Praca Konkursowa
na program i plan kształcenia
z informatyki dla zespołów uczniów (kół)
uzdolnionych informatycznie
na I rok nauczania
wraz z obudową metodyczną

„Mówi się często, że człowiek dotąd nie zrozumie czegoś, zanim nie nauczy tego - kogoś innego. W rzeczywistości, człowiek nie zrozumie czegoś naprawdę, zanim nie zdoła nauczyć tego – komputera.”

Donald E. Knuth

Od autorów

Szanowni Państwo,

Niniejsza praca zawiera program pracy z uczniem uzdolnionym. Przygotowując go założyliśmy jak największy udział samodzielnej pracy praktycznej uczniów, ograniczając do minimum treści przekazywane przez nauczyciela. Aby to osiągnąć skupiliśmy się przede wszystkim na przygotowaniu odpowiednich zadań praktycznych. Wszystkie będą zrealizowane w formacie zgodnym z SIO2. Uczniowie w trakcie rozwiązywania tych problemów zapoznają się z najważniejszymi pojęciami matematycznymi, technikami programowania oraz algorytmami i strukturami danych. Każde zajęcia zostaną również wzbogacone odnośnikami do ogólnodostępnych zadań (na przykład zamieszczonych w serwisie szkopuł.edu.pl) do samodzielnej pracy uczniów.

Taka forma realizacji zajęć kół przedmiotowych oraz lekcji informatyki sprawdza się od lat w VI LO w Radomiu. Mamy nadzieję, że wspomoże również pracę kolejnych nauczycieli informatyki w Polsce.

Agnieszka Król

Mirosław Mortka

Spis treści

1. Imiona i nazwiska autorów:

Agnieszka Król – askrol@op.pl, tel. 604232321

Mirosław Mortka – mmortka@kochanowski.radom.pl, tel. 501588811

Miejsce pracy:

VI Liceum Ogólnokształcące z Oddziałami Dwujęzycznymi im. Jana Kochanowskiego w Radomiu, ul. Kilińskiego 25

2. Krótki opis doświadczenia w pracy z uczniami uzdolnionymi informatycznie.

Agnieszka Król – 22 lata pracy w VI LO w Radomiu. Od 2005 r. jest czynnym egzaminatorem CKE z przedmiotu informatyka. Od kilkunastu lat prowadzi w szkole koła zainteresowań dla uzdolnionej młodzieży. Współpracowała przy organizacji zajęć w ramach projektu Mazowieckie Talenty w macierzystej szkole. Od 2013 roku współorganizuje (z panem Mirosławem Mortką) dniowe Warsztaty Informatyczne Liceum Kochanowskiego WILK (7 edycji; w tegorocznej po raz 3 wzięli również udział uczniowie z I LO w Lublinie, łącznie 101 uczestników).

Była opiekunem 6 finalistów Olimpiady Informatycznej (w tym 2 laureatów), kilku finalistów Olimpiady Informatycznej Gimnazjalistów oraz kilkunastu finalistów LOGII (w tym 7 laureatów).

Mirosław Mortka – 17 lat pracy w VI LO w Radomiu, w latach 2002-2008 pracownik Politechniki Radomskiej – obecnie Uniwersytet Techniczno-Humanistyczny – (prowadzenie zajęć ćwiczeniowych i laboratoryjnych: podstawy programowania, języki C i C++, algorytmy i struktury danych). Jest czynnym egzaminatorem CKE z przedmiotu informatyka. Od początku pracy w szkole prowadzi koła zainteresowań dla uzdolnionej młodzieży. Od 2013 roku organizuje Warsztaty Informatyczne Liceum Kochanowskiego WILK (7 edycji).

W edycjach od XVI do XXVI był opiekunem 44 finalistów Olimpiady Informatycznej (w tym 21 laureatów), 4 laureatów Międzynarodowej Olimpiady Informatycznej, 5 laureatów BOI oraz 4 CEOI. Był również opiekunem uczniów w innych konkursach (Olimpiada Informatyczna Gimnazjalistów czy Akademickie Mistrzostwa Polski w Programowaniu Zespołowym). Do jego absolwentów należą m. in. Karol Farbiś i Mateusz Radecki.

3. Główny cel kształcenia w ramach koła, opis głównych efektów uczenia się.

Cele kształcenia – wymagania ogólne

Program koła koncentruje się na najważniejszym założeniu nowej podstawy programowej: „Najważniejszym celem kształcenia informatycznego uczniów jest rozwój umiejętności myślenia komputacyjnego, skupionego na kreatywnym rozwiązywaniu problemów z różnych dziedzin ze świadomym i bezpiecznym wykorzystaniem przy tym metod i narzędzi wywodzących się z informatyki". Stąd w niniejszym programie nauczania kładzie się największy nacisk na logiczne i abstrakcyjne myślenie, myślenie algorytmiczne, programowanie i rozwiązywanie problemów z wykorzystaniem komputera, układanie i programowanie algorytmów. Równie ważne jest zwrócenie uwagi na przestrzeganie prawa i zasad bezpieczeństwa. Celem koła jest również rozwijanie kompetencji społecznych, takich jak: komunikacja i współpraca w grupie, w tym w środowiskach wirtualnych, udział w projektach zespołowych.

Opis głównych efektów uczenia się (zna podstawę programową z przedmiotu informatyka):

Uczeń:

- sprawnie posługuje się zintegrowanym środowiskiem programistycznym oraz systemem sprawdzającym rozwiązania uczniowskie przy pisaniu, uruchamianiu i testowaniu programów;
- do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;
- porównuje działanie różnych algorytmów dla wybranego problemu, analizuje algorytmy na podstawie ich gotowych implementacji;
- sprawdza poprawność działania algorytmów dla przykładowych danych;
- objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;
- ilustruje i wyjaśnia rolę pojęć, obiektów i operacji matematycznych w projektowaniu rozwiązań problemów informatycznych i z innych dziedzin;
- objaśnia sposoby wykonywania przez komputer działań arytmetycznych i operacji logicznych;
- implementuje w wybranym języku programowania poznane na kole algorytmy.

4. Opis kompetencji uczniów przystępujących do programu i propozycja sposobu weryfikacji ich posiadania.

Program zakłada, że uczniowie startują od podstaw programowania.

Ze względu na zróżnicowany poziom nauczania matematyki i informatyki w różnych szkołach podstawowych weryfikacji najlepiej dokonywać w trakcie trwania koła obserwując przede wszystkim:

- podejmowanie prób wykonywania zadań również poza zajęciami koła,
- wyniki osiągane w trakcie konkursów, zawodów, testów,
- samodzielność pracy w trakcie zajęć.

5. Plan zajęć (Temat, treści programowe z matematyki (M), programowania (P) i algorytmiki A)

Częstotliwość systematycznych zajęć, czas trwania jednych zajęć.

Program przewiduje realizację zadań w cyklu 2 godzin tygodniowo przez przynajmniej 30 tygodni - łącznie co najmniej 60 godzin lekcyjnych. Liczebność grupy powinna być nie większa niż 15 osób.

Proponowany plan:

1. Podstawy programowania. Potrzebne oprogramowanie. Pierwszy program w C++. Wypisywanie i odczytywanie. Typy w C++. (90 minut)

   P. Środowiska programistyczne (do wyboru, zalecane CodeBlocks). Wprowadzenie do programowania. Obsługa środowiska programistycznego. System sprawdzający rozwiązania uczniowskie. Wejście i wyjście (strumieniowe), formatowanie wyjścia. Operatory arytmetyczne. Zmienne i typy zmiennych (całkowity, rzeczywisty).

   Zadania do wykonania

   a. James Bond
   b. Klasy
   c. Obwód trójkąta
   d. Stopnie

   Zadania do samodzielnego wykonania (w scenariuszu zajęć)

2. Instrukcje warunkowe (90 minut)

   M. Rachunek zdań, kwantyfikatory (rozumienie). Podzielność. Arytmetyka modulo.

   P. Instrukcja warunkowa i operatory logiczne, typ logiczny. Wejście i wyjście (strumieniowe), formatowanie wyjścia.

   a. Szachy – liczby pól.
   b. Łamanie czekolady.
   c. Szachy – wieża.

   Zadania do samodzielnego wykonania (w scenariuszu zajęć)

3. Pętla for, while, do-while 1 (90 minut)

   M. Podzielność. Systemy liczbowe (reprezentacja liczb w komputerze), schemat Hornera, arytmetyka modulo.

P. Instrukcje iteracyjne. Instrukcja warunkowa i operatory logiczne, typ logiczny.

A. Metody projektowania algorytmów i strategie algorytmiczne: specyfikacja, projektowanie pętli – niezmienniki.

a. Dwójki

b. Litery w liczbie

c. Małpki na wybiegu

d. Szachownica

e. Najbliższy palindrom

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

4. Pętla for, while, do-while 2 (90 minut)

M. Podzielność. Schemat Hornera, arytmetyka modulo.

P. Synchronizacja wejścia/wyjścia strumieniowego. Instrukcje iteracyjne. Instrukcja warunkowa i operatory logiczne, typ logiczny.

A. Metody projektowania algorytmów i strategie algorytmiczne: specyfikacja, projektowanie pętli – niezmienniki. Elementy złożoności obliczeniowej: operacja dominująca, złożoność pesymistyczna, podstawowe klasy funkcji złożoności (logarytmiczna, liniowa, stała, pierwiaskowa)

a. Dzielniki (szkopuł), podstawowa arytmetyka (złożoność $\sqrt{n}$ vs. n)

b. Ile podzielnych przez 3 i przez 5 w przedziale (złożoność 1 vs. n)

c. Przenoszenie

d. Ścieżka Sterna-Brocota (złożoność ln n)

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

5. Tablice 1 (90 minut)

M. Zbiory i operacje na zbiorach.

P. Tablice statyczne jednowymiarowe.

A. Proste przetwarzanie tablic jednowymiarowych: przeszukiwanie, znajdowanie minimum/maksimum.

a. Temperatury

b. Najlepsze sumy (n zamiast n^2)

c. Bankiet (I OIG)

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

6. Tablice 2 (90 minut)

M. Zbiory i operacje na zbiorach.

P. Tablice statyczne jednowymiarowe. Tablice dwuwymiarowe.

A. Proste przetwarzanie tablic jednowymiarowych: wypełnianie, przeszukiwanie, przeszukiwanie cykliczne (rotating), znajdowanie minimum/maksimum, sumy prefiksowe, zliczanie. Złożoność obliczeniowa (podstawowe klasy funkcji złożoności (liniowa, kwadratowa, sześcienna). Proste przetwarzanie tablic dwuwymiarowych.

a. Pociąg

b. Bitib, Bajtjab i Palindrom

c. Odciski palca

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

7. Łańcuchy znaków (proste) (90 minut)

M. Ciągi arytmetyczne, zbiory.

P. Łańcuchy znaków (typ char, string, rzutowanie typów)

A. Zamiana systemu liczbowego. arytmetyka wielkich liczb. Proste algorytmy dla tekstów, wyszukiwanie wzorca w tekście algorytmem naiwnym. Relacje (porządek częściowy, porządek leksykograficzny).

a. Brakująca cyfra

b. Ciąg monotoniczny

c. System 36

d. James i potęgi dwójki

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

8. Funkcje 1 (90 minut)

M. Podzielność, NWD, algorytm Euklidesa, NWW, liczby pierwsze i złożone, testowanie pierwszości (algorytm „pierwiastkowy”), systemy liczbowe, schemat Hornera.

P. Funkcje, parametry funkcji, zasięg zmiennych.

A. Podstawowe algorytmy dla liczb całkowitych: zamiana systemu liczbowego, algorytm Euklidesa (iteracyjny), pierwiastkowy test pierwszości.

a. Dodawanie ułamków (NWD i NWW)

b. Liczby Super-B-pierwsze (matura)

c. Trzy liczby rosnąco.

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

9. Funkcje 2. Rekurencja (90 minut)

M. NWD, NWW. Techniki dowodowe, indukcja.

P. Funkcje, parametry funkcji, zasięg zmiennych. Funkcje rekurencyjne.

A. Elementy złożoności obliczeniowej, podstawowe klasy funkcji złożoności. Podstawowe algorytmy dla liczb całkowitych: zamiana systemu liczbowego, algorytm Euklidesa (rekurencyjny). Definicje i równania rekurencyjne, proste metody rozwiązywania równań rekurencyjnych (rozwijanie, argumenty kombinatoryczne).

a. Struś pędziwiatr (NWD - rekurencyjnie)

b. Liczby parzystocyfrowe (OIG, rekurencyjnie, kolejność wykonania)

c. Drzewo Sterna-Brocota

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

10. Zawody 1. Powtórzenie i podsumowanie. (90 minut)

Zawody (przygotowane przykładowe zadania).

11. Rekurencja. Metoda nawrotów (90 minut)

M. Kombinatoryka: zbiory i operacje na zbiorach, porządek leksykograficzny, liczby Fibonacciego, zliczanie obiektów kombinatorycznych, definicje i równania rekurencyjne, proste metody rozwiązywania równań rekurencyjnych. Techniki dowodowe. Indukcja.

P. Funkcje, parametry funkcji, zasięg zmiennych. Funkcje rekurencyjne. Łańcuchy znaków. Tablice dwuwymiarowe.

A. Podstawowe klasy funkcji złożoności. Proste algorytmy dla tekstów. Proste przetwarzanie tablic dwuwymiarowych: problem 8 hetmanów i inne. Przeszukiwanie drzew ukorzenionych (prefiksowe, infiksowe i postfiksowe). Metody projektowania algorytmów i strategie algorytmiczne: przeszukiwanie z powrotami.

a. Roztargniony profesor

b. Mysz w labiryncie

c. Problem 8 hetmanów

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

12. Sortowanie 1 (90 minut)

M. Zbiory i operacje na zbiorach, relacje (równoważności, porządek częściowy, porządek liniowy), ciągi (arytmetyczny), dowody kombinatoryczne. Sortowanie

P. Biblioteka algorithm (sortowanie)

A. Elementy złożoności obliczeniowej. Sortowanie przez wstawianie, sortowanie przez scalanie, sortowanie szybkie.

a. Latarnie

b. Najczęściej występujący (najc2)

c. Ciąg Farey'a (fare)

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

13. Sortowanie 2 (90 minut)

M. Zbiory i operacje na zbiorach, relacje (równoważności, porządek częściowy, porządek liniowy), ciągi (arytmetyczny), dowody kombinatoryczne. Sortowanie

P. Biblioteka algorithm (sortowanie)

A. Elementy złożoności obliczeniowej. Sortowanie przez wstawianie, sortowanie przez scalanie, sortowanie szybkie.

a. Wirgiliusz

b. Układanie kart

c. Równoważne teksty (rtezad)

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

14. Wyszukiwanie binarne (90 minut)

M. Techniki dowodowe.

P. Biblioteka algorithm (sortowanie, wyszukiwanie binarne).

A. Elementy złożoności obliczeniowej. Sortowanie. Wyszukiwanie binarne.

a. Znaczki

b. Tunele

c. Bierki

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

15. Wyszukiwanie binarne po wyniku (90 minut)

M. Techniki dowodowe.

P. Biblioteka algorithm (sortowanie, wyszukiwanie binarne).

A. Elementy złożoności obliczeniowej. Sortowanie. Wyszukiwanie binarne.

a. Zawody wędkarskie (zwezad)

b. Marchewki (marzad)

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

16. Programowanie zachłanne, dynamiczne 1 (90 minut)

M. Zbiory i operacje na zbiorach, ciągi, permutacje, kombinacje, zliczanie obiektów kombinatorycznych, dowody kombinatoryczne. Techniki dowodowe, indukcja.

P. Tablice jedno- i dwuwymiarowe. Zliczanie. Funkcje, funkcje rekurencyjne.

A. Przetwarzanie tablic dwuwymiarowych: wyznaczanie najtańszej drogi z dolnego lewego rogu do górnego prawego rogu w tablicy liczbowej przy ruchach tylko prawo i do góry, operacje macierzowe. Metody projektowania algorytmów i strategie algorytmiczne: specyfikacja, algorytmy zachłanne, rekurencja, dziel i rządź, programowanie dynamiczne, przeszukiwanie z powrotami, zamiatanie. Elementy złożoności obliczeniowej.

a. Pszczółki

b. iGruszka

c. Spadek

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

17. Programowanie zachłanne, dynamiczne 2 (90 minut)

M. Zbiory i operacje na zbiorach, ciągi, permutacje, kombinacje, zliczanie obiektów kombinatorycznych, dowody kombinatoryczne. Techniki dowodowe, indukcja.

P. Tablice jedno- i dwuwymiarowe. Zliczanie. Funkcje, funkcje rekurencyjne.

A. Przetwarzanie tablic dwuwymiarowych: wyznaczanie najtańszej drogi z dolnego lewego rogu do górnego prawego rogu w tablicy liczbowej przy ruchach tylko w prawo i do góry, operacje macierzowe. Metody projektowania algorytmów i strategie algorytmiczne: specyfikacja, algorytmy zachłanne, rekurencja, dziel i rządź, programowanie dynamiczne, przeszukiwanie z powrotami, zamiatanie. Elementy złożoności obliczeniowej.

a. Poszukiwania (szu)

b. Stadion

c. Zbiórka

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

18. Struktury danych 1 (90 minut)

M. Zbiory i operacje na zbiorach, ciągi. Techniki dowodowe, indukcja.

P. Struktury danych: stos, kolejka, vector, pair. Biblioteka STL.

A. Podstawowe abstrakcyjne struktury danych i ich implementacje: wektor, stos, kolejka, lista. Reprezentacja zbiorów rozłącznych (problem Find-Union). Kolejka priorytetowa.

a. Mnóż, odejmuj, dodawaj (mod)

b. Wodzirej (wod)

c. Gorące pudełko (gpuzad)

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

19. Struktury danych 2 (90 minut)

    M. Zbiory i operacje na zbiorach, ciągi. Techniki dowodowe, indukcja.

    P. Struktury danych: stos, kolejka, vector, pair. Biblioteka STL.

    A. Podstawowe abstrakcyjne struktury danych i ich implementacje: wektor, stos, kolejka, lista. Reprezentacja zbiorów rozłącznych (problem Find-Union). Kolejka priorytetowa.

    a. Kamienie

    b. Loginy

    c. Izba przyjęć

    Zadania do samodzielnego wykonania (w scenariuszu zajęć)

20. Zawody 2. Powtórzenie i podsumowanie. (90 minut)

    Zawody (przygotowane przykładowe zadania).

21. Grafy 1 (90 minut)

    M. Grafy nieskierowane, wierzchołki, stopień wierzchołka, krawędzie, ścieżki, metody przechodzenia grafów.

    P. Struktury danych. Metody projektowania algorytmów i strategie algorytmiczne: algorytmy zachłanne. Elementy złożoności obliczeniowej.

    A. Przeszukiwanie grafów (w głąb i wszerz), drzewo przeszukiwania. Spójność, sortowanie topologiczne.

    a. Dwukolorowanie

    b. Prehistoria

    Zadania do samodzielnego wykonania (w scenariuszu zajęć)

22. Grafy 2 (90 minut)

    M. Grafy nieskierowane, wierzchołki, stopień wierzchołka, krawędzie, ścieżki, cykle, spójność, metody przechodzenia grafów.

    P. Struktury danych. Metody projektowania algorytmów i strategie algorytmiczne: algorytmy zachłanne, rekurencja. Elementy złożoności obliczeniowej.

    A. Przeszukiwanie grafów (w głąb i wszerz), drzewo przeszukiwania. Spójność.

    a. Mysz w labiryncie

    b. Grand Prix Bajtocji

    Zadania do samodzielnego wykonania (w scenariuszu zajęć)

23. Grafy 3 (90 minut)

M. Grafy nieskierowane i skierowane, wierzchołki, stopień wierzchołka, krawędzie, ścieżki, cykle, spójność, metody przechodzenia grafów.

P. Struktury danych. Metody projektowania algorytmów i strategie algorytmiczne: algorytmy zachłanne, rekurencja, dziel i rządź, programowanie dynamiczne, przeszukiwanie z powrotami. Elementy złożoności obliczeniowej.

A. Przeszukiwanie grafów (w głąb i wszerz), drzewo przeszukiwania. Spójność, sortowanie topologiczne, cykl Eulera, silna spójność. Najkrótsze ścieżki (algorytm Dijkstry).

a. Głuchy telefon

b. Porządek alfabetyczny

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

24. Grafy 4 (90 minut)

M. Grafy nieskierowane i skierowane, wierzchołki, stopień wierzchołka, krawędzie, ścieżki, cykle, spójność, silna spójność, grafy ważone, podgrafy, metody przechodzenia grafów.

P. Struktury danych. Metody projektowania algorytmów i strategie algorytmiczne: algorytmy zachłanne, rekurencja, dziel i rządź, programowanie dynamiczne, przeszukiwanie z powrotami. Elementy złożoności obliczeniowej.

A. Przeszukiwanie grafów (w głąb i wszerz), drzewo przeszukiwania. Spójność, sortowanie topologiczne, cykl Eulera, silna spójność. Najkrótsze ścieżki (algorytm Dijkstry).

a. Mapa

b. Średniowiecze

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

25. Drzewa (90 minut)

M. Drzewa i lasy, drzewa rozpinające, drzewa ukorzenione, drzewa binarne i ich własności, metody przechodzenia drzew w tym drzew ukorzenionych.

P. Struktury danych. Metody projektowania algorytmów i strategie algorytmiczne: algorytmy zachłanne, rekurencja, dziel i rządź, programowanie dynamiczne, przeszukiwanie z powrotami. Elementy złożoności obliczeniowej.

A. Statyczne, zrównoważone drzewa przeszukiwań binarnych (drzewa przedziałowe, drzewa licznikowe), dynamiczne.

a. Samochody

b. Nasionka

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

26. Algorytmy geometryczne 1 (90 minut)

M. Geometria analityczna na płaszczyźnie: punkt, odcinek, prosta, wektor, okrąg i koło, wielokąt i jego własności, reprezentacja obiektów geometrycznych w komputerze, metryki Eucklidesowa i miejska, iloczyn skalarny i wektorowy wraz z zastosowaniami.

P. Struktury danych. Biblioteka STL. Typ struct. Tablice.

A. Proste algorytm geometryczne na płaszczyźnie: lokalizacja punktu w wielokącie, wypukła otoczka, pola wielokątów, wypukła otoczka.

a. Proste prostopadłe

b. Fioletowe lasery

c. Wiśniowy sad

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

27. Algorytmy geometryczne 2 (90 minut)

M. Geometria analityczna na płaszczyźnie: punkt, odcinek, prosta, wektor, okrąg i koło, wielokąt i jego własności, reprezentacja obiektów geometrycznych w komputerze, metryki Eucklidesowa i miejska, iloczyn skalarny i wektorowy wraz z zastosowaniami.

P. Struktury danych. Biblioteka STL. Typ struct. Tablice.

A. Proste algorytm geometryczne na płaszczyźnie: lokalizacja punktu w wielokącie, wypukła otoczka, wypukła otoczka.

a. Bombardowanie

b. Mur

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

28. Algorytmy tekstowe 1 (90 minut)

M. Zbiory i operacje na zbiorach. Relacje (równoważności, porządek częściowy, porządek liniowy, porządek leksykograficzny). Techniki dowodowe (wprost, nie wprost), indukcja.

P. Łańcuchy znaków (typ char, string, rzutowanie typów). Tablice. Biblioteka STL.

A. Proste algorytmy dla tekstów, palindromy, wyszukiwanie wzorca w tekście algorytmem naiwnym, algorytm KMP, haszowanie.

a. Czołgiści

b. Ści(ą)gany

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

29. Algorytmy tekstowe 2 (90 minut)

M. Zbiory i operacje na zbiorach. Relacje (równoważności, porządek częściowy, porządek liniowy, porządek leksykograficzny). Techniki dowodowe (wprost, nie wprost), indukcja.

P. Łańcuchy znaków (typ char, string, rzutowanie typów). Tablice. Biblioteka STL.

A. Proste algorytmy dla tekstów, palindromy, wyszukiwanie wzorca w tekście algorytmem naiwnym, algorytm KMP, haszowanie.

a. Czy tu już byłem?

b. Numer telefonu

Zadania do samodzielnego wykonania (w scenariuszu zajęć)

30. Zawody 3. Powtórzenie i podsumowanie. (90 minut)

Zawody (przygotowane przykładowe zadania).

6. Koncepcja i opis innych form kształcenia (prace samodzielne, obozy, grywalizacja itp.).

Program koła zakłada po każdych kolejnych 9 kołach zajęcia podsumowujące w formie zawodów (po 9, 19 i 29 zajęciach).

Do każdych zajęć zostały wskazane konkretne ogólnodostępne zadania, zwykle o podwyższonym stopniu trudności, do samodzielnej pracy.

Zaleca się również jak najszybszy start w zawodach programistycznych (turnieje w ramach MAP, zawody w serwisach internetowych, na przykład CodeForces).

Ponadto zaleca się udział w obozie/warsztatach informatycznych (4-5 dniowe) organizowanych przez podmiot zewnętrzny, jeśli nauczyciel nie ma doświadczenia w tej dziedzinie.

7. Szczegółowe opisy poszczególnych zajęć

Zajęcia 1

Temat: Podstawy programowania. Potrzebne oprogramowanie. Pierwszy program w C++. Wypisywanie i odczytywanie. Typy w C++.

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin (matematyka, fizyka, informatyka), stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, testuje poprawność programów dla różnych danych;

Efekty:

- umie wybrać środowisko programistyczne,
- umie uruchomić potrzebne oprogramowanie,
- zna zasady pracy na portalu szkopuł.edu.pl,
- umie napisać prosty program w C++,
- zna podstawowe typy w C++,
- umie formatować wyjście

Formy i metody pracy: praca samodzielna, wykład, omówienie

Zadania do wykonania na zajęciach	Treści programowe
1. Bond	M.1, P.1, P.2.1, P.2.2, P.2.6
2. Klasy	M.1, P.2.3, P.2.6
3. Obwód trójkąta	M.1, P.2.3, P.2.6
4. Stopnie	M.1, P.2.3

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/6/4/

https://www.main2.edu.pl/main2/courses/show/6/7/

https://www.main2.edu.pl/main2/courses/show/6/12/

Zadania do wykonania w domu:

Zadania z ogólnodostępnych serwisów z zadaniami.

Choinka:
https://szkopul.edu.pl/problemset/problem/byKw2vwQV3stdD_23STSTQVq/site/?key=statement

Read/Write:

https://szkopul.edu.pl/problemset/problem/VYKVdv984yra7FZWHFpzfy6j/site/?key=statement

Koło:

https://szkopul.edu.pl/problemset/problem/wzEgXE9QizulWQd-HtbMhs8R/site/?key=statement

ZADANIA I ROZWIĄZANIA

Zadanie 1. Bond

Poniżej widzisz kod programu, który na ekranie wypisuje komunikat: "HELLO WORLD!". Zmodyfikuj treść programu tak, aby wypisywał w pierwszej linii komunikat "My name is Bond.", zaś w drugiej "James Bond.".

```
#include<iostream>
using namespace std;
int main()
{
  cout << "HELLO WORLD!";
  return 0;
}
```

Wejście

Twój program nie powinien oczekiwać żadnych danych.

Wyjście

W pierwszej linii wypisz komunikat: "My name is Bond.", zaś w drugiej "James Bond.".

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z obsługą środowiska programistycznego, strukturą programu w C++ oraz systemu sprawdzającego rozwiązania uczniowskie.
W zadaniu zwracamy uwagę na dokładność wypisywanych komunikatów przez ucznia (formatowanie wyjścia).

Zadanie 2. Klasy

Dostępna pamięć: 256MB

W liceum w Bajtomiu przyjęto nowych uczniów do trzech klas pierwszych. Zapamiętaj liczby uczniów w każdej klasie, a później je wypisz.

Wejście

W pierwszej linii wejścia znajdują się trzy liczby całkowite a, b oraz c ($1 \le a, b, c \le 50$), odpowiednio liczba uczniów w klasie a, b i c.

Wyjście

W pierwszym wierszu wyjścia wypisz liczby uczniów w klasach a, b i c. W kolejnych trzech liniach wypisz nazwy klas (mała litera) oraz (po odstępie) liczbę uczniów w każdej z klas.

Przykład

Wejście	Wyjście
12 34 23	12 34 23 a 12 b 34 c 23

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z całkowitymi typami zmiennych (int), wejściem i wyjściem (strumieniowym) oraz formatowaniem wyjścia.

W zadaniu zwracamy uwagę na dokładność wypisywanych komunikatów przez ucznia.

Zadanie 3. Obwód trójkąta

Dostępna pamięć: 32MB

Dla danych długości boków trójkąta oblicz jego obwód.

Wejście

W pierwszym wierszu wejścia znajduje się trzy liczby całkowite a, b i c ($1 \le a, b, c \le 10^9$) – długości boków trójkąta. Możesz założyć, że z podanych długości boków zawsze będzie można zbudować trójkąt.

Wyjście

Długość obwodu trójkąta.

Przykład

Wejście	Wyjście
4 3 2	9

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z całkowitymi typami zmiennych (long long), wejściem i wyjściem (strumieniowym) oraz formatowaniem wyjścia.

Zadanie 4. Stopnie

Limit pamięci: 64MB

Napisz program, który dla podanej temperatury w stopniach Fahrenheita wypisze temperaturę w stopniach Celsjusza.

Możesz wykorzystać wzór: °C = 5 / 9 (°F – 32)

Wejście

Pierwszy wiersz wejścia zawiera jedną liczbę całkowitą f ($-459 \le f \le 10^9$) – temperaturę w stopniach Fahrenheita.

Wyjście

Na wyjściu wypisz temperaturę w stopniach Celsjusza zaokrągloną do 2 miejsc po przecinku.

Przykład

Wejście	Wyjście

32	0.00

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z całkowitymi oraz rzeczywistymi typami zmiennych (float oraz double) oraz z formatowaniem wyjścia dla liczb rzeczywistych.

Należy zwrócić uwagę na „pułapkę” w zadaniu: 5 / 9 dla liczb całkowitych zawsze daje wynik 0.

Zajęcia 2

Temat: Instrukcje warunkowe

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin (matematyka, informatyka), stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, różne typy zmiennych; testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- zna zasady pracy na portalu szkopuł.edu.pl, main2.edu.pl
- zna podstawowe typy w C++,
- umie formatować wyjście,
- zna podstawowe operatory logiczne,
- zna zasadę podzielności i parzystości liczb,
- umie napisać prosty program z wykorzystaniem instrukcji warunkowej w C++,

Formy i metody pracy: praca samodzielna, omówienie

Zadania do wykonania na zajęciach	Treści programowe
1. Szachy – Liczby pól	M.2, P.2.3, P.2.9
2. Łamanie czekolady	M.2, P.2.3, P.2.9
3. Szachy – Wieża	M.1, P.2.3, P.2.9

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/6/10/

Zadania do wykonania w domu:

Zadania z ogólnodostępnych serwisów z zadaniami.

Przestępność:

https://szkopul.edu.pl/problemset/problem/XyXPTzp1WMMdDFnomLVq8k_u/site/?key=statement

Ćwiartka:

https://main2.edu.pl/c/konkurs-wstepu-do-programowania/p/cwi/

Trójkąt:

https://szkopul.edu.pl/problemset/problem/dvaxj3WZQniteVixM49HlSJt/site/?key=statement

Stół:

https://main2.edu.pl/c/konkurs-wstepu-do-programowania/p/sto/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Szachy – Liczby pól

Limit pamięci: 64MB

Wiele gier korzysta z planszy mającej kształt kwadratu i złożonej z równej ilości wierszy i kolumn. Pola są malowane naprzemiennie w kolorach jasnym i ciemnym. Plansza taka nazywana jest szachownicą. Czy zastanawiałeś się kiedyś, ile pól na takiej szachownicy jest jasnych, a ile ciemnych?

Wejście

Na standardowym wejściu znajduje się jedna liczba całkowita n określająca rozmiar szachownicy ($1 \leq n \leq 10^3$).

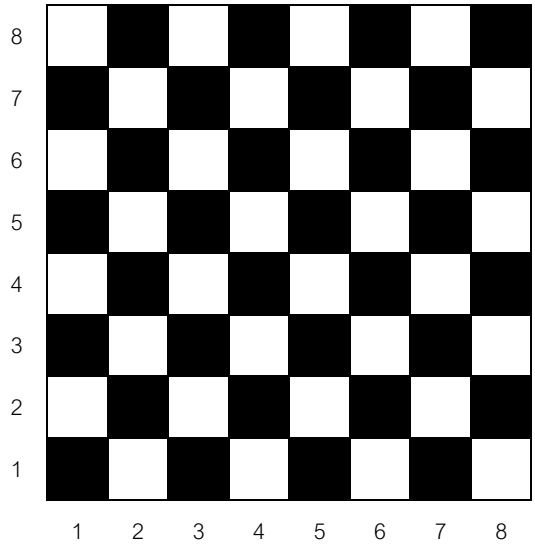
Wyjście

Na wyjściu w pierwszym wierszu wypisz liczbę pól jasnych, w drugim – ciemnych. Zakładamy, że pole o współrzędnych [1, 1] jest polem ciemnym.

Przykład

Wejście	Wyjście
8	32 32

Ilustracja przykładu:



Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z całkowitymi typami zmiennych, instrukcją warunkową, typami logicznymi oraz podzielnością liczb.

Rozwiązanie zadania zależne jest od parzystości liczby n (w przypadku nieparzystego n pól czarnych jest o 1 więcej niż białych).

```
wczytaj n
wypisz ((n ·n) div 2)
jeżeli n mod 2 = 0
     wypisz ((n· n) div 2)
w przeciwnym wypadku
     wypisz ((n · n )div 2) + 1
```

Zadanie 2. Łamanie czekolady

Limit pamięci: 64MB

Pan Integer kupił swoją ulubioną czekoladę z nadzieniem toffi. Czekolada ma kształt prostokąta o rozmiarze n na m kawałków. Pan Integer chciałby teraz odłamać jednym ruchem dokładnie k kawałków. Czy jest to możliwe?

Wejście

Pierwszy wiersz wejścia zawiera dwie liczby całkowite n oraz m – rozmiar czekolady ($1 \le n, m \le 10^6$). W kolejnej linii znajduje się całkowita liczba k kawałków czekolady, które chce odłamać pan Integer ($1 \le k \le 10^6$).

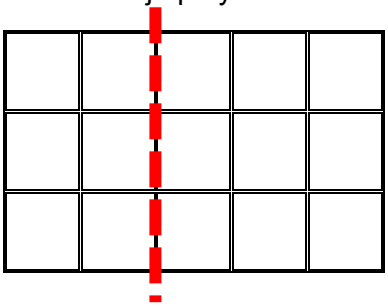
Wyjście

Na wyjściu wypisz odpowiedź na pytanie, czy pan Integer może jednym przełamaniem oderwać k kawałków czekolady (TAK lub NIE).

Przykład

Wejście	Wejście
3 5 6	4 8 6
Wyjście	Wyjście
TAK	NIE

Ilustracja przykładu 1:



Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z całkowitymi typami zmiennych, instrukcją warunkową, operatorami logicznymi oraz podzielnością liczb.

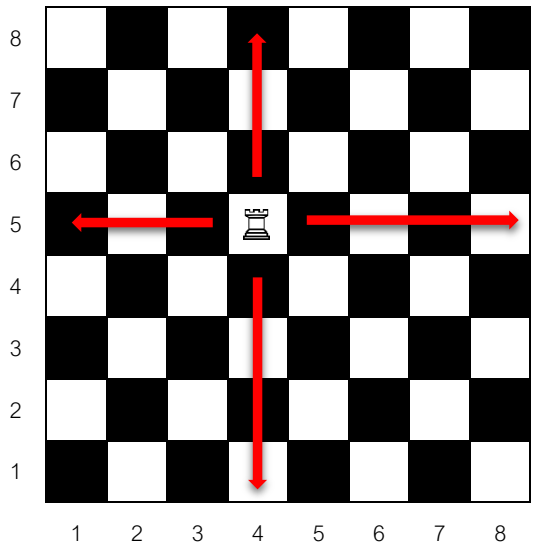
Rozwiązanie zadania zależne jest od podzielności liczby k przez któryś z rozmiarów czekolady. Trzeba również zwrócić uwagę na to, czy czekolada jest wystarczająco duża.

```
wczytaj n, m, k
jeżeli (k mod m = 0 lub k mod n = 0) i n · m ≥ k
     wypisz TAK
w przeciwnym wypadku
     wypisz NIE
```

Zadanie 3. Szachy – Wieża

Limit pamięci: 64MB

Wieża po szachownicy przesuwa się pionowo lub poziomo. Znając początkowe położenie wieży określ, czy w jednym ruchu można przenieść ją na wybrane pole.



Wejście

Pierwszy wiersz wejścia zawiera dwie oddzielone spacją liczby naturalne x_1 oraz y_1 ($1 \le x_1, y_1 \le 8$) – współrzędne pola, na którym stoi wieża. Drugi wiersz zawiera dwie liczby naturalne x_2 oraz y_2 ($1 \le x_2, y_2 \le 8$) – współrzędne pola, na które chcemy przesunąć wieżę.

Wyjście

Wypisz informację, czy z pola $[x_1, y_1]$ można przesunąć wieżę na pole $[x_2, y_2]$ w jednym ruchu: TAK lub NIE.

Przykład

Dla danych wejściowych:	poprawną odpowiedzią jest:
4 5 5 4	NIE

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z całkowitymi typami zmiennych, instrukcją warunkową oraz operatorami logicznymi (alternatywa rozłączna – xor).

Wystarczy sprawdzić, czy zmienia się wyłącznie jedna współrzędna jednocześnie.

```
wczytaj x1, y1
wczytaj x2, y2
jeżeli x1 ≠ x2 albo y1 ≠ y2
     wypisz TAK
w przeciwnym wypadku
     wypisz NIE
```

Zajęcia 3

Temat: Pętla for, while, do-while 1

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- zna różne systemy liczbowe i potrafi dokonać zamiany,
- umie napisać program z wykorzystaniem instrukcji warunkowej i iteracyjnej,

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Dwójki	M.1, P.2.10
2. Litery w liczbie	M.2, P.2.10, A.1, A.3.1
3. Małpki na wybiegu	P.2.10
4. Szachownica	M.1, P.2.10
5. Najbliższy palindrom	P.2.10, A.1, A.3.1

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/6/14/

https://www.main2.edu.pl/main2/courses/show/6/17/ (część dot. pętli)

Zadania do wykonania w domu:

Zadania z ogólnodostępnych serwisów z zadaniami.

Maksymalna różnica:

https://szkopul.edu.pl/problemset/problem/ewrYvFHd7H87PgwhxdWtmaBv/site/?key=statement

Prostokąt:

https://szkopul.edu.pl/problemset/problem/ihvm-DVIj3xdP2fKuCm4cJeO/site/?key=statement

Ciąg sum częściowych:

https://szkopul.edu.pl/problemset/problem/B6X2pKq-yacHF3AvjcDs48Lk/site/?key=statement

ZADANIA I ROZWIĄZANIA

Zadanie 1. Dwójki

Limit pamięci: 64MB

Dana jest liczba naturalna x ($x<10^{18}$). Dla każdego x wypisz liczbę potęg 2 mniejszych bądź równych każdego z x.

Przykład

Wejście 10 Wyjście 4	Wejście 64 Wyjście 7

Rozwiązanie

Zadanie wymaga użycia pętli `while`.
W zadaniu zaczynamy od wartości potęgi równej (dla x równego 0). Tak długo mnożymy kolejne dwójki, aż uzyskany wynik będzie większy niż zadana liczba. Jednocześnie zliczamy wykonane mnożenia.

```
potęga ← 1, liczba_potęg ← 0
wczytaj x
dopóki potęga < x wykonaj
     potęga ← potęga • x
     liczba_potęg ← liczba_potęg + 1
wypisz liczba_potęg - 1
```

Zadanie 2. Litery w liczbie

Janek bada właściwości liczb i ostatnio szuka takich liczb, które zapisane w postaci szesnastkowej zawierają co najmniej jedną literę.

Wejście

Pierwszy wiersz danych zawiera liczbę naturalną mniejszej od tryliona.

Wyjście

Wypisz komunikat TAK, jeśli w postaci szesnastkowej występują litery lub NIE – jeśli ich brak.

Wejście 54 Wyjście NIE	Wejście 255 Wyjście TAK

Rozwiązanie

Zadanie wymaga użycia pętli `while`.
Wykorzystamy fakt, że aby zamienić liczbę z systemu dziesiętnego na liczbę w systemie o dowolnej innej podstawie wystarczy dzielić przez nową podstawę. Jednak zamiast

wypisywać reszt, wystarczy sprawdzić, czy w wyniku wystąpiłaby litera (reszta wynosi co najmniej 10).

```
wartownik ← 0
wczytaj x
dopóki x > 0 i wartownik = 0 wykonaj
     jeżeli x mod 16 ≥ 10 to
          wartownik ← 1
     x ← x div 16
jeżeli wartownik = 1 to
     wypisz „TAK”
w przeciwnym wypadku
     wypisz „NIE”
```

Zadanie 3. Małpy na wybiegu

Dostępna pamięć: 32MB

Mały Bitek trenuje małpki. Umie już je ustawić w jednym rzędzie. Planuje teraz kolejne sztuczki. Na noc jednak Bitek musi schować małpki na wybiegu. Pomóż mu!

Wejście

W pierwszym i jedynym wierszu wejścia znajduje się jedna liczba całkowita n ($1 \leq n \leq 10^3$) – liczba małpek.

Wyjście

Wypisz n znaków @ obok siebie otoczonych płotkiem ze znaków #.

Przykład

Wejście	Wyjście
5	#######
	#@@@@@#
	#######

Rozwiązanie

Zadanie pozwala na trzykrotne użycie pętli `for`.
Zauważmy, że będziemy wypisywać kolejne znaki @ oraz #. W pierwszej linii musimy wypisać n+2 kratki (ogrodzenie jest większe z lewej i prawej strony o jeden element). W drugiej linii wypiszemy jedną kratkę, n małpek i znowu jedną kratkę. W trzeciej linii ponownie musimy wypisać n+2 kratki.

```
wczytaj n
dla i=1 do n+2 wykonaj
     wypisz #
przejdź do nowej linii
wypisz #
dla i=1 do n wykonaj
     wypisz @
wypisz #
przejdź do nowej linii
dla i=1 do n+2 wykonaj
```

```
wypisz #
```

Zadanie 4. Szachownica

Dostępna pamięć: 32MB

Napisz program, który dla podanej na standardowym wejściu liczby całkowitej n, narysuje szachownicę z cyfr 0 i 1 o boku n.

Wejście

Jedyny wiersz danych zawiera liczbę całkowitą n ($1 \leq n \leq 200$).

Wyjście

Program powinien wypisać szachownicę o wielkości n.

Przykład

Wejście	Wyjście
5	01010 10101 01010 10101 01010

Rozwiązanie

Zadanie wymaga użycia zagnieżdżonej pętli, na przykład `for`.
Zewnętrzna pętla będzie odpowiadać za rysowanie kolejnych wierszy (iterator y), zaś wewnętrzna kolejnych elementów w wierszu (iterator x). Zauważmy, że cyfry 0 występują zawsze, gdy numer wiersza i kolumny ma tę samą parzystość. W zadaniu wykorzystamy fakt, że suma dwóch liczb o tej samej parzystości jest parzysta.

```
wczytaj n
dla y=1 do n wykonaj
     dla x=1 do n wykonaj
          jeżeli (x + y) mod 2 = 0
               wypisz 0
          w przeciwnym wypadku
               wypisz 1
     przejdź do nowej linii
```

Analizując warunek możemy dokonać uproszczenia:

```
wczytaj n
dla y=1 do n wykonaj
     dla x=1 do n wykonaj
          wypisz (x + y) mod 2
     przejdź do nowej linii
```

Zadanie 5. Najbliższy palindrom

Mały Bitek przeczytał na Wikipedii: Palindrom (gr. palindromeo – biec z powrotem) – wyrażenie brzmiące tak samo czytane od lewej do prawej i od prawej do lewej. Janek szybko

ułożył sobie jeden palindrom (biorąc pod uwagę tylko litery): „U Izydy żądze na wyrku co noc ukrywane, zdąży Dyziu?”. Potem zaczął wymyślać całą masę następnych. Niestety – dużo gorzej mu idzie z liczbami. Chciałby szybko zamienić dowolna liczbę (jeśli nie jest ona palindromem) na najbliższy większy od niej palindrom. Czy mu pomożesz?

Wejście

Pierwszy wiersz danych zawiera liczbę całkowitą k ($10 \leq k \leq 1\,000\,000$) do zamiany na palindrom.

Wyjście

Program powinien wypisać w jedną liczbę całkowitą – najmniejszą możliwą liczbę nieujemną, którą należy dodać do danej liczby, aby otrzymać palindrom.

Przykład

Wejście 150 Wyjście 1	Wejście 55555 Wyjście 0	Wejście 142 Wyjście 9

Rozwiązanie

Zacznijmy od sprawdzenia, czy podana liczba k jest palindromem. W tym celu wyznaczymy liczbę odwrócona_k (z odwróconą kolejnością cyfr w liczbie). Będziemy „odrywać” po jednej cyfrze z liczby o wartości k (operacje mod 10 oraz div 10).

```
k' ← k
odwrócona_k ← 0
dopóki k' > 0 wykonaj
      odwrócona_k ← odwrócona_k • 10 + k' mod 10
      k' ← k' div 10
```

Zauważmy, że w przypadku, gdy odwrócona_k jest równa k mamy do czynienia z palindromem. Możemy więc dodawać kolejne wartości tak długo, aż otrzymamy palindrom.

```
wartownik ← 0
i ← 0
dopóki wartownik = 0 wykonaj
      OBLICZ odwrócona_k dla wartości k+i
      jeżeli odwrócona_k = k+1
            wartownik ← 1
      w przeciwnym wypadku
            i ← i + 1
wypisz i
```

Zajęcia 4

Temat: Pętla for, while, do-while 2

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów; bierze pod uwagę złożoność obliczeniową

Efekty: ,

- umie napisać program z wykorzystaniem instrukcji warunkowej i iteracyjnej,
- wie jak optymalizować program,
- umie policzyć złożoność obliczeniową

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Dzielniki	M.2, P.2.10, A.2
2. Podzielne	M.2, P.2.10, A.2
3. Przenoszenie	M.2, P.2.10, A.2
4. Ścieżka Sterna-Brocota	M.2, P.2.10, A.2

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/6/14/

https://www.main2.edu.pl/main2/courses/show/6/17/ (część dot. pętli)

Zadania do wykonania w domu:

Choinka 2:

https://szkopul.edu.pl/problemset/problem/LNqt3rNp7kiR-65zBzZS3_sL/site/?key=statement

Wężyk:

https://szkopul.edu.pl/problemset/problem/T8JDkgJLZX_h9Hd_ead-sTum/site/?key=statement

ZADANIA I ROZWIĄZANIA

Zadanie 1. Dzielniki

Dostępna pamięć: 256MB. Źródło: szkopuł.edu.pl

Zadaniem Twojego programu będzie wypisanie wszystkich naturalnych dzielników zadanej liczby. Napisz program, który wczyta ze standardowego wejścia liczbę naturalną n, a następnie wypisze na standardowe wyjście wszystkie dzielniki liczby uporządkowane rosnąco.

Wejście

W pierwszej linii wejścia znajduje się jedna liczba całkowita n ($1 \leq n \leq 10^{12}$).

Wyjście

W i-tym wierszu wyjścia należy wypisać i-ty z kolei dzielnik liczby n.

Przykład

Wejście	Wyjście
12	1 2 3 4 6 12

Rozwiązanie

Należy zwrócić uwagę na szybkość oraz poprawność rozwiązania.

Rozwiązanie wolne sprawdza wszystkie dzielniki z zakresu [1, n]. Wymaga więc wykonania n operacji sprawdzenia podzielności.

```
dla i=1 do n wykonaj
     jeżeli n mod i = 0
          wypisz i
```

Rozwiązanie szybsze korzysta z właściwości: $a \cdot b = n$ dla $a \leq \sqrt{n}$ i $b \geq \sqrt{n}$.

Stąd wystarczy sprawdzać wyłącznie liczby mniejsze bądź równe $\sqrt{n}$.

Ponadto funkcję matematyczną pierwiastek(n) można wyeliminować z poniższego zapisu podnosząc strony nierówności do kwadratu (rozwiązanie wzorcowe w C++).

```
dla i=1 do √n wykonaj
     jeżeli n mod i = 0
          wypisz i
dla i=√n do i wykonaj
     jeżeli n mod i = 0 ∧ i·i<n
          wypisz n div i
```

Zadanie 2. Podzielne

Dostępna pamięć: 32MB

Dane są liczby naturalne a i b. Wypisz, ile liczb w przedziale jest podzielnych przez 3 lub przez 5.

Wejście

W pierwszej linii wejścia znajdują się dwie liczby całkowite a i b ($1 \le a \le b \le 10^{15}$).

Wyjście

Na standardowym wyjściu należy wypisać jedną liczbę całkowitą.

Przykład

Wejście	Wyjście
25 75	24

Rozwiązanie

Należy zwrócić uwagę na szybkość oraz poprawność rozwiązania.

Rozwiązanie wolne sprawdza wszystkie liczby z zakresu [a, b]:

```
ile ← 0
dla i=a do b wykonaj
      jeżeli i mod 3 = 0 lub i mod 5 = 0
            ile ← ile + 1
wypisz ile
```

Wymaga więc wykonania b – a + 1 operacji sprawdzenia podzielności. W skrajnym wypadku będziemy być może musieli wykonać aż 10^{15} operacji. Zakładając, że komputer wykonuje około 10^8 operacji sprawdzenia podzielności w ciągu sekundy, potrzebowalibyśmy na to rozwiązanie około 10^7 sekund, co daje prawie 116 dni.

Zauważmy jednak, że liczb podzielnych przez 3 w przedziale [1, b] jest b div 3. A zatem w przedziale [a, b] liczb podzielnych przez 3 jest b div 3 – (a-1) div 3. Podobnie postępujemy dla 5. Pamiętajmy, że musimy wyeliminować liczby jednocześnie podzielne przez 3 i przez 5, które policzyliśmy dwukrotnie.

```
podzielne_przez_3 ← b div 3 – (a – 1) div 3
podzielne_przez_5 ← b div 5 – (a – 1) div 5
podzielne_przez_15 ← b div 15 – (a – 1) div 15
wypisz podzielne_przez_3 + podzielne_przez_5 - podzielne_przez_15
```

Ten program wykonuje tylko 3 operacje przypisania, działa więc w czasie stałym, niezależnym od liczb a oraz b.

Zadanie 3. Przenoszenie

Jasio uczy się dodawać wielocyfrowe liczby od prawej do lewej, po jednej cyfrze. Dla Jasia operacja przeniesienia, podczas której jedynka jest przenoszona z jednej pozycji do następnej, stanowi poważne wyzwanie. Twoim zadaniem jest policzenie, ile operacji przeniesienia wystąpi w każdym z dodawań w danym zestawie. Pomoże to Jasiowi w oszacowaniu trudności zadań.

Wejście

W pierwszej linii wejścia znajduje się liczba n (1 ≤ n ≤ 25) – liczba zestawów testowych. W każdym z n następnych wierszy znajdują się po dwie liczby całkowite bez znaku, każda z nich ma mniej niż 18 cyfr.

Wyjście

Dla każdego z n zestawów liczb wypisz liczbę operacji przeniesienia występujących podczas dodawania dwóch liczb.

Przykład

Wejście	Wyjście
3 234 342 654 456 191 111	0 3 1

Rozwiązanie

Zauważmy, że dla każdej z par liczb wystarczy sprawdzać kolejne cyfry począwszy od ostatniej do pierwszej dodając je do siebie wraz z bitem przeniesienia (na początku równym 0). Bit przeniesienia pojawia się wówczas, gdy suma dwóch ostatnich cyfr jest równa co najmniej 10. Wynikiem jest liczba bitów przeniesienia równych 1. Ponieważ liczby mogą składać się z różnej liczby cyfr, musimy liczyć względem większej z nich.

Zadanie dodatkowe dla uczniów: udowodnij, że bit przeniesienia nie będzie większy niż 1.

Operacja zliczania przeniesień dla dwóch liczb:

```
wczytaj a, b
ile ← 0
bit ← 0
dopóki a ≠ 0 lub b ≠ 0 wykonuj
     bit ← (bit + a mod 10 + b mod 10) div 10
     ile ← ile + bit
wypisz ile
```

Jak szybko działa nasze rozwiązanie? Pętla wykonuje liczbę „obrotów” zgodną liczbą cyfr większej liczb, jest więc równa $\lfloor \lg(\max(a,b)) \rfloor + 1$.

Zadanie 4. Ścieżka Sterna-Brocota

Drzewo Sterna-Brocota to drzewo binarne zawierające wszystkie dodatnie ułamki nieskracalne. Struktura ta posiada wiele ciekawych właściwości. Jeśli liczby a oraz b są względnie pierwsze, to ułamek $\frac{a}{b}$ występuje w drzewie dokładnie jeden raz. Ponadto każdą

liczbę rzeczywistą dodatnią możemy zapisać jako ciąg symboli L oraz P tak, że początkowe fragmenty tego ciągu symbolizują liczby wymierne przybliżające tę liczbę. Na przykład liczbę $\frac{5}{7}$ opiszemy jako LPPL.

Zaczynamy od $\frac{0}{1}$ symbolizującego zero i $\frac{1}{0}$ symbolizującego nieskończoność. Następnie na kolejnych piętrach drzewa wpisujemy „pomiędzy" wartości $\frac{a}{b}$ oraz $\frac{c}{d}$ wartość $\frac{a+c}{b+d}$.

Naszymi wartościami startowymi jest $\frac{0}{1}, \frac{1}{1}, \frac{1}{0}$.

Zatem w pierwszym kroku mamy: $\frac{0}{1} \quad \frac{1}{1} \quad \frac{1}{0}$

W drugim kroku: $\frac{0}{1} \quad \frac{1}{2} \quad \frac{1}{1} \quad \frac{2}{1} \quad \frac{1}{0}$

W trzecim: $\frac{0}{1} \quad \frac{1}{3} \quad \frac{1}{2} \quad \frac{2}{3} \quad \frac{1}{1} \quad \frac{3}{2} \quad \frac{2}{1} \quad \frac{3}{1} \quad \frac{1}{0}$

Zaś w czwartym: $\frac{0}{1} \quad \frac{1}{4} \quad \frac{1}{3} \quad \frac{2}{5} \quad \frac{1}{2} \quad \frac{3}{5} \quad \frac{2}{3} \quad \frac{3}{4} \quad \frac{1}{1} \quad \frac{4}{3} \quad \frac{3}{2} \quad \frac{5}{3} \quad \frac{2}{1} \quad \frac{5}{2} \quad \frac{3}{1} \quad \frac{4}{1} \quad \frac{1}{0}$

Napisz program, który czyta dwie liczby m oraz n i wypisuje ścieżkę Sterna-Brocota.

Wejście

Jedyny wiersz danych zawiera dwie *względnie pierwsze* liczby całkowite naturalne m i n ($1 \le m, n \le 10^5$, $m \neq n$).

Wyjście

Program powinien wypisać ścieżkę z drzewa Sterna-Brocota.

Przykład

Wejście 5 3 Wyjście PLP	Wejście 7 2 Wyjście PPPL

Rozwiązanie

Zadanie wymaga zapamiętanie lewego i prawego zakresu $(\frac{0}{1}, \frac{1}{0})$, a następnie wyznaczania nowego „środka" i porównywanie go z $\frac{m}{n}$ (obliczona wartość może być mniejsza – P, większa – L, lub równa, co kończy nasze zadanie).

Wykorzystamy pętlę `do-while`:

```
a ← 0, b ← 1, c ← 1, d ← 0
wykonuj
      e ← a + c
      f ← b + d
      jeżeli m/n > e/f
            a ← e, b ← f
```

```
            wypisz 'P'
      jeżeli m/n < e/f
            c ← e, d ← f
            wypisz 'L'
dopóki m/n ≠ e/f
```

Aby wszystkie operacje wykonywać wyłącznie na liczbach rzeczywistych wykorzystamy:

$$\frac{x1}{y1} = \frac{x2}{y2} \quad \equiv \quad x1 \cdot y2 = y1 \cdot x2 \qquad dla\ y1, y2 \geq 1$$

Zadanie dla uczniów: ile „obrotów” pętli wykona powyższy algorytm w skrajnym wypadku?

Zajęcia 5

Temat: Tablice 1

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem instrukcji warunkowej, iteracyjnej i tablic w C++,
- umie wyczytywać dane do tablicy i wykonywać proste operacje,

Formy i metody pracy: praca samodzielna,

Zadania do wykonania na zajęciach	Treści programowe
1. Temperatury	M.2, P.2.13, A.3.2
2. Najlepsze sumy	M.2, P.2.13, A.3.2
3. Bankiet	M.2, P.2.13, A.3.2

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/6/17/

Zadania do wykonania w domu:

Średnia:

https://szkopul.edu.pl/problemset/problem/YYAbpdvNnjQKBQQEdhKHKR7W/site/?key=statement

Szuflady:

https://szkopul.edu.pl/problemset/problem/ERbofDxdUsKn2g8Tnwc25t6R/site/?key=statement

Krążki:

https://szkopul.edu.pl/problemset/problem/fYXVXOreVxlXTRoHZJXyXF2l/site/?key=statement

ZADANIA I ROZWIĄZANIA

Zadanie 1. Temperatury

Dostępna pamięć: 256MB

Bajtek na lekcji przyrody uczy się określać różne zjawiska pogodowe. Jednym z zadań, które otrzymał, jest codzienne notowanie temperatury. Bajtek rzetelnie zapisywał liczby na karteczce każdego dnia. Teraz przygląda się uzyskanym wynikom i zastanawia, ile razy i w które dni uzyskał określoną temperaturę. Pomożesz mu?

Wejście

W pierwszej linii wejścia znajduje się jedna liczba całkowita n ($1 \le n \le 10^3$), oznaczająca liczbę dni pomiarowych. W drugiej linii wejścia znajduje się n liczb całkowitych - zanotowanych w kolejnych dniach temperatur t_i ($-50 \le t_i \le 100$). W trzeciej linii znajduje się jedna liczba x ($-50 \le x \le 100$) - szukana temperatura.

Wyjście

Na wyjściu w jednej linii powinna znaleźć się liczba dni k, w które została zmierzona temperatura x, oraz k liczb oznaczających numery dni, w które zmierzono temperaturę x. Liczby należy podać w kolejności rosnącej i oddzielić pojedynczymi spacjami.

Przykład

Wejście	Wyjście
5 -2 0 1 -2 3 -2	2 1 4

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z przetwarzaniem i przeszukiwaniem tablic jednowymiarowych.

Wymuszone zostało dwukrotne przeszukanie tablicy: zliczenie wartości oraz wypisanie indeksów.

```
wczytaj n
dla i=1 do n wykonaj
     wczytaj a[i]
ile ← 0
wczytaj x
dla i=1 do n wykonaj
     jeżeli a[i] = x
          ile ← ile + 1
wypisz ile
dla i=1 do n wykonaj
     jeżeli a[i] = x
          wypisz i
```

Zadanie 2. Najlepsze sumy

Dostępna pamięć: 32MB. Na podstawie zadania cke.gov.pl

Najlepszą sumą ciągu liczb a_1, a_2, …, a_n nazywamy największą wartość wśród sum złożonych z sąsiednich elementów tego ciągu. Na przykład dla ciągu: 1, 2, -5, 7 mamy następujące sumy:

```
1
1+2 = 3
1+2+(-5) = -2
1+2+(-5)+7 = 5
2
2+(-5) = -3
2+(-5)+7 = 4
-5
-5+7 = 2
7
```

Zatem najlepszą sumą jest 7 (zwróć uwagę, że jeden element też uznajemy za sumę).

Zaproponuj algorytm wyznaczania najlepszej sumy dla dowolnego ciągu liczb całkowitych. Na jego podstawie napisz program do obliczenia najlepszych sum ciągów liczb.

Wejście

W jednej linii znajduje się ciąg liczb całkowitych zakończony liczbą 0. Liczb jest nie więcej niż 500000, ich wartość mieści się w przedziale [-1000, 1000].

Wyjście

Największa suma złożona z kolejnych elementów ciągu.

Przykład

Wejście	Wyjście
1 2 -5 7 0	7

Rozwiązanie

Zauważmy, że w naszym wypadku wynik nie będzie nigdy gorszy niż 0 kończące ciąg. Najprostszym rozwiązaniem wydaje się policzenie wszystkich możliwych sum i wybranie największej z nich. Niech k będzie liczbą elementów.

```
k ← 1
wykonaj
      k ← k + 1
      wczytaj a[k]
dopóki a[k] ≠ 0
maksimum ← 0
```

```
dla i=1 do k wykonaj
      suma ← 0
      dla j=i do k wykonaj
            suma ← suma + a[j]
            jeżeli suma > maksimum
                  maksimum ← suma
wypisz maksimum
```

Zadanie dla uczniów: Ile operacji sumowania wykona nasz program?

W pierwszym przebiegu zewnętrznej pętli wykona k dodawań, w drugim k – 1, w trzecim k – 2, … w k-tym – jedno. Łącznie więc zostanie wykonanych k + k – 1 + k – 2 + … + 1 = (k·(k+1)) div 2 operacji. Widać więc, że liczba operacji rośnie wraz z kwadratem rozmiaru danych wejściowych.

Rozwiązanie optymalne zakłada sumowanie kolejnych elementów dopóki się to „opłaca”, to znaczy dopóki suma częściowa jest większa od 0. W przeciwnym wypadku opłaca się pozbyć zmniejszającej końcowy wynik wartości i liczyć od nowa.

```
k ← 1
wykonaj
      k ← k + 1
      wczytaj a[k]
dopóki a[k] ≠ 0
maksimum ← 0
suma ← 0
dla i=1 do k wykonaj
      suma ← suma + a[i]
      jeżeli suma > maksimum
            maksimum ← k
      jeżeli suma < 0
            suma ← 0
wypisz maksimum
```

Zadanie dla uczniów: Ile operacji sumowania wykona nasz program?

Zadanie dla uczniów: Dokonaj analizy liczby wykonywanych operacji w zależności od rozmiaru danych wejściowych w dotychczas wykonanych zadaniach: Dzielniki, Ile podzielnych przez 3 i przez 5, Podstawowa arytmetyka oraz Najlepsze sumy.

Dla wnikliwych uczniów: Czy w ogóle była nam potrzebna tablica?

Zadanie 3. Bankiet

Dostępna pamięć: 32MB. I OIG (dostępne w serwisie szkopuł.edu.pl)

W restauracji Utalentowany Miś zaplanowano bankiet dla finalistów OIG. Goście zasiądą przy okrągłych stołach w ściśle określony sposób. Kierownik sali otrzymał listę gości wraz z informacją, kto ma siedzieć z lewej strony każdego z nich. Ile stołów musi przygotować na bankiet?

Zadanie

Opracuj program, który:

- wczyta ze standardowego wejścia informacje o rozmieszczeniu gości,
- obliczy ile stołów trzeba przygotować,
- wypisze wynik na standardowe wyjście.

Wejście

W pierwszym wierszu zapisano liczbę gości N ($1 \leq N \leq 30\,000$). Goście są ponumerowani kolejnymi liczbami naturalnymi od 1 do N. W drugim wierszu zapisano numer gościa siedzącego po lewej stronie pierwszego gościa. W trzecim wierszu zapisano numer gościa siedzącego po lewej stronie drugiego gościa itd. W i-tym wierszu zapisano numer gościa siedzącego po lewej stronie (i–1)-tego gościa. W N+1-szym wierszu zapisano numer gościa siedzącego po lewej stronie N-tego gościa.

Wyjście

W pierwszym wierszu wypisz liczbę stolików potrzebnych do usadzenia wszystkich gości.

Przykład

Wejście	Wyjście
12 4 10 7 3 2 6 1 5 11 8 12 9	4

Rozwiązanie

Otrzymaliśmy już gotową listę (rzeczywiste ustawienie), więc możemy założyć, że każde dzieci wskazało kogoś innego. Wystarczy więc przeprowadzić symulację kolejnych dzieci aż natrafimy na już usadzone (będziemy oznaczać je liczbą 0). Liczba symulacji to szukana liczba stołów. Algorytm wygląda więc następująco:

```
wczytaj n
dla i=1 do n wykonaj
     wczytaj a[i]
ile ← 0
dla i=1 do n wykonaj
     jeżeli a[i] ≠ 0
          ile ← ile + 1
          k ← a[i]
          dopóki k ≠ 0
```

```
                j ← k
                k ← a[j]
                a[j] ← 0
wypisz ile
```

Zadanie dla uczniów: Ile operacji sumowania wykona nasz program mimo „pętli w pętli”? Odpowiedź uzasadnij.

Zajęcia 6

Temat: Tablice 2

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem instrukcji warunkowej i iteracyjnej i tablic w C++,

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Pociąg	M.2, P.2.13, A.3.2
2. Bitib, Bajtjab i Palindrom	M.2, P.2.13, A.3.2
3. Odciski palca	M.2, P.2.13, A.3.2

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/6/17/

Zadania do wykonania w domu:

Żarówki (zliczanie):

https://szkopul.edu.pl/c/plo155/p/zar/25364/

Przyciski (VI OIG – zliczanie i maksimum):

https://szkopul.edu.pl/problemset/problem/aPqgk8oaUsM4nB6FLjxehRPe/site/?key=statement

Zapałki (V OIG, sumy prefiksowe od lewej i prawej):

https://szkopul.edu.pl/problemset/problem/ZLG7FB_afACLMh8-zsupw5zV/site/?key=statement

ZADANIA I ROZWIĄZANIA

Zadanie 1. Pociąg

Dostępna pamięć: 64MB

„Stoi na stacji lokomotywa,
Ciężka, ogromna i pot z niej spływa -
Tłusta oliwa.
[…]
A tych wagonów jest ze czterdzieści,
Sam nie wiem, co się w nich jeszcze mieści.
Lecz choćby przyszło tysiąc atletów
I każdy zjadłby tysiąc kotletów,
I każdy nie wiem jak się natężał,
To nie udźwigną - taki to ciężar!”

Julian Tuwim, Fragment wiersza ‘Lokomotywa’

Jaś pracuje na rampie kolejowej i przeładowuje wagony. Wiedząc, jak ciężkie mogą być wagony, postanowił zakupić specjalny dźwig. Może on podnieść dowolną liczbę wagonów jednocześnie, ważne, żeby stały one obok siebie. Niestety, nawet taki super-sprzęt posiada maksymalny udźwig. Jak duży ciężar musi podnieść dźwig Jasia?

Wejście

W pierwszym wierszu wejścia znajduje się liczba wagonów n pewnego pociągu ($2 \le n \le 10^6$). W drugiej linii znajduje się n liczb całkowitych w_i – ciężar każdego z wagonów ($1 \le w_i \le 10_6$). W trzeciej linii znajduje się jedna liczba całkowita k – liczba zestawów wagonów ($1 \le k \le 10^6$). W kolejnych k liniach znajdują się informacje o zestawach wagonów, które należy podnieść dźwigiem, odpowiednio numer pierwszego w_p i ostatniego wagonu w_k do podniesienia (należy jednocześnie podnieść wszystkie wagony od w_p do w_k włącznie).

Wyjście

Twój program powinien w k liniach ciężar każdego z zestawów wagonów.

Przykład

Wejście	Wyjście
5 3 5 4 5 4 2 2 3 3 5	9 13

Rozwiązanie

Najprostszym i najbardziej intuicyjnym rozwiązaniem wydaje się zliczanie ciężarów wagonów od pierwszego w_p do ostatniego wagonu w_k (dla uproszczenia zadania wagony numerujemy od 1).

```
wczytaj n
dla i=1 do n wykonaj
     wczytaj a[i]
wczytaj k
```

```
dla i=1 do k wykonaj
      wczytaj wp, wk
      suma ← 0
      dla j=wp do wk wykonaj
            suma ← suma + a[j]
      wypisz suma
```

Zadanie dla uczniów: Uzasadnij, że liczba operacji rośnie wraz z kwadratem danych wejściowych.

Rozwiązanie szybkie: Wykorzystajmy fakt, że łatwo jest już w trakcie wczytania policzyć, ile ważą wszystkie wagony od pierwszego do aktualnie wczytanego (wystarczy do dotychczasowej sumy dodać ciężar aktualnego wagonu). Przykład:

nr wagonu	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
ciężar wagonu:	0	1	1	2	1	1	2	1	1	1	2
suma ciężarów wszystkich wagonów od pierwszego do aktualnego:	0	1	2	4	5	6	8	9	10	11	13

Zauważ, że aby obliczyć ciężar wagonów od w_p do w_k wystarczy odjąć od sumy ciężarów wagonów do w_k sumę wagonów znajdujących się przed w_p.

```
wczytaj n
dla i=1 do n wykonaj
      wczytaj a[i]
      suma[i] ← suma[i-1] + a[i]
wczytaj k
dla i=1 do k wykonaj
      wczytaj wp, wk
      wypisz suma[wk] – suma[wp – 1]
```

Zadanie dla uczniów: Ile operacji dodawania i odejmowania w zależności od n i od k wykona nasz algorytm?

Zadanie 2. Bitib, Bajtjab i Palindrom

Dostępna pamięć: 64MB.

Bitib i Bajtjab lubią takie ciągi liczb, które obaj mogą czytać jednocześnie - Bitib od pierwszej do ostatniej, zaś Bajtjab od ostatniej do pierwszej, a mimo to czytają to samo. Czy liczby na kartce, którą właśnie znaleźli, można tak poprzestawiać, aby otrzymać ulubiony przez chłopców sposób ich ustawienia?

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba naturalna n nie większa niż milion. W następnej linii znajduje się n liczb naturalnych nie większych niż milion.

Wyjście

Jedyny wiersz wyjścia zawiera słowo „TAK” – jeśli z elementów ciągu można zbudować palindrom, lub „NIE” – jeśli nie jest to możliwe.

Przykład

Wejście	Wejście
5 15 15 81 18 81 Wyjście TAK Wyjaśnienie: Przykładowy możliwy do uzyskania palindrom: 81 15 18 15 81	6 15 15 15 15 18 81 Wyjście NIE

Rozwiązanie

Spostrzeżenie: Aby było możliwe czytanie liczb od początku i od końca w tym samym momencie, dana liczba musi wystąpić dokładnie dwa razy. Zatem jeżeli liczba występuje więcej niż dwa razy, musimy zaobserwować parzystą liczbę wystąpień. Jedyny wyjątek stanowi liczba o nieparzystej liczbie wystąpień w przypadku nieparzystej n.

Zatem musimy policzyć, ile razy wystąpiła każda z liczb. Zapamiętajmy też największą liczbę, jaka znalazła się na kartce (wykorzystujemy w drugiej pętli):

```
wczytaj n
największa ← 0
dla i=1 do n wykonaj
     wczytaj x
     ile_razy_wystąpiła[x] ← ile_razy_wystąpiła[x] + 1
     jeżeli x > największa
          największa ← x
ile_nieparzystych ← 0
dla i=1 do największa wykonaj
     jeżeli ile_razy_wystąpiła[i] mod 2 = 0
          ile_nieparzystych ← ile_nieparzystych + 1
jeżeli ile_nieparzystych > 1
     wypisz NIE
w przeciwnym wypadku
     wypisz TAK
```

Zadanie dla uczniów: Jaka jest złożoność obliczeniowa programu? Jakie warunki muszą być spełnione, by takie rozwiązanie było możliwe?

Zadanie 3. Odciski palca

Dostępna pamięć: 32MB

Pan Integer pracuje nad programem do odczytywania odcisków palców. Najwięcej uwagi poświęca modułowi odpowiedzialnemu za sprawdzanie, czy pewien kwadratowy

czteroelementowy fragment odnalezionego obrazu (zdjęty z przedmiotu odcisk) może być jakąś częścią dużego obrazu (pełnego wzoru). Zastanawia się przy tym, ile razy fragment pojawił się w jakiejkolwiek formie na dużym obrazie. Pomóż mu to obliczyć!

Wejście

W pierwszej i drugiej linii wejścia znajdują się po dwie dwucyfrowe liczby pierwsze – odnaleziony fragment obrazu. W drugiej linii wejścia znajdują się dwie liczby całkowite h oraz w (1 ≤ h, w ≤ 1000), odpowiednio wysokość i szerokość dużego obrazu.

W kolejnych h liniach znajduje się po w rozdzielonych spacją dwucyfrowych liczb pierwszych.

Wyjście

Liczba wystąpień dowolnej permutacji fragmentu obrazu we wzorze.

Przykład

Wejście	Wejście	Wejście
11 13 17 19 2 3 23 11 13 29 17 19	11 13 17 19 3 4 23 11 13 17 29 17 19 11 11 17 17 13	11 13 17 19 1 4 11 13 17 19
Wyjście 1	Wyjście 3	Wyjście 0

Rozwiązanie

Zadanie dla uczniów: Uzasadnij, że iloczyn czterech dowolnych liczb pierwszych (niekoniecznie różnych) będzie unikatowy (to znaczy, że nie można znaleźć innej czwórki liczb pierwszych, której iloczyn miał taką samą wartość).

Korzystając z powyższego spostrzeżenia wystarczy sprawdzać kolejne kwadratowe obszary, czy ich iloczyn jest równy iloczynowi wzorcowej czwórki.

```
wczytaj p1, p2
wczytaj p3, p4
palec ← p1 ·p2 · p3 · p4
wczytaj h, w
dla y=1 do h wykonaj
     dla x=1 do w wykonaj
          wczytaj a[x][y]
wynik ← 0
dla y=1 do h-1 wykonaj
     dla x=1 do w-1 wykonaj
     jeżeli palec = a[x][y] · a[x][y+1] · a[x+1][y] · a[x+1][y+1]
          wynik ← wynik + 1
wypisz wynik
```

Zajęcia 7

Temat: Łańcuchy znaków (proste)

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, łańcuchy znaków, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem tablic ii łańcuchów znaków w C++,
- potrafi zaimplementować własną arytmetykę dużych liczb.
- zna algorytm przeszukiwania tekstu,
- potrafi dokonać zamiany pomiędzy różnymi systemami liczbowymi,
- zna schemat Hornera

Formy i metody pracy: praca samodzielna,

Zadania do wykonania na zajęciach	Treści programowe
1. Brakująca cyfra	M.3, P.2.15, A.3.3
2. Ciąg monotoniczny	M.3, P.2.15, A.3.2
3. System 36	M.3, P.2.15, A.3.1, A.3.2
4. James i potęgi dwójki	M.3, P.2.15, A.3.2

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/6/17/

Zadania do wykonania w domu:

Statystyki (OIG):

https://szkopul.edu.pl/problemset/problem/XAJ_VTHmix0ioDPo5ygaq1sc/site/?key=statement

Sumujący Jaś (OIG):

https://szkopul.edu.pl/problemset/problem/Tczhl-p0p4d8QI5QKSByWTME/site/?key=statement

ZADANIA I ROZWIĄZANIA

Zadanie 1. Brakująca cyfra

Dostępna pamięć: 64MB

Mamusia ciągle powtarzała Jankowi: ‘Odrabiając pracę domową z matematyki nie pij nad zeszytem mleka!’. Janek oczywiście nie słuchał mamy i mleko nad zeszytem pił. Aż do czasu, kiedy mleko rozlało się na zadanie domowe! Całe szczęście, że rozmyła się tylko jedna cyfra w całej liczbie! Janek pamięta, że każda liczba na kartce była podzielna przez 9. Twoim zadaniem jest odnalezienie brakującej cyfry. Jeśli warunek spełnia kilka cyfr, pomóż odnaleźć Jankowi najmniejszą z nich.

Wejście

Pierwszy wiersz wejścia zawiera jedną liczbę k z pracy domowej($1 \le k \le 10^{1000}$). Zagubiona cyfra zastąpiona jest znakiem ‘X’. Możesz przyjąć, że dla 40% testów zachodzi warunek $k \le 10^{18}$.

Wyjście

Na wyjściu wypisz brakującą cyfrę.

Przykład

Wejście	Wyjście
23X54	4

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z obsługą łańcuchów znaków (typ char, string, rzutowanie typów).

Zauważmy, że liczba jest podzielna przez 9, jeżeli suma jej cyfr jest podzielna przez 9. Naszym zadaniem jest zatem obliczenie sumy cyfr bez X i wypisanie brakującej wartości do 9. Wyjątek stanowią sytuacje, gdy suma jest równa 9. Wówczas, gdy X umieszczono na pierwszym miejscu, wypiszemy 9, w każdym innym wypadku 0.

```
wczytaj s
suma ← 0
dla i=0 do długość(s)-1 wykonaj
     jeżeli s[i] ≠ 'X'
          suma ← suma + s[i] - 48
brakująca_cyfra ← 9 - suma mod 9
jeżeli brakująca_cyfra = 9 i s[0] ≠'X'
     brakująca_cyfra ← 0
wypisz brakująca_cyfra
```

Zadanie 2. Ciąg monotoniczny

Dostępna pamięć: 32MB

Ciąg liczbowy nazywamy monotonicznym jeżeli jest rosnący, albo malejący, albo stały.

W naszym zadaniu dany jest ciąg liter. Jakie litery należy w nim zmienić, aby nasz ciąg był stały, zastępując przy tym jak najmniej znaków? Jeżeli jest kilka takich znaków, które można pozostawić, nie zmieniaj alfabetycznie największej z nich. Możesz założyć, że zawsze będzie co najmniej jedna litera do zmiany.

Wejście

W pierwszej linii wejścia znajduje się ciąg co najmniej dwóch liter (wyłącznie małe litery alfabetu łacińskiego) nie dłuższy niż 10^6 elementów.

Wejście

Na wyjściu w pierwszym wierszu wypisz alfabetycznie wszystkie litery do zmiany.

Przykład

Wejście	Wyjście
kajak	aaj

Wyjaśnienie: Należy pozostawić litery ‘k’.

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z obsługą łańcuchów znaków (typ char, string, rzutowanie typów).

W zadaniu należy zliczyć wystąpienia liter, odszukać ostatnią najczęściej występującą i wypisać każdą literę zgodnie z liczbą jej wystąpień (patrz algorytm: sortowanie przez zliczanie).

```
wczytaj s
dla i=0 do długość(s) wykonaj
     ile_wystąpień[ s[i] ] ← ile_ wystąpień[ s[i] ] + 1
najczęściej_wyst ← 0
dla i=’a’ do ’z’ wykonaj
     jeżeli ile_wystąpień[i] ≥ ile_wystąpień[ najczęściej_wyst ]
          najczęściej_wyst ← i
dla i=’a’ do ’z’ wykonaj
     jeżeli najczęściej_wyst ≠ i
          dla j=0 do ile_wystąpień[i]
               wypisz znak i
```

Zadanie 3. System 36

Dostępna pamięć: 32MB

Bitek sprawdza przydatność obliczeń w różnych systemach obliczeń. Stwierdził, że korzystając z cyfr i dużych liter alfabetu łacińskiego może korzystać nawet z systemu trzydziestoszóstkowego! Próbuje teraz napisać program, który szybko przeliczałby wartości pomiędzy dowolnymi systemami liczbowymi.

Wejście

Pierwszy wiersz zawiera trzy liczby: X, Y i Z, gdzie X jest liczbą zapisaną w systemie o podstawie Y (wartość X nie przekracza 10^{15} w systemie dziesiętnym). Z jest podstawą systemu, na który należy zamienić liczbę X ($1 < Y, Z < 37$).

Wyjście

Jedna liczba całkowita - zapis X w systemie o podstawie Z.

Przykład

Wejście	Wyjście
`37826876 10 36`	`MIREK`

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z zamianą liczb pomiędzy poszczególnymi systematami pozycyjnymi oraz schematu Hornera.

Najprostszym rozwiązaniem jest zamiana wskazanego tekstu zapisanego w systemie o podstawie Y na jego reprezentacją w systemie dziesiętnym (z użyciem schematu Hornera).

```
wczytaj X, Y, Z
dziesiętnie ← 0
dla i=0 do długość(X) wykonaj
      dziesiętnie ← dziesiętnie ·Y + s[i]-48
```

Następnie należy zamienić otrzymaną wartość na system o podstawie Z (zmienna tekstowa `wynik`). Zakładamy, że instrukcja `(znak) wartość` rzutuje wartość całkowitoliczbową na jej znak w kodzie ASCII. Łączenie tekstów pozwala na umieszczanie nowo obliczonych cyfr na początku zapisu liczby.

```
wynik ← ""
dopóki dziesiętnie ≠ 0 wykonuj
     jeżeli dziesiętnie mod Z < 10
          c ← (znak) (dziesiętnie mod Z + '0')
     w przeciwnym wypadku
          c ← (znak) (dziesiętnie mod Z + 'A')
     wynik ← c + wynik
     dziesiętnie ← dziesiętnie div Z
wypisz wynik
```

Zadanie 4. James i potęgi dwójki

Dostępna pamięć: 64MB

To był już prawie koniec misji. James Blond rozejrzał się uważnie po pokoju. Pomiędzy porozrzucanymi kartkami znajdowała się TA JEDNA, poszukiwana przez wszystkich. „Jak mogli jej nie zauważyć?"- zapytał sam siebie. Schylił się i wszystko zrozumiał. Na przedartej stronie widać było krótkie ciągi liczb. Ich końcowe cyfry musiały znajdować się na brakującej większej części. W panice zaczął przegarniać papiery. „Nie ma! Nie ma!" – krzyczał do siebie bezgłośnie. Usiadł bezsilnie wpatrując się ciągi znaków. I wtedy… przypomniał sobie ostatnie słowa Profesora: „Tylko potęgi dwójki!". A więc o to mu chodziło! Każda z liczb to potęga dwójki! Wystarczy znaleźć najmniejszą potęgę dwójki, której początkowe cyfry widział na kartce! James wyjął swój smartphone, włączył aplikację kalkulatora w widoku programisty i… Nie mógł uwierzyć! Android się zawiesił! Czy coś mu jeszcze może pomóc?

Zadanie

Napisz program, który dla zadanej dodatniej liczby całkowitej n wyznaczy najmniejszy wykładnik X taki, że pierwsze cyfry liczby 2^X zgadzają się z zadaną liczbą. Pamiętaj, że oderwano większą część kartki, więc na pewno brakuje ponad połowy cyfr.

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba całkowita bez znaku n ($1 \le n < 10^{50}$).

Wyjście

Dla podanej n wyznacz wykładnik X taki, że pierwsze cyfry liczby 2^X zgadzają się z zadaną liczbą. Możesz założyć, że wynik zawsze będzie istniał i $2^X \le 10^{100}$. Możesz założyć, że dla 40% testów zachodzi warunek $2^X < 10^{18}$.

Przykład

Wejście 1 Wyjście 7	Wejście 5 Wyjście 9	Wejście 16 Wyjście 14

Rozwiązanie

Rozwiązanie zadania wymaga zaimplementowania własnej arytmetyki dużych liczb (dodawania), rzutowania typów i przeszukiwania tekstu.

W zadaniu liczymy kolejne potęgi dwójki. Będziemy więc po prostu dodawać je do siebie i naiwnie sprawdzać zgodność kolejnych znaków, stąd liczbę n wczytujemy jako tekst. Zauważmy, że najmniejszą potęgą, którą możemy wypisać, jest 2^7 (musi to być liczba co najmniej 3-cyfrowa w zapisie dziesiętnym). Wygodnie będzie więc zacząć od 2^6. Zakładamy, że instrukcja `(znak) wartość` rzutuje wartość całkowitoliczbową na jej znak w kodzie ASCII.

```
wczytaj n
potęga ← ”64”
X ← 6
wartownik ← 0
```

```
dopóki wartownik = 0 wykonuj
      // obliczamy nową potęgę
      // dodając do siebie cyfry począwszy od ostatniej
      X ← X + 1
      c ← 0 // kolejne wyznaczane cyfry
      dla i=długość(n)-1 do 0
            c ← 2 ·(potęga[i]-48) + c
            potęga [i] ← (znak) (c mod 10 + ’0’)
            c  ← c div 10
      // po dodaniu wszystkich cyfr został nam bit przeniesienia
      jeżeli c > 0
            potęga ← ’1’ + potęga
      //sprawdzamy zgodność początkowych cyfr
      jeżeli długość(potęga) > 2 ·długość(n)
            wartownik ← 1
            dla i=0 do długość(n)-1 wykonuj
                  jeżeli n[i] ≠ potęga[i]
                        wartownik ← 0
wypisz X
```

Zajęcia 8

Temat: Funkcje 1

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, łańcuchy znaków, algorytm Euklidesa, funkcje, NWD, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie napisać program z wykorzystaniem instrukcji warunkowej i iteracyjnej,
- zna algorytm Euklidesa,
- umie pisać własne funkcje,
- zna zasadę podzielności, NWD,

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Suma ułamków	M.2, P.2.16, A.3.1
2. Liczby super-B-pierwsze	M.2, P.2.16, A.3.1
3. Trzy liczby rosnąco	M.2, P.2.16,

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/6/21/

Zadania do wykonania w domu:

Odwróć i dodaj (przygotowana paczka do SIO2)

Liczby zaprzyjaźnione (przygotowana paczka do SIO2)

Doskonałość (przygotowana paczka do SIO2)

ZADANIA I ROZWIĄZANIA

Zadanie 1. Suma ułamków

Limit pamięci: 64MB

Napisz program, który czyta dwa ułamki zwykłe, a następnie wypisuje ułamek będący ich sumą.

Wejście

Dwa ułamki zapisane w postaci czterech liczb oddzielonych pojedynczym odstępem (odpowiednio licznik i mianownik pierwszej oraz licznik i mianownik drugiej liczby). Liczniki i mianowniki są liczbami naturalnymi nieprzekraczającymi miliarda.

Wyjście

Wypisywany ułamek (suma) w postaci licznika i mianownika przedzielonego znakiem /. Ułamek powinien być zapisany za pomocą liczb względnie pierwszych.

Przykład

Wejście
`2 3 5 7`
 	
Wyjście
`29/21`
	Wejście
	`1 4 7 4`
	
	Wyjście
	`2/1`

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z podzielnością, NWD, algorytmem Euklidesa oraz z funkcjami, parametrami funkcji i zasięgiem zmiennych.

W zadaniu korzystamy z podstawowych umiejętności związanych z obliczaniem sumy ułamków. Dla otrzymanych licznika i mianownika należy policzyć NWD (by skrócić ułamek).

W tym celu napiszemy własną funkcję NWD korzystając z algorytmu Euklidesa: dopóki dwie liczby nie są sobie równe, od większej odejmuj mniejszą; uzyskana końcowa wartość to NWD.

```
funkcja nwd (a, b)
      dopóki a ≠ b
            jeżeli a > b
                  a ← a − b
            w przeciwnym wypadku
                  b ← b − a
      zwróć a
```

Zadanie dla uczniów: Ile operacji musi wykonać podana funkcja w skrajnym (pesymistycznym) wypadku? Podaj taki przypadek.

Spostrzeżenie: przy dużej różnicy wartości między a i b wielokrotne odejmowanie liczby mniejszej można zastąpić operacją `mod`.

```
funkcja nwd (a, b)
      dopóki b ≠ 0
            r ← a mod b
            a ← b
            b ← r
      zwróć a
```

Nasz program wygląda więc następująco:

```
wczytaj l1, m1, l2, m2
l ← l1·m2+l2·m1
m ← m2·m1
wypisz l/nwd(l,m), '/', m/nwd(l,m)
```

Zadanie 2. Liczby super-B-pierwsze

Limit pamięci: 64MB. Źródło: cke.gov.pl

Liczba „super-B-pierwsza” to taka liczba naturalna, która spełnia następujące warunki:

- jest liczbą pierwszą,
- suma cyfr tej liczby jest liczbą pierwszą,
- suma cyfr w jej zapisie binarnym jest liczbą pierwszą.

Oblicz, ile jest liczb „super-B-pierwszych” w przedziale [a, b].

Wejście

Pierwszy wiersz danych zawiera dwie liczby całkowite a i b ($2 \le a < b \le 1\,000\,000$).

Wyjście

Program powinien wypisać ilość liczb „super-B-pierwszych” w zadanym przedziale.

Przykład

Wejście	Wejście
2 100	8

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z testowaniem pierwszości liczb oraz przypomnienie związane z systemami liczbowymi oraz z funkcjami i parametrami funkcji.

Napiszmy funkcję sprawdzającą pierwszość liczby x (brak dzielnika mniejszego lub równego √x poza 1). Pamiętajmy, by funkcję matematyczną pierwiastek(n) zastąpić w poniższym zapisie podnosząc strony nierówności do kwadratu (rozwiązanie wzorcowe w C++).

```
funkcja pierwsza (x)
      dla i=2 do √n wykonaj
            jeżeli x mod i = 0
                  zwróć FAŁSZ
      zwróć PRAWDA
```

Potrzebujemy również funkcji zliczających sumę cyfr w zapisie dziesiętnym oraz binarnym. Zauważmy, że w każdym wypadku wykonujemy te same operacje zmieniając jedynie dzielnik, którym jest podstawa systemu (10 lub 2). Możemy zatem napisać jedną funkcję dla obu tych operacji:

```
funkcja suma_cyfr (x, podstawa)
      suma ← 0
      dopóki x ≠ 0
            suma ← suma + x mod podstawa
            x ← x div podstawa
      zwróć suma
```

Możemy teraz sprawdzić kolejne liczby w przedziale i wypisać wynik na wyjście:

```
wczytaj a, b
ile ← 0
dla k=a do b wykonaj
      jeżeli   pierwsza( suma_cyfr(k,10) )
             i pierwsza( suma_cyfr(k,2) )
             i pierwsza (k)
           ile ← ile + 1
wypisz ile
```

Zadanie dla uczniów: Jakie znaczenie dla liczby wykonanych operacji w zadaniu ma kolejność wywoływanych funkcji w instrukcji `jeżeli`?

Zadanie 3. Trzy liczby rosnąco

Limit pamięci: 64MB.

Napisz program, który czyta trzy liczby całkowite, a następnie wypisuje je w kolejności niemalejącej.

Wejście

Dane wejściowe zawierają trzy liczby oddzielone pojedynczym odstępem. Dane są dodatnimi liczbami całkowitymi nieprzekraczającymi miliona.

Wyjście

Trzy liczby.

Przykład

Wejście	Wejście
7 5 3	3 5 7

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z funkcjami i parametrami funkcji.

Napiszmy procedurę zamieniającą ze sobą wartościami dwie liczby, jeżeli druga liczba jest mniejsza od pierwszej (w C++ użyjemy dla funkcji typu void, parametry przekażemy przez referencję):

```
procedura zamień (&a, &b)
     jeżeli b < a
          pom ← a
          a ← b
          b ← pom
```

Teraz dla naszych trzech liczb dokonamy trzech prostych zamian.

```
wczytaj, x, y, z
zamień (x, y)
zamień (y, z)
zamień (x, y)
```

Zadanie dla uczniów: Jak moglibyśmy wykorzystać powyższą funkcję dla ciągu liczb?

Zajęcia 9

Temat: Funkcje 2

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, algorytm Euklidesa, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem funkcji rekurencyjnej, równaniami rekurencyjnymi,
- zna algorytm Euklidesa,

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Struś pędziwiatr	M.2, P.2.16, P.2.17, A.3.1
2. Liczby parzystocyfrowe	M.2, P.2.16, P.2.17, A.3.2
3. Drzewo Sterna-Brocota	M.2, P.2.16, P.2.17, A.3.2

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/6/21/

https://www.main2.edu.pl/main2/courses/show/7/8/

Zadania do wykonania w domu:

Szybkie potęgowanie (przygotowana paczka do SIO2)

ZADANIA I ROZWIĄZANIA

Zadanie 1. Struś pędziwiatr

Limit pamięci: 64MB

Struś Pędziwiatr porusza się z niesamowitą prędkością. Jest w stanie całe miasto przemierzyć w mniej niż jedną sekundę. Na przeszkodzie stoją mu jednak sygnalizatory świetlne. Nie są ze sobą zsynchronizowane, przez co Struś wciąż rusza i wciąż się zatrzymuje. Każde skrzyżowanie w mieście działa w ten sam sposób: przez x_i sekund na i-tym skrzyżowaniu zapalone jest czerwone, a następnie przez jedną sekundę zielone światło. Struś zauważył, że zdarzają się takie momenty, kiedy wszystkie światła w mieście świecą na zielono. Ile sekund w najgorszym przypadku musi czekać na taki moment Struś?

Wejście

W pierwszym wierszu wejścia znajduje się liczba całkowita n ($1 < n < 10$) – liczba skrzyżowań w mieście. W kolejnych n liniach znajduje się po jednej liczbie całkowitej x_i – czas trwania zmiany czerwonych świateł na i-tym skrzyżowaniu ($1 < x_i < 50$). Możesz założyć, że każde ze skrzyżowań działa z inną częstotliwością zmian świateł.

Wyjście

Jedna liczba całkowita – najmniejszy czas, jaki upłynie pomiędzy dwoma momentami, kiedy zielone światła zapalą się jednocześnie na wszystkich skrzyżowaniach.

Przykład

Wejście	Wejście
2 3 5	11

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z algorytmem Euklidesa, funkcjami rekurencyjnymi oraz definicjami i równaniami rekurencyjnymi.

Warto zauważyć, że pełna zmiana świateł na i-tym skrzyżowaniu trwa x_i+1 sekund. Skoro światła się zmieniają cyklicznie i zają się takie momenty, kiedy wszystkie światła w mieście świecą na zielono, to znaczy że pełny cykl musi trwać NWW ze wszystkich czasów x_i+1. Przez ostatnią sekundę wszystkie światła będą świeciły się na zielono.

Spróbujmy w naszym zadaniu zdefiniować funkcję NWD (wykorzystywaną do wyznaczenia NWW) rekurencyjnie. Skorzystajmy ze wzoru:

$$nwd(a,b) = \begin{cases} a \; dla \; b = 0 \\ nwd(b, a \bmod b) \; dla \; b \neq 0 \end{cases}$$

Funkcja NWD rekurencyjnie:

```
funkcja nwd (a, b)
     jeżeli b = 0
          zwróć a
```

```
    nwd (b, a mod b)
```

Teraz wystarczy obliczyć NWW dla wszystkich wartości z zadania.

```
wczytaj n
wczytaj x
wynik ← x + 1
dla i=1 do n-1 wykonaj
    wczytaj x
    wynik ← (x+1)·wynik / nwd(x+a,wynik)
zwróć wynik – 1
```

Zadanie 2. Liczby parzystocyfrowe

Limit pamięci: 256MB. Źródło: OIJ (szkopuł.edu.pl)

Dodatnią liczbę całkowitą nazywamy parzystocyfrową, jeśli wszystkie jej cyfry są parzyste. Na przykład: liczby 6, 42, 2020 są parzystocyfrowe, zaś 7, 34, 2019 lub 13 579 nie są. Gdyby wszystkie liczby parzystocyfrowe ustawić w kolejności rosnącej, która liczba byłaby N-ta w tym porządku?

Napisz program, który: wczyta liczbę naturalną N, wyznaczy N-tą liczbę parzystocyfrową i wypisze wynik na standardowe wyjście.

Wejście

W pierwszym (jedynym) wierszu wejścia znajduje się jedna liczba naturalna N. Liczba będzie równa co najmniej 1 i co najwyżej 10^{18}.

Wyjście

W pierwszym (jedynym) wierszu wyjścia powinna się znaleźć jedna liczba naturalna – N-ta liczba parzystocyfrowa.

Przykład

Wejście	Wejście
12	7921
Wejście	Wejście
44	446282

Rozwiązanie

Omówienie rozwiązania: https://oij.edu.pl/oij14/zadania/par/parroz.pdf

Zadanie można w prosty sposób rozwiązać rekurencyjnie:

```
procedura dziesiętny_na_piątkowy (x)
    jeżeli x ≥ 5
        dziesiętny_na_piątkowy (x div 2)
wypisz (x mod 5) · 2
```

Zadanie dla uczniów: Jakie znaczenie ma rekurencyjne wywołanie funkcji przed lub po wypisaniu kolejnej cyfry?

Główna część programu wygląda więc następująco:

```
wczytaj n
procedura dziesiętny_na_piątkowy (n)
```

Zadanie 3. Drzewo Sterna-Brocota

Limit pamięci: 128MB

Drzewo Sterna-Brocota to drzewo binarne zawierające wszystkie dodatnie ułamki nieskracalne. Własności (według Wikipedii):

- W drzewie występują wszystkie dodatnie liczby wymierne zapisane jako ułamki nieskracalne.
- Jeśli liczby a oraz b są względnie pierwsze, to ułamek $\frac{a}{b}$ występuje w drzewie dokładnie raz.

Zaczynamy od $\frac{0}{1}$ - symbolizującego zero i $\frac{1}{0}$ symbolizującego nieskończoność. Następnie na kolejnych piętrach drzewa wpisujemy „pomiędzy" wartości $\frac{a}{b}$ oraz $\frac{c}{d}$ wartość $\frac{a+c}{b+d}$.

Zatem w pierwszym kroku mamy: $\frac{0}{1}\ \frac{1}{1}\ \frac{1}{0}$

W drugim kroku: $\frac{0}{1}\ \frac{1}{2}\ \frac{1}{1}\ \frac{2}{1}\ \frac{1}{0}$

Zaś w trzecim: $\frac{0}{1}\ \frac{1}{3}\ \frac{1}{2}\ \frac{2}{3}\ \frac{1}{1}\ \frac{3}{2}\ \frac{2}{1}\ \frac{3}{1}\ \frac{1}{0}$

Napisz program, który czyta liczbę naturalną n i wypisuje sekwencję ułamków odpowiadającą temu numerowi iteracji.

Wejście

Jedyny wiersz danych zawiera liczbę całkowitą n (0 ≤ n ≤ 20) – numer iteracji.

Wyjście

Program powinien wypisać wiersz tekstu zawierający sekwencję ułamków (zapisanych z użyciem znaku „/") oddzielonych pojedynczymi odstępami.

Przykład

Wejście
4
Wejście
0/1 1/4 1/3 2/5 1/2 3/5 2/3 3/4 1/1 4/3 3/2 5/3 2/1 5/2 3/1 4/1 1/0

Rozwiązanie

Rozwiązanie zadania wymaga omówienia podstawowych informacji związanych z funkcjami rekurencyjnymi, parametrami funkcji oraz zasięgiem zmiennych.

Wyznaczmy rozmiar drzewa w n-tej iteracji. Zauważmy, że przed pierwsza iteracją mamy 2 wyrazy. W każdym kolejnym kroku pomiędzy wszystkie dotychczasowe wyrazy wstawiamy

o jeden mniej wyraz: 0 – 2 wyrazy. 1 – 2+1 wyrazy, 2 – 3+2 wyrazy, 3 – 5+4 wyrazy, 4 – 9+8 wyrazów. Łatwo zauważyć, że dla n>0 mamy 2^n+1 wyrazów.

Zauważmy, że $a^n=a^{n-1}\cdot a$, możemy więc obliczyć potęgę ze wzoru rekurekurencyjnego:

$$potęga(a,n)=\begin{cases}1\ dla\ n=0\\ potęga(a,n-1)\cdot a\ dla\ n\neq 0\end{cases}$$

Funkcja potęga rekurencyjnie:

```
funkcja potęga (a, n)
     jeżeli n = 0
          zwróć 1
     zwróć potęga(a, n-1)·a
```

Jak wyznaczyć wartość „środka” pomiędzy dwoma elementami tablicy?

```
środek ← (lewy+prawy)/2
licznik[środek] ← licznik[lewy]+licznik[prawy]
mianownik[środek] ← mianownik[lewy]+ mianownik[prawy]
```

Teraz wystarczy wyznaczyć rekurencyjnie wszystkie „środki” dla wszystkich zadanych wyrazów pomiędzy wskazanym lewym i prawym elementem. Kolejne „środki” będziemy wyznaczać tak długo, dopóki pomiędzy lewym i prawym elementem pozostanie jeszcze co najmniej jeden wyraz wolny. Dla uproszczenia zakładamy, że tablice licznik i mianownik zostały zdefiniowane jako zmienne globalne.

```
procedura dsb (lewy, prawy)
środek ← (lewy+prawy)/2
licznik[środek] ← licznik[lewy]+licznik[prawy]
mianownik[środek] ← mianownik[lewy]+ mianownik[prawy]
jeżeli prawy-lewy>1
     dsb(lewy, środek)
     dsb(środek, prawy)
```

Główna część programu:

```
wczytaj n
rozmiar ← potęga(2,n)
licznik[0] ← mianownik[rozmiar] ← 0
licznik[rozmiar] ← mianownik[0] ← 1
dsb(0,rozmiar)
dla i=0 do n wykonuj
     wypisz licznik[i], '/', mianownik[i]
```

Dla wnikliwego ucznia: czym różnią się tablice licznik i mianownik? Jak znacząco można oszczędzić pamięć?

Zajęcia 10

Temat: Zawody 1. Powtórzenie i podsumowanie. Sparing.

Czas trwania: 2X45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem instrukcji warunkowej i iteracyjnej i tablic, rekurencji, własnych funkcji w C++,
- zna podstawowe algorytmy: Euklidesa, NWD

Formy i metody pracy: praca samodzielna w formie zawodów, sparingu

Zadania do wykonania na zajęciach	Treści programowe
1. Cyfry 2356	M.2, P.2.10, A.1
2. Podatki	M.2, P.2.10, A.1
3. Pogotowie lotnicze	M.2, P.2.13, A.3.2
4. Kolorowe żaróweczki	M.2, P.2.13, A.3.2

Podpowiedzi do rozwiązań:

Zadanie 1. Znaleźć najmniejszą liczebność cyfr 2, 5 i 6 (liczba liczb 256). Pozostałe 2 i 3 to liczba liczb 32 (instrukcja jeżeli)

Zadanie 2. Należy wykorzystać hipotezę Goldbacha. Wówczas łatwo zauważyć, że wynik to 1, 2 lub 3.

Zadanie 3. Sumy prefiksowe. Wskazujemy jedno lub dwa środkowe miasta.

Zadanie 4. Jeden ze sposobów: Musimy pamiętać pierwsze, bieżące oraz ostatnie wystąpienie każdej z liczb oraz ostatnią minimalną odległość każdej z liczb do poprzedniego wystąpienia.

ZADANIA

Zadanie 1. Cyfry 2356

Dostępna pamięć: 32MB

Pan Integer znalazł w swoim pokoju pudełko z cyframi. W pudełku jest k_2 cyfr 2, k_3 cyfr 3, k_5 cyfr 5 i k_6 cyfr 6.

Ulubionymi liczbami całkowitymi pana Integera są 32 i 256. Postanowił poskładać te liczby z cyfr znajdujących się w pudełku. Każda cyfra może być użyta nie więcej niż jeden raz, tzn. wszystkie liczby powinny zawierać nie więcej niż k_2 cyfr 2, k_3 cyfr 3 i tak dalej. Chce przy tym, aby suma uzyskanych liczb była jak największa. Cyfry nieużywane nie są doliczane do sumy. Pomóż mu rozwiązać to zadanie!

Wejście

Jedyna linia wejścia zawiera cztery liczby całkowite k_2, k_3, k_5 i k_6 - odpowiednio liczbę cyfr 2, 3, 5 i 6 ($0 \le k_2, k_3, k_5, k_6 \le 5 \cdot 10^6$)$.

Wyjście

Wypisz jedną liczbę całkowitą - maksymalną możliwą sumę ulubionych liczb całkowitych pana Integera, które można uzyskać za pomocą cyfr z pudełka.

Przykład

Wejście	Wejście
5 1 3 4 Wejście 800	1 1 1 1 Wejście 256

Wyjaśnienie do przykładu pierwszego: Mamy pięć cyfr 2, jedną 3, trzy 5 i cztery 6. Pan Integer może złożyć 3 liczby 256 i jedną 32, co daje $3 \cdot 256 + 32 = 800$.

Zadanie 2. Podatki

Dostępna pamięć: 64MB

Pan Longint zamieszkał w Bitolandii. Bardzo lubi tamtejsze widoki. Bitolandia to kraj o bardzo dziwnym systemie podatkowym. Kwota podatku, jaką musi zapłacić pan Longint od umowy opiewającej na n bitów (miejscowa waluta), jest równa maksymalnemu dzielnikowi właściwemu n. Na przykład dla trzech umów odpowiednio na 25, 6 i 2 bity należy zapłacić 9 bitów (5 bitów dla umowy na 25 bitów, 3 dla 6 i 1 dla 2 bitów).

Pan Longint dostrzegł jednak pewną lukę w przepisach. Otóż może on podzielić umowę na kilka różnych (mniejszych) umów i (być może) zapłacić mniejszy podatek opodatkowując każdą umowę oddzielnie! Czy mu się to opłaca?

Oblicz, jaki łączny (możliwie najmniejszy) podatek dla podanej kwoty zapłaci pan Longint!

Wejście

Pierwszy i jedyny wiersz wejścia zawiera jedną liczbę n ($2 \le n \le 10^{10}$) – kwota, na jaką opiewa umowa.

Wyjście

Wypisz jedną liczbę całkowitą – najniższy podatek, jaki może zapłacić Longint.

Przykład

Wejście	Wejście
4 Wejście 2	27 Wejście 3

Zadanie 3. Pogotowie lotnicze

Dostępna pamięć: 64MB

Bajtazar, król Bitolandii, chce podwyższyć poziom bezpieczeństwa w państwie. Postanowił powołać do życia lotnicze pogotowie ratunkowe. Ma nadzieję, że w ten sposób lekarze będą mogli śmigłowcami szybko dotrzeć do każdego mieszkańca. To na pewno podniesie poczucie bezpieczeństwa ludności.

Bitolandia to ciekawy kraj. Całe państwo przecina jedna prosta droga, a wszystkie miasta położone są wzdłuż tej drogi. Król Bajtazar chciałby umieścić bazę lotniczego pogotowia w takim mieście, żeby w razie nagłej potrzeby można było śmigłowcem jak najszybciej dolecieć w dowolne miejsce w kraju.

Pomóż mu i wskaż wszystkie miasta, które się do tego nadają!

Wejście

W pierwszym wierszu wejścia znajduje się liczba miast w Bitolandii n ($1 \le n \le 10^6$). W drugiej linii wejścia znajduje się n-1 liczb całkowitych x_i $(1 \le x_i \le 10^6)$$, gdzie x_i oznacza odległość między miastami o numerze i oraz i+1.

Wyjście

W jednej linii wypisz w kolejności rosnącej wszystkie miasta, w których można umieścić bazę lotniczego pogotowia ratunkowego.

Przykład

Wejście	Wejście
7 2 3 2 2 4 1	4

Zadanie 4. Kolorowe żaróweczki

Dostępna pamięć: 64MB

W Bitocji mieszkańcy lubią ozdabiać domy kolorowymi żaróweczkami. Tworzą z nich różne wzory, ale zawsze dbają o to, by żarówki znajdowały się w jednej linii obok siebie. Czasem trzeba dokonywać napraw i wówczas ważne jest, by żarówki w tym samym kolorze

znajdowały się blisko siebie. Każdy mieszkaniec chce więc wiedzieć, w jakiej odległości żarówki tego samego koloru są najbliżej oraz najdalej siebie.

Wejście

W pierwszym wierszu wejścia znajduje się liczba n ($1 \le n \le 10^6$). W drugiej linii znajduje się n liczb całkowitych k_i – kolory żaróweczek ($1 \le k_i \le n$).

Wyjście

Twój program powinien zapisać dwie liczby całkowite oznaczające odpowiednio najmniejszą odległość pomiędzy dwoma żaróweczkami dowolnego ale tego samego koloru oraz największą możliwą odległość pomiędzy dwoma żaróweczkami dowolnego, ale tego samego koloru. Jeżeli nie jest możliwe wyznaczenie którejś z odległości, wypisz 0.

Przykład

Wejście	Wejście
7 1 2 3 3 1 1 3	1 5

Wyjaśnienie: najbliższe żarówki w kolorze 3 znajdują się w odległości 1 (1 2 3 3 1 1 3), najdalsze żarówki w kolorze 1 znajdują się w odległości 5 (1 2 3 3 1 1 3).

Zajęcia 11

Temat: Rekurencja. Metoda nawrotów

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice dwuwymiarowe, rekurencję, indukcję, pisze własne funkcje rekurencyjne, umie wyznaczyć liczby Fibonacciego, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem funkcji rekurencyjnej, równaniami rekurencyjnymi,
- zna porządek leksykograficzny,
- zna przeszukiwanie drzew ukorzenionych,
- zna liczby Fibonacciego (potrafi obliczyć),

Formy i metody pracy: praca samodzielna, wykład, omówienie

Zadania do wykonania na zajęciach	Treści programowe
1. Roztargniony profesor	M.3, P.2.16, P.2.17, P.2.19, A.3.5, A.3.6
2. Mysz w labiryncie	M.3, P.2.16, P.2.17, P.2.19, A.3.5, A.3.6
3. Problem 8 hetmanów	M.3, P.2.16, P.2.17, P.2.19, A.3.5, A.3.6

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/7/8/

https://www.main2.edu.pl/main2/courses/show/7/18/

Zadania do wykonania w domu:

Czy koń doskoczy (gotowa paczka do SIO2)

ZADANIA I ROZWIĄZANIA:

Zadanie 1. Roztargniony profesor

Dostępna pamięć: 64MB

Pewien profesor, aby dostać się do swego gabinetu, musi pokonać N schodów. Profesor pokonuje zazwyczaj jeden lub dwa schody na raz. Oblicz, na ile różnych sposobów profesor może pokonać schody. Podaj te sposoby.

Wejście

Jedna liczba całkowita N (1 ≤ N ≤ 25) - liczba schodów.

Wyjście

Wyjście zawiera w pierwszym wierszu liczbę możliwych kombinacji, w kolejnych wierszach wszystkie możliwe kombinacje w porządku leksykograficznym.

Przykład

Wejście	Wyjście
3	3 1 1 1 1 2 2 1

Rozwiązanie

Profesor zgodnie z treścią zadania ma do pokonania N schodów. Napiszmy procedurę profesor, które będzie symulowała kolejne ruchy profesora. Jakie kroki może wykonać? Zgodnie z treścią zadania pokonuje jednocześnie jeden schodek (wówczas zostanie mu do pokonania jeszcze N-1 schodów) lub 2 schody (analogicznie pozostanie jeszcze N-1 schodów) – oczywiście profesor wykona ruch, jeżeli pozostała przed nim wystarczająca liczba schodów. Jak zapamiętać wszystkie sposoby, na które profesor się porusza? Dodajmy jako parametr funkcji listę dotychczas wykonanych ruchów i po każdym kroku dodawajmy kolejny. Po tym, jak dojdziemy na sam koniec schodów (pozostanie 0 schodów), wypiszmy uzyskaną kombinację oraz zliczmy kolejny sposób dojścia na koniec schodów.

```
ile_kombinacji = 0

profesor (N, ruchy)
     jeżeli (N = 1)
          ile_kombinacji ← ile_kombinacji + 1
          wypisz ruchy
     jeżeli (N ≥ 1) profesor (N-1, ruchy + ”1”)
     jeżeli (N ≥ 2) profesor (N-2, ruchy + ”2”)
```

Zadanie dla uczniów: Na ile sposobów profesor może przejść schody? Jaki znany ciąg opisuje tę liczbę?

Zadanie 2. Mysz w labiryncie

Dostępna pamięć: 32MB

Do labiryntu trafiła mysz. Pomóż jej odnaleźć wyjście!

Dane:

W pierwszej linii wejścia znajdują się dwie liczby całkowite n i m (2 ≤ n, m ≤ 1000) oznaczające rozmiar labiryntu: n kolumn i m wierszy. W kolejnych liniach znajdują się znaki oznaczające kolejno:

- `-` - możliwość przejścia (korytarz);
- `x` - brak przejścia (ściana);

- o - początkowe położenie myszy;
- w - wyjście (końcowe położenie myszy).

Wynik

Pierwszy i jedyny wiersz wyjścia powinien wypisać „**TAK**” – jeśli istnieje wyjście z labiryntu, lub „**NIE**” – jeżeli warunek nie będzie spełniony.

Wejście	Wyjście
8 8 w--x-x-- xx-x-x-- ---x---- --xxxx-- --x--x-- -----x-o xxxx---- ------x-	TAK

Rozwiązanie

Umieśćmy mysz i labirynt w kwadratowej tablicy `t[n+2][m+2]`. Miejsca, gdzie stoją ściany, oznaczmy wartością 1. Podobnie dodajmy ściany dookoła obszaru, po którym porusza się mysz (będzie to nasz wartownik – wartości 1 w pierwszym i ostatnim wierszu oraz kolumnie). Teraz wystarczy miejsce, gdzie mysz stoi (wejście), oznaczyć jako odwiedzone (wartość 1) i spróbować przejść rekurencyjnie w lewo, w prawo, w dół oraz w górę (jeżeli to miejsce było jeszcze nieoznaczone). Na koniec wystarczy sprawdzić, czy kiedykolwiek odwiedziliśmy miejsce o współrzędnych wyjścia (w algorytmie poniżej pominęliśmy wczytywanie).

```
ruch_myszy (X, Y)
     t[X][Y] ← 1
     jeżeli (t[X-1][Y] = 0) ruch_myszy(X-1, Y)
     jeżeli (t[X+1][Y] = 0) ruch_myszy(X+1, Y)
     jeżeli (t[X][Y-1] = 0) ruch_myszy(X, Y-1)
     jeżeli (t[X][Y+1] = 0) ruch_myszy(X, Y+1)
```

Główna część programu:

```
ruch_myszy (X_wejścia, Y_wejścia)
jeżeli (t[Xwyjścia][Ywyjścia] = 1) wypisz ”TAK”
w przeciwnym wypadku wypisz ”NIE”
```

Zadanie 3. Problem 8 hetmanów

Janek uczy się grać w szachy. Poznaje sposób poruszania się różnych figur. Najśmieszniejszy wydaje mu się styl skoków konika. Zastanawia się również, co byłoby, gdyby na szachownicy znalazło się kilka figur hetmanów. Analizuje, czy na planszy o wymiarach **n** x **n** możliwe jest takie rozstawienie **n** tych figur, aby żaden z nich się nie szachował, a jeśli tak, to na ile sposobów można to zrobić?

Wejście

n - rozmiar szachownicy, liczba naturalna mniejsza niż 15.

Wyjście

Liczba sposobów, na jakie można rozmieścić figury hetmanów na szachownicy tak, by się wzajemnie nie szachowały.

Przykład

Wejście	Wyjście
8	92

Rozwiązanie

To klasyczny problem, dla którego wynik liczbowy bardzo łatwo odnaleźć w Internecie. Warto jednak go przeanalizować jako klasyczny algorytm z nawrotami.

Dysponujemy szachownicą o rozmiarze n na n pól. Chcemy na niej rozmieścić w taki sposób n królowych, aby żadna z nich się wzajemnie nie szachowała. W tym celu ustawiamy pierwszą królową w pierwszej kolumnie w pierwszym wierszu. Przechodzimy (rekurencyjnie) do kolejnej kolumny i szukamy wiersza, który nie jest szachowany. I ponownie przechodzimy rekurencyjnie do kolejnej kolumny. Jeżeli uda się nam w ten sposób ustawić królową w n-tym wierszu, zliczamy tę sytuację jako kolejny sposób. Następnie usuwamy królową z szachownicy i próbujemy ustawić w kolejnym wierszu. Po przeanalizowaniu wszystkich wierszy w kolumnie wracamy do poprzedniej kolumny.

W jaki prosty sposób zapamiętać już zajęte pola? Dość łatwo zauważyć, że tablica `zajęty_wiersz[]` może spamiętywać wiersze, w których ustawiliśmy już hetmana. Podobna obserwacja pozawala zapamiętać zajęte przekątne.

Niech w oznacza numer badanego wiersza, zaś k – kolumny.

```
funkcja możliwe (w, k)
   zwróć zajęty_wiersz[w]=0 i przekątna1[w+k-1]=0 i przekątna2[k-
w+n]

procedura ustaw(w, k)
     zajęty_wiersz[w]←1, przekątna1[w+k-1]←1, przekątna2[k-w+n]←1

procedura zdejmij(w, k)
     zajęty_wiersz[w]←0, przekątna1[w+k-1]←0, przekątna2[k-w+n]←0

procedura próbuj (k)
     dla i=1,2,3,…,n wykonaj
          jeżeli możliwe (w, k)
               ustaw(w, k)
               jeżeli k < n
                    próbuj (k+1)
               przeciwnie
                    ilość_sposobów ← ilość_sposobów + 1
               zdejmij(w, k)
```

Główna część programu:

```
wczytaj n
próbuj(1)
```

```
wypisz ilość_sposobów
```

Zajęcia 12

Temat: Sortowanie 1

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, stosuje algorytmy sortowania, wyznacza złożoność obliczeniową, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem algorytmu sortującego,
- zna porządek częściowy i liniowy.

Formy i metody pracy: praca samodzielna, omówienie, wykład, dyskusja

Zadania do wykonania na zajęciach	Treści programowe
1. Latarnie	M.3, P.2.14, A.3.4
2. Najczęściej występujący (najc2)	M.3, P.2.14, A.3.4
3. Ciąg Farey'a (fare)	M.3, P.2.14, A.3.4

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/7/13/

Zadania do wykonania w domu:

Minimalna liczba (II OIG):

https://szkopul.edu.pl/problemset/problem/8-Z3rWfeUiuDVT7E9Yc3LTFt/site/

Deski kontratakują (XIV OIG):

https://szkopul.edu.pl/problemset/problem/2hydlzVm5wFFvOONGbTbm0Ow/site/

ZADANIA I ROZWIĄZANIA:

Zadanie 1. Latarnie

Dostępna pamięć: 32MB

Samotne długie noce i ostre noże w kieszeniach
Wskazówki zegarka świecą w ciemnościach
nie, nie, nie
Kroki na schodach, w bezruchu zamierasz
Tu czai się strach, tak ciemno tu teraz
nie, nie, nie
Właśnie tak, tak wygląda moje miasto nocą,
tak wygląda nocą świat
nie, nie, nie
tak, tak wygląda moje miasto nocą,
tak wygląda nocą świat

De Mono: Moje miasto nocą

Nie każdy lubi chodzić nocą ulicami, na których latarnie dają zbyt mało światła. Ciemności boi się również pan Integer. Wzdłuż ulicy, na której mieszka, postawiono k latarni. Pan Integer uważa, że jest ich stanowczo za mało. Albo za słabo świecą. Ponieważ zaplanowano właśnie wymianę wszystkich żarówek na nowe (energooszczędne), pan Integer postanowił zwrócić się do miejscowego zarządu dróg z nietypową prośbą. Chciałby, aby każda latarnia świeciła światłem z odpowiednią jasnością co najmniej o promieniu r tak, aby cała ulica była oświetlona. Nie chce przy tym narażać miasta na dodatkowe koszty, dlatego chce wyznaczyć r tak, by było jak najmniejsze. Pomożesz mu?

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite d oraz k – odpowiednio długość ulicy, na której mieszka pan Integer oraz liczba latarni ($1 \le k \le d \le 10^6$). W kolejnej linii znajduje się k liczb całkowitych x_i – odległości i-tej latarni od początku ulicy ($0 \le x_i \le d$). Latanie mogą stać w dowolnym miejscu ulicy. W jednym miejscu znajduje się tylko jedna latarnia.

Wyjście

Minimalny promień światła r, dla którego każdy punkt ulicy zostanie rozświetlony. Wynik wypisz z dokładnością do jednego miejsca po przecinku.

Przykład

Wejście	Wyjście
`10 3` `1 5 9`	`4.0`

Rozwiązanie

Dane należy wczytać do tablicy, posortować ją, a następnie znaleźć największą różnicę pomiędzy dwoma sąsiednimi elementami. Wynik (podzielony przez 2) należy przyrównać do odległości pierwszego elementu od początku oraz ostatniego od końca i wybrać największy z nich.

Do sortowania można użyć polecenia `sort` z biblioteki STL.

Zadanie 2. Najczęściej występujący

Dostępna pamięć: 32MB

Wyznacz najpopularniejszy element ciągu, to znaczy taki, który występuje w nim największą liczbę razy. W przypadku, gdy w ciągu jest więcej niż jeden najpopularniejszy element, podaj największy z nich.

Wejście

W jednej linii znajduje się ciąg liczb całkowitych zakończony liczbą 0. Liczb jest nie więcej niż 1000000, ich wartość mieści się w przedziale $[-10^{18}, 10^{18}]$.

Wyjście

Najpopularniejszy element ciągu.

Przykład

Wejście	Wyjście
1 2 -5 7 1 3 3 7 3 1 0	3

Rozwiązanie

Ze względu na zakres danych wejściowych nie będzie możliwe zliczenie w tablicy wystąpień każdego z elementów. Dane należy wczytać do tablicy `t[]` i posortować ją. Elementy o tej samej wartość znajdują się obok siebie. Wystarczy je zliczyć i wybrać wartość maksymalną.

```
sortuj(t,t+n)
m ← 0, ile ← 1
dla i=1,2,…,n-1
      jeżeli t[i]=t[i-1]
            ile ← ile + 1
      przeciwnie
            ile ← 1
      jeżeli m < ile
            m ← ile
wypisz m
```

Zadanie 3. Ciąg Farey'a

Dostępna pamięć: 32MB

Ciąg Farey'a dla liczby naturalnej N (N > 1) to ciąg ułamków, których licznik i mianownik są liczbami naturalnymi nieprzekraczającymi liczby N oraz licznik nie jest większy od mianownika. Ułamki powinny być skrócone, posortowane rosnąco i nie mogą się powtarzać. Oto przykład ciągu Farey'a dla N = 4:

$$\frac{0}{1}\ \frac{1}{4}\ \frac{1}{3}\ \frac{1}{2}\ \frac{2}{3}\ \frac{3}{4}\ \frac{1}{1}$$

Napisz program, który czyta liczbę N i wypisuje odpowiadający jej ciąg Farey'a.

Wejście

Pierwszy i jedyny wiersz danych zawiera jedną liczbę naturalną N (3 ≤ N ≤ 1000).

Wyjście

Program powinien wypisać w jednym wierszu ciąg ułamków zapisanych przy użyciu znaku „/” (np. „1/2”) oddzielonych pojedynczymi odstępami, stanowiącymi ciąg Farey’a dla liczby N.

Przykład

Wejście	Wyjście
4	0/1 1/4 1/3 1/2 2/3 3/4 1/1

Rozwiązanie

Najprostszym sposobem rozwiązania zadania będzie wygenerowanie wszystkich możliwych ułamków z podanego zakresu oraz zapamiętanie w tablicy tych nieskracalnych(funkcja `__gcd` z STL zwraca największy wspólny dzielnik).

W językach C/C++ stosunkowo proste wydaje się skorzystanie ze struktury, która będzie przechowywać liczniki i mianowniki.

```
struct ulamek { int l, m;};
ulamek u[1000000];
```

Aby skorzystać z sortowania z biblioteki STL, konieczne jest w tym wypadku skorzystanie z własnej funkcji porównującej (z porównania usunęliśmy dzielenie):

```
bool operator < (const ulamek &a, const ulamek &b) {
     return (a.l*b.m)<(b.l*a.m);
}
```

Główna część programu:

```
k=1;
for (int i=1; i<n;i++)
     for (int j=n; j>=2; j--)
          if (__gdc(i,j)==1 && j>i) {
               u[k].l=i; u[k].m=j;
               k++;

sort(u,u+k);
for(int i=0; i<k;i++) printf("%d/%d ", u[i].l, u[i].m);
```

Pamiętajmy o wypisaniu pierwszego i ostatniego elementu ciągu!

Zajęcia 13

Temat: Sortowanie 2

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, stosuje algorytmy sortowania, wyznacza złożoność obliczeniową, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem algorytmu sortującego,
- zna porządek częściowy i liniowy,
- zna algorytm sortowania przez wstawianie, scalanie i szybkie.

Formy i metody pracy: praca samodzielna, dyskusja, omówienie

Zadania do wykonania na zajęciach	Treści programowe
1. Wirgiliusz	M.3, P.2.14, A.3.4
2. Układanie kart	M.3, P.2.14, A.3.4
3. Równoważne teksty (rtezad)	M.3, P.2.14, A.3.4

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/7/13/

Zadania do wykonania w domu:

Czekolada (X OI):

https://szkopul.edu.pl/problemset/problem/8AKFvYX1GKjeakIidXGH5_h7/site

Lotniska (X OI):

https://szkopul.edu.pl/problemset/problem/YtuyhoaeeDNR5zcXbKEewOdA/site

ZADANIA I ROZWIĄZANIA:

Zadanie 1. Wirgiliusz

Limit pamięci: 128MB

Ojciec Wirgiliusz,
uczył dzieci swoje,
a miał ich wszystkich
prawdziwe roje,
takie malutkie,
i takie wielkie,
takie grubiutkie,
i takie cienkie.

Piosenka tradycyjna

Wszystkie dzieci Wirgiliusza postanowiły zamieszkać w Bajtomiu w pięknych domkach na nowo wybudowanej ulicy Bajtockiej. Razem z nimi zamieszka również Wirgiliusz. Ponieważ ojciec zamierza często odwiedzać wszystkie swoje dzieci, chciałby znaleźć taki dom jednego z nich, aby mieć możliwie blisko nich wszystkich.

Ojciec Wirgiliusz chce zminimalizować sumaryczną odległość do wszystkich swoich dzieci i poprosił Ciebie, abyś napisał stosowny program.

Wejście

Pierwszy wiersz zawiera liczbę całkowitą r ($1 \le r \le 10^6$) oznaczającą liczbę dzieci Wirgiliusza. Drugi wiersz zawiera r całkowitych numerów domów s_1, s_2, ..., s_i, ..., s_r, w których one mieszkają ($1 \le s_i \le 1\,000\,000\,000$). Zauważ, że różne dzieci mogą mieszkać w tych samych domach.

Wyjście

W pierwszym i jedynym wierszu wyjścia Twój program musi wypisać minimalną sumę odległości optymalnie położonego domu Wirgiliusza od wszystkich dzieci. Odległość między dwoma domami o numerach s_i i s_j wynosi $d_{ij} = |s_i - s_j|$.

Przykład

Wejście	Wyjście
3 2 4 6	4

Rozwiązanie

Rozwiązanie niepoprawne: wybór mieszkania leżącego najbliżej średniej wszystkich numerów.

Rozwiązanie wystarczające: Prosta obserwacja prowadzi do wniosku, że najlepszym wyborem miejsca zamieszkania dla Wirgiliusza będzie mediana wszystkich numerów. Wystarczającym rozwiązaniem będzie posortowanie tablicy `s[]` z `r` numerami mieszkań z użyciem STL, a następnie obliczenie sumy wszystkich różnic między miejscem zamieszkania Wirgiliusza i pozostałymi domami (poniżej fragment kodu).

```
  sort(s,s+r);
  vito = r / 2;
  for (int i=0; i<r; i++)
    d = d + abs( s[i] - s[vito] );
  cout << d;
```

Lepszym rozwiązaniem byłoby samodzielne wyznaczenie mediany z użyciem algorytmu Hoare'a (działającym w czasie O(n)) zamiast sortowania.

Zadanie 2. Układanie kart

Limit pamięci: 64MB

Mały Bitek dostał od rodziców talię kart. Ze zdziwieniem jednak zauważył, że karty zamiast tradycyjnych oznaczeń typu *król pik* są oznaczone kolejnymi liczbami całkowitymi. Szybko jednak bardzo mu się to spodobało.

Bitek grając w karty zawsze ustawia je od najmniejszej do największej. Stosuje przy tym popularną metodę:

- zakłada, że pierwsza karta jest już na swoim miejscu,
- każdą kolejną kartę przesuwa w lewo tak długo, aż znajdzie się ona na właściwej pozycji.

Bitek zauważył, że w trakcie porządkowania *i*-ta karta mija zawsze k_i innych kart. Bardzo go to zaintrygowało i teraz chciałby wiedzieć, ile łącznie takich „minięć" zostanie wykonanych dla całej potasowanej talii.

Wejście

W pierwszym wierszu znajduje się jedna liczba całkowita n – liczba kart ($1 \le n \le 10^6$). W kolejnej linii znajduje się n różnych liczb całkowitych x_i – wartości kolejnych kart ($1 \le x_i \le n$).

Wyjście

Jedna liczba całkowita oznaczająca łączną liczbę miniętych w trakcie układania kart.

Przykład

Wejście	Wejście
4 4 2 3 1	5

Rozwiązanie

Rozwiązanie wolne: Możemy zliczać wszystkie minięte elementy symulując sortowanie przez wstawianie. Takie rozwiązanie zyska 20 punktów.

```
insertion_sort (T[], n)
i ← 1
wynik ← 0
    dopóki i < n // wykonaj n-1 krotnie
    j ← i - 1   // liczby do i-1 włącznie są uporządkowane
```

```
    k ← T[i]    // zapamiętaj i-tą liczbę do wstawienia
    // dopóki nie dojdziesz do początku ciągu
    // i element badany jest większy niż element wstawiany
    dopóki j ≥ 0 i T[j] > k
        T [j + 1] ← T [j] // przesuń badany element o jeden w prawo
        j ← j - 1
        wynik ← wynik + 1 // zapamiętaj kolejne przesunięcie
        // element wstawiany zapisz na ostatniej zapamiętanej
pozycji
    T [j + 1] ← k
    i = i + 1
zwróć wynik
```

Rozwiązanie szybkie. Przeanalizujmy proces sortowania przez zliczanie. Zwróćmy przede wszystkim uwagę na proces scalania dwóch posortowanych ciągów. Jeżeli założymy, że scalane ciągi znajdują się w sortowanej tablicy bezpośrednio po sobie, to zabranie elementu z pierwszego z nich do sortowanej tablicy nie spowoduje „minięcia" żadnego z elementów drugiej tablicy. Z kolei element zabrany z drugiego ciągu mija jednocześnie wszystkie nie zabrane jeszcze elementy z ciągu pierwszego. Dzięki temu uzyskamy złożoność sortowania przez scalanie: O(*n* log *n*).

```
scal(t[],P,L)
  //przepisz dane do tablicy pomocniczej
  dla i=P, P+1, …, L wykonaj pom[i] ← t[i]
  //znajdź środek
  s ← (P+L)/2
  //zacznij od początku lewego i prawego ciągu
  i ← L, j ← s+1, k ← L
  //dopóki nie skończy się lewy lub prawy ciąg
  dopóki i ≤ s i j ≤ P
    //wybierz mniejszy element z prawego lub lewego ciągu
    //i przepisz go do tablicy wynikowej
    jeżeli pom[i] ≤ pom[j]
      t[k] ← pom[i], i←i+1,
    przeciwnie
      t[k] ←pom[j],j←j+1
      // policz elementy, które mija w tym momencie pom[j]
      policz ← policz + (s-i+1)
    k ← k+1
  //przepisz pozostałe elementy
  dopóki i ≤ s
    t[k] ← pom[i], i←i+1, k ← k+1
  dopóki j ≤ P
    t[k] ←pom[j], j←j+1, k ← k+1
```

Główna część algorytmu merge-sort:

```
mergesort(L, P)
jeśli jest więcej niż jedna liczba do posortowania
znajdź środek
posortuj lewą część
posortuj prawą część
scal posortowane ciągi
```

Zadanie 3. Równoważne teksty

Dostępna pamięć: 256MB

Dwa teksty *a* oraz *b* nazwiemy równoważnymi, jeżeli spełniają jeden z dwóch warunków:

1. są identyczne,
2. jeżeli tekst *a* podzielimy na dwie równej długości części a_1 i a_2, a tekst *b* podzielimy podobnie na teksty b_1 i b_2, to jeden z warunków będzie poprawny:
   - a_1 i b_1 oraz a_2 i b_2 będą równoważne

     lub
   - a_1 i b_2 oraz a_2 i b_1 będą równoważne.

Sprawdź, czy podane dwa teksty są równoważne według powyższej definicji!

Wejście

W dwóch wierszach wejścia znajduje się po jednym tekście. Teksty mają długość od 1 do $2 \cdot 10^5$ znaków i składają się wyłącznie z małych liter alfabetu łacińskiego. Długości tekstów są równe.

Wyjście

Wypisz jedno słowo: TAK lub NIE, odpowiedź na pytanie, czy podane teksty są równoważne.

Przykład

Dla danych wejściowych	poprawnym wynikiem jest
`abcd` `cdba`	`TAK`

Dla danych wejściowych	poprawnym wynikiem jest
`abcd` `acbd`	`NIE`

Objaśnienie przykładu 1

Tekst abcd rozkładamy na ab i cd. Tekst cdba rozkładamy na cd i ba. W tym wypadku cd jest równoważne cd. Postępując tą samą metodą stwierdzimy, że ab i ba są równoważne.

Rozwiązanie

Zauważmy, że znacznie łatwiej będzie nam stwierdzić, czy dwa teksty o parzystej długości są równoważne, jeśli ich porównywane części weźmiemy do porównania w kolejności leksykograficznej. W tym wypadku po prostu powinny być sobie równe. To samo możemy zrobić rekurencyjnie dla ich podciągów. Spróbujmy więc uporządkować oba porównywane ciągi w opisany sposób i sprawdzić, czy po porównaniu otrzymujemy identyczne ciągi.

W poniższym algorytmie dla uproszczenia wykorzystana została funkcja `substr` oraz `min` z biblioteki STL.

```
sortuj (s[])
      n ← |s[]|
      jeżeli (n mod 2 = 1)
            zwróć s[]
      tekst x ← sortuj(s.substr (0, n/2))
      tekst y ← sortuj(s.substr (n/2))
      zwróć min (x+y, y+x)
```

Główna część programu:

```
tekst a, b
wczytaj (a, b)
jeżeli sortuj(a) = sortuj(b)
      wypisz „TAK”
przeciwnie
      wypisz „NIE”
```

Zajęcia 14

Temat: Wyszukiwanie binarne

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, stosuje algorytmy sortowania, wyszukiwanie binarne wyznacza złożoność obliczeniową, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem algorytmu sortującego,
- zna porządek częściowy i liniowy,
- zna algorytmy sortowania, wyszukiwanie binarne.

Formy i metody pracy: praca samodzielna, omówienie, wykład, dyskusja

Zadania do wykonania na zajęciach	Treści programowe
1. Znaczki	M.3, P.2.14, A.3.2, A.3.4
2. Tunele	M.3, P.2.14, A.3.2, A.3.4
3. Bierki	M.3, P.2.14, A.3.2, A.3.4

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/7/5/

Zadania do wykonania w domu:

Statki kosmiczne (II OIG):

https://szkopul.edu.pl/problemset/problem/isVbAEInYlrdxQNvE6mEnS9i/site/?key=statement

Kalendarze (III OIG):

https://szkopul.edu.pl/problemset/problem/LWpMcXylQBa6wHzcJ6U7axzK/site/?key=statement

ZADANIA I ROZWIĄZANIA:

Zadanie 1. Znaczki

Limit pamięci: 32MB

Bitek i Bajtek zbierają znaczki. Bitek planuje wziąć udział w wystawie połączonej z konkursem na największy zbiór znaczków. Aby zwiększyć swoje szanse na odniesienie spektakularnego sukcesu, postanowił pożyczyć od Bajtka te znaczki, których sam nie posiada. Chciałby teraz wiedzieć, o ile znaczków zwiększy się jego kolekcja.

Wejście

W pierwszej linii wejścia znajdują się dwie liczby naturalne *n* (mniejsza niż 100 000) i *m* (mniejsza niż 10 000 000), oznaczające odpowiednio liczbę znaczków Bitka oraz znaczków Bajtka. W drugiej linii wejścia znajduje się *n* różnych liczb naturalnych mniejszych niż miliard – typy znaczków Bitka. W trzeciej linii wejścia znajduje się *m* różnych liczb naturalnych mniejszych niż miliard – typy znaczków Bajtka. Możesz założyć, że w Bitek i Bajtek podali każdy posiadany znaczek tylko raz.

Wyjście

W jedynym wierszu wyjścia znajduje się jedna liczba całkowita: liczba znaczków w kolekcji Bajtka, których nie posiada Bitek.

Przykład

Dla danych wejściowych:	poprawną odpowiedzią jest:
6 4 1 2 3 4 5 6 2 4 5 7	1

Rozwiązanie

Rozwiązanie poprawne: wystarczy posortować pierwszy zestaw *n* znaczków (`a[]`), a następnie dla każdego z *m* znaczków z drugiego zestawu sprawdzać wyszukiwaniem binarnym czy wystąpił on w pierwszym zestawie (funkcja sort w bibliotece STL).

```
znajdź (x)
      L ← 0, P ← n − 1
      dopóki (L≤P)
            k ← (L+P) div 2
            jeżeli a[k] = x
                  zwróć PRAWDA
            jeżeli a[k] < x
                  L ← k+1
            przeciwnie
                  P ← k-1
      zwróć  FAŁSZ
```

Główna część programu:

```
sort(a, a+n)
dla i=0,1,…,m
```

```
      wczytaj x
      jeżeli znajdź(x)=FAŁSZ
           licznik ← licznik + 1
wypisz licznik
```

Zadanie 2. Tunele

Limit pamięci: 32MB

Janek zarządza dużą firmą dostawczą. Właśnie zamierza opracować nowy plan dostaw do swoich kontrahentów. Do wszystkich można dojechać nowo wybudowaną autostradą. Niestety samochody na autostradzie muszą pokonywać również tunele. Każdy z samochodów oraz każdy z tuneli ma określoną wysokość. Aby ciężarówka Janka przejechała pod tunelem, musi być niższa niż wysokość tunelu. Janek zna wysokości ciężarówek i tuneli i zastanawia się, do którego miejsca autostrady dojedzie każde z jego aut.

Wejście

W pierwszej linii wejścia znajduje się dwie liczby całkowite a i b ($1 \le a, b \le 500\,000$), odpowiednio ilość tuneli i samochodów. W drugiej linii znajduje się a oddzielonych spacjami liczb całkowitych $t_1, t_2, \ldots, t_a$ ($1 \le t_i \le 1\,000\,000\,000$), gdzie t_i oznacza wysokość i-tego tunelu. W trzeciej linii znajduje się b oddzielonych spacjami liczb całkowitych $s_1, s_2, \ldots, s_b$ ($1 \le s_i \le 1\,000\,000\,000$), gdzie s_i oznacza wysokość i-tego samochodu.

Wyjście

Pierwszy i jedyny wiersz wyjścia zawiera b liczb całkowitych $d_1, d_2, \ldots, d_i$, gdzie d_i oznacza numer ostatniego tunelu, przez który może przejechać i-ty samochód lub 0, jeśli nie może przejechać przez żaden z tuneli.

Przykład

Dla danych wejściowych:	poprawną odpowiedzią jest:
5 3 8 6 5 7 9 4 7 9	5 1 0

Rozwiązanie

Rozwiązanie wolne: Możemy symulować dla każdego z samochodów jazdę przez kolejne tunele, dopóki wysokość tunelu jest wystarczająca. To rozwiązanie będzie działać w czasie $O(a \cdot b)$.

Rozwiązanie szybkie: Przez kolejny tunel może przejechać samochód równy jego wysokości (minus jeden), jeżeli po lewej jego stronie nie znajdzie się żaden niższy tunel. W przeciwnym wypadku ten niższy tunel zadecyduje o maksymalnej możliwej wysokości samochodu. Zapamiętajmy więc dla każdego tunelu najniższą możliwą wysokość prześwitu w tablicy `t[]`. Teraz możemy dla każdego samochodu wyszukać odpowiedni tunel binarnie. Możemy wykorzystać funkcje `reverse` oraz `upper_bound` z biblioteki STL w celu uproszczenia obliczeń:

```
min=1000000000;
```

```
scanf("%d %d",&a,&b);
for(int i=0; i<a; i++) {
    scanf("%d",&t[i]);
    if (t[i]<min) min=t[i];
    t[i]=min;
}
reverse(t,t+a);
for (int i=0; i<b; i++) {
    scanf("%d",&s);
    int *p = upper_bound(t, t+a, s);
    printf("%d ",a-(int)(p-t));
}
```

Zadanie 3. Bierki (OIG)

Limit pamięci: 32MB

Jaś lubi budować trójkąty z bierek. W tym celu trzyma je w worku, z którego wybiera trzy bierki na chybił-trafił. Bierki mogą mieć różne długości i nie zawsze Jaś może zbudować trójkąt, a wtedy wpada w histerię. Mama Jasia ma dość histerycznych napadów synka i dlatego poprosiła Ciebie o pomoc. Należy odrzucić niektóre bierki w taki sposób, aby z pozostałych zawsze dało się ułożyć trójkąt, jednocześnie zostawiając jak najwięcej bierek w worku.

Zadanie

Opracuj program, który:

- wczyta ze standardowego wejścia liczbę bierek w worku oraz ich długości,
- obliczy największą liczbę bierek, którą można pozostawić w worku, tak żeby z każdych trzech z nich można było utworzyć trójkąt,
- wypisze wynik na standardowe wyjście.

Wejście

W pierwszym wierszu zapisano liczbę n ($5 \leq n \leq 30\,000$), oznaczającą liczbę bierek w worku. W każdym z n następnych wierszy zapisano długość jednej bierki. Długość bierki jest liczbą całkowitą z przedziału [1, 500].

Wyjście

W pierwszym wierszu wypisz liczbę bierek, które powinny zostać w worku.

Przykład

Dla danych wejściowych:	poprawną odpowiedzią jest:
10 7 1 2 8 10 6	7

1 7 9 9	

Rozwiązanie

Rozwiązanie wolne: Dla każdej pary dwóch różnych bierek sprawdźmy, ile bierek jest mniejszych niż suma dwóch pierwszych. Takie rozwiązanie będzie działało w czasie $O(n^3)$.

Rozwiązanie szybkie: Posortujmy bierki niemalejąco. Dla każdej bierki i oraz i+1 znajdźmy (wyszukiwaniem binarnym) pierwszą bierkę, której długość jest równa lub większa od sumy bierek i oraz i+1. Największy otrzymany w ten sposób przedział stanowi rozwiązanie zadania.

```
sort(a,a+n);
for (int i=0;i<n-1;i++)
    {
       int* p = lower_bound(a,a+n,a[i]+a[i+1]);
       ilosc=(p-a)-i;
       if(ilosc > bierki)
            bierki=ilosc;
    }
```

Zajęcia 15

Temat: Wyszukiwanie binarne po wyniku

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, stosuje algorytmy sortowania, wyszukiwanie binarne wyznacza złożoność obliczeniową, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem algorytmu sortującego,
- zna porządek częściowy i liniowy,
- zna algorytmy sortowania, wyszukiwanie binarne.

Formy i metody pracy: praca samodzielna, dyskusja, wykład, omówienie

Zadania do wykonania na zajęciach	Treści programowe
1. Zawody wędkarskie (zwezad)	M.3, P.2.14, A.3.2, A.3.4
2. Marchewki (marzad)	M.3, P.2.14, A.3.2, A.3.4

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/7/5/

Zadania do wykonania w domu:

Tort (V OIG):

https://szkopul.edu.pl/problemset/problem/G50OwjoYk8gUak-2HGHGll4o/site/?key=statement

ZADANIA I ROZWIĄZANIA:

Zadanie 1. Zawody wędkarskie

Dostępna pamięć: 32MB

Jasio organizuje zawody wędkarskie. Dokonał właśnie przeglądu wszystkich n stanowisk, jakimi dysponuje. Stanowiska umiejscowione są w linii prostej nad jeziorem w odległościach $x_1, x_2, ..., x_n$ od początku jeziora. W czasie konkursu zawodnicy nie mogą sobie nawzajem przeszkadzać. Dlatego Jaś chciałby teraz tak rozmieścić w wędkarzy, aby najmniejsza odległość pomiędzy dwoma najbliższymi z nich była jak największa. Pomóż znaleźć Jasiowi taki rozkład wędkarzy.

Wejście

W pierwszej linii wejścia znajdują się dwie liczby całkowite: n oraz w ($2 \le w \le n \le 10^6$). W kolejnej linii znajduje się n liczb całkowitych x_i ($0 \le x_i \le 10^9, x_i - 1 < x_i$). Każda z wartości oznacza odpowiednio odległość i-tego stanowiska od początku jeziora.

Wyjście

Jedna liczba całkowita wyznaczająca najlepszą możliwą odległość pomiędzy dwoma najbliższymi stanowiskami.

Przykład

Dla danych wejściowych	poprawnym wynikiem jest
6 3 2 5 10 13 18 19	8

Ocenianie

Podzadanie	Ograniczenia	Liczba punktów
1	$n \le 10^4$	40
2	brak dodatkowych założeń	60

Rozwiązanie

Aby ułatwić rozwiązania zadania, w tablicy `a[]` będziemy pamiętać odległości pomiędzy kolejnymi stanowiskami zamiast odległości od początku jeziora oraz napiszemy funkcję `rozmieść(x)` sprawdzającą, czy jest możliwe rozmieszczenie wszystkich wędkarzy tak, aby odległość między nimi wynosiła co najmniej x. Sprawdzenie wykonamy zachłannie, usadzając wszystkich wędkarzy jak najbliżej początku jeziora.

```
oblicz_odległości()
//pamiętam największą odległość oraz odległości pomiędzy kolejnymi
//stanowiskami zamiast odległości od początku jeziora
     m ← a[n-1]
     dla i=n-1, n-2, …, 0
          a[i] ← a[i] - a[i-1];

rozmieść (x)
     suma ← 0
```

```
      //pierwszy wędkarz na pewno zajął miejsce na stanowisku 0
      //pozostało więc jeszcze W-1 wędkarzy do rozmieszczenia
      //począwszy od stanowiska 1
      j ← W-1, i ← 1
      //jeżeli nie doszedłeś do końca, a masz jeszcze wędkarzy
      dopóki i<n i j>0 wykonuj
            //dodawaj kolejne odległości aż osiągniesz x
            dopóki i<N i suma<x wykonuj
                  suma ← suma +a[i]
                  i ← i+1
            jeżeli suma ≥ x
                  j ← j-1
                  suma ← 0
      jeżeli j=0
            zwróć PRAWDA
      przeciwnie
            zwróć FAŁSZ
```

Rozwiązanie wolne: Korzystając z funkcji rozmieść będziemy próbowali rozmieścić wędkarzy o kolejne odległości. Ostatnia możliwa odległość jest rozwiązaniem zadania (wczytywanie jest pominięte).

```
oblicz_odległości()
i ← 1
dopóki rozmieść(i) = PRAWDA
      i ← i + 1
wypisz i - 1
```

Rozwiązanie szybkie: Zauważmy, że największym problemem jest wykonywanie funkcji rozmieść dla zbyt wielu różnych odległości i. Możemy znacząco zmniejszyć liczbę sprawdzanych odległości korzystając z wyszukiwania binarnego. Zauważmy, że najmniejszą możliwą do uzyskania odległością jest jeden, zaś największą – zapamiętana w funkcji `oblicz_odległości` odległość m.

Wyszukując binarnie ostatnią odległość, dla której funkcja `rozmieść(x)` zwraca prawdę musimy uważać, by w każdym kroku właściwie zawężać zakres poszukiwań. Zauważmy, że jeżeli jakieś x nie spełnia warunków zadania, to odpowiedź może wynosić co najwyżej x-1 (zmniejszamy prawy zakres). Jeżeli x spełnia warunki zadania, to również nadaje się na wynik. Aby jednak również lewy zakres zawsze ulegał zawężeniu, zaokrąglajmy wyznaczanie środka w górę (w innym wypadku program mógłby się nam „zapętlić” na dwóch ostatnich wartościach):

```
oblicz_odległości()
L ← 1, P ← m
dopóki L < P
      środek ← (L+P+1) div 2
      jeżeli rozmieść(środek) = PRAWDA
            L ← x
      przeciwnie
            P ← x - 1
wypisz środek
```

Zadanie dla uczniów: określ złożoność obliczeniową dla obu metod.

Zadanie 2. Marchewki

Dostępna pamięć: 64MB

Bitomir zasadził na swoim polu marchewki. Każda marchewka jest oddalona od innej marchewki o jeden bitometr (jednostka długości w Bajtocji) w poziomie i w pionie. Bitomir chce zakupić specjalny zbierak do marchewek. Zbierak zamontowany jest na długim ramieniu i potrafi zebrać wszystkie marchewki w promieniu swojego działania. Bitomir zastanawia się teraz, jakiej długości ramię (w bitometrach) musi posiadać zbierak, by jednorazowo zebrał co najmniej *m* marchewek. Cena zbieraka zależy od długości ramienia. Bitomir chce zakupić jak najtańsze urządzenie. Pomóż mu wyznaczyć najlepszy zbierak!

Wejście

W pierwszej linii wejścia znajduje się jedna liczba całkowita m ($2 \le m \le 5 \cdot 10^{12}$), oznaczająca liczbę marchewek, którą chcemy jednorazowo zebrać.

Wyjście

Na wyjściu powinna znaleźć się jedna liczba całkowita oznaczająca minimalną długość ramienia zbieraka.

Przykład

Dla danych wejściowych	poprawnym wynikiem jest
13	2

Ocenianie

Podzadanie	Ograniczenia	Liczba punktów
1	$n \le 10^5$	15
2	$n \le 10^8$	15
3	brak dodatkowych założeń	60

Rozwiązanie

Rozwiązanie wolne 1: W każdym kwadracie o boku 2r obliczamy liczbę marchewek wewnątrz koła o promieniu r. Wypisujemy takie r, dla którego udało zebrać się co najmniej m marchewek:

```
r ← 0
wykonuj
      r ← r + 1
      liczba_marchewek ← 0
      dla x=-r, -r+1, …, r wykonuj
            dla y=-r, -r+1, …, r wykonuj
                  jeżeli x^2 + y^2 ≤ r^2
                        liczba_marchewek ← liczba_marchewek + 1
dopóki liczba_marchewek < m
wypisz r
```

Rozwiązanie ulepszone: Spróbujmy znacząco przyśpieszyć obliczanie liczby marchewek wewnątrz koła o promieniu r. Zauważmy, że dla x równego 0 nad osią 0x znajduje się dokładnie r marchewek. Po zwiększeniu x o 1 liczba ta może wyłącznie zmaleć. Odnajdźmy to y znajdujące się wewnątrz koła o promieniu r. Następnie dalej zmniejszajmy x aż do r. Zauważmy, że w ten sposób każde x oraz każde y zmniejszamy dokładnie r razy. Działanie to zawrzyjmy w funkcji ile_marchewek(r).

```
ile_marchewek (r)
      ile ← 0
      y ← r
      dla x=0,1,…, r wykonuj
            dopóki x^2 + y^2 ≤ r^2 wykonuj
                  y ← y – 1
            ile ← ile + y
      //rozwiązanie dla pierwszej ćwiartki mnożymy przez 4
      //dla czterech ćwiartek i dodajemy punkt (0,0)
      zwróć 4·ile+1
```

Główna część programu:

```
r ← 0
dopóki ile_marchwek(r) < m wykonuj
      r ← r + 1
wypisz r
```

Nadal jednak nasz program działa za wolno.

Rozwiązanie szybkie: Ponownie największy problem stanowi sprawdzanie wszystkich możliwych r (obliczania liczby marchewek wewnątrz okręgu niestety nie da się już przyśpieszyć). Zauważmy jednak, że potrafimy założyć najmniejszy oraz największy rozmiar ramienia urządzenia. Możemy też wyznaczać liczby marchewek dla długości pośrednich. Z pomocą przyjdzie nam więc wyszukiwanie binarne.

Jako minimalny promień przyjmijmy 0, jako maksymalny 2 000 000. Wyznaczmy środek dla tych dwóch wartości i obliczmy liczbę marchewek wewnątrz (funkcja ile_marchewek(r) z poprzedniego rozwiązania). Jeżeli jest równa m, zakończmy wyszukiwanie, za duża - promień może być równy bądź mniejszy, jeżeli jest zbyt mała, promień na pewno jest większy. Zadbajmy przy tym o zmniejszanie przedziału przeszukiwań w każdym kroku zarówno z lewej, jak i z prawej strony:

```
L ← 0, P ← 2000000
dopóki L < P wykonuj
      r ← (L+P) div 1 // zaokrąglenie w dół
      ile ← ile_marchewek(r)
      jeżeli  ile = m
            przerwij pętlę
      jeżeli ile > m
            P ← r
      przeciwnie
            L ← r + 1
wypisz r
```

Zadanie dla uczniów: określić liczbę operacji wykonywaną przez funkcję ile_marchewek w zależności od r. Określić złożoność obliczeniową każdego z trzech rozwiązań.

Zajęcia 16

Temat: Programowanie zachłanne, dynamiczne 1

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, przetwarza tablice dwuwymiarowe, algorytmy zachłanne testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem funkcji rekurencyjnej, równaniami rekurencyjnymi,
- zna operacje na zbiorach,
- zna przeszukiwanie z powrotami

Formy i metody pracy: praca samodzielna, dyskusja, omówienie, pogadanka

Zadania do wykonania na zajęciach	Treści programowe
1. Pszczółki	M.3, P.2.17, P.2.19, A.3.5
2. iGruszka	M.3, P.2.17, P.2.19, A.3.5
3. Spadek	M.3, P.2.17, P.2.19, A.3.5

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/7/20/

https://www.main2.edu.pl/main2/courses/show/7/22/

Zadania do wykonania w domu:

Urzędnicy (V OIG):

https://szkopul.edu.pl/problemset/problem/cSV0kK7zLib42_NGIA7DvOoZ/site/

ZADANIA I ROZWIĄZANIA:

Zadanie 1. Pszczółki

Dostępna pamięć: 256MB

Pszczoły rozmnażają się w specyficzny sposób: z niezapłodnionych jaj rodzą się samce (trutnie), a z zapłodnionych – samice (pszczoły). Rodzina pszczoły jest więc nietypowa – ma ona co prawda i ojca, i matkę, ale już tylko jednego dziadka i dwie babki.

Janek dogłębnie bada ten temat i zastanawia się, ile męskich przodków żyło w rodzinie pszczoły w n-tym pokoleniu. Pomóż Jankowi dokonać obliczeń!

Wejście

Pierwszy wiersz zawiera jedną liczbę całkowitą n ($0 < n \leq 10^5$) – numer pokolenia pszczoły, dla którego szukamy liczby trutni.

Wyjście

Jedna liczba całkowita – liczba trutni w n-tym pokoleniu. Ponieważ liczba trutni bardzo szybko rośnie, wystarczy, że wypiszesz wynik mod 10^9+7.

Przykład

Wejście	Wyjście
5	5

Rozwiązanie

Łatwo zauważyć, że liczba trutni w n-tym pokoleniu to n-ty wyraz ciągu Fibonacciego. Możemy oczywiście wyznaczyć wskazaną wartość ze wzoru iteracyjnego, ale w celach dydaktycznych pokażmy, jak w łatwy sposób wykorzystać rekurencję w połączeniu z programowaniem dynamicznym tak, by nie stracić na złożoności obliczeniowej.

Aby wyznaczyć liczbę trutni w n-tym pokoleniu skorzystajmy ze wzoru:

$$pszczółki(n) = \begin{cases} 1 \; dla \; n \leq 2 \\ pszczółki(n-1) + pszczółki(n-2) \; dla \; n > 2 \end{cases}$$

Kod funkcji:

```
pszczółki(n)
      jeżeli n ≤ 2
            zwróć 1
      zwróć pszczółki(n-1)+pszczółki(n-2)
```

Łatwo zauważyć, że liczba wywołań rekurencyjnych funkcji będzie rosła bardzo szybko (w zbliżonym tempie do wartości n-tego wyrazu ciągu Fibonacciego). Co stanowi największy problem? Wielokrotne wywoływanie funkcji dla już wcześniej obliczonych parametrów. Aby temu zapobiec zapamiętujmy każdą obliczoną wartość w tablicy `trutnie[]`. Wówczas obliczenie nowej wartości funkcji będzie konieczne jedynie wówczas, gdy nie została jeszcze wyznaczona wartość w tabeli `trutnie[]`. Pamiętajmy, by po każdym kroku zapamiętywać wynik mod 10^9+7.

```
pszczółki(n)
      jeżeli n < 2
```

```
zwróć 1
jeżeli trutnie[n]=0
     trutnie[n] ← (pszczółki(n-1)+pszczółki(n-2) ) mod 10^9+7
zwróć trutnie[n]
```

Zadanie dla uczniów: Oszacować złożoność obliczeniową każdego z rozwiązań.

Zadanie 2. iGruszka

Limit pamięci: 64MB

Firma *iGruszka* wyprodukowała nowy model smartfona. Telefon ma dotykowy ekran, całkiem niezłe parametry i śliczne logo w kształcie nadgryzionej gruszki. Zarząd firmy postanowił jak najwięcej zarobić na swoim sztandarowym produkcie. Przeprowadzono więc bardzo szczegółowe badania sprawdzające, jaką kwotę są gotowi zapłacić poszczególni klienci za aparat. Sprawdzono też, jaką liczbę telefonów mogą wyprodukować firmowe fabryki. Znając koszty produkcji urządzenia, wyniki badań klientów oraz liczbę możliwych do zamówienia smartfonów, oblicz najlepszą cenę urządzenia (tzn. taką, by firma *iGruszka* zarobiła jak najwięcej) oraz zysk firmy.

Wejście

Pierwszy wiersz wejścia zawiera trzy liczby całkowite s, k oraz n – odpowiednio liczba możliwych do wyprodukowania telefonów, koszt jednego telefonu oraz liczba potencjalnych klientów ($0 \le s, k, n \le 10^6$). W kolejnych n liniach znajdują się maksymalne kwoty x_i, które i-ty klient jest gotowy zapłacić za telefon ($0 \le x_i \le 10^6$).

Wyjście

Na wyjściu w jednym wierszu wypisz jedną liczbę całkowitą – maksymalny możliwy do uzyskania zysk firmy.

Przykład

Dla danych wejściowych:	poprawną odpowiedzią jest:
5 8 5 15 11 10 12 16	14

Rozwiązanie

Zadanie to możemy rozwiązać metodą zachłanną – będziemy próbowali sprzedać jak największą liczbę telefonów za jak największą możliwą kwotę (maksymalizacja zysków). W tym celu posortujemy zyski (kwota, jaką gotów zapłacić jest klient minus koszt produkcji telefonu), jakie wygeneruje każdy klient za nowy telefon i będziemy analizować zarobek firmy począwszy od najwyższego. Zauważmy, że telefon za podaną cenę mogą kupić wszyscy klienci, którzy zadeklarowali *co najmniej* podaną kwotę. Zapamiętamy przy tym największy osiągnięty wynik.

W poniższym kodzie pominięto wczytywanie:

```
dla i=0,1,…, n-1 wykonuj
     x[i] ← x[i] – k
sort(a,a+n)
max_zarobek ← 0
dla i=0,1,…, n-1 wykonuj
     ilu_klientów ← n – i
     jeżeli ilu_klientów > s
          ilu_klientów ← s
     wartość ← ilu_klientów · a[i]
     jeżeli wartość > max_zaronek
          max_zarobek ← wartość
wypisz max_zarobek
```

Zadanie dla uczniów: Oszacować złożoność obliczeniową rozwiązania.

Zadanie 3. Spadek

Stary Jan ma dwóch synów. Ponieważ chłopcy często się kłócą, postanowił zapobiec następnym niesnaskom i sam podzieli majątek między nich. Jan chce rozdzielić majątek na dwie części tak, aby różnica wartości pomiędzy nimi była jak najmniejsza. Jeśli nie da się podzielić po równo, starszy syn otrzyma więcej.

Wejście

W pierwszym wierszu standardowego wejścia zapisano liczbę naturalną p ($1 \le p \le 150$) – liczbę składników majątku, a następnie ich wartości w_i ($1 \le w_i \le 10\,000$, $i = 1, 2, 3 \ldots p$).

Wyjście

W jednym wierszu standardowego wyjścia zapisz kwoty wartości części majątku starszego i młodszego syna, rozdzielając je spacją.

Przykłady

Wejście	Wejście
4 2 2 2 5	5 10 5 10 5 5
Wyjście 6 5	Wyjście 20 15

Rozwiązanie

Zadanie to klasyczny problem wydawania reszty. Zauważmy, że dysponując skończoną liczbą przedmiotów o określonych wartościach nie zawsze możemy uzyskać satysfakcjonującą nas sumę. Błędne byłby również w tym wypadku użycie metody zachłannej – wybieranie przedmiotu o jak największej i dodawanie go do wybranej sumy.

Spróbujmy zatem wyznaczyć wszystkie możliwe do uzyskania sumy. Zawsze możemy uzyskać sumę 0 (nie biorąc żadnego przedmiotu). Zapamiętajmy ją jako pierwsze maksimum m. Teraz dla każdego aktualnego przedmiotu zaktualizujmy dotychczas osiągnięte kwoty

o jego wartość (od prawej do lewej; w ten sposób unikniemy błędnego wielokrotnego aktualizowania tego samego wyniku).

```
m ← 0
wartości[0] ← 1
wczytaj(n)
dla i=1,2,…, n wykonuj
      wczytaj aktualny
      k ← m //zapamiętuję ostatni element do aktualizacji
      m ← aktualny + m //aktualizuję maksimum
      wykonuj
            jeżeli wartości[k] = 1
                  wartości[k+altualny] ← 1
            k ← k – 1
      dopóki k ≥ 0
```

Na koniec nie pozostało nam nic więcej, niż wybrać te dwie kwoty, które różnią się jak najmniej od siebie:

```
j ← m div 2
dopóki wartości[k] ≠ 1
      j ← j – 1
wypisz m – j, j
```

Zadanie dla uczniów: Oszacować złożoność obliczeniową rozwiązania.

Zajęcia 17

Temat: Programowanie zachłanne, dynamiczne 2

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, przetwarza tablice dwuwymiarowe, algorytmy zachłanne testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem funkcji rekurencyjnej, równaniami rekurencyjnymi,
- zna operacje na zbiorach,
- zna przeszukiwanie z powrotami

Formy i metody pracy: praca samodzielna,

Zadania do wykonania na zajęciach	Treści programowe
1. Poszukiwania (szu)	M.3, P.2.17, P.2.19, A.3.5
2. Stadion	M.3, P.2.17, P.2.19, A.3.5
3. Zbiórka	M.3, P.2.17, P.2.19, A.3.5

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/7/20/

https://www.main2.edu.pl/main2/courses/show/7/22/

Zadania do wykonania w domu:

Bracia (V OIG):

https://szkopul.edu.pl/problemset/problem/KkZDP16mELVF692Bf-NJsN1L/site/

Monety (V OIG):

https://szkopul.edu.pl/problemset/problem/GMBfzRQAdw3b7wfvC9Xf7YE4/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Poszukiwania

Dostępna pamięć: 32MB

Książe Bratumił II Wstydliwy przez stosunek do białogłów swój przydomek otrzymał. Przeto jego rodziciel, król Rufin IV Rzadkowąsy, martwił się wielce. 40 lat Bratumiła dojrzałością późną na pewno zwać nie można, wszelako żona i dzieci dla syna chęcią przedłużenia rodu królewskiego nieodzowne Rufinowi wydawać się mogły. Toteż synowi nakazał, by królestwo całe przewędrował i za żonę jakową dorodną infantkę pojął. Syn wolę ojca spełnić zaprzysiągł, aliści warunek także wysunął: spędzi w podróży na poszukiwaniach księżniczki o dwa dni mniej, niż monarchia ojca hrabstw w szerokości i w długości w sumie liczy, a we wszelkim odwiedzonym hrabstwie dokładnie jeden dzionek pobędzie. Rufin tą przymusowością strapił się bardzo, azaliż synowi przyrzeczenie dał. Wszelako sam mu szlak zdeklarował się wyznaczyć taki, ażeby Bratumił na jak największą liczbę księżniczek okiem rzucił.

Wtórą noc już Rufin oka zmrużyć nie może. Jak trasę wyznaczyć? Na szczęście informatyk z eskapady dalekiej akuratnie przybył i laptop najnowszej generacji z procesorem wielordzeniowym sprowadził. Tedy król pchnął umyślnego po nadwornego, by mu ową trasę ustalił.

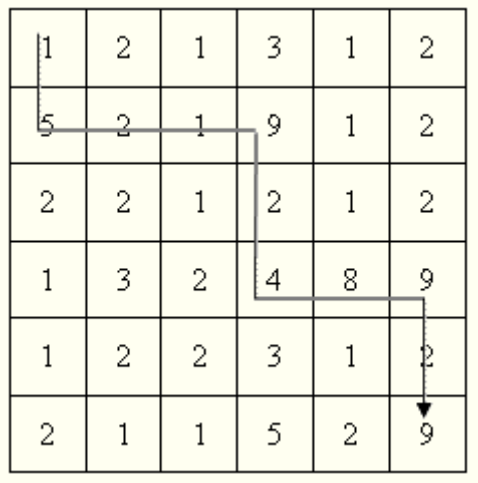
Zadanie

Mapa państwa to tablica kwadratowa o wymiarach n x n. Każdy kwadrat określa jedno hrabstwo, w każdej komórce znajduje się liczba zamieszkujących go księżniczek.

Napisz program, który dla danej tablicy liczb n x n oblicza ścieżkę o maksymalnej liczbie księżniczek prowadzącą od lewego górnego pola tablicy do prawego dolnego. Ścieżka składa się z sekwencji kroków. Liczba kroków wynosi dokładnie $2 \cdot n - 2$.

Maksymalną liczbą księżniczek ścieżki jest suma liczb w każdym z odwiedzonych pól tablicy.

Maksymalna ścieżka w przykładowej tablicy o wymiarze 6 x 6 pokazana jest poniżej.



Wejście

W pierwszym wierszu wejścia zapisana została jedna liczba całkowita n, oznaczającą liczbę wierszy oraz kolumn tablicy. Dalej następuje n x n liczb naturalnych nie większych niż 1000, w porządku wiersz-kolumna, tj. pierwszych n liczb tworzy pierwszy wiersz tablicy, następnych n liczb tworzy drugi wiersz itd. Liczby w wierszu są od siebie oddzielone przynajmniej jedną spacją. Liczba wierszy i kolumn jest z zakresu od 1 do 1000 włącznie.

Wyjście

Pierwsza i jedyna liczba określa maksymalną liczbę księżniczek, jakie może poznać Bratumił.

Przykład

Wejście	Wyjście
3 1 1 1 1 1 1 1 1 1	5
Wejście	Wyjście
6 1 2 1 3 1 2 5 2 1 9 1 2 2 2 1 2 1 2 1 3 2 4 8 9 1 2 2 3 1 2 2 1 1 5 2 9	52

Rozwiązanie

Obserwacja: Zauważmy, że Bratumił może się poruszać po planszy wyłącznie w prawo lub w dół (w innym wypadku liczba odwiedzonych hrabstw byłaby zbyt duża).

Rozwiązanie wolne: Możliwe jest generowanie wszystkich możliwych do uzyskania ścieżek i wyznaczanie dla nich sumy.

Zadanie dla uczniów: ile różnych ścieżek jest możliwych do wygenerowania dla tablicy n x n?

Rozwiązanie szybkie: Korzystając z obserwacji podanej na wstępie można zauważyć, że do każdego hrabstwa w pierwszym wierszu możemy przyjść wyłącznie z lewej strony. Podobnie do każdego hrabstwa w pierwszej kolumnie możemy przyjść wyłącznie z góry. Pozwala to w prosty sposób wyznaczyć liczbę księżniczek dla każdego pola w pierwszym wierszu i w pierwszej kolumnie. Idąc tym tokiem myślenia można zauważyć, że do hrabstwa o współrzędnych (2,2) możemy przyjść wyłącznie z hrabstw (2,1) lub (1,2). Oczywiście aby zmaksymalizować liczbę księżniczek będziemy wybierać to hrabstwo, które oferuje nam wyznaczoną dotychczas większą sumę. Podobnie postąpimy dla kolejnych pól w wierszu nr 2, a następnie kolejno w pozostałych wierszach.

Deklarując tablicę z pustym wierszem i kolumną o indeksie 0 (z wartościami 0 dla tych pól) nasz algorytm pozostanie stały bez dodatkowego wyznaczania wartości dla pierwszego wiersza i pierwszej kolumny.

```
Tablica a[0..n]
wczytaj n
dla y=1,2,…,n wykonaj
      dla x=1,2,…,n wykonaj
```

```
          wczytaj a[x,y]
dla y=1,2,…,n wykonaj
     dla x=1,2,…,n wykonaj
          a[x,y] ← maksimum(a[x-1,y],a[x,y-1]) + a[x,y]
wypisz a[x,y]
```

Zadanie dla uczniów: Jak odtworzyć ścieżkę, po której poruszał się Bratumił?

Zadanie 3. Zbiórka

Limit pamięci: 64MB

Pałac króla Bitolandii jest w ruinie, a kasa państwa świeci pustkami! Obywatele Bitolandii postanowili przeprowadzić zbiórkę pieniędzy na ratowanie cennego zabytku. W centrum stolicy, Bitogrodu, ustawiono wielką skrzynię, do której każdy może wrzucić datek. Zbiórka trwa już jakiś czas. Nikt jednak nie wie, jaka kwota mogła się dotychczas uzbierać. Gdyby tyko społeczny komitet ratowania zabytku miał pewność, że zebrane środki są wystarczające, zakończono by kwestę i przystąpiono do prac. W Bitolandii używa się tylko monet, a każda ma określoną wagę i nominał. Czy znając wagę zebranych monet możliwe jest określenie minimalnej kwoty, jaką do tej pory zebrano?

Wejście

Pierwszy wiersz zawiera jedną liczbę całkowitą c – ciężar zgromadzonych pieniędzy, nie więcej niż 10000. W drugiej linii znajduje się jedna liczba całkowita n – liczba różnych monet używanych w danej walucie ($1 \le n \le 1000$). W kolejnych liniach znajdują się opisy monet zawierające po dwie liczby całkowite, nominał m oraz wagę w ($1 \le m \le 50000$, $1 \le w \le 10000$).

Wyjście

Wypisz minimalną ilość pieniędzy, które można osiągnąć za pomocą monet z danej masy całkowitej. Jeśli waga nie może być osiągnięta, należy wypisać komunikat "NIEMOZLIWE".

Przykład

Wejście:	Wyjście:
100 2 1 10 20 5	10

Wejście:	Wyjście:
100 2 10 1 20 5	400

Wejście:	Wyjście:
5 2 2 2 4 4	NIEMOZLIWE

Rozwiązanie

Zauważmy, że rozwiązanie zadania to klasyczny problem plecakowy, jednak zamiast szukać wartości maksymalnej kwoty dla podanej wagi, szukamy wartości minimalnej.

W tablicach `waga_monet[]` oraz `wartość_monet[]` zapamiętajmy odpowiednio wagi i wartości poszczególnych monet (nie więcej niż 1000). Zauważmy, że maksymalna waga wyniesie 10000, potrzebujemy więc jeszcze tablicy `najlepsza[]` przechowywujące najmniejsze wartości dla każdej możliwej do uzyskania wagi.

Najlepsza możliwa wartość do uzyskania wagi 0 wynosi oczywiście 0. Teraz dla każdej wagi aż do wagi całej zawartości skrzyni będziemy sprawdzali, czy biorąc po kolei każdą monetę o określonej wadze mogę uzyskać lepszą (mniejszą) kwotę.

Na koniec sprawdzamy, czy w `najlepsza[szukana_waga]` została przez nas znaleziona jakaś waga.

```
tablica waga_monet[1..1000]
tablica wartość_monet[1..10000]
tablica najlepsza[1.1000]
wczytaj c, n
dla i=1,2,…,n
  wczytaj wartość_monet[i], waga_monet[i]
najlepsza[0] ← 0
dla j=1,2,…,c
  najlepsza[j] ← ∝
    dla i=1,2,…,n
      jeżeli waga_monet[i] ≤ j
        jeżeli najlepsza[j-waga[monet[i]]+wartość_monet[i]<najlepsza[j]
          najlepsza[j] ← najlepsza[j-waga[monet[i]]+wartość_monet[i]
jeżeli najlepsza[c] < ∝
  wypisz najlepsza[c]
przeciwnie
  wypisz „NIEMOŻLIWE”
```

Zajęcia 18

Temat: Struktury danych 1

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, zbiory i operacje na zbiorach, struktury danych, biblioteka STL, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem funkcji rekurencyjnej, równaniami rekurencyjnymi,
- zna struktury danych: stos, kolejkę, vector, pair, kolejka priorytetowa

Formy i metody pracy: praca samodzielna, wykład, omówienie

Zadania do wykonania na zajęciach	Treści programowe
1. Mnóż, odejmuj, dodawaj (mod)	M.3, P.2.16, P.2.18, A.4.1, A.4.2
2. Wodzirej (wod)	M.3, P.2.16, P.2.18, A.4.2, A.4.5
3. Gorące pudełko (gpuzad)	M.3, P.2.16, P.2.20, A.4.5, A.4.9

Materiały do zajęć:

https://en.cppreference.com/w/

Zadania do wykonania w domu:

Plakatowanie (OI)

https://szkopul.edu.pl/problemset/problem/au-E9FH96-3U9rCKhcNsD5n9/site

ZADANIA I ROZWIĄZANIA:

Zadanie 1. Mnóż, odejmuj, dodawaj

Limit pamięci: 32MB

Czy dodawanie, odejmowanie i mnożenie może sprawiać kłopoty? Oczywiście że nie! Dlatego mam dla ciebie proste zadanie: dla podanego w specjalnym zapisie wyrażenia arytmetycznego oblicz jego wartość! Specjalność tego zapisu polega na tym, że najpierw zawsze pojawiają się argumenty, a dopiero później działania M, O lub D (mnożenie, odejmowanie lub dodawanie). W wyrażeniu tym jednak nie musisz się martwić kolejnością działań arytmetycznych – po prostu wykonuj je zgodnie z wejściem!

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba N ($1 \le N \le 10$) – liczba wyrażeń do rozpatrzenia. W kolejnych N liniach znajdują się krótkie wyrażenia arytmetyczne. Długość żadnego z nich nie przekracza 1000 znaków. Wyrażenia składają się wyłącznie z wartości x_i ($0 \le x_i \le 10^{15}$) oraz działań opisanych odpowiednimi literami.

Wyjście

W oddzielnych liniach należy wypisać N liczb całkowitych – wyniki obliczeń. Możesz założyć, że końcowy wynik nigdy nie przekroczy $|10^{15}|$.

Przykład

Wejście	Wyjście
5 4 5 D 10 20 O 2 3 M 5 D 2 3 5 M D 5 5 M 20 O 2 2 D D	9 -10 11 17 9

Wyjaśnienie:

Kolejne wyrażenia to odpowiednio: 4+5, 10–20, 2*3+5, 2+3*5, 5*5–20+2+2

Rozwiązania

W zadaniu mamy do czynienia z zapisem wyrażeń arytmetycznych w odwrotnej notacji polskiej. Aby rozwiązać to zadanie wykorzystamy stos (można wykorzystać `stack <long long>` z biblioteki STL). Wyznaczanie wartości wyrażenia odbywa się według poniższego algorytmu (dla jednej linii tekstu):

```
stos liczb całkowitych S
dla wszystkich elementów w linii
     pobierz kolejny element z wejścia
     jeżeli element jest liczbą
          dodaj liczbę na stos
     jeżeli element jest operacją arytmetyczną
          zdejmij ze stosu ostatnia liczbę x1
          zdejmij ze stosu ostatnia liczbę x2
```

```
            jeżeli element = 'M'
                 dodaj na stos x1*x2
            jeżeli element = 'D'
                 dodaj na stos x1+x2
            jeżeli element = 'O'
                 dodaj na stos x2-x1
zdejmij ze stosu ostatnia liczbę x
wypisz x
```

Zadanie dla uczniów: Jak można przy wczytywaniu liczb wykorzystać schemat Hornera?

Zadanie 2. Wodzirej

Dostępna pamięć: 32 MB

Janek urządził imprezę. Tym razem postanowił wybrać jedną osobę, która będzie decydowała o przebiegu zabaw – wodzireja. Chciałby, aby osoba ta była lubiana przez jak największą grupę uczestników spotkania. Pomóż wskazać mu tę osobę oraz grono osób, które ją lubią.

Wejście

Pierwszy wiersz danych zawiera dwie liczby całkowite n i m, odpowiednio liczbę osób biorących udział w przyjęciu oraz liczbę wzajemnych sympatii ($2 \le n \le 10\,000$, $2 \le m \le 1\,000\,000$). W kolejnych m liniach znajdują się po dwie liczby całkowite u_i i v_i ($1 \le u_i, v_i \le 10\,000$) – informacje o parach osób, które się lubią wzajemnie.

Wyjście

Program powinien wypisać w pierwszym wierszu jedną liczbę – najbardziej lubianą osobę na przyjęciu (wodzireja). Jeśli jest więcej takich osób – tę o najmniejszej liczbie porządkowej.

W drugiej linii program powinien wypisać w kolejności rosnącej wszystkie osoby, które lubią wodzireja.

Przykład

Wejście	Wyjście
5 4 1 3 2 3 4 2 5 2	2 3 4 5

Rozwiązanie

Ponieważ nie możemy z góry założyć, z iloma uczestnikami lubi się każdy z imprezowiczów, wykorzystajmy do zapamiętywania znajomych tablicy dynamicznej (można wykorzystać `vector <int>` z biblioteki STL). Następnie odnajdziemy pierwszy najdłuższy ciąg, posortujemy go i wypiszemy. Poniżej kod zadania w C++ wraz komentarzami:

```
stos liczb całkowitych S

//lista znajomych
```

```
vector <int> V[10001];
cin >> n >> m;
//dla wszystkich par sympatii
for (int i=0; i<m; i++) {
  //wczytaj numery pary lubiących się wzajemnie znajomych
  cin >> u >> v;
  //dodaj do listy znajomych u znajomego v
  V[u].push_back(v);
  //dodaj do listy znajomych v znajomego u
  V[v].push_back(u);
}
//zakładamy, że nikt nie jest najbardziej lubiany
maximum = 0;
//dla wszystkich imprezowiczów
for (int i=1; i<=n; i++)
  //jeżeli jakiś imprezowicz ma więcej lubiących go znajomych
  //niż dotychczasowy najlepszy imprezowicz, zapamiętaj jego numer
  if (V[maximum].size() < V[i].size()) maximum = i;
//wypisz numer najbardziej lubianego imprezowicza
cout << maximum << endl;
//posortuj jego znajomy numerami rozną co
sort ( V[maximum].begin(), V[maximum].end());
//wypisz wysztkich znajomych naszego ulubieńca
for (int i=0; i< V[maximum].size(); i++)
  cout << V[maximum][i] << " ";
```

Uwaga: Warto wrócić do treści tego zadania na zajęciach nr 21 (grafy) i wykazać, że w zadaniu wodzirej zaimplementowaliśmy już graf.

Zadanie 3. Gorące pudełko

Dostępna pamięć: 256MB

Czarująca pani Char, przedszkolanka w przedszkolu Słoneczny Integer, lubi urządzać dla swoich podopiecznych różne zabawy. Jedną z nowych zabaw, które pani Char zaproponowała dzieciom, jest zabawa w gorące pudełko. Polega ona na tym, że pani Char wręcza pierwszemu dziecku kolorowe pudełko, ono podaje je drugiemu, drugie - trzeciemu, itd.. Ostatnie dziecko podaje pudełko pierwszemu.

W czasie zabawy gra muzyka. Gdy melodia ucichnie, dziecko, które trzyma pudełko, dostaje pyszne ciasteczko i kończy zabawę. Pani Char przerywa muzykę w różnych nieoczekiwanych dla dzieci momentach. Pudełko porusza się raz w prawo, raz w lewo. Zabawa trwa tak długo, aż zostanie tylko jedno dziecko - zwycięzca gry.

Każdy z maluchów chciałby oczywiście wygrać zabawę. Mały Bitek zauważył w pewnym momencie, że pani Char zawsze używa takiej samej sekwencji ruchów. Zauważył też, że może wybrać takie miejsce wśród dzieci, które zapewni mu wygraną. Zastanawia się teraz, które to miejsce. Pomożesz mu?

Wejście

W pierwszej linii wejścia znajduje się jedna liczba całkowita n ($2 \leq n \leq 10^6$), oznaczająca liczbę dzieci w przedszkolu.

W kolejnych n-1 liniach wejścia znajduje się sekwencja ruchów wykonywanych przez panią Char, przy czym i-ty ruch opisany jest przez liczbę całkowitą x_i ($1 \le x_i \le 100$) oznaczającą liczbę przekazań przez dzieci pudełka, po których ucichnie muzyka. Ruchy wykonywane są na przemian raz w prawo, raz w lewo, przy czym pierwszy ruch wykonywany jest zawsze w prawo. Dziecko, które odpadło, oddaje pudełko temu dziecku, od którego je otrzymało.

Wyjście

Na wyjściu powinna znaleźć się jedna liczba całkowita oznaczająca liczony od lewej numer miejsca, które powinien zająć mały Bitek, żeby wygrać zabawę.

Przykład

Wejście	Wyjście
5 3 2 5 1	3

Objaśnienie przykładu: Pogrubieniem oznaczmy położenie pudełka. Na początku otrzymuje gorące pudełko zawsze pierwsze dziecko: [1,2,3,4,5]. Muzyka cichnie po trzech podaniach, więc odpada dziecko czwarte: [1,2,3,5]. Teraz pudełko przechodzi o dwa w lewo, odpada dziecko pierwsze: [2,3,5]. Dalej - cyklicznie - pięć razy w prawo - odpada piąte: [2,3]. I raz w lewo - zostało tylko dziecko trzecie: [3].

Ocenianie

Podzadanie	Ograniczenia	Punkty
1	$n \le 10^3$	30
2	brak dodatkowych założeń	70

Rozwiązanie

Zauważmy, że często będziemy usuwać elementy ze zbioru. W zadaniu tym możemy więc skorzystać strukturę reprezentującą zbiór (np. set z STL). Ciekawszym rozwiązaniem będzie jednak wykorzystanie listy cyklicznej (użyjemy `list <int>` z biblioteki STL pamiętając, by zawsze ostrożnie przechodzić pomiędzy pierwszym i ostatnim elementem listy). Kod programu stanowi symulację przekazywania pudełka pomiędzy dziećmi.

```
list<int> lista;
list<int>::iterator p, q;
int n, x;
//dodaj n dzieci do listy
cin >> n;
for (int i=1; i<=n; i++) lista.push_back(i);
//zapamiętaj początek listy
p=lista.begin();
//dla n-1 dzieci
for (int i=0; i<n-1; i++){
  //wczytaj liczbę dzieci, które przekażą sobie pudełko
  cin >> x;
  //idź w prawo o jeden ruch za mało, iterator zwiększaj
```

```
  if (i%2==0) {
    for (int j=1; j<x; j++){
      p++;
      //jeżeli doszedłeś do końca listy dzieci, wróć na początek
      if (p==lista.end()) p=lista.begin();
    }
    //dziecko q do usunięcia z kolejki jest za dzieckiem p
    q=p;
    q++;
    //ostrożnie z końcem listy
    if (q==lista.end()) q=lista.begin();
    //teraz możesz usunąć dziecko z kolejki
    lista.erase(q);
  }
  //podobnie idziesz w lewo, iterator zmniejszasz
  else {
    for (int j=1; j<x; j++){
    //również ostrożnie z przejściem przez początek-koniec listy
      if (p==lista.begin()) p=lista.end();
      p--;
    }
    //usuń z kolejki poprzednie dziecko
    q=p;
    if (q==lista.begin()) q=lista.end();
    q--;
    lista.erase(q);
  }
//w kolejce zostało jedno dziecko, wypisz go
cout << lista.front();
```

Zajęcia 19

Temat: Struktury danych 2

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, zbiory i operacje na zbiorach, struktury danych, biblioteka STL, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem funkcji rekurencyjnej, równaniami rekurencyjnymi,
- zna struktury danych: stos, kolejkę, vector, pair, kolejka priorytetowa

Formy i metody pracy: praca samodzielna, pogadanka, dyskusja, omówienie

Zadania do wykonania na zajęciach	Treści programowe
1. Kamienie	M.3, P.2.16, P.2.18, A.4.1, A.4.2
2. Loginy	M.3, P.2.16, P.2.18, P.2.20, A.4.2, A.4.5
3. Izba przyjęć	M.3, P.2.16, P.2.18, P.2.20, A.4.5, A.4.9

Materiały do zajęć

https://en.cppreference.com/w/

Zadania do wykonania w domu:

Kodowanie permutacji (II OI):

https://szkopul.edu.pl/problemset/problem/ioeAqb7fRkdNzdUVS0cvMSKm/site/

Samochodziki (XII OI):

https://szkopul.edu.pl/problemset/problem/DNXEM9WX4TRCI0fFbWedf1qq/site/

ZADANIA I ROZWIĄZANIA:

Zadanie 1. Kamienie

Limit pamięci: 64MB

Janek jest kolekcjonerem kamieni. Zbiera je i kataloguje, nadając każdemu unikatowy numer. Dba o to, by każdy kamień był inny i nie pozwala, aby w kolekcji znalazły się dwa takie same. Kiedy chodzi plażą, gdzie może znaleźć nowe okazy do swoich zbiorów, podnosi każdy napotkany kamień, ocenia go, nadaje mu numer katalogowy i – jeśli jeszcze takiego w kolekcji nie posiada – zabiera go. Trudno jest mu jednak szybko ocenić, czy kolejny znaleziony kamień jest potrzebny. Pomóż mu!

Wejście

W pierwszym wierszu standardowego wejścia zapisana jest jedna liczba całkowita n ($2 \le n \le 1\,000\,000$) oznaczająca liczbę kamieni. Kolejne n wierszy zawiera po jednej liczbie całkowitej k_i ($0 \le k_i \le 10^9$) oznaczającej numer katalogowy i-tego kamienia.

Wyjście

W n wierszach standardowego wyjścia Twój program powinien zapisać po jednej literze ‘T’ lub ‘N’, oznaczającą przydatność kamienia w kolekcji Janka.

Przykład

Wejście	Wyjście
5 1 2 4 2 5	T T T N T

Rozwiązanie

Bardzo proste zadanie, którym możemy użyć struktury `set` z biblioteki STL.

```
wczytaj n
dla i=1,2,…,n
      wczytaj x
      jeżeli x nie należy do zbioru
            wypisz ‘T’
            dodaj x do zbioru
      przeciwnie
            wypisz ‘N’
```

Jeżeli będzie to wskazane, można wytłumaczyć na zajęciach ideę drzew BST (zasada działania, bez implementacji) oraz drzew samorównoważących się w odniesieniu do kontenerów set i map z biblioteki STL wraz z informacją o złożoności obliczeniowej poszczególnych operacji (wstawienia, usunięcia, szukania, przejścia do poprzedniego/następnego).

Zadanie 2. Loginy

Dostępna pamięć: 128MB

Pan Integer pracuje nad otwarciem nowego portalu internetowego. Jednym z elementów systemu jest poczta elektroniczna. Pan Integer rozdzielił już pracę pomiędzy zespoły. Twoim zadaniem będzie opracowanie modułu rejestracji loginów.

Moduł rejestracji otrzymuje proponowany login. Jeśli taka nazwa jeszcze nie wystąpiła w bazie, moduł wysyła informację *OK* do użytkownika i zapisuje go w bazie. Jeśli podana nazwa już występuje w bazie, wówczas tworzony jest nowy login, system wysyła go do użytkownika i zapisuje w bazie. Nowy login tworzony jest przez dodanie kolejnego jak najmniejszego numeru do proponowanej przez użytkownika nazwy.

Wejście

W pierwszej linii wejścia znajduje się jedna liczba całkowita n ($1 \; n \; 5 \cdot 10^5$), oznaczająca liczbę rozpatrywanych loginów.

W kolejnych n liniach znajdują się rozpatrywane loginy. Każdy login składa się wyłącznie z małych liter alfabetu łacińskiego i jest nie dłuższy niż 30 znaków.

Wyjście

Na wyjściu dla każdego loginu wypisywana jest odpowiedź wysyłana dla użytkownika: *OK* – w przypadku udanej rejestracji, lub nowy login dla już istniejącej nazwy.

Przykład

Wejście	Wyjście
6 mirek katarzyna kirek mirek nowy mirek	OK OK mirek1 mirek2 OK mirek3

Ocenianie

Podzadanie	Ograniczenia	Punkty
1	$n \leq 10^3$	30
2	brak dodatkowych założeń	70

Rozwiązanie

Bardzo proste zadanie, którym możemy użyć struktury `map` z biblioteki STL.

Zauważmy, że kluczem w naszej mapie będzie wczytany login, zaś wartością liczba dotychczas użytych takich loginów. Dla ułatwienia kod programu z komentarzami:

```
string s;
//lista loginów
map<string,int>a;
//liczba zapytań
```

```
cin>>n;
//dla każdego zapytania
for (int i=0; i<n; i++){
  //wczytaj oczekiwany login
  cin >> s;
  //jeżeli podany login istnieje, wartość elementu
  //o podanym kluczu będzie większa od 0
  if ( a[s]>0 )
  //wówczas wypisujemy login wraz z liczbą dotychczasowych wystąpień
    cout << s << a[s] << "\n";
  //jeżeli login jeszcze nie wystąpił
  else
    //wypisz login bez zmian
    //cout << "OK" << "\n";
  //zwiększ liczność aktualnego loginu o 1
  a[s]++;
}
```

Zadanie 3. Izba przyjęć

Dostępna pamięć: 256MB

Na izbie przyjęć chorzy pacjenci oczekują na przyjęcie z przejęciem patrząc na zegarek. Ale to nie czas decyduje o kolejności. Ważniejszy jest stan chorego. Dlatego często na SORach oznacza się pacjentów opaskami w kolorach czerwonym, żółtym i zielonym. Kolor czerwony oznacza, że pacjent musi być przyjęty w pierwszej kolejności - w trybie natychmiastowym. Taki kolor przydzielany jest osobom w bardzo złym stanie, często przywiezionym przez karetki. Kolor żółty jest przydzielany pacjentom, którzy wymagają pilnej interwencji lekarza. Kolorem zielonym są oznaczani pacjenci z najniższym stopniem zagrożenia życia, przyjmowani w trzeciej kolejności.

Pan Integer opracował dokładniejszy sposób określania stanu pacjentów. Wprowadził *k* kolorów opasek. Jeśli mamy pacjentów z opaskami w kolorach *i* oraz *j* i wiemy, że *i<j*, to pacjent z opaską w kolorze *i* zostanie przyjęty jako pierwszy. Dla pacjentów z tym samym kolorem opaski liczy się czas przyjęcia.

Pan Integer wymyślił system, który ma przyśpieszyć pracę pobliskiej izby przyjęć i obserwuje teraz, jak działa on w rzeczywistości. Co sekundę przychodzi nowy pacjent lub jeden z oczekujących pacjentów jest przyjmowany przez dyżurującego lekarza. Pan Integer potrafi natychmiast sklasyfikować przybyłych (określić kolor ich opasek). Nie nadąża jednak ze wskazywaniem pacjentów, którzy mają być w danym momencie obsłużeni. Czy potrafisz napisać program, który pomoże mu w pracy?

Wejście

W pierwszej linii wejścia znajduje się jedna parzysta liczba całkowita n ($1 \le n \le 2 \cdot 10^6$). W kolejnych n liniach znajduje się po jednej liczbie całkowitej *ki* oznaczającej kolor opaski pacjenta, który przyszedł w *i*-tej sekundzie ($1 \le k_i \le 109$) lub informacja o przyjęciu w tym momencie kolejnej osoby przez lekarza ($k_i = -1$).

Możesz założyć, że zawsze jest ktoś, kto oczekuje pomocy i że wszyscy pacjenci zostaną kiedy obsłużeni. Możesz również założyć, że testach ocenianych na 30 punktów zachodzi warunek $n \le 10^3$.

Wyjście

Na wyjściu powinno znaleźć się *n* liczb całkowitych oznaczających kolejność przyjęcia pacjentów przez lekarza. Pacjentów oznacz czasem przyjścia na izbę przyjęć.

Przykład

Wejście	Wyjście
10 3 5 2 -1 4 -1 1 -1 -1 -1	3 1 7 5 2

Rozwiązanie

W tym zadaniu najwygodniejsze będzie użycie kolejki priorytetowej (struktura `priority_queue` z biblioteki STL). Nasza kolejka uwzględnia przede wszystkim stan pacjenta (kolor), a dopiero później jego czas przybycia. Zadeklarujmy strukturę, która będzie przechowywać te dwie informacje. Zauważmy, że kolejka priorytetowa przechowuje na początku element najważniejszy. Uwzględnimy ten fakt przeciążając operator mniejszości (wymaganie kolejki priorytetowej). Teraz nie zostało nam nic innego niż zasymulować pracę izby przyjęć.

Kod programu wraz z komentarzami:

```
struct pacjent {int kolor, czas;};
//pacjent pomocniczy
pacjent p;
priority_queue <pacjent> q;
//przeciążamy operator mniejszości, liczy się
//w pierwszej kolejności kolor, mniej ważny jest ten o większym
numerze
bool operator <(const pacjent &a, const pacjent &b){
  return (a.kolor>b.kolor) || (a.kolor==b.kolor && a.czas>b.czas);
}
```

Główna część programu:

```
//liczba zdarzeń na izbie przyjęć
cin>>n;
for (int i=1; i<=n; i++){
  //kolor pacjenta z sekundy i
  cin >> p.kolor;
  //jeżeli kolor jest równy -1, to informacja o przyjęciu
  //najbardziej potrzebującego pacjenta
  if (p.kolor==-1) {
    //zdejmujemy z kolejki pierwszego oczekującego
```

```
    p = q.top(); q.pop();
    //wypisz jego czas przyjęcia na izbę
    cout << p.czas << "\n";
  }
  //nowy pacjent
  else {
    //zapamiętaj jego czas przyjęcia
    p.czas=i;
    //wstaw go do kolejki oczekujących
    q.push(p);
  }
}
```

Zajęcia 20

Temat: Zawody 2. Powtórzenie i podsumowanie.

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice dwuwymiarowe, rekurencję, pisze własne funkcje rekurencyjne, struktury danych, biblioteka STL, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem funkcji rekurencyjnej, równaniami rekurencyjnymi,
- zna struktury danych: tablica prefiksowa,
- zna porządek częściowy i liniowy,
- zna wyszukiwanie binarne

Formy i metody pracy: praca samodzielna w formie zawodów, sparingu

Zadania do wykonania na zajęciach	Treści programowe
1. Dobro i zło	M.3, P.2.13, P.2.16, A.3.2, A.4
2. Bursztyn	M.3, P.2.16, P.2.19, A.3.2, A.4
3. Na rzymskie	M.3, P.2.16, P.2.20, A.3.2, A.4

Podpowiedzi do rozwiązań

Zadanie 1. Dobro i zło. Proste wyszukiwanie binarne po wyniku. Utrudnieniem jest konieczność skorzystania z biblioteczki.

Zadanie 2. Bursztyn. Zadanie należy rozwiązać w oparciu o programowanie dynamiczne. Najłatwiej wykorzystać sumy prefiksowe w tablicy dwuwymiarowej.

Zadanie 3. Na rzymskie. Wystarczające będzie użycie algorytmu zachłannego. Przygotujmy sobie dwie tablice ze wszystkimi możliwymi wartościami „cyfr” oraz cyframi w zapisie rzymskim, a następnie wypisujmy każdą „cyfrę” odpowiednią liczbę razy od począwszy tej o największej wartości.

ZADANIA I ROZWIĄZANIA:

Zadanie 1. Dobro i zło

Limit pamięci: 64MB Limit czasu: 0.5s

Walka dobra ze złem w Vilolandzie trwa w najlepsze. Niedługo rozpęta się bitwa, która zadecyduje o wszystkim, tylko... no właśnie... gdzie mogłaby się odbyć?

Viloland, jak wiadomo, jest światem w kształcie prostokąta podzielonego na 1 x *n* kwadratowych pól. Każde pole jest kontrolowane albo przez dobro, albo przez zło, przy czym wiadomo, że pole numer 1 jest kontrolowane przez dobro, a pole numer *n* przez zło. To, przez kogo jest kontrolowane dane pole, może się dowolnie przeplatać i każde ze „środkowych" $n - 2$ pól nie jest w żaden sposób zależne od innych.

Wielka bitwa musi odbyć się na dwóch sąsiednich polach, z których lewe (to o mniejszym indeksie) jest kontrolowane przez dobro, a prawe przez zło. Póki co przywódcy obu stron tylko się kłócą, ale w końcu bitwa musi się odbyć. Pomóż znaleźć odpowiednie miejsce!

Komunikacja

Twój program powinien używać biblioteki, która pozwala na zadawanie pytań o to, kto kontroluje poszczególne pola, oraz na udzielenie ostatecznej odpowiedzi. Aby użyć biblioteki, należy wpisać na początku programu:

```
#include "doblib.h"
```

Biblioteka udostępnia następujące funkcje:

- `long long inicjuj();`

  Inicjalizuje bibliotekę. Należy wywołać ją tylko raz, na początku programu, przed użyciem pozostałych funkcji. Zwraca liczbę *n*, oznaczającą liczbę pól w Vilolandzie.

- `int sily(long long x);`

  Dla każdego *x* spełniającego warunek $1 \leq x \leq n$ zwraca 1, jeśli pole numer *x* jest kontrolowane przez siły dobra, 2, jeśli przez siły zła. W szczególności `sily(1)` zwróci 1, `sily(n)` zwróci 2. **Uwaga! Możesz użyć tej funkcji co najwyżej 100 razy!**

- `void odpowiedz(long long x);`

  Udziela odpowiedzi, że na polu numer *x* panują siły dobra, zaś na polu numer $x + 1$ - siły zła. Wywołanie tej funkcji zakończy działanie twojego programu.

Twój program **nie może** czytać żadnych danych (ani ze standardowego wejścia, ani z plików). **Nie może** również nic wypisywać do plików ani na standardowe wyjście. Może pisać na standardowe wyjście diagnostyczne (`stderr`) - pamiętaj jednak, że zużywa to cenny czas.

Przykładowy przebieg programu

Załóżmy, że $n = 7$, a rozkład sił na kolejnych polach wygląda tak:

1121222

gdzie podobnie jak wyżej 1 oznacza kontrolę sił dobra, a 2 - sił zła. Poprawne odpowiedzi to $x = 2$ oraz $x = 4$. Interakcja programu z biblioteką może wyglądać następująco:

Wywołanie	Zwrócona wartość
`inicjuj()`	-
`sily(2)`	1
`sily(3)`	2
`odpowiedz(2)`	-

Ocenianie

Podzadanie	Ograniczenia	Punkty
1	$n \leq 10^2$	20
2	$n \leq 10^{18}$	80

Eksperymenty

W zakładce Pliki znajduje się archiwum `dob_dlazaw.zip` zawierające:

- Przykładową bibliotekę `doblib.h`, która pozwoli Ci przetestować poprawność formalną rozwiązania. Aby jej użyć, umieść ją w tym samym folderze co swoje rozwiązanie, a następnie skompiluj je. Tak otrzymany program wczytuje z wejścia liczbę *n*, a następnie ciąg *n* znaków równych '1' lub '2', w zależności od tego czy dane pole kontrolowane jest przez siły dobra, czy przez siły zła. Pamiętaj jednak, że biblioteka ta różni się od tej, na której zostanie ostatecznie ocenione twoje rozwiązanie.
- Przykładowe, błędne rozwiązanie `dob.cpp`, ilustrujące poprawną komunikację z biblioteką.
- Test przykładowy `dob0.in` w opisanym wyżej formacie przykładowej biblioteki.

Zadanie 2. Bursztyn

Dostępna pamięć:32 MB

Czy wiedzieliście, że w województwie lubelskim są ogromne złoża bursztynu, porównywalne z tymi na wybrzeżu Bałtyku? Wójt jednej z lubelskich gmin postanowił na własną rękę eksploatować złoża znajdujące się na zarządzanym przez niego terenie. W tym celu zabezpieczył już w przyszłorocznym budżecie pewną kwotę na zakup terenów bursztynonośnych. Ze względu na zakupiony sprzęt teren ten musi mieć kształt kwadratu o określonym wymiarze *k*. Wójt sporządził już mapę terenu z informacjami o ilości bursztynu na poszczególnych działkach. Teraz tylko chciałby wybrać taki kwadrat, którego eksploracja przyniesie mu największy zysk.

Wejście

W pierwszym wierszu standardowego wejścia zapisano trzy liczby całkowite dodatnie, oddzielone pojedynczym odstępem: n , m i k, gdzie ($1 \leq n, m, k \leq 1000$). W kolejnych n wierszach zapisano po m liczb całkowitych nieujemnych, które odpowiadają ilości bursztynu na poszczególnych działkach (wielkości te nie przekraczają miliona).

Wyjście

W pierwszym wierszu standardowego wyjścia zapisz największą sumaryczną ilość bursztynu kwadratowego obszaru.

Przykłady

Wejście 5 6 2 1 2 3 4 5 6 5 4 3 2 1 6 3 4 5 6 7 1 4 5 6 7 8 1 8 9 1 2 3 1 Wyjście 28	Wejście 4 5 3 1 2 3 4 5 5 4 3 2 1 3 4 5 6 7 4 5 6 7 8 Wyjście 45	Wejście 4 4 2 1 2 3 4 5 4 3 2 3 4 5 6 4 5 6 7 Wyjście 24

Zadanie 3. Na rzymskie

Dostępna pamięć: 32MB

Wczytaj liczbę zapisaną w systemie dziesiętnym i wypisz jej wartość zapisaną cyframi rzymskimi. Wartości cyfr pojawiających się w testach przedstawione są w tabelce poniżej.

I	1
IV	4
V	5
IX	9
X	10
L	50
C	100
D	500
M	1000

Wejście

Liczba arabska n ($1 \leq n \leq 3000$).

Wyjście

Pierwszy i jedyny wiersz wyjścia powinien wypisać jedno słowo – wartość n w zapisie rzymskim.

Przykład

Wejście	Wyjście
55	LV

Zajęcia 21

Temat: Grafy

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, zbiory i operacje na zbiorach, struktury danych, biblioteka STL, grafy nieskierowane, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem struktury danych – graf nieskierowany, podgrafy, wierzchołek, krawędzie, ścieżki,
- zna metody przechodzenia grafów,
- zna strukturę Find-Union,

Formy i metody pracy: praca samodzielna, omówienie, pokaz, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Dwukolorowanie	M.5, P.2.18, A.3.7, A.3.8, A.4.3
2. Prehistoria	M.5, P.2.18, A.3.7, A. 3.8, A.4.3, A.4.5

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/7/26/

Zadania do wykonania w domu:

Gildie

https://szkopul.edu.pl/problemset/problem/Oys6jiVOIap59lYCHRwDMbNT/site/

Żabka Bajtozja

https://szkopul.edu.pl/problemset/problem/Pbxxhq_YBgWCAXYvKCfJb_HA/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Dwukolorowanie

Dostępna pamięć: 32MB

Twierdzenie o czterech barwach mówi, że każdą mapę przedstawioną na płaszczyźnie można pokolorować z użyciem czterech kolorów w taki sposób, żeby żadne dwa sąsiadujące obszary nie były pomalowane na ten sam kolor. Problem ten był otwarty przez ponad sto lat i został udowodniony dopiero w roku 1976 za pomocą komputera.

Twoim zadaniem jest rozwiązanie prostszego problemu. Sprawdź, czy dany graf (spójny, nieskierowany i bez pętli) jest dwukolorowalny, tzn. czy jego wierzchołki mogą być pomalowane na kolory czerwony i czarny w taki sposób, aby sąsiadujące ze sobą wierzchołki nigdy nie były tego samego koloru.

Wejście

Dane wejściowe składają się z pewnej liczby zestawów testowych. W pierwszym wierszu każdego zestawu znajduje się liczba wierzchołków n ($1 \le n \le 100\,000$). Etykietą każdego wierzchołka jest liczba z zakresu od 0 do n–1. Drugi wiersz zawiera liczbę krawędzi k ($1 \le k \le 1\,000\,000$). Każdy z kolejnych k wierszy zawiera numery dwóch wierzchołków – numery te opisują krawędź.

Wyjście

Sprawdź, czy graf jest dwukolorowalny, i wypisz wynik: TAK – jeśli graf jest dwukolorowalny, lub NIE – w przeciwnym wypadku.

Przykład

Wejście	Wejście
3 3 0 1 2 0 1 2 Wyjście NIE	9 8 0 1 0 2 0 3 0 4 4 5 4 6 4 7 4 8 Wyjście TAK

Rozwiązanie

Zadanie rozpoczyna cykl zajęć związanych z zadaniami grafowymi.

W jaki sposób reprezentować graf w pamięci komputera? W wielu zadaniach będzie używane podobne rozwiązanie jak w zadaniu Wodzirej z zajęć nr 18.

Do zapamiętywania ścieżek pomiędzy wierzchołkami wykorzystamy tablicę dynamiczną (można wykorzystać `vector <int>` z biblioteki STL). Poniżej kod w C++.

```
//lista sąsiedztwa
vector <int> V[100001];
odwiedzony[1000001];
cin >> n >> k;
//dla wszystkich par wierzchołków
for (int i=0; i<m; i++) {
```

```
  //wczytaj numery wierzchołków
  // i dodaj je naprzemiennie do listsąsiedzwa
  cin >> u >> v;
  V[u].push_back(v); //dodaj do listy znajomych u znajomego v
  V[v].push_back(u); //dodaj do listy znajomych v znajomego u
}
```

Teraz musimy pokolorować graf na dwa kolory. W tym celu użyjemy tablicy `odwiedzony[]`, która będzie przechowywała informację o stanie wierzchołka: 0 – nieodwiedzony, 1 lub 2 – kolor wierzchołka.

W zadaniu wykorzystamy metodę przeszukiwania grafu wszerz. Zaczniemy przeglądanie wierzchołków od dowolnego z nich (na przykład pierwszego) i nadamy mu kolor 1. Odwiedzimy jego sąsiadów. Każdy nieodwiedzony zostanie dodany do kolejki do sprawdzenia w przyszłości oraz nadamy mu kolor przeciwny do wierzchołka, z którego do niego dotarliśmy (1 lub 2). Co w przypadku, kiedy trafimy na wierzchołek już odwiedzony? Jeżeli jego kolor jest taki sam, jak wierzchołka, z którego przyszliśmy, nie jest możliwe dwukolorowanie grafu. W przeciwnym wypadku sprawdzamy dalej.

Poniżej uproszczony algorytm:

```
kolejka Q
BFS (w)
  odwiedzony[w] ← 1
  Q.dodaj(w)
  dopóki (Q nie jest pusta)
    w ← Q.zdejmij_pierwszy_wierzchołek()
    dla wszystkich sąsiadów v wierzchołka w
      jeżeli (odwiedzony[v] = 0)
        odwiedzony[v] ← 1+odwiedzony[w] mod 2 //kolor przeciwny do w
        Q.dodaj(v)
      przeciwnie
        jeżeli (odwiedzony[v] = odwiedzony[w])
          zwróć FAŁSZ i zakończ funkcję
  zwróć PRAWDA // przeszukaliśmy graf i nie znaleźliśmy kolizji
```

W głównej części programu (po wcześniejszym wczytaniu struktury grafu) wystarczy wywołać funkcję BFS i sprawdzić zwracany przez nią wynik:

```
jeżeli (BFS (1) = PRAWDA)
  wypisz TAK
przeciwnie
  wypisz NIE
```

Zadanie dla uczniów: Jaka jest złożoność obliczeniowa tego programu (przeanalizuj przede wszystkim, ile razy zostanie odwiedzony każdy z wierzchołków).

Zadanie 2. Prehistoria

Limit pamięci: 64 MB

Na dziewiczych, zamieszkałych przez dinozaury, mamuty, wiewiórki biegające za żołędziem i inne dziwne stwory terenach rozciąga się kraina zwana przez jej pierwotnych mieszkańców

Bajtolandią. Na owym obszarze żyje *n* plemion. Miedzy niektórymi z nich są dwukierunkowe drogi, których jest w sumie *m*. Wodzowie postanowili połączyć ich plemiona w sojusze, aby łatwiej było im przetrwać. Zasada łączenia jest prosta: plemię *a* zawiąże sojusz z plemieniem *b* jeśli:

- istnieje droga łącząca plemię *a* i plemię *b*,

lub

- istnieje takie plemię *c*, że zarówno plemię *a* i plemię *b* mają z nim sojusz.

Proces zawierania sojuszy trwa tak długo, aż nie da się zawrzeć żadnego innego.

Ludzie zamieszkujący te tereny są prymitywni, aczkolwiek bardzo inteligentni, więc zgadali się i założyli Koło Informatyczne Bajtolandii (KIB). Zasada przyjęcia do KIBu jest prosta. Obecni członkowie zadają kandydatowi *q* pytań o treści "Czy plemię *p* jest w sojuszu z plemieniem *k*?". Jaskiniowcy mają już swoje komputery stworzone z kamieni oraz śladowych ilości drewna, lecz mimo to odpowiadanie na owe pytania sprawia im trudność. Napisz odpowiedni program i pomóż im dostać się do KIBu.

Zadanie

Napisz program, który:

- wczyta ze standardowego wejścia ilość plemion, dróg, opisy tych dróg, ilość pytań KIBu, oraz ich opisy.
- wyliczy odpowiedzi na zapytania,
- wypisze wynik na standardowe wyjście.

Wejście

W pierwszym wierszu standardowego wejścia zapisane są dwie liczby całkowite *n* i *m* ($1 \le n, m \le 1000000$) oznaczające odpowiednio ilość plemion i dróg w Bajtolandii. W następnych m wierszach znajdują się opisy tych dróg, składające się z dwóch liczb całkowitych *a* i *b* ($1 \le a, b \le n$, $a \ne b$) oznaczające, że plemię *a* i plemię *b* łączy droga. W następnym wierszu znajduje się jedna liczba całkowita *q* ($1 \le q \le 1000000$), a w następnych *q* wierszach po dwie liczby całkowite *p* i *k* ($1 \le p, k \le n$, $p \ne k$), oznaczające pytanie KIBu. W testach wartych około 40% punktów zachodzi warunek ($1 \le n, m, q \le 1000$).

Wyjście

Twój program powinien wypisać na wyjście *q* wierszy, w *i*-tym z nich powinno znajdować się pojedyncze słowo "TAK" (bez cudzysłowów), jeśli odpowiedź na *i*-te pytanie KIBu jest twierdząca, lub "NIE" w przeciwnym wypadku.

Przykład

Wejście	Wyjście
8 6 1 2 5 7 3 2 5 8 1 3 2 4 4	TAK NIE TAK NIE

1 4 6 7 7 5 3 8	

Rozwiązanie

Reprezentacja grafu – lista sąsiedztwa (jak w zadaniu Dwukolorowanie). Aby dowiedzieć, się, które plemiona są ze sobą w sojuszu, będziemy przeglądać graf od dowolnego wierzchołka. Wszyscy sąsiedzi badanego wierzchołka oraz ich sąsiedzi będą częścią jednego sojuszu. Każde wywołanie przeszukiwania grafu odnajdzie nam jeden sojusz. Zamiast kolorować wierzchołki na dwa kolory będziemy w tablicy `odwiedzony[]` przechowywać numer sojuszu, do którego należy nasze plemię.

```
BFS (w, nr_sojuszu)
  kolejka Q
  Q.dodaj(w)
  dopóki (Q nie jest pusta)
    w ← Q.zdejmij_pierwszy_wierzchołek()
    odwiedzony[w] ← nr_sojuszu
    dla wszystkich sąsiadów v wierzchołka w
      jeżeli (odwiedzony[v] = 0)
        odwiedzony[v] ← odwiedzony[w] + 1
        Q.dodaj(v)
```

W głównej części programu (po wcześniejszym wczytaniu struktury grafu) każde plemię przydzielimy do sojuszu:

```
liczba_sojuszy ← 0
dla i=1,2,…,n wykonaj
  jeżeli (odwiedzony[i]=0)
    liczba_sojuszy ← liczba_sojuszy + 1
    BFS(i, liczba_sojuszy)
```

Teraz wystarczy dla q zapytań sprawdzić, czy plemiona p oraz k zostały oznaczone tym samym numerem sojuszu:

```
wczytaj q
dla i=1,2,…,q wykonaj
wczytaj p, k
jeżeli (odwiedzony[p]=odwiedzony[k])
  wypisz TAK
przeciwnie
  wypisz NIE
```

Zajęcia 22

Temat: Grafy

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, zbiory i operacje na zbiorach, struktury danych, biblioteka STL, grafy nieskierowane, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem struktury danych – graf nieskierowany, wierzchołek, krawędzie, ścieżki,
- zna metody przechodzenia grafów, cykle

Formy i metody pracy: praca samodzielna, omówienie, dyskusja

Zadania do wykonania na zajęciach	Treści programowe
1. Mysz w labiryncie	M.5, P.2.18, A.3.7, A.4.3
2. Grand Prix Bajtocji	M.5, P.2.18, A.3.7, A.4.3

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/7/26/

Zadania do wykonania w domu:

Szpiedzy

https://szkopul.edu.pl/problemset/problem/RiTxbbjmgXZ84FuD9qo15Aq8/site/

Nadajniki

https://szkopul.edu.pl/problemset/problem/sKmyIHBMNi9EV3WO6GQ4xoFt/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Mysz w labiryncie

Dostępna pamięć: 32MB

Do labiryntu trafiła mysz. Pomóż jej odnaleźć wyjście!

Dane:

W pierwszej linii wejścia znajdują się dwie liczby całkowite n i m (2 <= n, m <= 1000) oznaczające rozmiar labiryntu: n kolumn i m wierszy. W kolejnych liniach znajdują się znaki oznaczające kolejno:

- - możliwość przejścia (korytarz);
- x - brak przejścia (ściana);
- o - początkowe położenie myszy;
- w - wyjście (końcowe położenie myszy).

Wynik

Jedna liczba całkowita oznaczająca ilość pół odwiedzonych przez mysz, lub słowo „NIE” – jeżeli taka ścieżka nie istnieje.

Przykładowe dane:

Wejście	Wyjście
8 8 w--x-x-- xx-x-x-- ---x---- --xxxx-- --x--x-- -----x-o xxxx---- ------x-	17

Rozwiązanie

Graf w tym zadaniu do kwadratowa siatka – dwuwymiarowa tablica `mapa[][]`. Z każdego pola o współrzędnych (x,y) możemy przejść do czterech pól: na prawo i na lewo oraz w górę i w dół. Ze względu na konieczność znalezienia najkrótszej ścieżki między dwoma polami rozwiązanie zadania wymaga użycia algorytmu BFS. W kolejce wierzchołków do odwiedzenia będziemy przechowywali współrzędne kolejnych pól do odwiedzenia (punkty p składające się z dwóch pól: x i y). Wszystkie niedostępne pola na mapie oznaczymy dowolną liczbą (w naszym algorytmie -1). Każde odwiedzone pole będzie zawierało informację o odległości od pola startowego, zaś pola nieodwiedzone – wartość 0. Dodatkowo (aby ułatwić sobie pilnowanie brzegów planszy) w kolumnach i wierszach 0 oraz n+1 ustawimy „ścianę” (wartownika, wartości -1).

Poniżej uproszczony algorytm:

```
kolejka Q
mapa[][]
punkt p, pom
BFS (x,y)
  dla i=0,1,2,…,n+1
    mapa[0][i] ← mapa[n+1][i] ← mapa[i][0] ← mapa[i][n+1] ← -1
  mapa[x][y] ← 1
  p.x ← 1, p.y ← 1
  Q.dodaj(p)
  dopóki (Q nie jest pusta)
    p ← Q.zdejmij_pierwszy_punkt()
```

```
jeżeli (mapa[x-1][y]=0)
  mapa[x-1][y] ← mapa[x][y]+1 //zwiększamy odległość o 1
  pom.x ← p.x-1, pom.y ← pom.y
  Q.dodaj(pom)
jeżeli (mapa[x+1][y]=0)
  mapa[x+1][y] ← mapa[x][y]+1 //zwiększamy odległość o 1
  pom.x ← p.x+1, pom.y ← pom.y
  Q.dodaj(pom)
jeżeli (mapa[x][y-1]=0)
  mapa[x][y-1] ← mapa[x][y]+1 //zwiększamy odległość o 1
  pom.x ← p.x, pom.y ← pom.y-1
  Q.dodaj(pom)
jeżeli (mapa[x][y+1]=0)
  mapa[x][y+1] ← mapa[x][y]+1 //zwiększamy odległość o 1
  pom.x ← p.x, pom.y ← pom.y+1
  Q.dodaj(pom)
```

W głównej części programu poza wczytaniem mapy musimy wywołać algorytm BFS dla punktu startowego myszy, a następnie sprawdzić wartość w punkcie wyjścia (dla 0 mysz nigdy nie dotrarła, w przeciwnym wypadku w polu `mapa[x_wyjścia][y_wyjścia]` znajdziemy minimalną odległość od wejścia.

Zadanie 2. Grand Prix Bajtocji

Dostępna pamięć: 64 MB

W Bajtocji wybudowano nowoczesny tor wyścigowy. Składa się on połączonej ze sobą sieci dróg, które mogą się łączyć ze sobą wyłącznie na skrzyżowaniach. Pomiędzy każdą parą skrzyżowań zaprojektowana została dokładnie jedna droga. Drogi (i skrzyżowania) są na tyle szerokie, że można na nich wygodnie wyprzedzać inne samochody (nawet jeżdżąc w dwóch różnych kierunkach), ale nie można na nich zawracać.

Aby urozmaicić wyścigi właściciele toru starają się umieścić pole startu / mety na różnych skrzyżowaniach. Zastanawiają się teraz, czy w ogóle jest możliwe wytyczenie takiej zamkniętej trasy.

Wejście

Skrzyżowania ponumerowane są kolejnymi liczbami naturalnymi od 1 do n ($1 \le n \le 1000$).

Pierwszy wiersz danych zawiera dwie liczby naturalne n (liczba skrzyżowań) oraz k – liczba dróg ($1 \le k \le 10^6$). Kolejne k wierszy zawiera po dwie oddzielone pojedynczym odstępem liczby naturalne u i w oznaczające numery skrzyżowań połączonych drogą ($1 \le u, w \le 1000$, $u \ne w$).

Wyjście

Program powinien wypisać słowo TAK, jeśli można wytyczyć trasę o własnościach opisanych w temacie, lub słowo NIE – w przeciwnym przypadku.

Przykład

Wejście	Wejście
4 3 1 2 1 3 2 4 Wyjście NIE	4 4 1 2 1 3 1 4 2 4 Wyjście TAK

Rozwiązanie

Reprezentacja grafu – lista sąsiedztwa. Zadanie sprowadza się jedynie do sprawdzenia, czy graf reprezentujący drogi zawiera jakikolwiek cykl. Jednym ze sposobów odnalezienia takiego cyklu jest przeszukiwanie grafu (wykorzystamy w naszym wypadku ponownie algorytm przeszukiwania grafu wszerz – BFS). Zmiana standardowego algorytmu, którą wprowadzimy, wymaga sprawdzania, czy w przypadku trafienia na wierzchołek już odwiedzony jest to inny wierzchołek niż ten, z którego przyszliśmy. W tablicy `odwiedzony[]` będziemy przechowywać dodatkowo numer wierzchołka, z którego przyszliśmy do danego wierzchołka.

Poniżej uproszczony algorytm:

```
kolejka Q
BFS (w)
  odwiedzony[w] ← 1
  Q.dodaj(w)
  dopóki (Q nie jest pusta)
    w ← Q.zdejmij_pierwszy_wierzchołek()
    dla wszystkich sąsiadów v wierzchołka w
      jeżeli (odwiedzony[v] = 0)
        odwiedzony[v] ← w //przyszliśmy z wierzchołka w
        Q.dodaj(v)
      przeciwnie
        jeżeli (odwiedzony[w] = w)
          zwróć PRAWDA i zakończ funkcję
  zwróć FAŁSZ // przeszukaliśmy graf i nie znaleźliśmy cyklu
```

Zajęcia 23

Temat: Grafy

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, struktury danych, biblioteka STL, grafy skierowane, relacje porządku liniowego/częściowego/leksykograficznego, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem struktury danych – graf skierowany, podgrafy, wierzchołek, krawędzie, ścieżki, stopień wierzchołka
- zna metody przechodzenia grafów,

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Głuchy telefon	M.3, M.5, P.2.18, A.3.7, A.3.8, A.4.3
2. Porządek alfabetyczny	M.3, M.5, P.2.18, A.3.7, A.3.8, A.4.3

Materiały do zajęć:

http://algorytmika.wikidot.com/dfs
http://algorytmika.wikidot.com/sortowanie-topologiczne
http://www.rafalnowak.pl/wiki/index.php?title=DFS
http://www.rafalnowak.pl/wiki/index.php?title=Sortowanie_topologiczne

Zadania do wykonania w domu:

Randka

https://szkopul.edu.pl/problemset/problem/gIvRmapl7sX6di87092Rmjdw/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Głuchy telefon

Dostępna pamięć: 128MB

Śliczna pani Char, przedszkolanka w przedszkolu Słoneczny Integer, lubi urządzać dla swoich podopiecznych różne zabawy. Jedną z nowych zabaw, które pani Char zaproponowała dzieciom, jest zabawa w głuchy telefon. Polega ona na tym, że pani Char szepcze do ucha jakieś trudne słowo pierwszemu dziecku, ono powtarza je drugiemu, drugie - trzeciemu, itd... Ostatnie ma wymówić je na głos. Dzieci (jak to dzieci) od razu zaznaczyły, że nie mogą powtarzać podanego słowa dowolnemu dziecku, tylko będą je mówić wyłącznie swojemu najlepszemu koledze lub najlepszej koleżance. Pani Char nie chce zmuszać do niczego maluchów. Zauważyła też od razu, że utworzyły się grupki, podgrupki i jakieś kliki, i że każdej z nich musi podać oddzielnie nowe słowo. Zastanawia się teraz, jaka jest minimalna liczba różnych słów, jaką musi wyszeptać dzieciom do ucha, aby każde z nich wzięło udział w zabawie.

Wejście

Pierwszy wiersz wejścia zawiera jedną liczbę całkowitą n $(1 \le n \le 10^5)$ - liczbę dzieci. W kolejnych n wierszach znajduje się numer dziecka x_i, do którego chce wyszeptać słowo i-te dziecko $(1 \le x_i \le n)$.

Wyjście

Wypisz minimalną liczbę dzieci, którym musi wyszeptać do ucha słowa pani Char.

Przykład

Wejście	Wyjście
4 2 3 4 3	1

Rozwiązanie

Reprezentacja grafu w tym zadaniu to jednowymiarowa tablica `do_kogo[]` – numer dziecka, do którego i-te dziecko szepcze słowo. W trakcie wczytywania zliczmy, ile dzieci chciałby szeptać słowa do i-tego dziecka (`tablica ile[]`). Następnie przeprowadźmy symulację „szeptania” zaczynając od tych dzieci, do których nikt nie chce szeptać. W tym momencie w zabawie nie brały jeszcze udziału wyłącznie te dzieci, które wzajemnie szeptają do siebie w dwie lub więcej osób (cykle). Wywołajmy symulację dla nich. Do symulacji wykorzystamy algorytm przeszukiwania grafu w głąb (DFS).

Poniżej uproszczony algorytm:

```
wczytaj n
dla i=1,2,…,n
  wczytaj do_kogo[i]
  ile[do_kogo[i]] ← ile[do_kogo[i]]+1
dla i=1,2,…,n
  jeżeli (ile[i]=0)
```

```
    wynik ← wynik+1
    DFS(i)
dla i=1,2,…,n
  jeżeli (odwiedzony[i]=0)
    wynik ← wynik+1
    DFS(i)
```

Proces symulacji szeptania:

```
DFS (w)
  odwiedzony[w] ← 1
  jeżeli (odwiedzony[do_kogo[w]]=0)
    DFS(do_kogo[w])
```

Zadanie 2. Porządek alfabetyczny

Dostępna pamięć: 64MB

Bajtomirek, kolekcjoner niespotykanych książek, odnalazł niedawno starodruk napisany w dziwnym, nieużywanym już języku. Litery w książce były zapisane co prawda znakami łacińskimi, ale kolekcjoner nie potrafił ich rozczytać. Na szczęście książka zawierała na końcu krótki indeks występujących w niej słów, ale kolejność pozycji w indeksie był inny od tego, którego można by się spodziewać. Kolekcjoner opracował już sposób na odczytanie periodyku. Musi tylko poznać kolejność liter w alfabecie z książki. Umożliwi mu w tym indeks słów. Ale wydaje mu się to bardzo żmudnym i mało ciekawym zajęciem. Dlatego, aby zakończyć pracę kolekcjonerską, Bajtomirek poprosił Ciebie, byś na podstawie indeksu słów odtworzył pierwotną kolejność liter alfabetu użytego w książce.

Wejście

Wejście składa się z uporządkowanej listy słów złożonych wyłącznie z wielkich liter alfabetu łacińskiego. W każdej linii znajduje się jedno słowo nie dłuższe niż 20 znaków. Koniec listy słów sygnalizowany jest znakiem #. Bajtomirek nie wie, czy w książce zostały użyte wszystkie litery alfabetu łacińskiego. Dla każdego zestawu słów istnieje tylko jedno poprawne rozwiązanie.

Wyjście

Wynikiem działania programu powinien być jeden wiersz zawierający wielkie litery w kolejności zgodnej z alfabetem użytym w książce.

Przykład

Wejście	Wyjście
XWY ZX ZXY ZXW YWWX #	XZYW

Rozwiązanie

W zadaniu możemy doszukać się struktury grafu skierowanego: wierzchołkami są litery, kolejność pomiędzy nimi do krawędź. Krawędź zaobserwujemy wówczas, gdy dla wyrazów s1 i s2 znajdziemy pierwszą różną literę na pozycji i. Wówczas istnieje krawędź od s1[i] do s2[i]. Informacje o krawędziach będziemy przechowywali w kwadratowej macierzy sąsiedztwa (alfabet łaciński składa się z 26 liter, więc wystarczy macierz `a[26][26]`). W algorytmie uprościmy zapis i iiterę 'A' reprezentowała będzie wartość 0 (wiersz i kolumna 0), zaś 'Z' – 25. W tablicy `jest[]` zapamiętamy, czy dana litera wystąpiła w alfabecie.

```
wczytaj()
  wczytaj s1, s2
  jeżeli (|s1|=1) jest[s1[i]] ← 1 // jednoliterowe słowo i alfabet
  dopóki (s2 ≠ '#')
    m = min(|s1|, |s2|)
    dla i=0,1,2,…,m-1 wykonuj
      jeżeli (s2 ≠ '#' ∧ s1[i] ≠ s2[i])
        a[s1[i]][s2[i]] ← 1
        jest[s1[i]] ← 1
        jest[s2[i]] ← 1
        i ← m //kończy wewnętrzną pętlę
    s1 ← s2
    wczytaj s2
```

Aby ustalić kolejność liter w alfabecie skorzystamy z sortowania topologicznego opartego o algorytm przeszukiwania grafu w głąb DFS. Dla każdej nie użytej jeszcze litery znajdującej się naszą badaną literą c będziemy wywoływać DFS. Jeżeli dla jakiejś litery zabraknie liter znajdujących się za nią, dodamy ją na początek naszego rozwiązania (wyraz `wynik`).

```
DFS(litera c)
  odwiedzony[c] ← 1
  //szukamy nie odwiedzonych liter alfabetu za c
  dla i=0,1,2,…,25 wykonuj
    jeżeli (odwiedzony[i]=0 ∧ a[c][i]=1)
      DFS(i)
  wynik ← c + wynik

odwiedz()
  dla i=0,1,2,…,25 wykonuj
    jeżeli (odwiedzony[i]=1 ∧ jest[i]=1)
      DFS(i)
  wypisz wynik
```

Główna część programu to wywołanie funkcji `wczytaj()` i `odwiedz()`.

Zajęcia 24

Temat: Grafy

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, struktury danych, biblioteka STL, grafy skierowane, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem struktury danych – graf skierowany, wierzchołek, krawędzie, ścieżki, stopień wierzchołka
- zna metody przechodzenia grafów,

Formy i metody pracy: praca samodzielna, omówienie, dyskusja, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Mapa	M.5, P.2.18, A.4.7,
2. Średniowiecze	M.5, P.2.18, A.4.7,

Materiały do zajęć:

http://algorytmika.wikidot.com/dfs
http://algorytmika.wikidot.com/dijkstra

Zadania do wykonania w domu:

Przemytnicy

https://szkopul.edu.pl/problemset/problem/l8-ujU0a7HQFxy8UY32B4Kk_/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Mapa

Limit pamięci: 128MB

Jaś jak co wieczór spędza czas ze swoim tatą. Postanowił poprosić go o trochę pieniędzy, aby mógł pójść do kina wraz z przyjaciółmi. Tata Jasia, który dba o jego rozwój, wymyślił mu następującą grę.

Tata pokazał mu mapę lasu, który rośnie niedaleko ich domu. Mapa zawiera spis *n* polan, które znajdują się w lesie oraz *m* ścieżek łączących niektóre z nich. Tata Jasia bardzo lubi chodzić do lasu, żeby podbiegać, ponieważ jest to bardzo zdrowe i buduje kondycje, przed dorocznym biegiem po ulicach Wałbrzycha (przecież nie Bajtocji). Biega on tylko po ścieżkach, z polany na polanę. Ma on jednak swoje upodobania i każdą ścieżką biega tylko w jedną stronę, tzn. każda ścieżka jest dla niego jednokierunkowa.

Tata nie pokazał synkowi tej mapy bez powodu. Zadał on mu pytanie, o to, czy startując z pewnej polany i biegnąc pewnymi ścieżkami (w odpowiednim kierunku) da się wrócić na polanę z której się zaczęło. Innymi słowy tata pyta Jasia czy w tym lasku ma możliwość biegać w kółko. Pytanie jest jednak bardzo trudne i przerasta Jasia. Pomóż mu i napisz odpowiedni program.

Zadanie

Napisz program, który:

wczyta ze standardowego wejścia liczbę polan, ścieżek oraz ich opis, wyliczy odpowiedzi na zapytanie, wypisze wynik na standardowe wyjście.

Wejście

W pierwszym wierszu standardowego wejścia zapisane są dwie liczby całkowite nieujemne *n* (*1 ≤ n ≤ 100000*) i *m* (*0 ≤ m ≤ 500000*), oznaczające odpowiednio ilość polan oraz ścieżek. Następne *m* wierszy zawiera po dwie liczby całkowite a_i i b_i (*1 ≤* a_i, b_i *≤ n*), oznaczające, że w lasku istnieje ścieżka, którą można przebiec od polany a_i do polany b_i. Może się zdarzyć, że $a_i = a_j$ i $b_i = b_j$ dla $i \neq j$. W testach wartych około 33% punktów zachodzą dodatkowe warunki *(1 ≤ n, m ≤ 1000)*.

Wyjście

Jeśli w owym lesie da się biegać w kółko na wyjście należy wypisać pojedyncze słowo "TAK". W przeciwnym wypadku należy wypisać "NIE".

Przykład

Wejście	Wyjście
4 5 1 2 2 4 3 1 3 4 2 3	TAK

Rozwiązanie

W tym zadaniu graf skierowany reprezentował będzie polany (wierzchołki) i ścieżki w lesie (lista sąsiedztwa). W zadaniu musimy zbadać występowanie cykli. W grafie skierowanym użyjemy do tego celu algorytmu przeszukiwania w głąb DFS. Wierzchołki w grafie pokolorujemy na trzy kolory: nieodwiedzony: biały (0) oraz odwiedzone: czarny (1) i szary (2). Każdy wierzchołek, który wywołamy rekurencyjnie, oznaczymy kolorem czarnym. Po przeszukaniu wszystkich jego sąsiadów (koniec wywołań DFS dla danego wierzchołka) oznaczymy go kolorem szarym.

Jak zorientować się, że znaleźliśmy cykl? Zauważmy, że trafiając na wierzchołek w już odwiedzony w kolorze czarnym oznacza to, że jest on jakimś potomkiem badanego wierzchołka w (nie zakończyło się jeszcze wywołanie rekurencyjne DFS dla w) – mamy więc cykl. W przypadku koloru szarego wierzchołek v musiał być odwiedzony z innej ścieżki i nie możemy potwierdzić wystąpienia cyklu.

Poniżej uproszczony algorytm DFS:

```
DFS (w)
  odwiedzony[w] ← 1
  dla wszystkich sąsiadów v wierzchołka w
    jeżeli ( odwiedzony[v]=1 ∨ (odwiedzony[v]=0 ∧ DFS(v)=PRAWDA) )
      zwróć PRAWDA i zakończ funkcję
  odwiedzony[w] ← 2
  zwróć fałsz
```

Główna część programu (wczytywanie struktury grafu pominięte):

```
dla i=1,2,3,…,n //n - liczba wierzchołków
  jeżeli (odwiedzony[i]=0 ∧ DFS(i)=PRAWDA)
    wypisz TAK i zakończ program
wypisz NIE i zakończ program
```

Zadanie 2. Średniowiecze

Limit pamięci: 128MB

Ponure czasy nastały w Bajtolandii. Dzikie zwierzęta, zbóje i magiczne stwory szaleją i zagrażają bezpieczeństwu Bajtolandczyków, ale jak zwykle to nie to nas interesuje. Interesuje nas sam KIB, który nadal ma się dobrze. Ma też nowe wyzwania dla młodych algorytmików chcących wejść w ich szeregi.

W Bajtolandii istnieje wiele smoczych i tajemniczych jam, między nimi jedna, która znajduje się tuż obok siedziby KIBu. Jako, że mistrzowie stowarzyszenia byli w niej wiele razy to sporządzili jej dokładną mapę, która wisi przy wejściu do siedziby. Smocza jama wygląda następująco.

- Jest w niej *n* jaskiń, numerowanych od *1* do *n*.
- Między niektórymi z jaskiń są jednokierunkowe tunele (smok jest bardzo szybki i wszystkowidzący, więc pilnuje wszystkich tuneli, aby nikt nie chodził nimi w niewłaściwą stronę)(tunel może prowadzić od pewnej jamy z powrotem do niej).

- Każdy tunel ma swój koszt, czyli ilość złota, którą należy zapłacić smokowi, aby przejść danym tunelem (smok pilnuje również, aby każdy zawsze płacił).
- Wejście do jamy to jaskinia o numerze *1*, a smoczy skarbiec to jaskinia o numerze *n*.

Aby od wejścia dostać się do skarbca należy oczywiście iść różnymi tunelami i wiele zapłacić. Pytanie kwalifikacyjne do KIBu jest następujące. *„Ile należy minimalnie w sumie zapłacić, aby od wejścia dostać się do smoczego skarbca?”*To zadanie jest jednak bardzo trudne. Pomóż młodym algorytmikom i napisz odpowiedni program.

Zadanie

Napisz program, który:

- wczyta ze standardowego wejścia ilość jaskiń, tuneli oraz opisy tych tuneli.
- wyliczy odpowiedzi na pytanie KIBu,
- wypisze wynik na standardowe wyjście.

Wejście

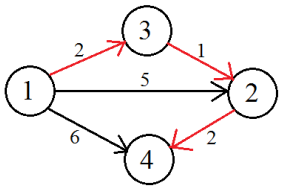
W pierwszym wierszu standardowego wejścia zapisane są dwie liczby całkowite *n* i *m* *(1≤n≤200000, 1≤m≤1000000)* oznaczające odpowiednio ilość jaskiń i dróg w smoczej jamie. W następnych *m* wierszach znajdują się opisy tych tuneli, składające się z trzech liczb całkowitych *a*,*b* i *c(1≤a,b≤n, 0≤c≤10^9)* oznaczające, że istnieje tunel zaczynający się w jaskini *a*, kończący się jaskini *b*, którego przejście kosztuje *c* sztuk złota.

Wyjście

Twój program powinien wypisać jedną liczbę całkowitą, będącą odpowiedzią na pytanie KIBu. Jeśli nie da się dotrzeć od wejścia do smoczego skarbca, program powinien wypisać - 1.

Przykład

Wejście	Wyjście
4 5 1 2 5 2 4 2 1 3 2 3 2 1 1 4 6	5



Rozwiązanie

W tym zadaniu skierowany graf ważony o dodatnich wagach reprezentował będzie jaskinie (wierzchołki) i drogi pomiędzy jaskiniami (lista sąsiedztwa; dla każdego wierzchołka w

będziemy pamiętać listę jego sąsiadów v wraz z odległością od w do v). Poniżej kod w C++ z komentarzami:

```
vector <pair <long long, int> > t[200007]; //lista sąsiedztwa
//first – waga, second – nr wierzchołka końcowego
cin >> n >> k; //liczba wierzchołków i krawędzi
for (int i=0; i<k; i++){
     cin >> a >> b >> w; // krawędź z a do b o wardze w
     t[a].push_back(make_pair(w,b));
}
dijkstra(1);
```

Algorytm Dijkstry:

```
long long INF=4000000000000000000LL; // nieskończoność
priority_queue <pair <long long, int> > q; // kolejka priorytetowa,
// umożliwia znalezienie najbliższego wierzchołka do 1
pair <long long, int> x; //pomocnicza
long long d[200007]; // odległości od wierzchołka 1 do pozostałych
int odw[200007];

void dijkstra(int v){
  // ustaw odległości z v do wszystkich miast na nieskończoność
  for (int i=1; i<=n; i++) d[i]=INF;
  // odległość do v ustawiam na 0 i dodaję do kolejki
  d[v]=0;
  q.push(make_pair(0,v));
  //dopóki kolejka nie jest pusta
  while (!q.empty()){
    //zdejmij najbliższego sąsiada i jeśli nie jest odwiedzony
    x=q.top(); q.pop();
    if (!odw[x.second]){
      //oznacz jako odwiedzony
      odw[x.second]=1;
      //dla wszystkich sąsiadów x.second
      for (int i=0; i<t[x.second].size(); i++){
        int u=t[x.second][i].second; //sąsiad
        long long k=t[x.second][i].first;
        //sprawdź, czy do v będzie bliżej przez x.second
        //do sąsiadów x.second niż bezpośrednio
        if (d[u]>k+d[x.second]){
          //zapamiętaj nową lepszą odległość
          d[u]=k+d[x.second];
          //dodaj do kolejki; odwracając wartość odległości
          //najbliższe znajdą się na początku kolejki
          q.push(make_pair(-d[u],u));
        }
      }
    }
  }
  //jeżeli zaktualizowano odległość z 1 do n,
  //w d[n] mamy szkuną wartość
  if (d[n]==INF) cout << "-1\n";
```

```
  else cout << d[n] << "\n";
}
```

Zajęcia 25

Temat: Drzewa

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, struktury danych, biblioteka STL, drzewa, drzewa ukorzenione, drzewo binarne, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem drzew ukorzenionych, binarnych, przedziałowych,

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Samochody	M.5, P.2.5, P.2.6, A.3.6, A.4.7,
2. Nasionka	M.5, P.2.5, P.2.6, A.3.6, A.4.7,

Materiały do zajęć:

https://www.main2.edu.pl/main2/courses/show/7/23/
https://www.youtube.com/watch?v=gfVDgd_LQn4&t=109s

Zadania do wykonania w domu:

Koleje

https://szkopul.edu.pl/problemset/problem/VYTSyRwgdwmLf56i_ffWGB0L/site/

Tetris 2D

https://szkopul.edu.pl/problemset/problem/-0A1PC3fNc2RK-qvoUQKGGLl/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Samochody

Dostępna pamięć: 32MB

W najbliższym czasie zaplanowane zostały zawody samochodowe. Wiadomo, że odbędzie *k* wyścigów. Chęć udziału w wyścigach zgłosiło *n* samochodów, którym nadano numery od 1 do *n*. Wszystkie samochody mają swoją własną maksymalną prędkość początkową. Jednak przed każdym wyścigiem po- zostawiono zawodnikom czas, w którym samochody mogą zostać ulepszone, a ich prędkość maksymalna może wówczas wzrosnąć.

Według zasad zawodów, w *i*-tym wyścigu wezmą udział pojazdy o numerach od a_i do b_i. Oczywiście ma wygrać najlepszy, co w tym przypadku najprawdopodobniej oznacza: osiągający największą prędkość.

Znasz początkowe prędkości maksymalne wszystkich *n* samochodów i plan ich ulepszeń. Wiesz zatem, ile samochodów zostanie ulepszonych przed *i*-tym wyścigiem, które to będą pojazdy i o ile wzrośnie prędkość maksymalna każdego z nich. Posiadasz też informację o tym, które samochody wezmą udział w każdym z wyścigów.

Tak rozległa wiedza pozwala wierzyć, że jesteś w stanie przewidzieć wyniki. Napisz program, który pomoże Ci to zrobić.

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite: *n i k* ($1 \le n \le 300\,000$, $1 \le k \le 300\,000$), oznaczające odpowiednio: liczbę wszystkich samochodów oraz liczbę mających się odbyć wyścigów.

W kolejnym wierszu znajduje się *n* liczb, gdzie *i*-ta liczba to v_i *(*$1 \le v_i \le 1\,000\,000$*)*, czyli prędkość samochodu o *i*-tym numerze.

W kolejnych wierszach znajdują się opisy kolejnych *k* wyścigów. Każdy taki opis składa się z wartości x_i ($0 \le x_1+x_2+...+x_n \le 300\,000$), czyli liczby ulepszeń przed *i*-tym wyścigiem, x_i wierszy zawierających dwie liczby: s_i, d_i *(*$1 \le s_i \le n$, $0 \le di \le 1\,000\,000$*)*, z których pierwsza oznacza numer ulepszanego samochodu, a druga wartość, o jaką zwiększy się jego prędkość maksymalna, w ostatnim wierszu opisu znajduje się dwie liczby a_i oraz b_i *(*$1 \le a_i \le b_i \le n$*)*, oznaczające, że w danym wyścigu wezmą udział samochody o numerach od a_i do b_i.

Wyjście

Twój program powinien wypisać na standardowe wyjście *k* wierszy. W *i*-tym wierszu ma znajdować się numer samochodu, który powinien wygrać *i*-ty wyścig. Możesz założyć, że odpowiedź zawsze będzie jednoznaczna.

Wejście	Wyjście
5 3 7 6 4 12 1 2 2 7 1 3 1 4 0	2 4 3

3 5 3 5 14 2 2 3 14 1 5	

Rozwiązanie

W tym zadaniu skorzystamy z drzew przedziałowych. Wystarczające będzie skorzystanie z drzewa maksimów (punkt-przedział). Liście w tym drzewie będą przechowywały informacje o prędkości poszczególnych samochodów, w węzłach będziemy przechowywać informację o samochodzie, który osiąga w zadanym przedziale (wśród wszystkich potomków węzła) największą prędkość.

Aktualizacja prędkości samochodu wymaga aktualizacji prędkości maksymalnych wszystkich węzłów-przodków samochodów.

Sprawdzenie zwycięzcy poszczególnych wyścigów wymaga wyszukania maksymalnej prędkości w zadanym przedziale.

Zadanie 2. Nasionka

Dostępna pamięć: 32MB

Ola znalazła pracę w pięknym ogrodzie. Do jej zadań należy troszczenie się o zaopatrzenie ogrodników.

Aktualnie ogrodnicy skończyli planować, jakie rabaty kwiatów urządzą w najbliższej przyszłości. Są bardzo dokładni, więc zrobili to naprawdę szczegółowo. Ola dostała listę, z której dowiedziała się, że chcą oni posiać n rodzajów kwiatów i będzie im potrzebne z_i nasionek i-tego rodzaju.

Teraz Ola, która naprawdę przejmuje się swoją pracą, obawia się, czy fundusze, które ma do wydania na nasionka, są wystarczające, żeby zrealizować całe zamówienie.

Na szczęście okazało się, że jeden ze sklepów ogrodniczych obchodzi dziesięciolecie swojego istnienia i z tej okazji planuje szereg promocji. W najbliższym czasie ma się odbyć k akcji promocyjnych. W ramach j-tej akcji będzie można kupić po x_j nasionek z rodzajów, których numery mieszczą się w przedziale od a_j do b_j.

Ola postanowiła skorzystać z okazji i pokazać, że jest odpowiednią osobą do tej pracy. Chce dołożyć wszelkich starań, by zrealizować całe zamówienie, w miarę możliwości nie przekraczając zaplanowanej na to sumy pieniędzy. W tym celu ma zamiar kupić jak najwięcej potrzebnych nasionek w promocyjnej cenie.

Pomóż Oli, która i tak ma bardzo dużo zajęć. Napisz program, który po każdej promocji będzie ją informował, ile rodzajów nasionek udało jej się właśnie skompletować oraz które to rodzaje, tak by mogła skreślić je z listy zakupów. Dzięki Twojej pomocy na pewno będzie jej łatwiej zorientować się, jaką część zamówienia ogrodników już zrealizowała. Ola będzie Ci bardzo wdzięczna.

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite: n i k ($1 \le n \le 500\,000$, $1 \le k \le 100\,000$), oznaczające odpowiednio: liczbę rodzajów nasionek oraz liczbę zaplanowanych akcji promocyjnych.

W kolejnym wierszu znajduje się n liczb całkowitych, gdzie i-ta liczba oznacza, że ogrodnicy zamówili z_i ($1 \le v_i \le 1\,000\,000\,000$) nasionek i-tego rodzaju.

Następne k wierszy to opisy kolejnych akcji promocyjnych. Każdy taki opis składa się z trzech liczb całkowitych a_j, b_j, x_j ($1 \le a_j \le b_j \le n$, $0 \le x_j \le 1\,000\,000\,000$), oznaczających, że w ramach i-tej promocji Ola będzie mogła kupić za połowę ceny po x_j nasionek z każdego rodzaju o numerze od a_j do b_j.

Wyjście

Twój program powinien wypisać na standardowe wyjście k wierszy.

W i-tym wierszu ma na początku znajdować się liczba m_i oznaczająca, ile rodzajów nasionek udało się skompletować dzięki i-tej akcji promocyjnej, a następnie m_i liczb (w kolejności rosnącej) informujących, jakiego rodzaju były to nasionka.

Wejście	Wyjście
8 4 10 8 14 21 6 73 1 16 2 5 7 1 6 1 3 7 20 4 8 4	1 5 1 2 3 3 4 7 0

Rozwiązanie

W tym zadaniu skorzystamy z drzew przedziałowych. Konieczne będzie skorzystanie z drzewa typu przedział-przedział. Musimy przechowywać informacje o liczbie każdego rodzaju nasionek Szukamy w każdym kroku i zliczamy te, których liczba po promocji osiągnie wartość mniejszą równą 0. W tym celu dodajemy (tak naprawdę odejmujemy) na przedziale liczbę promocyjnych nasionek. Następnie musimy wskazać, ile było takich nasionek oraz zapamiętać ich numery.

Zajęcia 26

Temat: Algorytmy geometryczne

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, struktury danych, biblioteka STL, algorytmy geometryczne: punkt, odcinek, prosta, wektor, wielokąt, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem algorytmów geometrycznych,
- zna iloczyn skalarny, wektorowy,
- zna wzór Picka.

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Proste prostopadłe	M.6, P.2.18, P.2.20, A.3.13,
2. Fioletowe lasery	M.6, P.2.18, P.2.20, A.3.13,
3. Wiśniowy sad	M.6, P.2.18,P.2.20, A.3.13,

Materiały do zajęć:

http://informatykaplus.edu.pl/upload/list/czytelnia/Przeglad_podstawowych_algorytmow.pdf
ss. 29-32

Zadania do wykonania w domu:

Radar (gotowa paczka do SIO2)

ZADANIA I ROZWIĄZANIA

Zadanie 1. Proste prostopadłe

Limit pamięci: 32MB

Dane są dwie proste, każda opisana za pomocą dwóch punktów. Mając dane współrzędne kartezjańskie punktów A, B (pierwsza prosta) oraz C i D (druga prosta) określ, czy te dwie proste są prostopadłe.

Wejście

Pierwszy wiersz danych zawiera liczbę naturalną W z zakresu od 1 do 1000, określającą ilość zestawów danych, czyli opisów par wektorów do wczytania. Każdy zestaw danych obejmuje dwa wiersze: w pierwszym znajdują się współrzędne pierwszej prostej (punkty A i B), a w drugim – współrzędne drugiej prostej (punkty C i D). Współrzędne punktów są liczbami całkowitymi, których wartość bezwzględna nie przekracza stu milionów. Liczby w wierszu oddzielone są od siebie pojedynczymi spacjami.

Wyjście

Program powinien wypisać dla każdego przypadku wiersz tekstu zawierający słowo TAK, jeśli dane wektory są prostopadłe, lub słowo NIE, jeśli nie są prostopadłe.

Przykład

Wejście	Wynik
2 0 0 1 1 2 0 1 1 0 0 2 0 0 0 2 1	TAK NIE

Rozwiązanie

Potraktujmy odcinki jako wektory i obliczmy ich współrzędne. W takim wypadku wystarczy sprawdzić, czy iloczyn skalarny tych wektorów ($x_1 \cdot x_2 + y_1 \cdot y_2$) jest równy 0.

Zadanie 2. Fioletowe lasery

Dostępna pamięć: 256MB. Limit czasu: 1s

Bajtomir zajmuje się w Bajtocji wywiadem. Dostał właśnie nowe zadanie. Na podstawie meldunków szpiegów może wskazać położenie czujników ruchu w pewnym niedostępnym budynku. Dowiedział się też, że czujniki można zneutralizować specjalnym fioletowym laserem, ale trzeba to zrobić jednym strzałem. Bajtomir zastanawia się teraz, czy jest to możliwe. Pomóż mu!

Wejście

W pierwszej linii wejścia znajduje się jedna liczba całkowita n ($1 \le n \le 10^6$), oznaczająca liczbę czujników ruchu w budynku. W kolejnych n liniach znajdują się po dwie liczby x_i oraz y_i ($-10^6 \le x_i, y_i \le 10^6$) współrzędne i-tego czujnika ruchu.

Wyjście

Na wyjściu powinno znaleźć się jedno słowo: TAK lub NIE - odpowiedź na pytanie z zadania.

Przykład

Wejście	Wyjście
3 0 0 -5 -5 5 5	TAK

Wejście	Wyjście
3 0 0 1 2 2 1	NIE

Rozwiązanie

Zauważmy, że sprawdzamy, czy wszystkie punkty są współliniowe. Zapamiętujemy pierwszy punkt, szukamy kolejnego punktu o współrzędnych różnych niż punkt pierwszy. Następnie korzystając z iloczynu wektorowego sprawdzamy współliniowość punktów.

```
punkt a, b, c
wczytaj n
wczytaj a.x, a.y
i←2
wykonuj
  wczytaj b.x, b.y
  i ← i+1
dopóki (i≤n ∧ a.x=b.x ∧ a.y=b.y)
dopóki (i≤n)
  //jeżeli iloczyn wektorowy dla 3 punktów nie jest równy 0
  //punkty nie są współliniowe
  wczytaj c.x, c.y
  jeżeli ( (c.x-a.x) · (b.y-a.y)-(c.y-a.y) · (b.x-a.x) ≠ 0 )
    wypisz NIE i zakończ program
wypisz TAK
```

Zadanie 3. Wiśniowy sad

Dostępna pamięć: 128 MB

Bajtomirek zakupił piękny wiśniowy sad. Sad jest piękny, ponieważ wszystkie drzewa posadzono obok siebie w odległości 1 zarówno w pionie, jak i w poziomie. Sad jest ogrodzony zwykłą siatką, a słupki, do których przymocowana jest siatka, są umieszczone w różnych miejscach, zawsze jednak o współrzędnych całkowitych względem położenia drzew. Żadne drzewo nie znajduje się na linii ogrodzenia. Bajtomirek zastanawia się teraz, ile dokładnie jest drzew wiśniowych w jego sadzie. Chciałby to szybko obliczyć, nie wie jednak jak. Pomóż mu!

Wejście

Pierwszy wiersz zawiera jedna liczbę całkowitą n – liczbę słupków ogrodzeniowych. Kolejnych n linii zawiera po dwie liczby całkowite x oraz y – współrzędne kolejnych słupków ($3 \le n \le 10^3$, $0 \le x, y \le 10^6$).

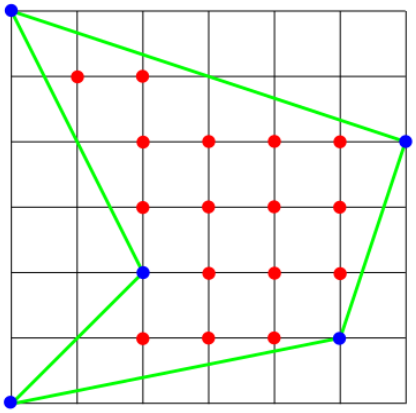
Wyjście

Program powinien wypisać liczbę drzew wiśniowych znajdujących się wewnątrz ogrodzenia.

Przykład

Wejście:	Wyjście
5 0 0 5 1 6 4 0 6 2 2	16

Ilustracja przykładu:



Rozwiązanie

W zadaniu rozwiązanie uzyskamy korzystając ze wzoru Picka:

$$P = W + \frac{1}{2}B - 1$$

Gdzie: P – pole powierzchni wielokąta, W – liczba punktów kratowych leżących wewnątrz wielokąta, B – liczba punktów kratowych leżących na brzegu wielokąta.

Musimy zatem wyznaczyć pole powierzchni i liczbę punktów kratowych leżących na brzegu wielokąta.

Iloczyn wektorowy wyznacza pole powierzchni równoległoboku. Pole powierzchni całkowitej wielokąta obliczymy sumując wszystkie pola trójkątów pomiędzy każda parą sąsiadujących ze sobą wierzchołków wielokąta i punktu (0,0) (ostateczny wynik podzielimy na 2, możemy otrzymać ujemny).

Liczbę punktów kratowych leżących na brzegu wielokąta wyznaczymy z NWD współrzędnych x i y odcinków pomiędzy każdą parą sąsiadujących ze sobą wierzchołków wielokąta.

```
funkcja det(x1, y1, x2, x2)
  zwróć x1·y2-x2·y1

wczytaj n
dla i=0,1,2,…,n-1 wykonuj
    wczytaj x[i]. y[i]
x[n] ← x[0], y[n] ← y[0]
P ← 0, B ← 0
dla i=0,1,2,…,n-1 wykonuj
    P ← P det(x[i],y[i],x[i+1],y[x+1])
    B ← B + NWD( |x[i]-x[x+1]|, |y[i]-y[i+1]| )
P ← |P|
wypisz (P + 2 – B)/2
```

Zajęcia 27

Temat: Algorytmy geometryczne

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, zbiory i operacje na zbiorach, struktury danych, biblioteka STL, algorytmy geometryczne: punkt, odcinek, prosta, wektor, wielokąt, iloczyn skalarny i wektorowy, otoczka wypukła, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem algorytmów geometrycznych,
- zna iloczyn skalarny, wektorowy, otoczkę wypukłą

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Bombardowanie	M.6, P.2.18, P.2.20, A.3.13,
2. Mur	M.6, P.2.18, P.2.20, A.3.13,

Materiały do zajęć:

http://informatykaplus.edu.pl/upload/list/czytelnia/Przeglad_podstawowych_algorytmow.pdf
ss. 29-32

Zadania do wykonania w domu:

Ołtarze (IV OI)

https://szkopul.edu.pl/problemset/problem/JR1lB-gdjNHcT5j3mtNRFhLa/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Bombardowanie

Limit pamięci: 128MB

Stało się! Bitocja wypowiedziała wojnę Bajtocji! Wychodząc z założenia, że najlepszą formą obrony jest atak, już pierwszego dnia wojny Bajtocjanie wysłali swoje bombowce z bombami bitowymi nad sąsiednią wyspę. Bomby bitowe zmieniają wszystkie bity w urządzeniach elektronicznych o wartości 1 na losowe siejąc popłoch w dowództwie przeciwnika. Niestety, z powodu gęstej mgły piloci zrzucili swoje ładunki w różnych miejscach wyspy i teraz nie wiadomo, ile z nich na pewno trafiło w terytorium Bitocji. Ponieważ Bitocja ma kształt wielokąta wypukłego o znanych wierzchołkach, a Ty znasz miejsca zrzucenia ładunków, możesz określić tę liczbę!

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby całkowite n i m ($3 \le n, m \le 10^5$), odpowiednio liczba punktów określających kształt Bitocji oraz liczba bomb bitowych zrzuconych przez Bajtocję. W kolejnych n liniach znajdują się po dwie liczby całkowite x_i i y_i ($-10^9 \le x_i, y_i \le 10^9$) – współrzędne wierzchołków Bitocji. W następnych m liniach znajdują się po dwie liczby całkowite x_j i y_j ($-10^9 \le x_j, y_j \le 10^9$) – współrzędne zrzutów bomb bitowych.

Wyjście

Jedna liczba całkowita oznaczająca, ile bomb na pewno trafiło w terytorium Bitocji.

Przykład

Wejście	Wyjście
4 3 0 0 5 0 5 5 0 5 1 1 6 1 1 4	2

Rozwiązanie

Naszym zadaniem jest sprawdzenie, ile punktów leży wewnątrz wielokąta. Zauważmy, że wielokąt ten stanowi otoczkę wypukłą.

Rozwiązanie wolne: Dla każdego z punktów będziemy sprawdzać, czy leży on wewnątrz któregokolwiek z trójkątów zbudowanych z punktu pierwszego i każdej pary kolejnych dwóch wierzchołków wielokąta.

Rozwiązanie szybkie: Spróbujmy podzielić wielokąt na dwie podobnej wielkości części i sprawdźmy, w której z nich może być położony każdy sprawdzany punkt (na lewo bądź na prawo od prostej dzielącej nasz wielokąt). Podobnie postępujmy z nowym fragmentem tak długo, dopóki nie pozostanie nam jeden trójkąt (wyszukiwanie binarne po wyniku. Na koniec sprawdźmy, czy punkt ten leży wewnątrz trójkąta (suma pól trzech powstałych małych trójkątów, gdzie jednym z wierzchołków jest badany punkt, musi być równa polu „dużego" trójkąta).

```
det(punkt a, punkt b, punkt c) // iloczyn wektorowy
    zwróć (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x)

wewnątrz_tr(punkt a, punkt b, punkt c, punkt x)
    jeżeli (|det(a,b,c)|=|det(a,b,x)|+|det(a,c,x)|+|(det(b,c,x)|
        zwróć PRAWDA
    zwróć FAŁSZ

punkt w[n] //wierzchołki wielokąta
wewnątrz (punkt x){
    L←1, P=n←1 //liczymy punkty od 0, ostatni jest n-1
    jeżeli
    jeżeli ( det(w[0],w[P],x)>0 ∨ det(w[0],w[L],x)<0) zwróć FAŁSZ
    // wyszukujemy binarnie
    dopóki (L<P-1)
        środek ← (L+P) div 2
        jeżeli (det(w[0],w[P],x)<=0 ∧ det(w[0],w[srodek],x)>=0)
            L←środek;
        przeciwnie P←środek;
    jeżeli (wewnątrz_tr(w[0],w[P],w[L],x)=PRAWDA) zwróć PRAWDA
    zwróć FAŁSZ
```

W głównej części programy wywołujemy funkcję `wewnątrz` dla każdej bomby i zliczamy.

Zadanie 2. Mur

Limit pamięci: 64MB

Kiedy w Bitomiu odkryto złoto, mieszkańcy miasta wpadli w popłoch. Do ich miasta zaczęły ciągnąć dziesiątki, później setki, a w końcu tysiące poszukiwaczy łatwego zarobku. Aby nie dopuścić do katastrofy, postanowiono ogrodzić Bitom murem. Jednocześnie postanowiono, że mur będzie możliwie najmniejszy.

Ktoś szybko naniósł współrzędne domów na mapę, ktoś drugi zaczął już wyrabiać cegły, inni zaczęli składać zamówienia na cement. Ale jaki ma być przebieg muru?

Zadanie

Napisz program, który:

- wczyta ze standardowego wejścia ilość domów i ich współrzędne,
- wyznaczy domy, obok których będzie bezpośrednio przebiegał mur,
- wypisze wynik na standardowe wyjście.

Wejście

W pierwszym wierszu standardowego wejścia zapisana jest jedna liczba całkowita n ($2 \le n \le 500\,000$) oznaczająca ilość domów. Kolejne n wierszy zawiera po dwie liczby całkowite a i b ($-10^9 \le a, b \le 10^9$) oznaczające współrzędne kolejnych domów. W części testów wartych około 40% punktów zachodzi warunek ($2 \le n \le 2000$) oraz ($-10000 \le a, b \le 10000$).

Wyjście

W pierwszym wierszu standardowego wyjścia Twój program powinien wypisać jedną liczbę całkowitą X, oznaczającą liczbę domów obok których stanie mur. W kolejnych X liniach znajdą się kolejne współrzędne domów. Jako pierwszy wypisz dom znajdujący się jak najbardziej na zachód. Jeśli jest takich kilka, wypisz ten, który znajduje się najbardziej na południu. Kolejne domy wypisuj w kolejności przeciwnej do ruchu wskazówek zegara.

Przykład

Wejście	Wyjście
10 1 4 3 3 4 2 3 5 1 2 5 4 1 0 6 6 5 3 6 4	8 1 0 4 2 5 3 6 4 6 6 3 5 1 4 1 2

Rozwiązanie

W zadaniu musimy wyznaczyć otoczkę wypukłą. Posortujmy wszystkie punkty od lewej do prawej (w przypadku takiej samej współrzędnej x posortujmy po y) w tablicy `punkty[]`. Dodajemy do otoczki dwa pierwsze punkty, a następnie dla wszystkich pozostałych punktów sprawdzamy, czy po ich dodaniu otoczka „skręca” w lewo (iloczyn wektorowy dla trzech ostatnich punktów musi być większa od 0). Jeżeli nie, usuwamy poprzednie punkty które w tym przeszkadzają. Działanie powtarzamy od prawej do lewej.

```
det(punkt a, punkt b, punkt c) // iloczyn wektorowy
    zwróć (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x)

punkt otoczka[], punkty[]
buduj_otoczkę()
    otoczka[0] ← punkty[0]
    otoczka[1] ← punkty[1]
    ostatni ← 1 //ostatni dodany punkt do otoczki
    dla i=2,3,…,n-1 wykonuj
        dopóki (ostatni>0
        ∧ det(otoczka[ostatni-1],otoczka[ostatni],punkty[i])<0)
            ostatni ← ostatni - 1
        ostatni ← ostatni + 1
        otoczka[ostatni] ← punkty[i]
    dla i=n-2,n-3,…,0 wykonuj
        dopóki (ostatni>0
        ∧ det(otoczka[ostatni-1],otoczka[ostatni],punkty[i])<0)
            ostatni ← ostatni - 1
        ostatni ← ostatni + 1
        otoczka[ostatni] ← punkty[i]
```

Liczbę domów pamięta zmienna ostatni, wypisujemy zawartość tablicy `otoczka[]`.

Zajęcia 28

Temat: Algorytmy tekstowe

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, struktury danych, biblioteka STL, algorytmy tekstowe KMP, haszowanie, łańcuchy znaków, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem algorytmów tekstowych,
- potrafi wyszukiwać wzorzec w tekście,
- zna algorytm naiwny i KMP, haszowanie.

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Czołgiści	M.3, P.2.15, A.3.3,
2. Ści(ą)gany	M.3, P.2.15, A.3.3,

Materiały do zajęć:

http://informatykaplus.edu.pl/upload/list/czytelnia/Zaawansowane_algorytmy_ALL.pdf ss. 13-15

Zadania do wykonania w domu:

Pociągi

https://szkopul.edu.pl/problemset/problem/_aNlLfqZBTY6FRf2_IGScTLn/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Czołgiści

Limit pamięci: 128MB

W Skaryszewie odbywa się doroczny ogólnoświatowy zjazd czołgistów. Przybyło na niego swoimi czołgami n czołgistów. Każdy czołg ma nazwę (typ czołgu, nazwa nadana przez czołgistę itp.). Znając wszystkie nazwy czołgów musisz stwierdzić, ile jest par czołgów o tej samej nazwie.

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba całkowita n ($1 \leq n \leq 1000$) oznaczająca liczbę czołgistów (i tym samym liczbę czołgów). W kolejnych n wierszach znajdują się nazwy czołgów. Nazwa składa się wyłącznie z małych liter alfabetu angielskiego i ma nie więcej niż 2000 znaków.

Wyjście

Na wyjście należy wypisać jedną liczbę całkowitą, oznaczającą ilość par czołgów o tych samych nazwach.

Przykład

Wejście	Wyjście
5 rudy tygrys rudy rudy tygrys	4

Rozwiązanie

Rozwiązanie wolne: Możemy posortować wszystkie napisy i policzyć takie same pary czołgów. Sortowanie długich napisów będzie jednak dość powolne.

Rozwiązanie szybkie: Dla każdego napisu obliczmy hasz (sumę kontrolną) dla wszystkich jego liter. Teraz możemy po prostu dla każdego elementu sprawdzić, ile jest takich samych po lewo od niego. Aby uniknąć kolizji trzymajmy dwa różne hasze dla każdego czołgu.

Jak obliczyć sumę kontrolną dla nazwy czołgu?

```
napis s
p ← 263
q ← 10^9+7
w ← 0
dla i=0,1,2,..., |s| wykonuj
    w ← (p · p + s[i]) mod q
```

Teraz możemy po prostu dla każdego elementu sprawdzić, ile jest takich samych po prawo od niego. Aby uniknąć kolizji trzymajmy dwa różne hasze dla każdego czołgu.

```
hasz[n] // obliczone hasze dla każdej nazwy czołgu
dla i=0,1,2,...,n-1 wykonuj
    dla j=0,1,2,...,i-1 wykonuj
        jeżeli (hasz[i]=hasz[j])
            wynik ← wynik + 1
```

Aby uniknąć kolizji wskazane jest obliczanie dwóch różnych haszy dla każdej nazwy czołgu.

Zadanie 2. Ści(ą)gany

Dostępna pamięć: 64MB

Janek wymyślił nowy aforyzm. Ostatnio stało się to jego hobby. Zauważył jednak, że niektórzy zaczęli wykorzystywać jego sentencję w swoich wypracowaniach bez pytania go o zgodę. Trochę go to irytuje, więc postanowił sprawdzić, kto ściąga i wypisuje jego maksymę.

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba całkowita n ($1 \le n \le 100$), oznaczająca liczbę wypracowań do sprawdzenia. W drugim wierszu zapisana jest sentencja Janka: ciąg znaków nie dłuższy niż 1000000 złożony wyłącznie z dużych liter alfabetu łacińskiego. W kolejnych n liniach znajdują się wypracowania do sprawdzenia (ciągi znaków nie dłuższe niż 1000000 złożony wyłącznie z dużych liter alfabetu łacińskiego, po jednym wypracowaniu w linii).

Wyjście

W kolejnych n znajdują się komunikaty *TAK* lub *NIE* - informacja, czy w danym wypracowaniu zawarta jest sentencja Janka.

Przykład

Wejście	Wyjście
3 ABC ABABABCAB ABABABABAB AB	TAK NIE NIE

Rozwiązanie

Naszym zadaniem jest wyszukanie wzorca w tekście. Spróbujmy sprawdzać to z użyciem haszowania. W tym celu najpierw wyznaczmy potęgę $p^{|wzorzec|}$ oraz hasz wzorca (oczywiście ze względu na wartości pamiętamy tylko resztę dzielenia przez dużą liczbę pierwszą q):

```
p←127, q←10^9+7LL
pn←1
dla i=1,2,3,…,|wzorzec| wykonuj
    pn←(pn · p) mod q
dla i=|wzorzec|-1,…,2,1,0 wykonuj
    h←(pn · p + wzorzec[i]) mod q
```

Liczymy również hasze sufiksowe dla przeszukiwanego tekstu:

```
H[] // hasze sufiksowe
H[|tekst|]←0
dla i=|tekst|-1,…,2,1,0 wykonuj
    H[i]←(H[i+1] · p + tekst[i]) mod q
```

Teraz dla wskazanego tekstu sprawdźmy, czy hasz wycinka tekstu jest równy haszowi wzorca (pamiętajmy o mnożeniu przez potęgę `pn`). Gdybyśmy odejmując otrzymali wartość ujemną, dodajmy do wyniku `q`).

```
dla i=0,1,2,…,|tekst|-|wzorzec| wykonuj
    haszWycinka ← (H[i]-H[i+|tekst|*pn]) mod q
    dopóki (haszWycinka<0) haszWycinka ← haszWycinka + q
    jeżeli (h = haszWycinka)
            wypisz TAK
```

Oczywiście hasze sufiksowe obliczamy i sprawdzamy dla każdego z tekstów.

Zajęcia 29

Temat: Algorytmy tekstowe

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, struktury danych, biblioteka STL, algorytmy tekstowe, łańcuchy znaków, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem algorytmów tekstowych,
- potrafi wyszukiwać wzorzec w tekście,
- zna algorytm naiwny i KMP,

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Czy tu już byłem?	M.3, P.2.15, A.3.3,
2. Numer telefonu	M.3, P.2.15, A.3.3,

Materiały do zajęć:

http://informatykaplus.edu.pl/upload/list/czytelnia/Przeglad_podstawowych_algorytmow.pdf s. 28

Zadania do wykonania w domu:

Szablon

https://szkopul.edu.pl/problemset/problem/a3IarwgOdubufXQ89OsQz3v_/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Numer telefonu

Dostępna pamięć: 256MB

W Bajtolandii każdy numer telefonu składa się tylko z cyfr 3 oraz 5 i zawsze zaczyna się od 3. Bitek nie lubi pamiętać długich i monotonnych numerów - zamiast tego woli zapamiętywać prefikso-sufiksy dla każdej z cyfr. Zastanawia się teraz, jak odtworzyć numer telefonu i czy dobrze zapamiętał numer.

Wejście

W pierwszej linii wejścia znajduje się jedna liczba całkowita n ($1 \le n \le 10^6$), oznaczająca długość tablicy prefikso-sufiksów. W drugiej linii wejścia znajduje się n-elementowa tablica prefikso-sufiksów s_i ($0 \le s_i \le 10^6$), a jej elementy są oddzielone pojedynczymi odstępami.

Wyjście

Na wyjściu powinien w pierwszej linii znaleźć się n-cyfrowy numer telefonu odtworzony z tablicy prefikso-sufiksów lub napis NIE - odpowiedź na pytanie, czy tablica była poprawnie zapamiętana.

Przykład

Wejście	Wyjście
8 0 0 1 1 2 3 2 3	35335353

Wejście	Wyjście
3 0 0 2	NIE

Rozwiązanie

Zacznijmy od konstrukcji numeru telefonu. Pierwszą jego cyfrą jest 3. Kolejne cyfry zależą od tablicy prefikso-sufiksów. Jeżeli wartość w tablicy na i-tej pozycji wynosi 0, oznacza to, że nie mamy żadnej cyfry zgodnej z początkiem, więc musimy użyć cyfry 5. Dla wartości `pref[i]` z tablicy prefikso-sufiksów do numeru telefonu dodamy znak o indeksie `pref[i]` z dotychczasowego numeru telefonu.

```
wynik[] // tablica, w której zapamiętamy numer telefonu
wynik[0] ← 3
dla i=1,2,…,n-1 wykonuj
    jeżeli (pref[i] = 0)
        wynik[i] ← 5
    przeciwnie
        wynik[i] ← pref[wynik[i]]
```

Teraz obliczmy tablicę prefikso-sufiksów `K[]` (algorytm KMP) dla obliczonej tablicy `wynik[]`.

```
K[0] ← 0
dla i=1,1,2,...,n-1 wykonuj
    //zakładamy zgodność dla następnej cyfry
    K[i] ← K[i-1]
    // jeżeli litery w naszym zapisie się nie zgadzają,
    //zmniejszamy wartość aktualnego na poprzedni  prefikso-sufiks
    dopóki (K[i]>0 ∧ wynik[K[i]] ≠ wynik[i])
        K[i] ← K[K[i]-1]
    //jeżeli bieżąca litera się zgadza,
    //zwiększamy długość aktualnego prefikso-sufiksu o 1
    jeżeli (wynik[K[i]] = wynik[i])
        K[i] ← K[i]+1
```

Teraz wystarczy sprawdzić, czy tablica `K[]` oraz `pref[i]` są równe.

Zadanie 2. Czy tu już byłem?

Dostępna pamięć: 256MB

Bajtek zwiedzał Bajtogród przez cały dzień. Wyraźnie widać już po nim zmęczenie. Podobają się mu jednak tutejsze skrzyżowania. Wokół każdego z nich stoi zawsze dokładnie *n* budynków. Skrzyżowania różnią się od siebie tylko wysokościami poszczególnych domów. Każde skrzyżowanie jest otoczone budynkami o charakterystycznym dla siebie ciągu wysokości.

Bajtek znalazł się właśnie na kolejnym skrzyżowaniu, ale w żaden sposób nie może sobie przypomnieć, czy wcześniej już tu był. Na szczęcie każde skrzyżowanie, które dzisiaj odwiedził, opisał w zeszycie. Który to już raz Bajtek odwiedził aktualne skrzyżowanie?

Wejście

W pierwszej linii wejścia znajdują się dwie liczby całkowite n oraz k ($1 \le n \le 10^5, 1 \le k \le 100$), oznaczające odpowiednio liczbę budynków na każdym ze skrzyżowań oraz liczbę skrzyżowań, które wcześniej odwiedził Bajtek. W drugiej linii wejścia znajduje się n-elementowy ciąg wysokości kolejnych budynków na aktualnym skrzyżowaniu s_i ($1 \le s_i \le 10^6$), a jego elementy są oddzielone pojedynczymi odstępami. W kolejnych k liniach znajdują się opisy odwiedzonych wcześniej przez Bajtka skrzyżowań. Każdy opis to n-elementowy ciąg wysokości kolejnych budynków na jednym z odwiedzonych skrzyżowań o_i ($1 \le o_i \le 10^6$).

Wyjście

Na wyjściu w kolejnych k liniach wypisz komunikat TAK lub NIE - odpowiedź na pytanie, czy Bajtek był już wcześniej na tym skrzyżowaniu. Bajtek mógł odwiedzić aktualne skrzyżowanie wcześniej więcej niż raz.

Przykład

Wejście	Wyjście
5 3 1 2 3 4 5 2 3 4 5 1 3 4 5 1 2 3 5 4 1 2	TAK TAK NIE

Ocenianie

Podzadanie	Ograniczenia	Liczba punktów
1	$n, k \le 10^2$	25
2	brak dodatkowych założeń	75

Rozwiązanie

W zadaniu musimy sprawdzić równoważność cykliczną dwóch słów. Najprostszym sposobem będzie wyszukanie wzorca `w` w tablicy `tt` (podwójna tablica `t`). Możemy tu użyć algorytmu KMP na tablicach `w+t+t` (+ oznacza połączenie tablic w jedną).

```
wynik[] //tablica złożona z tablic w, t oraz t;
```

```
        //pomiędzy tablicę w i t dopiszemy wartość -1
K[0] ← 0
dla i=1,2,…,3·n wykonuj
    //zakładamy zgodność dla następnej cyfry
    K[i] ← K[i-1]
    // jeżeli litery w naszym zapisie się nie zgadzają,
    //zmniejszamy wartość aktualnego na poprzedni prefikso-sufiks
    dopóki (K[i]>0 ∧ wynik[K[i]] ≠ wynik[i])
        K[i] ← K[K[i]-1]
    //jeżeli bieżąca litera się zgadza,
    //zwiększamy długość aktualnego prefikso-sufiksu o 1
    jeżeli (wynik[K[i]] = wynik[i])
        K[i] ← K[i]+1
```

Jeżeli teraz w tablicy prefikso-sufiksów `K[]` na pozycjach większych niż `|w|` znajdziemy prefikso-sufiks o długości `|w|`, oznacza to, że znaleźliśmy się na już odwiedzonym wcześniej skrzyżowaniu.

Można również użyć algorytmu działającego także w czasie O(n) i nie zużywającego dodatkowej pamięci. Jego opis można znaleźć tu:

http://www.rafalnowak.pl/wiki/index.php?title=R%C3%B3wnowa%C5%BCno%C5%9B%C4%87_cykliczna_dw%C3%B3ch_s%C5%82%C3%B3w

Zajęcia 30

Temat: Zawody 3. Powtórzenie i podsumowanie.

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, struktury danych, biblioteka STL, grafy, łańcuchy znaków, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem grafów, struktur danych,

Formy i metody pracy: praca samodzielna, sparing

Zadania do wykonania na zajęciach	Treści programowe
1. Nawiasy	M.1, P.2.18, A.1
2. Wojna domowa	M.5, P.2.18, A.3.7, A.3.8

Nawiasy

Limit pamięci: 64MB

Kiedy czasem dostaniemy do obliczenia wartość wyrażenia arytmetycznego, już sam pierwszy rzut oka sprawia, że mamy ochotę rozwiązać to zadanie pojutrze i mieć dzięki temu dwa dni wolnego. Najgorsze są jednak przypadki takie, w których po długich i wyczerpujących obliczeniach dowiadujesz się, że otrzymałeś błędny zapis wyrażenia!

Napisz program, który sprawdzi, czy dla podanego zapisu użyte w nim nawiasy umieszczone są w możliwy do przyjęcia sposób (zachowana jest właściwa kolejność nawiasów otwierających i zamykających, każdy nawias musi być sparowany z nawiasem tego samego typu).

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba n ($1 \leq n \leq 10$). W kolejnych n wierszach znajduje się po jednym wyrażeniu złożonym z nie więcej niż 1 000 000 znaków. Wyrażenie składa się z nawiasow: (,), { ,}, [,], <, >.

Wyjście

W n wierszach standardowego wyjścia Twoj program powinien zapisać jedną literę ‘T’ lub ‘N’, oznaczającą poprawność zapisu nawiasow w wyrażeniu arytmetycznym.

Przykład

Wejście 3	Wejście T

([]) <(>) {()()}	N T

Wojna domowa

Dostępna pamięć: 256MB

Bajtocja przez lata rozwijała się przekształcając się stopniowo w nowoczesny kraj. Jedyne co przeszkadzało w dalszym rozwoju państwa, to słabo rozwinięta sieć drogowa. Król Baltazar postanowił ulepszyć transport w kraju. W tym celu każda droga została zamieniona w jednokierunkowy trakt. W ten sposób zwiększyła się przepustowość dróg, ale nie zawsze było już możliwe dotarcie z dowolnego miasta do każdego innego! Doprowadziło to dalszych napięć w kraju, aż w końcu w Bajtocji wybuchła wojna domowa! W rezultacie rewolucji powstało wiele nowych królestw. Miasta podzieliły się na nowe państwa według bardzo prostej zasady: jeśli istnieje droga z miasta **a** do **b** i z miasta **b** do **a**, to są one częścią tego samego królestwa.

Gildia kupiecka stara się odnaleźć w nowej rzeczywistości. Aby dalej prowadzić interesy, musi wiedzieć, ile królestw powstało w wyniku rewolucji, oraz czy istnieje droga (w dwóch kierunkach) miedzy dwoma wskazanymi miastami.

Napisz program, który pomoże w działalności gildii kupieckiej!

Wejście

W pierwszej linii wejścia znajdują się dwie liczby całkowite **n** oraz **k** ($1 \le \mathbf{n} \le 2 \cdot 10^5$, $1 \le \mathbf{k} \le 10^6$) oznaczające liczbę państw w Bajtocji oraz liczbę istniejących dróg. W kolejnych **k** liniach znajdują się po dwie liczby całkowite **u** oraz **w** ($1 \le \mathbf{u}, \mathbf{w} \le \mathbf{n}$, $\mathbf{u} \ne \mathbf{w}$) oznaczające odpowiednio miasto początkowe oraz miasto końcowe, pomiędzy którymi istnieje droga po reformach Bajtazara.

W **k** + 1 linii wejścia znajduje się jedna liczba całkowita **m** ($1 \le \mathbf{m} \le 10^5$) oznaczająca liczbę par miast do sprawdzenia. W kolejnych m liniach znajduje się po dwie liczby całkowite **a** oraz **b** ($1 \le \mathbf{a}, \mathbf{b} \le$ n) oznaczające numery miast, dla których należy sprawdzić, czy istnieje droga (w obu kierunkach).

Wyjście

Na wyjściu w pierwszym wierszu powinna znaleźć się jedna liczba całkowita oznaczająca liczbę różnych państw, na które podzieliła się Bajtocja.

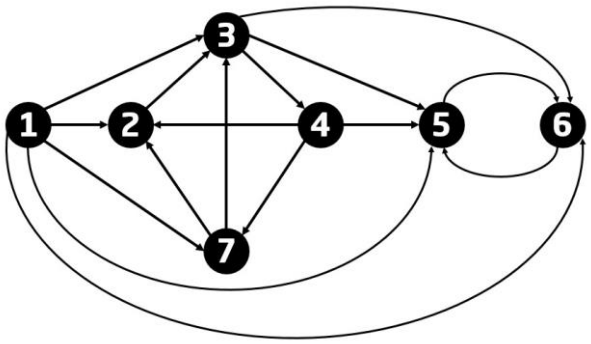
W kolejnych **m** liniach znajdują się odpowiedzi na pytania, czy istnieje dwukierunkowa droga miedzy kolejną parą miast **a** i **b** (**TAK** lub **NIE**).

Przykład

Dla danych wejściowych:	poprawnym wynikiem jest:
7 16 1 2 2 3 3 4 4 2 7 2 4 5	3 TAK NIE TAK TAK

5 6 6 5 4 7 1 7 1 3 3 5 3 6 7 3 1 5 1 6 4 7 2 1 5 6 5 4 2	

Przykładowe państwo:



Ocenianie

Podzadanie	Ograniczenia	Punkty
1	$n,k,m \leq 10^3$	30
2	brak dodatkowych założeń	70

Podpowiedzi do rozwiązań

Zadanie 1. Wojna domowa. Zadanie to wymaga wyznaczenia liczby silnie spójnych składowych w grafie skierowanym, a następnie sprawdzanie, czy pary liczb należą do tej samej spójnej. .

Zadanie 2. Nawiasy. W rozwiązaniu zadania należy wykorzystać strukturę stosu. Każdy nawias otwierający wstawiamy na stos. Każdy nawias zamykający powinien zdjąć ze stosu swój odpowiednik. Wyrażenie jest poprawne, jeśli nigdy nie trafimy na niepoprawna parę, a po jego całkowitym przetworzeniu stos będzie pusty.

8. Warunki, w tym infrastruktura, niezbędne do realizacji zajęć (sprzęt, oprogramowanie, zasoby internetowe).

Realizacja programu musi odbywać się w pracowniach komputerowych wyposażonych w zestawy komputerowe dostępne indywidualnie dla każdego ucznia. Komputer musi być wyposażony w podstawowe oprogramowanie umożliwiające realizację programu koła:

- przeglądarkę internetową,
- środowisko programistyczne C++ (on-line: ideone.com, repl.it, desktop: DevC++, CodeBlocks),

oraz zalecane:

- edytor tekstu,
- prosty edytor plików tekstowych (darmowy Notepad++),
- system Linux (ewentualnie dostępny w wersji bootowalnej zamieszczony na przenośnym nośniku danych).

Nauczyciel powinien mieć założone konto nauczycielskie na portalu szkopul.edu.pl (sugerowane założenie własnego konkursu), zaś uczeń - konto uczniowskie na www portalu.

9. Literatura i inne zasoby edukacyjne.

portal: szkopul.edu.pl

portal: main2.edu.pl

portal: smurf.mimuw.edu.pl/uczesie

portal: algorytm.org

Wiki Rafała Nowaka: www.rafalnowak.pl>wiki

Olimpiada Informatyczna Juniorów: oij.edu.pl

Olimpiada Informatyczna: oi.edu.pl

Zaprzyjaźnij się z algorytmami. Jacek Tomasiewicz. PWN Warszawa 2015.

Przygody Bajtazara. 25 lat Olimpiady Informatycznej. Praca zbiorowa. Wybór zadań. PWN Warszawa 2018.

Wprowadzenie do algorytmów. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, PWN Warszawa 2018.

Karol Kuczmarski, Kurs C++ od zera do gier kodera

Witold Lipski, Kombinatoryka dla programistów, WNT

Godzina Kodowania http://godzinakodowania.pl