Program kształcenia z informatyki dla kół informatycznych - I rok nauczania

Projekt zgłoszony do konkursu Fundacji Rozwoju Informatyki i Centrum Rozwiązań Strategicznych im. Jana Łukasiewicza

Autor projektu

dr Lech Duraj
Wydział Matematyki i Informatyki,
Uniwersytet Jagielloński
lech.duraj@gmail.com
tel.888205025

Doświadczenie w pracy z uzdolnionymi uczniami

Od 2007 roku prowadzę zajęcia w V Liceum Ogólnokształcącym w Krakowie, w klasach o profilu “matematyczno-algorytmicznym”. Są to 2 godziny tygodniowo przedmiotu “algorytmika”, na prawach przedmiotu fakultatywnego. Współorganizuję też wszelkiego rodzaju obozy dla uczniów V LO, w których biorą udział (lokalnie i zdalnie) uczniowie z innych miast. W czasie mojej pracy w tej szkole opiekowałem się kilkudziesięcioma laureatami i finalistami Olimpiady Informatycznej, w tym medalistami Międzynarodowej Olimpiady Informatycznej.
Od 2017 roku współorganizuję na wydziale Jagiellońskie Warsztaty Olimpijskie, przeznaczone dla uczniów spoza Krakowa, zapoznające uczestników z podstawami algorytmiki oraz “olimpijskiego” programowania.
Od 2011 roku jestem członkiem kadry naukowej Obozu Naukowo-Treningowego im. Antoniego Kreczmara, a od 2015 roku kierownikiem naukowym obozu.

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

Podstawowym celem zaprezentowanego programu kształcenia jest zapoznanie uczniów z podstawami algorytmiki, w szczególności - chociaż nie tylko - pojęciami i technikami przydatnymi na Olimpiadzie Informatycznej i innych zawodach programistycznych. Szczegółowa lista omawianych zagadnień podana jest w sekcji “Plan kursu”.

Dodatkowe cele, którym służą zajęcia według podanego programu:

- Nauka efektywnego pisania kodu - zwracanie uwagi na istotne w programowaniu drobiazgi: odpowiednie wcięcia, unikanie kopiowania i powtarzającego się kodu, planowanie programu tak, aby był czytelny i zrozumiały etc.
- Naprawianie błędów w programach (debugging) - programowanie z użyciem sprawdzarki zachęca do poprawnej i efektywnej implementacji algorytmów, a także wyszukiwania i poprawiania błędów, w tym tworzenia własnych testów.

Podstawowe założenia programu są następujące:

- **Najlepsze zrozumienie algorytmów osiąga się przez ich własnoręczne zaprogramowanie**. Dlatego też centralną rolę w kursie pełni system *online judge* (“sprawdzarka”), do którego uczniowie wysyłają wszystkie napisane kody. Do każdego omawianego tematu dołączone są 1-2 przykładowe zadania do implementacji przez uczniów. Program ucznia musi działać prawidłowo (przejść wszystkie testy na sprawdzarce), ale można próbować dowolną ilość razy.
- **Algorytmy powinny być dokładnie wytłumaczone na zajęciach, ale implementacja przez ucznia musi być samodzielna**. Wymaga to zachowania równowagi podczas prowadzenia zajęć - z jednej strony na tablicy powinien pojawić się opis algorytmu pozwalający wszystkim uczniom na dokładne pojęcie jego idei, z drugiej strony - nie może to być kod algorytmu, który stwarza pokusę przepisania go bez zrozumienia.
- **Przy uczeniu się algorytmiki uczniowie wymagają nieustannej pomocy.** Nauka poprzez sprawdzarkę może być frustrująca, szczególnie przy szukaniu dobrze ukrytych błędów. Trzeba zachęcić uczniów do szukania pomocy, zarówno u nauczyciela (zależnie od indywidualnych możliwości, przez dodatkowe konsultacje, przez e-mail albo nawet przez komunikator internetowy), jak i u siebie nawzajem. Warto podkreślić, że programowanie jest na ogół działaniem zespołowym, a umiejętności komunikacji są ważne na równi z algorytmiką i kodowaniem. Dobrze jednak przy tym zaznaczyć granicę: kopiowanie lub przepisywanie cudzego kodu jest zarówno nieuczciwe, jak i szkodliwe dla wszystkich.
- **Pojęcia abstrakcyjne należy wprowadzać ostrożnie.** Do takich pojęć zaliczam, między innymi:
    - złożoność algorytmu
    - notację O()
    - niezmiennik algorytmu
    - strukturę danych

  W wypadku takich pojęć nie należy zaczynać od definicji, a tym bardziej od formalnej definicji, która w najlepszym wypadku zostanie natychmiast zapomniana, a w najgorszym nastawi uczniów negatywnie i utrudni zrozumienie pojęcia w przyszłości. Konieczne jest budowanie intuicji przez wprowadzenie najpierw odpowiedniej liczby przykładów (najlepiej na przestrzeni kilku zajęć), a dopiero potem wspólnej nazwy i ewentualnie definicji.
- **Konstrukcje języka C++ i pojęcia matematyczne wprowadzane są tam, gdzie są potrzebne** - każda nowa konstrukcja powinna być w takim miejscu kursu, aby od razu

przydała się w implementowanym algorytmie. Dlatego np. wektor jest wprowadzany przy własnej arytmetyce, struktura (*struct*) przy sortowaniu, a pojęcie funkcji/podprocedury tam, gdzie pierwszy raz opłaca się w taką funkcję wydzielić część kodu. Poza pierwszymi 3-4 zajęciami, kurs unika tłumaczenia nowych konstrukcji języka samodzielnie, w oderwaniu od algorytmów - w ten sposób uczniowie od razu zapamiętują, do czego dana instrukcja służy i jak się jej używa w programie. Analogiczne podejście przyjąłem przy pojęciach matematycznych - nie ma osobnych zajęć poświęconych np. logice matematycznej lub kombinatoryce. To jednak może zależeć od poziomu i intensywności nauczania matematyki w danej szkole i w tym zakresie program może wymagać modyfikacji.

- **Kiedy to tylko możliwe, należy stwarzać uczniom możliwość, aby do algorytmu doszli sami.** Sprawdzoną praktyką jest określenie problemu do rozwiązania (np. “chcemy mieć posortowaną tablicę”), zbieranie od uczniów pomysłów na rozwiązanie go i odpowiednie moderowanie dyskusji.

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

W pierwotnej wersji program nie ma wstępnych wymagań z zakresu programowania i algorytmiki - jest przeznaczony dla uczniów, którzy mogli nawet w ogóle nie mieć styczności z programowaniem. Bardzo łatwo go przystosować do grupy uczestników, którzy mają już obycie z językiem C++, pomijając niektóre z początkowych zajęć, przy innych wybierając trudniejsze warianty zadań, ewentualnie dokładając dodatkowe zadania ćwiczące sprawność programistyczną uczniów. W V LO tradycyjnie na początku I klasy dzieli się uczniów na dwie grupy (“znający C++” i “nie znający C++”), przy czym po roku zajęć grupy łączą się ze sobą.
Uczniowie przystępujący do programu powinni mieć obycie matematyczne pozwalające zrozumieć pojawiające się w algorytmice pojęcia z teorii liczb i kombinatoryki (funkcje, liczby pierwsze, dzielniki, logarytm dwójkowy, systemy zapisu w różnych podstawach etc.) Są to jednak raczej podstawowe pojęcia i można z dobrym przybliżeniem uznać, że jeśli uczeń radzi sobie z rozszerzoną matematyką w szkole średniej, to może przystąpić do programu.
Trudniej zweryfikować, czy uczniowie radzą sobie z samodzielnym programowaniem i rozumieniem algorytmów. W zasadzie jedyną rozsądną metodą jest weryfikacja po kilku zajęciach, czy poszczególni uczniowie “nadążają” za grupą (w V LO w Krakowie kwestia ta jest rozwiązywana poprzez możliwość względnie swobodnej zmiany profilu klasy przez pierwsze 2-3 miesiące nauki, a także możliwość rezygnacji po I roku z fakultetu algorytmicznego).

Częstotliwość systematycznych zajęć, czas trwania jednych zajęć

Jedne zajęcia planowane są na 2 godziny lekcyjne (1h 30m) odbywające się co tydzień. Do każdych zajęć dołączone są zadania do rozwiązania w systemie *online judge*. Zależnie od “siły” grupy i poziomu trudności konkretnych zadań, zadania powinny mieć termin od 1 do 2 tygodni.

Krótkie terminy zachęcają do systematycznej pracy na bieżąco, z kolei termin dwutygodniowy pozwala na powrót, po tygodniu, do ostatnio omawianego tematu i wyjaśnienie wątpliwości, które pojawiły się podczas prób samodzielnej implementacji przez uczniów.
Odradzane są dłuższe terminy - uczniowie mają tendencję do zabierania się do zadań blisko ostatecznego terminu, a po kilku tygodniach nie będą już pamiętali, jak wyglądał algorytm.

Plan kursu

Kurs jest podzielony na działy, w każdym jest 8-10 zadań. Przy każdym temacie podana jest szacunkowa liczba godzin, którą jednak należy traktować z pewną rezerwą. Tempo kursu może się bardzo różnić w zależności od grup uczniów, czasem też zachodzi potrzeba dorzucenia więcej zadań z jakiegoś działu/tematu.
W osobnym pliku załączone są treści wszystkich zadań wspomnianych w poniższej tabeli.

Część A: “Nauka programowania”				
Lp.	Temat zajęć	Liczba godzin	Treści programowe	Zadania na sprawdzarce
1	Wprowadzenie	2-4	● zapoznanie uczniów z obsługą komputera, systemem plików, środowiskiem programistycznym ● dyskusja o tym, co to jest algorytm i czym zajmuje się algorytmika ● gra w “za dużo, za mało” i strategia połowienia (“wyszukiwanie binarne”) jako przykład efektywnego algorytmu ● pojęcie logarytmu dwójkowego	
2	Zapoznanie z językiem C++ i sprawdzarką	2-4	● konstrukcja programu ● pojęcie zmiennej ● wczytywanie i wypisywanie ● instrukcja warunkowa ● sprawdzarka: wysyłanie programów, komunikaty	*Hello, world!* *Blackjack*
3	Pętle, tablice	2	● pętla *for* i *while* ● tablica	*Suma 10 liczb* *Odwróć liczby*
4	Pętle II, łańcuchy	2	● podwójne pętle ● zmienne typu *string*	*Prostokąt* *Palindrom*

			● rozpoznawanie palindromów	
5	Algorytm Euklidesa	2	● algorytm Euklidesa w wersji z pętlą *while* i odejmowaniem ● poprawność i efektywność (szukanie trudnych testów) ● algorytm w wersji efektywnej (z resztą z dzielenia) ● [opcjonalnie] dowód, że algorytm działa w czasie logarytmicznym	*Algorytm Euklidesa*
6	Funkcje, sprawdzanie pierwszości	4	● sprawdzanie pierwszości algorytmem naiwnym (wszystkie dzielniki) ● algorytm “pierwiastkowy” i porównanie efektywności ● wydzielenie fragmentu ze sprawdzaniem pierwszości do osobnej funkcji ● typ logiczny (*bool*)	*Zielone Butelki* *Faktoryzacja [opcjonalnie]*

Część B: “Rozgrzewka algorytmiczna”				
Lp.	Temat zajęć	Liczba godzin	Treści programowe	Zadania na sprawdzarce
1	Własna arytmetyka	4	● Wektory jako alternatywa dla tablicy, metody klasy *vector* ● Reprezentacja dużych liczb - wczytywanie jako łańcuch, zamiana na wektor ● Dodawanie pisemne - pętla z przeniesieniem ● [opcjonalnie] Odejmowanie pisemne ● Mnożenie pisemne - algorytmem “każda cyfra przez każdą, usuń przeniesienia”	*Dodawanie* *Odejmowanie* *Mnożenie*
2	Rekursja, szybkie potęgowanie	2	● Rekurencja na przykładzie silni ● Potęgowanie - analogicznie do silni, iteracyjnie i	*Szybkie potęgowanie*

			rekurencyjnie ● Przyśpieszenie potęgowania przez zmianę wywołania rekurencyjnego	
3	Systemy liczbowe	2	● Zapis liczby w różnych podstawach ● Szczególna rola systemu dwójkowego ● Algorytm zamiany pomiędzy systemami, wersja iteracyjna i rekurencyjna	*Systemy liczbowe*
4	Proste sortowanie	2-4	● sortowanie bąbelkowe, przez selekcję, przez wstawianie ● liczenie porównań i przestawień w każdym z tych algorytmów ● zastosowania praktyczne (małe/duże dane)	*Sortowanie*
5	Wyszukiwanie binarne	4	● idea wyszukiwania binarnego: szukanie “właściwej” liczby (rozwiązywanie równań) ● szukanie elementu w tablicy ● warianty: pierwszy i ostatni element	*Zagadka Nicolo Tartaglii* *Naczelny Statystyk*

Część C: “Struktury danych, sortowanie i STL”				
Lp.	Temat zajęć	Liczba godzin	Treści programowe	Zadania na sprawdzarce
1	Sortowanie za pomocą STL-a	2	● Funkcja *sort()* ● Własna funkcja porównująca (komparator) ● Typ strukturalny (*struct*) ● Sortowanie kątowe (w najprostszej wersji) jako przykład sortowania z komparatorem	*Panorama*
2	Kolejka	2	● Struktura kolejki, klasa *queue* i jej metody ● Własna implementacja kolejki: wariant nieograniczony	*Wojna*

			(wektor) i cykliczny, konsekwencje dla efektywności i pamięci	
3	Stos	2-4	● Struktura stosu, klasa *stack* i jej metody ● Własna implementacja (przez tablicę/wektor) ● Odwrotna Notacja Polska: algorytm obliczania wartości ● [opcjonalnie] Zamiana wyrażenia na ONP (“algorytm górki rozrządowej”)	*Harry Potter i struktury danych* *Odwrotna Notacja Polska* *Odwrotna Notacja Polska kontratakuje*
4	Kolejka priorytetowa	2	● Klasa *priority_queue* ● [bardzo opcjonalnie] pojęcie kopca, idea sortowania przez kopcowanie	*Wybory*
5	Sortowanie przez scalanie	2	● Scalanie posortowanych wektorów/tablic ● Algorytm MergeSort ● [opcjonalnie] Dowód złożoności O(n log n) ● Liczenie inwersji	*Sierżant*
6	Sortowanie szybkie	2	● Element dzielący i podział tablicy ● Algorytm QuickSort ● Dobre i złe podziały, losowy wybór elementu dzielącego ● Problem 2-SUM (dwa elementy w tablicy sumujące się do zadanej liczby)	*Randka w ciemno*

Część D: “Grafy, podstawowe algorytmy”				
Lp.	Temat zajęć	Liczba godzin	Treści programowe	Zadania na sprawdzarce
1	Pojęcie grafu, algorytm BFS	4	● Pojęcie grafu ● Przykłady grafów “z życia”: mapa połączeń, sieć społecznościowa ● Spójność, spójne składowe ● Ścieżka, odległość	*facepalm.bt*

			● Implementacja grafu: przez macierz i przez listę sąsiedztwa ● Algorytm BFS	
2	Dwukolorowalność	2	● Algorytm dwukolorowalności ● DFS jako prostszy w implementacji wariant przeszukiwania	*Anty-portal* *Autostrady i polityka*
3	BFS na nietypowych grafach	2	● Graf punktów kratowych(*grid*) ● Niejawna (tworzona w czasie algorytmu) lista sąsiadów ● Modyfikacja grafu do BFS (na przykładzie podwojenia wierzchołków)	*Loch [początkującego] czarnoksiężnika*
4	DFS i sortowanie topologiczne	2	● DFS na nieskierowanych i skierowanych grafach ● Krawędzie drzewowe i wsteczne, kolejność wejścia i wyjścia ● Wykrywanie cykli ● Sortowanie topologiczne jako “efekt uboczny” DFS	*Komisja śledcza*
5	Silnie spójne składowe	2	● Pojęcie silnie spójnej składowej ● Algorytm Kosaraju (DFS+transpozycja grafu+DFS) ● [bardzo opcjonalnie] Szkic dowodu poprawności	*Euro 2102*

Część E: “Programowanie dynamiczne - wprowadzenie”				
Lp.	Temat zajęć	Liczba godzin	Treści programowe	Zadania na sprawdzarce
1	Wprowadzenie do programowania dynamicznego	2	● Technika bottom-up: podzadania, kolejność obliczania ● Zamiana rekursji na programowanie dynamiczne (np. liczby Fibonacciego)	*Piramida* *Mosiężny Most*

2	Przyśpieszanie algorytmów - przydatne triki	2	● Ograniczenie sprawdzanych podzadań, na przykładzie zadania *Remont autostrady* ● Najdłuższy podciąg rosnący	*Remont autostrady* *Chorąży*
3	Dwuwymiarowe programowanie dynamiczne	2-4	● Proste problemy z dwuwymiarową tablicą (*Diamenty*) ● Najdłuższy wspólny podciąg (LCS) ● Optymalizacja pamięciowa w algorytmach dynamicznych ● Odtwarzanie wyniku w algorytmach dynamicznych	*Diamenty* *Neon* *[dwie wersje]*
4	Problem plecakowy	2	● Klasyczny problem plecakowy ● Optymalizacja pamięciowa i odtwarzanie wyniku dla problemu plecakowego	*Skarb faraona*

Część F: “Drzewa i drzewa przedziałowe”				
Lp.	Temat zajęć	Liczba godzin	Treści programowe	Zadania na sprawdzarce
1	Drzewa nieukorzenione	2	● Drzewa jako grafy, liczba krawędzi w drzewie ● DFS i BFS na drzewach ● Średnica drzewa, najdalsze wierzchołki	*Za siedmioma górami*
2	Drzewa z korzeniem	2	● Reprezentacja drzew (przez listę dzieci), albo przez pierwszego syna/pierwszego brata ● Porządek preorder i postorder	*Rzeki Bajtocji*
3	Drzewa przedziałowe - wprowadzenie	4	● Drzewo turniejowe i inne drzewa punkt-przedział, implementacja w tablicy ● Suma/maksimum na przedziale, procedura rekurencyjna “od dołu drzewa” ● Drzewa przedział-punkt	*Powrót Naczelnego* *Statystyka* *Podatki*

			• Procedura rekurencyjna “od góry drzewa”	
4	Zaawansowane drzewa przedziałowe	4-6	• Drzewa typu “przedział-przedział” • Ogólna postać procedury w węźle drzewa (rekursja+naprawienie) • Opóźnione spychanie (*lazy propagation*) • Drzewa na nietypowych obiektach kombinatorycznych (np. permutacje)	*Koleje* *Siła złego* *Tetris 2D* *Skrętka*

Materiał na dalsze lata kursu:

G. Grafy, najkrótsze ścieżki

1. Algorytm Forda-Bellmana, cykle ujemne
2. Algorytm Dijkstry
3. Algorytm Floyda-Warshalla
4. Problem Find-Union
5. Minimalne drzewo rozpinające i algorytm Kruskala

H. Algorytmy tekstowe

1. Palindromy i algorytm Manachera
2. Tablica prefiksowo-sufiksowa, okres słowa, algorytm KMP
3. Tablica prefiksowo-prefiksowa
4. Hashowanie i algorytm Karpa-Rabina
5. Algorytm Karpa-Millera-Rosenberga

I. Algorytmy dynamiczne i zachłanne

1. Algorytmy typu “wszystkie przedziały” (triangulacja wielokąta i pokrewne)
2. Gry na przedziałach
3. Algorytmy na maskach bitowych
4. Algorytmy dynamiczne z sortowaniem topologicznym
5. Algorytmy zachłanne i dowody poprawności

J. Grafy, zaawansowane algorytmy

1. Problem 2-SAT
2. Punkty artykulacji, mosty i dwuspójne składowe

3. Cykle Eulera
4. Najniższy wspólny przodek
5. Skojarzenia w grafach dwudzielnych

Szczegółowy plan zajęć

W ramach pracy konkursowej załączam, w osobnym dokumencie, szczegółowy plan zajęć od A-3 do B-5 (24 godziny zajęć), a także archiwum z treściami wszystkich proponowanych zadań.

Koncepcja i opis innych form kształcenia

Podany program kursu koncentruje się na zapoznaniu uczniów z podstawowymi algorytmami. Należy zadbać o to, aby rozwijać też umiejętność samodzielnego rozwiązywania zadań i projektowania algorytmów. W tym celu proponuję następujące dodatkowe formy aktywności:

1. **Dodatkowe zadania do samodzielnego rozwiązania** - najlepsze są zadania olimpijskie, powiązane z aktualnie omawianym tematem. Zadania nie są rozwiązywane na zajęciach i nie są obowiązkowe - uczniowie rozwiązują je we własnym zakresie i we własnym tempie.
2. **Mini-sparingi** - co jakiś czas warto poświęcić zajęcia na zorganizowanie mini-zawodów. Na sprawdzarkę należy przygotować ok. 4-5 zadań, koniecznie o zróżnicowanym stopniu trudności - tak aby najprostsze zrobili wszyscy uczniowie, a najtrudniejsze co najwyżej 1-2 osoby. Na swoich zajęciach preferuję konkursy typu ACM (analogicznie jak na lekcjach, zadania sprawdzane w skali 0/1, czyli akceptacja/brak akceptacji, zgłoszenia do skutku), ale konkursy olimpijskie (skala 0-100) też mogą się sprawdzić. Warto też rozważyć konkurs zespołowy (3-osobowe drużyny z jednym komputerem) - na początku uczniom będzie w takich zespołach szło gorzej, ale oprócz uczenia umiejętności komunikacji w zespole, ta forma daje uczniom znacznie więcej radości z konkursu.
3. **Warsztaty/obozy** - według tradycyjnego i sprawdzonego schematu: całodniowe konkursy w warunkach zbliżonych do Olimpiady Informatycznej.

Warunki, w tym infrastruktura, niezbędne do realizacji zajęć

Do nauki programowania potrzebna jest pracownia komputerowa. Po opanowaniu przez uczniów podstaw programowania, zajęcia można prowadzić przy tablicy, pozostawiając uczniom pisanie programów we własnym zakresie.
Komputery w pracowni mogą pracować pod systemem Windows lub Linux, konieczny jest jedynie kompilator C++ oraz środowisko umożliwiające pisanie i kompilację w tym języku (np. Codeblocks/Geany).
Centralnym elementem kursu jest system *online judge*, w którym nauczyciel musi mieć możliwość tworzenia konkursów i dodawania zadań. W V LO w Krakowie używamy systemu

Satori napisanego i utrzymywanego na Uniwersytecie Jagiellońskim. System ten jest wprawdzie otwarty i dostępny dla chętnych, ale technicznie bardzo trudny w obsłudze. W praktyce zalecam jedno z dwóch poniższych rozwiązań:

- Do większości zastosowań dobrze nadaje się serwis szkopul.edu.pl, z możliwością tworzenia konkursów oraz zamieszczania własnych zadań, w formacie wprowadzonym przez Olimpiadę Informatyczną.
- Jeśli jest potrzeba niestandardowych rozwiązań/modyfikacji istniejących, można rozważyć postawienie "od zera" systemu DOMJudge - sprawdzarki *open source*, z powodzeniem używanej na zawodach informatycznych na całym świecie.

Literatura i inne zasoby edukacyjne

Niestety, na rynku jest stosunkowo niewiele dobrych podręczników algorytmiki dla początkujących. Zdecydowanie najlepszą dostępną w języku polskim pozycją jest:

- J. Tomasiewicz, Zaprzyjaźnij się z algorytmami, PWN, Warszawa, 2019.

Aby mieć wyczucie, które algorytmy są ważne z punktu widzenia potencjalnych olimpijczyków, warto zapoznać się z "niebieskimi książeczkami" - opracowaniami zadań z Olimpiady Informatycznej, dostępnymi na stronie www.oi.edu.pl. Wybór najciekawszych zadań, a także kilka wykładów na interesujące tematy algorytmiczne można znaleźć w książce:

- Przygody Bajtazara. 25 lat Olimpiady Informatycznej – wybór zadań, PWN, Warszawa 2018.

Zadania na różne poruszone na kursie tematy - zarówno zadania do wykorzystania jako "podstawowe" jak i "dodatkowe" można znaleźć w różnych miejscach internetu:

- www.szkopul.edu.pl - zadania z Olimpiady Informatycznej i innych polskich konkursów;
- ICPC Live Archive - zadania ze studenckich konkursów ICPC;
- codeforces.com - rosyjski serwis z regularnymi sparingami i bardzo aktywną społecznością programistów-zawodników.