Webinarium przeprowadzone w ramach projektu "Mistrzostwa w Algorytmice i Programowaniu - Uczniowie", **finansowanego przez:**

Piotr Chrząstowski

Zbioru z punktu widzenia informatyka

Zbiory

- Co to jest zbiór?
- Z czego można zbiór utworzyć?
- Czy każdy zbiór da się poznać?

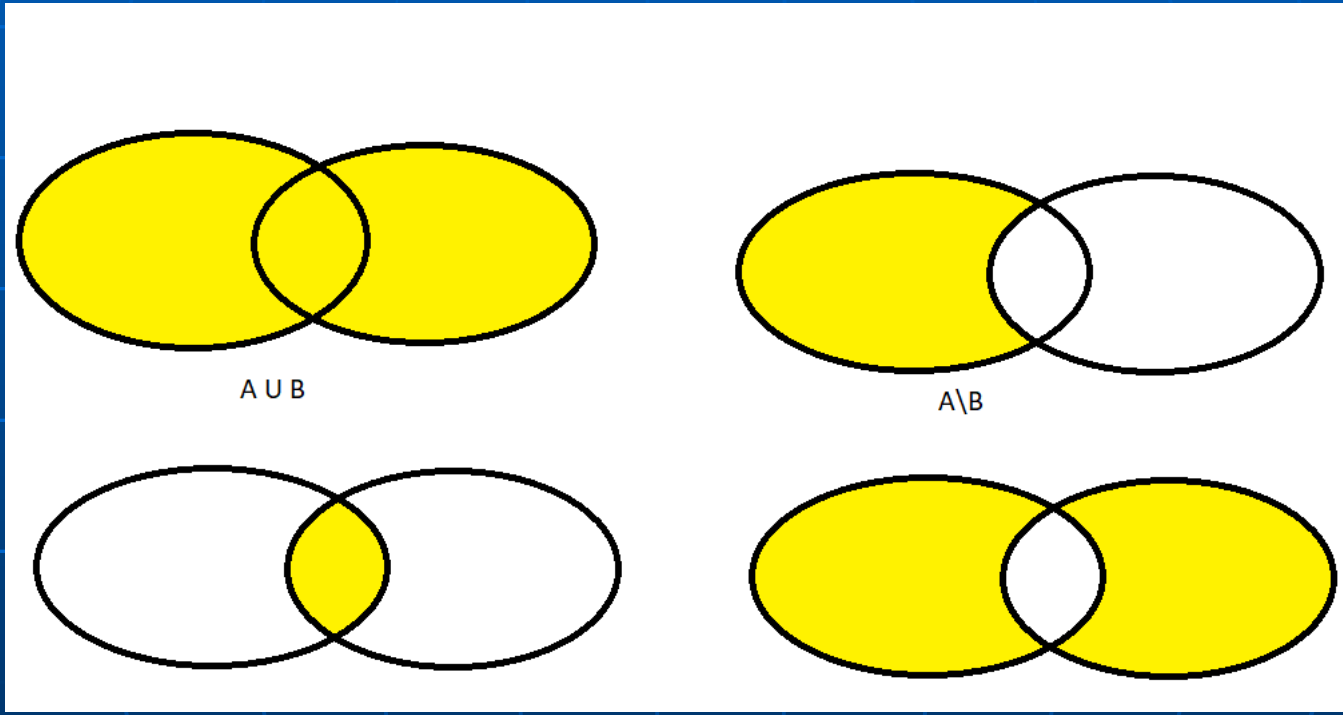
Czego chcielibyśmy od zbiorów?

- Żeby dla każdego elementu móc określić, czy do danego zboru należy, czy nie.
- Żeby można było dla niego określić podzbiory z dowolnych jego elementów.
- Żeby można było na nich rachować – określić sumę, przecięcie, różnicę zbiorów i dopełnienie zbioru.

Notacja

- Piszemy $x \in A$, gdy element x do danego zboru należy (x może być zbiorem!)
- $A \subseteq B$, gdy każdy element zbioru A jest jednocześnie elementem zbioru B, czyli dla każdego x: $x \in A \rightarrow x \in B$
- Ponadto dla wszystkich podzbiorów A,B pewnego zbioru U określamy działania sumy zbiorów: $A \cup B$, przecięcia zbiorów: $A \cap B$, różnicy zbiorów $A \setminus B$ i dopełnień zbiorów A', B'.

Działania na zbiorach



Zbiór pusty

Na ogół przyjmujemy, że zbiór pusty jest jeden w całej matematyce i jest podzbiorem każdego zbioru. Oznaczamy go przez $\varnothing$.

Jeśli $A=\varnothing$, to $P(A)=\{\varnothing\}$

To wcale nie jest oczywiste dla informatyków.

Czy pusta tablica, to to samo co puste drzewo, pusty napis, pusta lista?

Podzbiory

Podzbiory zbioru A oznaczamy przez P(A).

Jeśli $A=\emptyset$, to $P(A)=\{\emptyset\}$

Jeśli $A=\{a\}$, to $P(A)=\{\emptyset,\{a\}\}$

Jeśli $A=\{a,b\}$, to $P(A)=\{\emptyset,\{a\},\{b\},\{a,b\}\}$

Jeśli $A=\{a,b,c\}$, to $P(A)=\{\emptyset,\{a\},\{b\},\{c\},\{a,b\},\{a,c\},\{b,c\},\{a,b,c\}\}$

itd.

Widzimy, że ogólnie jesli zbiór ma n elementów, to liczba jego podzbiorów wynosi 2^n.

Ważne pytanie

Czy zbiór wśród swoich elementów może zawierać siebie samego?

Czyli, czy może się zdarzyć, że $A \in A$?

$$A = \{A, \ldots..\}$$

Ważne pytanie

Czy zbiór wśród swoich elementów może zawierać siebie samego?

Czyli, czy może się zdarzyć, że $A \in A$?

$$A = \{A, \ldots..\}$$

albo na przykład

$$A=\{\{\{...\{\varnothing\}...\}\}\}$$

Przyjmijmy na razie, że takie zbiory istnieją

- Koncepcja zbioru samego w sobie wydaje się nieco nienaturalna.
- Na pewno takie zbiory nie mogą być skończone.
- W przyrodzie i na lekcjach matematyki raczej nie występują
- Nazwijmy je dziwnymi

Zbiory dziwne i normalne

Pozostałe zbiory, czyli takie, że $A \notin A$ nazwijmy zbiorami normalnymi.

Większość znanych nam zbiorów jest normalna.

Utwórzmy zbiór zbiorów normalnych i nazwijmy go N.

Zbiór zbiorów normalnych

- Spróbujmy zastanowić się, jakie są własności zbioru zbiorów normalnych?
- W szczególności spytajmy, czy zbiór zbiorów normalnych jest normalny, czyli

$$\text{czy } N \in N?$$

W czym problem?

Jeżeli $N \in N$, to N jest z definicji dziwny, zatem $N \notin N$.

Przyjęcie założenia, że doprowadziło do sprzeczności, więc prawdziwe jest zdanie przeciwne.

Zatem $N \notin N$, więc N jest dziwny.

Czyżby?

Jeżeli $N \in N$, to N jest z definicji dziwny, zatem $N \notin N$.

Zaś jeśli $N \notin N$, to N jest z definicji normalny, czyli $N \in N$.

Sprzeczność!

Zbiór N nie jest ani normalny, ani dziwny, ale innych zbiorów nie ma, czyli go po prostu nie ma!

Mamy problem

Wygląda na to, że dowodzenie nie wprost nie zawsze działa!

Mamy problem

Aby uniknąć problemu dziwnych zbiorów, matematycy pogodzili się z tym, że

- nie ze wszystkiego można utworzyć zbiór, w szczególności ze zbiorów normalnych
- należy zabronić tego, aby zbiór był elementem siebie samego, czyli zbiory dziwne są zakazane, a zatem
- wszystkie zbiory powinny być normalne.

A może nie ma zbiorów dziwnych?

A może nie ma zbiorów dziwnych?

SĄ !!!

Co to jest katalog w komputerze?

Katalog jest to zbiór składający się z dwóch rodzajów elementów:

- plików
- innych katalogów

Co to jest katalog w komputerze?

Katalog jest to zbiór składający się z dwóch rodzajów elementów:

- plików
- innych katalogów

Czy na pewno innych???

A co by było gdyby...

… gdyby jeden z tych podkatalogów był katalogiem, w którym właśnie się znajduje?

A co by było gdyby...

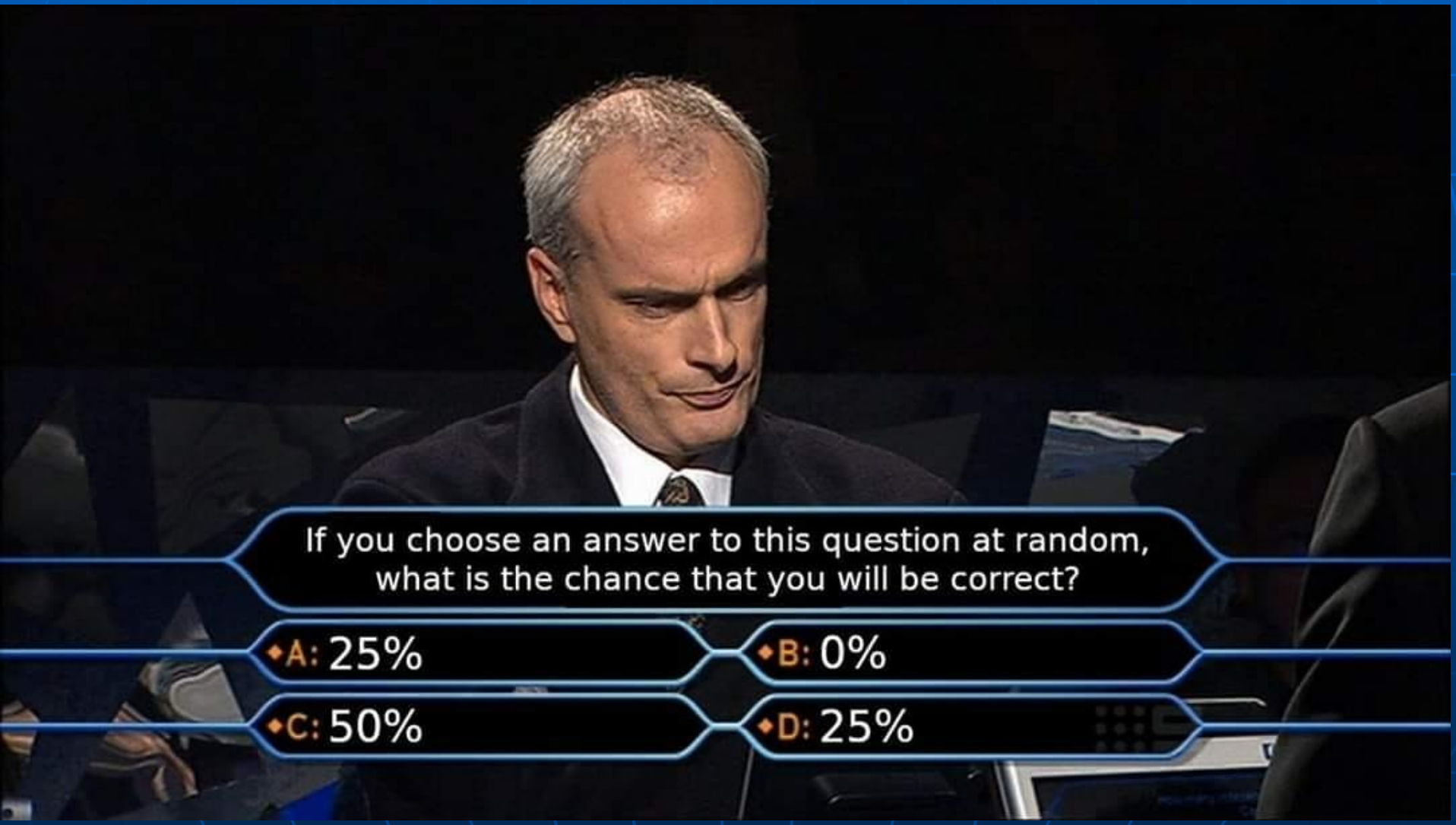
… gdyby jeden z tych podkatalogów był katalogiem, w którym właśnie się znajduje?

Niemożliwe?

Czy możemy poznać istotę zbioru, który jest sam w sobie?

- mamy problem ze zrozumieniem nieskończoności, a taki zbiór wymusza otarcie się o nią
- nie można zamykać oczu myśląc, że katastrofa się nie wydarzy.

Przyczyna paradoksów



Jak reprezentować zbiory w komputerze?

- Podstawowe operacje, których zazwyczaj potrzebujemy:
  - Utwórz zbiór pusty A
  - Sprawdź, czy w zbiorze A jest dany element x
  - Dodaj element x do zbioru A
  - Usuń ze zbioru A element x
- Ponadto często przydatne są niektóre operacje na zbiorach, szczególnie suma i przecięcie.

Reprezentujemy zbiory skończone

- Póki są niewielkie, możemy reprezentować je za pomocą map bitowych. Trzeba wtedy elementy przestrzeni ponumerować od zera do n-1 i zaalokować tablicę odpowiedniej długości.
- Zbiór pusty, to same zera
- Istnienie elementu zaznaczamy „zapalając” odpowiedni bit.

Przykład

- Niech U={0,1,2,...,31}.
- Taki zbiór można reprezentować w jednej zmiennej (unsigned int z).
- Przypisanie z=0; przypisuje zmiennej z reprezentację zbioru pustego
- Teraz wystarczy utworzyć maski (maska na n-ty bit to 1 shl n;) i zastosować odpowiednie operatory, np. w języku C: if (z&(1 shl 4)) ...; sprawdzi, czy element 5 jest w zbiorze reprezentowanym przez z.

Operacje na maskach

- W tym ujęciu mamy bardzo szybkie operacje na bitmaskach: suma zbiorów reprezentowanych przez zmienne v i z, to v|z, iloczyn, to v&z itd.
- Dla większych zbiorów działamy na tablicach grupując elementy w paczki po np. 32 bity.

Nie zawsze jednak możemy tak łatwo ponumerować elementy

- Jeśli tworzymy na przykład graf stanów jakiegoś systemu na żywo, to trudno jest przewidzieć, jak będą wyglądały jego wierzchołki.
- Podstawowym i operacjami będą wtedy:
  - sprawdzenie, czy nowo wygenerowany wierzchołek już jest w naszym grafie
  - dodanie do zbioru wierzchołków nowego
  - czasem usunięcie
- I to wszystko się dzieje dynamicznie.

Potrzeba innych struktur danych

- Pierwszy pomysł: tablica (lista) nieposortowana.
  - Wszystkie operacje są drogie:
    - sprawdzanie przynależności do zbioru
    - dodawanie
    - usuwanie
  - mają złożoność liniową

Potrzeba innych struktur danych

- Drugi pomysł: tablica (lista, choć to nieoczywiste) posortowana.
  - Nie wszystkie operacje są drogie:
    - dodawanie
    - usuwanie
  - mają złożoność liniową, ale
    - sprawdzanie przynależności do zbioru
  - ma złożoność logarytmiczną (binsearch)

Drzewa BST

- Drzewa binarnych wyszukiwań, gdzie w każdym węźle, w którym przechowujemy wartość v, wszystkie wartości lewego syna są mniejsze od v, a wszystkie wartości prawego syna są większe od v.

Miłe własności drzew BST

- Wyszukiwanie wartości, ich wstawianie i usuwanie mają złożoność nie przekraczającą wysokości drzewa
- Losowo wstawionych n wartości tworzy drzewo o oczekiwanej wysokości logarytmicznej

Może warto zrandomizować dane?

Znana metoda, którą stosuje się czasem przed sortowaniem metodą quicksort – randomizacja porządku danych – tu niekoniecznie jest możliwa.

Dane bowiem generowane są często na bieżąco.

Co gorsza: w typowych porządkach, które się w wielu sytuacjach stosuje, element, który wstawia kolejną wartość ma często bardzo podobną wartość, więc ścieżka dostępu jest prawie identyczna.

Jak zmusić drzewo BST do logarytmicznej wysokości?

Dwa rozwiązania:

- Drzewa AVL
- Drzewa lewicowe (splay)

Drzewa AVL

Wymyślone na początku at 70-tych przez Adelsona-Velskiego i Landisa procedury pozwalają na zachowanie zrównoważenia drzewa, a co za tym idzie, wymuszają logarytmiczną wysokość.

Drzewo jest zrównoważone, jeśli w każdym węle wysokości obu synów różnią się nie więcej niż o 1.

Oznacza to, że każda operacja (wyszukiwanie, wstawianie, usuwanie) ma koszt logarytmiczny.

Drzewa splay

- Wymyślone w 1985r przez Daniela Sleatora i Roberta Tarjana
- Bazują na jednej operacji: dla zadanej wartości x w drzewie przebuduj drzewo tak, aby x znalazło się w korzeniu.
- Uzyskujemy ten efekt dokonując odpowiednich rotacji na ścieżce od x do korzenia.

Drzewa splay

Drzewa splay są wyjątkowo dobre gdy zapytania dotyczą ostatnich elementów, które w nich były poszukiwane – są one blisko korzenia.

Własności drzew splay

- Ciekawe: choć wysokość drzewa splay może być liniowa, to zamortyzowany pesymistyczny koszt wykonania n operacji jest O(log n)
- W przypadku gdy poszukujemy elementów popularnych, koszt może zejść poniżej logarytmicznego
- Można łatwo sumować zbiory o tej własności, że elementy pierwszego są mniejsze od elementów drugiego zbioru.

Projekt „Mistrzostwa w Algorytmice i Programowaniu – Uczniowie" jest finansowany ze środków pochodzących z „Programu Rozwoju Talentów Informatycznych na lata 2019-2029"

Dofinansowanie Projektu: 4.887.850,50 zł

Całkowita wartość Projektu: 5.460.850,50 zł