

# Teoretyczne podstawy informatyki



## Wykład 6a: Model danych oparty na zbiorach

<http://hibiscus.if.uj.edu.pl/~erichter/Dydaktyka2010/TPI-2010>

# Model danych oparty na zbiorach

- Zbiór jest najbardziej podstawowym modelem danych w matematyce.
- Wszystkie pojęcia matematyczne, od drzew po liczby rzeczywiste można wyrazić za pomocą specjalnego rodzaju zbioru.
- Jest więc naturalne że jest on również podstawowym modelem danych w informatyce.
- **Dotychczas wykorzystaliśmy to pojęcie mówiąc o**
  - **zdarzeniach w przestrzeni probabilistycznej,**
  - **słowniku,** który także jest rodzajem zbioru na którym możemy wykonywać tylko określone operacje: wstawiania, usuwania i wyszukiwania

## Podstawowe definicje

---

- ❑ W matematyce pojęcie **zbioru** nie jest zdefiniowane wprost.
- ❑ Zamiast tego, podobnie jak punkt czy prosta w geometrii, **zbiór jest zdefiniowany za pomocą swoich własności.**
- ❑ W szczególności istnieje pojęcie **przynależności**, które jest sensowne tylko i wyłącznie dla zbiorów. Jeśli **S** jest **zbiorem** oraz **x** jest **czymkolwiek**, zawsze możemy odpowiedzieć na pytanie  
    **„Czy x należy do zbioru S?”**
- ❑ Zbiór **S** składa się więc z wszystkich takich elementów **x**, dla których **x** należy do zbioru **S**.

## Podstawowe definicje

### □ Notacja:

- Wyrażenie  $\mathbf{x} \in \mathbf{S}$  oznacza, że element  $\mathbf{x}$  należy do zbioru  $\mathbf{S}$ .
- Jeśli elementy  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$  należą do zbioru  $\mathbf{S}$ , i żadne inne, to możemy zapisać:

$$\mathbf{S} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$$

- Każdy  $\mathbf{x}$  musi być inny, nie możemy umieścić w zbiorze żadnego elementu dwa lub więcej razy. Kolejność ułożenia elementów w zbiorze jest jednak całkowicie dowolna.
- Zbiór pusty, oznaczamy symbolem  $\emptyset$ , jest zbiorem do którego nie należą żadne elementy. Oznacza to że  $\mathbf{x} \in \emptyset$  jest zawsze fałszywe.

# Podstawowe definicje

---

## □ Definicja za pomocą abstrakcji:

- Wyliczenie elementów należących do zbioru nie jest jedynym sposobem jego definiowania. Bardzo wygodne jest wyjście od definicji że istnieje zbiór **S** oraz że jego elementy spełniają własność **P**, tzn.  $\{x : x \in S \text{ oraz } P(x)\}$  czyli „zbiór takich elementów **x** należących do zbioru **S**, które spełniają własność **P**.”

## □ Równość zbiorów:

- Dwa zbiory są **równe** (czyli są tym samym zbiorem), jeśli zawierają **te same elementy**.

## □ Zbiory nieskończone:

- Zwykle wygodne jest przyjęcie założenia że zbiory są skończone. Czyli że istnieje pewna skończona liczba **N** taka, że nasz zbiór zawiera dokładnie **N** elementów. Istnieją jednak również zbiory nieskończone np. liczb naturalnych, całkowitych, rzeczywistych, itd.

# Operacje na zbiorach

- Operacje często wykonywane na zbiorach:
  - **Suma:** dwóch zbiorów  $S$  i  $T$ , zapisywana  $S \cup T$ , czyli zbiór zawierający elementy należące do zbioru  $S$  lub do zbioru  $T$ .
  - **Przecięcie (iloczyn):** dwóch zbiorów  $S$  i  $T$ , zapisywana  $S \cap T$ , czyli zbiór zawierający należące elementy do zbioru  $S$  i do zbioru  $T$ .
  - **Różnica:** dwóch zbiorów  $S$  i  $T$ , zapisywana  $S \setminus T$ , czyli zbiór zawierający tylko te elementy należące do zbioru  $S$ , które nie należą do zbioru  $T$ .
- Jeżeli  $S$  i  $T$  są zdarzeniami w przestrzeni probabilistycznej,
  - suma, przecięcie i różnica mają naturalne znaczenie,
  - $S \cup T$  jest zdarzeniem polegającym na zajściu zdarzenia  $S$  lub  $T$ ,
  - $S \cap T$  jest zdarzeniem polegającym na zajściu zdarzenia  $S$  i  $T$ ,
  - $S \setminus T$  jest zdarzeniem polegającym na zajściu zdarzenia  $S$  ale nie  $T$ ,
  - Jeśli  $S$  jest zbiorem obejmującym całą przestrzeń probabilistyczna,  $S \setminus T$  jest dopełnieniem zbioru  $T$ .

## Prawa algebraiczne dla sumy, przecięcia i różnicy zbiorów

Prawo przemienności dla sumy:  $(S \cup T) = (T \cup S)$

Prawo łączności dla sumy:  $(S \cup (T \cup R)) = ((S \cup T) \cup R)$

- Prawo przemienności i łączności dla sumy zbiorów określają, że możemy obliczyć sumę wielu zbiorów, wybierając je w dowolnej kolejności.
- Wynikiem zawsze będzie taki sam zbiór elementów, czyli takich które należą do jednego lub więcej zbiorów będących operandami sumy.

Prawo przemienności dla przecięcia:  $(S \cap T) = (T \cap S)$

Prawo łączności dla przecięcia:  $(S \cap (T \cap R)) = ((S \cap T) \cap R)$

- Przecięcie dowolnej liczby zbiorów nie zależy od kolejności ich grupowania

Prawo rozdzielności przecięcia względem sumy:

$$(S \cap (T \cup R)) = ((S \cap T) \cup (S \cap R))$$

Prawo rozdzielności sumy względem przecięcia:

$$(S \cup (T \cap R)) = ((S \cup T) \cap (S \cup R))$$

## Prawa algebraiczne dla sumy, przecięcia i różnicy zbiorów

Prawo łączności dla sumy i różnicy:  $(S \setminus (T \cup R)) = ((S \setminus T) \cup (S \setminus R))$   
Prawo rozdzielności różnicy względem sumy:  $((S \cup T) \setminus R) = ((S \setminus R) \cup (T \setminus R))$

- Zbiór pusty jest elementem neutralnym sumy:  $(S \cup \emptyset) \equiv S$
- Idempotencja sumy:  $(S \cup S) = S$
- Idempotencja przecięcia:  $(S \cap S) = S$
- $(S \setminus S) \equiv \emptyset$
- $(\emptyset \setminus S) \equiv \emptyset$
- $(\emptyset \cap S) \equiv \emptyset$

### □ Relacja podzbioru:

- Istnieje relacja zawierania się jednego zbioru w drugim zbiorze, co oznacza że wszystkie elementy pierwszego są również elementami drugiego.

### □ Zbiór potęgowy:

- $P(S)$  zbioru  $S$  to zbiór wszystkich podzbiorów zbioru  $S$ .
- Jeśli  $S = \{1,2,3\}$  to:
- $P(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$ .



## Zbiory a listy

---

- Istotne różnice między pojęciem zbiorów  $S = \{x_1, x_2, \dots, x_n\}$  a listą  $L = \{x_1, x_2, \dots, x_n\}$ :
  - **Kolejność elementów** w zbiorze jest nieistotna (a dla listy jest istotna).
  - Elementy należące do listy mogą się **powtarzać** (a dla zbioru nie mogą).

## Obiekty niepodzielne

---

- ❑ To nie jest pojęcie z teorii zbiorów ale bardzo wygodne dla dyskusji o strukturach danych i algorytmach opartych na zbiorach.
- ❑ Zakładamy istnienie pewnych **obiektów niepodzielnych**, które nie są zbiorami. Obiektem niepodzielnym może być element zbioru, jednak nic nie może należeć do samego obiektu niepodzielnego. Podobnie jak zbiór pusty, obiekt niepodzielny nie może zawierać żadnych elementów. Zbiór pusty jest jednak zbiorem, obiekt niepodzielny nim nie jest.
- ❑ Kiedy mówimy o strukturach danych, często wygodne jest wykorzystanie skomplikowanych typów danych jako typów obiektów niepodzielnych. Obiektami niepodzielnymi mogą więc być struktury lub tablice, które przecież wcale nie mają „niepodzielnego” charakteru.

Abstrakcyjny typ danych = zbiór  
Abstrakcyjna implementacja = lista  
Implementująca struktura danych = lista jednokierunkowa

## □ Suma, przecięcie i różnica:

- Podstawowe operacje na zbiorach, np. suma, mogą wykorzystywać jako przetwarzaną strukturę danych listę jednokierunkową, chociaż właściwa technika przetwarzania zbiorów powinna się nieco różnić od stosowanej przez nas do słownika. W szczególności posortowanie elementów wykorzystywanych list znacząco skraca czas wykonywania operacji sumy, przecięcia i różnicy zbiorów. (Takie działanie niewiele zmienia czas wykonywania operacji na słowniku).

## Zbiory a listy

---

### □ Zbiory jako nieposortowane listy:

- Wyznaczenie sumy, przecięcia czy różnicy **zbiorów** o **rozmiarach** **m** i **n** wymaga czasu  $O(mn)$ .
- Aby stworzyć listę **U** reprezentująca np. sumę pary **S** i **T**, musimy rozpocząć od skopiowania listy reprezentującej zbiór **S** do początkowo pustej listy **U**. Następnie każdy element listy ze zbioru **T** musimy sprawdzić aby przekonać się, czy nie znajduje się on na liście **U**. Jeśli nie to dodajemy ten element do listy **U**.

# Zbiory a listy

---

## □ Zbiory jako posortowane listy

- Operacje wykonujemy znacznie szybciej jeżeli elementy są posortowane. Za każdym razem porównujemy ze sobą tylko dwa elementy (po jednym z każdej listy).
- Wyznaczenie sumy, przecięcia czy różnicy **zbiorów o rozmiarach  $m$  i  $n$**  wymaga czasu  **$O(m+n)$** .
- Jeżeli listy nie były pierwotnie posortowane to sortowanie list zajmuje  **$O(m \log m + n \log n)$** .
- Operacja ta może nie być szybsza niż  **$O(m n)$**  jeśli ilość elementów list jest bardzo różna.
- Jeżeli liczby  $m$  i  $n$  są porównywalne to  **$O(m \log m + n \log n) < O(m n)$**

## Implementacja zbiorów oparta na wektorze własnym

---

- Definiujemy **uniwersalny zbiór  $U$**  w którym zawierają się wszystkie zbiory na których będziemy przeprowadzać operacje. Np. talia kart (zbiór 52 kart) jest uniwersalny dla różnych możliwych zbiorów kart.
- **Porządkujemy elementy zbioru  $U$**  w taki sposób, by każdy element tego zbioru można było związać z **unikatową „pozycją”**, będącą liczbą całkowitą od **0** do  **$n-1$**  (gdzie  **$n$**  jest liczba elementów w zbiorze uniwersalnym). Liczba elementów w zbiorze  **$S$**  jest  **$m$** .
- Wówczas, **zbiór  $S$**  zawierający się w zbiorze  **$U$** , możemy **reprezentować za pomocą wektora własnego** złożonego z zer i jedynek – dla każdego elementu  **$x$**  należącego do zbioru  **$U$** , jeśli  **$x$**  należy także do zbioru  **$S$** , odpowiadająca temu elementowi pozycja zawiera wartość **1**; jeśli  **$x$**  nie należy do  **$S$** , na odpowiedniej pozycji mamy wartość **0**.

## Implementacja zbiorów oparta na wektorze własnym

---

- Czas potrzebny na wykonanie operacji sumy, przecięcia i różnicy jest  $O(n)$ .
- Jeśli przetwarzane zbiory są dużą częścią zbioru uniwersalnego to jest to dużo lepsze niż  $O(m \log m)$  (posortowanie listy) lub  $O(m^2)$  (nieposortowane listy).
- Jeśli  $m \ll n$  to jest to oczywiście nieefektywne.
- Ta implementacja również niepraktyczna jeżeli wymaga zbyt dużego  $U$ .

## Przykład z kartami

---

### □ **Przykład:**

- Niech  $U$  będzie talią kart.
- Umawiamy się że porządkujemy karty w talii w następujący sposób:
  - Kolorami: trefl, karo, kier, pik.
  - W każdym kolorze wg schematu: as, 2, 3, ..., walet, dama, król.
- Przykładowo pozycja:
  - as trefl to 0,
  - król trefl to 12,
  - as karo to 13,
  - walet pik to 49.



## Przykład z kartami

- Zbiór wszystkich kart koloru trefl  
11111111111111000
- Zbiór wszystkich figur  
0000000000111000000000011100000000001110000000000111
- Poker w kolorze kier (as, walet, dama, król)  
000000000000000000000000000000001000000000111000000000000
- Każdy element zbioru kart jest związany z unikatową pozycją.

## Przykład z jabłkami

	Odmiana	Kolor	Dojrzewa
0	Delicious	czerwony	późno
1	Granny Smith	zielony	wcześnie
2	Jonathan	czerwony	wcześnie
3	McIntosh	czerwony	wcześnie
4	Gravenstein	czerwony	późno
5	Pippin	zielony	późno

Czerwone = 101110

Wcześnie = 011100

Czerwone  $\cup$  Wcześnie = 111110

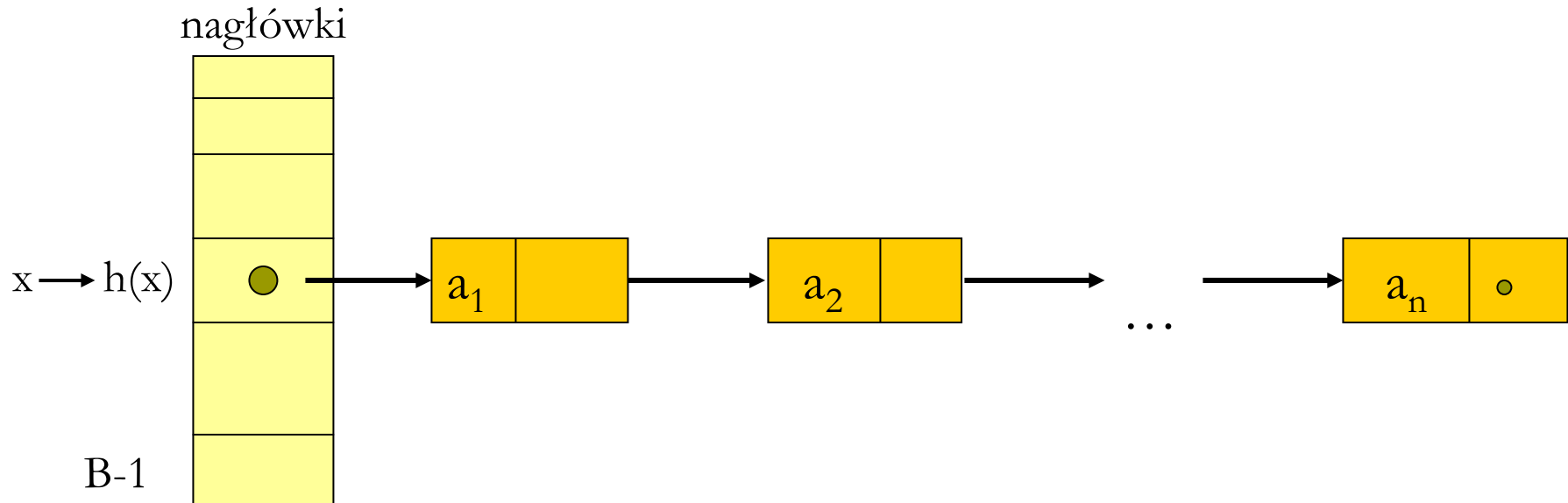
Czerwone  $\cap$  Wcześnie = 001100

Czas potrzebny do wyznaczenia sumy, przecięcia, różnicy jest proporcjonalny do długości wektora własnego.

## Struktura danych tablicy mieszającej

- Reprezentacja słownika oparta o wektor własny, jeśli tylko możliwa, umożliwiłaby bezpośredni dostęp do miejsca w którym element jest reprezentowany.
  - Nie możemy jednak wykorzystywać zbyt dużych zbiorów uniwersalnych ze względu na pamięć i czas inicjalizacji.
  - Np. słownik dla słów złożonych z co najwyżej 10 liter. Ile możliwych kombinacji:  $26^{10} + 26^9 + \dots + 26 = 10^{14}$  możliwych słów.  
Faktyczny słownik: to tylko około  $10^6$ .
- Co robimy?**
- Grupujemy, każda grupa to jedna komórka z „nagłówkiem” + lista jednokierunkowa z elementami należącymi do grupy.
  - Taka strukturę nazywamy tablicą mieszającą (ang. hash table)

# Struktura danych tablicy mieszającej



- ❑ Istnieje **funkcja mieszająca** (ang. **hash function**), która jako argument pobiera element  $x$  i zwraca liczbę całkowitą z przedziału **0 do B-1**, gdzie **B** jest liczbą komórek w tablicy mieszającej.
- ❑ Wartością zwracaną przez  **$h(x)$**  jest komórka, w której umieszczamy element  $x$ .
- ❑ Ważne aby funkcja  **$h(x)$**  „mieszała”, tzn. aby komórki zawierały tę samą przybliżoną liczbę elementów.

## Struktura danych tablicy mieszającej

- Każda komórka składa się z listy jednokierunkowej, w której przechowujemy wszystkie elementy zbioru wysłanego do tej komórki przez funkcje mieszającą.
- Aby odnaleźć element  $x$  obliczamy wartość  $h(x)$ , która wskazuje na numer komórki.
- Jeśli tablica mieszająca zawiera element  $x$ , to możemy go znaleźć przeszukując listę która znajduje się w tej komórce.
- Tablica mieszająca pozwala na wykorzystanie reprezentacji zbiorów opartej na liście (wolne przeszukiwanie), ale dzięki podzieleniu zbioru na  $B$  komórek, czas przeszukiwania jest  $\sim 1/B$  potrzebnego do przeszukiwania całego zbioru.
- W szczególności może być nawet  $O(1)$ , czyli taki jak w reprezentacji zbioru opartej na wektorze własnym.

## Implementacja słownika oparta na tablicy mieszającej

---

- Aby wstawić, usunąć lub wyszukać element  $\mathbf{x}$  w słowniku zaimplementowanym przy użyciu tablicy mieszającej, musimy zrealizować proces złożony z trzech kroków.
  - wyznaczyć właściwą komórkę przy użyciu funkcji  $\mathbf{h}(\mathbf{x})$
  - wykorzystać tablice wskaźników do nagłówek w celu znalezienia listy elementów znajdującej się w komórce wskazanej przez  $\mathbf{h}(\mathbf{x})$
  - wykonać na tej liście operacje tak jakby reprezentowała cały zbiór

- **Listy jednokierunkowe, wektory własne** oraz **tablice mieszające** to trzy najprostsze sposoby reprezentowania zbiorów w języku programowania.
- **Listy jednokierunkowe** oferują największą elastyczność w przypadku większości operacji na zbiorach, nie zawsze są jednak rozwiązaniem najbardziej efektywnym.
- **Wektory własne** są najszybszym rozwiązaniem dla pewnych operacji, mogą jednak być wykorzystywane tylko w sytuacjach, gdy zbiór uniwersalny jest mały.
- Często złotym środkiem są **tablice mieszające**, które zapewniają zarówno oszczędne wykorzystanie pamięci, jak i satysfakcjonujący czas wykonania operacji.

# Podsumowanie

---

- Pojęcie zbioru ma zasadnicze znaczenie w informatyce.
- Najczęściej wykonywanymi operacjami na zbiorach są: suma, przecięcie oraz różnica.
- Do modyfikowania i upraszczania wyrażeń złożonych ze zbiorów i zdefiniowanych na nich operacji możemy wykorzystywać prawa algebraiczne.
- Trzy najprostsze sposoby implementacji struktury danych dla zbiorów.
  - Listy jednokierunkowe oferują największą elastyczność w przypadku większości operacji na zbiorach, nie zawsze są najbardziej efektywne.
  - Wektory własne są najszybszym rozwiązaniem dla pewnych operacji, mogą jednak być wykorzystane tylko w sytuacjach, gdy zbiór uniwersalny jest mały.
  - Tablice mieszające są często złotym środkiem, który zapewnia oszczędne wykorzystanie pamięci i satysfakcjonujący czas wykonania.