Teoretyczne podstawy informatyki

Wykład 7:
Model danych oparty na zbiorach

http://hibiscus.if.uj.edu.pl/~erichter/Dydaktyka2009/TPI-2009

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 wiec naturalne że jest on również podstawowym modelem danych w informatyce.
- **Dotychczas wykorzystaliśmy to pojecie 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 pojecie **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 $x \in S$ oznacza, że element **x** należy do zbioru **S**.
- Jeśli elementy $x_1, x_2, \ldots, x_n$ należą do zbioru **S**, i żadne inne, to możemy zapisać:
  $S = \{x_1, x_2, \ldots, x_n\}$
- Każdy **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 $\varnothing$, jest zbiorem do którego nie należą żadne elementy. Oznacza to że $x \in \varnothing$ 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 ∈ S 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 ∪ T**, czyli zbiór zawierający elementy należące do zbioru **S** lub do zbioru **T**.
  - **Przecięcie (iloczyn):** dwóch zbiorów **S** i **T**, zapisywana **S ∩ T**, czyli zbiór zawierający należące elementy do zbioru **S** i do zbioru **T**.
  - **Różnica:** dwóch zbiorów **S** i **T**, zapisywana **S \ 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 ∪ T** jest zdarzeniem polegającym na zajściu zdarzenia **S** lub **T**,
  - **S ∩ T** jest zdarzeniem polegającym na zajściu zdarzenia **S** i **T**,
  - **S \ T** jest zdarzeniem polegającym na zajściu zdarzenia **S** ale nie **T**,
  - Jeśli **S** jest zbiorem obejmującym całą przestrzeń probabilistyczna, **S \ 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: $(S \cup \varnothing) \equiv S$
- Idempotencja sumy: $(S \cup S) = S$
- Idempotencja przecięcia: $(S \cap S) = S$
- $(S \setminus S) \equiv \varnothing$
- $(\varnothing \setminus S) \equiv \varnothing$
- $(\varnothing \cap S) \equiv \varnothing$

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:
- $P(S) = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$.

Zbiory a listy

- Istotne różnice miedzy pojęciem zbiór $S = \{x_1, x_2, \ldots, x_n\}$ a listą $L = \{x_1, x_2, \ldots, x_n\}$:
  - **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 wiec 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*(m n).**
  - 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 << 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

  1111111111110000000000000000000000000000000000

- Zbiór wszystkich figur

  000000000011100000000011100000000011100000000111

- Poker w kolorze kier (as, walet, dama, król)

  000000000000000000000000100000000011100000000000

- 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

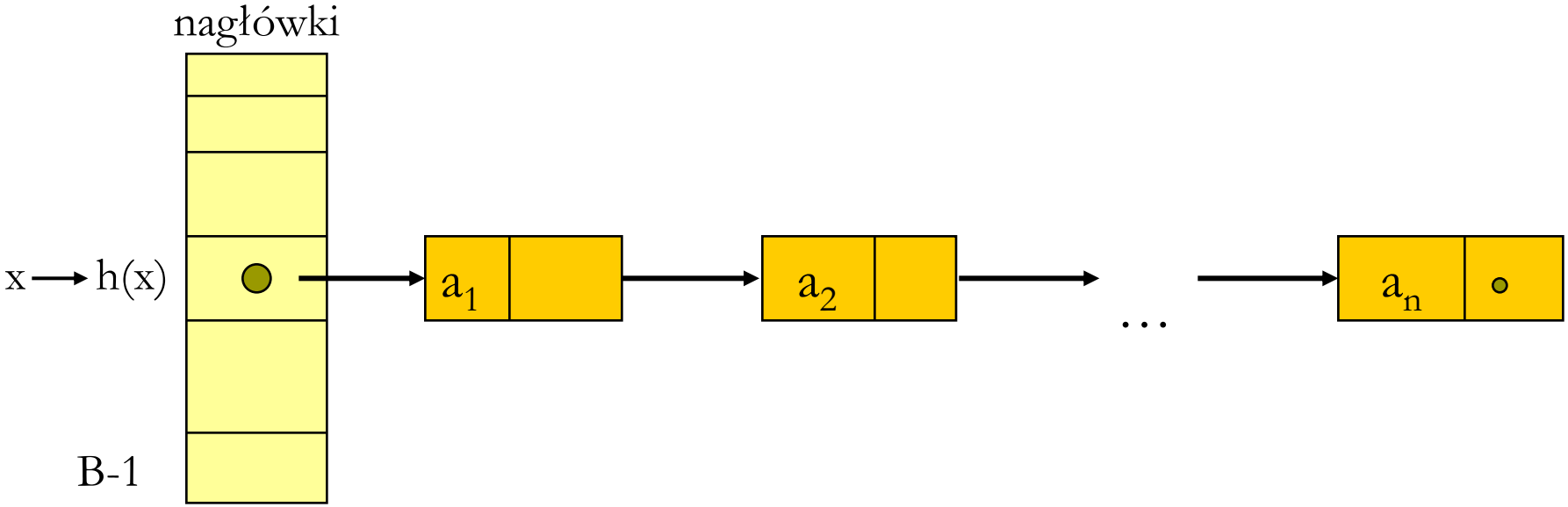
Czerwone = 101110
Wcześnie = 011100
Czerwone ∪ Wcześnie = 111110
Czerwone ∩ Wcześnie = 001100

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: $26^{10}+26^{9}+ \ldots + 26 = 10^{14}$ 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 mieszająca (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 **~ 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 **x** w słowniku zaimplementowanym przy użyciu tablicy mieszającej, musimy zrealizować proces złożony z trzech kroków.
  - wyznaczyć właściwą komórkę przy użyciu funkcji **h(x)**
  - wykorzystać tablice wskaźników do nagłówków w celu znalezienia listy elementów znajdującej się w komórce wskazanej przez **h(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.

Relacje i funkcje

- Chociaż założyliśmy, że w ogólności elementy należące do zbiorów są niepodzielne, w praktyce często korzystnym rozwiązaniem jest przypisanie elementom pewnych struktur.
- Ważną strukturą dla elementów jest **lista o stałej długości zwana krotką.**
- Każdy element takiej listy nazywamy **składową krotki**.
- Zbiór elementów, z których każdy jest krotką o takiej samej liczności – powiedzmy **k** - nazywamy **relacją**. Licznością takiej relacji jest **k**.
- **Jeśli liczność wynosi 2 mówimy o krotce lub relacji binarnej.**
- **Iloczyn kartezjański A x B:**
  - Jest to zbiór par, z których pierwszy element pochodzi ze zbioru **A**, drugi ze zbioru **B**, czyli
    A x B = { (a, b) : a ∈ A oraz b ∈ B}
  - Iloczyn kartezjański nie ma własności przemienności, **A x B ≠ B x A**
  - K-elementowy iloczyn kartezjański A_1 x A_2 x A_3 … x A_k to zbiór k-krotek $(a_1,a_2, \ldots, a_k)$.

Funkcje

- **Relacje binarne:**
  - **Relacja binarna jest zbiorem par**, będącym podzbiorem iloczynu kartezjańskiego dwóch zbiorów **A i B**.
    Jeśli relacja **R** jest podzbiorem **A x B**, mówimy, ze **R jest relacja z A do B.**
  - O zbiorze **A** mówimy że jest dziedziną, zaś o zbiorze **B** że jest przeciwdziedziną.
- **Funkcje:**
  - Mówimy, że relacja **R** jest **funkcja częściową z dziedziny A do przeciwdziedziny B** jeśli dla dowolnego elementu **a** należącego do zbioru **A istnieje co najwyżej** jeden taki element **b** należący do zbioru **B**, ze spełniona jest relacja **aRb**.
  - O relacji **R** mówimy ze jest **funkcją zupełną** jeśli dla każdego elementu **a** należącego do zbioru **A** istnieje **dokładnie jeden** element **b** należący do zbioru **B**, że spełniona jest relacja **aRb**.
- **Różnica** miedzy funkcją częściowa i funkcją zupełną jest taka że ta pierwsza może nie być zdefiniowana dla niektórych elementów swojej dziedziny.

Odpowiedniości wzajemnie jednoznaczne

- Niech **F** będzie funkcją częściową z dziedziny **A** do przeciwdziedziny **B** charakteryzującą się następującymi własnościami:
  1. Dla każdego elementu **a** należącego do zbioru **A** istnieje taki element **b** należący do zbioru **B**, że **F(a) = b**
  2. Dla każdego elementu **b** należącego do zbioru **B** istnieje pewien element **a** należący do zbioru **A**, że **F(a) = b**
  3. Dla każdego elementu **b** należącego do zbioru **B** nie istnieją takie różne elementy a_1, a_2, należące do zbioru **A**, dla których zarówno $F(a_1)$, jak i $F(a_2)$ są równe **b**.
- W takim przypadku o funkcji **F** mówimy że jest **odpowiedniością wzajemnie jednoznaczną z A do B** (**bijekcja**).
- Warunek 1) oznacza że funkcja **F** jest zupełna z **A do B**.
- Warunek 2) i 3) oznacza że funkcja **F** jest zupełna z **B do A**.

Przykład

- Funkcja **$S = \{(a, b): b=a^2; a \in Z, b \in Z\}$ NIE JEST** wzajemnie jednoznaczna.
- Funkcja spełnia własność 1) ponieważ dla każdej liczby całkowitej **$a=i$** istnieje liczba całkowita **$b=i^2$** taka że **$b=S(a)$**.
- Funkcja **nie** spełnia własności 2) ponieważ istnieją wartości **b** (wszystkie liczby ujemne) które, dla żadnego **a**, nie należą do **S(a).**
- Funkcja **nie** spełnia własności 3), ponieważ istnieje wiele przykładów par różnych **a**, dla których **S(a)** jest równe takiej samej wartości **b**.
  $S(a_1 = 3) = 9$; $S(a_2 = -3) = 9$.

Przykład

- Funkcja **P = {(a, b): b=a+1; a ∈ Z, b ∈ Z}** **JEST** wzajemnie jednoznaczna.
- Funkcja spełnia własność 1) ponieważ dla każdej liczby całkowitej **a** istnieje liczba całkowita **b = a + 1** taka że **b = P(a)**.
- Funkcja spełnia własność 2) ponieważ dla każdej wartości **b** istnieje pewna liczba całkowita **a = b - 1** dla której **b = P(a)**.
- Funkcja spełnia własność 3), ponieważ dla żadnej liczby całkowitej **b** nie mogą istnieć dwie różne liczby całkowite, które po dodaniu do nich **1** są równe **b**.

Implementowanie funkcji w formie danych

- W językach programowania funkcje są zazwyczaj implementowane w formie kodu. Jeśli jednak ich dziedzina jest odpowiednio mała, mogą być także implementowane za pomocą technik całkiem podobnych do wykorzystywanych dla zbiorów.
- **Funkcja jako kod**:
  Określa sposób wyznaczania wartości **F(x)** dla dowolnej wartości **x** należącej do dziedziny.

Implementowanie funkcji w formie danych

- **Funkcja jako dane:**
  - Reprezentacja składa się ze **skończonego zbioru par**. Te pary **(a_1,b_1), (a_2,b_2), …, (a_n,b_n)** przechowujemy w określonej strukturze np. listy jednokierunkowe, wektory własne, tablice mieszające.
- **Operacje na funkcjach (reprezentowanych jako dane):**
- Operacje które chcemy wykonywać są bardzo podobne do tych wykonywanych na słownikach:
  - wstawić nowa parę **(a, b)**, taka że **b=F(a)** (sprawdzamy czy nie ma **(a,c)**)
  - usunąć parę powiązana z wartością **a**, czyli **(a, b=F(a))**
  - wyszukać wartość **F(a)** związaną z wartością **a**, czyli wskazać na parę **(a, b=F(a))**

Efektywność operacji na funkcjach (reprez. jako dane)

- Czas potrzebny na wykonanie operacji na funkcjach reprezentowanych za pomocą struktur:

  listy jednokierunkowej, wektora własnego, tablicy mieszającej

  są takie same jak czasy adekwatnych operacji wykonywanych na **słownikach** reprezentowanych za pomocą tych samych struktur.
- Jeśli funkcja składa się z **n** par, operacja na **liście jednokierunkowej** reprezentującej tą funkcje wymaga średnio czasu ***O*(n)**.

Efektywność operacji na funkcjach (reprez. jako dane)

- Reprezentacja oparta na **wektorze własnym** wymaga średnio czasu ***O*(1)** dla każdej operacji (ale można ją stosować tylko jeśli zbiór uniwersalny czyli dziedzina ma ograniczony rozmiar).
- Operacje na funkcji reprezentowanej za pomocą **tablicy mieszającej** złożonej z **B** komórek wymagają średnio czasu ***O*(n/B)**.
  Jeśli istnieje możliwość zbliżenia wartości **B** do wartości **n**, to średni czas wykonywania operacji maleje nawet do ***O*(1)**.

Implementowanie relacji binarnych

- Implementowanie relacji binarnych różni się w pewnych elementach od implementacji funkcji.
- Zarówno relacje binarne jak i funkcje są zbiorami par, jednak funkcja dla każdego należącego do dziedziny elementu a zawiera co najwyżej jedną parę w postaci **(a,b)** dla dowolnego **b**.
- Relacja może zawierać dowolną liczbę należących do przeciwdziedziny elementów powiązanych z danym elementem a należącym do dziedziny relacji.

Implementowanie relacji binarnych

- **Operacje na relacjach binarnych:**
  - **wstawianie:**
    wstawiamy parę **(a,b)** i nie musimy sprawdzać, czy w relacji **R** istnieje już para **(a,c)** dla pewnej wartości **b≠c,**
  - **usuwanie:**
    podobnie jak dla zbioru,
  - **wyszukiwanie:**
    pobiera należący do dziedziny element a i zwraca wszystkie należące do przeciwdziedziny elementy **b**, takie ze **(a,b)** znajduje się w relacji binarnej.
- Powyższa interpretacja operacji wyszukiwania daje nowy abstrakcyjny typ danych różniący się nieco od słownika. Implementacje są oparte o strukturę listy jednokierunkowej, wektora lub tablicy mieszającej.

Operacje słownikowe na funkcjach i relacjach

- Zbiór par może być traktowany jako zbiór, jako funkcja lub jako relacja.
- Dla każdego z tych przypadków zdefiniowaliśmy właściwe operacje wstawiania, usuwania i wyszukiwania elementów.
- Operacje te różnią się w swojej formie.

	zbiór par	**funkcja**	**relacja**
wstawianie	dziedzina i przeciwdziedzina	dziedzina i przeciwdziedzina	dziedzina i przeciwdziedzina
usuwanie	dziedzina i przeciwdziedzina	dziedzina	dziedzina i przeciwdziedzina
wyszukiwanie	dziedzina i przeciwdziedzina	dziedzina	dziedzina i przeciwdziedzina

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 największa 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.

Podsumowanie

- Relacje (binarne) są zbiorami par. Funkcja jest relacją, w której istnieje co najwyżej jedna krotka dla danej wartości pierwszej składowej
- Bijekcja pomiędzy dwoma zbiorami jest funkcją, która z każdą wartością pierwsze składowej łączy unikatowy element drugiej składowej i odwrotnie.
- Istnieje wiele istotnych własności relacji binarnych – do najważniejszych należą zwrotność, przeciwzwrotność, symetrai, asymetria, antysymetria, spójność oraz przechodniość.
- Specyficzne rodzaje relacji binarnych to:
  - relacja porządku częściowego,
  - relacja porządku całkowitego,
  - relacja równoważności.