ALGORYTMY I STRUKTURY DANYCH

Temat 2: **Typy i struktury danych. Modele danych oparte na drzewach.**

Wykładowca: dr inż. Zbigniew TARAPATA

e-mail: Zbigniew.Tarapata@isi.wat.edu.pl
http://www.tarapata.strefa.pl/p_algorytmy_i_struktury_danych/

Treść wykładu

- Typy proste
- Typy złożone
- Definicja struktury danych
- Liniowe struktury danych
- Tablice z haszowaniem
- Struktury drzewiaste
- Realizacje struktur

Proste typy danych

Type $T=(c_1,c_2,\ldots,c_n)$

Moc typu T – wartość funkcji $moc(T)=n$

Type *kształt*= (*prostokąt, kwadrat,elipsa,okrąg*)

Integer

Boolean

Char

Real

Definicja struktury danych

Definicja 2.1:

Strukturą danych nazywamy trójkę uporządkowaną
S=(D, R, e),

gdzie:

- ***D*** – oznacza zbiór danych elementarnych $\boldsymbol{d_{i,\ i=1,..,|D|}}$ (i – jest indeksem poszczególnych danych),
- ***R={r_{we}, N}*** – jest zbiorem dwóch relacji określonych na strukturze danych:
  - $r_{we} \subset \{e\} \times D$ – relacja wejścia do struktury danych S,
  - $N \subset D \times D$ – relacja następstwa (porządkująca) strukturę S,
- *e* – jest elementem wejściowym do struktury danych S (nie jest to dana wchodząca w skład struktury danych S).

Zbiór danych elementarnych *D*

- Jak widać z definicji struktury danych, zbiór danych elementarnych jest skończony i dla wygody operowania oznaczeniem jego elementów poszczególne dane są indeksowane od wartości 1 w kierunku wartości większych.
- Weźmy zatem dla przykładu pięcioelementową strukturę danych *S*. Zapis zbioru jej danych elementarnych może wyglądać następująco:

$$D=\{d_1, d_2, d_3, d_4, d_5\}$$

- Liczność zbioru danych elementarnych wynosi 5, co zapisujemy:
  - **|*D*|=5**

Dana elementarna, zmienna programowa

- *Dana elementarna* d_i jest pojęciem abstrakcyjnym, rozumianym jako tzw. „nazwana wartość”. Jest ona określana jako uporządkowana dwójka elementów:

$$d_i = \langle n_i, w_i \rangle$$

- , gdzie:
  - n_i – nazwa danej,
  - w_i – aktualna wartość danej z określonej dziedziny wartości.
- *Zmienna programowa* jest pojęciem związanym z realizacją danej w konkretnym środowisku programistycznym. Posiada ona zdefiniowaną etykietę (nazwę zmiennej), wraz z określeniem dziedziny wartości, którą może przyjmować dana reprezentowana przez zmienną, a także zdefiniowaną dla tej dziedziny wartości dziedziną algorytmiczną.

Relacja r_{we} – wskazanie korzenia struktury S

- Relacja r_{we}, jest opisywana poprzez jedno- lub wieloelementowy zbiór dwójek uporządkowanych elementów, z których pierwszym jest zawsze element wejściowy **e**, natomiast drugim elementem jest jedna z danych elementarnych ze zbioru **D**.
- W sytuacji opisanej powyżej mówimy, że
  - „element **e** należy do dziedziny relacji r_{we}" - $Dz(r_{we})$,
  - „dana d_i należy do przeciwdziedziny r_{we}" – $PDz(r_{we})$
- Element (elementy) należące do $PDz(r_{we})$ są nazywane *korzeniem (korzeniami) struktury danych S*. Struktura musi mieć zdefiniowany co najmniej jeden korzeń.

Przykład: Załóżmy, że struktura S posiada dwa korzenie, według opisu:

$$r_{we} = \{\langle e, d_2 \rangle, \langle e, d_5 \rangle\}$$

Relacja N – ustalenie porządku struktury S

- Relacja następstwa N opisuje wzajemne uporządkowanie elementów w strukturze danych S. Porządek struktury opisujemy w postaci zestawów dwójek uporządkowanych elementów.

Przykład: Opiszmy porządek naszej pięcioelementowej struktury danych S:

$$N = \{\langle d_2, d_1 \rangle, \langle d_1, d_3 \rangle, \langle d_1, d_4 \rangle, \langle d_3, d_4 \rangle, \langle d_5, d_3 \rangle\}$$

UWAGA: Kor[illegible]ktury S nie m[illegible]elementami należącymi do

poprzednik

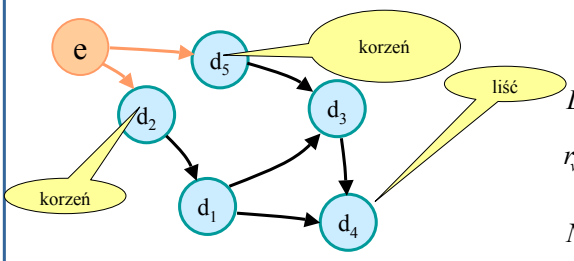
następnik

$PDz(N)$

Model grafowy struktury danych

- Aby łatwiej zobrazować strukturę danych i w ten sposób lepiej ją zrozumieć, można zbudować dla niej *model grafowy*. W modelu tym:
  - węzły (kółka) oznaczają poszczególne dane elementarne,
  - łuki (strzałki) służą do odwzorowania następstw poszczególnych danych elementarnych w strukturze S.

Przykład: Model grafowy dla opisywanej do tej pory struktury S:



$$D=\{d_1,d_2,d_3,d_4,d_5\}$$

$$r_{we}=\{\langle e,d_2\rangle,\langle e,d_5\rangle\}$$

$$N=\{\langle d_2,d_1\rangle,\langle d_1,d_3\rangle,\langle d_1,d_4\rangle,\langle d_3,d_4\rangle,\langle d_5,d_3\rangle\}$$

Definicja liniowej struktury danych

Definicja 2.2:

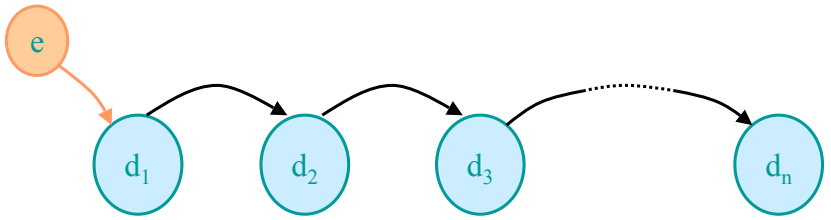
Liniową strukturą danych nazywamy strukturę danych ***S=(D, R, e)***, w której relacja porządkująca ***N*** opisuje powiązania pomiędzy elementami odpowiednio dla poszczególnych rodzajów list.

Wyróżniamy cztery rodzaje list (jednopoziomowych):

- jednokierunkowe listy niecykliczne
- dwukierunkowe listy niecykliczne
- jednokierunkowe listy cykliczne (pierścienie jednokierunkowe)
- dwukierunkowe listy cykliczne (pierścienie dwukierunkowe)

Jednokierunkowe listy niecykliczne

- Model grafowy listy jednokierunkowej:

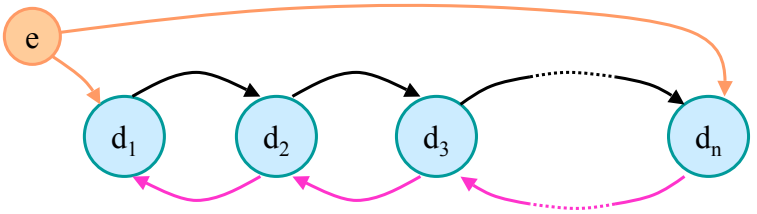


- Relacja następstwa dla listy jednokierunkowej *L*:

$$N = N_L$$
$$N_L = \left\{ \langle d_i, d_{i+1} \rangle : d_i, d_{i+1} \in D, i = 1 \ldots |D| - 1 \right\}$$

Dwukierunkowe listy niecykliczne

- Model grafowy listy dwukierunkowej:

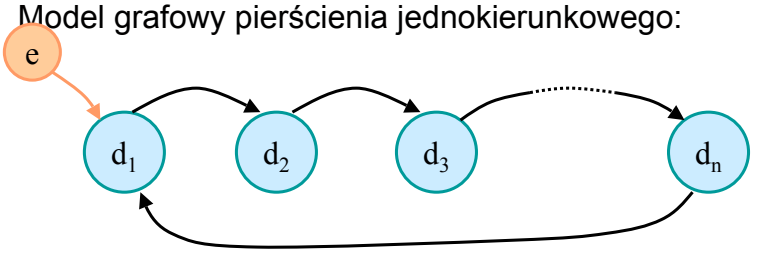


- Relacja następstwa dla listy dwukierunkowej *Ld:*

$$N_{Ld} = N_L \cup N_W$$
$$N_L = \left\{ \langle d_i, d_{i+1} \rangle : d_i, d_{i+1} \in D, i = 1 \ldots |D| - 1 \right\}$$
$$N_W = N_L^{-1}$$

Pierścień jednokierunkowy

- Model grafowy pierścienia jednokierunkowego:

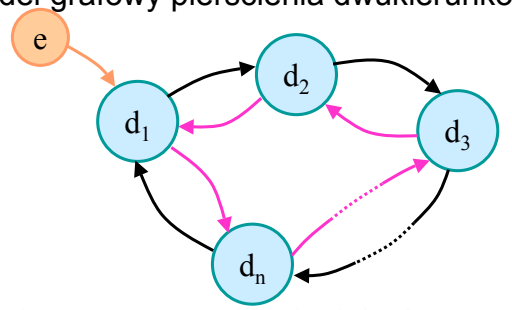


  - Relacja następstwa dla pierścienia jednokierunkowego *P:*

$$N_P = N_L \cup \left\{ \langle d_n, d_1 \rangle : d_n, d_1 \in D, n = |D| \right\}$$

Pierścienie dwukierunkowe

- Model grafowy pierścienia dwukierunkowego:



  - Relacja następstwa dla pierścienia dwukierunkowego *Pd:*

$$N_{PD} = N_P \cup N_{Pw}$$

$$N_{Pw} = N_P^{-1}$$

Definicja tablicy rozproszonej (z haszowaniem)

Definicja 2.3:

- Tablicą rozproszoną nazywamy trójkę uporządkowaną

$$T = (K, D, h)$$, gdzie

K = {k_1, k_2, k_3,..., k_n} - zbiór kluczy,
D = {d_1, d_2, d_3,..., d_n} - zbiór adresów,
h - funkcja *mieszająca (haszująca)* zdefiniowana następująco:

$$h : K \rightarrow D$$

Tradycyjnym obszarem zastosowań tablic rozproszonych są zagadnienia związane z przetwarzaniem danych.

Tablice rozproszone, funkcja haszująca

Zadaniem funkcji haszującej *h* jest w miarę równomierne obciążanie tablicy rozproszonej (jej komórek). Zagadnienie definiowania funkcji mieszającej jest istotne dla efektywności obliczeń realizowanych na bazie tablic rozproszonych.

- Ma to szczególnie duże znaczenie dla tablic rozproszonych przetwarzanych bezpośrednio w nośnikach zewnętrznych (taśmach, dyskach)

- Nie można jednak wykluczyć powstawania tzw. *konfliktów* w tablicach rozproszonych.

Konflikty w tablicach rozproszonych

- *Kolizją (konfliktem)* w tablicy rozproszonej nazywamy sytuację powstałą wtedy, gdy:

$$\exists_{k_i,k_j \in K,\, k_i \neq k_j} \bullet \left(h(k_i) = h(k_j)\right)$$

  - Elementy k_i, k_j biorące udział w kolizji nazywamy *synonimami*

Dużo więcej o listach i tablicach rozproszonych będziemy mówić na wykładzie 3 i 4

Drzewiaste struktury danych

Definicja 2.4:

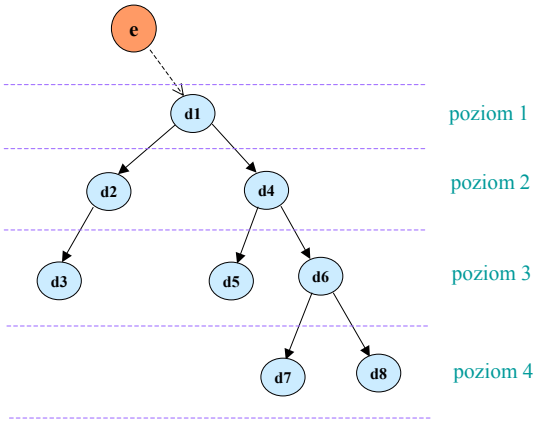
Drzewiastą strukturą danych nazywamy strukturę danych ***S=(D, R, e)***, w której relacja porządkująca ***N*** opisuje kolejne, rekurencyjne powiązania pomiędzy danymi elementarnymi drzewa, tworzącymi kolejne „poddrzewa".

Wniosek: Drzewo ze swojej natury jest strukturą hierarchiczną (rekurencyjną). Niezwykle istotne jest tutaj odpowiednie przypisanie danych elementarnych ze zbioru ***D*** do kolejnych poziomów drzewa i opisanie porządku w relacji ***N***.

Drzewiaste struktury danych – kilka pojęć podstawowych

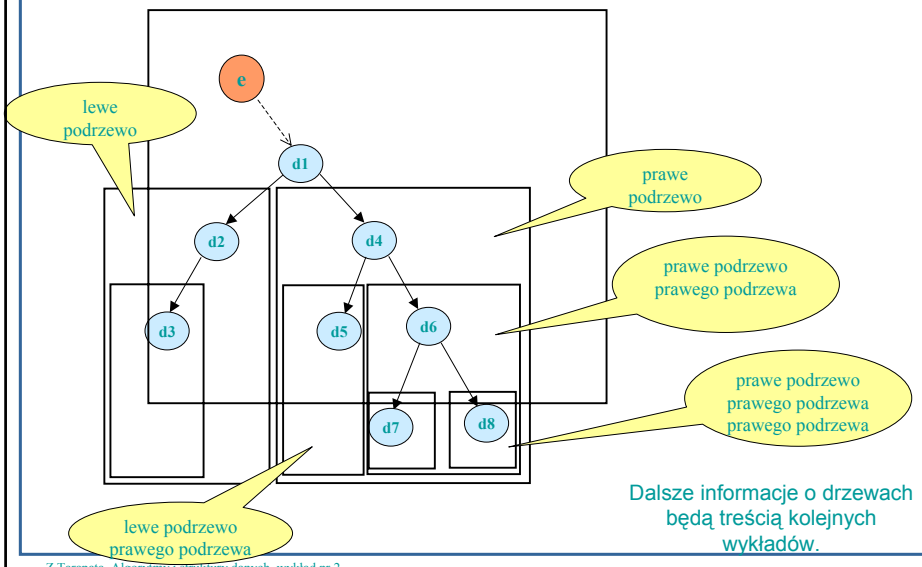
- *Korzeń drzewa* – jest tylko i wyłącznie jeden dla drzewa. Jest to dana wskazywana przez element wejściowy **e**,
- *Liść drzewa* – jest to dana elementarna, która nie posiada swojego następnika w w sensie relacji ***N***,
- *Stopień drzewa* – maksymalna liczba możliwych następników dla dowolnego elementu drzewa. Najczęściej przyjmuje się, że stopień drzewa jest potęgą liczby 2 (drzewa dwójkowe, czwórkowe, ósemkowe, szesnastkowe),
- *Droga w drzewie* – kolejne łuki pomiędzy wskazanymi dwoma elementami drzewa, które trzeba pokonać, aby dojść od jednego elementu drzewa do innego
- *Poziom drzewa* – elementy ułożone w tej samej odległości (długości drogi) od korzenia drzewa,
- *Drzewo zupełne* – takie drzewo, którego wszystkie elementy (oprócz liści) mają taką liczbę następników, ile wynosi stopień drzewa

Przykład modelu grafowego drzewa dwójkowego (binarnego)



Dla powyższego drzewa: wskaż korzeń, liście, opisz zbiór D, relacje r_{we} i N

Rekurencja w drzewie



Dalsze informacje o drzewach będą treścią kolejnych wykładów.

Realizacje struktur danych

- ***Realizacje sekwencyjne (statyczne)*** - wtedy, gdy z góry znamy maksymalny rozmiar przetwarzanej struktury liniowej i z góry chcemy zarezerwować dla niej określony zasób (pamięć operacyjne, pamięć zewnętrzna. W czasie wytwarzania programów komputerowych bazujemy wtedy na *zmiennych statycznych*,
- ***Realizacje łącznikowe (dynamiczne)*** - wtedy, gdy rozmiar struktury nie jest z góry znany i w czasie jej przetwarzania może istnieć konieczność dodawania do niej nowych elementów lub ich usuwania. W czasie wytwarzania programów komputerowych bazujemy wtedy na *zmiennych dynamicznych (wskaźnikowych)*,
- ***Realizacje łącznikowo-sekwencyjne (hybrydowe)*** - połączenie obu powyższych metod - wtedy gdy konieczny jest odpowiedni balans pomiędzy zmiennymi statycznymi i dynamicznymi

Przykłady liniowych struktur danych

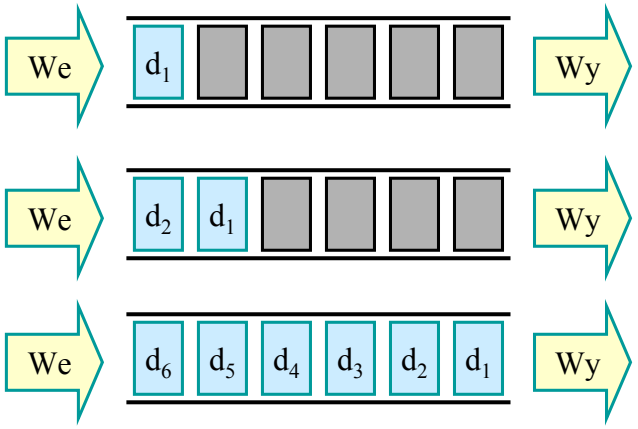
- Definicja listy stanowi podstawę dla zdefiniowania struktury liniowej. Wszystkie przypadki struktur liniowych można zdefiniować bazując na ich odpowiednich reprezentacjach listowych
- Bazując na pojęciu listy powiemy sobie o innych liniowych strukturach danych, będą to:
  - kolejki,
  - stosy,
  - napisy,
  - tablice sekwencyjne
  - tablice rzadkie
  - tablice rozproszone (z haszowaniem).

Kolejki

- Kolejka (queue) jest strukturą danych wykorzystywaną najczęściej jako bufor przechowujący dane określonych typów.
- Dyscypliny kolejek:
  - **FIFO** (First In First Out) - pierwszy element wchodzący staje się pierwszym wychodzącym
  - **Round-Robin** - cykliczna kolejka z dyscypliną czasu obsługi komunikatu przechowywanego w kolejce (w systemach operacyjnych, sieciach komputerowych)
  - **LIFO** (Last In First Out) - ostatni wchodzący staje się pierwszym wychodzącym (tak działa *stos*)
  - **kolejki priorytetowe** - dodatkowo w standardowym mechanizmie kolejki uwzględnia się wartości priorytetów przechowywanych danych

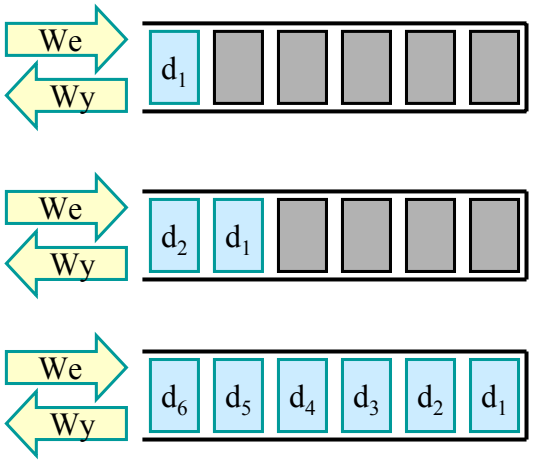
Kolejki FIFO

- Dyscyplina First In First Out:



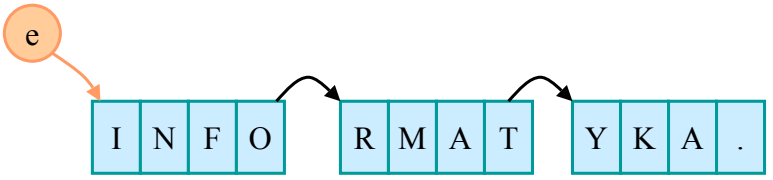
Stos (kolejka LIFO)

- Dyscyplina Last In First Out:



Napisy

- Napis (ang. string) może byś realizowany na trzy sposoby:
  - w postaci sekwencyjnej (np. tablica statyczna)
  - w postaci łącznikowej (każdy znak jest oddzielnym elementem) listy dynamicznej,
  - w postaci łącznikowo-sekwencyjnej (paczki napisów sekwencyjnych połączone łącznikami):



Łącznikowo - sekwencyjna realizacja napisu

Tablice sekwencyjne

- W praktyce programowania najczęściej spotykamy się z tablicami:
  - jednowymiarowymi (wektory),
  - dwuwymiarowymi (macierze)
  - trójwymiarowymi (prostopadłościany)
- Bardzo często używa się pojęcia *tablica* dla określenia implementacji macierzy
- Liczna grupa metod numerycznych wykorzystuje pojęcie tablicy (ciągu, sekwencji liczb)
- Zajmiemy się teraz bardzo ważnymi dla praktyki programowania algorytmami sortowania tablic

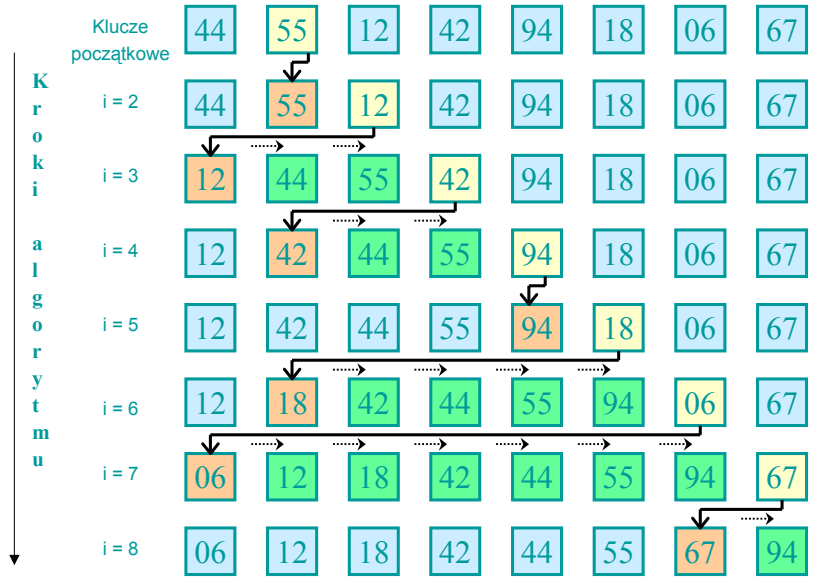
Algorytmy sortowania tablic sekwencyjnych

- Omówimy ideę czterech algorytmów sortowania tablic (wg. N.Wirth: „Algorytmy + struktury danych = programy”):
  - sortowanie przez *wstawianie* (ang. insertion sort)
  - sortowanie przez *wybieranie (selekcję)* (ang. selection sort)
  - sortowanie przez *zamianę* (ang. exchange sort). Sortowanie to nosi również nazwę sortowania *bąbelkowego* (ang. bubble sort),
  - sortowanie *mieszane* (ang. shake sort)

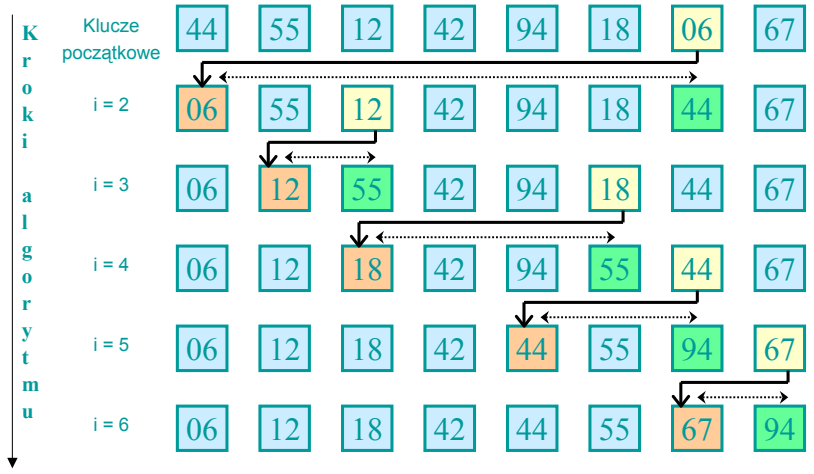
- We wszystkich przypadkach, w których sortujemy rosnąco, posłużymy się następującym przykładem tablicy liczb naturalnych:

Klucze początkowe	44	55	12	42	94	18	06	67

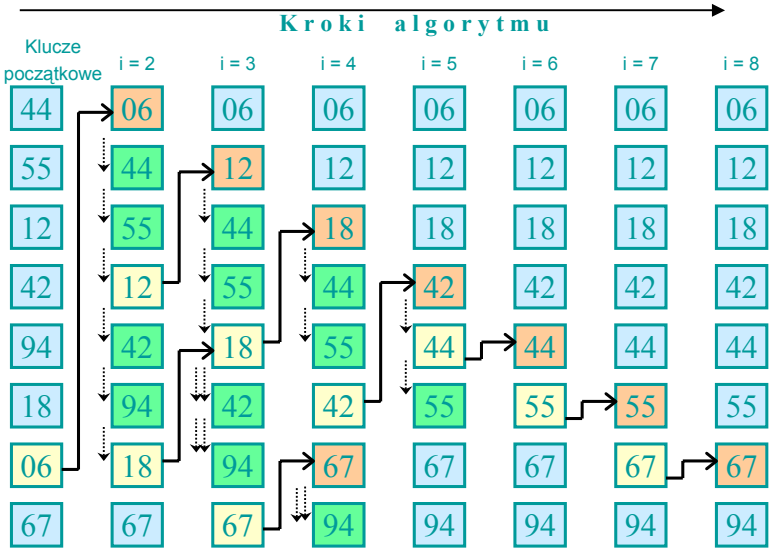
Sortowanie przez wstawianie



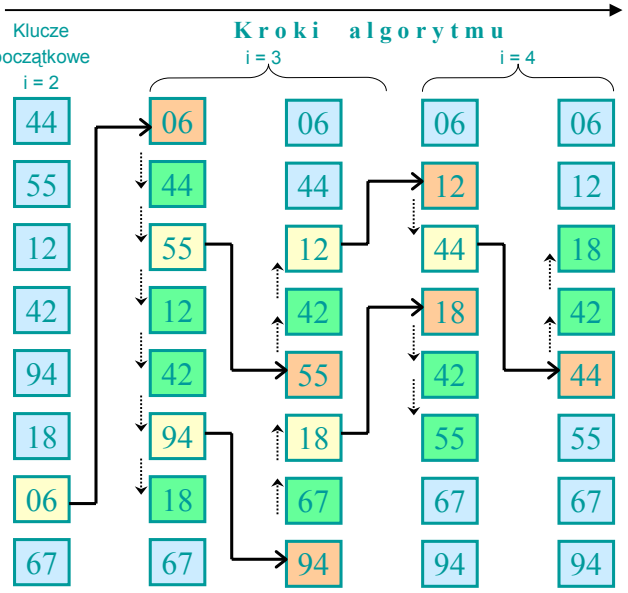
Sortowanie przez wybieranie



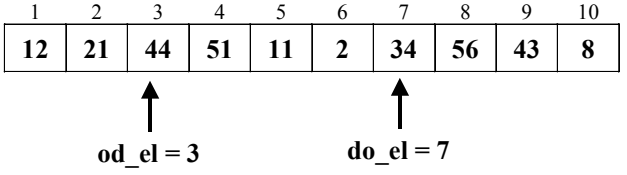
Sortowanie przez zamianę (bąbelkowe)



Sortowanie mieszane



Przykład implementacji algorytmu sortowania przez wybieranie – iteracyjnie - język Pascal

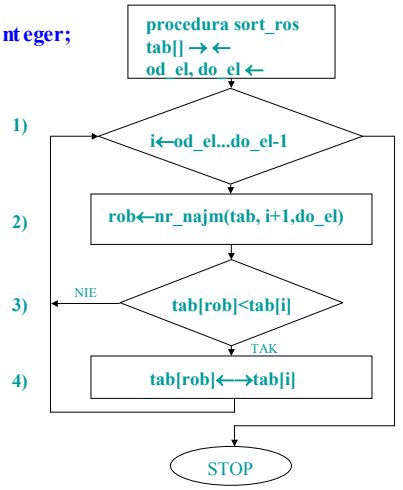


sortowanie rosnąco elementów w przedziale tablicy:

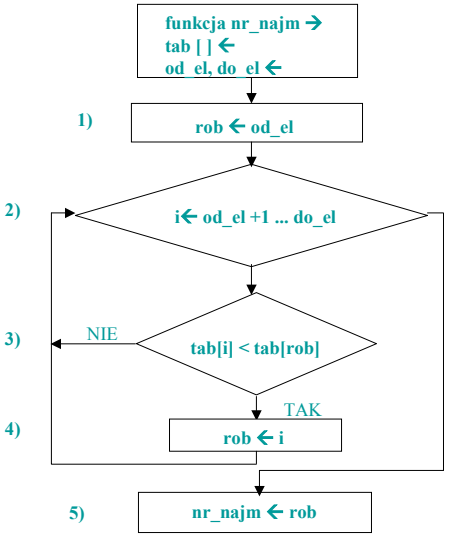
1. dla każdego elementu począwszy od numeru ***od_el*** aż do numeru ***do_el - 1***:
   1.1. znajdź najmniejszy z pozostałych tzn. ***od_el+1, od_el+2, ... ,do_el***
   1.2. jeśli znaleziony jest mniejszy od bieżącego:
      1.2.1. zamień miejscami znaleziony z bieżącym

Przykład implementacji algorytmu sortowania przez wybieranie - iteracyjnie, cd.

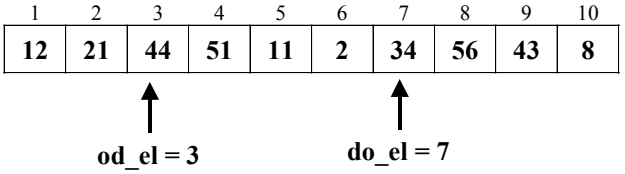
```
procedure sort_ros(var tab : array[1..N] of integer;
                   od_el, do_el : integer);
var
  i, rob, pom: integer;

begin
  for i := od_el to do_el-1 do
    begin
      rob := nr_najm(tab, i+1, do_el);
      if tab[rob] < tab[i] then
        begin
          pom := tab[rob];
          tab[rob] := tab[i];
          tab[i] := pom;
        end;
    end;
end;
```



Przykład implementacji algorytmu sortowania przez wybieranie - iteracyjnie, cd.



Przykład implementacji algorytmu sortowania przez wybieranie – rekurencyjnie - język Pascal

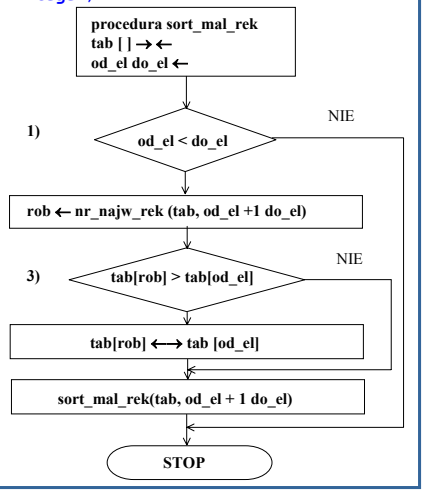


sortowanie rekurencyjne malejąco elementów w przedziale tablicy:

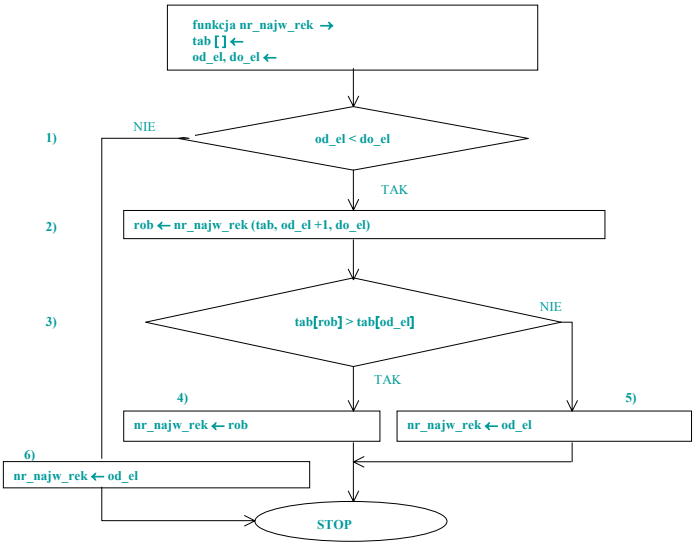
1. dla elementu bieżącego znajdź największy z pozostałych tzn. ***od_el+1, ... ,do_el***
2. jeśli znaleziony jest mniejszy od bieżącego:
   2.1. zamień miejscami znaleziony z bieżącym
3. posortuj pozostałe tzn. ***od_el+1, ... ,do_el***

Przykład implementacji algorytmu sortowania przez wybieranie - rekurencyjnie, cd.

```
procedure sort_mal_rek(var tab : array[1..N] of integer;
                       od_el,do_el : integer);
var
  rob,pom : integer;

begin
  if od_el < do_el then
    begin
      rob := nr_najw_rek(tab,od_el+1,do_el);
      if tab[rob] > tab[od_el] then
        begin
          pom := tab[rob];
          tab[rob] := tab[od_el];
          tab[od_el] := pom;
        end;
      sort_mal_rek(tab,od_el+1,do_el);
    end;
end;
```



Przykład implementacji algorytmu sortowania przez wybieranie - rekurencyjnie, cd.



Przykład implementacji algorytmu sortowania przez wybieranie - złożona struktura danych

```
const
  M_il_graczy = 10;

type
  T_il_graczy = 0..M_il_graczy;

T_pozycje =
   (srodkowy,
    skrzydlowy,
    rozgrywajacy);

T_gracz =
   record
     imie,
     nazwisko : string;
     wzrost : integer;
     gra_jako : set of T_pozycje;
   end;

 T_lista_graczy =
    array [1..M_il_graczy] of T_gracz;

 T_druzyna =
    record
       ilosc : T_il_graczy;
       lista : T_lista_graczy;
    end;
```

```
procedure sort_druzyny_ros (var d : T_druzyna;
                            od_el, do_el : integer);
var
   i, rob : integer;
   pom : T_gracz;

begin
 if (od_el < d.ilosc) and (do_el <= d.ilosc) then
    begin
       for i := od_el to do_el-1 do
          begin
             rob := nr_najm_gracza (d, i+1, do_el);
             if d.lista[rob].wzrost < d.lista[i].wzrost then
                begin
                   pom := d.lista[rob];
                   d.lista[rob] := d.lista[i];
                   d.lista[i] := pom;
                end;
          end;
    end;
end;
```

Złożoność obliczeniowa algorytmów sortowania

- Sortowanie przez wstawianie

$P_{Omin} = n - 1$ $\quad P_{Rmin} = 2\,(n - 1)$

$P_{Ośr} = 0.25\,(n^2 + n - 2)$ $\quad P_{Rśr} = 0.25\,(n^2 + 9n - 10)$

$P_{Omax} = 0.5\,(n^2 + n) - 1$ $\quad P_{Rmax} = 0.5\,(n^2 + 3n - 4)$

- n - ilość elementów w tablicy,
- P_{Oi} - liczba porównań klucza,
- P_{Ri} - liczba przesunięć elementów w tablicy

Złożoność obliczeniowa algorytmów sortowania, cd.

- Sortowanie przez wybieranie

$P_O = 0.5\,(n^2 - n)$ $\quad P_{Rmin} = 3\,(n - 1)$

$P_{Rśr} = n\,(\ln n + \gamma)$

$\gamma = 0.577216...$ stała Eulera

$P_{Rmax} = \text{trunc}\,(n^2/4) + 3\,(n - 1)$

- n - ilość elementów w tablicy,
- P_O - liczba porównań klucza (niezależna od liczności elementów),
- P_{Ri} - liczba przesunięć elementów w tablicy

Złożoność obliczeniowa algorytmów sortowania, cd.

- Sortowanie przez zamianę (bąbelkowe)

$P_O = 0.5 (n^2 - n)$

$P_{Rmin} = 0$

$P_{Rśr} = 0.75 (n^2 - n)$

$P_{Rmax} = 1.5 (n^2 - n)$

- n - ilość elementów w tablicy,
- P_O - liczba porównań klucza,
- P_{Ri} - liczba przesunięć elementów w tablicy

Złożoność obliczeniowa algorytmów sortowania, cd.

- Sortowanie mieszane

$P_{Omin} = n - 1$

$P_{Rmin} = 0$

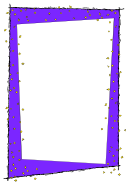
$P_{Ośr} = 0.5 (n^2 - n (k_2 + \ln n))$

$P_{Rśr} = n - k_1 \, sqrt(n)$

- n - ilość elementów w tablicy,
- k_1 - indeks poniżej którego tablica jest posortowana
- k_2 - indeks powyżej którego tablica jest posortowana
- P_O - liczba porównań klucza,
- P_{Ri} - liczba przesunięć elementów w tablicy

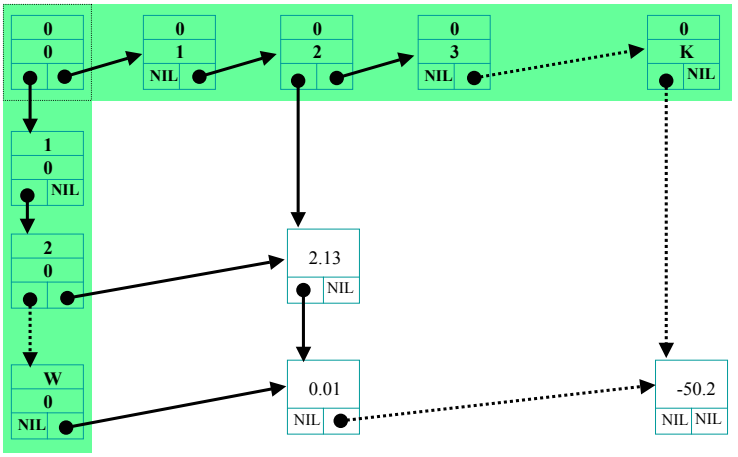
Pojęcie, zastosowania tablic rzadkich

- *Tablice rzadkie* są jednym z przypadków liniowych struktur danych. Ich cechą charakterystyczną jest to, że przechowują one jedynie tzw. *wartości niezerowe,*
- Po prostu pozycja tablicy z wartością zerową nie występuje (brak pozycji w macierzy oznacza wartość zerową),
- W tablicy są więc przechowywane wyłącznie wartości znaczące (tzn. różne od ustalonej z góry wartości domyślnej)
- Tablice rzadkie realizujemy wyłącznie w postaci łącznikowej (dynamiczne struktury danych)



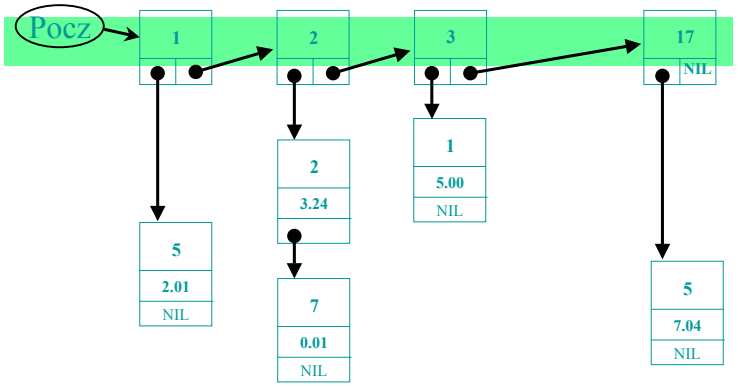
Pojęcie, zastosowania tablic rzadkich, cd.

- Przykład realizacji tablic rzadkich (tablica dynamiczna):



Pojęcie, zastosowania tablic rzadkich, cd.

- Przykład realizacji tablic rzadkich (lista dwupoziomowa):



Pojęcie, zastosowania tablic rzadkich, cd.

- Macierze rzadkie mogą być wykorzystywane do implementacji macierzy incydencji grafów
- Częstym zastosowaniem jest przechowywanie obrazów rastrowych (szczególnie wtedy, gdy na obrazie jest mało „zapalonych” punktów)
- *? Czy potrafisz wyobrazić sobie inne zastosowania macierzy rzadkich?*
  Wymień kilka Twoich propozycji w tym zakresie.

Konflikty w tablicach rozproszonych

- *Kolizją (konfliktem)* w tablicy rozproszonej nazywamy sytuację powstałą wtedy, gdy:

$$\exists_{k_i,k_j \in K,\, k_i \neq k_j} \bullet \left(h(k_i) = h(k_j)\right)$$

  - Elementy k_i, k_j biorące udział w kolizji nazywamy *synonimami*.

Metody haszowania w tablicach rozproszonych

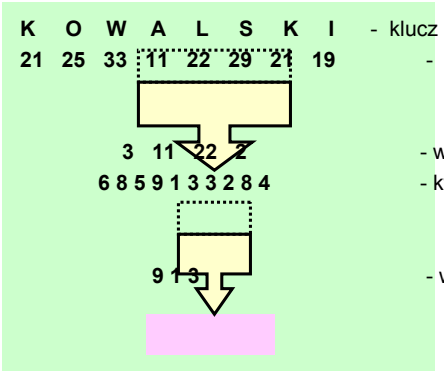
- Omówimy przykłady funkcji niezależnych od rozkładu klucza. Pierwszy z nich, to *randomizacja*:

$$d_i = \textit{randomize}(k_i)$$

- Jest ona dość prosta, ale rzadko wykorzystywana

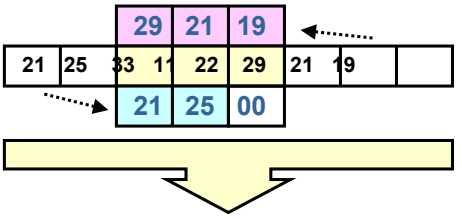
Metody haszowania w tablicach rozproszonych

- Metoda kwadratu środka:



Metody haszowania w tablicach rozproszonych, cd.

K O W A L S K I - klucz
21 25 33 11 22 29 21 19 - kody znaków



- d = 212500 + 331122 + 292119 = 834741 - wyliczony adres

Metody haszowania w tablicach rozproszonych, cd.

- Metoda dzielenia
  - adres wyliczany według formuły:

$d = wart(k) \bmod p$, gdzie:

d - adres w tablicy rozproszonej (czasami: indeks)
wart(k) - wartość wyliczona na podstawie klucza - inną metodą, np. metodą składania lub kwadratu środka
p - liczba pozycji w tablicy

- w zastosowaniach praktycznych metoda ta jest bardzo efektywna

Metody haszowania w tablicach rozproszonych, cd.

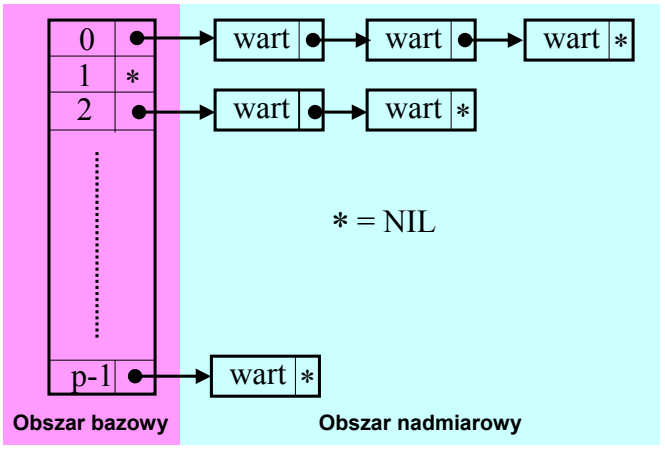
- Przykład zastosowania metody dzielenia
  - tablica ma 1000 pozycji
  - zakończenie przykładu przedstawionego dla metody *kwadratu środka*:
    - **$d_1 = 913 \bmod 1000 = 913$**
  - zakończenie przykładu przedstawionego dla metody *składania*:
    - **$d_2 = 834741 \bmod 1000 = 741$**

Organizacja tablic rozproszonych

- Tablice rozproszone bez obszaru nadmiarowego - tutaj dane znajdują się wyłącznie w *obszarze bazowym*
- Tablice rozproszone z obszarami nadmiarowymi:
  - z listami synonimów
  - rozproszone tablice indeksowe

Organizacja tablic rozproszonych, przykład

- Rozproszone tablice indeksowe



Usuwanie konfliktów w tablicach rozproszonych

- Bez obszarów nadmiarowych - szukanie liniowe
- $d_i = (d_0 + a * i) \bmod p$, gdzie

  d0 = h(k) - wynik początkowego haszowania klucza
  di - nowy adres, wyliczany wtedy, gdy w (i-1)-szym kroku wystąpił konflikt
  a - stały współczynnik empiryczny
  i - numer kolejnego kroku szukania (0 <= i <= p)
  p - liczba pozycji w tablicy

Usuwanie konfliktów w tablicach rozproszonych, cd.

- Bez obszarów nadmiarowych - szukanie kwadratowe
  - $d_i = (d_0 + a * i + b * i^2) \bmod p$,
  - lub wzór uproszczony:
    - $d_i = (d_0 + i^2)$, gdzie

      d0 = h(k) - wynik początkowego haszowania klucza
      di - nowy adres, wyliczany wtedy, gdy w (i-1)-szym kroku wystąpił konflikt
      a, b - stałe współczynniki empiryczny
      i - numer kolejnego kroku szukania (0 <= i <= p)
      p - liczba pozycji w tablicy

Usuwanie konfliktów w tablicach rozproszonych, cd.

- Bez obszarów nadmiarowych - szukanie sześcienne
  - $d_i = (d_0 + i^3)$, gdzie
    - $d_0 = h(k)$ - wynik początkowego haszowania klucza d_i - nowy adres, wyliczany wtedy, gdy w (i-1)-szym kroku wystąpił konflikt

      i - numer kolejnego kroku szukania (0 <= i <= p)

      p - liczba pozycji w tablicy

Usuwanie konfliktów w tablicach rozproszonych, cd.

- Z wykorzystaniem obszarów nadmiarowych:
  - *listy synonimów* : pierwsze wstawienie do wolnego miejsca w obszarze bazowym. Kiedy wyliczona w haszowaniu pozycja z obszaru bazowego jest zajęta, to wstawiamy nowy element do listy synonimów podwieszonej pod tą pozycję w obszarze bazowym. Listy synonimów tworzą obszary nadmiarowe
  - *rozproszona tablica indeksowa* - wszystkie dane są wstawiane do obszaru nadmiarowego
- Obszary nadmiarowe są organizowane w postali zestawu posortowanych *wykazów liniowych* lub posortowanych i zrównoważonych *wykazów leksykograficzych*

Dziękuję za uwagę