Fundusze Europejskie Wiedza Edukacja Rozwój | Rzeczpospolita Polska | Unia Europejska Europejski Fundusz Społeczny

Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Materiały do wykładu z przedmiotu:
Informatyka
Kod: EDS1B1007

WYKŁAD NR 13

Opracował: dr inż. Jarosław Forenc

Białystok 2020

Materiały zostały opracowane w ramach projektu „PB2020 - Zintegrowany Program Rozwoju Politechniki Białostockiej" realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

Plan wykładu nr 13

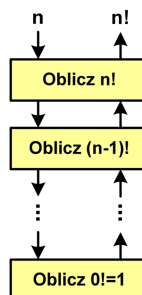
- Algorytmy komputerowe
  - rekurencja
  - złożoność obliczeniowa
  - algorytmy sortowania (proste wstawianie, proste wybieranie, bąbelkowe)
- Klasyfikacja systemów komputerowych (Flynna)
- Architektura von Neumanna i architektura harwardzka

Rekurencja

- Rekurencja lub rekursja - jest to odwoływanie się funkcji lub definicji do samej siebie
- Rozwiązanie danego problemu wyraża się za pomocą rozwiązań tego samego problemu, ale dla danych o mniejszych rozmiarach
- W matematyce mechanizm rekurencji stosowany jest do definiowania lub opisywania algorytmów
- Silnia:

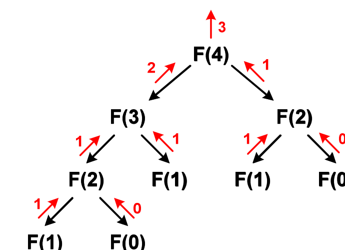
$$n! = \begin{cases} 1 & dla \quad n = 0 \\ n(n-1)! & dla \quad n \geq 1 \end{cases}$$

```
int silnia(int n)
{
    return n==0 ? 1 : n*silnia(n-1);
}
```



Rekurencja - ciąg Fibonacciego

$$F_n = \begin{cases} 0 & dla \quad n = 0 \\ 1 & dla \quad n = 1 \\ F_{n-1} + F_{n-2} & dla \quad n > 1 \end{cases}$$




```
int F(int n)
{
    if (n==0) return 0;
    if (n==1) return 1;
    return F(n-1) + F(n-2);
}
```

Złożoność obliczeniowa

- W celu rozwiązania danego problemu obliczeniowego szukamy algorytmu najbardziej efektywnego czyli:
  - najszybszego (najkrótszy czas otrzymania wyniku)
  - o możliwie małym zapotrzebowaniu na pamięć
- Problem: Jak ocenić, który z dwóch różnych algorytmów rozwiązujących to samo zadanie jest efektywniejszy?
- Do oceny efektywności służy złożoność obliczeniowa algorytmu (koszt algorytmu) czyli ilość zasobów potrzebnych do jego działania (czas, pamięć)

Złożoność obliczeniowa

Złożoność czasowa

- Czas wykonania algorytmu wyrażony w liczbie wykonywanych operacji (jednostkach czasu, liczbie cykli procesora) w zależności od wielkości danych
- Jej miarą jest zazwyczaj liczba podstawowych operacji (dominujących) - pozostałe operacje są pomijane
- Podstawowe operacje: porównanie, podstawienie, operacja arytmetyczna

Złożoność pamięciowa

- Jest miarą wykorzystania pamięci (liczba komórek pamięci)
- Wyrażana jest w liczbie bajtów lub liczbie zmiennych określonego typu w zależności od wielkości danych

Złożoność obliczeniowa

- Złożoność obliczeniowa algorytmu jest funkcją opisującą zależność między liczbą danych a liczbą operacji wykonywanych przez ten algorytm
- W praktyce stosuje się oszacowanie powyższej funkcji - są to tzw. notacje (klasy złożoności):
  - O (duże O) - najbardziej popularna
  - Ω (omega)
  - Θ (theta)

Notacja O („duże O”)

- Wyraża złożoność matematyczną algorytmu
- Do wyznaczenia złożoności bierze się pod uwagę liczbę dominujących operacji wykonywanych w algorytmie
- Przykład zapisu: $O(n^2)$
  - po literze O występuje wyrażenie w nawiasach zawierające literę n, która oznacza liczbę elementów, na których działa algorytm
- W funkcji opisującej złożoność bierze się pod uwagę tylko najistotniejszy składnik, np.

  $f(n) = n^2+2n \rightarrow O(n^2)$ $f(n) = n^2+n-5 \rightarrow O(n^2)$

- W powyższych przykładach dla dużego n wpływ składnika liniowego i stałego na wartość funkcji jest nieistotny w porównaniu ze składnikiem głównym n^2

Notacja O („duże O”)

- Porównanie najczęściej występujących złożoności:

Elementy (n)	O(log n)	O(n)	O(n log n)	$O(n^2)$	$O(n^3)$	$O(2^n)$
10	3	10	33	100	1 000	1024
100	7	100	664	10 000	1 000 000	$1{,}27 \cdot 10^{30}$
1 000	10	1 000	9 966	1 000 000	10^9	$1{,}07 \cdot 10^{301}$
10 000	13	10 000	132 877	10^8	10^{12}	$1{,}99 \cdot 10^{3010}$

O(log n) - logarytmiczna (np. przeszukiwanie binarne)
O(n) - liniowa (np. porównywanie łańcuchów znaków)
O(n log n) - liniowo-logarytmiczna (np. sortowanie szybkie)
$O(n^2)$ - kwadratowa (np. proste algorytmy sortowania)
$O(n^3)$ - sześcienna (np. mnożenie macierzy)
$O(2^n)$ - wykładnicza (np. problem komiwojażera)

Sortowanie

- Sortowanie polega na uporządkowaniu zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru (wartości każdego elementu)
- W przypadku liczb, sortowanie polega na znalezieniu kolejności liczb zgodnej z relacją ≤ lub ≥

Przykład:

- Tablica nieposortowana:

6	4	5	2	3	1

- Tablica posortowana zgodnie z relacją ≤ (od najmniejszej do największej liczby):

1	2	3	4	5	6

- Tablica posortowana zgodnie z relacją ≥ (od największej do najmniejszej liczby):

6	5	4	3	2	1

Sortowanie

- W przypadku słów sortowanie polega na ustawieniu ich w porządku alfabetycznym (leksykograficznym)

Przykład:

- Tablica nieposortowana:

Paweł	Piotr	Adrian	Ela	Ola	Henryk

- Tablice posortowane:

Adrian	Ela	Henryk	Ola	Paweł	Piotr

Piotr	Paweł	Ola	Henryk	Ela	Adrian

Sortowanie

- W praktyce sortowanie sprowadza się do porządkowanie danych na podstawie porównania - porównywany element to klucz

Przykład:

- Tablica nieposortowana (imię, nazwisko, wiek):

Piotr	Ola	Paweł	Jan	Ela	Magda
Kowalski	Nowak	Wójcik	Kamiński	Król	Mazur
25	18	23	20	22	15

- Tablica posortowana (klucz - nazwisko):

Jan	Piotr	Ela	Magda	Ola	Paweł
Kamiński	Kowalski	Król	Mazur	Nowak	Wójcik
20	25	22	15	18	23

- Tablica posortowana (klucz - wiek):

Magda	Ola	Jan	Ela	Paweł	Piotr
Mazur	Nowak	Kamiński	Król	Wójcik	Kowalski
15	18	20	22	23	25

Sortowanie

Po co stosować sortowanie?

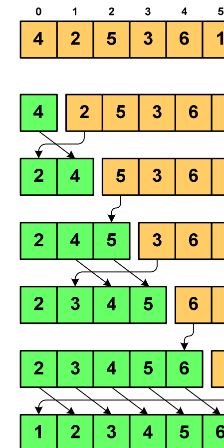
- Posortowane elementy można szybciej zlokalizować
- Posortowane elementy można przedstawić w czytelniejszy sposób

Klasyfikacje algorytmów sortowania

- Złożoność obliczeniowa algorytmu - zależność liczby wykonywanych operacji od liczebności sortowanego zbioru n
- Złożoność pamięciowa - wielkość zasobów zajmowanych przez algorytm (sortowanie w miejscu - wielkość zbioru danych podczas sortowania nie zmienia się lub jest tylko nieco większa)
- Sortowanie wewnętrzne (odbywa się w pamięci komputera) i zewnętrzne (nie jest możliwe jednoczesne umieszczenie wszystkich elementów zbioru sortowanego w pamięci komputera)

Proste wstawianie (insertion sort)

Przykład:



Program w języku C:

```c
int main(void)
{
    int tab[N],i,j,tmp;
    // ...
    for (i=1; i<N; i++)
    {
        j=i;
        tmp=tab[i];
        while (tab[j-1]>tmp && j>0)
        {
            tab[j]=tab[j-1];
            j--;
        }
        tab[j]=tmp;
    }
}
```

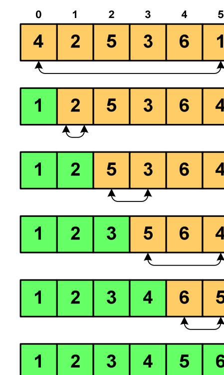

Proste wstawianie (insertion sort)

- Złożoność algorytmu: $O(n^2)$
  + wydajny dla danych wstępnie posortowanych
  + wydajny dla zbiorów o niewielkiej liczebności
  + małe zasoby zajmowane podczas pracy (sortowanie w miejscu)
  + prosty w implementacji

  − mała efektywność dla normalnej i dużej ilości danych.

Proste wybieranie (selection sort)

Przykład:



Program w języku C:

```c
int main(void)
{
    int tab[N],i,j,k,tmp;
    // ...
    for (i=0;i<N-1;i++)
    {
        k=i;
        for (j=i+1; j<N; j++)
            if (tab[k]>=tab[j])
                k = j;
        tmp = tab[i];
        tab[i] = tab[k];
        tab[k] = tmp;
    }
}
```

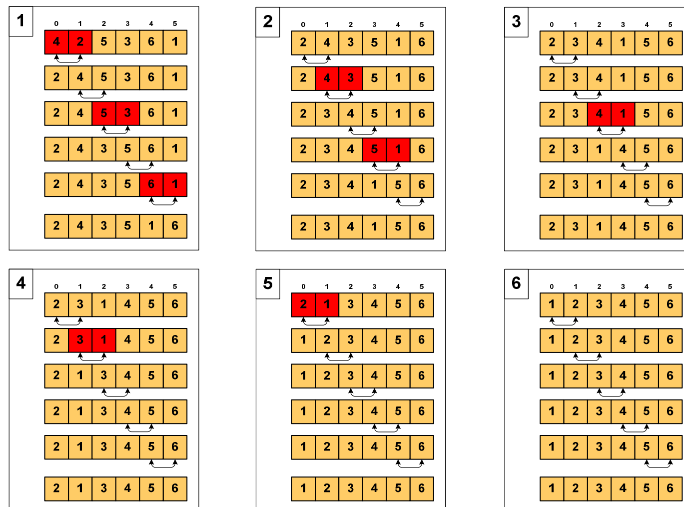
Proste wybieranie (selection sort)

- Złożoność algorytmu: $O(n^2)$
  - \+ szybki w sortowaniu niewielkich tablic
  - \+ małe zasoby zajmowane podczas pracy (sortowanie w miejscu)
  - \+ prosty w implementacji
  - – liczba porównań elementów jest niezależna od początkowego rozmieszczenia elementów w tablicy
  - – w algorytmie może zdarzyć się, że wykonywana jest zamiana tego samego elementu ze sobą.

Bąbelkowe (bubble sort)

- Sortowanie bąbelkowe (ang. bubble sort), nazywane jest także:
  - sortowaniem pęcherzykowym
  - sortowaniem przez prostą zamianę (ang. straight exchange)
- Metoda ta polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności jeśli jest to konieczne
- Nazwa metody wzięła się stąd, że kolejne porównania powodują „wypychanie” kolejnego największego elementu na koniec („wypłynięcie największego bąbelka”)

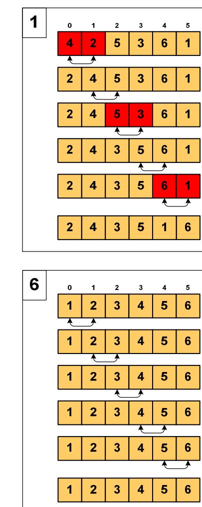
Bąbelkowe (bubble sort)



Bąbelkowe (bubble sort)

Program w języku C:

```c
int main(void)
{
   int tab[N],i,j,tmp,koniec;
   // ...
   do {
      koniec=1;
      for (i=0;i<N-1;i++)
         if (tab[i]>tab[i+1])
         {
            tmp=tab[i];
            tab[i]=tab[i+1];
            tab[i+1]=tmp;
            koniec=0;
         }
   } while (!koniec);
}
```



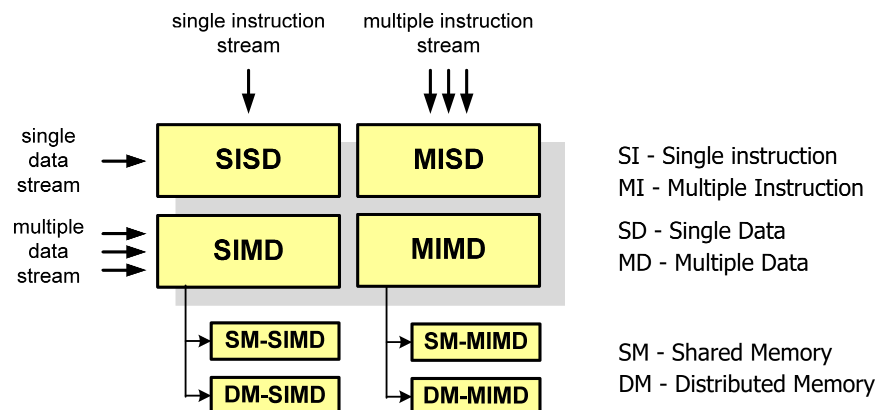
Bąbelkowe (bubble sort)

- Złożoność algorytmu: $O(n^2)$
  - \+ prosta realizacja
  - \+ wysoka efektywność użycia pamięci (sortowanie w miejscu)
  - – mała efektywność.

Klasyfikacja systemów komputerowych

- Taksonomia Flynna - pierwsza, najbardziej ogólna klasyfikacja architektur komputerowych (1972):
  - Flynn M.J.: „Some Computer Organizations and Their Effectiveness”, IEEE Transactions on Computers, Vol. C-21, No 9, 1972.
- Opiera się na liczbie przetwarzanych strumieni rozkazów i strumieni danych:
  - strumień rozkazów (Instruction Stream) - odpowiednik licznika rozkazów; system złożony z n procesorów posiada n liczników rozkazów, a więc n strumieni rozkazów
  - strumień danych (Data Stream) - zbiór operandów, np. system rejestrujący temperaturę mierzoną przez n czujników posiada n strumieni danych

Taksonomia Flynna



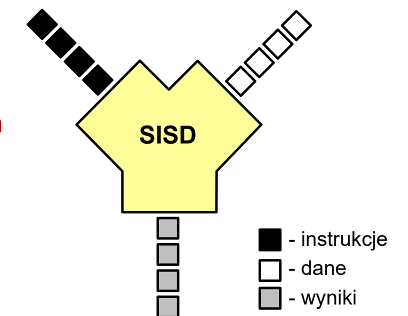
SI - Single instruction
MI - Multiple Instruction

SD - Single Data
MD - Multiple Data

SM - Shared Memory
DM - Distributed Memory

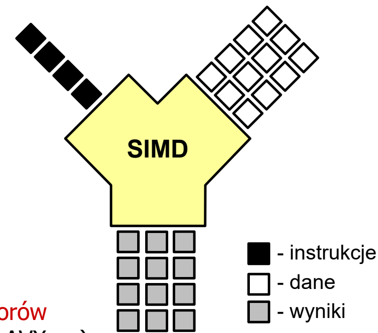
SISD (Single Instruction, Single Data)

- Jeden wykonywany program przetwarza jeden strumień danych
- Klasyczne komputery zbudowane według architektury von Neumanna
- Zawierają:
  - jeden procesor
  - jeden blok pamięci operacyjnej zawierający wykonywany program.



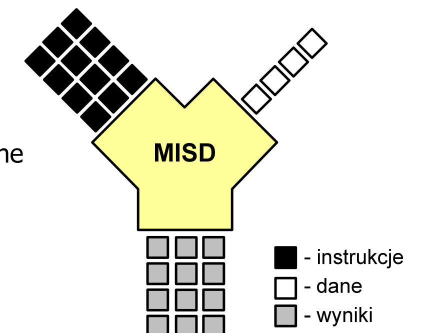
SIMD (Single Instruction, Multiple Data)

- Jeden wykonywany program przetwarza wiele strumieni danych
- Te same operacje wykonywane są na różnych danych
- Podział:
  - SM-SIMD (Shared Memory SIMD):
    - komputery wektorowe
    - rozszerzenia strumieniowe procesorów (MMX, 3DNow!, SSE, SSE2, SSE3, AVX, …)
  - DM-SIMD (Distributed Memory SIMD):
    - tablice procesorów
    - procesory kart graficznych (GPGPU)



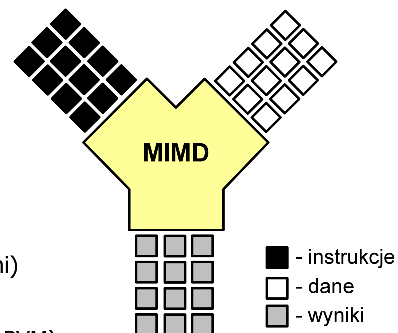
MISD (Multiple Instruction, Single Data)

- Wiele równolegle wykonywanych programów przetwarza jednocześnie jeden wspólny strumień danych
- Systemy tego typu nie są spotykane



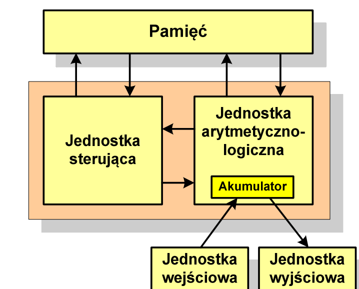
MIMD (Multiple Instruction, Multiple Data)

- Równolegle wykonywanych jest wiele programów, z których każdy przetwarza własne strumienie danych
- Podział:
  - SM-MIMD (Shared Memory):
    - wieloprocesory (np. komputery z procesorami wielordzeniowymi)
  - DM-MIMD (Distributed Memory):
    - wielokomputery (biblioteki MPI, PVM)
    - klastry
    - gridy



Architektura von Neumanna

- Rodzaj architektury komputera, opisanej w 1945 roku przez matematyka Johna von Neumanna
- Inne nazwy: architektura z Princeton, store-program computer (koncepcja przechowywanego programu)
- Zakłada podział komputera na kilka części:
  - jednostka sterująca (CU - Control Unit)
  - jednostka arytmetyczno-logiczna (ALU - Arithmetic Logic Unit)
  - pamięć główna (memory)
  - urządzenia wejścia-wyjścia (input/output)



Architektura von Neumanna - podstawowe cechy

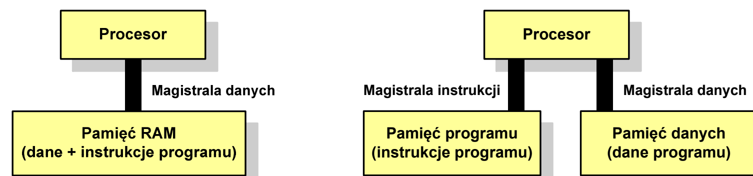
- Informacje przechowywane są w komórkach pamięci (cell) o jednakowym rozmiarze, każda komórka ma numer - adres
- Dane oraz instrukcje programu (rozkazy) zakodowane są za pomocą liczb i przechowywane w tej samej pamięci
- Dane i instrukcje czytane są przy wykorzystaniu tej samej magistrali
- Praca komputera to sekwencyjne odczytywanie instrukcji z pamięci komputera i ich wykonywanie w procesorze
- Wykonanie rozkazu:
  - pobranie z pamięci słowa będącego kodem instrukcji
  - pobranie z pamięci danych
  - wykonanie instrukcji
  - zapisanie wyników do pamięci

Architektura harwardzka

- Nazwa architektury pochodzi od komputera Harward Mark I:
  - zaprojektowany przez Howarda Aikena
  - pamięć instrukcji - taśma dziurkowana, pamięć danych - elektromechaniczne liczniki
- Architektura komputera, w której pamięć danych jest oddzielona od pamięci instrukcji
- Pamięci danych i instrukcji mogą różnić się:
  - technologią wykonania
  - strukturą adresowania
  - długością słowa
- Procesor może w tym samym czasie czytać instrukcje oraz uzyskiwać dostęp do danych

Architektura harwardzka i von Neumanna

- W architekturze harwardzkiej pamięć instrukcji i pamięć danych:
  - zajmują różne przestrzenie adresowe
  - mają oddzielne szyny (magistrale) do procesora
  - zaimplementowane są w inny sposób



Architektura von Neumanna

Architektura harwardzka

- Zmodyfikowana architektura harwardzka:
  - oddzielone pamięci danych i rozkazów, lecz wykorzystujące wspólną magistralę

Koniec wykładu nr 13

Dziękuję za uwagę!