Krzysztof Regulski
AGH, WIMiIP

ALGORYTMY

1. Algorytm

- formalny zapis ciągu czynności, które należy wykonać, aby na podstawie określonych danych uzyskać żądany wynik.
- zwykle algorytmy pracują na **danych wejściowych** i uzyskują z nich **dane wyjściowe**. Informacje zapisane w pamięci maszyny traktuje się jako jej **stan wewnętrzny**. Niektóre algorytmy mają za zadanie wyłącznie przeprowadzanie komputera z jednego stanu wewnętrznego do innego.
- algorytm jest "niezależnym" od jego implementacji przepisem, a program może zostać zinterpretowany i wykonany przez komputer.
- ograniczenia:
  - struktury danych
  - czas realizacji
- kryterium jakości (złożoność obliczeniowa):
  - wydajność czasowa
  - zużycie pamięci

2. Metody algorytmów

- **dziel i zwyciężaj** – dzielimy problem na kilka mniejszych, a te znowu dzielimy, aż ich rozwiązania staną się oczywiste,
- **programowanie dynamiczne** – problem dzielony jest na kilka, ważność każdego z nich jest oceniana i po pewnym wnioskowaniu wyniki analizy niektórych prostszych zagadnień wykorzystuje się do rozwiązania głównego problemu,
- **metoda zachłanna** – nie analizujemy podproblemów dokładnie, tylko wybieramy najbardziej obiecującą w tym momencie drogę rozwiązania,
- **programowanie liniowe** – oceniamy rozwiązanie problemu przez pewną funkcję jakości i szukamy jej minimum,
- **poszukiwanie i wyliczanie** – kiedy przeszukujemy zbiór danych aż do odnalezienia rozwiązania,
- **algorytm probabilistyczny** – algorytm działa poprawnie z bardzo wysokim prawdopodobieństwem, ale wynik nie jest pewny,
- **heurystyka** – człowiek na podstawie swojego doświadczenia tworzy algorytm, który działa w najbardziej prawdopodobnych warunkach, rozwiązanie zawsze jest przybliżone.

3. Najważniejsze techniki implementacji algorytmów komputerowych

- **proceduralność** – algorytm dzielimy na szereg podstawowych procedur, wiele algorytmów współdzieli wspólne biblioteki standardowych procedur, z których są one wywoływane w razie potrzeby,
- **praca sekwencyjna** – wykonywanie kolejnych procedur algorytmu, według kolejności ich wywołań, na raz pracuje tylko jedna procedura,
- **praca wielowątkowa** – procedury wykonywane są sekwencyjnie, lecz kolejność ich wykonania jest trudna do przewidzenia dla programisty
- **praca równoległa** – wiele procedur wykonywanych jest w tym samym czasie, wymieniają się one danymi,
- **rekurencja** – procedura lub funkcja wywołuje sama siebie, aż do uzyskania wyniku lub błędu,
- **obiektowość** – procedury i dane łączymy w pewne klasy reprezentujące najważniejsze elementy algorytmu oraz stan wewnętrzny wykonującego je urządzenia.

4. Przykłady

- algorytmy sortowania
- problem plecakowy
- algorytmy wyszukiwania
- algorytmy ścieżki krytycznej
- algorytmy sztucznej inteligencji
- algorytmy przeszukiwania drzew
- algorytmy kryptograficzne
- algorytm genetyczny
- algorytm kwantowy
- algorytm mrówkowy

5. Sposoby zapisu algorytmów

- Zapis w postaci ciągu kroków (języka naturalnego)
- Zapis w postaci graficznej - schematy blokowe
- Zapis w języku symbolicznym (pseudokodu)
- Zapis w języku programowania

6. Złożoność obliczeniowa

- Jest to jeden z najważniejszych parametrów charakteryzujących algorytm. Decyduje on o efektywności całego programu. Podstawowymi zasobami systemowymi uwzględnianymi w analizie algorytmów są czas działania oraz obszar zajmowanej pamięci. Na złożoność czasową składają się dwie wartości: **pesymistyczna**, czyli taka, która charakteryzuje najgorszy przypadek działania oraz **oczekiwana**.

Najczęściej algorytmy mają złożoność czasową proporcjonalną do funkcji:

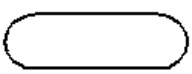
- log(n) - złożoność logarytmiczna
- n - złożoność liniowa
- nlog(n) - złożoność liniowo-logarytmiczna
- n^2 - złożoność kwadratowa
- n^k - złożoność wielomianowa
- 2^n - złożoność wykładnicza
- n! - złożoność wykładnicza, ponieważ $n! > 2^n$ już od n=4

N (złożoność liniowa)	$\log_2(N)$ (złożoność logarytmiczna)	2^N (złożoność wykładnicza)
1	0	2
2	1.00	4
3	1,58	8
4	2,00	16
10	3,32	1024
11	3,46	2048
12	3,58	4096
20	4,32	1048576
30	4,91	1073741824
40	5,32	1099511627776
60	5,91	1152921504606846976
100	6,64	26765060022822940149670320 53 76
1000	9,97	ok. $1*10^{301}$

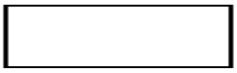
7. Schematy blokowe

- Schematy blokowe są tzw. metajęzykiem. Oznacza to, że jest to język bardzo ogólny, służy do opisywania algorytmów w taki sposób, by na jego podstawie można było je zaimplementować w każdym języku.

- Poszczególne elementy schematu łączy się za pomocą strzałek. W większości przypadków blok ma jedną strzałkę wchodzącą i jedną wychodzącą, lecz są także wyjątki (omówię je poniżej).

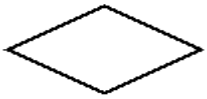


- Ta figura oznacza początek lub koniec algorytmu. W każdym algorytmie musi się znaleźć dokładnie jedna taka figura z napisem "Start" oznaczająca początek algorytmu oraz **dokładnie jedna** figura z napisem "Stop" oznaczająca koniec algorytmu

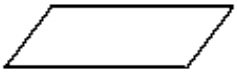


- Jest to figura oznaczająca proces. W jej obrębie umieszczamy wszelkie obliczenia lub podstawienia.

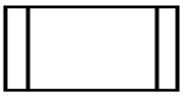
Proces ma dokładnie jedną strzałkę wchodzącą i dokładnie jedną strzałkę wychodzącą.



- Romb symbolizuje blok decyzyjny. Umieszcza się w nim jakiś warunek (np. `x>2`). Z dwóch wybranych wierzchołków rombu wyprowadzamy dwie możliwe drogi: gdy warunek jest spełniony (strzałkę wychodzącą z tego wierzchołka należy opatrzyć etykietą "Tak") oraz gdy warunek nie jest spełniony. Każdy romb ma dokładnie jedną strzałkę wchodzącą oraz dokładnie dwie strzałki wychodzące.



- Równoległobok jest stosowany do odczytu lub zapisu danych. W jego obrębie należy umieścić stosowną instrukcję np. `Read(x)` lub `Write(x)` (można też stosować opis słowny np. "`Drukuj x na ekran`"). Figura ta ma dokładnie jedną strzałkę wchodzącą i jedną wychodzącą.



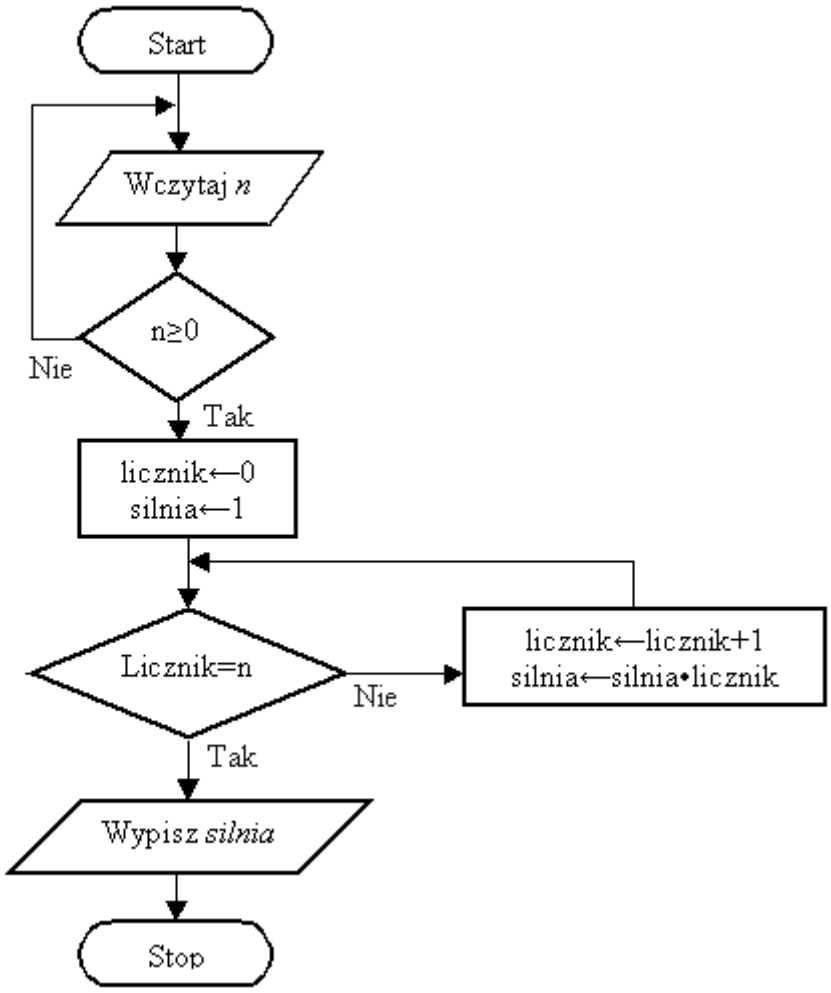
- Ta figura symbolizuje proces, który został już kiedyś zdefiniowany. Można ją porównać do procedury, którą definiuje się raz w programie, by następnie móc ją wielokrotnie wywoływać. Warunkiem użycia jest więc wcześniejsze zdefiniowanie procesu. Podobnie jak w przypadku zwykłego procesu i tu mamy jedno wejście i jedno wyjście.

- Koło symbolizuje tzw. łącznik stronicowy. Może się zdarzyć, że chcemy "przeskoczyć" z jednego miejsca na kartce na inne (np. by nie krzyżować strzałek). Możemy w takim wypadku posłużyć się łącznikiem. Umieszczamy w jednym miejscu łącznik z określonym symbolem w środku (np. cyfrą, literą) i doprowadzamy do nie go strzałkę. Następnie w innym miejscu kartki umieszczamy drugi łącznik z takim samym symbolem w środku i wyprowadzamy z niego strzałkę. Łącznik jest często porównywany do teleportacji (z jednego miejsca na kartce do drugiego). Łączniki występują więc w parach, jeden ma tylko wejście a drugi wyjście.

- Ten symbol to łącznik międzystronicowy. Działa analogicznie jak pierwszy, lecz nie w obrębie strony. Przydatne w złożonych algorytmach, które nie mieszczą się na jednej kartce. Uwaga: jeśli stosujemy oba typy łączników w schemacie, to najlepiej jest stosować liczby do identyfikowania jednych i litery do drugich. Dzięki temu nie dojdzie do pomyłki.

8. Algorytm: Silnia

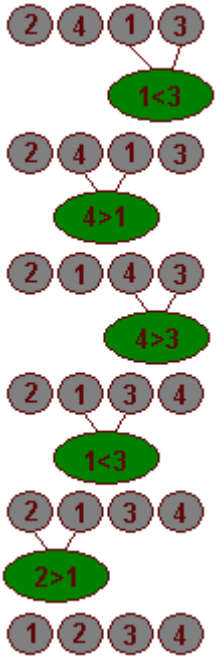
- Rekurencyjna definicja silni wygląda następująco:
```
silnia(0)=1
silnia(n)=n*silnia(n-1)
```



- **Zadanie 1** (1 pkt): napisz program liczący silnię zgodnie z powyższym algorytmem

9. Algorytm: Sortowanie bąbelkowe

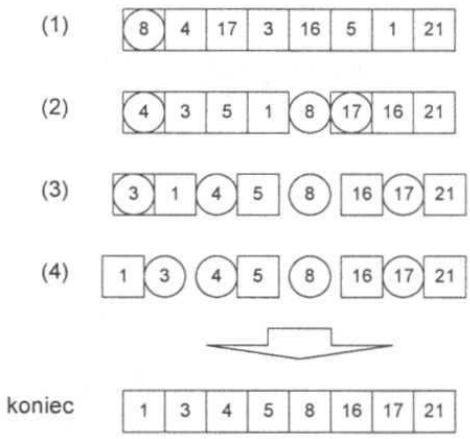
- Jest to jeden z prostszych algorytmów sortowania. Sprawdzamy całą tablicę od końca, jeżeli trafimy na parę elementów, w której większy poprzedza mniejszy to zamieniamy je miejscami i znów zaczynamy przeszukiwać tą tablicę od końca. Czynność powtarzamy tak długo aż podczas sprawdzania całej tablicy, nie zajdzie ani jedna zamiana elementów. Realizuje się to najczęściej za pomocą zmiennej logicznej. Algorytm nosi nazwę bąbelkowego, gdyż najmniejsze liczby "wypływają" z dołu tablicy na jej szczyt. Oto przykład zastosowania dla nieuporządkowanego ciągu liczb <2, 4, 1, 3>.



- Przy następnym przebiegu nie zajdzie ani jedna zmiana, to znak, że ciąg jest już posortowany. Z powyższego zdania można wyciągnąć wniosek, że gdy ciąg wejściowy będzie posortowany, to algorytm wykona tylko jeden przebieg. Jest to duża zaleta tego sposobu sortowania, niektóre metody będą sortowały ciąg nawet jeśli będzie on posortowany.
  Z kolei najgorszym zestawem danych dla tego algorytmu jest ciąg posortowany nierosnąco.
- **Zadanie 2** (1 pkt): Narysuj algorytm sortowania bąbelkowego, a następnie napisz program sortujący macierz 1x10 wypełnioną liczbami losowymi

10. Algorytm: Sortowanie szybkie

- Algorytm sortowania szybkiego jest uważany za najszybszy algorytm dla danych losowych. Zasada jego działania opiera się o metodę **dziel i zwyciężaj**. Zbiór danych zostaje podzielony na dwa podzbiory i każdy z nich jest sortowany niezależnie od drugiego.
- Wybieramy element (osiowy, dowolny) tablicy, a następnie przegrupowujemy ją tak, aby wszystkie mniejsze elementy znalazły się z lewej strony, a większe z prawej. Następnie w obu podzbiorach wybieramy elementy osiowe i ponawiamy grupowanie. Sortowanie wykonujemy tak długo, jak długo porządkowane podzbiory zawierają co najmniej dwa elementy.




```
program quicksort_1;
{ Program porządkuje tablicę liczb typu Byte metodą szybkiego sortowania. }
uses  Crt;
const
  IleLiczb = 200;      { dla wartości typu Byte - max. 65000 }
type
  TTablica = array[1..IleLiczb] of Byte;
var
  Tablica : TTablica;
  i : Word;
procedure QuickSort(var T : TTablica; Poczatek, Koniec : Word);
{ Procedura sortuje tablicę liczb T metodą szybkiego sortowania. }
{ Początek i Koniec określają granice sortowanego podprzedziału. }
  var
    Element, i : Word;
    Pomoc : Byte;
  begin
    if (Poczatek < Koniec) then
    begin
      Element := Poczatek;
      for i := Poczatek+1 to Koniec do
      if (T[i] < T[Poczatek]) then      { zamień elementy miejscami }
      begin
        Element := Element + 1;
        Pomoc := T[Element]; T[Element] := T[i]; T[i] := Pomoc;
      end;
      Pomoc := T[Poczatek]; T[Poczatek] := T[Element]; T[Element] := Pomoc;
      { wywołaj rekurencyjnie dla obu podciągów pozostałych po rozdzieleniu
}
      QuickSort(T, Poczatek, Element-1);
      QuickSort(T, Element+1, Koniec);
    end;
end; {      OuickSort -}
procedure WypiszTablice(var T : TTablica; Ile : Word);
{ Procedura wypisuje zawartość tablicy T. Sposób wypisywania dostosowano }
{ do zmiennych typu Byte (4 pola na wartość). }
  var
    i : Word;
    begin
      for i := 1 to Ile do
      write(T[i]:4);
      writeln;
end; {      WypiszTablice -}
begin
  ClrScr;
  { wypełnij tablicę liczbami i wypisz jej zawartość }
```

```
  Randomize;
  for i := 1 to IleLiczb do
  Tablica[i] := Random(256);
  WypiszTablice(Tablica, IleLiczb);
  readln;
  { sortuj tablicę i ponownie wypisz jej zawartość }
  QuickSort(Tablica, 1, IleLiczb);
  WypiszTablice(Tablica, IleLiczb);
  readln;
end.
```

- **zadanie 3** (1 pkt): Narysuj schemat blokowy algorytmu sortowania szybkiego