Przetwarzanie danych

Tworzenie programów

Wiemy:

- Co to jest komputer
- Z jakich elementów jest skonstruowany.
- Co to jest procesor. . .
- I jak działa
- O tym, że komputery potrzebują programów

Nie bardzo wiemy:

- Co to jest program
- Skąd się bierze
- Czy na pewno jest niezbędny. . .

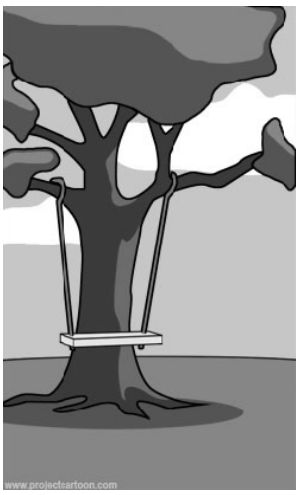
Tworzenie programów

… nie jest łatwe …

Na podstawie: http://www.projectcartoon.com/

Jak powstaje program…

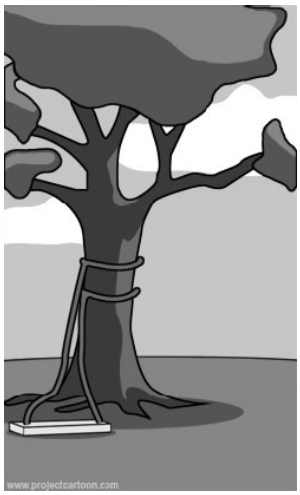
Tak opowiedział klient



Tak zrozumiał kierownik projektu

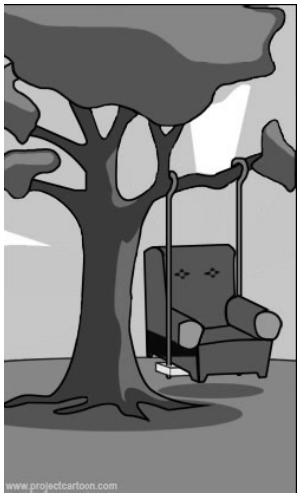
Tak zaprojektowali analitycy

Jak powstaje program…



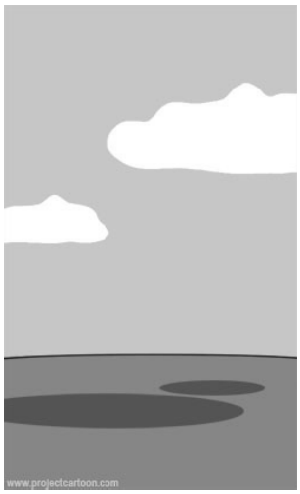
Tak zaprogramowali programiści

Tak zobaczyli to beta-testerzy

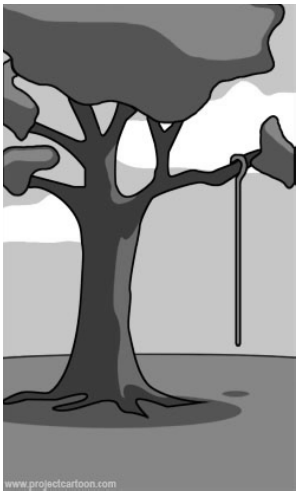


Tak przedstawia marketing

Jak powstaje program…



Tak wygląda dokumentacja

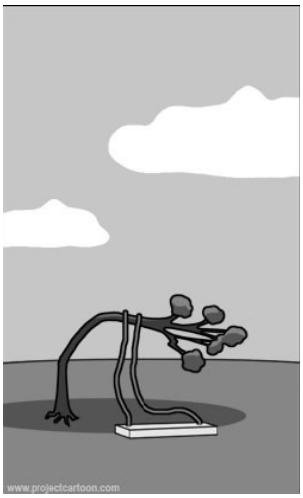


Tak wygląda po zainstalowaniu u klienta

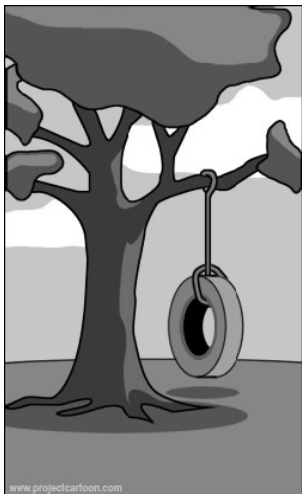
Za to zapłacił klient

Jak powstaje program…

Tak wygląda wsparcie



Tak wyglądają procedury awaryjne



Tego klient potrzebował

Kolejne kroki tworzenia programu

- Sformułowanie zadania
- Określenie danych wejściowych
- Określenie celu, czyli wyniku
- Poszukiwanie metody rozwiązania, czyli algorytmu
- Zapisanie algorytmu
  - opisu słownego
  - listy kroków
  - schematu blokowego
  - języka programowania
- Analiza poprawności rozwiązania
- Testowanie rozwiązania dla różnych danych.
- Ocena efektywności przyjętej metody.

Co to jest algorytm ?

Algorytm (potocznie) to przepis rozwiązania zadania, zawierający opis danych wraz z opisem czynności, które należy wykonać z tymi danymi, aby osiągnąć zamierzony cel w skończonym, ściśle określonym czasie.

Termin algorytm wywodzi się od zlatynizowanej formy (Algorismus, Algorithmus) nazwiska matematyka arabskiego z IX w., Al-Chuwarizmiego (أبو عبد الله محمد بن موسى الخوارزمي)

Algorytm cd.

- Algorytm jest pewną ściśle określoną procedurą obliczeniową, która dla zestawu właściwych danych wejściowych wytwarza żądane dane wyjściowe
- Algorytm jest to zbiór reguł postępowania umożliwiających rozwiązanie określonego zadania w skończonej liczbie kroków i w skończonym czasie.

Algorytm musi być …

Kompletny:

algorytm musi uwzględniać wszystkie możliwe przypadki, które mogą pojawić się podczas jego realizacji.

- uwzględnienie rożnych przypadków oznacza zapewnienie dalszej realizacji algorytmu, zgodnie z przewidzianymi na taką okoliczność instrukcjami. Oznacza to:
  - przewidzenie wystąpienia błędów numerycznych i logicznych
  - opracowanie systemu reakcji (komunikaty o błędach, odpowiednie zakończenie działania).

np. Obliczanie rozwiązań równania kwadratowego wymaga uwzględnienia przypadków: $b^2 - 4ac > 0$, $b^2 - 4ac < 0$. Czy to wystarczy?

Algorytm musi być …

Skończony:

algorytm musi zapewnić osiągnięcie rozwiązania w skończonej liczbie kroków (a więc też w skończonym czasie).

- Skończona liczba kroków nie oznacza, ze z góry wiadomo po ilu krokach algorytm się zakończy.
- Komunikat o błędzie lub braku rozwiązania też jest jednym z możliwych poprawnych zakończeń realizacji algorytmu.
- Algorytm musi posiadać warunek zakończenie operacji (np. kryterium dokładności) aby nie wykonywał się, mimo że poprawnie, to w nieskończoność.

Algorytm musi być …

Jednoznaczny:

dla tych samych danych wejściowych algorytm musi zawsze dawać te same wyniki.

- oznacza to niezależność działania programu od momentu jego wykonania, wpływu innych programów realizowanych równocześnie przez system operacyjny oraz, co najtrudniejsze, od sprzętu realizującego dany algorytm.
- np. algorytmy wykonujące obliczenia arytmetyczne powinny dawać dokładnie takie same wyniki - jest to bardzo trudne do spełnienia (różne kodowanie liczb, różne algorytmy ich przetwarzania)
- np. algorytmy formatujące tekst (procesory tekstu) powinny dawać taki sam wygląd strony (układ tekstu, łamanie wyrazów, etc.) zgodny z informacją zapisaną w pliku

Z czego składa się algorytm?

- Dane - obiekty podlegające przekształceniom podczas wykonywania algorytmu
- Wynik – ostateczny rezultat wykonania algorytmu
- Instrukcje – opis ciągu czynności, które muszą być wykonane w określonej kolejności

Algorytm zapisany przy pomocy języka programowania jest programem.

Algorytmy i łamigłówki

- A ma wyznaczyć wiek trójki dzieci kolegi B
- B informuje, że iloczyn wieku dzieci to 36
  - A zastanawia się i prosi o dodatkową informację
- B podaje sumę wieku dzieci
  - A zastanawia się, ale dalej nie może ustalić wieku dzieci, prosi o dodatkową informację
- B informuje, że najstarsze dziecko gra na gitarze
- A podaje rozwiązanie zagadki

Rozwiązanie

- Jakie są możliwe kombinacje dające iloczyn 36?

  (1,1,36) (1,2,18) (1,3,12) (1,4,9)
  (1,6,6) (2,2,9) (2,3,6) (3,3,4)

- Jakie są możliwe sumy wieku dzieci?

  1+1+36=38 1+2+18=21 1+3+12=16 1+4+9=14
  1+6+6=13 2+2+9=13 2+3+6=11 3+3+4=10

- Rozwiązaniem jest jedno z dwóch przypadków:

  1+6+6=13 2+2+9=13

- Jedno najstarsze dziecko jest w kombinacji (2,2,9)

Algorytmy i łamigłówki cd…

- A, B, C i D będą się ścigać
  - A przewidywał, że wygra B
  - D przewidywał, że B będzie ostatni
  - C przewidywał, że A będzie trzeci
  - D przewidywał, że A będzie miał rację
- Tylko jedno przewidywanie było trafne
- Było to przewidywanie zwycięzcy

- W jakiej kolejności A,B,C,D ukończyli wyścig?

Reprezentacja algorytmu

- Algorytmy powinny być tak przedstawiane, aby było możliwe ich jednoznaczne odczytanie i zastosowanie.

- Niektóre algorytmy można opisać w języku potocznym, zwłaszcza wtedy, gdy jego wykonawcą ma być człowiek

Sposoby zapisu algorytmów

- lista kroków - najbardziej naturalny sposób zapisu algorytmu,
- graficznie (tzw. schematy blokowe) - z użyciem symbolicznych elementów będących odpowiednikiem czynności,
- w pseudojęzyku programowania,
- w konkretnym języku np. C++, TP, Java, itp.

Przykład listy kroków

1. Wlać do garnka zimną wodę.
2. Zapalić gaz.
3. Gotować wodę do wrzenia.
4. Włożyć jajko.
5. Odczekać trzy minuty.
6. Zgasić gaz.
7. Wyjąć jajko

Lista kroków cd.

1. Podnieś słuchawkę.
2. Wybierz cyfrę 9.
3. Wybierz cyfrę 9.
4. Wybierz cyfrę 7.
5. Przekaż informacje.
6. Odłóż słuchawkę.

- Algorytmy liniowe mają opisy składające się z kroków, które nie zależą od żadnych warunków i są wykonywane w zapisanej kolejności.
- Istnieją jednak sytuacje, w których dalsze postępowanie w algorytmie zależy od spełnienia, bądź nie, określonych warunków.
- Czasami musimy powtórzyć pewne kroki algorytmu kilka razy.

Instrukcja warunkowa

Działa według jednego z dwóch przedstawionych schematów:

- Jeśli spełniony jest warunek *W*, wykonaj instrukcję *A*.
- Jeśli spełniony jest warunek W, to wykonaj instrukcję A; w przeciwnym razie wykonaj instrukcję B.

Instrukcja warunkowa cd.

- Instrukcje A i B opisują pojedynczą instrukcję lub instrukcję składającą się z ciągu instrukcji wykonywanych sekwencyjnie.
- Instrukcja warunkowa pozwala dokonać wyboru jednej z dwóch dalszych dróg wykonania algorytmu.

Lista kroków - instrukcja warunkowa

1. Podnieś słuchawkę.
2. Wybierz cyfrę 6.
3. Wybierz cyfrę 1.
4. Wybierz cyfrę 6.
5. Wybierz cyfrę 2.
6. Wybierz cyfrę 2.
7. Wybierz cyfrę 2.
8. Wybierz cyfrę 2.
9. Czy połączyłeś się z koleżanką ?
    A. Jeśli TAK, to przejdź do kroku 10.
    B. Jeśli NIE, to przejdź do kroku 11.
10. Zaproś koleżankę.
11. Odłóż słuchawkę.

Pętla

- wielokrotne powtarzanie niektórych instrukcji jest cechą charakterystyczną wielu algorytmów
- nie zawsze (tak jak w tym algorytmie) możemy określić dokładnie liczbę powtórzeń.
- liczba powtórzeń może zależeć od spełnienia pewnych warunków.
- wielokrotne powtarzanie instrukcji umożliwiają instrukcje iteracyjne (pętle). Działają one według schematu:

  Wykonuj instrukcję A dokładnie n razy.

Pętla

Iteracja to technika algorytmiczna polegająca na wykonaniu tej samej instrukcji dla n zmiennych.

Lista kroków - pętla

1. Podnieś słuchawkę.
2. Wybierz cyfrę 6.
3. Wybierz cyfrę 1.
4. Wybierz cyfrę 6.
5. Wykonaj czynność cztery razy
    A. Wybierz cyfrę 2.
6. Czy połączyłeś się z koleżanką ?
    A. Jeśli tak, to przejdź do kroku 7.
    B. Jeśli nie, to przejdź do kroku 8.
7. Zaproś koleżankę.
8. Odłóż słuchawkę.

Pętla

Powtarzamy wybieranie numeru aż do uzyskania połączenia. Dopiszemy w tym celu polecenie będące drugim rodzajem instrukcji iteracyjnej:

Powtarzaj wykonywanie instrukcji A aż do spełnienia warunku W.

Czym jest instrukcja A, czym warunek W ?

- Instrukcja A - podniesienie słuchawki, wybranie numeru
- Warunek W - uzyskanie połączenia z wybranym numerem

Lista kroków - pętla

1. Czy słuchawka jest odłożona ?
   A. Jeśli tak, to przejdź do kroku 2.
   B. Jeśli nie, to odłóż słuchawkę.
2. Podnieś słuchawkę.
3. Wybierz cyfrę 6.
4. Wybierz cyfrę 1.
5. Wybierz cyfrę 6.
6. Wykonaj czynność cztery razy
   A. Wybierz cyfrę 2.
7. Czy połączyłeś się z koleżanką ?
   A. Jeśli tak, to przejdź do kroku 8.
   B. Jeśli nie, to przejdź do kroku 9.
8. Zaproś koleżankę.
9. Odłóż słuchawkę.

Pętla

Jeśli okazałoby się, że nadal słychać w słuchawce sygnał zajętości linii czynność należałoby powtórzyć. Musielibyśmy wykonywać te czynności dopóki linia nie byłaby "czysta". W takim przypadku stosujemy instrukcję, która działa według schematu:

Dopóki warunek W jest spełniony, wykonuj instrukcję A.

Lista kroków - pętla

1. Czy słuchawka jest odłożona ?
    A. Jeśli tak, to przejdź do kroku 2.
    B. Jeśli nie, to odłóż słuchawkę.
2. Podnieś słuchawkę.
3. Czy linia jest zajęta ?
    A. Jeśli Tak, to:
        a. Odłóż słuchawkę.
        b. Podnieś słuchawkę.
        c. Przejdź do kroku 3.
    B. Jeśli Nie, to przejdź do kroku 4.
4. Wybierz cyfrę 6.
5. Wybierz cyfrę 1.
6. Wybierz cyfrę 6.
7. Wykonaj czynność cztery razy
    A. Wybierz cyfrę 2.
8. Czy połączyłeś się z koleżanką ?
    A. Jeśli tak, to przejdź do kroku 9.
    B. Jeśli nie, to przejdź do kroku 1.
9. Zaproś koleżankę.
10. Odłóż słuchawkę.

Schematy blokowe

W przypadku algorytmów skomplikowanych zapis w postaci języka potocznego będzie nieczytelny. Bardziej przejrzysty sposób – to schematy blokowe .

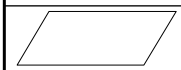
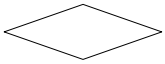
Schemat blokowy to graficzny zapis algorytmu rozwiązania zadania, przedstawiający opis i kolejność wykonywania czynności realizujących dany algorytm.

Schematy blokowe

W schemacie blokowym poszczególne operacje przedstawione są za pomocą odpowiednio połączonych skrzynek (klocków, bloków). Połączenia określają kolejność i sposób wykonywania operacji realizujących dany algorytm.

W literaturze informatycznej przyjęto pewne standardowe oznaczenia poszczególnych działań (są to figury geometryczne), ale można również używać innych oznaczeń (muszą one jednak być takie same dla określonego typu operacji).

Schematy blokowe

Symbol	Nazwa	Opis
	Początek, koniec	Oznaczenie miejsca rozpoczęcia i zakończenia algorytmu
	Operator	Działanie (operacja) do wykonania
	Operator wejścia/wyjścia	Wprowadzenie danych do pamięci lub ich wyprowadzenie
	Element decyzyjny	Operacja wyboru jednej z alternatywnych dróg działania
	Łącznik	Symbol połączenie dwóch fragmentów sieci działania
	Komentarz	Oznaczenie miejsca na komentarz
	Linia	Połączenie poszczególnych symboli działania

Schematy blokowe

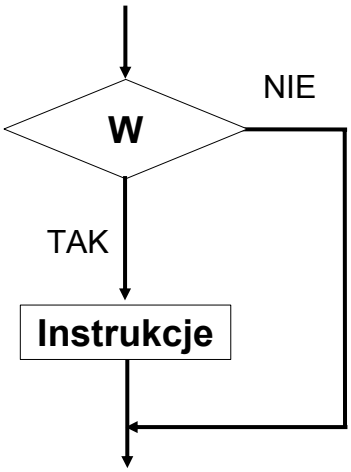
Na schemacie blokowym sytuacje warunkowe realizujemy przez skrzynkę warunkową.

W skrzynce wpisujemy warunek logiczny, stosując znaki:

- "=" równy,
- "<>" różny,
- "<" mniejszy,
- ">" większy,
- "<=" mniejszy lub równy,
- ">=" większy lub równy,

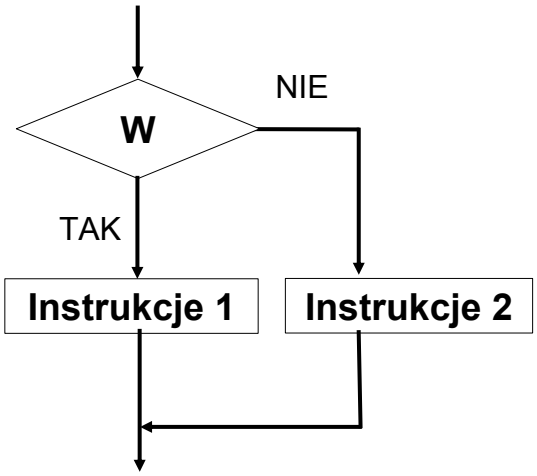
np: (a > 5) lub (a <= 20), (a < 5) OR (a <= 20)

Instrukcja warunkowa



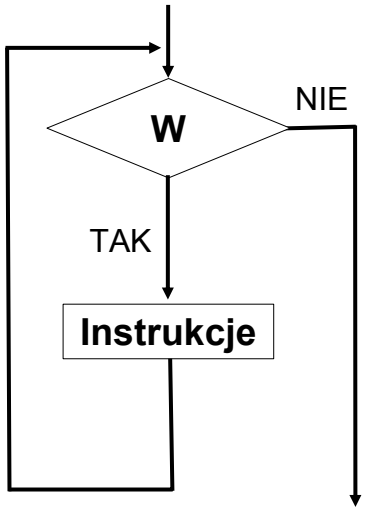
Instrukcje są realizowane gdy spełniony jest warunek W

Instrukcja warunkowa



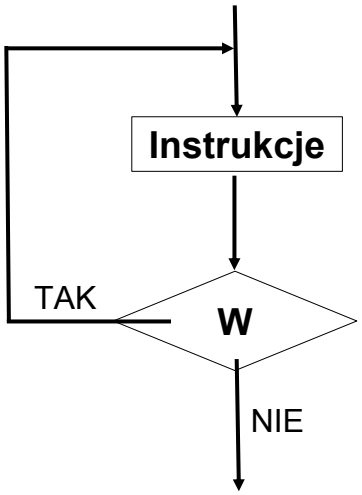
Jeżeli spełniony jest warunek W realizowane są Instrukcje 1, w przeciwnym wypadku realizowane są Instrukcje 2

Instrukcja pętli „dopóki”



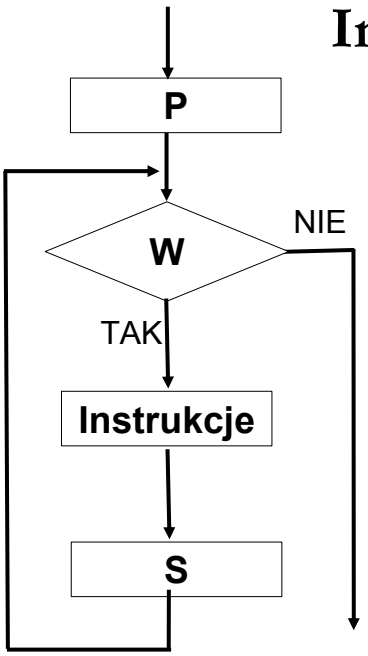
Dopóki spełniony jest warunek W, powtarzany jest ciąg instrukcji

Instrukcja pętli „aż do”



Powtarzaj wykonanie ciągu instrukcji aż do spełnienia warunku W

Instrukcja pętli „for”



Powtarzaj określoną ilość razy wykonanie ciągu instrukcji.

Liczba powtórzeń tych działań może być z góry określona lub zależeć od spełnienia warunku.

Klasyfikacja algorytmów (wybrane kategorie)

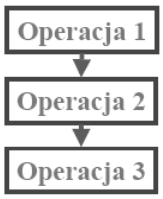
- algorytmy proste – rozgałęzione (nie występują albo występują rozgałęzienia),
- algorytmy cykliczne – mieszane (z powrotami albo bez powrotów),
- algorytmy równoległe – sekwencyjne
  - sekwencyjne – algorytmy, w których instrukcje wykonywane są w porządku, w jakim zostały wprowadzone;
  - niesekwencyjne (równoległe, współbieżne) – algorytmy, w których następstwo między pewnymi operacjami nie jest określone.

Klasyfikacja algorytmów (wybrane kategorie)

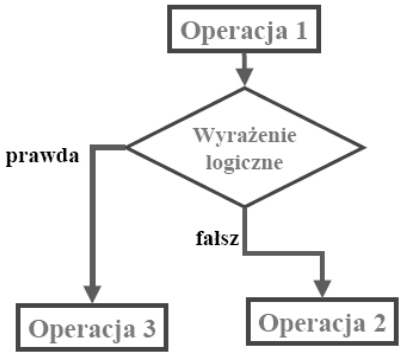
- algorytmy numeryczne - nienumeryczne (wykonywanie obliczeń lub przetwarzanie danych),
- algorytmy rekurencyjne – iteracyjne
  - iteracyjne – rodzaj algorytmu i programu, w których wielokrotnie wykonuje się pewne instrukcje, dopóki nie zostanie spełniony określony warunek;
  - rekurencyjne – takie procedury, które w swojej definicji posiadają wywołanie samej siebie;

Algorytm prosty a rozgałęziony

Proste:

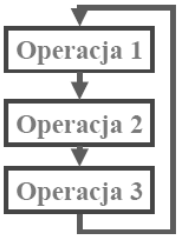


Rozgałęzione:

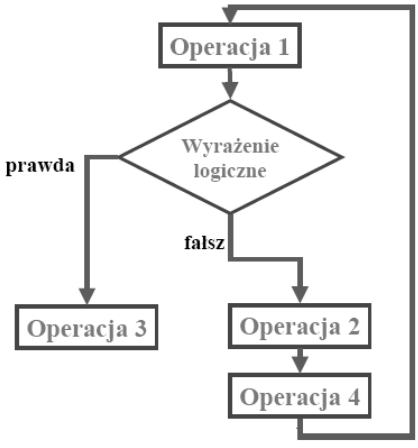


Algorytm cykliczny a mieszny

Cykliczne:

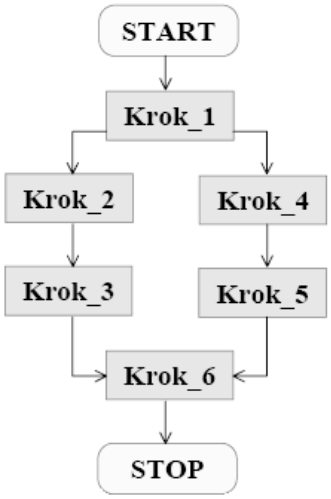


Mieszane:

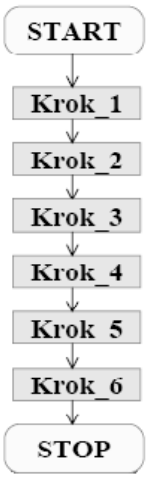


Algorytm równoległy a sekwencyjny

Równoległy:

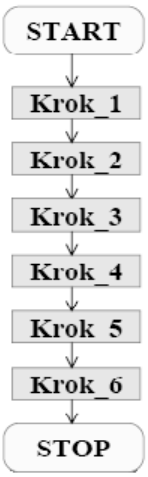


Sekwencyjny:

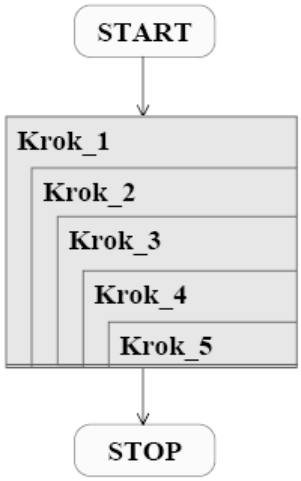


Algorytm iteracyjny a rekurencyjny

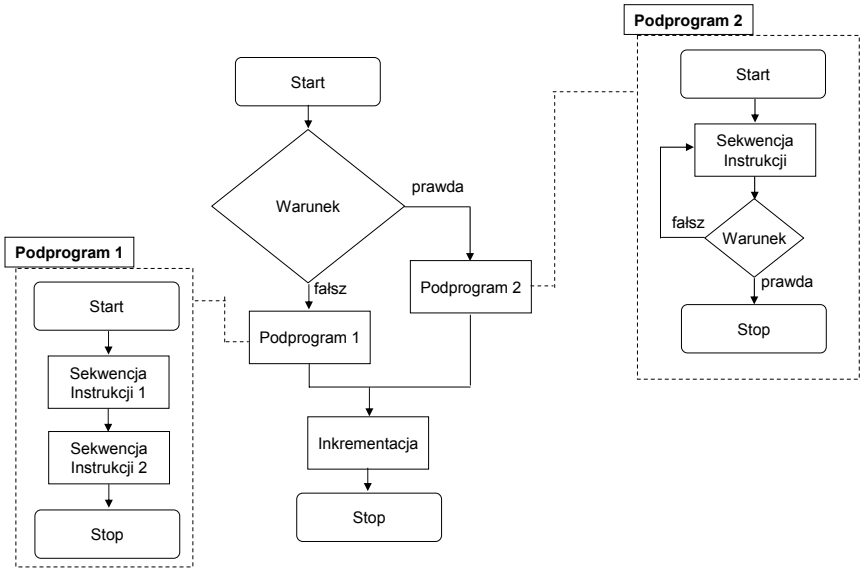
Iteracyjny:

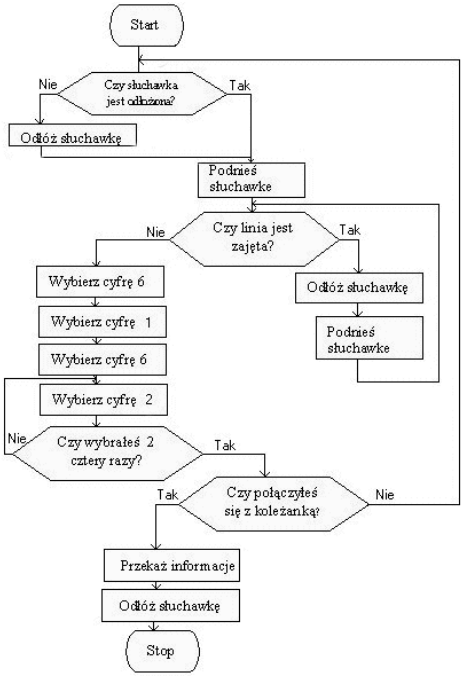


Rekurencyjny:



Logika rozgałęziania





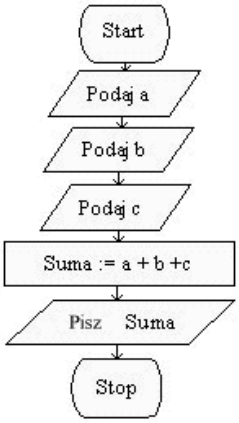
1.Czy słuchawka jest odłożona ?
   A.Jeśli tak, to przejdź do kroku 2.
   B.Jeśli nie, to odłóż słuchawkę.
2.Podnieś słuchawkę.
3.Czy linia jest zajęta ?
   A.Jeśli Tak, to:
      a.Odłóż słuchawkę.
      b.Podnieś słuchawkę.
      c.Przejdź do kroku 3.
   B.Jeśli Nie, to przejdź do kroku 4.
4.Wybierz cyfrę 6.
5.Wybierz cyfrę 1.
6.Wybierz cyfrę 6.
7.Wykonaj czynność cztery razy
   A.Wybierz cyfrę 2.
8.Czy połączyłeś się z koleżanką ?
   A.Jeśli tak, to przejdź do kroku 9.
   B.Jeśli nie, to przejdź do kroku 1.
9.Zaproś koleżankę.
10.Odłóż słuchawkę.

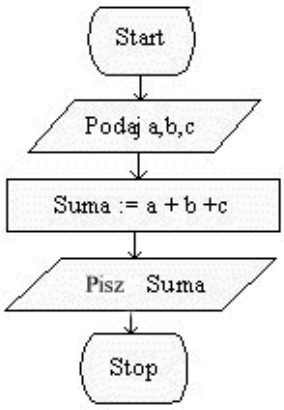
Przykład 1

Opracuj algorytm obliczający sumę 3 wprowadzonych z klawiatury liczb.

Algorytm w postaci ciągu kroków do wykonania:

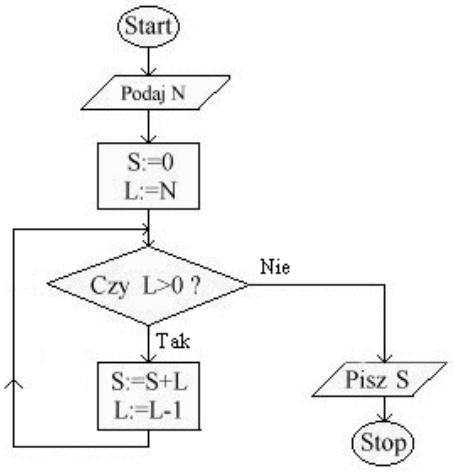
1. Podaj pierwszą liczbę
2. Podaj drugą liczbę
3. Podaj trzecią liczbę
4. Dodaj do siebie liczby i wynik zapamiętaj
5. Wypisz otrzymany wynik



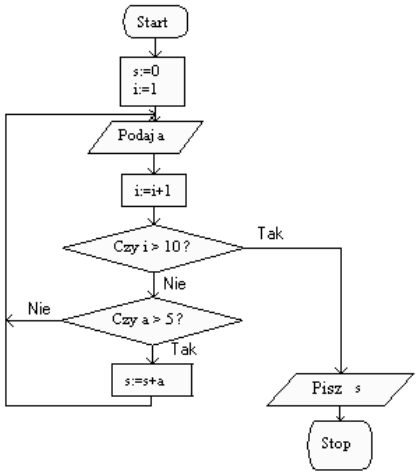


Podstawowe algorytmy

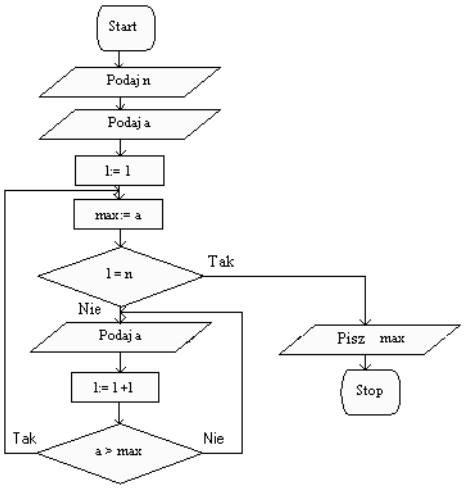
Algorytm sumujący N liczb



Algorytm sumujący liczby większe od 5 spośród 10 wprowadzonych.



Dany jest ciąg n-elementowy. Elementy tego ciągu są różne. Wskaż największy element ciągu.



Algorytm Euklidesa

Największy Wspólny Dzielnik (NWD) dwóch liczb jest największą liczbą naturalną spośród tych, które dzielą obie te liczby bez reszty.
Np. NWD(24,18) = 6.

Algorytm wyznaczania NWD podał i udowodnił jego poprawność już w starożytności grecki uczony Euklides.

Algorytm Euklidesa

W codziennej praktyce NWD służy nam do skracania ułamków do postaci właściwej: $^{12}/_{18} = {}^{2}/_{3.}$

Ponieważ największą liczbą naturalną dzielącą bez reszty liczby 12 oraz 18 jest 6, więc po podzieleniu licznika i mianownika ułamka przez NWD otrzymujemy jego postać właściwą, której dalej już nie można uprościć.

Dwie liczby nie posiadające wspólnego dzielnika różnego od 1 są względnie pierwsze.

Algorytm Euklidesa

Aby znaleźć NWD dwóch liczb, od większej należy odejmować mniejszą dotąd, aż obie liczby będą sobie równe. Wynik jest ich największym wspólnym podzielnikiem.

Obliczmy tym sposobem NWD liczb 24 i 15.
24 - 15 = 9
15 - 9 = 6
9 - 6 = 3
6 - 3 = 3

Para 3 i 3 - otrzymaliśmy równość, więc liczba 3 jest największym wspólnym podzielnikiem liczb 24 i 15.

NWD(24,15)=3

Algorytm Euklidesa

Dane wejściowe
a,b- liczby, dla których wyznaczamy NWD, $a,b \in N$

Dane wyjściowe
Liczba naturalna równa NWD(*a,b*)

Lista kroków ?

Schemat blokowy ?

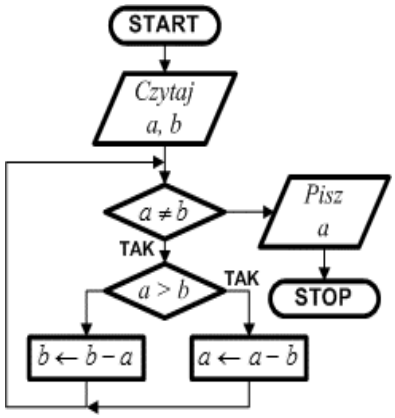
Algorytm Euklidesa

Lista kroków:

krok 1: Czytaj *a,b*

krok 2: Dopóki $a \neq b$, wykonuj krok 3.
Inaczej pisz *a* i zakończ algorytm

krok 3: Jeżeli $a > b$, to $a \leftarrow a - b$.
Inaczej $b \leftarrow b - a$



- Czy podany algorytm jest bezpieczny dla wszystkich możliwych wartości *a* i *b*?
- Co się stanie jeśli *a* lub *b* będzie równe 0?
- Jak zmodyfikowałbyś algorytm, aby uwzględniał takie przypadki?

Przeszukiwanie sekwencyjne

W *n* elementowym zbiorze wyszukać element o zadanych własnościach.

- Zadanie przeszukiwania sekwencyjnego polega na przeglądaniu kolejnych elementów zbioru.
- Znaleziony element zostaje zwrócony (zwykle interesuje nas nie sam element, ale jego pozycja w zbiorze)
- W przeciwnym wypadku algorytm zwraca informację, iż poszukiwanego elementu w zbiorze nie ma.

Przeszukiwanie sekwencyjne

Dane wejściowe

n - liczba elementów w zbiorze wejściowym, $i \in N$

d[ ] - zbiór wejściowy, w którym dokonujemy poszukiwań (Indeksy elementów rozpoczynają się od 1)

x - wartość poszukiwana

Dane wyjściowe

p- pozycja elementu *x* w zbiorze *d*[ ] (Jeśli $p = 0$, to element *x* w zbiorze nie występuje), $p \in C$

Lista kroków?

Schemat blokowy ?

Przeszukiwanie sekwencyjne

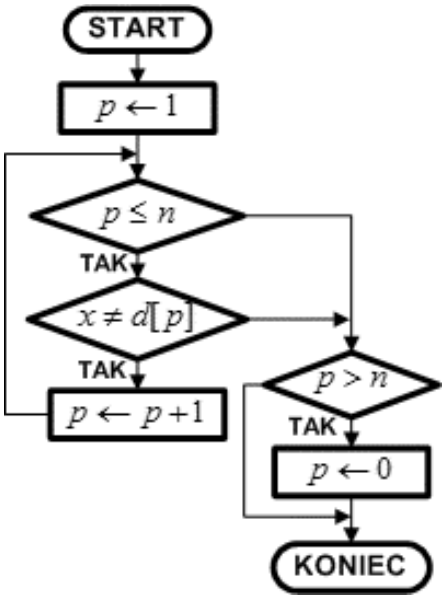
Lista kroków:

krok 1: $p \leftarrow 1$

krok 2: Dopóki ($p \le n$) i ($x \neq d[p]$) wykonuj $p \leftarrow p + 1$

krok 3: Jeśli $p > n$, to $p \leftarrow 0$

krok 4: Zakończ algorytm



Przeszukiwanie z wartownikiem

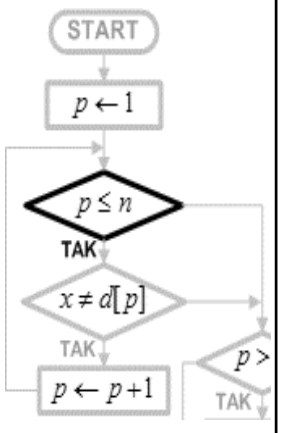
Warunek $p \leq n$ jest potrzebny tylko wtedy, gdy zbiór $d[\,]$ nie zawiera poszukiwanego elementu o wartości x (gwarantuje on zakończenie pętli)

Jeśli moglibyśmy zagwarantować, iż poszukiwany element zawsze zostanie znaleziony, to wtedy warunek ten stałby się zbędny.

Przeszukiwanie z wartownikiem

Jeśli dodamy na końcu zbioru poszukiwany element, to pętla przeszukująca zakończy się albo na elemencie x leżącym wewnątrz zbioru, albo na elemencie dodanym elemencie x:

- W pierwszym przypadku zmienna p będzie wskazywała pozycję znalezionego elementu i pozycja ta będzie mniejsza lub równa n
- Gdy element x w zbiorze nie występuje, zmienna p wskaże pozycję dodanego przez nas elementu, czyli $n + 1$.



Przeszukiwanie z wartownikiem

Dane wejściowe

n - liczba elementów w zbiorze wejściowym, $n \in N$

d[ ] - zbiór wejściowy, w którym dokonujemy poszukiwań (Indeksy elementów rozpoczynają się od 1) Zbiór musi posiadać miejsce na dodatkowy element, który zostanie dopisany na końcu

x - wartość poszukiwana

Dane wyjściowe

p - pozycja elementu *x* w zbiorze *d*[ ] (Jeśli *p* = 0, to element *x* w zbiorze nie występuje), $p \in C$

Lista kroków ?

Schemat blokowy ?

Przeszukiwanie z wartownikiem

Lista kroków:

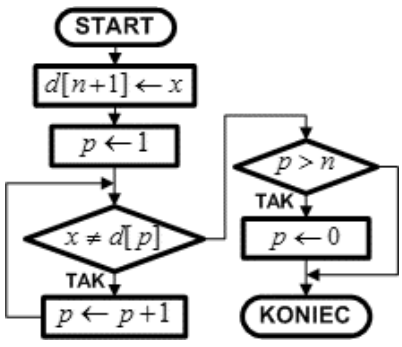
krok 1: $d[n + 1] \leftarrow x$

krok 2: $p \leftarrow 1$

krok 3: Dopóki ($x \neq d[p]$) wykonuj $p \leftarrow p + 1$

krok 4: Jeśli $p > n$, to $p \leftarrow 0$

krok 5: Zakończ algorytm



Wyszukiwanie elementu maksymalnego

W *n* elementowym zbiorze znaleźć element o największej wartości

- Algorytm może zwracać pozycję tego elementu, lub częściej tylko wartość (albo jedno i drugie).
- Bierzemy pierwszy element zbioru jako tymczasowy element maksymalny zapamiętując jego wartość oraz pozycję w zbiorze.
- Porównujemy go kolejno z pozostałymi elementami.
- Jeśli porównywany element zbioru jest większy od tymczasowego, to za nowy tymczasowy element maksymalny przyjmujemy element ze zbioru. Zapamiętujemy również pozycję tego elementu.
- Gdy porównamy wszystkie elementy zbioru, element tymczasowy stanie się rzeczywistym elementem maksymalnym.

Wyszukiwanie elementu maksymalnego

Dane wejściowe

n - liczba elementów w zbiorze wejściowym, $n \in N$, $n>0$
d[ ] - zbiór w którym poszukujemy elementu maksymalnego (Indeksy elementów rozpoczynają się od 1)

Dane wyjściowe

w_{max} - wartość wyznaczonego elementu maksymalnego
p_{max} - pozycja elementu maksymalnego, $p_{max} \in N$, $1 \le p_{max} \le n$

Zmienne pomocnicze

i - przechowuje indeksy porównywanych elementów; $i \in N$

Lista kroków ?

Schemat blokowy ?

Wyszukiwanie elementu maksymalnego

Lista kroków:

krok 1: $w_{max} \leftarrow d[1]$; $p_{max} \leftarrow 1$

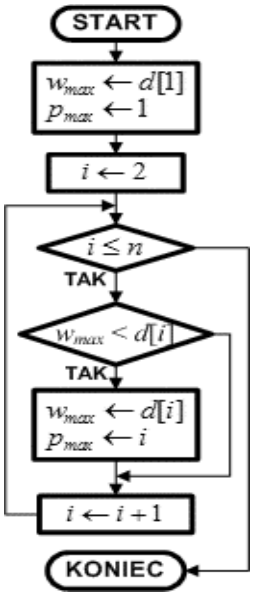
krok 2: Dla $i = 2,3,...,n$:

jeśli $w_{max} < d[i]$, to

$w_{max} \leftarrow d[i]$;

$p_{max} \leftarrow i$

krok 3: Zakończ algorytm



Wyszukiwanie najczęstszego elementu

W n elementowym zbiorze znaleźć element który powtarza się najczęściej

Podejście bezpośrednie:

Wybieramy ze zbioru kolejne elementy i zliczamy ich wystąpienia - wynikiem jest element, który występował najczęściej.

Wyszukiwanie najczęstszego elementu

Dane wejściowe

n - liczba elementów w zbiorze wejściowym, $n \in N$, $n>0$
$d[\ ]$ - zbiór w którym poszukujemy najczęstszego elementu

Dane wyjściowe

w_n - wartość elementu powtarzającego się najczęściej
p_n - pierwsza pozycja elementu najczęstszego, $p_n \in N$, $1 \le p_n \le n$
l_n - liczba wystąpień elementu najczęstszego, $l_n \in N$, $1 \le l_n \le n$

Zmienne pomocnicze

i, j – zmienne licznikowe pętli $i, j \in N$
licznik – licznik wystąpień elementu

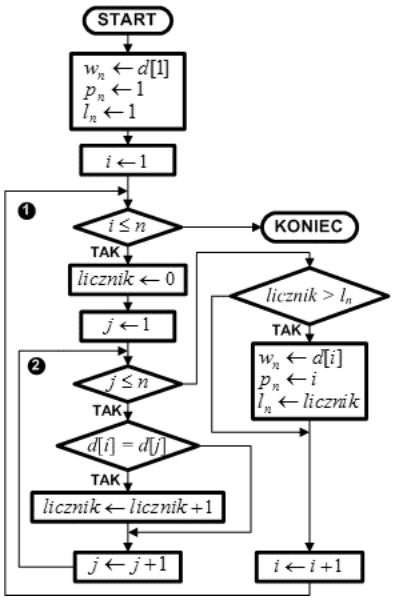
Lista kroków ?

Schemat blokowy ?

Wyszukiwanie najczęstszego elementu

Lista kroków:

krok 1: $w_n \leftarrow d[1]$; $p_n \leftarrow 1$; $l_n \leftarrow 1$
krok 2: Dla $i = 1,2,...,n$: wykonuj kroki 3...5
krok 3: *licznik* ← 0
krok 4: Dla $j = 1,2,...,n$:
jeśli $d[i] = d[j]$, to
licznik ← *licznik* + 1
krok 5: Jeśli *licznik* > l_n , to
$w_n \leftarrow d[i]$;
$p_n \leftarrow i$;
$l_n \leftarrow$ *licznik*
krok 6: Zakończ algorytm



Liczby pierwsze – sito Erastotenesa

- W czasach starożytnych grecki uczony Eratostenes z Cyreny zaproponował wyrzucanie ze zbioru liczb naturalnych wielokrotności kolejnych liczb, które nie zostały wcześniej wyrzucone.
- To, co zostanie, będzie zbiorem liczb pierwszych, które nie posiadają innych podzielników jak 1 i same siebie.
- Metoda ta została nazwana sitem Eratostenesa i jest najszybszą metodą wyszukiwania liczb pierwszych w ograniczonym zbiorze.

Liczby pierwsze – sito Erastotenesa

Znajdź wszystkie liczby pierwsze w zbiorze 20 początkowych liczb naturalnych:
{ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 }

- Najpierw usuniemy ze zbioru liczbę 1
- Bierzemy wolną liczbę 2 i usuwamy ze zbioru wszystkie jej wielokrotności (liczby parzyste):
  { 2 3 5 7 9 11 13 15 17 19 }
- Następną wolną liczbą jest 3 (pozostaną liczby niepodzielne przez 2 i przez 3):{ 2 3 5 7 11 13 17 19 }
- W zbiorze pozostały same liczby pierwsze.

Liczby pierwsze – sito Erastotenesa

Dane wejściowe

g - górny kres przedziału, w którym poszukujemy liczb pierwszych, $g \in$ N

Dane wyjściowe

Kolejne liczby pierwsze zawarte w przedziale od 2 do *g*.

Zmienne pomocnicze

i – służy do sterowania pętlami iteracyjnymi, $i \in$ N

t[ ] – tablica logiczna odwzorowująca zbiór liczbowy (Indeksy elementów przebiegają wartości od 2 do g; liczbę określa indeks elementu tablicy, np. *t*[5] = *true* - liczba 5 jest w zbiorze; początkowo wszystkie liczby są zbiorze)

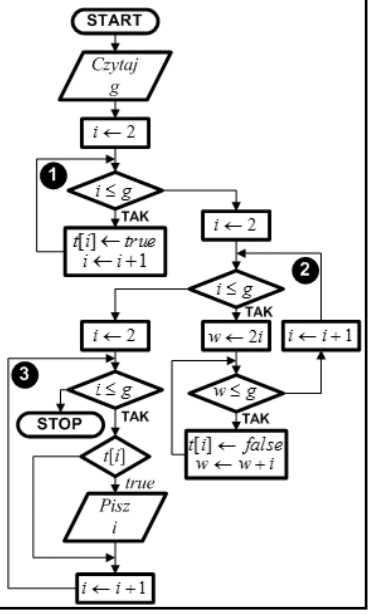
w – służy do tworzenia kolejnych wielokrotności, $w \in$ N

Lista kroków ?
Schemat blokowy ?

Liczby pierwsze – sito Erastotenesa

Lista kroków:

krok 1: Czytaj *g*
krok 2: Dla *i* = 2,3,...,*g* wykonuj *t*[*i*] ← *true*
krok 3: Dla *i* = 2,3,...,*g* wykonuj kroki 4...7
krok 4: *w* ← 2*i*
krok 5: Dopóki *w* ≤ *g* wykonuj kroki 6,7.
Inaczej wykonaj następny obieg pętli z kroku 3
krok 6: *t*[*w*] ← *false*
krok 7: *w* ← *w* + *i*
krok 8: Dla *i* = 2,3,...,*g* jeżeli *t*[*i*] = *true*, to pisz *i*
krok 9: Zakończ algorytm



Sortowanie bąbelkowe

- Algorytm sortowania bąbelkowego jest jednym z najstarszych algorytmów sortujących.
- Zasada działania opiera się na cyklicznym porównywaniu par sąsiadujących elementów i zamienianie ich kolejności w przypadku niespełnienia kryterium porządkowego zbioru.
- Operację tę wykonujemy dotąd, aż cały zbiór zostanie posortowany.

Sortowanie bąbelkowe

Przykład
Należy posortować 5-cio elementowy zbiór liczb {5 4 3 2 1}, który wstępnie jest posortowany w kierunku odwrotnym.

5 4 3 2 1	4 3 2 1 5	3 2 1 4 5	2 1 3 4 5
4 5 3 2 1	3 4 2 1 5	2 3 1 4 5	1 2 3 4 5
4 3 5 2 1	3 2 4 1 5	2 1 3 4 5	1 2 3 4 5
4 3 2 5 1	3 2 1 4 5 .	2 1 3 4 5	1 2 3 4 5
4 3 2 1 5	3 2 1 4 5	2 1 3 4 5	1 2 3 4 5
Stan po pierwszym obiegu.	Stan po drugim obiegu.	Stan po trzecim obiegu.	

.

Sortowanie bąbelkowe

Dane wejściowe

n - liczba elementów w przeszukiwanym zbiorze, $n \in N$

d[ ] – zbiór *n*-elementowy, który będzie sortowany
(elementy zbioru mają indeksy od 1 do *n*)

Dane wyjściowe

d[ ]- posortowany zbiór *n*-elementowy

Zmienne pomocnicze

i,j – zmienne sterujące pętli $i,j \in N$

Lista kroków ?

Schemat blokowy ?

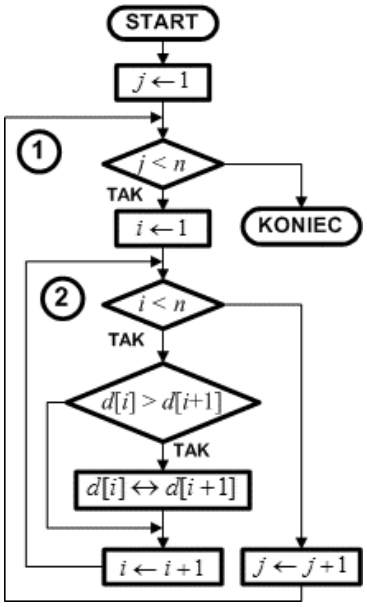
Sortowanie bąbelkowe

Lista kroków:

krok 1: Dla *j* = 1,2,...,*n* - 1:
wykonuj krok 2

krok 2: Dla *i* = 1,2,...,*n* - 1:
jeśli *d*[*i*] > *d*[*i* + 1],
to zamień *d*[*i*] *d*[*i* + 1]

krok 3: Zakończ algorytm



Sortowanie przez wybór

Załóżmy, iż chcemy posortować zbiór liczbowy rosnąco.

Element najmniejszy powinien znaleźć się na pierwszej pozycji.

- Szukamy w zbiorze elementu najmniejszego i wymieniamy go z elementem na pierwszej pozycji. W ten sposób element najmniejszy znajdzie się na swojej docelowej pozycji.
- W identyczny sposób postępujemy z resztą elementów należących do zbioru.

Sortowanie przez wybór

Przykład
Należy posortować zbiór liczb {4 7 2 9 3}.

4 7 **2** 9 3	Wyszukujemy najmniejszy element w zbiorze. Jest nim liczba 2.
2 7 4 9 3	Znaleziony element minimalny wymieniamy z pierwszym elementem zbioru - liczbą 4
2 7 4 9 **3**	Wśród pozostałych elementów wyszukujemy element najmniejszy. Jest nim liczba 3.
2 3 4 9 7	Znaleziony element minimalny wymieniamy z drugim elementem zbioru - liczbą 7.
2 3 4 9 7	Znajdujemy kolejny element minimalny - liczbę 4.
2 3 4 9 7	Element ten nie zmienia zatem swojej pozycji w zbiorze.
2 3 4 9 **7**	Znajdujemy kolejny element minimalny - liczbę 7.
2 3 4 7 9	Wymieniamy go z liczbą 9
2 3 4 7 9	Sortowanie skończone

Sortowanie przez wybór

Dane wejściowe

n - liczba elementów w przeszukiwanym zbiorze, $n \in N$

d[ ] – zbiór *n*-elementowy, który będzie sortowany
(elementy zbioru mają indeksy od 1 do *n*)

Dane wyjściowe

d[ ]- posortowany zbiór *n* - elementowy

Zmienne pomocnicze

i,j – zmienne sterujące pętli $i,j \in N$

p_{min}- pozycja elementu minimalnego w zbiorze *d*[ ], $p_{min} \in N$

Lista kroków ?

Schemat blokowy ?

Sortowanie przez wybór

Lista kroków:

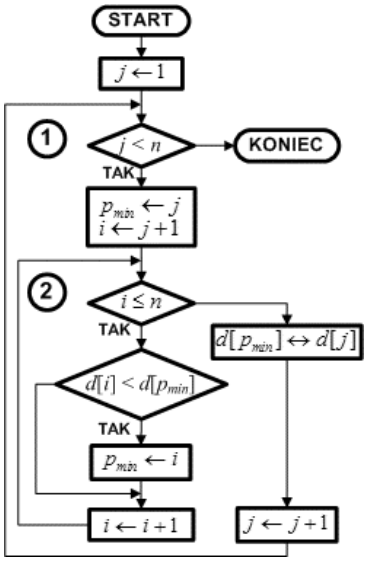
krok 1: Dla $j = 1, 2, ..., n - 1$:
wykonuj kroki 2...4

krok 2: $p_{min} \leftarrow j$

krok 3: Dla $i = j + 1,\ j + 2, ..., n$:
jeśli $d[i] < d[p_{min}]$, to $p_{min} \leftarrow i$

krok 4: $d[j] \leftarrow d[p_{min}]$

krok 5: Zakończ algorytm



Rekurencja

Rekurencja, rekursja (ang. recursion) - cecha algorytmu, polegająca na tym, że w którymś kroku algorytmu następuje odwołanie do całego algorytmu.

Obiekt rekurencyjnym – obiekt, który częściowo składa się z siebie samego lub jego definicja odwołuje się do jego samego (np. płatek śniegu, kalafior, fraktale).

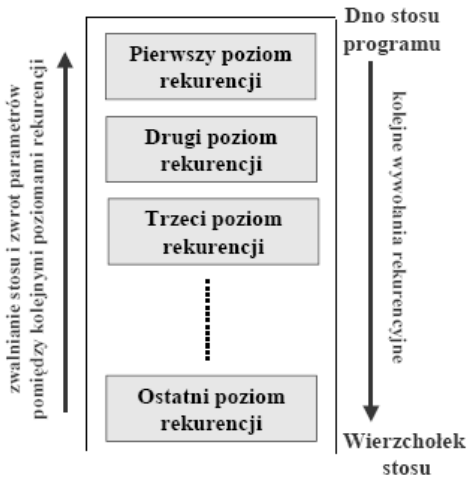
Funkcja (procedura) jest rekurencyjna, jeśli wywołuje sama siebie.

Rekurencja

Ważne jest, aby kolejne wywołania funkcji (procedury) rekurencyjnej były realizowane dla kolejnych wartości parametrów formalnych w taki sposób, aby nie doszło do zjawiska „nieskończonej pętli rekurencyjnych wywołań funkcji”

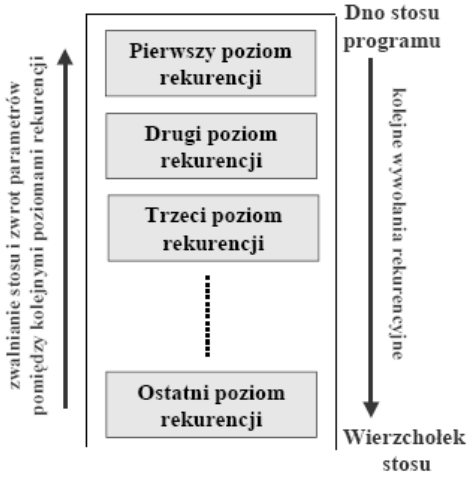
Wywołanie funkcji rekurencyjnej

- Kolejne wywołania funkcji rekurencyjnej są związane z odkładaniem na stosie programu kolejnych rekordów aktywacji procedury
- W wyniku kończenia działania poszczególnych funkcji na kolejnych poziomach rekurencji kolejne rekordy aktywacji są zdejmowane ze stosu



Wywołanie funkcji rekurencyjnej

- Kolejne wywołania funkcji rekurencyjnej są związane z odkładaniem na stosie programu kolejnych rekordów aktywacji procedury
- W wyniku kończenia działania poszczególnych funkcji na kolejnych poziomach rekurencji kolejne rekordy aktywacji są zdejmowane ze stosu



Definicja rekurencja

Definicja rekurencyjna składa się z dwóch części:

- W pierwszej, zwanej podstawową lub warunkiem początkowym, są wyliczone elementy podstawowe, stanowiące części składowe wszystkich pozostałych elementów zbioru.
- W drugiej części, zwanej krokiem indukcyjnym, są podane reguły umożliwiające konstruowanie nowych obiektów z elementów podstawowych lub obiektów zbudowanych wcześniej. Reguły te można stosować wielokrotnie, tworząc nowe obiekty.

Rekurencyjne obliczanie silnii

Silnię liczby n należącej do zbioru liczb naturalnych definiujemy następująco:

$$n! = 1 \times 2 \times 3 \times \ldots \times (n-1) \times n$$

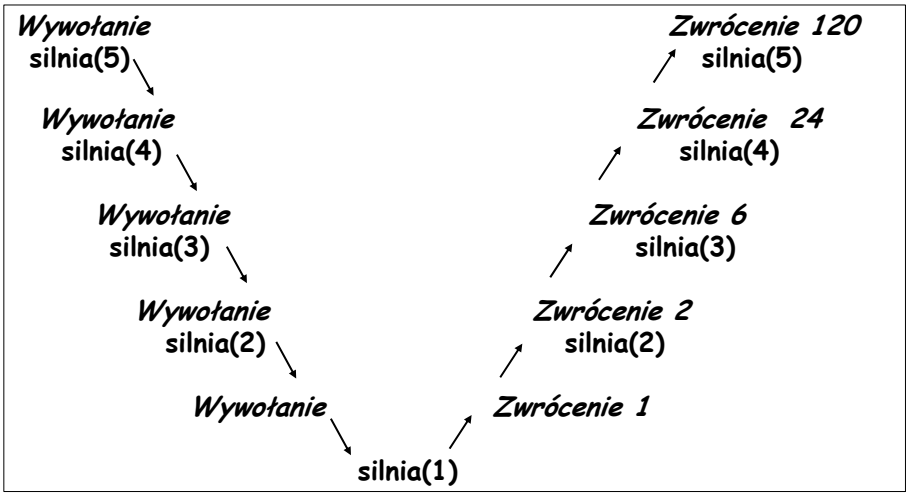
Rekurencyjna definicja funkcji silnia:

$$n! = \begin{cases} 1, & \text{jeśli } n = 0 \text{ (podstawa)} \\ n \times (n-1)! & \text{jeśli } n > 0 \text{ (indukcja)} \end{cases}$$

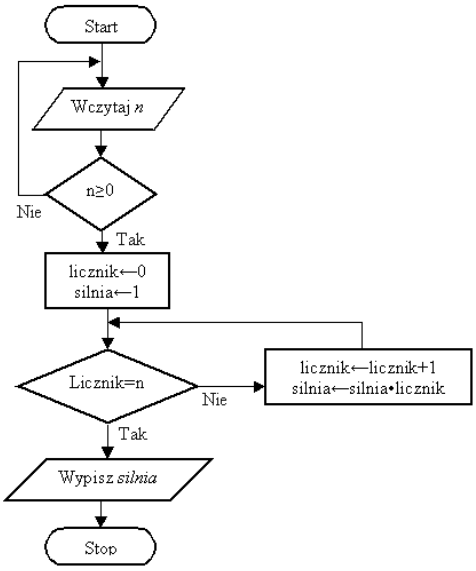
Przykład: $5! = 5 \times 4! = 5 \times 4 \times 3! = 5 \times 4 \times 3 \times 2! = 5 \times 4 \times 3 \times 2 \times 1! = 120$

Rekurencyjne obliczanie silnii

krok 1: Jeśli n < 2, silnia(n) ← 1 i zakończ algorytm
krok 2: silnia(n) ← n x silnia(n - 1) i zakończ algorytm



Rekurencyjne obliczanie silnii



Literatura

- Cormen T.H., Leiserson Ch.E., Rivest R.L., Stein C.: „*Wprowadzenie do algorytmow*". WNT, Warszawa, 2005
- Wroblewski P.: „*Algorytmy, struktury danych i techniki programowania, Wydanie III*", Helion, Gliwice, 2003
- N. Wirth: „*Algorytmy + struktury danych = programy*". WNT, Warszawa, 2004.
- http://we.pb.edu.pl/~jforenc/dydaktyka.html
- http://pl.wikipedia.org