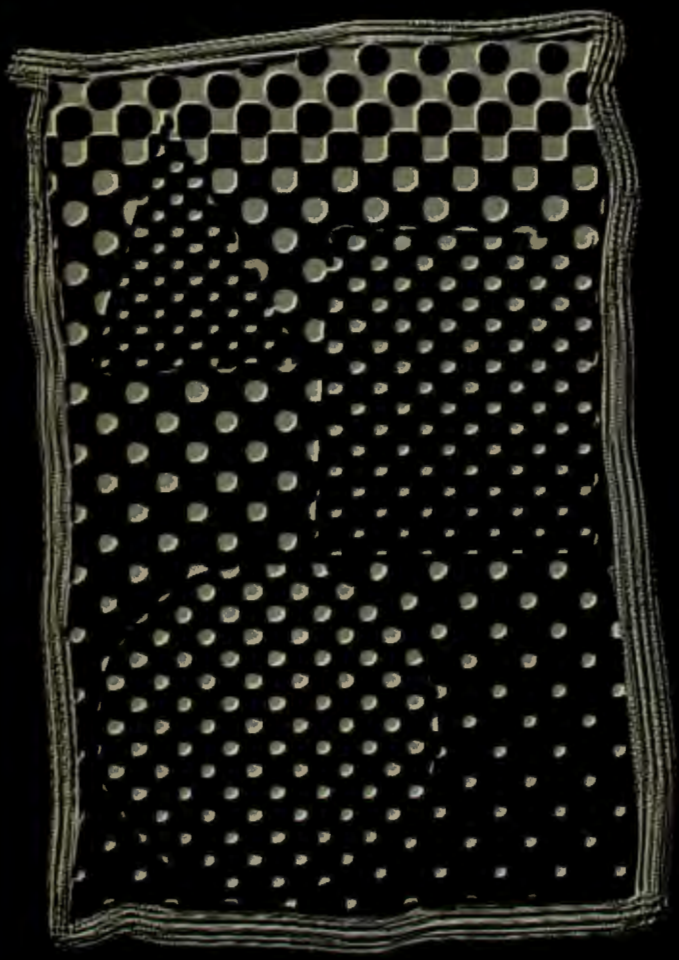
WYŻSZA SZKOŁA
INFORMATYKI STOSOWANEJ
I ZARZĄDZANIA



Henryk Spustek

ELEMENTY INFORMATYKI

WARSZAWA 2000

18.-

Seria: Skrypty WSISiZ

Skrypt zgłoszony przez
Dziekana Wydziału Zarządzania i Marketingu
dr Barbarę Mażbic-Kulmę

WYŻSZA SZKOŁA
INFORMATYKI STOSOWANEJ I ZARZĄDZANIA

Henryk Spustek

ELEMENTY INFORMATYKI

Warszawa 2000

ISBN 83-88311-17-4

Projekt graficzny okładki: Jan Młynarczyk

Druk:
Zakład Poligraficzny Jerzy Kosiński
Warszawa

5. ALGORYTMY BUDOWA I PRZYKŁADY

Głównym zastosowaniem komputera jako maszyny liczącej jest rozwiązywanie problemów dających się zapisać w postaci algorytmu. Algorytm jest "przepisem" na rozwiązanie danego problemu. W literaturze można znaleźć wiele definicji słowa algorytm, jednakże wszystkie one odzwierciedlają sobą to co powiedziano powyżej[16]. Przytoczymy kilka z nich.

Definicje cybernetyczne

- Algorytm stanowi dokładny i szczegółowy przepis jednoznacznie określający rodzaj i kolejność działań niezbędnych dla uzyskania wymaganych wyników.

- Algorytm jest dokładnym przepisem wykonania skończonej ilości operacji w określonym porządku, pozwalającym na rozwiązanie każdego zadania danego typu.

Definicja matematyczna

- Algorytm jest regułą przekształcania wyrażeń matematycznych przez powtarzanie tych samych działań na kolejno otrzymywanych wynikach działań poprzednich.

Słowo *algorytm* pochodzi od nazwiska matematyka perskiego Muchameda Ibu Al Chrozemiego (nazwisko to w formie zlatynizowanej brzmi Algorismus, Algorithmus[17]).

---

[16] P.Wróblewski Algorytmy struktury danych i techniki programowania, Wydawnictwo Helion 1997

[17] Nowa Encyklopedia Powszechna PWN, Warszawa 1994

Każdy algorytm, będący zbiorem reguł postępowania w celu rozwiązania danego problemu powinien posiadać szereg cech wśród których na szczególna uwagę zasługują:

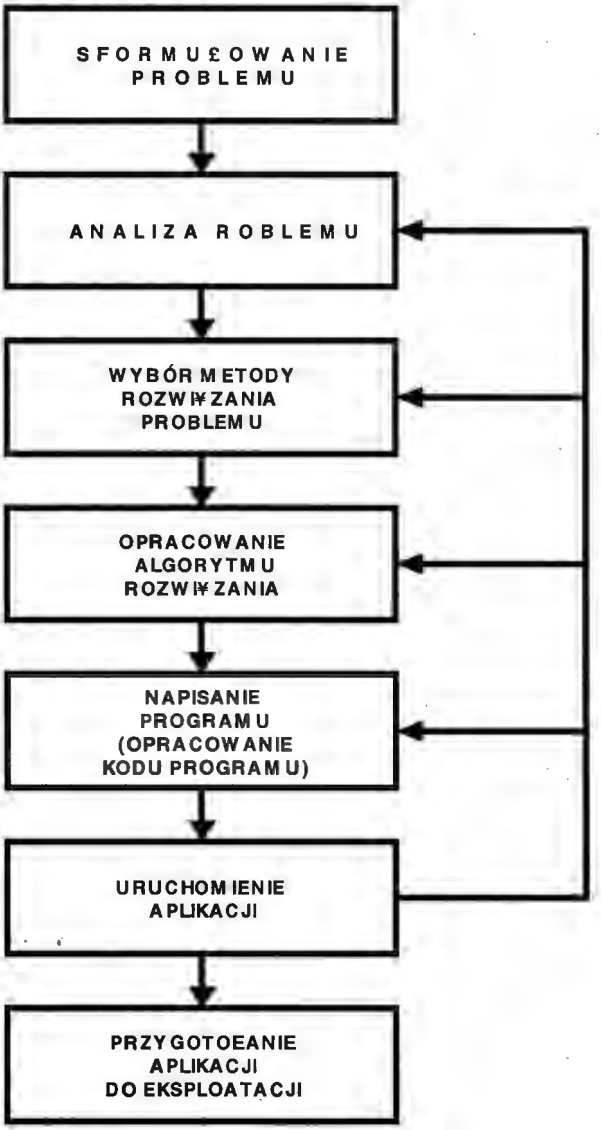
- *określoność* - rozumiana w ten sposób, że znane są wszystkie przypadki jakie mogą zaistnieć w czasie realizacji algorytmu,
- *skończoność* - wszystkie obliczenia zawarte w algorytmie powinny zostać zrealizowane w skończonej liczbie kroków,
- *wykonalność* - zawiera się w dobrym (właściwym) zdefiniowaniu poszczególnych kroków algorytmu, tak aby możliwym było jego efektywne wykonanie.

Podczas budowy algorytmów należy brać pod uwagę następujące zasady:

- prostoty budowy,
- minimalizacji liczby wykonywanych działań,
- maksymalizacji dokładności przeprowadzanych obliczeń.

Sam proces budowy algorytmu polega na dobrym (właściwym) wyodrębnieniu etapów budowy, doskonałym ich opisaniu (w sensie podanym wyżej) oraz określeniu porządku w jakim mają być realizowane. Nieodzowną czynnością - już po napisaniu algorytmu - jest sprawdzenie, czy algorytm właściwie realizuje założone zadania, czy wyniki uzyskane przy jego pomocy są dokładne a przebieg wykonywanych operacji jest optymalny.

Główne etapy budowy algorytmu przedstawia rys.11.



Rys.11. Główne etapy budowy algorytmu.

źródło: dzięki uprzejmości dr R.Wieleby

Krótko opiszemy wymienione powyżej etapy budowy algorytmu.

Sformułowanie problemu jest tym etapem na którym następuje sprecyzowanie celu działania algorytmu. Na tym etapie należy odpowiedzieć na podstawowe pytania dotyczące założeń, rodzaju danych wejściowych i wyjściowych oraz formy przedstawienia wyników.

Analiza problemu polega na zbudowaniu lub znalezieniu (dobraniu) wśród już istniejących, modelu matematycznego opisującego w zadowalający sposób problem sformułowany na etapie pierwszym.

Wybór metody rozwiązania problemu - etap na którym decydujemy jaka metoda zostanie zastosowana w opracowywanym algorytmie. Zwykle istnieje więcej niż jedna metoda rozwiązania danego problemu (patrz algorytm znajdowania $\sqrt{2}$ - przedstawiony w rozdziale 5.4 – *przykład 25*). Przeważnie o wyborze metody rozwiązania problemu decyduje to, która z metod dostarcza wyników obarczonych najmniejszym błędem. Zatem, właściwą metodę wybieramy kierując się wiedzą i doświadczeniem oraz informacjami zawartymi w literaturze przedmiotu.

Opracowanie algorytmu rozwiązania - opracowanie krok po kroku kolejności wykonywanych działań.

Napisanie programu - polega na zapisaniu opracowanego algorytmu w kodzie maszynowym adekwatnym dla konkretnego języka programowania.

Uruchomienie aplikacji - sprawdzenie czy napisany program działa prawidłowo, czyli porównanie wyników uzyskanych w wyniku działania aplikacji z wynikami działania algorytmu. W przypadku niezgodności należy szukać błędów, stąd strzałki "do góry" na rys.11. Etap ten jest bardzo ważny. Dobre (solidne) testowanie aplikacji "procentuje" w późniejszej jej eksploatacji.

Przygotowanie aplikacji do eksploatacji - etap wdrożeniowy, wiąże się z opracowaniem dokumentacji do istniejącej, sprawnie działającej aplikacji.

Wśród algorytmów rozróżnia się (ze względu na budowę) następujące ich rodzaje:

- liniowy,
- rozgałęziony,
- cykliczny,
- proceduralny,
- rekurencyjny.

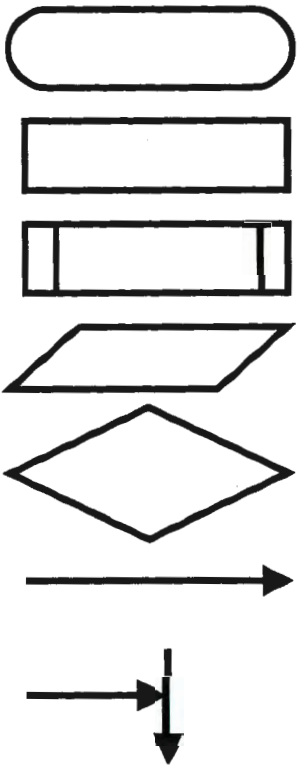
5.1 NOTACJE ZAPISU ALGORYTMÓW

Algorytmy można zapisywać na różne sposoby. W zasadzie, sposób zapisu algorytmu nie jest istotny. Ważne jest aby algorytm był optymalny, tak ze względu na czas wykonania jak i jakość uzyskiwanych wyników. Nie mniej jednak, należy zadbać o to aby zapis algorytmu był czytelny. Ułatwi to dalsze prace jak również spowoduje, że na ewentualną powtórną analizę problemu spożytkujemy mniej czasu.

Wyróżnia się następujące, podstawowe notacje zapisu algorytmów. Są to:

- język naturalny,
- sieci działań,
- diagramy strukturalne (schematy blokowe),
- pseudo - kod,
- języki programowania.

PODSTAWOWE SYMBOLE UŻYWANE PRZY KONSTRUOWANIU SIECI DZIAŁAŃ



Początek / Koniec

Przetwarzanie

Proces uprzednio zdefiniowany

WE / WY

Decyzja

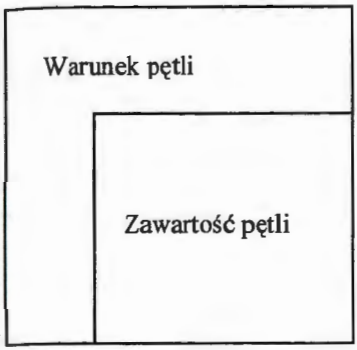
Kierunek przepływu danych

Łączenie dróg przepływu danych

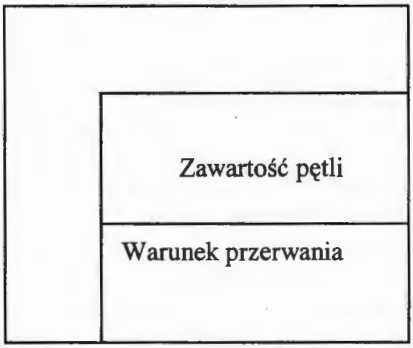
PODSTAWOWE SYMBOLE UŻYWANE PRZY KONSTRUOWANIU SCHEMATÓW BLOKOWYCH (DIAGRAMÓW STRUKTURALNYCH)[18]

W podstawowych blokach diagramów strukturalnych znajdziemy między innymi pętle programowe realizowane na dwa sposoby (patrz poniżej A i B) oraz proste sprawdzenie warunku, dające w rezultacie dwie możliwości odpowiedzi "tak" oraz "nie" (patrz poniżej C).

A

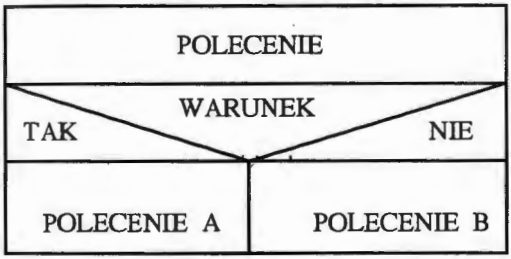


B.



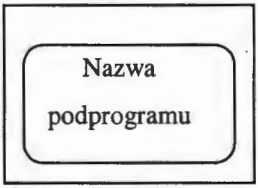
Pętla programowa z warunkiem przerwania umieszczonym:
A. na początku B. Na końcu

C.



Sprawdzenie warunku - dwa ciągi logiczne w przypadku prawdy i fałszu

D.



Wywołanie podprogramu

[18] H.Feichtinger Mikrokomputery poradnik, WKŁ Warszawa 1988

5.2 ELEMENTY SKŁADNIOWE PSEUDO-KODU

Algorytmy można zapisywać na wiele sposobów. Między innymi można je formułować przy użyciu języka naturalnego bądź języka programowania. Pseudokod jest narzędziem częściowo sformalizowanym, gdzie obok pewnych ustalonych struktur można używać języka naturalnego[19].

Pseudo - kod zawiera następujące elementy zwane zdaniami:

1) zdanie określające czynność - zdanie zapisane w języku naturanym,

2) zdanie decyzyjne "jeśli" (pozwala na podejmowanie decyzji):

a) proste,
jeśli **warunek** *to*
zdanie
b) z alternatywą.
jeśli **warunek** *to*
zdanie_1
w przeciwnym wypadku
zdanie_2

3) zdanie iteracyjne "podczas gdy" (zdanie to pozwala na wykonywanie danych czynności dopóki spełniony jest zadany warunek)
podczas gdy **warunek** *wykonuj*
zdanie

4) zdanie iteracyjne "dla" (zadanie to umożliwia wykonywanie działań określoną liczbę razy lub dla kolejnych wartości z danego zbioru)
dla **lista sytuacji**
wykonuj
zdanie

---

[19] A.Struzińska-Walczak, K.Walczak Nauka programowania dla początkujących Turbo Pascal, Wydawnictwo W & W Warszawa 1993

5) zdanie pozwalające na wybór

wybierz
przypadek *warunek_1*

zdanie_1
przypadek *warunek_2*

zdanie_2

.....

przypadek *warunek_n*

zdanie_n
w przeciwnym wypadku
zdanie_awaryjne
koniec wyboru

6) zdanie grupujące (jest to zdanie zawierające w sobie kilka zdań zgrupowanych nawiasami klamrowymi)

{

zdanie_1

zdanie_2

....

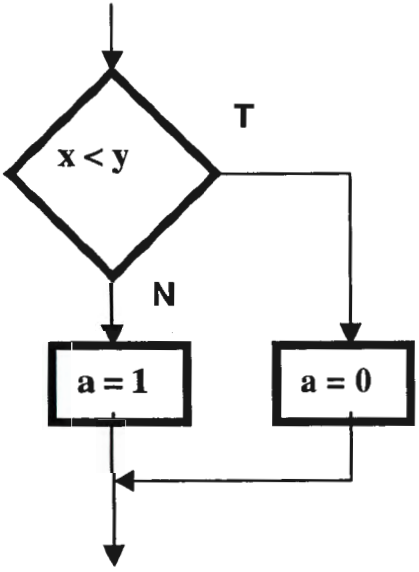
zdanie_n
}

Uwagi dotyczące pseudo - kodu:

- przy zapisie w pseudo - kodzie niezbędne jest stosowanie wcięć celem optycznego uwidocznienia zależności pomiędzy poszczególnymi zdaniami,
- zdania równorzędne powinny być wcięte o tyle samo znaków,
- nawiasy grupujące powinny jednoznacznie wskazywać na to, które z nich tworzą parę.

5.3 PODSTAWOWE RODZAJE INSTRUKCJI – FRAGMENTY SIECI DZIAŁAŃ I PSEUDO-KODU[20]

INSTRUKCJA (WARUNKOWA)
„jeśli"



Jeśli *warunek* **to**
instrukcja 1
w przeciwnym przypadku
instrukcja 2

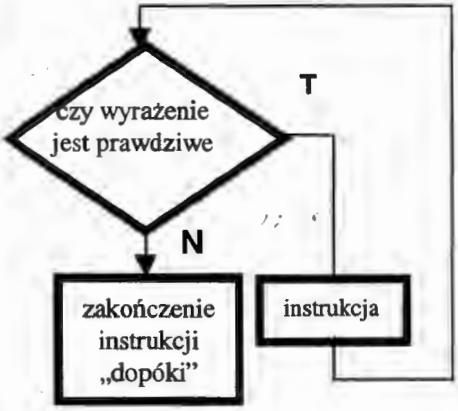
[20] A. Marciniak Object Pascal – jezyk programowania w środowisku Borland Delphi 2.0, Nikom, Poznań 1997

INSTRUKCJA ITERACYJNA
"powtarzaj"

Powtarzaj
ciąg instrukcji

sprawdź wyrażenie
zakończ jeśli
wyrażenie prawdziwe

INSTRUKCJA ITERACYJNA
"dopóki"



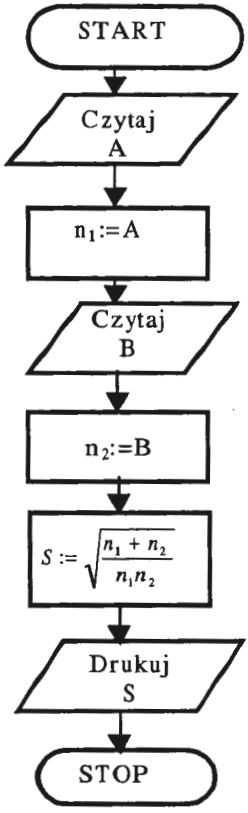
Dopóki
spełniony jest warunek
wykonuj
ciąg instrukcji

Koniec instrukcji

5.4 PRZYKŁADY SIECI DZIAŁAŃ

Przykład 22

Sieć działań bez rozwidlenia - "od góry do dołu". Wszystkie polecenia wykonywane są kolejno jedno po drugim.



- UWAGA -

$n_2 := B$ należy czytać

"za n_2 podstaw B"

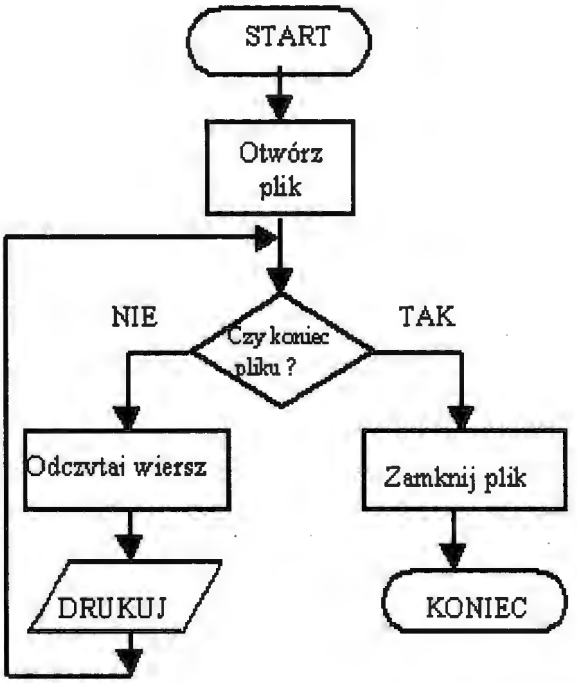
zaś $n := n + 1$

"za n podstaw $n+1$"

"n zwiększ o jeden"

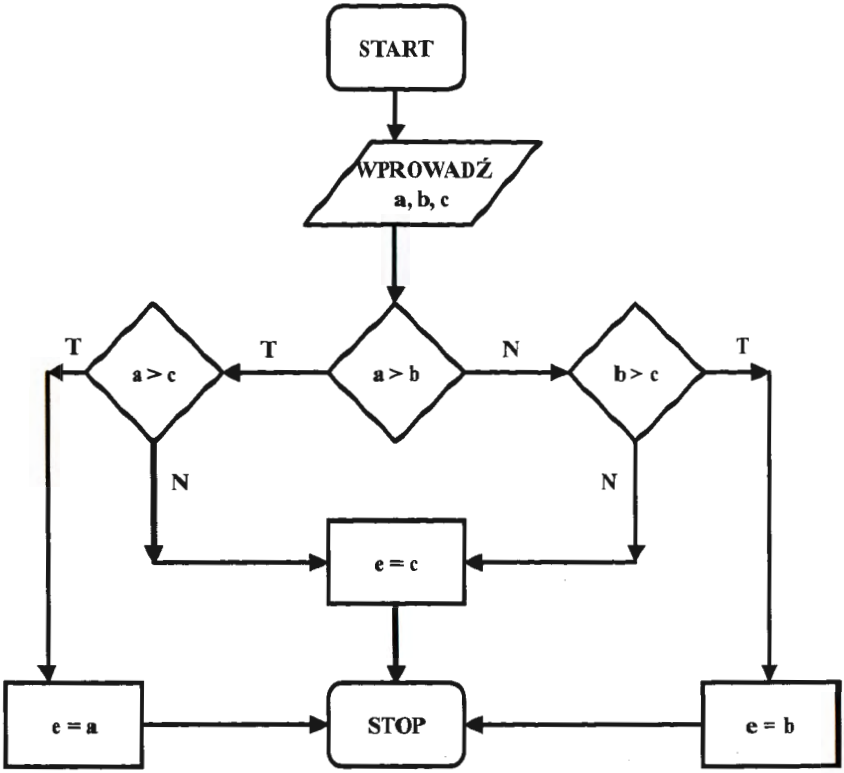
Przykład 23

Przykład sieci działań z rozwidleniem



Przykład 24

Algorytm obliczania wartości maksymalnej spośród trzech liczb a, b, c[21].

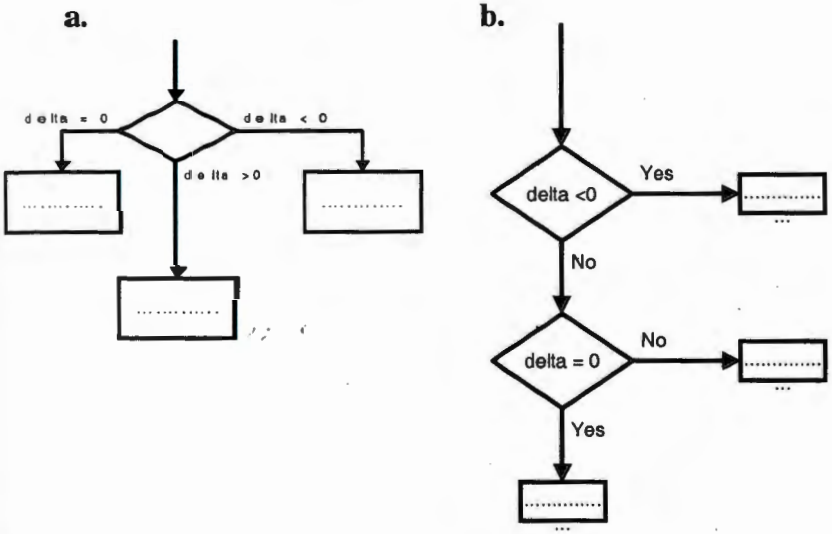


[21] Algorytm zaczerpnięto z Dróbka, Szymański zbiór zadań dla klasy 1 i 2 LO

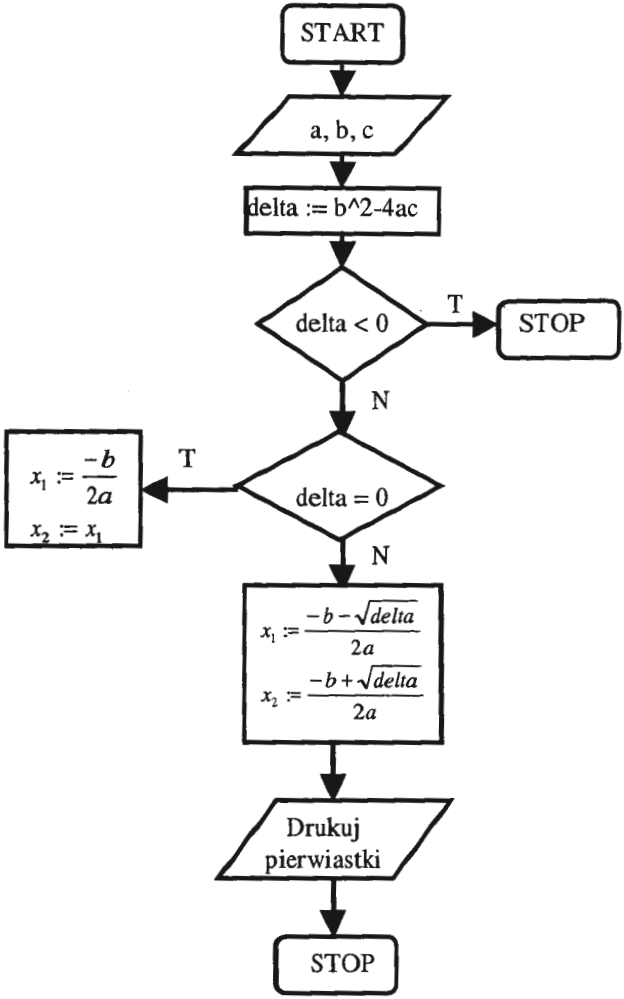
Wiele problemów numerycznych może zostać rozwiązanych przy pomocy różnorodnych algorytmów. Oczywistym jest, że zależnie od rodzaju problemu stosuje się różne metody numeryczne. Stąd można wyróżnić trzy zasadnicze grupy algorytmów:

- algorytmy oparte na metodach dokładnych,
- algorytmy oparte na metodach przybliżonych,
- algorytmy stałopunktowe (ang. fixed - point).

Typowym przedstawicielem algorytmów dokładnych jest algorytm rozwiązania trójmianu kwadratowego. Jest to uwarunkowane tym, że istnieją ściśle określone zależności matematyczne pozwalające w jednoznaczny sposób określić dokładne rozwiązanie (rozwiązania) równania kwadratowego. Przy budowie sieci działań ilustrującej proces rozwiązywania równania kwadratowego brakuje nam elementu decyzyjnego posiadającego jedno wejście i trzy a nie dwa wejścia (rys.12 a). Element taki musimy "złożyć" z dwóch symboli decyzji (rys.12 b). Algorytm rozwiązania trójmianu kwadratowego przedstawia rys.13.



Rys.12. Problem "rozgałęzienia" bloków decyzyjnych.



Rys.13. Sieć działań dla algorytmu rozwiązywania trójmianu kwadratowego.

Niekiedy wzoty analityczne będące dokładnym rozwiązaniem problemu są bardzo skomplikowane i nie jest możliwe podanie ich w postaci prostych równości. Wówczas podaje się poszczególne kroki postępowania opisane prostymi równościami. Klasycznym przykładem jest algorytm rozwiązania układu równań liniowych metodą eliminacji, tj. poprzez takie przekształcanie go aby otrzymać następującą postać:

$$\begin{cases} x_1 + 0 \cdot x_2 + 0 \cdot x_3 + \ldots + 0 \cdot x_n = A_1 \\ 0 \cdot x_1 + x_2 + 0 \cdot x_3 + \ldots + 0 \cdot x_n = A_2 \\ \ldots\ldots\ldots\ldots\ldots \\ 0 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 + \ldots + x_n = A_n \end{cases}$$

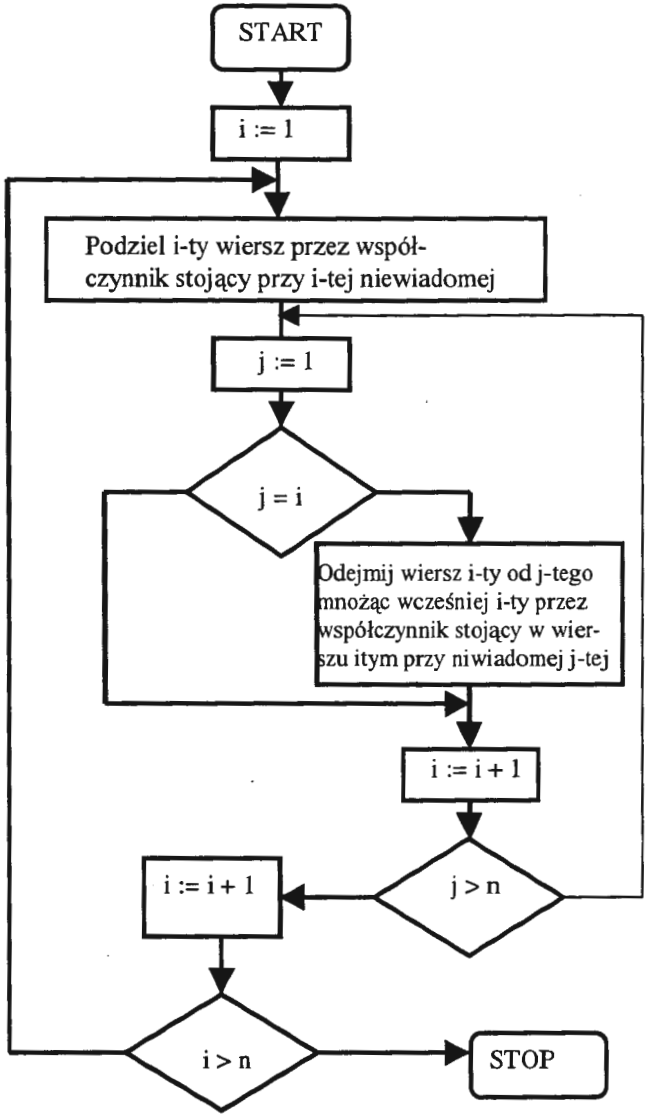
Postać tę otrzymuje się odejmując równania stronami i odpowiednio je przekształcając.

Otrzymane wówczas rozwiązanie ma postać:

$$\begin{cases} x_1 = A_1 \\ x_2 = A_2 \\ \ldots\ldots\ldots \\ x_n = A_n \end{cases} .$$

Problemem o wiele bardziej złożonym jest rozwiązywanie układów równań nieliniowych. Układów takich (w wielu przypadkach) nie potrafimy rozwiązać analitycznie. Nie potrafimy również podać wzorów analitycznych określających dokładne wartości pierwiastków wielomianu stopnia wyższego niż piąty. W tego typu przypadkach należy skorzystać z algorytmów opartych o metody przybliżone. Przykładem takiego algorytmu jest algorytm oparty o metodę siecznych (patrz *przykład* 28). Przy pomocy tej metody rozwiązuje się równania postaci : $W(x) = 0$, przy założeniach, że określony jest przedział poszukiwań <a, b> oraz wartości funkcji w tych punktach mają przeciwne znaki. Założenia te stanowią gwarancję istnienia jednego pierwiastka w rozpatrywanym przedziale <a, b>.

Algorytmy stałopunktowe są algorytmami iteracyjnymi. Typowym przedstawicielem takiego algorytmu jest algorytm oparty na metodzie kolejnych przybliżeń, zwanej metodą iteracji prostej.



Rys.14. Sieć działań dla algorytmu rozwiązywania układu równań liniowych metodą eliminacji.

Przykład 25

Napisać algorytm obliczający wartość $\sqrt{2}$ z zadaną dokładnością.

Najpierw skorzystamy z zapisu liczby w postaci łańcuchowej.

UŁAMKI ŁAŃCUCHOWE[22]

Reprezentacja dodatniej liczby a jako ułamka łańcuchowego ma postać:

$$a = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cfrac{1}{\ldots}}}},$$

gdzie $a_1, a_2, a_3, \ldots$ są ciągiem dodatnich liczb całkowitych zwanych *widmem.*

Przykład 26

Znaleźć reprezentację liczby a w postaci ułamka łańcuchowego:

1) $a = \frac{5}{3}$,

2) $a = \frac{2}{3}$.

[22] Donald B.Small, M.Hosack Ćwiczenia z analizy matematycznej z zastosowaniem systemów obliczeń symbolicznych , WNT, Warszawa 1991 str.130 - 134

Ad. 1)

Krok 1

$$\frac{5}{3}=1+\frac{2}{3}$$

Krok 2

$$\frac{5}{3}=1+\frac{1}{\frac{3}{2}}$$

Krok 3

$$\frac{5}{3}=1+\frac{1}{1+\frac{1}{2}}$$

Czyli:

$$\frac{5}{3}=1+\frac{2}{3}=1+\frac{1}{\frac{3}{2}}=1+\frac{1}{1+\frac{1}{2}}. \qquad [1,1,2]$$

Ad. 2)

$$\frac{2}{3}=0+\frac{1}{\frac{3}{2}}=0+\frac{1}{1+\frac{1}{2}}. \qquad [0,1,2]$$

Reprezentacja liczby w postaci ułamka łańcuchowego przy użyciu komputera

$a=[a]+(a-[a])$, gdzie $[a]$ jest częścią całkowitą liczby a.

Można to zapisać dalej w sposób następujący:

$$a=[a]+(a-[a])=[a]+\frac{1}{\frac{1}{a-[a]}}$$

jeśli zdefiniujemy

$$f(x)=\frac{1}{1-[x]}$$

dla dodatnich niecałkowitych wartości x to

$$a = [a] + \frac{1}{f(a)}.$$

Powtarzając ten proces dla nowej wartości $f(a)$, mamy:

$$a = [a] + \cfrac{1}{[f(a)] + \cfrac{1}{[f^2(a)] + \cfrac{1}{\ldots}}}$$

gdzie $f^n(a) = \cfrac{1}{f^{n-1}(a) - [f^{n-1}(a)]}$.

Stąd liczbę $\sqrt{2}$ możemy zapisać w następującej postaci:

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{\ldots}}}}$$

Postać ta jest punktem wyjścia naszego algorytmu. Sieć działań została zamieszczona poniżej (rys.15), dokładność oznaczono symbolem Δ.

Istnieje również inny algorytm obliczania przybliżenia pierwiastka kwadratowego z liczby naturalnej n. Można przedstawić go następująco:

Krok 1

Za a podstaw 1.

Krok 2

Oblicz a^2.

Krok 3

Powtarzaj Krok 2 dopóki $a^2 < n$.

Krok 4

Sprawdź czy $a^2 = n$.

Jeśli tak, to $\sqrt{n} = a$. STOP

Jeśli nie, to wykonuj dalsze kroki.

Krok 5

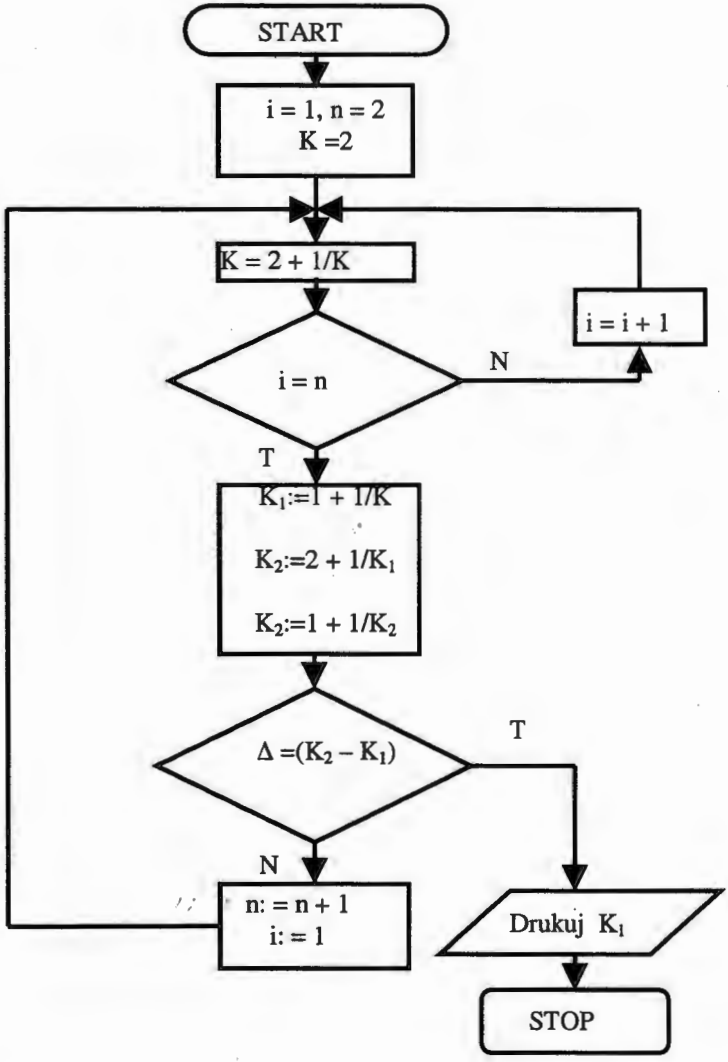
Oblicz: $b_1 = n-(a-1)^2$ oraz $b_2 = a^2 - n$

Krok 6

Sprawdź czy $b_2 < b_1$.

Jeśli tak, to $\sqrt{n} = (a-1)+\frac{b_1}{2(a-1)}$ STOP

Jeśli nie, to $\sqrt{n} = a-\frac{b_2}{2a}$ STOP



Rys.15. Sieć działań algorytmu obliczającego $\sqrt{2}$ z zadaną dokładnością.

Zadania

1. Który z zamieszczonych wyżej algorytmów obliczania pierwiastka drugiego stopnia z liczby naturalnej daje lepsze wyniki i dlaczego ?
2. Narysować sieć działań dla opisanego wyżej (w 6 krokach) algorytm znajdowania $\sqrt{2}$.
3. Napisać algorytm obliczający $\sqrt{7}$ oraz $\sqrt{11}$ bazując na zapisie liczby w postaci ułamka łańcuchowego.
4. Udowodnić, że widmem $\sqrt{3}$ jest $[1,1,2,1,2,...]$.
5. Co jest efektem działania algorytmu zamieszczonego na rys.16 ?
6. Sporządź tabelę o której jest mowa w algorytmie - rys.17.
7. Co jest efektem działania algorytmu zamieszczonego na rys.18 ?

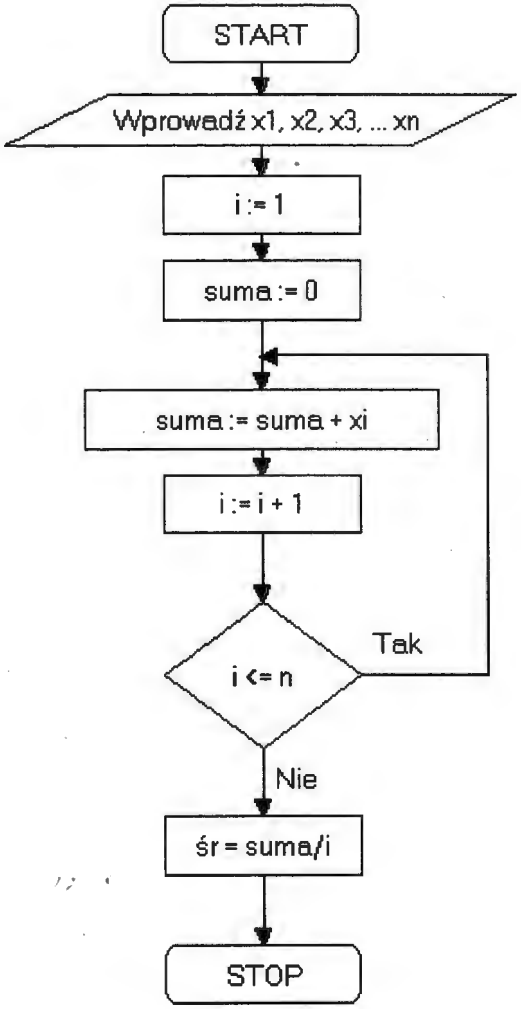
Jako uzupełnienie informacji dotyczących reprezentacji liczb całkowitych dodatnich w postaci ułamków łańcuchowym podamy twierdzenie Eulera, które znacznie ułatwia obliczenia.

Tw. Eulera

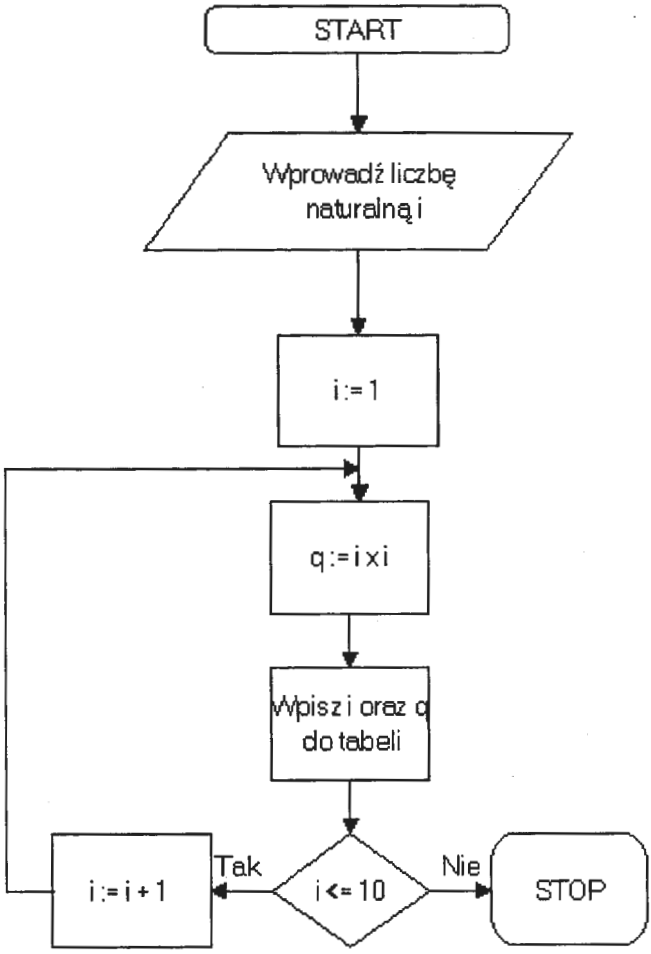
Niech n będzie liczbą całkowitą dodatnią. Wtedy reprezentacją $\sqrt{1+n^2}$ w postaci ułamka łańcuchowego będzie: $[n,2n,2n,2n,...]$.

Przykład 27

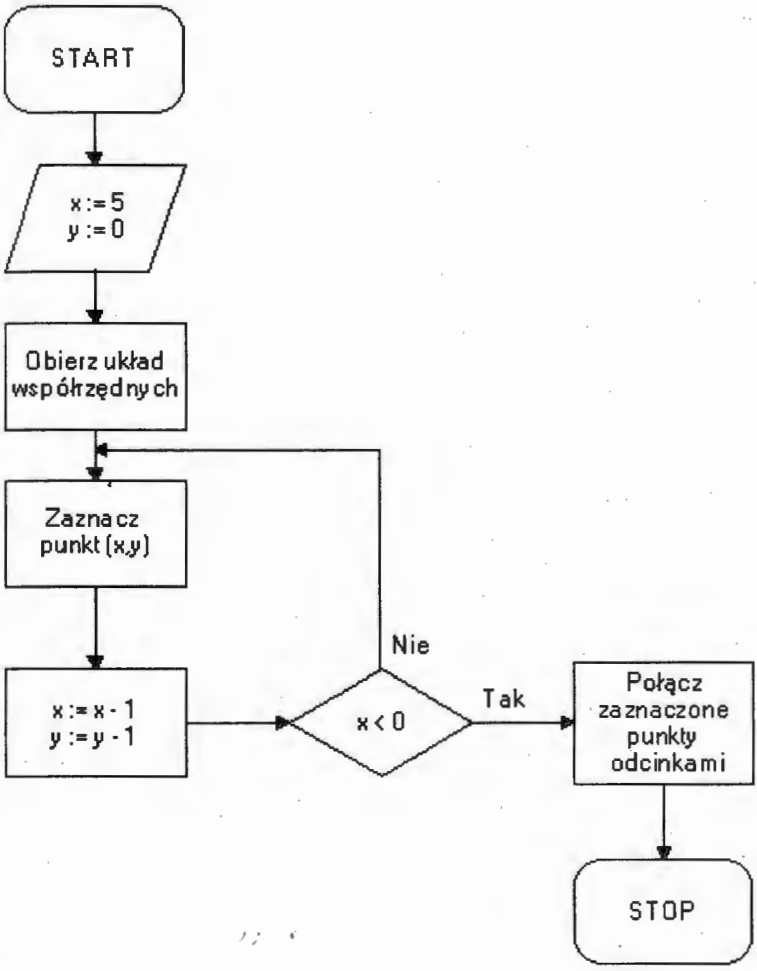
$\sqrt{2}=\sqrt{1+1^2}$; $[1,2,2,2,...]$ $\sqrt{10}=\sqrt{1+3^2}$; $[3,6,6,6,...]$



Rys. 16. Sieć działań do zadania5.



Rys. 17. Sieć działań do zadania6.



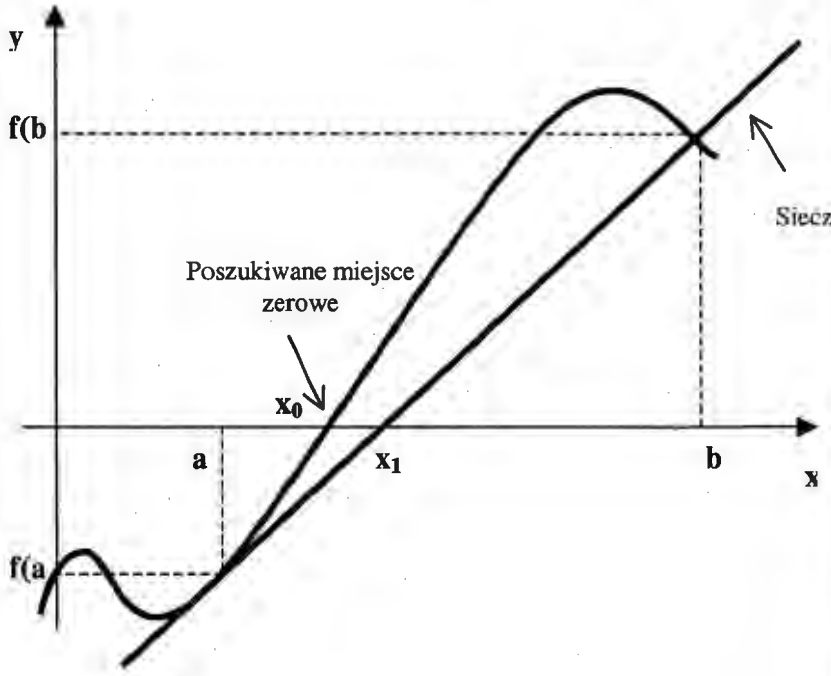
Rys.18. Sieć działań do zadania 7.

Przykład 28

Napisać algorytm wyznaczania miejsca zerowego funkcji metodą siecznych.

Formułujemy problem

Poszukujemy miejsca zerowego funkcji $f(x)$ w przedziale $\langle a,b \rangle$. Zakładamy dla ułatwienia, że w tym przedziale funkcja posiada tylko jedno miejsce zerowe - rys.19.



Rys.19. Ilustracja graficzna metody siecznych.

Opis algorytmu w języku naturalnym

Krok 1 znajdujemy równanie siecznej:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

podstawiając $a = x_1$ oraz $b = x_2$.

Krok 2 znajdujemy współrzędną x_1 punktu w którym sieczna przecina oś OX

$$y = 0 \Rightarrow x_1 = a + \frac{f(b)(a-b)}{f(b) - f(a)}$$

Krok 3 obliczamy $f(x_1)$

Jeżeli $f(x_1) < \varepsilon$, gdzie ε oznacza założoną dokładność

to znaleziono miejsce zerowe $x_0 = x_1$, w przeciwnym wypadku:

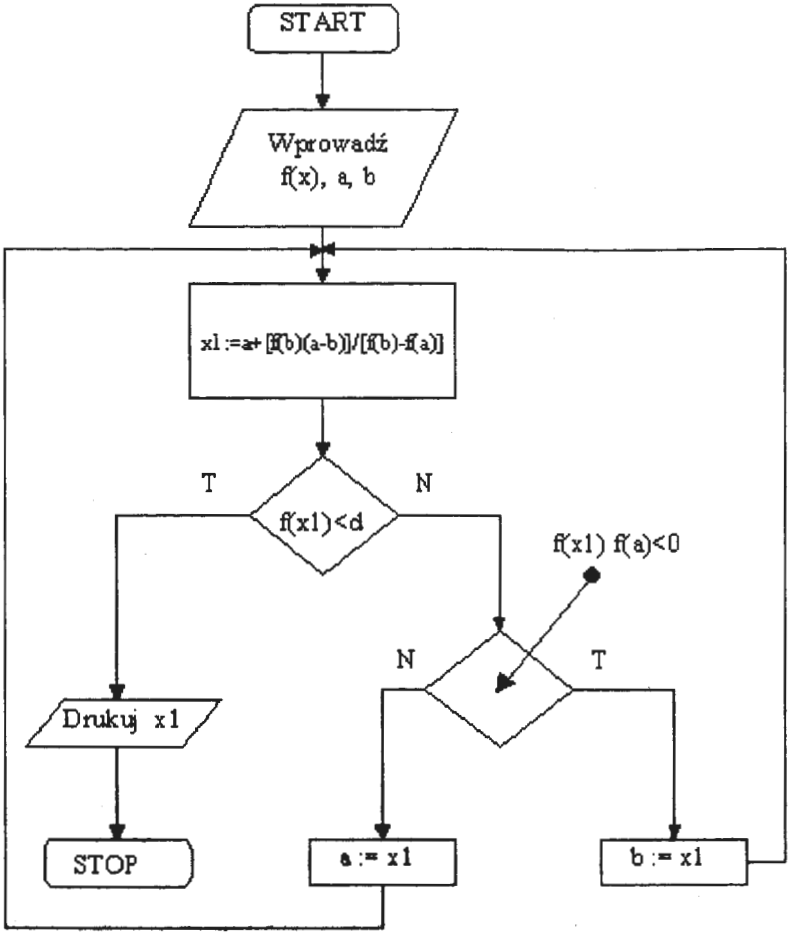
jeżeli $f(x_1) \cdot f(a) < 0$ to w miejsce a podstaw x_1,

w przeciwnym wypadku w miejsce b podstaw x_1.

Powróć do kroku 2.

Krok 4 powtarzamy kroki 1 do 3 aż do osiągnięcia wymaganej dokładności obliczeń.

Poniżej przedstawiono sieć działań odpowiadającą przedstawionemu algorytmowi.

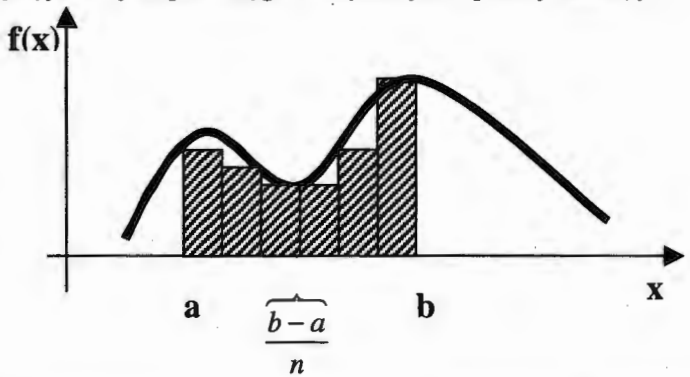


Rys.20. Sieć działań dla problemu wyznaczania miejsc zerowych funkcji metodą siecznych.

Przykład 29

Napisać algorytm obliczania całki oznaczonej $S = \int_a^b f(x)dx$. Założyć, że funkcja $f(x)$ jest co najwyżej drugiego stopnia. Do obliczenia całki wykorzystać metodę prostokątów.

Metodę wyjaśnimy za pomocą geometrycznej interpretacji całki (rys. 21).



Rys. 21. Idea całkowania metodą prostokątów - interpretacja geometryczna

Opis algorytmu w języku naturalnym

Krok 1

Przedział $\langle a,b \rangle$ dzielimy na n - odcinków o długości: $\frac{b-a}{n}$.

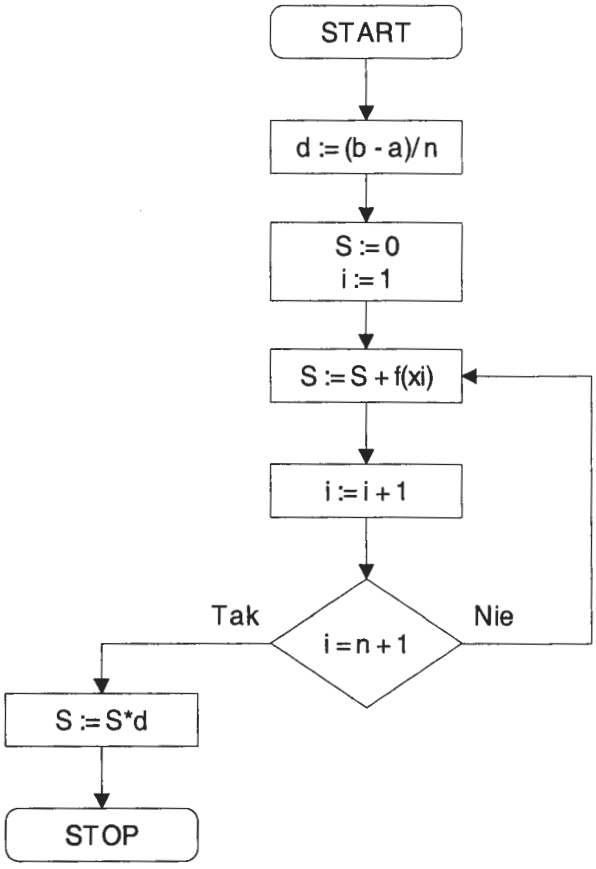
Krok 2

Obliczamy pole jednego z powstałych prostokątów: $S_i = f(x_i) \cdot \frac{b-a}{n}$.

Krok 3

Obliczamy sumę pól będących przybliżeniem szukanej całki oznaczonej

$$\int_a^b f(x)dx = \sum_{i=1}^{n} f(x_i) \cdot \Delta x = \frac{b-a}{n} \sum_{i=1}^{n} f(x_i) \cdot$$



Rys.22. Sieć działań dla problemu wyznaczania całki oznaczonej w zadanych granicach całkowania metodą prostokątów.

ZADANIA

1. Sprawdzić działanie powyższego algorytmu dla następujących danych:

a) $f(x)=2x$, $a=0$, $b=5$, $n=5$;

b) $f(x)=x^2-1$, $a=0$, $b=5$, $n=5$.

Czy otrzymałeś właściwe wyniki ?

Oblicz popełniony błąd.

Co należy poprawić w algorytmie ?

2. Napisać algorytm obliczania całki oznaczonej $S=\int_a^b f(x)dx$. Założyć, że funkcja $f(x)$ jest co najwyżej drugiego stopnia. Do obliczenia całki wykorzystać metodę trapezów. (Algorytm jest całkowicie podobny do poprzedniego, lecz pola przybliżane są trapezami zamiast prostokątami.)

Przykład 30

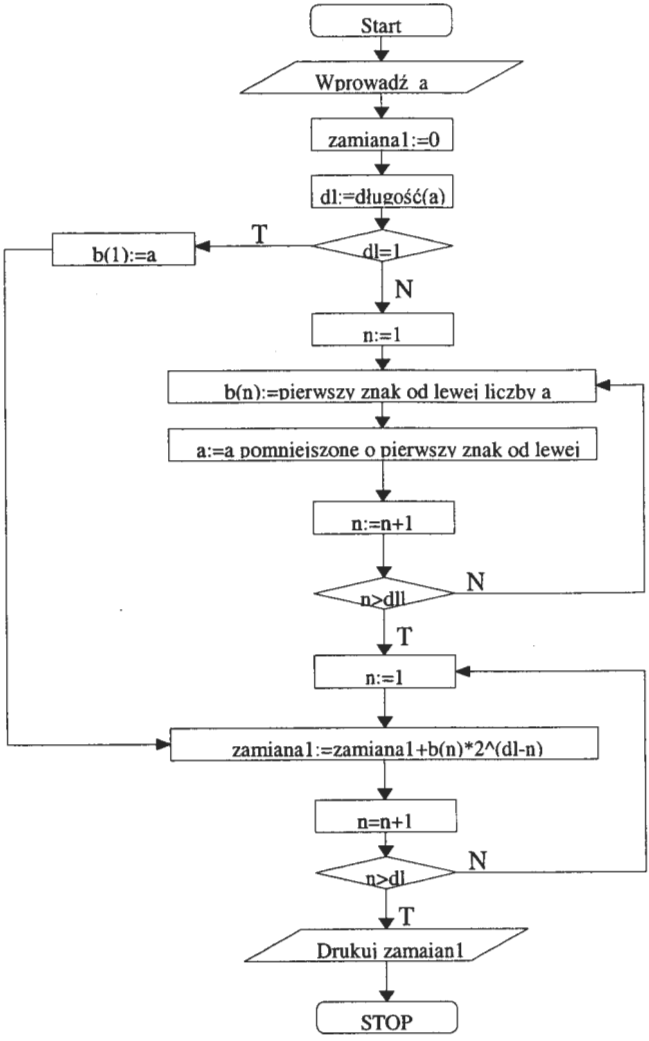
Napisać algorytm zamiany liczby binarnej a na decymalną. Założyć, że liczba a jest całkowita. Algorytm przedstawić w postaci:

a) sieci działań,

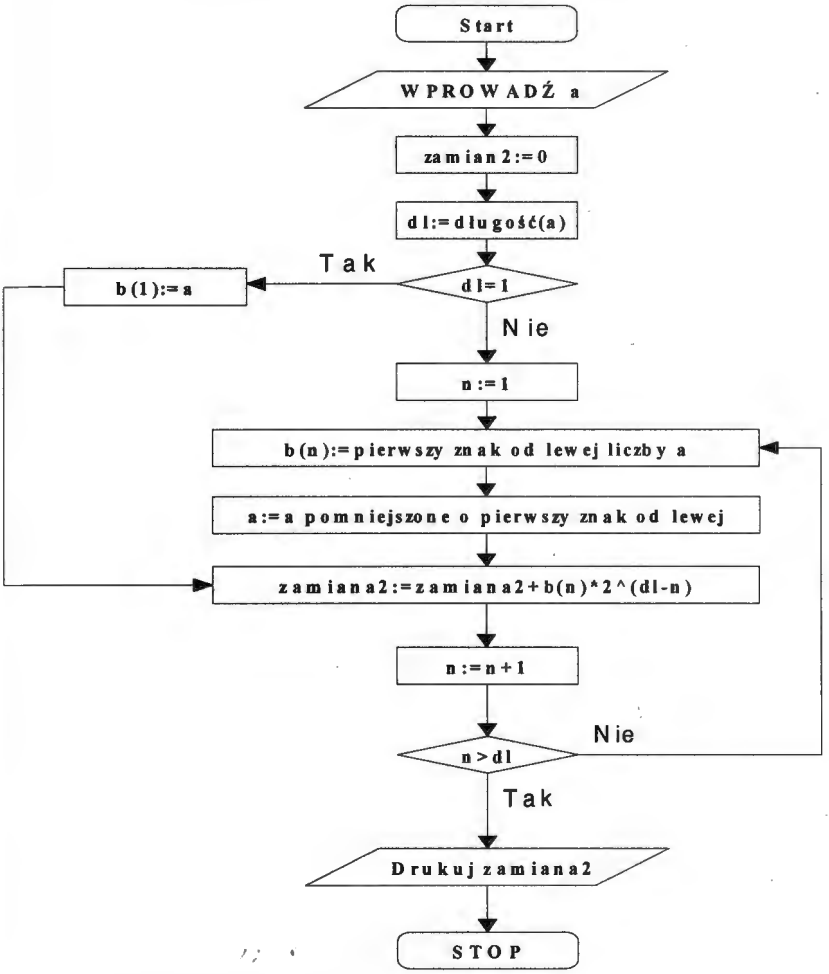
b) schematu blokowego.

Uwagi

Przy budowie algorytmu założono, że liczba **a** została wprowadzona w postaci tekstu, w związku z tym możliwe były typowe operacje tekstowe wykonywane na fragmentach tekstu. Dla uproszczenia, w algorytmie pominięto fazę zamiany łańcucha tekstowego zawartego w kolejnych elementach tablicy **b(n)** na liczbę całkowitą. Dopiero po takiej zamianie możliwe do wykonania staje się działanie mnożenia zawarte w schemacie powyżej. Jednakże, nawet pobieżna analiza powyższej sieci działań wskazuje na pewne jej mankamenty. Widać, że operacje obliczeniowe (ostatnia pętla) można było wykonać równolegle z operacjami tekstowymi wykonywanymi wcześniej. Zmodyfikowana sieć działań widoczna jest poniżej.



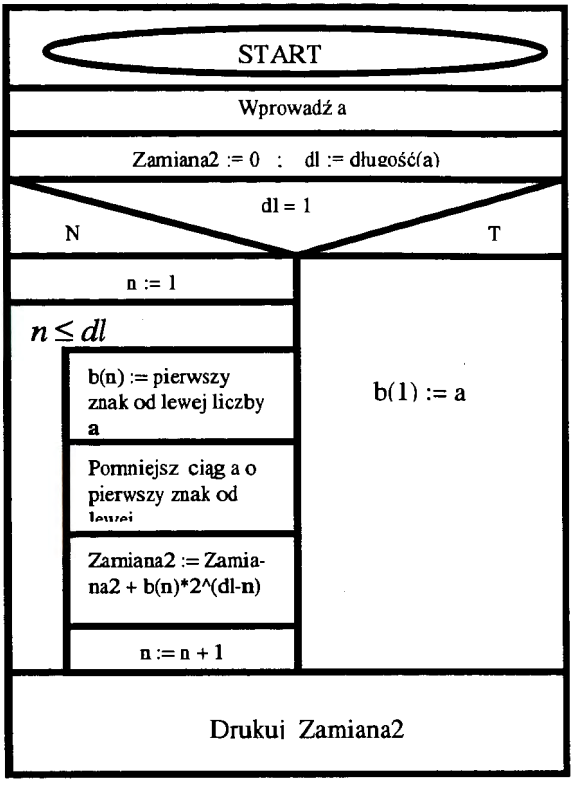
Rys.23. Sieć działań dla problemu zamiany liczby binarnej na decymalną.



Rys.24. Zmodyfikowana sieć działań z rys.23.

Ad b)

Dla zmodyfikowanej sieci działań można narysować odpowiedni schemat blokowy (jak niżej).



Rys.25. Schemat blokowy dla algorytmu zamiany liczby binarnej na decymalną.

Zadania

1. Algorytm z przykładu 28 przedstawić w postaci pseudo - kodu.
2. Napisać funkcję użytkownika (EXCEL) rozwiązującą problem z przykładu 28.

UWAGA

Funkcja użytkownika o nazwie *zamiana1* wraz z jej modyfikacją (*zamiana2*) znajduje się w rozdziale 8.5

5.5 WYSZUKIWANIE BINARNE

Przykład 31

Pomyślałem liczbę całkowitą z zakresu 1 – 100. Ktoś zgaduje. Pięćdziesiąt ? Za mało – odpowiadam. Siedemdziesiąt pięć ? Za dużo. I tak dalej ... aż zgaduję o jaką liczbę chodziło[23].

W przypadku gdy liczba pochodzi z przedziału 1 do N, to możliwe jest jej odgadnięcie po zadaniu maksymalnie $\log_2 N$ pytań.

Dla N = 1000 wystarczy 10 pytań, zaś dla N = 1000000, potrzebujemy najwyżej 20 pytań.

Powyższy przykład ilustruje pewną technikę służącą do rozwiązywania wielu problemów związanych z programowaniem – *wyszukiwanie binarne*.

ZADANIE

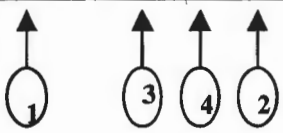
Dlaczego liczba prób w powyższym przykładzie wynosi: $\log_2 N$?

Najczęstszym zastosowaniem wyszukiwania binarnego jest (w programowaniu) szukanie elementu w posortowanej tablicy.

Przykład 32

Dana jest następująca tablica:

26	26	31	31	32	38	38	41	43	46	50	53	58	59	79	97



[23] Przykład zaczerpnięto z J. Bentley Perełki oprogramowania WNT, Warszawa 1992

Poszukując liczby 50, algorytm podejmuje cztery próby pokazane powyżej.

Tok postępowania:

Krok 1

16 liczb – wybieramy liczbę 41

Krok 2

8 liczb – wybieramy liczbę 53

Krok 3

4 liczby – wybieramy liczbę 46

Krok 4

2 liczby – wybieramy liczbę 50

Wyszukiwanie binarne zwykle okazuje się trudne do poprawnego sformułowania. Do przeszukiwania tablicy N – elementowej potrzeba średnio N/2 porównań stosując metodę sekwencyjną, zaś stosując wyszukiwanie binarne nigdy nie dokonujemy więcej niż około $\log_2 N$ porównań. Rola wyszukiwania binarnego nie kończy się jednak na błyskawicznym przeszukiwaniu posortowanej tablicy.

Przykłady zastosowania wyszukiwania binarnego:

- odnalezienie brakującego elementu w danym zbiorze (np. liczb),
- w procedurze obliczania pierwiastka danej liczby,
- do rozwiązywania równania z jedną niewiadomą, wykorzystuje się tu metodę wyszukiwania binarnego dzieląc stopniowo na połowy przedział, w którym zawiera się poszukiwana wartość – analiza numeryczna nazywa to metodą bisekcji,
- gdy algorytm selekcji dokonuje podziałów wokół losowo wybranego elementu, a następnie wywołuje sam siebie dla wszystkich elementów znajdujących się po jednej stronie wybranego elementu, to używa wyszukiwania

binarnego „o losowym wyborze elementu porównywanego",

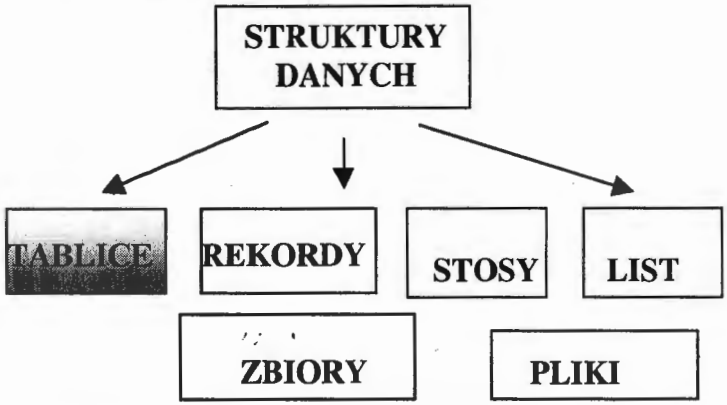
- metoda śledzenia prawidłowości działań algorytmu – gdzie umieścić instrukcję drukowania by zlokalizować błąd ?

Algorytm wyszukiwania binarnego jest efektywny i może być stosowany w pamięci głównej i na dysku; jego wadą jest to, że cała tablica danych musi być znana i wcześniej posortowana.

5.6 SORTOWANIE

Sortowanie to proces ustawiania zbioru obiektów w określonym porządku. Celem stosowania sortowania jest ułatwienie późniejszego wyszukiwania elementów sortowanego zbioru. Metody sortowania podzielono na dwie klasy, a mianowicie: na metody *sortowania tablic* i metody *sortowania plików*. Różnicę w podejściu do sortowania tablic i plików można sobie łatwo uświadomić, wystarczy wyobrazić sobie tablicę danych jako planszę z poukładanymi na niej kartkami zapisanymi liczbami, zaś plik jako stos kart ułożonych na stole jena na drugiej. Na planszy widoczne są wszystkie karty jednocześnie - mamy do nich bezpośredni dostęp, zaś w przypadku stosu kart dostępna jest jedynie karta umieszczona na wierzchołku stosu. Stąd aby znaleźć odpowiednią kartę należy zdejmować kolejno wszystkie karty porównywać je i kolejno odkładać na drugi stos. Tak więc, różnica w podejściu do problemu sortowania tablic i plików sekwencyjnych jest istotna.

Główne struktury danych przedstawia rys.26. Omówione zostaną wybrane metody sortowania tablic, stąd zostały one tam wyróżnione.



Rys.26. Struktury danych.

*) źródło: N.Wirth Algorytmy + struktury danych = programy

5.7 WYBRANE METODY SORTOWANIA TABLIC[24]

Wymagania stawiane metodom sortowania:

a) oszczędne korzystanie z dostępnej pamięci,

b) duża efektywność.

Dobrą miarą efektywności jest liczba P_0 koniecznych porównań i liczba P_r koniecznych przesunięć (przestawień) obiektów. Wielkości te są funkcjami liczby n sortowanych obiektów. Efektywne algorytmy sortowania wymagają liczby porównań rzędu $n \cdot \log n$.

Proste metody sortowania wymagają aż ok. n^2 porównań *kluczy*.

Klucz jest cechą obiektu (jedną z wielu cech charakteryzujących obiekt). Przykładowo, sortowana rosnąco tablica zawierająca liczby dwucyfrowe, może być tablicą kluczy stanowiących o priorytecie wykonywania robót u pewnej grupy klientów. Tak więc, sortując rosnąco ciąg liczb układamy pewną listę zawierającą szereg danych "podpiętych" pod sortowane klucze.

Trzy ważne powody dla których warto przedstawić proste metody sortowania

1. Proste metody bardzo dobrze nadają się do wyjaśnienia głównych zasad sortowania
2. Ich programy są proste a co za tym idzie krótkie i łatwe do zrozumienia.
3. W prawdzie proste metody wymagają większej liczby operacji niż metody skomplikowane, to jednak operacje te są zazwyczaj mniej skomplikowane, co dla małej liczby sortowanych obiektów daje lepsze rezultaty.

Sortowanie przez proste wstawianie

Metoda powszechnie stosowana przez grających w karty. Obiekty (karty) są podzielone umownie na ciąg wynikowy i ciąg źródłowy. W każdym kroku, począwszy od $i = 2$ i zwiększając i o jeden, *i-ty* element ciągu źródłowego przenosi się do ciągu wynikowego wstawiając go w odpowiednim miejscu.

---

[24] N.Wirth Algorytmy + struktury danych = programy, WNT Warszawa 1989

Sortowanie przez proste wybieranie

Etapy postępowania:

1. wybieramy obiekt o najmniejszym kluczu,
2. wymieniamy go z pierwszym obiektem,
3. powtarzamy powyższe operacje z pozostałymi n-1 obiektami, następnie z n-2 obiektami, aż pozostanie tylko jeden obiekt - największy.

> - Uwaga -
> sortowanie jest tu rozumiane jako układanie
> obiektów w kolejności rosnącej

Sortowanie przez prostą zamianę

Algorytm prostej zamiany opiera się na zasadzie porównywania i zamiany par sąsiadujących ze sobą obiektów dopóki wszystkie obiekty nie zostaną posortowane. Metoda ta znana jest też pod nazwą sortowania bąbelkowego. Sortowanie bąbelkowe jest łatwe do przedstawienia z powodu prostej interpretacji fizycznej. Otóż, występuje tam całkowita analogia do cieczy w której znajdują się pęcherzyki gazu o różnych ciężarach. „Bąbelki" wędrują kolejno do góry, najpierw najlżejsze, później cięższe itd. aż uzyskamy posortowaną tablicę.

Sortowanie metodą Shella

Krok 1

Grupowanie i sortowanie wszystkich obiektów oddalonych od siebie o cztery miejsca.
Proces ten nazywa się sortowaniem „co 4".

Krok 2

Sortujemy tak jak w kroku 1 z tą różnicą, że „co 2".

Krok 3

Sortowanie odbywa się „co 1".

Sortowanie metodą malejących przyrostów - zmodyfikowana metoda Shella

Modyfikacja polega na tym, że kolejne przyrosty są liczbami nieparzystymi, a ich ilość określa się w sposób jak niżej.

Oznaczmy kolejne przyrosty jako: $h_1, h_2, h_3, ..., h_t$, przy czym $h_t = 1$, oraz spełniona jest zależność: $h_{k-1} = 2h_k + 1$ przy czym $h_{i+1} < h_i$. Ponadto liczba przyrostów wynosi: $t = (\log_2 n) - 1$, gdzie n jest liczbą sortowanych obiektów.

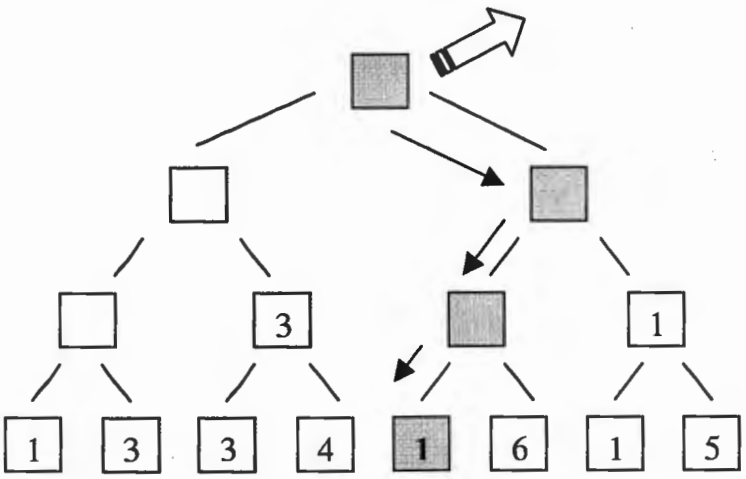
Przykład 33

Obliczyć kolejne przyrosty dla tablicy zawierającej 16 liczb.

Obliczamy liczbę przyrostów $t = (\log_2 16) - 1 = 4 - 1 = 3$. Kolejne przyrosty będące liczbami nieparzystymi wynoszą: 5, 3 i 1.

Sortowanie drzewiaste

W "pniu" drzewa umieszczamy sortowane obiekty. Następnie obiekty porównuje się parami i obiekt o mniejszym kluczu "wędruje w kierunku korony drzewa". W kolejnym kroku następuje "zejście" po ścieżce wyznaczonej przez najmniejszy klucz i eliminacja najmniejszego klucza. Eliminacja najmniejszego klucza polega na zastąpieniu najmniejszego klucza, wszędzie gdzie on występuje, przez klucz o wartości nieskończenie wielkiej. Klucz ten oznaczono symbolicznie pustym kwadratem (patrz rys.24 - zaznaczono wybrany klucz; zacienione prostokąty oznaczają klucz o wartości $+\infty$; czarne strzałki oznaczają drogę „zejścia" po drzewie; na drugim etapie należy powtórzyć proces porównywania kluczy parami itd. aż całe drzewo wypełni się szarymi prostokątami).



Rys.27. Sortowanie drzewiaste

Sortowanie szybkie

Metoda dająca bardzo dobre wyniki pod względem czasu działania przy sortowaniu dużej liczby elementów tablic ułożonych przypadkowo. Algorytm tej metody można zapisać następująco:

Krok 1

Wybierz dowolny element **x** z sortowanej tablicy **T**.

- Uwaga -

Zwykle element x określany jest następująco:

x = a[(lewy + prawy) div 2],

gdzie: div – oznacza dzielenie całkowite,

lewy – jest lewym numerem elementu tablicy,

prawy – jest prawym (skrajnym) numerem elementu tablicy,

a[i] – kolejny element tablicy **T**.

Krok 2

Podziel tablicę T na dwie spójne części: część lewostronną T1 z elementami mniejszymi niż x oraz część prawostronną T2 z elementami większymi lub równymi x.

Tablica T będzie wyglądać następująco:

T1 < x	x	T2 ≥ x

Krok 3

Powtórz kroki 1 i 2 dla T1 i T2 jeśli części T1 i T2 zawierają więcej niż jeden element.

Przykład 34

Dana jest tablica zawierająca następujące liczby:

7 1 8 6 4 9 5

Wyznaczamy element środkowy: x = a[4], a[4] = 6.

Wykonujemy następujące zamiany: a[1] z a[7] oraz a[3] z a[5].

- Uwaga -

Kierujemy się zasadą efektywności – wymieniamy obiekty położone daleko od siebie.

Po wykonaniu powyższych zamian otrzymujemy tablicę w której można wyróżnić dwie części: część pierwszą (lewostronną) od elementu pierwszego do czwartego w której występują elementy mniejsze od sześciu i drugą od elementu piątego do siódmego z elementami większymi równymi sześć.

Algorytm powyższy jest powtarzany dla każdej części tablicy oddzielne tak długo aż otrzymamy części jednoelementowe.

Przykład 35

Dana jest tablica zawierająca następujące liczby:

7 6 5 4 3 2 1

Wyznaczamy element środkowy: x = 4. Wykonujemy następujące zamiany: a[1] z a[7], a[2] z a[6] oraz a[3] z a[5].

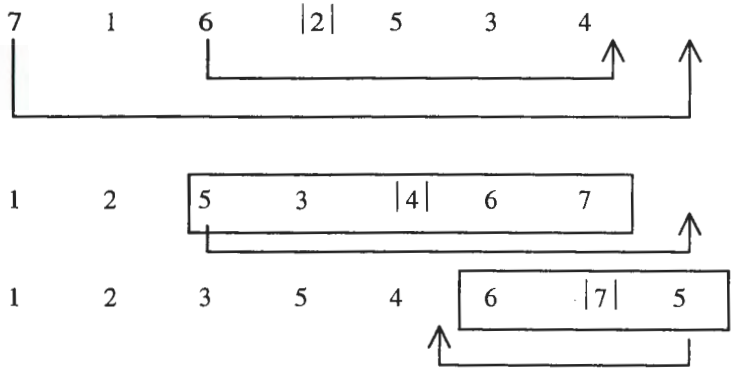
Przykład 36

Dana jest tablica zawierająca następujące liczby:

7 1 6 2 5 3 4

Wyznaczamy element środkowy: x = 2. Dokonujemy stosownych zamian i przestawień.

Sytuację tę można przedstawić jak poniżej.

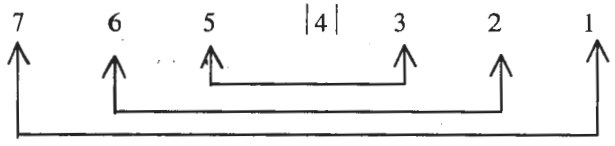


Przykład 37

Dana jest tablica zawierająca następujące liczby:

7 6 5 4 3 2 1

Wyznaczamy element środkowy: x = 4. Dokonujemy stosownych zamian i przestawień.



Tablica została posortowana wyjątkowo sprawnie (zauważmy, że elementy ułożone były w odwrotnej kolejności tj. malejąco).

5.8 REKURENCJA

Rekurencja polega na wywołaniu danego podprogramu w jego treści. Rekurencja może być *bezpośrednia* i *pośrednia*. Rekurencja *bezpośrednia* polega na tym, że tekst danego podprogramu zawiera odwołanie do samego siebie. Z rekurencją *pośrednią* mamy do czynienia wtedy, gdy tekst podprogramu zawiera wywołanie innego podprogramu, w wyniku działania którego następuje wywołanie danego podprogramu.

Rekurencję można wyobrazić sobie podobnie jak nieskończoność. Załóżmy, że mamy przed sobą obrazek na którym widoczny jest krasnoludek trzymający obrazek na którym znajduje się krasnoludek trzymający obrazek ... . Jest to nic innego jak nieskończoność. Z rekurencją jest zupełnie podobnie z tym, że ilość obrazków z krasnoludkami jest ograniczona - jest liczbą skończoną.

Funkcje rekurencyjne definiuje się w kategoriach ich samych. Aby miało to sens i aby zapobiec nieskończonemu cofaniu się (kiedy funkcja nieskończenie odwołuje się do niezdefiniowanych wcześniej wartości), musi być znany przypadek bazowy, który rozpoczyna definicje oraz krok indukcyjny definiujący wartość w zależności od wartości bieżącej[25].

Przykład 38

W pliku tekstowym umieszczone są cztery linijki tekstu w postaci:

aaaaaaaaaaaaaaaaaa

bbbbbbbbbbbbbbbbbb

ccccccccccccccccccc

ddddddddddddddddddd

Naszym zadaniem jest stworzenie takiej procedury rekurencyjnej, która z powyżego pliku odczyta linię po linii a następnie wydrukuje wszystkie linie w

[25] Donald B.Small, Jonh M.Hosack Ćwiczenia z analizy matematycznej z zastosowaniem systemów obliczeń symbolicznych WNT, Warszawa 1995

kolejności odwrotnej[26].

Procedura rekurencyjna wykonująca powyższe zadanie realizuje jedynie trzy funkcje:

- Wczytuje kolejną linię tekstu,
- Rekurencyjnie wywołuje samą siebie,
- Drukuje zapamiętaną linię.

Schemat działania powyższej procedury można znaleźć w literaturze podanej w przypisach.

Przykład 39

Funkcja SILNIA. Definiujemy funkcję SILNIA w następujący sposób:

$$SILNIA(0) = 1 \text{ (przypadek bazowy)}$$

$$SILNIA(n) = n \cdot SILNIA(n-1) \text{ dla } n > 0 \text{ (krok indukcyjny)}$$

Przykład 40

Znaleźć dziesiąty wyraz ciągu Fibonaciego.

Ciąg Fibonnacciego : $r_1 = 1,\ r_2 = 1$ (przypadek bazowy)

$r_n = r_{n-1} + r_{n-2}$, gdzie $r_1, r_2, r_3, \ldots$ - kolejne wyrazy ciągu.

$n = 3 \Rightarrow r_3 = r_2 + r_1 = 1 + 1 = 3,$

$n = 4 \Rightarrow r_4 = r_3 + r_2 = 3 + 1 = 4,$

$n = 5 \Rightarrow r_5 = r_4 + r_3 = 4 + 3 = 7,$

$n = 6 \Rightarrow r_6 = r_5 + r_4 = 7 + 4 = 11,$

$n = 7 \Rightarrow r_7 = r_6 + r_5 = 11 + 7 = 18,$

---

[26] Przykład zaczerpnięto z książki W.W.Walczak Programowanie w języku Pascal dla już nie całkiem początkujących Wydawnictwo W&W Warszawa 1999

$n=8 \Rightarrow r_8 = r_7 + r_6 = 18+11=29$,

$n=9 \Rightarrow r_9 = r_8 + r_7 = 29+18=47$,

$n=10 \Rightarrow r_{10} = r_9 + r_8 = 47+29=76$.

Przykład 41

Przedstawić następujące ciągi w postaci wzoru rekurencyjnego:

a) $a_n = n+1$,

b) $b_n = 2^n$,

c) $c_n = e^n$.

Ad. a)

$a_{n+1} = n+1+1 = a_n + 1$, $a_1 = 2$.

Możemy zatem napisać:

$$\begin{cases} a_1 = 2 \\ a_{n+1} = a_n + 1 \end{cases}.$$

Ad. b)

$a_{n+1} = 2^{n+1} = 2^n \cdot 2 = 2a_n$, $a_1 = 2$.

$$\begin{cases} a_1 = 2 \\ a_{n+1} = 2a_n \end{cases}.$$

Ad. c)

$a_{n+1} = e^{n+1} = e \cdot e^n = e \cdot a_n$, $a_1 = e$.

$$\begin{cases} a_1 = e \\ a_{n+1} = e \cdot a_n \end{cases}.$$

Przykład 42

GENERATOR FIBONACCIEGO

Przypomnijmy, ciąg rekurencyjny Fibonacciego ma postać:

$$\begin{cases} a_n = a_{n-2} + a_{n-1}, & n \geq 2 \\ a_0 = a_1 = 1 \end{cases}.$$

W generatorze liczb losowych próbowano zastosować ciąg reszt:

$$x_n = (x_{n-2} - x_{n-1}) \mod m, \quad n \geq 2.$$

Znaleźć sześć początkowych wyrazów ciągu reszt przedstawionego powyższym wzorem rekurencyjnym dla $m = 2$.

$x_1 = 1,$
$x_2 = (x_0 + x_1) \mod 2 = (1+1) \mod 2 = 0,$
$x_3 = (x_1 + x_2) \mod 2 = (1+0) \mod 2 = 1,$
$x_4 = (x_2 + x_3) \mod 2 = (0+1) \mod 2 = 1,$
$x_5 = (x_3 + x_4) \mod 2 = (1+1) \mod 2 = 0,$
$x_6 = (x_4 + x_5) \mod 2 = (1+0) \mod 2 = 1.$

Ciąg reszt x_n zachowuje się na tyle bezładnie (przyjmuje losowe wartości), że od dawna już interesował matematyków[27]. Ciąg powyższy przeszedł pomyślnie test równomierności rozkładu, zaś niestety nie spełnił innych wymaganych testów, tj. testu niezależności i testu serii. Stąd też, został on zmodyfikowany do postaci:

$$x_n = x_{n-r} \Diamond x_{n-s} \quad n \geq r \qquad r > s \geq 1,$$

gdzie dodawanie zastąpiono działaniem $\Diamond$. Działanie $\Diamond$ jest dowolnym działaniem modulo m. Stosuje' się następujący umowny zapis: $F(r, s, \Diamond)$. Przyjmuje się też, że: $m = 2^L$ gdzie $L \in N$.

Istnieją takie parametry r i s, dla których okres generatora (tzn. długość niepowtarzalnego ciągu cyfr) jest maksymalny.

Przykładowo, maksymalny okres uzyskuje się w przypadku generatora typu

---

[27] R.Wieczorkowski R.Zieliński Komputerowe generatory liczb losowych, WNT Warszawa 1997

„+" / "-" dla r = 17, s = 5 i $m = 2^{32}$. Okres ten wynosi: $T_{MAX} = (2^{17} - 1) \cdot 2^{31}$.

W przypadku generatora $F(r, s, \bullet)$, gdzie działanie $\bullet$ oznacza mnożenie otrzymuje się:

$$T_{MAX} = (2^r - 1) \cdot 2^{n-3}.$$

Ciekawego spostrzeżenia dostarcza analiza działania logicznego XOR (patrz rozdział 3.3 niniejszego skryptu).

Okazuje się bowiem, że dla zmiennych boolowskich mamy:

$$a \ \ xor \ \ b = (a + b) \bmod 2.$$

Dla generatora z działaniem XOR $T_{MAX} = 2^r - 1$.

Generatory liczb losowych oparte o ciąg rekurencyjny Fibonacciego przeżywają obecnie swoisty renesans, co związane jest między innymi z łatwością ich implementacji. Generator liczb losowych, aby mógł wystartować musi zostać zainicjowany. Oznacza to, że należy zadać mu pewne wartości początkowe - wartości startowe. Przykładowo, liczby startowe można uzyskać bazując na parametrach związanych z datą i czasem. Parametry te są w komputerze bezpośrednio dostępne. W 1990 r. Anderson zaproponował następującą formułę dla liczby początkowej x_0 inicjującej generator liczb losowych:

$$x_0 = rr + 100(mm - 1 + 12(dd - 1 + 31(gg + 24(\min + 60s)))),$$

gdzie: *rr* - dwie ostatnie cyfry roku,

mm - miesiąc (od 1 do 12),

dd - dzień (od 1 do 31),

gg - godzina (od 0 do 23),

min - minuta (od 0 do 59),

s - sekunda (od 0 do 59).

Dotychczasowe wydawnictwa
WYŻSZEJ SZKOŁY INFORMATYKI STOSOWANEJ I ZARZĄDZANIA

- Z. Stachowiak: *Ekonomia. Zarys podstawowych problemów.* 1998; Wyd. 2. 2000.
- Z. Mikołejko: *Elementy filozofii.* 1998; Wyd. 2 popr. i rozsz. 1998; Wyd. 3 popr. i rozsz. 1999.
- W. Arczewska: *Bazy danych Oracle* 1998; Wyd. 2 popr. i rozsz. 1999; Wyd. 3 popr. 1999.
- S. Bożek, P. Cholajda, G. Szkatuła: *Wstęp do bazy danych MS Access dla Windows 95.* 1998.
- T. Łuba: *Podstawy układów logicznych.* 1998; Wyd. 2 popr. 1999.
- G. Szkatuła, A. Pogorzelec: *Ćwiczenia z bazy danych Microsoft Access 97.* 1999; Wyd. 2 rozsz. 1999.
- A. Żochowski: *L E M Laboratorium eksperymentów matematycznych.* 1999; Wyd. 2. popr. 2000.
- J. Hołubiec, red.: *Analiza systemowa w finansach i zarządzaniu. Wybrane problemy.* 1999.
- M. Doros: *Przetwarzanie obrazów. Materiały pomocnicze.* Cz.1, 2. 1999; Wyd. 2 popr. 1999.
- L. Oleksyn: *Istota, zakres i cechy rachunku kosztów.* 1999.
- L. Oleksyn: *Zadania rachunku kosztów w zarządzaniu.* 1999.
- L. Oleksyn: *Ekonomia - zarys wykładu.* 1999.
- Z. Nahorski: *Metoda najmniejszych kwadratów.* Cz. 1, 2. 1999.
- O. Hryniewicz: *Wykłady ze statystyki.* 1999.
- P. Cholajda: *Systemy informatyczne w MS ACCESS 97 PL.* 1999.
- K. Liderman: *Bezpieczeństwo informacji w systemach informatycznych.* 2000.
- M. Barszczewski: *Zarządzanie sieciami telekomunikacyjnymi.* 2000.
- J. Borkowski, M. Dyrda, L. Kanarski, B. Rokicki: *Wybrane problemy psychologii organizacji. O konflikcie i negocjacjach.* 2000.
- J. Jarmakiewicz: *Sieci teleinformatyczne.* Cz. 1, 2. 2000.
- T. Łuba: *Synteza układów logicznych.* 2000.
- H. Spustek: *Elementy informatyki.* 2000.

**WYŻSZA SZKOŁA
INFORMATYKI STOSOWANEJ
I ZARZĄDZANIA**

pod auspicjami
Polskiej Akademii Nauk

ZAŁOŻYCIELEM
Wyższej Szkoły Informatyki Stosowanej i Zarządzania
jest
Fundacja Krzewienia Nauk Systemowych
powołana z inicjatywy
Prezesa
POLSKIEJ AKADEMII NAUK

FUNDATOREM
Fundacji Krzewienia Nauk Systemowych
jest
POLSKA AKADEMIA NAUK

ORGANEM
sprawującym nadzór jest
MINISTERSTWO EDUKACJI NARODOWEJ

Wyższa Szkoła Informatyki Stosowanej i Zarządzania
prowadzi studia wyższe na kierunkach:
INFORMATYKA
ZARZĄDZANIE I MARKETING

SIEDZIBA
Instytut Badań Systemowych
Polskiej Akademii Nauk
ul. Newelska 6, 01-447 Warszawa

ISBN 83-88311-17-4