Wykład 5

- Funkcja silnia, współczynniki rozwinięcia dwumianu, trójkąt Pascal'a
- Ciąg Fibonacciego
- Wyszukiwanie binarne i interpolacyjne
- Problem "plecakowy"
- Poszukiwanie miejsc zerowych funkcji
- Interpolacja funkcji metodą Lagrange'a
- Całkowanie numeryczne

Funkcja silnia, współczynniki rozwinięcia dwumianu, trójkąt Pascal'a

Dla nieujemnej liczby całkowitej *n* definiuje się funkcję "silnia" - *n!* jako:

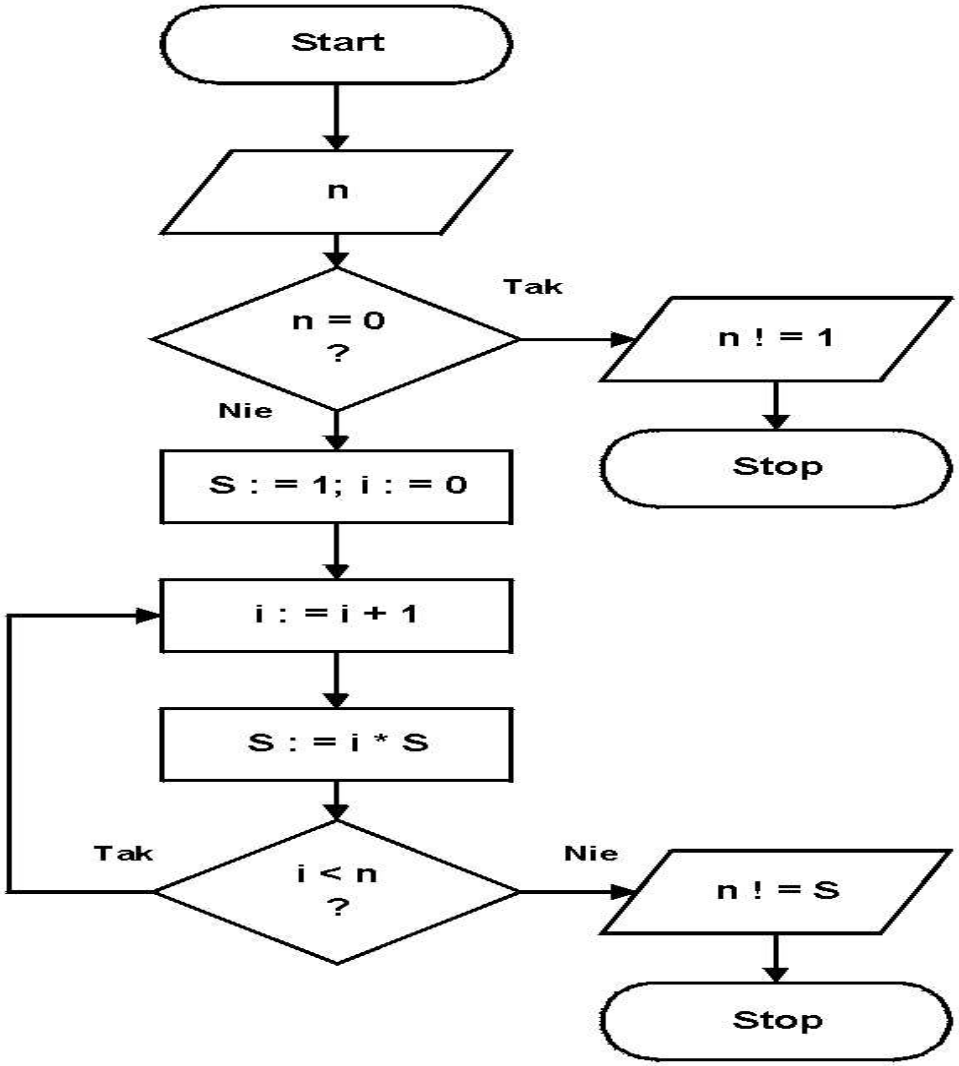
$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n & \text{dla } n > 0 \end{cases}$$

Schemat blokowy algorytmu obliczania wartości *n!* w wersji iteracyjnej dla danego, nieujemnego *n* pokazuje rysunek 5.1.

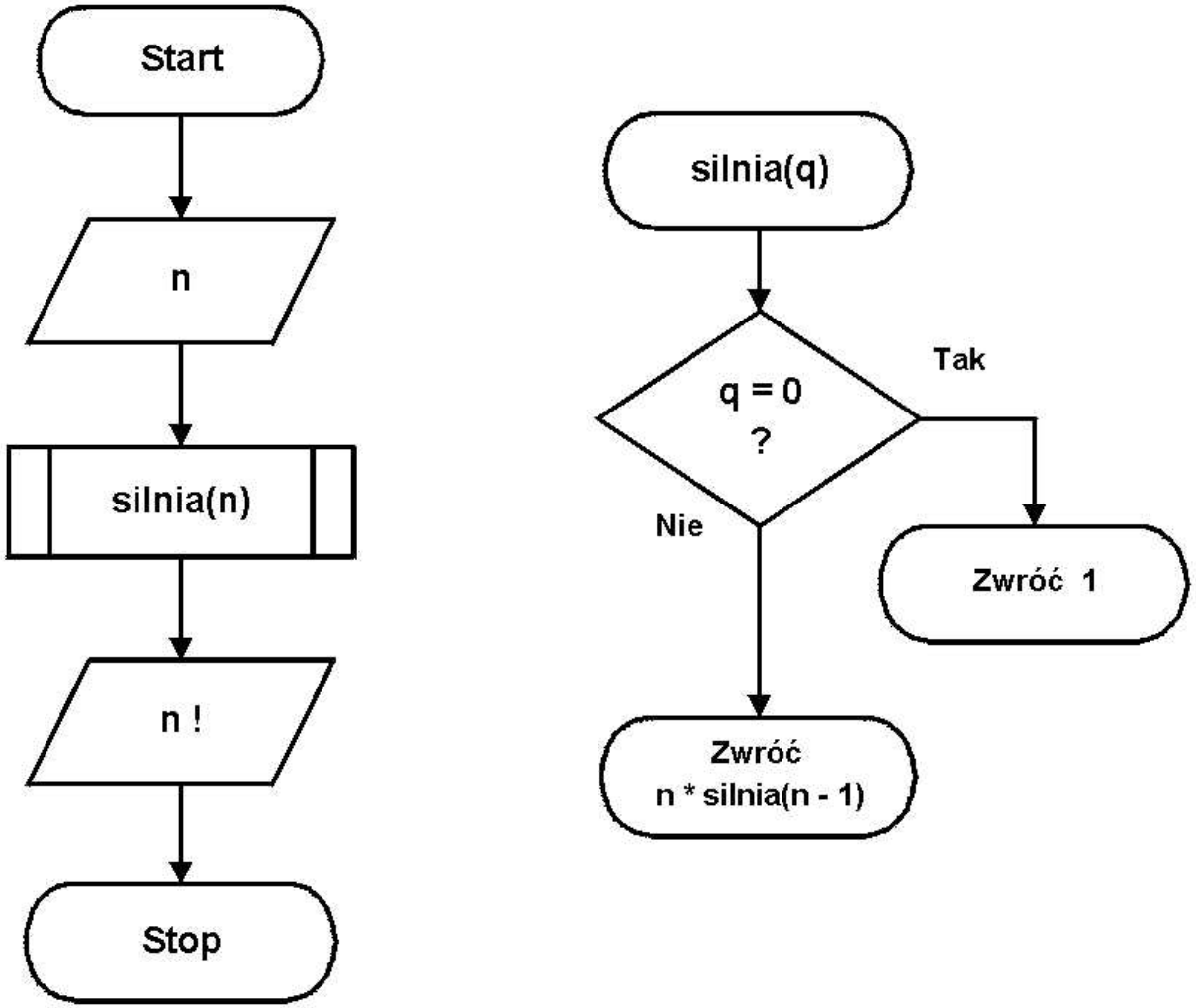
Rekurencyjna definicja funkcji silnia ma postać:

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n \cdot (n-1)! & \text{dla } n > 0 \end{cases}$$

Schemat blokowy rekurencyjnej wersji algorytmu obliczania funkcji "silnia" (na schemacie zamiast symbolu ! użyto nazwy silnia) pokazuje rys. 5.2.



Rys.5.1. Wersja "iteracyjna" algorytmu obliczania funkcji silnia



Rys.5.2. Wersja rekurencyjna algorytmu obliczania silni

Rozwinięcie tzw. *dwumianu Newtona* ma postać:

$$(a+b)^n = \binom{n}{0}a^n + \binom{n}{1}a^{n-1}b + \binom{n}{2}a^{n-2}b^2 + \quad + \binom{n}{k}a^{n-k}b^k + \quad + \binom{n}{n-1}ab^{n-1} + \binom{n}{n}b^n$$

$\binom{n}{k}$ (czytaj "n po k") oblicza się ze wzoru:

$$\binom{n}{k} = \frac{n!}{k!\cdot(n-k)!}$$

Współczynniki rozwinięcia dwumianu Newtona dla kolejnych n = 0, 1, 2, . . . tworzą tzw. *trójkąt Pascala* (rys.5.3)

n = 0 **1**
n = 1 **1 1**
n = 2 **1 2 1**
n = 3 **1 3 3 1**
n = 4 **1 4 6 4 1**
n = 5 **1 5 10 10 5 1**
n = 6 **1 6 15 20 15 6 1**

Rys. 5.3. Kilka początkowych wierszy "trójkąta Pascal'a"

Ciąg Fibonacciego

Ciąg ten powstaje w ten sposób, że wartość kolejnego wyrazu (oprócz pierwszych dwóch) jest sumą dwóch poprzednich. Wyrazy pierwszy i drugi są z definicji równe 1:

$$F_k = \begin{cases} 1 & dla\ k = 1,2 \\ F_{k-2} + F_{k-2} & dla\ \ k \geq 3 \end{cases}$$

Jest to postać rekurencyjna definicji ciągu Fibnacciego, a jej wykorzystanie do obliczania wartości k – go wyrazu wymaga wielokrotnego obliczania tych samych elementów. Złożoność obliczeniowa procedury obliczania wartości k-tego wyrazu ma tutaj charakter wykładniczy, wymaga wykonania nie mniej niż 2^k sumowań.

Kilka początkowych wyrazów ciągu to zgodnie z definicją: 1, 1, 2, 3, 5, 8, 13, 21, . . .

Interesująca (i zaskakująca) jest bezpośrednia postać wyrażenia na *k* – ty wyraz ciągu Fibonacciego:

$$F_k = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k\right]$$

Algorytm iteracyjny obliczania k – go wyrazu ciągu Fibonacciego to lista kroków:

Dana: Liczba naturalna *k* nie mniejsza niż 1

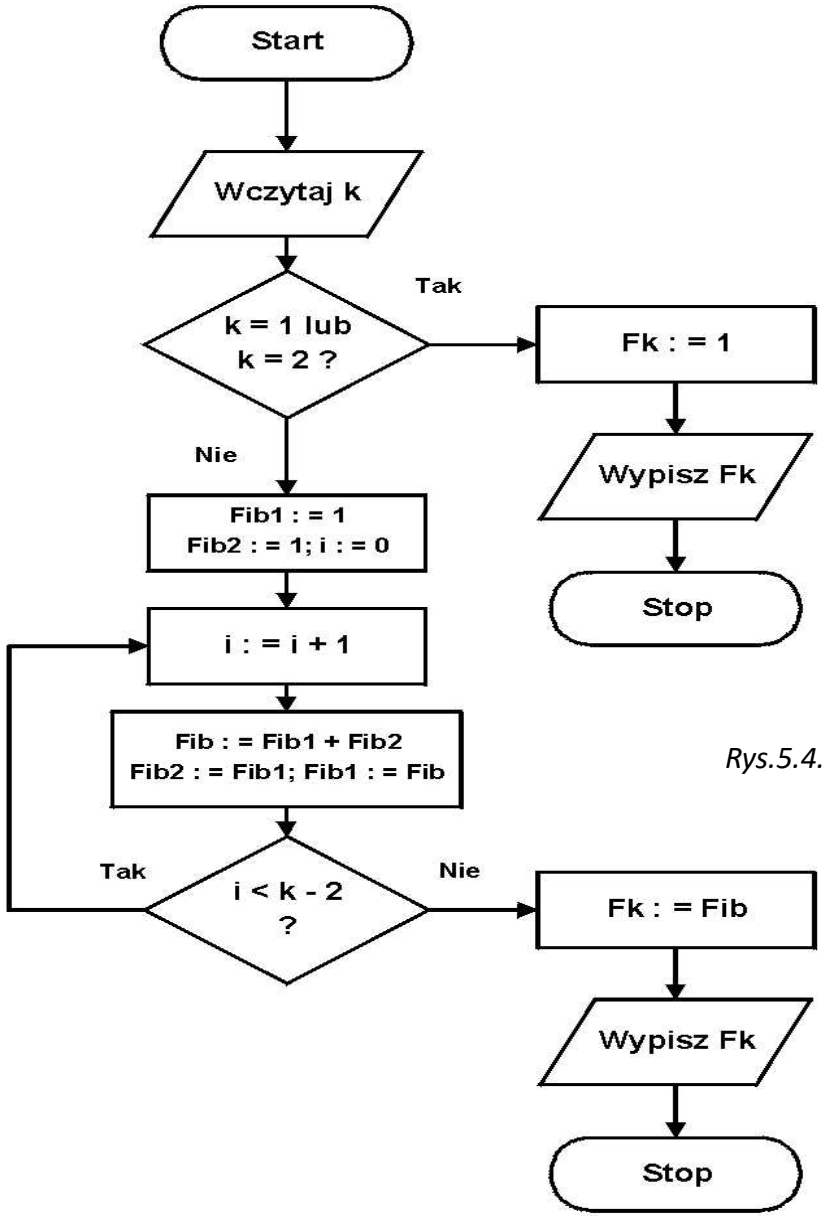
Wynik: Wartość F_k

1º Jeżeli $k = 1$ lub $k = 2$, to przyjmij $F_k = 1$ i zakończ obliczenia

2º Przyjmij *Fib1* := 1 oraz *Fib2* := 1

3º $k - 2$ razy wykonuj sekwencję przypisań: *Fib := Fib1 + Fib2; Fib2 := Fib1; Fib1 := Fib*

4º Wynik to *Fib*.



Rys.5.4. Iteracyjny algorytm obliczania k – go wyrazu ciągu Fibonacciego

Wyszukiwanie binarne i interpolacyjne

Na jednym z pierwszych wykładów - algorytm wyszukiwania określonego elementu w zbiorze, w którym ułożenie – kolejność elementów jest przypadkowa.

Znacznie szersze zastosowanie w informatyce mają algorytmy wyszukiwania określonego elementu w zbiorze uporządkowanym.

Do rozwiązania takiego problemu doskonale nadają się algorytmy klasy "dziel i zwyciężaj".

Najprostszym, chociaż nie najbardziej efektywnym jest algorytm *wyszukiwania binarnego*.

Zakładamy, że dany jest w postaci tablicy $a[k..l]$ gdzie $k \leq l$, uporządkowany ciąg elementów, tzn. $a_k \leq a_{k+1} \leq \ldots \leq a_l$ oraz element y.

Zadanie polega na wskazaniu pozycji s, na której w ciągu A znajduje się element y.

Metoda wyszukiwania binarnego sprowadza się do tego, że znajdujemy element, który znajduje się w połowie (lub prawie w połowie) badanego ciągu i sprawdzamy, czy szukany wyraz jest równy, mniejszy lub większy od niego.

Pierwszy przypadek oznacza, że znaleźliśmy położenie wyszukiwanego wyrazu,

w drugim powtarzamy połowienie "lewej" części ciągu i przeszukiwanie go,

a w trzecim - przeszukujemy część ciągu na prawo od punktu podziału.

Powtarzając tą procedurę, albo znajdziemy wreszcie pozycję szukanego wyrazu ciągu,

albo stwierdzimy, że dany element nie występuje w nim.

Uporządkowany opis algorytmu wyszukiwania binarnego ma postać:

Dane: Tablica $A = a[k], a[k+1], \ldots, a[l]$ zawierająca uporządkowany ciąg elementów oraz element y taki że $a_k \leq y \leq a_l$

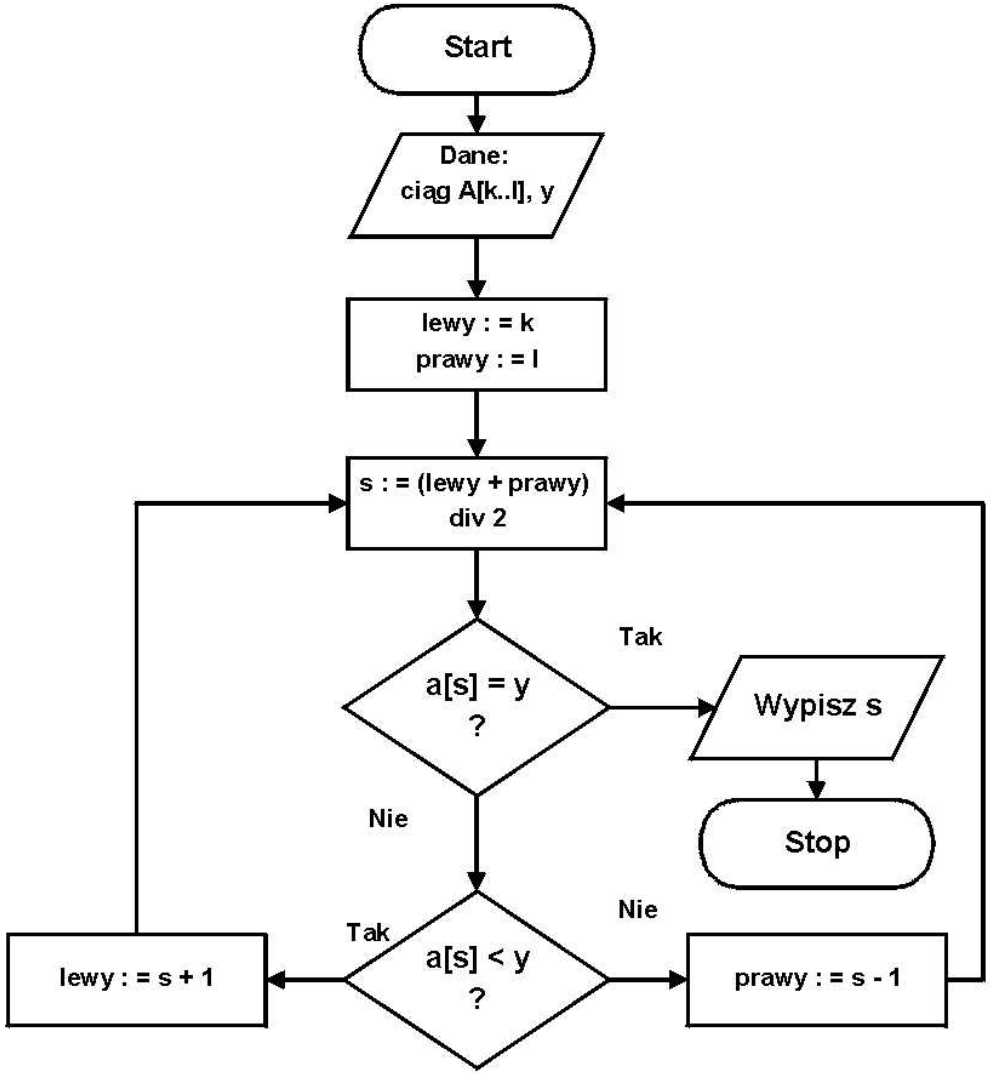
Wynik: *Takie* s ($k \leq s \leq l$), że $a_s = y$ lub $s = -1$, *jeżeli* s nie występuje w ciągu A

1º *Przyjmij lewy := k, prawy := l; lewy, prawy to bieżące (i początkowe) krańce przedziałów poszukiwań*

2º *Jeżeli lewy > prawy, to przyjmij s :=* -1 i zakończ poszukiwanie
3º *Przyjmij s := (lewy + prawy)* ***div*** *2 (****div*** *- dzielenie całkowite).*

Jeżeli $a_s = y$, to s jest szukana pozycją i zakończ działania.

Jeżeli $a_s < y$, *to lewy :=* $s + 1$, *w przeciwnym przypadku prawy :=* s - 1 i wróć do kroku ***2º.***



Rys. 5.5. Algorytm wyszukiwania binarnego

Uogólnieniem algorytmu binarnego przeszukiwania jest (równie ważny z praktycznego punktu widzenia) algorytm *binarnego umieszczania*.

Pozwala on rozwiązać zadanie znalezienia miejsca dla elementu *y* w uporządkowanym ciągu tak, aby po umieszczeniu *y* na znalezionej pozycji, ciąg był nadal uporządkowany.

Poniżej opis algorytmu; podobny do metody binarnego wyszukiwania:

Dane: Tablica $A = a[k], a[k+1], \ldots, a[l]$ zawierająca uporządkowany ciąg elementów oraz element *y* taki że $y \geq a_k$

Wynik: *Miejsce dla y* w ciągu *A*, czyli największe *r takie, że* $a_r \leq y \leq a_{r+1}$ jeśli $k \leq r \leq l-1$ lub $r = l$, jeżeli $a_l \leq y$

1º *lewy := k, prawy := l*

2º Jeśli $a_s \leq y$, to *lewy := s*, w przeciwnym przypadku *prawy := s* – 1. Jeśli *lewy = prawy, to zakończ działania i przypisz r := lewy, w przeciwnym przypadku powtórz krok* ***2º***

Różnica pomiędzy wyszukiwaniem *interpolacyjnym* a binarnym polega tylko na innym sposobie obliczania indeksu *s* dzielącego podciągi, w których poszukujemy elementu *y*:

- w algorytmie binarnym wartość *s* wyznacza się z zależności:

$$s = lewy + \frac{1}{2}(prawy - lewy)$$

- w algorytmie interpolacyjnym jest to wzór:

$$s = lewy + \frac{y - a_{lewy}}{a_{prawy} - a_{lewy}}(prawy - lewy)$$

Problem "plecakowy"

Zadanie nazywane w literaturze "problem plecakowy" polega na zapakowaniu do plecaka o ograniczonej pojemności zbioru najbardziej wartościowych (przydatnych) rzeczy.

Jego dokładna definicja:

Dane: n rzeczy $R_1, R_2, \ldots, R_n$, każda z nich występuje w nieograniczonej ilości.

Rzecz R_i waży (zajmuje miejsce o wielkości) w_i jednostek i ma wartość p_i. maksymalna pojemność plecaka wynosi W jednostek.

Wynik: Ilości $q_1, q_2, \ldots, q_n$ poszczególnych rzeczy (mogą być zerowe), których całkowita waga (objętość) nie przekracza W jednostek.

Sformułowanie matematyczne problemu plecakowego ma postać:

Należy znaleźć wartości $q_1, q_2, \ldots, q_n$, dla których wartość sumy:

$$p_1q_1 + p_2q_2 + \ldots + p_nq_n \qquad (5.1)$$

jest największa, a łączna objętość spełnia warunek:

$$w_1q_1 + w_2q_2 + \ldots + w_nq_n \leq W \qquad (5.2)$$

gdzie $q_1, q_2, \ldots, q_n$ są nieujemnymi liczbami całkowitymi, a W jest "pojemnością plecaka".

Wartości $q_1, q_2, \ldots, q_n$ są *rozwiązaniem dopuszczalnym* jeżeli spełniają warunki (5.2),

a jeżeli suma (5.1) ma wartość maksymalną, to są one *rozwiązaniem optymalnym*.

Tak sformułowana wersja problemu plecakowego nosi nazwę *ogólna* – zakłada się, że liczba poszczególnych przedmiotów, jakie możemy wybrać jest nieograniczona.

W wersji *decyzyjnej* na rozwiązanie nakłada się dodatkowe ograniczenie - istnieje dokładnie jeden egzemplarz każdego przedmiotu. Należy każdorazowo podejmować decyzję, czy dany przedmiot wybrać, czy odrzucić.

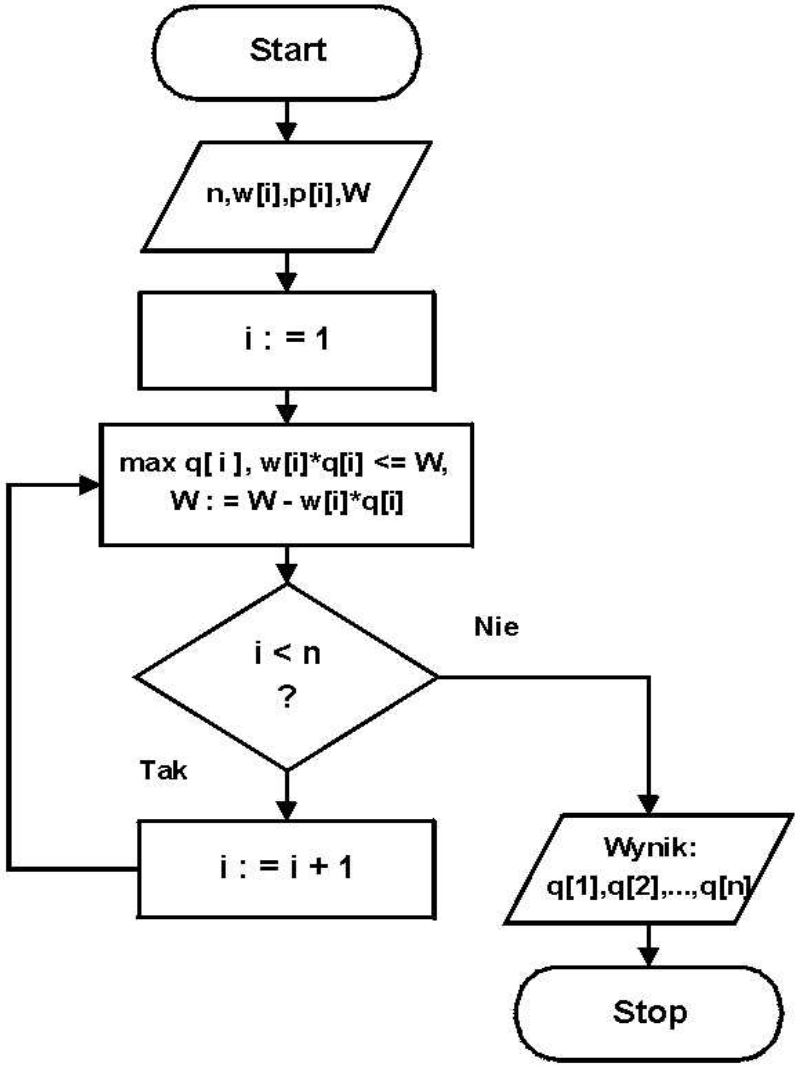
Kryteria wyboru kolejnych przedmiotów do "zapakowania" plecaka mogą być następujace:

- wybierać rzeczy najcenniejsze – w kolejności nierosnących wartości p_i,
- wybierać rzeczy zajmujące najmniej miejsca – w kolejności niemalejących wag w_i,
- wybierać rzeczy o największej wartości jednostkowej – w kolejności nierosnących wartości ilorazu p_i / w_i.

Wymienione wyżej kryteria stanowią strategię *zachłanną*.

Przykładowy schemat blokowy algorytmu zachłannego dla ogólnego problemu plecakowego (rys. 5.6) wykorzystuje trzecie spośród wymienionych wyżej kryteriów wyboru kolejnych przedmiotów:

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \quad \geq \frac{p_n}{w_n}$$



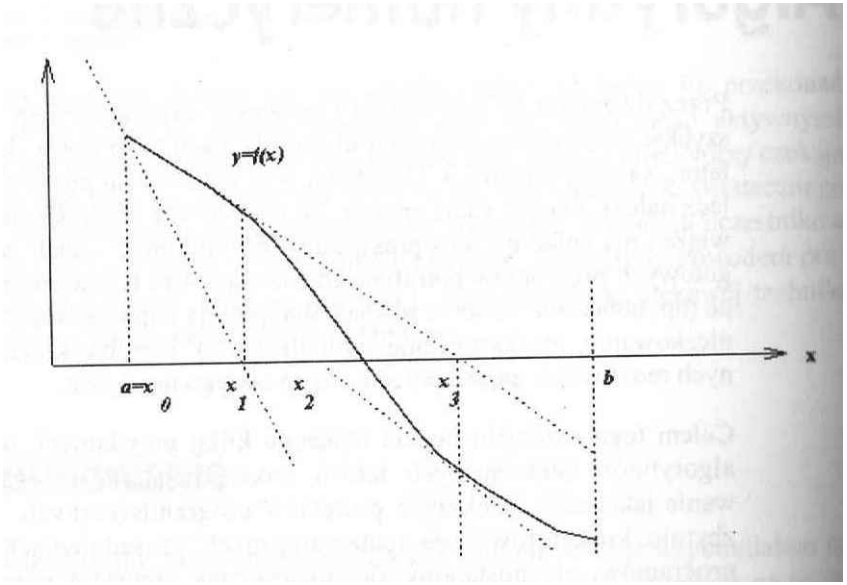
Rys.5.6. Schemat blokowy algorytmu zachłannego dla ogólnego problemu plecakowego

Poszukiwanie miejsc zerowych funkcji

Wiele metod numerycznych rozwiązania tego problemu – najprostsza to

metoda Newtona.

Polega na kolejnym zbliżaniu się do miejsca zerowego funkcji przy pomocy stycznych do krzywej (rys. 5.7).



Rys. 5.7. Algorytm Newton'a poszukiwania miejsc zerowych

Algorytm *Newtona* sprowadza się do iteracyjnego powtarzania następujących obliczeń:

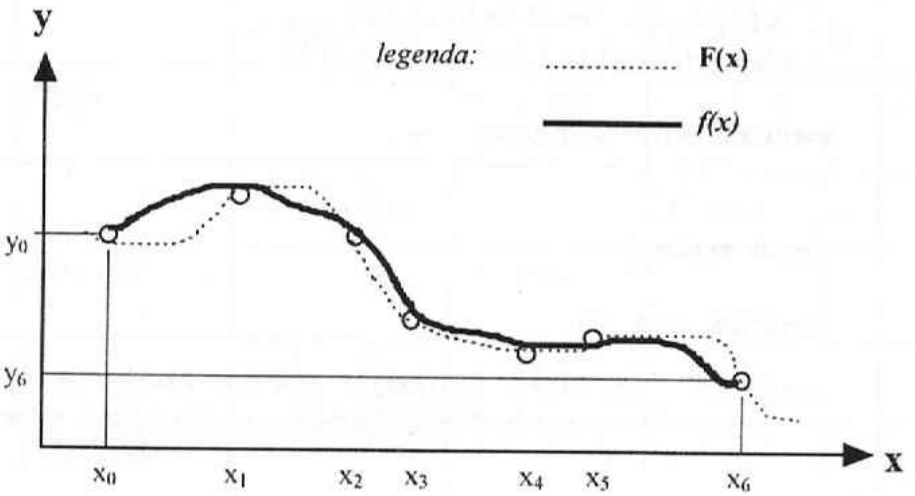
1^o $$z_i = z_{i-1} - \frac{f(z_{i-1})}{f'(z_{i-1})}$$

2^o *stop, jeżeli* $f(z_i) < \varepsilon$.

Gdzie ε jest zadaną stałą, która określa kiedy należy przerwać iteracje. Jest to dokładność przybliżenia miejsca zerowego funkcji f. W punkcie startowym inicjujemy obliczenia przyjmując jakąś wartość z_o. Definicje funkcji f i jej pierwszej pochodnej f' należy "wpisać" w program.

Interpolacja funkcji metodą Lagrange'a

Jeżeli dysponujemy "fragmentem" jakiejś funkcji – znamy jej wartości dla skończonego zbioru argumentów, możemy "przybliżyć" przebieg funkcji za pomocą innej mniej skomplikowanej funkcji (np. wielomianu) w ten sposób, że funkcja interpolująca przechodzi dokładnie przez punkty o zadanych współrzędnych (rys.5.8).



Rys. 5.8. Interpolacja funkcji f(x) za pomocą wielomianu F(x)

Znalezienie współczynników wielomianu interpolacyjnego wymaga skomplikowanych obliczeń tzw. wyznacznika Vandermond'a. Jeżeli potrzebujemy znać wartość tego wielomianu tylko w określonym punkcie z, to możemy skorzystać z metody Lagrange'a opisanej wyrażeniem:

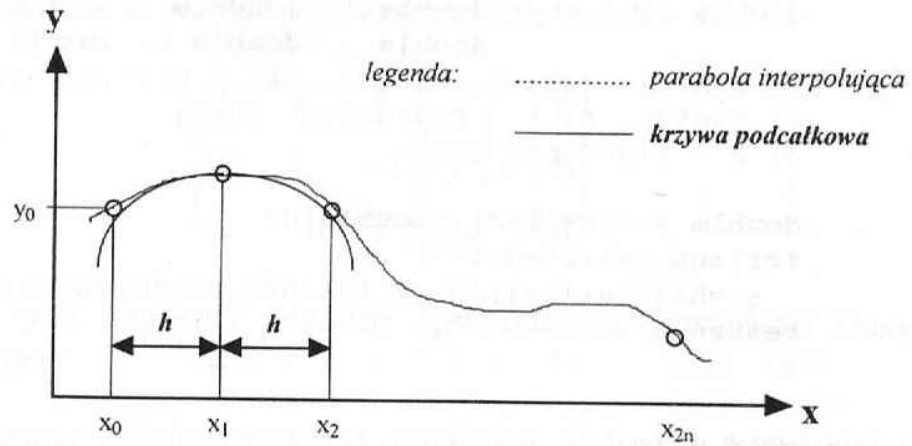
$$F(z) = (z - x_0)(z - x_1)...(z - x_n)\sum_{j=0}^{n} \frac{y_j}{(z - x_j)\prod_{i=0.i\neq j}^{n}(x_j - x_i)}$$

Całkowanie numeryczne

Obliczanie wartości całki oznaczonej może okazać się bardzo trudne ze względu skomplikowane, a czasem wręcz niemożliwe symboliczne obliczanie oryginału funkcji podcałkowej. Często zdarzają się sytuacje, kiedy znane są wartości funkcji podcałkowej tylko w niektórych punktach przedziału całkowania, Wtedy do rozwiązania takiego zadania korzysta się z procedur całkowania numerycznego.

Spośród wielu stosowanych w praktyce metod – metoda trapezów, metoda Romberga, metoda Simpsona itp. Dość dobrą dokładnością charakteryzuje się metoda Simpsona. Rys. 5.9. przedstawia ideę tej metody. W pokazanym na rysunku przedziale całkowania przybliżoną wartość całki oznaczonej oblicza się wg formuły:

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}\left(f(x_0)+4f(x_1)+f(x_2)\right)$$



Rys. 5.9. Idea całkowania numerycznego metodą Simpsona

Wystarczy obliczyć wg pokazanego wyżej wzoru całkę "częściową" w kolejnych podprzedziałach 3 punktowych, na które podzielono przedział całkowania od a do b.

Przy podziale na 2n odcinków mamy h = (b – a)/2n. Łączna wartość pola powierzchni pod krzywą f(x) w przedziale a do b będzie sumą wartości "całek częściowych".

Dokładność obliczeń zależy oczywiście od liczby trzypunktowych przedziałów cząstkowych.

W tego typu metodach przyjmuje się zwykle jako miarę dokładności uzyskanego wyniku względny błąd ε jaki popełniamy kończąc obliczenia po wykonaniu np. n podziałów przedziału całkowania.

Powtarzanie obliczeń kończy się jeżeli zostanie spełniony warunek:

$$\frac{\left|F_{n+1} - F_n\right|}{\left|F_n\right|} \leq \varepsilon$$

gdzie F_n , F_{n+1} – sumy całek częściowych odpowiednio po n i n+1 podziałach.