BFS

Kolejka

Lista sasiadow

Tablica odwiedzonych

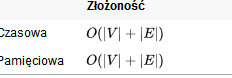
•przeglądanie grafu z kolejką nazywamy metodą wszerz(BFS, czyli breadth-firstsearch)

•to, czy wierzchołek oznaczamy jako obsłużony w momencie wejścia do czy wyjścia z kolejki nie wpływa na kolejność przetwarzania

•rozmiar kolejki jest ograniczony przez |V| (wejście) lub |E| (wyjście)

•liczba odwołań do wierzchołków to O(|E|)

Ponieważ w najgorszym przypadku przeszukiwanie wszerz musi przebyć wszystkie krawędzie prowadzące do wszystkich węzłów





DFS

Stos niejawny rekurencji

Lista sasiadow

Kolory (biały-nieodwiedzony, szary-przetwarzany, czarny-przetworzony)

czasy wejscia

czasy wyjscia

tablica ojcow

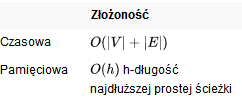
•przeglądanie grafu ze stosem i oznaczaniem wierzchołka w momencie zdjęcia nazywamy metodą w głąb(DFS, czyli deapth-firstsearch)

•to, czy wierzchołek oznaczamy jako obsłużony w momencie położenia czy zdjęcia ze stosu wpływa na kolejność przetwarzania

•rozmiar stosu jest ograniczony przez |V| (położenie) lub |E| (zdjęcie)

•liczba odwołań do wierzchołków to O(|E|)

Algorytm musi odwiedzić wszystkie wierzchołki oraz wszystkie krawędzie, co oznacza, że złożoność wynosi O(|V|+|E|).



****

Find union

**Struktura zbiorów rozłącznych** to [struktura danych](https://pl.wikipedia.org/wiki/Struktura_danych), która przechowuje dla ustalonego [zbioru](https://pl.wikipedia.org/wiki/Zbi%C3%B3r) (uniwersum) jego podział na mniejsze, rozłączne zbiory. Struktura taka umożliwia dwie operacje:

* **Find**: Wyznacza, w którym zbiorze jest dany element, pozwalając na sprawdzenie, czy dwa elementy są w tym samym zbiorze.
* **Union**: Łączy dwa zbiory w jeden.

*Find* wymaga w pesymistycznym przypadku przeszukania wszystkich list (czas O ( n ) {\displaystyle O(n)} ).

Możliwą modyfikacją tej metody jest trzymanie w każdym węźle listy wskaźnika do jej początku – wtedy *Find* działa w czasie stałym, zaś *Union* wymaga poprawienia takich wskaźników na jednej z łączonych list. Jeśli dołącza się zawsze mniejszą listę do większej, operacja *Union* działa w zamortyzowanym czasie O ( log ⁡ n ) {\displaystyle O(\log n)}  (innymi słowy, sekwencja m {\displaystyle m} m operacji na tej strukturze dla n {\displaystyle n} n elementów działa łącznie w czasie O ( m + n log ⁡ n ) {\displaystyle O(m+n\log n)} ).

Naszym zadaniem jest określać, w dowolnym momencie, czy dwa elementy znajdują się w tej samej grupie. Mówiąc formalnie, musimy zaimplementować dwie operacje:

* *Union(x,y)* — połącz grupy, w których są elementy x i y, w jedną grupę,
* *Find(x)* — znajdź grupę elementu x. W praktyce chodzi o sprawdzenie, czy dwa elementy są w tej samej grupie (sprawdzamy zawsze warunek, czy *Find(x) = Find(y)* dla pewnych dwóch elementów x,y).

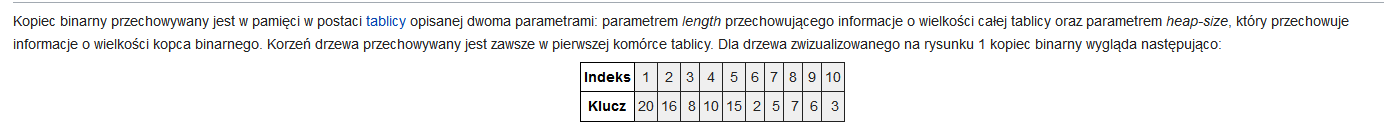
Kopiec

tablica pary

Kopiec binarny (zwykły) to drzewo binarne, w zachowana jest w nim zależność między węzłem a jego potomkami.

W kopcu klucz rodzica jest niewiększy od kluczy jego dzieci (co implikuje taką zależność w stosunku do wszystkich potomków).

Kopiec binarnymożna zapisać i przetwarzać jako wektor tworzący ze statycznymi połączeniami, będzie on miał najmniejszą możliwą wysokość dla danej liczby elementów i łatwy dostęp do liści.



### Dodawanie nowych wierzchołków

Załóżmy, że kopiec składa się z *n* elementów, zaś elementy uporządkowane są od największych (warunek kopca brzmi więc: każdy element jest większy od swoich dzieci). Dodawany wierzchołek ma klucz równy *k*:

1. wstaw wierzchołek na pozycję *n*+1
2. zamieniaj pozycjami z rodzicem (przepychaj w górę) aż do przywrócenia warunku kopca (czyli tak długo, aż klucz rodzica jest większy niż *k*, lub element dotrze na pozycję 1)

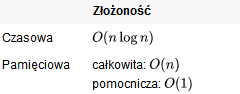
Dodawanie nowego wierzchołka w pseudokodzie:

### Usuwanie wierzchołka ze szczytu kopca

1. usuń wierzchołek ze szczytu kopca
2. przestaw ostatni wierzchołek z pozycji *n* na szczyt kopca; niech *k* oznacza jego klucz
3. spychaj przestawiony wierzchołek w dół, zamieniając pozycjami z większymi z dzieci, aż do przywrócenia warunku kopca (czyli aż dzieci będą mniejsze od *k* lub element dotrze na spód kopca)

Zarówno wstawianie jak i usuwanie obiektów ze szczytu kopca ma złożoność *O*(log*n*).

Sortowanie na kopcu



Sortowanie topologiczne

Lista sasiadow

Kolejka

jezeli ktos nie ma sasiadow to dodajemy go do kolejki

dopoki kolejka nie jest pusta idziemy do kolejnych sasiadow i zmniejszamy ich ilosc sasiadow

jezeli ktorys sasiad nie ma juz drog skierowanych w swoja strone to dodajemy go do kolejki

jezeli przejzelismy wszystkich sasiadkow to usuwamy wierzcholek z kolejki

•Acykliczne grafy skierowane można posortować topologicznie

•Wynikiem jest dowolny ciąg wierzchołków, w którym nie ma żadnych dróg od późniejszych do wcześniejszych (linearyzacja częściowego porządku będącego domknięciem tranzytywnym)

•Można do tego zaadaptować BFS, wstawiając do kolejki wszystkie wierzchołki o stopniu wejściowym 0, a później obniżać stopnie wejściowe oglądanych sąsiadów i dodawać ich do kolejki, kiedy osiągną 0

Drzewo rozpinające

Drzewem rozpinającym nieskierowanego grafu spójnego G=(V,E) nazywamy dowolny jego podgraf spójny T=(V,E'), który jest drzewem.

ST(G) -zbiór wszystkich drzew rozpinających grafu G Najkrótsza

Minimalne drzewo rozpinające

dane: Graf nieskierowany G=(V,E) oraz funkcja w:E→R przyporządkowująca krawędziom wagi

wynik: (V,Em) –minimalny element ST(G),wszystkie takie drzewa oznaczymy przez MST(G)

rozmiar danych: |V| + |E|

definicja formalna: ∀(V,E')∈ST(G) ∑e∈E'≤ ∑e∈Em

Algorytm Kruskala

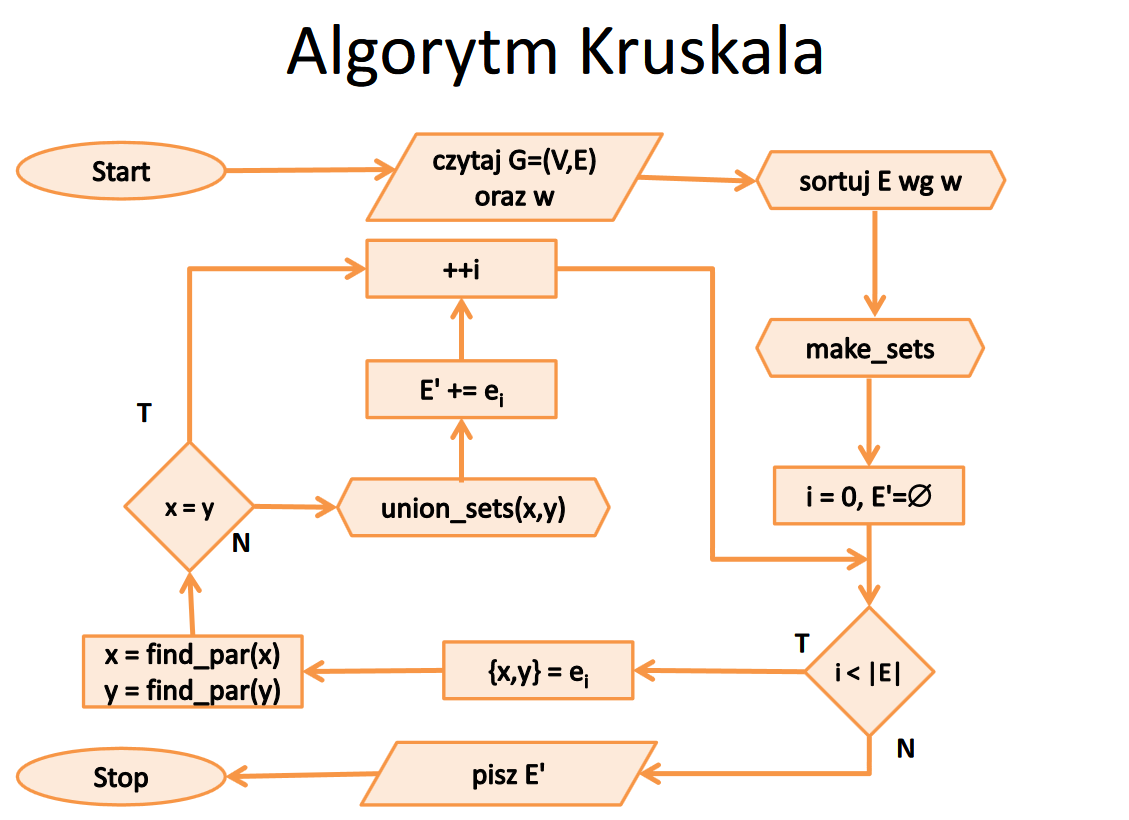
(minimalne drzewo rozpinające)

Graf ważony spójny o dodatnich wagach

znajduje drzewo zawierające wszystkie wierzchołki grafu, którego waga jest najmniejsza możliwa

•Posortuj krawędzie ze zbioru E względem wagi w(e).

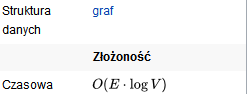
•Buduj podgraf G' przeglądaj te krawędzie i dodając je do G' jeśli nie powoduje to stworzenia cyklu.

*•Testowanie cykli można zrealizować za pomocą struktury zbiorów rozłącznych. *

## Algorytm

* Utwórz las L z wierzchołków oryginalnego grafu – każdy wierzchołek jest na początku osobnym drzewem.
* Utwórz zbiór S zawierający wszystkie krawędzie oryginalnego grafu.
* Dopóki S nie jest pusty oraz L nie jest jeszcze drzewem rozpinającym:
  + Wybierz i usuń z S jedną z krawędzi o minimalnej wadze.
  + Jeśli krawędź ta łączyła dwa różne drzewa, to dodaj ją do lasu L, tak aby połączyła dwa odpowiadające drzewa w jedno.
  + W przeciwnym wypadku odrzuć ją.

Po zakończeniu algorytmu L jest minimalnym drzewem rozpinającym.



Algorytm Prima

wyznaczający tzw. minimalne drzewo rozpinające

graf nieskierowany i spójny

•Przeglądaj graf G=(V,E) używając jako kontenera kolejki priorytetowej

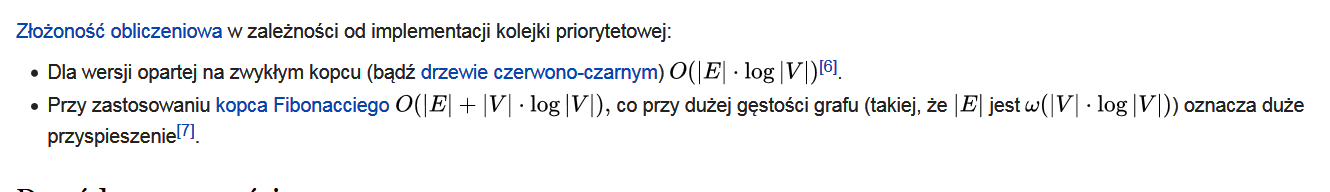
•Kolejne przeglądane wierzchołki, wraz z krawędziami używanymi w przeglądaniu, budują poszukiwany podgraf

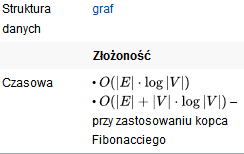
•Wagi ponownie odwiedzanych wierzchołków, które jeszcze nie zostały dodane do drzewa należy poprawiać lub ponownie dodawać wierzchołek do kontenera

## Algorytm

Schemat działania:

* Utwórz drzewo zawierające jeden wierzchołek, dowolnie wybrany z grafu.
* Utwórz kolejkę priorytetową, zawierającą wierzchołki osiągalne z MDR (w tym momencie zawiera jeden wierzchołek, więc na początku w kolejce będą sąsiedzi początkowego wierzchołka), o priorytecie najmniejszego kosztu dotarcia do danego wierzchołka z MDR.
* Powtarzaj, dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:
  + wśród nieprzetworzonych wierzchołków (spoza obecnego MDR) wybierz ten, dla którego koszt dojścia z obecnego MDR jest najmniejszy.
  + dodaj do obecnego MDR wierzchołek i krawędź realizującą najmniejszy koszt
  + zaktualizuj kolejkę priorytetową, uwzględniając nowe krawędzie wychodzące z dodanego wierzchołka





Poprawność obu algorytmów

Podgraf bezpieczny, to podgraf który możemy rozszerzyć do MST

Krawędź bezpieczna dla podgrafu bezpiecznego, to taka krawędź, którą możemy dodać do podgrafu aby nadal był bezpieczny

droga w grafie

dane: Graf skierowany G=(V,E) oraz funkcja w:E→R przyporządkowująca łukom wagi, dwa zbiory wierzchołków S⊆V oraz T⊆V

wynik: A∈M|S|×|T|– długości minimalnych dróg (lub informacja o braku drogi -∝)rozmiar danych: |V| + |E|

definicja formalna:

∀P=(vi,e1,...,vj)-droga prosta ∑ek∈P w(ek) ≤ A[i,j]

Wersje problemu

•Najkrótsze drogi z jednym źródłem (|S|=1, T=V)

•Najkrótsze drogi z jednym celem(S = V, |T|=1)

•Najkrótsza droga między parą wierzchołków(|S|=|T|=1)

•Najkrótsze drogi między wszystkimi parami(S = T = V)

Wersje problemu

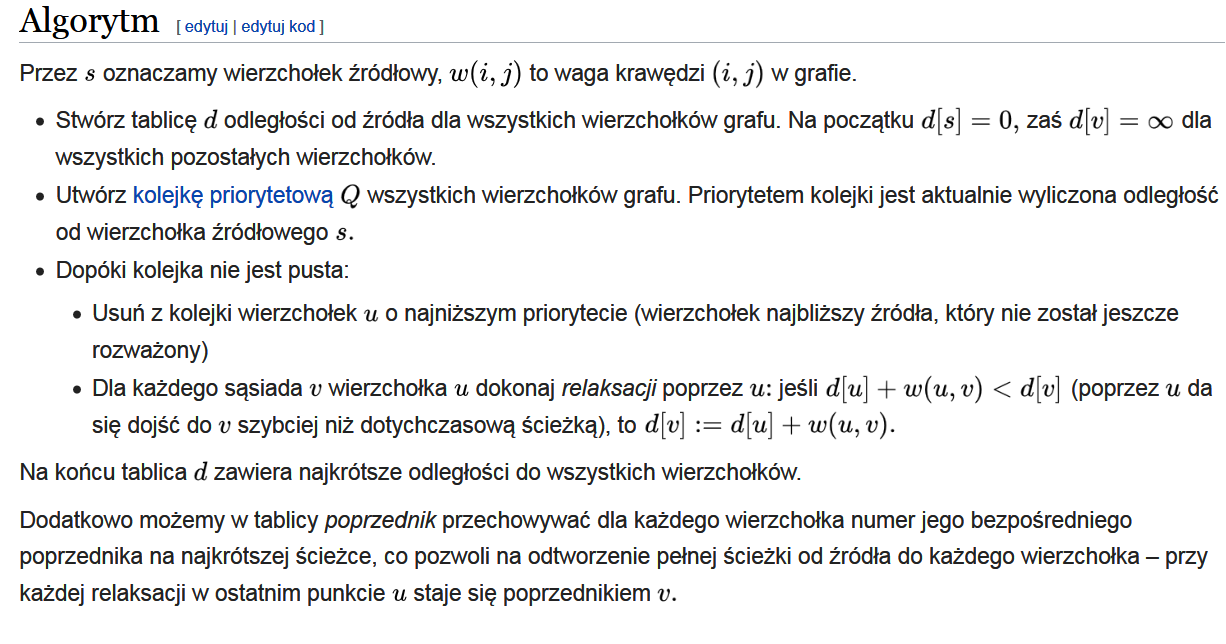
•Rozwiązanie dla S = T = V jest najbardziej ogólne i daje wszystkie pozostałe rozwiązania

•Problemy dla |S|=1 i T = V oraz S = V i |T|=1 są dualne (identyczne dla grafów nieskierowanych)

•Problem dla |S|=|T|=1 jest najbardziej szczegółowe i można je rozwiązać korzystając z rozwiązań któregokolwiek z pozostałych problemów (być może przerywając je wcześniej)

Idea algorytmu Dijkstry

znajdowania [najkrótszej ścieżki](https://pl.wikipedia.org/wiki/Problem_najkr%C3%B3tszej_%C5%9Bcie%C5%BCki) z pojedynczego źródła w [grafie](https://pl.wikipedia.org/wiki/Graf_(matematyka)) o nieujemnych wagach krawędzi.



•Utwórz tablicę A z aktualnie wyliczonymi najkrótszymi drogami z wierzchołka v0,

•Zainicjuj tablicę A wartościami +∝

•Ustaw A[v0]=0

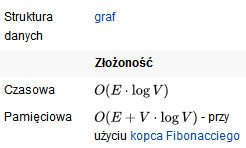
•Dopóki istnieje choć jeden nieobsłużony wierzchołekv, taki że A[v] ≠+∝

–Weź nieobsłużone v o minimalnym A[v]

–Oznacz v jako obsłużony

–Dla każdego v', który jest sąsiadem v relaksuj A, czyli

•Jeśli A[v]+w(v,v') < A[v'] to ustaw A[v']=A[v]+w(v,v')



Analiza złożoności

•Złożoność zależy od struktury grafu

•Dla grafów gęstych opłaca się użyć macierzy sąsiedztwa, wówczas element minimalny wśród wierzchołków nieobsłużonych może być liczony naiwnie (liniowo)

–Złożoność czasowa: O(|V|2) = O(|E|)

–Złożoność pamięciowa: O(|V|+|E|)

•Dla grafów rzadkich lepsze są listy sąsiedztwa, wówczas element minimalny opłaca się uzyskiwać z kolejki priorytetowej

–Złożoność czasowa: O(|E|lg|V|) dla zwyczajnego kopcaO(|E|+|V|lg|V|) dla kopca Fibonacciego

–Złożoność pamięciowa: O(|V|+|E|)

•Dla grafów bez ujemnych wag (algorytm Dijkstry)

–Złożoność czasowa to O(|E||V| + |V|2lg|V|)

–Złożoność pamięciowa to O(|V|2)

•Algorytmy dla grafów bez ujemnych wag:

–Z jednym źródłem: O(|E|+|V|lg|V|)

–Dla wszystkich par: O(|V||E|+|V|2lg|V|)

Przeglądanie drzewa

Klasyczne metody przeglądania grafów

•Wierzchołki są zawsze dodawane przez rodzica

•Nie trzeba sprawdzać przetworzenia wierzchołka

•Stos jako kontener –BFS

–można użyć stosu rekurencyjnego

–kilka metod przeglądania

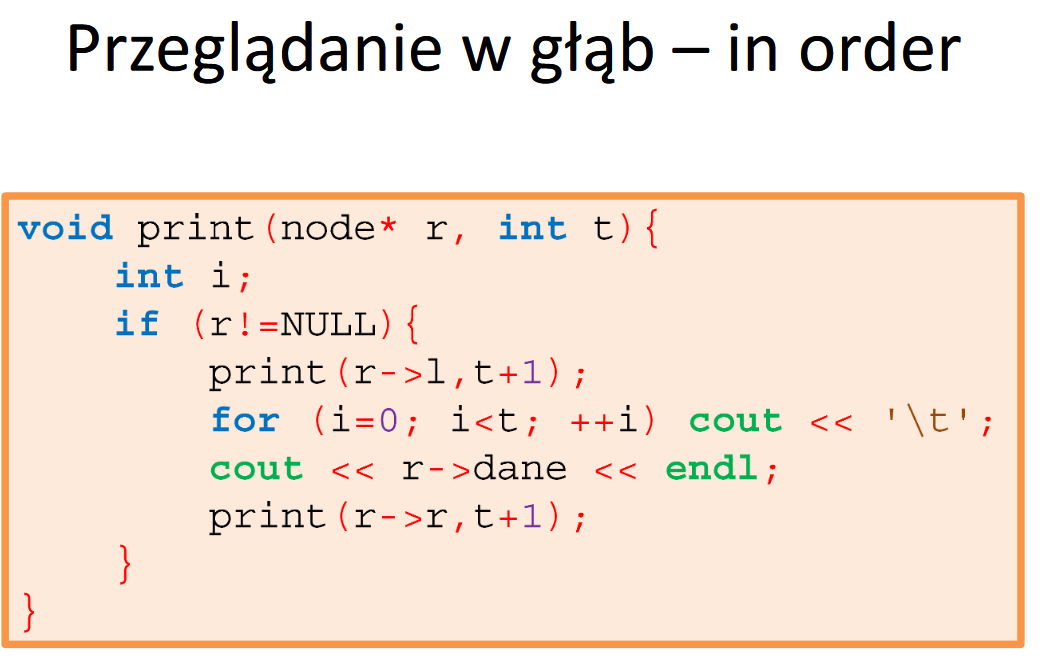
•rozdzielenieprzetwarzania (drugi odczyt) od wstawiania nowych elementów do kontenera (pierwszy odczyt)

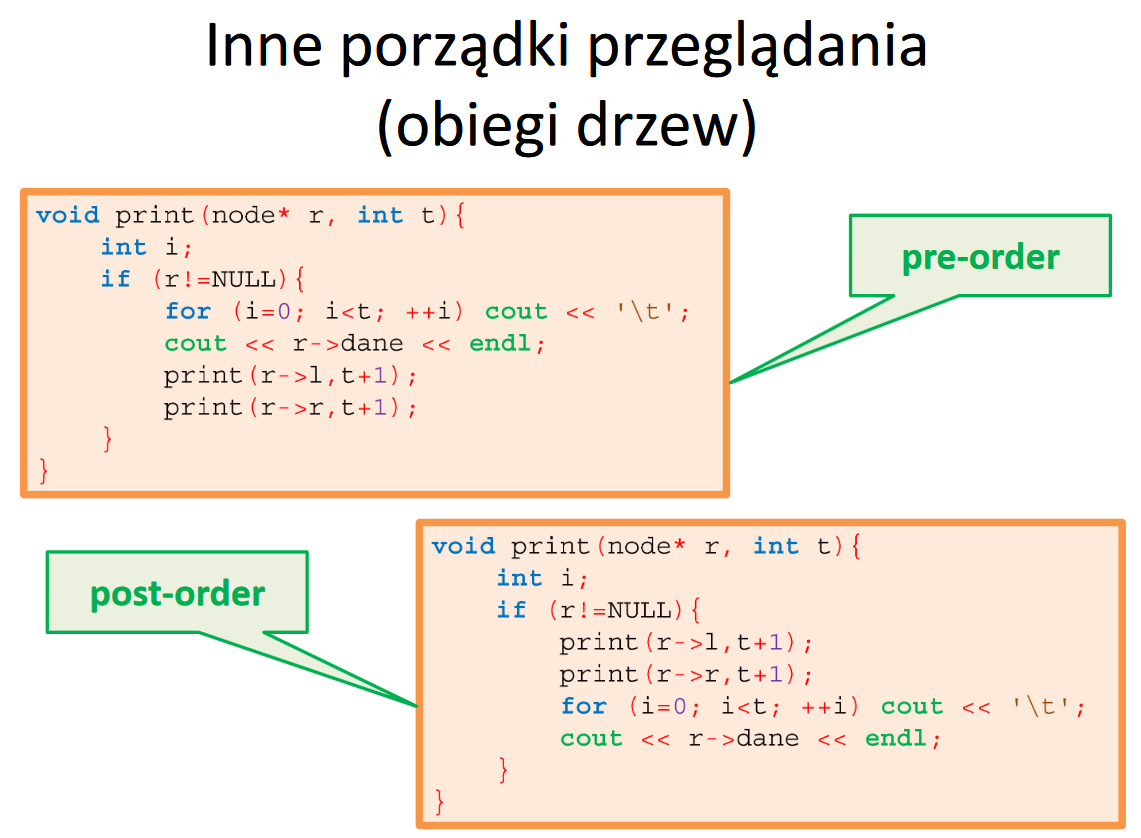
•różna kolejność wstawiania

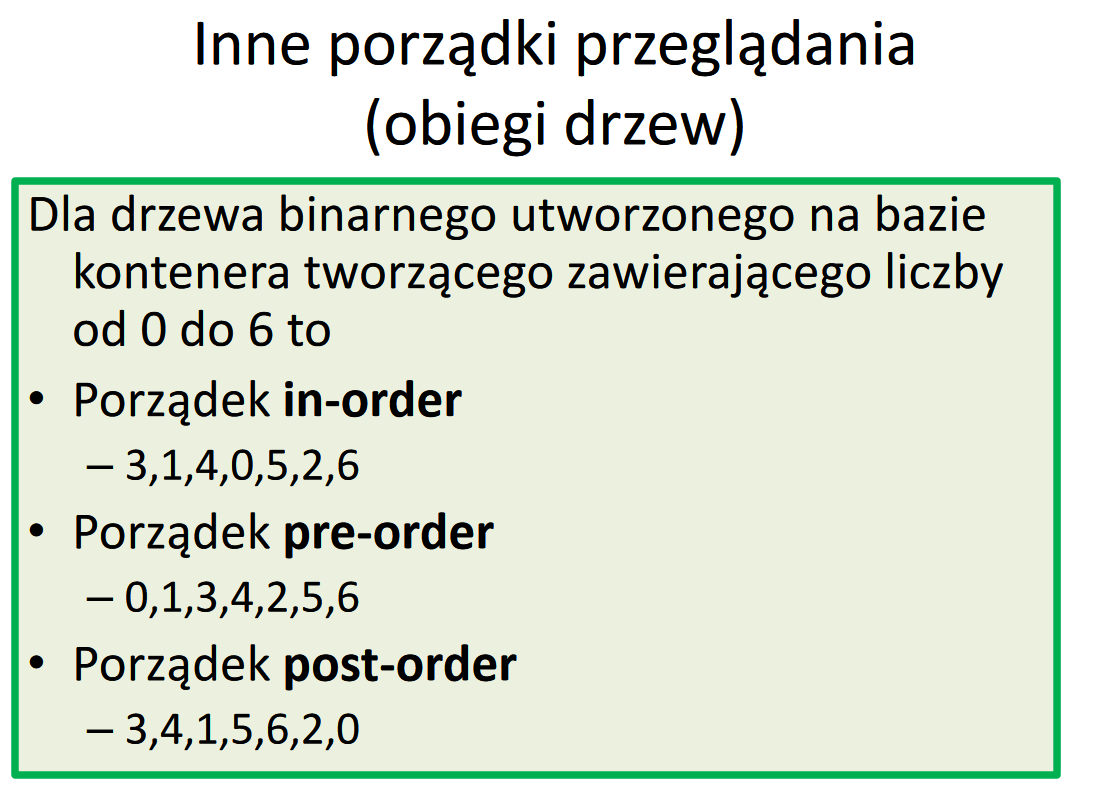
•Kolejka jako kontener –DFS

–przeglądanie warstwami

–kolejność jak w wektorze tworzącym (przykład)







Drzewo poszukiwań binarnych(BST)

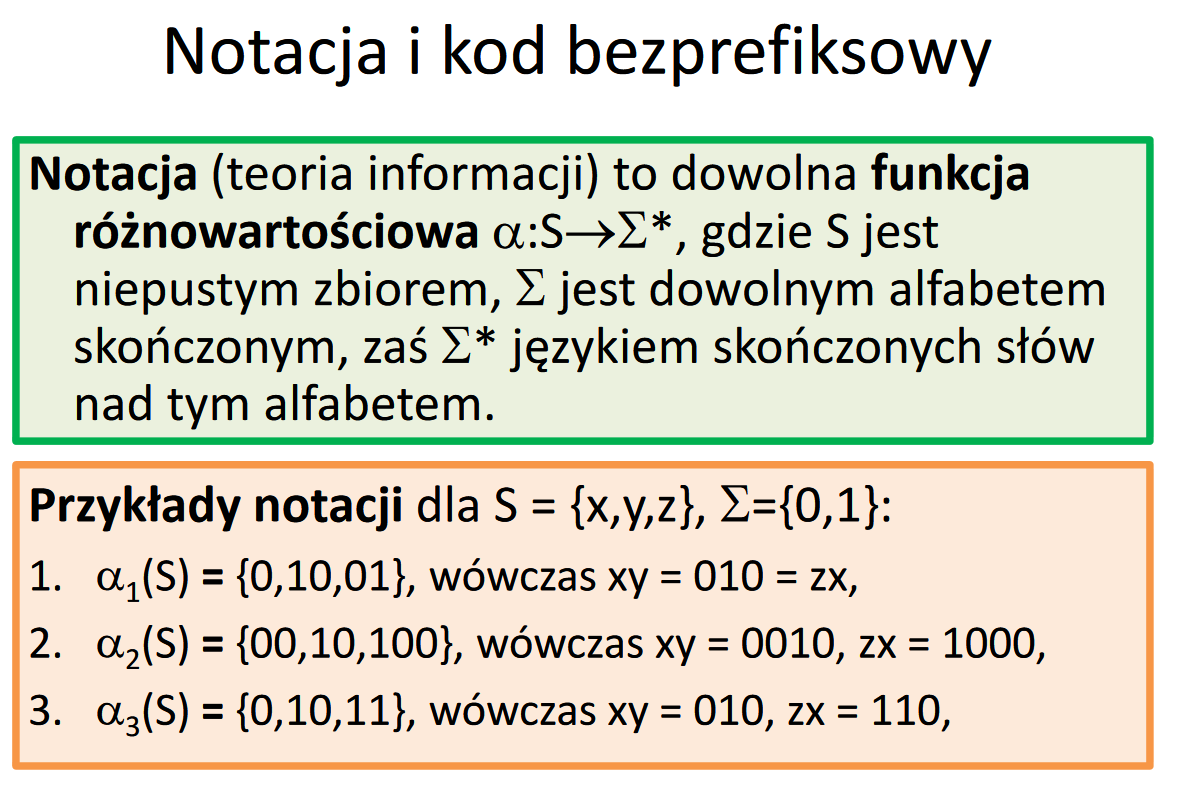
Koszt wykonania podstawowych operacji w drzewie BST jest proporcjonalny do wysokości drzewa h.  dla drzewa o n {\displaystyle n} n węzłach optymistyczny koszt każdej z podstawowych operacji wynosi

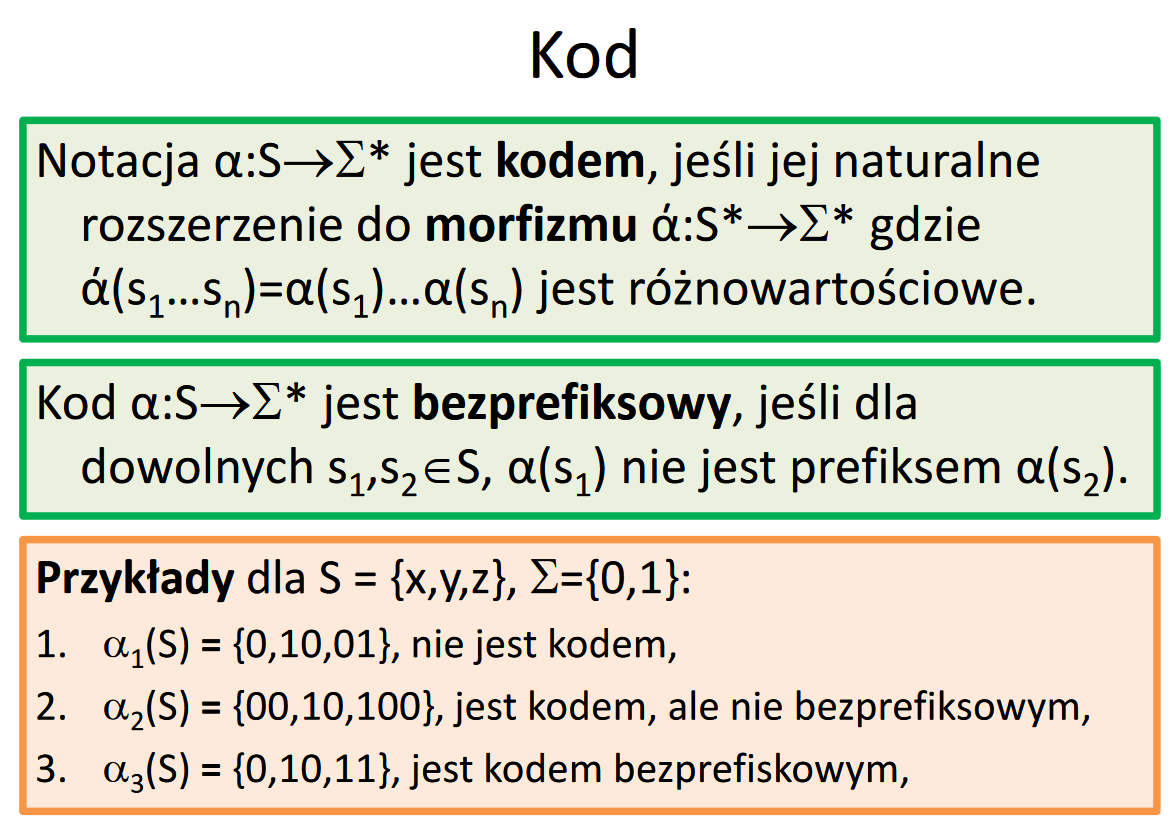
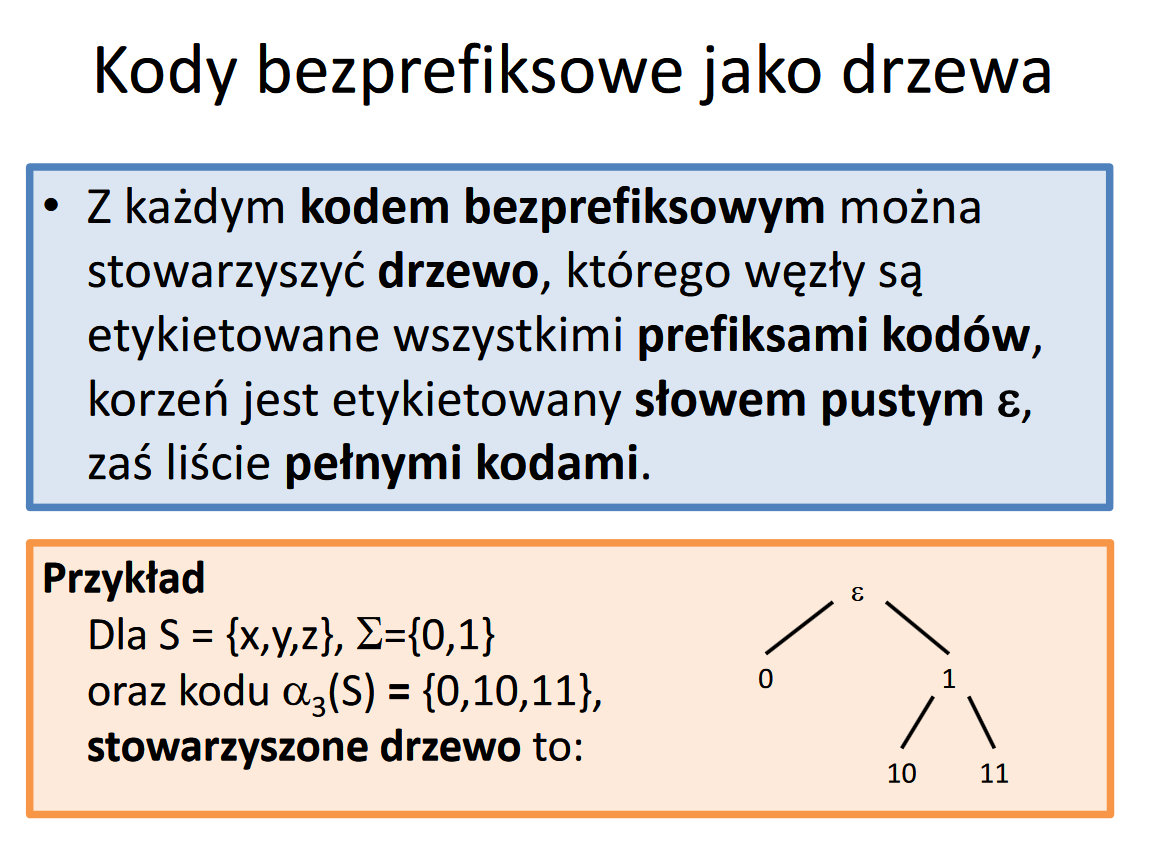
 h , {\displaystyle h,} hhhg

dynamiczna struktura danych będąca [drzewem binarnym](https://pl.wikipedia.org/wiki/Drzewo_binarne), w którym lewe poddrzewo każdego węzła zawiera wyłącznie elementy o kluczach mniejszych niż klucz węzła a prawe poddrzewo zawiera wyłącznie elementy o kluczach nie mniejszych niż klucz węzła. Węzły, oprócz klucza, przechowują wskaźniki na swojego lewego i prawego syna oraz na swojego ojca.

Drzewo poszukiwań binarnych(BST) to drzewo binarne w którym zachowana jest zależność między węzłem a jego potomkami.

W BST klucz rodzica jest niemniejszy od kluczy lewych potomków oraz niewiększy od kluczy prawych potomków.

Drzewo poszukiwań binarnych jest zbudowane efektywnie jeśli jest drzewem zrównoważonym(o najmniejszej możliwej wysokości), gdyż podstawowe operacje zależą od tej wysokości. 

 Optymalne kodowanie

dane: słowo x=x1x2...xn;

wynik: minimalne kodowanie bezprefiksowedla x

;rozmiar danych: n;

definicja formalna: poszukujemy α:alph(x)→{0,1}\*takiego, że dla każdego kodowania bezprefiksowegoβ:alph(x)→{0,1}\*mamy #(β(x))≥#(α(x))

algorytm Huffmana

metoda [kompresji bezstratne](https://pl.wikipedia.org/wiki/Kompresja_bezstratna)

•Analogicznie do minimalnego sklejania par

•Rozwiązanie zachłanne

•Kolejne sklejane elementy mają identyczne prefiksy

•Sklejenie wydłuża długość kodu o jeden bit na każdy znak, czyli o sumę sklejanych wartości

•Budujemy drzewo kodu od liści, ostatnie sklejanie „za darmo"

Algorytm Huffmana

•Drzewo Huffmana można utworzyć algorytmem zachłannym, złożoność czasowa to O(n lgn)

•Jeśli zbiór wag częstości występowania liter jest posortowany, to można je utworzyć szybciej korzystając z dwóch kolejek FIFO i postępując podobnie jak w scalaniu, złożoność czasowa to O(n)



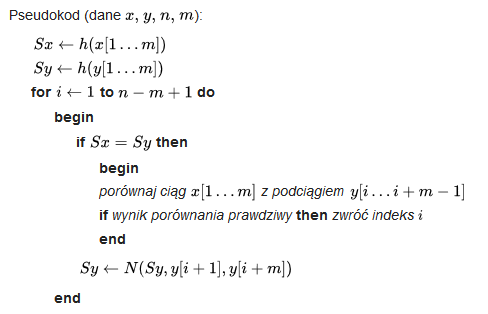
Algorytm Karpa-Rabina

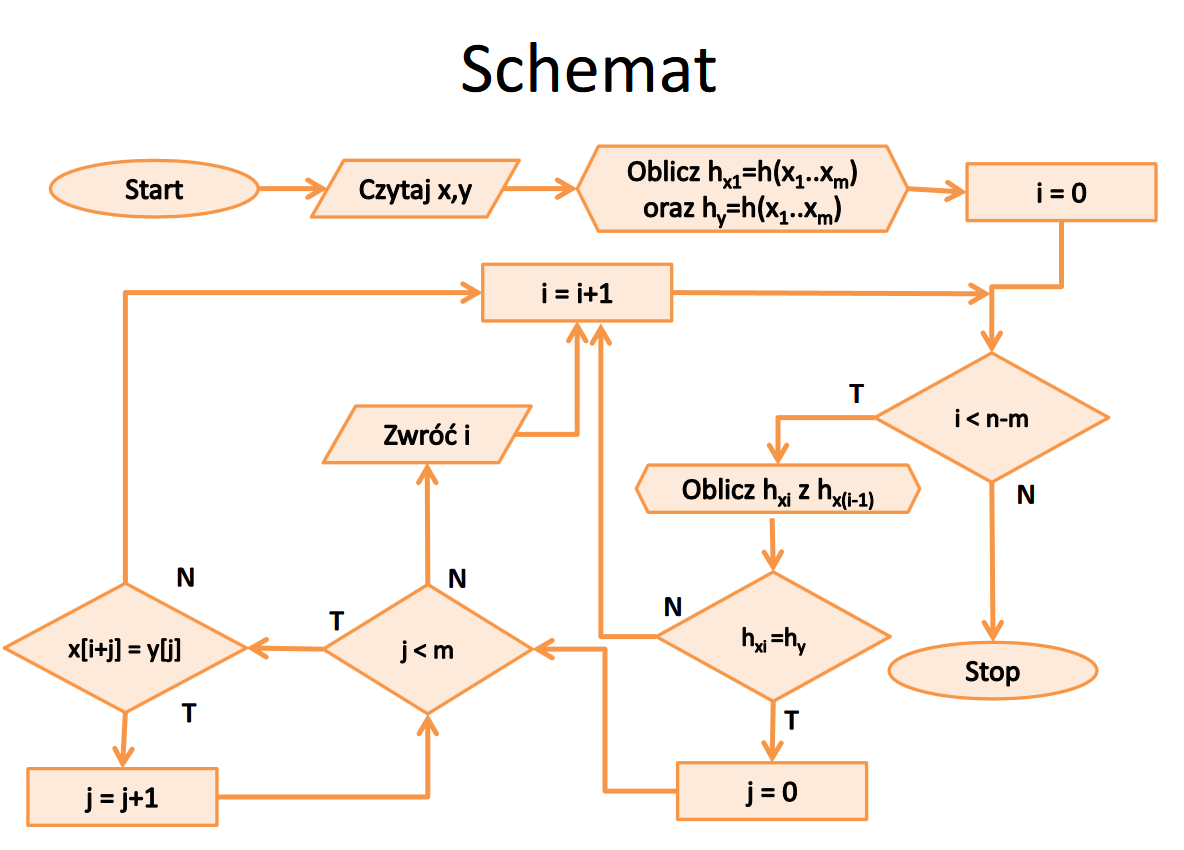
[algorytmem](https://pl.wikipedia.org/wiki/Algorytm) dopasowania wzorca – służy do lokalizowania w tekście określonego podciągu

•Dzięki porównywaniu funkcji skrótu dla okna rozmiaru m, można uniknąć wewnętrznej pętli, nawet gdy duża część wzorca pasuje

•Prosta funkcja skrótu dla wzorca y to h(y1y2...ym) = (dm-1y1+dm-2y2+...+d0ym) mod p, gdzie d to baza (zazwyczaj rozmiar alfabetu), zaś p moduł (zazwyczaj liczba pierwsza)

•Możemy wyrazić h(xi+1xi+2...xi+m) jako ((h(xixi+1...xi+m-1) -dm-1xi)⋅d+ xi+m) mod p. 



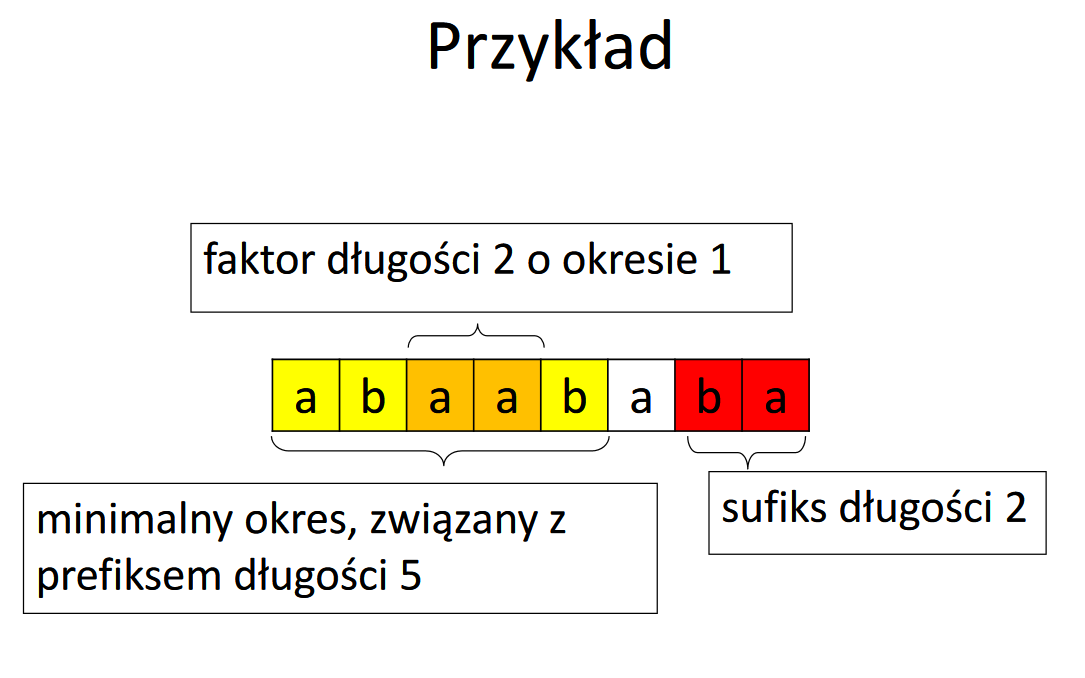


Faktory

Faktorem słowa w nazywamy dowolny zwarty fragment słowa w. Faktorami trywialnymi słowa w są całe słowo w oraz słowo puste ε.

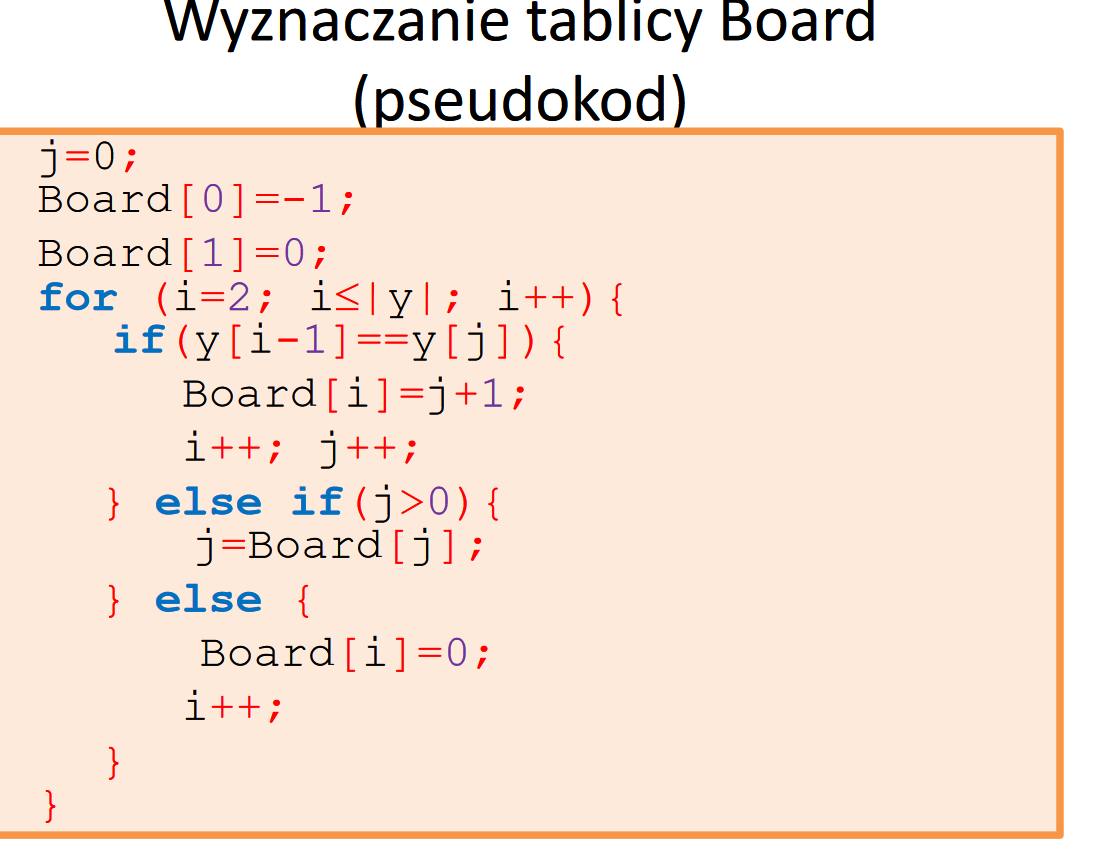
Prefiksy i sufiksy to szczególne faktory słowa –prefiksy znajdują się na początku słowa, zaś sufiksy znajdują się na końcu słowa. Prefiks długości n słowa w oznaczamy w(n).

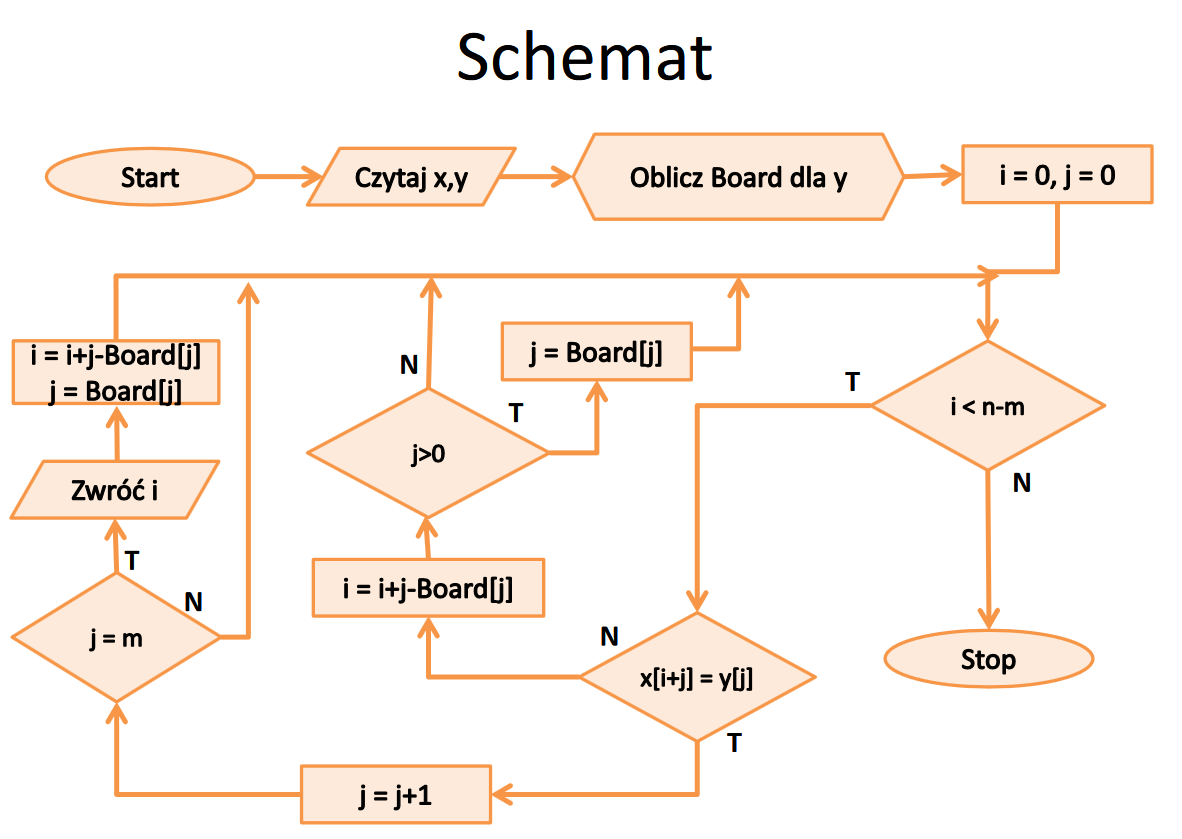
Okresem słowa w nazywamy długość jego prefiksu u, który powtarza się do końca słowa (w[i]=w[i+#u])

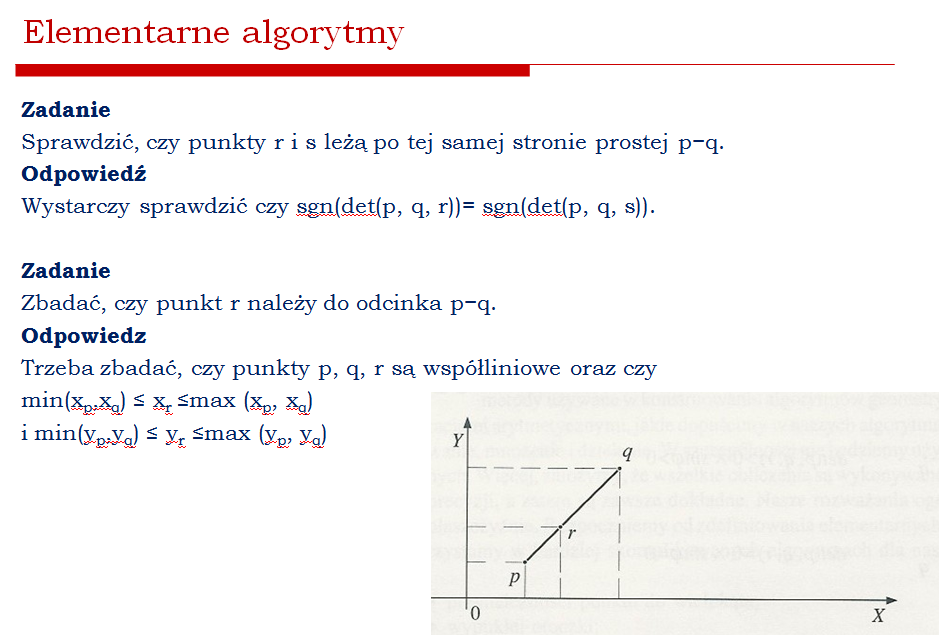
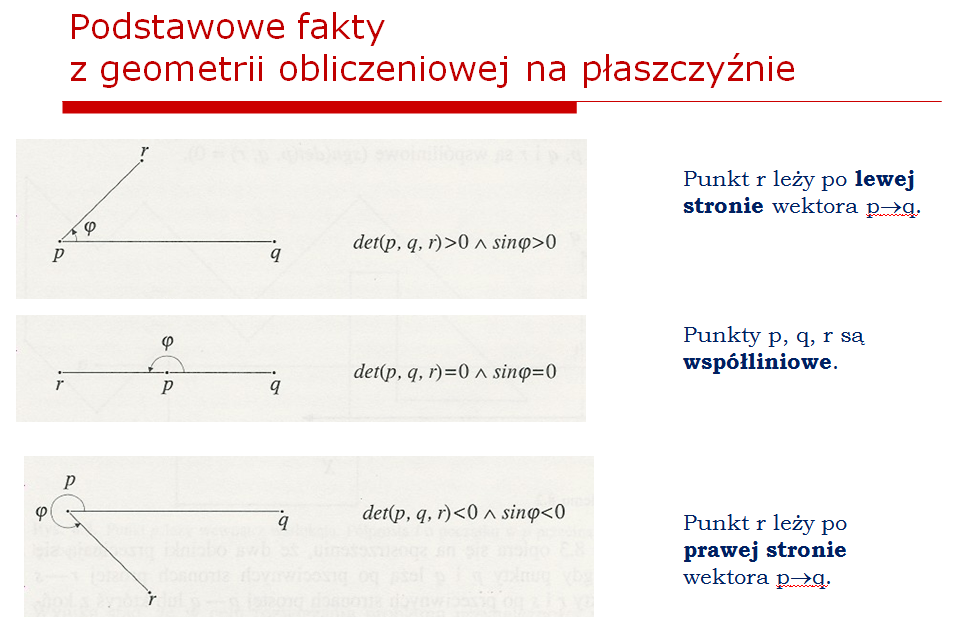
Tablica prefikso-sufiksów

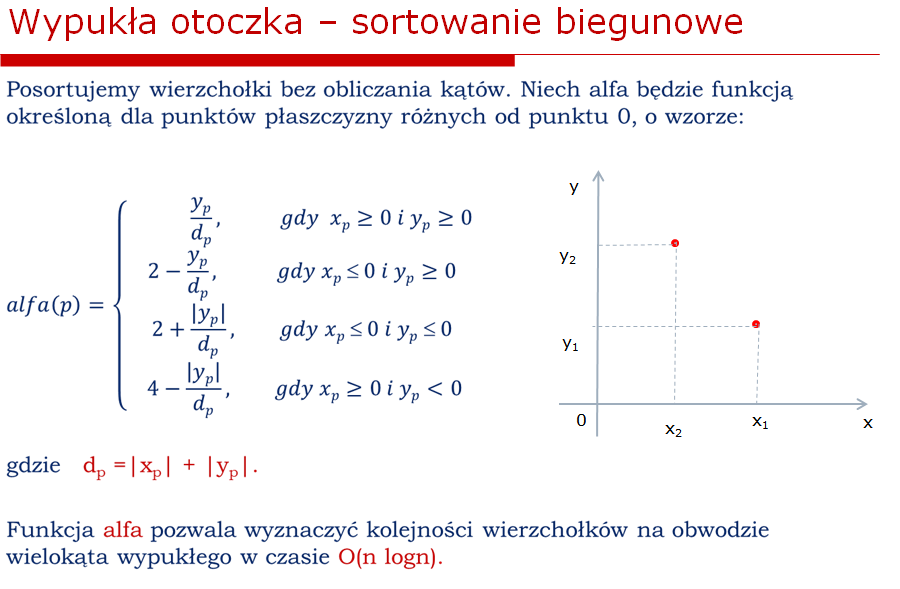
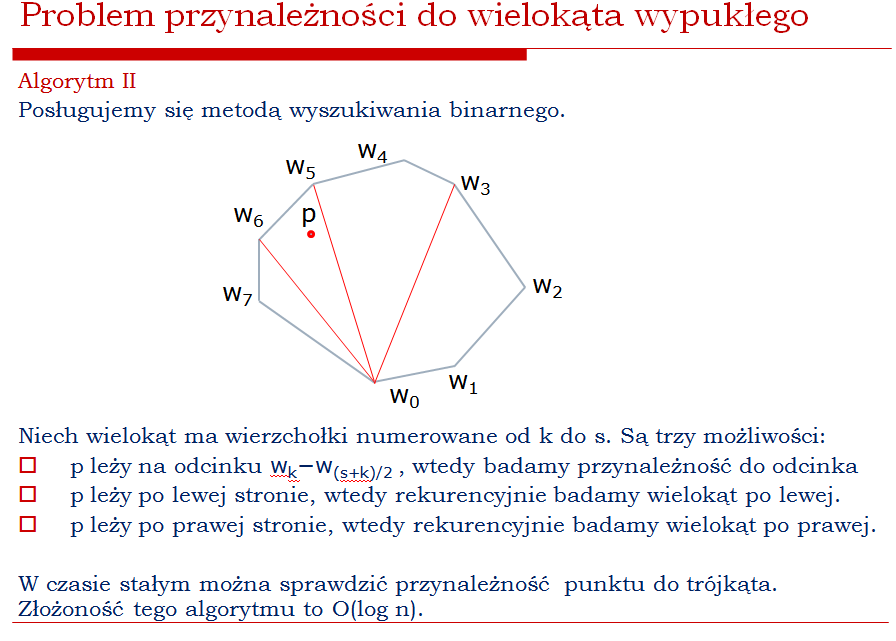
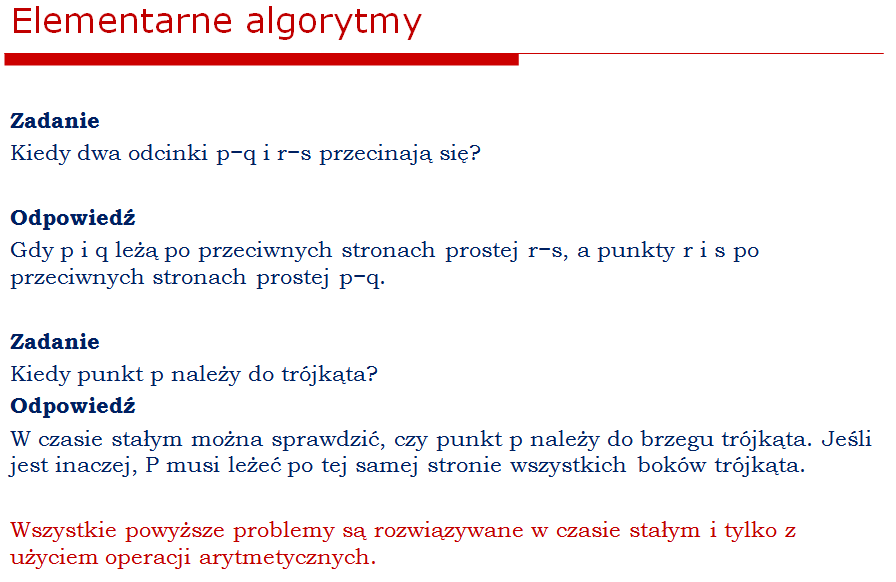
Prefikso-sufiksem słowa w nazywamy jego właściwy (różny od w) prefiks, który jednocześnie jest jego sufiksem. Każde słowo posiada przynajmniej jeden prefikso-sufiks długości 0.

Funkcja Board:[1..|w|]→[0..|w|] wyznacza długości maksymalnych prefikso-sufiksów kolejnych prefiksów słowa w.



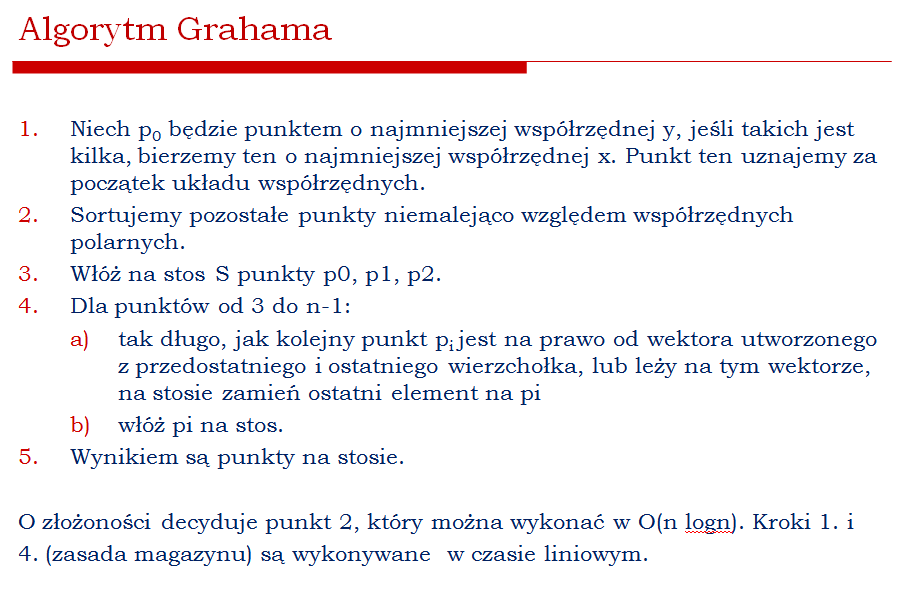




* 

Algorytm Grahama

* W algorytmie Grahama używamy stosu, który zawiera kandydatów na wierzchołki otoczki.
* Każdy punkt z wejściowego zbioru jest raz wkładany na stos, natomiast punkty nie będące wierzchołkami otoczki są ze stosu zdejmowane.
* W momencie zakończenia działania algorytmu stos zawiera punkty występujące na otoczce w kolejności odwrotnej do ruchu wskazówek zegara.



Algorytm Jarvisa

Niech p0 będzie punktem o najmniejszej współrzędnej y, jeśli takich jest kilka, bierzemy ten o najmniejszej współrzędnej x.

Weź dowolny punkt różny od p0

Powtarzaj dla punktów, które jeszcze nie są w otoczce: Dla punktów pi, jeśli pi leży na prawo od wektora, weź go jako koniec wektora.

Powtarzaj, aż znajdziesz całą otoczkę.

Przecinanie się par odcinków

Algorytm korzysta z metody przez zamiatanie i polega na tym, że:

sortujemy odcinki niemalejąco względem współrzędnych x początków odcinków.

prowadzimy miotłę po tych współrzędnych.

Rozpoczynający się odcinek wkładamy do miotły binarnie, zgodnie z relacją leżenia po lewej/prawej – po lewej wyżej, po prawej niżej. Sprawdzamy czy sąsiadujące odcinki nie przecinają się.

Gdy odcinek się kończy, wyrzucamy go z miotły i sprawdzamy, czy jego dwaj sąsiedzi nie przecinają się.