Sposoby projektowania algorytmów

„Dziel i zwyciężaj”
Przykłady: MergeSort, QuickSort
Dzielimy problem na mniejsze pod-problemy + rekurencja

„Programowanie dynamiczne”
Jak „dziel i zwyciężaj” ale unikamy wywołań rekurencyjnych
z tymi samymi parametrami;
zapamiętujemy wyniki tych wywołań (tzw. spamiętywanie)
gdzie zapamiętujemy? tablica, słownik (?)
Przykłady: Fibonacciego lub obl. symbolu Newtona

„Algorytmy zachłanne”
Dla problemów optymalizacyjnych (rozwiązania + funkcja kosztu)
Budujemy/modyfikujemy rozwiązanie na podstawie
lokalnych informacji (inaczej niż w metodzie „siłowej”)
Przykłady: alg. grafowe dla MST (Prima i Kruskala)
Nie zawsze prowadzi do globalnie optymalnego rozwiązania!!

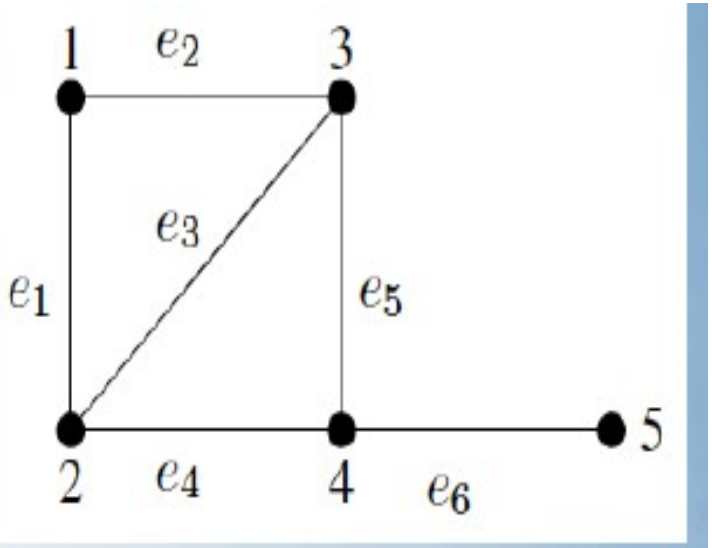
Cormen: pojęcie „optymalnej pod-struktury”...

Algorytmy grafowe

Reprezentowanie grafu w pamięci

Macierz sąsiedztwa grafu...

$$V = \{1,2,3,4,5\}$$



$$B(G) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \quad \begin{array}{l} d(1) = 2 \\ d(2) = 3 \\ d(3) = 3 \\ d(4) = 3 \\ d(5) = 1 \end{array}$$

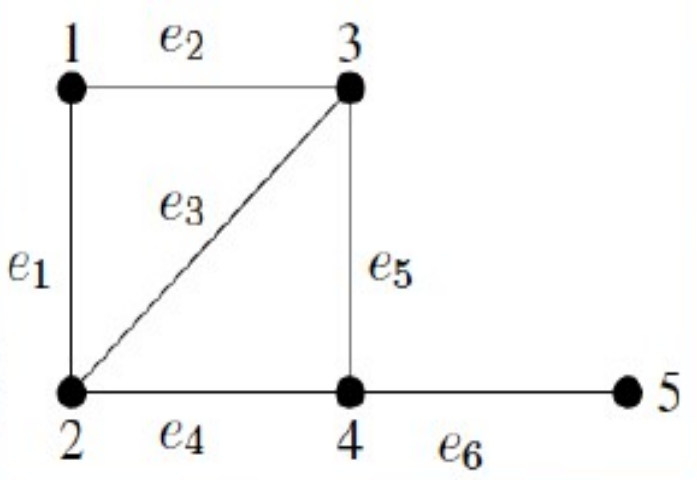
Algorytmy grafowe

Reprezentowanie grafu w pamięci

Macierz incydencji grafu...

$$V = \{1,2,3,4,5\}$$

$$E = \{e_1, e_2, e_3, e_4, e_5, e_6\} = \{\{1,2\},\{1,3\},\{2,3\},\{2,4\},\{3,4\},\{4,5\}\}$$



$$A(G) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{l} d(1) = 2 \\ d(2) = 3 \\ d(3) = 3 \\ d(4) = 3 \\ d(5) = 1 \end{array}$$

Algorytmy grafowe

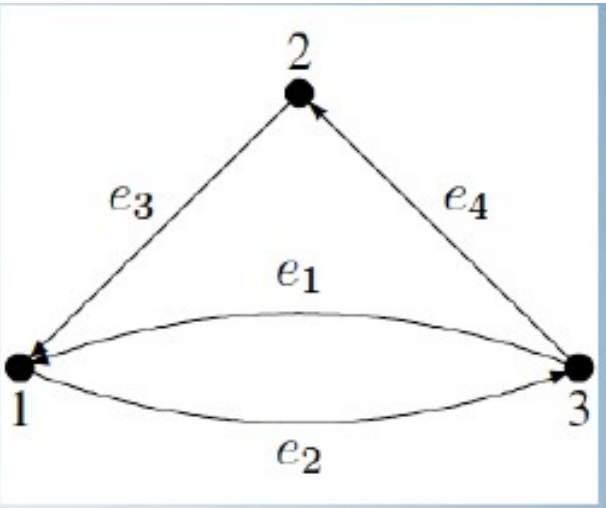
Reprezentowanie grafu w pamięci

Macierz incydencji digrafu...

$$V = \{1,2,3\}$$

$$E = \{e_1, e_2, e_3, e_4\} = \{(3,1), (1,3), (2,1), (3,2)\}$$

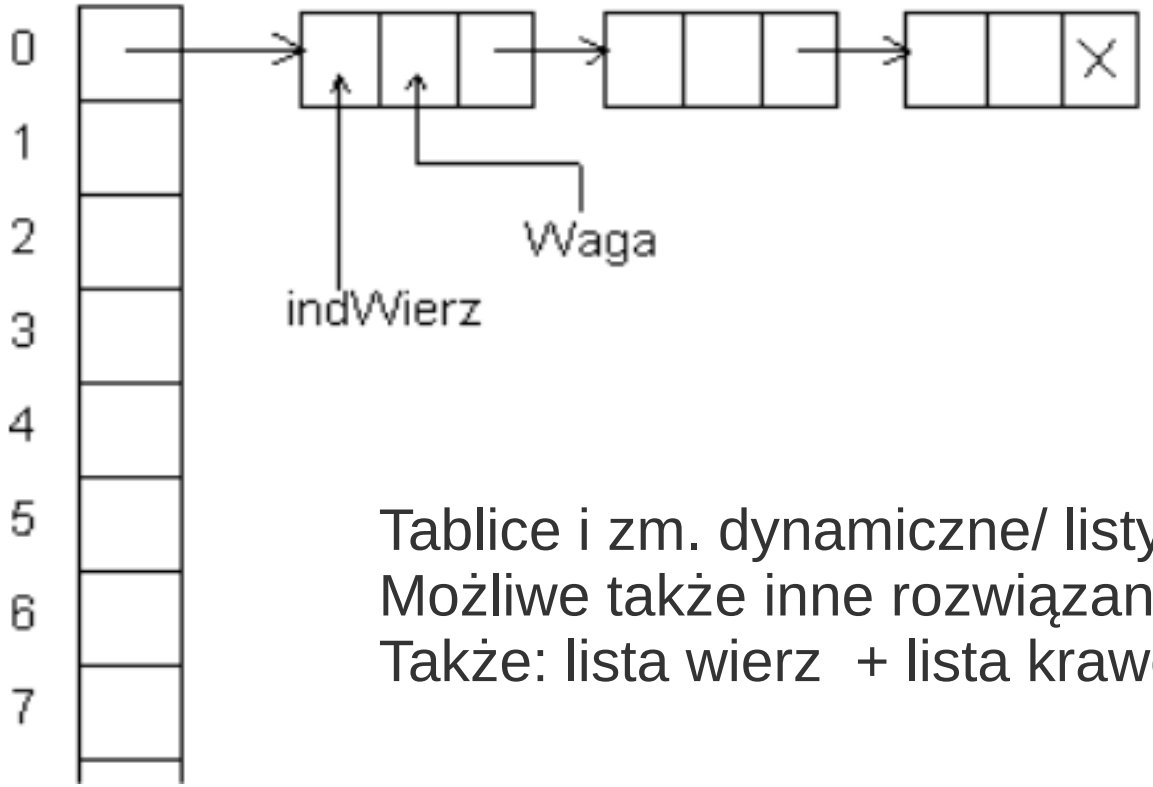
$$A(G) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{bmatrix} -1 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 1 \end{bmatrix} \end{array}$$



Algorytmy grafowe

Reprezentowanie grafu w pamięci

Listy sąsiedztw grafu (kraw z wagami)...



Tablice i zm. dynamiczne/ listy sąsiadów...
Możliwe także inne rozwiązania mieszane!
Także: lista wierz + lista krawędzi...

Algorytmy grafowe

Przeszukiwanie grafów i drzew

Przeszukiwanie czego? drzewa, grafu (niekoniecznie drzewa)
Sposoby przeszukiwania: wszerz (BFS), w głąb (DFS)

DFS (depth first search)

przeszukiwanie **w głąb**, napotykając nieodwiedzony wierzchołek rekurencyjnie odwiedzamy jego jeszcze nieodwiedzonych następników

BFS (breadth first search)

przeszukiwanie **wszerz**, napotykając nieodwiedzony wierzchołek zapisujemy w kolejce jego następników i następnie odwiedzamy kolejny wierzchołek według kolejki

Przeszukiwanie DFS drzew BST (inna kolejność kluczy):

BST_**Inorder**(var korzen)

najpierw rekurencyjne wywołanie dla lewego poddrzewa, potem wypisanie klucza bieżącego wier, potem rekurencyjne wywołanie dla prawego poddrzewa

BST_**Preorder**(var korzen)

najpierw wypisanie klucza, potem rekurencyjne wywołanie dla lewego poddrzewa, potem rekurencyjne wywołanie dla prawego poddrzewa

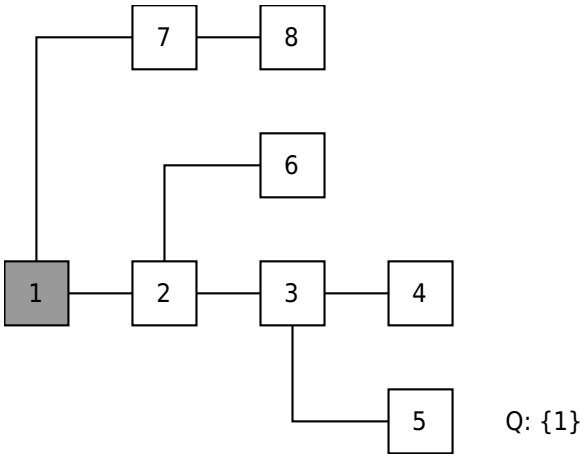
Algorytmy grafowe

Przeszukiwanie grafów i drzew

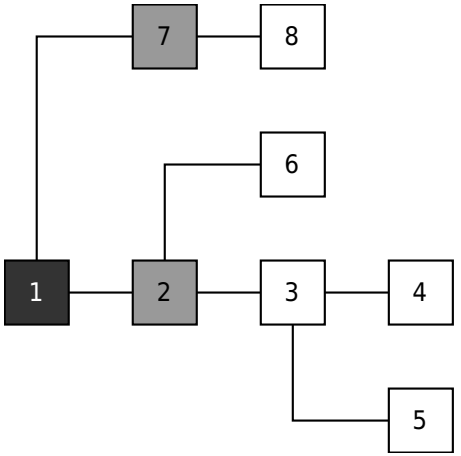
Przeszukiwanie BFS drzew?

używa się pomocniczo kolejki Q...

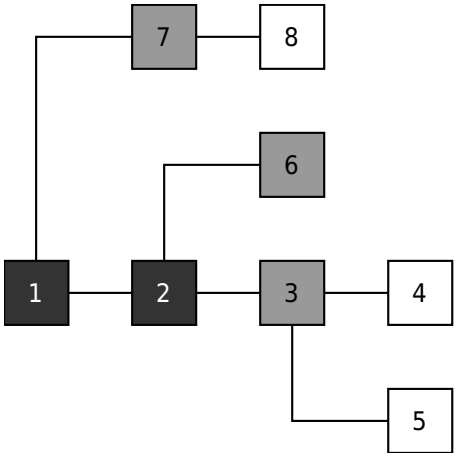
1. Q:= korzeń drzewa
2. czytamy „v” z kolejki Q i przetw go
3. zapisujemy sąsiadów v do kolejki Q (na koniec)
4. goto 2, aż do wyczerpania Q



Q: {1}



Q: {2, 7}



Q: {7, 3, 6}

Algorytmy grafowe

Przeszukiwanie grafów i drzew

Przeszukiwanie BFS grafu (niekoniecznie drzewa)?

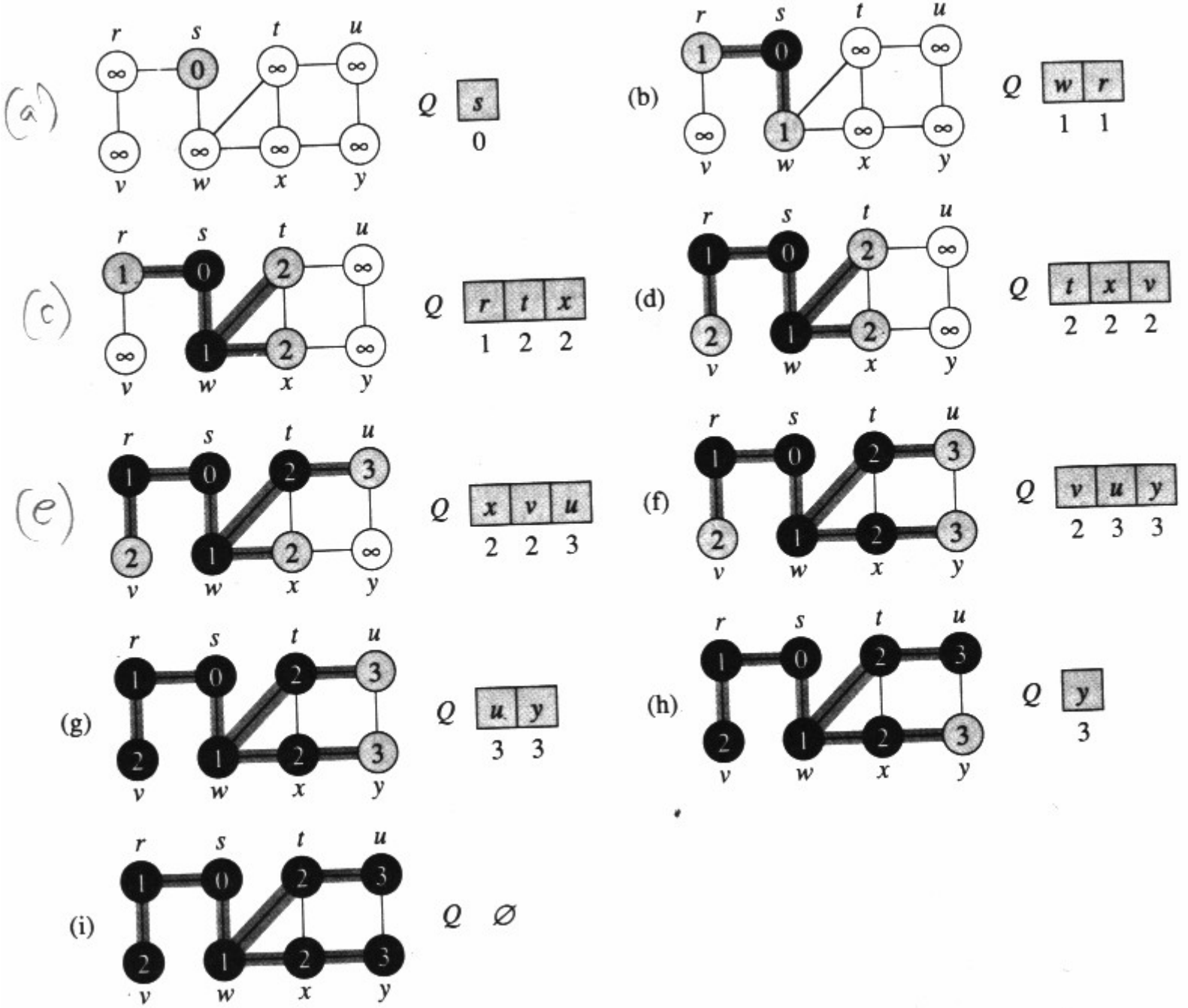
bardziej złożone niż dla drzew...

wierz mają kolory:

biały (nieprzetworzone), szary (otoczka) i czarny (przetw)

```
BFS(G, s)
 1  for każdy wierzchołek u ∈ V[G] − {s}
 2      do color[u] ← BIAŁY
 3         d[u] ← ∞
 4         π[u] ← NIL
 5  color[s] ← SZARY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← {s}
 9  while Q ≠ Ø
10      do u ← head[Q]
11         for każdy v ∈ Adj[u]
12             do if color[v] = BIAŁY
13                then color[v] ← SZARY
14                     d[v] ← d[u] + 1
15                     π[v] ← u
16                     ENQUEUE(Q, v)
17         DEQUEUE(Q)
18         color[u] ← CZARNY
```

Przeszukiwanie BFS grafu... (interpretacja kolorów)



Algorytmy grafowe

Przeszukiwanie grafów i drzew

Przeszukiwanie DFS grafu (niekoniecznie drzewa)?

nieco bardziej złożone...

można najpierw zrobić przeszukanie BFS,

potem przeszukiwać drzewo spinające metodą DFS !!

Algorytmy grafowe

Problem MST (Minimum Spanning Tree)

Krawędzie grafu mają wagi (>=0)

Problem MST: znaleźć drzewo spinające o minimalnej wadze.

def drzewa spinającego w G: drzewo zawierające całe V(G)

def wagi drzewa: suma wag wszystkich kraw drzewa (zadanie 19)

Dwa algorytmy znajdujące MST w grafie G:

Opis algorytmu Kruskala

narzędzie: op na zbiorach rozłącznych; czas = O(m*n)

powstaje wiele drzew (las), które są fragmentami MST

Opis algorytmu Prima

narzędzie: kolejka prior; czas= O(m*lg(n)) // kopce bin

powstaje 1 drzewo, które się powiększa o 1 kraw

Oba te^ algorytmy należą do kategorii „zachłannych”:

rozbudowujemy „rozwiązanie częściowe”,

każdy dodany element należy do MST;

(jeśli wagi kraw różne to jest dokładnie 1 drzewo MST)

Algorytmy grafowe

Problem MST (Minimum Spanning Tree)

Algorytm **Kruskala** – pseudokod:

```
MST-KRUSKAL(G, w)
1  A ← Ø
2  for każdy wierzchołek v ∈ V[G]
3     do MAKE-SET(v)
4  posortuj krawędzie z E niemalejąco względem wag w
5  for każda krawędź (u, v) ∈ E, w kolejności niemalejących wag
6     do if FIND-SET(u) ≠ FIND-SET(v)
7        then A ← A ∪ {(u, v)}
8             UNION(u, v)
9  return A
```

Op Union(u,v) = łączenie zbiorów rozłącznych (fragmentów MST)
działa w czasie O(n)

Zb „A” zawiera krawędzie drzewa MST

Dowód poprawności algorytmu:

zakładamy, że wagi są różne, czyli jest 1 drzewo MST

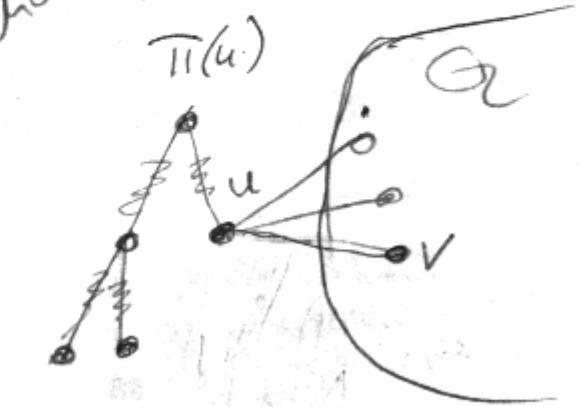
każda kraw z A należy do MST (przeczenie prowadzi do sprzeczności)

Algorytmy grafowe

Problem MST (Minimum Spanning Tree)

Algorytm **Prima** – pseudokod:

```
MST-PRIM(G, w, r)
 1  Q ← V[G]
 2  for każdy u ∈ Q
 3      do key[u] ← ∞
 4  key[r] ← 0
 5  π[r] ← NIL
 6  while Q ≠ Ø
 7      do u ← EXTRACT-MIN(Q)
 8          for każdy v ∈ Adj[u]
 9              do if v ∈ Q i w(u, v) < key[v]
10                  then π[v] ← u
11                       key[v] ← w(u, v)
```



Zał: kolejka prior jest zbudowana z kopca bin...
Zmiana klucza w linii 11 to op DecreaseKey na kopcu (czas=O(logn))
Wynik to krawędzie (v, pi(v))
Czas działania: n*logn // ExtractMin + m*logn // DecreaseKey (??) = O(m*log(n))
Dowód poprawności algorytmu:
dowodzimy, że MOE fragmentu należy do MST... (MOE=Min Outgoing Edge)