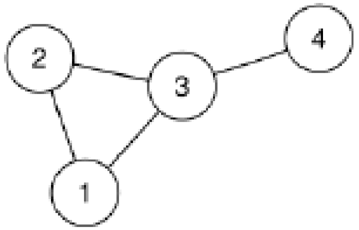
Struktury danych

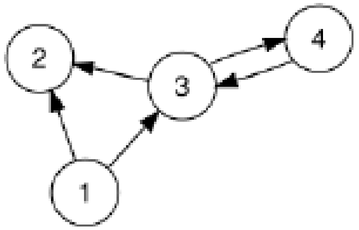
Grafy

Graf

- Graf $G = (V, E)$
  - struktura danych, zbudowana z wierzchołków (*vertices* $\in V$) oraz krawędzi (*edges* $\in E \subset V \times V$) = odnośników do innych węzłów grafu
  - krawędzie mogą być skierowane (graf skierowany) lub etykietowane wagami (graf etykietowany, z wagami)

- Implementacja
  - macierz sąsiedztwa
  - macierz incydencji
  - listy sąsiedztwa





Reprezentacja grafów

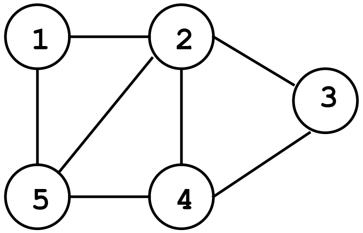
Graf (V, E)

- macierzy sąsiedztwa, dla grafów **gęstych**, tzn. $|E|$ jest bliskie $|V|^2$
- **Listy sąsiedztwa**
  - dla grafów rzadkich, tzn. $|E| << |V|^2$
  - tablica o rozmiarze $|V|$ składająca się z list, po jednej dla każdego wierzchołka
  - złożoność pamięciowa = $\Theta(|V| + |E|)$
  - wyszukiwanie krawędzi: przeglądanie list
- **Macierze sąsiedztwa**
  - dla grafów gęstych, $|E| \sim |V|^2$
  - $$m_{ij} = \begin{cases} 1 & \exists e = (v_i, v_j) \\ 0 & \text{wpp} \end{cases}$$
  - złożoność pamięciowa = $\Theta(|V|^2)$
  - wyszukiwanie krawędzi: $O(1)$
- **Macierz incydencji**
  - $$b_{ij} = \begin{cases} -1 & \text{jeżeli krawędź } j \text{ wychodzi z węzla } i \\ 1 & \text{jeżeli krawędź } j \text{ wchodzi do węzla } i \\ 0 & \text{wpp} \end{cases}$$
  - złożoność pamięciowa = $\Theta(|V| \cdot |E|)$
  - wyszukiwanie krawędzi: $O(1)$.

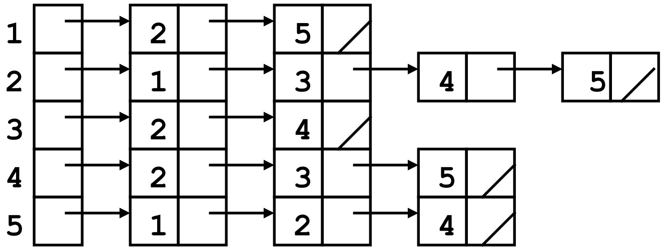
Reprezentacja grafów

- Reprezentacja grafów ważonych
  - na liście sąsiedztwa waga krawędzi (v_i , v_j) jest pamiętana z wierzchołkiem v_j
  - w macierzy sąsiedztwa $m_{ij} = w(v_i , v_j)$

Reprezentacja grafów. Przykład



Graf nieskierowany



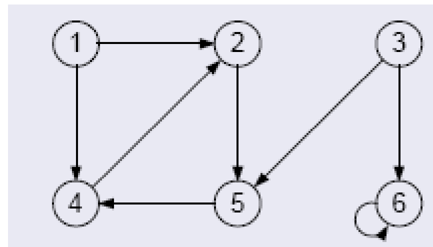
Lista sąsiedztwa
grafu G

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

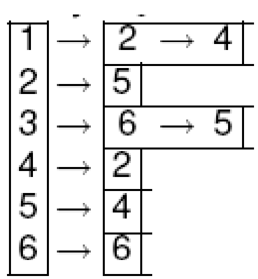
Macierz sąsiedztwa
grafu G

Macierz sąsiedztwa A grafu nieskierowanego jest symetryczna względem głównej przekątnej: $A = A^T$ (A^T – macierz transponowana)

Reprezentacja grafów. Przykład



Graf skierowany



Lista sąsiedztwa grafu G

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Macierz sąsiedztwa grafu G

Przeszukiwane wszerz grafów skierowanych i nieskierowanych

Przeszukiwanie wszerz

• Jeden z najprostszych algorytmów przeszukiwania grafu

• Dany jest graf G = (V, E) i jeden wyróżniony wierzchołek *s* (**źródło**).

W przeszukiwaniu wszerz badane są krawędzie G w celu odwiedzenia każdego wierzchołka osiągalnego z *s*. Obliczane są też odległości (najmniejsza liczba krawędzi) od *s* do wszystkich osiągalnych wierzchołków.

• Wynik algorytmu – drzewo o korzeniu *s*, zawierające wszystkie wierzchołki osiągalne z *s*.

• Dla każdego wierzchołka *v* osiągalnego z *s* ścieżka w drzewie przeszukiwania od *s* do *v* jest równa najkrótszej ścieżce od *s* do *v* w grafie G.

Założenia algorytmu

• W algorytmie wierzchołki są kolorowane na biało, szaro lub czarno. Podczas przeszukiwania każdy nowo napotkany wierzchołek staje się **odwiedzony** i zmienia kolor na szary lub czarny.

Na początku wszystkie wierzchołki są białe.

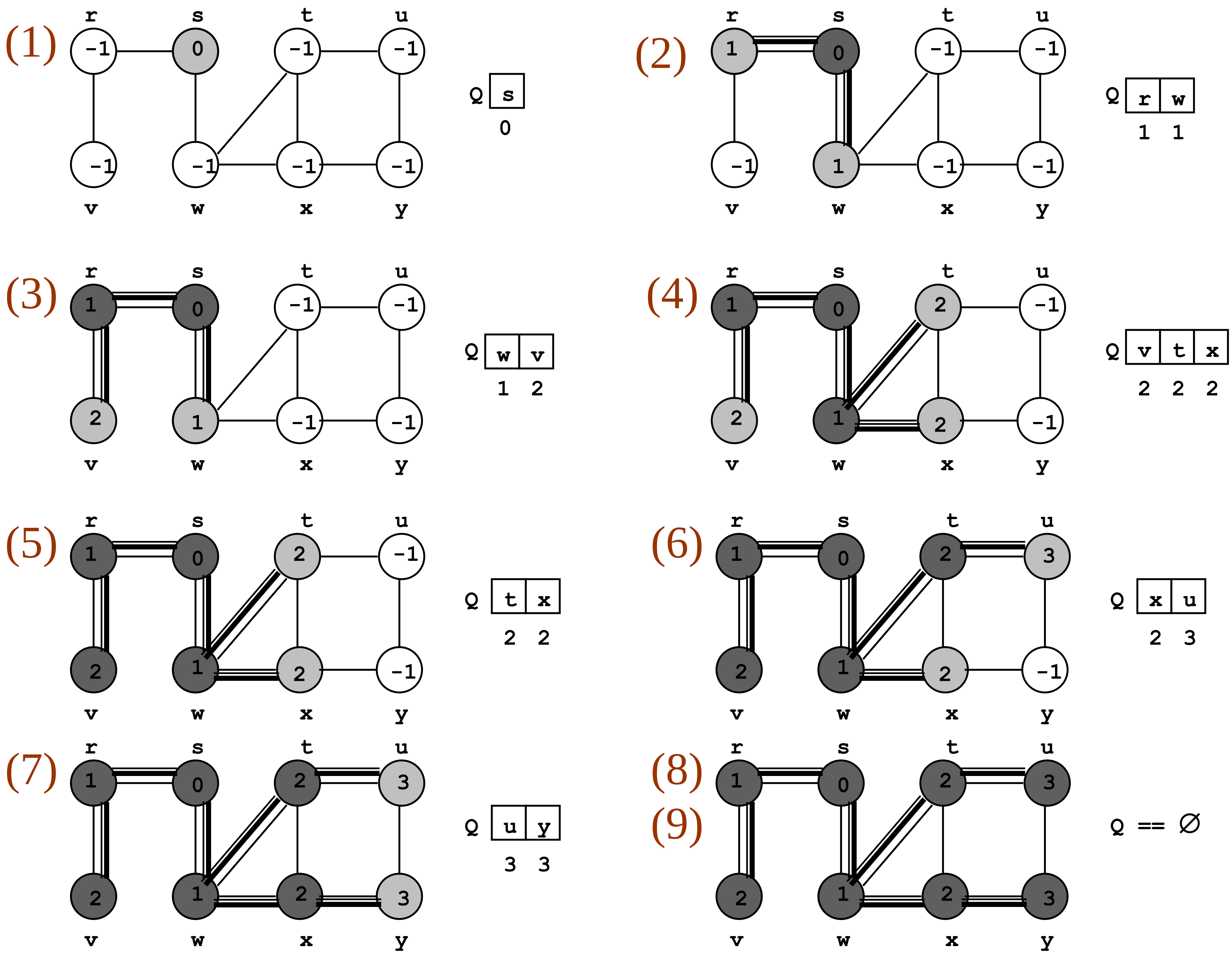
Jeśli (u, v)$\in$E i wierzchołek *u* jest czarny, to *v* jest albo szary albo czarny, tzn. wszystkie wierzchołki sąsiadujące z *v* są już odwiedzone.

Wierzchołki szare mogą mieć białych sąsiadów.

Algorytm przeszukiwania wszerz grafów

• Dany jest graf G = (V, E) reprezentowany przez listy sąsiedztwa. Dla każdego wierzchołka $u \in V$ dana jest lista sąsiedztwa Ls[*u*].
• Kolor wierzchołka $u \in V$ jest zapamiętany w zmiennej kolor[*u*].
• Poprzednik wierzchołka $u \in V$ jest zapamiętany w zmiennej poprz[*u*]. Jeśli *u* nie ma poprzednika, to poprz[*u*] = -1.
• Odległość od źródła *s* do wierzchołka *u* jest obliczana w zmiennej odl[*u*].
• Szare wierzchołki są zapamiętywane w kolejce Q typu FIFO.

```
for(każdy wierzchołek u∈V[G]-{s}){
  kolor[u] = bialy;
  odl[u] = -1;
  poprz[u] = -1;
}
kolor[s] = szary;
odl[s]= 0;
poprz[s] = -1;
Q = {s};
while(Q != ∅){
  u = weź wierzchołek z kolejki Q;
  for(każdy wierzchołek v∈Ls[u]){
    if(kolor[v]==bialy){
      kolor[v]=szary;
      odl[v] = odl[u] + 1;
      poprz[v] = u;
      dodaj do kolejki Q wierzchołek v;
    }
  }
    kolor[u] = czarny;
}
```



(1)
r s t u
-1 0 -1 -1
-1 -1 -1 -1
v w x y
Q s
0
(2)
r s t u
1 0 -1 -1
-1 1 -1 -1
v w x y
Q r w
1 1
(3)
r s t u
1 0 -1 -1
2 1 -1 -1
v w x y
Q w v
1 2
(4)
r s t u
1 0 2 -1
2 1 2 -1
v w x y
Q v t x
2 2 2
(5)
r s t u
1 0 2 -1
2 1 2 -1
v w x y
Q t x
2 2
(6)
r s t u
1 0 2 3
2 1 2 -1
v w x y
Q x u
2 3
(7)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q u y
3 3
(8)
(9)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q == ∅

Przeszukiwane wgłąb grafów

Przeszukiwanie wgłąb

Przy przeszukiwaniu wgłąb badane są krawędzie ostatnio odwiedzonego wierzchołka *v*, z którego jeszcze wychodzą nie zbadane krawędzie. Gdy wszystkie krawędzie opuszczające wierzchołek *v* są zbadane, przeszukiwanie wraca do wierzchołka, z którego *v* został odwiedzony.

Definiowany jest poprzednik wierzchołka $u \in V$, który jest zapamiętany w zmiennej poprz[*u*]. Podgraf poprzedników może składać się z kilku drzew (przeszukiwanie może być wykonywane z wielu źródeł).

Podgraf poprzedników definiowany jest jako:

$E_{poprz} = \{(\text{poprz}[v], v) : v \in V \text{ i } \text{poprz}[v] \neq \varnothing \}$

Podgraf poprzedników w przeszukiwaniu w głąb jest **lasem przeszukiwania wgłąb**, złożonym z **drzew przeszukiwania wgłąb**. Krawędzie ze zbioru E_{poprz} to **krawędzie drzewowe**.

Założenia algorytmu

- W algorytmie wierzchołki są kolorowane na biało, szaro lub czarno.
- Kolor wierzchołka zmienia się na szary, gdy jest **odwiedzany** po raz pierwszy.
- Wierzchołek jest kolorowany na czarno, gdy zostaje **przetworzony**, tzn. jeśli lista jego sąsiedztwa zostaje całkowicie zbadana.
- Każdemu wierzchołkowi *v* przypisuje się dwie etykiety:

d[*v*] – numer kroku obliczeń, w którym *v* jest odwiedzany po raz pierwszy (gdy wierzchołek kolorowany jest na szaro)

f[*v*] – numer kroku w przeszukiwaniu, w którym kończy się badanie listy sąsiedztwa wierzchołka *v* (gdy wierzchołek kolorowany jest na czarno)

Wierzchołek *v* jest biały do kroku d[*v*], szary w krokach od d[*v*] do f[*v*], a potem czarny.

```
Przesz_wglab(G){
  for(każdy wierzchołek u∈V[G]){
    kolor[u] = bialy;
    poprz[u] = -1;
  }
  czas=0;
  for(każdy wierzchołek u∈V[G]){
    if(kolor(u)==bialy)
       Odwiedz(u);
  }
}

Odwiedz(u){
  kolor[u] = szary;
  d[u] = ++czas;
  for(każdy wierzchołek v∈Ls[u]{
    if(kolor[v]==bialy){
      poprz[v]=u;
      Odwiedz(v);
    }
  }
  kolor[u] = czarny;
  f[u] = ++czas;
}
```

Sortowanie topologiczne

Sortowanie w głąb zastosowane do sortowania topologicznego acyklicznych grafów skierowanych.

Sortowanie topologiczne grafu G=(V, E) polega na uporządkowaniu wszystkich jego wierzchołków w taki sposób, że jeśli w G istnieje krawędź (*u*, *v*), to w tym porządku *u* występuje przed wierzchołkiem *v*.

Zastosowanie - określenie kolejności wykonywania czynności.

```
Sortowanie_topologiczne(G){
  lista L = ∅;
  Przesz_w_glab(G); //dla ustalenia czasów przetworzenia f[v] dla wszystkich
                    //wierzcholkow v
  wstaw każdy wierzchołek v na poczatek listy L, kiedy zostanie przetworzony;
  return L;
}
```

Minimalne drzewa rozpinające (MDR)

Drzewo rozpinające grafu

- Dany jest spójny graf nieskierowany $G=(V, E)$
- Z każdą krawędzią (u, v) związana jest waga $w(u,v)$
- Cel: znajdź acykliczny podzbiór $T \subseteq E$, który łączy wszystkie wierzchołki i którego łączna waga

  $$w(T) = \sum_{(u,v)\in T} w(u,v)$$

  jest najmniejsza
- T – acykliczny i łączy wszystkie wierzchołki $\Rightarrow$ jest drzewem i jest nazywane **drzewem rozpinającym**
- Problem wyznaczania drzewa T – **problem minimalnego drzewa rozpinającego**

Drzewo rozpinające - algorytmy

- Algorytmy
  - Algorytm Kruskala
  - Algorytm Prima
- Oparte na technice zachłannej budowy algorytmu

- Minimalne drzewo rozpinające
  - nie musi być wyznaczone jednoznacznie
  - może istnieć wiele minimalnych drzew rozpinających w grafie

Rozrastanie się MDR

- Dany jest $G=(V, E)$, funkcja wagowa w: $E \to \mathbf{R}$
- Algorytm tworzenia MDR (GenerujMDR)
  - MDR rozrasta się w wyniku dodawania pojedynczych krawędzi
  - W trakcie algorytmy tworzony jest zbiór A, który jest zawsze podzbiorem pewnego MDR
  - W każdym kroku algorytmu dodawana jest krawędź (u, v) nienaruszająca niezmiennika, że zbiór $A \cup \{(u, v)\}$ jest dalej podzbiorem MDR. Krawędź (u, v) nazywamy krawędzią bezpieczną dla zbioru A

Algorytm tworzenia MDR

Generuj-MDR(G, w)

1. $A := \emptyset$
2. Dopóki A nie tworzy DR

   Znajdź krawędź (u, v) bezpieczną dla A

   $A := A \cup \{(u, v)\}$

3. Zwróć zbiór A

Używane pojęcia

Definicje

- **Przekrój** (S, $V - S$) grafu nieskierowanego G=(V, E) to podział V na zbiory S i $V - S$
- Krawędź (u, v) $\in E$ **krzyżuje się** z przekrojem (S, $V - S$) wtw $u \in S$ i $v \in$ V – S lub odwrotnie
- Krawędź krzyżująca się z przekrojem jest **krawędzią lekką**, jeśli jej waga jest najmniejsza spośród wszystkich krawędzi krzyżujących się z tym przekrojem

Algorytm Kruskala

- Algorytm oparty na schemacie obliczania minimalnego drzewa rozpinającego
- W algorytmie do rozrastającego się lasu dodawana jest krawędź (u, v) o najmniejszej wadze spośród krawędzi łączących różne drzewa w lesie. Niech A_1 i A_2 będą drzewami, które łączy krawędź (u, v). Ponieważ (u, v) jest krawędzią lekką łączącą A_1 z innym drzewem, to (u, v) jest krawędzią bezpieczną dla A_1
- Algorytm Kruskala jest algorytmem zachłannym
  - w każdym kroku dodawana jest krawędź o najmniejszej wadze

Algorytm Kruskala

MDR-KRUSKAL(G,w)

1. $A := \varnothing$
2. Dla każdego wierzchołka $v \in V[G]$
3. *Twórz-zbiór*(v)
4. Posortuj krawędzie zbioru E niemalejąco względem wag
5. Dla każdej krawędzi $(u, v) \in E$, w kolejności niemalejących wag
6. jeśli *Znajdź-zbiór*$(u) \neq$ *Znajdź-zbiór*(v) to // Spr., czy zbiory wierzchołków // należą do tego samego drzewa
7. $A := A \cup \{(u, v)\}$
8. *Połącz*(u, v) // Łączenie drzew
9. Zwróć A

Algorytm Prima

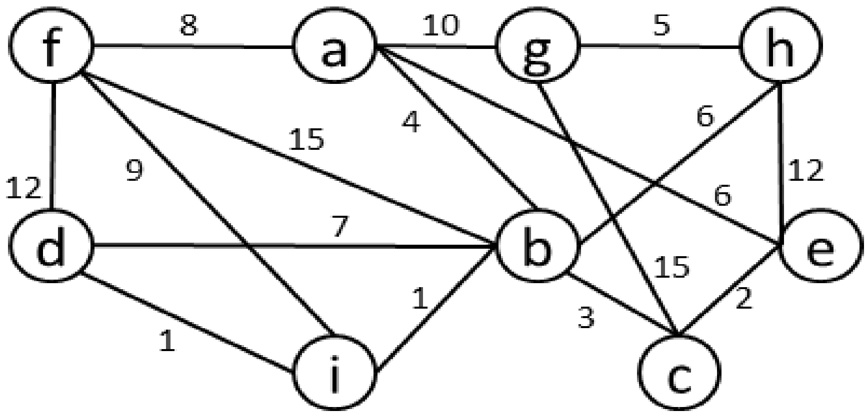
- Algorytm jest podobny do algorytmu Dijkstry dla problemu najkrótszych ścieżek w grafie
- W algorytmie krawędzie ze zbioru A tworzą zawsze pojedyncze drzewo
- Na początku drzewo tworzy dowolnie wybrany wierzchołek, a potem rośnie do chwili, w której rozpina wszystkie wierzchołki z V
- W każdym kroku krawędź lekka łącząca wierzchołek z A z wierzchołkiem z V – A jest dodawana do drzewa.
- Algorytm korzysta z:
  – Kolejki priorytetowej Q (decyduje o efektywności całego alg., np. implementacja jako kopiec binarny)
  – Dla wierzchołka *v* kluczem klucz[*v*] wyznaczającym pozycję *v* w kolejce jest minimalna waga spośród wag krawędzi łączących *v* z wierzchołkami drzewa. Zał., że klucz[*v*]=∞, jeśli takiej krawędzi nie ma.
  – Tablica ojciec[*v*] pamięta ojca *v* w budowanym drzewie

Algorytm Prima

MDR-PRIM(G,w)

1. $Q := V[G]$
2. Dla każdego wierzchołka $u \in Q$
3. klucz[u] $:= \infty$
4. klucz[r] $:= 0$
5. ojciec[r] := nil
6. Dopóki $Q \neq \emptyset$
7. $u :=$ *Wyciągnij-min*(Q)
8. dla każdego wierzchołka $v \in$ przyległy[u]
9. jeśli $v \in Q$ i $w(u,v) <$ klucz[v] to // wyznaczanie krawędzi lekkiej
10. ojciec[v] $:= u$
11. klucz[v] $:= w(u,v)$ $A = \{(v, \text{ojciec}(v)):\ v \in V - \{r\} - Q\}$

$A = \{(v, \text{ojciec}(v)):\ v \in V - \{r\}\}$ – drzewo wyznaczane niejawnie


f
a
g
h
d
b
e
i
c
8
10
5
15
4
6
9
12
12
6
7
15
1
3
2
1