Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wykład 11.

Grafy

Podstawowe pojęcia, reprezentacje, problemy

Uwagi wstępne:

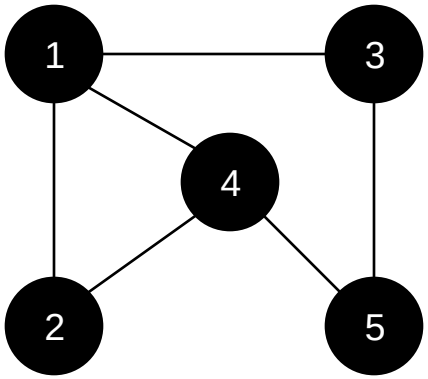
1. Na slajdach przedstawiono jedynie hasła – będą one rozwijane podczas wykładu.
2. Szczegółową wiedzę należy czerpać np. z 5. pozycji literatury podstawowej, zalecanej dla kursu (Cormen T.H. i inni, *Wprowadzenie do algorytmów,* WNT, Warszawa, 2007, wyd. 8)

Grafy – podstawowe pojęcia

Graf G (zwany też grafem nieskierowanym) to uporządkowana para:

$$G = (V, E)$$

gdzie: V - niepusty zbiór wierzchołków,
E - zbiór krawędzi, z których każda łączy określoną parę wierzchołków (inaczej: *rodzina* dwuelementowych podzbiorów zbioru V). Wierzchołki należące do krawędzi nazywane są *końcami*.



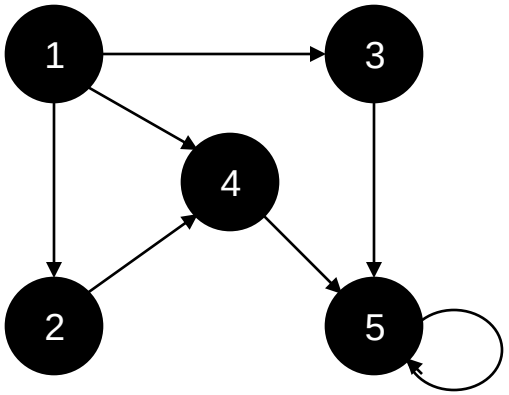
Zazwyczaj *V* i *E* są określane jako zbiory skończone. W praktyce rozważa się też czasami grafy o nieskończonej liczbie wierzchołków (wtedy liczba krawędzi może być skończona lub nieskończona).

Grafy – podstawowe pojęcia

Graf skierowany (digraf) to uporządkowana para:

$$G = (V, A)$$

gdzie: *V* - zbiór *wierzchołków*,
A - zbiór uporządkowanych par różnych wierzchołków ze zbioru *V*, zwanych *krawędziami skierowanymi* lub *łukami*.
Krawędź $e=(x,y)$ jest skierowana z *x* do *y*, czyli wychodzi z *x*, a wchodzi do *y*.



Zazwyczaj *V* i *E* są określane jako zbiory skończone. W praktyce rozważa się też czasami grafy o nieskończonej liczbie wierzchołków (wtedy liczba krawędzi może być skończona lub nieskończona).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Grafy – podstawowe pojęcia

Graf mieszany to uporządkowana trójka:

$$G = (V, E, A)$$

gdzie: zbiory *V*, *E*, *A* są zdefiniowane jak wyżej,

Graf mieszany może zawierać jednocześnie krawędzie skierowane i nieskierowane.

Wiele problemów można zamodelować za pomocą grafu.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Grafy – podstawowe pojęcia

- Droga
  Wyznaczona przez krawędzie *trasa* polegająca na podróżowaniu od wierzchołka do wierzchołka po łączących je krawędziach. Droga może być jednoznacznie zapisana jako ciąg krawędzi.
- Droga prosta
  Droga nie zawierająca dwóch tych samych krawędzi
- Długość drogi/ścieżki
  to liczba krawędzi/wierzchołków tworzących daną drogę/ścieżkę
- Ścieżka
  Intuicyjnie bardzo podobna do *drogi*, z tym, że jest wyznaczona przez *wierzchołki* (czyli można ją opisać poprzez ciąg wierzchołków).
- Ścieżka prosta
  Ścieżka wyznaczona tak, by żaden wierzchołek na *trasie* nie powtarzał się
- Ścieżka zamknięta
  Ścieżka kończąca się w początkowym wierzchołku
- Usunięcie wierzchołka
  Usunięcie wierzchołka z grafu wraz ze wszystkimi krawędziami z nim powiązanymi
- Cykl
  Zamknięta droga prosta, czyli taka, że ostatnia krawędź drogi kończy się w wierzchołku należącym do początkowej krawędzi drogi

Grafy – podstawowe pojęcia

- Droga acykliczna
  Droga nie zawierająca cyklu
- Pętla
  Krawędź zaczynająca i kończąca się w tym samym wierzchołku
- Gęstość grafu
  Stosunek liczby krawędzi do największej możliwej liczby krawędzi.
  Używa się również określeń: *graf gęsty*, jeżeli ma on *dużo* krawędzi w stosunku do liczby wierzchołków i podobnie *graf rzadki*, jeżeli ma on *mało* krawędzi w stosunku do liczby wierzchołków. Przy czym znaczenie słów mało i dużo może zależeć od kontekstu.
- Podgraf grafu H
  Graf *G* uzyskany poprzez usunięcie części wierzchołków z *H*, wraz z kończącymi się w nich krawędziami.
- Nadgraf grafu H
  Taki graf, że *H* jest jego podgrafem.
- Klika
  Podzbiór wierzchołków danego grafu, z których każdy jest sąsiadem każdego innego (czyli podgraf pełny).
- Kolorowanie grafu
  Nadanie każdemu wierzchołkowi *koloru*, tak, by żadne sąsiadujące ze sobą wierzchołki nie były *pokolorowane* tym samym kolorem.

Grafy – podstawowe pojęcia

- Liczba chromatyczna

  Najmniejsza liczba kolorów potrzebna do prawidłowego pokolorowania grafu.

- Krawędzie sąsiednie

  Krawędzie kończące się w jednym wierzchołku. W przypadku grafów skierowanych zazwyczaj wymagana jest "zgodność kierunków" krawędzi, tj. dwie krawędzie są sąsiednie, jeżeli odpowiednio kończą się i zaczynają w tym samym wierzchołku.

- Stopień wierzchołka v

  Liczba kończących się w nim krawędzi: *deg(v)*. W przypadku grafów skierowanych mówi się o stopniach wejściowym i wyjściowym – *degIn(v)*, *degOut(v)*

- Graf r-regularny

  Graf, w którym każdy wierzchołek grafu jest stopnia *r*.

- Sąsiad (sąsiedztwo)

  Dwa wierzchołki są sąsiadami, jeśli istnieje krawędź pomiędzy nimi.

- Waga krawędzi

  Często od grafu reprezentującego np. sieć połączeń komunikacyjnych oczekuje się nie tylko informacji o istniejącym połączeniu (krawędzi lub ścieżki), ale też o np. długości połączenia. Wprowadza się wtedy *wagi* - wartość przypisaną każdej krawędzi. Graf taki można wykorzystać np. do wyznaczenie optymalnej, w sensie przejechanych kilometrów trasy, lub, ogólniej rozwiązanie problemu komiwojażera, wyznaczenia optymalnego rozłożenia kabli w sieci, koordynowania wysyłania plików metodą peer to peer, itp.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Grafy – podstawowe pojęcia

- Wierzchołek izolowany

  Wierzchołek o stopniu 0, czyli nie będący końcem żadnej krawędzi.

- Wierzchołek pokrywający krawędź

  Wierzchołek *v pokrywa* krawędź *e*, jeżeli *e* kończy się w *v*. W analogiczny sposób definiuje się krawędź pokrywającą dany wierzchołek – krawędź *e* kryje wierzchołek *v*, gdy się w nim kończy.

- Acentryczność wierzchołka grafu

  Maksymalna odległości wierzchołka do innych wierzchołków grafu, lub inaczej długość najdłuższej ścieżki prostej zaczynającej się w danym wierzchołku.

- Minimalny pokrywający podzbiór krawędzi/wierzchołków

  Możliwie najmniejszy podzbiór krawędzi/wierzchołków grafu, taki, że pokrywają one wszystkie wierzchołki/krawędzie danego grafu.

  Liczność minimalnego zbioru pokrywającego krawędzi/wierzchołków nazywa się *indeksem pokrycia wierzchołkowego/krawędziowego*. Wszystkie podzbiory o tej liczności i własności nazywa się *pokryciem minimalnym*.

- Wierzchołek rozspajający

  Wierzchołek, po usunięciu którego zwiększa się liczba spójnych składowych grafu.

-Most

"Krawędziowy odpowiednik" wierzchołka rozspajającego – krawędź, po usunięciu której wzrasta liczba spójnych składowych grafu.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Grafy – podstawowe pojęcia

- Izomorfizm grafów

Graficzna reprezentacja grafów (w postaci kropek i łączących je krzywych) jest tylko sposobem przedstawienia relacji zachodzących między wierzchołkami. Dla każdego grafu istnieje nieskończenie wiele przedstawiających go jednoznacznie wykresów (*rysunków)*. Właściwości grafów są niezależne od sposobu numerowania wierzchołków, kolejności ich rysowania itp. Grafy różniące się tylko sposobem ich przedstawienia, lub indeksami nadanymi wierzchołkom, nazywamy *izomorficznymi*.

- Homeomorfizm grafów

Dwa grafy są *homeomorficzne*, jeśli z jednego grafu można otrzymać drugi zastępując wybrane krawędzie *łańcuchami prostymi* lub łańcuchy proste pojedynczymi krawędziami ("*dorysowywanie"* na krawędziach dowolnej liczby wierzchołków bądź *wymazywanie* ich).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Grafy – podstawowe pojęcia

Wybrane klasy grafów (wydzielone ze względu na różne właściwości):

- Graf prosty (właściwy)
  Graf nie zawierający pętli ani krawędzi wielokrotnych. Z reguły zdanie *G jest grafem* oznacza w domyśle, że *G* jest grafem prostym
- Graf pełny
  Graf, którego każdy wierzchołek jest połączony bezpośrednio krawędzią z każdym innym wierzchołkiem.
- Graf regularny stopnia *k*
  Graf, którego każdy wierzchołek jest stopnia *k*
- Graf acykliczny
  Graf nie zawierający żadnej *drogi zamkniętej*
- Graf spójny
  Graf, w którym dla każdego wierzchołka istnieje *droga* do każdego innego wierzchołka
- Graf k-spójny
  Graf posiadający *k* spójnych składowych
- Drzewo
  Spójny graf acykliczny
- Las
  Graf, którego wszystkie spójne składowe są drzewami

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Grafy – podstawowe pojęcia

Wybrane klasy grafów (c.d.):

- Graf dwudzielny
  Graf, którego wierzchołki mogą być podzielone na dwa zbiory, tak by w obrębie jednego zbioru żaden wierzchołek nie był połączony z innym
- Graf dwudzielny pełny
  Graf dwudzielny taki, że każdy wierzchołek z jednego zbioru jest połączony krawędzią z każdym wierzchołkiem ze zbioru drugiego.
- Graf k-dzielny
  To naturalne rozszerzenie klasy grafów dwudzielnych – jest to graf, którego zbiór wierzchołków można podzielić na *k* parami rozłącznych podzbiorów takich, że żadne dwa węzły należące do tego samego zbioru nie są połączone krawędzią
- Pełny graf k-dzielny
  Jeżeli zbiór wierzchołków dzieli się na *k* nie połączonych między sobą podzbiorów wierzchołków, to jeżeli dla każdego wierzchołka z *j-tego* przedziału jest połączony z każdym wierzchołkiem z każdego z przedziałów poza *j*, to jest to pełny graf k-dzielny
- Graf eulerowski
  Graf posiadający *drogę prostą* przechodzą przez każdą *krawędź*.
- Graf hamiltonowski
  Graf posiadający *ścieżkę prostą* przechodzą przez każdy *wierzchołek*.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Grafy – podstawowe pojęcia

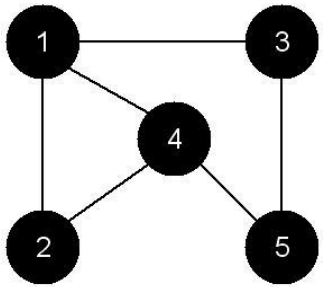
Wybrane klasy grafów (c.d.):

- Graf planarny
  Graf, dla którego istnieje *graf izomorficzny*, który można przedstawić na płaszczyźnie tak, by żadne krawędzie się nie przecinały.
- Graf płaski
  To izomorficzne przedstawienie grafu takie, że żadne dwie krawędzie się nie przecinają.
- Graf platoński
  Graf, którego przedstawienie tworzy siatkę wielościanu foremnego.
- Graf komórkowy
  Graf płaski, którego wszystkie ściany są utworzone przez drogi zamknięte tej samej długości.
- Graf symetryczny
  Graf skierowany taki, że jeżeli istnieje krawędź (u,v) to istnieje też krawędź (v,u).
- Graf asymetryczny
  Graf asymetryczny to taki, że jeżeli istnieje krawędź (u,v) to nie istnieje krawędź (v,u).
- Graf podstawowy grafu skierowanego
  To (niemal) ten sam graf, ale nieskierowany, czyli bez zwrotów na krawędziach

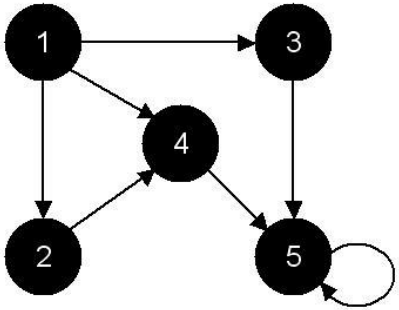
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Grafy – podstawowe reprezentacje

- Macierz sąsiedztwa



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

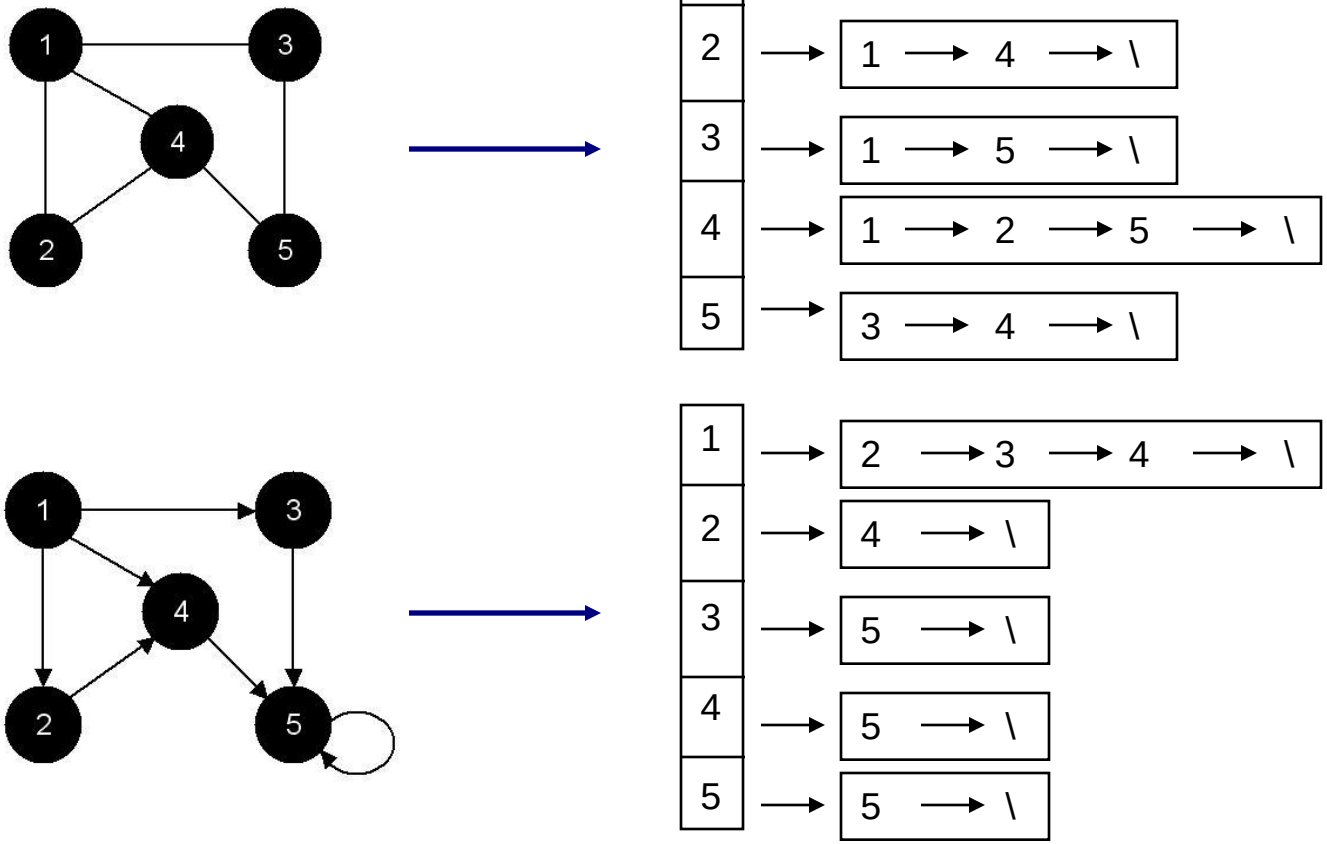


	1	2	3	4	5
1	0	1	1	1	0
2	-1	0	0	1	0
3	-1	0	0	0	1
4	-1	-1	0	0	1
5	0	0	-1	-1	1

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Grafy – podstawowe reprezentacje

- Lista sąsiedztwa



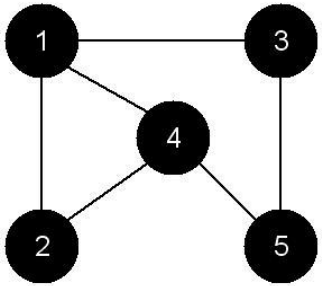
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Grafy – podstawowe reprezentacje

- Macierz incydencji

Krawędzie:

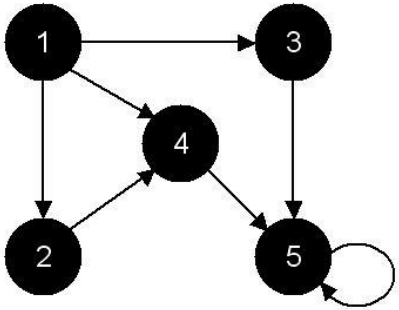
e1 = {1,2}
e2 = {1,3}
e3 = {1,4}
e4 = {2,4}
e5 = {3,5}
e6 = {4,5}



Krawędzie

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

e1 = {1,2}
e2 = {1,3}
e3 = {1,4}
e4 = {2,4}
e5 = {3,5}
e6 = {4,5}



Krawędzie

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

W każdej z przedstawionych reprezentacji można przedstawić **wagę krawędzi** określoną przez **funkcję wagową E -> R**

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przykładowe problemy i algorytmy grafowe

1. Przeszukiwanie grafu

a) Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)
b) Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

2. Wyznaczanie najkrótszej drogi w grafie

Długość drogi jest rozumiana jako suma wag krawędzi w drodze.

3. Wyznaczanie najkrótszej drogi pomiędzy dwoma wierzchołkami

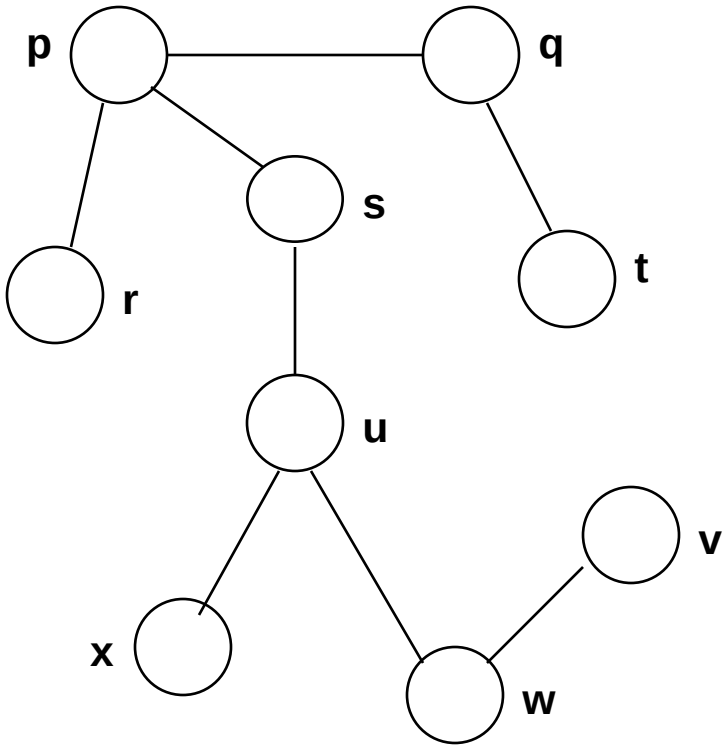
4. Wyznaczanie minimalnego drzewa rozpinającego **MST** (Minimal Spanning Tree).

-> Algorytm Kruskala.

-> Algorytm Prima

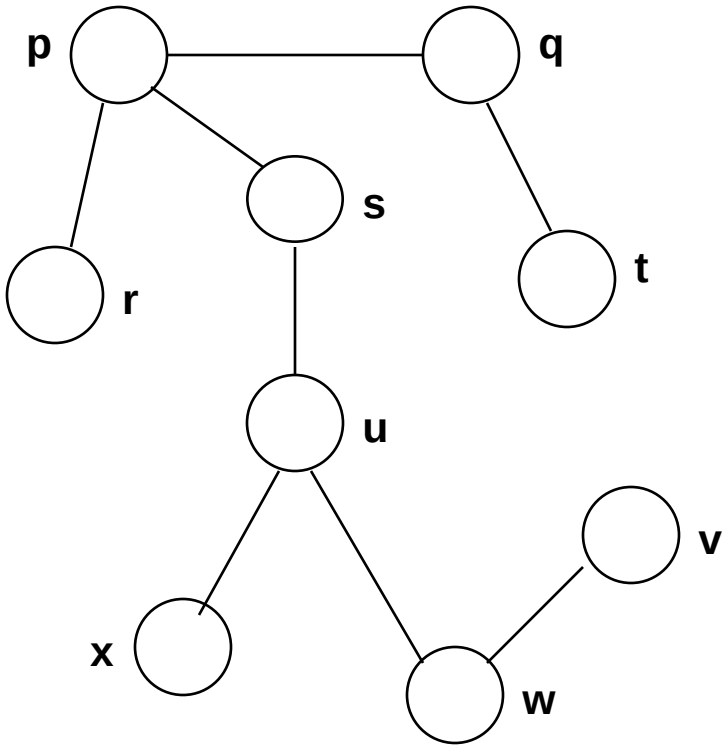
Wynikiem jest zbiór MST (Minimal Spanning Tree) zawierający krawędzie tworzące drzewo rozpinające o minimalnej sumie wag krawędzi.

Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)




```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
                oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

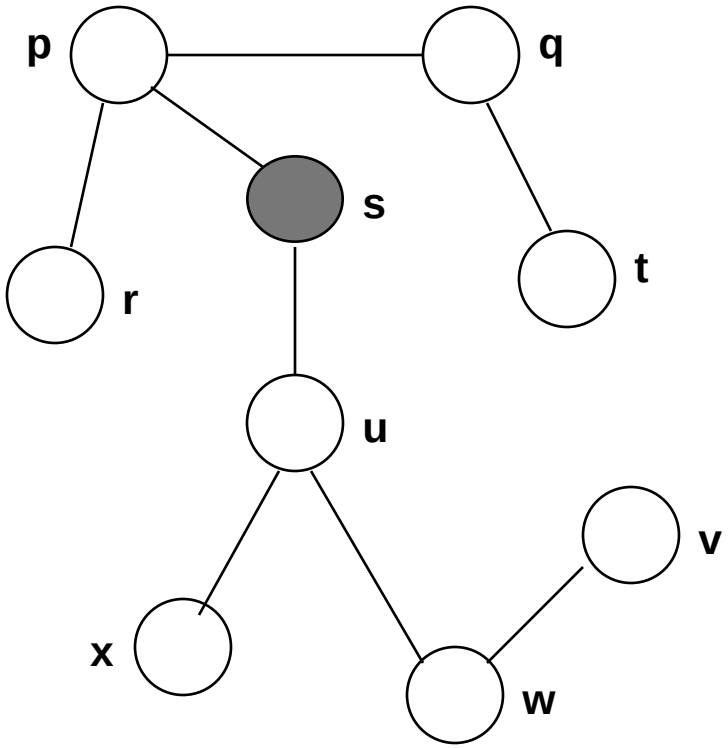
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)




```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

Kolejka: { }

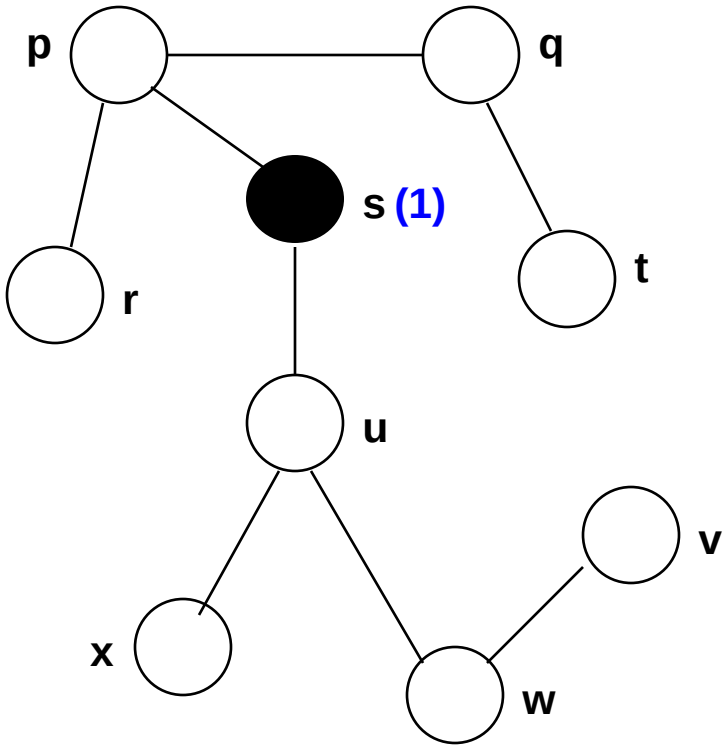
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)




```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

Kolejka: { s }

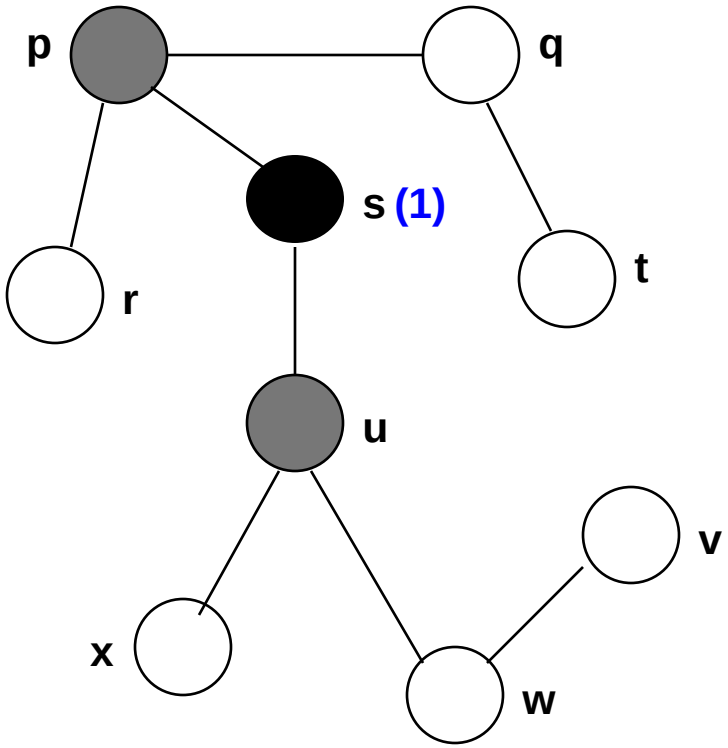
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)




```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

Kolejka: { }

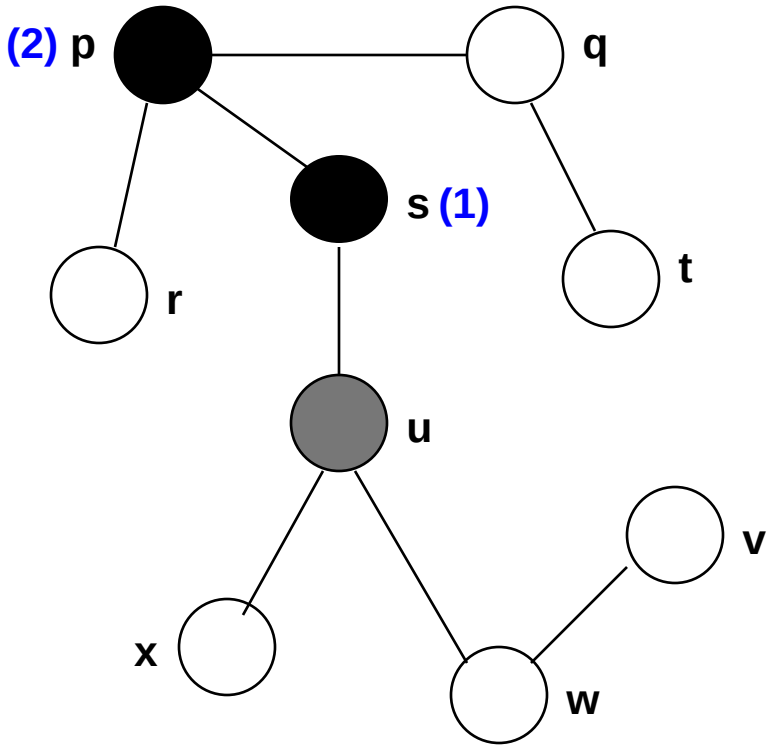
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { p, u }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

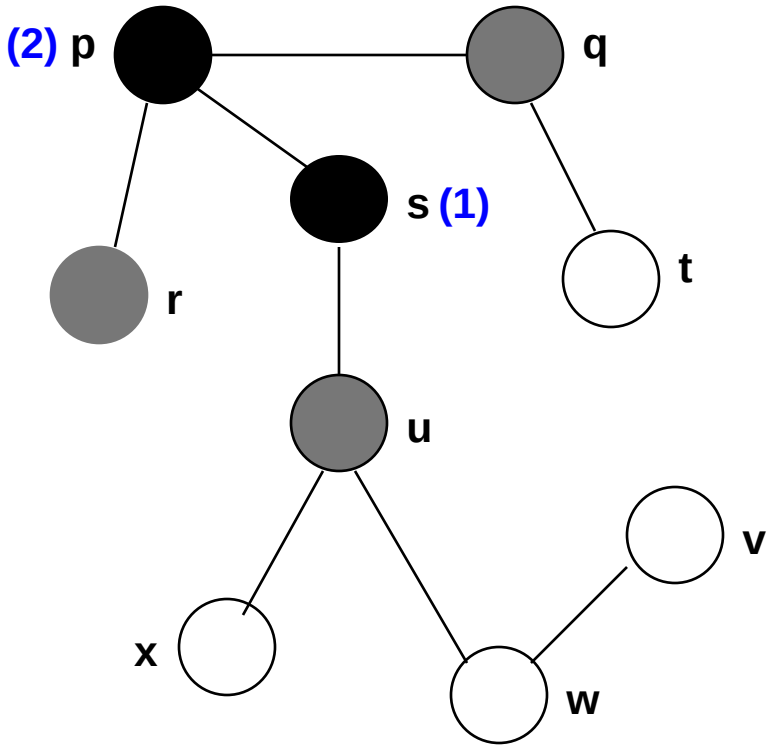
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { u }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

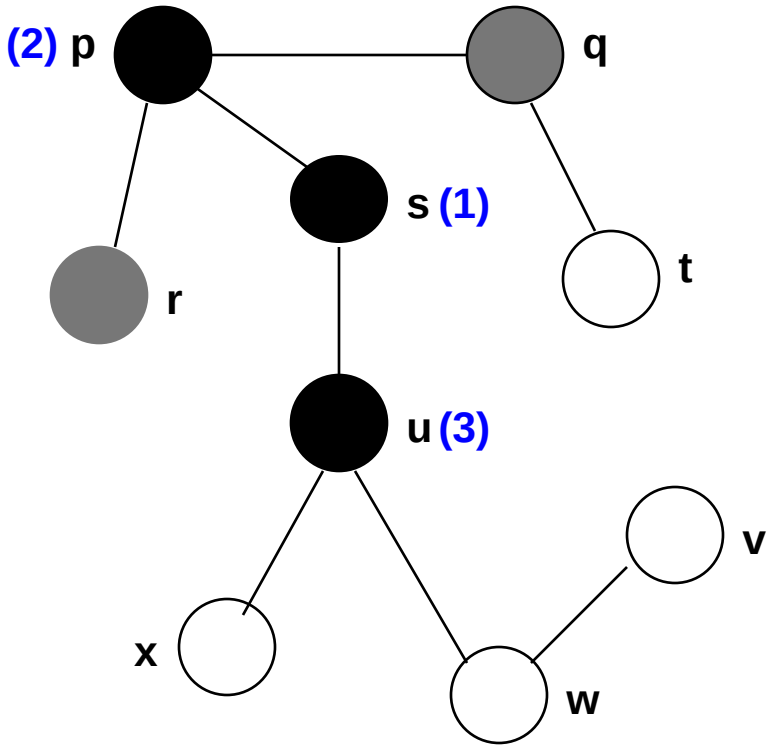
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { u, r, q }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

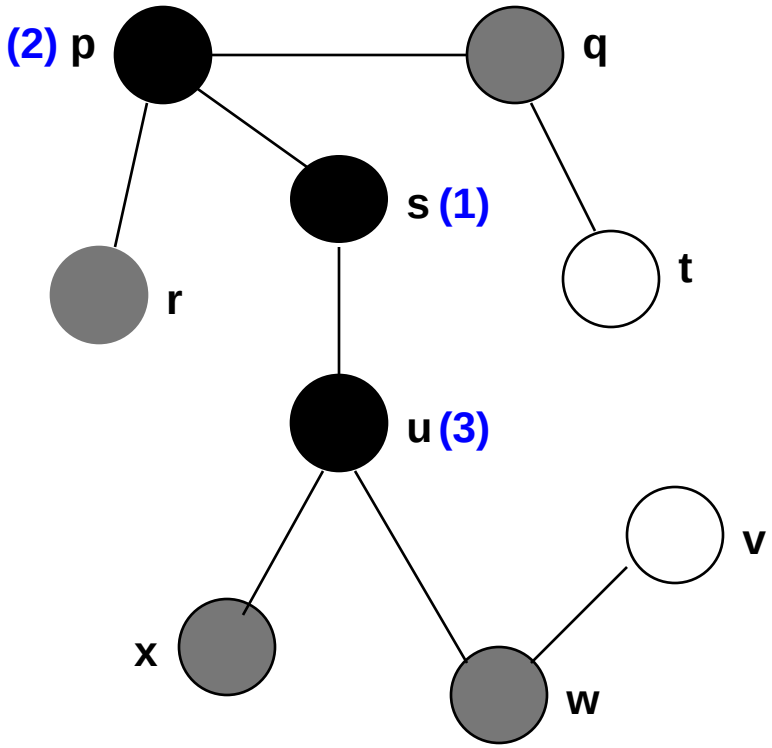
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { r, q }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

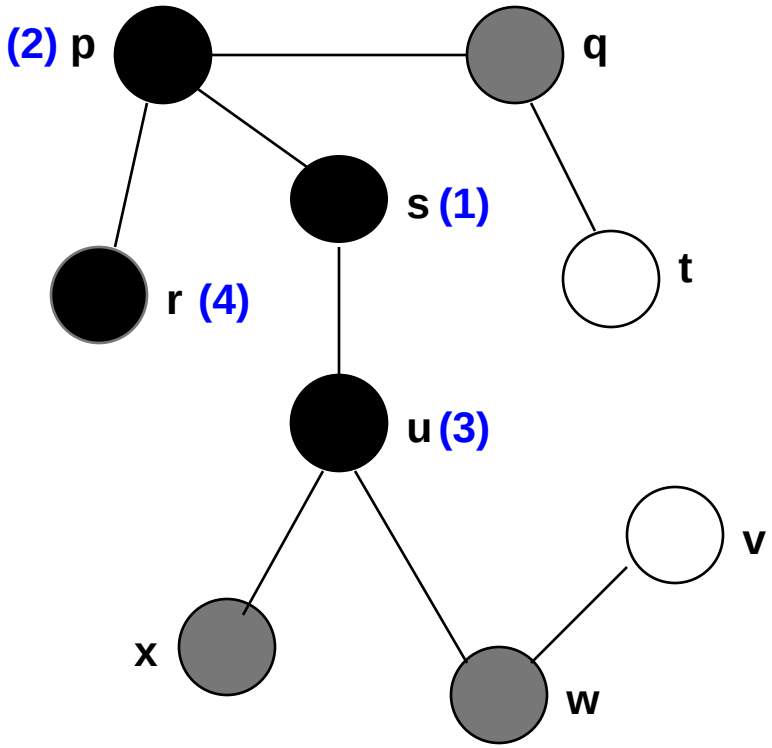
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { r, q, x, w }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
                oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

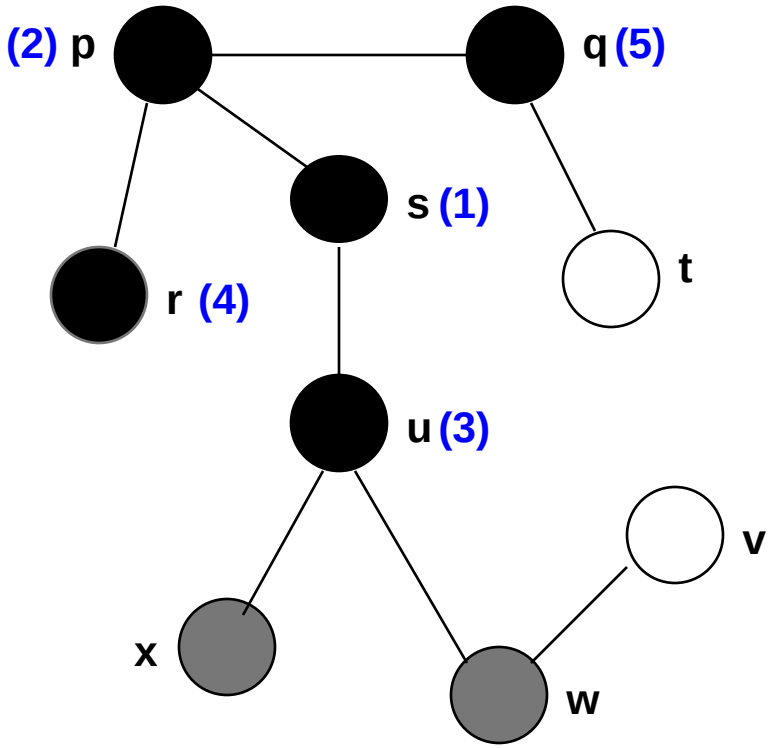
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { q, x, w }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

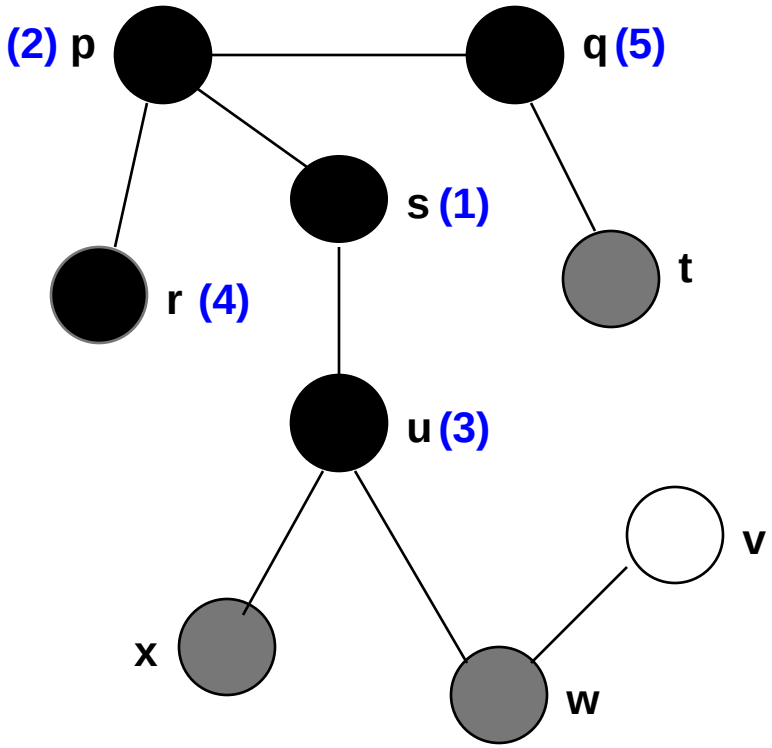
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { x, w }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

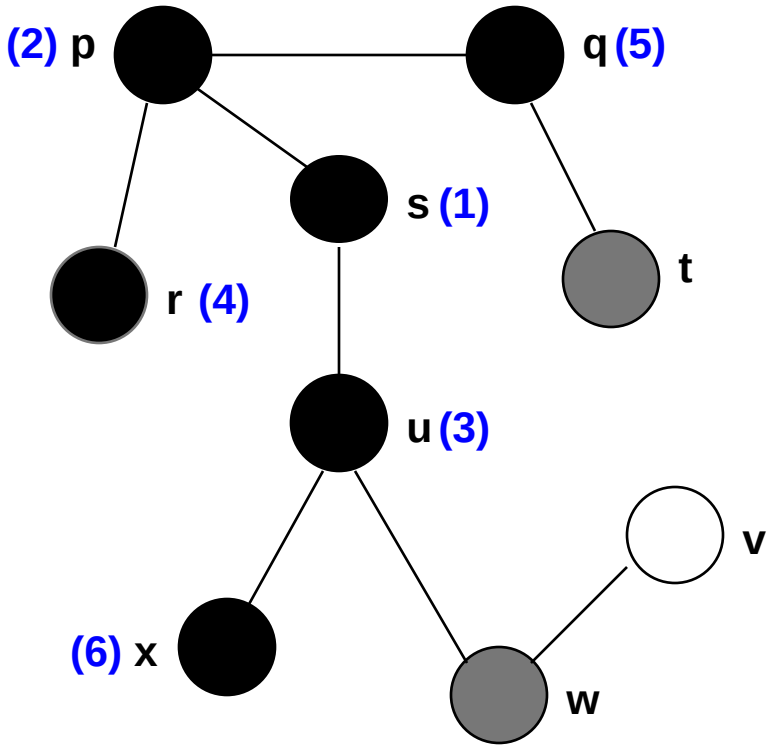
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { x, w, t }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

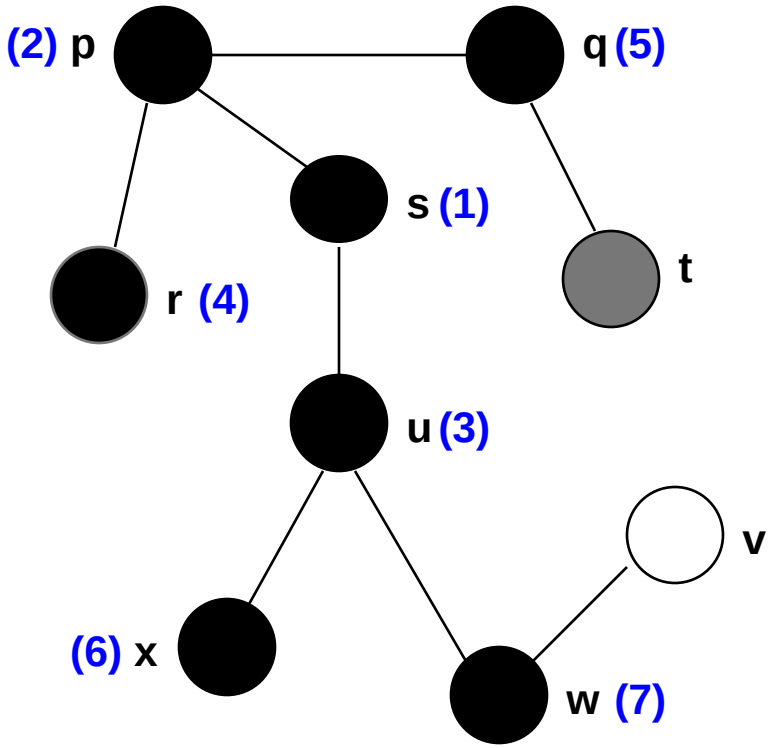
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { w, t }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

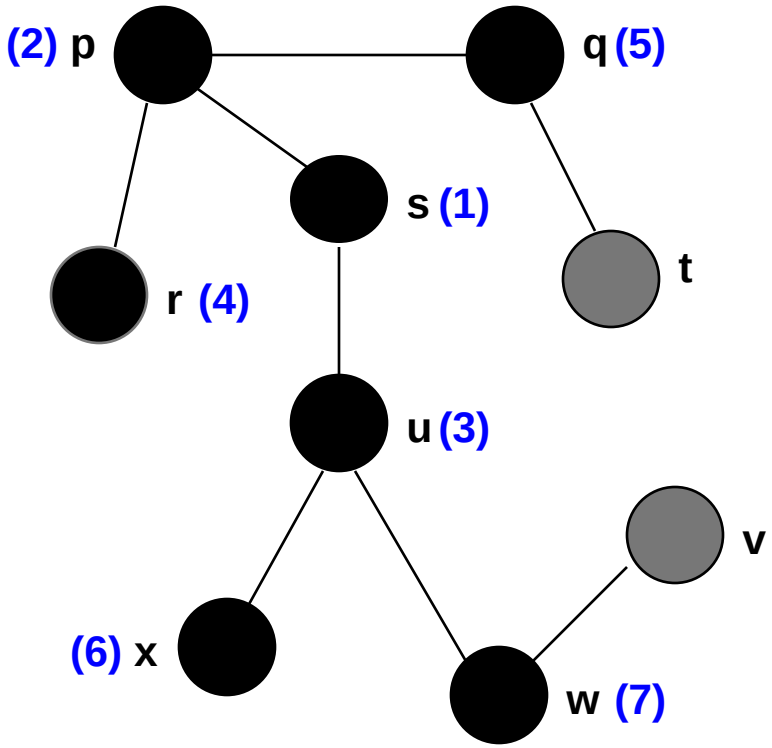
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { t }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

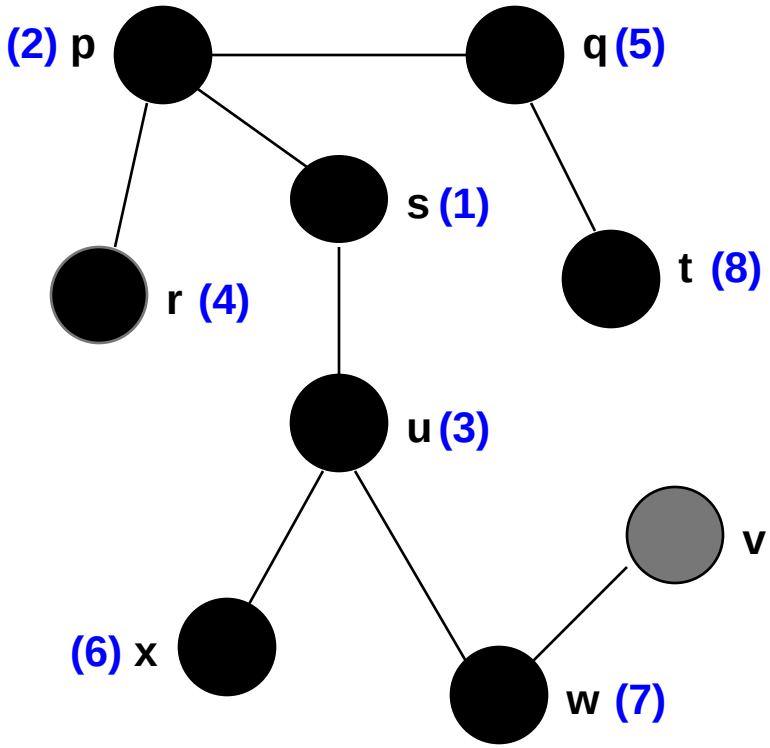
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { t, v }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

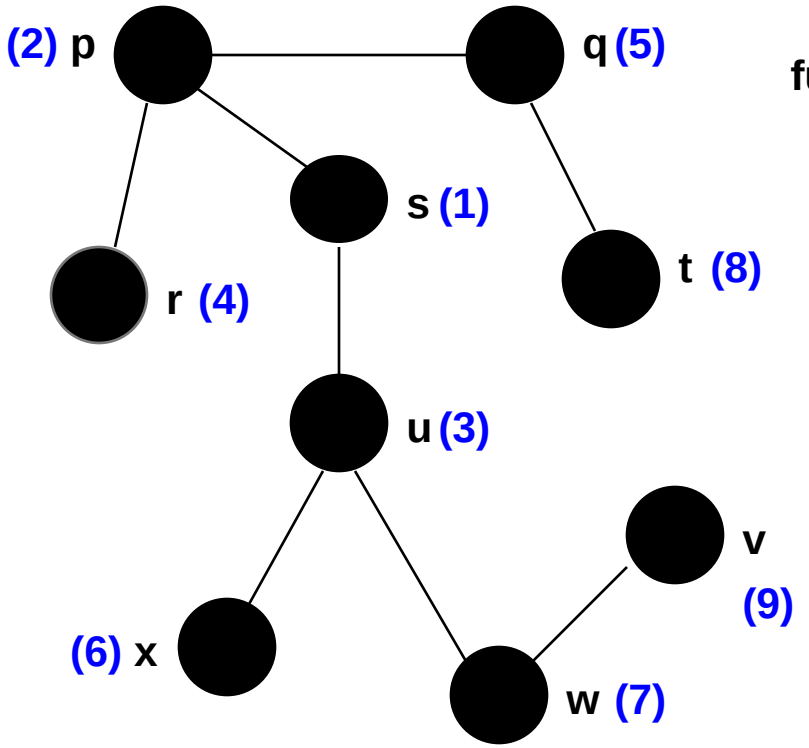
Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)



Kolejka: { v }

```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)




```
funkcja BFS (s, Cel)
    wyczyść(Kolejka)
    dla każdego wierzchołka x innego niż s
            oznacz x jako nie odwiedzony
    wstaw_do_kolejki(Kolejka, s)
    dopóki nie_pusta(Kolejka)
        w = pobierz_z_kolejki(Kolejka)
        jeżeli w = Cel
                zwróć w /* i zakończ funkcję */
        dla każdego wierzchołka u ∈ LS[w]
            jeżeli nie_odwiedzony(u)
                zapamiętaj_że_odwiedzony(u)
                wstaw_do_kolejki(Kolejka, u)
```

Kolejka: { } -> KONIEC

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)

Złożoność pamięciowa BFS: O(|V| + |E|),
gdzie |V| to liczba węzłów, a |E| to liczba krawędzi w grafie,
ponieważ trzeba utrzymywać listę węzłów które się już odwiedziło.
Z powodu tak dużego zapotrzebowania na pamięć BFS
jest niepraktyczne dla dużych danych.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)

Złożoność pamięciowa BFS: O(|V| + |E|),
gdzie |V| to liczba węzłów, a |E| to liczba krawędzi w grafie,
ponieważ trzeba utrzymywać listę węzłów które się już odwiedziło.
Z powodu tak dużego zapotrzebowania na pamięć BFS
jest niepraktyczne dla dużych danych.

Złożoność czasowa BFS: O(|V| + |E|),
gdzie |V| to liczba węzłów, a |E| to liczba krawędzi w grafie,
ponieważ trzeba przebyć wszystkie krawędzie i odwiedzić
wszystkie wierzchołki.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)

Złożoność pamięciowa BFS: O(|V| + |E|),
gdzie |V| to liczba węzłów, a |E| to liczba krawędzi w grafie,
ponieważ trzeba utrzymywać listę węzłów które się już odwiedziło.
Z powodu tak dużego zapotrzebowania na pamięć BFS
jest niepraktyczne dla dużych danych.

Złożoność czasowa BFS: O(|V| + |E|),
gdzie |V| to liczba węzłów, a |E| to liczba krawędzi w grafie,
ponieważ trzeba przebyć wszystkie krawędzie i odwiedzić
wszystkie wierzchołki.

Kompletność: BFS jest kompletne, bo jeśli istnieje rozwiązanie,
to niezawodnie zostanie znalezione (niezależnie od grafu).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przeszukiwanie wszerz (algorytm **BFS** - **B**readth **F**irst **S**earch)

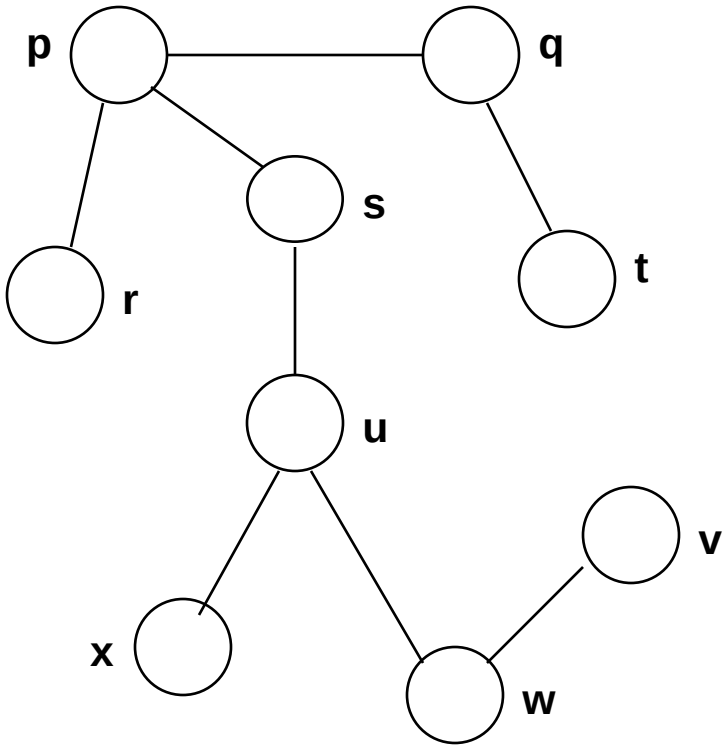
Złożoność pamięciowa BFS: O(|V| + |E|),
gdzie |V| to liczba węzłów, a |E| to liczba krawędzi w grafie,
ponieważ trzeba utrzymywać listę węzłów które się już odwiedziło.
Z powodu tak dużego zapotrzebowania na pamięć BFS
jest niepraktyczne dla dużych danych.

Złożoność czasowa BFS: O(|V| + |E|),
gdzie |V| to liczba węzłów, a |E| to liczba krawędzi w grafie,
ponieważ trzeba przebyć wszystkie krawędzie i odwiedzić
wszystkie wierzchołki.

Kompletność: BFS jest kompletne, bo jeśli istnieje rozwiązanie,
to niezawodnie zostanie znalezione (niezależnie od grafu).

Przykładowe zastosowania:
- znajdowanie wszystkich połączonych węzłów w grafie,
- znajdowanie najkrótszej ścieżki pomiędzy dwoma węzłami,
- sprawdzanie, czy graf jest dwudzielny.

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

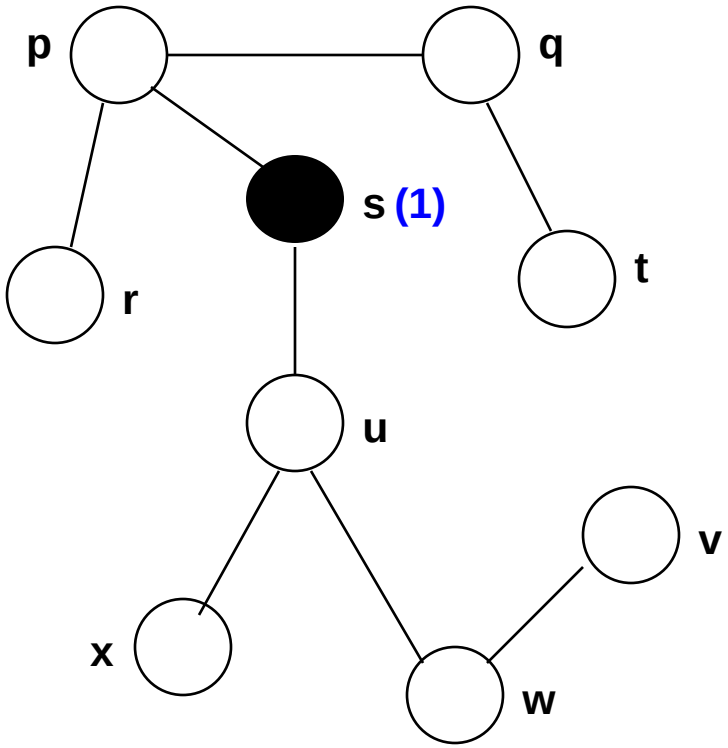


funkcja DFS (s, Cel)
  dla każdego wierzchołka x innego niż s
    oznacz x jako nie odwiedzony
  DFS_VERT(s)

Gdzie:

DFS_VERT(u)
  /* wykorzystaj u, np.:
    jeśli u=Cel to .... */
  zapamiętaj_że_odwiedzony(u)
  dla każdego wierzchołka v∈LS[u]
    jeżeli nie_odwiedzony(u)
      DFS_VERT(v)

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

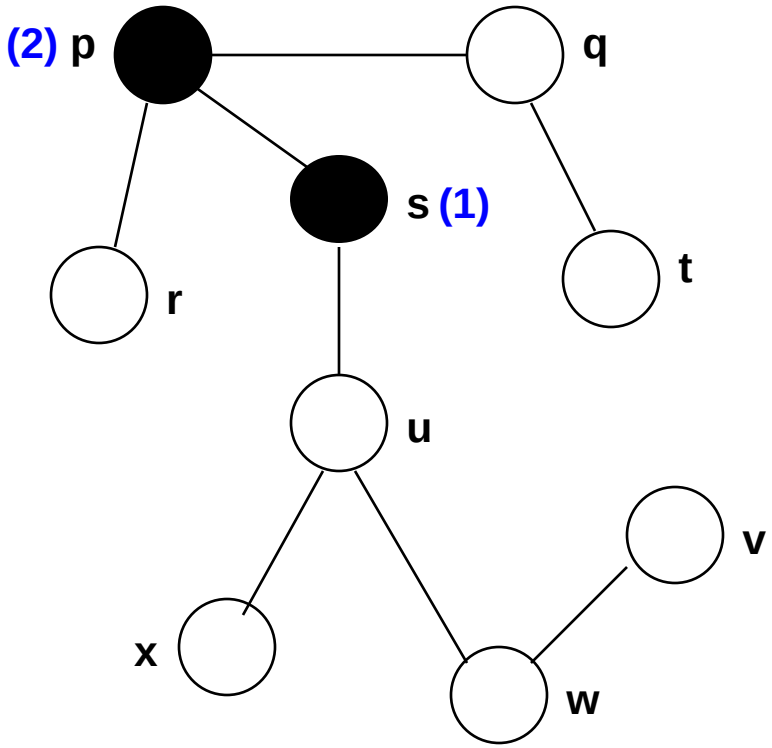


funkcja DFS (s, Cel)
  dla każdego wierzchołka x innego niż s
    oznacz x jako nie odwiedzony
  DFS_VERT(s)

Gdzie:

DFS_VERT(u)
  /* wykorzystaj u, np.:
    jeśli u=Cel to .... */
  zapamiętaj_że_odwiedzony(u)
  dla każdego wierzchołka v∈LS[u]
    jeżeli nie_odwiedzony(u)
      DFS_VERT(v)

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

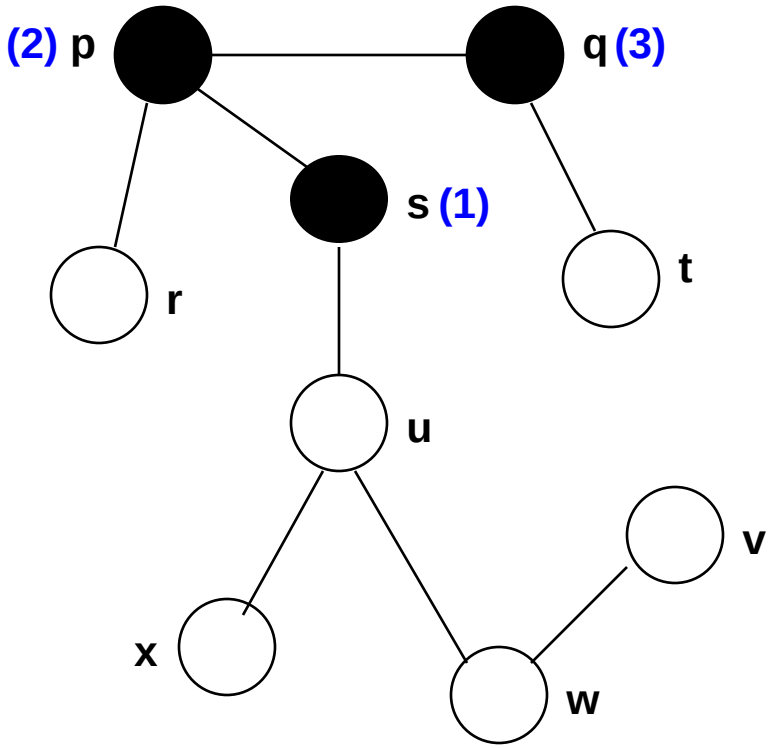


funkcja DFS (s, Cel)
  dla każdego wierzchołka x innego niż s
    oznacz x jako nie odwiedzony
  DFS_VERT(s)

Gdzie:

DFS_VERT(u)
  /* wykorzystaj u, np.:
    jeśli u=Cel to .... */
  zapamiętaj_że_odwiedzony(u)
  dla każdego wierzchołka v∈LS[u]
    jeżeli nie_odwiedzony(u)
      DFS_VERT(v)

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

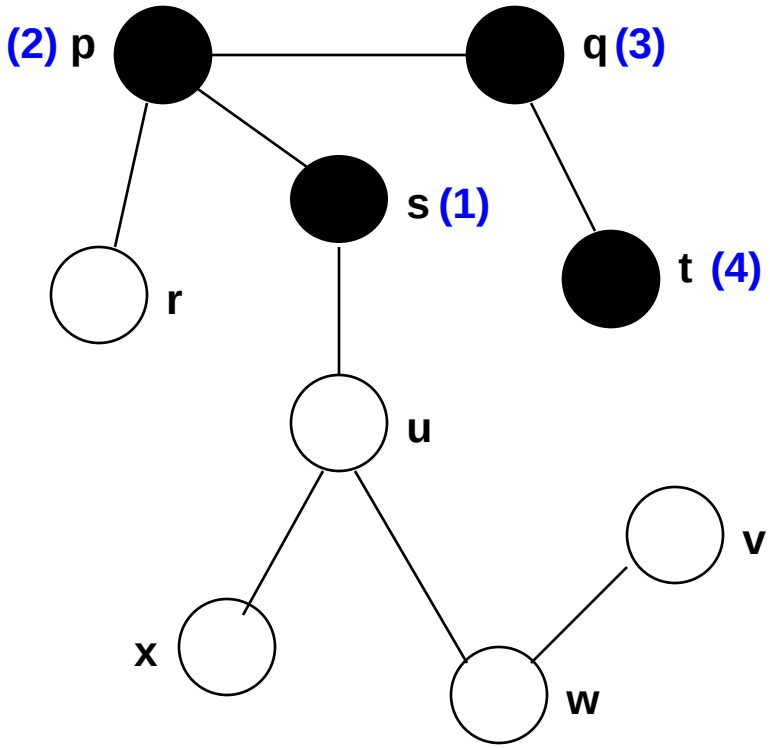


funkcja DFS (s, Cel)
  dla każdego wierzchołka x innego niż s
    oznacz x jako nie odwiedzony
  DFS_VERT(s)

Gdzie:

DFS_VERT(u)
  /* wykorzystaj u, np.:
    jeśli u=Cel to .... */
  zapamiętaj_że_odwiedzony(u)
  dla każdego wierzchołka v∈LS[u]
    jeżeli nie_odwiedzony(u)
      DFS_VERT(v)

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

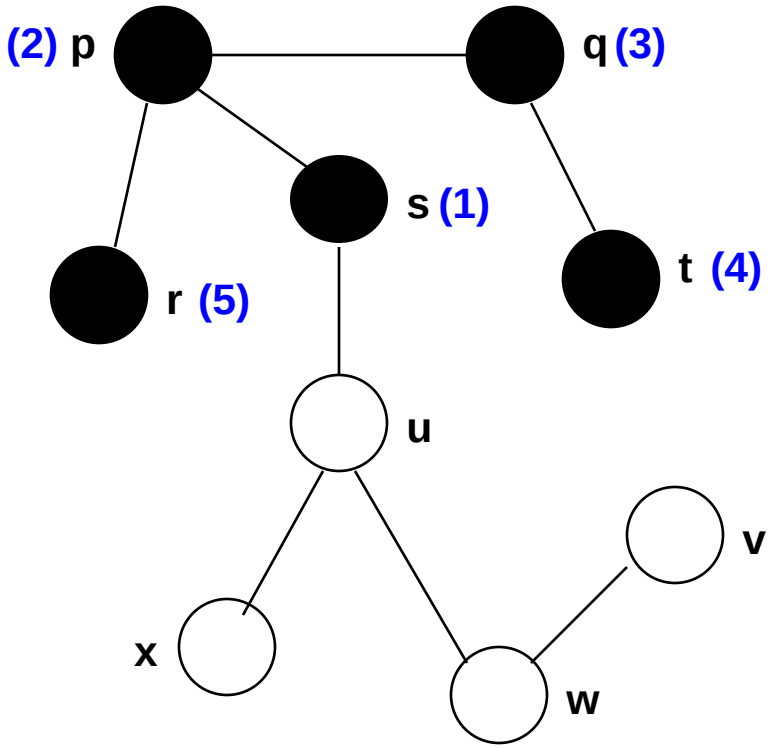


funkcja DFS (s, Cel)
  dla każdego wierzchołka x innego niż s
    oznacz x jako nie odwiedzony
  DFS_VERT(s)

Gdzie:

DFS_VERT(u)
  /* wykorzystaj u, np.:
    jeśli u=Cel to .... */
  zapamiętaj_że_odwiedzony(u)
  dla każdego wierzchołka v∈LS[u]
    jeżeli nie_odwiedzony(u)
      DFS_VERT(v)

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

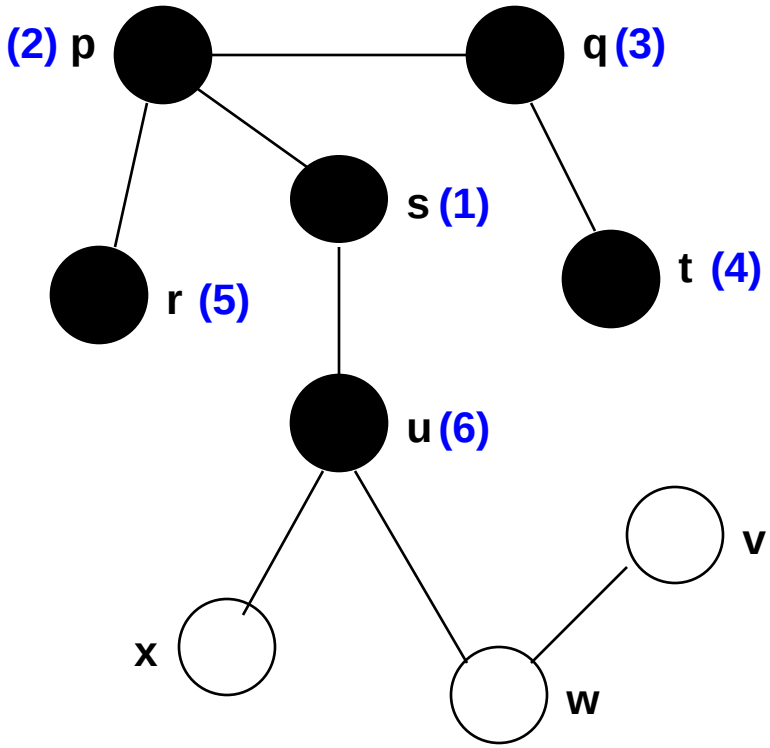


funkcja DFS (s, Cel)
  dla każdego wierzchołka x innego niż s
    oznacz x jako nie odwiedzony
  DFS_VERT(s)

Gdzie:

DFS_VERT(u)
  /* wykorzystaj u, np.:
    jeśli u=Cel to .... */
  zapamiętaj_że_odwiedzony(u)
  dla każdego wierzchołka v∈LS[u]
    jeżeli nie_odwiedzony(u)
      DFS_VERT(v)

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

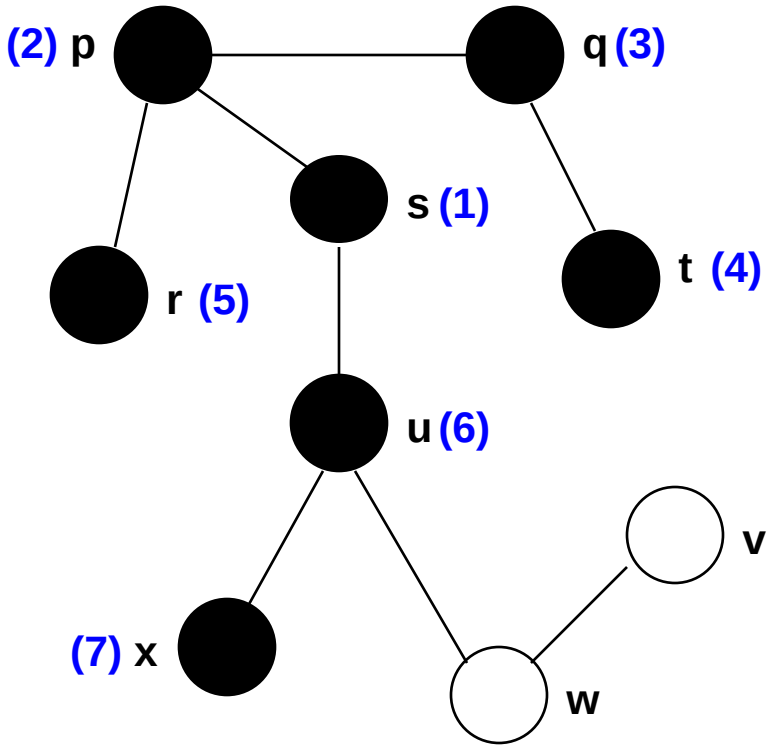


funkcja DFS (s, Cel)
  dla każdego wierzchołka x innego niż s
    oznacz x jako nie odwiedzony
  DFS_VERT(s)

Gdzie:

DFS_VERT(u)
  /* wykorzystaj u, np.:
    jeśli u=Cel to .... */
  zapamiętaj_że_odwiedzony(u)
  dla każdego wierzchołka v∈LS[u]
    jeżeli nie_odwiedzony(u)
      DFS_VERT(v)

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

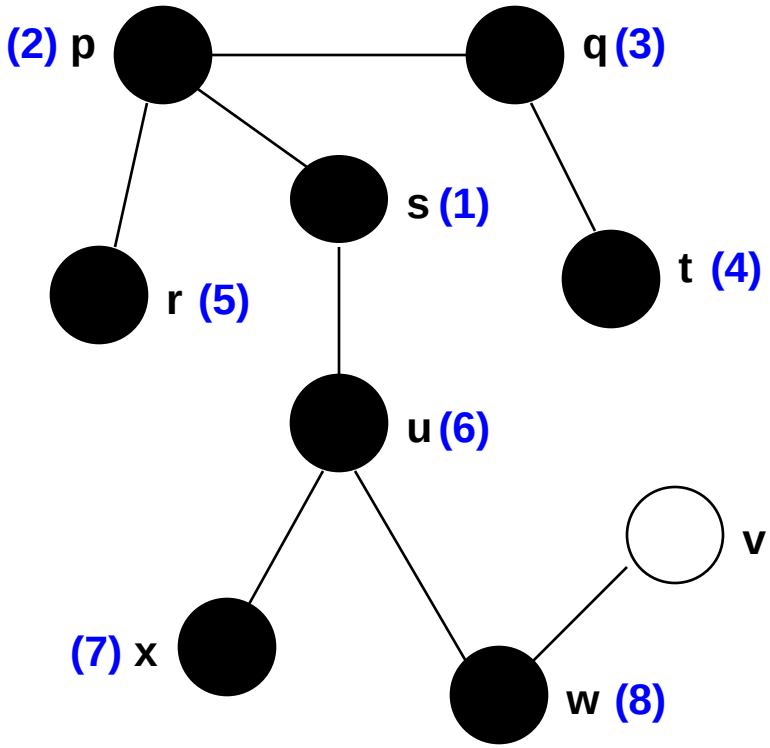


funkcja DFS (s, Cel)
  dla każdego wierzchołka x innego niż s
    oznacz x jako nie odwiedzony
  DFS_VERT(s)

Gdzie:

DFS_VERT(u)
  /* wykorzystaj u, np.:
    jeśli u=Cel to .... */
  zapamiętaj_że_odwiedzony(u)
  dla każdego wierzchołka v∈LS[u]
    jeżeli nie_odwiedzony(u)
      DFS_VERT(v)

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

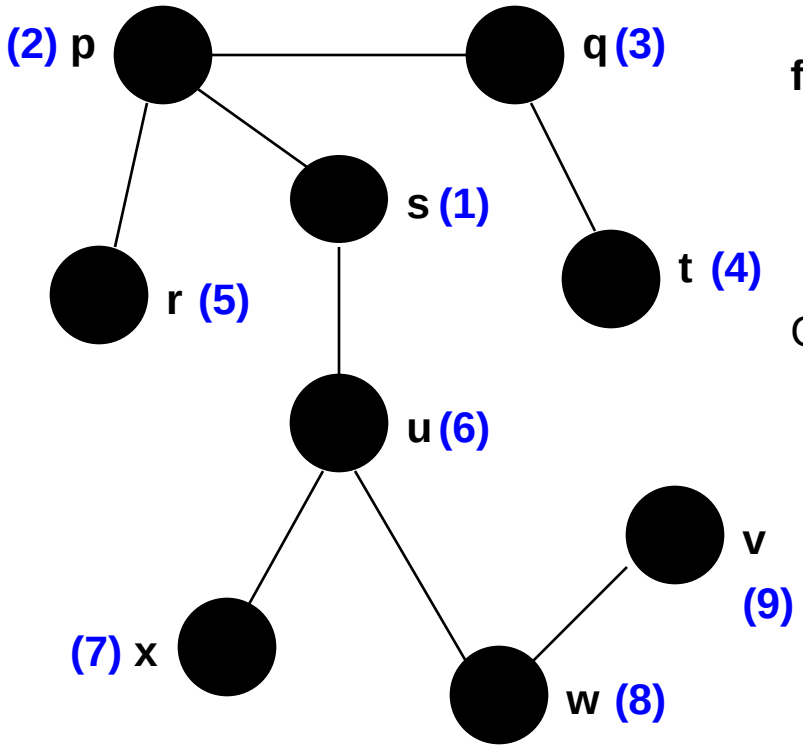


funkcja DFS (s, Cel)
  dla każdego wierzchołka x innego niż s
    oznacz x jako nie odwiedzony
  DFS_VERT(s)

Gdzie:

DFS_VERT(u)
  /* wykorzystaj u, np.:
    jeśli u=Cel to .... */
  zapamiętaj_że_odwiedzony(u)
  dla każdego wierzchołka v∈LS[u]
    jeżeli nie_odwiedzony(u)
      DFS_VERT(v)

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)



funkcja DFS (s, Cel)
  dla każdego wierzchołka x innego niż s
    oznacz x jako nie odwiedzony
  DFS_VERT(s)

Gdzie:

DFS_VERT(u)
  /* wykorzystaj u, np.:
    jeśli u=Cel to .... */
  zapamiętaj_że_odwiedzony(u)
  dla każdego wierzchołka v∈LS[u]
    jeżeli nie_odwiedzony(u)
      DFS_VERT(v)

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

Złożoność pamięciowa DFS w przypadku drzewa jest o wiele mniejsza niż dla BFS, ponieważ algorytm w każdym momencie wymaga zapamiętania tylko ścieżki od korzenia do bieżącego węzła a nie, jak BFS, zapamiętywania wszystkich węzłów w danej odległości od korzenia (co rośnie lawinowo w funkcji długości ścieżki).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

Złożoność pamięciowa DFS w przypadku drzewa jest o wiele mniejsza niż dla BFS, ponieważ algorytm w każdym momencie wymaga zapamiętania tylko ścieżki od korzenia do bieżącego węzła a nie, jak BFS, zapamiętywania wszystkich węzłów w danej odległości od korzenia (co rośnie lawinowo w funkcji długości ścieżki).

Złożoność czasowa DFS: O(|V| + |E|) (jak dla BFS),
gdzie |V| to liczba węzłów, a |E| to liczba krawędzi w grafie,
ponieważ trzeba przebyć wszystkie krawędzie i odwiedzić wszystkie wierzchołki.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przeszukiwanie w głąb (algorytm **DFS** - **D**eep **F**irst **S**earch)

Złożoność pamięciowa DFS w przypadku drzewa jest o wiele mniejsza niż dla BFS, ponieważ algorytm w każdym momencie wymaga zapamiętania tylko ścieżki od korzenia do bieżącego węzła a nie, jak BFS, zapamiętywania wszystkich węzłów w danej odległości od korzenia (co rośnie lawinowo w funkcji długości ścieżki).

Złożoność czasowa DFS: O(|V| + |E|) (jak dla BFS),
gdzie |V| to liczba węzłów, a |E| to liczba krawędzi w grafie, ponieważ trzeba przebyć wszystkie krawędzie i odwiedzić wszystkie wierzchołki.

Przykładowe zastosowania:
- znajdowanie najkrótszej ścieżki pomiędzy dwoma węzłami,
- sprawdzanie, czy istnieje ścieżka pomiędzy dwoma węzłami,
- jako strategia poszukiwania rozwiązań problemów.

Wyznaczanie minimalnego drzewa rozpinającego
MST (Minimal Spanning Tree)

Algorytm Prima

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

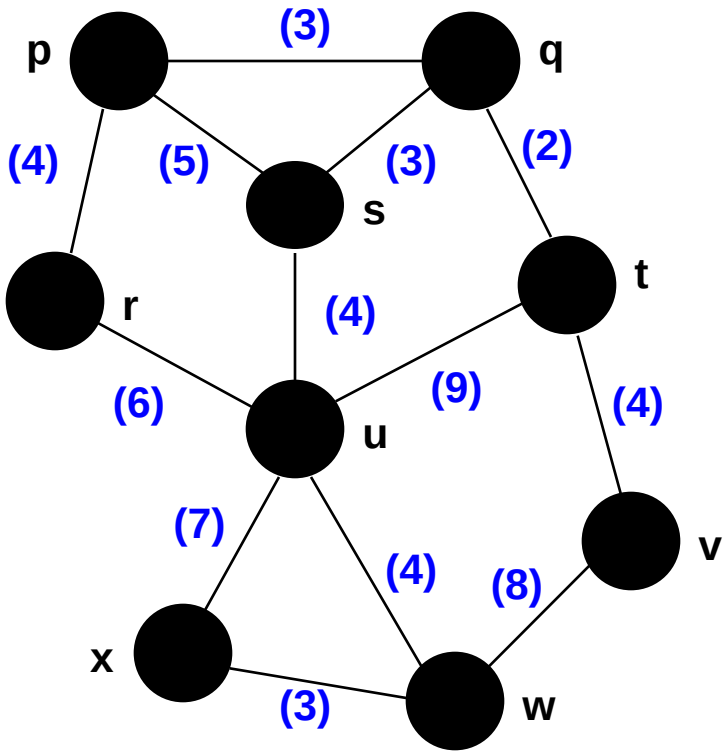
Po zakończeniu algorytmu utworzone drzewo jest minimalnym drzewem rozpinającym (MST).

Jest to algorytm zachłanny o złożoności czasowej $O(|E| \log(|V|))$.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład

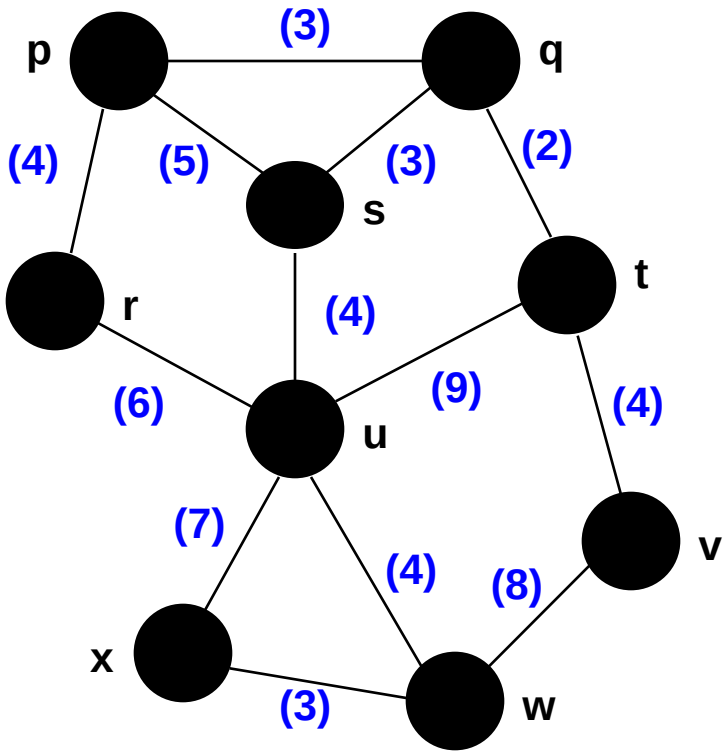


Graf, dla którego należy znaleźć MST

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład

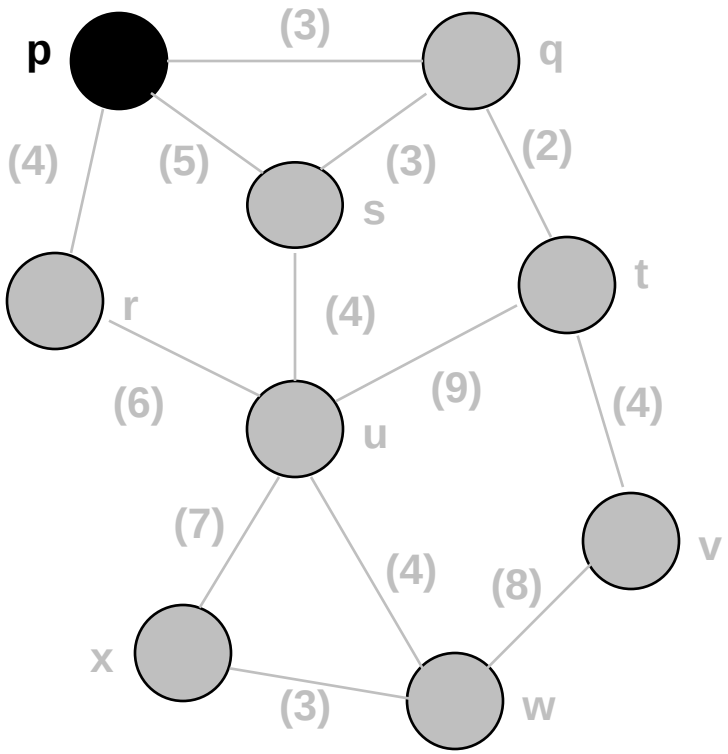


1. Utwórz drzewo zawierające jeden wierzchołek, dowolnie wybrany z grafu.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład

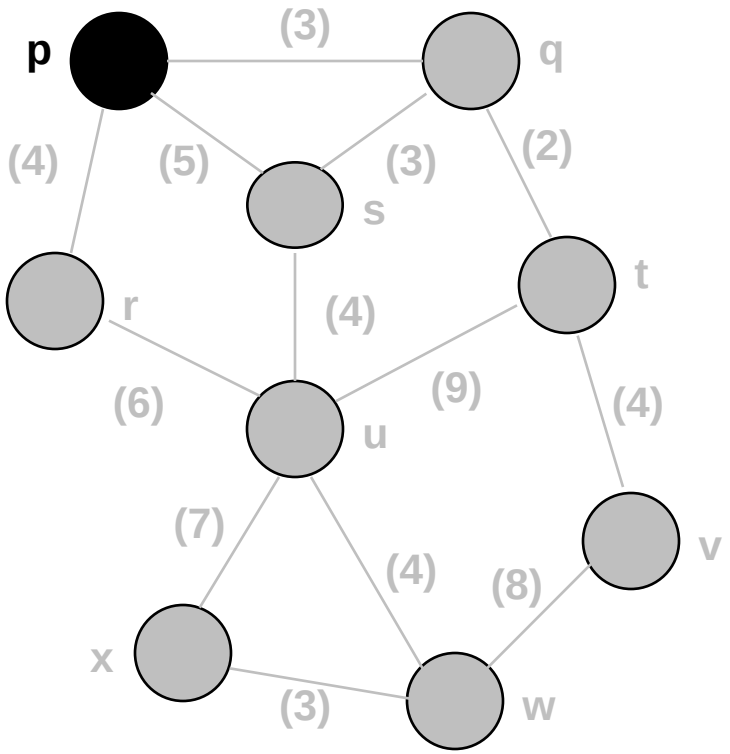


1. Utwórz drzewo zawierające jeden wierzchołek, dowolnie wybrany z grafu.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład



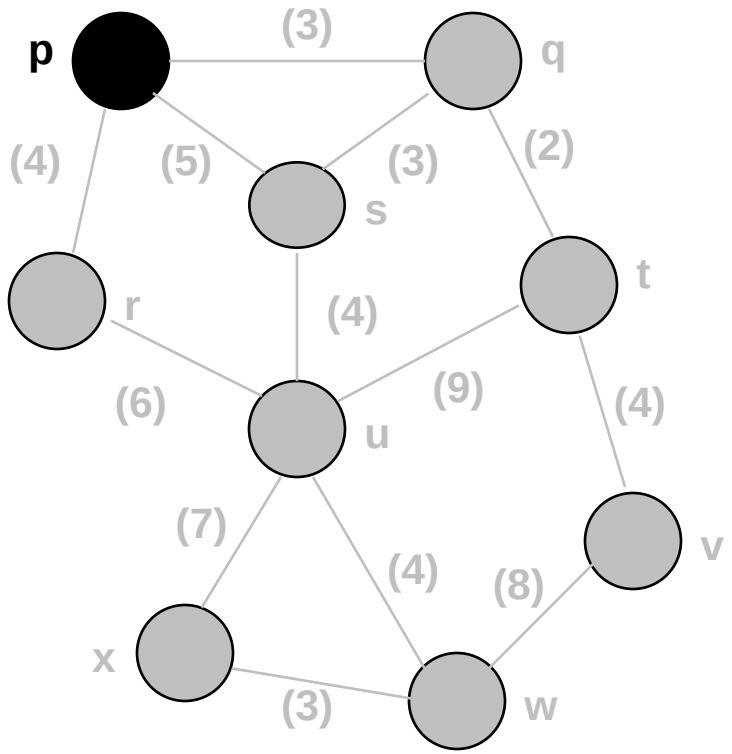
3 (q)
4 (r)
5 (s)

2 (q, t)
3 (p, q)
3 (q, s)
3 (w, x)
4 (p, r)
4 (s, u)
4 (t, v)
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

2. Utwórz kolejkę priorytetową, zawierającą wierzchołki osiągalne z MST (w tym momencie zawiera jeden wierzchołek, więc na początku w kolejce będą sąsiedzi początkowego wierzchołka), o priorytecie najmniejszego kosztu dotarcia do danego wierzchołka z MST.

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład



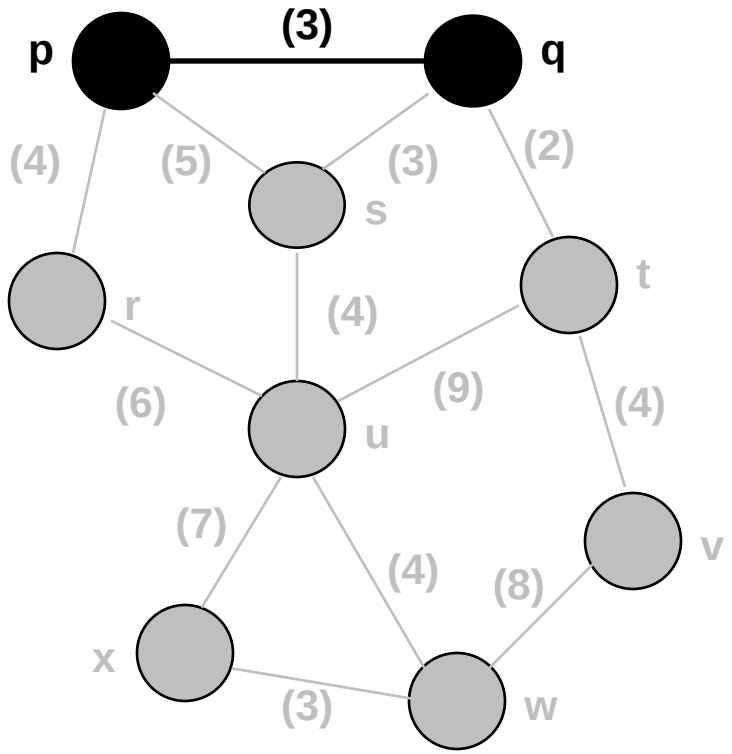
3 (q)
4 (r)
5 (s)

2 (q, t)
3 (p, q)
3 (q, s)
3 (w, x)
4 (p, r)
4 (s, u)
4 (t, v)
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:
   - wśród nieprzetworzonych wierzchołków (spoza obecnego MST) wybierz ten, dla którego koszt dojścia z obecnego MST jest najmniejszy,
   - dodaj go do obecnego MST i zaktualizuj kolejkę priorytetową, uwzględniając nowe krawędzie wychodzące z dodanego wierzchołka

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład



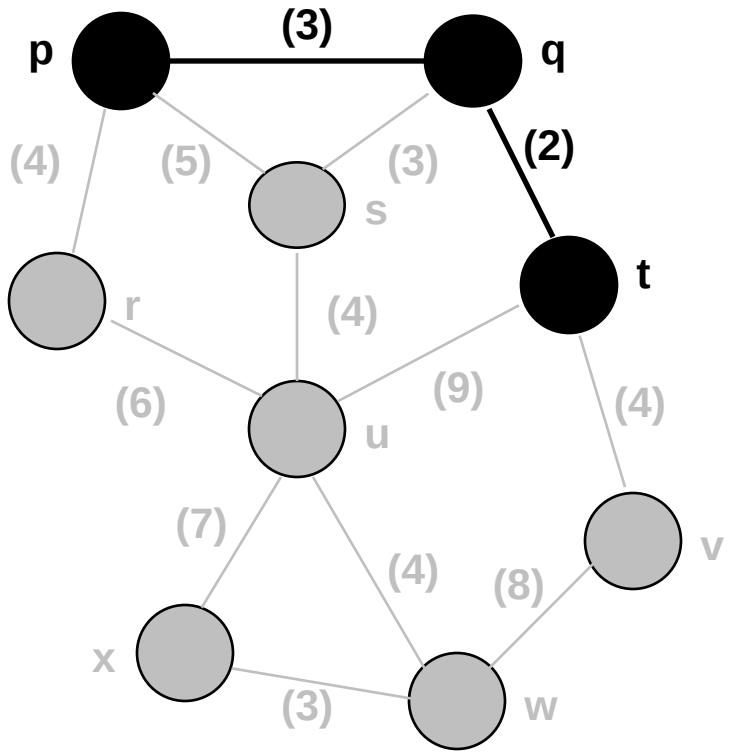
~~3 (q)~~	**2 (q, t)**
2 (t)	~~**3 (p, q)**~~
3 (s)	**3 (q, s)**
4 (r)	3 (w, x)
	4 (p, r)
	4 (s, u)
	4 (t, v)
	4 (u, w)
	~~5 (p, s)~~
	6 (r, u)
	7 (u, x)
	8 (v, w)
	9 (t, u)

3. Dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:
   - wśród nieprzetworzonych wierzchołków (spoza obecnego MST) wybierz ten, dla którego koszt dojścia z obecnego MST jest najmniejszy,
   - dodaj go do obecnego MST i zaktualizuj kolejkę priorytetową, uwzględniając nowe krawędzie wychodzące z dodanego wierzchołka

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład

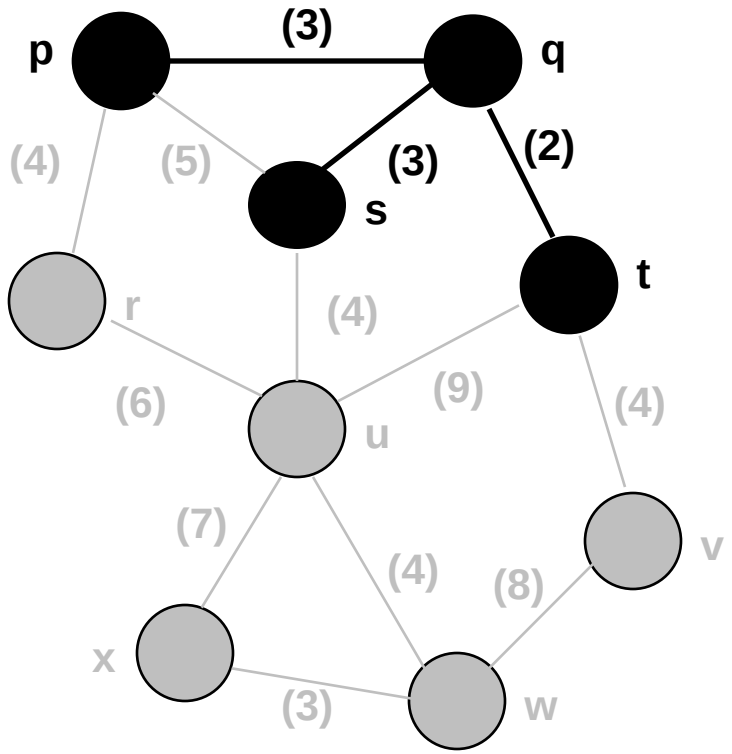


~~**2 (t)**~~	~~**2 (q, t)**~~
3 (s)	3 (q, s**)**
4 (r)	3 (w, x)
4 (v)	4 (p, r)
9 (u)	4 (s, u)
	4 (t, v)
	4 (u, w)
	6 (r, u)
	7 (u, x)
	8 (v, w)
	9 (t, u)

3. Dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:
   - wśród nieprzetworzonych wierzchołków (spoza obecnego MST) wybierz ten, dla którego koszt dojścia z obecnego MST jest najmniejszy,
   - dodaj go do obecnego MST i zaktualizuj kolejkę priorytetową, uwzględniając nowe krawędzie wychodzące z dodanego wierzchołka

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład



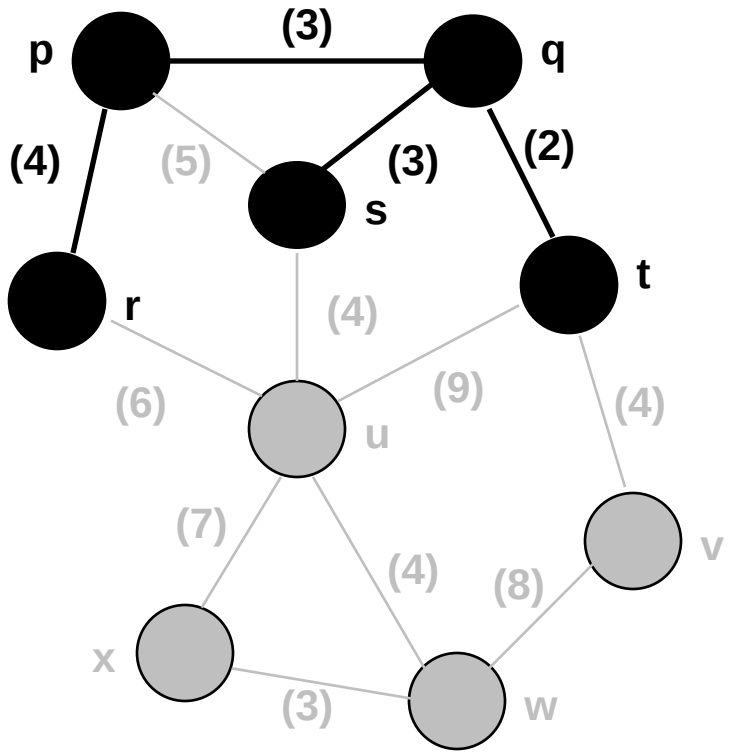
~~**3 (s)**~~	~~**3 (q, s)**~~
4 (r)	3 (w, x)
4 (u)	4 (p, r)
4 (v)	**4 (s, u)**
	4 (t, v)
	4 (u, w)
	6 (r, u)
	7 (u, x)
	8 (v, w)
	9 (t, u)

3. Dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:
   - wśród nieprzetworzonych wierzchołków (spoza obecnego MST) wybierz ten, dla którego koszt dojścia z obecnego MST jest najmniejszy,
   - dodaj go do obecnego MST i zaktualizuj kolejkę priorytetową, uwzględniając nowe krawędzie wychodzące z dodanego wierzchołka

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład



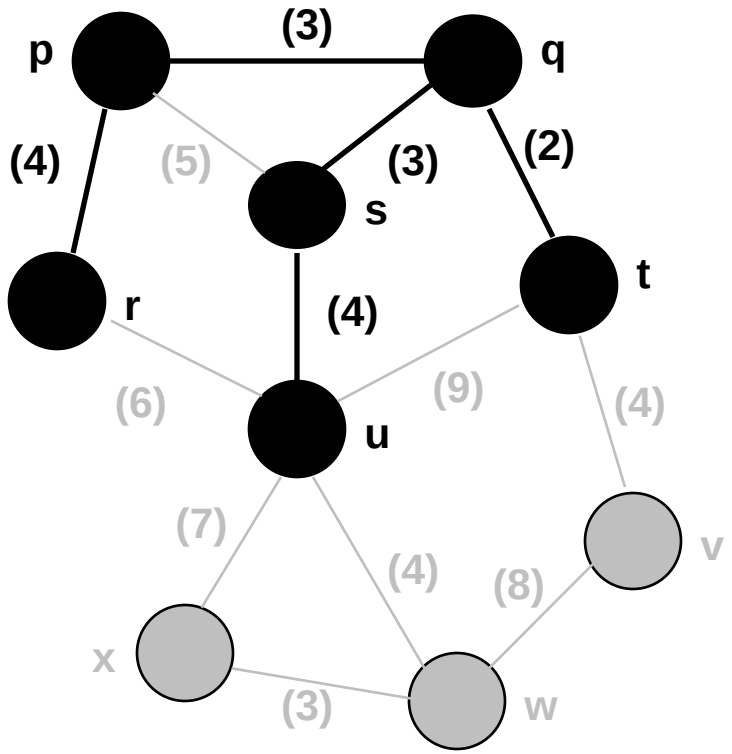
~~**4 (r)**~~
4 (u)
4 (v)

3 (w, x)
~~**4 (p, r)**~~
4 (s, u)
4 (t, v)
4 (u, w)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:
   - wśród nieprzetworzonych wierzchołków (spoza obecnego MST) wybierz ten, dla którego koszt dojścia z obecnego MST jest najmniejszy,
   - dodaj go do obecnego MST i zaktualizuj kolejkę priorytetową, uwzględniając nowe krawędzie wychodzące z dodanego wierzchołka

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład

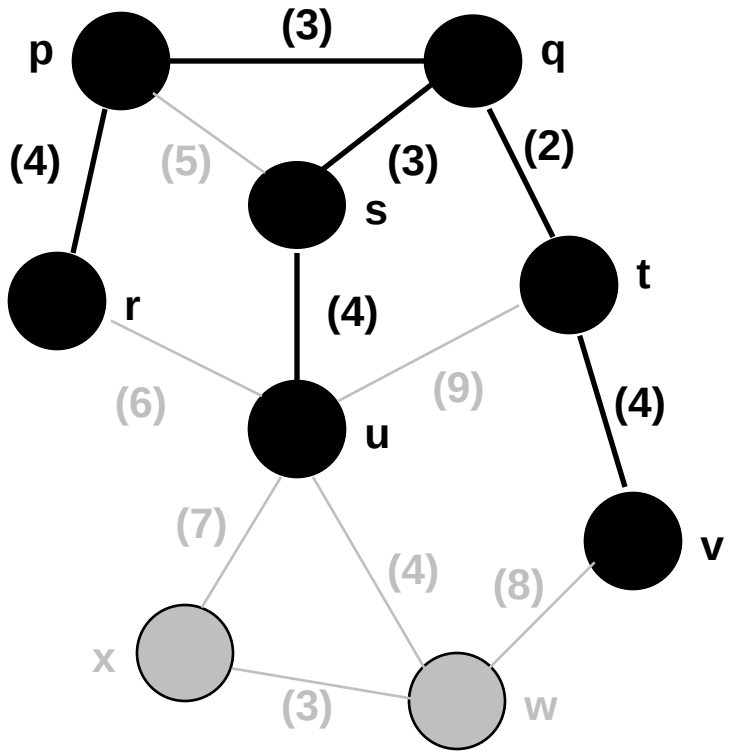


~~**4 (u)**~~	3 (w, x)
4 (v)	~~**4 (s, u)**~~
4 (w)	4 (t, v)
	4 (u, w)
	6 (r, u)
	7 (u, x)
	8 (v, w)
	9 (t, u)

3. Dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:
   - wśród nieprzetworzonych wierzchołków (spoza obecnego MST) wybierz ten, dla którego koszt dojścia z obecnego MST jest najmniejszy,
   - dodaj go do obecnego MST i zaktualizuj kolejkę priorytetową, uwzględniając nowe krawędzie wychodzące z dodanego wierzchołka

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład



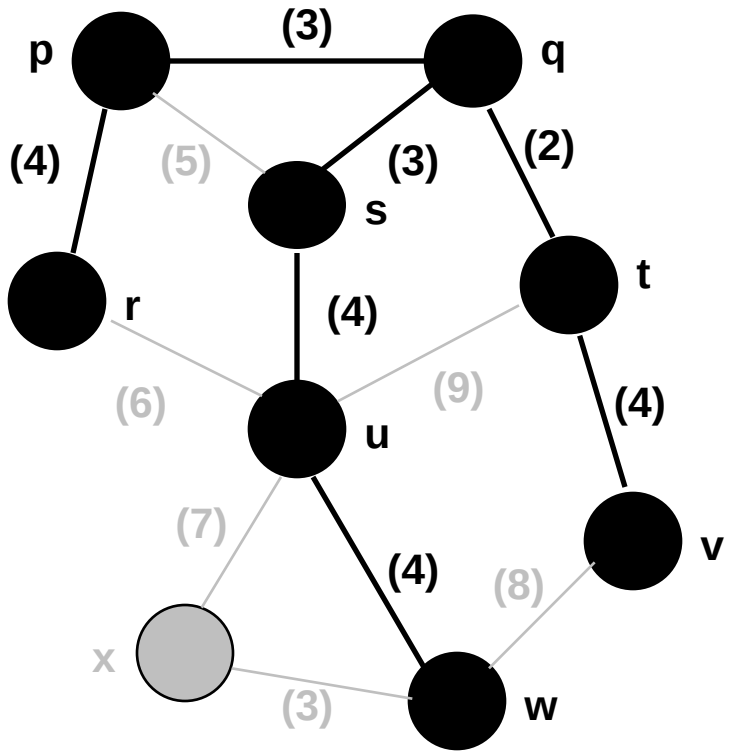
~~**4 (v)**~~
4 (w)

3 (w, x)
~~**4 (t, v)**~~
4 (u, w)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:
   - wśród nieprzetworzonych wierzchołków (spoza obecnego MST) wybierz ten, dla którego koszt dojścia z obecnego MST jest najmniejszy,
   - dodaj go do obecnego MST i zaktualizuj kolejkę priorytetową, uwzględniając nowe krawędzie wychodzące z dodanego wierzchołka

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład



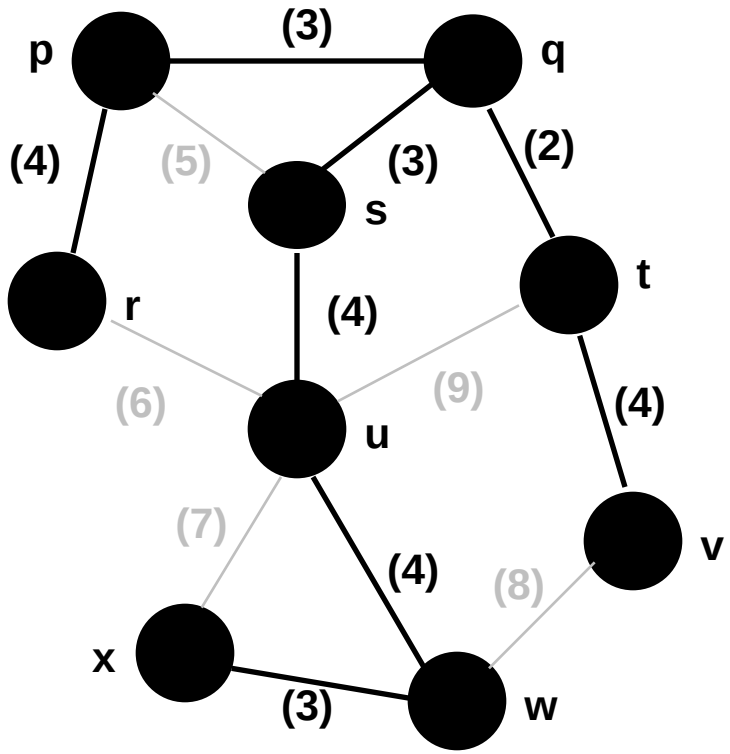
~~**4 (w)**~~
3 (x)

3 (w, x)
~~**4 (u, w)**~~
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:
   - wśród nieprzetworzonych wierzchołków (spoza obecnego MST) wybierz ten, dla którego koszt dojścia z obecnego MST jest najmniejszy,
   - dodaj go do obecnego MST i zaktualizuj kolejkę priorytetową, uwzględniając nowe krawędzie wychodzące z dodanego wierzchołka

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Prima - przykład



~~3 (x)~~

~~3 (w, x)~~
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

Drzewo obejmuje wszystkie wierzchołki grafu - utworzone drzewo jest minimalnym drzewem rozpinającym (MST).

Wyznaczanie minimalnego drzewa rozpinającego

MST (Minimal Spanning Tree)

Algorytm Kruskala

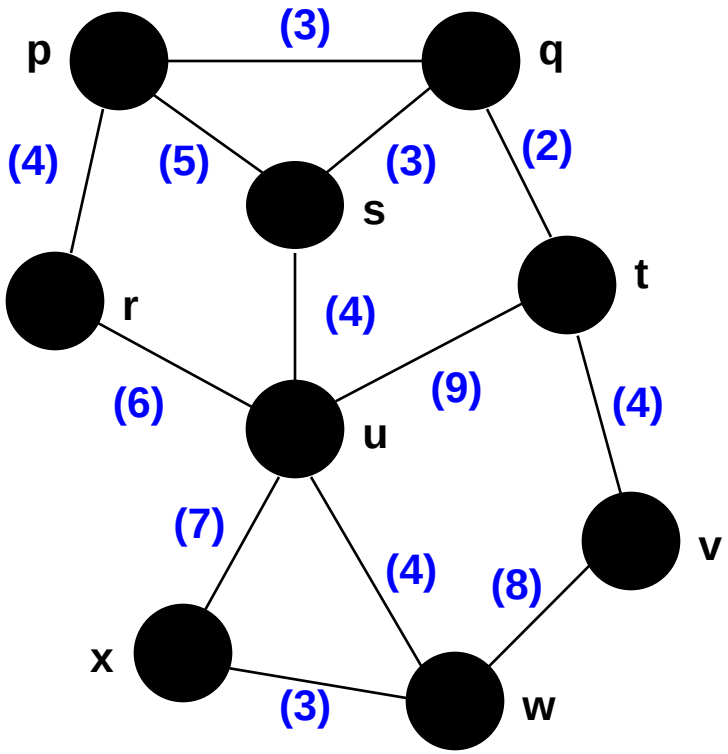
1. Utwórz las L z wierzchołków oryginalnego grafu – każdy wierzchołek jest na początku osobnym drzewem.
2. Utwórz zbiór S zawierający wszystkie krawędzie oryginalnego grafu.
3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź o minimalnej wadze.
   - Jeśli krawędź ta łączyła dwa różne drzewa, 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.

Jest to algorytm zachłanny o złożoności czasowej $O(|E| \log(|V|))$.

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład

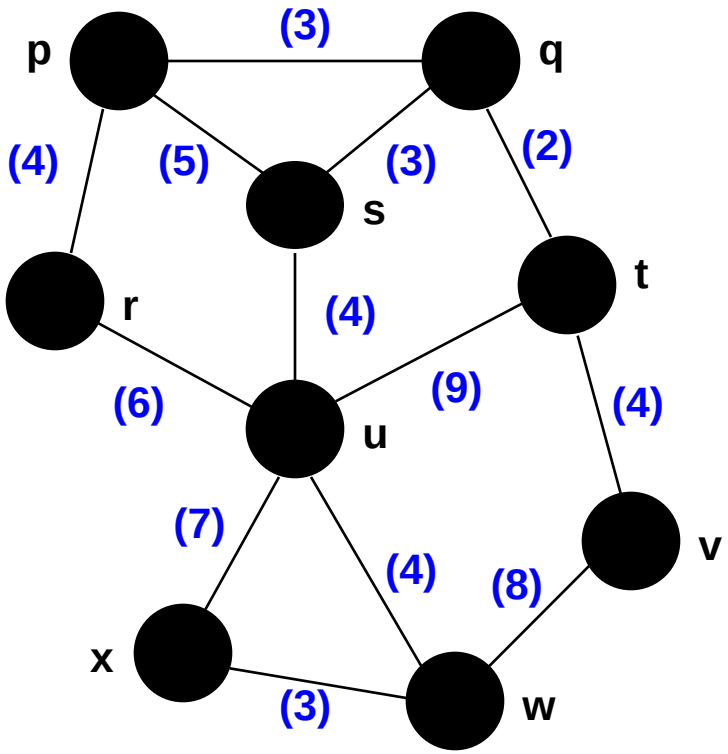


Graf, dla którego należy znaleźć MST

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład

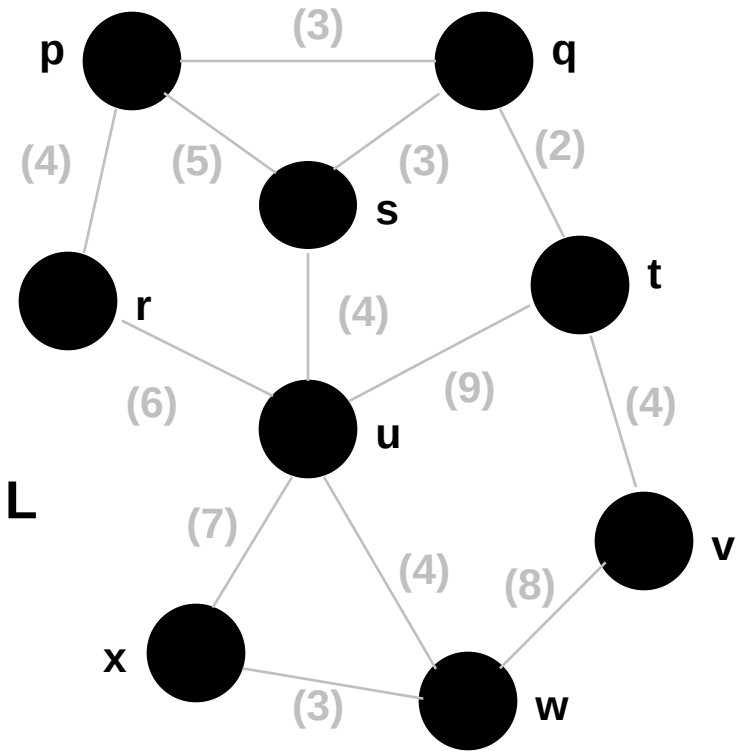


1. Utwórz las L z wierzchołków oryginalnego grafu – każdy wierzchołek jest na początku osobnym drzewem.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

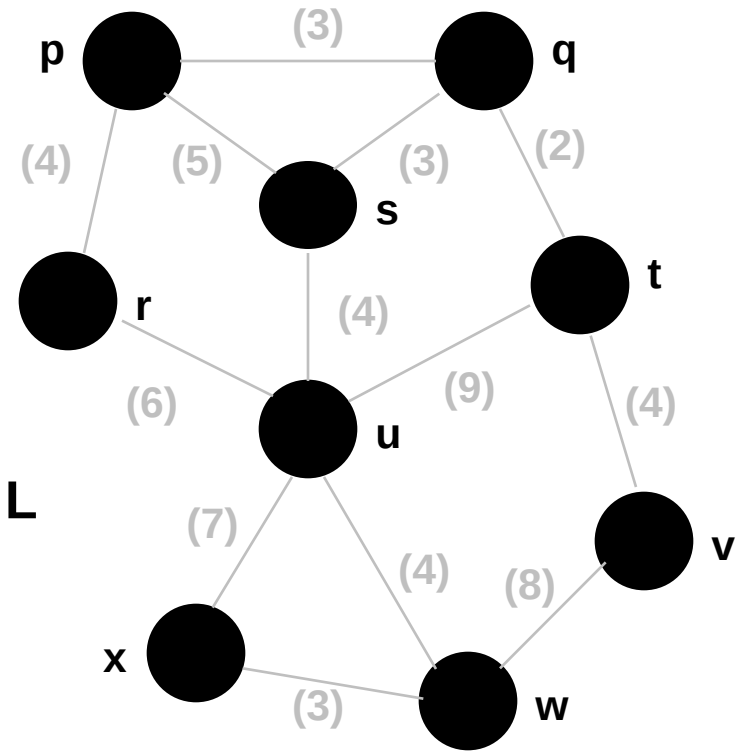
Algorytm Kruskala - przykład



1. Utwórz las L z wierzchołków oryginalnego grafu – każdy wierzchołek jest na początku osobnym drzewem.

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



L

Zbiór S

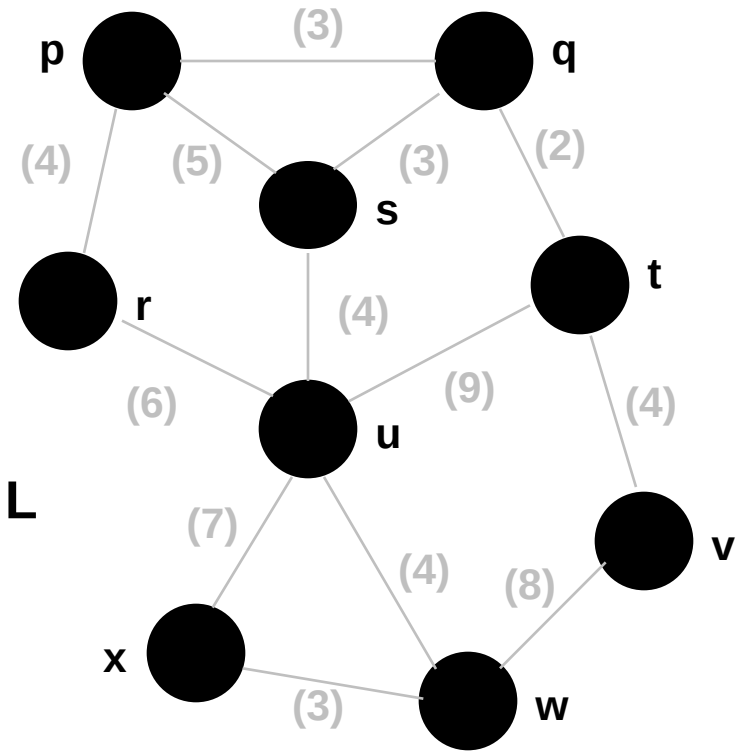
2 (q, t)
3 (p, q)
3 (q, s)
3 (w, x)
4 (p, r)
4 (s, u)
4 (t, v)
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

2. Utwórz zbiór S zawierający wszystkie krawędzie oryginalnego grafu.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



Zbiór S

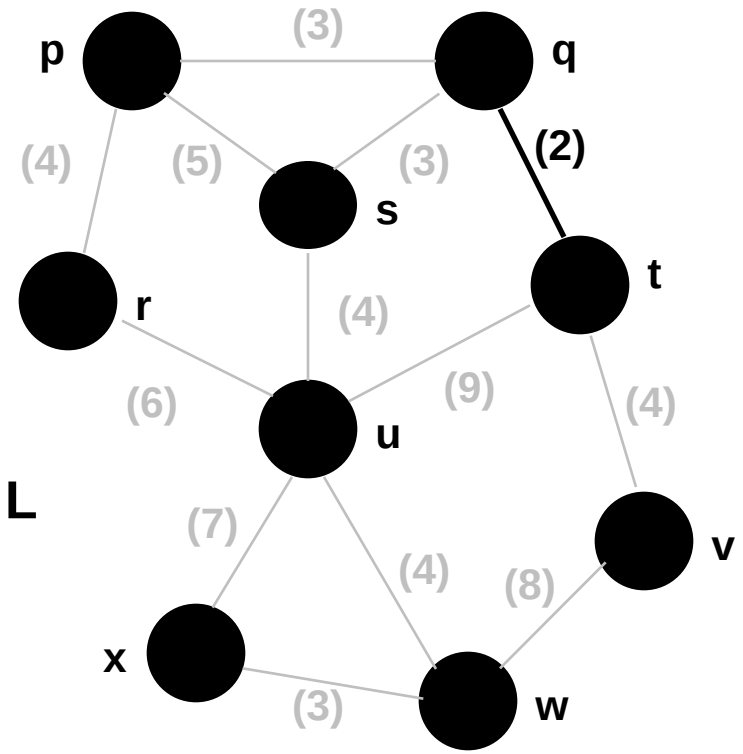
2 (q, t)
3 (p, q)
3 (q, s)
3 (w, x)
4 (p, r)
4 (s, u)
4 (t, v)
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



Zbiór S

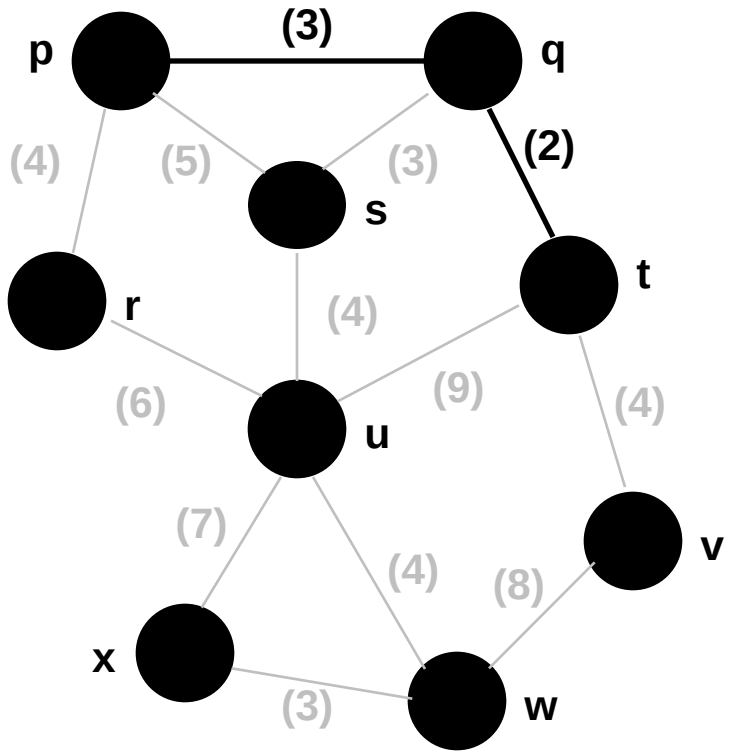
~~2 (q, t)~~
3 (p, q)
3 (q, s)
3 (w, x)
4 (p, r)
4 (s, u)
4 (t, v)
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



Zbiór S

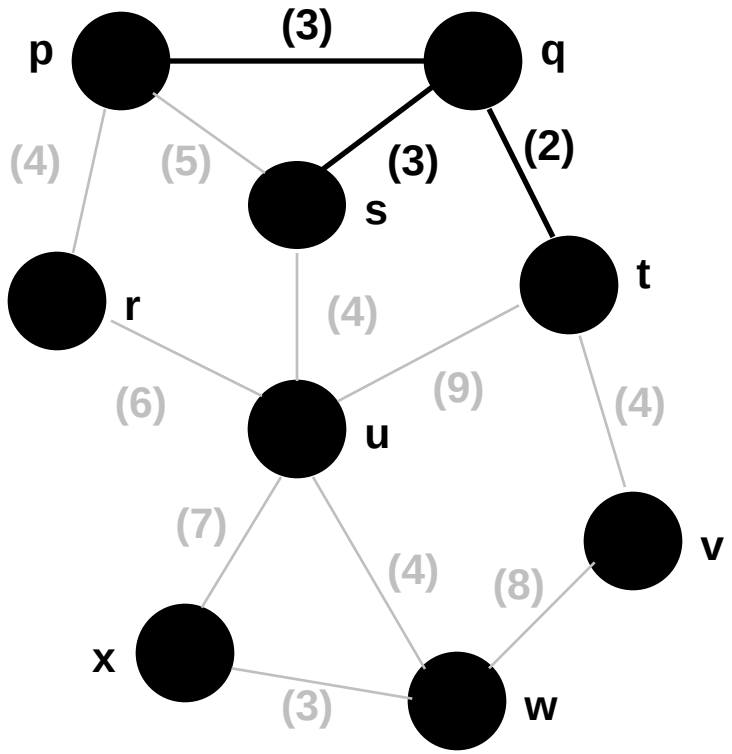
~~2 (q, t)~~
~~3 (p, q)~~
3 (q, s)
3 (w, x)
4 (p, r)
4 (s, u)
4 (t, v)
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



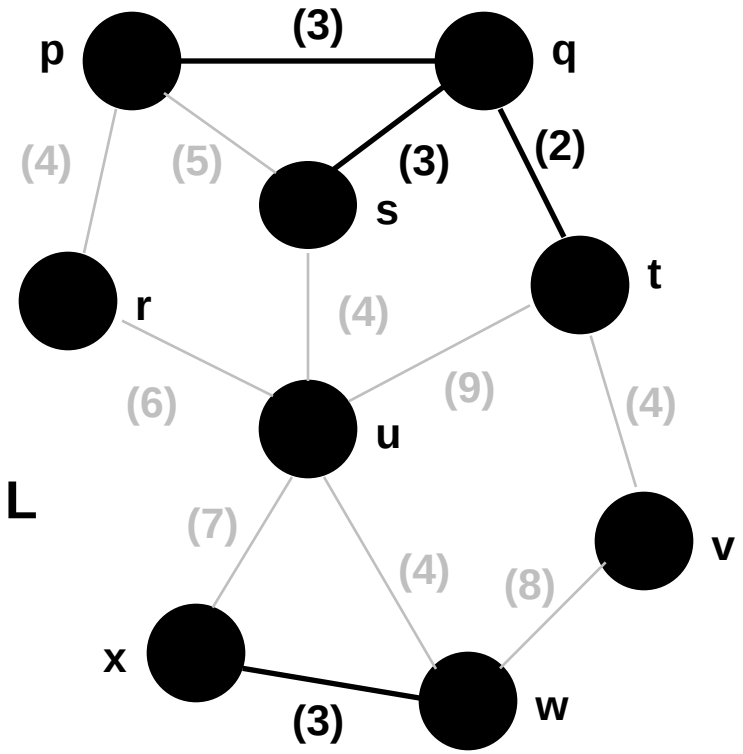
Zbiór S

~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
3 (w, x)
4 (p, r)
4 (s, u)
4 (t, v)
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



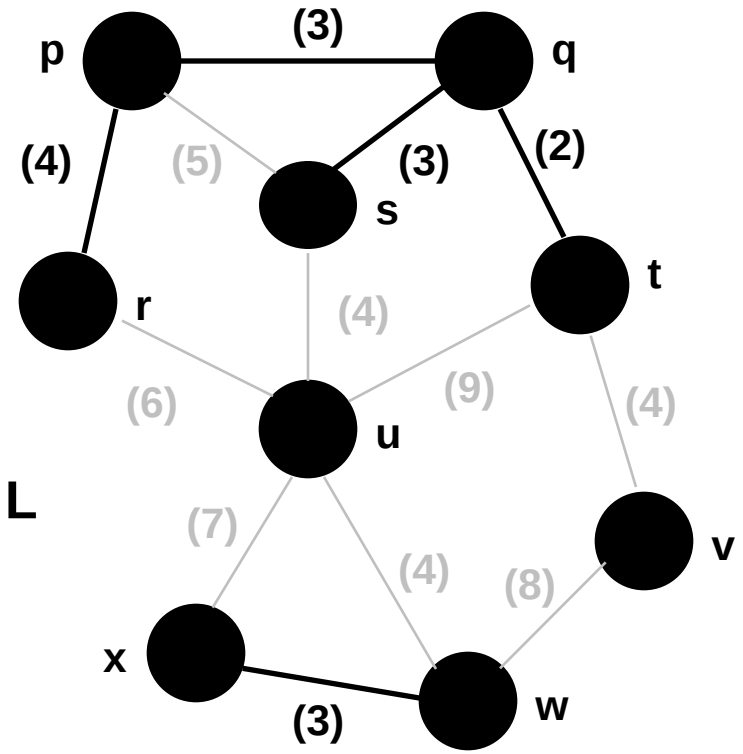
Zbiór S

~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
~~3 (w, x)~~
4 (p, r)
4 (s, u)
4 (t, v)
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



Zbiór S

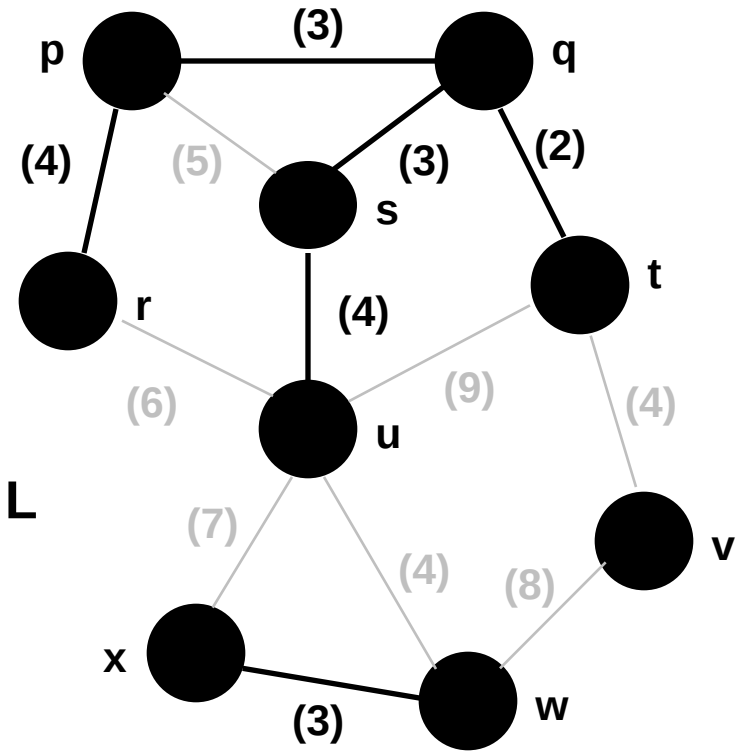
~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
~~3 (w, x)~~
~~4 (p, r)~~
4 (s, u)
4 (t, v)
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



Zbiór S

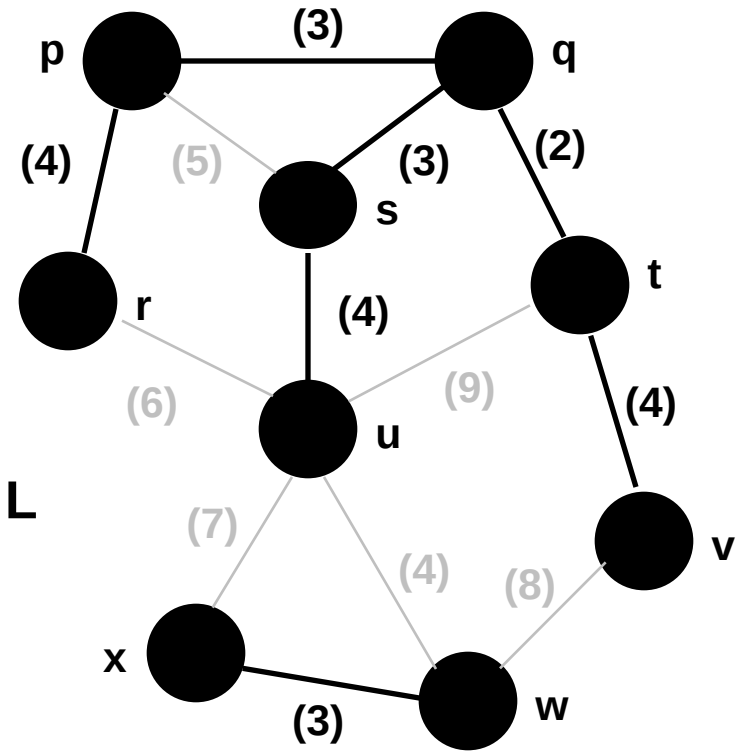
~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
~~3 (w, x)~~
~~4 (p, r)~~
~~4 (s, u)~~
4 (t, v)
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



Zbiór S

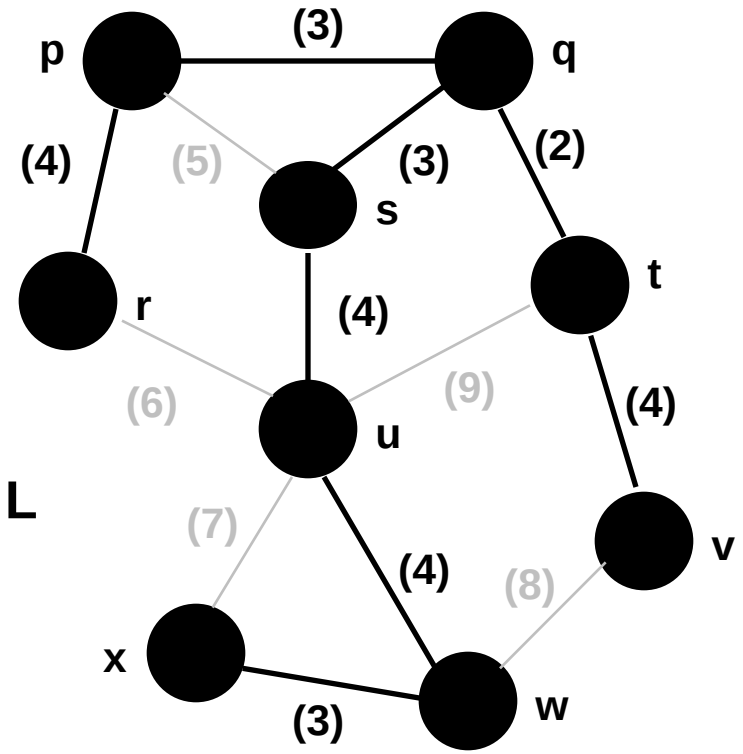
~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
~~3 (w, x)~~
~~4 (p, r)~~
~~4 (s, u)~~
~~4 (t, v)~~
4 (u, w)
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



Zbiór S

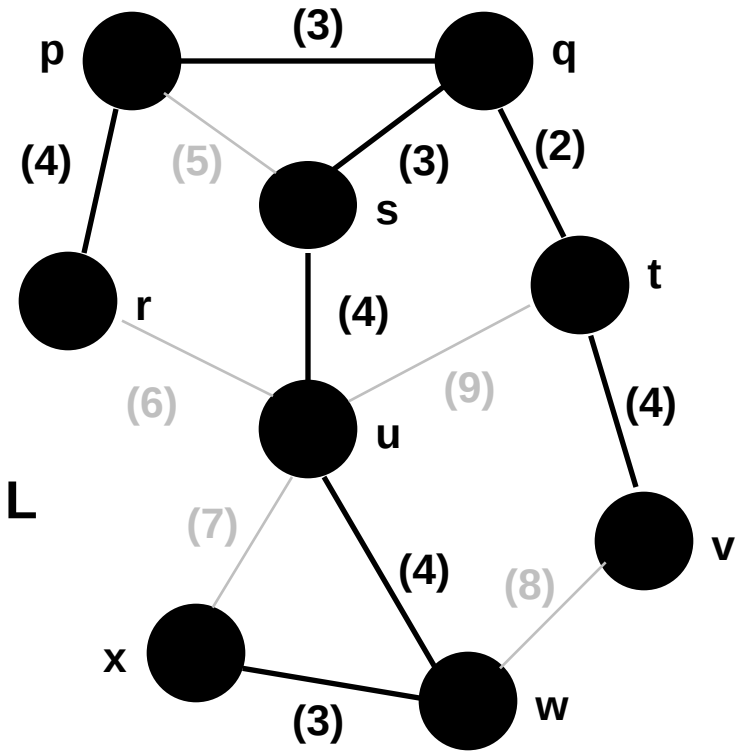
~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
~~3 (w, x)~~
~~4 (p, r)~~
~~4 (s, u)~~
~~4 (t, v)~~
~~4 (u, w)~~
5 (p, s)
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



Zbiór S

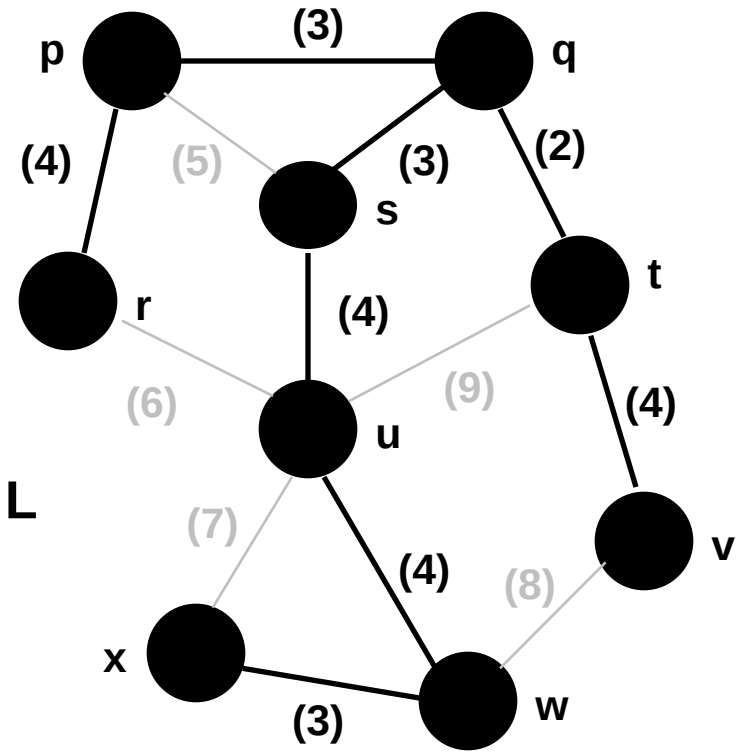
~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
~~3 (w, x)~~
~~4 (p, r)~~
~~4 (s, u)~~
~~4 (t, v)~~
~~4 (u, w)~~
~~**5 (p, s)**~~ **<= nie łączy drzew!**
6 (r, u)
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



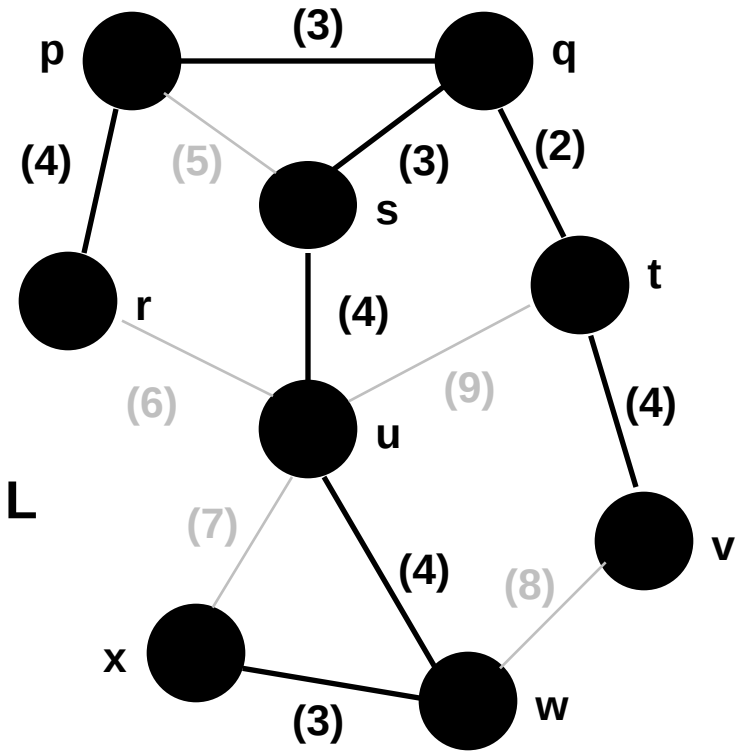
Zbiór S

~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
~~3 (w, x)~~
~~4 (p, r)~~
~~4 (s, u)~~
~~4 (t, v)~~
~~4 (u, w)~~
~~5 (p, s)~~ <= nie łączy drzew!
~~6 (r, u)~~ <= nie łączy drzew!
7 (u, x)
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



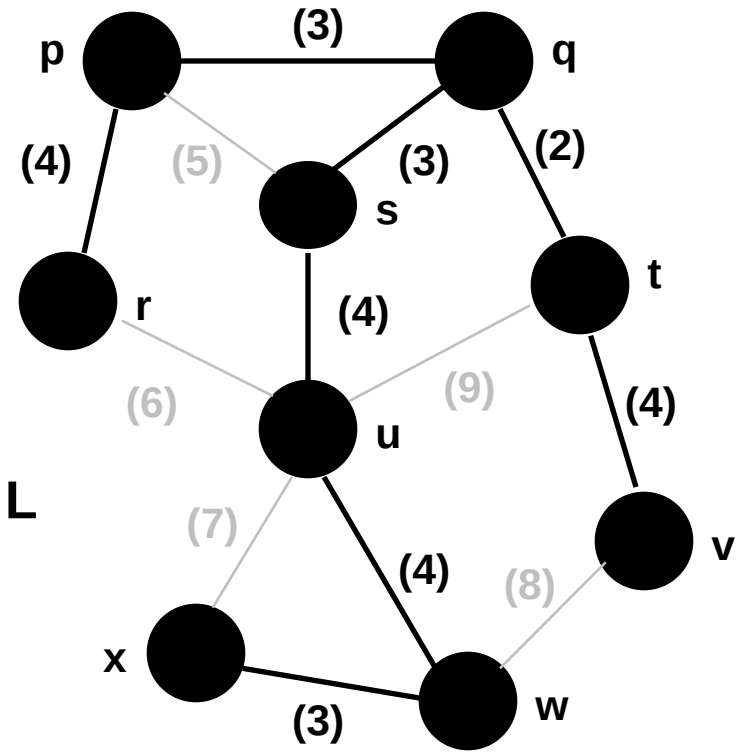
Zbiór S

~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
~~3 (w, x)~~
~~4 (p, r)~~
~~4 (s, u)~~
~~4 (t, v)~~
~~4 (u, w)~~
~~5 (p, s)~~ <= nie łączy drzew!
~~6 (r, u)~~ <= nie łączy drzew!
~~7 (u, x)~~ <= nie łączy drzew!
8 (v, w)
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



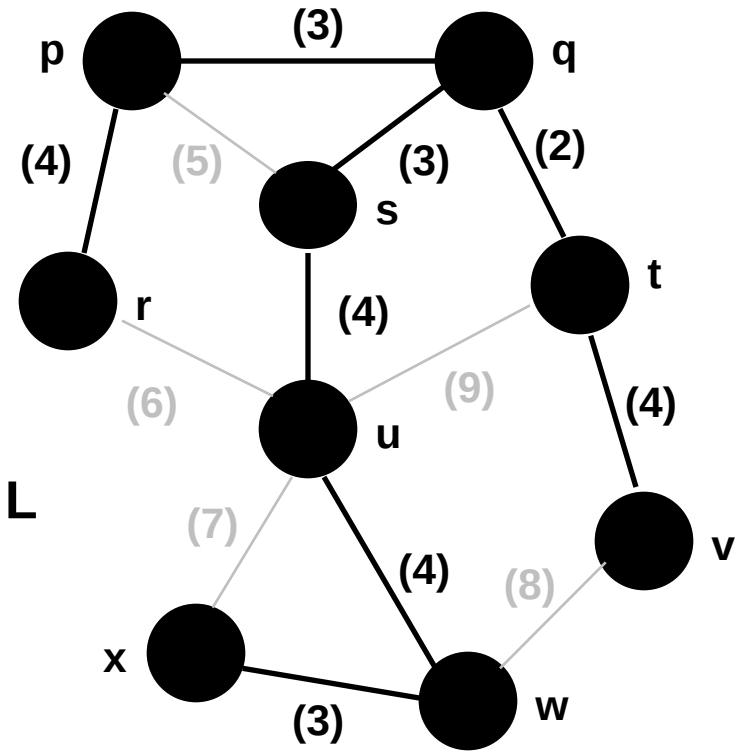
Zbiór S

~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
~~3 (w, x)~~
~~4 (p, r)~~
~~4 (s, u)~~
~~4 (t, v)~~
~~4 (u, w)~~
~~5 (p, s)~~ <= nie łączy drzew!
~~6 (r, u)~~ <= nie łączy drzew!
~~7 (u, x)~~ <= nie łączy drzew!
~~8 (v, w)~~ <= nie łączy drzew!
9 (t, u)

3. Dopóki S nie jest pusty:
   - Wybierz i usuń z S krawędź 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.

Wyznaczanie minimalnego drzewa rozpinającego **MST**

Algorytm Kruskala - przykład



Zbiór S

~~2 (q, t)~~
~~3 (p, q)~~
~~3 (q, s)~~
~~3 (w, x)~~
~~4 (p, r)~~
~~4 (s, u)~~
~~4 (t, v)~~
~~4 (u, w)~~
~~5 (p, s)~~ <= nie łączy drzew!
~~6 (r, u)~~ <= nie łączy drzew!
~~7 (u, x)~~ <= nie łączy drzew!
~~8 (v, w)~~ <= nie łączy drzew!
~~9 (t, u)~~ <= nie łączy drzew!

S jest pusty - L jest minimalnym drzewem rozpinającym

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie najkrótszej ścieżki z pojedynczego źródła w grafie o nieujemnych wagach krawędzi

Algorytm Dijkstry

Oznaczenia: *s* - wierzchołek źródłowy, *w*(*i*,*j*) - waga krawędzi (*i*,*j*) w grafie.

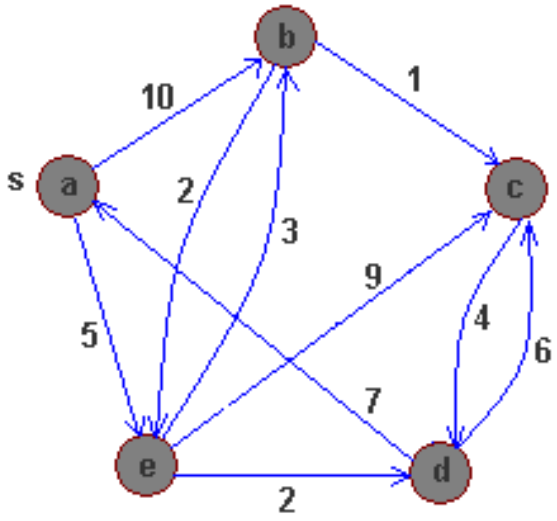
1. Utwórz tablicę *d* odległości od źródła dla wszystkich wierzchołków grafu. Na początku *d*[*s*] = 0, zaś *d*[*v*] = *nieskończoność* dla wszystkich pozostałych wierzchołków.
2. Utwórz kolejkę priorytetową *Q* wszystkich wierzchołków grafu. Priorytetem kolejki jest aktualnie wyliczona odległość od wierzchołka źródłowego *s*.
3. Dopóki kolejka nie jest pusta:
   - Usuń z kolejki wierzchołek *u* o najniższym priorytecie (wierzchołek najbliższy źródła, który nie został jeszcze rozważony)
   - Dla każdego sąsiada *v* wierzchołka *u* dokonaj *relaksacji* poprzez *u*: jeśli *d*[*u*] + *w*(*u*,*v*) < *d*[*v*] (poprzez *u* da się dojść do *v* szybciej niż dotychczasową ścieżką), to *d*[*v*]: = *d*[*u*] + *w*(*u*,*v*).

Na końcu tablica *d* zawiera najkrótsze odległości do wszystkich wierzchołków.

Dodatkowo, możemy w tablicy *poprzednik* przechowywać dla każdego wierzchołka numer jego bezpośredniego poprzednika na najkrótszej ścieżce, co pozwoli na odtworzenie pełnej ścieżki od źródła do każdego wierzchołka – przy każdej relaksacji w ostatnim punkcie, *u* staje się poprzednikiem *v*.

Wyznaczanie najkrótszej ścieżki z pojedynczego źródła w grafie o nieujemnych wagach krawędzi

Algorytm Dijkstry



	a	b	c	d	e
a	0	10	∞	∞	5
b	∞	0	1	∞	2
c	∞	∞	0	4	∞
d	7	∞	6	0	∞
e	∞	3	9	2	0

Q	D(a)	D(b)	D(c)	D(d)	D(e)
{b,c,d,e}	0	10	*	*	5
{b,c,d}	0	8	14	7	5
{b,c}	0	8	13	7	5
{c}	0	8	9	7	5
{}	0	8	9	7	5

Należy teraz zastosować algorytm konstrukcji drogi ze źródła s do wierzchołka v o danym „koszcie minimalnym” D(v) (z tabeli j/w)

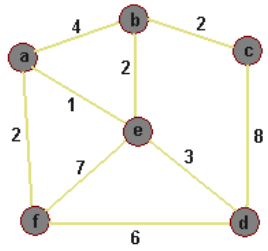
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wyznaczanie minimalnego drzewa rozpinającego

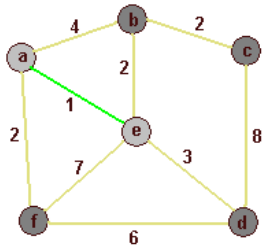
MST (Minimal Spanning Tree)

(przykłady z Wikipedii)

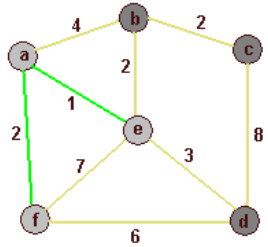
Algorytm Prima



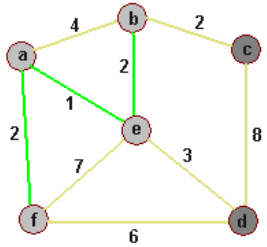
Wybieramy wierzchołek *a*.
Tworzymy posortowaną listę
L={[a,e,1],[a,f,2],[a,b,4]}
Wybieramy krawędź (a,e).



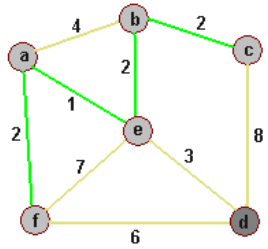
Dodajemy nowe krawędzie:
L={[a,f,2],[e,b,2],[e,d,3],[a,b,4],[e,f,7]}
Wybieramy krawędź (a,f).



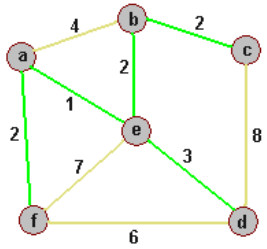
Krawędź [f,e,7] jest już na liście:
L={[e,b,2],[e,d,3],[a,b,4],[f,d,6],[e,f,7]}
Wybieramy krawędź (e,b).



Dodajemy krawędź (b,c):
L={[b,c,2],[e,d,3],[a,b,4],[f,d,6],[e,f,7]}
Wybieramy krawędź (b,c).



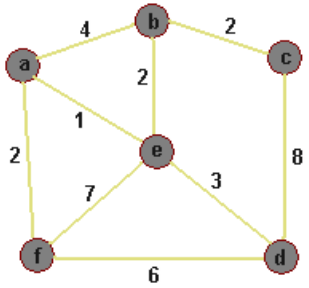
Dodajemy krawędź (c,d):
L={[e,d,3],[a,b,4],[f,d,6],[e,f,7],[c,d,8]}
Wybieramy krawędź (e,d).



Drzewo utworzone.

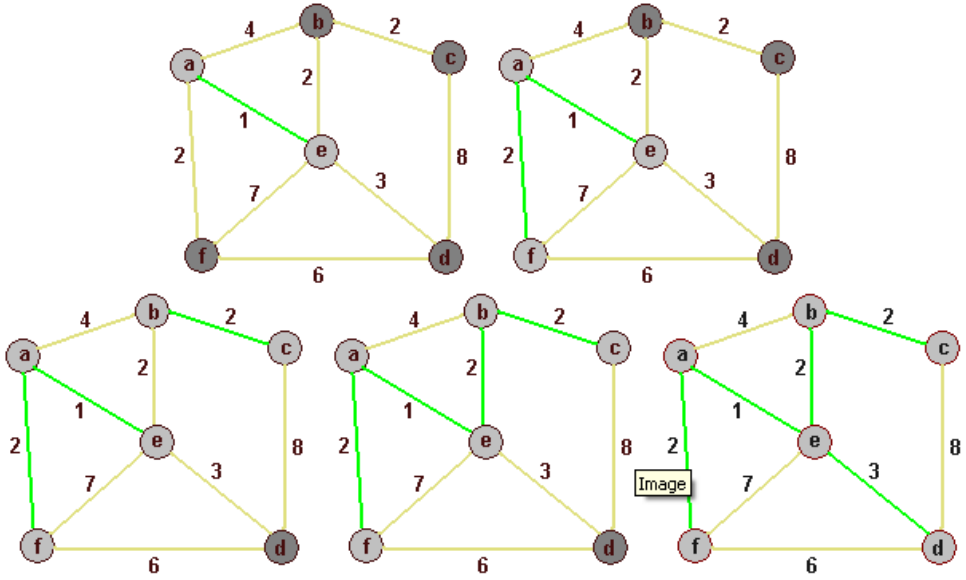
Algorytm Kruskala

Dany jest spójny graf nieskierowany:



Po posortowaniu krawędzi wg. wag otrzymamy:

1. Krawędź ae=1
2. Krawędź af=2
3. Krawędź bc=2
4. Krawędź be=2
5. Krawędź de=3
6. Krawędź ab=4
7. Krawędź fd=6
8. Krawędź ef=7
9. Krawędź cd=8



Wszystkie wierzchołki należą do jednego drzewa- minimalnego drzewa rozpinającego.
Suma wag krawędzi wchodzących w skład drzewa wynosi 10.