Algorytmy grafowe

Przypomnienie. Graf możemy reprezentować w pamięci na dwa sposoby:

- macierz sąsiedztwa
- listy sąsiedztwa

W algorytmach nie będziemy jawnie odwoływać się do tych struktur danych. Będziemy dopuszczać wyrażenia typu u–$v \in E$ oraz konstrukcje **for all** $v \in X$, na oznaczenie pętli, która wykonuje się $|X|$ razy, za każdym razem zmienna v oznacza inny wierzchołek ze zbioru X.

Przykład problemu grafowego. Dla danego grafu nieskierowanego obliczyć liczbę spójnych składowych i każdemu wierzchołkowi przypisać numer jego spójnej składowej.

Przeszukiwanie grafu

Często przy rozwiązywaniu problemów grafowych konieczne jest zebranie informacji o strukturze grafu. W tym celu najczęściej przeszukujemy graf, tzn. „wędrujemy” po grafie przechodząc wzdłuż krawędzi i odwiedzając kolejne wierzchołki.

Podczas przeszukiwania grafu wierzchołki będą odwiedzane w kolejności charakterystycznej dla wybranego algorytmu przeszukiwania. Nie jest obojętne, jaki algorytm wybierzemy – dobieramy najwłaściwszy algorytm do problemu. Poznamy dwie najważniejsze metody przeszukiwania grafu:

- Przeszukiwanie w głąb (DFS, Depth First Search)
- Przeszukiwanie wszerz (BFS, Breadth First Search).

Przeszukiwanie w głąb

for all $v \in V$ **do**
    $Odwiedzony[u] \leftarrow False$
for all $v \in V$ **do**
    if not $Odwiedzony[v]$ **then**
        OdwiedźDFS(v)

procedure OdwiedźDFS(u)
$Odwiedzony[u] \leftarrow True$
for all $x \in Sasiedzi[u]$ **do**
    if not $Odwiedzony[x]$ **then**
        OdwiedźDFS(x)

Za każdym razem, gdy wykonuje się obrót pętli **for all** $x \in Sasiedzi[u]$, sprawdzana jest jedna krawędź grafu (od u do x). Jeśli wierzchołek x nie jest jeszcze odwiedzony, przechodzimy od u do x.

Algorytm sprawdza każdą krawędź grafu co najwyżej dwa razy. Dlatego złożoność czasowa algorytmu jest $\Theta(n + m)$.

Zastosowanie: spójne składowe

for all $v \in V$ **do**
    $Odwiedzony[u] \leftarrow False$
$Numer \leftarrow 0$ {zmienna globalna}
for all $v \in V$ **do**
    if not $Odwiedzony[v]$ **then**
        $Numer \leftarrow Numer + 1$
        OdwiedźDFS(v)

procedure OdwiedźDFS(u)

$Odwiedzony[u] \leftarrow True$
$Skladowa[u] \leftarrow Numer$
for all $x \in Sasiedzi[u]$ **do**
    if not $Odwiedzony[x]$ **then**
        OdwiedźDFS(x)

Po zakończeniu powyższego algorytmu zmienna $Numer$ zawiera liczbę spójnych składowych grafu. Dla każdego wierzchołka grafu u, w komórce tablicy $Skladowa[u]$ znajduje się numer spójnej składowej, do której należy u.

Przeszukiwanie wszerz

Podczas przeszukiwania wszerz nie zawsze odwiedzane są wszystkie wierzchołki grafu. Odwiedzamy jedynie wierzchołki *osiągalne* z pewnego ustalonego wierzchołka s, tzn. takie wierzchołki, do których istnieje ścieżka z wierzchołka s.

W przypadku grafu nieskierowanego oznacza to po prostu, że odwiedzamy wszystkie wierzchołki w spójnej składowej zawierającej s.

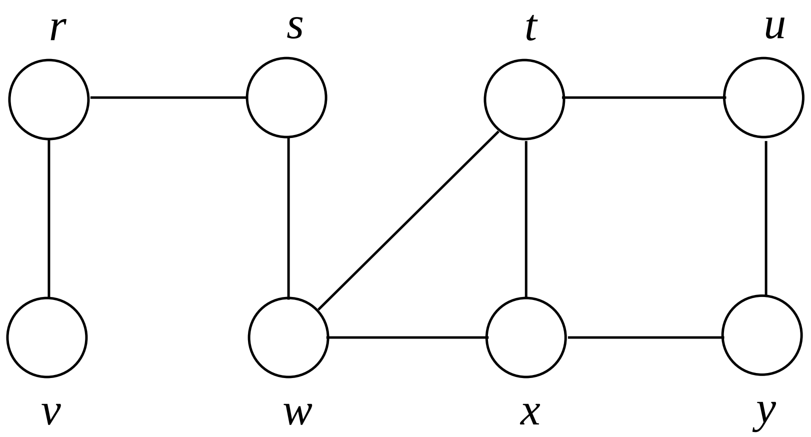
Idea algorytmu: najpierw odwiedzamy wierzchołki odległe o 1 od s, potem odległe o 2, 3, 4, itd...

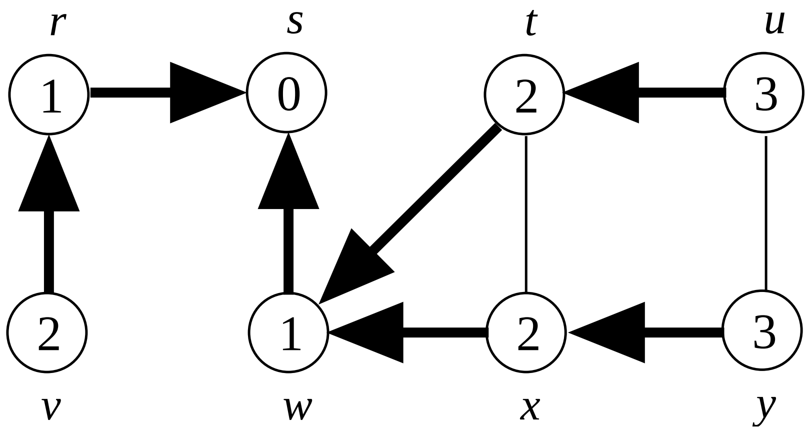
Natychmiastowe zastosowanie przeszukiwania wszerz: Dla każdego wierzchołka x w grafie znajdziemy odległość od s do x; jej wartość zapiszemy w tablicy $d[x]$. W tablicy $Skad[x]$ zapiszemy numer wierzchołka, z którego weszliśmy do x.

Przeszukiwanie wszerz

```
for all v ∈ V − {s} do
    Odwiedzony[v] ← False
    d[v] ← +∞
    Skad[v] ← nil
Odwiedzony[s] ← True
d[s] ← 0
Skad[s] ← nil
Q.Utwórz {tworzy pustą kolejkę FIFO Q}
Q.Dodaj(s)
while not Q.Pusta do
    u ← Q.Usuń
    for all x ∈ Sasiedzi[u] do
        if not Odwiedzony[x] then
            Odwiedzony[x] ← True
            d[x] ← d[u] + 1
            Skad[x] ← u
            Q.Dodaj(x)
```

Drzewo najkrótszych ścieżek





Strzałki odpowiadają zawartości tablicy $Skad$. Widzimy, że tworzą one drzewo o korzeniu w s. Nazywamy je *drzewem najkrótszych ścieżek*. Z takiego drzewa morzemy łatwo odczytać najkrótsze ścieżki od s do innych wierzchołków.

Problem najkrótszej ścieżki

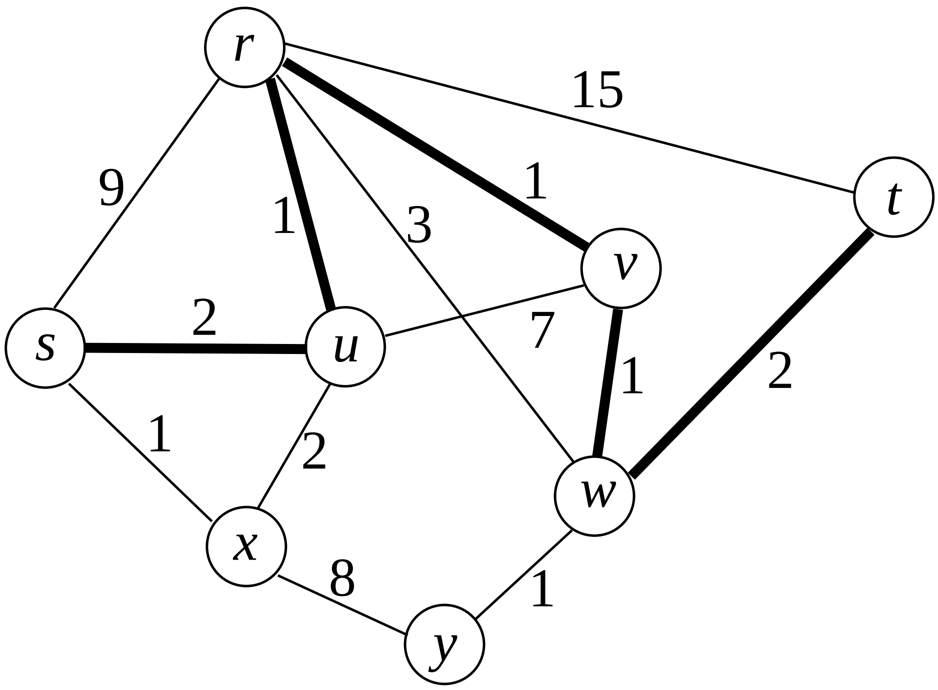
Dane:

- $G = (V, E)$ graf nieskierowany,
- funkcja $w : E \to [0, \infty)$, tzn. dla każdej krawędzi e mamy jej „długość”, czyli nieujemną liczbę rzeczywistą $w(e)$,
- wyróżniony wierzchołek s.

W takim grafie za długość ścieżki przyjmiemy sumę długości jej krawędzi. Zmienia wtedy swój sens także pojęcie odległości dwóch wierzchołków (długości najkrótszej ścieżki łączącej wierzchołki). Taką „nową” odległość będziemy oznaczać przez $d_w(u, v)$.

Problem: Dla każdego $v \in V$ obliczyć $d_w(s, v)$. Obliczyć drzewo najkrótszych ścieżek (w sensie odległości d_w).

przykład: najkrótsza ścieżka



Najkrótsza ścieżka od s do t (złożona z pogrubionych krawędzi) ma długość 7.

Algorytm Dijkstry

W każdym momencie zbiór V podzielony jest na dwa zbiory rozłączne: $V = L \cup P$.

L – wierzchołki, dla których już znamy odległość od s

P – pozostałe wierzchołki.

Dodatkowo wierzchołki L leżą bliżej s niż wierzchołki z P, tzn. jeśli $l \in L$ i $p \in P$ to $d_w(s,l) \leq d_w(s,p)$. Co więcej, dla każdego wierzchołka z L najkrótsza ścieżka z s przechodzi *tylko* po wierzchołkach z L.

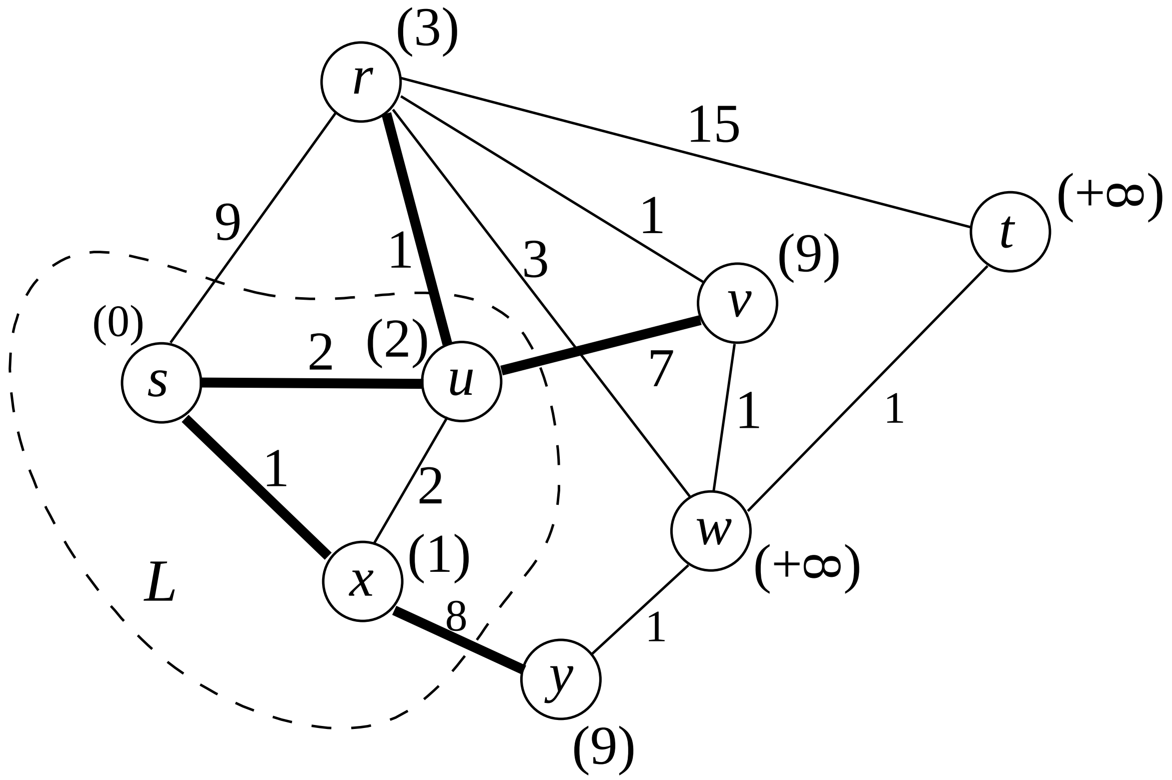
Na początku $L = \{s\}$, $P = V - \{s\}$.

Algorytm w każdym kroku przenosi jeden wierzchołek z P do L.

Algorytm Dijkstry, cd.

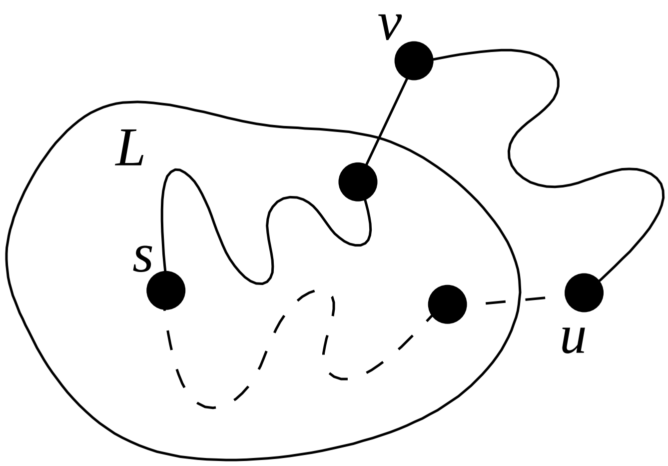
W tablicy $LOdl[v]$ będziemy dla każdego wierzchołka v przechowywać długość najkrótszej ścieżki z s do v przechodzącej *tylko* po wierzchołkach L, lub $+\infty$ jeśli takiej drogi nie ma.

Problem: Jak znaleźć w P wierzchołek do przeniesienia?



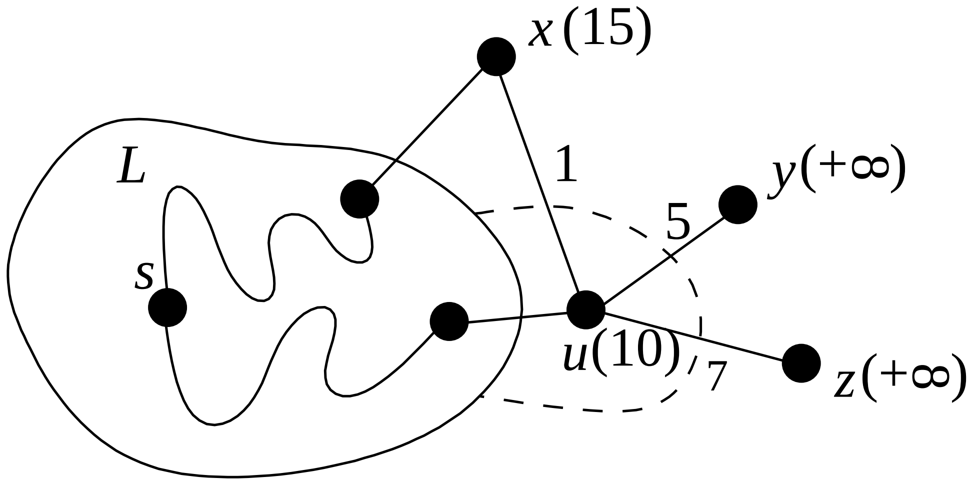
Algorytm Dijkstry, cd.

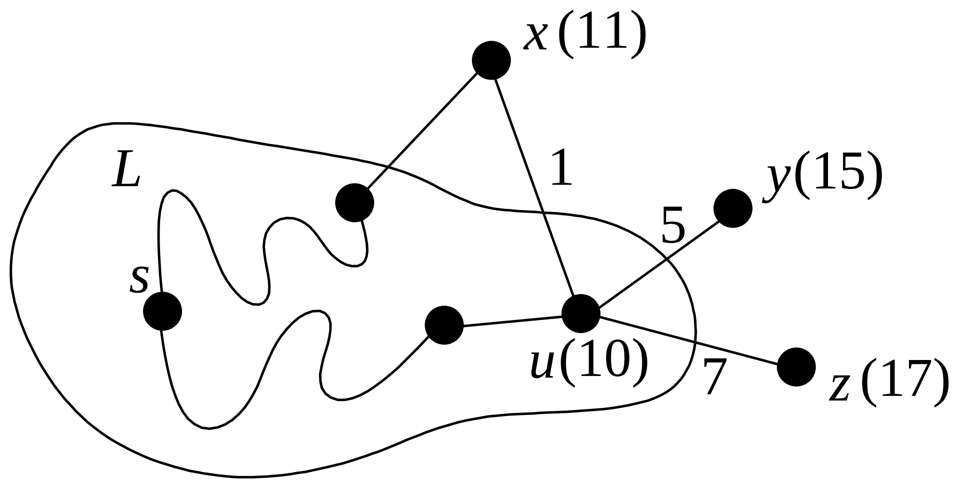
Wybieramy z P taki wierzchołek u, dla którego wartość $LOdl[u]$ jest najmniejsza. Wtedy $LOdl[u] = d_w(s, u)$ (dlaczego?). Dla takiego wierzchołka znamy już poprawną odległość od s i możemy go przenieść do L.



Algorytm Dijkstry, cd.

Uwaga! Po przeniesieniu u z P do L musimy być może odświeżyć niektóre wartośc w tablicy $LOdl$. Zmienić mogły się tylko wartości $LOdl$ dla sąsiadów u.





Algorytm Dijkstry, cd.

$L \leftarrow \{s\}$; $P \leftarrow V - \{s\}$
for all $u \in P$ **do**
    if $s\text{–}u \in E$ **then**
        $LOdl[u] \leftarrow w(s\text{–}u)$
    else
        $LOdl[u] \leftarrow +\infty$
for $i \leftarrow 2$ **to** n **do**
    $u \leftarrow$ wierzchołek z P t.ż. $LOdl[u]$ jest najmniejsze
    $P \leftarrow P - \{u\}$
    $L \leftarrow L \cup \{u\}$
    for all $x \in Sasiedzi[u]$ **do**
        if $x \in P$ **and** $LOdl[u] + w(x\text{–}u) < LOdl[x]$ **then**
            $LOdl[x] \leftarrow LOdl[u] + w(x\text{–}u)$

Algorytm Dijkstry

Modyfikacja: drzewo najkrótszych ścieżek.

$L \leftarrow \{s\}$; $P \leftarrow V - \{s\}$
for all $u \in P$ **do**
    if s–$u \in E$ **then**
        $LOdl[u] \leftarrow w(s\text{–}u)$
        $Skad[u] \leftarrow s$
    else
        $LOdl[u] \leftarrow +\infty$
        $Skad[u] \leftarrow nil$
for $i \leftarrow 2$ **to** n **do**
    $u \leftarrow$ wierzchołek z P t.ż. $LOdl[u]$ jest najmniejsze
    $P \leftarrow P - \{u\}$
    $L \leftarrow L \cup \{u\}$
    for all $x \in Sasiedzi[u]$ **do**
        if $x \in P$ **and** $LOdl[u] + w(x\text{–}u) < LOdl[x]$ **then**
            $LOdl[x] \leftarrow LOdl[u] + w(x\text{–}u)$
            $Skad[x] \leftarrow u$