Podstawowe algorytmy grafowe

Piotr Kasprowicz

Politechnika Wrocławska
Wydział Podstawowych Problemów Techniki

Seminarium, Listopad 2019

Podstawowe pojęcia

Graf

Graf (ang. graph) jest strukturą danych składającą się z dwóch zbiorów: zbioru wierzchołków V (ang. vertices) i zbioru krawędzi E (ang. edges), co matematycznie zapisujemy w postaci uporządkowanej pary $G = (V, E)$.

Podstawowe pojęcia

Graf

Graf (ang. graph) jest strukturą danych składającą się z dwóch zbiorów: zbioru wierzchołków V (ang. vertices) i zbioru krawędzi E (ang. edges), co matematycznie zapisujemy w postaci uporządkowanej pary $G = (V, E)$.

Graf spójny

Graf spełniający warunek, że dla każdej pary wierzchołków istnieje łącząca je ścieżka.

Podstawowe pojęcia

Graf

Graf (ang. graph) jest strukturą danych składającą się z dwóch zbiorów: zbioru wierzchołków V (ang. vertices) i zbioru krawędzi E (ang. edges), co matematycznie zapisujemy w postaci uporządkowanej pary $G = (V, E)$.

Graf spójny

Graf spełniający warunek, że dla każdej pary wierzchołków istnieje łącząca je ścieżka.

Minimalne drzewo rozpinające

Drzewo rozpinające danego grafu o najmniejszej z możliwych wag, tj. takie, że nie istnieje dla tego grafu inne drzewo rozpinające o mniejszej sumie wag krawędzi.

Reprezentacje grafów

Lista sądziedztwa

- grafy rzadkie
- złożonośc pamięciowa $\Theta(V + E)$
- prosta modyfikacja przy grafie z wagami
- brak możliwości szybkiego sprawdzenia czy istnieje krawędź łącząca dwa wierzchołki

Reprezentacje grafów

Lista sądziedztwa

- grafy rzadkie
- złożonośc pamięciowa $\Theta(V + E)$
- prosta modyfikacja przy grafie z wagami
- brak możliwości szybkiego sprawdzenia czy istnieje krawędź łącząca dwa wierzchołki

Macierz sądziedztwa

- grafy gęste
- złożonośc pamięciowa $\Theta(V^2)$
- istnieje możliwość szybkiego sprawdzenia czy istnieje krawędź łącząca dwa wierzchołki

Przeszukiwanie grafu

Przeszukiwanie grafu lub inaczej przechodzenie grafu oznacza odwiedzenie każdego wierzchołka dokładnie raz w pewnym określonym porządku.

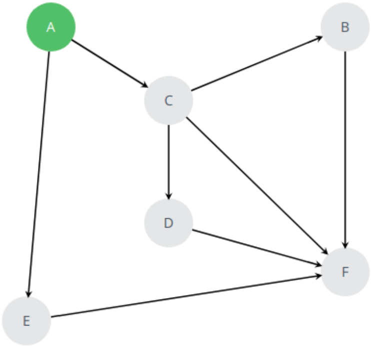
Przeszukiwanie wszerz

Algorytm

let Q be a queue;
label start v as visited;
Q.enqueue(start v);
while *Q is not empty* **do**
    v = Q.dequeue();
    for *all edges from v to w in G.adjacentEdges(v)* **do**
        if *w is not labeled as visted* **then**
            label w as visited;
            Q.enqueue(w)
        end
    end
end

Przeszukiwanie wszerz - przykład

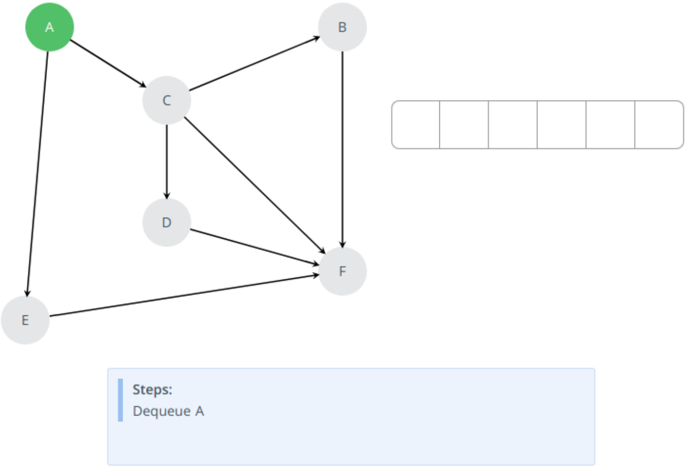
Order: A,



Steps:
Mark and enqueue A

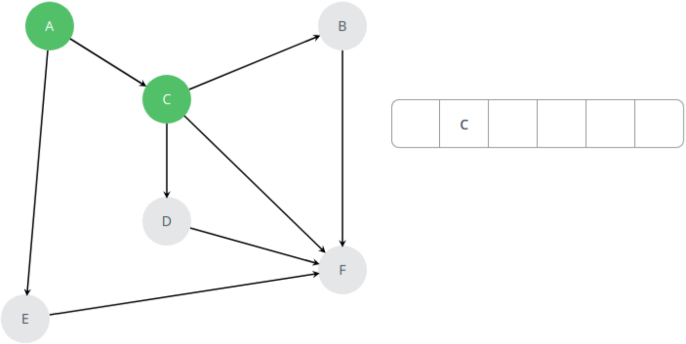
Przeszukiwanie wszerz - przykład

Order: A,



Przeszukiwanie wszerz - przykład

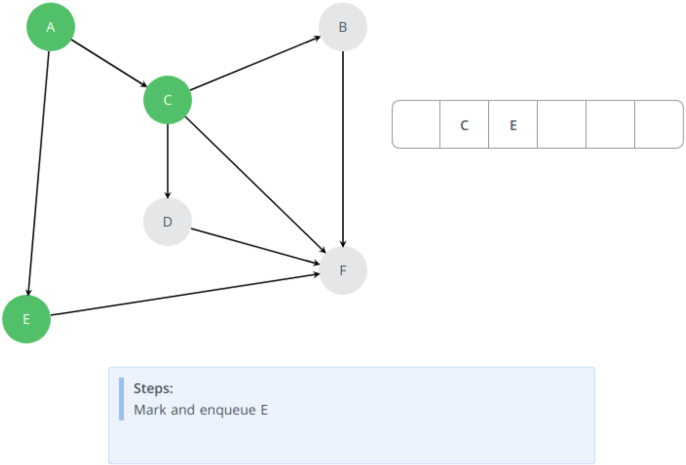
Order: A, C,



Steps:
Mark and enqueue C

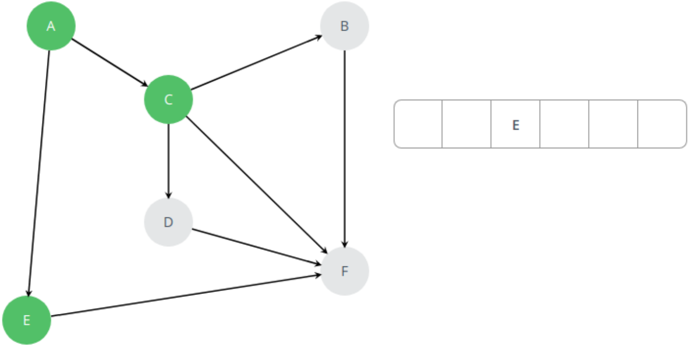
Przeszukiwanie wszerz - przykład

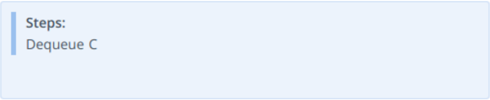
Order: A, C, E,



Przeszukiwanie wszerz - przykład

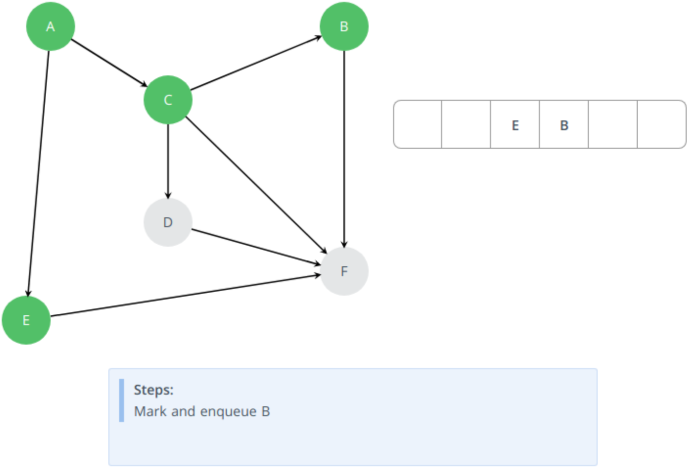
Order: A, C, E,





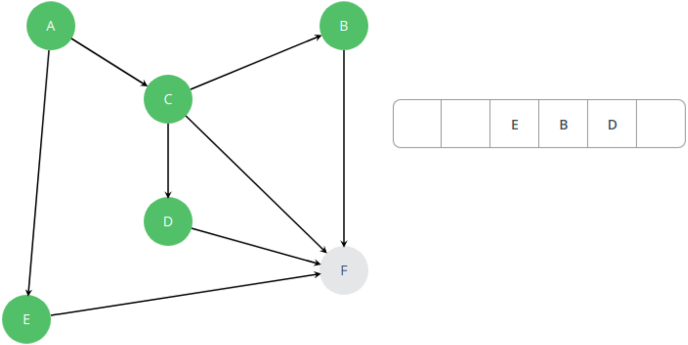
Przeszukiwanie wszerz - przykład

Order: A, C, E, B,



Przeszukiwanie wszerz - przykład

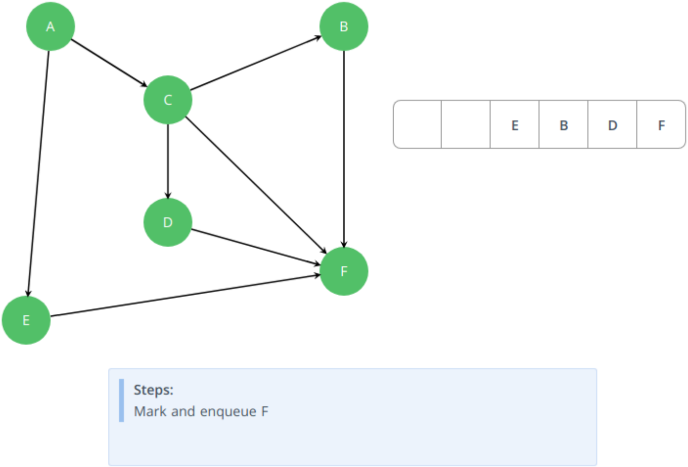
Order: A, C, E, B, D,



Steps:
Mark and enqueue D

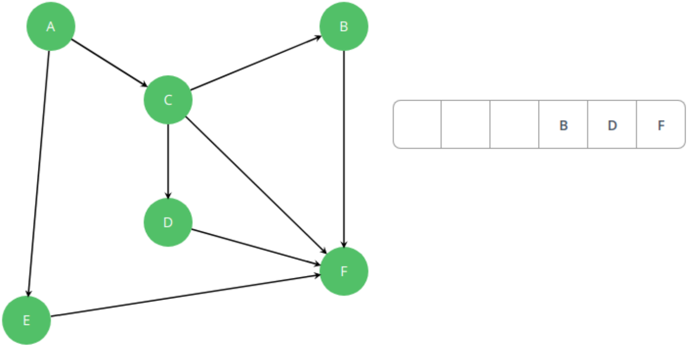
Przeszukiwanie wszerz - przykład

Order: A, C, E, B, D, F



Przeszukiwanie wszerz - przykład

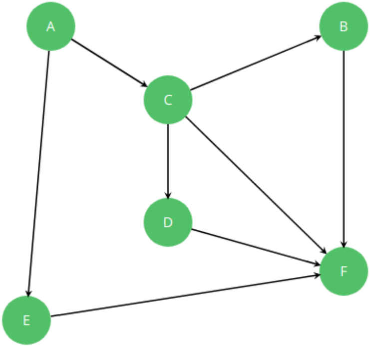
Order: A, C, E, B, D, F



Steps:
Dequeue E

Przeszukiwanie wszerz - przykład

Order: A, C, E, B, D, F

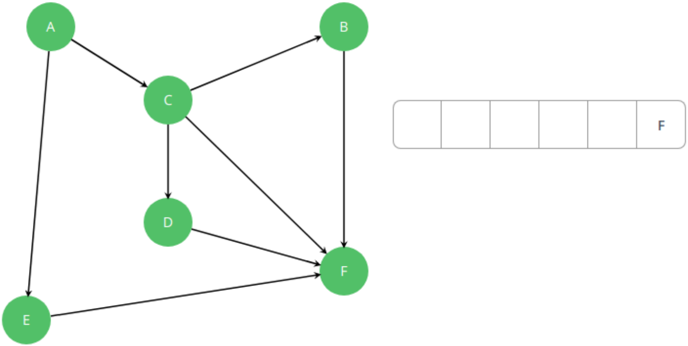


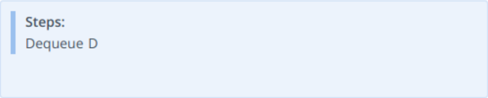
				D	F

Steps:
Dequeue B

Przeszukiwanie wszerz - przykład

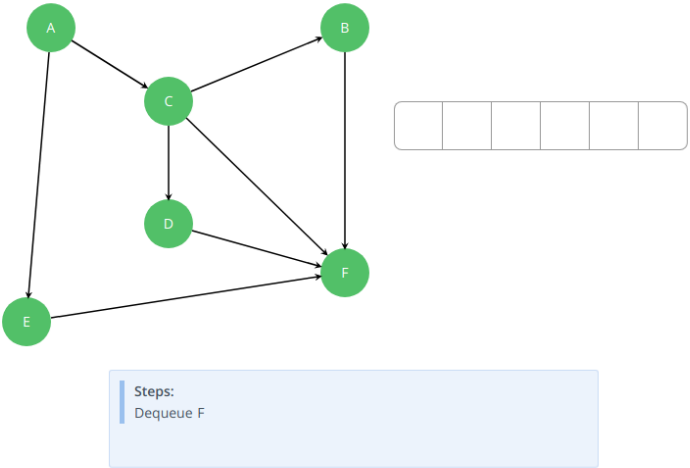
Order: A, C, E, B, D, F



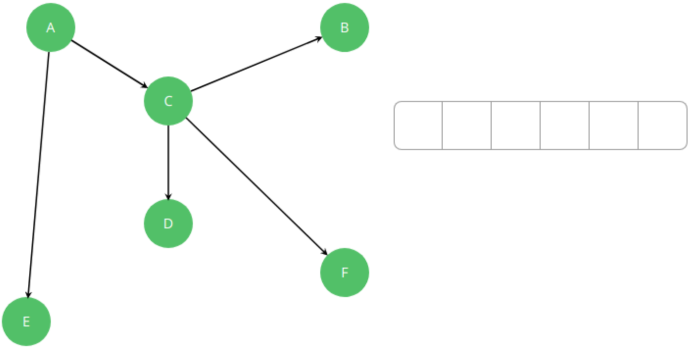


Przeszukiwanie wszerz - przykład

Order: A, C, E, B, D, F



Order: A, C, E, B, D, F



Steps:
Completed breadth first search graph

Przeszukiwanie wszerz

Algorytm stosowany jest:

- do wyznaczania silnych spójnych składowych grafu skierowanego

Przeszukiwanie wszerz

Algorytm stosowany jest:

- do wyznaczania silnych spójnych składowych grafu skierowanego
- w algorytmie sortowania topologicznego skierowanego grafu acyklicznego

Przeszukiwanie wszerz

Algorytm stosowany jest:

- do wyznaczania silnych spójnych składowych grafu skierowanego
- w algorytmie sortowania topologicznego skierowanego grafu acyklicznego
- do testowania dwudzielności grafu

Przeszukiwanie w głąb - rekurencyjnie

DFS-recursive

mark s as visited;
for *all neighbours w of s in Graph G* **do**
    if *w is not visited* **then**
        DFS-recursive(G, w);
    end
end

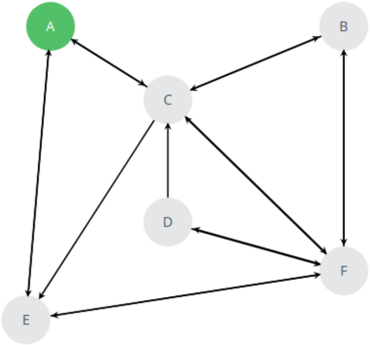
Przeszukiwanie w głąb - iteracyjnie

DFS-iterative

let S be a stack;
S.push(s);
mark s as visited;
while *S is not empty* **do**
    v = S.pop();
    label v as visited;
    for *all edges from v to w in G.adjacentEdges(v)* **do**
        if *v is not labeled as visted* **then**
            S.push(w);
            mark w as visited;
        end
    end
end

Przeszukiwanie w głąb - przykład

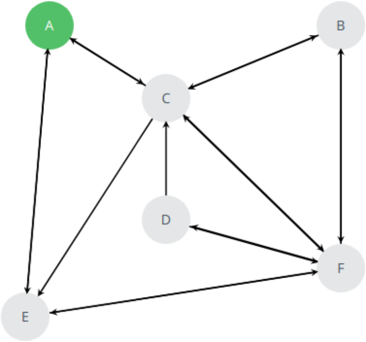
Order: A,



Steps:
Mark node A

Przeszukiwanie w głąb - przykład

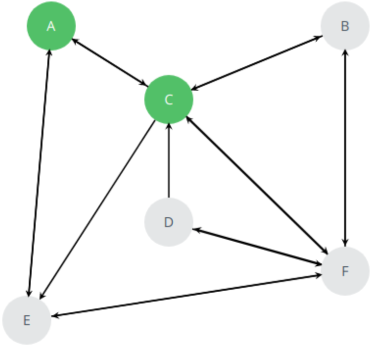
Order: A,



Steps:
Add A to the stack

Przeszukiwanie w głąb - przykład

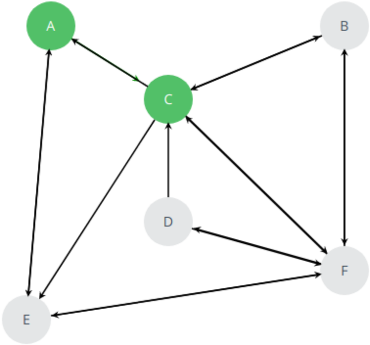
Order: A, C,



Steps:
Mark node C

Przeszukiwanie w głąb - przykład

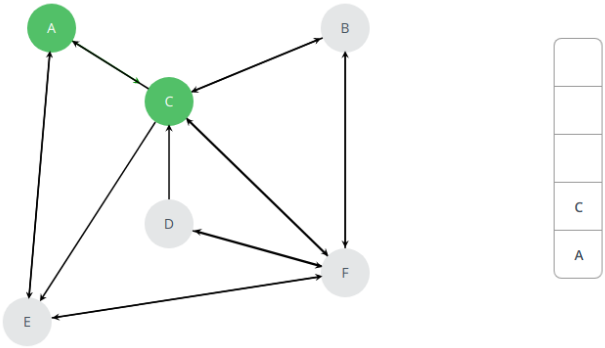
Order: A, C,

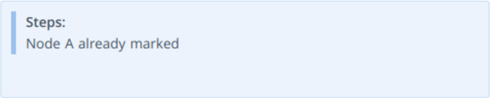


Steps:
Add C to the stack

Przeszukiwanie w głąb - przykład

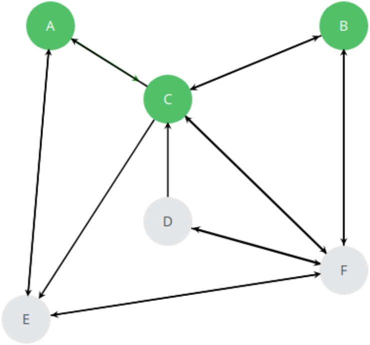
Order: A, C,





Przeszukiwanie w głąb - przykład

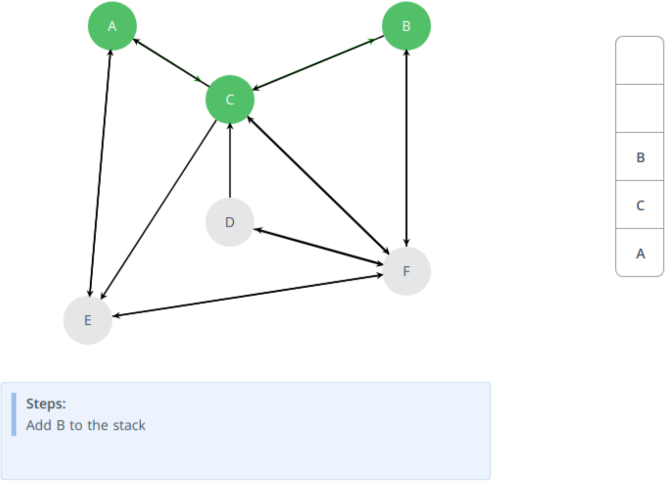
Order: A, C, B,



Steps:
Mark node B

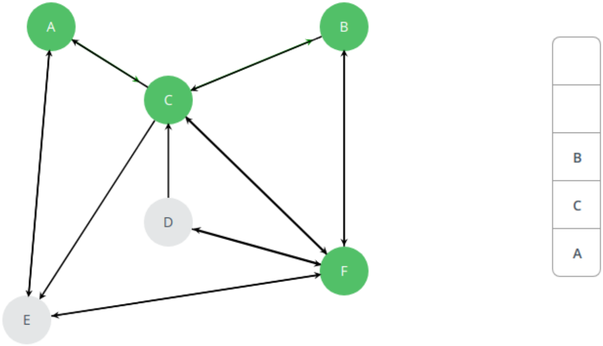
Przeszukiwanie w głąb - przykład

Order: A, C, B,



Przeszukiwanie w głąb - przykład

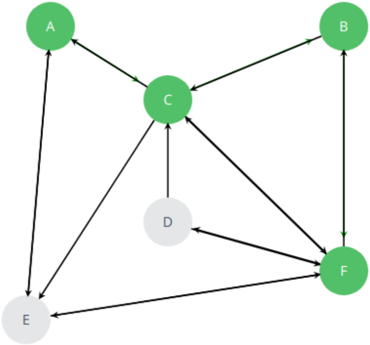
Order: A, C, B, F,



Steps:
Mark node F

Przeszukiwanie w głąb - przykład

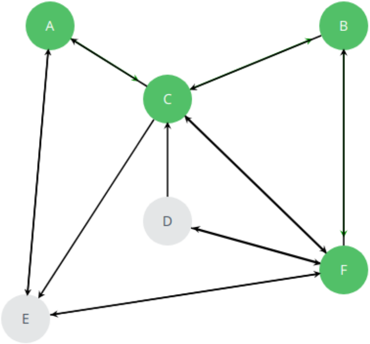
Order: A, C, B, F,



Steps:
Add F to the stack

Przeszukiwanie w głąb - przykład

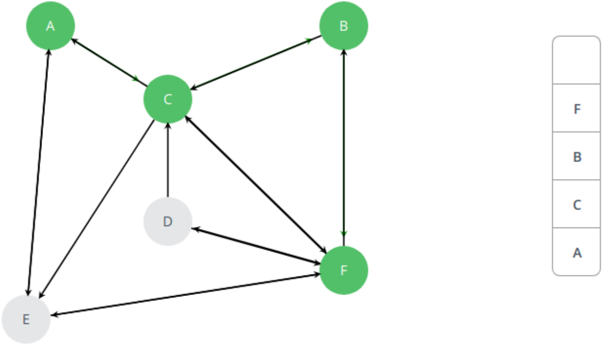
Order: A, C, B, F,

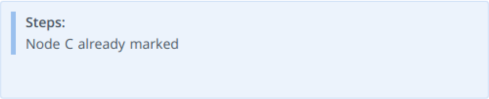


Steps:
Node B already marked

Przeszukiwanie w głąb - przykład

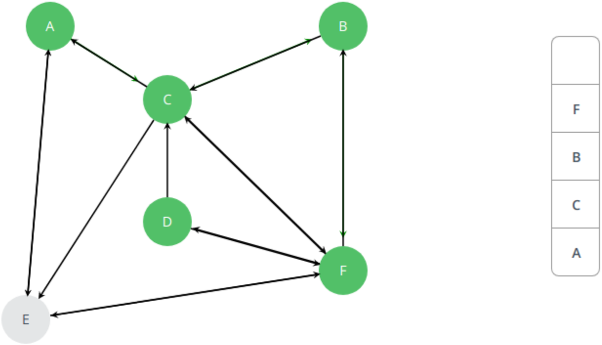
Order: A, C, B, F,

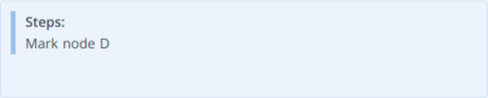




Przeszukiwanie w głąb - przykład

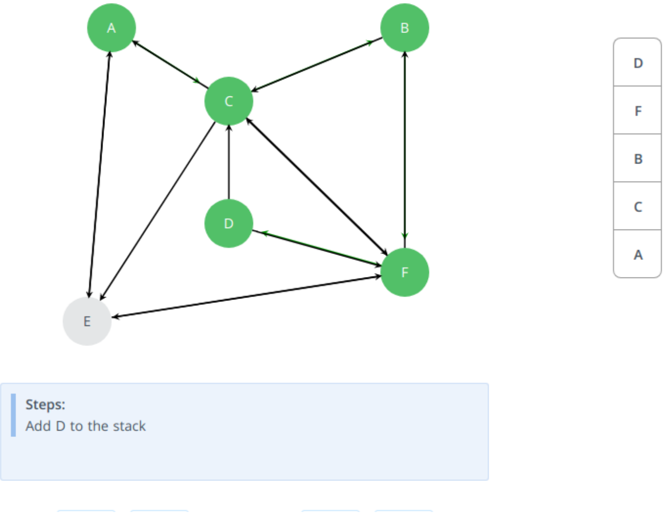
Order: A, C, B, F, D,





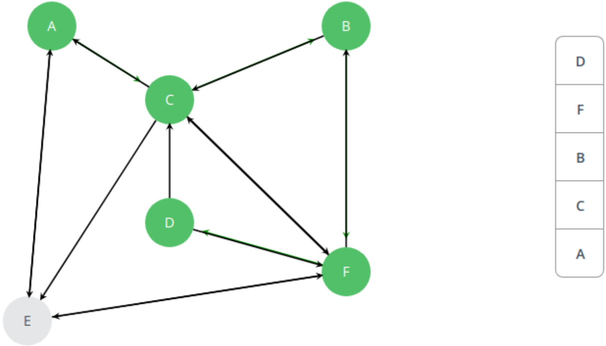
Przeszukiwanie w głąb - przykład

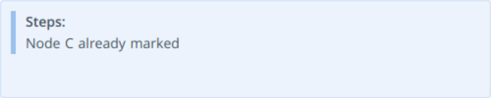
Order: A, C, B, F, D,



Przeszukiwanie w głąb - przykład

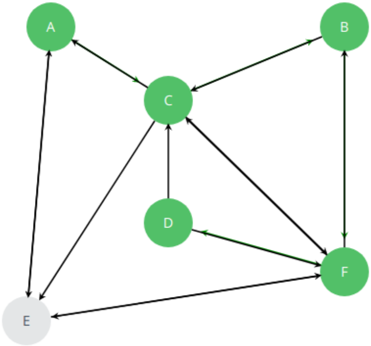
Order: A, C, B, F, D,





Przeszukiwanie w głąb - przykład

Order: A, C, B, F, D,

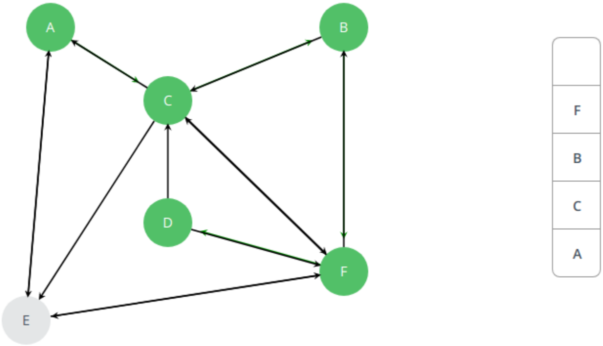


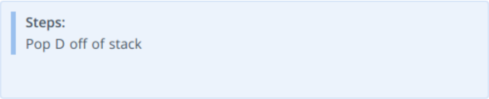
Steps:

Node F already marked

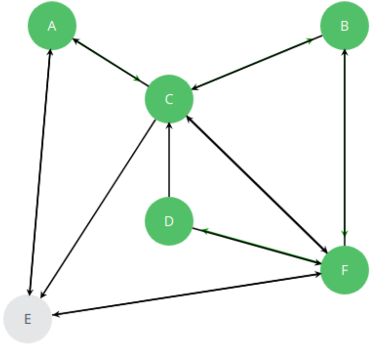
Przeszukiwanie w głąb - przykład

Order: A, C, B, F, D,





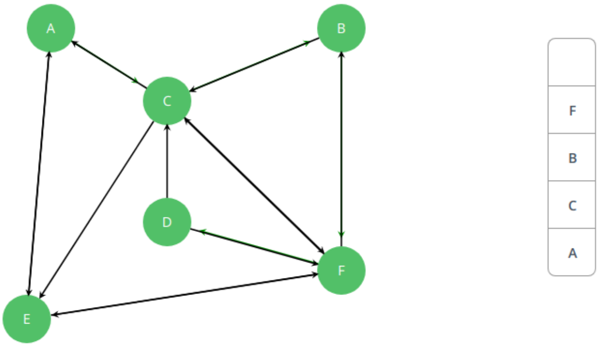
Order: A, C, B, F, D,

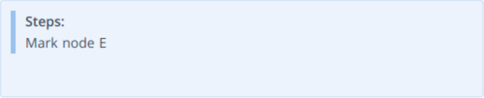


Steps:
Print (F,E) and call depth first search on E

Przeszukiwanie w głąb - przykład

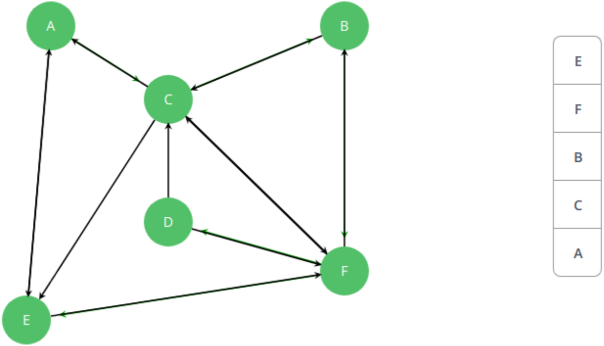
Order: A, C, B, F, D, E





Przeszukiwanie w głąb - przykład

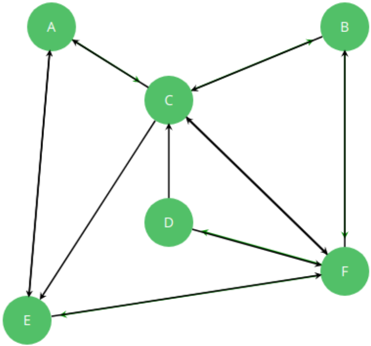
Order: A, C, B, F, D, E



Steps:
Add E to the stack

Przeszukiwanie w głąb - przykład

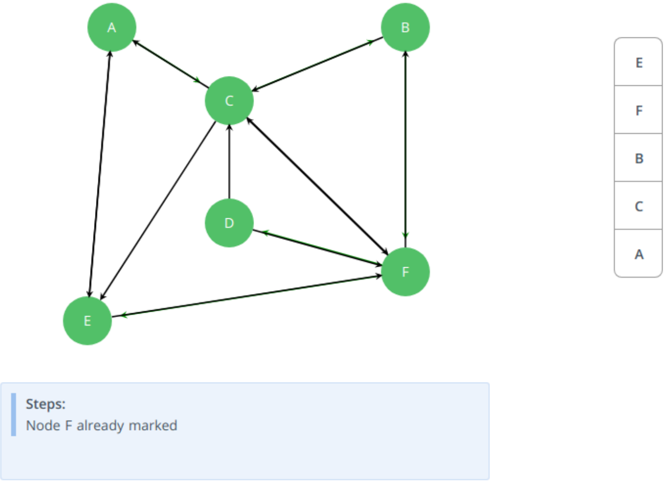
Order: A, C, B, F, D, E



Steps:
Node A already marked

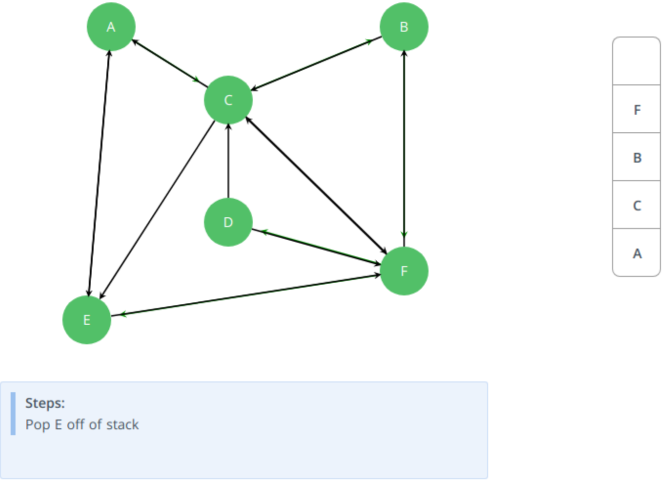
Przeszukiwanie w głąb - przykład

Order: A, C, B, F, D, E



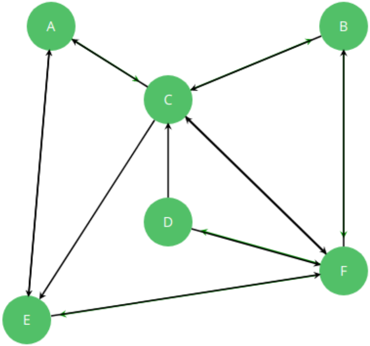
Przeszukiwanie w głąb - przykład

Order: A, C, B, F, D, E



Przeszukiwanie w głąb - przykład

Order: A, C, B, F, D, E

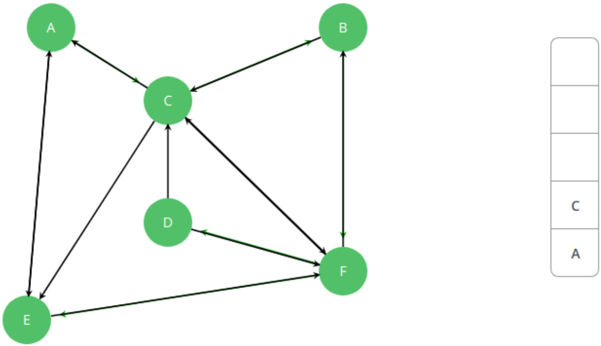


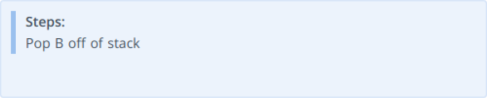
Steps:

Pop F off of stack

Przeszukiwanie w głąb - przykład

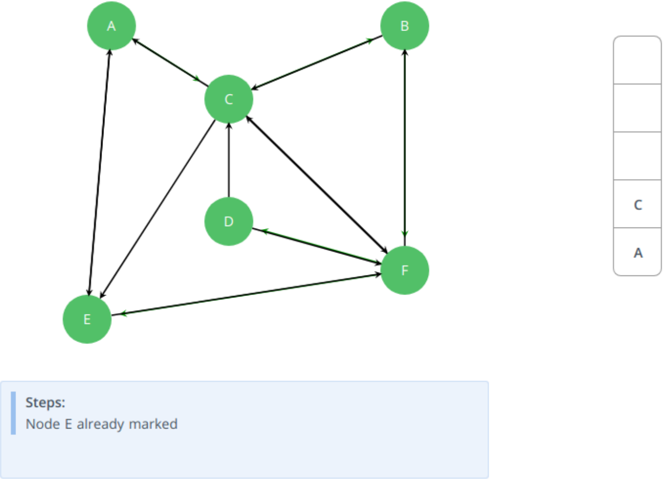
Order: A, C, B, F, D, E



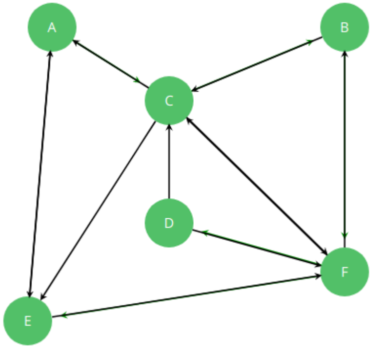


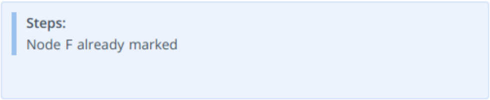
Przeszukiwanie w głąb - przykład

Order: A, C, B, F, D, E



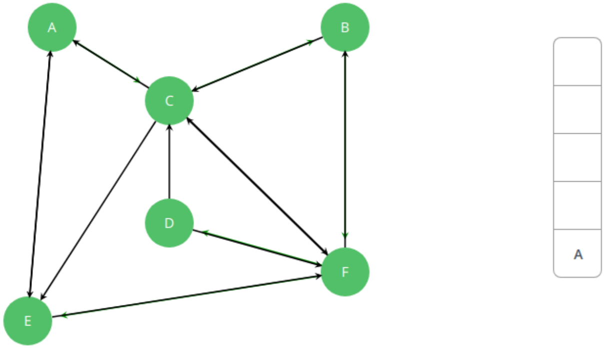
Order: A, C, B, F, D, E

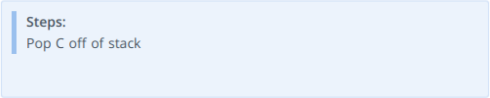




Przeszukiwanie w głąb - przykład

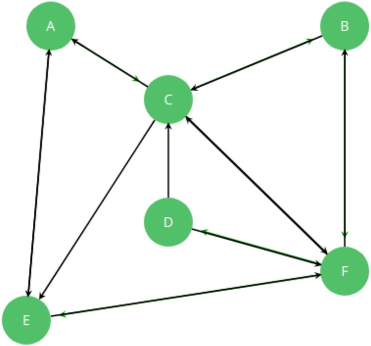
Order: A, C, B, F, D, E





Przeszukiwanie w głąb - przykład

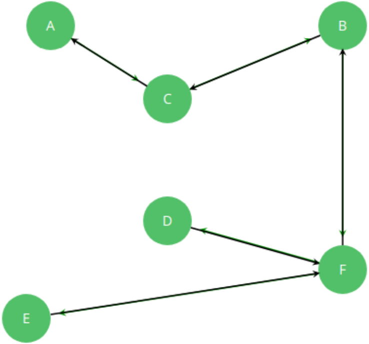
Order: A, C, B, F, D, E



Steps:
Pop A off of stack

Przeszukiwanie w głąb - przykład

Order: A, C, B, F, D, E



Steps:
Completed depth first search graph

Problem najkrótszej ścieżki

Algorytm Dijkstry

- Stwórz tablicę d odległości od źródła dla wszystkich wierzchołków grafu. Na początku $d[s] = 0$, zaś $d[v] = \infty$ dla wszystkich pozostałych wierzchołków.

Problem najkrótszej ścieżki

Algorytm Dijkstry

- Stwórz tablicę d odległości od źródła dla wszystkich wierzchołków grafu. Na początku $d[s] = 0$, zaś $d[v] = \infty$ dla wszystkich pozostałych wierzchołków.
- Utwórz kolejkę priorytetową Q wszystkich wierzchołków grafu. Priorytetem kolejki jest aktualnie wyliczona odległość od wierzchołka źródłowego s.

Problem najkrótszej ścieżki

Algorytm Dijkstry

- Stwórz tablicę d odległości od źródła dla wszystkich wierzchołków grafu. Na początku $d[s] = 0$, zaś $d[v] = \infty$ dla wszystkich pozostałych wierzchołków.
- Utwórz kolejkę priorytetową Q wszystkich wierzchołków grafu. Priorytetem kolejki jest aktualnie wyliczona odległość od wierzchołka źródłowego s.
- Dopóki kolejka nie jest pusta:
  - Usuń z kolejki wierzchołek u 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 $d[v] := d[u] + w(u, v)$.

Algorytm Dijkstry

Jeśli graf nie jest ważony (wszystkie wagi mają wielkość 1), zamiast algorytmu Dijkstry wystarczy algorytm przeszukiwania grafu wszerz.

Algorytm Dijkstry

Jeśli graf nie jest ważony (wszystkie wagi mają wielkość 1), zamiast algorytmu Dijkstry wystarczy algorytm przeszukiwania grafu wszerz.
Złożność obliczeniowa:

Algorytm Dijkstry

Jeśli graf nie jest ważony (wszystkie wagi mają wielkość 1), zamiast algorytmu Dijkstry wystarczy algorytm przeszukiwania grafu wszerz.
Złożność obliczeniowa:

- zwykła tablica $O(V^2)$

Algorytm Dijkstry

Jeśli graf nie jest ważony (wszystkie wagi mają wielkość 1), zamiast algorytmu Dijkstry wystarczy algorytm przeszukiwania grafu wszerz.
Złożoność obliczeniowa:

- zwykła tablica $O(V^2)$
- kolejka priorytetowa $O(ElogV)$

Algorytm Dijkstry

Jeśli graf nie jest ważony (wszystkie wagi mają wielkość 1), zamiast algorytmu Dijkstry wystarczy algorytm przeszukiwania grafu wszerz.
Złożoność obliczeniowa:

- zwykła tablica $O(V^2)$
- kolejka priorytetowa $O(ElogV)$
- kopiec fibonacciego $O(E + VlogV)$

Algorytm Dijkstry

Jeśli graf nie jest ważony (wszystkie wagi mają wielkość 1), zamiast algorytmu Dijkstry wystarczy algorytm przeszukiwania grafu wszerz.
Złożoność obliczeniowa:

- zwykła tablica $O(V^2)$
- kolejka priorytetowa $O(ElogV)$
- kopiec fibonacciego $O(E + VlogV)$

Ujemne wagi

Przez fakt, że algorytm Dijkstry jest algorytmem zachłannym nie działa dla ujemnych wag. Rozwiązaniem jest algorytm Bellmana-Forda.

Algorytm Prima

Algorytm

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

Algorytm Prima

Algorytm

- Utwórz drzewo zawierające jeden wierzchołek, dowolnie wybrany z grafu.
- Utwórz kolejkę priorytetową, zawierającą wierzchołki osiągalne z MDR (w tym momencie zawiera jeden wierzchołek, więc na początku w kolejce będą sąsiedzi początkowego wierzchołka), o priorytecie najmniejszego kosztu dotarcia do danego wierzchołka z MDR.

Algorytm Prima

Algorytm

- Utwórz drzewo zawierające jeden wierzchołek, dowolnie wybrany z grafu.
- Utwórz kolejkę priorytetową, zawierającą wierzchołki osiągalne z MDR (w tym momencie zawiera jeden wierzchołek, więc na początku w kolejce będą sąsiedzi początkowego wierzchołka), o priorytecie najmniejszego kosztu dotarcia do danego wierzchołka z MDR.
- Powtarzaj, dopóki drzewo nie obejmuje wszystkich wierzchołków grafu:

Algorytm Prima

Algorytm

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

Algorytm Prima

Algorytm

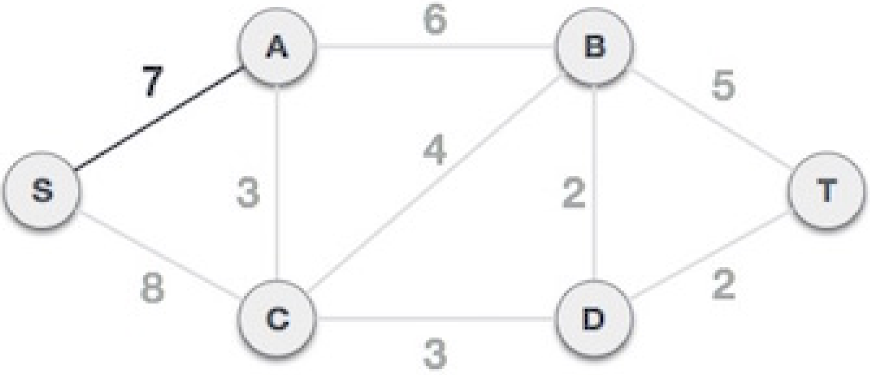
- Utwórz drzewo zawierające jeden wierzchołek, dowolnie wybrany z grafu.
- Utwórz kolejkę priorytetową, zawierającą wierzchołki osiągalne z MDR (w tym momencie zawiera jeden wierzchołek, więc na początku w kolejce będą sąsiedzi początkowego wierzchołka), o priorytecie najmniejszego kosztu dotarcia do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

Algorytm Prima

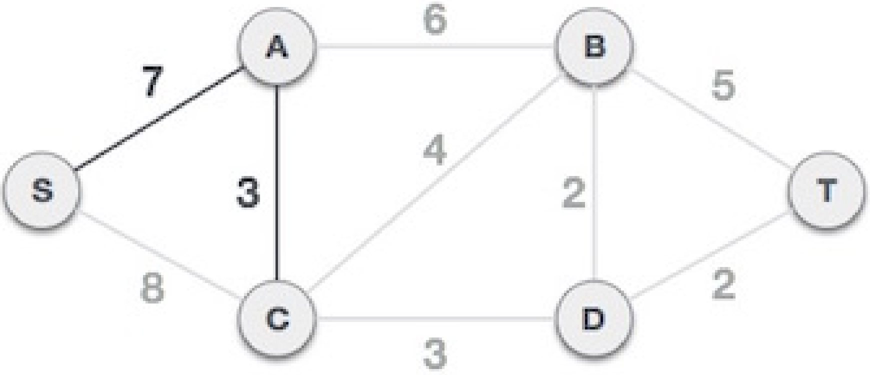
Algorytm

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

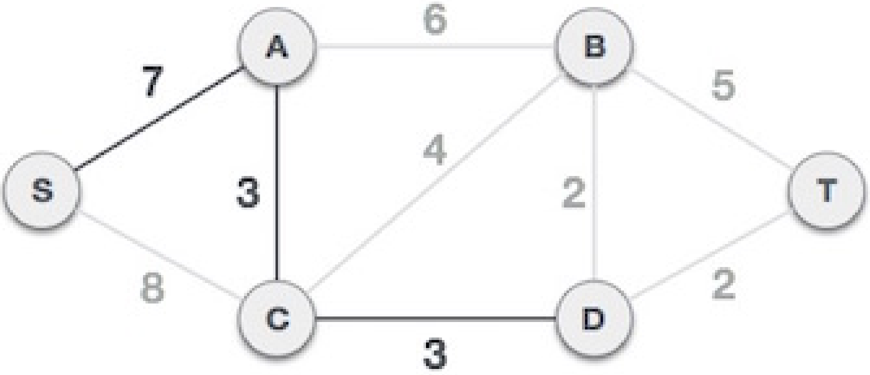
Algorytm Prima - przykład



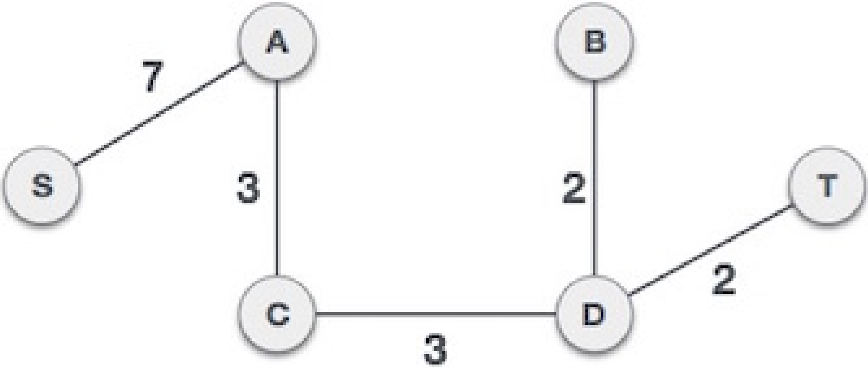
Algorytm Prima - przykład



Algorytm Prima - przykład



Algorytm Prima - przykład



Algorytm Kruskala

Algorytm

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

Algorytm Kruskala

Algorytm

- Utwórz las L z wierzchołków oryginalnego grafu – każdy wierzchołek jest na początku osobnym drzewem.
- Utwórz zbiór S zawierający wszystkie krawędzie oryginalnego grafu.

Algorytm Kruskala

Algorytm

- Utwórz las L z wierzchołków oryginalnego grafu – każdy wierzchołek jest na początku osobnym drzewem.
- Utwórz zbiór S zawierający wszystkie krawędzie oryginalnego grafu.
- Dopóki S nie jest pusty oraz L nie jest jeszcze drzewem rozpinającym:

Algorytm Kruskala

Algorytm

- Utwórz las L z wierzchołków oryginalnego grafu – każdy wierzchołek jest na początku osobnym drzewem.
- Utwórz zbiór S zawierający wszystkie krawędzie oryginalnego grafu.
- Dopóki S nie jest pusty oraz L nie jest jeszcze drzewem rozpinającym:
  - Wybierz i usuń z S jedną z krawędzi o minimalnej wadze.

Algorytm Kruskala

Algorytm

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

Algorytm Kruskala

Algorytm

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

Algorytm Kruskala

Algorytm

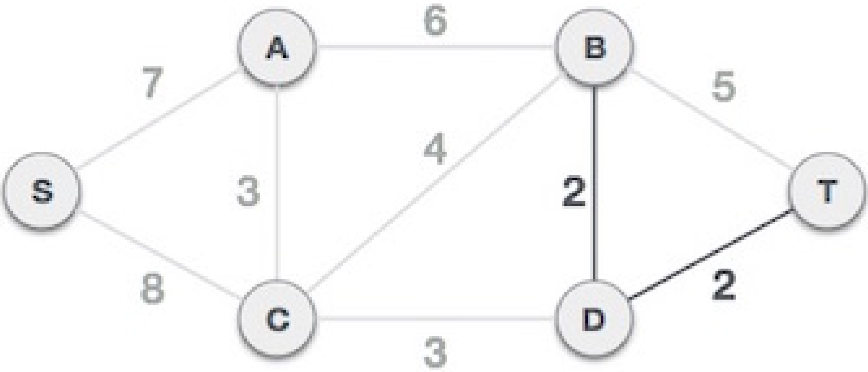
- Utwórz las L z wierzchołków oryginalnego grafu – każdy wierzchołek jest na początku osobnym drzewem.
- Utwórz zbiór S zawierający wszystkie krawędzie oryginalnego grafu.
- Dopóki S nie jest pusty oraz L nie jest jeszcze drzewem rozpinającym:
    - Wybierz i usuń z S jedną z krawędzi o minimalnej wadze.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

Złożoność czasowa

Zarówno alogorytm Prima i Kruskala posiada złożoność: $O(ElogV)$, zastosowanie kopca fibonacciego dla algorytmu prima redukuje złożoność do $O(E + VlogV)$, co jest opłacalne przy gęstych grafach.

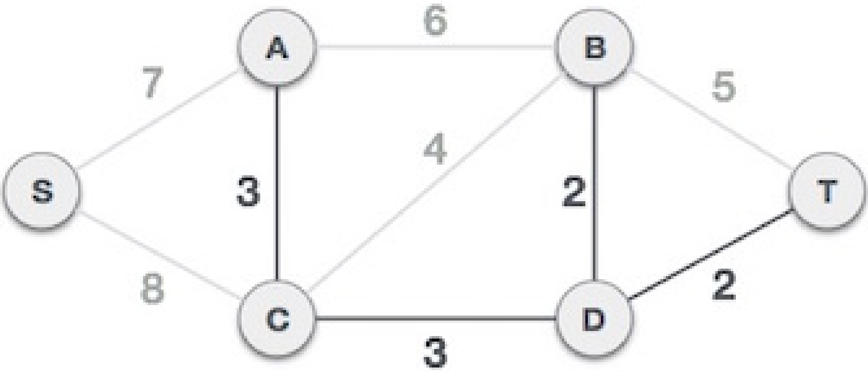
Algorytm Kruskala - przykład

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8



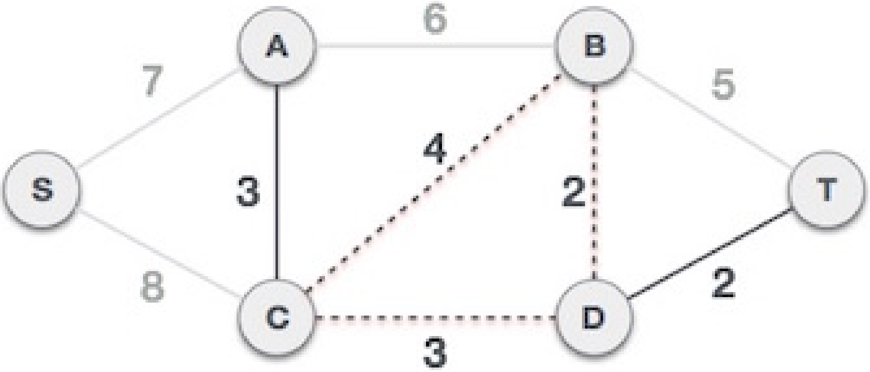
Algorytm Kruskala - przykład

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8



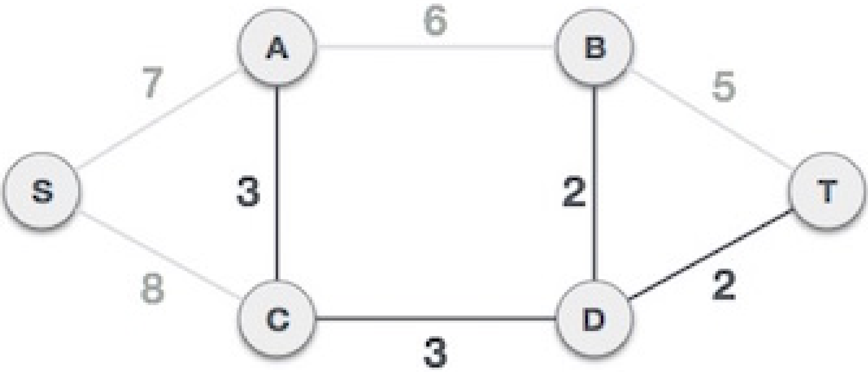
Algorytm Kruskala - przykład

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8



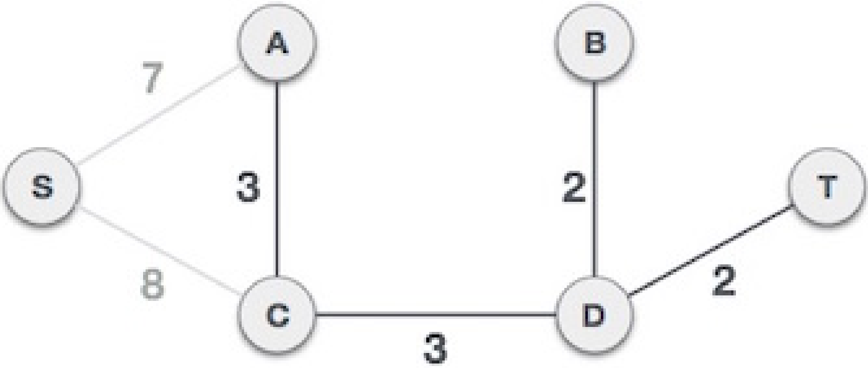
Algorytm Kruskala - przykład

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8



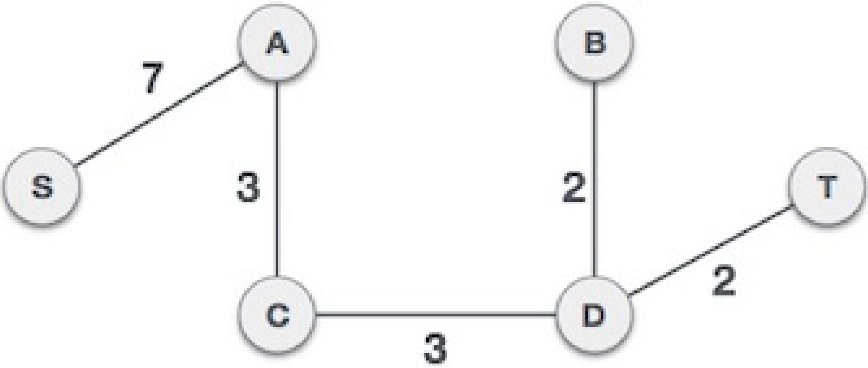
Algorytm Kruskala - przykład

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8



Algorytm Kruskala - przykład

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8



Sieci przepływowe

Sieć przepływowa jest spójnym grafem skierowanym $G = (V, E)$, w którego krawędziach odbywa się **przepływ** jakiegoś czynnika (wody, gazu, informacji itp.). Przepływ jest możliwy tylko w kierunku zwrotu krawędzi grafu. W sieci przepływowej wyróżnia się jeden wierzchołek s, z którego wychodzą przepływy - jest to tzw. **źródło** , oraz jeden wierzchołek t, do którego zbiegają się przepływy - jest to tzw. **ujście**.

Sieć rezydualna składa się z krawędzi, którymi można jeszcze przesłać przepływ.

Przepustowość oznacza maksymalną ilość czynnika mogącego przez krawędź przeływać.

Przepustowość rezydualna oznacza ile przez dany kanał już czynnika płynie.

Maksymalny przepływ w sieci

W problemie tym musimy określić maksymalną wielkość przepływu ze źródła do ujścia sieci przy ograniczeniach przepustowości nałożonych na poszczególne kanały.

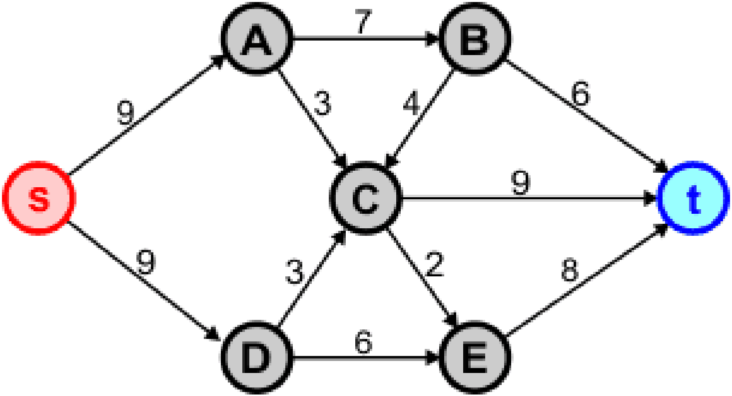
Algorytm Forda-Fulkersona

- Wyzeruj wszystkie przepływy w sieci.
- Dopóki w sieci rezydualnej istnieje ścieżka rozszerzająca p, zwiększaj przepływ o $c_f(p)$ wzdłuż kanałów zgodnych z kierunkiem ścieżki, a zmniejszaj przepływ wzdłuż kanałów przeciwnych (wygaszanie przepływu). Przepływ sieciowy rośnie o $c_f(p)$.

Algorytm Edmondsa-Karpa

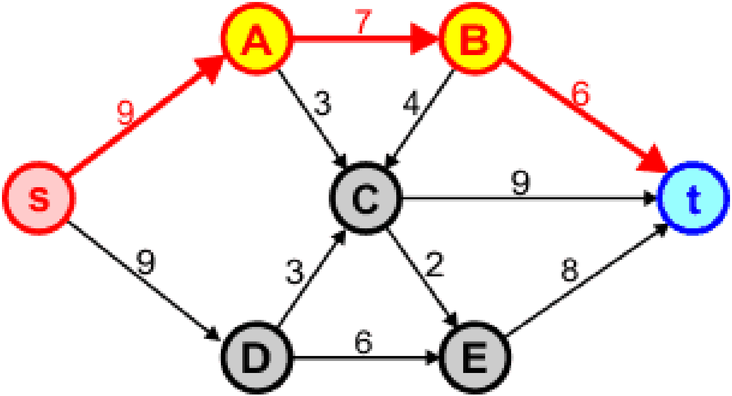
To algorytm Forda-Fulkersona z zastosowaniem BFS, który jest lepszy od wersji z DFS, ponieważ preferowane ścieżki krótkie, czyli zawierające jak najmniejszą liczbę krawędzi grafu sieci przepływowej.

Algorytm Forda-Fulkersona - przykład



Oto nasza sieć przepływowa. W kanałach zaznaczyliśmy ich przepustowości. Dla zerowych przepływów sieć rezydualna jest identyczna z siecią pierwotną. Przepływ sieci $|f| = 0$.

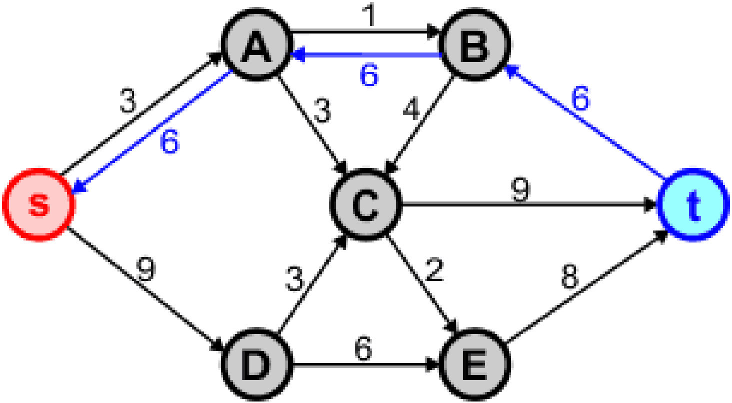
Algorytm Forda-Fulkersona - przykład



Szukamy ścieżki rozszerzającej np. może to być ścieżka: $p = \{s \rightarrow A \rightarrow B\}$. Na ścieżce p znajdują się trzy kanały sieci rezydualnej: (s, A), (A, B) i (B, t). Przepustowość rezydualna $c_f(p)$ ścieżki jest równa najmniejszej przepustowości rezydualnej jej kanałów.

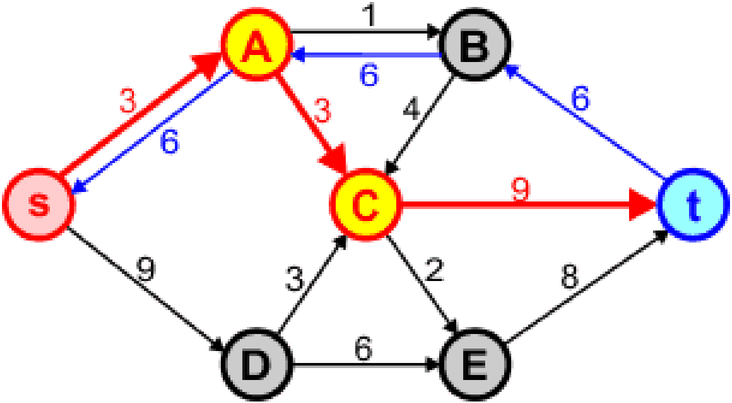
$$|f_{nowy}| = |f_{stary}| + c_f(p) = 0 + 6 = 6$$

Algorytm Forda-Fulkersona - przykład



Zwiększenie przepływu w kanale sieci pierwotnej o $c_f(p)$ odpowiada zmniejszeniu przepustowości rezydualnej tego kanału. Jednocześnie wraz z pojawieniem się przepływu w kanale sieci pierwotnej powstaje kanał przeciwny w sieci rezydualnej o przepustowości rezydualnej równej przepływowi.

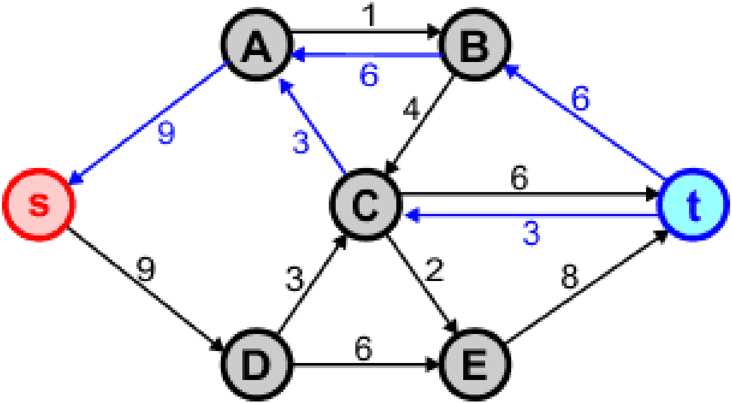
Algorytm Forda-Fulkersona - przykład



W nowej sieci rezydualnej szukamy kolejnej ścieżki rozszerzającej:

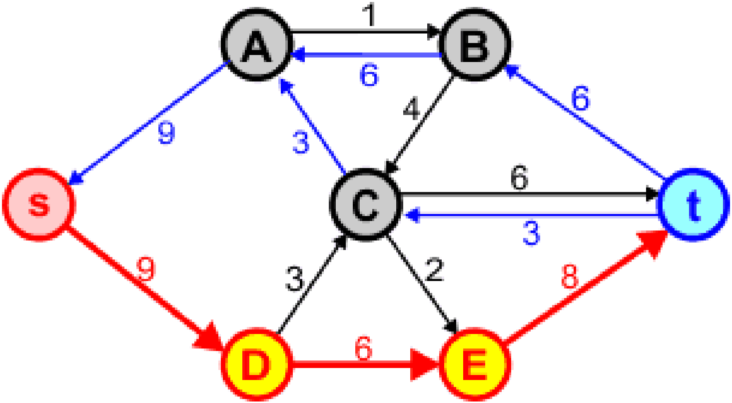
$$p = s \to A \to C \to t, c_f(p) = 3$$

Algorytm Forda-Fulkersona - przykład



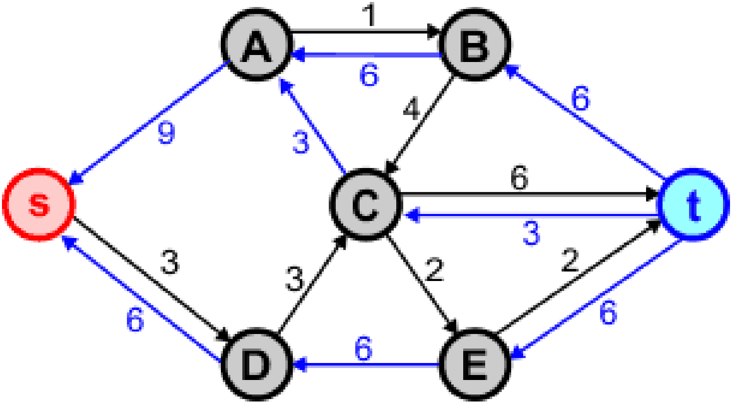
Przepływ zwiększamy: $|f_{nowy}| = |f_{stary}| + c_f(p) = 6 + 3 = 9$ i modyfikujemy przepustowości rezydualne krawędzi ścieżki rozszerzającej otrzymując nową sieć rezydualną. Znikają z niej kanały (s, A) i (A, C) - wykorzystały już swój potencjał zwiększania przepływu.

Algorytm Forda-Fulkersona - przykład

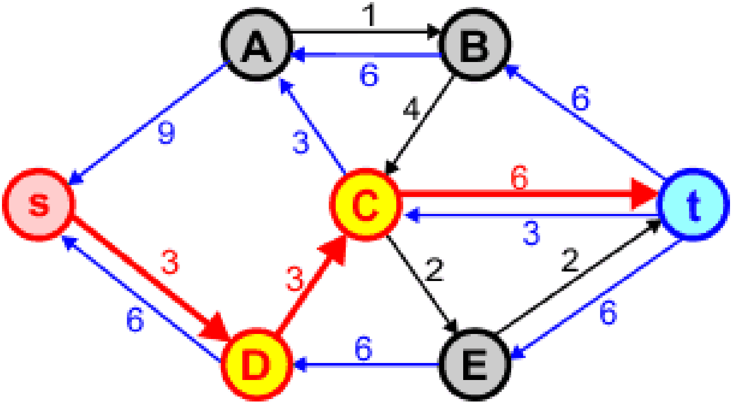


Szukamy kolejnej ścieżki rozszerzającej: $p = s \to D \to E \to t, c_f(p) = 6$

$$|f_{nowy}| = |f_{stary}| + c_f(p) = 9 + 6 = 15$$



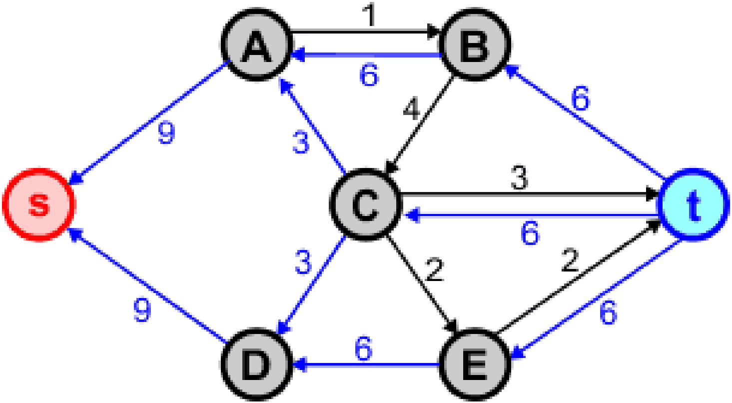
Ponownie modyfikujemy przepustowości rezydualne krawędzi ścieżki rozszerzającej otrzymując nową sieć rezydualną. W nowej sieci rezydualnej zniknął kanał (D, E).



Wciąż jednakże możemy znaleźć nową ścieżkę rozszerzającą:

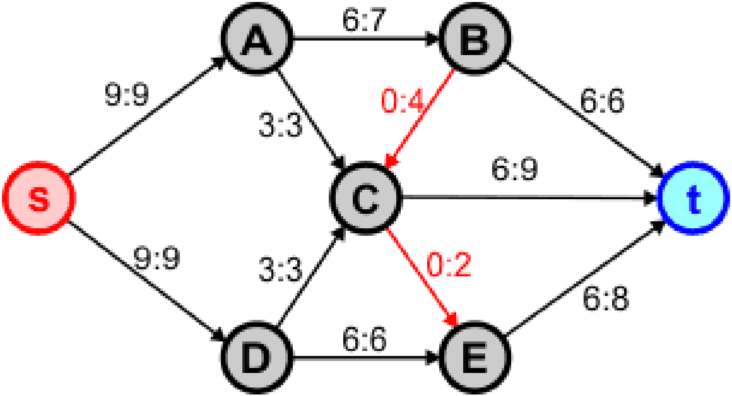
$p = s \to D \to C \to t, c_f(p) = 3 \ |f_{nowy}| = |f_{stary}| + c_f(p) = 15 + 3 = 18$

Algorytm Forda-Fulkersona - przykład



Ponownie po zmodyfikowaniu sieci rezydualnej otrzymujemy nową sieć rezydualną. W tej sieci rezydualnej nie znajdziemy już żadnej nowej ścieżki rozszerzającej - ze źródła *s* nie wychodzi żaden kanał.

Algorytm Forda-Fulkersona - przykład



Otrzymaliśmy maksymalny przepływ. Kanały z przepustowością rezydualna równą zero są zbędne . Kanały (B,C) i (C,E) można zlikwidować.

Dziękuję