Podstawowe algorytmy grafowe i ich zastosowania

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Autor projektu: dr Andrzej Mróz (UMK)

Projekt pn. „*Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych*" realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu

Przeszukiwanie (przeglądanie) grafu = systematyczne przechodzenie wzdłuż jego krawędzi w celu odwiedzenia wszystkich wierzchołków.

- Służy m.in. do zbierania informacji o strukturze grafu.
- Algorytmy grafowe często zaczyna się od przeszukania wejściowego grafu.
- Wiele bardziej zaawansowanych algorytmów grafowych jest modyfikacją podstawowych algorytmów przeszukiwania.

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu

Dwie metody przeszukiwania grafu:

- **w głąb** (ang. *depth-first search*, DFS),
- **wszerz** (ang. *breadth-first search*, BFS).

W obu metodach będziemy iterować po sąsiadach danego wierzchołka. Dlatego najwygodniejszą reprezentacją grafu są listy sąsiedztwa.

Oznaczenia: Adj = tablica list sąsiedztwa;
dla $u \in V$, Adj$[u]$ = lista sąsiadów u w grafie G.

Uwaga

Iterację po sąsiadach danego wierzchołka można łatwo zrealizować również na macierzy sąsiedztwa. Jest to jednak bardziej kosztowne w sensie złożoności obliczeniowej.

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu w głąb

Ustalmy graf niezorientowany $G = (V, E)$.

Z ustalonego wierzchołka **źródłowego** w sięgamy coraz głębiej w graf, jeżeli jest to tylko możliwe.

- badamy wszystkie niezbadane dotąd krawędzie ostatnio odwiedzonego wierzchołka v,
- gdy wszystkie krawędzie v są zbadane, wracamy do wierzchołka, z którego v został odwiedzony,
- proces kontynuujemy dopóki wszystkie wierzchołki osiągalne z wierzchołka źródłowego w nie zostaną odwiedzone.

Jeżeli po powyższym procesie pozostanie jakikolwiek nie odwiedzony wierzchołek u, kontynuujemy przeszukiwanie traktując go jako nowy wierzchołek źródłowy.
Cały proces powtarzamy, aż wszystkie wierzchołki w grafie zostaną odwiedzone.

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu w głąb

Należy zatem zapamiętywać stany, w jakich znajdują się w danej chwili wierzchołki.

Do zapisania aktualnego stanu wierzchołka używamy jednego z trzech kolorów:

- **biały** – wszystkie wierzchołki na początku;
- wierzchołek odwiedzany po raz pierwszy kolorujemy na **szaro**;
- wierzchołek przetworzony (= lista jego sąsiadów jest całkowicie zbadana) kolorujemy na **czarno**.

Uwaga

Tak naprawdę w najprostszej implementacji wystarczą dwa stany: nieodwiedzony i odwiedzony. Jednak dokładniejsze rozróżnienie stanów pozwala lepiej zrozumieć ideę DFS oraz jest wykorzystywane w niektórych zastosowaniach.

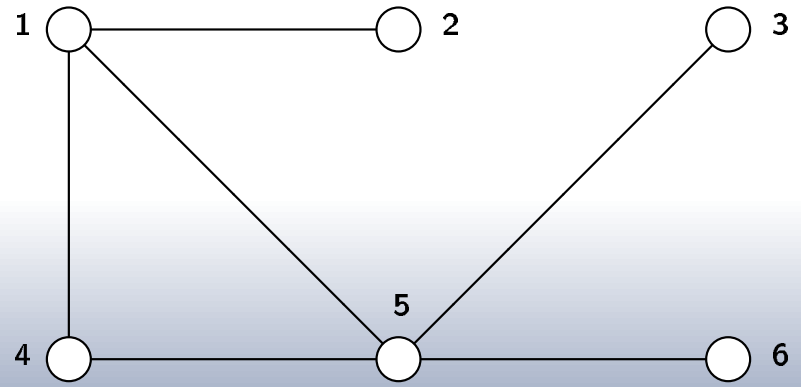
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```

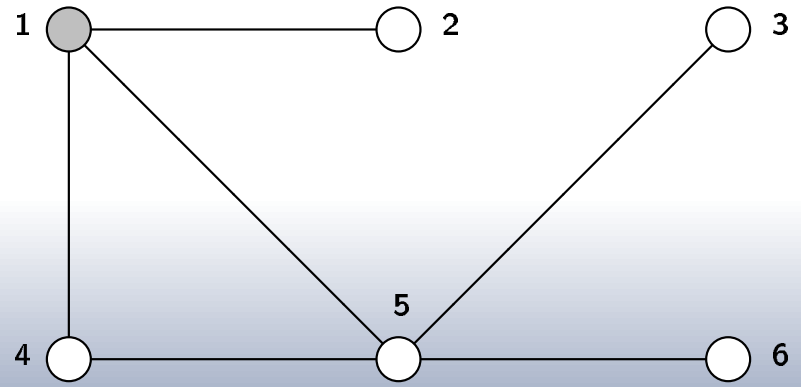
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



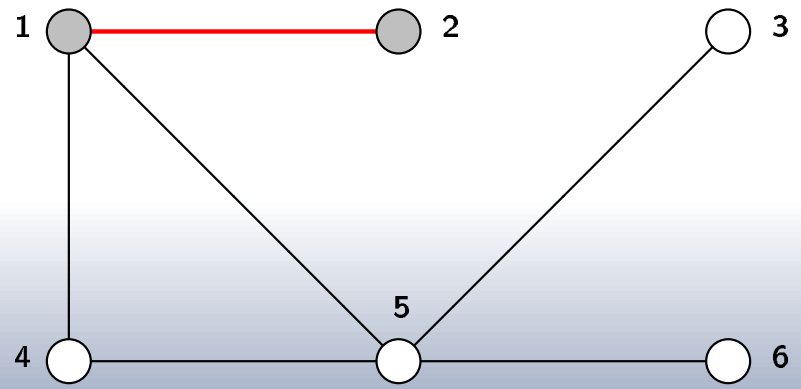
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



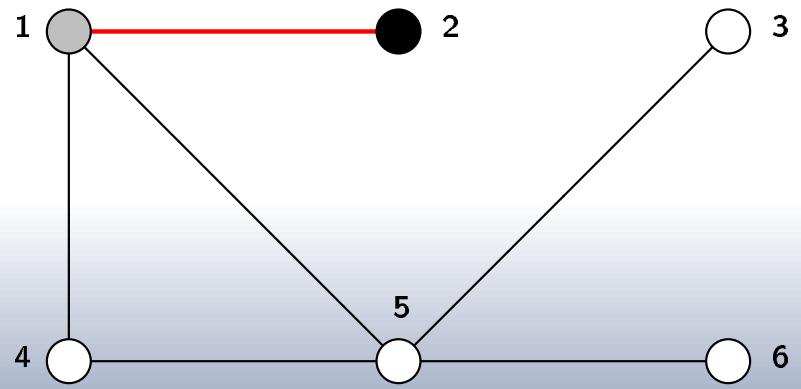
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



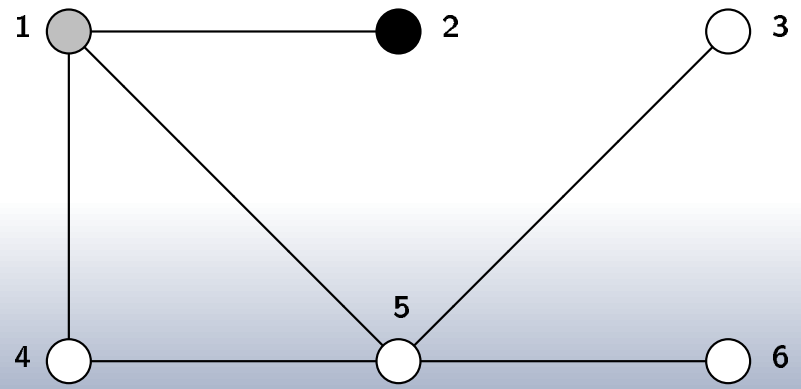
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



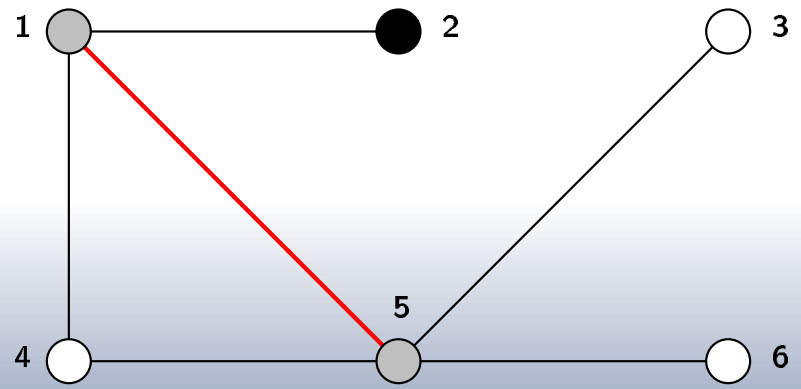
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



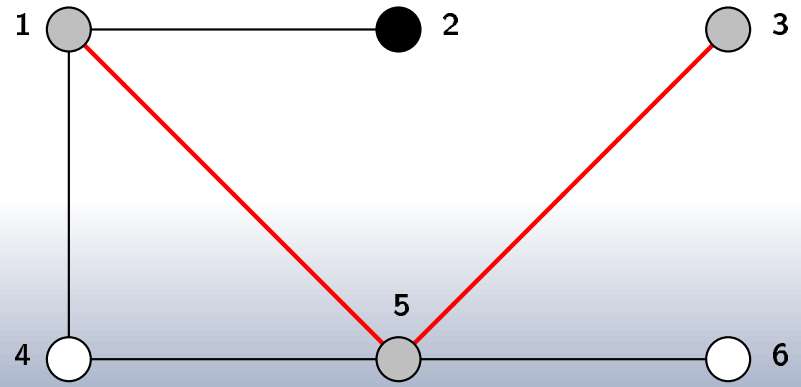
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



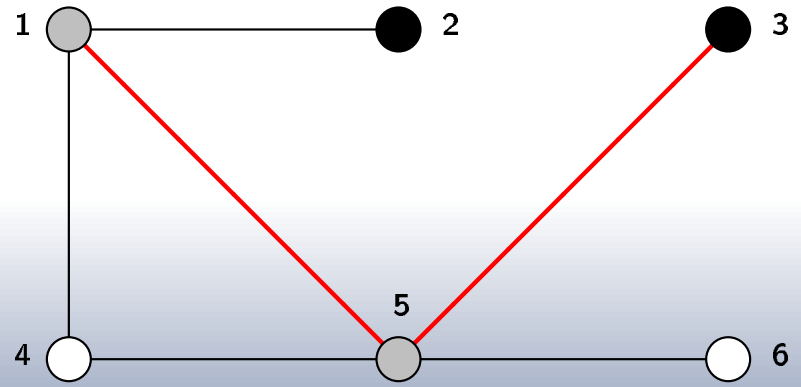
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



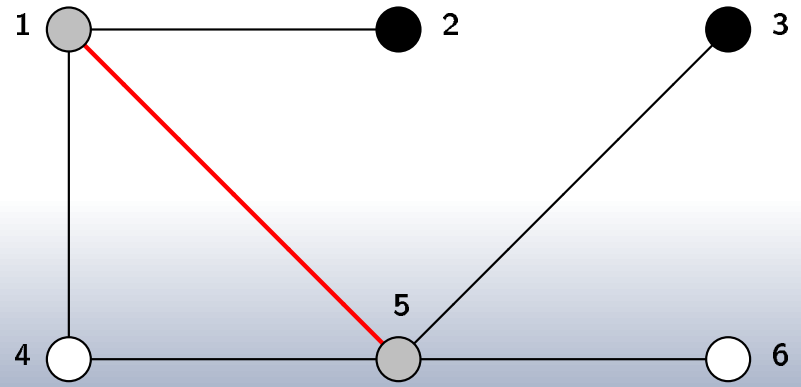
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



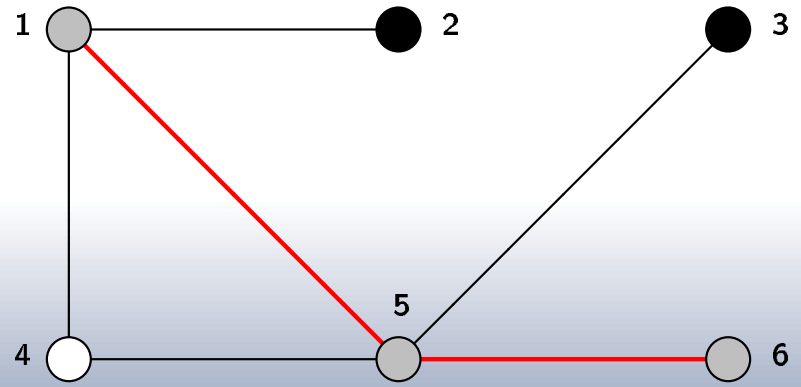
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



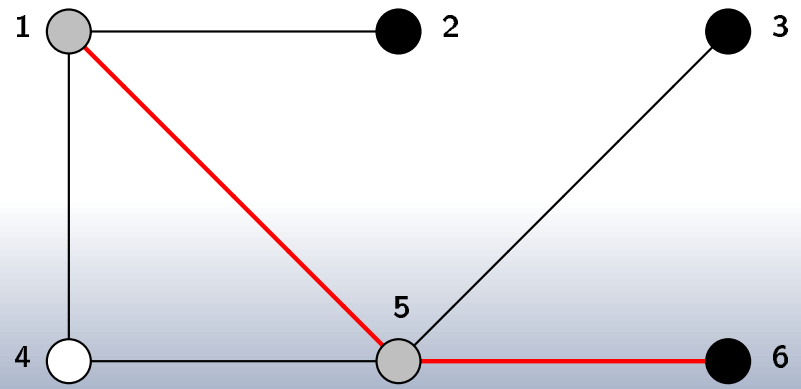
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



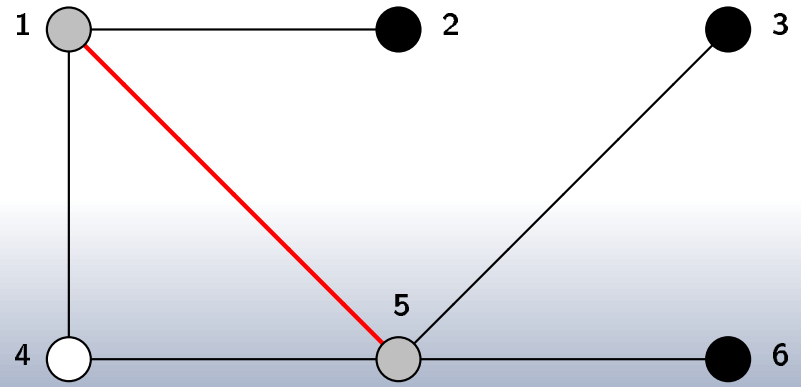
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



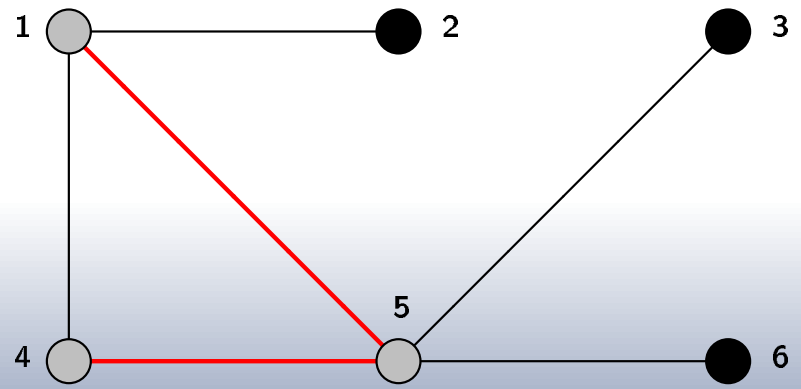
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



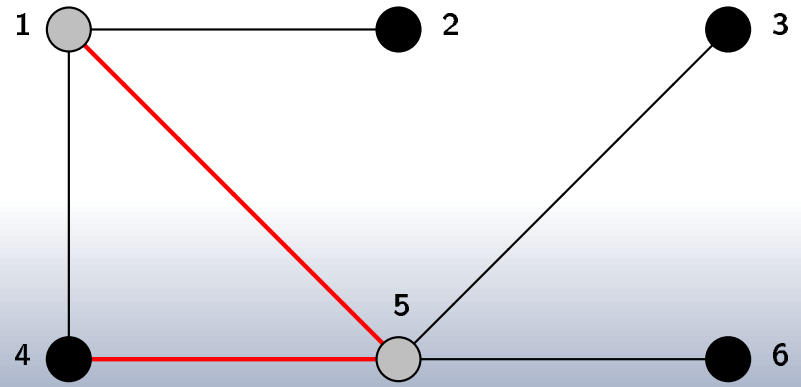
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



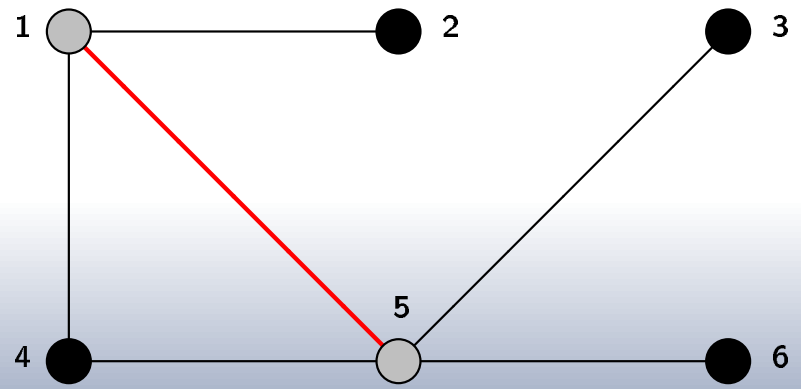
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



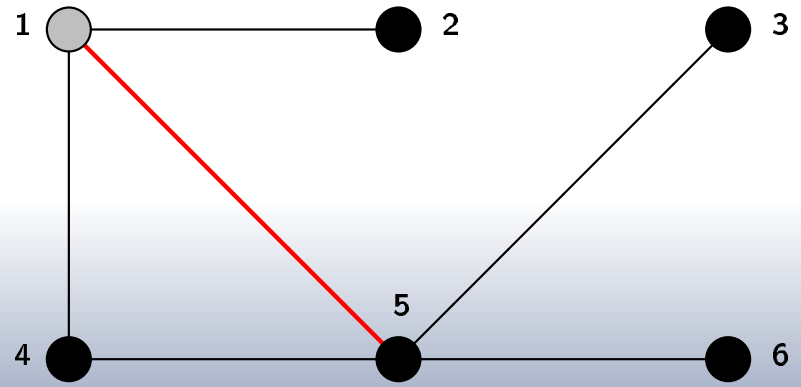
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



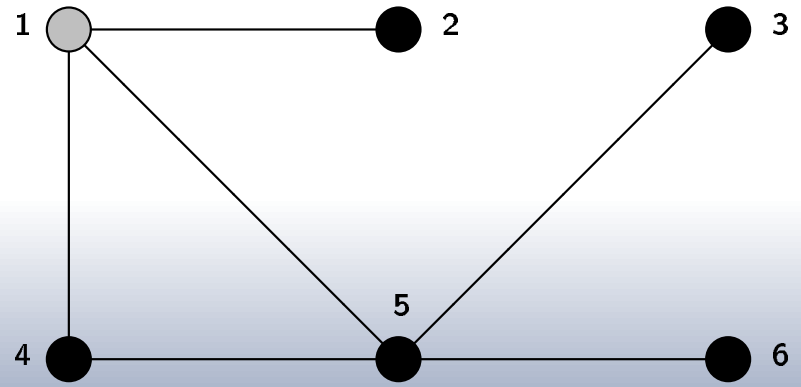
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



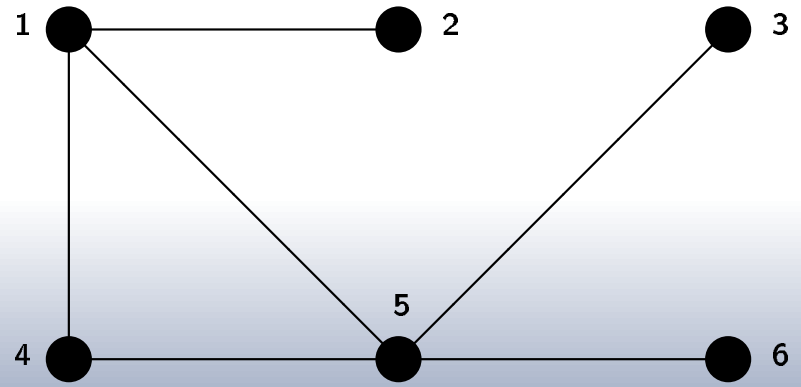
Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



Przeszukiwanie grafu w głąb

```
DFS-Visit(G, u)
 1   begin
 2       kolor(u) := szary;
 3       for każdy v ∈ Adj[u] do
 4           if kolor(v) = biały then DFS-Visit(G, v);
 5       kolor(u) := czarny;
 6   end;
```



Przeszukiwanie grafu w głąb

Zauważmy, że po wywołaniu `DFS-Visit` dla wierzchołka u wierzchołki, które nie są osiągalne z u pozostaną nieodwiedzone (białe). Aby zatem przejrzeć cały graf, należy wywoływać `DFS-Visit` dopóki będą białe wierzchołki.

Pełen przebieg algorytmu DFS jest realizowany przez poniższą procedurę:

```
DFS(G)
 1   begin
 2      for każdy u ∈ V(G) do
 3         kolor(u) := biały;
 4      for każdy u ∈ V(G) do
 5         if kolor(u) = biały then
 6            DFS-Visit(G, u);
 7   end;
```

Przeszukiwanie grafu w głąb

- **Złożoność czasowa:** $\mathcal{O}(|V| + |E|)$: odwiedzamy każdy wierzchołek i (dwukrotnie!) każdą krawędź.
- **Złożoność pamięciowa:** $\mathcal{O}(|V|)$: przechowywanie kolorów, przechowywanie wierzchołków na stosie rekurencji.

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Badanie spójności

Podstawowe zastosowania:

- Sprawdzanie spójności grafu.
- Wyznaczanie składowych spójności grafu.

Zauważmy, że w procedurze `DFS` wywołujemy `DFS-Visit` dokładnie tyle razy, ile jest składowych spójności w grafie G.

W szczególności, gdy graf jest spójny, `DFS-Visit` zostanie wywołana dokładnie raz przez procedurę `DFS`.

Nietrudno uzupełnić procedurę `DFS` o kod zliczający składowe spójności, jak również przypisujący wierzchołkom „numer” składowej.

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

DFS - inne zastosowania

Inne zastosowania:

- Wyznaczanie silnie spójnych składowych (w wersji dla grafu skierowanego).
- Sortowanie topologiczne grafu zorientowanego (bez zorientowanych cykli).
- Generowanie labiryntów.
- Znajdowanie drogi w labiryncie.

Zainteresowanych odsyłamy do literatury (patrz też dodatkowe materiały do wykładu i zajęć laboratoryjnych).

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu wszerz

Z ustalonego wierzchołka **źródłowego** s przeglądamy kolejne wierzchołki z niego osiągalne.

- wierzchołki w odległości (=najmniejszej liczbie krawędzi) k od źródła są odwiedzane przed wierzchołkami w odległości $k+1$,
- granica między wierzchołkami odwiedzonymi i nieodwiedzonymi jest przekraczana „jednocześnie" na całej jej szerokości.

Jeżeli po powyższym procesie pozostanie jakikolwiek nie odwiedzony wierzchołek u, kontynuujemy przeszukiwanie traktując go jako nowy wierzchołek źródłowy.

Cały proces powtarzamy, aż wszystkie wierzchołki w grafie zostaną odwiedzone.

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu wszerz

Każdy wierzchołek posiada jeden z 3 kolorów: biały, szary lub czarny (podobnie jak w DFS). Na początku wszystkie wierzchołki są białe.

Wierzchołki szare są przechowywane w kolejce FIFO.

Po jej opuszczeniu kolorujemy je na czarno.

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Kolejka FIFO

Kolejka FIFO Q jest dynamiczną strukturą danych w formie ciągu, do której można dołączyć składnik tylko w jednym końcu (na końcu kolejki - **tail**), a usunąć tylko w drugim końcu (na początku kolejki - **head**). Mamy zatem dwie podstawowe operacje:

- $\mathrm{Enqueue}(Q, v)$ = dodanie elementu v na końcu kolejki,
- $\mathrm{Dequeue}(Q)$ = usunięcie elementu z początku kolejki.

Dodatkowo wykorzystamy operację

- $\mathrm{Head}(Q)$ = zwrócenie elementu z początku kolejki (bez usuwania go).

Uwaga

Kolejkę implementujemy przy użyciu struktur wskaźnikowych. Można też zasymulować jej działanie na zwykłej (statycznej) tablicy, albo wykorzystać gotowe struktury biblioteczne, np. queue biblioteki STL w C++: `http://www.cplusplus.com/reference/queue/queue/`.

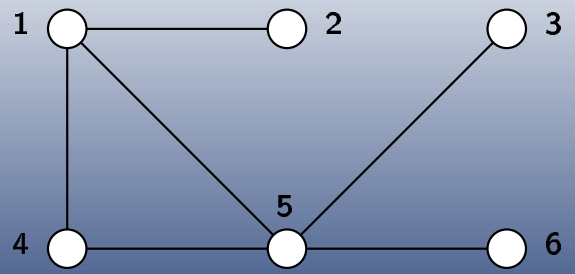
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu wszerz

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```

Przeszukiwanie grafu wszerz

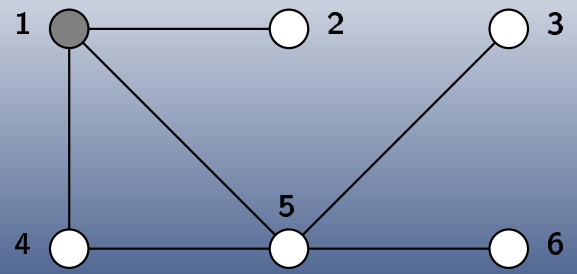
```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```



Q:

Przeszukiwanie grafu wszerz

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```

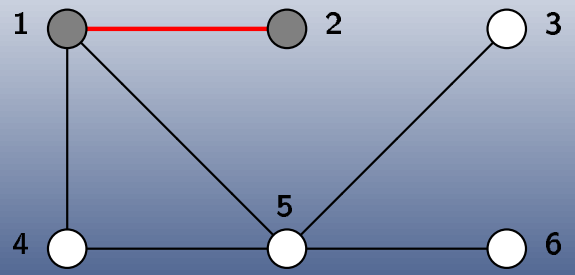


Q: 1

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu wszerz

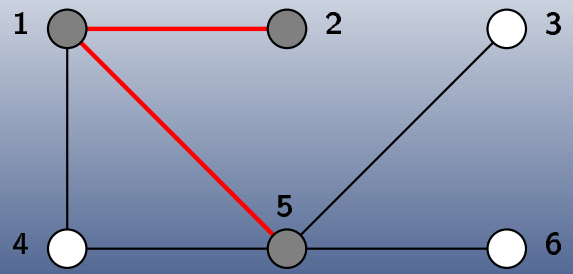
```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```



Q: 1 2

Przeszukiwanie grafu wszerz

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```

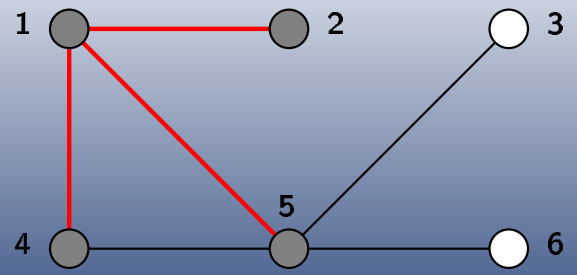


Q: 1 2 5

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu wszerz

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```

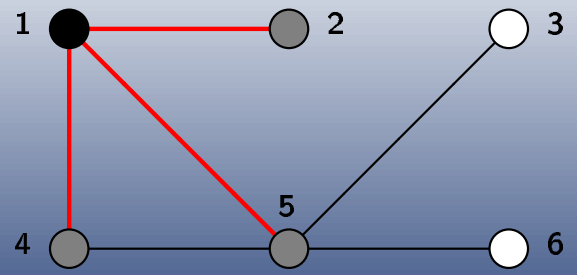


Q: 1 2 5 4

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu wszerz

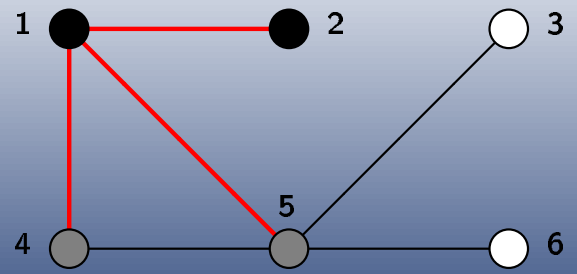
```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```



Q: 2 5 4

Przeszukiwanie grafu wszerz

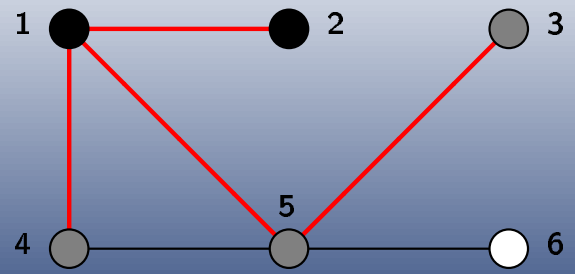
```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```



Q: 5 4

Przeszukiwanie grafu wszerz

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```

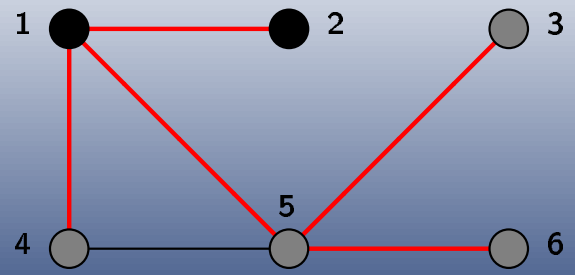


Q: 5 4 3

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Przeszukiwanie grafu wszerz

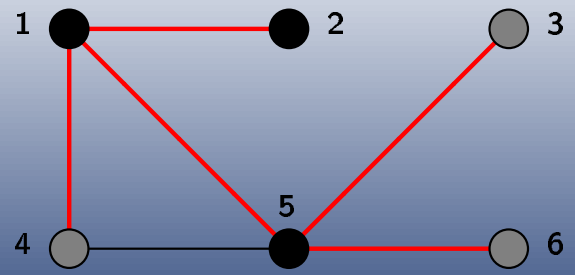
```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```



Q: 5 4 3 6

Przeszukiwanie grafu wszerz

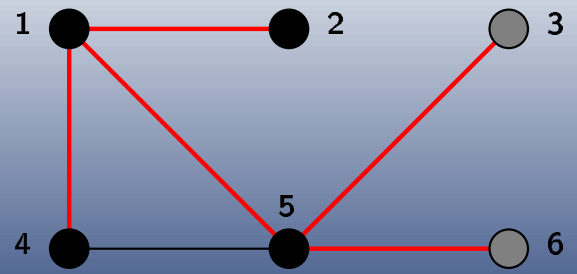
```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```



Q: 4 3 6

Przeszukiwanie grafu wszerz

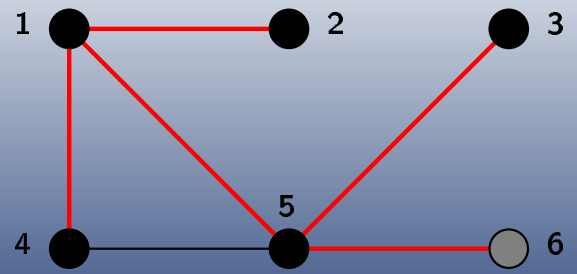
```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```



Q: 3 6

Przeszukiwanie grafu wszerz

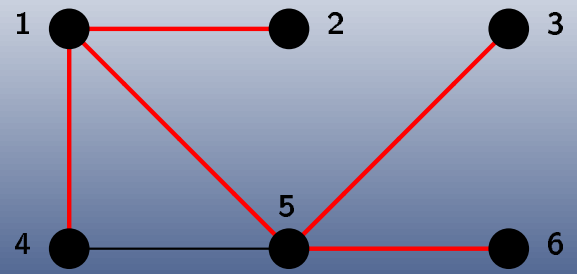
```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```



Q: 6

Przeszukiwanie grafu wszerz

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      Enqueue(Q, v)
10                end;
11             Dequeue(Q);
12             kolor(u) := czarny
13       end
14   end
```



Q:

Przeszukiwanie grafu wszerz

Zauważmy, że po wywołaniu `BFS-Visit` dla wierzchołka u wierzchołki, które nie są osiągalne z u pozostaną nieodwiedzone (białe). Aby zatem przejrzeć cały graf, należy wywoływać `BFS-Visit` dopóki będą białe wierzchołki.

Pełen przebieg algorytmu BFS jest realizowany przez poniższą procedurę:

```
BFS(G)
 1   begin
 2      for każdy u ∈ V(G) do
 3         kolor(u) := biały;
 4      for każdy u ∈ V(G) do
 5         if kolor(u) = biały then
 6            BFS-Visit(G, u);
 7   end;
```

Przeszukiwanie grafu wszerz

- **Złożoność czasowa:** $\mathcal{O}(|V| + |E|)$ (analogicznie jak dla DFS).
- **Złożoność pamięciowa:** $\mathcal{O}(|V|)$ (przechowywanie kolorów, kolejka).

Najkrótsza droga

Zastosowanie: znajdowanie najkrótszej drogi pomiędzy wierzchołkami s i v.

Modyfikacja procedury `BFS-Visit`: przechowywanie **wektora poprzedników** π.
Tj. poprzednikiem wierzchołka v na najkrótszej drodze z s do v jest wierzchołek $\pi[v]$;
poprzednikiem wierzchołka $\pi[v]$ jest wierzchołek $\pi[\pi[v]]$...

Najkrótsza droga

Przygotowanie:

```
for każdy u ∈ V(G) do
   begin
      kolor(u) := biały;
      π[u] := ∞
   end;
```

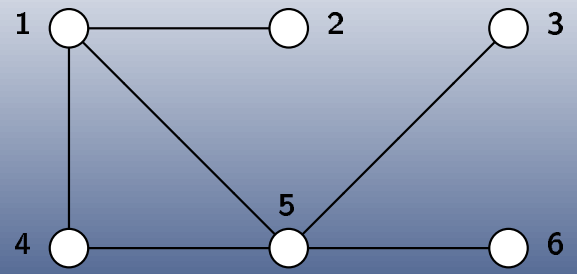
Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞



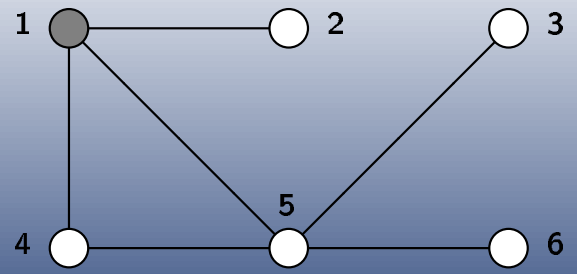
Q:

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞



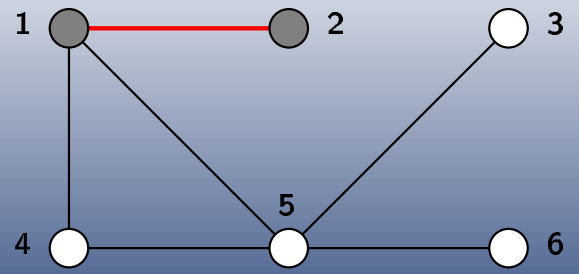
Q: 1

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	∞
4	∞
5	∞
6	∞



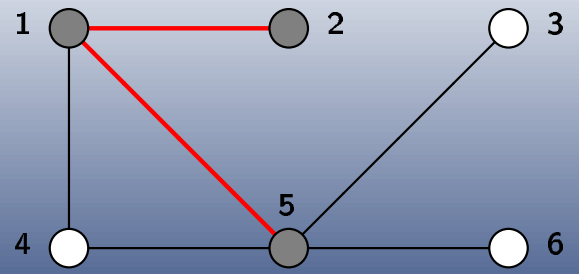
Q: 1 2

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	∞
4	∞
5	1
6	∞



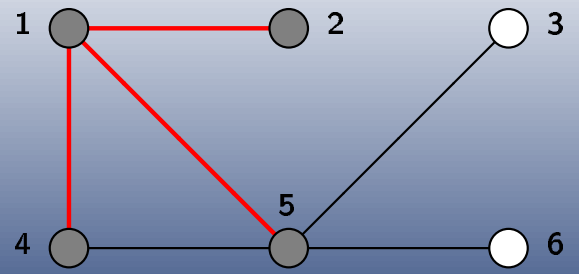
Q: 1 2 5

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	∞
4	1
5	1
6	∞



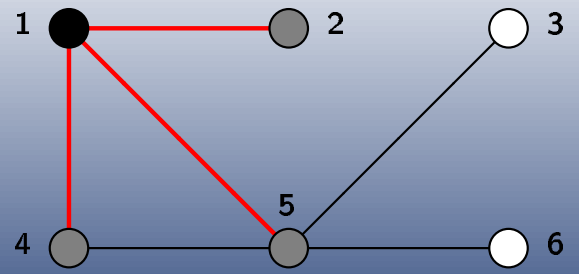
Q: 1 2 5 4

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	∞
4	1
5	1
6	∞



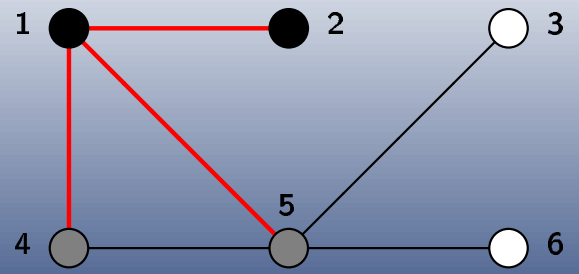
Q: 2 5 4

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	∞
4	1
5	1
6	∞



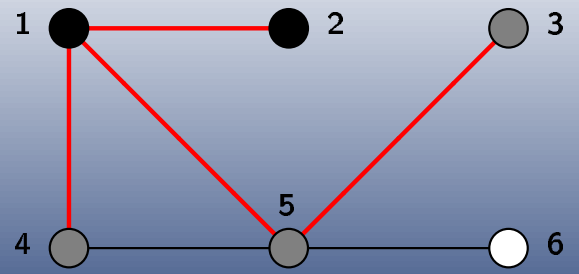
Q: 5 4

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	5
4	1
5	1
6	∞



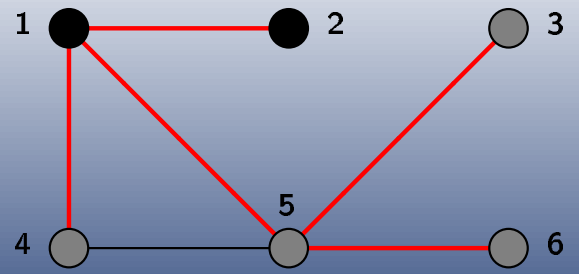
Q: 5 4 3

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	5
4	1
5	1
6	5

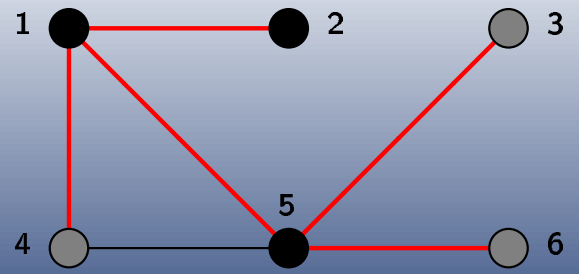


Q: 5 4 3 6

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	5
4	1
5	1
6	5



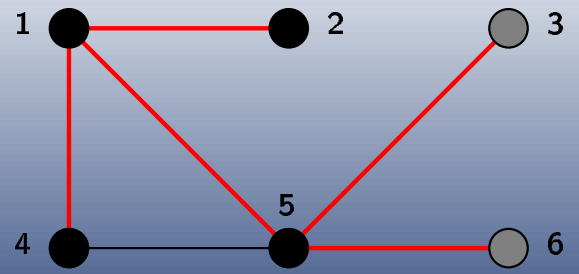
Q: 4 3 6

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	5
4	1
5	1
6	5



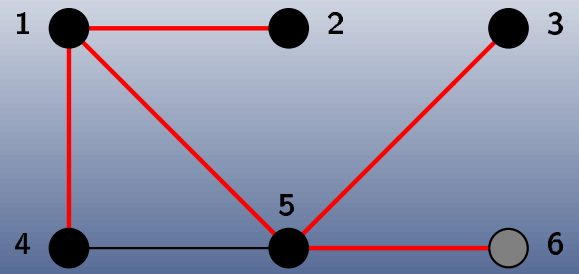
Q: 3 6

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	5
4	1
5	1
6	5



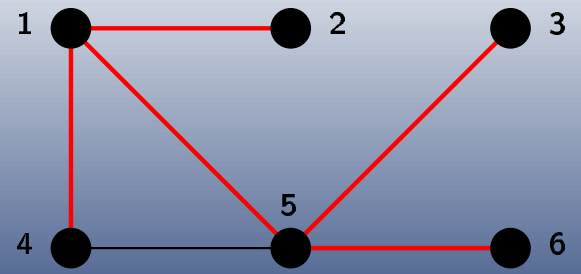
Q: 6

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zmodyfikowany BFS

```
BFS-Visit(G, s)
 1   begin
 2       kolor(s) := szary;
 3       Q := {s};
 4       while Q <> ∅ do begin
 5             u := Head(Q);
 6             for każdy v ∈ Adj[u] do
 7                if kolor(v) = biały then begin
 8                      kolor(v) := szary;
 9                      π[v] := u;
10                      Enqueue(Q, v)
11                end;
12             Dequeue(Q);
13             kolor(u) := czarny
14       end
15   end
```

w	$\pi[w]$
1	∞
2	1
3	5
4	1
5	1
6	5



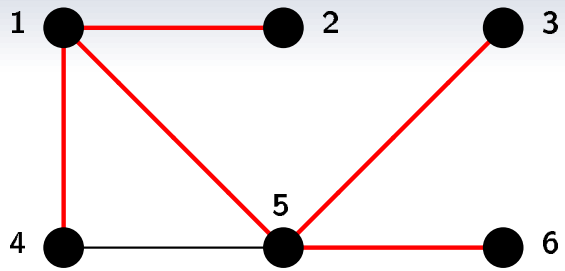
Q:

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Odczytanie dróg

Po wykonaniu `BFS-Visit(G, s)` w wektorze π zakodowane są najkrótsze drogi z wierzchołka źródłowego s do wszystkich wierzchołków w osiągalnych z s. W naszym przykładzie $s = 1$:

w	$\pi[w]$
1	∞
2	1
3	5
4	1
5	1
6	5



Np. najkrótsza droga z $s = 1$ do 6 to:
$6 \leftarrow \pi[6] \leftarrow \pi[\pi[6]]$, czyli
$6 \leftarrow 5 \leftarrow 1$.

A najkrótsza droga z $s = 1$ do 3 to:
$3 \leftarrow \pi[3] \leftarrow \pi[\pi[3]]$, czyli
$3 \leftarrow 5 \leftarrow 1$.

Odczytanie dróg

Odczytanie i wypisanie drogi z s do v „od końca” może być zrealizowane przez poniższy pseudokod:

```
PrintPathRev(s, v)
 1   begin
 2      if s = v then wypisz(v)
 3      else if π[v] = ∞ then wypisz('Nie ma drogi')
 4      else begin
 5         while v <> s do begin
 6            wypisz(v);
 7            v := π[v]
 8         end;
 9         wypisz(s)
10      end
11    end;
```

Odczytanie dróg

Jednak bardziej naturalnym byłoby wypisanie drogi z s do v „od początku”. Do tego może posłużyć poniższa elegancka procedura rekurencyjna:

```
PrintPath(s, v)
 1   begin
 2      if s = v then wypisz(v)
 3      else if π[v] = ∞ then wypisz('Nie ma drogi')
 4      else begin
 5         PrintPath(s, π[v]);
 6         wypisz(v)
 7      end
 8   end;
```

Podsumowanie

Jeżeli interesuje nas najkrótsza droga pomiędzy dwoma ustalonymi wierzchołkami s i t, to:

- uruchamiamy `BFS-Visit` dla wierzchołka s,
- odczytujemy drogę z s do t z wektora π.

Wektor π będzie zawierał, jako „skutek uboczny”, najkrótsze drogi z s do wszystkich wierzchołków osiągalnych z s (nie tylko t!). Wyliczania tej nadmiarowej informacji nie da się tu uniknąć.

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Podsumowanie

Zauważmy, że tu przez najkrótszą drogę rozumieliśmy drogę o najmniejszej liczbie krawędzi. Można rozważać grafy, w których krawędzie mają „długość” (lub inny „koszt”) i poszukiwać najkrótszych dróg względem tego parametru.

Tym zagadnieniem zajmiemy się na następnym wykładzie.

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego