Wstęp do programowania

Zastosowania stosów i kolejek

FIFO - LIFO

- Kolejki i stosy służą do przechowywania wartości zbiorów dynamicznych, czyli takich, które powstają przez dodawanie i usuwanie elementów zaczynając od zbioru pustego.
- Organizacja tych struktur danych odpowiada dwóm podstawowym metodom pobierania elementów ze zbioru w zależności od kolejności ich wkładania.
- FIFO: First In First Out
- LIFO: Last In First Out

Przechodzenie grafu

- Grafy najczęściej reprezentujemy za pomocą list sąsiedztwa. Wygodnie jest ponumerowac węzły grafu od 1 do n i w tablicy `S[1..n]of węzeł` przechowywać informację o sąsiadach węzłów.
-
```
type węzeł = record
        val:typ_elementu;
odwiedzony:Boolean;
  sąsiedzi:lista;
end;
```
- Zakładamy też, że mamy dostęp do struktury danych reprezentującej zbiór. Nazwijmy go magazynem.

Przechodzenie grafu

- Ogólny schemat przejścia grafu jest więc taki. Zaczynamy od inicjalizacji struktury danych:

```
S[1].odwiedzony:=true;
for i:=2 to n do
  S[i].odwiedzony:=false;
magazyn:= [1];
```

- Następnie w pętli przetwarzamy graf następująco:

Przechodzenie grafu

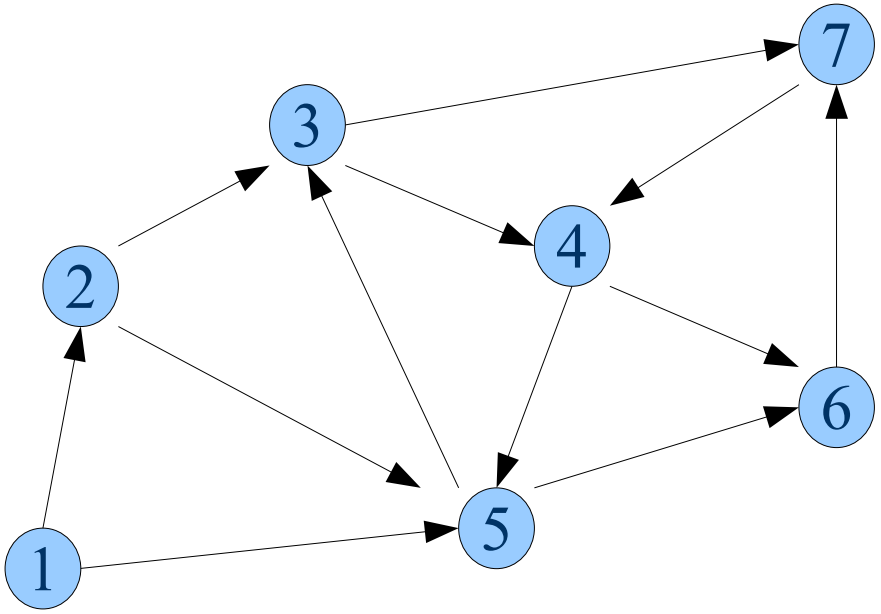
```
While magazyn <> [] do begin
 pobierz(x,magazyn); {jakieś x}
 przetwórz(x);
 l:=S[x].sąsiedzi;
 while l<> nil do begin
   if not S[l^.w].odwiedzony then begin
     S[l^.w].odwiedzony:=true;
     wstaw(x,magazyn)
   end;
   l:=l^.nast
 end
end
```

DFS - BFS

- Ten sposób przejścia grafu gwarantuje nam
  - odwiedzenie każdego węzła osiągalnego z węzła o numerze 1.
  - przetworzenie każdego węzła dokładnie raz
- Jeśli magazyn zrealizujemy za pomocą stosu, to otrzymamy algorytm przechodzenia grafu *wgłąb* (DFS: Depth-First-Search), a jesli za pomocą kolejki, to otrzymamy algorytm przechodzenia grafu *wszerz* (BFS: Breadth-First-Search).
- Oczywiście można sobie wyobrazić i dowolną inną kolejność pobierania elementów z magazynu, ale te dwie są najważniejsze.

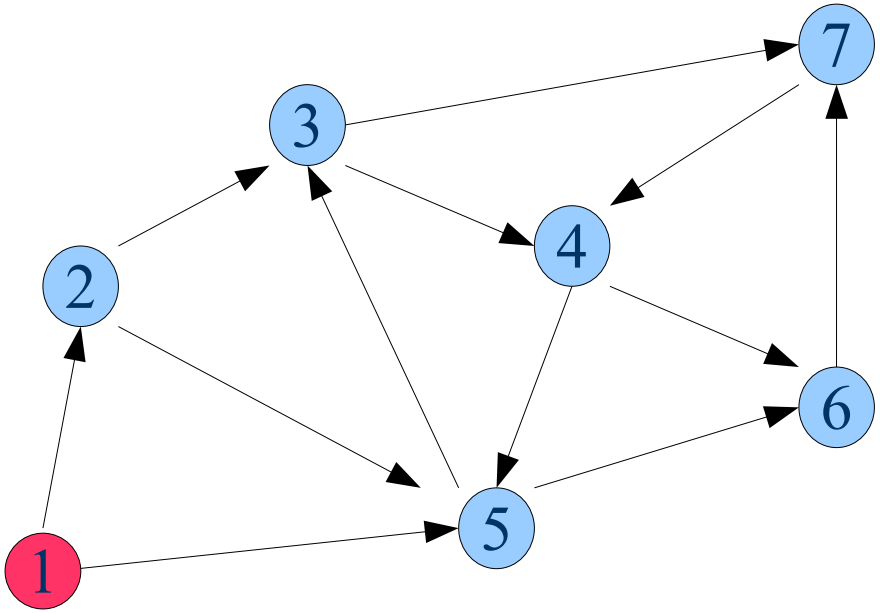
Przykład

- Naszym zadaniem jest wypisanie wszystkich numerów węzłów grafu w porządku DFS
- Graf wejściowy:



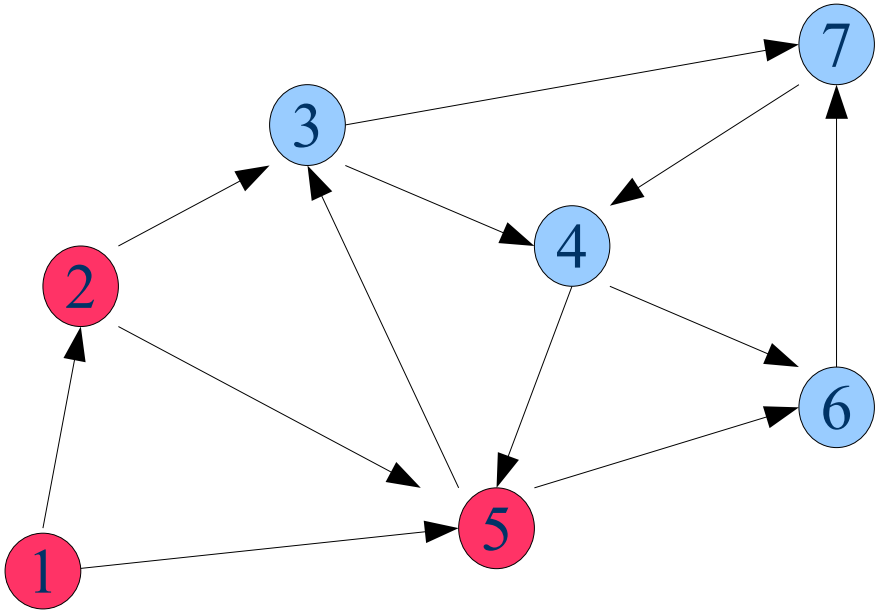
Przykład: DFS

- Stos: 1
- Wyjście:



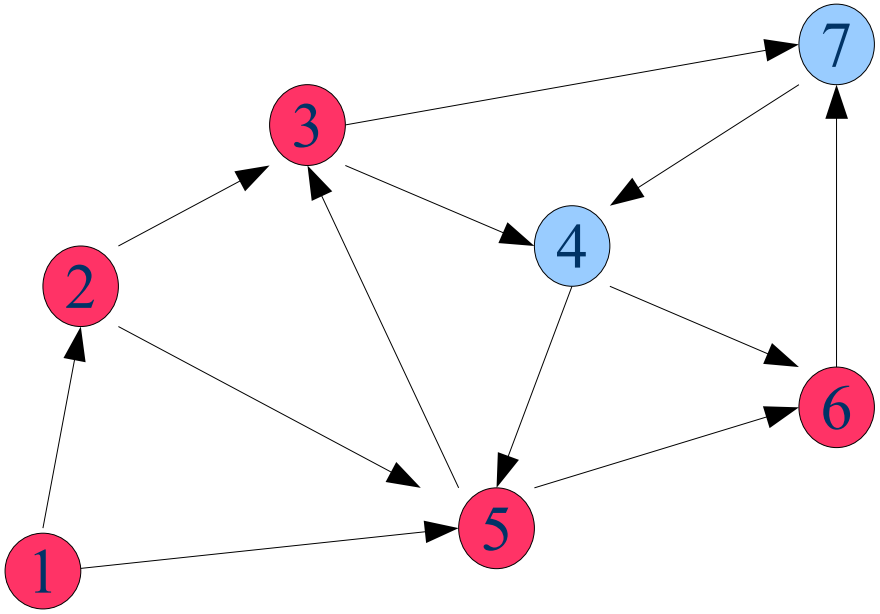
DFS

- Stos: 2 5
- Wyjście: 1



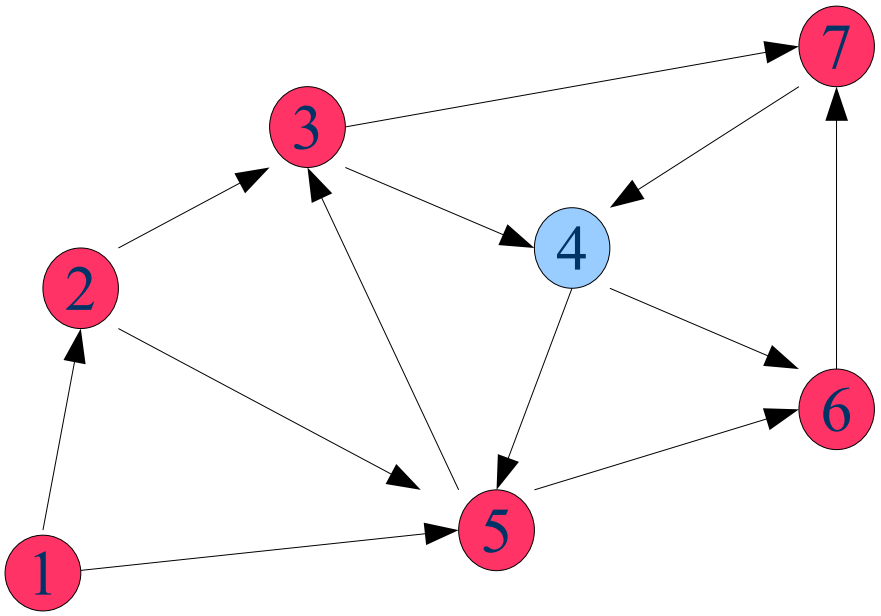
DFS

- Stos: 2 3 6
- Wyjście: 1 5



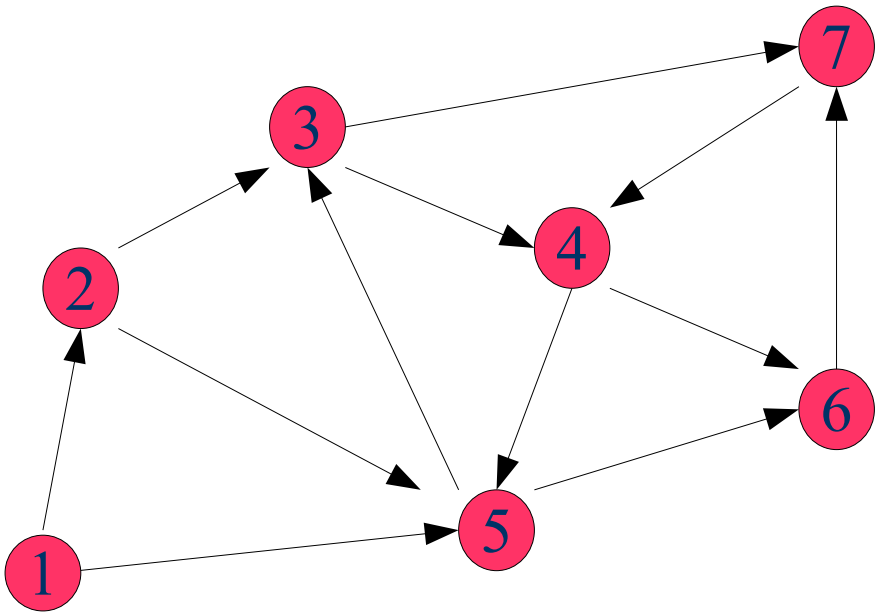
DFS

- Stos: 2 3 7
- Wyjście: 1 5 6



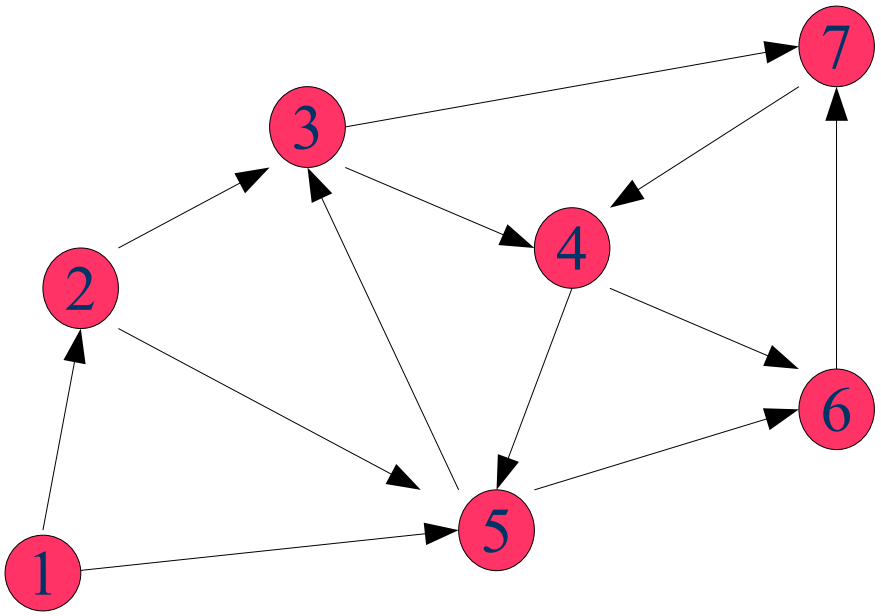
DFS

- Stos: 2 3 4
- Wyjście: 1 5 6 7



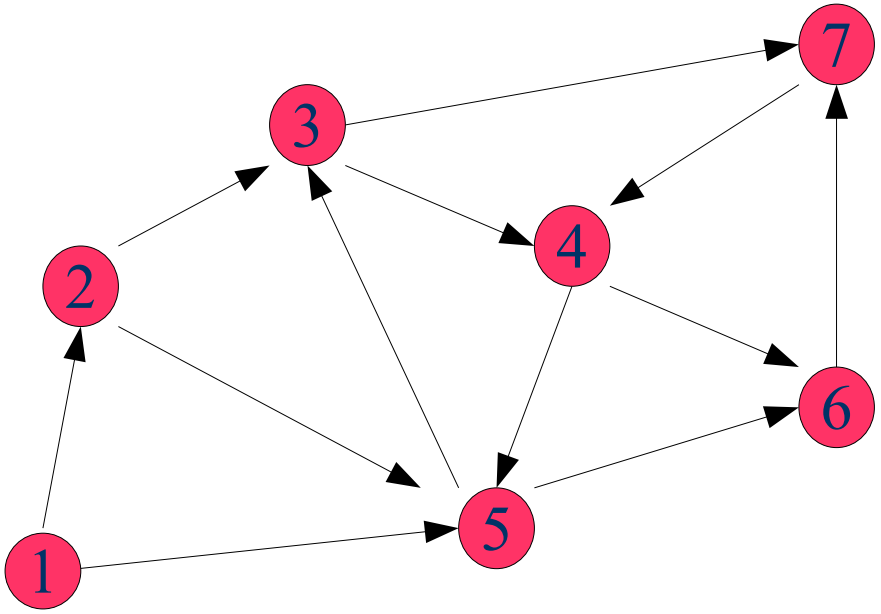
DFS

- Stos: 2 3
- Wyjście: 1 5 6 7 4



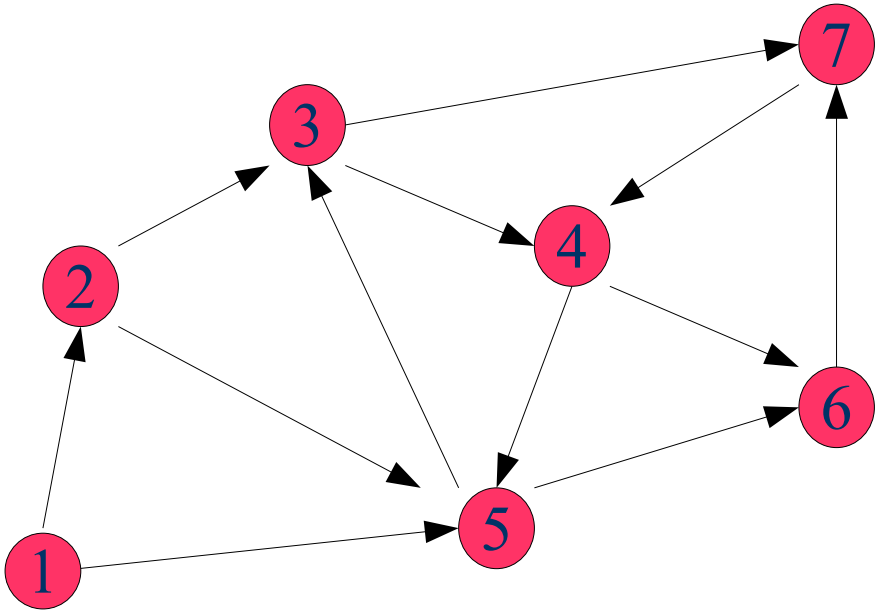
DFS

- Stos: 2
- Wyjście: 1 5 6 7 4 3



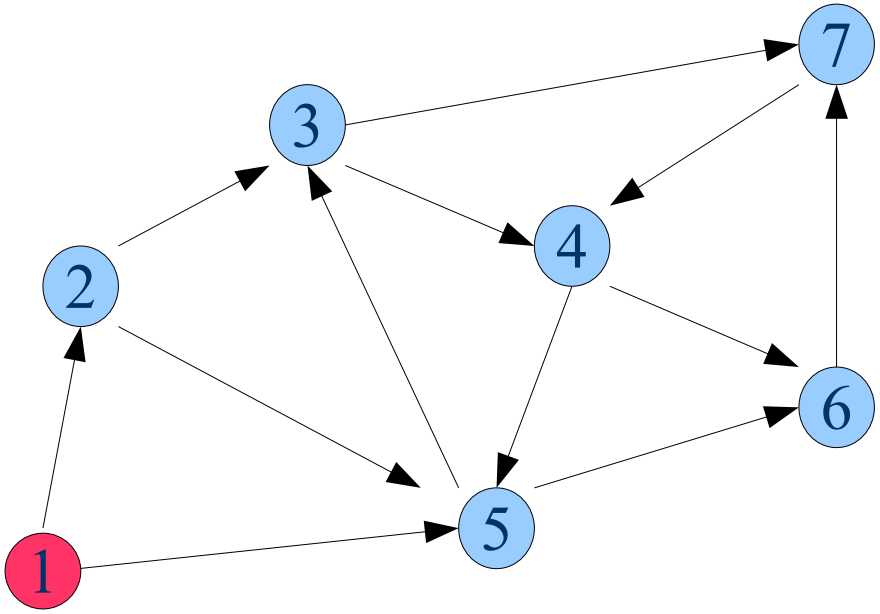
DFS

- Stos:
- Wyjście: 1 5 6 7 4 3 2



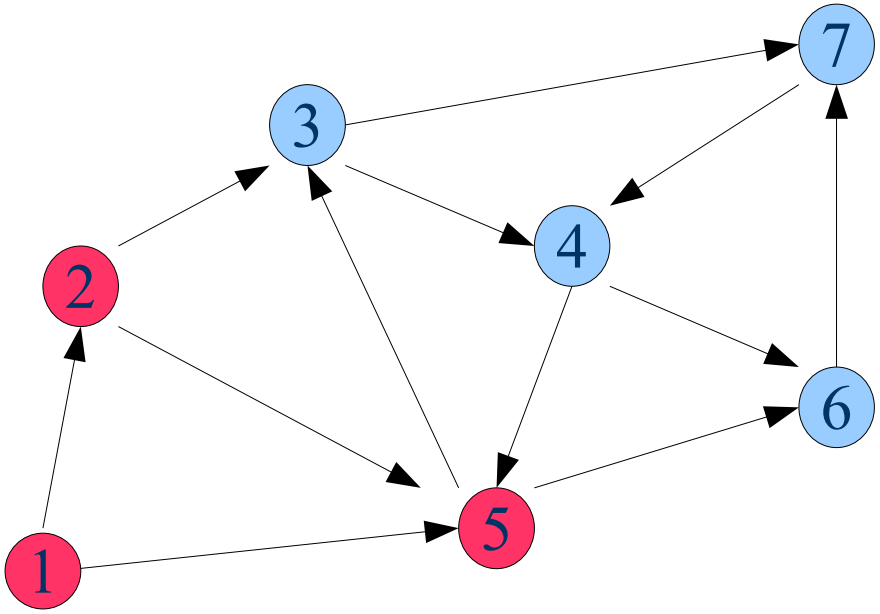
Przykład: BFS

- Kolejka: 1
- Wyjście:



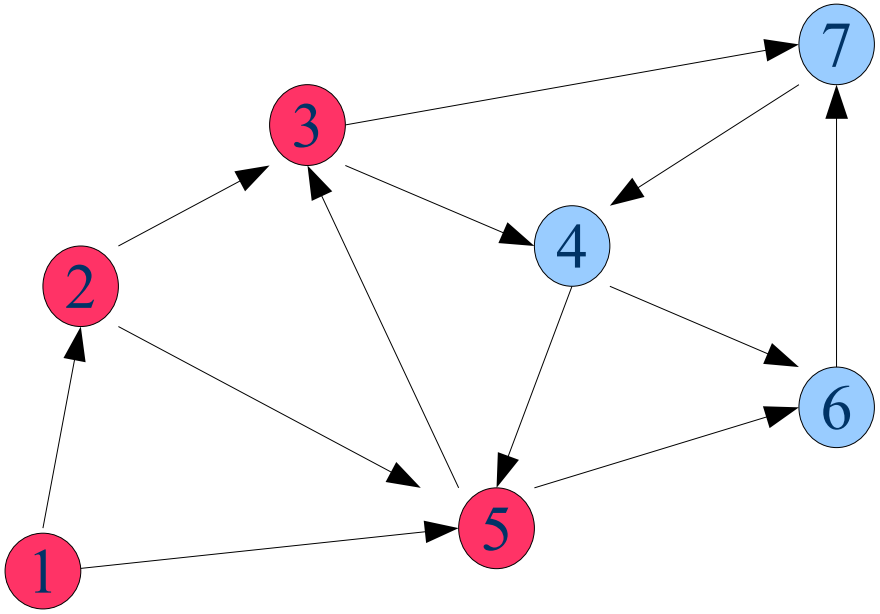
Przykład: BFS

- Kolejka: 2 5
- Wyjście: 1



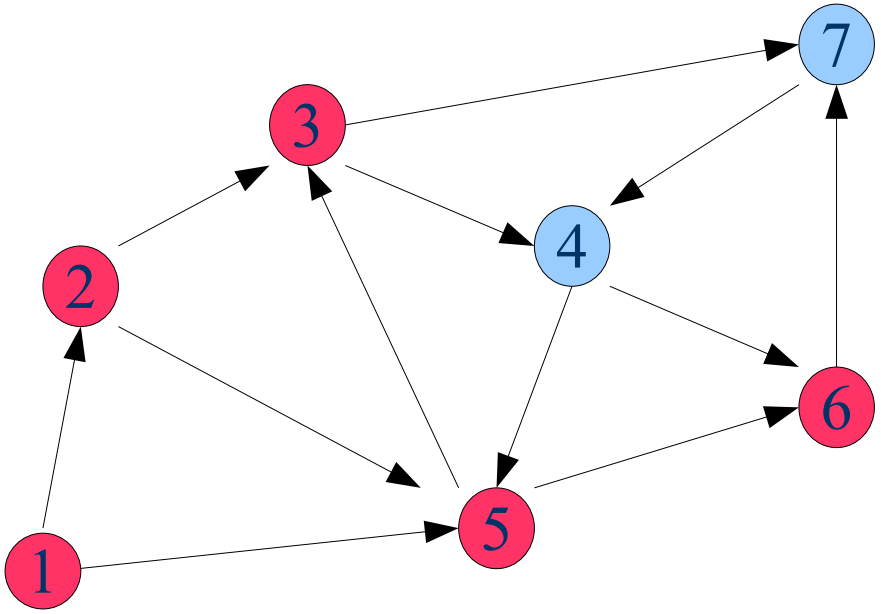
Przykład: BFS

- Kolejka: 5 3
- Wyjście: 1 2



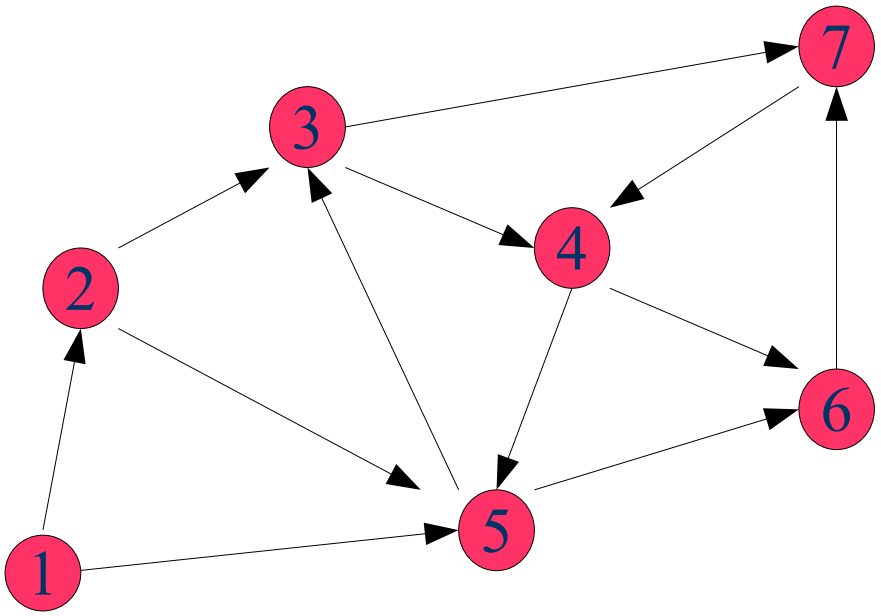
Przykład: BFS

- Kolejka: 3 6
- Wyjście: 1 2 5



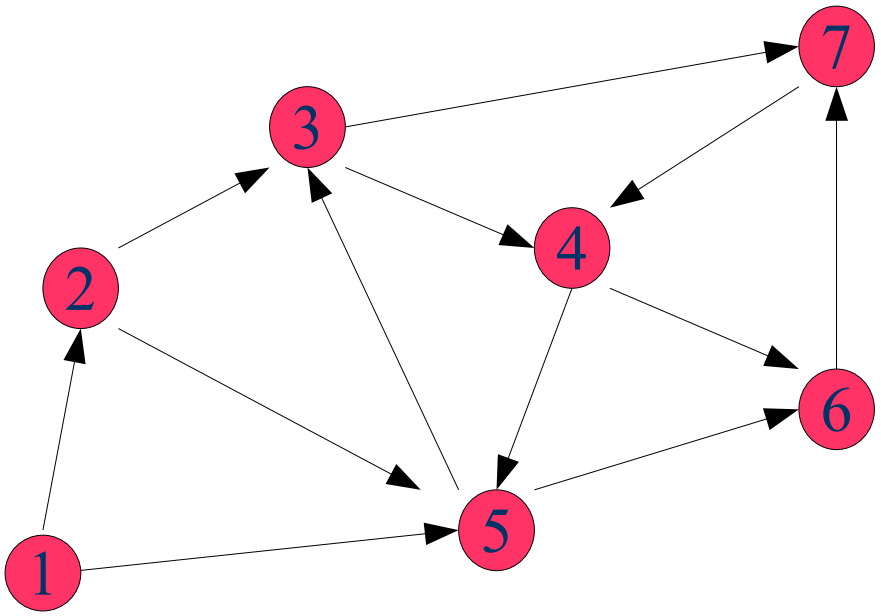
Przykład: BFS

- Kolejka: 6 4 7
- Wyjście: 1 2 5 3



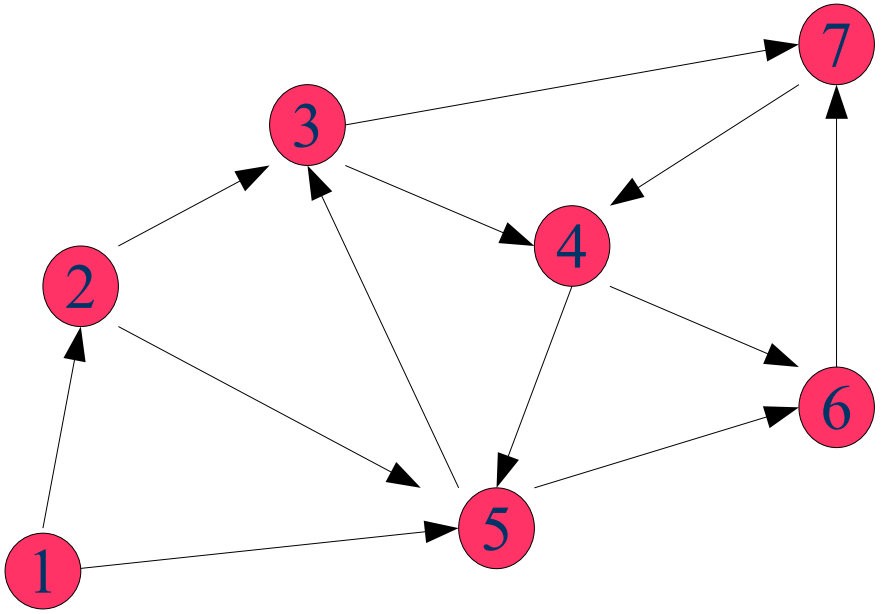
Przykład: BFS

- Kolejka: 4 7
- Wyjście: 1 2 5 3 6



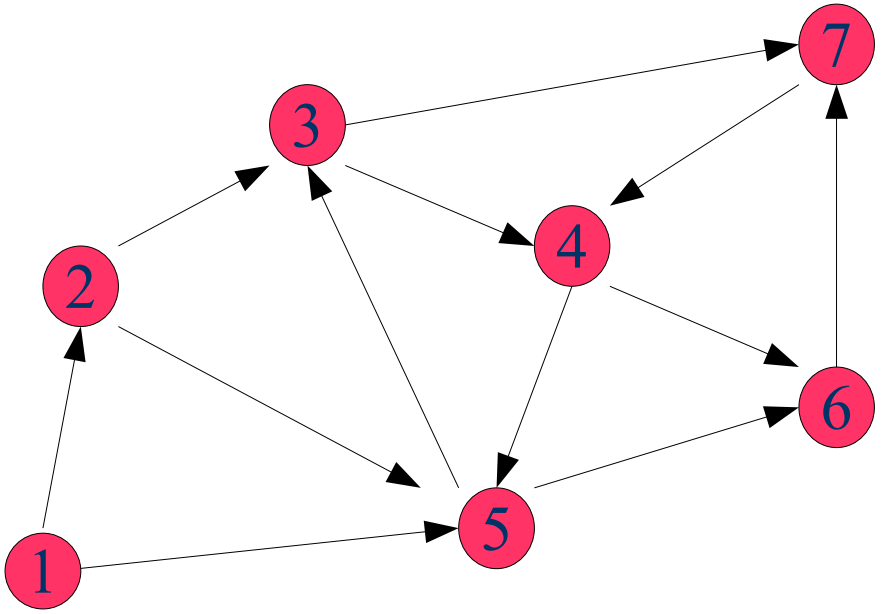
Przykład: BFS

- Kolejka: 7
- Wyjście: 1 2 5 3 6 4



Przykład: BFS

- Kolejka:
- Wyjście: 1 2 5 3 6 4 7



Cecha przeszukiwania wszerz

- Przy przeszukiwaniu wszerz węzły są odwiedzane w kolejności zgodnej z odległością od źródła (liczoną liczbą krawędzi).
- Dowód (indukcja) tezy, że w momencie wstawienia do kolejki węzła v wszystkie węzły bliższe niż v znajdują się w kolejce lub zostały już przetworzone:
  - Pierwszym węzłem przetworzonym jest źródło.
  - Aby węzeł v o numerze niebędący źródłem mógł być przetworzony, musi być wstawiony do kolejki przez inny węzeł v', o odległości mniejszej o 1, który właśnie został pobrany z kolejki. Na mocy założenia indukcyjnego nie było w niej węzłów bliższych niż v' – zostały już przetworzone.

Antyczne zadanie

- Sprzedawca wina ma trzy amfory o pojemności 3,5 i 8 litrów. Największa z nich jest wypełniona winem, pozostałe dwie są puste. Chciałby odlać z największej amfory do średniej 4 litry. Jak to można zrobić, przy założeniu że nie ma litrowej miarki, a to co pozostaje to wykonywanie przelewań dwóch typów:
  - Nalanie wina z jednej amfory do drugiej do pełna
  - Przelanie całej zawartości jednej amfory do drugiej

Rozwiązanie:

- Konstruujemy graf konfiguracji (zawartości wina w 3 amforach). Źródłem jest konfiguracja (8,0,0).
- Konfiguracje dozwolone, to takie trójki (x,y,z). że
  - x+y+z=8
  - xyz(x-8)(y-5)(z-3)=0
- Konfiguracja (x,y,z) jest połączona krawędzią z (x',y',z') jeśli z (x,y,z) można otrzymać (x',y',z') za pomocą jednego przelania.
- Chodzi o znalezienie najkrótszej ścieżki w tym grafie łączącej źródło z węzłem (4,4,0).

Zadanie o katastrofach lotniczych

- Mamy chronologiczne dane z katastrof lotniczych, w których kolejno są zapisane daty i liczby ofiar. Naszym zadaniem jest określić dla każdej katastrofy, od ilu lat nie było większej.
- Przykład:

```
47 48 50 50 51 52 53 55 56 57 57 58 60
 8  3  6  4  7  2  4  1  3  9  4  5  2
```

Zadanie o katastrofach lotniczych

- Mamy chronologiczne dane z katastrof lotniczych. Dane zawierają kolejno daty i liczby ofiar. Naszym zadaniem jest określić dla każdej katastrofy, od ilu lat nie było większej.
- Przykład:

47	48	50	50	51	52	53	55	56	57	57	58	60
8	3	6	4	7	2	4	1	3	9	4	5	2
-1	1	3	0	4	1	2	2	3	-1	0	1	2

Parę obserwacji

- Po pierwsze zauważmy, że przeglądając dane kolejno od nastarszej musimy odwoływać się do wcześniej napotkanych, ale z zastrzeżeniem, że jeśli jakaś katastrofa X jest wcześniejsza i ma mniejszą liczbę ofiar niż Y, to do X już nigdy później niż Y się nie odwołamy.
- Po drugie z danego roku warto pamiętać tylko największą katastrofę, nawet jeśli po niej tego samego roku zdarzyła się mniejsza.
- Z tych dwóch uwag wynika, że będą nam potrzebne informacje o katastrofach o malejących liczbach ofiar i rosnących latach. Umieścimy je na stosie.

Typy danych

```
type katastrofa = record
        lofiar,
        rok,
        odkiedy : Integer;
     end;
typ = katastrofa; {typ elementu listy}
var l:lista; {katastrof}
    s:stos; {katastrof}
    d:katastrofa; {rekord roboczy}
```

- Zkładamy, że dane w liście `l` mają wypełnione pola `lofiar` i `rok`, a mamy uzupełnić pole `odkiedy`.

Rozwiązanie zadania o katastrofach lotniczych

```
MakeEmpty(s);
d.lofiar:= ∞; Push(s,d); {strażnik}
while l<>nil do begin
 repeat  Pop(s,d)
 until d.lofiar > l^.w.lofiar;
 if d.lofiar = ∞ then l^.w.odkiedy:=-1
 else l^.w.odkiedy:=d^.w.rok - l^.w.rok;
 Push(s,d);
 if d^.w.rok<l^.w.rok then Push(s,l^.w);
 l:=l^.nast
end
```

Wizualizacja rozwiązania

47	48	50	50	51	52	53	55	56	57	57	58	60
8	3	6	4	7	2	4	1	3	9	4	5	2

∞

Wizualizacja rozwiązania

47 48 50 50 51 52 53 55 56 57 57 58 60

8 3 6 4 7 2 4 1 3 9 4 5 2

-1

(8,47)

(∞,?)

Wizualizacja rozwiązania

47 48 50 50 51 52 53 55 56 57 57 58 60

8 3 6 4 7 2 4 1 3 9 4 5 2

-1 1

(3,48)

(8,47)

(∞,?)

Wizualizacja rozwiązania

```
47  48  50  50  51  52  53  55  56  57  57  58  60
 8   3   6   4   7   2   4   1   3   9   4   5   2
-1   1   3
```

(6,50)

(8,47)

(∞,?)

Wizualizacja rozwiązania

```
47  48  50  50  51  52  53  55  56  57  57  58  60
 8   3   6   4   7   2   4   1   3   9   4   5   2
-1   1   3   0
```

(6,50)

(8,47)

(∞,?)

Wizualizacja rozwiązania

47	48	50	50	51	52	53	55	56	57	57	58	60
8	3	6	4	7	2	4	1	3	9	4	5	2
-1	1	3	0	4								

(7,51)

(8,47)

(∞,?)

Wizualizacja rozwiązania

47 48 50 50 51 52 53 55 56 57 57 58 60

8 3 6 4 7 2 4 1 3 9 4 5 2

-1 1 3 0 4 1

(2,52)

(7,51)

(8,47)

(∞,?)

Wizualizacja rozwiązania

47	48	50	50	51	52	53	55	56	57	57	58	60
8	3	6	4	7	2	4	1	3	9	4	5	2
-1	1	3	0	4	1	2						

(4,54)

(7,51)

(8,47)

(∞,?)

Wizualizacja rozwiązania

47	48	50	50	51	52	53	55	56	57	57	58	60
8	3	6	4	7	2	4	1	3	9	4	5	2
-1	1	3	0	4	1	2	2					

(1,55)
(4,54)
(7,51)
(8,47)
(∞,?)

Wizualizacja rozwiązania

47	48	50	50	51	52	53	55	56	57	57	58	60
8	3	6	4	7	2	4	1	3	9	4	5	2
-1	1	3	0	4	1	2	2	3				

(3,56)
(4,54)
(7,51)
(8,47)
(∞,?)

Wizualizacja rozwiązania

47	48	50	50	51	52	53	55	56	57	57	58	60
8	3	6	4	7	2	4	1	3	9	4	5	2
-1	1	3	0	4	1	2	2	3	-1			

(9,57)

(∞,?)

Wizualizacja rozwiązania

47	48	50	50	51	52	53	55	56	57	57	58	60
8	3	6	4	7	2	4	1	3	9	4	5	2
-1	1	3	0	4	1	2	2	3	-1	0		

(9,57)

(∞,?)

Wizualizacja rozwiązania

47	48	50	50	51	52	53	55	56	57	57	58	60
8	3	6	4	7	2	4	1	3	9	4	5	2
-1	1	3	0	4	1	2	2	3	-1	0	1	

(5,58)

(9,57)

(∞,?)

Wizualizacja rozwiązania

47	48	50	50	51	52	53	55	56	57	57	58	60
8	3	6	4	7	2	4	1	3	9	4	5	2
-1	1	3	0	4	1	2	2	3	-1	0	1	2

(2,60)
(5,58)
(9,57)
(∞,?)

Analiza złożoności

- Na pierwszy rzut oka wygląda na to, że nasz algorytm ma złożoność kwadratową. Mamy bowiem w pętli while o liczbie obrotów O(n) wewnętrzną pętlę repeat o liczbie obrotów pwsymistycznie O(n). Wiemy przecież, że O(n)*O(n) jest $O(n^2)$. Pokażemy, że wbrew pierwszej intuicji złożoność całego algorytmu jest O(n).
- Dowód jest banalny. Zauważmy, że operacją dominującą jest instruksja Pop(s,d) wewnątrz pętli repeat. Ale ile razy można zdjąć coś ze stosu? Na pewno nie więcej, niż się włożyło, a łącznie operacji Push wykonaliśmy co najwyyzej 2n.

Analiza złożoności pamięciowej

- Z pamięcią jest nawet nieco lepiej. Ponieważ lata elementów na stosie maleją w kierunku dna, więc nigdy rozmiar stosu nie przekroczy zakresu dat.

Zadanie o optymalnym dodawaniu

- Z wykładu o arytmetyce liczb rzeczywistych pamiętamy zadanie o optymalnym (najdokładniejszym) dodawaniu n liczb. Załóżmy, że oryginalne dane są posortowane.
- Algorytm optymalnego dodawania może stać się liniowy, jesli użyjemy stosu i kolejki! Wkładamy na stos malejąco do góry wszystkie elementy. Naszym zadaniem jest pobranie dwóch najmniejszych elementów, dodanie ich i włożenie z powrotem sumy do zbioru argumentów.
- Uzyskane sumy wkładamy do kolejki. Bedą one rosnące.
- Zawsze dwa najmniejsze elementy będą się znajdowały pośród czterech: 2 na szczycie stosu i 2 na początku kolejki! (lub mniej)