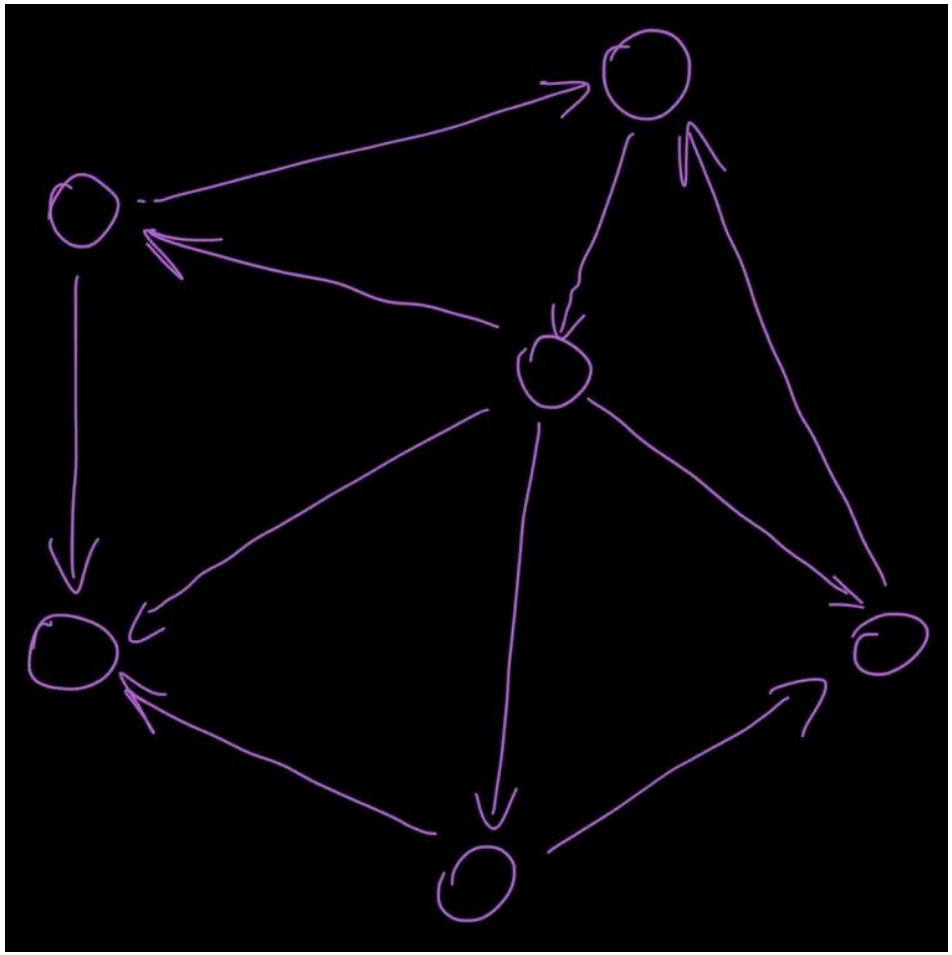
Algorytmika Praktyczna. Grafy.

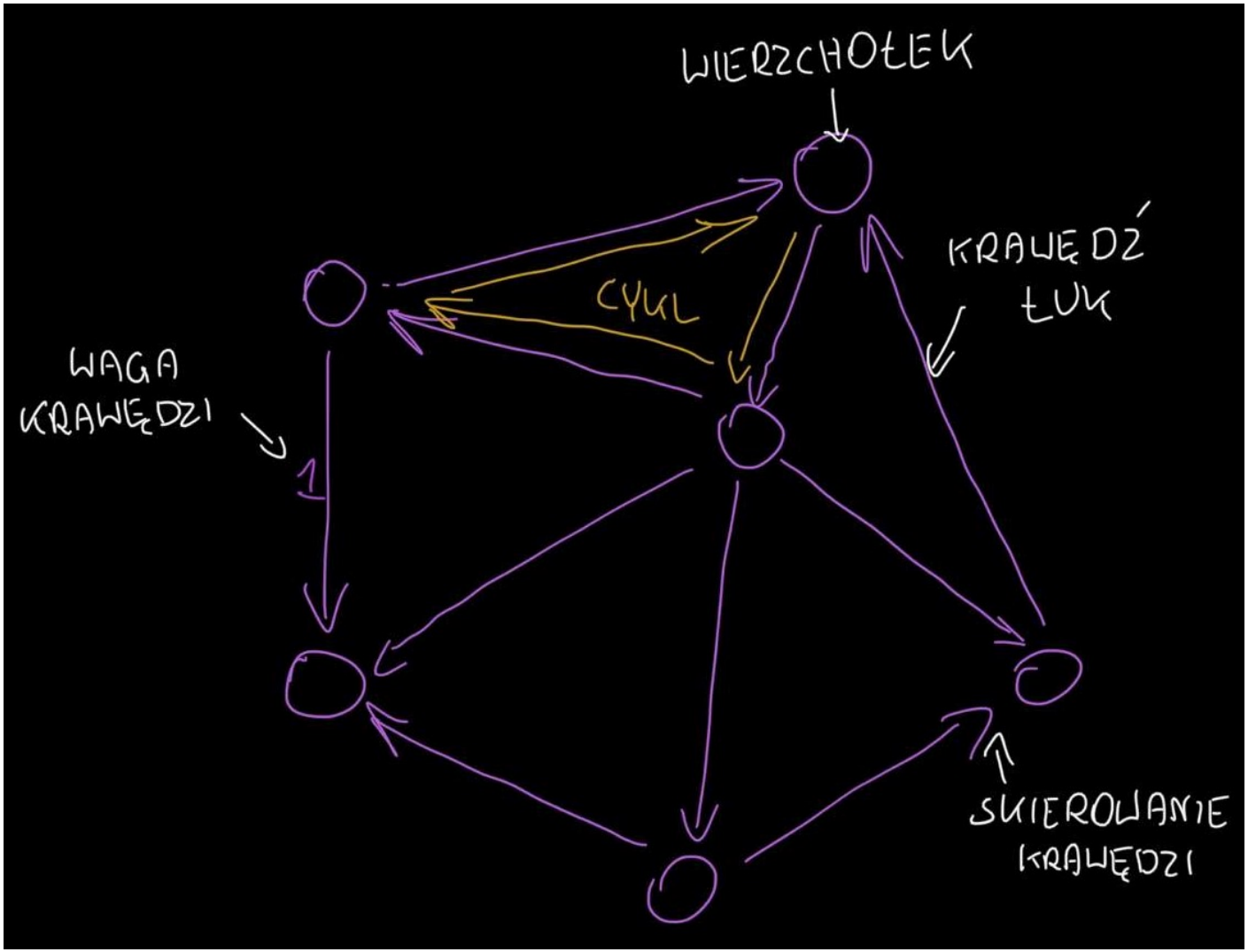
Artur Laskowski

17 marca 2022, Poznań

Graf

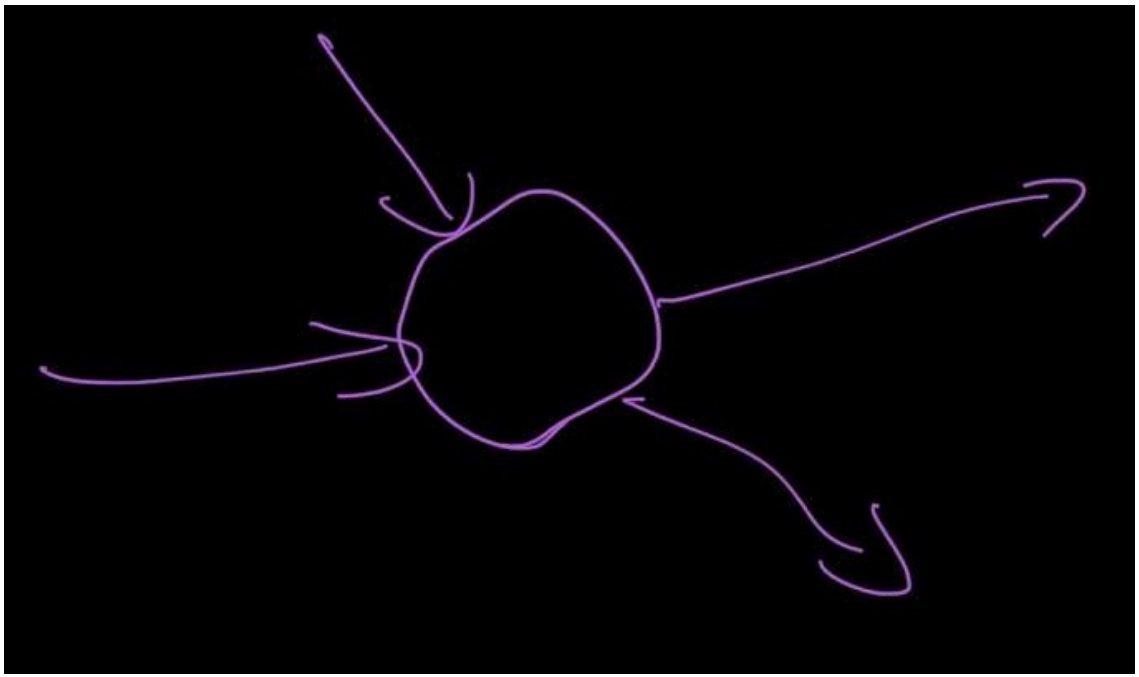


Graf


WIERZCHOŁEK
KRAWĘDŹ
ŁUK
CYKL
WAGA
KRAWĘDZI
1
SKIEROWANIE
KRAWĘDZI

Stopień wierzchołka

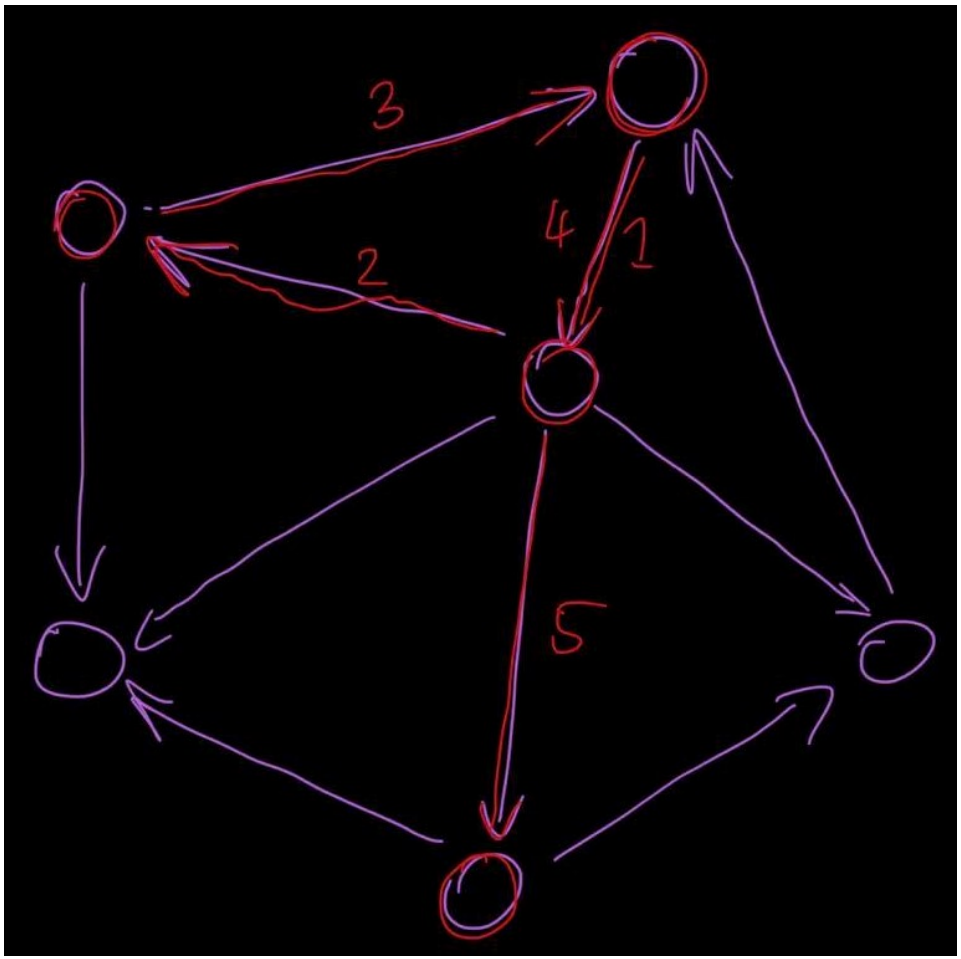
Stopień wejściowy
Stopień wyjściowy



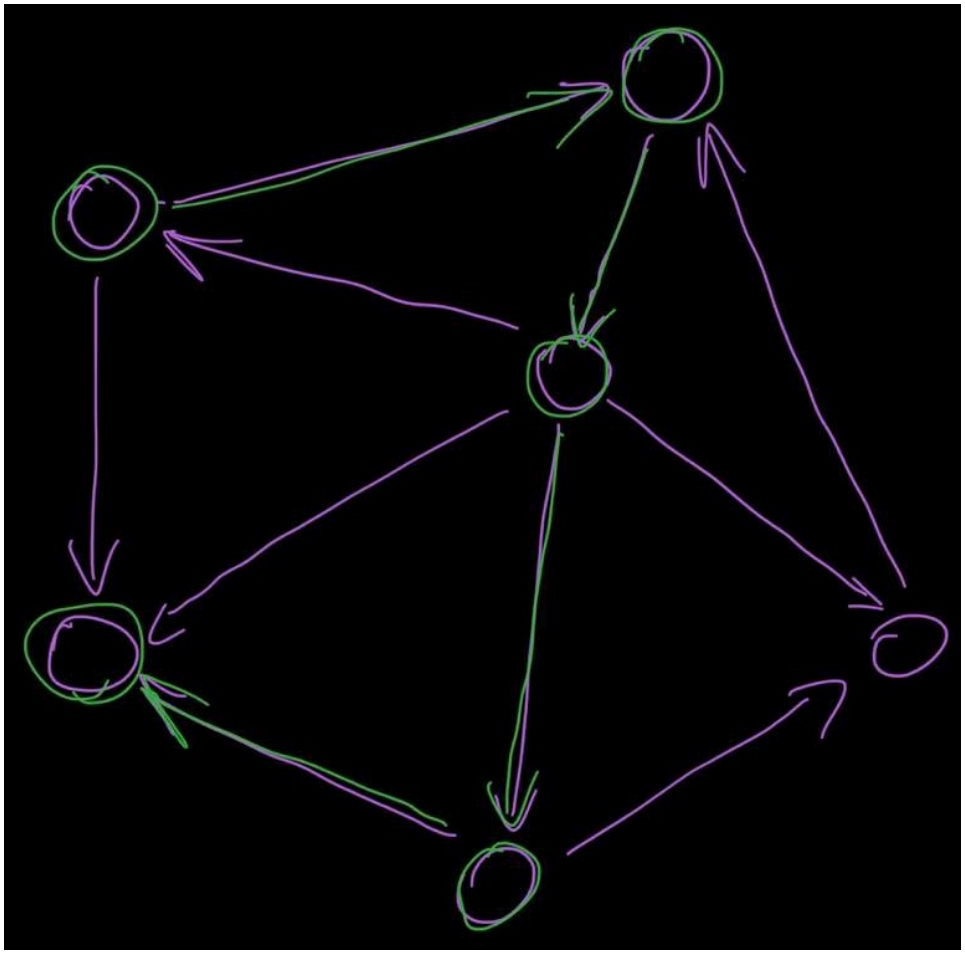
Ścieżki

Ścieżka	ciąg przejść przez kolejne wierzchołki wzdłuż krawędzi.
Ścieżka prosta	każdy wierzchołek występuje na niej najwyżej raz.

Ścieżki



Ścieżki

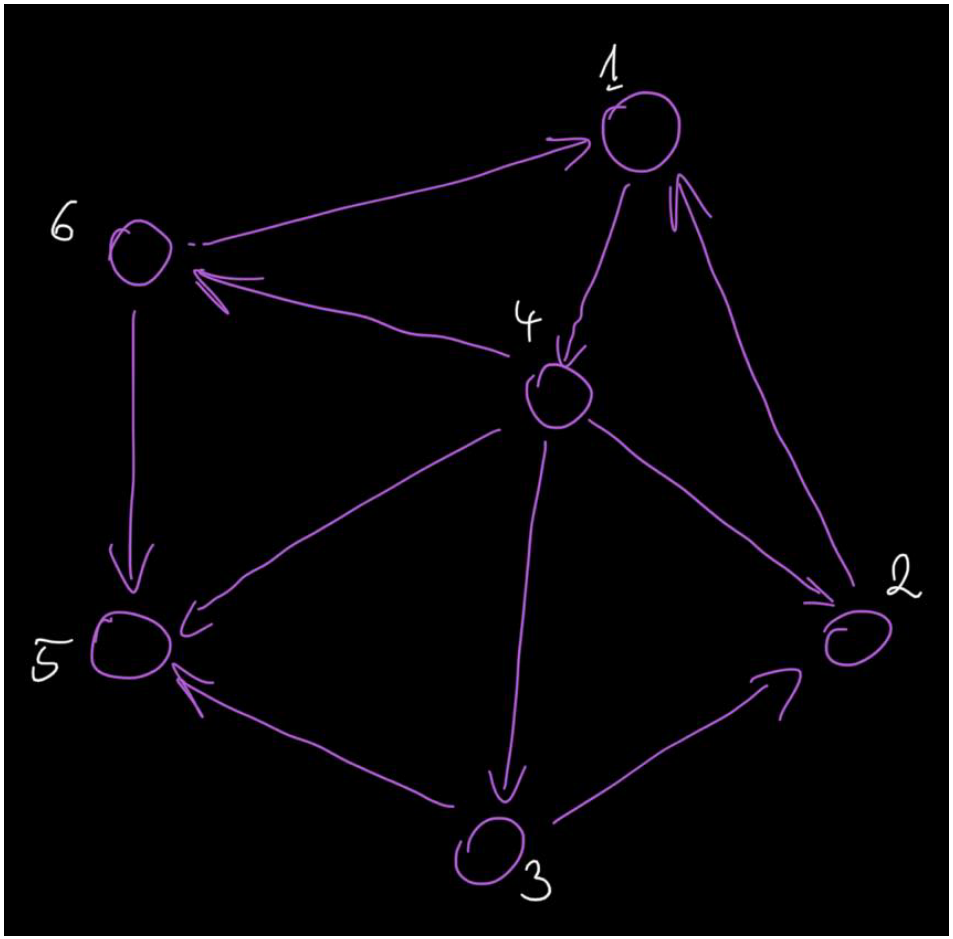


Gęstość grafu

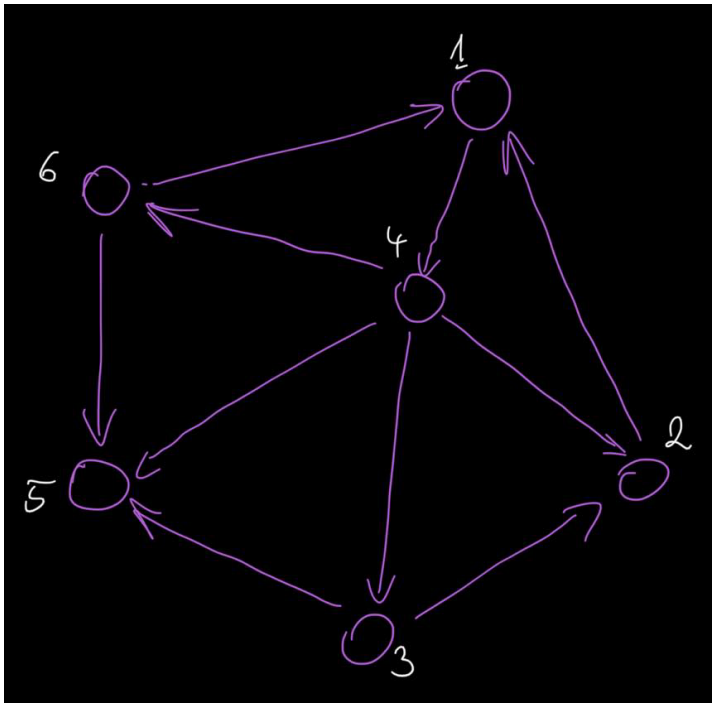
Grafy gęste: $E \approx V^2$

Grafy rzadkie: $E \approx V$

Reprezentacja grafu - macierz sąsiedztwa



Reprezentacja grafu - macierz sąsiedztwa



	1	2	3	4	5	6
1				true		
2	true					
3		true			true	
4		true	true		true	true
5						
6	true				true	

Reprezentacja grafu - macierz sąsiedztwa

```cpp
#include <iostream>

#define MAXN 1'000

using namespace std;

bool graph[MAXN][MAXN];

int main() {
    graph[2][3] = true;     // dodanie krawędzi skierowanej z wierzchołka 2 do wierzchołka 3
    graph[5][1] = true;     // dodanie krawędzi skierowanej z wierzchołka 5 do wierzchołka 1


    int x = 4;
    for(int i = 0; i < MAXN; ++i) {
        if(graph[x][i]) {
            continue;       // istnieje krawędź od x do i
        }
    }

    return 0;
}
```

Reprezentacja grafu - macierz sąsiedztwa

Zalety:

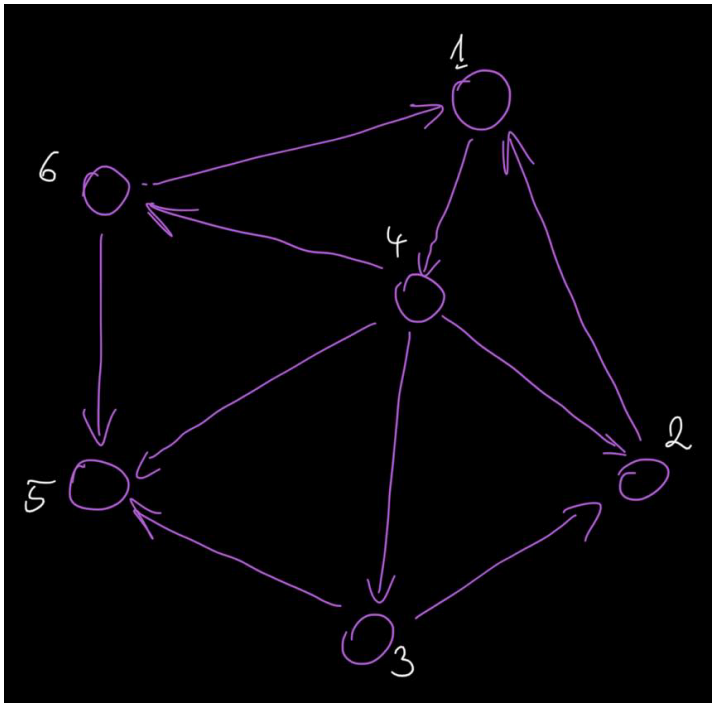
- Szybkie sprawdzenie, czy dana krawędź istnieje $O(1)$
- Prosta implementacja

Wady:

- Wolny przegląd wszystkich krawędzi $O(V^2)$
- Wolny przegląd następników/poprzedników $O(V)$
- Duża złożoność pamięciowa $O(V^2)$

Dla grafów gęstych wady mało istotne

Reprezentacja grafu - lista sąsiedztwa



1	{4}
2	{1}
3	{2, 5}
4	{2, 3, 5, 6}
5	{}
6	{1, 5}

Reprezentacja grafu - lista sąsiedztwa

```cpp
#include <iostream>
#include <vector>

#define MAXN 1'000

using namespace std;

vector<int> graph[MAXN];

int main() {
    graph[2].push_back(3);  // dodanie krawędzi skierowanej z wierzchołka 2 do wierzchołka 3
    graph[5].push_back(1);  // dodanie krawędzi skierowanej z wierzchołka 5 do wierzchołka 1


    int x = 4;
    for(int i = 0; i < graph[x].size(); ++i) {
        if(graph[x][i]) {
            continue;       // istnieje krawędź od x do graph[x][i]
        }
    }

    for(auto el: graph[x]) {
        continue;           // istnieje krawędź od x do el
    }

    return 0;
}
```

Reprezentacja grafu - lista sąsiedztwa

Zalety:

- Szybki przegląd wszystkich krawędzi $O(E)$
- Szybki przegląd następników/poprzedników*, średnio $O(\frac{E}{V})$
- Złożoność pamięciowa $O(E)$

Wady:

- Wolne sprawdzenie, czy dana krawędź istnieje, średnio $O(\frac{E}{V})$

Reprezentacja grafu - krawędzie z wagami

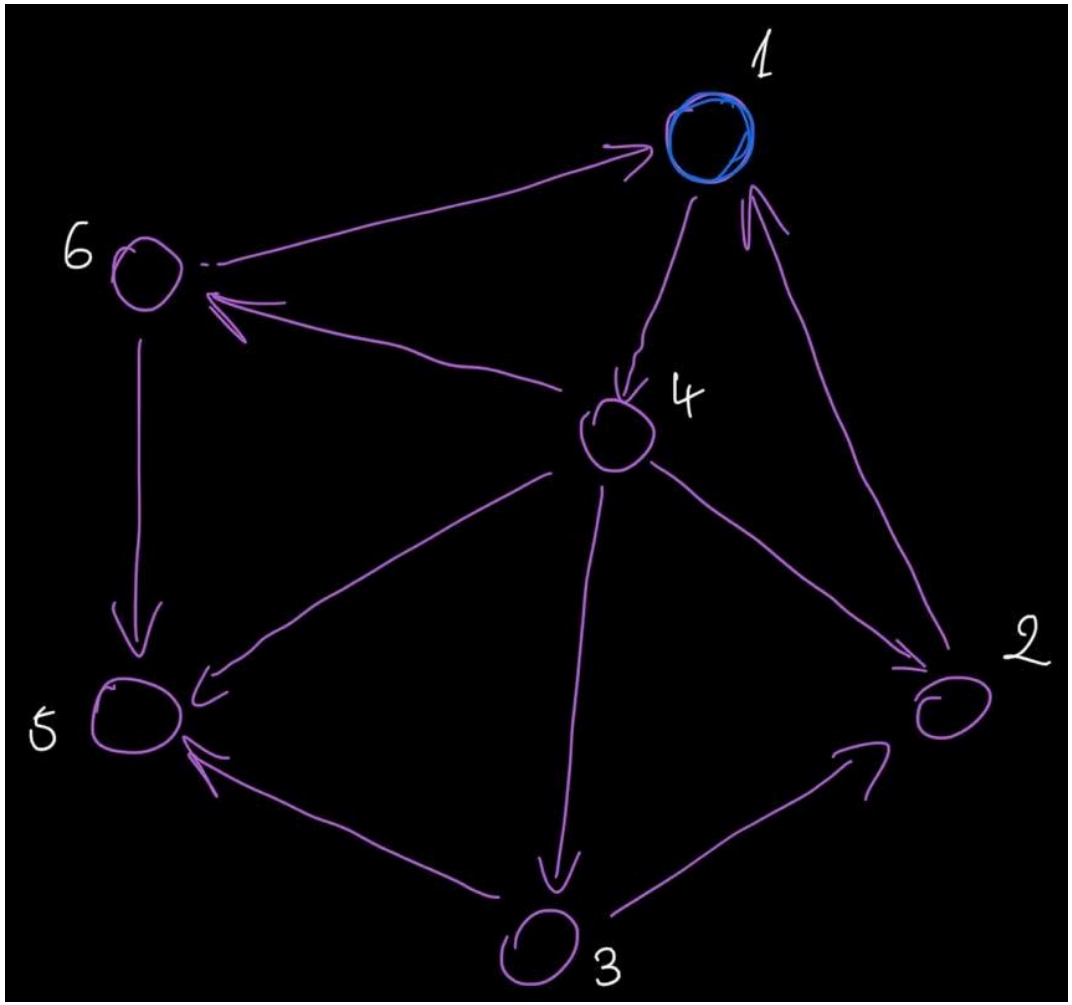
```
pair<bool, int> g[MAXN][MAXN];

vector<pair<int, int> > g[MAXN];
```

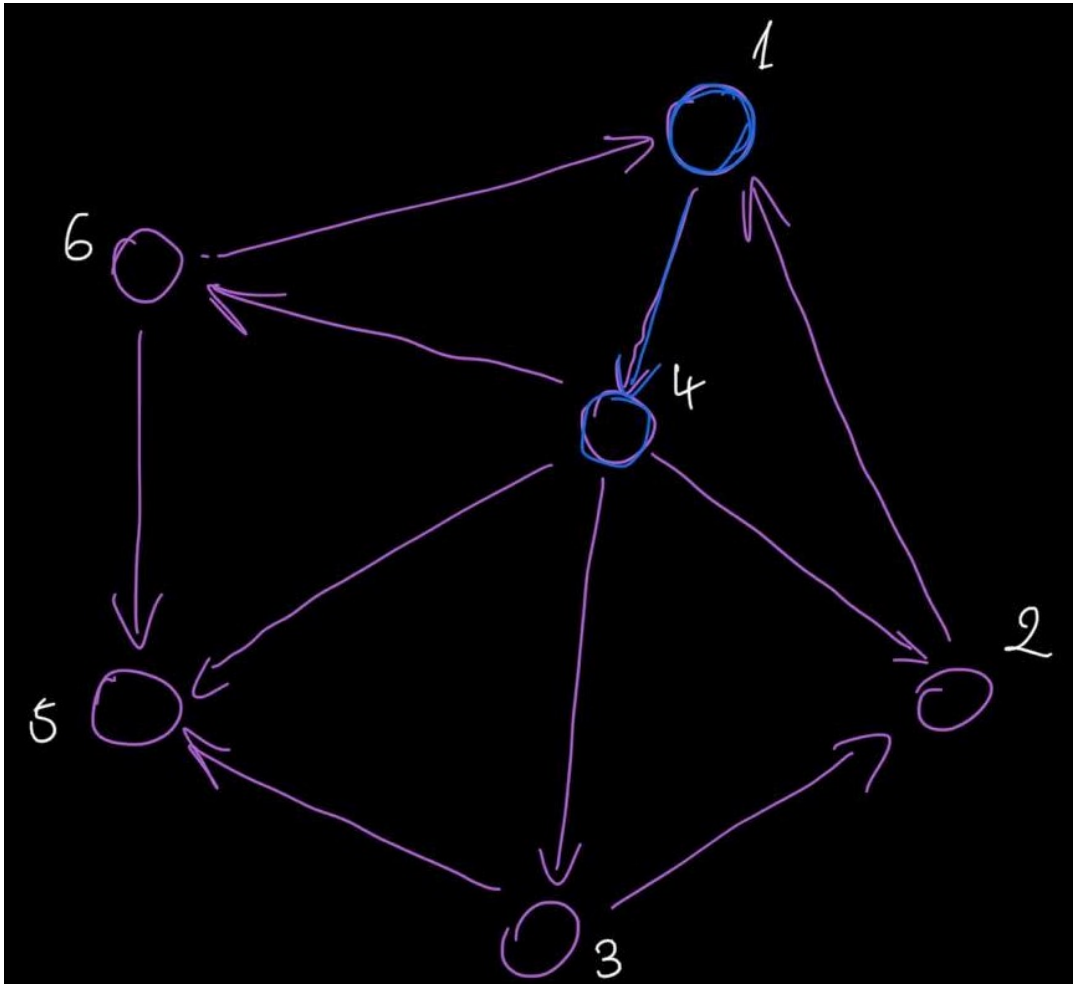
Algorytm DFS

- Depth-first search - przeszukiwanie "w głąb"
- Podobne do przeszukiwania drzew
- Generuje numerację pre- praz post-order
- Oznaczamy wierzchołki jako odwiedzone (ang. visited)
- Wyznaczamy drzewo - tzw. "Drzewo DFS"
- Nie zawsze jedna iteracja musi wystarczyć

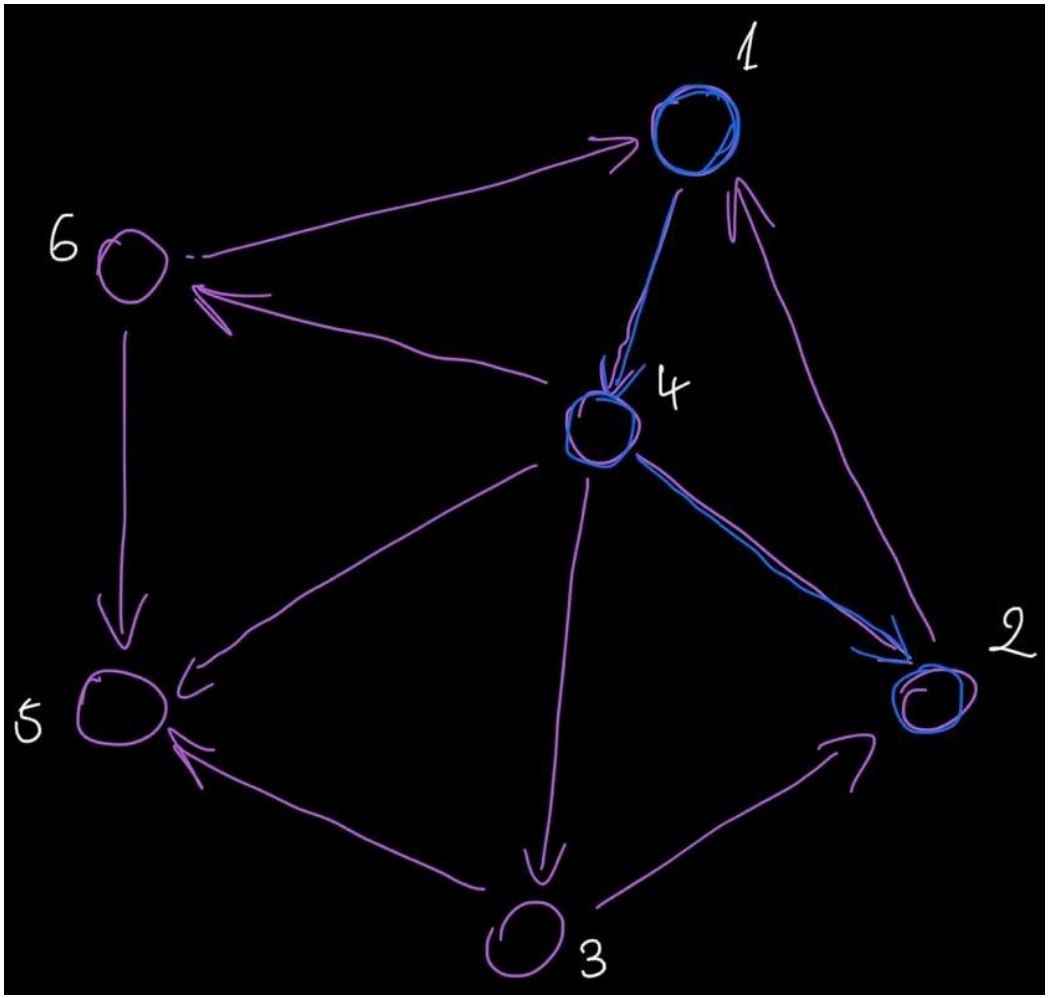
Algorytm DFS



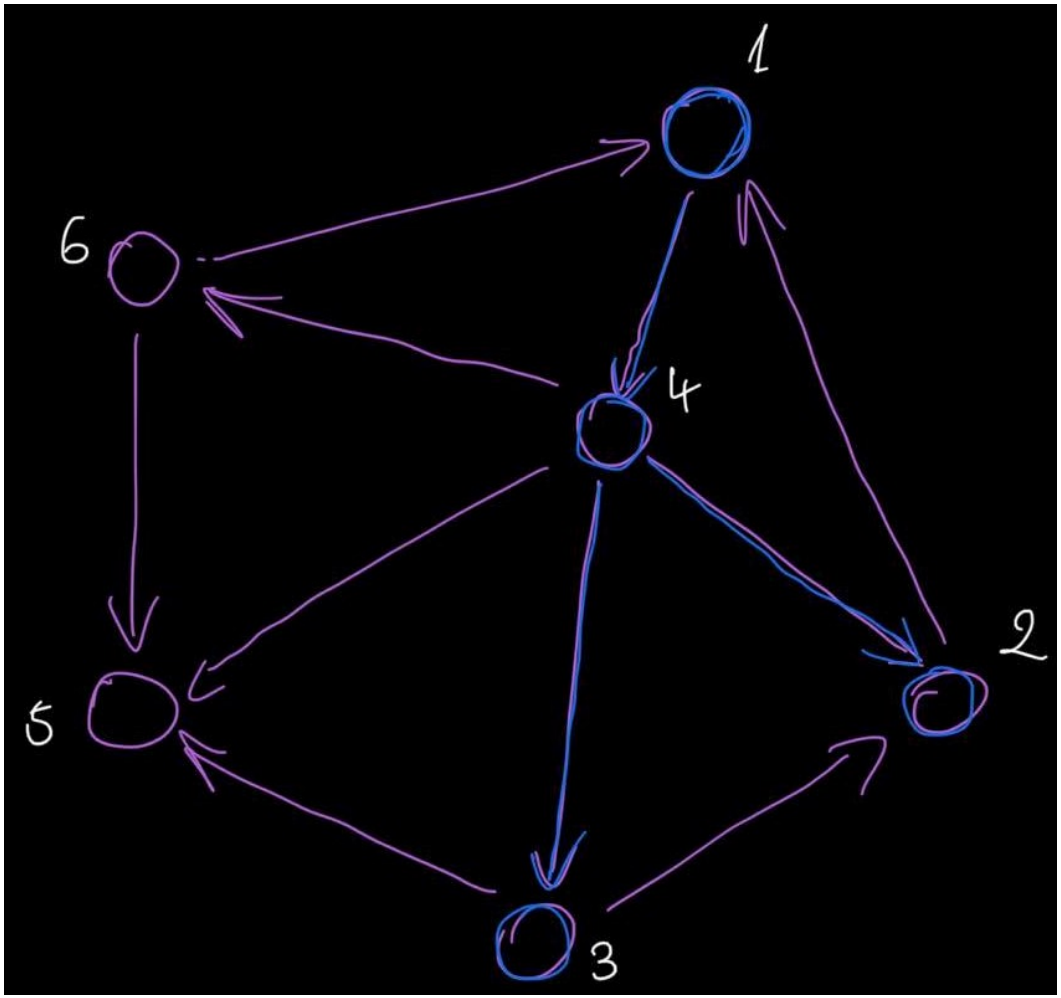
Algorytm DFS



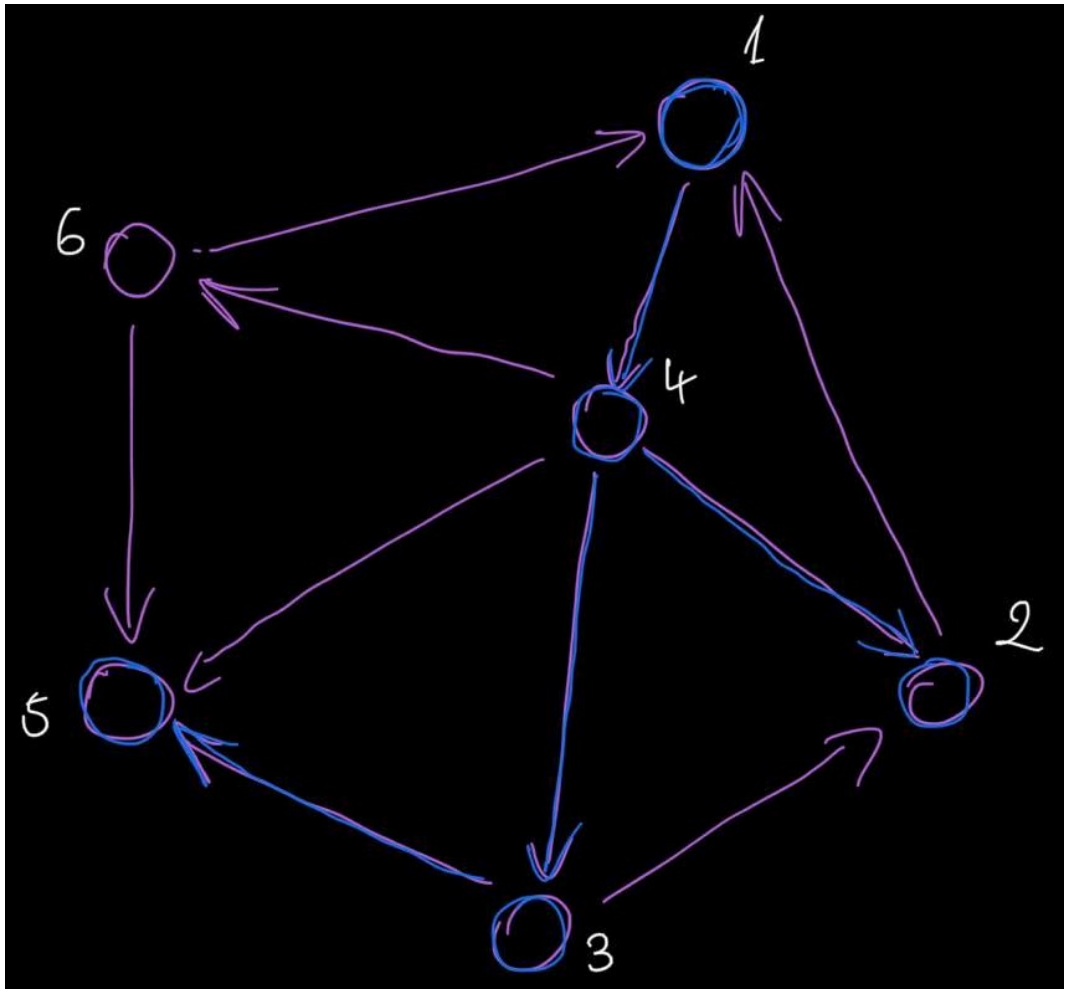
Algorytm DFS



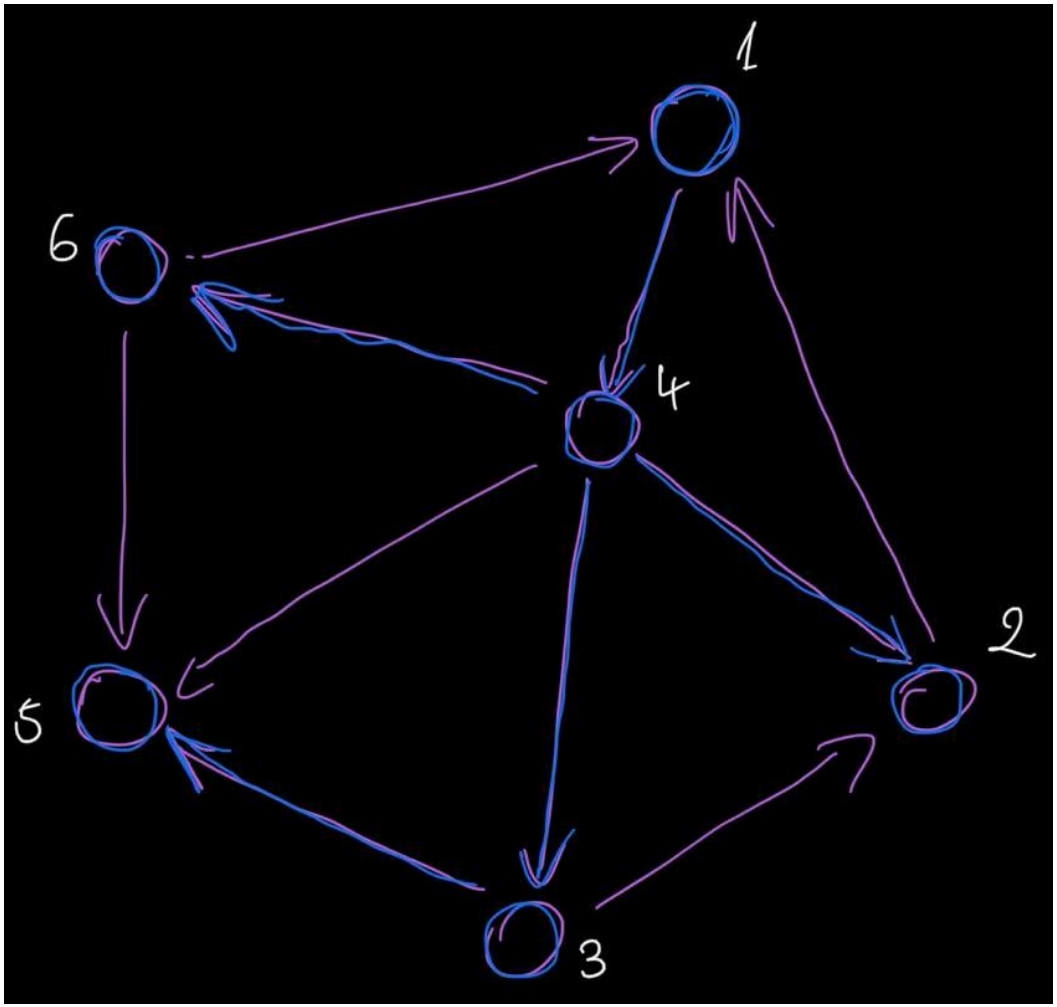
Algorytm DFS



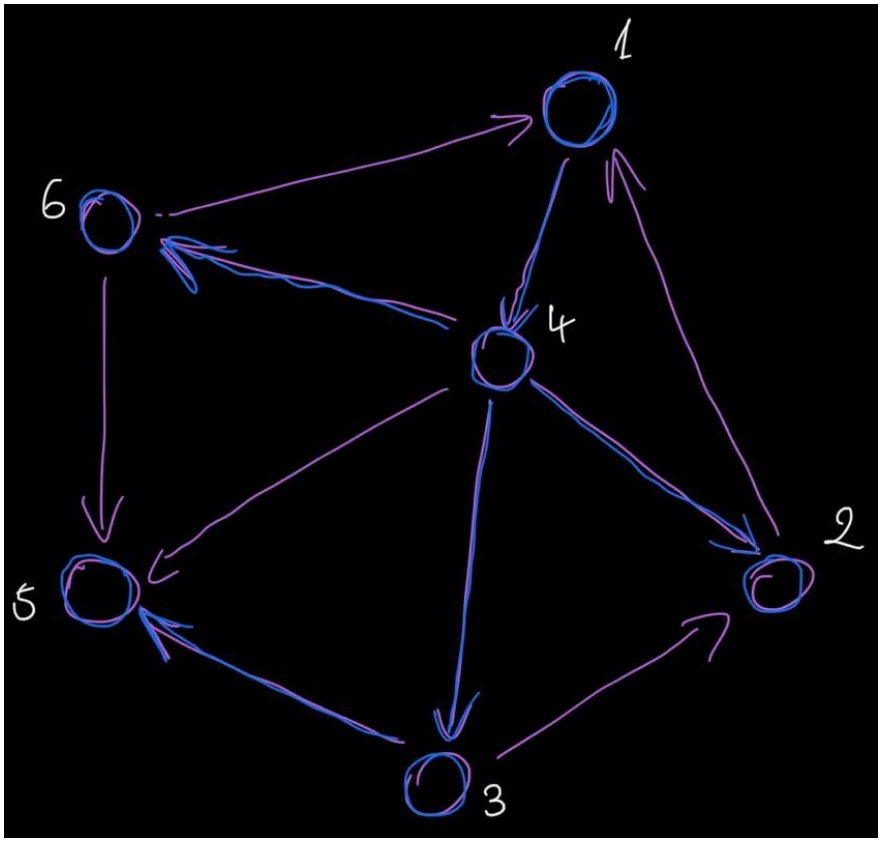
Algorytm DFS

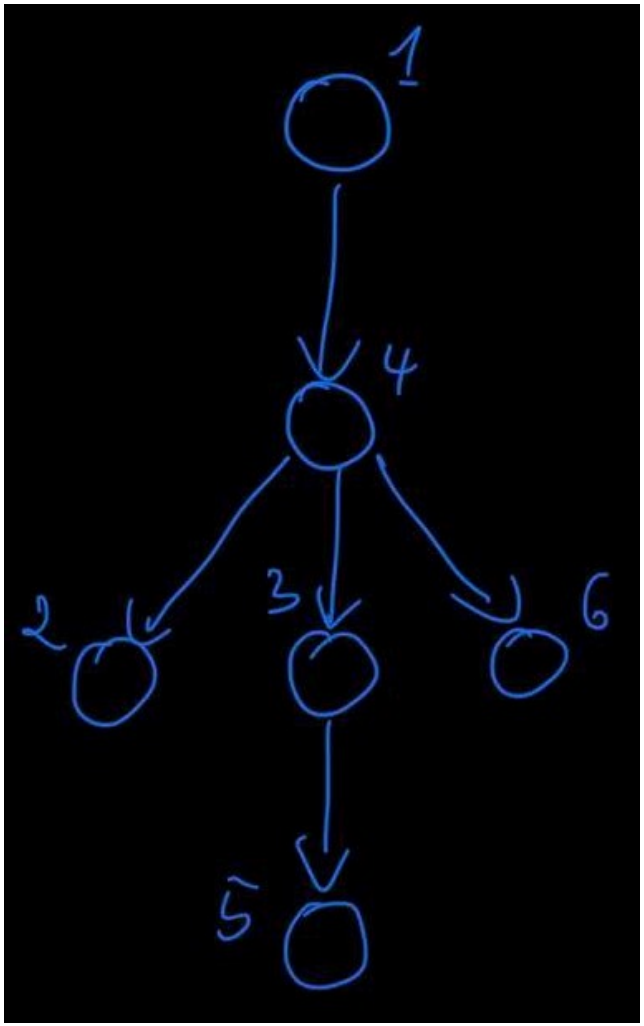


Algorytm DFS



Algorytm DFS - drzewo DFS





Algorytm DFS - implementacja

```
bool vis[MAXN];
for(int i = 0; i < n; ++i) {
    vis[i] = false;
}

dfs(1);
```

Algorytm DFS - implementacja

```
36  void dfs(int x) {
37      vis[x] = true;
38      for(int i = 0; i < graph[x].size(); ++i) {
39          int neighbour = graph[x][i];
40          if(vis[neighbour] == false) {
41              dfs(neighbour);
42          }
43      }
44  }
```

Algorytm DFS - implementacja

```
36  void dfs(int x) {
37      // wypisując tutaj mamy pre-order
38      vis[x] = true;
39      for(int i = 0; i < graph[x].size(); ++i) {
40          int neighbour = graph[x][i];
41          if(vis[neighbour] == false) {
42              dfs(neighbour);
43          }
44      }
45      // wypisując tutaj mamy post-order
46  }
```

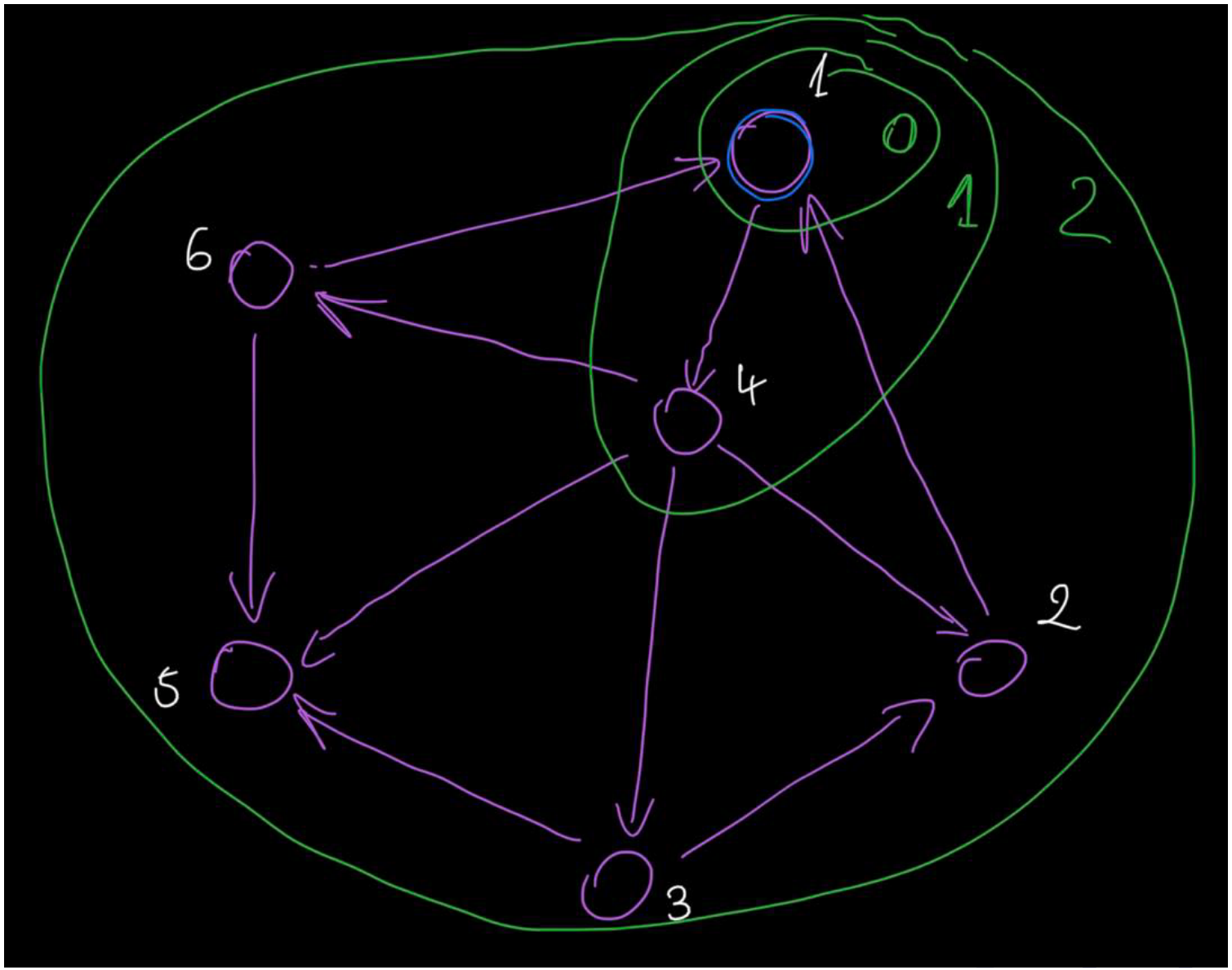
Algorytm BFS

- Breadth-first search - przeszukiwanie "wszerz"
- Wyznaczanie odległości od pewnego wierzchołka
- Wierzchołki rozpatrywane w kolejności powyższej odległości

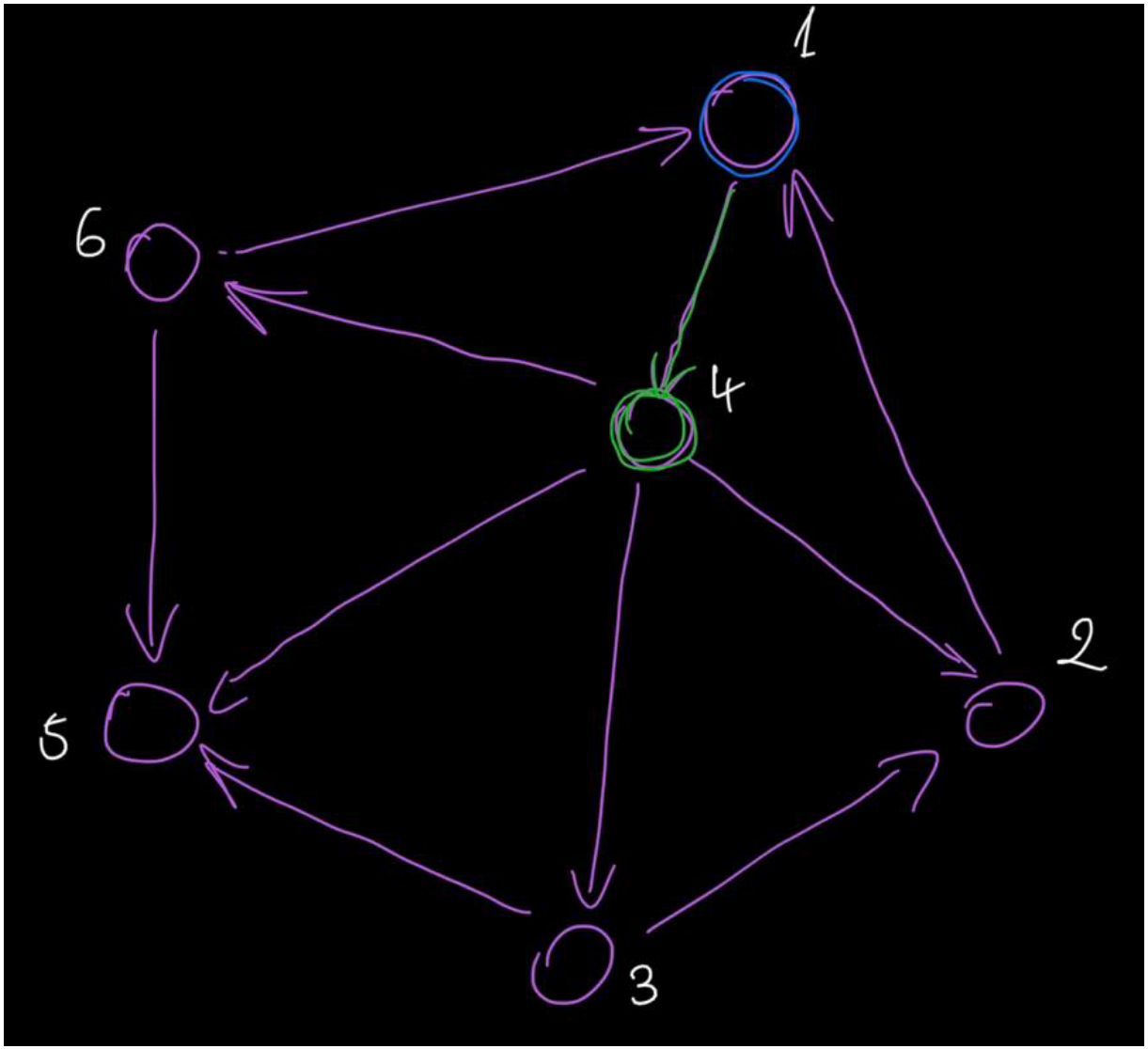
Algorytm BFS - implementacja

```
48  void bfs(int x) {
49      queue<int> q;
50      vis[x] = true;
51      q.push_back(x);
52      while(!q.empty()) {
53          int curr = q.front();
54          q.pop();
55          for(int i = 0; i < graph[curr].size(); ++i) {
56              int neighbour = graph[curr][i];
57              if(!vis[neighbour]) {
58                  vis[neighbour] = true;
59                  q.push(neighbour);
60              }
61          }
62      }
63  }
```

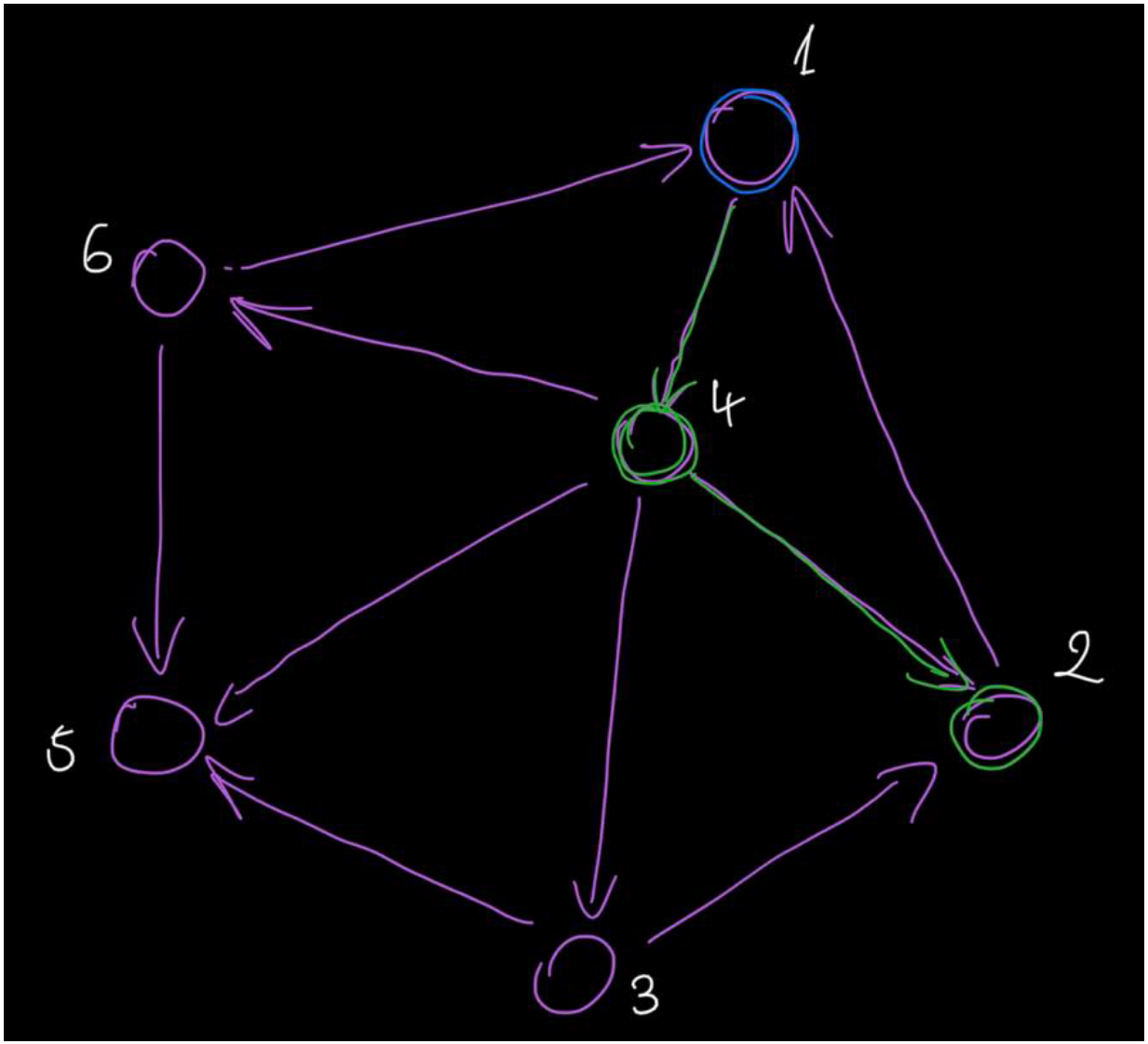
Algorytm BFS - odległości



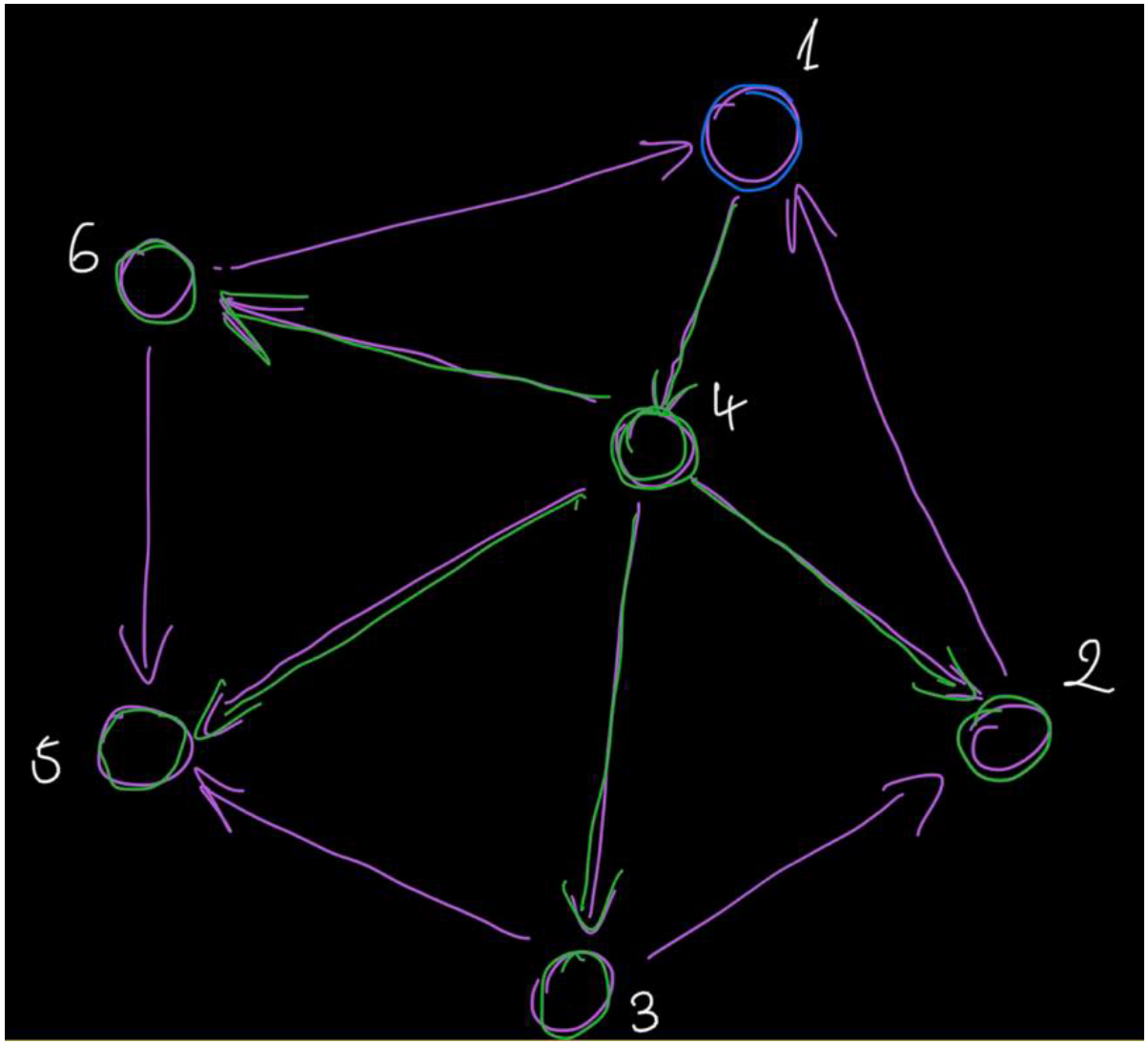
Algorytm BFS



Algorytm BFS



Algorytm BFS



Laboratoria

https://www.hackerrank.com/ap-02-2022