

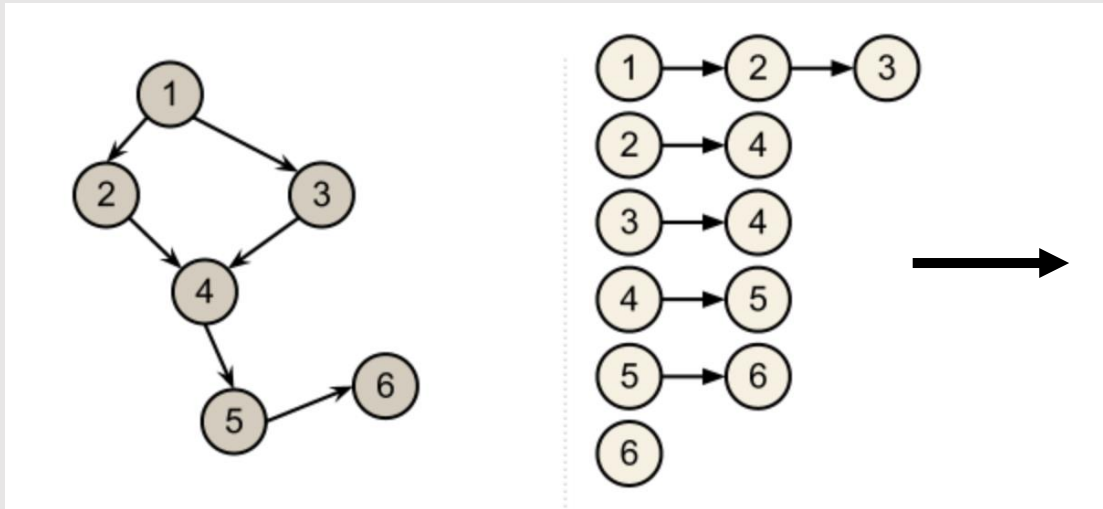


# ▼ Алгоритм пошуку в ширину

Маслова Надія, ФВЕ 1 к. маг.

# Представлення графів

- Існує 2 способи представити граф  $G = (V, E)$  – як набір списків суміжних вершин або як матрицю суміжності
- 1. Для кожної вершини  $u \in V$  список суміжних вершин містить в собі (вказівники на) всі суміжні з нею вершини ( $(u, v) \in E$ )
- 2. Матриця  $A = (a_{ij})$  розміром  $|V| \times |V|$ , для якої  $a_{ij} = \begin{cases} 1, \text{ if } (i, j) \in E, \\ 0 \text{ otherwise} \end{cases}$



	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	1	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	1
6	0	0	0	0	0	0

# Пошук в ширину (Breadth-first search)

- Один з базових алгоритмів, що складає основу багатьох інших (алгоритм Дейкстри, алгоритм Прима можуть розглядатись як його узагальнення).
- Нехай заданий граф  $G = (V, E)$  і фіксована початкова вершина  $s$ . Алгоритм BFS перелічує всі доступні вершини з  $s$ . В процесі пошуку виділяється частина, що називається «деревом пошуку в ширину» з коренем  $s$ . Вона містить всі досяжні вершини з  $s$  і лише їх. Для кожної з них шлях з кореня в дереві пошуку буде одним з найкоротших шляхів (з початкової вершини у графі).

# Пошук в ширину (Breadth-first search)

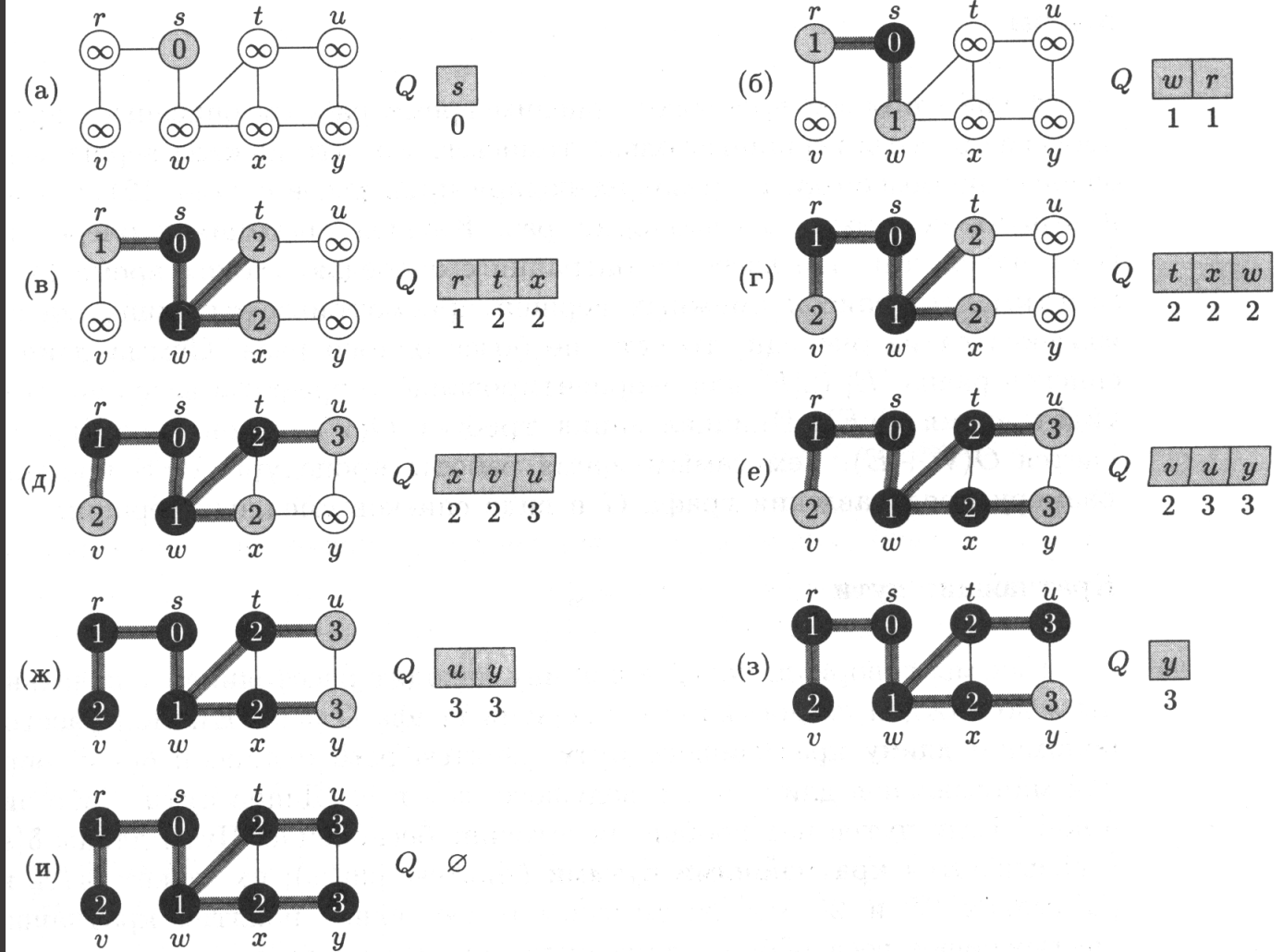
- Будемо вважати, що кожна вершина в процесі роботи алгоритма може мати певний колір: білий, сірий або чорний. На початку всі вершини білі. Біла вершина може стати сірою, а сіра чорною, але не навпаки. Таким чином регулюється порядок обходу: лише сірі вершини можуть мати суміжні незнайдені вершини.
- На початку дерево пошуку складається лише з початкової вершини. Коли зустрічається нова біла вершина  $v$ , суміжна з раніше знайденою вершиною  $u$ , то вершина  $v$  додається в дерево пошуку, стаючи при цьому дочкою вершини  $u$ . Оскільки кожен вузол можна знайти лише один раз, то двох батьків в неї не може бути.
- Для реалізації алгоритму зазвичай використовують чергу (FIFO).

# Псевдокод

BFS( $G, s$ )

```

1  for (для) каждой вершины  $u \in V[G] - \{s\}$ 
2      do  $color[u] \leftarrow$  БЕЛЫЙ
3           $d[u] \leftarrow \infty$ 
4           $\pi[u] \leftarrow \text{NIL}$ 
5   $color[s] \leftarrow$  СЕРЫЙ
6   $d[s] \leftarrow 0$ 
7   $\pi[s] \leftarrow \text{NIL}$ 
8   $Q \leftarrow \{s\}$ 
9  while  $Q \neq \emptyset$ 
10     do  $u \leftarrow head[Q]$ 
11         for (для) всех  $v \in Adj[u]$ 
12             do if  $color[v] =$  БЕЛЫЙ
13                 then  $color[v] \leftarrow$  СЕРЫЙ
14                      $d[v] \leftarrow d[u] + 1$ 
15                      $\pi[v] \leftarrow u$ 
16                     ENQUEUE( $Q, v$ )
17     DEQUEUE( $Q$ )
18      $color[u] \leftarrow$  ЧЁРНЫЙ
    
```



# Час роботи

- Під час роботи вершину можна покласти в чергу лише один раз, відповідно і забрати з черги теж тільки один раз. Кожна операція з чергою займає  $O(1)$ , тоді на всі операції потрібен час  $O(V)$ .
- Список суміжних вершин проглядається не більше одного разу, коли вершина виймається з черги. Сума довжин всіх списків складає  $E$ , всього обробка займає  $O(E)$ .
- Ініціалізація потребує час  $O(V)$ .
- Таким чином час роботи алгоритму складає  $O(V+E)$ , тобто пропорційне розміру представлення графа у вигляді списку суміжних вершин.

# Приклад

```
#include <iostream>
#include <chrono>
using namespace std::chrono;
int main() {
    int N, S, F, a;
    std::cin >> N >> S >> F;
    S--;
    F--;
    int arr[N][N];

    for (int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            std::cin >> a;
            arr[i][j] = a;}}
    auto start = steady_clock::now();
    int color[N];
    int d[N];
    int pi[N];
    for (int i=0; i<N; i++){
```

```
        color[i] = 0;
        d[i]=0;
        pi[i]=-1;
    }
    color[S] = 1;
    d[S] = 0;
    std::queue<int> Q;
    Q.push(S);
    while (!Q.empty()){
        int u = Q.front();
        for (int j=0; j<N; j++){
            if (arr[u][j] == 1){
                if(color[j] == 0){
                    color[j] = 1;
                    d[j] = d[u]+1;
                    pi[j] = u;
                    Q.push(j);}
            }
        }
    }
```

```
        Q.pop();
        color[u] = 2;
    }
    auto finish = steady_clock::now();
    auto duration = finish - start;
    double time =
        duration_cast<microseconds>(duration).count();
    std::cout << "Minimal distance = " << d[F] <<
        std::endl;
    std::cout << "Time = " << time << "
        microseconds" << std::endl;
    return 0;
}
```

# Результат роботи прикладу

```
nadila@nadila-Mi:~/OOP_basic$ ./DFS
```

```
4 4 3
0 1 1 1
1 0 1 0
1 1 0 0
1 0 0 0
```

```
Minimal distance = 2
Time = 3 microseconds
```

```
nadila@nadila-Mi:~/OOP_basic$ ./DFS
```

```
30 1 29
0 0 1 1 0 1 1 1 1 1 1 0 1 1 1 0 0 0 0 1 0 1 0 1 1 0 1 0 0 0
0 0 0 1 1 1 0 1 0 1 0 0 1 1 0 1 1 0 1 0 1 1 1 0 0 1 0 0 1 1
1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 1 1 1 0 1 0 0 1 1 1 0 1 1 0 0
1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 0 0 0 0
0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 1 1 1 0 1 1 0
1 1 1 0 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 0 1 0 1 1
1 0 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 0
1 1 0 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 1 1 0 1 0 1 1 0 0 1 0
1 0 0 0 0 0 1 0 0 1 1 1 1 0 1 0 0 1 0 0 0 1 0 1 0 1 0 1 0 1
1 1 1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 1 0 1 0 1 1 1 0 1 0 1
1 0 1 1 1 1 1 1 1 0 0 1 0 1 1 0 1 0 1 0 0 1 0 0 1 1 1 0 0 0
0 0 1 1 0 1 0 1 1 0 1 0 1 1 0 1 0 1 0 0 0 1 1 1 0 1 0 0 0 1
1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 1 0 1 1 0 0 1 1 0 1 0 1 1 0
1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 1 1 1 0 0 1
1 0 0 1 0 1 1 0 1 1 1 0 1 0 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 1
0 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1
0 1 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 0 1 0 1
0 0 1 1 0 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1 0 1 1 1
0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 1 0 0 0 1 1 1
1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 0 0
0 1 0 1 1 1 1 1 0 0 0 1 0 1 1 1 1 1 0 1 0 1 1 0 1 1 0 0 1 0
1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 1 0 1 1 1 1 1 1
0 1 1 1 0 1 0 1 0 1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 0 0 1 1 1 1
1 0 1 1 1 0 0 0 1 1 1 1 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1
1 0 1 0 1 0 0 1 0 0 0 0 0 1 1 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1
0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 0 1 1 0 1 0 1 1
1 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 0 1 1
0 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 0 0 0 1 1
0 1 0 0 1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 1 1 0 1 1 1 0 0 0
0 1 0 0 0 1 0 0 1 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0
```

```
Minimal distance = 2
Time = 9 microseconds
```