KAZAKH KBTU BRITISH
TECHNICAL
UNIVERSITY

LECTURE 10 – GRAPHS & BFS

Prepared by: Izbassar Assylzhan

Course: Algorithms & Data Structures – Fall 2025

LECTURE 10

KAZAKH KBTU BRITISH TECHNICAL UNIVERSITY

In this lecture, we'll cover the basic notations on graphs: matrix representation, adjacency & edge list. Then, we'll look up how to traverse the given graph with a help of Breadth-First Search (BFS) algorithm.

TOPICS WE'LL COVER:

- Introduction to Graphs
- Matrix representation
- Adjacency & Edge list
- Breadth-First Search
- Practice Problems

GOALS FOR THIS LECTURE:

- Understand how to represent graphs with matrices and lists.
- Implement traversing upon the graph using BFS algorithm.
- Solving some tasks using BFS algorithm.

INTRODUCTION TO GRAPHS

A **graph** is a collection of *vertices* (nodes) and *edges* (connections) between them.

$$G = (V, E) \quad (1)$$

where, V – set of vertices, and E – set of edges (pair of vertices).

Types:

- Directed / Undirected;
- Weighted / Unweighted.

Example:

Vertices = {1, 2, 3, 4, 5, 6};
Edges = {(1, 2), (1, 3), (2, 4), (2, 6), (3, 4), (3, 5)}

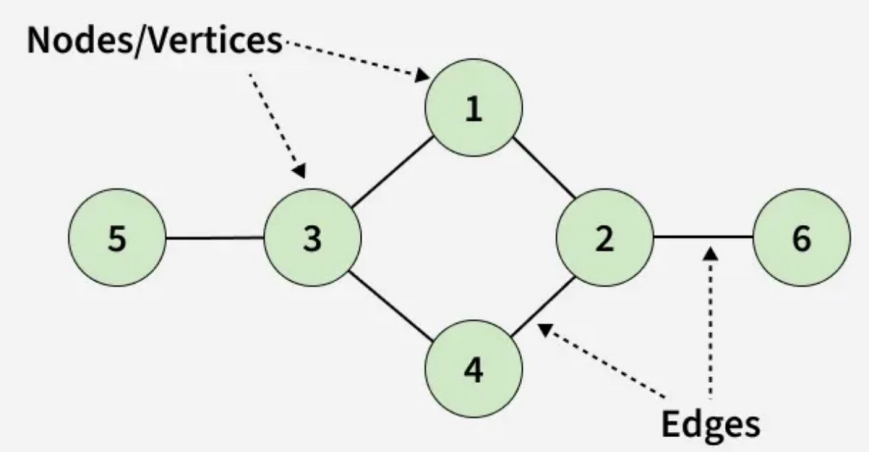


Figure 1 – Example of unweighted, undirected graph[1]

[1] https://www.geeksforgeeks.org/dsa/introduction-to-graphs-data-structure-and-algorithm-tutorials/

MATRIX REPRESENTATION

One of the simple representation of the graph is to use **2D matrix** of size VxV, where each cell

$$(i,j) = \begin{cases} 1, if\ there\ is\ an\ edge\ from\ vertex\ i\ to\ j \\ 0, otherwise \end{cases} \quad (2)$$

Example:

$$AdjM = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad (3)$$

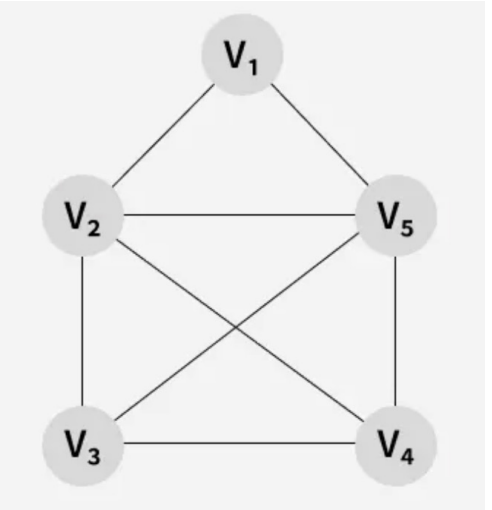


Figure 2 – Example graph [1]

Pros:

- Easy to check if edge exists → O(1);
- Simple implementation.

Cons:

- Requires O(V^2) space.

ADJACENCY & EDGE LIST

KAZAKH BRITISH TECHNICAL UNIVERSITY

In **adjacency list** each vertex stores a list of all the vertices it's connected to.

Example:

0: [1, 2],
1: [0, 2, 3],
2: [0, 1, 4],
3: [1, 4],
4: [2, 3]

Pros:

- Space-efficient for sparse graphs (O(V + E))
- Easier for traversal (like BFS/DFS)

Cons:

- Slower edge existence check → O(deg(V))

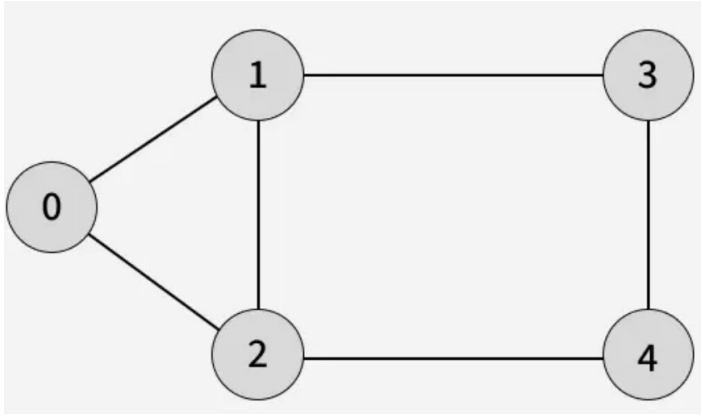


Figure 3 – Example undirected graph [2]

[2] https://www.geeksforgeeks.org/dsa/breadth-first-search-or-bfs-for-a-graph/

ADJACENCY & EDGE LIST (CONT.)

The **edge list** store all edges as pairs (u, v).

Example:

Edges = [(0, 1), (0, 2),
(1, 2), (1, 3),
(3, 1), (3, 4)]

Use Cases:

- Easy for algorithms like Kruskal's MST
- Compact representation

Pros: Simple & compact.

Cons: Not efficient for neighbor lookups.

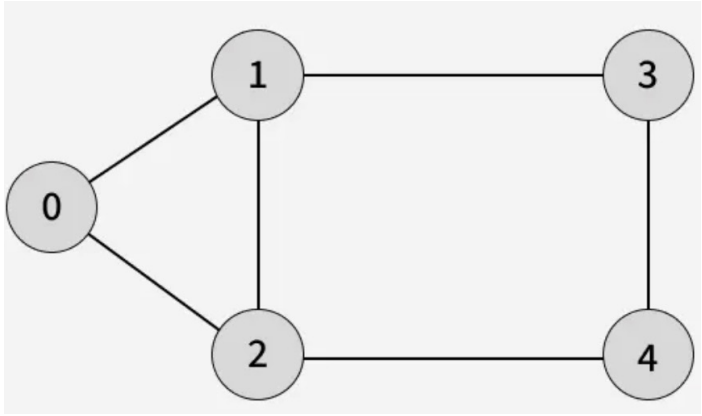


Figure 3 – Example undirected graph [2]

[2] https://www.geeksforgeeks.org/dsa/breadth-first-search-or-bfs-for-a-graph/

BREADTH-FIRST SEARCH

The Breadth-First Search (BFS) algorithm uses a **queue** to traverse a graph **level by level**, and great for finding shortest paths in **unweighted graphs**.

Steps of algorithm:

1. Starts from initial vertex;
2. Visit its all neighbors;
3. Move to next level neighbors;
4. Repeat the 2nd & 3rd steps until all reachable nodes are visited.

Time Complexity: O(V+E)

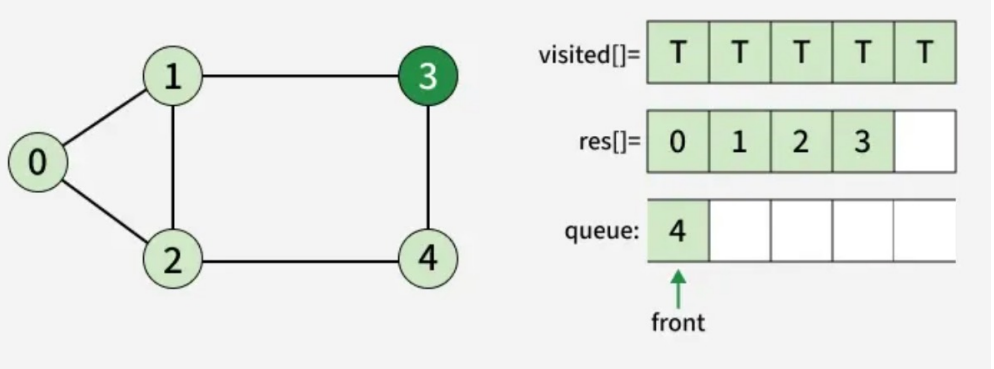


Figure 4 – Algorithm representation of BFS

[2] https://www.geeksforgeeks.org/dsa/breadth-first-search-or-bfs-for-a-graph/

BREADTH-FIRST SEARCH (CONT.)

- Introduction to Graphs
- Matrix representation
- Adjacency & Edge list
- Breadth-First Search
- Practice Problems

Given the graph as shown in Figure 5, and we should traverse upon that graph.

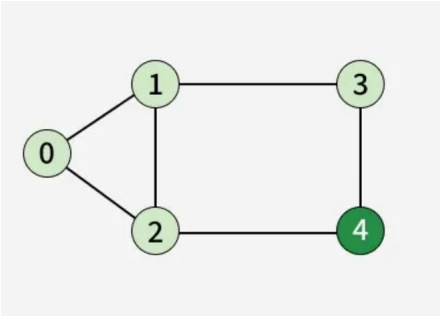
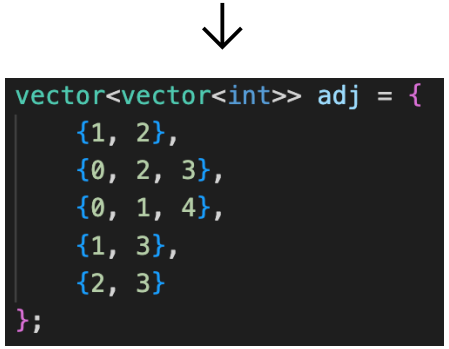


Figure 5 – Given graph [2].



```
vector<vector<int>> adj = {
    {1, 2},
    {0, 2, 3},
    {0, 1, 4},
    {1, 3},
    {2, 3}
};
```

Figure 6 – Graph representation.

```
void bfs(int start, vector<vector<int>> adj) {
    vector<bool> visited(adj.size(), false);
    queue<int> q;

    visited[start] = true;
    q.push(start);

    while(!q.empty()) {
        int vertex = q.front();
        q.pop();
        cout << vertex << " ";

        for (int i = 0; i < adj[vertex].size(); ++i) {
            int neighbor = adj[vertex][i];
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
}
```

Figure 7 – BFS algorithm.

```
(base) assylzhan@AIs-MacBook-Air L10 % ./a.out
0 1 2 3 4
```

Figure 8 – output of the BFS algorithm.

[2] https://www.geeksforgeeks.org/dsa/breadth-first-search-or-bfs-for-a-graph/

KAZAKH KBTU BRITISH TECHNICAL UNIVERSITY

PRACTICE PROBLEMS

We have two tasks to solve:

- 104. Maximum Depth of Binary Tree;
- 463. Island Perimeter;
- 547. Number of Provinces.

Q & A