FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION ITMO UNIVERSITY

Report
on the practical task No. 5

“Algorithms on graphs. Introduction to graphs and basic algorithms on graphs”

Performed by
Kozlov Alexey
Dmitriy Koryakov
J4133c
Accepted by
Dr Petr Chunaev

St.Petersburg
2022

Table of Contents

Goal

The use of different representations of graphs and basic algorithms on graphs (Depth-first search and Breadth-first search)

Problems

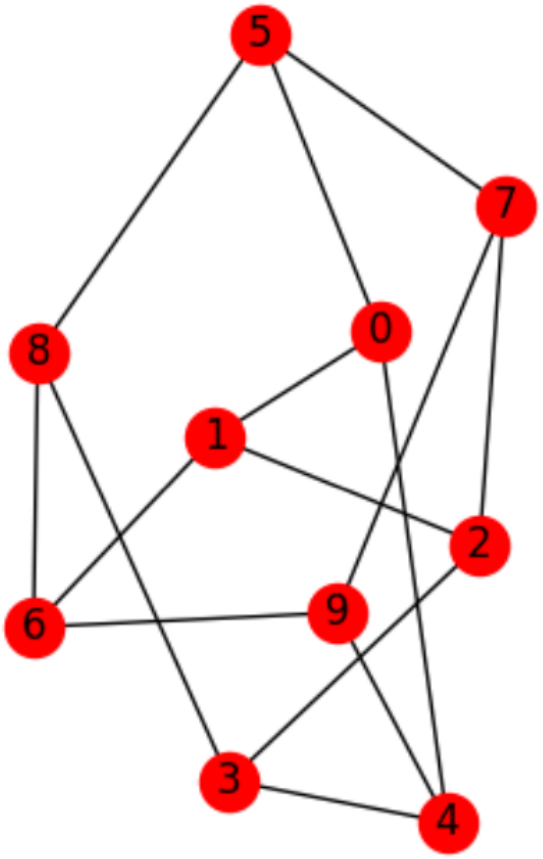
I. Generate a random adjacency matrix for a simple undirected unweighted graph with 100 vertices and 200 edges (note that the matrix should be symmetric and contain only 0s and 1s as elements). Transfer the matrix into an adjacency list. Visualize the graph and print several rows of the adjacency matrix and the adjacency list. Which purposes is each representation more convenient for?

II. Use Depth-first search to find connected components of the graph and Breadth- first search to find a shortest path between two random vertices. Analyse the results obtained.

III. Describe the data structures and design techniques used within the algorithms.

Brief theoretical part

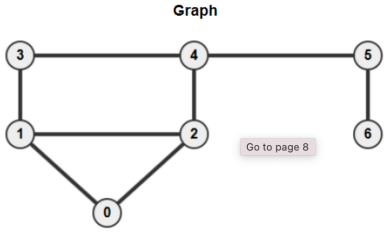
An (undirected) graph is a pair G = (V, E), where V is a set whose elements are called vertices (or nodes), and E is a set of two-sets (sets with two distinct elements) of vertices, whose elements are called edges (or links). The number of vertices is usually denoted by |V|. The number of edges is usually denoted by |E|.



The adjacency matrix is a matrix whose rows and columns are indexed by vertices and whose cells contain a Boolean value that indicates whether the corresponding vertices are adjacent (for weighted graphs, it contains corresponding weights instead of 1s.). The matrix (stored as a 2D array) requires $O(|V|^2)$ of space.

The adjacency list is a collection of lists containing the set of adjacent vertices of a vertex. The list (stored as an 1D array of lists) requires O ($|V|+|E|$) of space.

For a sparse graph, i.e. a graph in which most pairs of vertices are not connected by edges, $|E| \ll |V|^2$, an adjacency list is significantly more space-efficient than an adjacency matrix.



Adjacency matrix

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

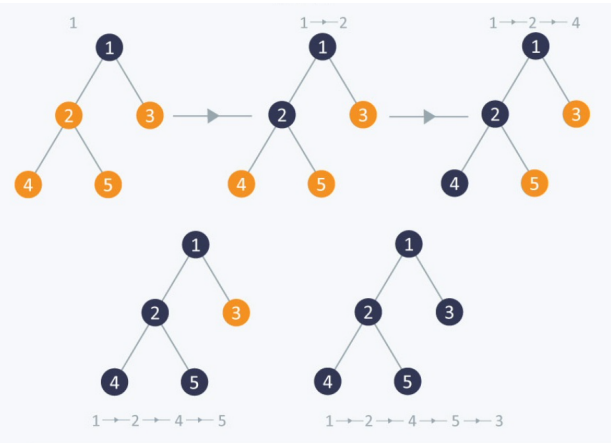
Adjacency list

Vertex	Adjacent vertices
0	1, 2
1	0, 2, 3
2	0, 1, 4
3	1, 4
4	2, 3, 5
5	4, 6
6	5

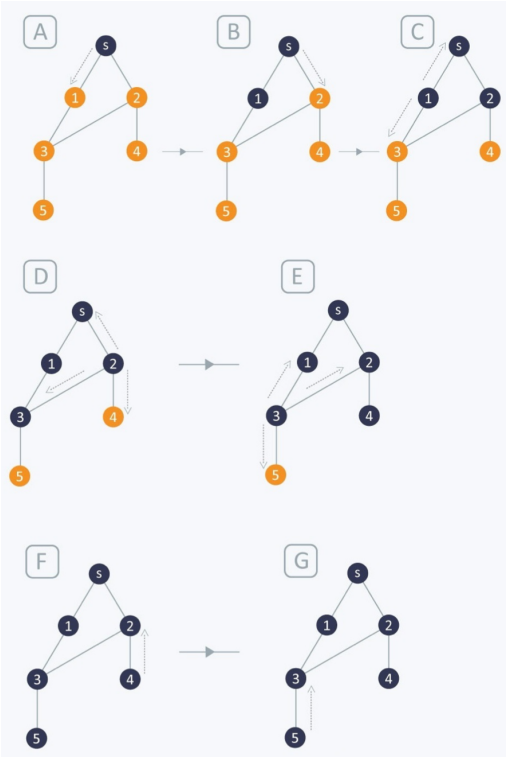
Depth-first search (DFS) is an algorithm for traversing a graph. The algorithm starts at a chosen root vertex and explores as far as possible along each branch before backtracking.

Applied for: searching connected components, searching loops in a graph, testing bipartiteness, topological sorting, etc.

The time complexity of DFS is $O(|V|+|E|)$.



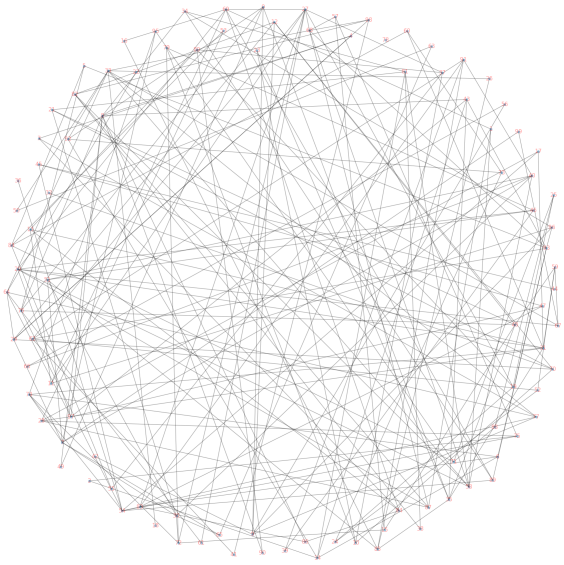
Breadth-first search (BFS) is an algorithm for traversing or searching a graph. The algorithm starts at a chosen root vertex and explores all of the neighbour vertices at the present depth prior to moving on to the vertices at the next depth level. It uses an opposite strategy to DFS, which instead explores the vertex branch as far as possible before being forced to backtrack and expand other vertices.



Applied for: searching shortest path

Solution

1. Generated a random adjacency matrix for a simple undirected unweighted graph with 100 vertices and 200 edges.



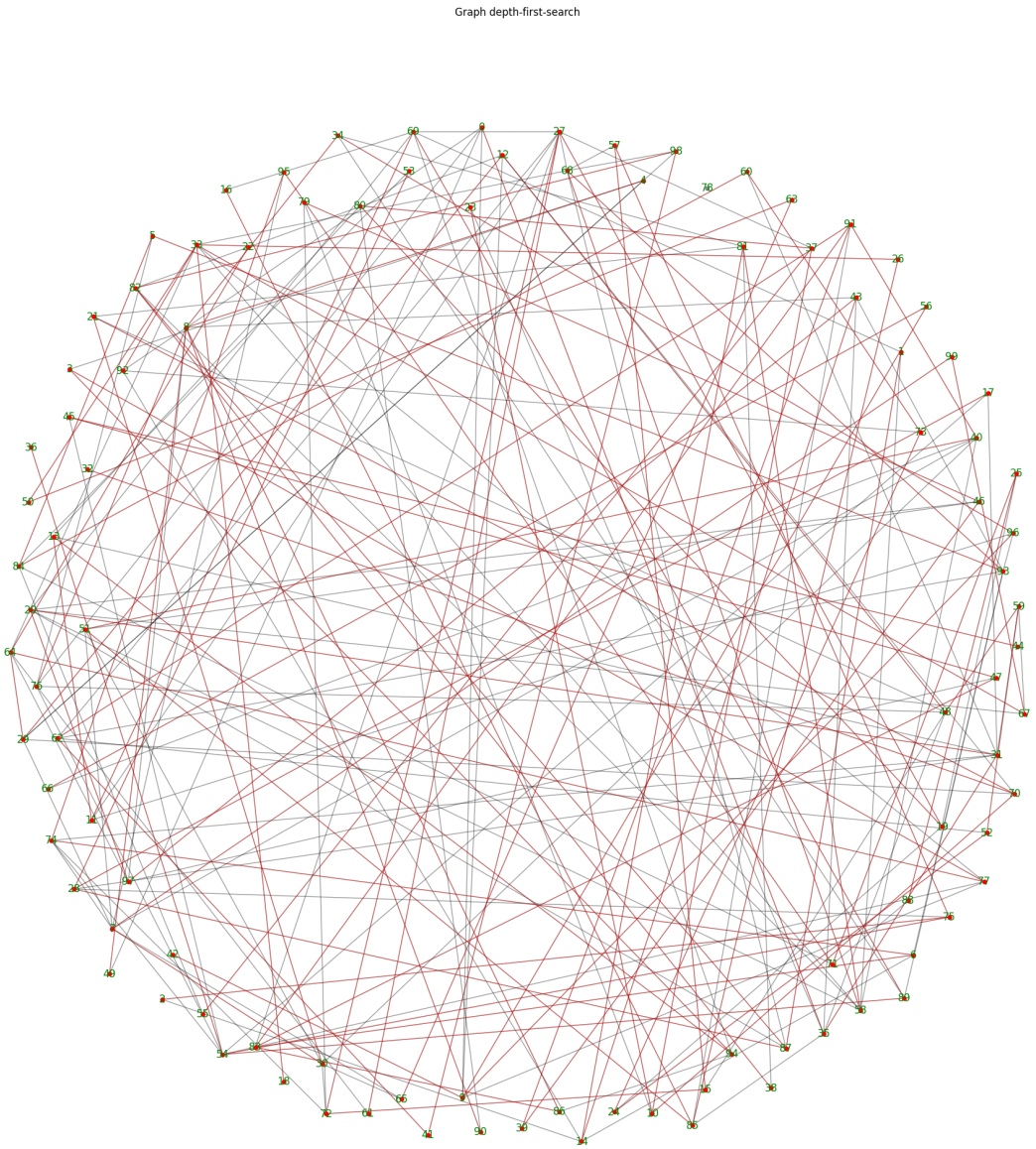
several rows of the adjacency matrix

```
matrix([[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

several rows of the adjacency list

```
{0: [9, 35, 49, 76, 92],
 1: [58, 69, 87],
 2: [14, 75],
 3: [4, 77, 94],
 4: [3, 8, 29],
 5: [76, 84, 93],
 6: [14, 74, 83, 93],
 7: [29, 30, 40, 51, 65, 74, 95, 99],
 8: [4, 34, 43, 49, 76, 86, 90, 97],
 9: [0, 12, 42, 56, 77, 96]}
```

2. Depth-first search, connected components of the graph.

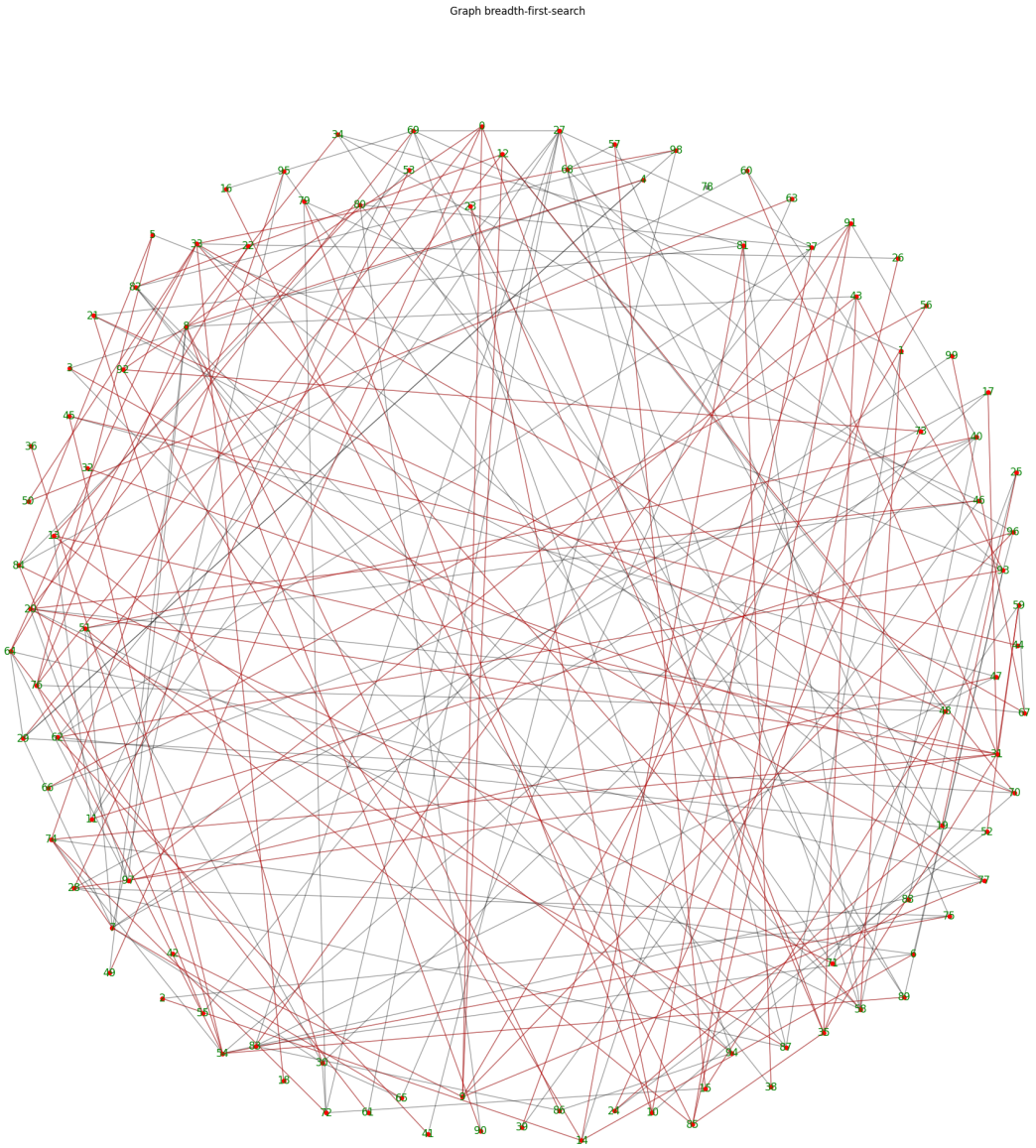


The first 10 edges and the last 10 edges of connected components.

```
path_edges[:10], path_edges[-10:]
```

```
Out[14]: ([(0, 35),
           (35, 31),
           (31, 20),
           (20, 97),
           (97, 43),
           (43, 14),
           (14, 81),
           (81, 58),
           (58, 68),
           (68, 93)],
          [(82, 38),
           (85, 16),
           (33, 50),
           (50, 63),
           (63, 65),
           (10, 23),
           (37, 55),
           (27, 61),
           (75, 2),
           (54, 36)])
```

Breadth-first search, a shortest path between two vertices.
The source is the zero node for all destinations.

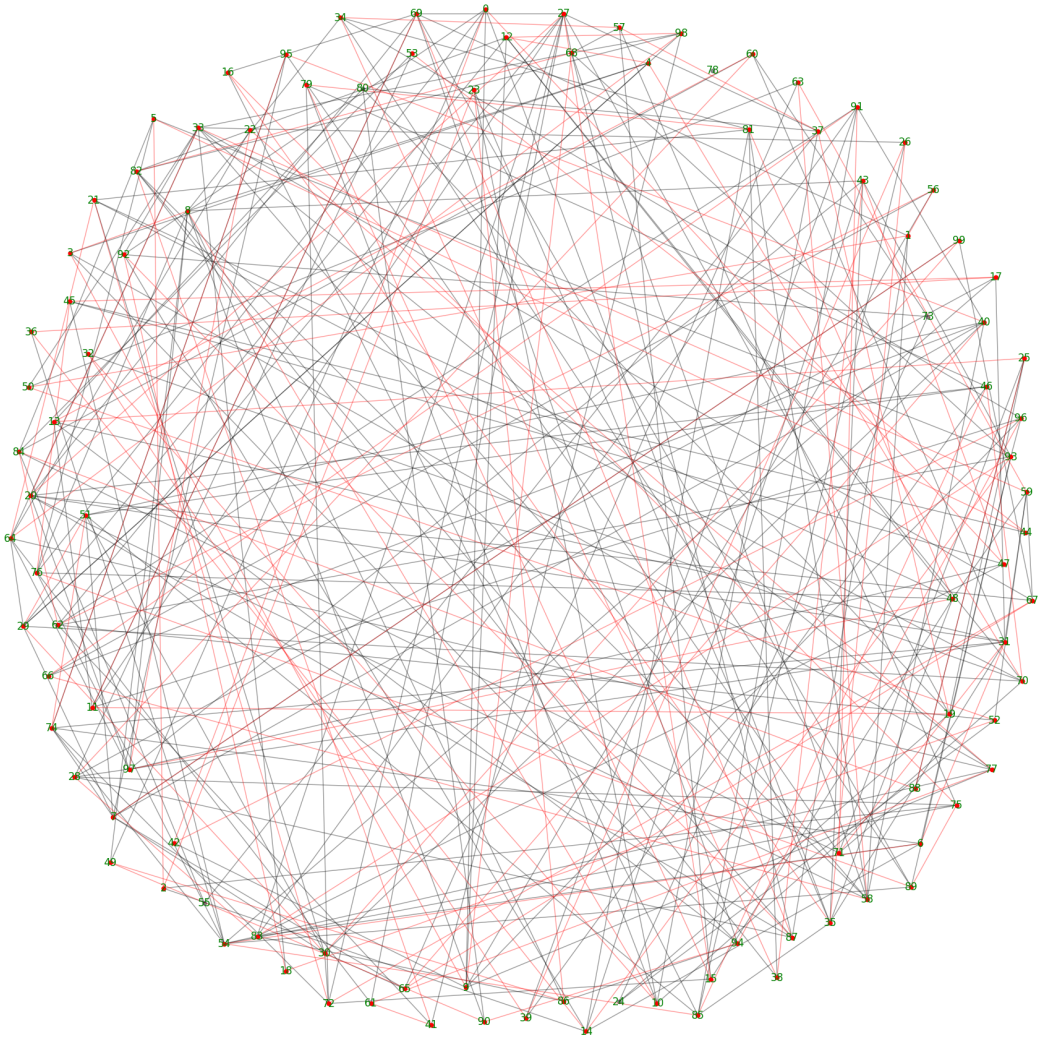


The shortest path between two random vertices.

```
source, destination, path
```

```
Out[24]: (7,
          53,
          array([ 7, 99, 30, 29, 51, 74, 95, 40, 65, 67, 72, 21, 64, 98, 12,  4, 70,
                 46, 85, 54,  6, 31, 71, 91, 66, 58, 63, 59, 33, 20, 27, 15, 79, 81,
                 48, 97, 22, 77, 61, 41, 82, 68,  9,  3,  8, 10, 32, 39, 26, 35, 16,
                 75, 89, 76, 45, 17, 36, 14, 93, 83, 28, 60, 13, 25, 88, 84, 62, 56,
                  1, 50, 52, 18, 92, 87, 69, 11, 19, 37, 57, 34, 38, 43, 47,  0, 42,
                 96, 94, 90, 49, 86, 23,  2,  5, 44, 53]))
```

Graph The shortest path between two random verticese



Conclusions

The time complexity of the Depth-First Search algorithm is represented within the sort of O(V+E), where V is that the number of nodes and E is that the number of edges. The space complexity of the algorithm is O(V).

The time complexity of the Breadth first Search algorithm is in the form of O(V+E), where V is the representation of the number of nodes and E is the number of edges. Also, the space complexity of the BFS algorithm is O(V).

Pros and cons of each presentation:

Adjacency matrix

- Uses O(n^2) memory
- It's quick to search and check for the presence or absence of a particular edge between any two nodes O(1)
- It's slow to go over all the edges
- Slow adding/removing a node is a complex O(n^2) operation
- It's fast to add a new edge O(1)

Adjacency list

- Use memory in anticipation for the number of edges (not the number of nodes), which can save a lot of memory if the adjacency matrix is sparse
- Finding the presence or absence of a specific edge between any two nodes slightly slower than with an O(k) matrix where k is the number of adjacent nodes
- Quickly iterate over all edges Because you can access any neighbor node directly
- Quickly add/remove node easier than matrix view
- It's fast to add a new edge O(1)

Depth-First Search Algorithm has a wide range of applications for practical purposes. Some of them are as discussed below:

1. For finding the strongly connected components of the graph
2. For finding the path
3. To test if the graph is bipartite
4. For detecting cycles in a graph
5. Topological Sorting
6. Solving the puzzle with only one solution.
7. Network Analysis
8. Mapping Routes
9. Scheduling a problem

Breadth-first Search Algorithm has a wide range of applications in the real-world. Some of them are as discussed below:

1. In GPS navigation, it helps in finding the shortest path available from one point to another.
2. In pathfinding algorithms
3. Cycle detection in an undirected graph
4. In minimum spanning tree
5. To build index by search index
6. In Ford-Fulkerson algorithm to find maximum flow in a network.

Appendix

Link to github - https://github.com/OnlyOneUseAcc/algorithms-RD-practise