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

Konstantin Krechetov

Academic group: j4132c

Accepted by

Dr Petr Chunaev

St. Petersburg

2020

Goal

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

Formulation of the problem

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

3.1 Depth-first search (DFS)

Depth-first search (DFS*) is an algorithm for traversing or searching 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 and so on.*

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

A connected component of a graph is a subgraph in which any two vertices are connected by paths.

3.2 Breadth-first search (BFS)

Breadth-first search (BFS) *is an algorithm for traversing or searching a graph. The algorithm starts at a chosen root vertex and explores all the neighbor 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.*

The time complexity *of BFS is* ***O(|V | + |E|).***

Results

In theory for sparse graph adjacency matrix consume more memory than adjacency list, but there are also some existing sparse data structures which erase this drawback.

The main drawback of the adjacency list is that there is no quick way to check existence of edge (u, v).

As we can see on example of graph with 100 nodes and 200 edges there is no significant benefit in visualization.

Using DFS we can obtain number of components in graph and connected problems while BFS is one of the ways which finds shortest path between 2 nodes.

In this example, dictionaries and lists for programming visualization of above algorithms were used. Besides, I implemented recursive technique (for example in DFS).

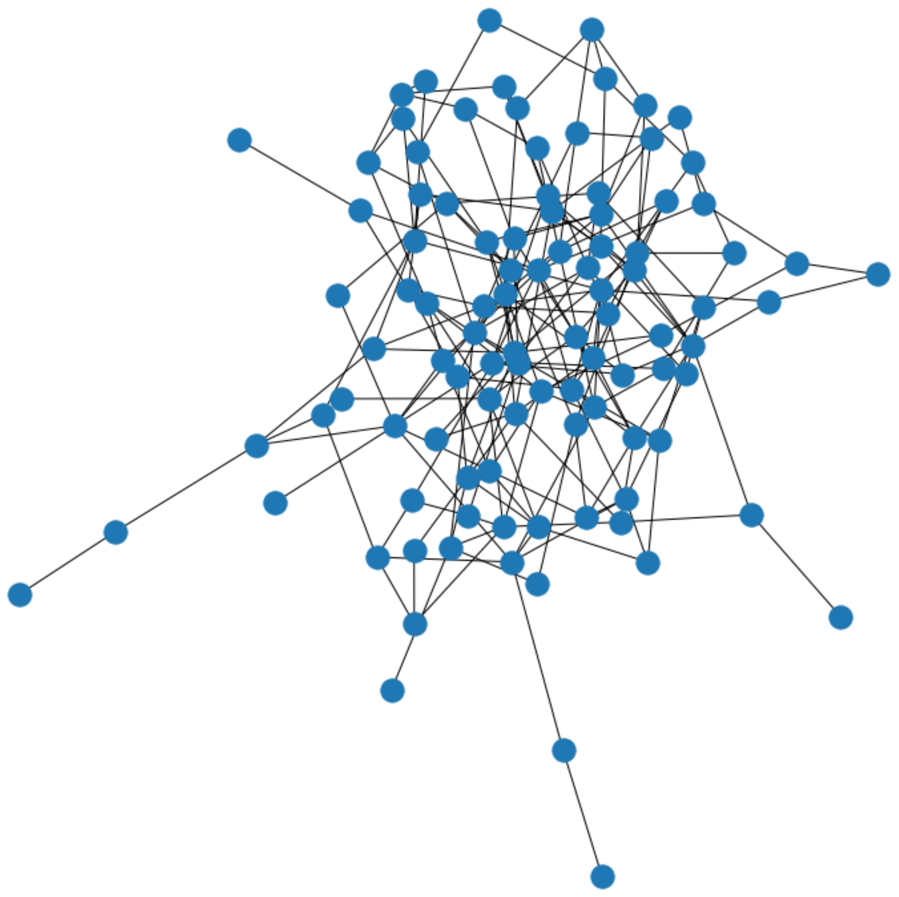
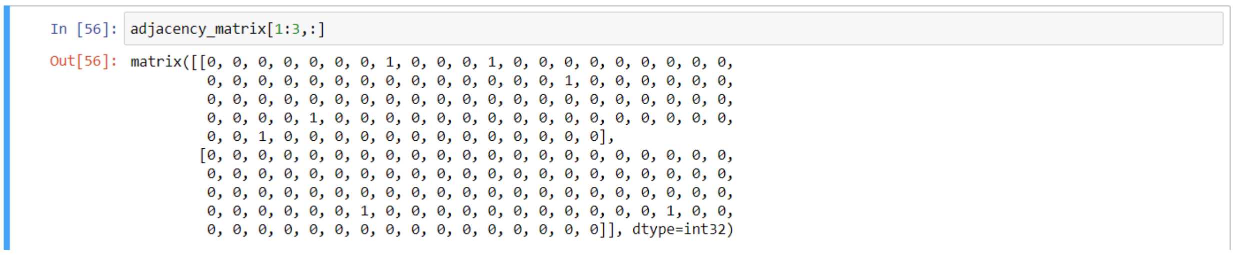


Figure 1*: Graph visualization.*



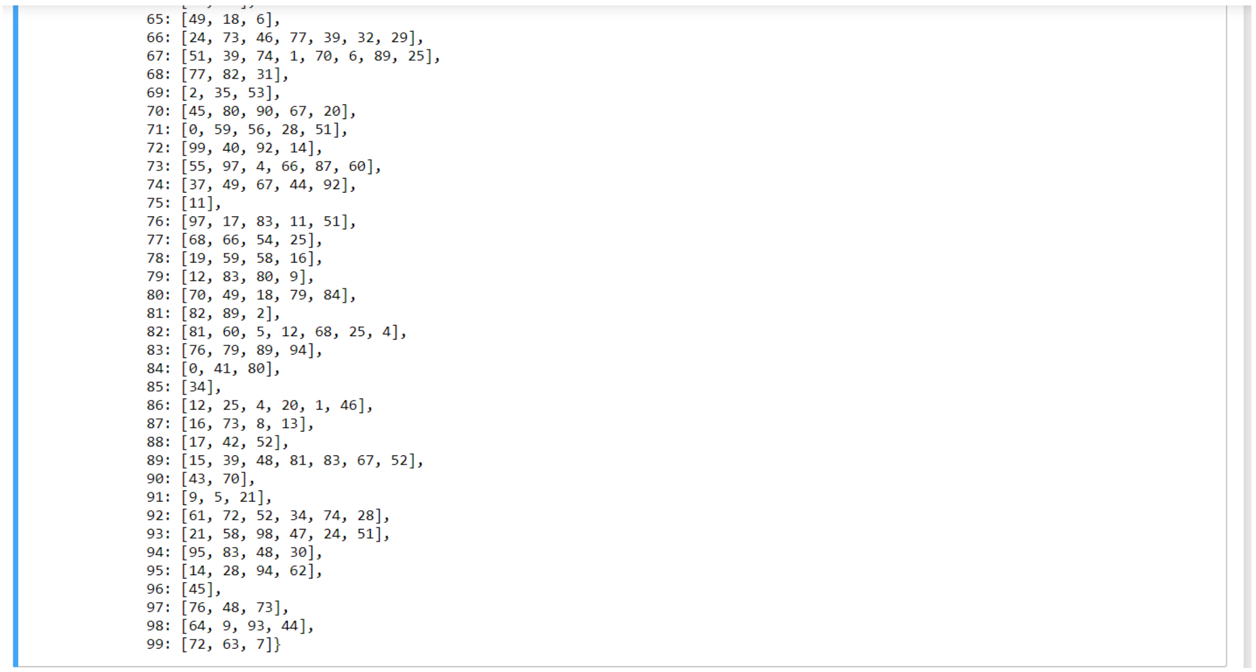
```
In [56]: adjacency_matrix[1:3,:]

Out[56]: matrix([[0, 0, 0, 0, 0, 0, 0, 1, 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, 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, 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, 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, 1, 0, 0,
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int32)
```

Figure 2: *Adjacency matrix.*

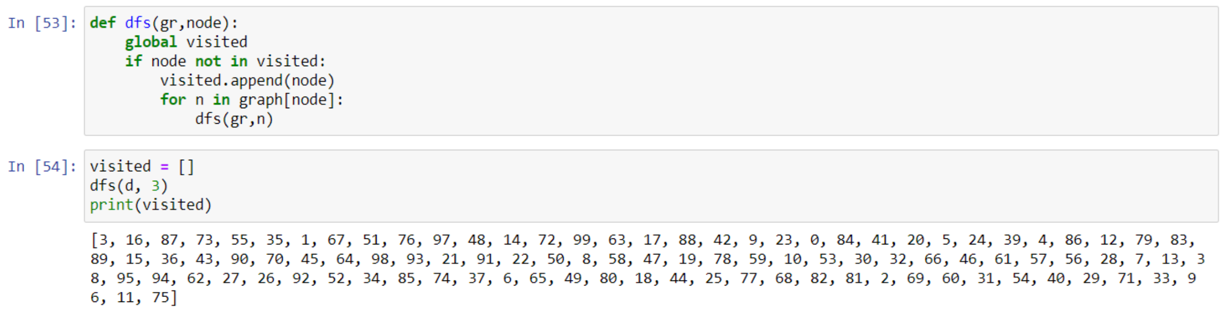
In [52]:
```
d = {}
for vertex in graph.adjacency():
    d[vertex[0]] = [*vertex[1].keys()]
d
```

Out[52]:
```
{0: [84, 52, 21, 71, 23],
 1: [35, 67, 11, 7, 86],
 2: [69, 81],
 3: [16, 48],
 4: [86, 73, 39, 13, 23, 82],
 5: [24, 82, 47, 20, 91, 50],
 6: [10, 67, 65, 37],
 7: [28, 13, 12, 1, 99],
 8: [22, 24, 87],
 9: [42, 23, 91, 98, 79, 50, 30],
 10: [53, 56, 6, 59, 54, 47],
 11: [76, 75, 1],
 12: [86, 79, 82, 7],
 13: [4, 7, 87],
 14: [48, 72, 95],
 15: [89, 36, 42],
 16: [3, 87, 41, 48, 78],
 17: [88, 76, 20, 47, 63],
 18: [80, 58, 65],
 19: [78, 47],
 20: [17, 5, 41, 86, 70],
 21: [93, 91, 0, 22],
 22: [50, 36, 8, 21],
 23: [9, 0, 4],
 24: [5, 39, 28, 66, 30, 93, 8],
 25: [86, 44, 77, 39, 67, 82],
 26: [28],
 27: [62],
 28: [56, 24, 7, 38, 95, 26, 92, 71],
 29: [40, 66],
 30: [53, 24, 32, 9, 57, 94],
 31: [68],
 32: [30, 66],
 33: [46],
 34: [85, 92],
 35: [1, 55, 69],
 36: [43, 15, 60, 22],
 37: [74, 61, 6],
 38: [28, 58],
 39: [24, 4, 89, 67, 66, 25],
 40: [72, 54, 29],
 41: [16, 84, 20],
 42: [9, 15, 88],
 43: [36, 90],
 44: [56, 25, 74, 98],
 45: [64, 96, 70, 57, 54],
 46: [66, 61, 59, 33, 54, 60, 86],
 47: [5, 58, 19, 17, 93, 10],
 48: [97, 14, 89, 16, 3, 94],
 49: [65, 80, 74],
 50: [22, 5, 9],
 51: [67, 76, 57, 93, 71],
 52: [0, 92, 88, 89],
 53: [10, 30, 69],
 54: [45, 40, 77, 10, 46, 63],
 55: [73, 35],
 56: [28, 10, 61, 44, 57, 71],
 57: [61, 45, 51, 56, 30],
 58: [47, 93, 18, 38, 78],
 59: [10, 78, 46, 71],
 60: [82, 36, 73, 46],
 61: [57, 92, 56, 46, 37],
 62: [27, 95],
 63: [99, 17, 54],
 64: [98, 45],
```



```
65: [49, 18, 6],
66: [24, 73, 46, 77, 39, 32, 29],
67: [51, 39, 74, 1, 70, 6, 89, 25],
68: [77, 82, 31],
69: [2, 35, 53],
70: [45, 80, 90, 67, 20],
71: [0, 59, 56, 28, 51],
72: [99, 40, 92, 14],
73: [55, 97, 4, 66, 87, 60],
74: [37, 49, 67, 44, 92],
75: [11],
76: [97, 17, 83, 11, 51],
77: [68, 66, 54, 25],
78: [19, 59, 58, 16],
79: [12, 83, 80, 9],
80: [70, 49, 18, 79, 84],
81: [82, 89, 2],
82: [81, 60, 5, 12, 68, 25, 4],
83: [76, 79, 89, 94],
84: [0, 41, 80],
85: [34],
86: [12, 25, 4, 20, 1, 46],
87: [16, 73, 8, 13],
88: [17, 42, 52],
89: [15, 39, 48, 81, 83, 67, 52],
90: [43, 70],
91: [9, 5, 21],
92: [61, 72, 52, 34, 74, 28],
93: [21, 58, 98, 47, 24, 51],
94: [95, 83, 48, 30],
95: [14, 28, 94, 62],
96: [45],
97: [76, 48, 73],
98: [64, 9, 93, 44],
99: [72, 63, 7]}
```

Figure 3*: Adjacency list (three pictures in a row)*

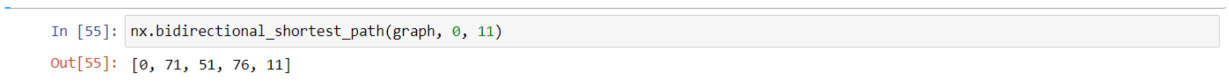


```
In [53]: def dfs(gr,node):
             global visited
             if node not in visited:
                 visited.append(node)
                 for n in graph[node]:
                     dfs(gr,n)

In [54]: visited = []
         dfs(d, 3)
         print(visited)

         [3, 16, 87, 73, 55, 35, 1, 67, 51, 76, 97, 48, 14, 72, 99, 63, 17, 88, 42, 9, 23, 0, 84, 41, 20, 5, 24, 39, 4, 86, 12, 79, 83,
         89, 15, 36, 43, 90, 70, 45, 64, 98, 93, 21, 91, 22, 50, 8, 58, 47, 19, 78, 59, 10, 53, 30, 32, 66, 46, 61, 57, 56, 28, 7, 13, 3
         8, 95, 94, 62, 27, 26, 92, 52, 34, 85, 74, 37, 6, 65, 49, 80, 18, 44, 25, 77, 68, 82, 81, 2, 69, 60, 31, 54, 40, 29, 71, 33, 9
         6, 11, 75]
```

Figure 4*: Results of DFS calculations*



```
In [55]: nx.bidirectional_shortest_path(graph, 0, 11)

Out[55]: [0, 71, 51, 76, 11]
```

Figure 5*: Results of BFS calculations*

Conclusions

In this work two most common graph methods were used: BFS and DFS. Both of these algorithms are proved to work correctly and provide such results as shortest path between any two nodes and number of components in graph.

Appendix

Source code is available on

https://github.com/KostyaKrechetov/ITMO-Analysis-and-development-of-algorithms/tree/master/Task5