FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report

on the practical task 5

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

Performed by

Ilya Lyalinov

Academic group J4134c

Accepted by

Dr Petr Chunaev

St. Petersburg

2021

Goal

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

Problems and methods

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

$G = (V, E)$ -- graph
A) DFS method:
We assume that initially every node is painted white.
For each $v \in V$:
Do DFS(v)

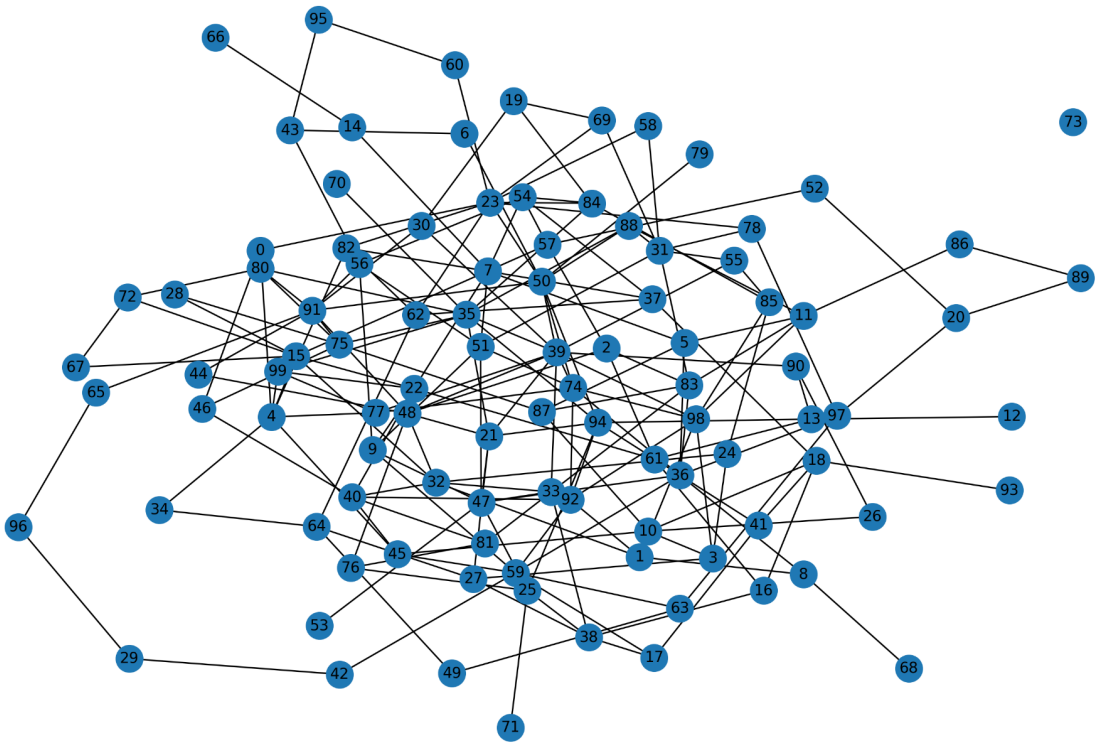
DFS(v):
1) Paint v to gray (partially visited)
2) For each neighbor of v (w) colored white, recursively perform the DFS(w) procedure
3) Paint v to black

B) BFS method:
For each $v \in V$:
1) Put v to queue
2) Pick the first element of queue *u* and delete it from queue
3) add all neighbors of *u* to queue
4) end if queue is empty

Time complexity estimation for both methods: $O(|V| + |E|)$

Results

I. First 5 rows of adjacency matrix and first 5 elements of adjacency list are available at this link: *https://github.com/ilyalinov/Algorithms/blob/master/5/1.txt*.
Graph visualization:



Let n be the number of vertices in our graph, m -- number of edges. Below we give estimates of the time and amount of memory required to work with the adjacency matrix and adjacency list:

	Adjacency matrix	Adjacency list
memory	O(n^2)	O(n + m)
edge belongs to a graph	O(1)	O(n)
calculate node's degree	O(n)	O(1)
graph traversal	O(n^2)	O(n + m)

Adjacency matrix requires more memory than adjacency list as it stores more values. Traversing the graph and calculating node's degree is faster with adjacency list.

However, adjacency matrix is better suited for checking whether the edge belongs to the graph.

II. Execution time of DFS for connected components was ~0.00076 seconds.
Components found:
1: [0, 4, 23, 46, 15, 34, 45, 77, 82, 99, 50, 58, 60, 62, 69, 78, 84, 40, 75, 22, 28, 32, 57, 67, 72, 64, 10, 59, 39, 74, 7, 35, 6, 79, 88, 91, 94, 31, 95, 37, 19, 97, 54, 85, 81, 92, 80, 87, 9, 61, 1, 24, 33, 48, 2, 52, 27, 49, 3, 18, 26, 17, 36, 42, 63, 21, 30, 55, 90, 98, 51, 5, 14, 70, 43, 11, 44, 56, 65, 13, 25, 41, 83, 20, 76, 47, 16, 8, 38, 93, 29, 66, 86, 96, 12, 71, 89, 53, 68]

2: [73]

Execution time of BFS for finding shortest path between randomly chosen vertices 75 and 23 was ~0.00025
Shortest path found:
[75, 35, 84, 23]; shortest path length = 3.
DFS executed longer because it examined every edge and node. However, BFS has finished on a search depth of three.

III. BFS internally uses queue to order the vertices that need to be traversed. DFS may be implemented using stack to avoid the usage of recursion. In this case, the algorithm is exactly the same as BFS, but a stack is used instead of queue. Both algorithms use adjacency list. Design technique used with DFS and BFS is "decrease and conquer".

Conclusion

We generated random adjacency matrix with 100 vertices and 200 edges. We transferred the matrix to adjacency list, visualized the graph and printed 5 first elements of adjacency list and 5 first rows of adjacency matrix. We compared when it is more convenient to use a matrix and when it is more convenient to use a list. We used BFS and DFS to find connected components and the shortest path between 2 vertices and analyzed the results. We described used data structures and algorithms design techniques.

Appendix

https://github.com/ilyalinov/Algorithms