Algorithms and Data Structures - Assignment 6

Graph theory, graph algorithms

Task A) Theory questions

1) Which of the following are true with respect to a Breadth First Search from a given node v?

a. A queue can be used to store the nodes and traverse the graph.
b. The graph is partitioned in layers of distance to the original node.
c. A stack can be used to store the nodes and traverse the graph.
d. Every connected component in the graph is eventually reached.

2) Which of the following are true with respect to a Depth First Search from a given node v?

a. A queue can be used to store the nodes and traverse the graph.
b. The graph is partitioned in layers of distance to the original node.
c. A stack can be used to store the nodes and traverse the graph.
d. Every connected component in the graph is eventually reached.

3) Which of the following are true with respect to shortest paths?

a. The shortest path between two nodes is unique.
b. Every path in a Minimum Spanning Tree is itself a shortest path.
c. Every concatenation of two shortest paths is itself a shortest path.
d. Every subpath of a shortest path is itself a shortest path.

4) Given one tree, which are valid ways to obtain a forest?

a. Planting more trees.
b. Merging another tree with it.
c. Cutting away an edge from it.

d. Taking an element which has two children and split the tree at this position by doubling this element.

5) Which statement has to be true, so that a graph becomes a tree?

a. The graph has cycles in it.
b. The graph has no cycles in it.
c. It is a directed graph.
d. It is not a directed graph.

6) Which statements are true?

a. Edge-weighted graphs are graphs where edges have weight values.
b. Node-weighted graphs are graphs where nodes have weight values.
c. Weighted graphs are trees with each node having a weight value.
d. Weighted graphs are directed graphs.

Task B) Minimum Spanning Tree

The aim of this task is to get an overview of the Minimum Spanning Tree data structure and the algorithm of Jarnik, Prim and Dijkstra.

1) Jarnik, Prim und Dijkstra algorithm

First write down a description, in natural language, of the Jarnik, Prim und Dijkstra algorithm (max. 100 words).

2) Practical example

Find the minimum spanning tree in the graph in figure 1. Show each of your steps in a small sketch of the graph. Also, calculate the weight of the graph and the weight of the MST.

3) Time complexity

What is the time complexity of the Jarnik, Prim and Dijkstra algorithm in its most basic implementation? How can it be decreased? Name a couple of important situations where the MST (and in particular, the Jarnik, Prim and Dijkstra algorithm) are used.

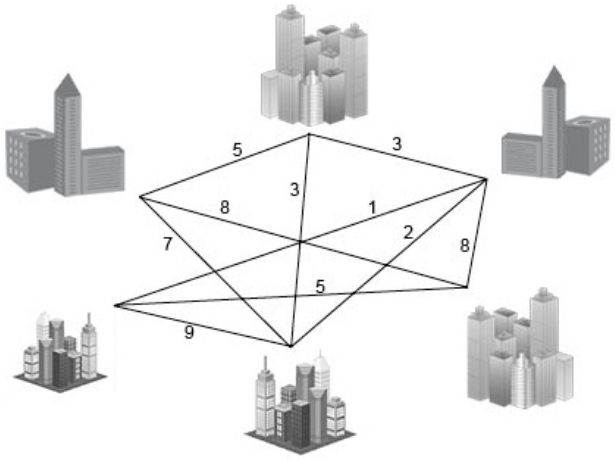

Figure 1: Sample graph for MST (I)

Task C) Kruskal's algorithm for minimum spanning trees

1) Use Kruskal's algorithm

Use the algorithm to find all minimum-weight spanning trees in the graph in figure 2, listing all the edges in the order in which you consider them, and indicating which edges are and are not included in each tree.

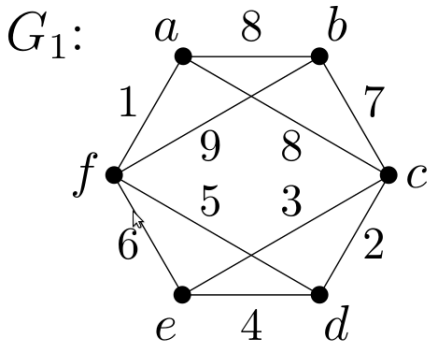

Figure 2: Sample graph for MST (II)

2) Another example

Now, do the same for the graph in figure 3.

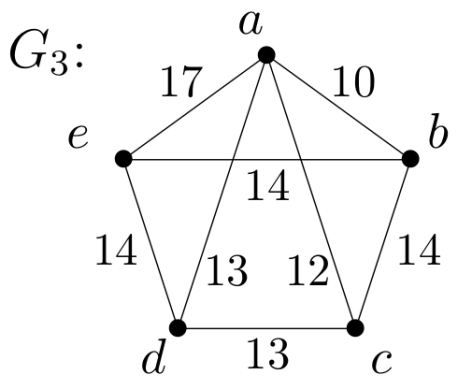

Figure 3: Sample MST graph (III)

Task D) Topological sort using Depth First Search (DFS)

In the folder *Task D*, fill the skeleton **void topologicalDFS(VVI& graph, int v)** in *main.cpp* so that it performs a topological DFS. The input graph is the one in figure 4. VVI is a typedef for a vector of vectors (the graph is represented with an adjacency matrix). Also, two sets for storing the explored nodes and the explored edges are already declared for you (an edge is represented by means of an std::pair<int, int>).

Note: whenever you have to visit several nodes or edges of equal priority in you algorithm, do so in numerical order. That way, the output for the sample graph should be $0, 1, 7, 3, 2, 6, 8, 5, 4$.

Hint: have a look at the "Topological Sorting Algorithm using DFS" in the slides *12_ Digraphs.pdf* in OLAT.

Task E) Breadth First Search (BFS)

In the folder *Task E*, fill the skeleton **void bfs(VVI& graph, int v)** in *main.cpp* so that it performs a BFS starting on the given vertex v. The input graph is the one in figure 5 (it is an undirected version of the previous graph), and the starting vertex is 0.

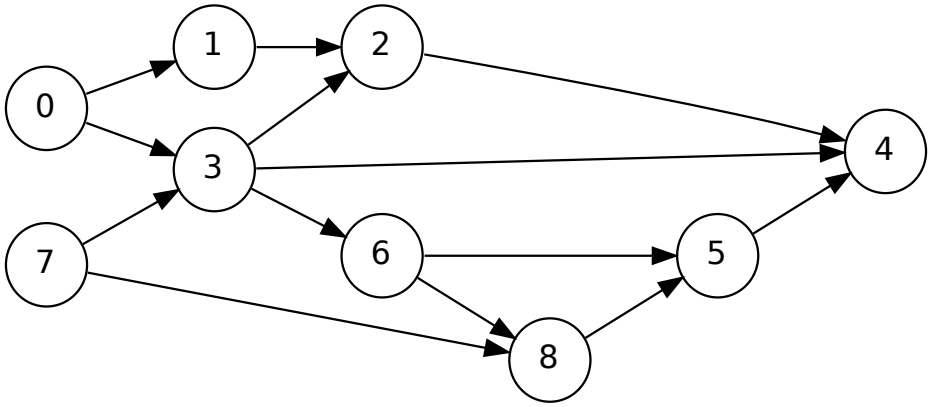

Figure 4: Sample graph for topological DFS

Note: whenever you have to visit several nodes or edges of equal priority in you algorithm, do so in numerical order. That way, the output for the sample graph should be $0, 1, 3, 2, 4, 6, 7, 5, 8$.

Hint: have a look at the slides *12_BFS.pdf* in OLAT.

Task F) Shortest path: Dijkstra

In the folder *Task F*, fill the skeleton **VI dijkstra(VVI& graph, int v)** in *main.cpp* so that it performs the Dijkstra shortest path algorithm. The return value should be a vector containing the distances from the given vertex to every vertex (VI is a typedef for **vector<int>**). The input is the undirected, edge-weighted graph in figure 6; the output vector for this graph should be $0, 3, 4, 7, 13, 15, 18, 12, 16$.

Hint: have a look at the slides *12_ShortestPath.pdf* in OLAT.

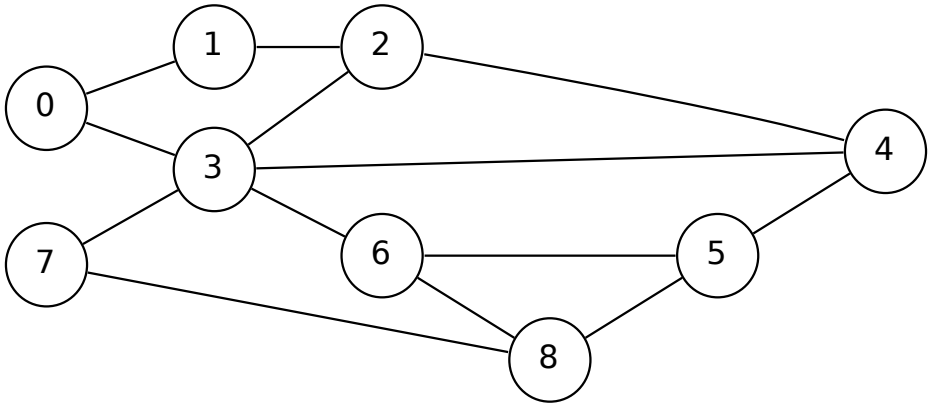

Figure 5: Sample graph for BFS

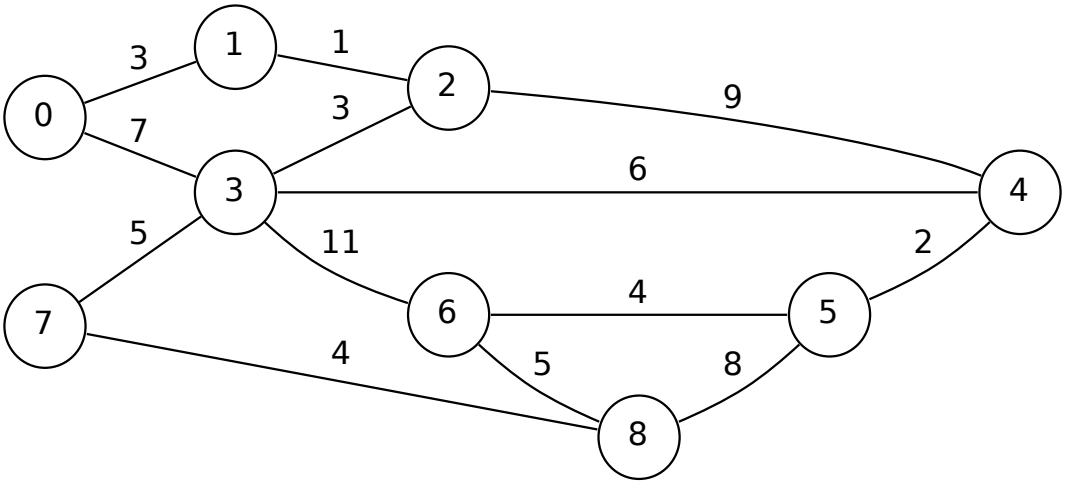

Figure 6: Sample graph for Dijkstra