1. A weighted connected graph G1 is given. There is a unique MST in G1. Next, all edge weights of G1 are (A) increased by positive constant c or (B) multiplied by positive constant c. What is the relation between the MST of the original and updated graph in case (A) and (B)?

2. The task is to find a spanning tree T of a given weighted graph. Also an interval <*c*1, *c*2> is given and the condition is stated that weight of each edge of T must be within <*c*1, *c*2>. T is therefore not necessarily a MST and might not be unique. Can this task be solved by some standard MST approach or is there some other more efficient method?

3. Recall the asymptotic complexities of Prim's and Kruskal's algorithms. Consider a graph G in which number of edges is four times bigger than the number of nodes. Which of these two algorithms processes G asymptotically faster?

4. Let us consider a weighted undirected graph G which is represented by the weight matrix C. Determine the asymptotic complexity of Kruskal's algorithm under assumption that the time of access to any element of C is constant but on the other hand each operation Union and Find is proportional to the number of nodes of G.

5. You are asked to find a minimum spanning tree of a complete weighted graph with the help of your personal computer. The result should be available next morning and the machine can run overnight. Estimate the maximum number of nodes in the graph with which the task can be finished in time. It is sufficient to give only order of magnitute of that number (like 103, 104, ..., etc.). Explain your estimation.

6. We have to find a minimum spanning tree T of a connected weighted graph G with *n* nodes and *m* = 5*n* edges. Next we have to remove the cheapest edge of T and thus create an unconnected graph T2. Finally, we have to create a spanning tree T3 by finding the most expensive edge possible in G and adding it into graph T2. Describe the algorithm which creates T3 and determine its asymptotic complexity.

7. Military headquarters in Graphland is looking for the cheapest spanning tree T of graph G with the additional demand that some specified edges must be included in T no matter what their real costs are. Therefore it is expected that T probably will not be a MST. Nonetheless, even under these conditions some standard algorithm for MST can be used with only minor modification. Describe your solution and determine its asymptotic complexity. Fortunately, no subset of the specified edges forms a cycle in the graph.

8. It turns out in the previous problem, that some of the specified edges in fact do form a cycle in the graph and maybe even more cycles than just one. Now the headquarters is not looking for a tree but instead for a cheapest subgraph of G which connects all nodes. The headqurters insists that all specified edges must be present in the subgraph. Show that even under these conditions the solution of the previous problem will work with virtually no difference.

9. Suppose that a graph is specified by its weight matrix. A special value (infinity or NaN etc.) in the matrix indicates that there is no edge between the corresponding nodes. Describe a modification of the Prim's algorithm which input is the weight matrix and which asymptotic complexity is proportional to the number of elements of the matrix.

10. Graph G was brought in by one of our customers and we found its MST and sent it to him before noon. In the afternoon, there is a call from the customer who apologizes and tells us that the weight of edge between the nodes 2075154 and 11439446 was originally wrongly specified and it is in fact by 17% smaller. Now our task is to decide whether this change will affect the resulting MST and if it does the task is to issue a correction of the original MST and its weight. All this can be done in linear time relative to the number on nodes of the graph, we are told by the boss, and he demands us to give the customer the updated results even still during the phone call.

Describe the algorithm of the correction, remember, its asymptotic complexity should be O(|V(G)|).

11. Solve the problem which is very similar to the previus one, with the only difference that the corrected edge weight is in fact bigger than the originally specified weight. Repeat your analysis and be aware that this time the correction algorithm complexity migth be higher than in previous case.

12. When all weights of edges in the graph are different is it automatically granted that there is only one unique minimum spanning tree in the graph? Explain your reasoning.

13. Recall what is the maximum number of edges in the planar graph with *n* nodes and discuss the question:

Which algorithm for finding MST is best suited for planar graphs? What is its asymptotic complexity relative to the number of nodes?

14. A greedy bandit is stealing edges one by one from the graph (and leaves the nodes untouched) and always takes away the most expensive edge available. He must not steal an edge which removal would disconnect the graph otherwise an alarm would be raised he would be caught. When the bandit's work is finished there remains not too much of the original graph. Is the remnant necessarily a minimum spanning tree of the original graph?

15. How do you apply an algorithm for finding a minimum spanning tree to the task of finding a maximum spanning tree of a given graph? Is it possible at all?

16. The structure of graph G1 and G2 is identical, G1 and G2 are isomorphic. In G1 nearly all edge weights are different and in G2 nearly all edges share the same weight. Which of MST algorithms you know is best suited for G1 and which is best suited for G2?

17. A backLink set of edges of an undirected graph is defined as a set of edges which contains at least one edge from each cycle in the graph. Therefore, when we remove any backLink set of edges from the graph there will be no cycles left in the graph (it may, of course, become disconnected). Is there any effective algorithm for finding a cheapest backLink set of edges in a weighted graph? (The cost of a set of edges is a sum of costst of all edges in the set.)

18. Explain why the Boruvka's algorithm for finding a MST of a graph contains a condition demanding that all edge weights are different. Draw an example of a graph in which modified Boruvka's algorithm without the condition would fail.

19. Let us consider a weighted undirected graph G which is represented by the weight matrix W. Determine the asymptotic complexity of Prim's algorithm under assumption that the time of access to any element of W is not constant but it is proportional to the number of nodes of G.

Example 1.

Let us consider a weigted undirected graph G which is represented by weight matrix W. Determine the asymptotic complexity of Kruskal's algorithm under assumption that the time of access to any element of W is not constant but it is proportional to the logarithm of number of nodes of G.

Let us suppose |V(G)| = *n*, |E(G)| = *m*.

Before we start sorting the edges we must first identify them in matrix W and transfer them into some data structure S which can be later sorted. Identification and transfer will take time proportional to *n*2 ∙log(*n*). Next, S must be sorted which takes time proportional to *m* ∙ log(*m*). For each edge *e* in the sorted structure it must be decided whether *e* belongs to MST. When the Union-Find structure is implemented effectively, the time of decision for one edge is at most proportional to log(*n*). In the best case we will be lucky and the first (*n*−1) edges in the sorted S will belong to MST and therefore the total decision time will be proportional to *n* ∙ log(*n*). We presume that we can add an edge to the MST in constant time. In the worst case it will be necessary to scan the whole sorted S before MST is found and consequently the total time of all decisions will be proportional to *m* ∙ log(*n*).

In the best case the complexity will be expressed by a function belonging to the class

Θ(*n*2 ∙log(*n*) + *m* ∙ log(*m*) + *n* ∙ log(*n*)) = Θ(*n*2 ∙log(*n*) + *m* ∙ log(*m*)).

In the worst case the complexity will be expressed by a function belonging to the class

Θ(*n*2 ∙log(*n*) + *m* ∙ log(*m*) + *m* ∙ log(*n*)) = Θ(*n*2 ∙log(*n*) + *m* ∙ log(*m*

Both variants lead to the same resulting complexity which is not that much surprising because both the terms *n* ∙ log(*n*) and *m* ∙ log(*n*) can be neglected due to the asymptotically more or equally important term *m* ∙ log(*m*).

It holds that the logarithm of number of edges is asymptotically proportional to the logarithm of number of nodes and further, it also holds that *m* < *n*2. Thus, for any connected graph it holds that the value of the term *m* ∙ log(*m*) grows at most as fast as the value of the term *n*2 ∙ log(*n*). This means that the term *m* ∙ log(*m*)can be neglected in the resulting asymptotic complexity. The concluding asymptotic complexity is

Θ(*n*2 ∙log(*n*) + *m* ∙ log(*m*)) = Θ(*n*2 ∙log(*n*)).

Example 2.

The weight of each edge of the given weighted connected graph G with *n* nodes can be only either v1 or v2 and it holds v1 < v2. We have to decide wheteher G contains a MST which weight is exactly v1×(*n*−1) and if the MST exists we have to list its edges. Suggest an algorithm other than standard MST algorithms like Kruskal's's or Prim's's that solves the problem effectively and compare its asymptotic complexity with the asymptotic complexity of both mentioned standard algorithms.

A neccessary and sufficient condition for the existence of MST with weight v1×(*n*−1) is existence of connected factor F (= subgraph containing all nodes of G) with edge weights all equal to v1. The existence of F can be easily decided by application of a graph search algorithm like BFS or DFS. We start in any node of the graph apply the search algorithm and during the search consider only edges with weight v1. Finally, we check if all nodes have been visited. If the answer is yes than the factor F exists and any of its spanning trees is also a minimum spanning tree of G. If BFS or DFS fail to visit all nodes the then the spanning tree with weight v1×(*n*−1) does not exist. Because both BFS and DFS build a search tree as an integral part of their work we can directly use that tree as a MST we look for in case of success.

Asymptotic complexity of both BFS and DFS is O(|E(G)|), asymptotic complexity of Kruskal's algorithm is

O(|E(G)| ∙ log(|V(G)|) which is obviously no less than O(|E(G)|) in any graph. We conclude that the modification suggested above is superior to or at least as good as the Kruskal's algorithm in all cases.

The asymptotic complexity of Prim's algorithm depends on the data structures used in its implementation. Let us consider the more or less standard implementation which uses priority queue where insert and search operations take at most time proportional to the logarithm of queue size. In such case the complexity of Prim's algorithm is O(|E(G)| + |V(G)| ∙ log( |V(G)|). Looking at this complexity formula we can see that for graphs in which holds |V(G)| ∙ log(|V(G)|) ∈ O(|E(G)|) this formula can be simplified to O(|E(G)|), which is the same complexity as the proposed BFS/DFS approach has. When the number of edges is asymptotically smaller than |V(G)| ∙ log(|V(G)|)

that is when |E|∉ Ω(|V(G)| ∙ log(|V(G)|)) (which represents a relatively sparse graph, for example a planar one) then the modification using BFS/DFS is asymptotically faster.