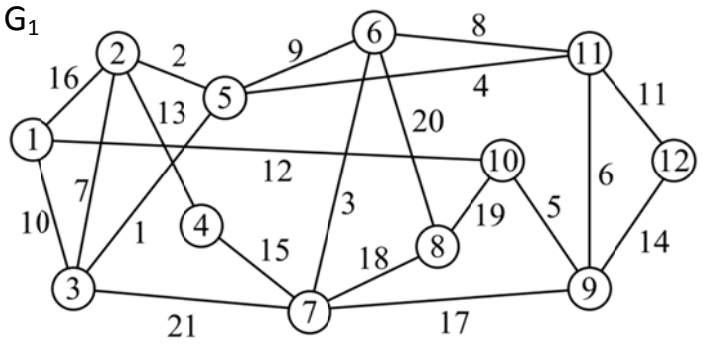
1. Determine the cost of the MST in the graph G_1 on the right [1].

2. In the same graph, determine the cost of a spanning tree which cost is maximum possible.

3. In the same graph, we add a new adge between vertices 6 and 9 with positive integer weight W. We want the edge 6 - 9 to be a part of the MST of the updated graph.
List all possible values of W.



4. Boruvka's algorithm is run of graph G_1. Determine exactly how many times will be performed the body of the central while loop in Boruvka's algorithm.

5. Prim's algorithm is run on graph G1. It starts in node 8. Determine the last node which the algoritm adds to the (gradually growing) MST.

6. Professor Maverick is convinced that the previous problem 5. can be solved correctly without constructing the MST of the graph. Decide if he is right and give some explanation of your thoughts.

7. 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 it might not be unique. Can this task be solved by some standard MST approach or is there some other more efficient method?

8. We have to find a minimum spanning tree T of a connected weighted graph G with n nodes and m = 5n 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.

9. 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.

10 A greedy bandit is stealing edges one by one from the given graph (and leaves the nodes untouched) and always takes away the most expensive edge available. However, he must not steal any edge which removal would disconnect the graph, otherwise an alarm would be raised and he would be caught and jailed. 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?

Examples of calculating asymptotic complexity of a MST related problem

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 $n2 \cdot \log(n)$. Next, S must be sorted which takes time proportional to $m \cdot \log(m)$. 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 $n \cdot \log(n)$. 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 $m \cdot \log(n)$.

In the best case the complexity will be expressed by a function belonging to the class
$\Theta(n2 \cdot \log(n) + m \cdot \log(m) + n \cdot \log(n)) = \Theta(n2 \cdot \log(n) + m \cdot \log(m))$.
In the worst case the complexity will be expressed by a function belonging to the class
$\Theta(n2 \cdot \log(n) + m \cdot \log(m) + m \cdot \log(n)) = \Theta(n2 \cdot \log(n) + m \cdot \log(m$
Both variants lead to the same resulting complexity which is not that much surprising because both the terms $n \cdot \log(n)$ and $m \cdot \log(n)$ can be neglected due to the asymptotically more or equally important term $m \cdot \log(m)$.

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 $m < n2$. Thus, for any connected graph it holds that the value of the term $m \cdot \log(m)$ grows at most as fast as the value of the term $n2 \cdot \log(n)$. This means that the term $m \cdot \log(m)$can be neglected in the resulting asymptotic complexity. The resulting asymptotic complexity is
$\Theta(n2 \cdot \log(n) + m \cdot \log(m)) = \Theta(n2 \cdot \log(n))$.
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 $v1 \times (n-1)$ 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 $v1 \times (n-1)$ 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 $v1 \times (n-1)$ 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 $O(|E(G)| \cdot \log(|V(G)|)$ 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 $O(|E(G)| + |V(G)| \cdot \log(|V(G)|)$. Looking at this complexity formula we can see that for graphs in which holds $|V(G)| \cdot \log(|V(G)|) \in O(|E(G)|)$ 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 $|V(G)| \cdot \log(|V(G)|)$ that is when $|E| \notin \Omega(|V(G)| \cdot \log(|V(G)|))$ (which represents a relatively sparse graph, for example a planar one) then the modification using BFS/DFS is asymptotically faster.

[1] Problems on Algorithms, Second Edition, by Ian Parberry and William Gasarch, 2002