Universitetet i Oslo
Institutt for Informatikk

A. Maus, R.K. Runde, I. Yu

INF2220: algorithms and data structures

Series 6

Topic Shortest paths and minimal spanning trees (Exercises with hints for solution)

Issued: 28. Sept. 2017

Classroom

Exercise 1 (Dijkstra's algorithm) Apply Dijkstra's algorithm to find the *shortest* path from the given start nodes to all others in Figures 1 and 2. Construct a table for the algorithms with entries `vertex`, `known`, $\mathtt{d}_v$ and $\mathtt{p}_v$.

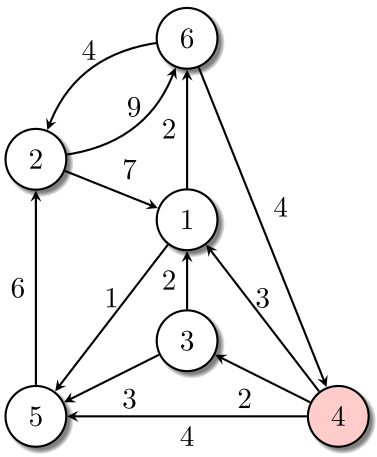

Figure 1: Starting from node 4

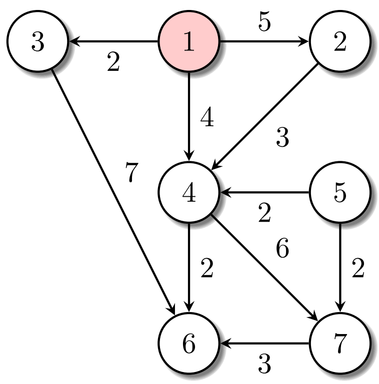

Figure 2: Starting from node 1

Solution: [for Exercise 1 (Dijkstra)] Running Dijkstra results in the tables from Figure 3 and 4.

Exercise 2 (Prim's algorithm) Apply Prim's algorithm to the graphs in Figures 5 and 6 to construct a minimum spanning tree, respectively. Show the final table produced by Prim's algorithm for each tree.

Solution: [for Exercise 2 (Prim)] The results are shown in Figures 7 and 8. □

Vertex	Known	d_v	p_v
1	T	3	4
2	T	9	6
3	T	2	4
4	T	0	-
5	T	4	4/1
6	T	5	1

Figure 3: Table for Figure 1, starting from node 4

Vertex	Known	d_v	p_v
1	T	0	-
2	T	5	1
3	T	2	1
4	T	4	1
5	F	∞	-
6	T	6	4
7	T	10	4

Figure 4: Table for Figure 2, starting from node 1

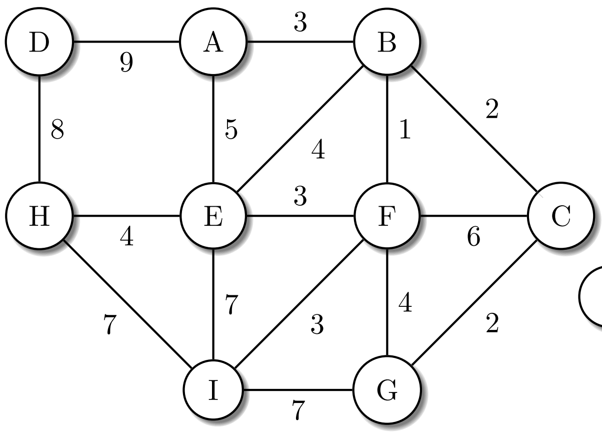

Figure 5: Undirected, weighted graph

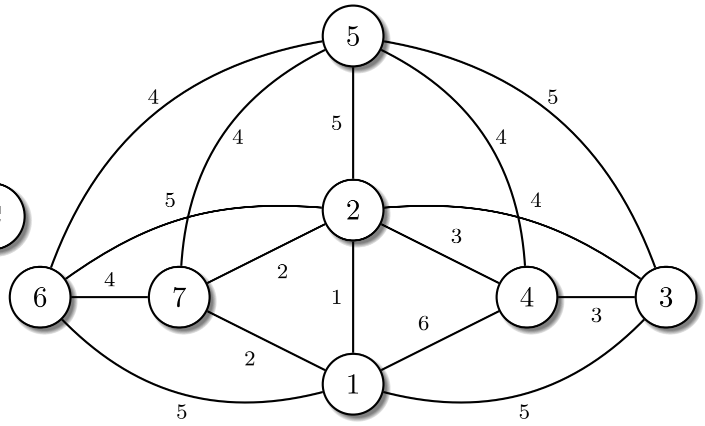

Figure 6: Undirected, weighted graph

Exercise 3 (Kruskal's algorithm) Apply Kruskal's algorithm to the graphs of Figure 5 and 6 to construct the minimum spanning tree, respectively. Calculate the cost of each of the minimum spanning trees. Are the costs same as those of the spanning trees resulted in Exercise 2?

Solution: [of Exercise 3 (Kruskal)] The resulting graphs are shown in Figures 9 and 10. The *costs* are 26 and 17, respectively

□

Exercise 4 (Spanning tree) Let $G = (V, E)$ be a connected, undirected graph and let T be a minimum spanning tree of G. Assume now that we change the weight of the edge $e = (u, v)$ in G.

1. Explain what changes of e will cause T to no longer be a minimum spanning tree of G. (NB. e is not necessarily an edge in T).

2. Explain an efficient algorithm that will with minimal changes to T make T again be a minimum spanning tree. Your algorithm should not change the weights in the graph.

Solution:

For 1:

- case 1: Suppose $e = (u, v)$ is in T, then T may no longer be an MST if the cost of e becomes larger than the cost of an edge not in T that is in any path between u and v.

Vertex	Known	d_v	p_v
A	T	3	B
B	T	2	C
C	T	2	G
D	T	8	H
E	T	3	F
F	T	1	B
G	T	0	-
H	T	4	E
I	T	3	F

Figure 7: Table for Prim on Fig. 5

Vertex	Known	d_v	p_v
1	T	1	2
2	T	0	-
3	T	3	4
4	T	3	2
5	T	4	7
6	T	4	7
7	T	2	2

Figure 8: Table for Prim on Fig. 6

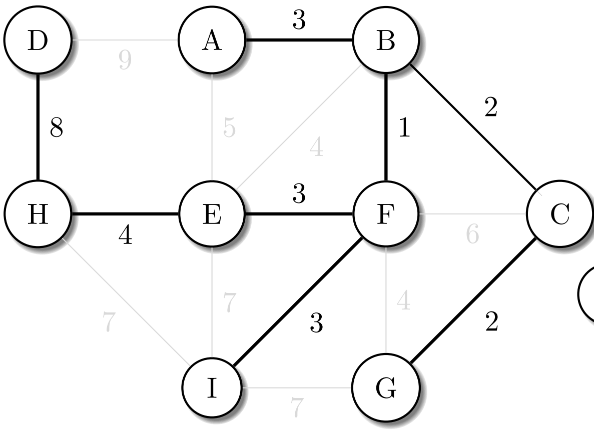

Figure 9: MST for Fig. 5 (Kruskal)

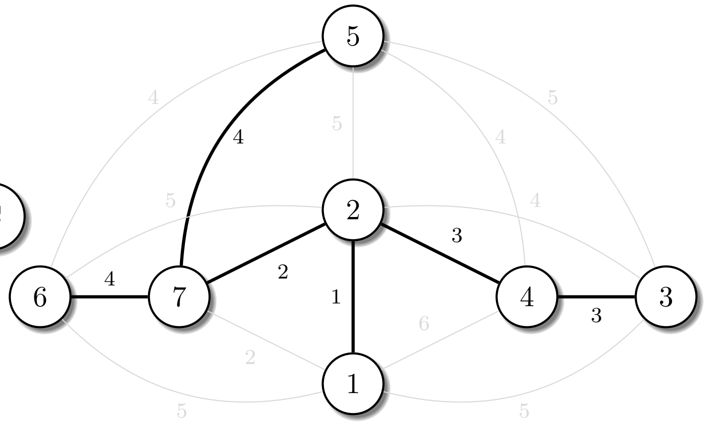

Figure 10: MST for Fig. 6 (Kruskal)

- case 2: Suppose $e = (u, v)$ is not in T, then T may not be an MST if the cost of e becomes smaller than the largest cost in the path between u and v in T.

For 2:

- for case 1: e is removed such that now T is disconnected into trees T_1 and T_2(you can show that these are MSTs for the corresponding induced subgraphs), then the minimum cost edge joining a vertex in T_1 to a vertex in T_2 is added to make the MST.
- for case 2: the edge with largest cost in the path between u and v in T is removed and new edge is added to T as before.

Lab

Exercise 5 We have a look at some particular kind of *directed graphs*, obeying the restriction that all nodes have maximally *one successor*, but possibly more that one *predecessors*. That class of graphs contains ordinary, linear *lists*, simple cyclic structures such as loops/cycles, and a form of *trees* where the edges are directed towards the root. There are a lot of further kinds of graphs. Draw a couple of different examples.

Assume that such graphs are represented by nodes of the following type:

```
class Node {
```

```
    Node succ;   // it is null if there is no successor
    int mark;
  }
```

A given graph consists of n nodes and to access them we use an *array:*

```
Node [] graph = new Node[n];
```

The array contains the nodes in arbitrary order.

The exercise requires to program the following three boolean methods (i.e., methods with boolean return type):

1. Checking whether or not the graph is a simple, linear lists ending with null.
2. Checking whether or not the graph is a simple cycle/loop?
3. Checking whether or not the graph is one single tree (with the edges pointing towards the root).

Assume that in addition to the code fragment sketched above there is also an integer variable `mark`, which should be intially null. You can use that variable in the implementation of the methods, if you wish. Try to solve the problems using as little different values for that variable, or even better using it at all.

Exercise 6 Write a method which reads an *adjacency matrix* representation of a graph and applies *Prim's* algorithm to find a minimum spanning tree of the input graph. The method should print the table produced by Prim's algorithm which is similar to the one you have seen in the lecture.

References