Report Project 3

Course code: IX1500 - Discrete Mathematics
Date: 2017-10-10
Evan Saboo, saboo@kth.se

Task Level C: Graph Application

Summary

Task

Every city listed below represents a router that forwards packet to their final destination. The weight (cost) between two linked cities is assumed to be inversely proportional to the free capacity of the link.

This task is about creating an optimal tree so that each router can be accessed with minimal weight. The tree should show how data packets are routed so that each city can be reached by the information flow that is based on Stockholm.

```
city = {"Abisko", "Boden", "Falun", "Goteborg", "Hoganas",
   "Hudiksvall", "Jonkoping", "Kalmar", "Kiruna", "Lidkoping", "Linkoping",
   "Lulea", "Lund", "Malmo", "Mariestad", "Ostersund", "Stockholm",
   "Strangnas", "Timra", "Uppsala", "Umea", "Varberg", "Visby"};
```

```
links = {{15, 18}, {13, 17}, {3, 16}, {6, 3}, {18, 6}, {12, 2}, {15, 11}, {19, 21}, {7, 8},
   {19, 2}, {7, 4}, {21, 17}, {17, 11}, {23, 11}, {10, 7}, {16, 17}, {10, 20},
   {13, 5}, {3, 17}, {7, 22}, {15, 10}, {16, 18}, {11, 16}, {17, 23}, {18, 17},
   {20, 6}, {11, 7}, {9, 2}, {3, 20}, {4, 17}, {19, 17}, {7, 20}, {22, 4}, {14, 7},
   {15, 17}, {6, 17}, {20, 5}, {16, 15}, {11, 3}, {9, 1}, {4, 10}, {5, 14}, {6, 16},
   {16, 19}, {13, 8}, {8, 23}, {16, 10}, {4, 13}, {2, 16}, {15, 3}, {20, 18}};
```

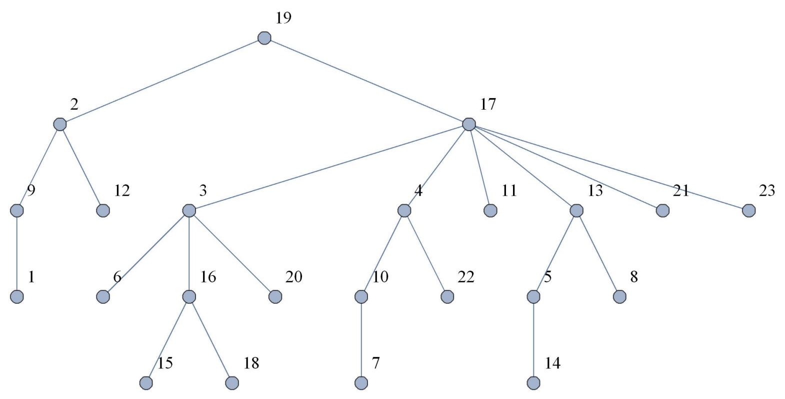
The list above shows how every city is linked (has an edge) to one or more cities.

```
city⟦links⟦4⟧⟧
{Hudiksvall, Falun}
```

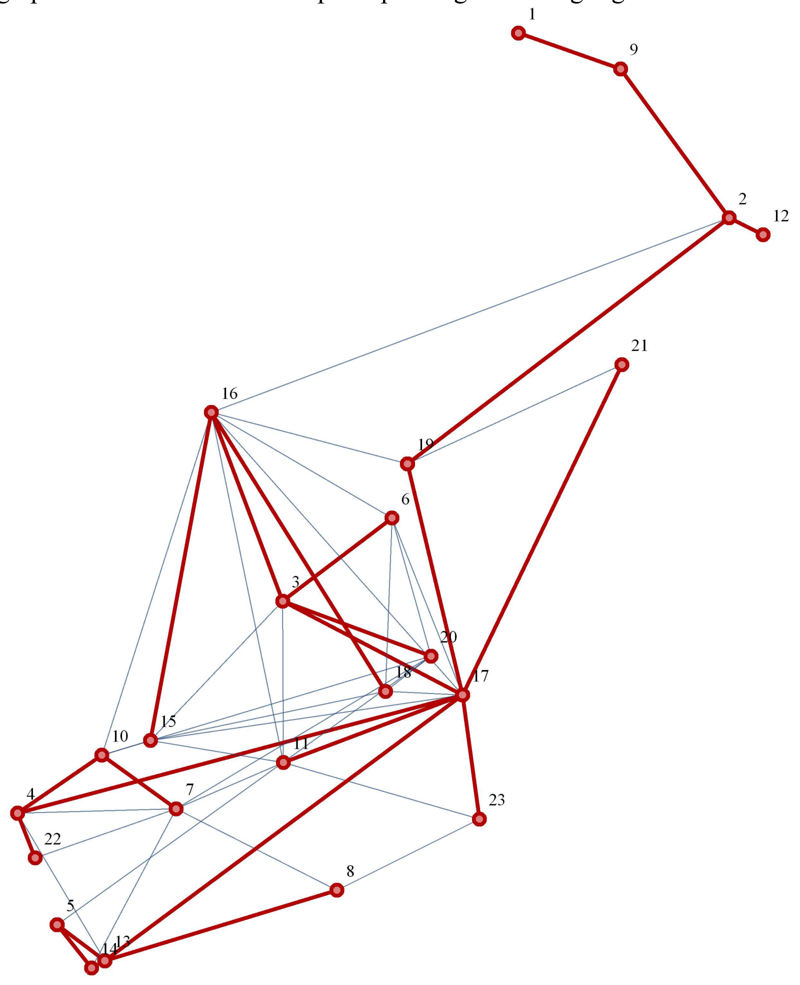
Result

By using Dijkstra’s algorithm (shortest path) we can construct a tree that connect to from Stockholm to every other city with minimal weight (cost).

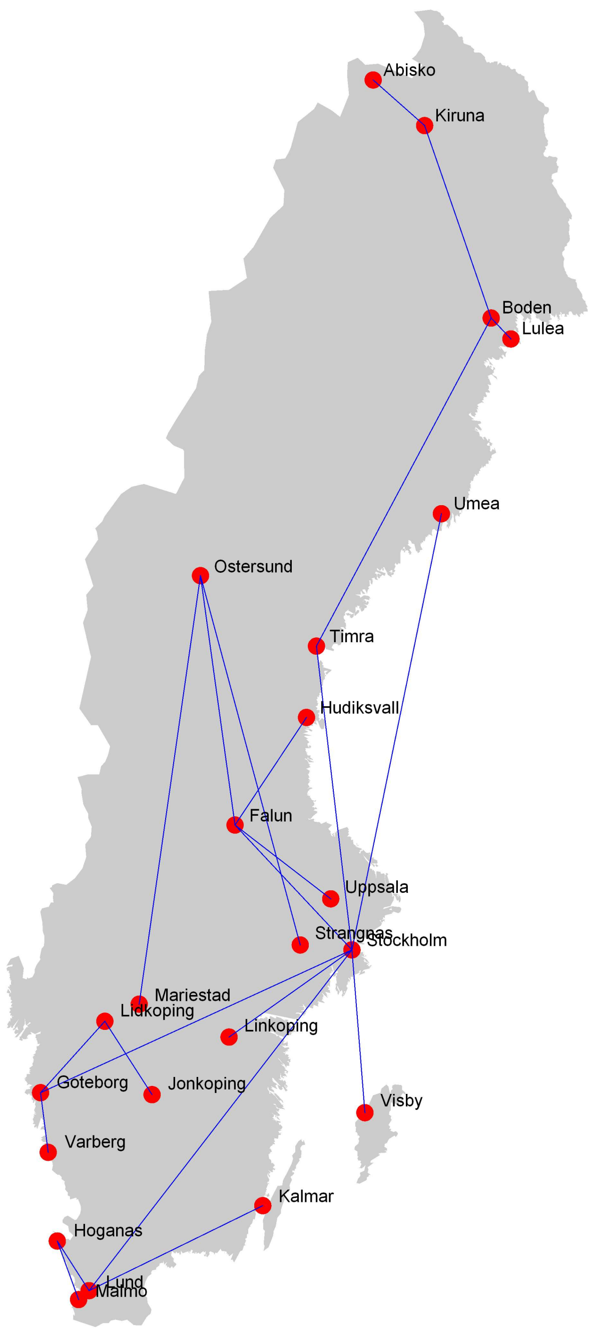
The picture below demonstrates the shortest path spanning tree where every city is presented as its corresponding index number in the “city list” above.



We can illustrate a graph by placing every city in its corresponding coordinates with their connected links. The graph also shows the shortest path spanning tree as highlighted.

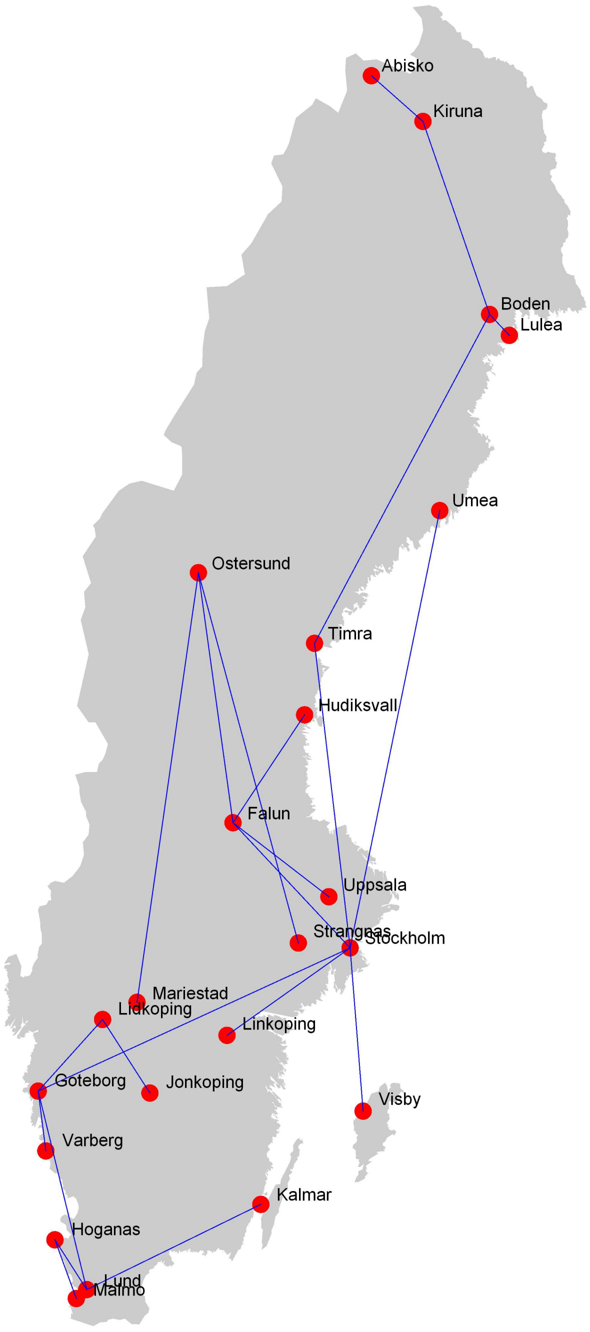


I have also illustrated the shortest path spanning tree with the map of Sweden and its connected cities (routers).



If the link between Stockholm and Lund stops working then the shortest path spanning tree changes. The shortest path from Stockholm to Lund in the new tree is:

Stockholm → Göteborg → Lund



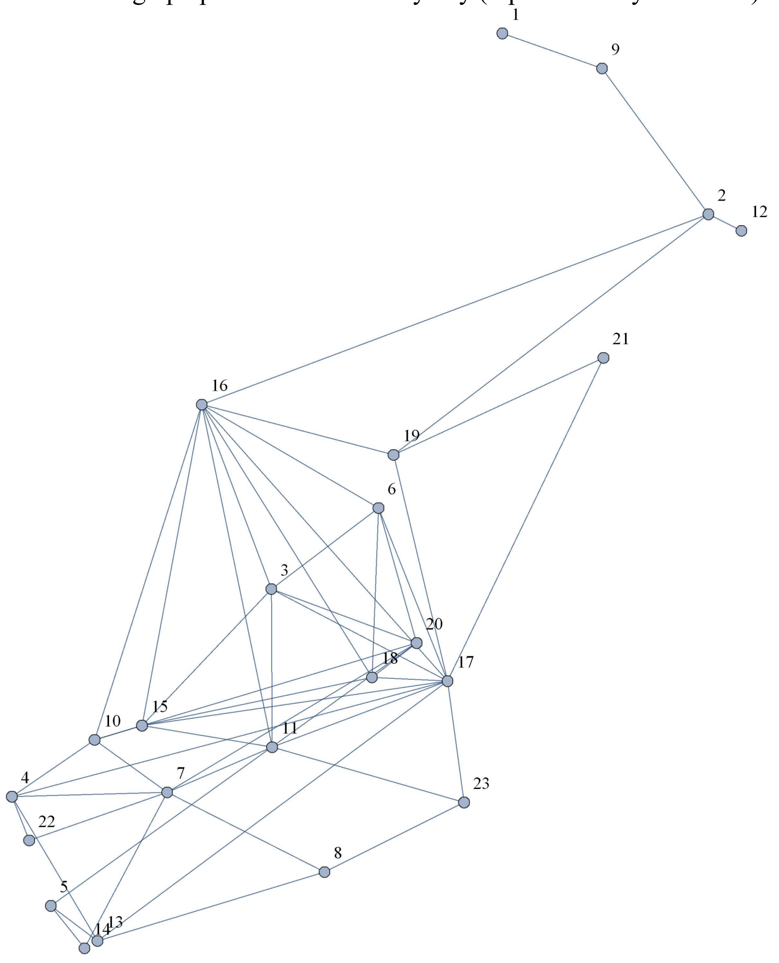
Routing

The first thing we need to do is to give every link its own weight (cost).
We do that by dividing a constant with the free capacity of the link.
In this case we suppose that the constant is 1 and the available capacity for each link is:

Stockholm	–	Göteborg	25
Stockholm	–	Lund	25
Göteborg	–	Lund	25
Stockholm	–	Falun	15
Falun	–	Östersund	15

The other links have a capacity ranging from 1 to 5 at random.

Below is the graph presentation of every city (represented by a number) connected with the provided links.



Creating shortest path spanning tree

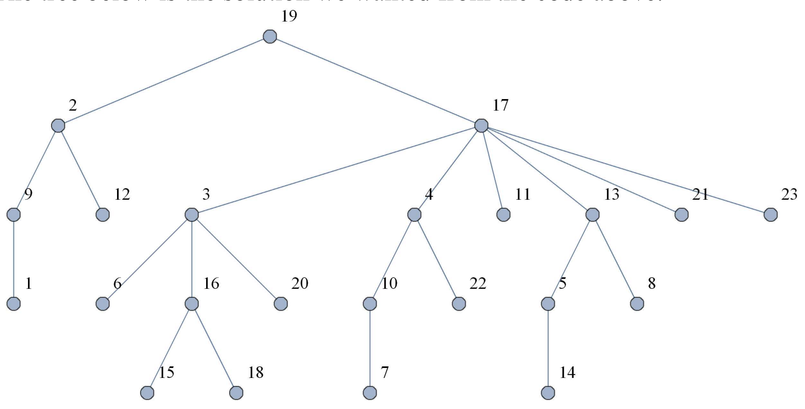
The tree we want to create needs to have the shortest path from Stockholm to every other city. We can compute the shortest path by using Dijkstra’s shortest path algorithm. The algorithm goes through every weighted edge connecting the starting node to the target node and chooses the path with the lowest total weight.
It can also find the shortest path from one vertex to every other vertex to create the shortest path spanning tree, where the starting vertex is the root.

By combining every path we can construct the shortest path spanning tree. The pseudo code below creates a list of links which represents the shortest path spanning tree:

FindShortestPathTree:

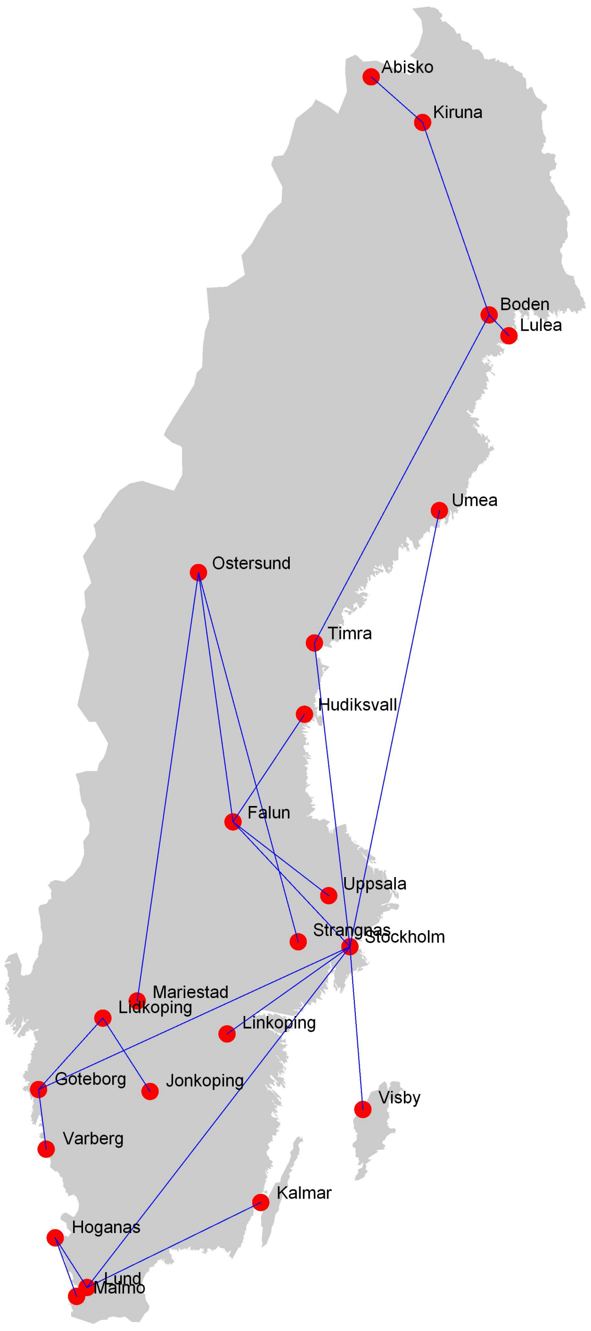
```
For $x$ = 1 to length(amountOfLinks)
        $P$ = Shortest Path From Stockholm to city[$x$];
                For edge = 1 to (length($P$) − 1)
                        if($P$[link] is not member of shortestPathTreeList)
                        Add $P$[link] to shortestPathTreeList;
                End for link
End for $x$
Return shortestPathTreeList
```

The tree below is the solution we wanted from the code above.



The reason why Dijkstra’s algorithm was used was because we wanted a spanning tree which guaranteed us the shortest path from a specific vertex to every other connected vertexes. We could have used Kruskal’s algorithm to create a minimum spanning tree but it wouldn’t guarantee us the shortest path from the root vertex to every other vertexes, we would only get a tree whose sum of all edge weights is the smallest possible weight.

We an also illustrate the shortest path tree with the map of Sweden and its cities:

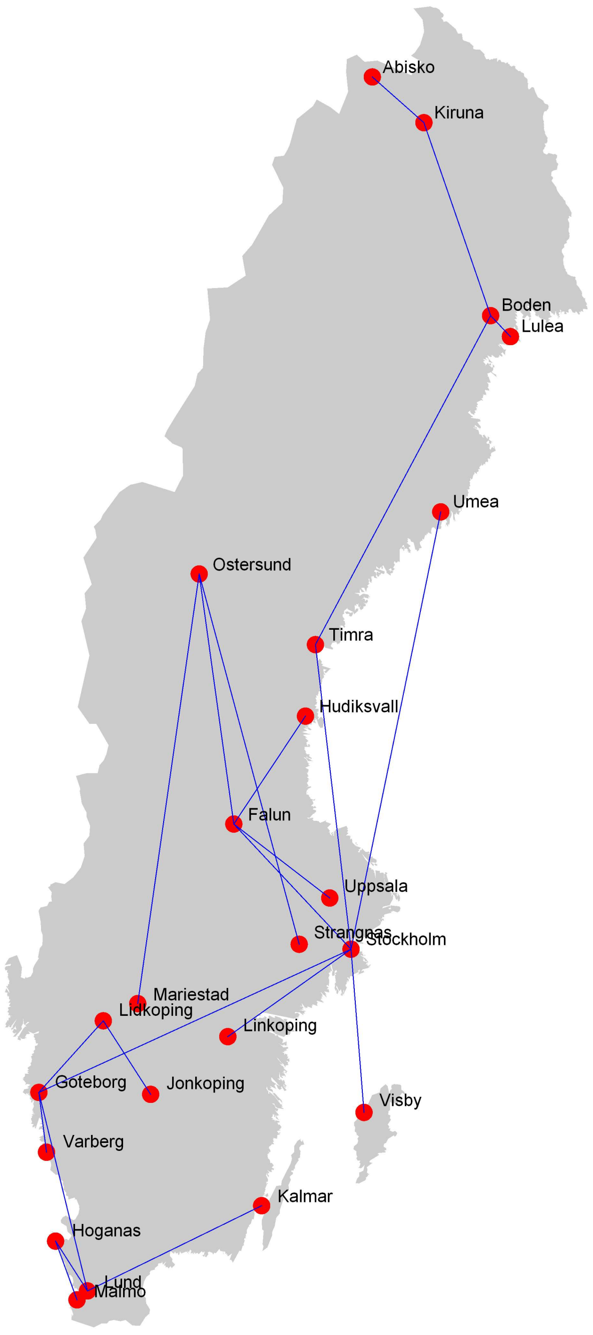


If a link stops working

What will happen if the link between Stockholm and Lund stops working?

When the link stops working the shortest path tree will split up into two subtrees and all routers (cities) in the second subtree will also lose their shortest path to Stockholm,
in other words they would lose the link which connected them to Stockholm.
We now need to find an alternative shortest path for every router not connected to Stockholm.

We will the same algorithm to construct an alternative shortest path spanning tree which does not include the link between Stockholm and Lund:



We can see above that The shortest path from Stockholm to Lund is now:
Stockholm → Göteborg → Lund

We can also see that the other routers who lost connection to Stockholm now have other shortest paths.

Code