1. Introduction

The topics covered in the assignment is linked-state routing, which is part of the most used routing protocol for Internet routers. Routing can be found in all networks, since it is a required function for communication over the Internet, i.e. sending packages over the internet. Routing is the process where an algorithm decides the shortest path for the route of a package through a whole network of nodes (i.e. routers). This is done by doing router hops one by one in the network and for each hop, the algorithm calculates the next hop, always considering the shortest path. Based on the traffic of the network, the algorithm dynamically updates the information about the network in order to detect failures or changes - in this case, we update the algorithm manually. As mentioned, in this assignment we are using link-state routing, which is a type of routing functionality. Link-state routing is calculating the best route for router A to B, where A stores information about its neighbouring routers. (Coulouris, Dollimore & Kindberg, 2012)

2. Main problems encountered

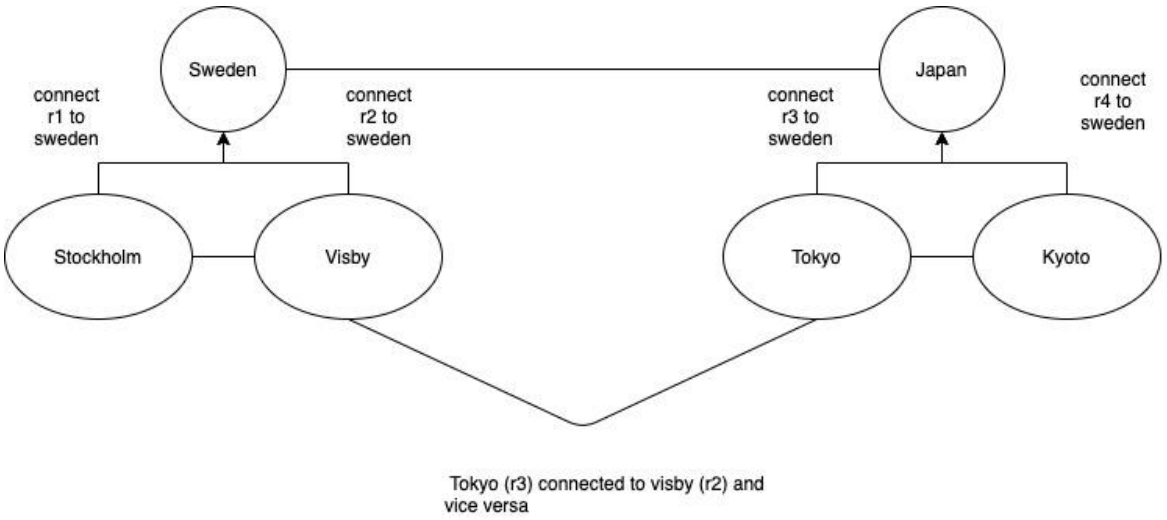
In this assignment, there was less code provided in the instructions. This was time and energy consuming, yet fun for the problem solving aspect. The main problems faced were:

- Lists module
    - Initially it took some time to figure out how lists' functions operate, especially the foldl/3 function since I have no experience in accumulators nor similar in-built functions in different programming languages. Hence, other solutions were tried using tail recursion instead:

```
all_nodes(Map) ->
        case Map of
                [] ->
                        [];
                [{H, [K,V]} | T ] ->
                        [H, K, V | all_nodes(T)]
        end.
```

    - The function outputs:
    - >map:all_nodes([{berlin,[london,paris]}]).
    - [paris,london,berlin]

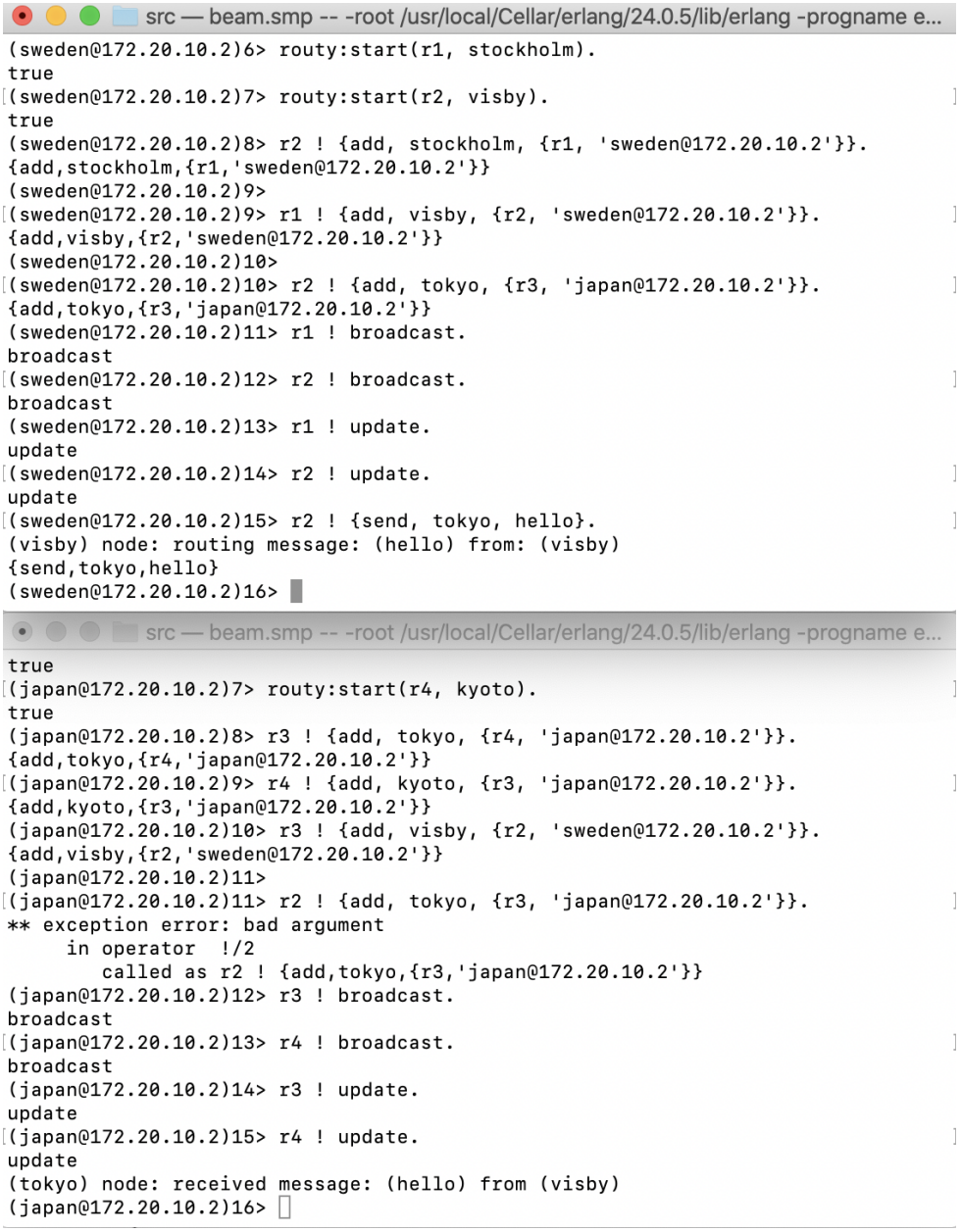
- Task 5
    - This was a bit overwhelming, a lot of code and theory to interpret and understand. But after breaking down everything into smaller pieces, it was feasible. Drawing graphs completely facilitated the process and to get an understanding of what was going on. The following graph was drawn.



- We see that each country consists of several routers who are connected. In order to send a message to another country, we must connect two routers from each country, viewed above. This will be demonstrated in the presentation.
- Once the connection between the routers is added successfully, we can soon start to send messages between the countries. First, we need to write the following commands for each involved router:
    - routerName ! broadcast.
        - This is done in order to check what information the router has about itself. This is because it is necessary for the router's incoming or sending messages - it has to know what other routers it is connected to. It also provides necessary details for the map, which is used to find nodes directly connected to a given node.
    - routerName ! update.
        - First time run, this command computes a routing table for each router. The routing table consists of information regarding which way to forward traffic, e.g. messages. This command also updates the routing table. The routing table gives information about how a node (e.g. Sweden) is reached by other nodes (Japan) - e.g. which gateway (city) should be used to reach the node.
    - routerName ! {send, tokyo, "hello"}
        - Now, one should be able to send messages in between the connected countries and cities. See part 3 for screenshots of the process.
- iteration()
- route()
- table()
- hist:New(Name), hist:Update(Name, N, History)
    - It was not clear what 'a new history' is, what a message actually is nor located, where N is located and what to compare N with.
- router(Name, 0, Msgs, Intf, Table, Map)
    - It was not clear where this function was implemented. Initially, it looked like an abstract data type *or* an instance of an object.
- Connecting the dots
    - Probably the most challenging part was to fully interpret the instructions and code.

3. Evaluation

Each terminal should be considered as a country. In this case, we are considering Sweden and Japan. One can see that the procedure described above works, since Tokyo has received a message from Visby seen below.



One problem encountered. If Visby sends a message to Kyoto, the message does not show up in Japan's shell, although {send,kyoto,hello} is returned by the program. This is strange, since the Kyoto router is linked with Visby through Tokyo. Sending a message to a non-existing node, e.g. berlin, seems not to be handled by the program (seen below), which could be a future improvement.

> r3 ! {send, berlin, hello}.
(tokyo) node: routing message: (hello) from: (tokyo)
{send,berlin,hello}

4. Conclusion

Link-state routing is challenging, and so was the assignment. Although some parts are managed manually in the assignment, the program still computes most of the functionality automatically which means that a lot of the link-state routing complexity was implemented. It was interesting and fun to dig further into Erlang, and it was especially useful to implement dijkstra's algorithm. The assignment definitely concretized and clarified the concept of link-state routing and its associated terms.

Reference

Coulouris, George F., Dollimore, J., Kindberg, T., Computer scientist. (2012). *Distributed systems: Concepts and design* (5., International ed.). Pearson Education ; Addison-Wesley.