Report - Homework 2

Emil Karlsson

September 29, 2022

1 Introduction

This homework included implementing pathfinding, link state propagation, and interprocess communication.

2 Prestudy

Many aspects of the Erlang programming language had to be studied prior to starting this homework. This mostly included the Erlang standard library, such as their lists library. It included many useful methods such as *lists:keyfind*, *lists:keydelete*, and especially *lists:foldl*. The latter method was especially useful considering the functional nature of Erlang, as this provided a for-each like alternative.

3 Method

The Dijkstra algorithm was the more challenging part of the homework. Initially, the idea behind the solution was firstly based on a iterative solution including queues from previous programming languages. However, this was quickly changed as such as solution would be cumbersome. The implementation used as instead a sort of scan, which attempted to updated the entire list with a *proposed* better path. The key part to the algorithm was how the list of routes was sorted prior to Dijkstra's algorithm. Specifically it was sorted based on the cost, which in this context was hops.

Further, a design choice was made where the method *entry/2* returned 0 if a node was found. This was done since a route should not be updated if it is not reachable. Furthermore, if this would have been for example *inf*, it would have lead to an immediate override, since a high hop count, for all the program knows, indicates a bad route. Finally, using 0 as a return value, only two branchen was needed in the if statement:

```
update(Node, N, Gateway, Sorted) ->
  Length = entry(Node, Sorted),
  if
    N < Length -> replace(Node, N, Gateway, Sorted);
    true -> Sorted
  end.
```

4 Discussion

This homework proved to be far more challenging than the previous homework. This was due to the fact that a bottom-up approach was used, rather than top-down, similar to the previous homework. This caused some questions regarding the choice of data structures, such as *map*.

5 Result

A network was setup as:

```
        |--> Stockholm   <--|
Lund <--|                   |--> Kiruna
        |--> Gothenburg  <--|
```

and a message was sent from Kiruna to Lund, forcing a forwarding to take place in either Gothenburg or Stockholm. Then killing the chosen router was killed, an every remaining node were updated.

```
routy:start(r1, kiruna).
routy:start(r2, stockholm).
routy:start(r3, gothenburg).
routy:start(r4, lund).

r1 ! {add, gothenburg, r2}.
r1 ! {add, stockholm, r1}.

r2 ! {add, kiruna, r1}.
r2 ! {add, lund, r4}.

r3 ! {add, kiruna, r1}.
r3 ! {add, lund, r4}.

r4 ! {add, gothenburg, r2}.
r4 ! {add, stockholm, r1}.
```

```
r1 ! broadcast.
r2 ! broadcast.
r3 ! broadcast.
r4 ! broadcast.

r1 ! update.
r2 ! update.
r3 ! update.
r4 ! update.

-----

r1 ! {send, lund, hellomyfriend}.

[ROUTE] (kiruna -> goteborg) hellomyfriend
[RECEIVE] (goteborg -> lund) hellomyfriend

%% Disrupting the path that was chosen
r2 ! stop

%% Calculating new route
r1 ! broadcast.
r3 ! broadcast.
r4 ! broadcast.

r1 ! update.
r3 ! update.
r4 ! update.

r1 ! {send, lund, hellomyfriend}.

[ROUTE] (kiruna -> stockholm) hellomyfriend
[RECEIVE] (stockholm -> lund) hellomyfriend
```