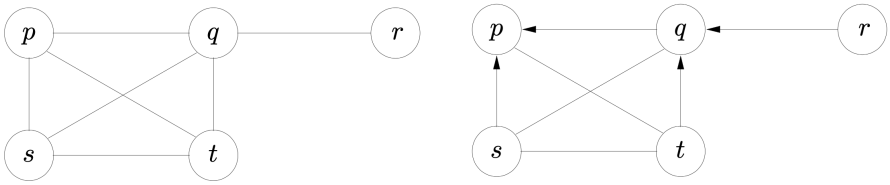
CS 511 – Quiz 7: Message Passing in Erlang

16 November 2022

Names:
Pledge:

Exercise 1

The following distributed algorithm called *echo* computes the spanning tree of a network[1]. One node in the network, called the *initiator*, starts the process. The initiator starts by sending a message to all its neighbors. Intuitively, these messages travel in all directions, and bounce back from the corners of the network toward the initiator. This is achieved as follows. When a noninitiator receives a message for the first time, it makes the sender its parent, and sends a message to all neighbors except its parent. When a noninitiator has received messages from all its neighbors, it sends a message to its parent. Finally, when the initiator has received messages from all its neighbors, it prints the result.



A sample execution could be:

- p sends messages to q, s, and t.
- p's message arrives at q, who makes p its parent and sends messages to r, s, and t.
- q's message arrives at t, who makes q its parent and sends messages to p and s.
- q's message arrives at r, who makes q its parent. Since r has no other neighbors, it sends a message to its parent q straightaway.
- p's message arrives at s, who makes p its parent and sends messages to q and t.
- p's and s's message arrive at t, who sends a message to its parent q.
- r's, s's, and t's message arrive at q, who sends a message to its parent p.
- q's and t's message arrive at s, who sends a message to its parent p.
- q's, s's, and t's message arrive at p, who decides.

Sample output from the erlang shell:

```
> echo:start().
P is <0.408.0>
Q is <0.409.0>
R is <0.410.0>
S is <0.411.0>
T is <0.412.0>
done [{<0.408.0>,parent_of,<0.409.0>},{<0.408.0>,parent_of,<0.411.0>},
{<0.409.0>,parent_of,<0.412.0>},{<0.409.0>,parent_of,<0.410.0>}]
```

[1]It is part of a class of algorithms known as *wave algorithms*, see: Distributed Algorithms: An Intuitive Approach (The MIT Press) Wan Fokkink, 2013

Here is the stub you have to complete.

```
-module(echo).
-compile(export_all).

start() ->  %% Set up sample network
    P = spawn(?MODULE,setup_neighbors,[initiator]),
    Q = spawn(?MODULE,setup_neighbors,[noninitiator]),
    R = spawn(?MODULE,setup_neighbors,[noninitiator]),
    S = spawn(?MODULE,setup_neighbors,[noninitiator]),
    T = spawn(?MODULE,setup_neighbors,[noninitiator]),
    io:format("P is ~p~n",[P]), io:format("Q is ~p~n",[Q]),
    io:format("R is ~p~n",[R]), io:format("S is ~p~n",[S]),
    io:format("T is ~p~n",[T]),
    P![Q,S,T],
    Q![P,S,T,R],
    R![Q],
    S![P,Q,T],
    T![S,P,Q],
    ok.

setup_neighbors(Init) ->
    receive
        NPIDs ->
            case Init of
                initiator ->  %% Initiator node: send first wave
                    [ Pid!{self(),[]} || Pid <- NPIDs ];
                noninitiator ->    %% Noninitiator node
                    ok
            end,
            loop(Init,NPIDs,0,undef,[])  %% start main node loop
    end.

%% Noninitiator node and parent undef
%% - Wait for first message.
%% - Set parent
%% - If there is more than one neighbor, send wave forward to them (except parent)
%% - Otherwise, send wave back to parent
loop(noninitiator,NPIDs,0,undef,[]) ->
    receive
        {From,_W} ->
            if
                length(NPIDs)>1 ->  %% have more than one neighbor, send wave forward
                    [ Nei!{self(),[]} || Nei <- lists:delete(From,NPIDs) ],
                    loop(noninitiator,NPIDs,1,From,[]);
                true ->             %% only one neighbor, send wave to parent
                    From!{self(),[{From,parent_of,self()}]}
            end
    end;
%% Noninitiator node with parent already defined
%% - receive {_From,W} message
%% - if got replies from all neighbors, send parent [{Parent,parent_of,self()} | W++Tree]
%% - otherwise, wait for more messages (recursive call)
loop(noninitiator,NPIDs,Rec,Parent,Tree) ->  complete;
%% Initiator node
%% - receive {_From,W} message
%% - if got replies from all neighbors, print out spanning tree (io:format("done ~w~n",[W++Tree]
%% - otherwise, wait for more messages (recursive call)
loop(initiator,NPIDs,Rec,undef,Tree) ->  complete.
```