```
-module(dp10a_gy7).
-author('patai@iit.bme.hu, hanak@inf.bme.hu').
-vsn('2010-12-12').
-compile(export_all).

%%                    Deklaratív programozás, 7. gyakorlat
%%                         Erlang programozás: gráfok
%%                             2010. 12. 06. & 08.

%% Élsúlyozott irányított gráfban legrövidebb út keresése

% @type node()   = atom().
% @type weight() = integer().
% @type graph()  = [{node(), [{node(), weight()}]}].

% körmentes irányított gráf (DAG)
test_graph(0) ->
    [{a, [{b, 10}, {c, 5}]}, {c, [{b, 2}]}, {b, [{d, 1}]}];

% kört tartalmazó irányított gráf
test_graph(1) ->
    [{a, [{b, 10}, {c, 5}]}, {c, [{b, 2}]}, {b, [{d, 1}, {a, 8}]}].

% körmentes irányított gráf (DAG)
test_graph_2(0) ->
    [{a, b, 10}, {a, c, 5}, {c, b, 2}, {b, d, 1}];

% kört tartalmazó irányított gráf
test_graph_2(1) ->
    [{a, b, 10}, {a, c, 5}, {c, b, 2}, {b, d, 1}, {b, a, 8}].

% M=dp10a_gy7.
% M:shortest_route(M:test_graph(0), b, c) == 'No route'.
% M:shortest_route(M:test_graph(1), b, c) == 13.

% @spec shortest_route(G::graph(),S::node(),T::node()) -> W::weight()
% @doc  S-ből T-be a legrövidebb út költsége a G-ben W;
%   ha nincs ilyen, W = 'No route'
shortest_route_1(Graph, StartNode, TargetNode) ->
    case lists:keysearch(TargetNode, 1,
                         lists:sort(reached(Graph, {StartNode, 0}, [StartNode]))
                        ) of
        {value, {_, Value}} -> Value;
%       {value, {_, Value, Path}} -> {Value, lists:reverse(Path)};
        _ -> 'No route'
    end.

shortest_route(Graph, StartNode, TargetNode) ->
    Ls = [W || {S,W} <- reached(Graph, {StartNode, 0}, [StartNode]),
               S == TargetNode
         ],
    lists:foldl(fun erlang:min/2, 'No route', Ls).
```



```
% @spec reached(G::graph(),{N::node(),W0::weight()},VNs::[node()]) ->
%                                                  [{T::node(),W::weight()}]
% @doc  T elérhető N-ből G-ben a CL-beli csomópontok érintése nélkül
% W-W0 költséggel
reached(Graph, {Node, Weight}, VisitedNodes) ->
    {_, Succs} = case lists:keysearch(Node,1,Graph) of
                     {value, Tuple} -> Tuple;
                     false -> {false,[]}
                 end,
    [{Node, Weight} |
%    [{Node, Weight, VisitedNodes} |
%     [ R || {N, Succs} <- Graph, N =:= Node,
%            {Succ, W0} <- Succs,
     [ R || {Succ, W0} <- Succs,
            not(lists:member(Succ, VisitedNodes)),
            R <- reached(Graph, {Succ, W0 + Weight}, [Succ|VisitedNodes])
     ]
    ].


% @type graph_2()  = [{node(), node(), weight()}].

% @spec reached_2(G::graph_2(),{N::node(),W0::weight()},VNs::[node()]) ->
%                                                  [{T::node(),W::weight()}]
% @doc  T elérhető N-ből G-ben a CL-beli csomópontok érintése nélkül
% W-W0 költséggel
reached_2(Graph, {Node, Weight}, VisitedNodes) ->
    [{Node, Weight} |
%    [{Node, Weight, VisitedNodes} |
     [ R || {N, Succ, W0} <- Graph,
            N =:= Node,
            not(lists:member(Succ, VisitedNodes)),
            R <- reached_2(Graph, {Succ, W0 + Weight}, [Succ|VisitedNodes])
     ]
    ].


shortest_route_2(Graph, StartNode, TargetNode) ->
    case
        [W || {S,W} <- reached_2(Graph, {StartNode, 0}, [StartNode]),
              S == TargetNode
        ] of
        [] -> 'No route';
        Ls -> lists:min(Ls)
    end.

% M=dp10a_gy7.
% G0 = M:test_graph(0).
% G1 = M:test_graph(1).
% M:reached(G0,{b,0},[b]).
% G20 = M:test_graph_2(0).
% G21 = M:test_graph_2(1).
% M:reached_2(G0,{b,0},[b]).
```