Assignment – 2

Features:-

- Modularity of code – Avoided code duplication
- Implemented Breadth First Search also.
- Reduced Complexity – avoided by not sorting repeatedly, instead just finding the minimum element in case of Best-First Search.
- Tried to use inbuilt functions as much as possible.
- Well Commented Code.
- Used euclidean distance for heuristic.

Code

```
:- retractall(edges(_,_,_)), retractall(predecessor(_,_)), retractall(heuristic(_, _, _)).
:- consult("edges.pl").
:- consult("heuristic.pl").


indexOf([Element | _], Element, 0) :- !.
indexOf([_ | Tail], Element, Index) :-
        indexOf(Tail, Element, Index1),
        !,
        Index is Index1 + 1.


getMinElement(Open, H, Destination) :-
        findall(Heurstic_Cost, (member(S, Open), heuristic(S, Destination , Heurstic_Cost)),
List_of_heurstic_cost_to_destination),
        min_list(List_of_heurstic_cost_to_destination, M),
        indexOf(List_of_heurstic_cost_to_destination, M, Index),
        nth0(Index, Open, H1),
        H1 = H.


setLink(_, []).
setLink(S, [H | T]) :-
        retractall(predecessor(_, H)),
        asserta(predecessor(S, H)),
        setLink(S, T).


retracepath(Destination, Source, Cost) :-
        (
                (
                        \+ predecessor(_, Destination),
                        write("Cost = "),
                        write(Cost),
                        writeln(" KMs"),
                        write(Source),
                        !
                );
                (
                        predecessor(X, Destination),
                        edges(X, Destination, C),
```

```
                NewCost is Cost + C,
                retracepath(X, Source, NewCost),
                write(" --> "),
                write(Destination)
        )
    ).


search(Source, Source, _, _) :- writeln("You are at Destination only!"), !.
search(Source, Destination, Open, Closed, Choice) :-
    Source \= Destination,
    (
        (
            Open = [],
            writeln("No path found!"),
            !
        );
        (
            (
                (
                    Choice = 2,
                    getMinElement(Open, S, Destination)
                % get node, S from Open with minimum cost to Destination (Best
First Search)
                );
                (
                    nth0(0, Open, S)
                    % get first node, S from Open
                )
            ),
            select(S, Open, Open1),
                % remove node, S from Open
            append(Closed, [S], Closed1),
                % insert node, S in Closed

            findall(Successor, edges(S, Successor, _), ListOfSuccessors),          %
generate all successors of S
            subtract(ListOfSuccessors, Closed1, ListOfUnvisitedSuccessors),        %
remove all successors which are already visited
            (
                setLink(S, ListOfUnvisitedSuccessors),
            % Set link from successors of S to S
                (
                    member(Destination, ListOfUnvisitedSuccessors),
    % some successor of S is Destination
                    writeln("Path Found!"),
                    retracepath(Destination, Source, 0),
            % retrace back the path
                    !
                );
                (
```

```
                                        (
                                                (
                                                        Choice = 3,
                                                        append(Open1, ListOfUnvisitedSuccessors,
Open2)          % append successors in end of Open (Breadth First Search)
                                                );
                                                (
                                                        append(ListOfUnvisitedSuccessors, Open1,
Open2)          % append successors in front of Open
                                                )
                                        ),
                                        search(Source, Destination, Open2, Closed1, Choice)
                        % repeat
                                )
                        )
                )
        ).


query :-
        writeln("1.) for Depth First Search"),
        writeln("2.) for Best First Search"),
        writeln("3.) for Breadth First Search"),
        writeln("Enter Your Choice :"),
        read(Choice),
        (
                (
                        member(Choice, [1, 2, 3]),
                        writeln("Enter Source :"),
                        read(Source),
                        writeln("Enter Destination :"),
                        read(Destination),
                        search(Source, Destination, [Source], [], Choice)
                );
                (
                        query
                )
        ).

:- query.
```

Output

```
?- consult('main.pl').
1.) for Depth First Search
2.) for Best First Search
3.) for Breadth First Search
Enter Your Choice :
|: 1.
Enter Source :
|: amritsar.
Enter Destination :
|: bhopal.
Path Found!
Cost = 2676 KMs
amritsar --> pune --> bhopal
true.

?- consult('main.pl').
1.) for Depth First Search
2.) for Best First Search
3.) for Breadth First Search
Enter Your Choice :
|: 2.
Enter Source :
|: amritsar.
Enter Destination :
|: bhopal.
Path Found!
Cost = 1449 KMs
amritsar --> indore --> bhopal
true.

?- consult('main.pl').
1.) for Depth First Search
2.) for Best First Search
3.) for Breadth First Search
Enter Your Choice :
|: 3.
Enter Source :
|: amritsar.
Enter Destination :
|: bhopal.
Path Found!
Cost = 2676 KMs
amritsar --> pune --> bhopal
true.
```

```
?- consult('main.pl').
1.) for Depth First Search
2.) for Best First Search
3.) for Breadth First Search
Enter Your Choice :
|: 1.
Enter Source :
|: agra.
Enter Destination :
|: baroda.
Path Found!
Cost = 1759 KMs
agra --> pune --> baroda
true.

?- consult('main.pl').
1.) for Depth First Search
2.) for Best First Search
3.) for Breadth First Search
Enter Your Choice :
|: 2.
Enter Source :
|: agra.
Enter Destination :
|: baroda.
Path Found!
Cost = 1629 KMs
agra --> chandigarh --> baroda
true.
```