# Artificial Intelligence

## Assignment -2 Report

Kaushik Dey

MT22034

## Contents



## Logic of the Program

Step1: We start the program with csv rule and give the parameter as the input file name. Thereafter the user has to enter dosearch. to start the search operation.

Step2: The user will be given two options depth first search or best first search. The user needs to enter dfs for depth first search and bfs for best first search.

Step3: If the user enters dfs or bfs the user will be asked to enter the source and destination name.

Step4: The program outputs the path according to the dfs or bfs the user enters.

## Explanation of the code

```
(start) csv_ (FILE0)
:-
csv:csv_read_file(FILE0,[HEADER|ROWss]) ,
row__to__list(HEADER,HEADERs) ,
(loop) csv_ (HEADERs,ROWss)
.

(loop) csv_ (_HEADERs,[])
:-
true
.

(loop) csv_ (HEADERs,[ROW|ROWss])
:-
row__to__list(ROW,ROWs) ,
lists:nth1(1,ROWs,CITY_A) ,
QUERY_A=(lists:nth1(NTH,ROWs,DISTANCE)) ,
QUERY_B=(NTH > 1) ,
QUERY_C=(lists:nth1(NTH,HEADERs,CITY_B)) ,
QUERY=(QUERY_A,QUERY_B,QUERY_C) ,
ASSERT=assertz(distance(CITY_A,CITY_B,DISTANCE)) ,
forall(QUERY,ASSERT) ,
ASSERTA=assertz(distance(CITY_B,CITY_A,DISTANCE)) ,
forall(QUERY,ASSERTA) ,
ASSERTB=assertz(heuristics(CITY_A,CITY_B,DISTANCE-200)) ,
forall(QUERY,ASSERTB) ,
ASSERTC=assertz(heuristics(CITY_B,CITY_A,DISTANCE-200)) ,
forall(QUERY,ASSERTC) ,

(loop) csv_ (HEADERs,ROWss)
.
```

Here we see that the facts were dynamically generate from the csv file along with the heuristics , Here I have taken the heuristics to be 200 less than actual distance .

```
dosearch :-
      write('Enter depth first search or best first search'),nl,
      read(Search),
      method(Search).

method(dfs) :-
    write('Enter the source'),
    read(Source),
    write('Enter the destination'),
    read(Destination),
    append([],[Source],Open),
    finding(Open,Destination,[]).

finding([H|T],Destination,Closed):-

    add_tail(Closed,H,R),
    findall(X,distance(H,X,_),Z),sort(Z, X),
    (
      list_member(Destination,X) ->
      add_tail(R,Destination,M),print_list(M);
      append(T,X,D),finding(D,Destination,R)
     ).
```

dosearch rule takes the input search technique and then method dfs rule is called . The method(dfs) rule takes the source and destination as input and then appends the source to an open list and finding rule then adds the head element of the open list to closed list and then findall method calls all the matching facts and then list_member checks whether the destination is within the reachable distance or not. If yes then the closed list is printed or else recursively current locations are appended at the beginning and finding is called .

```
method(bfs) :-
    write('Enter the source'),
    read(Source),
    write('Enter the destination'),
    read(Destination),
    append([],[Source],Open),
    findingb(Open,Destination,[]).

findingb([H|T],Destination,Closed):-

    add_tail(Closed,H,R),
    findall(X,heuristics(H,_,X),Z),sort(Z,O),removeAll(0,O,W),check0(W,H,Destination,R,T).

check0(W,H,Destination,R,T) :-
    findall(G,heuristics(H,G,_),Z),sort(Z,X),
    (
        list_member(Destination,X) ->
        add_tail(R,Destination,M),print_list(M);
        check(W,H,Destination,R,T)
    ).
check([L|P],H,Destination,R,T):-
     findall(G,heuristics(H,G,L),F),sort(F,X),
     (
      common_elements(R,X) ->
      check(P,H,Destination,R,T);
      list_member(Destination,X) ->
      add_tail(R,Destination,M),print_list(M);
      print_list(X),append(X,T,D),findingb(D,Destination,R)
     ).
```

---

If the user inputs bfs technique then the method(bfs) is called and then the same procedure continues . here the findingb rule is executed and only those locations are chosen with least heuristics value. If common elements are found it is ignored and if location is found it is printed.

```
insert(X, [], [X]).
insert(X, [H|T], [H|U]):-
    insert(X, T, U).

add_tail([],X,[X]).
add_tail([H|T],X,[H|L]):-add_tail(T,X,L).

print_list([]):-nl.
print_list([H|T]):-write(H),write(->),print_list(T).


membereq(X, [H|_]) :-
    X == H.
membereq(X, [_|T]) :-
    membereq(X, T).


common_elements([H|_], L2) :-
    membereq(H, L2).
common_elements([_|T], L2) :-
    common_elements(T, L2).

removeAll(_, [], []).
removeAll(X, [X|T], L):- removeAll(X, T, L), !.
removeAll(X, [H|T], [H|L]):- removeAll(X, T, L ).
```

insert rule inserts the element in the head of the list , add_tail rule adds the rule at the end of the list and common_elements checks whether two lists have any common elements and removeAll removes all the occurrences of X in the list (it is used to remove all the occurences of zero distance .).

# Prolog features used in the program

- Facts and Rules

```
(loop) csv_ (HEADERs,[ROW|ROWss])
:-
row__to__list(ROW,ROWs) ,
lists:nth1(1,ROWs,CITY_A) ,
QUERY_A=(lists:nth1(NTH,ROWs,DISTANCE)) ,
QUERY_B=(NTH > 1) ,
QUERY_C=(lists:nth1(NTH,HEADERs,CITY_B)) ,
QUERY=(QUERY_A,QUERY_B,QUERY_C) ,
ASSERT=assertz(distance(CITY_A,CITY_B,DISTANCE)) ,
forall(QUERY,ASSERT) ,
ASSERTA=assertz(distance(CITY_B,CITY_A,DISTANCE)) ,
forall(QUERY,ASSERTA) ,
(loop) csv_ (HEADERs,ROWss)
.
```

- Recursion

```
add_tail([],X,[X]).
add_tail([H|T],X,[H|L]):-add_tail(T,X,L).

print_list([]):-nl.
print_list([H|T]):-write(H),write(->),print_list(T).


membereq(X, [H|_]) :-
    X == H.
membereq(X, [_|T]) :-
    membereq(X, T).
```

- Cut

```
removeAll(_, [], []).
removeAll(X, [X|T], L):- removeAll(X, T, L), !.
removeAll(X, [H|T], [H|L]):- removeAll(X, T, L ).
```

- Input/Output

```
method(bfs) :-
    write('Enter the source'),
    read(Source),
    write('Enter the destination'),
    read(Destination),
    append([],[Source],Open),
    findingb(Open,Destination,[]).
```

```
dosearch :-
      write('Enter depth first search or best first search'),nl,
      read(Search),
      method(Search).

method(dfs) :-
    write('Enter the source'),
    read(Source),
    write('Enter the destination'),
    read(Destination),
    append([],[Source],Open),
```

- List

```
method(bfs) :-
    write('Enter the source'),
    read(Source),
    write('Enter the destination'),
    read(Destination),
    append([],[Source],Open),
    findingb(Open,Destination,[]).

findingb([H|T],Destination,Closed):-

    add_tail(Closed,H,R),
    findall(X,distance(H,_,X),Z),sort(Z,O),removeAll(0,O,W),check0(W,H,Destination,R,T).
```

- Assert

```
(loop) csv_ (HEADERs,[ROW|ROWss])
:-
row__to__list(ROW,ROWs) ,
lists:nth1(1,ROWs,CITY_A) ,
QUERY_A=(lists:nth1(NTH,ROWs,DISTANCE)) ,
QUERY_B=(NTH > 1) ,
QUERY_C=(lists:nth1(NTH,HEADERs,CITY_B)) ,
QUERY=(QUERY_A,QUERY_B,QUERY_C) ,
ASSERT=assertz(distance(CITY_A,CITY_B,DISTANCE)) ,
forall(QUERY,ASSERT) ,
ASSERTA=assertz(distance(CITY_B,CITY_A,DISTANCE)) ,
forall(QUERY,ASSERTA) ,
(loop) csv_ (HEADERs,ROWss)
.
```

- discontiguous (to keep facts and rules apart)

```
:-initialization(use_module(library(csv))).
:-initialization(use_module(library(lists))).
:- discontiguous method/1.
:- dynamic(distance/3) .
```

- predefined functions used (findall , sort)

```
        add_tail(Closed,H,R),
        findall(X,distance(H,_,X),Z),sort(Z,O),removeAll(0,O,W),check0(W,H,Destination,R,T).
```

## Output of the program.

```
?-
% c:/Users/LENOVO/OneDrive/Documents/Assignments/Artificial Intelligence/Assignment2/Assignment2.pl compiled 0.00 sec, 1 clauses
?- csv('C:/Users/LENOVO/OneDrive/Documents/Assignments/Artificial Intelligence/Assignment2/road.csv').
true ;
false.

?- dosearch.
Enter depth first search or best first search
|: bfs.
Enter the source|: jamshedpur.
Enter the destination|: kolhapur.
calcutta->
jamshedpur->calcutta->kolhapur->
true .

?- dosearch.
Enter depth first search or best first search
|: dfs.
Enter the source|: jamshedpur.
Enter the destination|: kolhapur.
jamshedpur->ahmedabad->kolhapur->
true.

?- dosearch.
Enter depth first search or best first search
|: dfs.
Enter the source|: jamshedpur.
Enter the destination|: hyderabad.
jamshedpur->hyderabad->
true.

?- dosearch.
Enter depth first search or best first search
|: bfs.
Enter the source|: jamshedpur.
Enter the destination|: hyderabad.
jamshedpur->hyderabad->
true .
```

---

```
?- dosearch.
Enter depth first search or best first search
|: dfs.
Enter the source|: jamshedpur.
Enter the destination|: delhi.
jamshedpur->delhi->
true.

?- dosearch.
Enter depth first search or best first search
|: bfs.
Enter the source|: jamshedpur.
Enter the destination|: delhi.
jamshedpur->delhi->
true .

?- dosearch.
Enter depth first search or best first search
|: dfs.
Enter the source|: jamshedpur.
Enter the destination|: indore.
jamshedpur->indore->
true.

?- dosearch.
Enter depth first search or best first search
|: dfs.
Enter the source|: jamshedpur.
Enter the destination|: asansol.
jamshedpur->ahmedabad->asansol->
true.

?- dosearch.
Enter depth first search or best first search
|: bfs.
Enter the source|: jamshedpur.
Enter the destination|: asansol.
calcutta->
jamshedpur->calcutta->asansol->
true .
```

---