Nishant Kumar / 2017171

Assignment-2

Depth-first search and Best first search

Outputs:

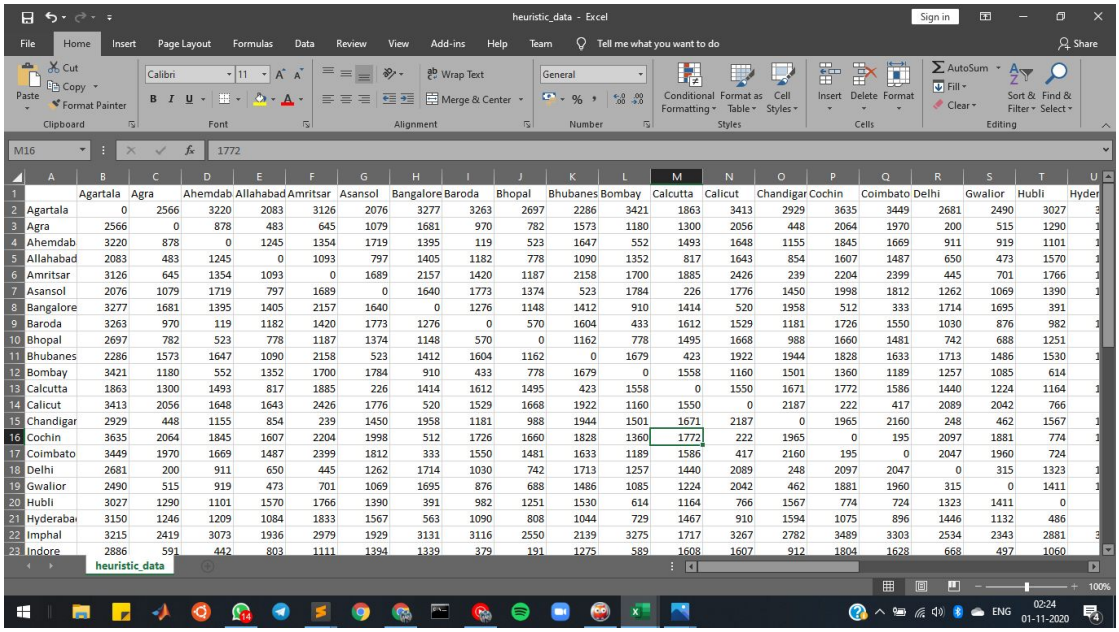
- Path to the destination city from the source city.
- The total cost incurred.

Prolog features used:

- Lists
- Recursion
- Backtracking
- Dynamic assertion
- Read / Write

Creation of heuristics:

- Heuristics are created using the *Dijkstra algorithm* as it always gives the shortest path between any two vertices while maintaining consistency and admissibility.
- The Dijkstra algorithm which is used for heuristics was coded in python and a CSV file was generated from it. Both (the CSV file and the python file) files are present in the zipped submission.
- A snapshot of the heuristic data is given below:



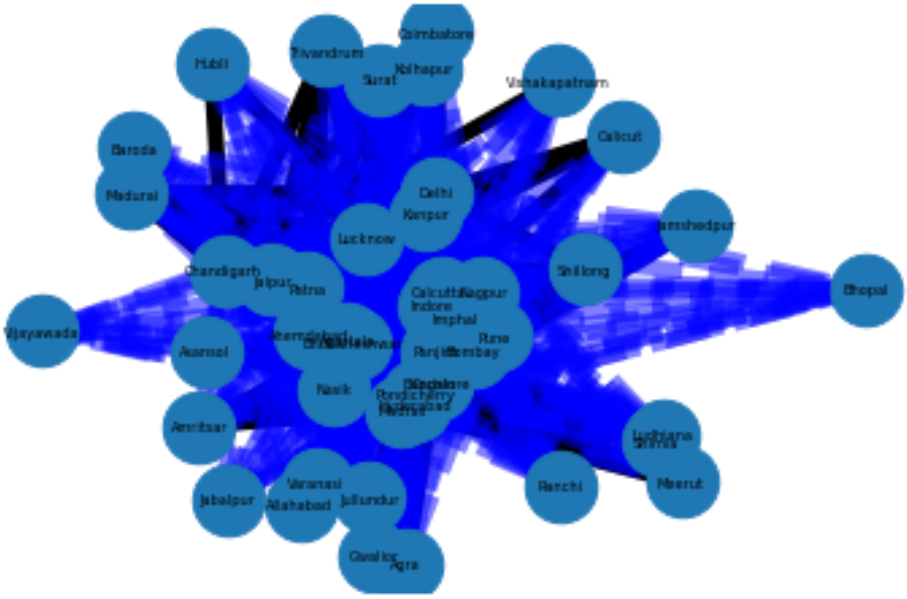
heuristic_data - Excel

	Agartala	Agra	Ahemdab	Allahabad	Amritsar	Asansol	Bangalore	Baroda	Bhopal	Bhubanes	Bombay	Calcutta	Calicut	Chandigar	Cochin	Coimbato	Delhi	Gwalior	Hubli
Agartala	0	2566	3220	2083	3126	2076	3277	3263	2697	2286	3421	1863	3413	2929	3635	3449	2681	2490	3027
Agra	2566	0	878	483	645	1079	1681	970	782	1573	1180	1300	2056	448	2064	1970	200	515	1290
Ahemdab	3220	878	0	1245	1354	1719	1395	119	523	1647	552	1493	1648	1155	1845	1669	911	919	1101
Allahabad	2083	483	1245	0	1093	797	1405	1182	778	1090	1352	817	1643	854	1607	1487	650	473	1570
Amritsar	3126	645	1354	1093	0	1689	2157	1420	1187	2158	1700	1885	2426	239	2204	2399	445	701	1766
Asansol	2076	1079	1719	797	1689	0	1640	1773	1374	523	1784	226	1776	1450	1998	1812	1262	1069	1390
Bangalore	3277	1681	1395	1405	2157	1640	0	1276	1148	1412	910	1414	520	1958	512	333	1714	1695	391
Baroda	3263	970	119	1182	1420	1773	1276	0	570	1604	433	1612	1529	1181	1726	1550	1030	876	982
Bhopal	2697	782	523	778	1187	1374	1148	570	0	1162	778	1495	1668	988	1660	1481	742	688	1251
Bhubanes	2286	1573	1647	1090	2158	523	1412	1604	1162	0	1679	423	1922	1944	1828	1633	1713	1486	1530
Bombay	3421	1180	552	1352	1700	1784	910	433	778	1679	0	1558	1160	1501	1360	1189	1257	1085	614
Calcutta	1863	1300	1493	817	1885	226	1414	1612	1495	423	1558	0	1550	1671	1772	1586	1440	1224	1164
Calicut	3413	2056	1648	1643	2426	1776	520	1529	1668	1922	1160	1550	0	2187	222	417	2089	2042	766
Chandigar	2929	448	1155	854	239	1450	1958	1181	988	1944	1501	1671	2187	0	1965	2160	248	462	1567
Cochin	3635	2064	1845	1607	2204	1998	512	1726	1660	1828	1360	1772	222	1965	0	195	2097	1881	774
Coimbato	3449	1970	1669	1487	2399	1812	333	1550	1481	1633	1189	1586	417	2160	195	0	2047	1960	724
Delhi	2681	200	911	650	445	1262	1714	1030	742	1713	1257	1440	2089	248	2097	2047	0	315	1323
Gwalior	2490	515	919	473	701	1069	1695	876	688	1486	1085	1224	2042	462	1881	1960	315	0	1411
Hubli	3027	1290	1101	1570	1766	1390	391	982	1251	1530	614	1164	766	1567	774	724	1323	1411	0
Hyderaba	3150	1246	1209	1084	1833	1567	563	1090	808	1044	729	1467	910	1594	1075	896	1446	1132	486
Imphal	3215	2419	3073	1936	2979	1929	3131	3116	2550	2139	3275	1717	3267	2782	3489	3303	2534	2343	2881
Indore	2886	591	442	803	1111	1394	1339	379	191	1275	589	1608	1607	912	1804	1628	668	497	1060

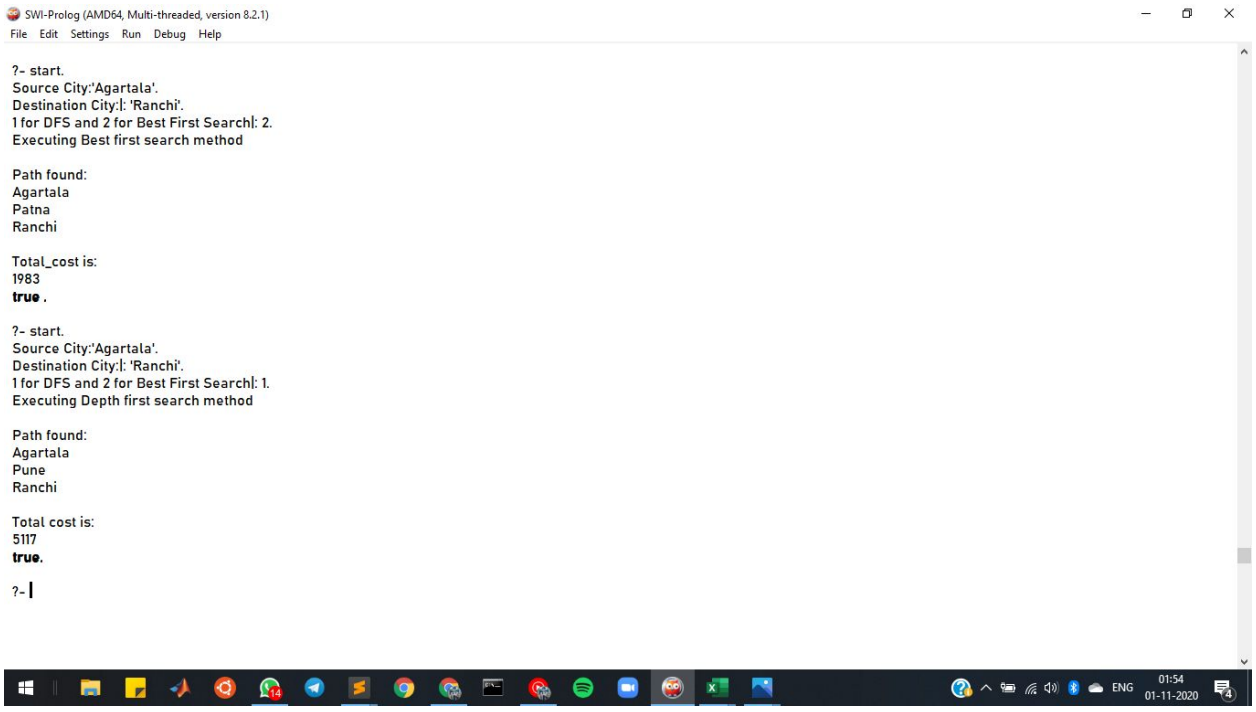
Steps to run the program:

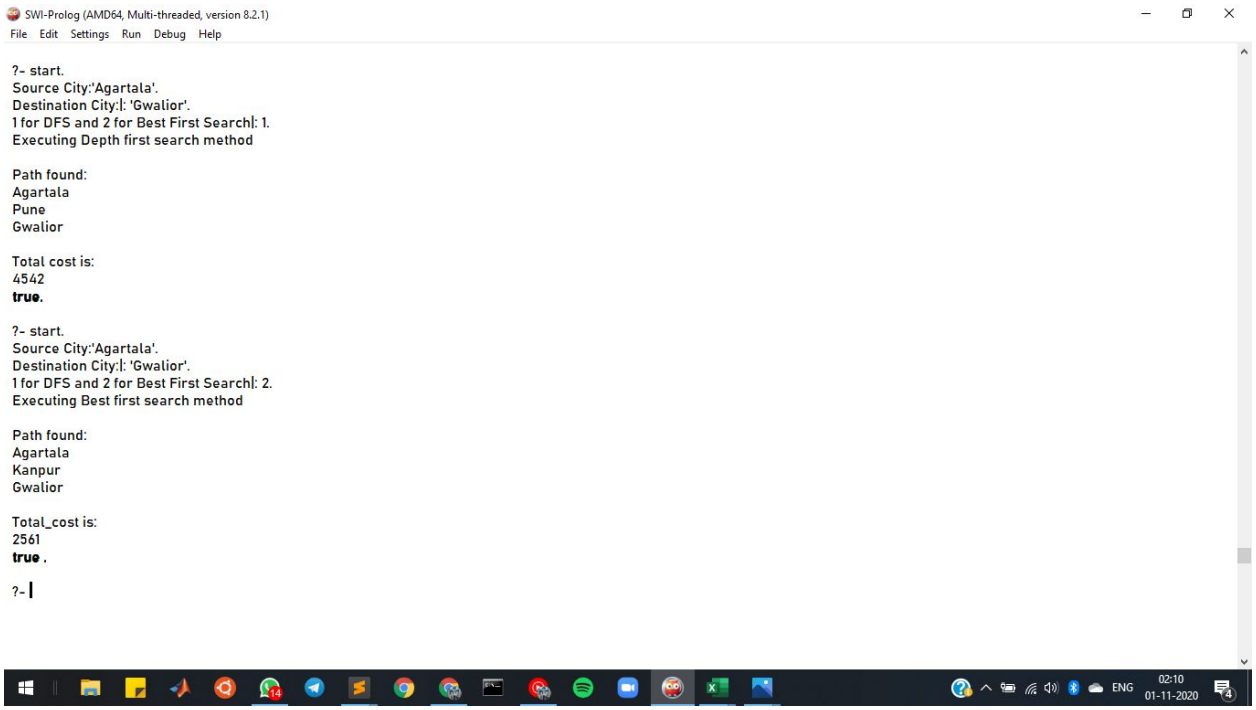
- The program is dynamic and will work for any city present in the knowledge base.
- The main program is in the "*best.pl*" file
- Run the predicates *'check'* and *'check_h'* to create the knowledge base (distances and heuristics).
- '*Start'* predicate controls the entire flow of the system.
- At the source city and the destination city, enter the city name in quotes as '*Agartala', 'Agra',* and end with a full stop and first alphabet as capital.
- Choose 1 for DFS and 2 for the Best first search.

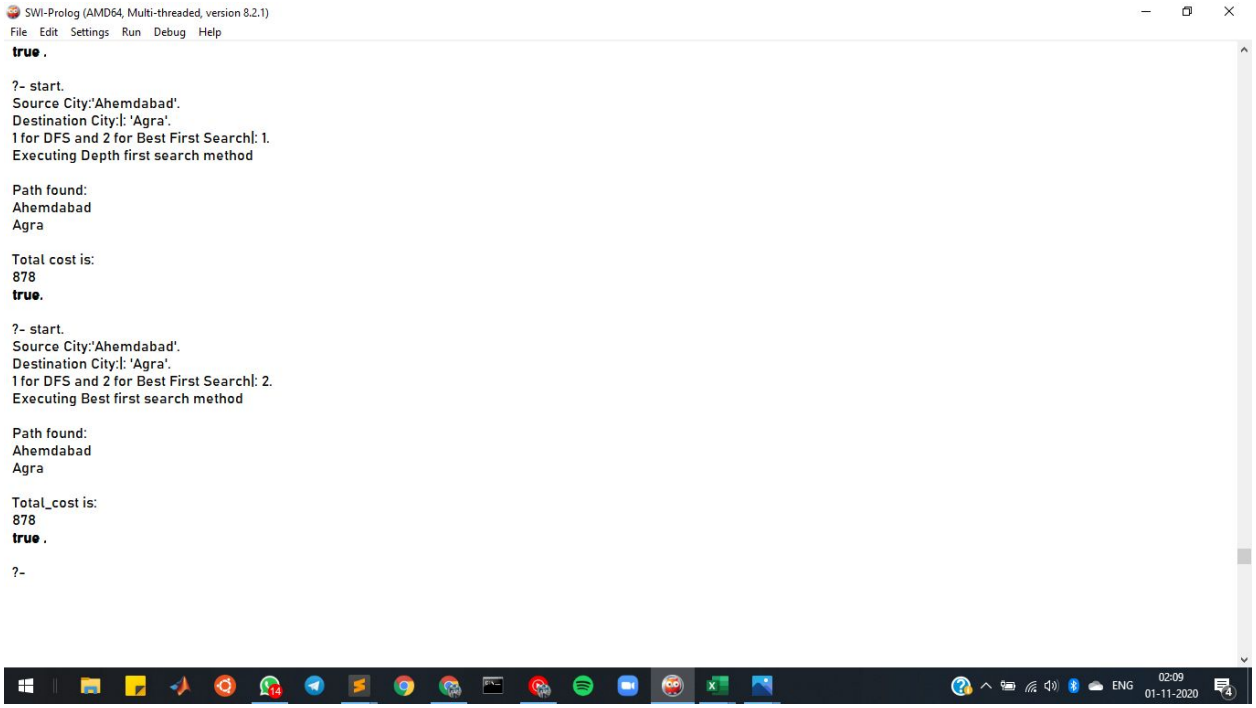
Graph Visualisation:



Screenshots of working of the program:







Source code:

```
%find path controls the whole flow of the initial system.
%State record is kept as [State, Parent, Distance from source, heuristic]
```

```
%[State,Paremt,Distance_from_parent,heuristic value]

:-dynamic(distance_gen/3).
:-dynamic(heuristics/3).
:-[depth].
%-----------------------------------------------------Creating knowledge
base----------------------------------------

check_h:-csv_read_file('C:/Users/hp/Desktop/IIITD/SEM-7/AI/heuristic_data.csv',Rows1,[functor(
heuristic_data), arity(48)]),maplist(assert, Rows1), print(Rows1),
setof([A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,AA,AB,AC,AD,AE,AF,AG,AH,AI,AJ,AK,AL
,AM,AN,AO,AP,AQ,AR,AS,AT,AU,AV],
heuristic_data(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,AA,AB,AC,AD,AE,AF,AG,AH,AI,
AJ,AK,AL,AM,AN,AO,AP,AQ,AR,AS,AT,AU,AV), AZ),nl,nl,nl,print(AZ),nl,nl,nl,extract_header_h(AZ).


assign_head_h([H|T],H).
remove_head_h([_|T],T).
extract_header_h(Data):-
       assign_head_h(Data,H),
       remove_head_h(Data,NewData),
       remove_head_h(H,Header),
       print(Header),nl,nl,nl,
       print(NewData),nl,nl,nl,
       %print_for_single_city(NewData,Header).
       pred_gen_all_city_h(NewData,Header).

extractdistances_h([Cityname|Distances],Cityname,Distances).

pred_gen_single_city_h(_,[]):-
       write('The end').
pred_gen_single_city_h([City1name|[CurDist|OtherDist]],[City2name|Tcity2]):-
       %atom_number(CurDist,CurDist1),
       asserta(heuristics(City1name,City2name,CurDist)),
       asserta(heuristics(City2name,City1name,CurDist)),
       %print(City1name),nl,print(City2name),nl,print(CurDist),nl,
       pred_gen_single_city_h([City1name|OtherDist],Tcity2).


print_for_single_city_h([H|T],Distances):-
       pred_gen_single_city_h(H,Distances).

pred_gen_all_city_h([],_):-
       write('Execution Stopped').

pred_gen_all_city_h([H|T],Distances):-
       pred_gen_single_city_h(H,Distances),
       pred_gen_all_city_h(T,Distances).

%-------------------------------------------------------------------
start:-
       write('Source City:'),
       read(City1),
       write('Destination City:'),
       read(City2),
       write('1 for DFS and 2 for Best First Search'),
       read(X),
       find_option(X,City1,City2).

find_option(1,City1,City2):-
       write('Executing Depth first search method'),nl,nl,
       find_dfs(City1,City2).
```

```
find_option(2,City1,City2):-
       write('Executing Best first search method'),nl,nl,
       find_path(City1,City2).
%-------------------------------------------------------------Best First
code-----------------------------------------

find_path(Start,Goal):-
       empty_closed(Closed),
       empty_open(Open),
       heuristics(Start,Goal,H),
       insert_state_in_open([Start,nil,0,H],Open,New_open),
       search_further(New_open,Closed,Goal).

empty_closed([]).
empty_open([]).

reverse_print_stack(S) :-
       empty_stack(S).
reverse_print_stack(S) :-
       stack([State,_,_,_], Rest, S),
       %reverse_print_stack(Rest),
       write(State), nl.

empty_stack([]).
stack(Start,Es,[Start|Es]).


%3 cases for search :- 1. what if open is empty. 2. What is the next state is goal, 3. more
search requried
search_further(Open,_,_):-
       empty_open(Open),
       write('No path found'),nl.

search_further(Open,Closed,Goal):-
       pop_from_open([Current_state,Parent,Costbob,_],Open,_),
       Current_state=Goal,
       write('Path found:'),nl,
       printsolution_h([Current_state, Parent,_, _],Closed),
       pop_from_open([Last_state,_,Cost,_],Closed,_),
       distance_gen(Last_state,Goal,X),
       Total_cost is X + Cost,nl,
       write('Total_cost is:'),nl,
       print(Costbob),!.
       %reverse_print_stack(Closed),
       %write(Goal).

search_further(Open,Closed,Goal):-
       pop_from_open([Current_state,Parent,D,H],Open,New_open),
       find_children_nodes([Current_state,Parent,D,H],New_open,Closed,Children,Goal),
       insert_list_in_open(Children,New_open,Updated_open),
       insert_in_closed([[Current_state,Parent,D,H]],Closed,New_closed),
       search_further(Updated_open,New_closed,Goal),!.

%define pop_from_open,
insert_in_open,insert_in_closed,find_children_nodes,member_closed,member_open

find_children_nodes([Parent_node,_,Parent_dist, _],Updated_open,Updated_closed,Children,Goal)
:-
       findall(Child, find_child([Parent_node,_,Parent_dist, _],Updated_open, Updated_closed,
Child,Goal),Children).
```

```
find_child([Parent_node,_,Parent_dist, _],
Updated_open,Updated_closed,[Next,Parent_node,New_Dist,H], Goal) :-
       distance_gen(Parent_node,Next,Dist),
       not(member_open([Next,_,_,_],Updated_open)),
       not(member_closed([Next,_,_, _],Updated_closed)),
       New_Dist is Parent_dist + Dist ,
       heuristics(Next, Goal, H).

%-----------------------------------------------HELPER
FUNCTIONS-----------------------------------------------------------

comparator([_,_,_,H1], [_,_,_,H2]):-
       H1=<H2.


insert_state_in_open(Current_state,[],[Current_state]):-!.
insert_state_in_open(Current_state,[H|T],[Current_state,H|T]):-
       comparator(Current_state,H).

insert_state_in_open(Current_state,[H|T],[H|T1]):-
       insert_state_in_open(Current_state,T,T1).

insert_list_in_open([],L,L).
insert_list_in_open([State|T],L,Lnew):-
       insert_state_in_open(State,L,Lmid),
       insert_list_in_open(T,Lmid,Lnew).

member_open(State,Open):-
       member(State,Open).

member_closed(State,Closed):-
       member(State,Closed).

add_if_not_in_set(X, S, S) :-
       member(X, S), !.

add_if_not_in_set(X, S, [X | S]).

insert_in_closed([], S, S).
insert_in_closed([H | T], S, S_new) :-
       insert_in_closed(T, S, S2),
       add_if_not_in_set(H, S2, S_new),!.

pop_from_open(E, [E | T], T).
pop_from_open(E, [E | T], _).

member(X, [X | _]).
member(X, [Y | T]) :- member(X, T).

printsolution_h([State, nil,_, _], _) :-
       write(State), nl,!.

printsolution_h([State, Parent, _, _], Closed_set):-
       not(same(State,Parent)),
       member_closed([Parent, Grandparent, _, _],Closed_set),
       printsolution_h([Parent, Grandparent,_, _],Closed_set),
       write(State), nl.

same(X1,X1).


:-dynamic(distance_gen/3).
```

```
%-----------------------------------------------------Creating knowledge
base----------------------------------------

check:-csv_read_file('C:/Users/hp/Desktop/IIITD/SEM-7/AI/roaddistance1.csv',Rows,[functor(city
dist), arity(21)]),maplist(assert, Rows), print(Rows),
setof([A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U],
citydist(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U),
Z),nl,nl,nl,print(Z),nl,nl,nl,extract_header(Z).


assign_head([H|T],H).
remove_head([_|T],T).
extract_header(Data):-
       assign_head(Data,H),
       remove_head(Data,NewData),
       remove_head(H,Header),
       print(Header),nl,nl,nl,
       print(NewData),nl,nl,nl,
       %print_for_single_city(NewData,Header).
       pred_gen_all_city(NewData,Header).

extractdistances([Cityname|Distances],Cityname,Distances).

pred_gen_single_city(_,[]):-
       write('The end').
pred_gen_single_city([City1name|[CurDist|OtherDist]],[City2name|Tcity2]):-
       %atom_number(CurDist,CurDist1),
       asserta(distance_gen(City1name,City2name,CurDist)),
       asserta(distance_gen(City2name,City1name,CurDist)),
       %print(City1name),nl,print(City2name),nl,print(CurDist),nl,
       pred_gen_single_city([City1name|OtherDist],Tcity2).


print_for_single_city([H|T],Distances):-
       pred_gen_single_city(H,Distances).

pred_gen_all_city([],_):-
       write('Execution Stopped').

pred_gen_all_city([H|T],Distances):-
       pred_gen_single_city(H,Distances),
       pred_gen_all_city(T,Distances).



%-------------------------------------------------------------DFS
CODE-----------------------------------------------------------------

find_dfs(Start,Goal):-
       empty_open(Start_open),
       stack([Start,nil,0],Start_open,Open),
       empty_closed(Closed),
       search(Open,Closed,Goal).

search(Open,_,_):-
       empty_open(Open),
       write('No path found').

search(Open,Closed,Goal):-
       stack([State,Parent,Cost],_,Open),
       State=Goal,
```

```
        write('Path found:'),nl,
        %print(Closed),nl,
        printsolution_dfs([State,Parent,Cost],Closed),nl,
        write('Total cost is:'),nl,
        print(Cost).

search(Open,Closed,Goal):-
        stack([State,Parent,D],Popped_open,Open),
        find_children([State,Parent,D],Popped_open,Closed,Children),
        add_to_stack(Children, Popped_open, Updated_open),
        insert_in_closed([[State,Parent,D]],Closed,Updated_closed),
        search(Updated_open,Updated_closed,Goal),!.

find_children([State,_,D], Rest_open_stack, Closed_set, Children) :-
        findall(Child, moves([State,_,D], Rest_open_stack, Closed_set, Child), Children).

moves([State,_,OldDist], Rest_open_stack, Closed_set, [Next,State,NewDist]) :-
        distance_gen(State,Next,Dist),
        NewDist is OldDist+Dist,
        %Dist is NewDist - OldDist,
        %is(NewDist,+(OldDist,Dist)),
        not(member_stack([Next,_,_], Rest_open_stack)),
        not(member_set([Next,_,_], Closed_set)).


printsolution_dfs([State, nil,_], _) :-
        write(State), nl.

printsolution_dfs([State, Parent,_], Closed_set) :-
        not(same(State,Parent)),
        member_set([Parent, Grandparent,_], Closed_set),
        printsolution_dfs([Parent, Grandparent,_], Closed_set),
        write(State), nl.

%----------------------------------------HELPER
FUNCTIONS-----------------------------------------------------

same(X1,X1).

empty_open([]).

stack(Start,Es,[Start|Es]).

empty_closed([]).

member(X, [X | _]).
member(X, [Y | T]) :- member(X, T).

member_set(State,Closed):-
        member(State,Closed).


add_to_stack(List, Stack, Result) :-
        append(List, Stack, Result).

member_stack(Element, Stack) :-
        member(Element, Stack).

member_closed(State,Closed):-
        member(State,Closed).

add_if_not_in_set(X, S, S) :-
```

```
        member(X, S), !.

add_if_not_in_set(X, S, [X | S]).

insert_in_closed([], S, S).
insert_in_closed([H | T], S, S_new) :-
        insert_in_closed(T, S, S2),
        add_if_not_in_set(H, S2, S_new),!.
```
