EXPERIMENT-25

PROJECT:

Solve the classical travelling salesman problem of AI.

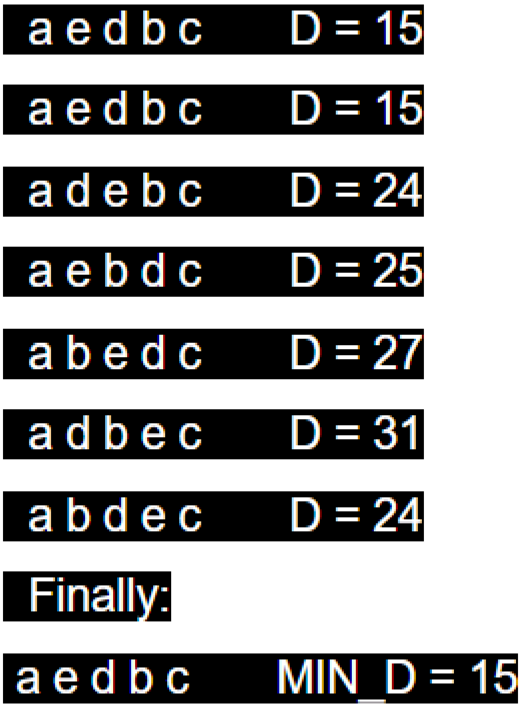
```
domains
/* will allow us cooperate with better names, for me this is like #typedef in C++ */
  town = symbol
  distance = unsigned
  rib = r(town,town,distance)
  tlist = town*
  rlist = rib*
predicates
  nondeterm way(town,town,rlist,distance)
  nondeterm route(town,town,rlist,tlist,distance)
  nondeterm route1(town,tlist,rlist,tlist,distance)
  nondeterm ribsmember(rib,rlist)
  nondeterm townsmember(town,tlist)
  nondeterm tsp(town,town,tlist,rlist,tlist,distance)
  nondeterm ham(town,town,tlist,rlist,tlist,distance)
  nondeterm shorterRouteExists(town,town,tlist,rlist,distance)
  nondeterm alltown(tlist,tlist)
  nondeterm write_list(tlist)
clauses
  /*
```

```
 Nothing special with write_list.

 If list is empty we do nothing,

 and if something there we write head and call ourselves for tail.

 */

 write_list([]).

 write_list([H|T]):-

   write(H,‘ ‘),

   write_list(T).

 /* Is true if town X is in list of towns… */

 townsmember(X,[X|_]).

 townsmember(X,[_|L]):-

   townsmember(X,L).

 /* Is true if rib X is in list of ribs…  */

 ribsmember(r(X,Y,D),[r(X,Y,D)|_]).

 ribsmember(X,[_|L]):-

   ribsmember(X,L).

 /* Is true if Route consists of all Towns presented in second argument */

 alltown(_,[]).

 alltown(Route,[H|T]):-

   townsmember(H,Route),

   alltown(Route,T).


 /* Is true if there is a way from Town1 to Town2, and also return distance
between them */

 way(Town1,Town2,Ways,OutWayDistance):-

   ribsmember(r(Town1,Town2,D),Ways),

   OutWayDistance = D.
```

```
%/*
 /* If next is uncommented then we are using non-oriented graph*/
 way(Town1,Town2,Ways,OutWayDistance):-
   ribsmember(r(Town2,Town1,D),Ways), /*switching direction here…*/
   OutWayDistance = D.
%*/

 /* Is true if we could build route from Town1 to Town2 */
 route(Town1,Town2,Ways,OutRoute,OutDistance):-
   route1(Town1,[Town2],Ways,OutRoute,T1T2Distance),
%SWITCH HERE
   way(Town2,Town1,Ways,LasDist), /* If you want find shortest way comment this line*/
   OutDistance = T1T2Distance + LasDist. /* And make this: OutDistance = T1T2Distance.*/

 route1(Town1,[Town1|Route1],_,[Town1|Route1],OutDistance):-
   OutDistance = 0.
 /* Does the actual finding of route. We take new TownX town and if it is not member of PassedRoute,
 we continue searching with including TownX in the list of passed towns.*/
 route1(Town1,[Town2|PassedRoute],Ways,OutRoute,OutDistance):-
   way(TownX,Town2,Ways,WayDistance),
   not(townsmember(TownX,PassedRoute)),
   route1(Town1,[TownX,Town2|PassedRoute],Ways,OutRoute,CompletingRoadDistance),
   OutDistance = CompletingRoadDistance + WayDistance.
 shorterRouteExists(Town1,Town2,Towns,Ways,Distance):-
    ham(Town1,Town2,Towns,Ways,_,Other),
        Other < Distance.
 /* calling tsp(a,a,…. picks any one connected to a town and calls another tsp*/
 tsp(Town1,Town1,Towns,Ways,BestRoute,MinDistance):-
   way(OtherTown,Town1,Ways,_),
       tsp(Town1,OtherTown,Towns,Ways,BestRoute,MinDistance).
 /*Travelling Salesman Problem is Hammilton way which is the shortes of other ones.*/
 tsp(Town1,Town2,Towns,Ways,BestRoute,MinDistance):-
       ham(Town1,Town2,Towns,Ways,Route,Distance),
       not(shorterRouteExists(Town1,Town2,Towns,Ways,Distance)),
   BestRoute = Route,
   MinDistance = Distance.
 /*Hammilton route from Town1 to Town2 assuming that Town2->Town1 way exists.*/
 ham(Town1,Town2,Towns,Ways,Route,Distance):-
   route(Town1,Town2,Ways,Route,Distance),
%SWITCH HERE
   alltown(Route,Towns), % if you want simple road without including all towns you could uncomment this line
   write_list(Route),
   write(” tD = “,Distance,“n“).
%   fail.


goal
/* EXAMPLE 1
```

```
  AllTowns = [a,b,c,d],

  AllWays = [r(a,b,1),r(a,c,10),r(c,b,2),r(b,c,2),r(b,d,5),r(c,d,3),r(d,a,4)],

*/

/* EXAMPLE 2 */

  AllTowns = [a,b,c,d,e],

  AllWays = [r(a,c,1),r(a,b,6),r(a,e,5),r(a,d,8),r(c,b,2),r(c,d,7),r(c,e,10),r(b,d,3),r(b,e,
9),r(d,e,4)],

  tsp(a,a,AllTowns,AllWays,Route,Distance),

%SWITCH HERE

%  tsp(a,b,AllTowns,AllWays,Route,Distance),

  write(“Finally:n“),

  write_list(Route),

  write(” tMIN_D = “,Distance,“n“)
```

OUTPUT



```
a e d b c     D = 15

a e d b c     D = 15

a d e b c     D = 24

a e b d c     D = 25

a b e d c     D = 27

a d b e c     D = 31

a b d e c     D = 24

Finally:

a e d b c     MIN_D = 15
```