Ahsanullah University of Science and Technology (AUST)
Department of Computer Science and Engineering

Assignment 2

Course No: CSE4108

Course Title: Artificial Intelligence Lab

Date of Submission-
06/06/2023

Submitted To-
Submitted To- Dr. S.M.A. Al-Mamun & Mr. Raihan Tanvir.

Submitted By-
MD Fardin Jaman Aranyak
190204093
B2
Year- 4th
Semester-1st
Department-CSE

<u>Question:</u> Define a recursive procedure in Python and in Prolog to find the length of a path between two vertices of a directed weighted graph.

<u>Solution:</u>

<u>Python Code:</u>

```
def find_path_length(graph, start, end, acc_distance=0):
    if start == end:
        return acc_distance

    if not graph[start]:
        return float('inf')

    min_distance = float('inf')
    for neighbor, weight in graph[start]:
        length = find_path_length(graph, neighbor, end, acc_distance + weight)
        min_distance = min(min_distance, length)

    return min_distance



graph = {
    'i': [('a', 35), ('b', 45)],
    'a': [('c', 22), ('d', 32)],
    'b': [('d', 28), ('e', 36), ('f', 27)],
    'c': [('d', 31), ('g', 47)],
    'd': [('g', 30)],
    'e': [('g', 26)],
    'g': []
}
```

```
path_length = find_path_length(graph, 'a', 'g')



print("Length of the path between 'a' and 'g':", path_length)
```

Prologue:

```
neighbor(i,a,35).
neighbor(i,b,45).
neighbor(a,c,22).
neighbor(a,d,32).
neighbor(b,d,28).
neighbor(b,e,36).
neighbor(b,f,27).
neighbor(c,d,31).
neighbor(c,g,47).
neighbor(d,g,30).
neighbor(e,g,26).

pathLength(X,Y,L):- neighbor(X,Y,L),!.
pathLength(X,Y,L):- neighbor(X,Z,L1), pathLength(Z,Y,L2), L is L1+L2.

findPathLength :- write('Source: '), read(Src),
        write('Destination: '), read(Des),
        pathLength(Src, Des, Length),
        write('Length of the path: '), write(Length), nl.

findPathLength.
```

<u>Question:</u> Modify the Python and Prolog codes demonstrated above to find h_2 and h_3 discussed above

<u>Python Code for H2:</u>

```
# Goal configuration
goal_config = {
    1: (1, 1, 1),
    2: (2, 1, 2),
    3: (3, 1, 3),
    4: (4, 2, 3),
    5: (5, 3, 3),
    6: (6, 3, 2),
    7: (7, 3, 1),
    8: (8, 2, 1)
}

goal_blank = (2, 2)

# Target configuration
target_config = {
    1: (1, 1, 2),
    2: (2, 1, 3),
    3: (3, 2, 1),
    4: (4, 2, 3),
    5: (5, 3, 3),
    6: (6, 2, 2),
    7: (7, 3, 2),
    8: (8, 1, 1)
}
```

```python
target_blank = (3, 1)

# Function to calculate the Manhattan distance between two cells
def manhattan_distance(cell1, cell2):
   x1, y1 = cell1[1:]
   x2, y2 = cell2[1:]
   return abs(x1 - x2) + abs(y1 - y2)

# Function to calculate the heuristic values
def calculate_heuristics():
   heuristics = []
   for tile in range(1, 9):
      tile_goal = goal_config[tile]
      tile_target = target_config[tile]
      distance = manhattan_distance(tile_goal, tile_target)
      heuristics.append(distance)
   return heuristics

# Calculate heuristics for each tile
heuristics = calculate_heuristics()

# Calculate the sum of heuristics
total_heuristic = sum(heuristics)

# Output
print("Heuristics:", total_heuristic)
```

Python Code for H3:

```
graph={
  'Q1':[(6,1)],
  'Q2':[(1,2)],
  'Q3':[(5,3)],
  'Q4':[(7,4)],
  'Q5':[(4,5)],
  'Q6':[(3,6)],
  'Q7':[(8,7)],
  'Q8':[(1,8)]
}
queens=['Q1','Q2','Q3','Q4','Q5','Q6','Q7','Q8']

def cheack(row,column):

  for queen in queens:

    for xrow,xcolumn in graph[queen]:
      if(row==xrow and column==xcolumn):
        return 1;
  return 0;

def attacking_queen():
  cnt =0;
  for j in range(8):
    for row,column in graph[queens[j]]:

      new_row=row
      new_col=column
```

```
        # In a same row
        for i in range(8):


            new_col=new_col+1;
            if(new_row<=8):
                cnt=cnt+cheack(new_row,new_col)


        new_row=row
        new_col=column
        #Diagonally up
        for i in range(8):
            new_row=new_row+1;
            new_col=new_col+1;
            if new_row<=8 and new_col<=8:
                cnt=cnt+cheack(new_row,new_col)
        #Diagonally Down
        new_row=row
        new_col=column
        for i in range(8):
            new_row=new_row-1;
            new_col=new_col+1;
            if new_row>=1 and new_col>=1:
                cnt=cnt+cheack(new_row,new_col)
        #print(queens[j]," ",cnt);
    return cnt;

print("Attacking pairs: ",attacking_queen())
```

Prologue Code for H3:

```
:-dynamic(hval/1).

/* Evaluates a 8-queens' state given as list of 8 digits */

evalState(L,V):- assert(hval(0)),hl(1,L),
di_up(1,L),di_dn(1,L),hval(V),
                  retractall(hval(_)).

hl(8,_):-!.  hl(I,L):- nthel(I,L,X), chk_incr(I,L,X), I1 is I+1,
hl(I1,L).

chk_incr(8,_,_):-!.  chk_incr(I,L,X):- I1 is I+1, nthel(I1,L,Y),
                                       do_incr(X,Y),chk_incr(I1,L,X).
do_incr(X,Y):- X=Y, incr_hval.  do_incr(_,_).

incr_hval:-hval(V), V1 is V+1, retract(hval(_)), assert(hval(V1)).

di_up(8,_):-!.  di_up(I,L):- nthel(I,L,X), chkup_incr(I,L,X,0), I1 is
I+1,
                             di_up(I1,L).
chkup_incr(8,_,_,_):-!.
chkup_incr(I,L,X,K):- I1 is I+1, nthel(I1,L,Y), K1 is K+1,
                      doup_incr(X,Y,K1), chkup_incr(I1,L,X,K1).

doup_incr(X,Y,K1):- X1 is X+K1, Y=X1, incr_hval.  doup_incr(_,_,_).

di_dn(8,_):-!.  di_dn(I,L):- nthel(I,L,X), chkdn_incr(I,L,X,0), I1 is
I+1,
                             di_dn(I1,L).

chkdn_incr(8,_,_,_):-!.
chkdn_incr(I,L,X,K):- I1 is I+1, nthel(I1,L,Y), K1 is K+1,
                      dodn_incr(X,Y,K1), chkdn_incr(I1,L,X,K1).

dodn_incr(X,Y,K1):- X1 is X-K1, Y=X1, incr_hval.  dodn_incr(_,_,_).


% A procedure to find the nth element of a list

nthel(N,[_|T],El):- N1 is N-1, nthel(N1,T,El).
nthel(1,[H|_],H):-!.
```

Prologue Code for H2:

```
/*Goal configaretion*/

gtp(1,1,1).

gtp(2,1,2).

gtp(3,1,3).

gtp(4,2,3).

gtp(5,3,3).

gtp(6,3,2).

gtp(7,3,1).

gtp(8,2,1).

gblnk(2,2).

/*Target configaretion*/

tp(1,1,2).

tp(2,1,3).

tp(3,2,1).

tp(4,2,3).

tp(5,3,3).

tp(6,2,2).

tp(7,3,2).

tp(8,1,1).

blnk(3,1).

go:- calcH(1,[],L), sumList(L,V),write('Heuristics: '),write(V).

calcH(9,X,X):-!.

calcH(T,X,Y):- dist(T,D), append(X,[D],X1), T1 is T+1, calcH(T1,X1,Y).
```

```
dist(T,V):-tp(T,A,B), gtp(T,C,D), V is abs(A-C) + abs(B-D).

sumList([],0):-!.
sumList(L,V):-L=[H|T], sumList(T,V1), V is V1+H.
```