Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

CSE4108

Artificial Intelligence

Submitted By:

Mayisha Farzana 16.02.04.028

Date of Submission: **9 March, 2020**

Ques 1.

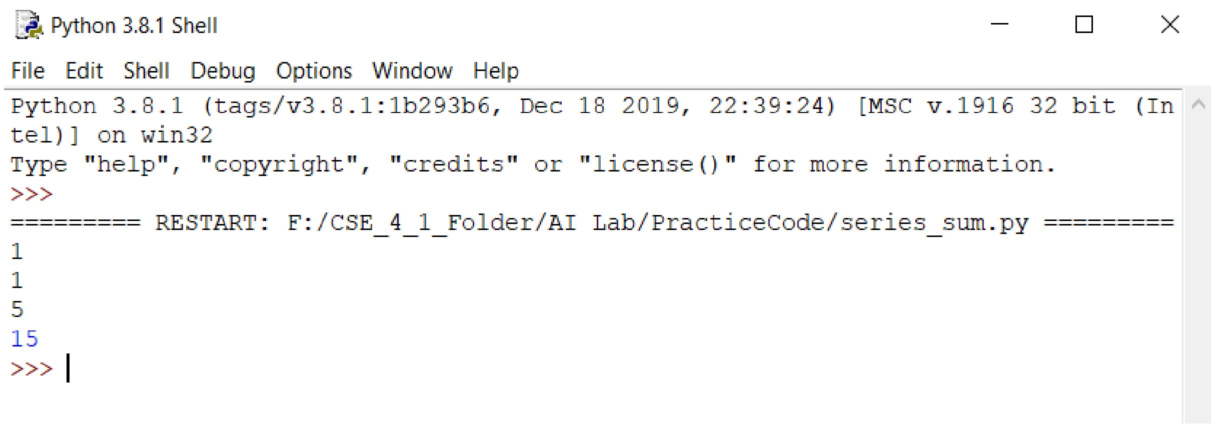
Define a recursive procedure in Python and in Prolog to find the sum of 1st n terms of an equal-interval series given the 1st term and the interval.

Description: In python code , first we to sum of the series . In formula , we know that nth term series is Term=first Term+(n-1)*interval . In here, we have used recursion function . So, when n=0 it will return 0 value . To get the total sum , we have use this formula , total = findSumOfSeries(n-1)+term.

Python Code:

```
 firstTerm=1
interval=1
def findSumOfSeries(n):
    if (n==0):
        return 0
    x=findSumOfSeries(n-1)
    term=firstTerm+((n-1)*interval)
    return x+term

firstTerm=int(input())
interval=int(input())
y=int(input())

print(findSumOfSeries(y))
```

Output :



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: F:/CSE_4_1_Folder/AI Lab/PracticeCode/series_sum.py =========
1
1
5
15
>>> |
```

Description:

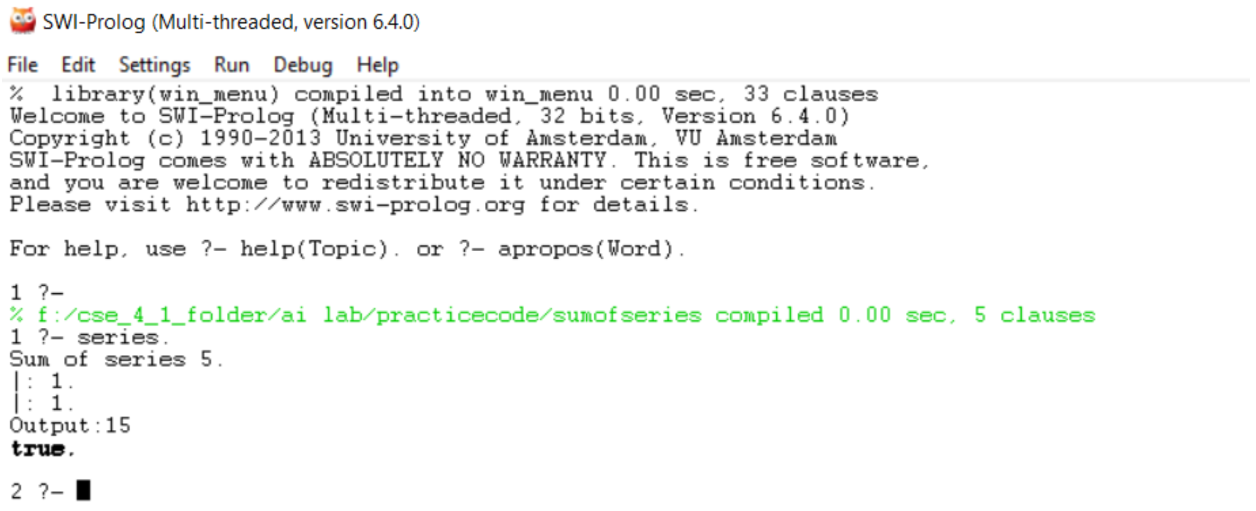
In prolog we know that , there is no return type , so we have to declare it as a parameter in the function . In the first sentence we have found that , if n is 0 it will return 0 value . In the next term

, to get the total sum of the series , we have count this till n>0 and we have assigned the last term N-1 in N . In findSumOfSeries , Y have saved the total sum till (N-1) . We have saved the last term in Term ; and X (The total sumOfSeries) will be Y+Term. We have taken input of SumOfSeries , firstTerm and interval .

Prolog Code:

```
findSumOfSeries(0,0,_,_):-!.
findSumOfSeries(N,X,F,I):- N>0 ,N1 is (N-1), findSumOfSeries(N1,Y,F,I),Term is F+(N-1)*I,
                        X is Y+Term .

series:-write('Sum of series '), read(S),read(F),read(I),write('Output:')
        ,findSumOfSeries(S,V,F,I),write(V),tab(5),fail.
series.
```

Output:


SWI-Prolog (Multi-threaded, version 6.4.0)
File Edit Settings Run Debug Help

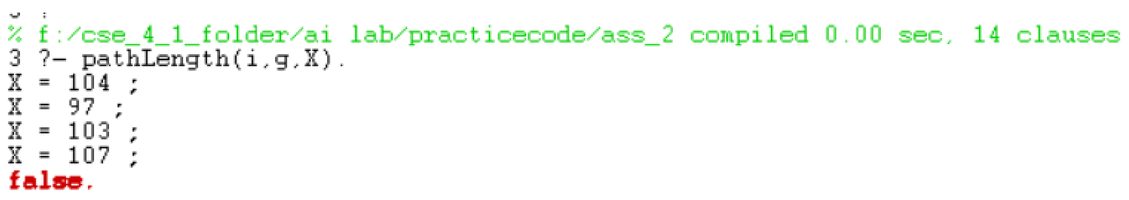
```
%  library(win_menu) compiled into win_menu 0.00 sec, 33 clauses
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.4.0)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
% f:/cse_4_1_folder/ai lab/practicecode/sumofseries compiled 0.00 sec, 5 clauses
1 ?- series.
Sum of series 5.
|: 1.
|: 1.
Output:15
true.

2 ?-
```

Ques 2: Define a recursive procedure in Python and in Prolog to find the length of a path between two vertices of a directed weighted graph.

Prolog Code :

In here , we have given the distance from one node to another node . then we have find out the path length . To find the length , we have used pathLength function . If X is the source and Y is destination ; then we find the total length in L . But if X travels to Z which is length L1 ; then Z travels to Y which is length L2; then the total length will be L=L1+L2.

```
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.
```



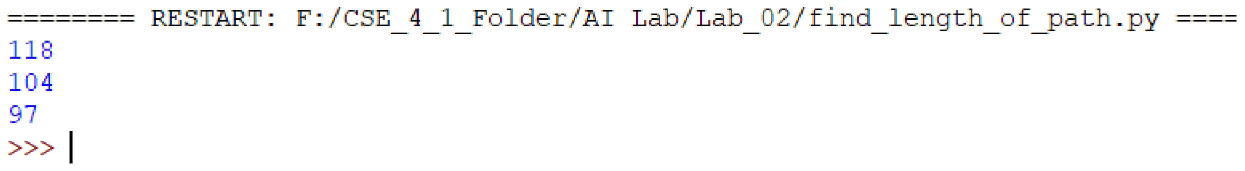
```
% f:/cse_4_1_folder/ai lab/practicecode/ass_2 compiled 0.00 sec, 14 clauses
3 ?- pathLength(i,g,X).
X = 104 ;
X = 97 ;
X = 103 ;
X = 107 ;
false.
```

Python Code :

In here , we have set the adjacent nodes and the corresponding weight . Our goal state is 7. At first , we have set all degrees in 0 value. Then we have seen the adjacent nodes and counts the degrees .If our current node is at goal state then we have printed the cost of it and terminate the program . But if it is the middle node ; then we have checked the degrees ; if it is more than 0 value . When the current node traverses the node ; we will decrease the degree values and will store the cost. We will continue this process through recursion , after that we will find out the total path length.

```
adj = [[(1, 35), (2, 45)],
      [(3, 22), (4, 32)],
      [(4, 28), (5, 36), (6, 27)],
      [(4, 31), (7, 47)],
      [(7, 30)],
      [(7, 26)],
      [],
      []]
goal = 7
start=0
degree = []
for i in range(len(adj)):
   degree.append(0)

for i in range(len(adj)):
   for k in adj[i]:
       degree[k[0]] += 1
def dfs(u, cost):
   if (u == goal):
       print(cost)
       return
   for node in adj[u]:
       if (degree[node[0]] > 0):
           degree[node[0]] -= 1
```

```
        dfs(node[0], cost + node[1])

dfs(start, 0)
```



```
======== RESTART: F:/CSE_4_1_Folder/AI Lab/Lab_02/find_length_of_path.py ====
118
104
97
>>> |
```

Ques 3. Modify the Python and Prolog codes demonstrated above to find h_2 and h_3 discussed above.

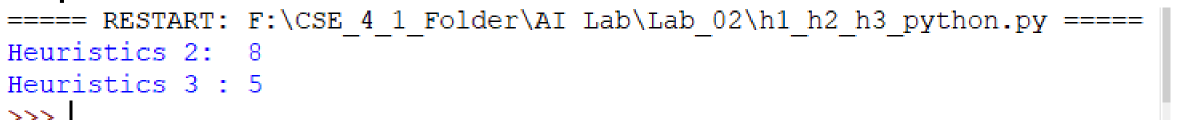
Python Code:

To find the h2 , first we have checked if the current gtp position matches with the selected gtp .If not then we have find out the distance abs(x1-x2)+abs(y1-y2) . Through this we have find out , h2 values.

To find out h3, first we have set position for queen . If there is a queen we have set it in 1 , and the others value are 0 .Then we have fix the row and check the columns if we find any queen ; and then we have fix the column and check the rows . After that we have checked it through diagonally if there is any queen . If we find any queen , we have counted the total numbers of queen .By this , we have found out the h3 value.

```
//H2
gtp=[(1,1,1), (2,1,2), (3,1,3), (4,2,3), (5,3,3), (6,3,2), (7,3,1), (8,2,1)]
gblnk = (2,2)
tp=[(1,1,2), (2,1,3), (3,2,1), (4,2,3), (5,3,3), (6,2,2), (7,3,2), (8,1,1)]
blnk = (3,1)
def H2():
    i,h=0,0
    while(i<=7):
        if ((gtp[i][1] != tp[i][1])|(gtp[i][2] != tp[i][2])):
            h += abs(gtp[i][1] - tp[i][1]) + abs(gtp[i][2] - tp[i][2])
        i=i+1
    print('Heuristics 2: ',h)
//H3
position = [[0, 0, 0, 0, 0, 0, 1, 0],
            [0, 0, 0, 1, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0],
            [0, 0, 0, 0, 0, 1, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0],
           [0, 1, 0, 0, 0, 0, 0, 1]]

def valid(x, y):
    if (x >= 0 and x < 8 and y >= 0 and y < 8):
        return True
    return False

def H3():
    ans = 0
    for i in range(8):
        for j in range(8):
            if (position[i][j] == 1):
                for k in range(j + 1, 8):
                    if (position[i][k] == 1):
                        ans = ans + 1
                for k in range(i + 1, 8):
                    if (position[k][j] == 1):
                        ans = ans + 1
                for d in range(1, 8):
                    x = i + d
                    y = j + d
                    if (valid(x, y) and position[x][y] == 1):
                        ans = ans + 1
                    x = i - d
                    y = j + d
                    if (valid(x, y) and position[x][y] == 1):
                        ans = ans + 1
   print('Heuristics 3 :',ans)
H2()
H3()
```

Output:



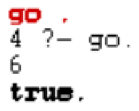
```
===== RESTART: F:\CSE_4_1_Folder\AI Lab\Lab_02\h1_h2_h3_python.py =====
Heuristics 2:  8
Heuristics 3 : 5
>>> 
```

Prolog Description :
In h2, we have checked if it returns 0 , that means the value is in the position. But if v is not in it’s position then we will return 1 value .In calH we have counted the total value .

```
//Prolog  H2
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).

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).

calcH(9,0):-!.
calcH(T,X):- T < 9, checkh(T,V), T1 is T+1, calcH(T1, X1), X is X1 + V.
```

```
checkh(T,V):-tp(T,A,B), gtp(T,C,D), A = C, B = D, V is 0, !. checkh(_,1):-!.
go:- calcH(1,V),write(V).
```

Output:



```
go .
4 ?- go.
6
true.
```