Expt No.

Page No
Date

Q. Facts, Fubby is a cat. Fubby has black spots.
Figaro is a dog. Figaro has white spots.

Rules: Mary owns a pet if it is a cat and it has black spots.
If someone owns something, he loves it.

Implement the rule base in SWI Prolog

Algorithm/
Flow Chart :—

(i) Identify fubby as cat
(ii) Identify figaro as dog
(iii) Then fubby has black spots
(iv) figaro has white spots.
(v) Identify mary owns a cat or a dog
(vi) Then check if it is a cat it has black spot
(vii) If mary owns that she loves that

Programme listing :—

```
cat (fubby),
dog (figaro),
spot (fubby, black),
```

SREEHARI

Teacher's Signature.....

Page No
Date

Expt No.

spot (figaro, white).

Owns (mary, Y) :-
cat(Y), spot(Y, Z), Z = black.

Loves (X, Y) :- Owns (X, Y).

Output :-

cat (X) → X = fubby

spot (fubby, X) → X → black
spot (figaro, X) → X → white.

Owns (mary, fubby) ⇒ True.

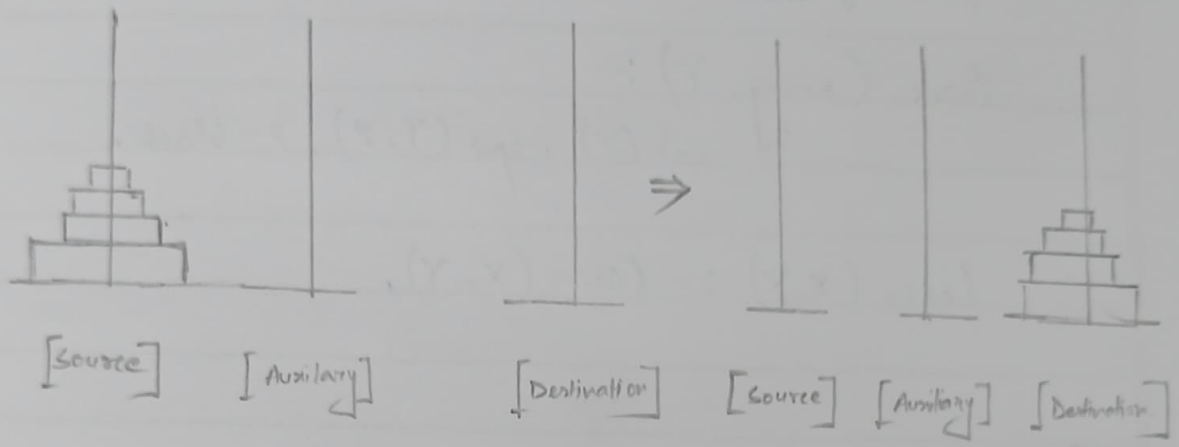
loves (mary, Y) ⇒ Y = fubby

EHARI

Teacher's Signature.....

[Tower of Hanoi]

Source to Destination Convention.



Output of Python Programme :—

for - n = 3

- Move disk 1 from pole A to pole C
- Move disk 2 from pole A to pole B
- Move disk 1 from pole C to pole B
- Move disk 3 from pole A to pole C
- Move disk 1 from pole B to pole A
- Move disk 2 from pole B to pole C
- Move disk 1 from pole A to pole C.

Expt No.

Page No	
Date	

Q. Implement the Tower of Hanoi using SWI prolog & python?

→ Algorithm :—

(i) Move n-1 disks from source to auxiliary

(ii) Move the N^{th} disk from source to destination

(iii) Move n-1 disks from auxiliary to destination.

START ⇒ Procedure TOH (disk, source, dest, aux)

IF (disk == 1), Then, move disk from source to destination.

ELSE, TOH (disk-1, source, ~~dest~~ aux, dest)

Move Disk (source to dest)

TOH (disk-1, aux, dest, source)

End IF, END Procedure

⇓

Stop

Program Listing :—

[Tower of Hanoi implementation in Python]

```
1. def toh (n, source, dest, aux)
    if (n==1): print ("Move disk 1 from tower", source, "to tower", dest)
    return
    toh (n-1, source, aux, dest)
    print ("Move disk", n, "from rod", source, "to rod", dest)
    toh (n-1, aux, dest, source)
n = 6/2/2/3/....
toh (n, 'A', 'B', 'C')
```

SREEHARI

Teacher's Signature.....

Expt No.

Page No
Date

[Tower of Hanoi implementation in Prolog]

```
move(1, X, Y, _) :-
    write('Move top disk from'), write(X), write(' to'), write(Y), nl.
move(N, X, Y, Z)
    N>1
    M is N-1
    move(M, X, Z, Y)
    move(1, X, Y, _)
    move(M, Z, X, Y)
```

Output :-

[PROLOG]

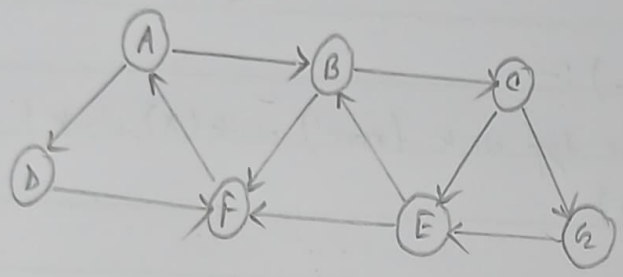
```
|?- move(4, source, target, Aux)
move top disk from source to Aux
move disk from source to target
move disk from aux to target.
move disk from source to aux
move disk from target to source
move disk from target to aux.
move disk from source to aux
move disk from source to target
move disk from aux to target
move disk from Aux to source
move disk from Aux target to source
move disk from aux to target
```

```
move disk from source to aux
move disk from source to target
move disk from aux to target
```

SREEHARI

Teacher's Signature.....

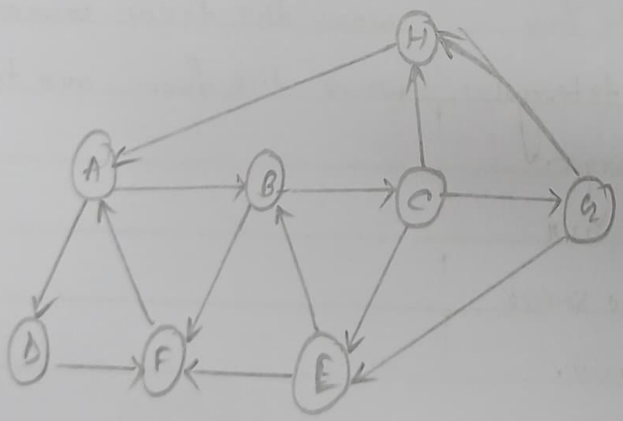
[BFS (Breath First Search)]



Adjacency Lists ⇒ A : B, D C : E, G G : B, F

B : C, F G : E F : A D : F

[DFS (Depth First Search)]



[Adjacency Lists]

A : B, D

B : C, F

C : E, G, H

E : B, F

G : E, H

F : A

D : F

H : A

Page No
Date

Q. Implement BFS & DFS using Python.

Algorithm :—

[BFS] →

Step 1 → Set status = 1 for each node in G.

Step 2 → Enqueue the starting node A and set its status = 2 (Waiting state)

Step 3 → Repeat steps 4 & 5 until Queue Empty

Step-4 → Dequeue a node N, Process it and set its status = 3

step-5 → Enqueue all the neighbours of N that are in the ready state (whose status = 1) and set their status = 2

[End of loop]

Step-6 → Exit

[DFS] → Step 1) set status = 1 (ready state) for each node in G

Step-2) Push the starting node A on the stack and set its status = 2 (waiting state)

step-3) Repeat steps 4 & 5 until Stack is empty.

step-4) Pop the top node N. process it and set its status = 3 (Processed step)

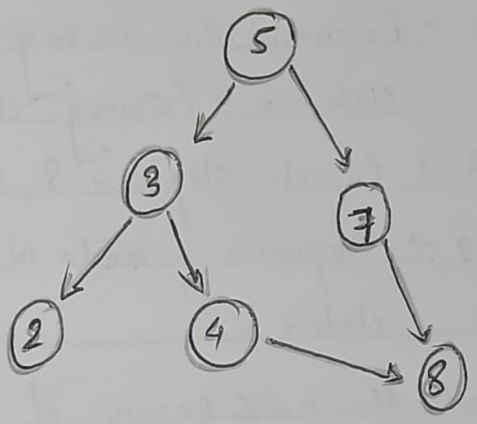
Step 5) Push on the stack all the neighbours of N that are in the ready state (whose status = 1) and set their status = 2

[End of loop]

Step-6 → Exit.

Teacher's Signature.....

[Given]



[Solve using BFS]

Expt No.

Page No
Date

Programme Listing :-

[BFS] →

```
graph = {
    '5' : ['3','7'],
    '3' : ['2','4'],
    '7' : ['8'],
    '2' : [],
    '4' : ['8'],
    '8' : [] }

visited = []
queue = []
def bfs (visited, graph, node):
    visited.append (node)
    queue.append (node)
    while queue :
        m = queue.pop(0)
        print (m, end = " ")
        for neighbour in graph[m]:
            if neighbour not in visited :
                visited.append (neighbour)
                queue.append (neighbour)
print ("Following is B.F.S")
bfs (visited, graph, '5')
```

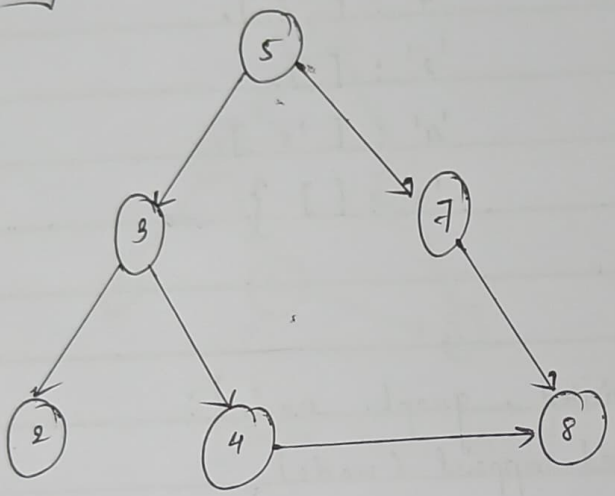
Output : Following is B.F.S

5 3 7 2 4 8

SREEHARI

Teacher's Signature.....

[Given]



Solve this with DFS ⇒

Expt No.

Page No
Date

[DFS] ⇒ graph = {
'5' : ['3', '7'],
'3' : ['2', '4'],
'7' : ['8'],
'2' : [],
'4' : ['8'],
'8' : [] }

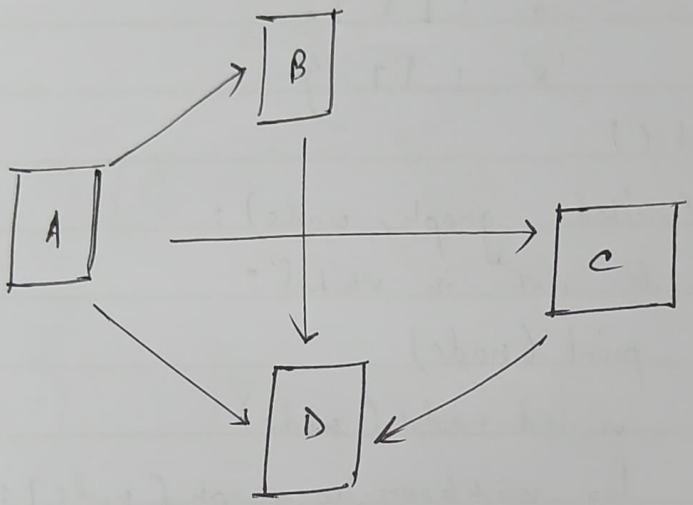
visited = set()
def dfs (visited, graph, node):
if node not in visited:
print (node)
visited.add (node)
for neighbour in graph [node]:
dfs (visited, graph, neighbour)
print ("Following is DFS");
dfs (visited, graph, '5');

Output :— Following is DFS

5 3 2 4 8 7

Teacher's Signature.....

[Given graph]



Solve using A* Algorithm.

Expt No.

Page No
Date

Q. Implementation of A* Algorithm.

→ [Algorithm]:=

<Step-i> Place the starting node in the OPEN list.

<Step-ii> Check if the open list is empty or not, if the list is empty then return failure and stop.

<Step-iii> Select the node from the "open" list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop. Otherwise.

<Step-iv> Expand Node n and generate all it's successors, and put n into the closed list for each successor n', check weather n' is already in the 'open' or 'closed', then if not they ~~attached to the black pointer which reflects the lowest~~, compute evaluation function for n' and place into open Open list.

<step-v> Else if node n' is already in 'Open' and 'closed', then it should be attached to the back pointer which reflects the lowest g(n') value.

<step-6> : Return to Step-2

[Programme Listing] →

```
from collections import deque
class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis
```

SREEHARI

Teacher's Signature.....

Expt No.

Page No

Date

```
def get_neighbours (self, v):
    return self.adjac_lis[v]
def h (self, n):
    H = {
        'A' : 1,
        'B' : 1,
        'C' : 1,
        'D' : 1 }
    return H[n]
def a_star_algorithm (self, start, stop):
    open_lst = set ([start])
    closed_lst = set ([])
    poo = {}
    poo[start] = start
    while len (open_lst) > 0:
        n = none
        for v in open_lst:
            if n == None or poo[v] + self.h (v) < poo[n]
                + self.h(n):
                n = v
    if n == None:
        Print ('Path doesnot exist')
        return n
    if n == stop
        reconst_path = []
        while par[n] != n:
```

SREEHARI

Teacher's Signature.....

Output :—

Input → (Before the code)

```
adjac_lis = {
        'A' : [('B',1),('C',3),('D',7)],
    'B' : [('D',5)],
    'C' : [('D',12)] }
graph1 = Graph(adjac_lis)
graph1.a-star_algorithm ('A','D')
```

Output →

```
path found : ['A','B','D']
```

reconst_path . append (n)
n = pare [n]
reconst reconst = path . append (start)
reconst_path . reverse()
print ('path found : {}'. format (reconst_path))
return reconst_path
for (m, weight) in self . get_neighbours (n):
if m not in open_lst and m not in closed_lst:
open_lst . add (m)
pare [m] = n
poo [m] = poo[n] + weight
else:
if poo[m] > poo[n] + weight:
poo [m] = poo[n] + weight
pare [m] = n
if m in closed_lst: ~~remove (m)~~
closed_lst . remove (m)
open_lst . add (m)
Open_lst . remove (n)
closed_lst . add (n)
Print ('Path doesnot exist')
return None

Expt No.

Page No
Date

reconst_path.append(n)
n = pare[n]
reconst reconst = path.append(start)
reconst_path.reverse()
print('path found: {}'.format(reconst_path))
return reconst_path
for (m, weight) in self.get_neighbours(n):
if m not in open_lst and m not in closed_lst:
open_lst.add(m)
pare[m] = n
poo[m] = poo[n] + weight
else:
if poo[m] > poo[n] + weight:
poo[m] = poo[n] + weight
pare[m] = n
if m in closed_lst: remove(m)
closed_lst.remove(m)
open_lst.add(m)
Open_lst.remove(n)
closed_lst.add(n)
Print('Path doesnot exist')
return None

Teacher's Signature.....

Page No
Date

Implementation of Greedy Algorithm using Python.

→ Best first Search. (Informed Search).

[Algorithm] :—

(i) Create 2 empty lists : OPEN & CLOSED

(ii) Start from the initial node (say N) and put it in the 'ordered' OPEN LIST.

(iii) Repeat the next steps until the GOAL node is reached

(a) If the openlist is empty, then exit the loop returning false.

(b) Select the first / top node (say N) in the OPEN openlist and move to the CLOSED list also capture the information of the parent-node.

(c) If N is a Goal Node, then move the node to the closed list and exit the loop returning True. The solution can be found by backtracking the path.

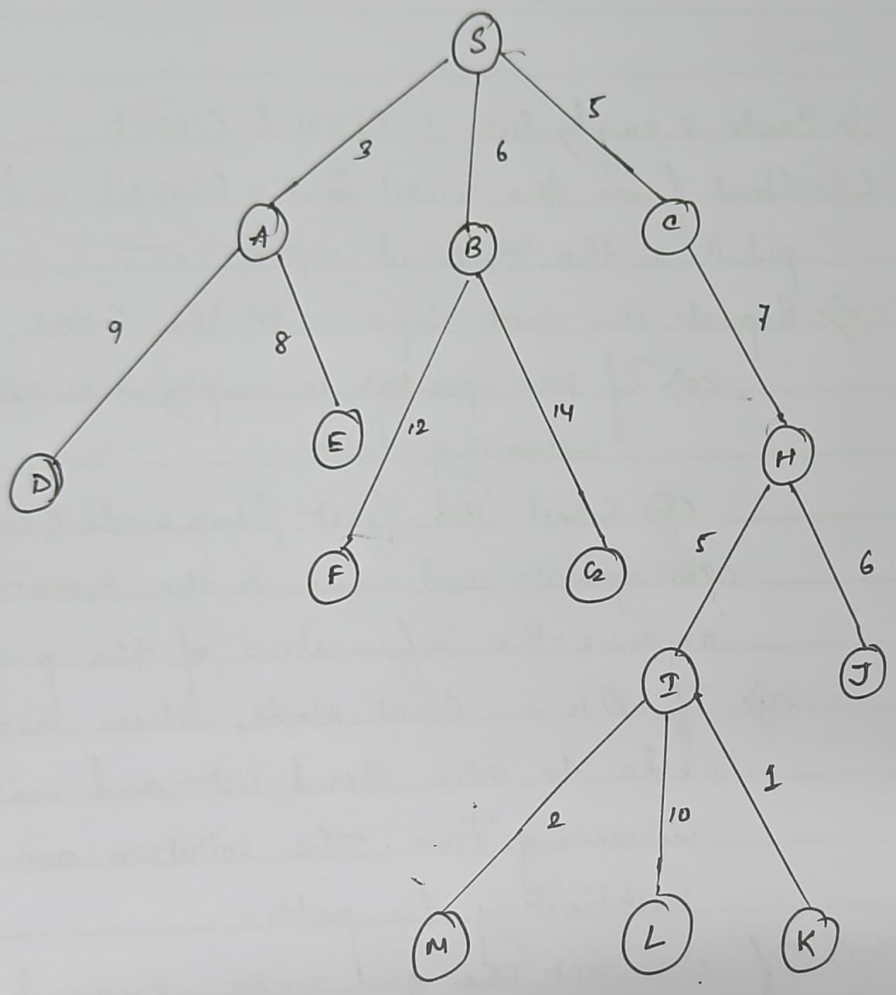
(d) If N is not the goal node, expand node N to generate the 'immediate' next node linked to the N and add all those to the OPEN list.

(e) Reorder the nodes to OPEN list in ascending order according to an evolution function.

The time complexity of this Algo is → $O(n * \log n)$

Teacher's Signature.....

Illustration :-



Page No
Date

[Programme Listing] :-

```
from queue import PriorityQueue
V = 14
graph = [[] for i in range(V)]

def best_first_search(actual_Src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_src] = True
    while pq.empty() == False:
    u = pq.get()[1]
    print(u, end = " ")
    if u == target:
        break
    for v, c in graph[u]:
        if visited[v] == False:
            visited[v] = True
            pq.put((c, v))

Print()
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))
addedge(0, 1, 3)
```

Teacher's Signature.....

Expt No.

Page No
Date

addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 5, 9)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)
source = 0
target = 9
best_first_search(source, target, v)

[Output]:=

dus ⇒ 0 1 3 2 8 9

SREEHARI

Teacher's Signature.....