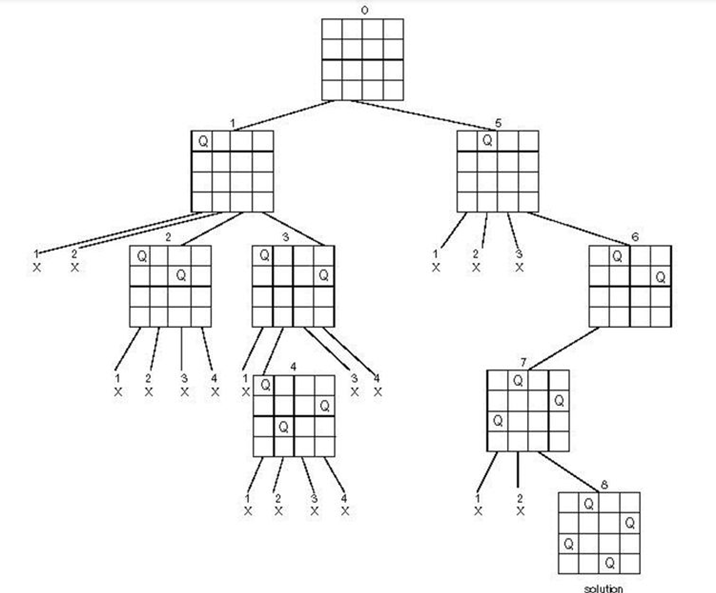
### Experiment – No-5

### *Problem Statement :* Write a program to implement 4 Queen Problems using BFS and DFS:



***Algorithm:***

1) Start in the leftmost column

2) If all queens are placed

return true

3) Try all rows in the current column.

Do following for every tried row.

a) If the queen can be placed safely in this row

then mark this [row, column] as part of the

solution and recursively check if placing

queen here leads to a solution.

b) If placing the queen in [row, column] leads to

a solution then return true.

c) If placing queen doesn't lead to a solution then

unmark this [row, column] (Backtrack) and go to

step (a) to try other rows.

3) If all rows have been tried and nothing worked,

return false to trigger backtracking.

***Program:***

graph={

'A':['B','D','E','C'],

'B':['F','G'],

'C':['H'],

'D':['I'],

'E':['J','K'],

'F':['L'],

'G':['M'],

'H':['N',],

'I':['O',],

'J':['P',],

'K':['Q',],

'L':[],

'M':['R'],

'N':['S'],

'O':['T'],

'P':['U'],

'Q':[],

'R':[],

'S':[],

'T':[],

'U':[]

}

Traversal=[]

visited=[]

queue=[]

def Bfs(graph,start,target,path):

queue.append(start)

while queue:

s=queue.pop(0)

path.append(s)

visited.append(s)

if s==target:

return path

for neighbour in graph[s]:

if neighbour not in visited:

queue.append(neighbour)

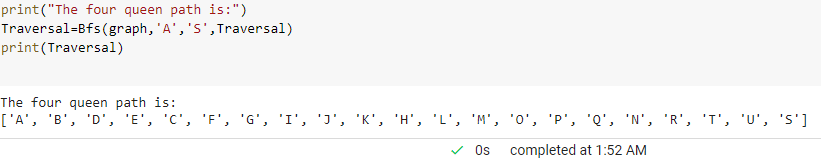
print("The four queen path is:")

Traversal=Bfs(graph,'A','S',Traversal)

print(Traversal)

### *Output :*

BY BFS:



BY DFS:

The four queen path is:

### ['A', 'C', 'H', 'N', 'S']