EX 3A Knight Tour & Count Path

DATE :

AIM :

To write a python program to find minimum steps to reach to specific cell in minimum moves by knight

Algorithm :

1.Create a list of 8 possible moves a knight can make (because a knight moves in "L" shapes).

2.Create a queue to keep track of where the knight can move next.

3.Start by putting the knight's starting position into the queue, with 0 steps taken so far.

4.Make a visited matrix to mark which positions the knight has already visited, so it doesn't repeat positions.

5.Start looping:Take the front position from the queue.

6.If it is the target position, return the number of steps taken so far — you're done!

7.If it's not the target:Try all 8 possible knight moves from the current position.

8.For each move:Check if the new position is inside the board boundaries.

9.If it's valid and not visited:Mark it as visited.

10.Add it to the queue with the step count increased by 1.

11.Keep repeating this process until the target is reached.

Program :

Developed by: GOWTHAM N

Register Number: 212223100008

```
class cell:

    def __init__(self, x = 0, y = 0, dist = 0):
        self.x = x
        self.y = y
        self.dist = dist

def isInside(x, y, N):
    if (x >= 1 and x <= N and
        y >= 1 and y <= N):
        return True
    return False
def minStepToReachTarget(knightpos,targetpos, N):
     # add your code here
    dx = [2, 2, -2, -2, 1, 1, -1, -1]
    dy = [1, -1, 1, -1, 2, -2, 2, -2]

    queue = []
    queue.append(cell(knightpos[0], knightpos[1], 0))
    visited = [[False for i in range(N + 1)]
                      for j in range(N + 1)]
    visited[knightpos[0]][knightpos[1]] = True
    while(len(queue) > 0):

        t = queue[0]
        queue.pop(0)
        if(t.x == targetpos[0] and
           t.y == targetpos[1]):
            return t.dist
```

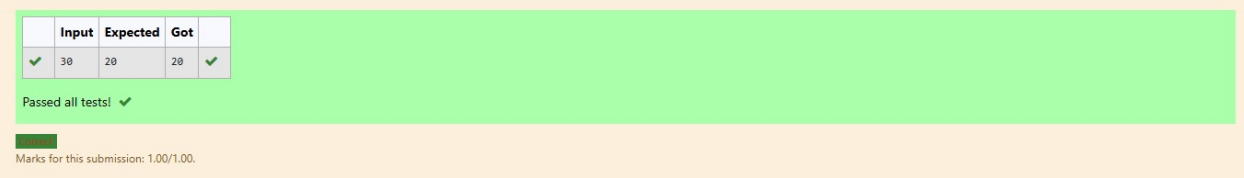
```
        # iterate for all reachable states
        for i in range(8):

            x = t.x + dx[i]
            y = t.y + dy[i]

            if(isInside(x, y, N) and not visited[x][y]):
                visited[x][y] = True
                queue.append(cell(x, y, t.dist + 1))


if __name__=='__main__':
    N = 30
    knightpos = [1, 1]
    targetpos = [30, 30]
    print(minStepToReachTarget(knightpos,
                               targetpos, N))
```

Output :



Result :

The program executed successfully, and the minimum number of steps for the knight to reach the target was calculated.

EX 3B Hamiltonian Circuit Problem

DATE :

AIM :

To write a python program to check whether Hamiltonian path exits in the given graph.

Algorithm :

1.Start with a path list, which keeps track of the order of visited vertices.

2.Initialize the first position in the path to vertex 0.

3.Use a recursive helper function (hamUtil) to build the path one vertex at a time.

4.Try every possible vertex as the next move.

5.For each vertex v, check if it's a valid move.

6.It must be connected to the previous vertex (adj[path[pos-1]][v] == 1).

7.It must not have been visited before (v not in path).

8.If a vertex is valid:Place it in the path.

9.Move to the next position (pos+1) and continue recursively.

10.If you reach a position equal to the number of vertices (pos == N):

11.You've found a valid Hamiltonian Path — return True.

12.If no valid path is found from a position, backtrack:

13.Remove the vertex from the path (path[pos] = -1) and try the next one.

14.If the recursion completes and no path is found, return False.

Program :

Developed by: GOWTHAM N

Register Number: 212223100008

```
def is_valid(v,pos,path,adj,N):

    if adj[path[pos-1]][v]==0:
        return False
    if v in path:
        return False
    return True

def hamUtil(adj,path,pos,N):
    if pos==N:
        return True
    for v in range(N):
        if is_valid(v,pos,path,adj,N):
            path[pos]=v
            if hamUtil(adj,path,pos+1,N):
                return True
            path[pos]=-1

def Hamiltonian_path(adj, N):
    path=[-1]*N
    path[0]=0

    if hamUtil(adj,path,1,N) == False:
        print ("Solution does not exist\n")
        return False
```

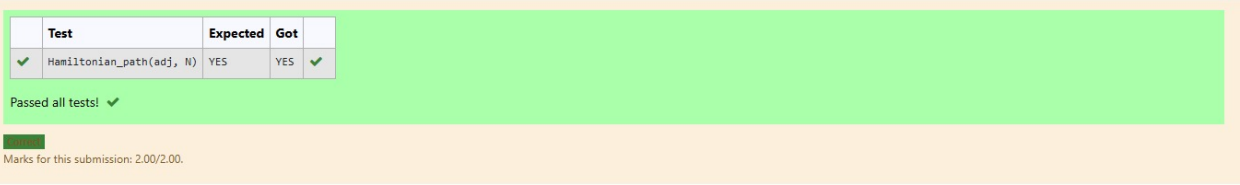
```
    # printSolution(path)
    return True

adj = [ [ 0, 1, 1, 1, 0 ] ,
        [ 1, 0, 1, 0, 1 ],
        [ 1, 1, 0, 1, 1 ],
        [ 1, 0, 1, 0, 0 ] ]

N = len(adj)

if (Hamiltonian_path(adj, N)):
    print("YES")
else:
    print("NO")
```

Output :



Result :

The Hamiltonian path program executed successfully, and it determined whether a Hamiltonian path exists in the given graph.

EX 3C Sudoku Solver

DATE :

AIM :

To write a python program to find the solution of sudoku puzzle using Backtracking.

Algorithm :

1.Find an empty cell (a cell with value 0).

2.Use isEmpty() to scan the board and return the position of the first empty cell.

3.If no empty cell is found, the board is complete!

4.Call printBoard() to display the solved Sudoku.

5.If there is an empty cell:Try placing each number from 1 to 9 in that cell.

6.For each number:Check if placing it is allowed using isPossible():

7.The number should not already be in the same row.

8.It should not be in the same column.

9.It should not be in the same 3x3 subgrid.

10.If the number is allowed:Place it in the cell.

11.Recursively call solve() to fill the next empty cell.

12.If solve() returns True, you're done.

13.If solve() returns False, it means that guess led to a dead end.

14.So, reset the cell back to 0 and try the next number (this is backtracking).

15.Repeat this process until the board is fully and correctly filled.

Program :

Developed by: GOWTHAM N

Register Number: 212223100008

```
board = [
    [0, 0, 0, 8, 0, 0, 4, 0, 3],
    [2, 0, 0, 0, 0, 4, 8, 9, 0],
    [0, 9, 0, 0, 0, 0, 0, 0, 2],
    [0, 0, 0, 0, 2, 9, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 7, 0, 6, 5, 0, 0, 0, 0],
    [9, 0, 0, 0, 0, 0, 0, 8, 0],
    [0, 6, 2, 7, 0, 0, 0, 0, 1],
    [4, 0, 3, 0, 0, 6, 0, 0, 0]
]

def printBoard(board):
    for i in range(0, 9):
        for j in range(0, 9):
            print(board[i][j], end=" ")
        print()

def isPossible(row, col, val):
    for j in range(0, 9):
        if board[row][j] == val:
            return False

    for i in range(0, 9):
        if board[i][col] == val:
            return False
```

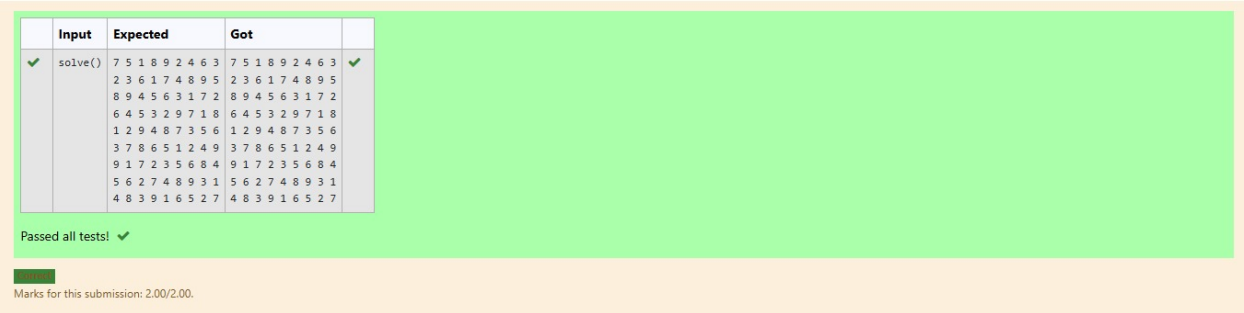
```
    startRow = (row // 3) * 3
    startCol = (col // 3) * 3
    for i in range(0, 3):
        for j in range(0, 3):
            if board[startRow+i][startCol+j] == val:
                return False
    return True
def isEmpty():
    for r in range(9):
        for c in range(9):
            if board[r][c] == 0:
                return r, c
    return None

def solve():

    empty = isEmpty()
    if empty is None:
        printBoard(board)
        return True

    row, col = empty
    for guess in range(1, 10):
        if isPossible(row, col, guess):
            board[row][col] = guess
            if solve():
                return True
            board[row][col] = 0
    return False
solve()
```

Output :



Result :

The Sudoku solver program executed successfully and found the solution for the given puzzle.

EX 3D Pattern Matching

DATE :

AIM :

To write a python program to implement pattern matching on the given string using Brute Force algorithm.

Algorithm :

1.Let m be the length of s1 (the big string).

2.Let n be the length of s2 (the pattern we want to find).

3.Loop through all possible starting positions in s1 where s2 could fit:From index 0 to m - n.

4.At each position i:Compare the substring of s1 starting at i with s2, character by character.

5.If all characters match (s1[i+j] == s2[j] for all j), then:

6.Return i as the starting index of the match.

7.If no match is found after checking all positions, return -1.

Program :

Developed by: GOWTHAM N

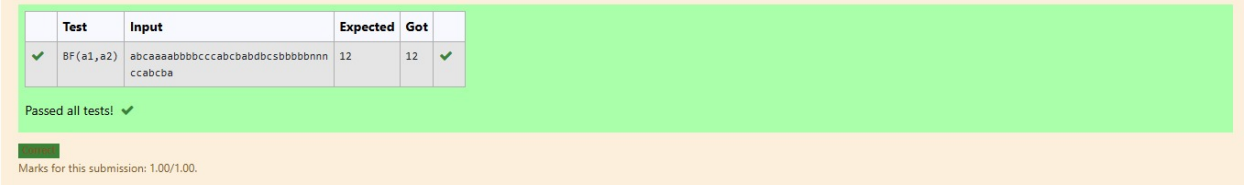
Register Number: 212223100008

```
def BF(s1,s2):

    m=len(s1)
    n=len(s2)
    for i in range(m-n+1):
        j=0
        while j<n and s1[i+j]==s2[j]:
            j+=1
        if j==n:
            return i
    return -1

if __name__ == "__main__":
    a1=input()
    a2=input()
    b=BF(a1,a2)
    print(b)
```

Output:



Result :

The brute force substring search program executed successfully and returned the starting index of the match or 0 if no match was found.

Started on	Monday, 21 July 2025, 1:52 PM
State	Finished
Completed on	Monday, 21 July 2025, 6:24 PM
Time taken	4 hours 32 mins
Overdue	2 hours 32 mins
Grade	**100.00** out of 100.00

Question **1**

Correct

Mark 20.00 out of 20.00

Write a python program to find minimum steps to reach to specific cell in minimum moves by knight.

Answer: (penalty regime: 0 %)

Reset answer

```
from collections import deque
class cell:
 def __init__(self, x=0, y=0, dist=0):
  self.x = x
  self.y = y
  self.dist = dist
def isInside(x, y, N):
 return x>=1 and x<=N and y>=1 and y<=N
def minStepToReachTarget(knightpos, targetpos, N):
 dx=[2,1,-1,-2,-2,-1,1,2]
 dy=[1,2,2,1,-1,-2,-2,-1]
 visited=[[False for i in range(N+1)] for j in range(N+1)]
 q=deque()
 q.append(cell(knightpos[0],knightpos[1],0))
 visited[knightpos[0]][knightpos[1]]=True
 while q:
  t=q.popleft()
  if t.x==targetpos[0] and t.y==targetpos[1]:
   return t.dist
  for i in range(8):
   x=t.x+dx[i]
   y=t.y+dy[i]
```

	Input	Expected	Got	
✔	30	20	20	✔

Passed all tests! ✔

Correct

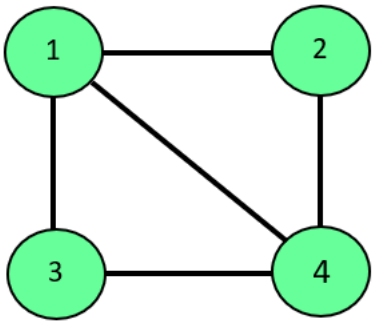
Marks for this submission: 20.00/20.00.

Question **2**

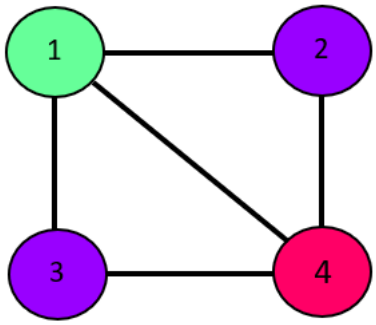
Correct

Mark 20.00 out of 20.00

The m-coloring problem states, **"We are given an undirected graph and m number of different colors. We have to check if we can assign colors to the vertices of the graphs in such a way that no two adjacent vertices have the same color."**



0	1	1	1
1	0	0	1
1	0	0	1
1	1	1	0



Node 1 -> color 1
Node 2 -> color 2
Node 3 -> color 2
Node 4-> color 3

For example:

Result
Solution Exists: Following are the assigned colors Vertex 1 is given color: 1 Vertex 2 is given color: 2 Vertex 3 is given color: 3 Vertex 4 is given color: 2

Answer: (penalty regime: 0 %)

Reset answer

```
def isSafe(graph, color):
    for i in range(4):
        for j in range(i + 1, 4):
            if (graph[i][j] and color[j] == color[i]):
                return False
    return True
def graphColoring(graph, m, i, color):
    if i == 4:
        if isSafe(graph, color):
            display(color)
            return True
        return False
    for j in range(1, m + 1):
        color[i] = j
        if graphColoring(graph, m, i + 1, color):
            return True
        color[i] = 0
    return False
def display(color):
    print("Solution Exists: Following are the assigned colors ")
    for i in range(4):
        print("Vertex", i + 1, " is given color: ", color[i])
```

	Expected	Got	
✔	Solution Exists: Following are the assigned colors Vertex 1 is given color: 1 Vertex 2 is given color: 2 Vertex 3 is given color: 3 Vertex 4 is given color: 2	Solution Exists: Following are the assigned colors Vertex 1 is given color: 1 Vertex 2 is given color: 2 Vertex 3 is given color: 3 Vertex 4 is given color: 2	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **3**

Correct

Mark 20.00 out of 20.00

Write a python program to implement Boyer Moore Algorithm with Good Suffix heuristic to find pattern in given text string.

For example:

Input	Result
ABAAABAACD ABA	pattern occurs at shift = 0 pattern occurs at shift = 4

Answer: (penalty regime: 0 %)

Reset answer

```
def preprocess_strong_suffix(shift,bpos,pat,m):
 i=m
 j=m+1
 bpos[i]=j
 while i>0:
  while j<=m and pat[i-1]!=pat[j-1]:
   if shift[j]==0:
    shift[j]=j-i
   j=bpos[j]
  i-=1
  j-=1
  bpos[i]=j
def preprocess_case2(shift,bpos,pat,m):
 j=bpos[0]
 for i in range(m+1):
  if shift[i]==0:
   shift[i]=j
  if i==j:
   j=bpos[j]
def search(text,pat):
 s=0
 m=len(pat)
```

	Input	Expected	Got	
✔	ABAAABAACD ABA	pattern occurs at shift = 0 pattern occurs at shift = 4	pattern occurs at shift = 0 pattern occurs at shift = 4	✔
✔	SaveethaEngineering Saveetha veetha	pattern occurs at shift = 2 pattern occurs at shift = 22	pattern occurs at shift = 2 pattern occurs at shift = 22	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **4**

Correct

Mark 20.00 out of 20.00

Write a Python program for Bad Character Heuristic of Boyer Moore String Matching Algorithm

For example:

Input	Result
ABAAAABCD ABC	Pattern occur at shift = 5

Answer: (penalty regime: 0 %)

Reset answer

```
NO_OF_CHARS=256
def badCharHeuristic(string,size):
 badChar=[-1]*NO_OF_CHARS
 for i in range(size):
  badChar[ord(string[i])]=i
 return badChar
def search(txt,pat):
 m=len(pat)
 n=len(txt)
 badChar=badCharHeuristic(pat,m)
 s=0
 while s<=n-m:
  j=m-1
  while j>=0 and pat[j]==txt[s+j]:
   j-=1
  if j<0:
   print("Pattern occur at shift = {}".format(s))
   s+=(m-badChar[ord(txt[s+m])] if s+m<n else 1)
  else:
   s+=max(1,j-badChar[ord(txt[s+j])])
def main():
 txt=input()
```

	Input	Expected	Got	
✔	ABAAAABCD ABC	Pattern occur at shift = 5	Pattern occur at shift = 5	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

Create a python program to implement Hamiltonian circuit problem using Backtracking.

For example:

Result
Solution Exists: Following is one Hamiltonian Cycle 0 1 2 4 3 0

Answer: (penalty regime: 0 %)

Reset answer

```
class Graph():
 def __init__(self,vertices):
  self.graph=[[0 for column in range(vertices)] for row in range(vertices)]
  self.V=vertices
 def isSafe(self,v,pos,path):
  if self.graph[path[pos-1]][v]==0:
   return False
  for vertex in path:
   if vertex==v:
    return False
  return True
 def hamCycleUtil(self,path,pos):
  if pos==self.V:
   if self.graph[path[pos-1]][path[0]]==1:
    return True
   else:
    return False
  for v in range(1,self.V):
   if self.isSafe(v,pos,path):
    path[pos]=v
    if self.hamCycleUtil(path,pos+1)==True:
     return True
```

	Expected	Got	
✔	Solution Exists: Following is one Hamiltonian Cycle 0 1 2 4 3 0	Solution Exists: Following is one Hamiltonian Cycle 0 1 2 4 3 0	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.