Name **:** Laukik Nitin Marathe
Roll No **:** TEAD21153
Sub **:** SL-1 Laboratory Group A
Class **:** TE (A)
Branch **:** AI&DS
Assignment No **:** 08

Title :
Implement depth first search algorithm and Breadth First Search algorithm. Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.

Code:
```
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

    # Insert Node
    def insert(self, data):
        if self.data:
            if data < self.data:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
        else:
            self.data = data
    # Print the Tree
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.data, end=" ")
        if self.right:
            self.right.PrintTree()

    # Inorder traversal
    def inorderTraversal(self):
        res = []
        if self.left:
            res = self.left.inorderTraversal()
        res.append(self.data)
        if self.right:
            res = res + self.right.inorderTraversal()
```

```python
        return res

    # Preorder traversal (Root -> Left -> Right)
    def PreorderTraversal(self):
        res = []
        res.append(self.data)
        if self.left:
            res = res + self.left.PreorderTraversal()
        if self.right:
            res = res + self.right.PreorderTraversal()
        return res

    # Postorder traversal
    def PostorderTraversal(self):
        res = []
        if self.left:
            res = self.left.PostorderTraversal()
        if self.right:
            res = res + self.right.PostorderTraversal()
        res.append(self.data)
        return res

# Function to print level order traversal of tree
def printLevelOrder(root):
    h = height(root)
    for i in range(1, h + 1):
        printCurrentLevel(root, i)

# Print nodes at a current level
def printCurrentLevel(root, level):
    if root is None:
        return
    if level == 1:
        print(root.data, end=" ")
    elif level > 1:
        printCurrentLevel(root.left, level - 1)
        printCurrentLevel(root.right, level - 1)

def height(node):
    if node is None:
        return 0
    else:
        lheight = height(node.left)
        rheight = height(node.right)
        if lheight > rheight:
            return lheight + 1
        else:
            return rheight + 1

# Input for tree
```

```
n = int(input("Enter the number of nodes in the tree: "))
flag = False

for i in range(n):
    if flag == False:
        r = int(input("Enter the value of root: "))
        root = Node(r)
        flag = True
    else:
        r = int(input("Enter the value of node: "))
        root.insert(r)

getInput = int(input("Enter the number of operation to perform (1.BFS, 2.DFS): "))

if getInput == 1:
    print("Level Order Traversal:")
    printLevelOrder(root)
elif getInput == 2:
    order = int(input("Enter the order of DFS (1.Inorder, 2.Pre-Order, 3.Post-order): "))
    if order == 1:
        print("Inorder Traversal:")
        print(root.inorderTraversal())
    elif order == 2:
        print("Preorder Traversal:")
        print(root.PreorderTraversal())
    elif order == 3:
        print("Postorder Traversal:")
        print(root.PostorderTraversal())
```

Output:

```
Enter the number of nodes in the tree: 6
Enter the value of root: 48
Enter the value of node: 12
Enter the value of node: 79
Enter the value of node: 36
Enter the value of node: 88
Enter the value of node: 25
Enter the number of operation to perform (1.BFS, 2.DFS): 2
Enter the order of DFS (1.Inorder, 2.Pre-Order, 3.Post-order): 2
Preorder Traversal:
[48, 12, 36, 25, 79, 88]
21153. Laukik Nitin Marathe


...Program finished with exit code 0
Press ENTER to exit console.
```

Name **:** Laukik Nitin Marathe
Roll No **:** TEAD21153
Sub **:** SL-1 Laboratory Group A
Class **:** TE (A)
Branch **:** AI&DS
Assignment No **:** 09

Title :

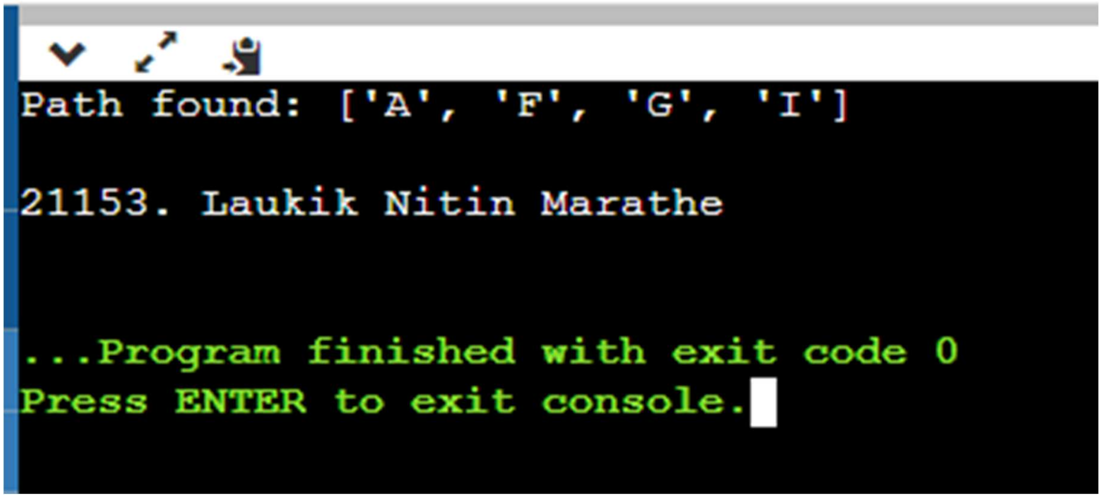
Implement A star (A*) Algorithm for any game search problem.

Code:

```
def aStarAlgo(start_node, stop_node):
    open_set = set([start_node])
    closed_set = set()
    g = {}
    parents = {}

    g[start_node] = 0
    parents[start_node] = start_node

    while len(open_set) > 0:
        n = None
        for v in open_set:
            if n is None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v

        if n is None:
            print('Path does not exist!')
            return None
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path

        open_set.remove(n)
        closed_set.add(n)

        for (m, weight) in get_neighbors(n):
            if m not in open_set and m not in closed_set:
                open_set.add(m)
                parents[m] = n
                g[m] = g[n] + weight
            else:
                if g[m] > g[n] + weight:
                    g[m] = g[n] + weight
```

```
                parents[m] = n

    print('Path does not exist!')
    return None

def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]

Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
    'F': [('A', 3), ('G', 1), ('H', 7)],
    'G': [('F', 1), ('I', 3)],
    'H': [('F', 7), ('I', 2)],
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
}

aStarAlgo('A', 'I')
```

Output:


Path found: ['A', 'F', 'G', 'I']
21153. Laukik Nitin Marathe
...Program finished with exit code 0
Press ENTER to exit console.

Name **:** Laukik Nitin Marathe
Roll No **:** TEAD21153
Sub **:** SL-1 Laboratory Group A
Class **:** TE (A)
Branch **:** AI&DS
Assignment No **:** 11

Title:
Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph colouring problem**.**

Code:
```
def reset():
    global board, counter
    board = [[0] * n for i in range(n)]
    counter = 0

def display(board):
    global counter
    counter += 1
    for i in range(0, n):
        for j in range(0, n):
            print(board[i][j], end=" ")
        print()
    print()

def check(board, row, column):
    for i in range(0, column):
        if board[row][i] == 1:
            return False

    for i, j in zip(range(row, -1, -1), range(column, -1, -1)):
        if board[i][j] == 1:
            return False

    for i, j in zip(range(row, n), range(column, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def marker(board, column):
    possibility = False
    if column == n:
        display(board)
        return True

    for i in range(0, n):
        if check(board, i, column):
            board[i][column] = 1
```

```
            possibility = marker(board, column + 1)
            board[i][column] = 0

    return possibility

n = int(input("Enter the size of the board: "))
counter = 0
reset()

if not marker(board, 0) and counter == 0:
    print("No feasible solution exists for the given dimensions")
else:
    print("There are a total of " + str(counter) + " possibilities for the given dimensions!")
```

Output:

input

```
Enter the size of the board: 6
0 0 0 1 0 0
1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0

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

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

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

There are a total of 4 possibilities for the given dimensions!

21153. Laukik Nitin Marathe


...Program finished with exit code 0
Press ENTER to exit console.
```

Name **:** Laukik Nitin Marathe
Roll No **:** TEAD21153
Sub **:** SL-1 Laboratory Group A
Class **:** TE (A)
Branch **:** AI&DS
Assignment No **:** 12

Title:

Implement Greedy search algorithm for any of the following application:

- Selection Sort
- Minimum Spanning Tree
- Single-Source Shortest Path Problem
- Job Scheduling Problem
- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm
- Dijkstra's Minimal Spanning Tree Algorithm

Code:

```
def findMin(V):
    # All denominations of Indian Currency
    deno = [1, 2, 5, 10, 20, 50, 100, 500, 1000]
    n = len(deno)
    # Initialize Result
    ans = []
    # Traverse through all denominations
    i = n - 1
    while(i >= 0):
        # Find denominations
        while (V >= deno[i]):
            V -= deno[i]
            ans.append(deno[i])
        i -= 1
    # Print result
    for i in range(len(ans)):
        print(ans[i], end=" ")

# Driver Code
if __name__ == '__main__':
    n = int(input("Enter the amount: "))
    print("Following is the minimal number of change for", n, ": ", end="")
    findMin(n)
```

Output:

```
input
Enter the amount: 1257
Following is the minimal number of change for 1257 : 1000 100 100 50 5 2
21153. Laukik Nitin Marathe


...Program finished with exit code 0
Press ENTER to exit console.
```