**COMSATS UNVERISTY ISLAMABAD**



**Artificial Intelligence**

**Lab 5**

**Submitted by:**

Hasaan Ahmad SP22-BSE-017

**Submitted to:**

**Sir Waqas Ali**

**Activity 1:**

def iterative\_deepening\_dfs(start,target):

    depth = 1

    bottom\_reached = False

    while not bottom\_reached:

        result, bottom\_reached= iterative\_deepening\_dfs\_rec(start, target,0,depth)

        if result is not None:

            return result

        depth \*=2

        print("Increasing Depth to " + str(depth))

    return None

def iterative\_deepening\_dfs\_rec(node,target, current\_depth,max\_depth):

    print("Visiting Node" + str(node["value"]))

    if node["value"]== target:

        print("found")

        return node, True

    if current\_depth== max\_depth:

        print("Current Maximum Depth Reached, Returning")

        if len(node["children"])>0:

            return None, False

        else:

            return None, True

    bottom\_reached = True

    for i in range(len(node["children"])):

        result,bottom\_reached\_rec = iterative\_deepening\_dfs\_rec(node["children"][i], target,current\_depth + 1, max\_depth)

        if result is not None:

            return result, True

        bottom\_reached= bottom\_reached and bottom\_reached\_rec

    return None, bottom\_reached

start = {

    "value": 0,

    "children": [

        {

            "value": 1,

            "children": [

                {"value": 3, "children": []},

                {"value": 4, "children": []}

            ]

        },

        {

            "value": 2,

            "children": [

                {"value": 5, "children": []},

                {"value": 6, "children": []}

            ]

        }

    ]

}

data = {

    "value": 0,

    "children": [

        {

            "value": 1,

            "children": [

                {"value": 3, "children": [

                    {"value": 4, "children": []},

                    {"value": 5, "children": []}

                ]},

                {"value": 6, "children": [

                    {"value": 7, "children": []}

                ]},

                {"value": 8, "children": [

                    {"value": 9, "children": []}

                ]},

                {"value": 10, "children": [

                    {"value": 11, "children": []}

                ]},

                {"value": 12, "children": [

                    {"value": 13, "children": []}

                ]},

                {"value": 14, "children": [

                    {"value": 15, "children": []}

                ]},

                {"value": 16, "children": []},

                {"value": 17, "children": [

                    {"value": 18, "children": []}

                ]},

                {"value": 19, "children": [

                    {"value": 20, "children": []},

                    {"value": 21, "children": []}

                ]},

                {"value": 22, "children": [

                    {"value": 23, "children": []},

                    {"value": 24, "children": []}

                ]},

                {"value": 25, "children": [

                    {"value": 5, "children": []}

                ]}

            ]

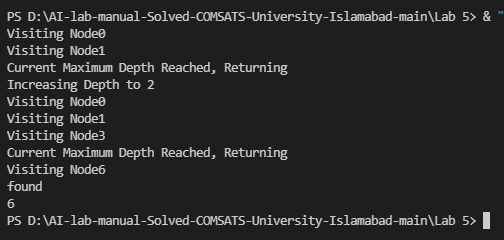
        }

    ]

}

print(iterative\_deepening\_dfs(data,6)["value"])

**Output:**

****

**Graded Task 1:**

# iterative deepening search

#

#SP22-BSE-017 HASAAN AHMAD

#

def iterative\_deepening\_dfs(dic\_graph, target):

    depth = 1

    bottom\_reached = False

    while not bottom\_reached:

        result, bottom\_reached = iterative\_deepening\_dfs\_rec(dic\_graph, target, 0, depth)

        if result is not None:

            return result

        depth \*= 2

        print("Increasing depth to "+ str(depth))

    return None

def iterative\_deepening\_dfs\_rec(node, target, current\_depth, max\_depth):

    print("Visiting Node " + str(node["value"]))

    if node["value"] == target:

        print("Found the node !")

        return node, True

    if current\_depth == max\_depth:

        print("Current maximum depth reached, returning...")

        if len(node["children"]) > 0:

            return None, False

        else:

            return None, True

    bottom\_reached = True

    for i in range(len(node["children"])):

        result, bottom\_reached\_rec = iterative\_deepening\_dfs\_rec(node["children"][i], target, current\_depth + 1, max\_depth)

        if result is not None:

            return result, True

        bottom\_reached = bottom\_reached and bottom\_reached\_rec

    return None, bottom\_reached

dictionary = {

    "value": "Arad",

    "children": [

        {"value": "Sibiu",

         "children": [

             {"value": "Fagaras", "children": []},

             {"value": "Rimnicu", "children": []}

         ]},

        {"value": "Zerind",

         "children": [

             {"value": "Oradea", "children": []}

         ]},

        {"value": "Timisoara",

         "children": [

             {"value": "Lugoj", "children": []}

         ]},

        {"value": "Oradea", "children": []},

        {"value": "Rimnicu",

         "children": [

             {"value": "Pitesti", "children": []},

             {"value": "Craiova", "children": []}

         ]},

        {"value": "Fagaras",

         "children": [

             {"value": "Bucharest", "children": []}

         ]},

        {"value": "Craiova",

         "children": [

             {"value": "Pitesti", "children": []}

         ]},

        {"value": "Pitesti",

         "children": [

             {"value": "Bucharest", "children": []}

         ]},

        {"value": "Bucharest",

         "children": [

             {"value": "Giurgiu", "children": []},

             {"value": "Urziceni", "children": []}

         ]},

        {"value": "Urziceni",

         "children": [

             {"value": "Hirsova", "children": []},

             {"value": "Vaslui", "children": []}

         ]},

        {"value": "Hirsova",

         "children": [

             {"value": "Eforie", "children": []}

         ]},

        {"value": "Eforie", "children": []},

        {"value": "Iasi",

         "children": [

             {"value": "Neamt", "children": []}

         ]},

        {"value": "Neamt", "children": []},

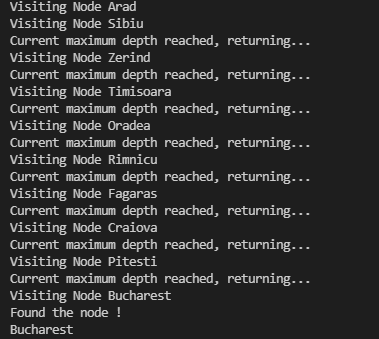
        {"value": "Vaslui", "children": []},

        {"value": "Giurgiu", "children": []}

    ]

}

**Output:**

****

**Graded Task 2:**

#SP22-BSE-017 HASAAN AHMAD

dictionary = ["START","NOTE", "SAND", 'STONED']

n = len(dictionary)

M = 4

N = 4

def isWord(Str):

    for i in range(n):

        if (Str == dictionary[i]):

            return True

    return False

def findWordsUtil(boggle, visited, i, j, Str, depth):

    if isWord(Str):

        print(Str)

    if depth <= 0:

        return

    row = i - 1

    while row <= i + 1 and row < M:

        col = j - 1

        while col <= j + 1 and col < N:

            if row >= 0 and col >= 0 and not visited[row][col]:

                findWordsUtil(boggle, visited, row, col, Str + boggle[row][col], depth - 1)

            col += 1

        row += 1

    visited[i][j] = False

def findWords(boggle):

    visited = [[False for \_ in range(N)] for \_ in range(M)]

    for depth in range(1, max(M, N) + 1):

        for i in range(M):

            for j in range(N):

                findWordsUtil(boggle, visited, i, j, boggle[i][j], depth)

boggle = [

    ["M", "S", "E", "F"],

    ["R", "A", "T", "D"],

    ["L", "O", "N", "E"],

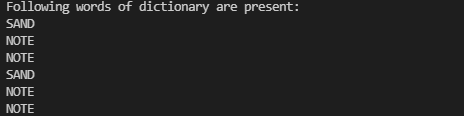
    ["K", "A", "F", "B"]

]

print("Following words of", "dictionary are present:")

findWords(boggle)

**Output:**

****