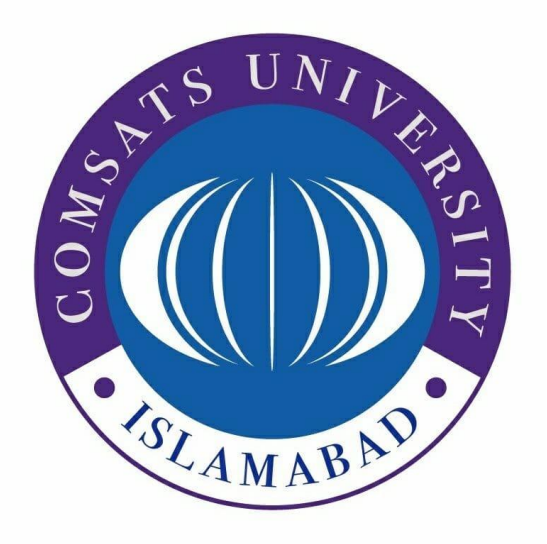
COMSATS UNVERISTY ISLAMABAD



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:

```
PS D:\AI-lab-manual-Solved-COMSATS-University-Islamabad-main\Lab 5> & "
Visiting Node0
Visiting Node1
Current Maximum Depth Reached, Returning
Increasing Depth to 2
Visiting Node0
Visiting Node1
Visiting Node3
Current Maximum Depth Reached, Returning
Visiting Node6
found
6
PS D:\AI-lab-manual-Solved-COMSATS-University-Islamabad-main\Lab 5>
```

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:

```
Visiting Node Arad
Visiting Node Sibiu
Current maximum depth reached, returning...
Visiting Node Zerind
Current maximum depth reached, returning...
Visiting Node Timisoara
Current maximum depth reached, returning...
Visiting Node Oradea
Current maximum depth reached, returning...
Visiting Node Rimnicu
Current maximum depth reached, returning...
Visiting Node Fagaras
Current maximum depth reached, returning...
Visiting Node Craiova
Current maximum depth reached, returning...
Visiting Node Pitesti
Current maximum depth reached, returning...
Visiting Node Bucharest
Found the node !
Bucharest
```

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:

```
Following words of dictionary are present:
SAND
NOTE
NOTE
SAND
NOTE
NOTE
```