Name: Dhruv Bheda **SAPID:** 60004200102

DIV: B/B1

A.I.

Exp2

Aim: Perform uninformed searching techniques like BFS, DFS, etc.

Theory:

BFS:

Breadth-first search is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. BFS or Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighbouring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

Algorithm:

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set

their STATUS = 2

(waiting state)

[END OF LOOP]

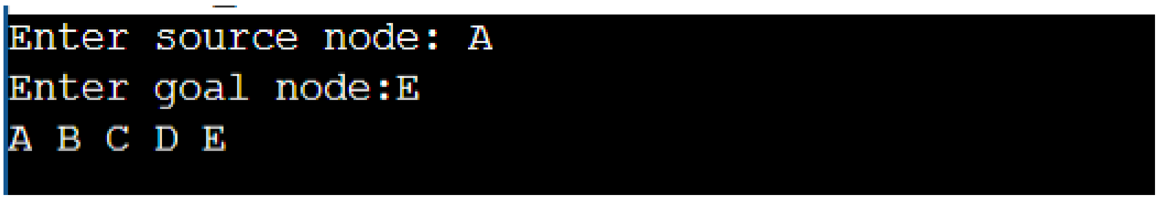
Step 6: EXIT

Code:

```
graph = {
   "A" : ["B","C"], "B" : ["D","E"],"C":["G"],"D":[" "], "E":["F"],"F":[" "],"G":[" "]
}

def bfs(visited, graph, src, goal):
  visited.append(src)
  queue.append(src)

  while queue:
    m = queue.pop(0)
    print (m, end = " ")
    if(m==goal):
        break

    for i in graph[m]:
      if(i==" " and i!=goal):
        continue
      elif i not in visited:
        visited.append(i)
        queue.append(i)
```

```
visited = []
queue = []
src=input("Enter source node: ")
goal=input("Enter goal node:")

bfs(visited, graph, src,goal)
```

<u>Output:</u>



```
Enter source node: A
Enter goal node:E
A B C D E
```

<u>DFS:</u>

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node and explores as far as possible along each branch before backtracking

It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children. Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

<u>Algorithm:</u>

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

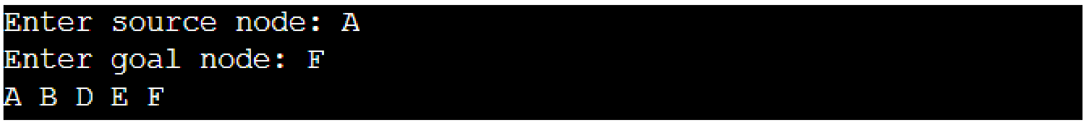
[END OF LOOP]

Step 6: EXIT

<u>Code:</u>

```
def dfs(graph,visited,src,goal):
    if src not in visited:
      print(src , end = " ")
      visited.append(src)
    if(src==goal):
      print()
      exit(dfs)
    for i in graph[src]:
        dfs(graph,visited,i,goal)

graph = {
   "A" : ["B","C"], "B" : ["D","E"],"C":["G"],"D":[], "E":["F"],"F":[],"G":[]
}

visited = []
src=input("Enter source node: ")
goal=input("Enter goal node: ")
dfs(graph,visited,src,goal)
```

Output:



```
Enter source node: A
Enter goal node: F
A B D E F
```

DFID:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found. This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency. The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Algorithm:

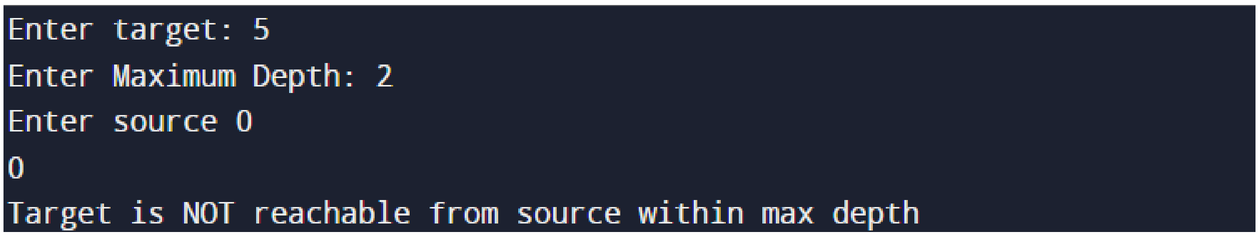
```
Step1: INPUT: START and GOAL states

Step2: LOCAL VARIABLE: Found

Step3: Initialise d = 1 and FOUND = False

Step4: while (FOUND = False) do
           perform DFS from start to depth d.

Step5: if goal state is obtained then FOUND = True
          else discard the nodes  generated in the search of depth d.

Step6: d = d + 1

Step7: if FOUND = true, then return the depth.

Step8: Stop
```

<u>Code:</u>

```
from collections import defaultdict
class Graph:
    def __init__(self,vertices):
        self.V = vertices
        self.graph = defaultdict(list)

    def addEdge(self,u,v):
        self.graph[u].append(v)

    def DLS(self,src,target,maxDepth):

        if src == target : return True

        if maxDepth <= 0 : return False

        for i in self.graph[src]:
                if(self.DLS(i,target,maxDepth-1)):
                    return True
        return False

    def IDDFS(self,src, target, maxDepth):
        for i in range(maxDepth):
            if (self.DLS(src, target, i)):
                return True
        return False
```

```
g = Graph (7);
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 3)
g.addEdge(1, 4)
g.addEdge(2, 5)
g.addEdge(2, 6)

target = int(input("Enter target: "));
maxDepth = int(input("Enter Maximum Depth: "));
src = int(input("Enter source"));

if g.IDDFS(src, target, maxDepth) == True:
    print ("Target is reachable from source " +
        "within max depth")
else :
    print ("Target is NOT reachable from source " +
        "within max depth")
```

Output:



```
Enter target: 5
Enter Maximum Depth: 2
Enter source 0
0
Target is NOT reachable from source within max depth
```

Conclusion:

Thus, we successfully studied Uninformed Searching Algorithms like BFS, DFS, DFID etc.