Prcatical No. 1: Manual Content

	Guru Gobind Singh Foundation's Guru Gobind Singh College of Engineering and Research Center, Nashik	

Experiment No: 01

Title of Experiment: 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..

Student Name:			
Class:	TE (Computer)		
Div:	-	Batch:	CO
Roll No.:			
Date of Attendance (Performance):			
Date of Evaluation:			
Marks (Grade) Attainment of CO Marks out of 10			
CO Mapped	CO1. Design a system using different informed search / uninformed search or heuristic approaches		
Signature of Subject Teacher			

TITLE: Implement DFS and BFS Algorithm. Use and Undirected Graph and develop a Recursive Algorithm for searching all the vertices of the graph or tree data structure.

AIM: Aim of this practical is to develop DFS and BFS algorithm programs in programming language.

OBJECTIVES: Based on above main aim following are the objectives

1. To study BFS
2. To study DFS
3. To apply algorithmic logic in implementation of program.

1. Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.

- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.

- The breadth-first search algorithm is an example of a general-graph search algorithm.

- Breadth-first search implemented using FIFO queue data structure.
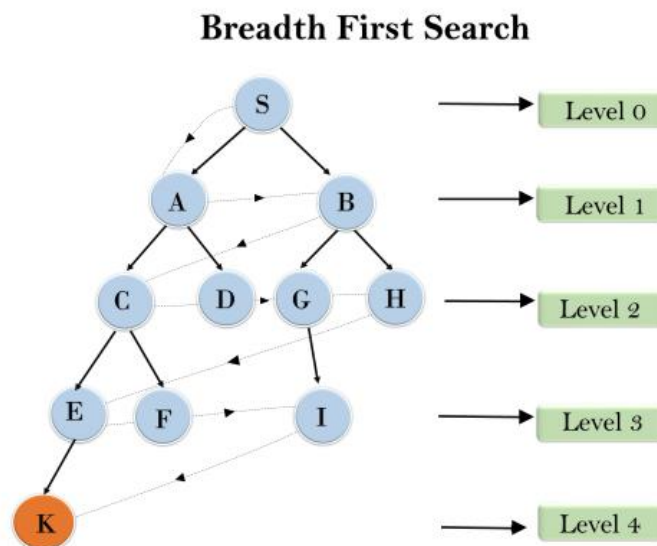
In the above tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K

- **Time Complexity:** Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.
- $T(b) = 1+b^2+b^3+.......+ b^d = O(b^d)$
- **Space Complexity:** Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.
- **Completeness:** BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.
- **Optimality:** BFS is optimal if path cost is a non-decreasing function of the depth of the node.

Algorithm:
1. Pick any node, visit the adjacent unvisited vertex, mark it as visited, display it, and insert it in a queue.
2. If there are no remaining adjacent vertices left, remove the first vertex from the queue.
3. Repeat step 1 and step 2 until the queue is empty or the desired node is found.

Program:

```
graph = {
  'A' : ['B','C'],
  'B' : ['D', 'E'],
  'C' : ['F'],
  'D' : [],
  'E' : ['F'],
  'F' : []
}
visited = [] # List to keep track of visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)
  while queue:
    s = queue.pop(0)
    print (s, end = " ")

    for neighbour in graph[s]:
      if neighbour not in visited:
```

```
            visited.append(neighbour)
            queue.append(neighbour)

# Driver Code
print("Following is the Path using Breadth-First Search")
bfs(visited, graph, 'A')
```

Output:
Following is the Path using Breadth-First Search
A B C D E F

2. Depth-first Search:

- Depth-first search isa recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Example: In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.

Depth First Search
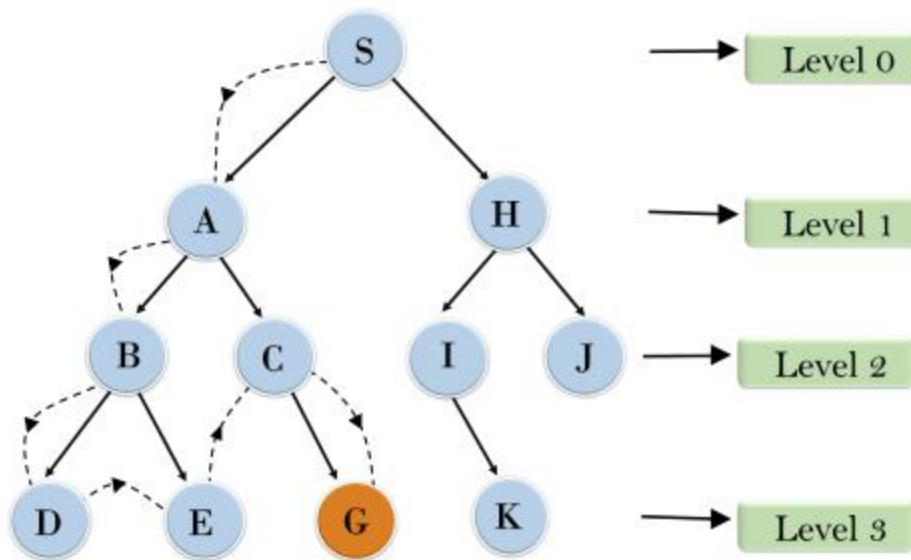
- **Completeness:** DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.
- **Time Complexity:** Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$$T(n)= 1+ n^2+ n^3 +.........+ n^m=O(n^m)$$

- **Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)**
- **Space Complexity:** DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

- **Optimal:** DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

Algorithm:
1. We will start by putting any one of the graph's vertex on top of the stack.   2. After that take the top item of the stack and add it to the visited list of the vertex.   3. Next, create a list of that adjacent node of the vertex. Add the ones which aren't in the visited list of vertexes to the top of the stack.   4. Lastly, keep repeating steps 2 and 3 until the stack is empty.
Program:

```
# Using a Python dictionary to act as an adjacency list
graph = {
    'A' : ['B','C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
print("Following is the Path using Depth-First Search")
dfs(visited, graph, 'A')
```

Output:
Following is the Path using Depth-First Search
A
B
D
E
F
C

Conclusion: Thus we have studied BFS & DFS algorithm in detail and implemented using recursive function.