Experiment No. 3
Implementation of Breadth first search for problem solving.
Date of Performance:
Date of Submission:

Aim: Implementation of Breadth first search for problem solving

Objective: To study the uninformed searching techniques and its implementation for problem solving.

Theory:

Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

A search problem consists of:

- **A State Space**. Set of all possible states where you can be.
- **A Start State**. The state from where the search begins.
- **A Goal Test**. A function that looks at the current state returns whether or not it is the goal state.

The Solution to a search problem is a sequence of actions, called the plan that transforms the start state to the goal state.

This plan is achieved through search algorithms.

Breadth First Search: BFS is a uninformed search method. It is also called blind search. Uninformed search strategies use only the information available in the problem definition. A search strategy is defined by picking the order of node expansion. It expands nodes from the root of the tree and then generates one level of the tree at a time until a solution is found. It is very easily implemented by maintaining a queue of nodes. Initially the queue contains just the root. In each iteration, node at the head of the queue is removed and then expanded. The generated child nodes are then added to the tail of the queue.

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbor nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbor nodes.

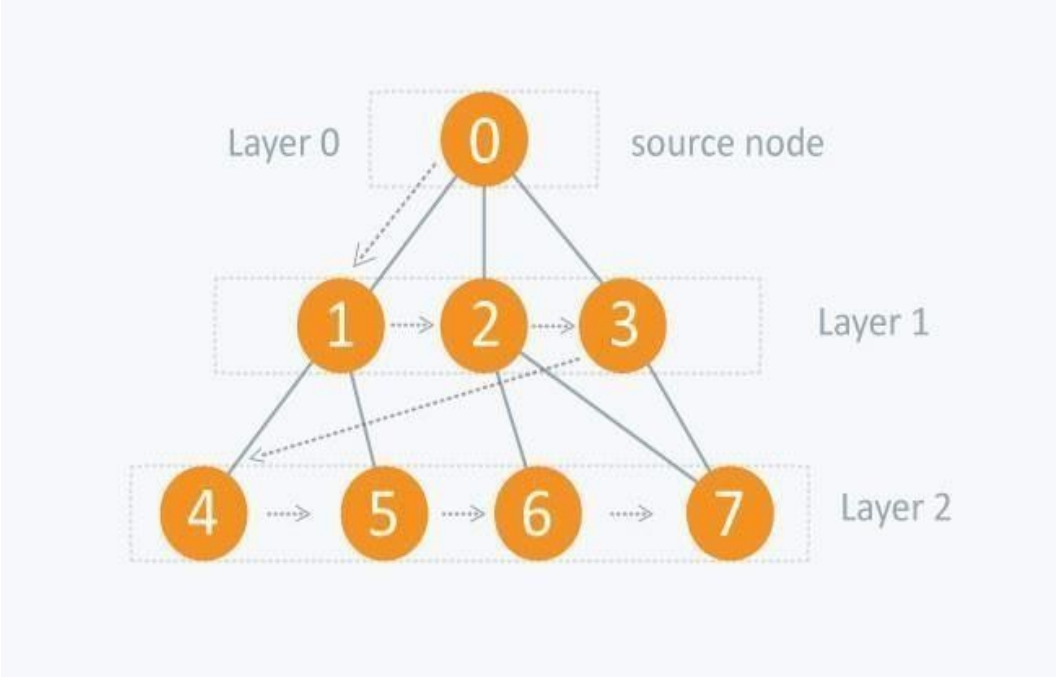
As the name BFS suggests, you are required to traverse the graph breadthwise as follows:
1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer

BFS Algorithm:

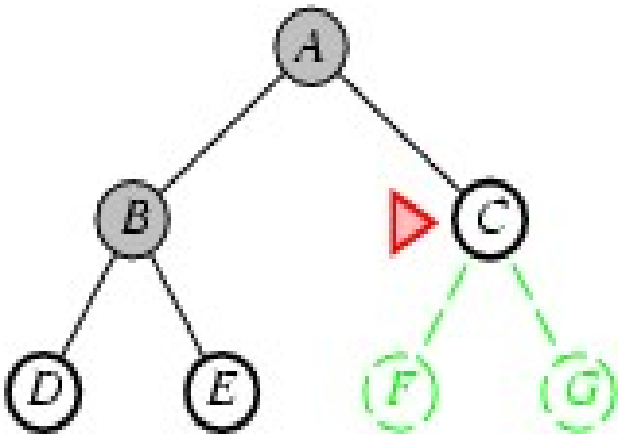
Pseudocode:

```
BFS (G, s) //Where G is the graph and s is the source node
let Q be queue.
Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.
mark s as visited.

while ( Q is not empty)
//Removing that vertex from queue,whose neighbour will be visited now v
= Q.dequeue( )

//processing all the neighbours of v
for all neighbours w of v in Graph G
if w is not visited
Q.enqueue( w ) //Stores w in Q to further visit its neighbour
mark w as visited
```

Working of BFS:



Example: Initial Node: A Goal Node: C



Searching Strategies are evaluated along the following dimensions:

1. Completeness: does it always find a solution if one exists?
2. Time complexity: number of nodes generated
3. Space complexity: maximum number of nodes in memory
4. Optimality: does it always find a least-cost solution?

Properties of Breadth-first search:

1. **Complete**: - Yes: if b is finite.
2. **Time Complexity**: O(b^d+1)
3. **Space Complexity**: O(b^d+1)
4. **Optimal**: Yes

Advantages of Breadth-First Search:

1. Breadth first search will never get trapped exploring the useless path forever.
2. If there is a solution, BFS will definitely find it out.
3. If there is more than one solution then BFS can find the minimal one that requires less number of steps.

Disadvantages of Breadth-First Search:

1. The main drawback of Breadth first search is its memory requirement. Since each level of the tree must be saved in order to generate the next level, and the amount of memory is proportional to the number of nodes stored.
2. If the solution is farther away from the root, breath first search will consume lot of time.

Conclusion:

```python
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)

    while queue:
        vertex = queue.popleft()
        print(vertex, end=' ')

        for neighbor in graph[vertex]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)

# Example usage:
if __name__ == "__main__":
    # Example graph as adjacency list
    graph = {
        'A': ['B', 'C'],
        'B': ['A', 'D', 'E'],
        'C': ['A', 'F'],
        'D': ['B'],
        'E': ['B', 'F'],
        'F': ['C', 'E']
    }

    print("BFS starting from vertex 'A':")
    bfs(graph, 'A')
```

```
BFS starting from vertex 'A':
A B C D E F
```

The experiment demonstrates the implementation of Breadth-First Search (BFS) using Python and a graph represented as an adjacency list. The BFS algorithm traverses the graph level by level, exploring all the neighbors of a vertex before moving to the next level. The provided implementation effectively explores the graph from the starting vertex 'A', visiting all connected nodes. The output shows the traversal order, which confirms the correct working of the BFS algorithm. This experiment highlights the importance of BFS for exploring unweighted graphs and solving problems like shortest path finding and network analysis.

CSL502: Artificial Intelligence Lab