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

Aim: Study and 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 layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour 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:

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100

#define initial 1
#define waiting 2

#define visited 3

int n;     /*Number of vertices in the graph*/ int adj[MAX][MAX]; /*Adjacency
Matrix*/ int state[MAX]; /*can be initial, waiting or visited*/

void create_graph(); void BF_Traversal(); void BFS(int v);

int queue[MAX], front=-1,rear=-1; void insert_queue(int vertex);
int delete_queue(); int isEmpty_queue();

int main()
{
      create_graph(); BF_Traversal();

     return 0; }/*End of main()*/
```

```
void BF_Traversal()
{
    int v;

    for(v=0; v<n; v++)
        state[v]=initial;

    printf("\nEnter starting vertex for Breadth First Search : "); scanf("%d", &v); BFS(v);
    for(v=0; v<n; v++) if(state[v] == initial)
        BFS(v); }/*End of BF_Traversal()*/

void BFS(int v)
{ int i;

    insert_queue(v); state[v]=waiting;

    while( !isEmpty_queue() )
    {
        v = delete_queue( ); printf("%d ",v); state[v] = visited;
        for(i=0; i<n; i++)
        {
            /* Check for adjacent unvisited vertices */ if( adj[v][i] == 1 && state[i] ==
            initial)
            {
                insert_queue(i); state[i] = waiting;
```

```
                }
            }
    } printf("\n"); }/*End of BFS()*/

void insert_queue(int vertex)
{
    if (rear == MAX-1) printf("Queue Overflow\n"); else
    {
        if (front == -1) /*If queue is initially empty */ front = 0; rear = rear+1;
        queue[rear] = vertex ;
    }
}/*End of insert_queue()*/

int isEmpty_queue()
{
    if(front == -1 || front > rear ) return 1; else return 0; }/*End
of isEmpty_queue()*/

int delete_queue()
{
    int del_item; if (front == -1 || front > rear)
    {
        printf("\nQueue Underflow\n");
        exit(1);
    }
```

```
    del_item = queue[front]; front = front+1; return

del_item; }/*End of delete_queue() */


void create_graph()
{
    int i,max_edges,origin,destin;


    printf("\nEnter number of vertices : "); scanf("%d",&n); max_edges =

    n*(n-1);


    for(i=1;i<=max_edges;i++)

    {

        printf("\nEnter edge %d( -1 -1 to quit ) : ",i); scanf("%d %d",&origin,&destin);


        if((origin == -1) && (destin == -1)) break;

        if( origin >= n || destin >= n || origin<0 || destin<0)

        {

            printf("\nInvalid edge!\n");

            i--;

        }

        else

        { adj[origin][destin]=1;

        }

    }/*End of for*/ }/*End of create_graph()*/ OUTPUT : :
```

Enter number of vertices : 5

Enter edge 1(-1 -1 to quit) : 0 1

Enter edge 2(-1 -1 to quit) : 0 2

Enter edge 3(-1 -1 to quit) : 0 3

Enter edge 4(-1 -1 to quit) : 1 3

Enter edge 5(-1 -1 to quit) : 3 2

Enter edge 6(-1 -1 to quit) : 4 4

Enter edge 7(-1 -1 to quit) : -1 -1

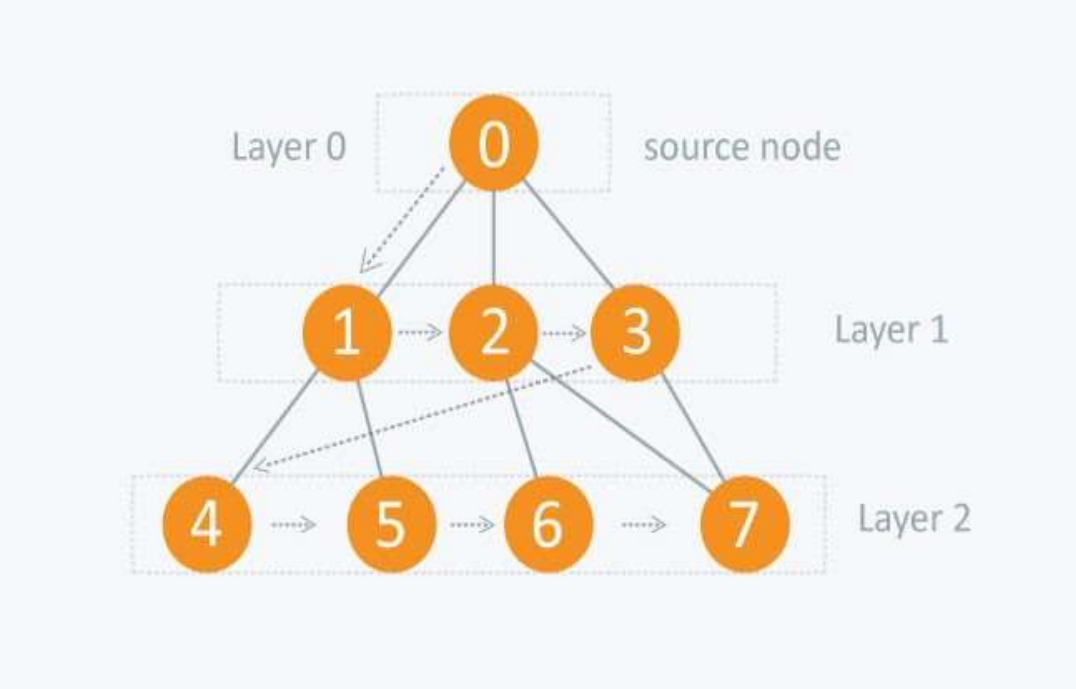
Enter starting vertex for Breadth First Search : 0

0 1 2 3

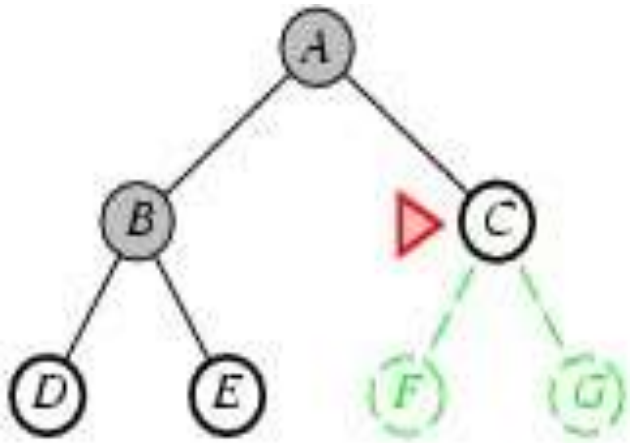
4

Process returned 0

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, the space complexity of BFS is O(bd).
2. If the solution is farther away from the root, breath first search will consume lot of time.

Applications:

How to determine the level of each node in the given tree?

As you know in BFS, you traverse level wise. You can also use BFS to determine the level of each node.

Conclusion:

In conclusion, the implementation of the breadth-first search (BFS) algorithm offers a powerful and efficient method for traversing and searching graphs or trees. By systematically exploring all vertices at each level of the graph, BFS ensures the shortest path from the start vertex to any other vertex is found. Its simplicity and effectiveness make it a popular choice in various applications such as shortest path finding, network traversal, and puzzle solving. With its broad applicability and straightforward implementation, BFS remains a fundamental tool in the arsenal of algorithms for graph traversal and search problems.