Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 20-12-2021	Date of Submission: 31-1-2022

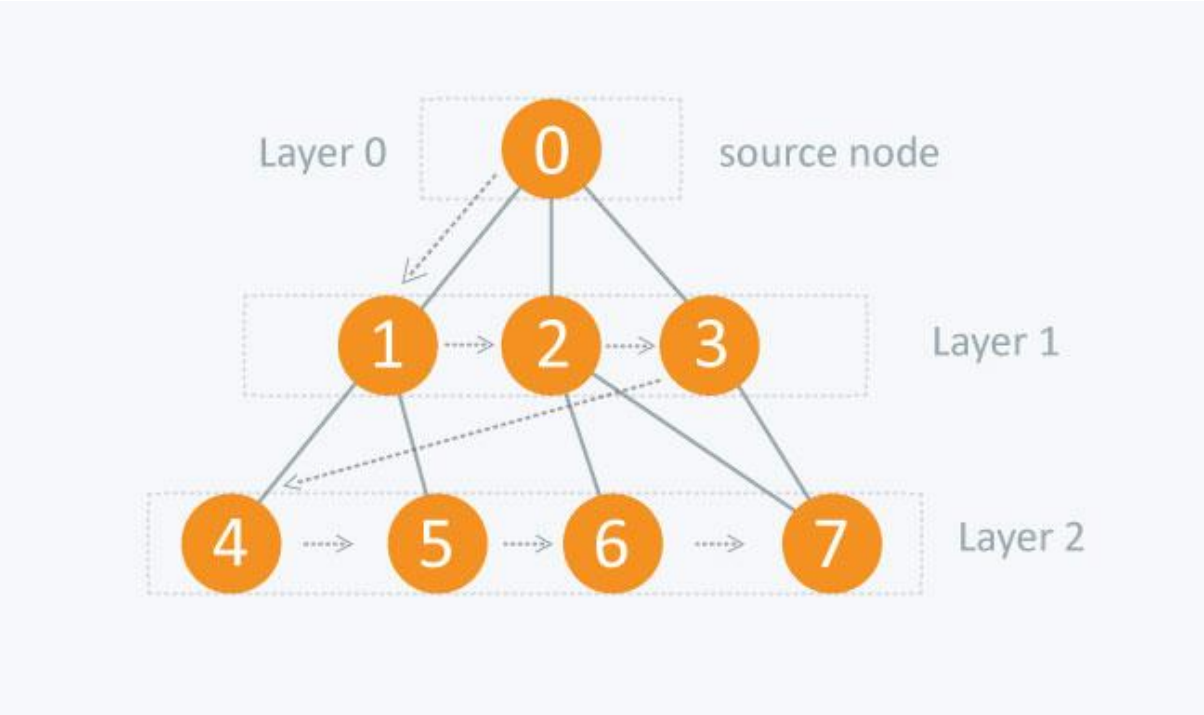
To implement Breadth First Search

Theory

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:

- First move horizontally and visit all the nodes of the current layer
- Move to the next layer



The distance between the nodes in layer 1 is comparitively lesser than the distance between the nodes in layer 2. Therefore, in BFS, you must traverse all the nodes in layer 1 before you move to the nodes in layer 2.

Traversing child nodes

A graph can contain cycles, which may bring you to the same node again while traversing the graph. To avoid processing of same node again, use a boolean array which marks the node after it is processed. While visiting the nodes in the layer of a graph, store them in a manner such that you can traverse the corresponding child nodes in a similar order.

In the earlier diagram, start traversing from 0 and visit its child nodes 1, 2, and 3. Store them in the order in which they are visited. This will allow you to visit the child nodes of 1 first (i.e. 4 and 5), then of 2 (i.e. 6 and 7), and then of 3 (i.e. 7) etc.

To make this process easy, use a queue to store the node and mark it as 'visited' until all its neighbours (vertices that are directly connected to it) are marked. The queue follows the First In First Out (FIFO) queuing method, and therefore, the neigbors of the node will be

visited in the order in which they were inserted in the node i.e. the node that was inserted first will be visited first, and so on.

Complexity

The time complexity of BFS is O(V + E), where V is the number of nodes and E is the number of edges.

Code

```
#include<stdio.h>
void adj();
void input();
void bfs(int);
int delQue();
void addQue(int);
int v,n,f=0,r=0,visited[10]={0},a[10][10],que[10]={0};
int main()
{
   printf("\tBFS TRAVERSAL\n\n");
        input();
        printf("\nEnter the starting vertex: \n");
        scanf("%d",&v);
        printf("\nBFS traversal Path: \n");
        bfs(v);
        return 0;
}
void input()
{
        int i, j;
        printf("\nEnter the number of nodes: ");
        scanf("%d",&n);
        printf("\nEnter the adjacent matrix: \n");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        scanf("%d",&a[i][j]);
}
```

```
void bfs(int v)
{
        printf("%d->", v);
        int j,u=v;
        visited[u]=1;
        while(1)
      {
        for(j=1; j<=n; j++)
        {
           if(a[u][j]!=0 && visited[j]==0)
           {
              visited[j]=1;
              addQue(j);
           }
        }
        u=delQue();
        printf("%d->", u);
        if(f==r)
           break;
      }
        printf("NULL\n");
}

void adj()
{
        int i, j;
        printf("\nAdjacency Matrix[][]: \n");
        for(i=1; i<=n; i++)
      {
        for(j=1; j<=n; j++)
           printf("%d\t", a[i][j]);
        printf("\n");
```

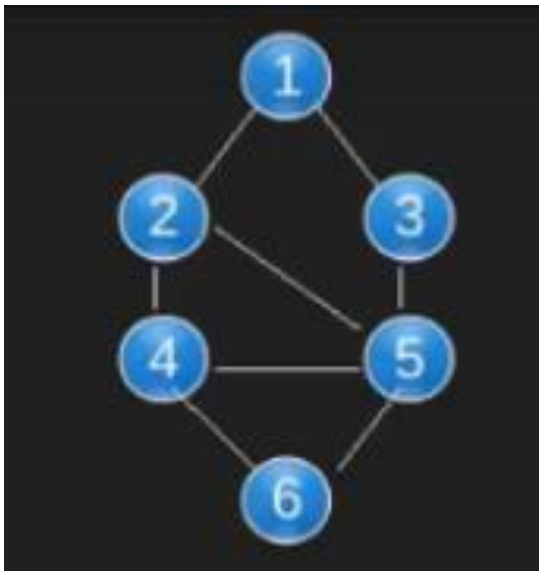
```
    }
}

void addQue(int x)
{
        que[r++]=x;
}

int delQue()
{
        return que[f++];
}
```

Output

Taking this example



```
        BFS TRAVERSAL


Enter the number of nodes: 6

Enter the adjacent matrix:
0 1 1 0 0 0
1 0 0 1 1 0
1 0 0 0 1 0
0 1 0 0 1 1
0 1 0 1 0 1
0 0 0 1 1 0

Enter the starting vertex:
1

BFS traversal Path:
1->2->3->4->5->6->NULL

Process returned 0 (0x0)   execution time : 110.237 s
Press any key to continue.
```