MIT Art Design and Technology University

MIT School of Computing, Pune

Department of Computer Science and Engineering

Second Year B. Tech

Academic Year 2022-2023. (SEM-II)

Subject: Advance Data Structures Laboratory

Assignment 5

Assignment Title: Represent a given graph using an adjacency list/matrix and perform DFS or BFS traversal.

Aim: Implement Graph and its BFS or DFS traversals.

Prerequisite:

1. Basic concepts of Graph.
2. DFS or BFS traversals of Graph

Objectives:

Implement a Program in C++ for the following operations on Graph:

1. Create a Graph of N nodes using Adjacency Matrix/adjacency list.
2. Recognize and define the basic attributes of a Graph.
3. Print all the nodes reachable from a given starting node in a digraph using BFS or DFS method

Outcomes:

Upon Completion of the assignment the students will be able to

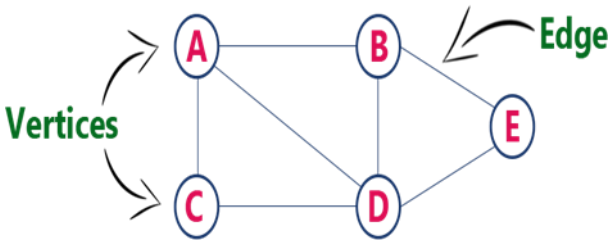
1. Create and implement Graph using adjacency list/adjacency matrix.
2. Understand and analyse DFS and BFS graph traversals.

Theory:

Graph: Graph is a non-linear data structure. It contains a set of points known as nodes (or vertices) and a set of links known as edges (or Arcs). Here edges are used to connect the vertices.

Generally, a graph G is represented as G = (V , E), where V is set of vertices and E is set of edges.

Example:



The figure shows is a graph with 5 vertices and 6 edges.

This graph G can be defined as G = (V , E)

Where V = {A,B,C,D,E} and

E = {(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.

Storage representation:

1. **Adjacency Matrix**
2. **Adjacency list**

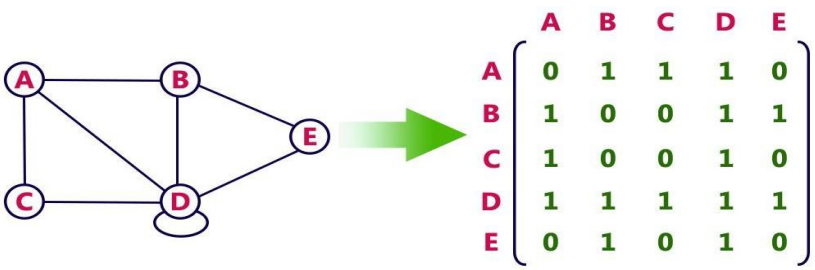
1.Adjacency Matrix

Let G = (V,E) be a graph with 'v' vertices v >= 1.

The adjacency matrix of graph G is a 2-dimensional n x n array say A[n:n]., with the property that,

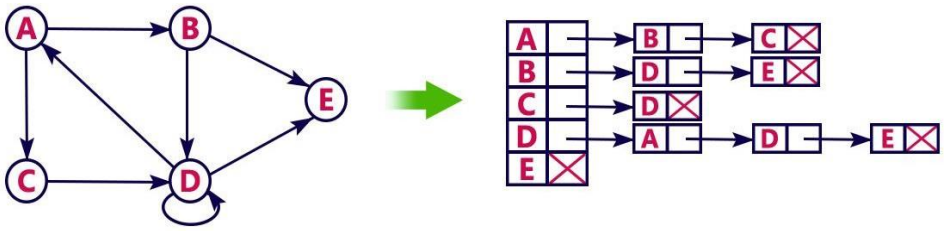
A(i, j) = 1 iff the edge (Vi, Vj) is in E(G) for undirected graph or the edge <Vi, Vj> is in E(G) for directed graph and

A(i, j) = 0 if there is no such edge in graph G from vertex Vi to Vj.



2. Adjacency list

In this representation the 'n' rows of adjacency matrix are represented as 'n' linked lists. There is one list for each vertex in a graph. The nodes in list i represent the vertices that are adjacent to vertex i.



Graph Traversal:

Graph traversal is a technique used for a searching vertex in a graph.

The graph traversal is used to decide the order of vertices is visited in the search process.

A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.

There are two graph traversal techniques and they are as follows...

- DFS (Depth First Search)
- BFS (Breadth First Search)

DFS traversal:

DFS traversal of a graph produces a spanning tree as final result. Spanning Tree is a graph without loops.

We use Stack data structure with maximum size of total number of vertices in the graph to implement DFS traversal.

We use the following steps to implement DFS traversal:

Step 1 - Define a Stack of size total number of vertices in the graph.

Step 2 - Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.

Step 3 - Visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it on to the stack.

Step 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.

Step 5 - When there is no new vertex to visit then use back tracking and pop one vertex from the stack.

Step 6 - Repeat steps 3, 4 and 5 until stack becomes Empty.

Step 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the grap

BFS traversal:

BFS traversal of a graph produces a spanning tree as final result. Spanning Tree is a graph without loops.

We use Queue data structure with maximum size of total number of vertices in the graph to implement BFS traversal.

We use the following steps to implement BFS traversal :

Step 1 - Define a Queue of size total number of vertices in the graph.

Step 2 - Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.

Step 3 - Visit all the non-visited adjacent vertices of the vertex which is at front of the Queue and insert them into the Queue.

Step 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.

Step 5 - Repeat steps 3 and 4 until queue becomes empty.

Step 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph.

Analysis of algorithms:

1) **Time Complexity:**

For the construction of an undirected graph with ‘n’ vertices using adjacency matrix is O (n^2), since for every pair (i, j) we need to store either ‘0’ or ‘1’ in an array of size n x n.

For the construction of an undirected graph with ‘v’ vertices & ‘e’ edges using adjacency list is O (v + e), since for every vertex v in G we need to store all adjacent edges to vertex v.

For DFS and BFS traversals of an undirected graph with ‘n’ vertices using adjacency matrix is O (n^2), since we visit to every value in an array of size n x n.

For DFS and BFS traversals of an undirected graph with ‘v’ vertices using adjacency list O(e), since we visit to each node in adjacency list exactly ones which is equal to number of edges in a graph.

2) Space Complexity:

Additional space required for DFS and BFS to store the intermediate elements in STACK and QUEUE respectively while traversing the graph.

Conclusion:

Hence, we have implemented DFS and BFS traversal. DFS uses Stack to find the shortest path. BFS is better when target is closer to Source. DFS is better when target is far from source.

Frequently ask questions:

1. Write down algorithm for DFS and BFS graph traversal.
2. Distinguish between DFS and BFS.
3. What is the time and space complexity of DFS and BFS?
4. List out applications of DFS and BFS.

MIT SCHOOL OF ENGINEERING
Rajbaug, Loni-Kalbhor, Pune

MIT-ADT UNIVERSITY

Name : Sourav Shailesh Toshniwal
Class : SY CSE - 1 (B)
Roll no : 2213047
Subject : ADSL. Assignment - 5

1. Write down algorithm of DFS and BFS graph traversal

Ans DFS Algorithm:
- Initialize a stack and visited set
- Push the starting node onto the stack and mark it as visited
- While the stack is not empty:
  - Pop a node from the top of stack
  - for each of its unvisited neighbours :
    - Mark the neighbour visited
    - Push the neighbour onto the stack.
- Return the visited node in the order they were visited.

BFS Algorithm:
- Initialize a queue and visited set.
- Enqueue the starting node onto the queue & mark it as visited
- While the queue is not empty.
  - Dequeue a node from the front of the queue.
  - for each of its unvisited node / neighbours :
    - Mark the neighbour visited
    - Enqueue the neighbour onto the back of queue.
- Return the visited nodes in the order they were visited.

2. Distinguish between DFS and BFS.

Ans i) Order of visited node : DFS nodes are visited in DF order while BFS nodes are visited in Breadth first order.
ii) Data structure used for traversal : DFS uses a stack to keep

track of the nodes to visit, BFS uses a queue.

iii) Space complexity: DFS has a lower space complexity, BFS has higher space complexity.

iv) Use cases: DFS is often used for solving problems that involve finding a path or cycle in a graph, BFS is used for finding shortest path.

3) What is the Time and space complexity of DFS & BFS?

Ans i) Time complexity (DFS): O(V+E)
Space complexity (DFS): O(V)
ii) Time Complexity (BFS): O(V+E)
Space complexity (BFS): O(V)
where V → no of vertices & E → no of edges.

4) List out applications of DFS & BFS.

Ans i) Applications of DFS:
- Pathfinding & maze solving
- Detecting cycles
- Topological sorting
- Finding Connected components.
- Finding strongly connected components.

ii) Applications of BFS:
- Shortest path finding
- Web crawlers
- Social Network Analysis
- Minimum Spanning Tree
- Game AI.

track of the nodes to visit, BFS uses a queue.

iii) Space complexity: DFS has a lower space complexity, BFS has higher space complexity.

iv) Use cases: DFS is often used for solving problems that involve finding a path or cycle in a graph, BFS is used for finding shortest path.

3) What is the time and space complexity of DFS & BFS?

Ans i) Time complexity (DFS): O (V+E)
Space complexity (DFS): O(V)
ii) Time Complexity (BFS): O(V+E)
Space complexity (BFS): O (V)
where V → no of vertices & E → no of edges.

4) List out applications of DFS & BFS.

Ans i) Applications of DFS:
- Pathfinding & maze solving
- Detecting cycles
- Topological sorting
- Finding Connected components.
- Finding strongly connected components.

ii) Applications of BFS:
- Shortest path finding
- Web crawlers
- Social Network Analysis
- Minimum Spanning Tree
- Game AI.

Code:

```cpp
#include <iostream>
using namespace std;

#define MAX 30

struct Node {
    string data;
    Node *next;
};

class BFS {
public:
    int n, edges;
    string vertex;
    Node *head[MAX];

    BFS() {
        for (int i = 0; i < MAX; i++) {
            head[i] = NULL;
        }
    }

    void graph() {
        cout << "Enter the number of vertices: ";
        cin >> n;
        for (int i = 0; i < n; i++) {
            cout << "Enter the vertex " << i << ": ";
            cin >> vertex;
            head[i] = new Node;
            head[i]->data = vertex;
            head[i]->next = NULL;
            cout << "Enter the number of adjacent edges to the vertex: ";
            cin >> edges;
            Node *curr = head[i];
            for (int j = 0; j < edges; j++) {
                Node *adj = new Node;
                cout << "Enter adjacent vertex " << j << " of " << vertex
<< ": ";
                cin >> adj->data;
                adj->next = NULL;
                curr->next = adj;
                curr = curr->next;
```

```
        }
    }
}


void adjacency_list() {
    for (int i = 0; i < n; i++) {
        cout << head[i]->data << "->";
        Node *curr = head[i]->next;
        while (curr != NULL) {
            cout << curr->data << "->";
            curr = curr->next;
        }
        cout << "NULL" << endl;
    }
}


void bfs(int start) {
    bool visited[MAX] = { false };
    visited[start] = true;
    int queue[MAX];
    int front = 0, rear = 0;
    queue[rear++] = start;


    while (front < rear) {
        int curr = queue[front++];
        cout << head[curr]->data << " ";


        Node *adj = head[curr]->next;
        while (adj != NULL) {
            int index = get_vertex_index(adj->data);
            if (!visited[index]) {
                visited[index] = true;
                queue[rear++] = index;
            }
            adj = adj->next;
        }
    }
    cout<<endl;
}


int get_vertex_index(string v) {
    for (int i = 0; i < n; i++) {
```

```
            if (head[i]->data == v) {
                return i;
            }
        }
        return -1;
    }
};



int main()
{
    int x,t=1;
    BFS b;
    while(t == 1)
    {
        cout<<"Enter\n1.Create graph\n2. Display Adjacency
list\n3.BFS\n4.exit: \n";
        cin>>x;
        switch(x)
        {
            case 1 : b.graph();
            break;
            case 2 : b.adjacency_list();
            break;
            case 3 : b.bfs(0);
            break;
            case 4 : t=0;
            break;
        }
    }
    return 0;
}
```

Output:

```
PS C:\SOURAV\CODE\C++ language codes\ADS assignment> cd "c:\SOURAV\CODE\C++ language codes\ADS assignment\" ; if ($?) {
Enter
1.Create graph
2. Display Adjacency list
3.BFS
4.exit:
1
Enter the number of vertices: 4
Enter the vertex 0: 2
Enter the number of adjacent edges to the vertex: 1
Enter adjacent vertex 0 of 2: 4
Enter the vertex 1: 7
Enter the number of adjacent edges to the vertex: 2
Enter adjacent vertex 0 of 7: 4
Enter adjacent vertex 1 of 7: 8
Enter the vertex 2: 4
Enter the number of adjacent edges to the vertex: 3
Enter adjacent vertex 0 of 4: 2
Enter adjacent vertex 1 of 4: 7
Enter adjacent vertex 2 of 4: 8
Enter the vertex 3: 8
Enter the number of adjacent edges to the vertex: 2
Enter adjacent vertex 0 of 8: 4
Enter adjacent vertex 1 of 8: 7
Enter
1.Create graph
2. Display Adjacency list
3.BFS
4.exit:
2
2->4->NULL
7->4->8->NULL
4->2->7->8->NULL
8->4->7->NULL
Enter
1.Create graph
2. Display Adjacency list
3.BFS
4.exit:
3
2 4 7 8
Enter
1.Create graph
2. Display Adjacency list
3.BFS
4.exit:
```