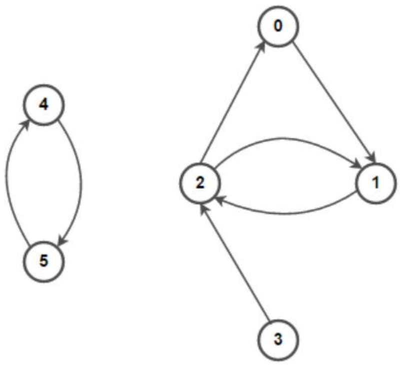
DATA STRUCTURES LABORATORY MANUAL – ICE 2144

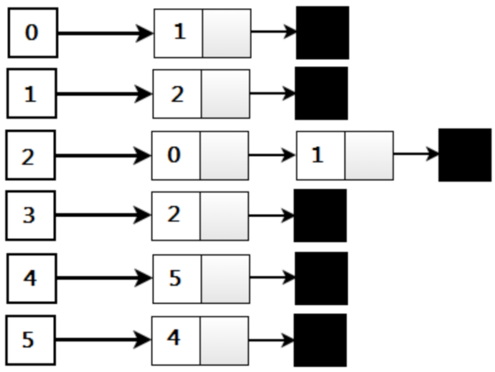
III SEMESTER B. TECH

EXPT. 9 GRAPHS

1) Given an undirected or a directed graph, write a program to implement the graph data structure.



(Graph structure)



(Adjacency list representation)

Program:
```
#include <iostream>
using namespace std;

// Data structure to store adjacency list nodes
class Node
{
        public:
                int val;
                Node* next;
};

// Data structure to store a graph edge
class Edge
{
        public:
                int src, dest;
};

class Graph
{
        // Function to allocate a new node for the adjacency list
        Node* getAdjListNode(int dest, Node* head)
        {
                Node* newNode = new Node;
                newNode→val = dest;

                // point new node to the current head
```

```
        newNode→next = head;
        return newNode;
}
int N;  // total number of nodes in the graph
public:
// An array of pointers to Node to represent the adjacency list
        Node **head;

// Constructor
        Graph(Edge edges[], int n, int N)
        {
                // allocate memory
                head = new Node*[N]();
                this→N = N;

                // initialize head pointer for all vertices
                for (int i = 0; i < N; i++)
                {
                        head[i] = NULL;
                }

                // add edges to the directed graph
                for (unsigned i = 0; i < n; i++)
                {
                        int src = edges[i].src;
                        int dest = edges[i].dest;

                        // insert at the beginning
                        Node* newNode = getAdjListNode(dest, head[src]);

                        // point head pointer to the new node
                        head[src] = newNode;

                // uncomment the following code for undirected graph

                        /*
                        newNode = getAdjListNode(src, head[dest]);

                        // change head pointer to point to the new node
                        head[dest] = newNode;
                        */
                }
        }
// Destructor
        ~Graph()
        {
```

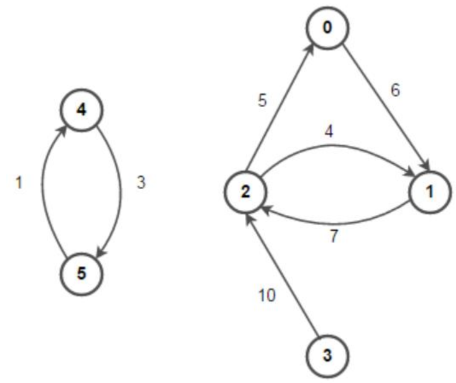
```
                for (int i = 0; i < N; i++)
                {
                        delete[] head[i];
                }

                delete[] head;
        }
};
// Function to print all neighboring vertices of a given vertex
void printList(Node* ptr)
{
        while (ptr != NULL)
        {
                cout << " → " << ptr→val;
                ptr = ptr→next;
        }
        cout << endl;
}
int main()
{
        // an array of graph edges as per the above diagram
        Edge edges[] =
        {
                // pair {x, y} represents an edge from `x` to `y`
                {0, 1}, {1, 2}, {2, 0}, {2, 1}, {3, 2}, {4, 5}, {5, 4}
        };
        // total number of nodes in the graph (labelled from 0 to 5)
        int N = 6;
        // calculate the total number of edges
        int n = sizeof(edges)/sizeof(edges[0]);

        // construct graph
        Graph graph(edges, n, N);

        // print adjacency list representation of a graph
        for (int i = 0; i < N; i++)
        {
                // print given vertex
                cout << i;
                // print all its neighboring vertices
                printList(graph.head[i]);
        }
        return 0;
}
```

2) Write a program to implement a weighted graph data structure.



Weighted graph

Program:

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

// Data structure to store adjacency list nodes
class Node
{
   public:
          int val, cost;
          Node* next;
};

// Data structure to store a graph edge
class Edge
{
   public:
          int src, dest, weight;
};

class Graph
{
          // Function to allocate a new node for the adjacency list
          Node* getAdjListNode(int value, int weight, Node* head)
          {
                    Node* newNode = new Node;
                    newNode→val = value;
                    newNode→cost = weight;

                    // point new node to the current head
                    newNode→next = head;

                    return newNode;
```

```
        }

        int N;  // total number of nodes in the graph

    public:

        // An array of pointers to Node to represent the
        // adjacency list
        Node **head;

        // Constructor
        Graph(Edge edges[], int n, int N)
        {
            // allocate memory
            head = new Node*[N]();
            this→N = N;

            // initialize head pointer for all vertices
            for (int i = 0; i < N; i++) {
                head[i] = NULL;
            }

            // add edges to the directed graph
            for (unsigned i = 0; i < n; i++)
            {
                int src = edges[i].src;
                int dest = edges[i].dest;
                int weight = edges[i].weight;

                // insert at the beginning
                Node* newNode = getAdjListNode(dest, weight, head[src]);

                // point head pointer to the new node
                head[src] = newNode;

                // uncomment the following code for undirected graph

                /*
                newNode = getAdjListNode(src, weight, head[dest]);

                // change head pointer to point to the new node
                head[dest] = newNode;
                */
            }
        }
```

```
        // Destructor
        ~Graph() {
                for (int i = 0; i < N; i++) {
                        delete[] head[i];
                }

                delete[] head;
        }
};
// Function to print all neighboring vertices of a given vertex
void printList(Node* ptr, int i)
{
        while (ptr != NULL)
        {
                cout << "(" << i << ", " << ptr→val << ", " << ptr→cost << ") ";
                ptr = ptr→next;
        }
        cout << endl;
}
// Graph implementation in C++ without using STL
int main()
{
        // an array of graph edges as per the above diagram
        Edge edges[] =
        {
                // (x, y, w) —> edge from `x` to `y` having weight `w`
                {0, 1, 6}, {1, 2, 7}, {2, 0, 5}, {2, 1, 4}, {3, 2, 10}, {4, 5, 1}, {5, 4, 3}
        };

        // total number of nodes in the graph (labelled from 0 to 5)
        int N = 6;

        // calculate the total number of edges
        int n = sizeof(edges)/sizeof(edges[0]);

        // construct graph
        Graph graph(edges, n, N);

        // print adjacency list representation of a graph
        for (int i = 0; i < N; i++)
        {
                // print all neighboring vertices of a vertex `i`
                printList(graph.head[i], i);
        }
        return 0;
}
```

DATA STRUCTURES LABORATORY MANUAL – ICE 2144

III SEMESTER B. TECH

Exercise:

1) a) Given an undirected or a directed graph, write a program to implement a graph data structure in C++ using STL.

   a) Implement for both weighted and unweighted graphs using the adjacency list representation of the graph.

2) Given an undirected graph with V vertices and E edges and a node X, write a program to find the level of node X in the undirected graph. If X does not exist in the graph then return -1. Note: Traverse the graph starting from vertex 0. (Hint: The given problem can be solved with the help of level order traversal. BFS traversal can be used in order to find the level of the given vertex.)