**Bahria University, Lahore Campus**

Department of Computer Science

Lab Journal 13

**(Spring 2023)**

|  |  |  |
| --- | --- | --- |
| Course: | **Data Structures and Algorithm - Lab** | Date: 9-06-2023\_\_\_\_\_\_ |
| Course Code: | CSL-221 | Max Marks: 10 |
| Faculty’s Name: | Fatima Zulfiqar |  |

Name: AFFAN AHMAD\_\_ Enroll No: 03-134221-003\_\_\_\_\_\_ Class: BS(cs)\_\_\_\_\_\_\_\_

Objective(s):

Upon completion of this lab session, learners will be able to:

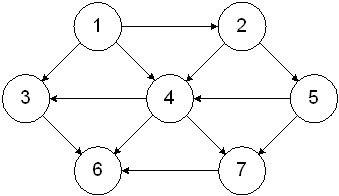
* Implement directed and undirected graph data structure
* Implement weighted directed / undirected graphs
* Insert new vertex and edges to weighted / non-weighted directed / undirected graphs
* Implement Depth First Search (DFS) algorithm and find path between two vertex

## Lab Tasks:

**Task 1**

Create separate classes for directed and undirected graph respectively using either adjacency list or adjacency matrix representation. Include methods to add a new vertex, add an directed/undirected edge, and determine if an directed/undirected edge exists between two vertices.

**Directed Graph**



#include <iostream>

using namespace std;

struct adjNode {

int val, cost;

adjNode\* next;

};

struct graphEdge {

int start, end, weight;

};

class DiaGraph{

adjNode\* getAdjListNode(int value, adjNode\* head) {

adjNode\* newNode = new adjNode;

newNode->val = value;

//newNode->cost = weight;

newNode->next = head;

return newNode;

}

int N;

public:

adjNode \*\*head;

DiaGraph(graphEdge edges[], int n, int N) {

head = new adjNode\*[N]();

this->N = N;

for (int i = 0; i < N; ++i)

head[i] = nullptr;

for (int i = 0; i < n; i++) {

int start = edges[i].start;

int end = edges[i].end;

// int weight = edges[i].weight;

adjNode\* newNode = getAdjListNode(end, head[start]);

head[start] = newNode;

}

}

};

void display\_AdjList(adjNode\* ptr, int i)

{

while (ptr != nullptr) {

cout << "(" << i << "-> " << ptr->val << ") ";

ptr = ptr->next;

}

cout << endl;

}

int main()

{

graphEdge edges[] = {

{ 1, 4 }, { 1, 3 }, { 1, 2 }, { 2, 5 }, { 2, 4 }, { 3, 6 }, { 4, 6 }, { 4, 7 }, { 5, 4 }, { 5, 7 }, { 7, 6 }

};

int N = 11;

int n = sizeof(edges) / sizeof(edges[0]);

DiaGraph diagraph(edges, n, N);

cout << "Graph adjacency list " << endl << "(start, end, weight):" << endl;

for (int i = 0; i < N; i++)

{

display\_AdjList(diagraph.head[i], i);

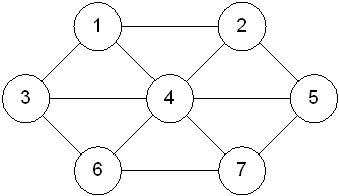
}

system("pause");

return 0;

}

**Undirected Graph**



#include <iostream>

using namespace std;

struct adjNode {

int val, cost;

adjNode\* next;

};

struct graphEdge {

int start, end, weight;

};

class DiaGraph{

adjNode\* getAdjListNode(int value, adjNode\* head) {

adjNode\* newNode = new adjNode;

newNode->val = value;

//newNode->cost = weight;

newNode->next = head;

return newNode;

}

int N;

public:

adjNode \*\*head;

DiaGraph(graphEdge edges[], int n, int N) {

head = new adjNode\*[N]();

this->N = N;

for (int i = 0; i < N; ++i)

head[i] = nullptr;

for (int i = 0; i < n; i++) {

int start = edges[i].start;

int end = edges[i].end;

// int weight = edges[i].weight;

adjNode\* newNode = getAdjListNode(end, head[start]);

head[start] = newNode;

}

}

};

void display\_AdjList(adjNode\* ptr, int i)

{

while (ptr != nullptr) {

cout << "(" << i << "-> " << ptr->val << ") ";

ptr = ptr->next;

}

cout << endl;

}

int main()

{

graphEdge edges[] = { { 1, 4 }, { 1, 3 }, { 1, 2 }, { 2, 1 }, { 2, 5 }, { 2, 4 }, { 3, 1 }, { 3, 4 }, { 3, 6 }, { 4, 6 }, { 4, 7 }, { 5, 4 }, { 5, 7 }, { 7, 6 }

};

int N = 11;

int n = sizeof(edges) / sizeof(edges[0]);

DiaGraph diagraph(edges, n, N);

cout << "Graph adjacency list " << endl << "(start, end, weight):" << endl;

for (int i = 0; i < N; i++)

{

display\_AdjList(diagraph.head[i], i);

}

system("pause");

return 0;

}

Check for edge :

**#include<iostream>**

**using namespace std;**

**bool A[10][10];**

**void initialize()**

**{**

**for(int i = 0;i < 10;++i)**

**for(int j = 0;j < 10;++j)**

**A[i][j] = false;**

**}**

**int main()**

**{**

**int x, y, nodes, edges;**

**initialize(); //Since there is no edge initially**

**cin >> nodes; //Number of nodes**

**cin >> edges; //Number of edges**

**for(int i = 0;i < edges;++i)**

**{**

**cin >> x >> y;**

**A[x][y] = true; //Mark the edges from vertex x to vertex y**

**}**

**cout<< "check the edge, enter the edge:";**

**cin>> x>>y;**

**if(A[x][y] == true)**

**cout << "There is an edge between "<<x<<" and "<<y << endl;**

**else**

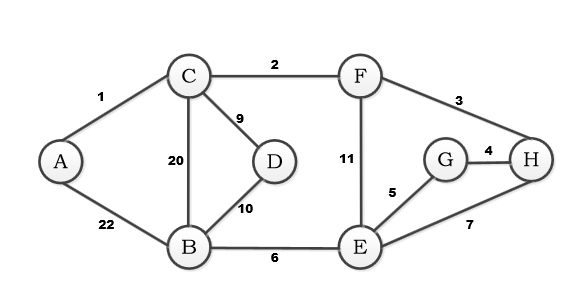
**cout << "There is no edge between "<<x<< " and "<<y << endl;**

**return 0;**

**}**

**Task 2**

Extend **Task 1** and Implement a weighted graph data structure using an adjacency list / adjacency matrix representation. Include methods to add a new vertex, add a weighted edge, and find the weight of an edge between two vertices.



#include <iostream>

using namespace std;

struct adjNode {

int val, cost;

adjNode\* next;

};

struct graphEdge {

int start, end, weight;

};

class DiaGraph{

adjNode\* getAdjListNode(int value, int weight, adjNode\* head) {

adjNode\* newNode = new adjNode;

newNode->val = value;

newNode->cost = weight;

newNode->next = head;

return newNode;

}

int N;

public:

adjNode \*\*head;

DiaGraph(graphEdge edges[], int n, int N) {

head = new adjNode\*[N]();

this->N = N;

for (int i = 0; i < N; ++i)

head[i] = nullptr;

for (int i = 0; i < n; i++) {

int start = edges[i].start;

int end = edges[i].end;

int weight = edges[i].weight;

adjNode\* newNode = getAdjListNode(end,weight, head[start]);

head[start] = newNode;

}

}

};

void display\_AdjList(adjNode\* ptr, int i)

{

while (ptr != nullptr) {

cout << "(" << i << "-> " << ptr->val << ","<<ptr->cost <<") ";

ptr = ptr->next;

}

cout << endl;

}

int main()

{

graphEdge edges[] = {

{ 1, 4,1 }, { 1, 3,2 }, { 1, 2,3 }, { 2, 5,4 }, { 2, 4,5 }, { 3, 6,6 }, { 4, 6,7 }, { 4, 7,8 }, { 5, 4,9 }, { 5, 7,10 }, { 7, 6 ,11}

};

int N = 11;

int n = sizeof(edges) / sizeof(edges[0]);

DiaGraph diagraph(edges, n, N);

cout << "Graph adjacency list " << endl << "(start, end, weight):" << endl;

for (int i = 0; i < N; i++)

{

display\_AdjList(diagraph.head[i], i);

}

system("pause");

return 0;

}

**Task 3**

Using Weighted Directed Graph designed in **Task 2,** implement a method to perform a depth-first search (DFS) traversal starting from vertex **A** towards vertex **G**.

include <iostream>

#include<vector>

using namespace std;

int main()

{

int cost[10][10], i, j, k, n, e, top, v, stk[10], visit[10], visited[10];

cout << "Enter the number of vertices in the Graph: ";

cin >> n;

cout << "\nEnter the number of edges in the Graph : ";

cin >> e;

cout << "\nEnter the start and end vertex of the edges: \n";

for (k = 1; k <= e; k++)

{

cin >> i >> j;

cost[i][j] = 1;

}

cout << "\nEnter the initial vertex to start the DFS traversal with: ";

cin >> v;

cout << "\nThe DFS traversal on the given graph is : \n";

cout << v << " ";

visited[v] = 1;

k = 1;

while (k < n)

{

for (j = n; j >= 1; j--)

{

if (cost[v][j] != 0 && visited[j] != 1 && visit[j] != 1)

{

visit[j] = 1;

stk[top] = j;

top++;

}

}

v = stk[--top];

cout << v << " ";

k++;

visit[v] = 0;

visited[v] = 1;

}

cout << "\n\n\n";

return 0;

}

**Note : Attempt all tasks and get them checked by your Lab Instructor. Also for each task, attach a screenshot of the output. You are free to use any other helping functions in your code.**

**Lab Grading Sheet :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Max Marks** | **Obtained Marks** | **Comments(*if any*)** |
| 1. | 4 |  |  |
| 2. | 2 |  |  |
| 3. | 4 |  |  |
| **Total** | **10** |  | **Signature** |