|  |
| --- |
| **Computer Engineering Department - ITU** |
| **CE200L: Data Structures & Algorithms Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 17/11/2022** |
| **Teaching Assistant: Muhammad Sufyan Ashraf** | **Semester: Fall 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 13A. Basic Operations on Directed Graph using Linked List**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 100)** | **Scaled to 10** | **Total**  **(out of 10)** |
| NIMRA MAQBOOL | BSCE21012 |  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Objective**

The objective of this lab is to provide knowledge of basic data structures and their implementations.

## **Equipment and Component**

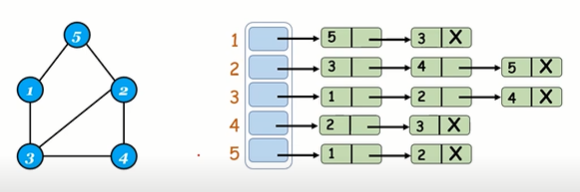
|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **Conduct of Lab**

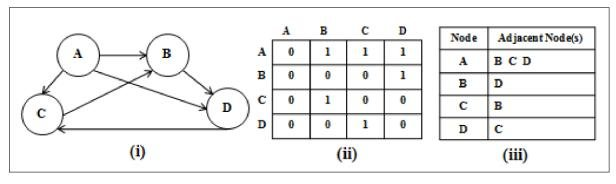
1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

## **Theory and Background**

In computer science, A **graph** data structure is a collection of nodes that have data and are connected to other nodes. Let's try to understand this through an example. On Facebook, everything is a node. That includes User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, note...anything that has data is a node. Every relationship is an edge from one node to another. Whether you post a photo, join a group, like a page, etc., a new edge is created for that relationship. More precisely, a graph is a data structure (V, E) that consists of a collection of vertices V and a collection of edges E, represented as ordered pairs of vertices (u, v).



A **directed** graph is graph, i.e., a set of objects (called vertices or nodes) that are connected together, where all the edges are directed from one vertex to another. A directed graph is sometimes called a digraph or a directed network.



A **pointer** is a variable that stores the address of another variable. Unlike other variables that hold values of a certain type, pointer holds the address of a variable. For example, an integer variable holds (or you can say stores) an integer value, however an integer pointer holds the address of a integer variable.

A **linked list** is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence.

**Templates** are a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one.

**Lab Task**

**Task A**

As you have implemented the undirected graph with arrays before, Now implement directed graph with adjancey list. Implement the following functions:

* Add elements (edges, nodes)
* Update elements (edges, nodes)
* Delete elements (edges, nodes)
* Check if the element is present in the graph
* Display directed graph

Make all necessary functions and handle all corner cases. Make a menu driven program.

|  |
| --- |
| // Paste your code here  **FUNCTION.H:**  #include <iostream>  using namespace std;  class Node { //made a class of node public:  int data; //declared data in public  Node \*nextPtr;   Node(int val) {  data = val; //made a constructor to set values  nextPtr = nullptr;  }   void setNextPtr(Node \*n) {  nextPtr = n; //set the next ptr  }   Node \*getNextPtr() {  return nextPtr; //get the value of next ptr  }   ~Node() {  nextPtr = nullptr;  } };  class linkList { //making 1 other class of linklist public:  Node \*tail;  Node \*head; //made some pointers  int count;   linkList() {  count = 0;  head = nullptr; //declared them to zero  tail = nullptr;  }   ~linkList() {  head = nullptr;  tail = nullptr;  }   Node \*getHead() const {  return head;  }   void append(int value) {  Node \*temp = new Node(value); //declaring  temp->data = value;  temp->nextPtr = nullptr; //initializing the next ptr in the next of the new initialized node  if (head == nullptr) {  head = temp; //if the head is null then store the temp in head  } else {  Node \*temp1 = head; //else make a new node  while (temp1->nextPtr != nullptr) //iterate it till the node is not null  temp1 = temp1->nextPtr; //store the temp to next ptr address  temp1->nextPtr = temp; //store pointer to the last one  }  count++;  }    bool checking(int edge) const {  bool flag = false; //declaring  if (!head) {  cout << "EMPTY LINK LIST." << endl;  } else {  Node \*temp = head; // creating and initializing  while (temp != nullptr) // loop till the null  {  if (temp->data == edge) {  flag = true;  }  temp = temp->nextPtr; // pointing to the next vertex  }  }  return flag; //returning  }   int deleteAtIndex(int index) {  int j;  if (index <= 0) {  cout << "THE INDEX MUST BE POSITIVE." << endl; //checking if it is less than zero  }  if (index == 1 && head != nullptr) { //if head is 1 or not null then make a new node  Node \*temp = head;  head = head->nextPtr; //and store the address of next ptr to that new node  delete temp; //delete new node  } else {  Node \*temp1 = head; //make a new node give it the address of head  int i = 1;  while (i < index - 1) { //iterate till 1 less position of the index  temp1 = temp1->nextPtr; //and storing the addresses  i++;  }  Node \*toDelete = temp1->nextPtr; //making new node and storing the address of the ptr of next ptr  temp1->nextPtr = temp1->nextPtr->nextPtr; //storing the value of next to next value to delete the middle one  delete toDelete; //deleting the ptr  j++;  }  count--;  return j; //returning  }   void display() const {  Node \*temp = head; //declaring  while (temp != nullptr) {  cout << temp->data << "\t"; //displaying the data  temp = temp->nextPtr; //storing the next address  cout << " "; //displaying space  }  cout << endl;  } };  class graph { public:  int size;  linkList \*array;   explicit graph(int s) {  size = s; //initializing  array = new linkList[size];   }   void input() const {  int value;  cout << "ENTER VALUES : \n";  for (int i = 0; i < size; i++) {  cout << "ENTER THE VALUE AT " << i + 1 << " = ";  cin >> value; //taking input from the user  array[i].append(value); //and appending it in array by passing that value in append function  }  }   void displayGraph() const {  cout << "\nTHE GRAPH IS. \n";  for (int i = 0; i < size; i++) {  array[i].display(); //displaying  }  }   void addElement(int vertex, int edge) const {  int temp = 0; //initializing temp and temp1 at zero  int temp1 = 0;  for (int i = 0; i < size; i++) {  if (array[i].checking(vertex) == 1) { //if the flag from the upper function is 1 then increment temp  temp++;  }  }  for (int i = 0; i < size; i++) {  if (array[i].checking(edge) == 1) { //if the flag from the upper function is 1 then increment temp1  temp1++;  }  }  if (temp > 0 && temp1 > 0) { //if tem and temp1 are 1 that means elements are present then we call the append function  for (int i = 0; i < size; i++) {  if (array[i].checking(vertex) == 1) { //double-checking  array[i].append(edge); //storing at last  }  }  }  }  void update (int node, int edge) const{  int temp=0,v;  if (edge>0){ //checking the edge is positive  for (int i=0; i<size; i++){  if(array[i].checking(node)==1){ //if the flag from the upper function is 1 then increment temp  temp++;  }  }  if (temp>0){ // if the element is present  for (int i=0; i<size; i++){  if(array[i].checking(node)==1){  temp = array[i].deleteAtIndex(edge); //getting the node to delete the node  if(temp>0){ //updating or not  cout << "ENTER THE VALUE YOU WANNA UPDATE WITH : ";  cin>> v;  array[i].append(v);  }  }  }  }  else {  cout << "VALUE DOSE NOT EXIST. \n";  }  }  else {  cout << "EDGE IS NEGATIVE.\n";  }  }    void deleteElement(int vertex, int edge) const {  int temp = 0;  if (edge > 0) { // if edge is a positive number  for(int i=0;i<size;i++){  if (array[i].checking(vertex) == 1) { //if the flag from the upper function is 1 then increment temp  temp++;  }  }  if (temp > 0) { // if the element is present  for(int i=0;i<size;i++){  if (array[i].checking(vertex) == 1) {  array[i].deleteAtIndex(edge); //getting the node to delete the node  }  }  } else {  cout << "DON'T EXIST.\n";  }  } else {  cout << "EDGE IS NEGATIVE.\n";  }  }   void check1(int value) const {  int temp = 0;  for (int i = 0; i < size; i++) {  if (array[i].checking(value) == 1) { //if the flag from the upper function is 1 then increment temp  cout << "ELEMENT IS PRESENT.\n";  temp++;  }  }  if (temp == 0) {  cout << "ELEMENT IS NOT PRESENT.\n";  }  }    ~graph() {  delete[]array; //deleting array  }   };  **MAIN.CPP:**  #include <iostream> #include "Functions.h"  using namespace std;  int main() {  graph g(5);  int opt;  int vertex;  int edge;  do {  cout << "CHOOSE OPTIONS." << endl;  cout << "1.INPUT." << endl;  cout << "2.ADD ELEMENT." << endl;  cout << "3.UPDATE ELEMENT." << endl;  cout << "4.DELETE ELEMENT." << endl;  cout << "5.CHECK IF THE VALUE IS PRESENT OR NOT." << endl;  cout << "6.EXIT." << endl;  cin >> opt;  if (opt == 1) {  g.input();  g.displayGraph();  }  if (opt == 2) {  cout << "ENTER VERTEX = ";  cin >> vertex;  cout << "ENTER EDGE = ";  cin >> edge;  g.addElement(vertex, edge);  g.displayGraph();  }  if (opt == 3) {  cout << "ENTER VERTEX = ";  cin >> vertex;  cout << "ENTER EDGE = ";  cin >> edge;  g.update(vertex, edge);  g.displayGraph();  }  if (opt == 4) {  cout << "ENTER VERTEX = ";  cin >> vertex;  cout << "ENTER EDGE = ";  cin >> edge;  g.deleteElement(vertex, edge);  g.displayGraph();  }  if (opt == 5) {  int value;  cout << "ENTER VALUE TO CHECK = ";  cin >> value;  g.check1(value);  }  if (opt == 6) {  cout << "YOU CHOOSE TO EXIT." << endl;  exit(4);   }  } while (opt >= 0 && opt <= 6);   return 0; }  // Paste your output here |

#### **Assessment Rubric for Lab**

**Method for assessment:**

Lab reports and instructor observation during lab sessions. Outcome assessed:

a. Ability to conduct experiments, as well as to analyze and interpret data (P) b. Ability to function on multi-disciplinary teams (A)

c. Ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (P)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Performance metric** | **Task** | **CLO** | **Description** | **Max marks** | **Exceeds expectation** | **Meets expectation** | **Does not meet expectation** | **Obtained marks** |
| 1. Realization of experiment (a) | 1 | 1 | Functionality | 40 | Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed (35-40) | Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed (20-34) | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed (0-19) |  |
| 2. Teamwork (b) | 1 | 3 | Group Performance | 5 | Actively engages and cooperates with other group member(s) in effective manner (4-5) | Cooperates with other group member(s) in a reasonable manner but conduct can be improved (2-3) | Distracts or discourages other group members from conducting the experiment (0-1) |  |
| 3. Conducting experiment (a, c) | 1 | 1 | On Spot Changes | 10 | Able to make changes (8-10) | Partially able to make changes (5-7) | Unable to make changes (0-4) |  |
| 1 | 1 | Viva | 10 | Answered all questions (8-10) | Few incorrect answers (5-7) | Unable to answer all questions (0-4) |  |
| 4. Laboratory safety and disciplinary rules (a) | 1 | 3 | Code commenting | 5 | Comments are added and does help the reader to understand the code (4-5) | Comments are added and does not help the reader to understand the code (2-3) | Comments are not added (0-1) |  |
| 5. Data collection (c) | 1 | 3 | Code Structure | 5 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap (4-5) | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables (2-3) | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy (0-1) |  |
| 6. Data analysis (a, c) | 1 | 4 | Algorithm | 20 | Solution is efficient, easy to understand, and maintain (15-20) | A logical solution that is easy to follow but it is not the most efficient (6-14) | A difficult and inefficient solution (0-5) |  |
| 7. Computer use (c) | 1 | 2 | Documentation & Github Submissions | 5 | Timely (4-5) | Late (2-3) | Not done (0-1) |  |
|  | Max Marks (total): | | | 100 | Obtained Marks (total): | | |  |

Lab Engineer Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_