|  |
| --- |
| **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 12A. Basic Operations on 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**

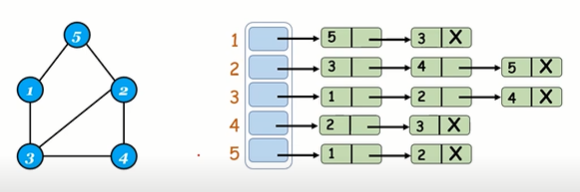
|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **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 **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 graph with arrays before, Now implement graph with adjacent linked 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 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 Edge;  class Vertex { public:  int data;  Vertex \*nextVertex;  Edge \*firstEdge; } \*start = nullptr;  class Edge { public:  Vertex \*destinationVertex;  Edge \*nextEdge;   void insertVertex(int key) {  Vertex \*temp;  Vertex \*ptr;  temp = new Vertex();  temp->data = key;  temp->nextVertex = nullptr;  temp->firstEdge = nullptr;  if (start == nullptr) {  start = temp;  return;  }  ptr = start;  while (ptr->nextVertex != nullptr){  ptr = ptr->nextVertex;  }  ptr->nextVertex = temp;  }   void deleteVertex(int key) {  Vertex \*temp;  Vertex \*ptr1;  Edge \*ptr2;  Edge \*temp1;  if (start == nullptr) {  cout<<"\nNO VERTEX PRESENT\n";  return;  }  if (start->data == key)  {  temp = start;  start = start->nextVertex;  } else  {  ptr1 = start;  while (ptr1->nextVertex != nullptr) {  if (ptr1->nextVertex->data == key)  break;  ptr1 = ptr1->nextVertex;  }  if (ptr1->nextVertex == nullptr) {  cout<<"VERTEX IS NOT PRESENT.\n";  return;  } else {  temp = ptr1->nextVertex;  ptr1->nextVertex = temp->nextVertex;  }  }  ptr2 = temp->firstEdge;  while (ptr2 != nullptr) {  temp1 = ptr2;  ptr2 = ptr2->nextEdge;  free(temp1);  }  free(temp);  }  void deleteIncomingEdges(int key) {  Vertex \*ptr;  Edge \*ptr1;  Edge \*temp;  ptr = start;  while (ptr != nullptr) {  if (ptr->firstEdge == nullptr) /\*Edge list for vertex ptr is empty\*/  {  ptr = ptr->nextVertex;  continue; /\* continue searching in other Edge lists \*/  }   if (ptr->firstEdge->destinationVertex->data == key) {  temp = ptr->firstEdge;  ptr->firstEdge = ptr->firstEdge->nextEdge;  free(temp);  continue; /\* continue searching in other Edge lists \*/  }  ptr1 = ptr->firstEdge;  while (ptr1->nextEdge != nullptr) {  if (ptr1->nextEdge->destinationVertex->data == key) {  temp = ptr1->nextEdge;  ptr1->nextEdge = temp->nextEdge;  free(temp);  continue;  }  ptr1 = ptr1->nextEdge;  }  ptr = ptr->nextVertex;  }   }    Vertex \*findVertex(int u) {  Vertex \*ptr;  Vertex \*location;  ptr = start;  while (ptr != nullptr) {  if (ptr->data == u) {  location = ptr;  return location;  } else  ptr = ptr->nextVertex;  }  location = nullptr;  return location;  }   void insertEdge(int u, int v) {  Vertex \*location;  Vertex \*location1;  Edge \*ptr;  Edge \*temp;   location = findVertex(u);  location1 = findVertex(v);   if (location == nullptr) {  cout<<"\nSTART VERTEX IS ABSENT, YOU HAVE TO ENTER VERTEX FIRST\n"<< u;  return;  }  if (location1 == nullptr) {  cout<<"\nEND VERTEX IS ABSENT, YOU HAVE TO ENTER VERTEX FIRST\n"<< v;  return;  }  temp = new Edge();  temp->destinationVertex = location1;  temp->nextEdge = nullptr;   if (location->firstEdge == nullptr) {  location->firstEdge = temp;  return;  }  ptr = location->firstEdge;  while (ptr->nextEdge != nullptr)  ptr = ptr->nextEdge;  ptr->nextEdge = temp;   }   void deleteEdge(int u, int v) {  Vertex \*location;  Edge \*temp;  Edge \*ptr1;  location = findVertex(u);  if (location == nullptr) {  cout<<"\nSTART VERTEX IS NOT PRESENT.\n";  return;  }  if (location->firstEdge == nullptr) {  cout<<"\nEDGE IS NOT PRESENT.\n";  return;  }  if (location->firstEdge->destinationVertex->data == v) {  temp = location->firstEdge;  location->firstEdge = location->firstEdge->nextEdge;  free(temp);  return;  }  ptr1 = location->firstEdge;  while (ptr1->nextEdge != nullptr) {  if (ptr1->nextEdge->destinationVertex->data == v) {  temp = ptr1->nextEdge;  ptr1->nextEdge = temp->nextEdge;  free(temp);  return;  }  ptr1 = ptr1->nextEdge;  }  cout<<"\nEDGE IS NOT PRESENT IN GRAPH.\n";  }   void display() {  Vertex \*ptr;  Edge \*ptr1;   ptr = start;  while (ptr != nullptr) {  printf("%d ->", ptr->data);  ptr1 = ptr->firstEdge;  while (ptr1 != nullptr) {  printf(" %d", ptr1->destinationVertex->data);  ptr1 = ptr1->nextEdge;  }  printf("\n");  ptr = ptr->nextVertex;  }  } };  **MAIN.CPP:**  #include<iostream> #include "Functions.h"  using namespace std;  int main() {  int choice, u, origin, destination;  Edge e;  while (1) {  cout << "\n1.INSERT A VERTEX.\n";  cout << "2.INSERT AN EDGE.\n";  cout << "3.DELETE A VERTEX.\n";  cout << "4.DELETE AN EDGE\n";  cout << "5.DISPLAY.\n";  cout << "6.EXIT.\n";  cout << "\nENTER YOUR CHOICE : ";  cin >> choice;  switch (choice) {  case 1:  cout<<"\nENTER THE VERTEX : ";  cin>>u;  e.insertVertex(u);  break;  case 2:  cout<<"\nENTER THE EDGES: ";  cin>>origin;  cin>>destination;  e.insertEdge(origin, destination);  break;  case 3:  cout<<"\nENTER THE VERTEX TO BE DELETED : ";  cin>>u;  /\*This function deletes all edges coming to this vertex\*/  e.deleteIncomingEdges(u);  /\*This function deletes the vertex from the vertex list\*/  e.deleteVertex(u);  break;  case 4:  cout<<"\nENTER THE EDGE TO BE DELETED : ";  cin>>origin;  cin>>destination;  e.deleteEdge(origin, destination);  break;  case 5:  e.display();  break;  case 6:  cout<<"YOU CHOOSE TO EXIT."<<endl;  exit(1);  default:  cout<<"\nINVALID ARGUMENT\n";  break;  } //End of switch  } //End of while }  // Paste your output here    Text  Description automatically generated  Text  Description automatically generated with low confidence  Text  Description automatically generated  Text  Description automatically generated |

#### **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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_