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

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 01/12/2022** |
| **Teaching Assistant: Muhammad Sufyan Ashraf** | **Semester: Fall 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 15A. Finding Different Solutions by Solving Open Ended Question**

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

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

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

## **Objective**

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

## **Equipment and Component**

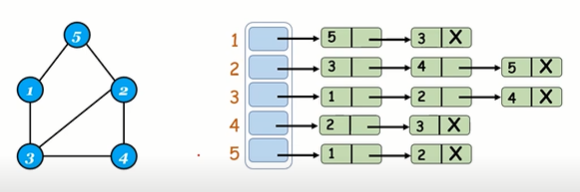
|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **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**

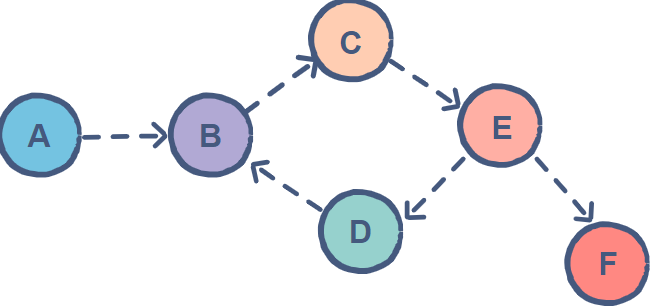
In computer science, A **graph** data structure is a collection of nodes that have data and are connected to other nodes. Let's try to understand this through an example. On Facebook, everything is a node. That includes User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, note...anything 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 a set of vertices (nodes) connected by edges, with each node having a direction associated with it. Edges are usually represented by arrows pointing in the direction the graph can be traversed. In the example on the right, the graph can be traversed from vertex A to B, but not from vertex B to A.

Icon

Description automatically generated



An **array** is a collection of similar types of data. For example, if we want to store the names of 100 people then we can create an array of the string type that can store 100 names. String[] array = new String[100]; Here, the above array cannot store more than 100 names.

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.

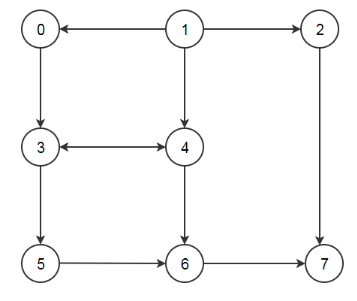
**Templates** are a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one.

**Lab Task**

**Task A**

Given a directed graph and two vertices (say source and destination vertex), determine if the destination vertex is reachable from the source vertex or not. If a path exists from the source vertex to the destination vertex, print it.

For example, there exist two paths [0—3—4—6—7] and [0—3—5—6—7] from vertex 0 to vertex 7 in the following graph. In contrast, there is no path from vertex 7 to any other vertex.



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

|  |
| --- |
| // Paste your code here  **FUNCTION.H:**  // // Created by Lenovo on 12/1/2022. //  #ifndef INC\_2022\_FALL\_CE\_DSA\_WEEK15\_LABTASK\_A\_BSCE21012\_FUNCTIONS\_H #define INC\_2022\_FALL\_CE\_DSA\_WEEK15\_LABTASK\_A\_BSCE21012\_FUNCTIONS\_H  #endif //INC\_2022\_FALL\_CE\_DSA\_WEEK15\_LABTASK\_A\_BSCE21012\_FUNCTIONS\_H  #include <iostream> #include <iomanip>  using namespace std;  class graph { public:  int ver; //declaring  int \*\*mat;  int \*\*solArray;  int rightPath;  int downPath;   explicit graph(int v) {  cout << "ENTER THE NUMBER OF VERTEX = ";  cin >> v;  ver = v; //copying  rightPath = 0;  downPath = 0;  mat = new int \*[ver]; //allocating memory to rows  for (int i = 0; i < ver; i++) {  mat[i] = new int[ver]; //allocating memory to each column  for (int j = 0; j < ver; j++) {  mat[i][j] = 0; //at first storing zeros  }  }  solArray = new int \*[ver]; //allocating memory to rows  for (int i = 0; i < ver; i++) {  solArray[i] = new int[ver]; //allocating memory to each column  for (int j = 0; j < ver; j++) {  solArray[i][j] = 0; //at first storing zeros  }  }  }   void displayMat() const {  cout << "\n\n" << setw(4) << " ";  for (int i = 0; i < ver; i++) {  cout << setw(3) << "( " << i << " )"; //displaying the nodes 0,1,2,3 etc.  }  cout << "\n\n";  for (int i = 0; i < ver; i++) {  cout << setw(3) << "( " << i << " )" << " "; //displaying the nodes  for (int j = 0; j < ver; j++) {  cout << setw(4) << mat[i][j]  << " "; //displaying that weather the nodes is present at that point or not by zero or 1  }  cout << "\n\n";  }  }   void displaySolutionMat() const {  cout << "\n\n" << setw(4) << " ";  for (int i = 0; i < ver; i++) {  cout << setw(3) << "( " << i << " )"; //displaying the nodes 0,1,2,3 etc.  }  cout << "\n\n";  for (int i = 0; i < ver; i++) {  cout << setw(3) << "( " << i << " )" << " "; //displaying the nodes  for (int j = 0; j < ver; j++) {  cout << setw(4) << solArray[i][j]  << " "; //displaying that weather the nodes is present at that point or not by zero or 1  }  cout << "\n\n";  }  }   void addEdges() const {  for (int i = 0; i < ver; i++) {  for (int j = 0; j < ver; j++) { //loops for adding an edge  if (i != j) { //if i and j are not equal.  cout << "ENTER 1 IF THE VERTEX " << i << " IS ADJACENT TO " << j << ",OTHERWISE ENTER 0";  cin  >> mat[i][j]; //entering zero and 1,1 if the nodes are present and zero if not  if (mat[i][j] != 1) {  if (mat[i][j] !=  0) { //checking if the user has enter a number other than 1 and zero  cout << "INVALID ARGUMENT." << endl;  exit(3);  }  }  }  if (i == j) {  cout << "ENTER 1 IF THE VERTEX " << i << " IS ADJACENT TO " << j << ",OTHERWISE ENTER 0";  cin >> mat[i][j];  if (mat[i][j] != 1) {  if (mat[i][j] !=  0) { //checking if the user has enter a number other than 1 and zero  cout << "INVALID ARGUMENT." << endl;  exit(3);  }  }  }  }  }  }   bool safePath(int sourceVertex1, int sourceVertex2, int destinationVertex1, int destinationVertex2) {  rightPath = sourceVertex1;  downPath = sourceVertex2;  if ((rightPath >= 0 && rightPath <= destinationVertex1) && (downPath >= 0 && downPath <= destinationVertex2) &&  mat[rightPath][downPath] == 1 && //checking that the points added are in the matrix and the matrix is 1 or not  solArray[rightPath][downPath] != 1) {  return true;  } else {  return false;  }  }   bool ratInMaze(int sourceVertex1, int sourceVertex2, int destinationVertex1, int destinationVertex2) {  rightPath = sourceVertex1;  downPath = sourceVertex2;  if (rightPath == destinationVertex1 && downPath == destinationVertex2 && mat[rightPath][downPath] == 1) {  solArray[rightPath][downPath] = 1; //the base condition  return true;  }  if (safePath(sourceVertex1, sourceVertex2, destinationVertex1, destinationVertex2)) {  solArray[rightPath][downPath] = 1; //checking if it is safe if it is then putting 1 at sol matrix   if (ratInMaze(sourceVertex1 - 1, sourceVertex2, destinationVertex1, destinationVertex2)) {  return true; //checking left side  }  if (ratInMaze(sourceVertex1 + 1, sourceVertex2, destinationVertex1, destinationVertex2)) {  return true; //checking right side  }  if (ratInMaze(sourceVertex1, sourceVertex2 + 1, destinationVertex1, destinationVertex2)) {  return true; //checking downward  }  if (ratInMaze(sourceVertex1, sourceVertex2 - 1, destinationVertex1, destinationVertex2)) {  return true; //checking upward  }  solArray[rightPath][downPath] = 0; //if none of them is true then put zero on sol mat and return false  return false;  }  return false;  }  bool findSolution() {  if (!ratInMaze(0, 0, 3, 2)) {  cout << "NO PATH FOUND." << endl; //checking if path exist  return false;  } else {  cout << "THE SOLUTION MATRIX IS . ";  displaySolutionMat(); //if it does then display  cout << "THE PATH TAKEN = ";  for (int i = 0; i < ver; i++) {  for (int j = 0; j < ver; j++) {  if (solArray[i][j] == 1) {  cout << "(" << i << ")" << "(" << j << ")" << "->"; //the path  }  }  }  cout << "REACHED." << endl;  }   return true;  }  **MAIN.CPP:**  // #include <iostream> #include "Functions.h"  using namespace std;  int main() {  graph g(4);  g.addEdges();  cout << "THE MATRIX IS . ";  g.displayMat(); //calling  g.findSolution(); }  // 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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_