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

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 03/11/2022** |
| **Teaching Assistant: Muhammad Sufyan Ashraf** | **Semester: Fall 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 10A. Breadth First Traversal and Depth First Traversal in Trees**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 the 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 **binary tree** is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

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 **binary** tree in which all nodes except leaf nodes have two children. A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children. It is also known as a proper binary tree. Traversing a tree means going through every node in it. The two most common ways to traverse a tree are **breadth-first** and **depth-first**. Breadth-first traversal means that we start from the top node, then go one level down, go through all of the children’s nodes from left to right.

**Heap** **sort** is a comparison-based sorting technique based on Binary Heap data structure. It is like the selection sort where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements. Heap sort is an in-place algorithm.

**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**

In this lab we want to implement heap sort through arrays. First thing to be done is that you create a function which converts an array into Max Heap and second thing is to do in place sorting in the array through it.

|  |
| --- |
| // Paste your code here  **FUNCTION.H:**  #include <iostream>  using namespace std;   class sort { public:  int \*arr;  int noOfElements; //we have declared the attributes of the class  int size;   sort(int s) {  size = s; //a constructor to det the size, no. of elements in the array and to allocate the memory  noOfElements = s;  arr = new int[s];  }   void display() {  for (int i = 0; i < size; i++) //a function to display the array  cout << arr[i] << " ";  cout << endl;  }   int parentNode(int i) { //a parent node which only get the parent  return (i - 1) / 2;  }   int leftNode(int i) {  return (2 \* i + 1); //which gives the left child  }   int rightNode(int i) {  return (2 \* i + 2); //which gives the right child  }   void minHeap(int i) {  int temp;  int l = leftNode(i); //storing the get left node in the l  int r = rightNode(i); //storing the right node in the r  int smallest = i; //making the i the smallest element  if (l < size && arr[l] < arr[i])  smallest = l; //comparing if the left child is less than size and the child value at l index is less than i then making smallest the l  if (r < size && arr[r] < arr[smallest]) //comparing if the right child is less than size and the child value at r index is less than smallest then making smallest the r  smallest = r;  if (smallest != i) { //if smallest is not w=equal to i  temp=arr[i];  arr[i]=arr[smallest]; //swapping  arr[smallest]=temp;  minHeap(smallest); //calling the min heap function  }  }   int extractMin() {  if (size <= 0) { //if the size is less than zero  cout << "SIZE IS NEGATIVE";  }  if (size == 1) { //if size is equal to 1  size--; //decrement the size and return the zero index  return arr[0];  }  int root = arr[0]; //making root as zero index  arr[0] = arr[size - 1]; // storing the second last element at zero index  size--; //decrementing the size  minHeap(0); //and calling  return root;  }   void addElements() {  cout << "ENTER " << noOfElements << " NO. OF ELEMENTS TO SORT USING HEAPSORT" << endl;  for (int i = 0; i < noOfElements; i++)  cin >> arr[i]; //taking elemnt from the user  }   void heapSort() {  int temp[noOfElements];  for (int j = 0; j < noOfElements; j++) { //making a loop till no of elements  temp[j] = extractMin(); //storing thr elements extracted from the extract min function  cout << temp[j] << " "; //displaying  }  }   void maxHeap(int arr[], int n, int i) {  int temp; //declaring  int largest, left, right;  largest = i;  left = (2 \* i) + 1; //initializing  right = (2 \* i) + 2;  if (left < n && arr[left] > arr[largest]) { //checking and comparing that if the array element at left index is greater than largest then store left in largest  largest = left;  }  if (right < n && arr[right] > arr[largest]) {  largest = right; //checking and comparing that if the array element at right index is greater than largest then store left in largest  }  if (largest != i) { //if largest is not equal to i  temp = arr[i];  arr[i] = arr[largest]; //swapping  arr[largest] = temp;  maxHeap(arr, n, largest); //calling the same function  }  }   void heapSortMax(int arr[], int n) {  int temp;  for (int i = n / 2 - 1; i >= 0; i--) {  maxHeap(arr, n, i); //calling  }  for (int i = n - 1; i >= 0; i--) {  temp = arr[0];  arr[0] = arr[i]; //swapping arr of zero and arr of i  arr[i] = temp;  maxHeap(arr, i, 0); //calling  }  } };  **MAIN.CPP:**  #include <iostream> #include "Functions.h"  using namespace std;  int main() {  int opt;  int size;  cout << "ENTER SIZE" << endl;  cin >> size;  sort obj(size);  do {  cout << "\nENTER OPTION." << endl;  cout << "1.BY USING MIN HEAP METHOD." << endl; //displaying  cout << "2.BY USING MAX HEAP METHOD." << endl;  cout << "3.EXIT." << endl;  cin >> opt;  if (opt == 1) {  obj.addElements();  cout << "UNSORTED ARRAY:" << endl; //displaying before swapping  obj.display();  for (int i = size / 2 - 1; i >= 0; i--) {  obj.minHeap(i);  }  cout << "SORTED ARRAY :" << endl;  obj.heapSort(); //calling  cout << endl;  }  if (opt == 2) {  int arr[size];  cout << "ENTER ELEMENTS = ";  for (int i = 0; i < size; i++) { //taking input  cin >> arr[i];  }  cout << "UNSORTED ARRAY :" << endl;  for (int i = 0; i < size; i++) { //displaying before swapping  cout << arr[i] << " ";  }  obj.heapSortMax(arr, size); //calling  cout << endl;  cout << "SORTED ARRAY :" << endl;  for (int i = 0; i < size; i++) { //displaying before swapping  cout << arr[i] << " ";  }  }  if (opt == 3) {  cout << "YOU CHOOSE TO EXIT." << endl;  exit(4);  }  if(opt!= 1 && opt!=2 && opt!=3){  cout<<"YOU HAVE ENTERED AN INVALID ARGUMENT."<<endl;  break;  }  } while (opt>=1 && opt<=3); }  // Paste your output here  **OUTPUT:** |

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