**Logo, company name

Description automatically generated**

**Department of (Computer Science)**

**Pak-Austria Fachhochschule: Institute of Applied Sciences and Technology, Haripur, Pakistan**

**COMP-112L Data Structure** **& Algorithm Lab**

**Lab Journal**

**Class: BS Computer Science**

**Name: Ahmed Raza**

**Registration No.: B20F0436CS031**

**Semester: 4th**

**Submission Date: 26th April 2022**

**Submitted to: Engr. Rafi-Ullah**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Instructor Signature**

**Lab No. 12**

**Implementing Graphs and Their Transversal**

**Objectives:**

In this lab we will be discussing about basic concepts and implementation of Graph in C++.in detail. This is one of the most important concepts in Data Structure C++ language.

**Tools/Software Required:**

* All the tasks are implemented on DEV C++.

**Introduction:**

**GRAPH:**

The graph is one non-linear data structure. That is consists of some nodes and their connected edges. The edges may be director or undirected.

The graph has two types of traversal algorithms. These are called the Breadth First Search and Depth First Search.

**Breadth First Search (BFS):**

The Breadth First Search (BFS) traversal is an algorithm, which is used to visit all the nodes of a given graph. In this traversal algorithm one node is selected and then all the adjacent nodes are visited one by one. After completing all the adjacent vertices, it moves further to check another vertex and checks its adjacent vertices again.

**Depth First Search (DFS):**

The Depth First Search (DFS) is a graph traversal algorithm. In this algorithm one starting vertex is given, and when an adjacent vertex is found, it moves to that adjacent vertex first and try to traverse in the same manner.

**Lab Tasks:**

**Task 1:**

Write a program for the Breadth First Search.

**Code:**

**#include<iostream>**

**using namespace std;**

**class GraphNode;**

**class List;**

**void displayList(List\*);**

**struct ListNode{**

**GraphNode\* storedNode;**

**ListNode\* next;**

**};**

**class List**

**{**

**public:**

**ListNode\* head;**

**List()**

**{**

**head = NULL;**

**}**

**void insert(GraphNode\* newGraphNode)**

**{**

**ListNode\* newListNode = new ListNode;**

**newListNode->storedNode = newGraphNode;**

**newListNode->next = head;**

**head = newListNode;**

**}**

**GraphNode\* getNode(int i)**

**{**

**if(!head)**

**return NULL;**

**int currIndex = 0;**

**ListNode\* currNode = head;**

**while(currNode && currIndex<i)**

**{**

**currNode = currNode->next;**

**currIndex++;**

**}**

**if(!currNode)**

**return NULL;**

**return currNode->storedNode;**

**}**

**void display(){**

**displayList(this);**

**}**

**int size(){**

**if(!head)**

**return 0;**

**int s;**

**ListNode\* currNode = head;**

**while(currNode!=NULL){**

**s++;**

**currNode = currNode->next;**

**}**

**return s;**

**}**

**};**

**class Queue{**

**public:**

**ListNode\* front;**

**ListNode\* rear;**

**Queue(){**

**front = rear = NULL;**

**}**

**ListNode\* enqueue(GraphNode\* newGraphNode){**

**ListNode\* newListNode = new ListNode;**

**newListNode->storedNode = newGraphNode;**

**newListNode->next = NULL;**

**if (front == NULL && rear == NULL){**

**front = rear = newListNode;**

**} else {**

**rear->next = newListNode;**

**rear = newListNode;**

**}**

**return newListNode;**

**}**

**GraphNode\* dequeue(){**

**if(isEmpty())**

**return NULL;**

**GraphNode\* dequeueNode = front->storedNode;**

**if (front == rear)**

**rear = NULL;**

**front = front->next;**

**//delete front;**

**return dequeueNode;**

**}**

**bool isEmpty(){**

**return (front == NULL);**

**}**

**};**

**class GraphNode{**

**public:**

**char value;**

**int color; //0=white/unvisisted 1=grey/discovered 2= black/explored**

**GraphNode\* parent;**

**int distance;**

**List\* adjacentNodes;**

**GraphNode(){**

**value = '-';**

**color = 0; //white, unvisited**

**parent = NULL;**

**distance = 0;**

**adjacentNodes = new List;**

**}**

**GraphNode(char v){**

**value = v;**

**color = 0; //white, unvisited**

**parent = NULL;**

**distance = 0;**

**adjacentNodes = new List;**

**}**

**};**

**void displayList(List\* list){**

**if(!list->head)**

**return;**

**ListNode\* currNode = list->head;**

**while(currNode!=NULL){**

**cout<<currNode->storedNode->value<<" ";**

**currNode = currNode->next;**

**}**

**cout<<endl;**

**}**

**int main(){**

**List listOfNodes;**

**GraphNode\* node1 = new GraphNode('Z');**

**GraphNode\* node2 = new GraphNode('Y');**

**GraphNode\* node3 = new GraphNode('X');**

**GraphNode\* node4 = new GraphNode('W');**

**GraphNode\* node5 = new GraphNode('V');**

**GraphNode\* node6 = new GraphNode('U');**

**node1->adjacentNodes->insert(node2); node1->adjacentNodes->insert(node3); node1->adjacentNodes->insert(node4);**

**node2->adjacentNodes->insert(node1); node2->adjacentNodes->insert(node5);**

**node3->adjacentNodes->insert(node1); node3->adjacentNodes->insert(node6);**

**node4->adjacentNodes->insert(node1); node4->adjacentNodes->insert(node6);**

**node5->adjacentNodes->insert(node2);**

**node6->adjacentNodes->insert(node3); node6->adjacentNodes->insert(node4);**

**listOfNodes.insert(node1);**

**listOfNodes.insert(node2);**

**listOfNodes.insert(node3);**

**listOfNodes.insert(node4);**

**listOfNodes.insert(node5);**

**listOfNodes.insert(node6);**

**//SEARCH STARTS HERE**

**//initialize vertices/nodes**

**GraphNode\* temp;**

**for (int i=0; i<5; i++){**

**temp = listOfNodes.getNode(i);**

**temp->color = 0;**

**temp->distance = 9999;**

**temp->parent = NULL;**

**}**

**temp=listOfNodes.getNode(5);**

**temp->color = 1;**

**temp->distance = 0;**

**temp->parent = NULL;**

**int size;**

**GraphNode \*u, \*v;**

**Queue Q;**

**Q.enqueue(node1);**

**while(!Q.isEmpty()){**

**u = Q.dequeue();**

**cout<<u->value<<" ";**

**size = u->adjacentNodes->size();**

**for (int i=0; i<size; i++){**

**v = u->adjacentNodes->getNode(i);**

**if (v->color == 0){**

**v->color = 1;**

**v->distance = u->distance + 1;**

**v->parent = u;**

**Q.enqueue(v);**

**}**

**}**

**u->color = 2;**

**}**

**}**

**Output:**

**Text

Description automatically generated**

**Task 2:**

Write a program for the Depth First Search.

**Code:**

**#include<iostream>**

**using namespace std;**

**class GraphNode;**

**class List;**

**void displayList(List\*);**

**struct ListNode{**

**GraphNode\* storedNode;**

**ListNode\* next;**

**};**

**class List{**

**public:**

**ListNode\* head;**

**List(){**

**head = NULL;**

**}**

**void insert(GraphNode\* newGraphNode){**

**ListNode\* newListNode = new ListNode;**

**newListNode->storedNode = newGraphNode;**

**newListNode->next = head;**

**head = newListNode;**

**}**

**GraphNode\* getNode(int i){**

**if(!head)**

**return NULL;**

**int currIndex = 0;**

**ListNode\* currNode = head;**

**while(currNode && currIndex<i){**

**currNode = currNode->next;**

**currIndex++;**

**}**

**if(!currNode)**

**return NULL;**

**return currNode->storedNode;**

**}**

**void display(){**

**displayList(this);**

**}**

**int size(){**

**if(!head)**

**return 0;**

**int s;**

**ListNode\* currNode = head;**

**while(currNode!=NULL){**

**s++;**

**currNode = currNode->next;**

**}**

**return s;**

**}**

**};**

**class Stack{**

**public:**

**ListNode\* top;**

**Stack(){**

**top = NULL;**

**}**

**void push(GraphNode\* newGraphNode){**

**ListNode\* newListNode = new ListNode;**

**newListNode->storedNode = newGraphNode;**

**newListNode->next = top;**

**top = newListNode;**

**}**

**GraphNode\* pop(){**

**if(isEmpty())**

**return NULL;**

**GraphNode\* dequeueNode = top->storedNode;**

**top = top->next;**

**return dequeueNode;**

**}**

**GraphNode\* peek(){**

**return top->storedNode;**

**}**

**bool isEmpty(){**

**return (top == NULL);**

**}**

**};**

**class GraphNode{**

**public:**

**char value;**

**int color; //0=white/unvisisted 1=grey/discovered 2= black/explored**

**List\* adjacentNodes;**

**GraphNode(){**

**value = '-';**

**color = 0; //white, unvisited**

**adjacentNodes = new List;**

**}**

**GraphNode(char v){**

**value = v;**

**color = 0; //white, unvisited**

**adjacentNodes = new List;**

**}**

**};**

**void displayList(List\* list){**

**if(!list->head)**

**return;**

**ListNode\* currNode = list->head;**

**while(currNode!=NULL){**

**cout<<currNode->storedNode->value<<" ";**

**currNode = currNode->next;**

**}**

**cout<<endl;**

**}**

**void depthFirstSearch(GraphNode\* source){**

**Stack S;**

**S.push(source);**

**while(!S.isEmpty()){**

**GraphNode\* u = S.pop();**

**if(u->color == 0){**

**cout<<u->value<<" ";**

**u->color = 1;**

**}**

**for (int i=0; i<u->adjacentNodes->size(); i++){**

**if(u->adjacentNodes->getNode(i)->color == 0)**

**S.push(u->adjacentNodes->getNode(i));**

**cout<<"";**

**}**

**}**

**}**

**int main(){**

**int numberOfNodes = 6;**

**List listOfNodes;**

**GraphNode\* node1 = new GraphNode('Z');**

**GraphNode\* node2 = new GraphNode('Y');**

**GraphNode\* node3 = new GraphNode('X');**

**GraphNode\* node4 = new GraphNode('W');**

**GraphNode\* node5 = new GraphNode('V');**

**GraphNode\* node6 = new GraphNode('U');**

**node1->adjacentNodes->insert(node2); node1->adjacentNodes->insert(node3); node1->adjacentNodes->insert(node6);**

**node2->adjacentNodes->insert(node1);**

**node3->adjacentNodes->insert(node1); node3->adjacentNodes->insert(node4); node3->adjacentNodes->insert(node5);**

**node4->adjacentNodes->insert(node3); node4->adjacentNodes->insert(node5);**

**node5->adjacentNodes->insert(node3); node5->adjacentNodes->insert(node4); node5->adjacentNodes->insert(node6);**

**node6->adjacentNodes->insert(node1); node6->adjacentNodes->insert(node5);**

**listOfNodes.insert(node1);**

**listOfNodes.insert(node2);**

**listOfNodes.insert(node3);**

**listOfNodes.insert(node4);**

**listOfNodes.insert(node5);**

**listOfNodes.insert(node6);**

**GraphNode\* temp;**

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

**listOfNodes.getNode(i)->color == 0;**

**cout<<"Source Node : "<<node1->value<<endl;**

**cout<<"Depth First Search Output : ";**

**for (int i=numberOfNodes-1; i>=0; i--) //A is on last index, so search starts from A**

**if (listOfNodes.getNode(i)->color==0)**

**depthFirstSearch(listOfNodes.getNode(i));**

**}**

**Output:**

**Text

Description automatically generated**

**Results & Observations:**

In this Lab graph data structure was introduced to us. A graph is made up of a finite number of vertices or nodes, as well as edges. The real data that we want to store is stored in vertices. Between these vertices, edges resemble each other. Each vertex has a table that stores the addresses of nodes nearby. Because of the non-linear character of the graph, it is perfect for real-life situations such as networking. The fact that graphs do not have a fixed starting point distinguishes them from other data structures. We have the option of selecting whatever source node we choose. A head exists in a linked list, and a root node exists in a binary tree, among other things. This results in various changes in graph output, but all those variants are valid.