TASK 1:

#include<iostream>

#include<string>

#include<queue>

#include<stack>

using namespace std;

int map[15][15] = { {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},{0,1,0,0,0,0,0,0,1,1,0,0,0,0,0},{1,1,0,0,1,1,0,0,1,1,0,0,0,1,1},{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},{0,0,0,0,1,1,1,1,1,1,0,0,0,1,1},{0,1,1,0,0,0,0,0,0,0,0,0,0,1,1},{0,1,1,0,0,0,1,1,0,0,1,1,0,0,0},{0,1,1,0,1,1,1,1,1,0,1,1,0,0,0},{0,1,1,0,1,1,1,1,1,0,1,1,0,0,0},{0,1,0,0,1,1,1,1,1,0,1,1,0,0,1},{0,0,0,0,0,0,0,0,0,0,0,0,0,1},{0,0,0,1,1,0,0,0,0,0,0,0,0,0,1},{0,0,0,1,1,0,0,0,0,0,1,1,0,0,1},{0,0,0,0,0,0,0,1,0,0,1,1,0,0,1},{0,0,0,0,0,0,0,1,0,0,1,1,0,0,1} };

int visited[15]; /// places went

int cost = 0;

class graph{

int\*\* arr;

public:

graph(){

int n = 15;

arr = new int\* [n + 1];

for (int i = 0; i <= n; i++){

arr[i] = new int[n];

}

for (int i = 0; i <= n; i++){

for (int j = 1; j <= n; j++){

arr[i][j] = map[i][j]; // maping the whole grid

}

}

}

void displayGrid(){

int n = 15;

cout << endl;

for (int i = 0; i < n; i++){

for (int j = 0; j <= n; j++){

cout << " " << arr[i][j] << " ";

}

cout << endl;

}

cout << endl;

}

void BSF(int sx, int sy, int destx, int desty, int nodes) {

bool visited[nodes]; //array to check if a node has been visited or not

bool visitedy[nodes];

queue<int> obj;

for (int i = 0; i < nodes; i++){

visited[i] = false;

visitedy[i] = false;

}

obj.push(sx); //enqued root (source) node

obj.push(sy);

visited[sx] = true;

visited[sy] = true;

while (!obj.empty()){

int x = obj.front();

obj.pop();

int y = obj.front();

obj.pop();

arr[x][y] = 7;

if (x == destx && y == desty) { //if desination has been reached

return;

}

for (int i = 0; i < nodes; i++) {

if (arr[x][i] == 0 && visited[i] != true){ //enque nodes adjacent to x in queue

visited[i] = true;

obj.push(x);

obj.push(i);

}

else if (arr[i][y] == 0 && visitedy[i] != true){

visitedy[i] = true;

obj.push(i);

obj.push(y);

}

}

}

}

void DFS(int sx, int sy, int destx, int desty, int nodes){

bool visited[nodes];

bool visitedy[nodes];

for (int i = 0; i < nodes; i++) {

visited[i] = false;

visitedy[i] = false;

}

stack<int> obj;

obj.push(sx); //push root node in stack

obj.push(sy);

visited[sx] = true;

visited[sy] = true;

int x;

int y;

while (!obj.empty()) {

y = obj.top();

obj.pop();

x = obj.top();

obj.pop();

arr[x][y] = 7;

if (x == destx && y == desty) {

return;

}

if (visited[x] == false) {

visited[x] = true; //every popped element will be marked as visited, popped node's adjacent nodes will be checked next

}

for (int i = 0; i < nodes; i++) {

if (visited[i] == false && arr[x][i] == 1) { //push adjacent nodes to x into stack

//the adjacent node pushed last will be poppedin next iteration and then it's adjacent nodes will be checked and so on...

obj.push(x);

obj.push(i);

}

else if (visitedy[i] == false && arr[i][x] == 1){

obj.push(i);

obj.push(x);

}

}

}

}

};

int main(){

graph g;

g.displayGrid();

g.DFS(1, 15, 15, 1, 15);

cout << "\nDFS:\n";

g.displayGrid();

cout << "\nNormal :\n";

graph hp;

hp.displayGrid();

cout << "\nBFS:\n";

hp.BSF(1, 15, 15, 1, 15);

hp.displayGrid();

}

QUESTION 2:

PART(A):

#include<iostream>

using namespace std;

class node{

public:

char data;

node\* next;

};

class list{

public:

node\* head;

list(){

head = NULL;

}

void insertInEnd(char d){

node\* x;

node\* temp = new node;

temp->data = d;

temp->next = NULL;

if (head == NULL){

head = temp;

return;

}

x = head;

while (x->next != NULL){

x = x->next;

}

x->next = temp;

}

void print(){

node\* n;

n = head;

while (n != NULL){

cout << "->" << n->data;

n = n->next;

}

}

};

int indexno(char x){

switch (x){

case 'A':

case 'a':

return 0;

break;

case 'B':

case 'b':

return 1;

break;

case 'C':

case 'c':

return 2;

break;

case 'D':

case 'd':

return 3;

break;

case 'E':

case 'e':

return 4;

case 'F':

case 'f':

return 5;

break;

case 'G':

case 'g':

return 6;

break;

case 'H':

case 'h':

return 7;

break;

case 'I':

case 'i':

return 8;

break;

case 'J':

case 'j':

return 9;

break;

case 'K':

case 'k':

return 10;

break;

case 'L':

case 'l':

return 11;

break;

case 'M':

case 'm':

return 12;

break;

case 'N':

case 'n':

return 13;

break;

case 'O':

case 'o':

return 14;

break;

case 'P':

case 'p':

return 15;

break;

case 'Q':

case 'q':

return 16;

break;

case 'R':

case 'r':

return 17;

break;

case 'S':

case 's':

return 18;

break;

case 'T':

case 't':

return 19;

break;

case 'U':

case 'u':

return 20;

break;

case 'V':

case 'v':

return 21;

break;

case 'W':

case 'w':

return 22;

break;

case 'X':

case 'x':

return 23;

break;

case 'Y':

case 'y':

return 24;

break;

case 'Z':

case 'z':

return 25;

break;

default:

return 0;

}

}

class graph{

public:

int\*\* arr;

list\* lst;

int size;

graph(int size1){

size = size1;

lst = new list[size];

arr = new int\* [size1];

for (int i = 0; i < size1; i++){

arr[i] = new int[size1];

}

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

for (int j = 0; j < size; j++){

arr[i][j] = 0;

}

}

}

void insert(char x, char y, int val){

int i = indexno(x);

int j = indexno(y);

arr[i][j] = val;

arr[j][i] = val;

lst[i].insertInEnd(y);

lst[j].insertInEnd(x);

}

void displayMat(){

cout << " ";

for (int k = 0; k < size; k++){

char x = k + 97;

cout << x << " ";

}

cout <<endl;

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

char x = i + 97;

cout << x << " ";

for (int j = 0; j < size; j++){

cout << arr[i][j] << " ";

}cout << "\n";

}

}

void displayList(){

cout << "\nAdjacency list \n";

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

char x = i + 97;

cout << x << " ";

lst[i].print();

cout << "\n";

}

}

};

int main(){

graph graphs(6);

graphs.insert('a', 'b', 7);

graphs.insert('a', 'c', 9);

graphs.insert('a', 'f', 14);

graphs.insert('b', 'c', 10);

graphs.insert('b', 'd', 15);

graphs.insert('c', 'd', 11);

graphs.insert('c', 'f', 2);

graphs.insert('d', 'e', 6);

graphs.insert('e', 'f', 9);

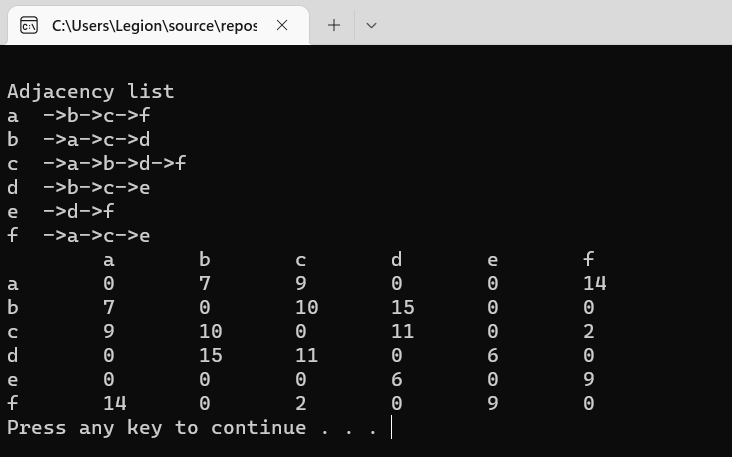
graphs.displayList();

graphs.displayMat();

system("pause");

return 0;

}



Q 2(b)i:

#include <iostream>

using namespace std;

int main() {

int\*\* arr;

int n, m;

cout << "Input number of vertices: ";

cin >> n;

arr = new int\* [n];

for (int i = 0; i < n; i++) {

arr[i] = new int[n];

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

arr[i][j] = 0;

}

}

cout << "Enter number of edges: ";

cin >> m;

int x, y;

for (int k = 0; k < m; k++) {

cout << "Enter 1st vertice: ";

cin >> x;

cout << "Enter 2nd vertice: ";

cin >> y;

arr[x - 1][y - 1] = 1;

arr[y - 1][x - 1] = 1;

}

cout << "Adjacency Matrix: " << endl;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

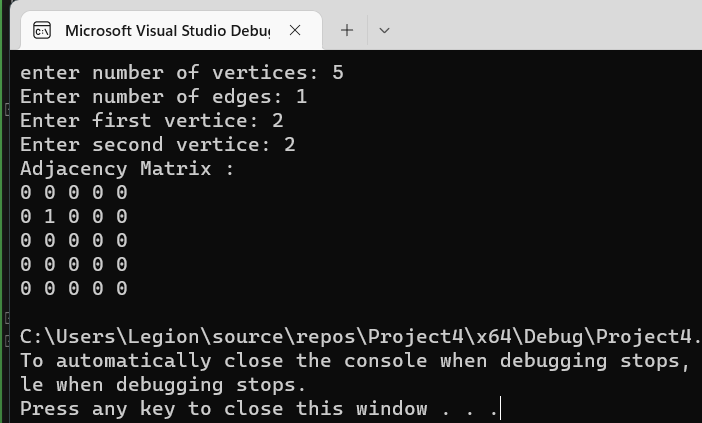
cout << arr[i][j] << " ";

}

cout << endl;

}

}



Q 2(b)ii:

#include <iostream>

using namespace std;

int main() {

int\*\* arr;

int\* count;

int n, m;

cout << "Input number of vertices: ";

cin >> n;

arr = new int\* [n];

count = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = new int[n];

count[i] = 0;

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

arr[i][j] = 0;

}

}

cout << "Enter number of edges: ";

cin >> m;

int x, y;

for (int k = 0; k < m; k++) {

cout << "Enter 1st vertice: ";

cin >> x;

cout << "Enter 2nd vertice: ";

cin >> y;

arr[x][y] = 1;

arr[y][x] = 1;

}

cout << "Adjacency Matrix: " << endl;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (arr[i][j] == 1) {

count[i]++;

}

cout << arr[i][j] << " ";

}

cout << endl;

}

int\*\* list;

list = new int\* [n];

for (int i = 0; i < n; i++) {

list[i] = new int[count[i]];

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < count[i]; j++) {

for (int k = 0; k < n; k++) {

if (arr[i][k] == 1) {

list[i][j] = k;

j++;

}

}

}

}

cout << "adjacency list: " << endl;

for (int i = 0; i < n; i++) {

cout << "for " << i << " : ";

for (int j = 0; j < count[i]; j++) {

cout << list[i][j];

}

cout << endl;

}

system("pause");

return 0;

}

Text

Description automatically generated

QUESTION 3:

#include<iostream>

using namespace std;

struct node{

char data;

node\* next;

};

class Queue{

public:

Queue(){

queueHead = nullptr;

}

bool isEmpty(){

return (queueHead == nullptr) ? true : false;

}

int front() { return queueHead->data; }

void Enqueue(int value){

node\* newNode = new node;

newNode->data = value;

if (queueHead == nullptr){

queueHead = newNode;

newNode->next = nullptr;

}

else{

node\* ptr = queueHead;

while (ptr->next != nullptr){

ptr = ptr->next;

}

ptr->next = newNode;

newNode->next = nullptr;

}

}

void Dequeue(){

if (isEmpty()) {

return;

}

else{

if (queueHead->next == nullptr){

delete queueHead;

queueHead = nullptr;

}

else{

node\* ptr = queueHead;

queueHead = queueHead->next;

ptr->next = nullptr;

delete ptr;

}

}

}

~Queue(){

delete[] queueHead;

}

private:

node\* queueHead;

};

struct Vertex{

char value;

Vertex\* Next;

};

class BFSgraph{

private:

Vertex\*\* list;

int nodes, edges;

bool\* visited;

public:

BFSgraph(const int n = 26){

nodes = n;

list = new Vertex \* [nodes];

for (int i = 0; i < nodes; i++)

list[i] = nullptr;

edges = nodes;

visited = new bool[n];

for (int i = 0; i < n; i++)

visited[i] = false;

}

void InsertEdge(char scr, char dest){

if (int(scr) >= 65 && int(scr) <= 90) scr -= 65;

else if (int(scr) >= 97 && int(scr) <= 122) scr -= 97;

else

cout << "Please Enter correct Source" << endl;

if (int(dest) >= 65 && int(dest) <= 90) dest -= 65;

else if (int(dest) >= 97 && int(dest) <= 122) dest -= 97;

else

cout << "Please Enter correct Destination" << endl;

Vertex\* newEle = new Vertex();

newEle->value = dest;

newEle->Next = nullptr;

if (list[scr] == nullptr){

list[scr] = newEle;

}

else{

Vertex\* root = list[scr];

while (root->Next != NULL){

if (root->value == dest){

cout << "Edge b/w Source & Destination Already Exist" << endl;

return;

}

root = root->Next;

}

root->Next = newEle;

cout << "Edge b/w Source & Destination is Created" << endl;

}

}

void BFS(int scr, int end){

if (int(scr) >= 65 && int(scr) <= 90) scr -= 65;

else if (int(scr) >= 97 && int(scr) <= 122) scr -= 97;

else

cout << "Please Enter correct Source" << endl;

cout << "BFS Treversal : ";

Queue queue;

visited[scr] = true;

queue.Enqueue(scr);

int s = 0, i = 0;

Vertex\* node;

while (!queue.isEmpty()){

s = queue.front();

cout << char(s + 65) << " ";

queue.Dequeue();

if (s + 65 == end){

cout << endl;

return;

}

node = list[s];

while (node != NULL){

if (!visited[node->value]){

visited[node->value] = true;

queue.Enqueue(node->value);

}

node = node->Next;

}

}

cout << endl;

}

void Display(){

cout << "Adjacency List : " << endl;

for (int i = 0; i < nodes; i++){

cout << char(i + 65);

Vertex\* root = list[i];

if (root == nullptr)

cout << " ";

while (root != nullptr){

if (root != nullptr)cout << "-->";

cout << char(root->value + 65);

root = root->Next;

}

cout << endl;

}

}

};

int main(){

BFSgraph g;

int choice = 0;

char scr = 0, dest = 0;

menu:

system("cls");

cout << "\n\t1. Insert Edge";

cout << "\n\t2. Display";

cout << "\n\t3. Exit";

cout << "\n\tEnter Choice => ";

try{

cin >> choice;

switch (choice){

case 1:

cout << "Enter Source : ";

cin >> scr;

if (!cin) { throw"Please Enter Correct Input"; }

cout << "Enter Destination : ";

cin >> dest;

if (!cin) { throw"Please Enter Correct Input"; }

g.InsertEdge(scr, dest);

break;

case 2:

g.Display();

g.BFS('A', 'B');

break;

case 3:

return 0;

break;

default:

if (choice < 1 || choice > 3){

throw"Please Enter Correct Choice";

}

break;

}

system("pause");

goto menu;

}

catch (const char\* error\_msg){

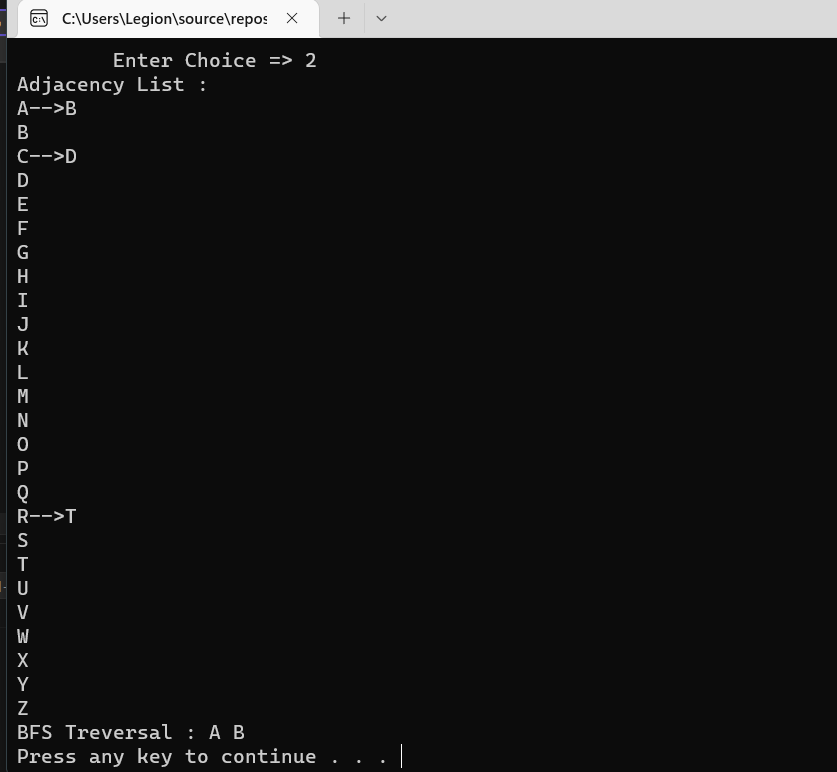
cout << "\n\t" << error\_msg << endl;

cin.clear(); cin.ignore(); system("pause");

goto menu;

}

}



QUESTION 4:

#include <iostream>

using namespace std;

class heap {

private:

int\* arr;

int size, count;

void mini\_heapify\_down(int root) {

int smallest = root;

int left = (root \* 2) + 1;

int right = (root \* 2) + 2;

if (left < count && arr[left] < arr[smallest]) {

smallest = left;

}

if (right < count && arr[right] < arr[smallest]) {

smallest = right;

}

if (smallest != root) {

swap(arr[smallest], arr[root]);

mini\_heapify\_down(smallest);

}

}

void mini\_heapify\_up(int root) {

int smallest = root;

int parent = (root - 1) / 2;

if (arr[parent] > arr[smallest]) {

smallest = parent;

}

if (smallest != root) {

swap(arr[root], arr[smallest]);

mini\_heapify\_up(smallest);

}

}

public:

heap(int s) {

size = s;

arr = new int[size];

count = 0;

}

bool isEmpty() {

return (count == -1);

}

bool isFull() {

return (count == size);

}

int extractMin() {

count--;

swap(arr[0], arr[count]);

cout << "deleted: " << arr[count] << endl;

mini\_heapify\_down(0);

return arr[count];

}

void insert(int value) {

if (isFull()) {

cout << "Heap is Full" << endl;

}

else

{

count++;

arr[count - 1] = value;

mini\_heapify\_up(count - 1);

}

}

void display() {

for (int i = 0; i < count; i++) {

cout << arr[i] << " ";

}

}

};

int main() {

int size, val;

cout << "Enter size: ";

cin >> size;

heap H(size);

cout << "Enter Values: " << endl;

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

cin >> val;

H.insert(val);

}

cout << "\nHeap is: ";

H.display();

cout << endl;

H.extractMin();

H.extractMin();

H.extractMin();

H.extractMin();

cout << endl;

system("pause");

return 0;

}

