**Question#1**

**Code:**

#include <iostream>

using namespace std;

class MinHeap{

private:

int\* heap\_array;

int max\_capacity;

int current\_size;

int parent(int index){

return (index - 1) / 2;

}

int leftChild(int index) {

return 2 \* index + 1;

}

int rightChild(int index) {

return 2 \* index + 2;

}

void heapifyDown(int index){

int smallest = index;

int left = leftChild(index);

int right = rightChild(index);

if (left < current\_size && heap\_array[left] < heap\_array[smallest])

smallest = left;

if (right < current\_size && heap\_array[right] < heap\_array[smallest])

smallest = right;

if (smallest != index){

swap(heap\_array[index], heap\_array[smallest]);

heapifyDown(smallest);

}

}

void heapifyUp(int index){

while (index != 0 && heap\_array[parent(index)] > heap\_array[index]) {

swap(heap\_array[index], heap\_array[parent(index)]);

index = parent(index);

}

}

public:

MinHeap(int max\_capacity){

this->max\_capacity = max\_capacity;

heap\_array = new int[max\_capacity];

current\_size = 0;

}

~MinHeap(){

delete[] heap\_array;

}

void insert(int key) {

if (current\_size == max\_capacity) {

cout << "Heap overflow!" << endl;

return;

}

current\_size++;

int index = current\_size - 1;

heap\_array[index] = key;

heapifyUp(index);

}

int extractMin() {

if (current\_size <= 0)

return 99999;

if (current\_size == 1) {

current\_size--;

return heap\_array[0];

}

int root = heap\_array[0];

heap\_array[0] = heap\_array[current\_size - 1];

current\_size--;

heapifyDown(0);

return root;

}

void extractTopThree() {

cout << "Extracting Top 3 Priority Tasks" << endl;

for (int i = 0; i < 3 && current\_size > 0; ++i) {

cout << "- Task with priority: " << extractMin() << endl;

}

}

void printHeap(){

cout << "Heap Content: ";

for (int i = 0; i < current\_size; i++) {

cout << heap\_array[i] << " ";

}

cout << endl;

}

};

int main() {

int max\_capacity = 50;

MinHeap min\_heap(max\_capacity);

int num\_tasks;

cout << "Enter the number of tasks: ";cin >> num\_tasks;

cout << "Enter the priorities of the tasks: ";

for (int i = 0; i < num\_tasks; i++) {

int task\_priority;

cout << "Enter:"; cin >> task\_priority;

min\_heap.insert(task\_priority);

}

cout << "Initial Heap Construction" << endl;

min\_heap.printHeap();

int choice;

do {

cout << "Menu:"<<endl;

cout << "1. Insert a new task"<<endl;

cout << "2. Extract the highest-priority task"<<endl;

cout << "3. Extract the top 3 priority tasks"<<endl;

cout << "4. Display the heap"<<endl;

cout << "5. Exit"<<endl;

cout << "Enter your choice : ";cin >> choice;

if (choice == 1) {

int new\_priority;

cout << "Enter the priority of the new task: ";cin >> new\_priority;

min\_heap.insert(new\_priority);

cout << "Task inserted successfully."<<endl;

}

else if (choice == 2) {

int min\_task = min\_heap.extractMin();

if (min\_task == INT\_MAX)

cout << "The heap is empty."<<endl;

else

cout << "Task extracted with priority: " << min\_task << endl;

}

else if (choice == 3) {

min\_heap.extractTopThree();

}

else if (choice == 4) {

min\_heap.printHeap();

}

else if (choice == 5) {

cout << "Exiting program. Goodbye!"<<endl;

}

else {

cout << "Invalid choice. Please try again."<<endl;

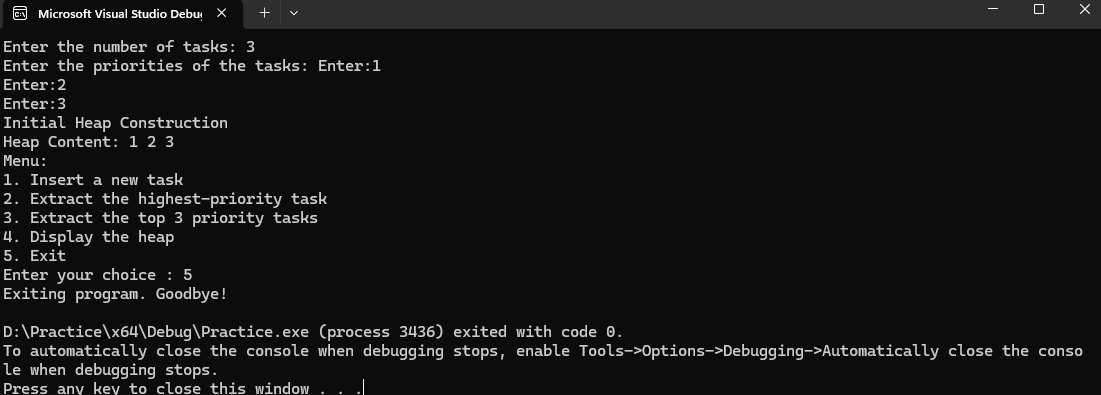
}

} while (choice != 5);

return 0;

}

**Output:**

****

**Question#2**

**Code:**

#include <iostream>

using namespace std;

int calculateRouteCost(int distanceMatrix[][4], int route[], int numCities) {

int totalCost = 0;

for (int i = 0; i < numCities - 1; i++) {

totalCost += distanceMatrix[route[i]][route[i + 1]];

}

totalCost += distanceMatrix[route[numCities - 1]][route[0]];

return totalCost;

}

void swap(int& a, int& b) {

int temp = a;

a = b;

b = temp;

}

void generatePermutations(int cityOrder[], int start, int end, int distanceMatrix[][4],int& minCost, int bestRoute[], int numCities) {

if (start == end) {

int currentRoute[4];

currentRoute[0] = 0;

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

currentRoute[i + 1] = cityOrder[i];

}

int currentCost = calculateRouteCost(distanceMatrix, currentRoute, numCities);

if (currentCost < minCost) {

minCost = currentCost;

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

bestRoute[i] = currentRoute[i];

}

}

return;

}

for (int i = start; i < end; i++) {

swap(cityOrder[start], cityOrder[i]);

generatePermutations(cityOrder, start + 1, end, distanceMatrix, minCost, bestRoute, numCities);

swap(cityOrder[start], cityOrder[i]);

}

}

void solveTsp(int distanceMatrix[][4], int numCities) {

int cityOrder[3];

for (int i = 1; i < numCities; i++) {

cityOrder[i - 1] = i;

}

int minCost =999999;

int bestRoute[4];

generatePermutations(cityOrder, 0, numCities - 1, distanceMatrix, minCost, bestRoute, numCities);

cout << "Shortest Route: ";

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

cout << bestRoute[i] << " -> ";

}

cout << bestRoute[0] << endl;

cout << "Total Cost: " << minCost <<endl;

}

int main() {

const int numCities = 4;

int distanceMatrix[4][4] = {

{0, 10, 15, 20},

{10, 0, 35, 25},

{15, 35, 0, 30},

{20, 25, 30, 0}

};

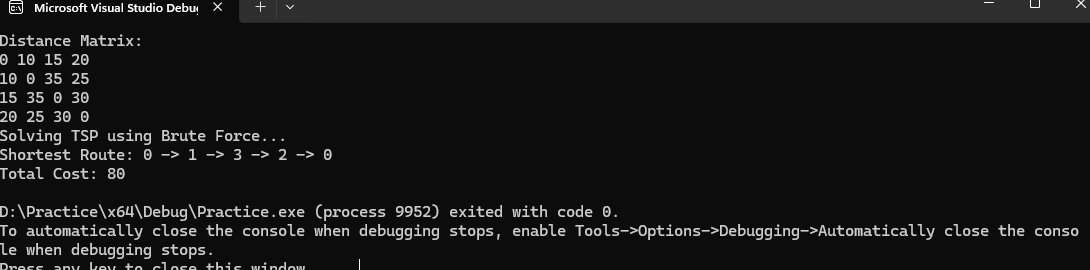
cout << "Solving TSP using Brute Force..."<<endl;

solveTsp(distanceMatrix, numCities);

return 0;

}

**Output:**

****

**Question#3**

**Code:**

#include <iostream>

using namespace std;

class Queue {

private:

int\* queue;

int front, rear, capacity;

public:

Queue(int cap) {

capacity = cap;

queue = new int[capacity];

front = -1;

rear = -1;

}

~Queue() {

delete[] queue;

}

bool empty() {

return front == -1;

}

void enqueue(int val) {

if (rear == capacity - 1) {

cout << "Queue is full!" << endl;

return;

}

if (front == -1)

front = 0;

queue[++rear] = val;

}

int dequeue() {

if (empty()) {

cout << "Queue is empty!" << endl;

return -1;

}

int val = queue[front++];

if (front > rear)

front = rear = -1;

return val;

}

};

class Stack {

private:

int\* stack;

int top, capacity;

public:

Stack(int cap) {

capacity = cap;

stack = new int[capacity];

top = -1;

}

~Stack() {

delete[] stack;

}

bool empty() {

return top == -1;

}

void push(int val) {

if (top == capacity - 1) {

cout << "Stack is full!" << endl;

return;

}

stack[++top] = val;

}

int pop() {

if (empty()) {

cout << "Stack is empty!" << endl;

return -1;

}

return stack[top--];

}

};

class Graph {

private:

int\*\* matrix;

char\* users;

int usr\_count, capacity;

int get\_user\_num(char user) {

for (int i = 0; i < usr\_count; ++i) {

if (users[i] == user)

return i;

}

return -1;

}

public:

Graph(int cap) {

capacity = cap;

usr\_count = 0;

matrix = new int\* [capacity];

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

matrix[i] = new int[capacity] {0};

}

users = new char[capacity];

}

~Graph() {

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

delete[] matrix[i];

}

delete[] matrix;

delete[] users;

}

void add\_new\_user(char user) {

if (usr\_count < capacity) {

users[usr\_count++] = user;

}

}

void add\_new\_edge(char u, char v) {

int x = get\_user\_num(u);

int y = get\_user\_num(v);

if (x != -1 && y != -1) {

matrix[x][y] = 1;

matrix[y][x] = 1;

}

}

void BFS(char start) {

int starting = get\_user\_num(start);

if (starting == -1) {

return;

}

bool\* visited\_arr = new bool[capacity] {false};

Queue q(capacity);

char\* bfsOrder = new char[capacity];

int bfsIndex = 0;

q.enqueue(starting);

visited\_arr[starting] = true;

while (!q.empty()) {

int temp = q.dequeue();

bfsOrder[bfsIndex++] = users[temp];

for (int i = 0; i < usr\_count; ++i) {

if (matrix[temp][i] == 1 && !visited\_arr[i]) {

q.enqueue(i);

visited\_arr[i] = true;

}

}

}

cout << "BFS Traversal: ";

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

cout << bfsOrder[i] << " ";

}

cout << endl;

delete[] visited\_arr;

delete[] bfsOrder;

}

void DFS(char start) {

int starting = get\_user\_num(start);

if (starting == -1) {

return;

}

bool\* visited\_arr = new bool[capacity] {false};

Stack s(capacity);

char\* dfsOrder = new char[capacity];

int dfsIndex = 0;

s.push(starting);

while (!s.empty()) {

int temp = s.pop();

if (!visited\_arr[temp]) {

dfsOrder[dfsIndex++] = users[temp];

visited\_arr[temp] = true;

for (int i = usr\_count - 1; i >= 0; --i) {

if (matrix[temp][i] == 1 && !visited\_arr[i]) {

s.push(i);

}

}

}

}

cout << "DFS Traversal: ";

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

cout << dfsOrder[i] << " ";

}

cout << endl;

delete[] visited\_arr;

delete[] dfsOrder;

}

void shortestPath(char start, char end) {

int starting = get\_user\_num(start);

int endIndex = get\_user\_num(end);

if (starting == -1 || endIndex == -1) {

return;

}

bool\* visited\_arr = new bool[capacity] {false};

Queue q(capacity);

int\* parent = new int[capacity];

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

parent[i] = -1;

}

q.enqueue(starting);

visited\_arr[starting] = true;

while (!q.empty()) {

int temp = q.dequeue();

for (int i = 0; i < usr\_count; ++i) {

if (matrix[temp][i] == 1 && !visited\_arr[i]) {

q.enqueue(i);

visited\_arr[i] = true;

parent[i] = temp;

if (i == endIndex)

break;

}

}

}

int\* path = new int[capacity];

int pathIndex = 0;

for (int at = endIndex; at != -1; at = parent[at]) {

path[pathIndex++] = at;

}

cout << "Shortest Path from " << start << " to " << end << ": ";

for (int i = pathIndex - 1; i >= 0; --i) {

cout << users[path[i]] << " ";

}

cout << endl;

delete[] visited\_arr;

delete[] parent;

delete[] path;

}

};

int main() {

int capacity = 10;

Graph g(capacity);

g.add\_new\_user('A');

g.add\_new\_user('B');

g.add\_new\_user('C');

g.add\_new\_user('D');

g.add\_new\_user('E');

g.add\_new\_user('F');

g.add\_new\_edge('A', 'B');

g.add\_new\_edge('A', 'C');

g.add\_new\_edge('B', 'D');

g.add\_new\_edge('B', 'E');

g.add\_new\_edge('C', 'F');

g.BFS('A');

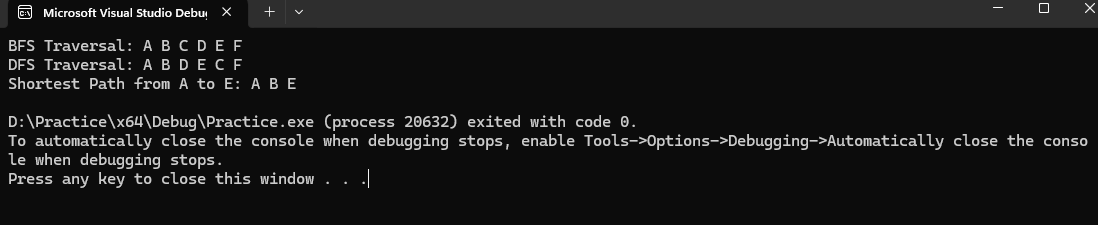
g.DFS('A');

g.shortestPath('A', 'E');

return 0;

}

**Output:**

****

**Question#4**

**Code:**

#include <iostream>

using namespace std;

class RoadNetwork {

private:

int\*\* adjacencyMatrix;

int numBlocks;

public:

RoadNetwork(int\*\* inputMatrix, int blocks) {

numBlocks = blocks;

adjacencyMatrix = new int\* [numBlocks];

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

adjacencyMatrix[i] = new int[numBlocks];

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

adjacencyMatrix[i][j] = inputMatrix[i][j];

}

}

}

~RoadNetwork() {

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

delete[] adjacencyMatrix[i];

}

delete[] adjacencyMatrix;

}

void findMst() {

int\* parent = new int[numBlocks];

int\* key = new int[numBlocks];

bool\* includedInMst = new bool[numBlocks];

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

key[i] = INT\_MAX;

includedInMst[i] = false;

parent[i] = -1;

}

key[0] = 0;

for (int count = 0; count < numBlocks - 1; ++count) {

int minKey = INT\_MAX, u;

for (int v = 0; v < numBlocks; ++v) {

if (!includedInMst[v] && key[v] < minKey) {

minKey = key[v];

u = v;

}

}

includedInMst[u] = true;

for (int v = 0; v < numBlocks; ++v) {

if (adjacencyMatrix[u][v] && !includedInMst[v] && adjacencyMatrix[u][v] < key[v]) {

key[v] = adjacencyMatrix[u][v];

parent[v] = u;

}

}

}

int totalCost = 0;

cout << " Minimum Spanning Tree (MST)"<<endl;

cout << "Edges in the MST and their distances:"<<endl;

for (int i = 1; i < numBlocks; ++i) {

cout << "Block " << parent[i] << " - Block " << i<< " : Distance " << adjacencyMatrix[i][parent[i]] << " km"<<endl;

totalCost += adjacencyMatrix[i][parent[i]];

}

cout << "Total cost of the road network: " << totalCost << " km"<<endl;

delete[] parent;

delete[] key;

delete[] includedInMst;

}

};

int main() {

int blocks = 5;

int\*\* adjacencyMatrix = new int\* [blocks];

int inputMatrix[5][5] = {

{0, 2, 0, 6, 0},

{2, 0, 3, 8, 5},

{0, 3, 0, 0, 7},

{6, 8, 0, 0, 9},

{0, 5, 7, 9, 0}

};

cout << "input Matrix:" << endl;

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

adjacencyMatrix[i] = new int[blocks];

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

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

}

cout << endl;

}

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

adjacencyMatrix[i] = new int[blocks];

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

adjacencyMatrix[i][j] = inputMatrix[i][j];

}

}

RoadNetwork network(adjacencyMatrix, blocks);

network.findMst();

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

delete[] adjacencyMatrix[i];

}

delete[] adjacencyMatrix;

return 0;

}

**Output:**

****