**REFERENCE CODES FOR GRAPHS EXPLAINED IN CLASS**

**Range Min Query using Segment Tree:**

#include <iostream>

#include <cmath>

#include <climits>

using namespace std;

int RMQUtil(int l, int r, int qs, int qe, int i,int st[]) {

if (qs <= l && qe >= r)

return st[i];

if (r < qs || l> qe)

return INT\_MAX;

int mid = (l+r)/2;

return min(RMQUtil(l, mid, qs, qe, 2 \* i + 1,st),

RMQUtil(mid + 1, r, qs, qe, 2 \* i + 2,st));

}

void constructSTUtil(int arr[], int l, int r, int i,int st[]) {

if (l == r) {

st[i] = arr[l];

return;

}

int mid = (l+r)/2;

constructSTUtil(arr, l, mid, i \* 2 + 1,st);

constructSTUtil(arr, mid + 1, r, i \* 2 + 2,st);

st[i] = min(st[i\*2+1],st[i\*2+2]);

}

int main() {

int n, i;

cin >> n;

int arr[n];

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

cin >> arr[i];

int height = ceil(log2(n));

int max\_size = 2 \*pow(2, height) - 1;

int st[max\_size];

constructSTUtil(arr, 0, n - 1,0,st);

int qs, qe;

cin >> qs >> qe;

cout << "Minimum of values in range [" << qs << ", " << qe << "] is = " <<RMQUtil(0, n - 1, qs, qe, 0,st)<< endl;

return 0;}

**Range Min Query using Sparse Table:**

#include <bits/stdc++.h>

using namespace std;

const int MAX = 500;

int st[MAX][MAX];

void buildSparseTable(int arr[], int n) {

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

st[i][0] = arr[i];

for (int j = 1; j <= log2(n); j++) {

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

st[i][j]=min(st[i][j - 1] , st[i + (1<<j-1)][j - 1]);

}

}

}

int query(int L, int R) {

int k = log2(R - L + 1);

return min(st[L][k], st[R-(1<<k)+1][k]);

}

int main() {

int n, s1, e1;

cin >> n;

int arr[n];

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

cin >> arr[i];

buildSparseTable(arr, n);

cin >> s1 >> e1;

cout << query(s1, e1) << endl;

return 0;

}

**Kruskal’s Minimum Spanning Tree (MST) Algorithm**

#include<iostream>

#include<algorithm>

using namespace std;

struct Edge {

int src, dest, weight;

};

bool compareEdges(Edge a, Edge b) {

return a.weight < b.weight;

}

int find(int parent[],int i){

while(parent[i]!=-1){

i=parent[i];

}

return i;

}

void unions(int parent[],int x,int y){

parent[x]=y;

}

void kruskalMST(int V,int E,Edge edge[]){

sort(edge, edge+E, compareEdges);

int parent[V];

fill(parent,parent+V,-1);

Edge res[V];

int eCount=0;

for(int i=0;i<E && eCount<V-1;i++){

int src=edge[i].src;

int dest=edge[i].dest;

int weig=edge[i].weight;

int parentsrc=find(parent,src);

int parentdest=find(parent,dest);

if(parentsrc != parentdest){

res[eCount].src=src;

res[eCount].dest=dest;

res[eCount].weight=weig;

eCount++;

unions(parent,parentsrc,parentdest);

}

}

cout<<"Following are the edges in the constructed MST\n";

int mincost=0;

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

cout<<res[i].src<<" -- "<<res[i].dest<<" == "<<res[i].weight<<endl;

mincost+=res[i].weight;

}

cout<<"Minimum Cost Spanning Tree: "<<mincost<<endl;

}

int main(){

int V, E;

cin >> V >> E;

Edge edges[E];

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

cin >> edges[i].src >> edges[i].dest >> edges[i].weight;

}

kruskalMST(V, E, edges);

return 0;

}

**Prim’s Algorithm for Minimum Spanning Tree (MST)**

#include <bits/stdc++.h>

using namespace std;

const int MAX = 10;

void printMST(int parent[], int graph[MAX][MAX], int V) {

for (int i = 1; i < V; i++)

cout << parent[i] << " " << i << " " << graph[i][parent[i]] << " \n";

}

void primMST(int graph[MAX][MAX], int V) {

int parent[V];

int key[V];

bool mstSet[V]={false};

fill(key, key + V, INT\_MAX);

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

parent[0] = -1;

pq.push({0,0});

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

int u =pq.top().second;

pq.pop();

mstSet[u] = true;

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

if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

pq.push({key[v],v});

}

} }

printMST(parent, graph, V);

}

int main() {

int V;

cin >> V;

int graph[MAX][MAX];

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

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

cin >> graph[i][j];

}

}

primMST(graph, V);

return 0;

}

**Dijkstra Algorithm:**

#include <bits/stdc++.h>

using namespace std;

#define V 5

void printSolution(int dist[]) {

cout<<"Vertex \t Distance from Source\n";

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

printf("%d %d\n", i, dist[i]);

}

void dijkstra(int graph[V][V], int src) {

int dist[V];

fill(dist, dist + V, INT\_MAX);

bool sptSet[V];

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

dist[src] = 0;

pq.push({0,src});

for (int node = 0;node<V;node++){

int u = pq.top().second;

pq.pop();

sptSet[u] = true;

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

if (graph[u][i] && !sptSet[i]){

dist[i] = min(dist[i],dist[u]+graph[u][i]);

pq.push({dist[i],i});

}

}

printSolution(dist);

}

int main() {

int graph[V][V],s;

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

for(int j=0;j<V;j++)

cin>>graph[i][j];

cin>>s;

dijkstra(graph, s);

return 0;

}

**Floyd Warshall Algorithm:**

// You are using GCC

#include<bits/stdc++.h>

using namespace std;

const int INF = 999;

int V;

void printmatrix(int graph2[][100]){

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

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

if (graph2[i][j] == INF)

cout << "INF ";

else

cout << graph2[i][j] << " ";

}

cout << endl;

}

}

void helper(int graph2[][100], int graph[][100]){

cout << "Original matrix" << endl;

printmatrix(graph);

for(int k=0; k<V; k++)

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

for(int j=0; j<V; j++)

graph2[i][j] = min(graph2[i][j], graph2[i][k]+graph2[k][j]);

cout << "Shortest path matrix" << endl;

printmatrix(graph2);

}

int main(){

cin>>V;

int edges;

cin>>edges;

int graph[100][100];

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

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

if(i==j){

graph[i][j] = 0;

}

else

graph[i][j] = INF;

}

}

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

int start, end, value;

cin>>start>>end>>value;

graph[start][end] = value;

graph[end][start] = value;

}

int graph2[100][100];

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

for(int j=0; j<V; j++)

graph2[i][j] = graph[i][j];

helper(graph,graph2);

}