Prim’s Algorithm

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX 100

typedef struct {

int vertex;

int key;

} MinHeapNode;

typedef struct {

MinHeapNode\* nodes[MAX];

int size;

int pos[MAX];

} MinHeap;

MinHeap\* createMinHeap() {

MinHeap\* heap = (MinHeap\*)malloc(sizeof(MinHeap));

heap->size = 0;

for (int i = 0; i < MAX; i++) heap->pos[i] = -1;

return heap;

}

void swapMinHeapNode(MinHeapNode\*\* a, MinHeapNode\*\* b) {

MinHeapNode\* t = \*a;

\*a = \*b;

\*b = t;

}

void minHeapify(MinHeap\* heap, int idx) {

int smallest = idx;

int left = 2\*idx + 1;

int right = 2\*idx + 2;

if (left < heap->size && heap->nodes[left]->key < heap->nodes[smallest]->key)

smallest = left;

if (right < heap->size && heap->nodes[right]->key < heap->nodes[smallest]->key)

smallest = right;

if (smallest != idx) {

MinHeapNode\* smallestNode = heap->nodes[smallest];

MinHeapNode\* idxNode = heap->nodes[idx];

heap->pos[smallestNode->vertex] = idx;

heap->pos[idxNode->vertex] = smallest;

swapMinHeapNode(&heap->nodes[smallest], &heap->nodes[idx]);

minHeapify(heap, smallest);

}

}

int isEmpty(MinHeap\* heap) {

return heap->size == 0;

}

MinHeapNode\* extractMin(MinHeap\* heap) {

if (heap->size == 0)

return NULL;

MinHeapNode\* root = heap->nodes[0];

MinHeapNode\* lastNode = heap->nodes[--heap->size];

heap->nodes[0] = lastNode;

heap->pos[root->vertex] = -1;

heap->pos[lastNode->vertex] = 0;

minHeapify(heap, 0);

return root;

}

void decreaseKey(MinHeap\* heap, int v, int key) {

int i = heap->pos[v];

heap->nodes[i]->key = key;

while (i && heap->nodes[i]->key < heap->nodes[(i - 1)/2]->key) {

heap->pos[heap->nodes[i]->vertex] = (i - 1)/2;

heap->pos[heap->nodes[(i - 1)/2]->vertex] = i;

swapMinHeapNode(&heap->nodes[i], &heap->nodes[(i - 1)/2]);

i = (i - 1)/2;

}

}

int isInMinHeap(MinHeap\* heap, int v) {

return heap->pos[v] < heap->size && heap->pos[v] != -1;

}

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

int parent[MAX];

int key[MAX];

MinHeap\* heap = createMinHeap();

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

parent[v] = -1;

key[v] = INT\_MAX;

heap->nodes[v] = (MinHeapNode\*)malloc(sizeof(MinHeapNode));

heap->nodes[v]->vertex = v;

heap->nodes[v]->key = key[v];

heap->pos[v] = v;

}

key[0] = 0;

heap->nodes[0]->key = 0;

heap->size = V;

while (!isEmpty(heap)) {

MinHeapNode\* minNode = extractMin(heap);

int u = minNode->vertex;

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

if (graph[u][v] && isInMinHeap(heap, v) && graph[u][v] < key[v]) {

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

parent[v] = u;

decreaseKey(heap, v, key[v]);

}

}

}

int totalCost = 0;

printf("Edge Weight\n");

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

printf("%d - %d %d\n", parent[i], i, graph[i][parent[i]]);

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

}

printf("Total Cost of MST = %d\n", totalCost);

}

int main() {

int V;

printf("Enter number of vertices: ");

scanf("%d", &V);

int graph[MAX][MAX];

printf("Enter the adjacency matrix:\n");

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

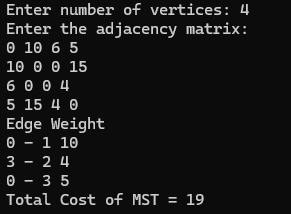
for (int j = 0; j < V; j++)

scanf("%d", &graph[i][j]);

primMST(graph, V);

return 0;

}



Kruskal’s

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX 100

struct Edge {

int src, dest, weight;

};

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

if (parent[i] != i)

parent[i] = find(parent, parent[i]);

return parent[i];

}

void unionSet(int parent[], int rank[], int x, int y) {

int xroot = find(parent, x);

int yroot = find(parent, y);

if (rank[xroot] < rank[yroot])

parent[xroot] = yroot;

else if (rank[xroot] > rank[yroot])

parent[yroot] = xroot;

else {

parent[yroot] = xroot;

rank[xroot]++;

}

}

int compareEdges(const void\* a, const void\* b) {

return ((struct Edge\*)a)->weight - ((struct Edge\*)b)->weight;

}

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

struct Edge edges[MAX \* MAX];

int edgeCount = 0;

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

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

if (graph[i][j] != 0) {

edges[edgeCount].src = i;

edges[edgeCount].dest = j;

edges[edgeCount].weight = graph[i][j];

edgeCount++;

}

}

}

qsort(edges, edgeCount, sizeof(edges[0]), compareEdges);

int parent[MAX], rank[MAX];

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

parent[i] = i;

rank[i] = 0;

}

int mstEdges = 0;

int totalCost = 0;

printf("Edge \tWeight\n");

for (int i = 0; i < edgeCount && mstEdges < V - 1; i++) {

int u = edges[i].src;

int v = edges[i].dest;

int setU = find(parent, u);

int setV = find(parent, v);

if (setU != setV) {

printf("%d - %d \t%d\n", u, v, edges[i].weight);

totalCost += edges[i].weight;

unionSet(parent, rank, setU, setV);

mstEdges++;

}

}

printf("Total cost of MST: %d\n", totalCost);

}

int main() {

int V, graph[MAX][MAX];

printf("Enter number of vertices: ");

scanf("%d", &V);

printf("Enter adjacency matrix (use 0 for no edge):\n");

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

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

scanf("%d", &graph[i][j]);

KruskalFromAdjMatrix(graph, V);

return 0;

}

