Московский авиационный институт

(Национальный исследовательский университет)

Факультет №3:

*Системы управления, информатика и электроэнергетика*.

Кафедра 304.

«Алгоритмы и обработка данных»

Отчёт по лабораторной работе №1

Тема

**«Поиск кратчайших путей в графе. Построение остовного дерева графа.»**1 вариант

Группа: М3О-225Б-23

Выполнили:

Евстифеев А.С.

Никулин Г.С.

Приняла:

*Дмитриева Е.А.*

Москва 2023 г.

Оглавление

[Задание 2](#__RefHeading___Toc356_3558528703)

[Структурные схемы алгоритмов 3](#__RefHeading___Toc361_3558528703)

[Дейкстра 3](#__RefHeading___Toc458_3558528703)

[Крускал 3](#__RefHeading___Toc460_3558528703)

[Код программы 3](#__RefHeading___Toc363_3558528703)

[Граф 7](#__RefHeading___Toc365_3558528703)

[Остовное дерево 7](#__RefHeading___Toc367_3558528703)

## Задание

Для взвешенного ориентированного графа, состоящего как минимум из 10 вершин, реализовать по вариантам:

1. алгоритм поиска кратчайшего пути;
2. сделав тот же самый граф неориентированным, построить его остовное дерево минимальной стоимости.

Граф для первого пункта задания придумывается самостоятельно (не нужно связывать каждую вершину со всеми остальными, не делайте граф полным).

Матрицу смежности ( значения весов каждого ребра) лучше определить в начале программы, не вводить значения с клавиатуры или из файла.

## Код программы

#include <iostream>

#define V 12

#define INF 1000000

struct Edge {

int u, v, weight;

};

auto findParent = [](int v, int parent[]) {

while (parent[v] != v)

v = parent[v];

return v;

};

void dijkstra(int source, int graph[][V]);

void kruskal(int graph[][V]);

int main() {

int graph[V][V] = {

{ 0, 7, 9, INF, INF, 14, INF, INF, INF, INF, INF, INF },

{ 7, 0, 10, 15, INF, INF, INF, INF, INF, INF, INF, INF },

{ 9, 10, 0, 11, INF, 2, INF, INF, INF, INF, INF, INF },

{ INF, 15, 11, 0, 6, INF, INF, INF, INF, INF, INF, INF },

{ INF, INF, INF, 6, 0, 9, 11, INF, INF, INF, INF, INF },

{ 14, INF, 2, INF, 9, 0, 1, INF, INF, INF, INF, INF },

{ INF, INF, INF, INF, 11, 1, 0, 8, INF, INF, INF, INF },

{ INF, INF, INF, INF, INF, INF, 8, 0, 7, INF, INF, INF },

{ INF, INF, INF, INF, INF, INF, INF, 7, 0, 5, INF, INF },

{ INF, INF, INF, INF, INF, INF, INF, INF, 5, 0, 12, INF },

{ INF, INF, INF, INF, INF, INF, INF, INF, INF, 12, 0, 3 },

{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, 3, 0 }

};

int source;

std::cout << "Enter start point (from 0 to " << V - 1 << "): ";

std::cin >> source;

dijkstra(source, graph);

std::cout << "\nCruskal:\n";

kruskal(graph);

return 0;

}

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

int dist[V], pred[V], visited[V];

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

dist[i] = INF;

pred[i] = -1;

visited[i] = 0;

}

dist[source] = 0;

printf("\t\t1\t2\t3\t4\t5\t6\t7\t8\t9\t10\t11\t12\n");

printf("distance:\t");

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

printf("%d\t",dist[j]);

}

printf("\nprevious:\t");

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

printf("%d\t",pred[j]);

}

printf("\n\n");

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

int minDist = INF;

int u = -1;

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

if (!visited[j] && dist[j] < minDist) {

minDist = dist[j];

u = j;

}

}

if (u == -1) break;

visited[u] = 1;

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

if (!visited[v] && graph[u][v] != INF && dist[u] + graph[u][v] < dist[v]) {

dist[v] = dist[u] + graph[u][v];

pred[v] = u;

}

}

///////////////////////////////////

printf("distance:\t");

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

printf("%d\t",dist[j]);

}

printf("\nprevious:\t");

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

printf("%d\t",pred[j]);

}

printf("\n\n");

/////////////////////////////

}

std::cout << "Shortest ways from point" << source << ":\n";

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

if (dist[i] == INF) {

std::cout << "point " << i << ": unaccesable\n";

}

else {

std::cout << "point " << i << ": length = " << dist[i] << ", way = ";

int path[V], pathIndex = 0;

for (int v = i; v != -1; v = pred[v])

path[pathIndex++] = v;

for (int j = pathIndex - 1; j >= 0; --j)

std::cout << path[j] << " ";

std::cout << std::endl;

}

}

}

void kruskal(int graph[][V]) {

Edge edges[V \* V];

int edgeCount = 0;

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

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

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

edges[edgeCount].u = i;

edges[edgeCount].v = j;

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

edgeCount++;

edges[edgeCount].u = j;

edges[edgeCount].v = i;

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

edgeCount++;

}

}

}

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

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

if (edges[j].weight > edges[j + 1].weight) {

Edge temp = edges[j];

edges[j] = edges[j + 1];

edges[j + 1] = temp;

}

}

}

int parent[V];

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

parent[i] = i;

std::cout << "MST:\n";

int mstWeight = 0;

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

int uParent = findParent(edges[i].u, parent);

int vParent = findParent(edges[i].v, parent);

if (uParent != vParent) {

std::cout << "edge: " << edges[i].u << " - " << edges[i].v << ", weight = " << edges[i].weight << std::endl;

mstWeight += edges[i].weight;

parent[uParent] = vParent;

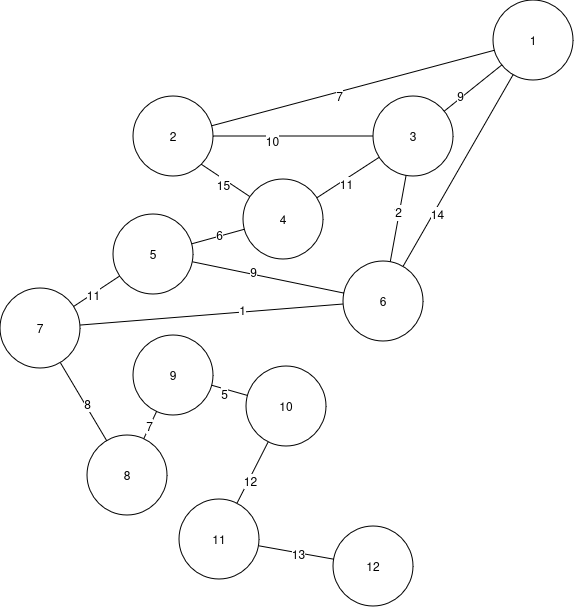
}

}

std::cout << "Wheight of MST: " << mstWeight << std::endl;

}

## Граф



## Остовное дерево

