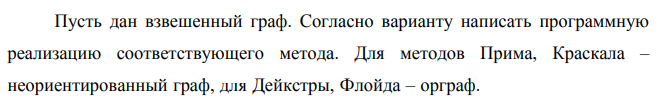
**Лабораторна робота №6**

Основные вычислительные алгоритмы на графах

**Мета роботи**: изучение оптимизирующих алгоритмов на графах: поиск кратчайшего пути, построение минимальных остовных деревьев.

*Завдання*



5. Алгоритм Дейкстры. Представление графа – список смежности.

**Ход работы**

1. **Текст программы**

#include <iostream>

#include <vector>

#include <string>

int const INF = 2'000'000'000;

struct Edge {

int a, b, w;

};

void run();

void printHello();

std::vector<Edge> getEdges(int verticesCount);

bool isCorrectEgde(Edge& edge, int verticesCount);

std::vector<int> Dijkstra(std::vector<Edge> edges, int verticesCount, int vertex);

int getNextCurrect(std::vector<int>& distances, std::vector<bool>& doneVertices);

template <typename T>

T prompt(const char label[]);

int main() {

printHello();

while (true) {

run();

if (prompt<std::string>("\nRepeat? (0 to exit): ") == "0") {

break;

}

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

}

return 0;

}

std::vector<int> Dijkstra(std::vector<Edge> edges, int verticesCount, int vertex) {

std::vector<int> distances(verticesCount);

std::vector<bool> doneVertices(verticesCount);

int current = vertex;

for (auto &item : distances) {

item = INF;

}

distances[current] = 0;

while (current != -1) {

doneVertices[current] = true;

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

if (doneVertices[i]) continue;

for (auto& e : edges) {

if (e.a != current || e.b != i) continue;

if (e.w >= distances[i]) continue;

distances[i] = distances[current] + e.w;

break;

}

}

current = getNextCurrect(distances, doneVertices);

}

return distances;

}

int getNextCurrect(std::vector<int> &distances, std::vector<bool> &doneVertices) {

int min = INF;

int index = -1;

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

if (doneVertices[i]) continue;

if (distances[i] >= min) continue;

min = distances[i];

index = i;

}

return index;

}

void run() {

int verticesCount = prompt<int>("Enter the number of vertices: ");

auto edges = getEdges(verticesCount);

int findingVertex = -1;

while (true) {

findingVertex = prompt<int>("\nEnter the number of finding vertex: ") - 1;

if (findingVertex >= 0 && findingVertex < verticesCount) {

break;

}

std::cout << "Wrong vertex number:\n";

}

auto minDistances = Dijkstra(edges, verticesCount, findingVertex);

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

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

std::string distance = (minDistances[i] < INF) ?

std::to\_string(minDistances[i]) : "Impossible";

std::cout << "To vertex " << i + 1 << ": " << distance << std::endl;

}

}

std::vector<Edge> getEdges(int verticesCount) {

int size = prompt<unsigned short>("Enter the number of edges: ");

std::vector<Edge> edges;

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

std::cout << "- Edge " << i + 1 << std::endl;

Edge edge;

edge.a = prompt<int>("From: ") - 1;

edge.b = prompt<int>("To: ") - 1;

edge.w = prompt<int>("Weight: ");

if (!isCorrectEgde(edge, verticesCount)) {

std::cout << "Wrong egde data. Try again:\n";

i--;

continue;

}

edges.push\_back(edge);

}

return edges;

}

bool isCorrectEgde(Edge &edge, int verticesCount) {

return edge.a >= 0 && edge.b >= 0 &&

edge.a < verticesCount && edge.b < verticesCount &&

edge.a != edge.b;

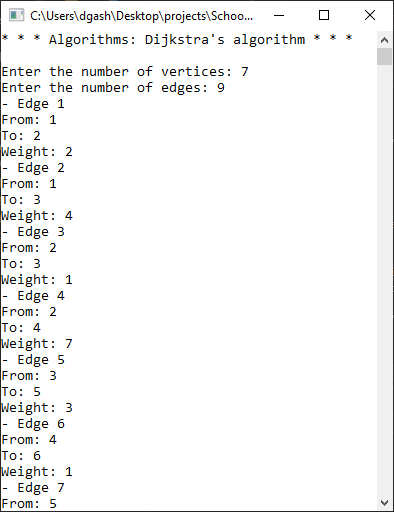
}

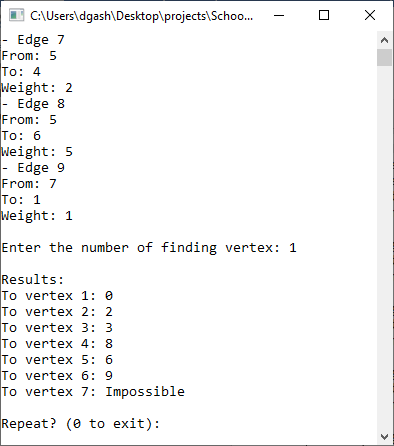
void printHello() {

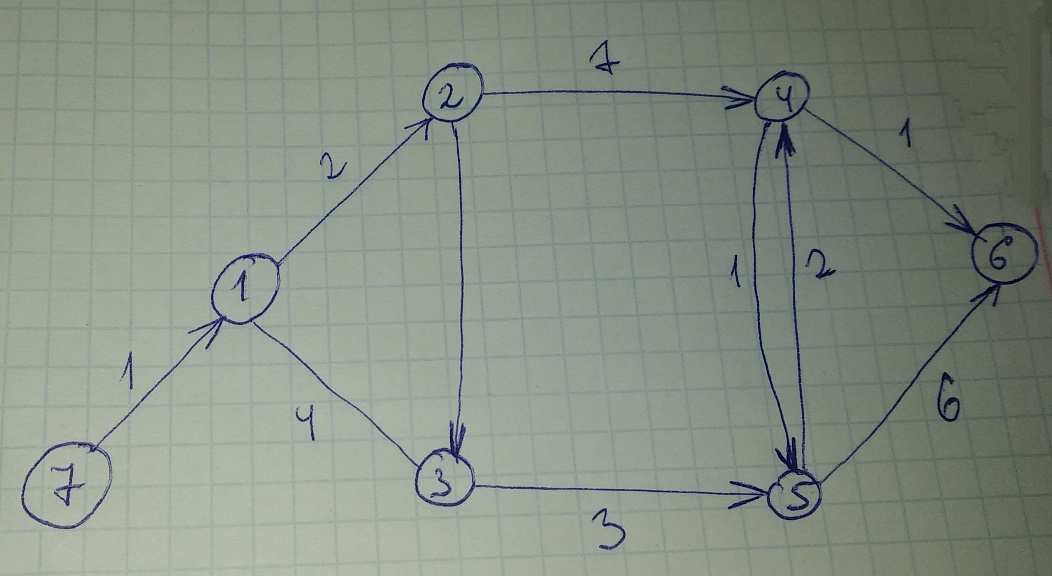
std::cout << "\* \* \* Algorithms: Dijkstra's algorithm \* \* \*\n\n";

}

1. **Результат выполнения программы**





****

**Итог:** на этой лабораторной работе я изучил оптимизирующие алгоритмы на графах: поиск кратчайшего пути, построение минимальных остовных деревьев.