|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
|  | | |
|  | | |
| Практическое задание №10 | | |
| по дисциплине «Методы построения и анализа алгоритмов» | | |
|  | | |
| **кафедра теоретической и прикладной информатики** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-03 |
| Бригада: | Место для ввода текста. |
| Студенты: | Сидоров Даниил, |
|  | Малыгин Сергей |
|  |  |
| Преподаватель: | Щукин Георгий Анатольевич |
|  |  |
|  | | |
| Новосибирск | | |
| 2021 | | |

**1.Результаты замеров:**

|  |  |  |
| --- | --- | --- |
| Вершины | Ребра | Алгоритм Дейкстры |
| 100 | 100 | 0.0003147 |
| 1 000 | 1 000 | 0.0102713 |
| 10 000 | 10 000 | 1.67457 |
|  |  |  |
| 100 | 200 | 0.0003578 |
| 1 000 | 2 000 | 0.0095412 |
| 10 000 | 20 000 | 2.46727 |
|  |  |  |
| 100 | 500 | 0.0003164 |
| 1 000 | 5 000 | 0.0097171 |
| 10 000 | 50 000 | 3.55913 |
|  |  |  |
| 100 | 1 000 | 0.0003687 |
| 1 000 | 10 000 | 0.0103529 |
| 10 000 | 100 000 | 4.81706 |
|  |  |  |
| 100 | 3 000 | 0.0003329 |
| 1 000 | 30 000 | 0.0116884 |
| 10 000 | 300 000 | 5.52706 |

**2.Программа:**

**shortest\_path.cpp**

#include "shortest\_path.h"

#include "graph.h"

#include <cmath>

#include <algorithm>

#include <numeric>

#include <random>

using namespace std;

vector<int> build\_path(map<int, int> parent, int start, int end)

{

if (start == end) return {};

vector<int> result = { end };

int vertice = end;

while (vertice != start)

{

vertice = parent[vertice];

result.insert(result.begin(),vertice);

}

return result;

}

vector<int> shortest\_path(const Graph &graph, int start\_vertex, int end\_vertex)

{

map<int, double> distance;

map<int, int> parent;

vector<int> adjacent\_vertices;

double min;

int u = 0;

vector<int> Q = graph.get\_vertices();

for (int v = 0; v <= graph.get\_vertices().size(); v++)

distance[v] = numeric\_limits<double>::infinity();

distance[start\_vertex] = 0;

while (Q.size() != 0)

{

min = numeric\_limits<double>::infinity();

for (const auto &elem : Q)

if (distance[elem] < min)

{

u = elem;

min = distance[elem];

}

if (min == numeric\_limits<double>::infinity()) return {};

if (u == end\_vertex) return build\_path(parent, start\_vertex, end\_vertex);

adjacent\_vertices = graph.get\_adjacent\_vertices(u);

for (const auto &v : adjacent\_vertices)

if (distance[v] > distance[u] + graph.edge\_weight(u, v))

{

distance[v] = distance[u] + graph.edge\_weight(u, v);

parent[v] = u;

}

auto it = find(Q.begin(), Q.end(), u);

Q.erase(it);

}

return {};

}

Graph random\_graph(Graph &graph, int num\_vertices, int num\_edges)

{

int remainingEdges = num\_edges - num\_vertices;

vector<int> vertices(num\_vertices);

iota(vertices.begin(), vertices.end(), 1);

for (int i = 1; i <= num\_vertices; i++)

graph.add\_vertex(i);

default\_random\_engine generator;

uniform\_int\_distribution<int> distributionEdge(1, 100);

uniform\_int\_distribution<int> distributionVertice(1, num\_vertices);

for (int i = 1; i <= num\_vertices - 1; i++)

graph.add\_edge(i, i + 1, distributionEdge(generator));

while (remainingEdges > 0)

{

int firstVertex = distributionVertice(generator);

int endVertex = distributionVertice(generator);

if (firstVertex != endVertex && !graph.has\_edge(firstVertex, endVertex))

{

graph.add\_edge(firstVertex, endVertex, distributionEdge(generator));

remainingEdges--;

}

}

return graph;

}

**main.cpp**

#define CATCH\_CONFIG\_RUNNER

#include "catch.hpp"

#include <iostream>

#include <chrono>

#include "shortest\_path.h"

using namespace std;

int main(int argc, char\* argv[]) {

int result = Catch::Session().run(argc, argv);

Graph graph;

for (int i = 100; i <=10000; i\*=10)

{

cout << "i: " << i << endl;

graph = random\_graph(graph, i, 30\*i);

auto t1 = std::chrono::high\_resolution\_clock::now();

vector<int> res = shortest\_path(graph, 0, i);

auto t2 = std::chrono::high\_resolution\_clock::now();

auto seconds = std::chrono::duration<double>(t2 - t1).count();

cout << "Shortest path: " << seconds << endl;

}

return result;

}