|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
|  | | |
|  | | |
| Практическое задание №11 | | |
| 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 Прима |
| 100 | 100 | 0.0008018 |
| 1 000 | 1 000 | 0.0368007 |
| 10 000 | 10 000 | 4.50882 |
|  |  |  |
| 100 | 200 | 0.0007969 |
| 1 000 | 2 000 | 0.0335717 |
| 10 000 | 20 000 | 6.19329 |
|  |  |  |
| 100 | 500 | 0.0009265 |
| 1 000 | 5 000 | 0.0381811 |
| 10 000 | 50 000 | 9.38024 |
|  |  |  |
| 100 | 1 000 | 0.0008319 |
| 1 000 | 10 000 | 0.0417493 |
| 10 000 | 100 000 | 11.9357 |
|  |  |  |
| 100 | 3 000 | 0.0012344 |
| 1 000 | 30 000 | 0.0583721 |
| 10 000 | 300 000 | 11.6807 |

**2.Программа:**

**min\_spanning\_tree.cpp**

#include "min\_spanning\_tree.h"

#include "graph.h"

#include <cmath>

#include <algorithm>

#include <numeric>

#include <random>

using namespace std;

vector<pair<int, int>> min\_spanning\_tree(const Graph &graph, int start) {

map<int, double> MinWeight;

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

MinWeight[i] = numeric\_limits<double>::infinity();

MinWeight[start] = 0;

map<int, int> Parent;

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

vector<int> adjacent\_vertices;

int u;

double min;

while (Q.size() > 0)

{

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

for (const auto &elem : Q)

{

if (MinWeight[elem] < min)

{

min = MinWeight[elem];

u = elem;

}

}

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

for (const auto &v : adjacent\_vertices)

{

if (find(Q.begin(), Q.end(), v) != Q.end() && MinWeight[v] > graph.edge\_weight(u, v))

{

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

Parent[v] = u;

}

}

Q.erase(find(Q.begin(), Q.end(), u));

}

vector<pair<int, int>> result;

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

if (i != start)

result.push\_back(make\_pair(i, Parent[i]));

return result;

}

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 <chrono>

#include "graph.h"

#include "min\_spanning\_tree.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, i);

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

vector<pair<int, int>> res = min\_spanning\_tree(graph, 0);

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

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

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

}

return result;

}