|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра параллельных вычислительных технологий | | |
| Практическое задание № 9 | | |
| по дисциплине «Методы построения и анализа алгоритмов» | | |
| **Алгоритм Дейкстры** | | |
|  | | |
|  |  | АРИФУЛОВ ТИГРАН |
| Группа ПМ-01 | ДЫЧКО АРСЕНИЙ |
|  | ЖИЖЧЕНКО ЛЕОНИД |
|  | САМСОНОВ СЕМЕН |
|  | ЯКОВЛЕВА ЕЛЕНА |
|  |  |
| Преподаватель | домников пётр александрович |
|  |  |
| Новосибирск, 2021 | | |

# Задание:

Алгоритм Дейкстры. Поиск кратчайшего пути между двумя вершинами в взвешенном графе. Поиск кратчайших путей между всеми парами вершин с распараллеливанием. Граф задается в файле в следующем формате. Первое число – количество вершин в графе. Второе число – количество ребер в графе. Далее перечислены ребра графа, заданные тройками чисел: первое число – начальная вершина ребра, второе число – конечная вершина ребра, третье число – вес ребра (натуральное число). На выходе файл с перечислением всех существующих путей от одной вершины до другой, с их расстоянием.

# Текст программы:

#include <vector>

#include <string>

#include <iostream>

#include <fstream>

#include <queue>

#include <omp.h>

#include <chrono>

#pragma region Defines

#define pb push\_back

#define mp make\_pair

#define F first

#define S second

#define all(x) (x).begin(),(x).end()

using pii = std::pair<int, int>;

#pragma endregion

using namespace std;

class Timer

{

private:

// Псевдонимы типов используются для удобного доступа к вложенным типам

using clock\_t = std::chrono::high\_resolution\_clock;

using second\_t = std::chrono::duration<double, std::ratio<1> >;

std::chrono::time\_point<clock\_t> m\_beg;

public:

Timer() : m\_beg(clock\_t::now())

{

}

void reset()

{

m\_beg = clock\_t::now();

}

double elapsed() const

{

return std::chrono::duration\_cast<second\_t>(clock\_t::now() - m\_beg).count();

}

};

inline void dijkstra(const vector < vector < pii > >& g, int f, vector < int >& distance, vector < int >& parent)

{

priority\_queue < pii > q; // [first]: weight, [second]: vertexTo

distance.resize((int)g.size(), INT\_MAX);

parent.resize((int)g.size());

distance[f] = 0;

q.push(mp(distance[f], f));

while (!q.empty())

{

int v = q.top().S;

int w = -q.top().F;

q.pop();

if (w > distance[v])

continue;

for (int i = 0; i < (int)g[v].size(); ++i)

{

int to = g[v][i].F;

int weight = g[v][i].S;

if (distance[v] + weight < distance[to])

{

distance[to] = distance[v] + weight;

q.push(mp(-distance[to], to));

parent[to] = v;

}

}

}

}

inline void getPath(const vector < int >& distance, const vector < int >& parent, const int f, const int s, string& outStr)

{

//outStr = to\_string(f) + "\t\t" + to\_string(s) + "\t\t" + to\_string(distance[s]);

outStr = "Begin: " + to\_string(f) + ",\tend: " + to\_string(s) + ",\tdistance: " + to\_string(distance[s]) + ",\tpath: ";

vector < int > path;

for (int x = s; x != f; x = parent[x])

path.pb(x);

path.pb(f);

reverse(all(path));

for (auto x : path)

outStr += to\_string(x) + " ";

}

inline void getAllRoutes(const vector < vector < pii > >& graph, ostream& out)

{

int graphSize = (int)graph.size();

#pragma omp parallel for schedule(dynamic, 100)

for (int startVertex = 1; startVertex < graphSize; ++startVertex)

{

vector <int> distance;

vector <int> parent;

string path;

dijkstra(graph, startVertex, distance, parent);

for (int i = 1; i < (int)graph.size(); ++i)

{

if (i == startVertex || distance[i] == INT\_MAX)

continue;

getPath(distance, parent, startVertex, i, path);

out << path + "\n";

}

}

}

int main()

{

setlocale(LC\_ALL, "");

Timer timer;

ifstream in; in.open("input.txt");

if (!in.is\_open()) throw exception("Input file is not open!");

int n, m;

in >> n >> m;

vector < vector < pii > > g(n + 1); // [first]: vertexTo, [second]: weight

for (int i = 1; i <= m; ++i)

{

int u, v, w;

in >> u >> v >> w;

g[u].pb(mp(v, w));

g[v].pb(mp(u, w));

}

in.close();

ofstream out("output.txt");

if (!out.is\_open()) throw exception("Output file is not open!");

timer.reset();

getAllRoutes(g, out);

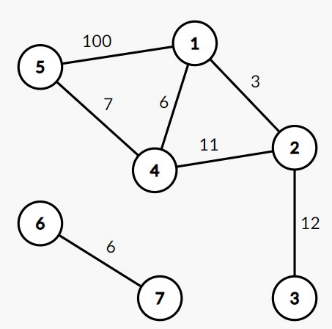
cout << timer.elapsed() << endl;

out.close();

return 0;

}

# Результат работы программы:

1. Тест №1
   1. Входные данные:

7 7  
1 2 3  
1 4 6  
1 5 100  
2 3 12  
2 4 11  
4 5 7  
6 7 6

* 1. Выходные данные:

Begin: 1, end: 2, distance: 3, path: 1 2

Begin: 1, end: 3, distance: 15, path: 1 2 3

Begin: 1, end: 4, distance: 6, path: 1 4

Begin: 1, end: 5, distance: 13, path: 1 4 5

Begin: 2, end: 1, distance: 3, path: 2 1

Begin: 2, end: 3, distance: 12, path: 2 3

Begin: 2, end: 4, distance: 9, path: 2 1 4

Begin: 2, end: 5, distance: 16, path: 2 1 4 5

Begin: 3, end: 1, distance: 15, path: 3 2 1

Begin: 3, end: 2, distance: 12, path: 3 2

Begin: 3, end: 4, distance: 21, path: 3 2 1 4

Begin: 3, end: 5, distance: 28, path: 3 2 1 4 5

Begin: 4, end: 1, distance: 6, path: 4 1

Begin: 4, end: 2, distance: 9, path: 4 1 2

Begin: 4, end: 3, distance: 21, path: 4 1 2 3

Begin: 4, end: 5, distance: 7, path: 4 5

Begin: 5, end: 1, distance: 13, path: 5 4 1

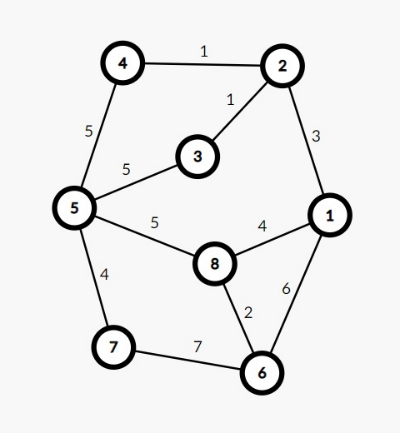
Begin: 5, end: 2, distance: 16, path: 5 4 1 2

Begin: 5, end: 3, distance: 28, path: 5 4 1 2 3

Begin: 5, end: 4, distance: 7, path: 5 4

Begin: 6, end: 7, distance: 6, path: 6 7

Begin: 7, end: 6, distance: 6, path: 7 6

1. Тест №2
   1. Входные данные:

8 11  
4 2 1  
1 2 3  
7 5 4  
2 3 1  
8 6 2  
8 5 5  
5 4 5  
7 6 7  
3 5 5  
1 6 6  
8 1 4

* 1. Выходные данные:

Begin: 1, end: 2, distance: 3, path: 1 2

Begin: 1, end: 3, distance: 4, path: 1 2 3

Begin: 1, end: 4, distance: 4, path: 1 2 4

Begin: 1, end: 5, distance: 9, path: 1 8 5

Begin: 1, end: 6, distance: 6, path: 1 6

Begin: 1, end: 7, distance: 13, path: 1 6 7

Begin: 1, end: 8, distance: 4, path: 1 8

Begin: 2, end: 1, distance: 3, path: 2 1

Begin: 2, end: 3, distance: 1, path: 2 3

Begin: 2, end: 4, distance: 1, path: 2 4

Begin: 2, end: 5, distance: 6, path: 2 4 5

Begin: 2, end: 6, distance: 9, path: 2 1 6

Begin: 2, end: 7, distance: 10, path: 2 4 5 7

Begin: 2, end: 8, distance: 7, path: 2 1 8

Begin: 3, end: 1, distance: 4, path: 3 2 1

Begin: 3, end: 2, distance: 1, path: 3 2

Begin: 3, end: 4, distance: 2, path: 3 2 4

Begin: 3, end: 5, distance: 5, path: 3 5

Begin: 3, end: 6, distance: 10, path: 3 2 1 6

Begin: 3, end: 7, distance: 9, path: 3 5 7

Begin: 3, end: 8, distance: 8, path: 3 2 1 8

Begin: 4, end: 1, distance: 4, path: 4 2 1

Begin: 4, end: 2, distance: 1, path: 4 2

Begin: 4, end: 3, distance: 2, path: 4 2 3

Begin: 4, end: 5, distance: 5, path: 4 5

Begin: 4, end: 6, distance: 10, path: 4 2 1 6

Begin: 4, end: 7, distance: 9, path: 4 5 7

Begin: 4, end: 8, distance: 8, path: 4 2 1 8

Begin: 5, end: 1, distance: 9, path: 5 8 1

Begin: 5, end: 2, distance: 6, path: 5 4 2

Begin: 5, end: 3, distance: 5, path: 5 3

Begin: 5, end: 4, distance: 5, path: 5 4

Begin: 5, end: 6, distance: 7, path: 5 8 6

Begin: 5, end: 7, distance: 4, path: 5 7

Begin: 5, end: 8, distance: 5, path: 5 8

Begin: 6, end: 1, distance: 6, path: 6 1

Begin: 6, end: 2, distance: 9, path: 6 1 2

Begin: 6, end: 3, distance: 10, path: 6 1 2 3

Begin: 6, end: 4, distance: 10, path: 6 1 2 4

Begin: 6, end: 5, distance: 7, path: 6 8 5

Begin: 6, end: 7, distance: 7, path: 6 7

Begin: 6, end: 8, distance: 2, path: 6 8

Begin: 7, end: 1, distance: 13, path: 7 6 1

Begin: 7, end: 2, distance: 10, path: 7 5 4 2

Begin: 7, end: 3, distance: 9, path: 7 5 3

Begin: 7, end: 4, distance: 9, path: 7 5 4

Begin: 7, end: 5, distance: 4, path: 7 5

Begin: 7, end: 6, distance: 7, path: 7 6

Begin: 7, end: 8, distance: 9, path: 7 5 8

Begin: 8, end: 1, distance: 4, path: 8 1

Begin: 8, end: 2, distance: 7, path: 8 1 2

Begin: 8, end: 3, distance: 8, path: 8 1 2 3

Begin: 8, end: 4, distance: 8, path: 8 1 2 4

Begin: 8, end: 5, distance: 5, path: 8 5

Begin: 8, end: 6, distance: 2, path: 8 6

Begin: 8, end: 7, distance: 9, path: 8 6 7

# Тестирование параллельной версии

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Число вершин | Число рёбер | Время при числе потоков, секунды | | | | | |
| 1 | 2 | 4 | 6 | 10 | 12 |
| 8 | 11 | 0.000125 | 0.000682 | 0.00112 | 0.001859 | 0.00275 | 0.004365 |
| 1000 | 917 | 0.007581 | 0.005004 | 0.005262 | 0.008784 | 0.007254 | 0.007904 |
| 10000 | 9035 | 0.298968 | 0.159570 | 0.105718 | 0.100554 | 0.076601 | 0.077286 |
| 210000 | 27063 | 70.07381 | 35.14008 | 20.76872 | 16.58304 | 24.08412 | 27.40349 |
| 910000 | 89974 | 2361.370 | 1391.036 | 990.96 | 940.358 | 972.754 | 1006.3 |

Тестирование проводилось на 6 ядерном процессоре с 12 потоками. В итоге по полученным данным стало ясно, что наибольшее ускорение программы достигается на 6 потоках. Причём видно, что ускорение от числа потоков нелинейное, и при переходе от 4 к 6 потокам практически не меняется. При изменении принципа вывода данных с 1 файла на несколько, каждый из которых принадлежит своему потоку, можно получить практически линейное ускорение.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Число вершин | Число рёбер | Время при числе потоков, секунды | | | | | |
| 1 | 2 | 4 | 6 | 10 | 12 |
| 10000 | 9035 | 0.302993 | 0.155389 | 0.108696 | 0.084954 | 0.063130 | 0.064686 |
| 210000 | 27063 | 78.808986 | 40.783123 | 21.458733 | 16.891294 | 25.438204 | 27.576759 |

Самый медленный случай – выделять для каждой вершины свой файл.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Число вершин | Число рёбер | Время при числе потоков, секунды | | | | | |
| 1 | 2 | 4 | 6 | 10 | 12 |
| 10000 | 9035 | 2.638012 | 1.445619 | 0.774654 | 1.254985 | 0.832395 | 1.362549 |
| 210000 | 27063 | 132.431708 | 74.418007 | 50.163248 | 39.732654 | 43.149573 | 42.278917 |