|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра параллельных вычислительных технологий | | |
| Практическое задание № 7 | | |
| по дисциплине «Методы построения и анализа алгоритмов» | | |
| **Представление графа в виде списка смежных вершин** | | |
|  | | |
|  |  | АРИФУЛОВ ТИГРАН |
| Группа ПМ-01 | ДЫЧКО АРСЕНИЙ |
|  | ЖИЖЧЕНКО ЛЕОНИД |
|  | САМСОНОВ СЕМЕН |
|  | ЯКОВЛЕВА ЕЛЕНА |
|  |  |
| Преподаватель | домников пётр александрович |
|  |  |
| Новосибирск, 2021 | | |

# Задания:

1. Дан неориентированный граф (напоминаю, что это не мультиграф и не псевдограф) в виде списка ребер. Представить его в виде списка смежных вершин. Использовать средства библиотеки STL: контейнер vector и файловые потоки.

Формат входного файла: первое число – количество вершин, второе число – количество ребер, далее идут ребра, перечисленные номерами вершин.

Формат выходного файла: количество строк в файле равно количеству вершин в графе. Первое число в строчке – номер вершины, двоеточие, далее идут номера смежных с ней вершин, отсортированные по возрастанию номеров.

Нумерация вершин в файлах начинается с 1, а внутри списков смежности – с 0.

1. Объединение графов. В файлах с именами 1.txt, 2.txt, 3.txt и так далее (не менее трех графов, причем количество входных файлов считывается из отдельного файла n.txt) хранятся неориентированные графы в следующем формате. Первое число – количество вершин, далее перечислены номера вершин, которые связаны по полному графу. Множества вершин из разных файлов должны пересекаться. Требуется построить граф, являющийся объединением всех этих графов и записать его в файл в формате, описанном в предыдущем задании. Использовать контейнер vector и файловые потоки.
2. Совместимость массивов и контейнеров (дополнение к предыдущему практическому заданию). Ниже приведена программа вычисления суммы элементов матрицы (n строк, m столбцов) двумя способами: на основе массива double\* и на основе контейнера vector<vector<double>>. Запустить программу при различных значениях n и m. Заполнить таблицу. В каком случае значения совпадают, верны/неверны? Объяснить результаты.

|  |  |  |  |
| --- | --- | --- | --- |
| n | m | SumVectorVector | Sum |
| 1 | 1 |  |  |
| 1 | 10 |  |  |
| 2 | 1 |  |  |
| 2 | 10 |  |  |
| 10 | 10 |  |  |

# Текст программы:

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#define pb push\_back

using namespace std;

void Task1()

{

constexpr char inputFilePath[] = "Task1/input.txt";

constexpr char outputFilePath[] = "Task1/output.txt";

int size\_n, size\_v;

ifstream in; in.open(inputFilePath);

if (!in.is\_open()) throw exception("Can't open input file.");

in >> size\_n >> size\_v;

vector<vector<int>> graph(size\_n);

int n\_first, n\_second;

for (size\_t i = 0; i < size\_v; ++i)

{

in >> n\_first >> n\_second;

graph[n\_first - 1].pb(n\_second - 1);

graph[n\_second - 1].pb(n\_first - 1);

}

in.close();

ofstream out; out.open(outputFilePath);

for (size\_t i = 0; i < graph.size(); ++i)

{

out << "[" << i + 1 << "] :";

for (auto k : graph[i])

{

out << " " << k + 1;

}

out << endl;

}

}

void Task2()

{

constexpr char inputFilePath[] = "Task2/n.txt";

ifstream in; in.open(inputFilePath);

int N;

in >> N;

in.close();

vector<vector<int>> g(1);

int mx\_sz = 1;

for (int id = 1; id <= N; ++id)

{

string path = "Task2/" + to\_string(id) + ".txt";

in.open(path);

if (!in.is\_open()) throw exception("Can't open input file.");

int cnt;

in >> cnt;

vector < int > nm;

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

{

int x;

in >> x;

mx\_sz = max(mx\_sz, x);

nm.pb(x - 1);

}

in.close();

g.resize(mx\_sz);

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

{

for (int j = i + 1; j < cnt; ++j)

{

g[nm[i]].pb(nm[j]);

g[nm[j]].pb(nm[i]);

}

}

}

constexpr char outputFilePath[] = "Task2/output.txt";

ofstream out; out.open(outputFilePath);

for (size\_t i = 0; i < g.size(); ++i)

{

out << "[" << i + 1 << "] :";

vector < bool > was(mx\_sz, false);

for (auto k : g[i])

{

if (!was[k])

{

out << " " << k + 1;

was[k] = true;

}

}

out << endl;

}

}

double Sum(int n, double\* a)

{

double s = 0;

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

{

s += a[i];

}

return s;

}

double SumVectorVector(vector< vector<double> >& a)

{

double s = 0;

int sz1 = a.size();

for (int i = 0; i < sz1; i++)

{

int sz2 = a[i].size();

for (int j = 0; j < sz2; j++)

{

s += a[i][j];

}

}

return s;

}

void TestSumMatrix(int n, int m)

{

vector< vector<double> > a;

a.resize(n, vector<double>(m, 1));

cout << SumVectorVector(a) << '\t' << Sum(n \* m, &a[0][0]) << endl;

}

void Task3(void)

{

cout << "N = 1, M = 1 : ";

TestSumMatrix(1, 1);

cout << endl << "N = 1, M = 10 : ";

TestSumMatrix(1, 10);

cout << endl << "N = 2, M = 1 : ";

TestSumMatrix(2, 1);

cout << endl << "N = 2, M = 10 : ";

TestSumMatrix(2, 10);

cout << endl << "N = 10, M = 10 : ";

TestSumMatrix(10, 10);

}

int main(void)

{

Task1();

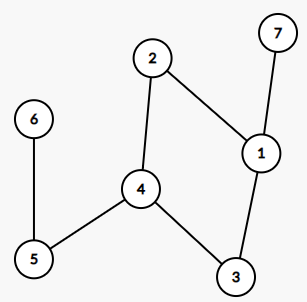
Task2();

Task3();

}

# Результат работы программы:

## Задание 1:

Входные данные:

7 7

1 2

1 3

2 4

4 5

6 5

4 3

1 7

Выходные данные:

[1] : 2 3 7

[2] : 1 4

[3] : 1 4

[4] : 2 5 3

[5] : 4 6

[6] : 5

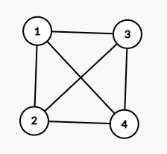
[7] : 1

## Задание 2:

Входные данные:

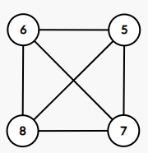
* Файл n.txt:

3

* Файл 1.txt:

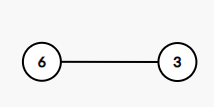
4

1 2 3 4

* Файл 2.txt:

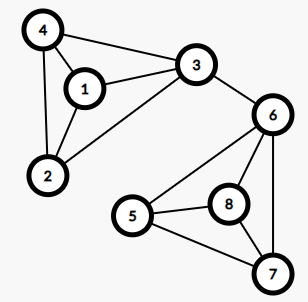
4

5 6 7 8

* Файл 3.txt:

2

3 6

Выходные данные:

[1] : 2 3 4

[2] : 1 3 4

[3] : 1 2 4 6

[4] : 1 2 3

[5] : 6 7 8

[6] : 5 7 8 3

[7] : 5 6 8

[8] : 5 6 7

## Задание 3:

|  |  |  |  |
| --- | --- | --- | --- |
| n | m | SumVectorVector | Sum |
| 1 | 1 | 1 | 1 |
| 1 | 10 | 10 | 10 |
| 2 | 1 | 2 | 1 |
| 2 | 10 | 20 | 10 |
| 10 | 10 | 100 | -7.84591e+298 |

При создании вектора векторов, данные внутри векторов не хранятся последовательно в памяти друг за другом, они находятся в разных участках памяти. Поэтому при попытке считать их последовательно, мы можем считать только *m* элементов первого массива, после которых скорее всего начинается либо чужая, либо неиспользуемая память компьютера. Следовательно, верный результат мы смогли получить только при *n = 1*, пока суммирование происходило только по одному последовательно хранящемуся в памяти массива.

