МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЕТ ПО ИНДИВИДУАЛЬНОМУ ЗАДАНИЮ № 2**

**по дисциплине  
 «Параллельное и низкоуровневое программирование»**

Выполнил студент группы 25/2                                       А.А. Козин

Направление подготовки  02.03.03  Математическое обеспечение и администрирование информационных систем

Курс    2

Отчет принял доктор физико-математических наук, профессор                                                                                       А.И. Миков

Краснодар

2022 г.

**Задача 1.**

Получение матрицы Z расстояний между вершинами графа по его матрице инцидентности B.

**Решение**:

Будем переводить матрицу инцидентности графа в матрицу смежности и с помощью алгоритма Флойда-Уоршелла искать минимальные пути между всеми парами вершин.

**Код программы:**

#define \_SILENCE\_AMP\_DEPRECATION\_WARNINGS

#include "profile.h"

#include <omp.h>

#include <ppl.h>

#include <amp.h>

#include <iostream>

#include <vector>

#include <random>

#include <algorithm>

#define INF INT16\_MAX

#ifdef max

#undef max

#undef min

#endif

vector<vector<bool>> gen\_incidence\_matrix(int32\_t vertex\_num, int32\_t edges\_num) {

vector<vector<bool>> incidence\_matrix(edges\_num, vector<bool>(vertex\_num, false));

random\_device dev;

mt19937 rng(dev());

uniform\_int\_distribution<mt19937::result\_type> edges\_idx(0, edges\_num - 1);

uniform\_int\_distribution<mt19937::result\_type> vertex\_idx(0, vertex\_num - 1);

int32\_t ei, vi;

for (size\_t i = 0; i < edges\_num; i++) {

vi = vertex\_idx(rng);

incidence\_matrix[i][vi] = true;

do {

vi = vertex\_idx(rng);

} while (incidence\_matrix[i][vi]);

incidence\_matrix[i][vi] = true;

}

return incidence\_matrix;

}

vector<vector<int32\_t>> find\_matrix\_of\_min\_routes(const vector<vector<bool>>& inc\_matrix, int32\_t vertex\_num, int32\_t edges\_num) {

vector<vector<int32\_t>> adj\_matrix(vertex\_num, vector<int32\_t>(vertex\_num, 0));

int32\_t idx1, idx2;

for (size\_t i = 0; i < edges\_num; i++) {

idx1 = 0;

idx2 = 0;

for (size\_t j = 0; j < vertex\_num; j++) {

if (inc\_matrix[i][j] && !idx1) {

idx1 = j;

}

else if (inc\_matrix[i][j] && idx1) {

idx2 = j;

break;

}

}

adj\_matrix[idx1][idx2] = true;

adj\_matrix[idx2][idx1] = true;

}

vector<vector<int32\_t>> d(vertex\_num, vector<int32\_t>(vertex\_num, 0));

for (int k = 0; k < vertex\_num; ++k)

for (int i = 0; i < vertex\_num; ++i)

for (int j = 0; j < vertex\_num; ++j)

d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

return d;

}

vector<vector<int32\_t>> find\_matrix\_of\_min\_routes\_omp(const vector<vector<bool>>& inc\_matrix, int32\_t vertex\_num, int32\_t edges\_num) {

vector<vector<int32\_t>> adj\_matrix(vertex\_num, vector<int32\_t>(vertex\_num, 0));

int32\_t idx1, idx2;

#pragma omp parallel for

for (int64\_t i = 0; i < edges\_num; i++) {

idx1 = 0;

idx2 = 0;

for (size\_t j = 0; j < vertex\_num; j++) {

if (inc\_matrix[i][j] && !idx1) {

idx1 = j;

}

else if (inc\_matrix[i][j] && idx1) {

idx2 = j;

break;

}

}

adj\_matrix[idx1][idx2] = true;

adj\_matrix[idx2][idx1] = true;

}

vector<vector<int32\_t>> d(vertex\_num, vector<int32\_t>(vertex\_num, 0));

#pragma omp parallel for shared(d)

for (int k = 0; k < vertex\_num; ++k)

for (int i = 0; i < vertex\_num; ++i)

for (int j = 0; j < vertex\_num; ++j)

d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

return d;

}

vector<vector<int32\_t>> find\_matrix\_of\_min\_routes\_ppl(const vector<vector<bool>>& inc\_matrix, int32\_t vertex\_num, int32\_t edges\_num) {

vector<vector<int32\_t>> adj\_matrix(vertex\_num, vector<int32\_t>(vertex\_num, 0));

int32\_t idx1, idx2;

concurrency::parallel\_for((size\_t)0, (size\_t)edges\_num, [&](size\_t i) {

idx1 = 0;

idx2 = 0;

for (size\_t j = 0; j < vertex\_num; j++) {

if (inc\_matrix[i][j] && !idx1) {

idx1 = j;

}

else if (inc\_matrix[i][j] && idx1) {

idx2 = j;

break;

}

}

adj\_matrix[idx1][idx2] = true;

adj\_matrix[idx2][idx1] = true;

});

vector<vector<int32\_t>> d(vertex\_num, vector<int32\_t>(vertex\_num, 0));

concurrency::parallel\_for((size\_t)0, d.size(), [&](size\_t k) {

for (int i = 0; i < vertex\_num; ++i)

for (int j = 0; j < vertex\_num; ++j)

d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

});

return d;

}

vector<int32\_t> find\_matrix\_of\_min\_routes\_amp(const vector<vector<bool>>& inc\_matrix, int32\_t vertex\_num, int32\_t edges\_num) {

vector<vector<int32\_t>> adj\_matrix(vertex\_num, vector<int32\_t>(vertex\_num, 0));

int32\_t idx1, idx2;

for (size\_t i = 0; i < edges\_num; i++) {

idx1 = 0;

idx2 = 0;

for (size\_t j = 0; j < vertex\_num; j++) {

if (inc\_matrix[i][j] && !idx1) {

idx1 = j;

}

else if (inc\_matrix[i][j] && idx1) {

idx2 = j;

break;

}

}

adj\_matrix[idx1][idx2] = true;

adj\_matrix[idx2][idx1] = true;

}

vector<int32\_t> d(vertex\_num \* vertex\_num, 0);

concurrency::array\_view<int, 2> d\_view(vertex\_num, vertex\_num, d);

d\_view.discard\_data();

concurrency::parallel\_for\_each(d\_view.extent,

[=](concurrency::index<2> idx) restrict(amp) {

int row = idx[0];

int col = idx[1];

for (int inner = 0; inner < idx[0]; inner++) {

if (d\_view[idx] < d\_view(row, inner) + d\_view(inner, col)) {

d\_view[idx] = d\_view(row, inner) + d\_view(inner, col);

}

}

});

d\_view.synchronize();

return d;

}

int main() {

int32\_t vertex\_num, edges\_num;

cin >> vertex\_num >> edges\_num;

auto im = gen\_incidence\_matrix(vertex\_num, edges\_num);

vector<vector<int32\_t>> mr;

vector<int32\_t> mr\_;

{

LOG\_DURATION("CPU single core computition");

mr = find\_matrix\_of\_min\_routes(im, vertex\_num, edges\_num);

}

{

LOG\_DURATION("CPU OpenMP multithreading computition");

mr = find\_matrix\_of\_min\_routes\_omp(im, vertex\_num, edges\_num);

}

{

LOG\_DURATION("CPU PPL multithreading computition");

mr = find\_matrix\_of\_min\_routes\_ppl(im, vertex\_num, edges\_num);

}

{

LOG\_DURATION("GPU computition");

auto mr\_ = find\_matrix\_of\_min\_routes\_amp(im, vertex\_num, edges\_num);

}

return 0;

}

**Пример вывода программы в консоль:**

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

**Задача 2.**

Создать классическое приложение Windows в Visual Studio. С использованием WinAPI описать и показать в окне параллельные процессы в следующей задаче:

**Шаттл (автобус в аэропорту). Едет по замкнутому маршруту (4 остановки). На каждой остановке часть людей выходит (случайное количество). На каждой остановке имеется очередь пассажиров случайной длины. Шаттл забирает людей на свободные места, едет дальше. Неограниченно.**

В описание включить вывод в графическое окно на экране состояний графических объектов задачи с частотой не менее 30 кадров в секунду.

**Решение:**

Создадим классы Bus – для хранения координат, скорости и размеров автобуса.

Client – класс, хранящий аналогичные данные для каждого пассажира.

Queue – для хранения положения очереди и обработки добавления удаления пассажиров в очередь на остановке.

Автобус, по заданию, будет ходить по кругу останавливаясь на каждой остановке. На каждой остановке случайное число людей могут сесть и выйти из автобуса. При это обновляется счётчик количества людей в автобусе, чтобы из него не вышли несуществующие люди. После посадки/высадки пассажиров автобус продолжает движение до следующей остановки.

Полный код программы в приложенном архиве.

Если не все картинки будут работать перекиньте их прямо в С диск C:\img.png.

**Задача 3.**

Создать классическое приложение Windows в Visual Studio. С использованием WinAPI описать и показать в окне параллельные процессы в следующей задаче:

**Защитник. Компьютерный процесс симулирует нападающего, который хочет забить мяч в ворота. Второй компьютерный процесс симулирует вратаря. Чем ближе нападающий к воротам, тем больше вероятность того, что вратарь не справится. Человек за клавиатурой (или мышью) симулирует защитника, старающегося отобрать мяч у нападающего.**

В описание включить вывод в графическое окно на экране состояний графических объектов задачи с частотой не менее 30 кадров в секунду.

**Решение:**

Зададим переменные для хранения положения и скоростей игроков на поле и мяча.

Для мяча пропишем коллизии для отскока от краёв поля и от игроков.

После гола в ворота мяч появляется на центре поля.

Защитники на воротах – компьютерные процессы.

Игрок – нападающий одной из команд, передвигающийся с помощью клавиш WASD.

Полный код программы в приложенном архиве.