

# **Правительство Российской Федерации**

Федеральное государственное автономное образовательное учреждение  
высшего образования «Национальный исследовательский университет «Высшая  
школа экономики»

Факультет компьютерных наук

## **Пояснительная записка заданию №3**

**По дисциплине**

**“Архитектура вычислительных систем”**

Исполнитель  
Студентка группы БПИ194  
А.И. Гиразутдинова  
Вариант 6

Москва 2020

## Оглавление

1. Постановка задачи.....	3
2. Уточнение задачи.....	3
3. Теоретическая справка.....	4
4. Реализация программы .....	5
5. Тестирование программы .....	7
6. Заключение .....	14
7. Использованная литература.....	15
Приложение.....	16

## **1. Постановка задачи**

- 1) Вычислить прямое произведение множеств  $A_1, A_2, A_3, A_4$ . Входные данные: множества чисел  $A_1, A_2, A_3, A_4$ , мощности множеств могут быть не равны между собой и мощность каждого множества больше или равна 1. Количество потоков является входным параметром.
- 2) Вывести результат алгоритма в консоль.

## **2. Уточнение задачи**

- 1) Реализовать программу, написанную на языке C++ 14.0, которая будет принимать на вход 4 подмножества множества целых чисел, мощности которых могут быть не равны между собой и мощность каждого множества больше или равна 1. Количество потоков является входным параметром.
- 2) Разработать программу с применением функций библиотеки POSIX Threads или стандартной библиотеки C++ и протестировать ее на нескольких примерах.
- 3) Формат входных и выходных данных на усмотрение разработчика.

### 3. Теоретическая справка

Прямое, или **декартово произведение** двух множеств — множество, элементами которого являются все возможные упорядоченные пары элементов исходных множеств. [1]

$$A_1 \times \dots \times A_n = \left\{ (a_{1_{i_1}}, \dots, a_{n_{i_n}}), a_{1_{i_1}} \in A_1, \dots, a_{n_{i_n}} \in A_n \right\}$$

**Итеративный параллелизм** используется для реализации нескольких процессов (часто идентичных), каждый из которых содержит циклы. Процессы программы, являющиеся итеративными программами, работают совместно над решением одной задачи; они взаимодействуют и синхронизируются либо с помощью разделяемых переменных, либо передачей сообщений. Такой параллелизм характерен, прежде всего, синхронизируемых параллельных вычислений.[2, стр. 21]

Если программа содержит большое число операций, которые могут выполняться параллельно, то такие алгоритмы обычно называют *алгоритмами с массовым параллелизмом*. [2, стр. 24]

## 4. Реализация программы

### 4.1. Выбор модели построения

Наиболее подходящая для этой программы модель построения многопоточных приложений – итеративная, так как потоки работают совместно над решением одной и той же задачи (нахождение прямого произведения множеств) и каждый из потоков выполняет итеративную функцию в виде вычисления элементов прямого произведения множеств.

### 4.2. Алгоритм

Имеем 4 множества:  $A, B, C, D$  (они же  $A_1, A_2, A_3, A_4$ ). Количество элементов в прямом произведении этих множеств  $\text{number\_of\_elems} = |A| \times |B| \times |C| \times |D|$ . Если  $\text{number\_of\_threads}$  – количество потоков, то  $\text{number\_of\_elems} \div \text{number\_of\_threads} = \text{num\_of\_operations\_per\_thread}$  – количество элементов, которые нужно вычислить каждому потоку<sup>1</sup>.

(\*) То есть первый поток вычисляет элементы с 0 по  $\text{num\_of\_operations\_per\_thread} - 1$ , второй – с  $\text{num\_of\_operations\_per\_thread}$  по  $\text{num\_of\_operations\_per\_thread} * 2 - 1$ , третий – с  $\text{num\_of\_operations\_per\_thread} * 2$  по  $\text{num\_of\_operations\_per\_thread} * 3 - 1$  и т. д.

Для вычисления элементов используется функция с 4 вложенными друг в друга циклами for: for1 – для перебора элементов из множества  $A$ , for2 – для перебора элементов из множества  $B$ , for3 – для перебора элементов из множества  $C$ , for4 – для перебора элементов из множества  $D$ .

Все элементы прямого произведения можно расположить по порядку:

$$\begin{array}{c} a_1, b_1, c_1, d_1 \\ a_1, b_1, c_1, d_2 \\ \dots \\ a_1, b_1, c_1, d_n \\ a_1, b_1, c_2, d_1 \\ a_1, b_1, c_2, d_2 \\ \dots \\ a_1, b_1, c_2, d_n \\ a_1, b_1, c_3, d_1 \\ \dots \\ a_1, b_1, c_3, d_n \\ \dots \\ a_1, b_1, c_m, d_n \\ a_1, b_2, c_1, d_1 \\ \dots \\ a_k, b_l, c_m, d_n \end{array}$$

, где  $k = |A|$ ,  $l = |B|$ ,  $m = |C|$ ,  $n = |D|$ ,  $a_i, b_i, c_i, d_i$  – элементы соответствующих множеств.

---

<sup>1</sup> Кроме последнего, так как  $\text{number\_of\_elems}$  не всегда может быть кратно  $\text{num\_of\_threads}$  и тогда последний поток должен вычислить все оставшиеся элементы

Следовательно, для каждого потока должны задаваться начальные значения итераторов циклов for1, for2, for3, for4, чтобы начинать вычисление с определенного элемента. Чтобы найти значения итераторов достаточно знать с какого по счету элемента поток должен вычислять элементы (см. (\*)). Далее вычисляем ind1 – начальное значение итератора for1, ind2 – начальное значение итератора for2, ind3 – начальное значение итератора for3, ind4 – начальное значение итератора for4:

Допустим, поток вычисляет элементы с  $h$ -го по  $h + \text{num\_of\_operations\_per\_thread} - 1$  включительно, тогда используется следующий алгоритм для вычисления начальных значений итераторов, с которых в функции начинается перебор элементов:

```
ind4 = h % n
h /= n
ind3 = h % m
h /= m
ind2 = h % l
h /= l
ind1 = h % k
```

, где  $k = |A|$ ,  $l = |B|$ ,  $m = |C|$ ,  $n = |D|$

Таким образом каждый поток вычисляет равное количество элементов (кроме последнего потока) и записывает элементы в общий двумерный вектор.

### 4.3.Реализация алгоритма

Для реализации потоков была выбрана стандартная библиотека C++.

Организация входных и выходных данных

Входные данные должны быть представлены в виде файла с данными. Результат выполнения также записывается в файл. Пользователь вводит в консоль путь к input и output файлам.

Требование к input файлу:

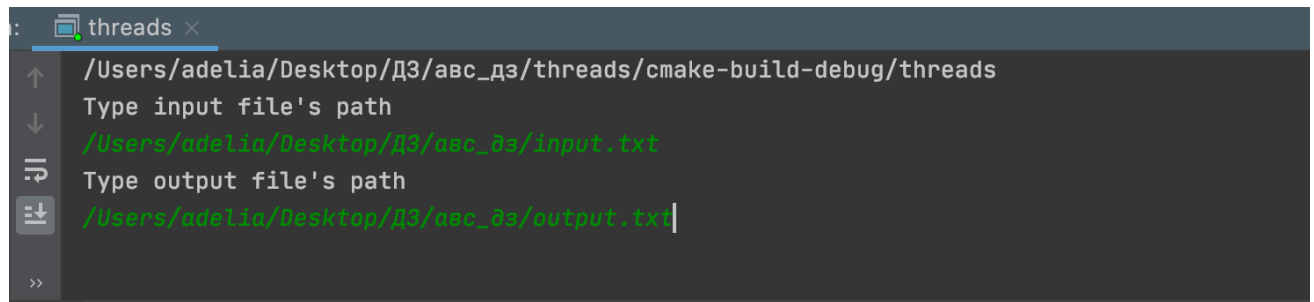
- В первой строке файла – элементы множества  $A$  через пробел;
- Во второй строке файла – элементы множества  $B$  через пробел;
- В третьей строке файла – элементы множества  $C$  через пробел;
- В четвертой строке файла – элементы множества  $D$  через пробел;
- В пятой строке файла число `num_of_operations_per_thread` – количество потоков;

Данные должны быть корректны (однако есть проверка на произвольные символы в строках и количество строк).

Сам код программы предоставлен в Приложении.

## 5. Тестирование программы

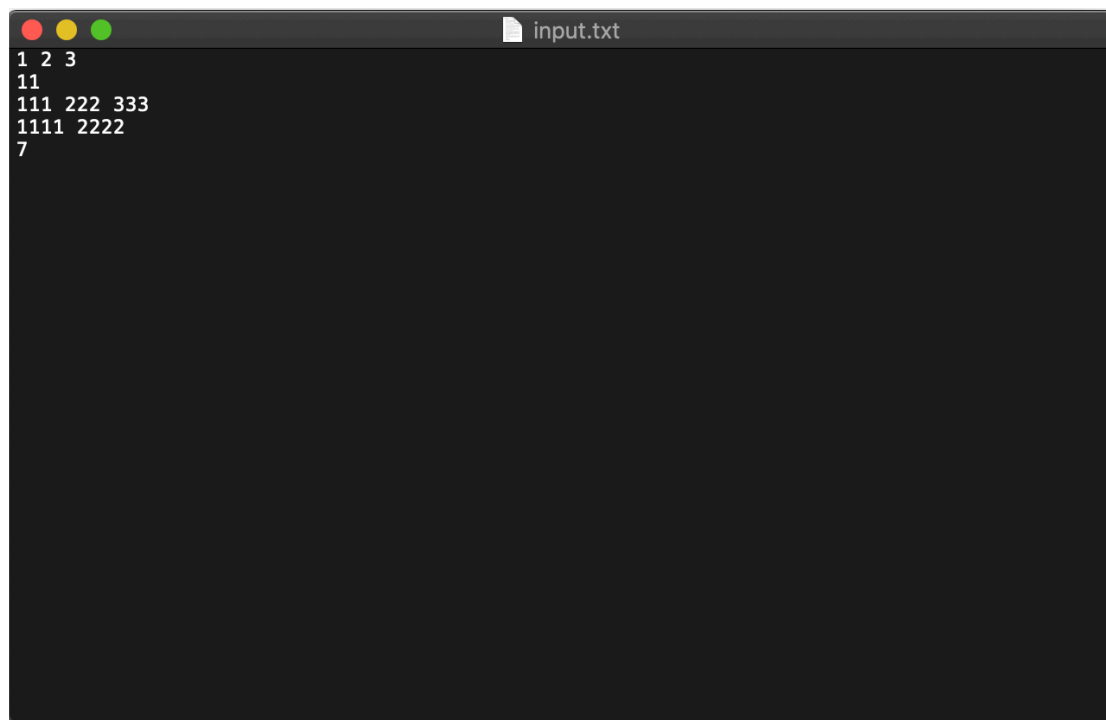
1) Тест на проверку работы программы с корректными данными



```
threads x
/Users/adelia/Desktop/ДЗ/авс_дз/threads/cmake-build-debug/threads
Type input file's path
/Users/adelia/Desktop/ДЗ/авс_дз/input.txt
Type output file's path
/Users/adelia/Desktop/ДЗ/авс_дз/output.txt
```

Рисунок 1 – ввод в консоль

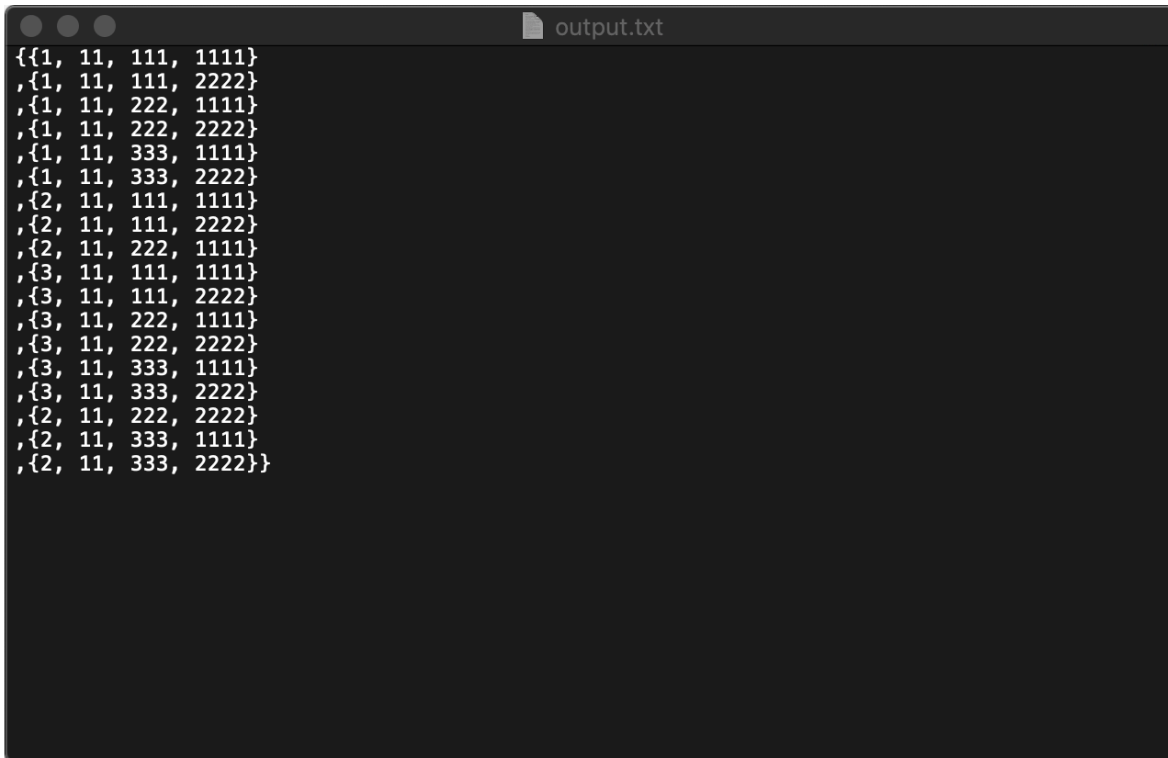
На рисунке 1 показан ввод пользователем путей к файлам input.txt – для считывания входных данных и output.txt – для записи результата.



```
input.txt
1 2 3
11
111 222 333
1111 2222
7
```

Рисунок 2 – содержимое файла input.txt

На рисунке 2 показаны корректные входные данные, которые программа будет обрабатывать.

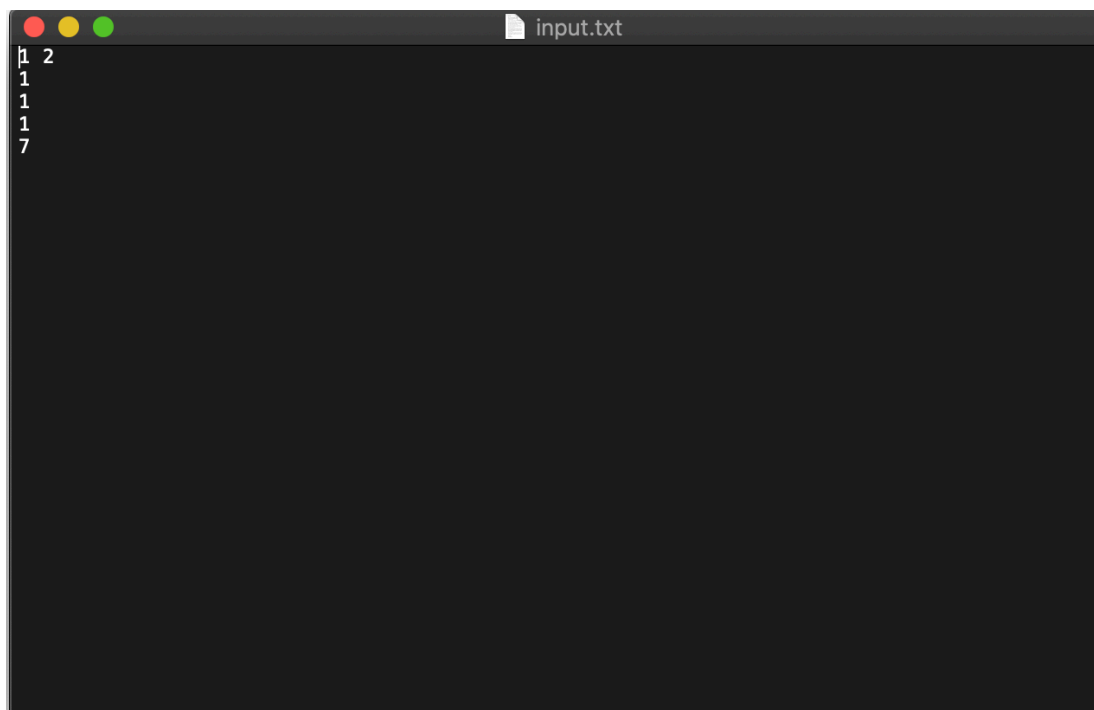


```
output.txt
{{1, 11, 111, 1111}
,{1, 11, 111, 2222}
,{1, 11, 222, 1111}
,{1, 11, 222, 2222}
,{1, 11, 333, 1111}
,{1, 11, 333, 2222}
,{2, 11, 111, 1111}
,{2, 11, 111, 2222}
,{2, 11, 222, 1111}
,{3, 11, 111, 1111}
,{3, 11, 111, 2222}
,{3, 11, 222, 1111}
,{3, 11, 222, 2222}
,{3, 11, 333, 1111}
,{3, 11, 333, 2222}
,{2, 11, 222, 2222}
,{2, 11, 333, 1111}
,{2, 11, 333, 2222}}
```

Рисунок 3 – содержимое файла output.txt после выполнения программы

На рисунке 3 показан результат выполнения программы. На рисунке 18 элементов, что корректно, так как  $3 \cdot 1 \cdot 3 \cdot 2 = 18$ , 3, 1, 3, 2 – мощности данных множеств.

## 2) Тест на проверку программы при $\text{number\_of\_elems} < \text{number\_of\_threads}$



```
input.txt
1 2
1
1
1
7
```

Рисунок 4 – содержимое файла input.txt



```
/Users/adelia/Desktop/ДЗ/авс_дз/threads/cmake-build-debug/threads
Type input file's path
/Users/adelia/Desktop/ДЗ/авс_дз/input.txt
Type output file's path
/Users/adelia/Desktop/ДЗ/авс_дз/output.txt
Too many threads
The program will use only 2 threads
Process finished with exit code 0
```

Рисунок 5 – информация в консоли при выполнении программы

На рисунке 4 показан файл с корректными данными, однако  $7 > 2*1*1*1$ , то есть заданное количество потоков больше максимального количества. Тогда программа использует максимальное количество потоков, а остальные отбрасывает. В консоли это также транслируется, что доказывает рисунок 5.

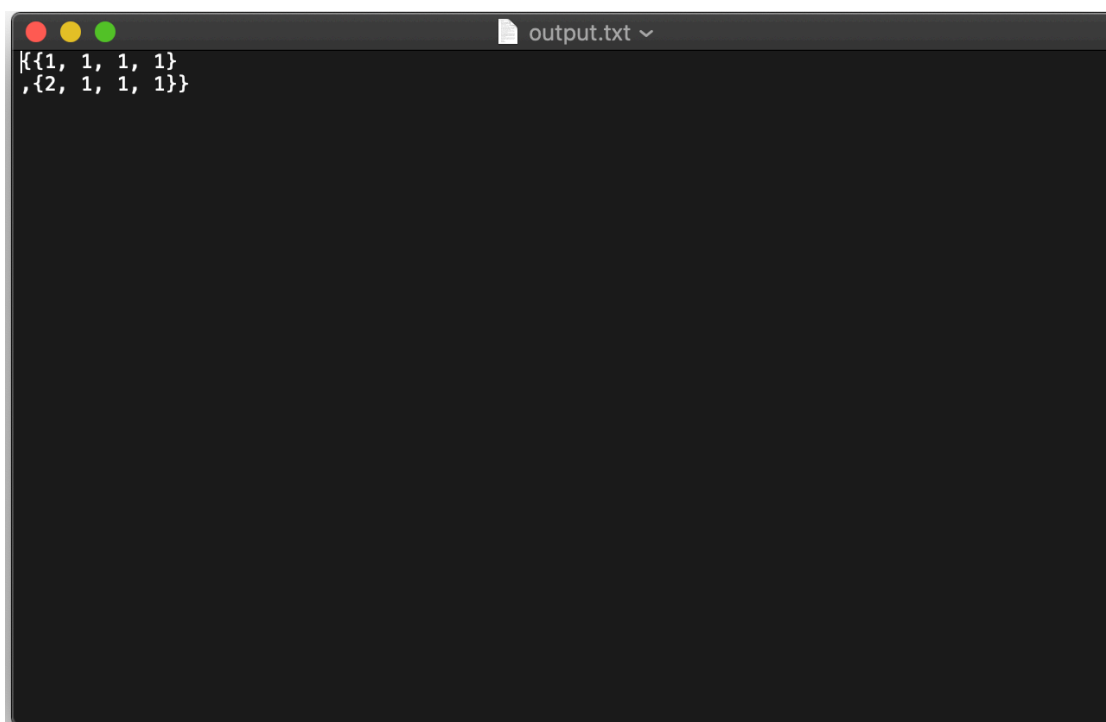


Рисунок 6 – содержимое файла output.txt после выполнения программы

На рисунке 6 показан корректный результат работы программы.

### 3) Тесты на реакцию программы на некорректные данные

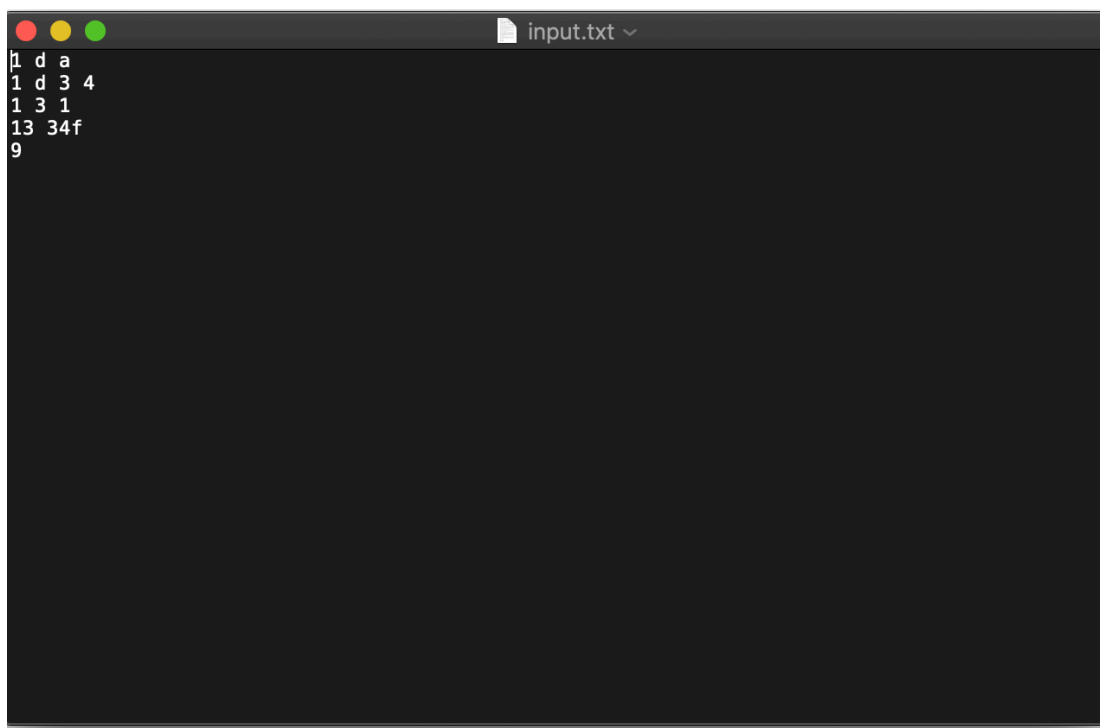


Рисунок 7 – содержимое файла input.txt с некорректным форматом данных

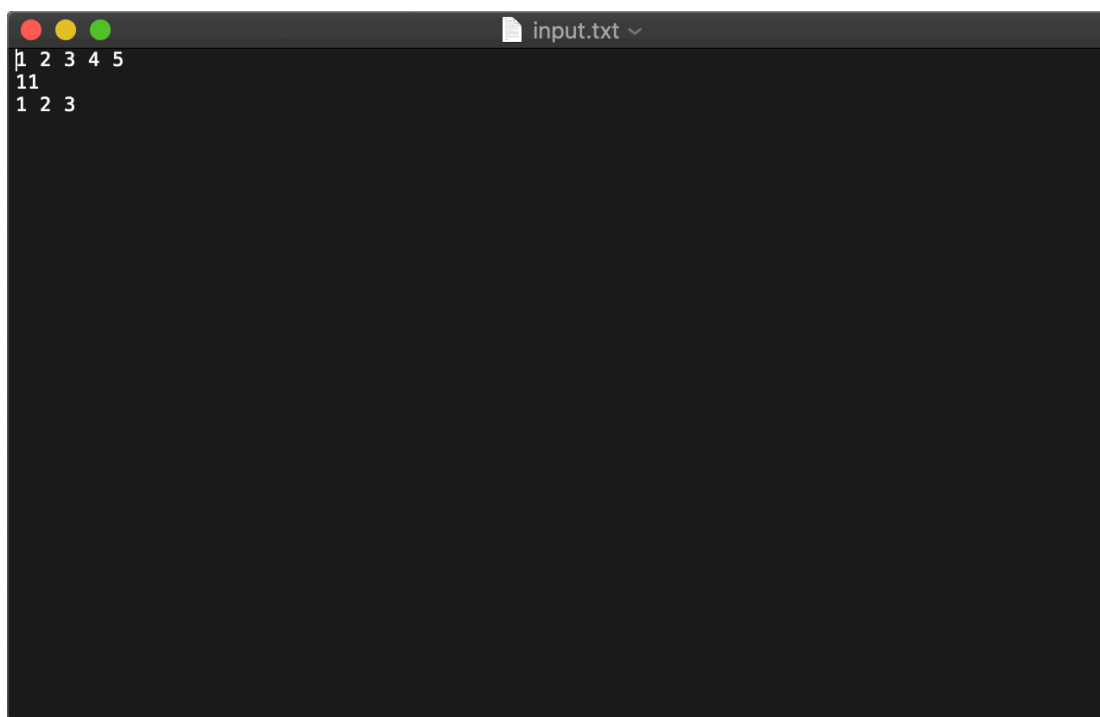


Рисунок 8 – содержимое файла input.txt с некорректным форматом данных

На рисунке 7 показан файл с некорректными входными данными: на некоторых позициях стоят латинские буквы вместо чисел. На рисунке 8 показан файл с некорректными входными данными: в файле 3 строки вместо требуемых 5.

```
/Users/adelia/Desktop/Д3/авс_д3/threads/cmake-build-debug/threads
Type input file's path
/Users/adelia/Desktop/Д3/авс_д3/input.txt
Type output file's path
/Users/adelia/Desktop/Д3/авс_д3/output.txt
Error while reading the file.
Check the correctness of input data
input data in file:
A1 A2 ... An
B1 B2 ... Bm
C1 C2 ... Ck
D1 D2 ... Dl
<num_of_threads>
where
Ai - elements of the first multiplicity (integer)
Bi - elements of the second multiplicity (integer)
Ci - elements of the third multiplicity (integer)
Di - elements of the fourth multiplicity (integer)
<num_of_threads> - number of threads (> 0)

Process finished with exit code 11
|
```

Рисунок 9 – реакция программы на некорректные данные

На рисунке 9 показано, как программа реагирует на файлы с некорректными данными (рисунки 7-8): в консоль выводится сообщение об ошибке и повторяется требуемый формат входных данных в файле.

В файл output.txt при этом ничего не записывается.

#### 4) Тест на нулевое количество потоков

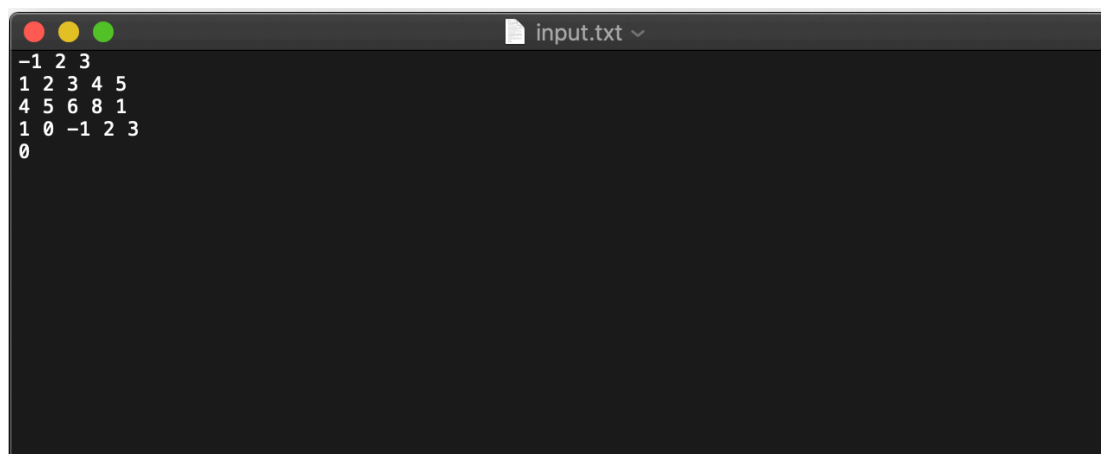


Рисунок 10 – содержимое файла input.txt с параметром number\_of\_threads = 0

```
/Users/adelia/Desktop/ДЗ/авс_дз/threads/cmake-build-debug/threads
Type input file's path
/Users/adelia/Desktop/ДЗ/авс_дз/input.txt
Type output file's path
/Users/adelia/Desktop/ДЗ/авс_дз/output.txt
Number of threads must be > 0
Process finished with exit code 0
|
```

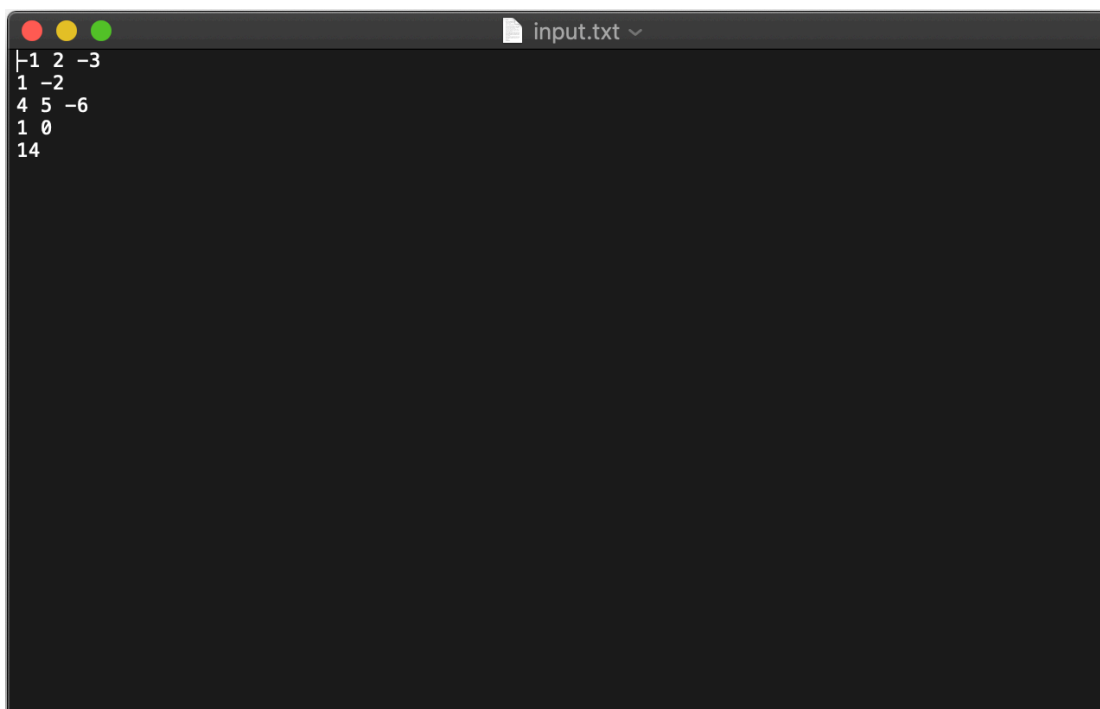
Рисунок 11 – реакция программы на number\_of\_threads = 0

На рисунке 10 показаны входные данные с количеством потоков = 0, что недопустимо в данной программе. На рисунке 11 показана реакция программы на данный входной параметр. Пользователю выводится сообщение о недопустимости такого значения.

В файл output.txt при этом ничего не записывается.

#### 5) Тест на отрицательные целые числа в множествах

Ввод путей к файлам в консоль как на рисунке 1.



```
input.txt
1 2 -3
1 -2
4 5 -6
1 0
14
```

Рисунок 12 – содержимое файла input.txt с корректными данными

На рисунке 12 показан файл с корректными входными данными, некоторые элементы множеств отрицательные.

```
output.txt
{{-1, 1, 5, 1}
,{-1, 1, -6, 1}
,{-1, 1, -6, 0}
,{-1, 1, 4, 1}
,{-1, -2, 4, 1}
,{-1, -2, 4, 0}
,{-1, 1, 4, 0}
,{-1, -2, 5, 1}
,{-1, -2, 5, 0}
,{-1, 1, 5, 0}
,{-1, -2, -6, 1}
,{-1, -2, -6, 0}
,{2, 1, 4, 1}
,{2, 1, 4, 0}
,{2, 1, 5, 1}
,{2, 1, -6, 1}
,{2, 1, -6, 0}
,{2, 1, 5, 0}
,{2, -2, 4, 1}
,{2, -2, 4, 0}
,{2, -2, 5, 1}
,{2, -2, 5, 0}
,{2, -2, -6, 1}
,{2, -2, -6, 0}
,{-3, 1, 4, 1}
,{-3, 1, 4, 0}
,{-3, 1, 5, 1}
,{-3, 1, 5, 0}
,{-3, 1, -6, 1}
,{-3, 1, -6, 0}
,{-3, -2, 4, 1}
,{-3, -2, 4, 0}
,{-3, -2, 5, 1}
,{-3, -2, 5, 0}
,{-3, -2, -6, 1}
,{-3, -2, -6, 0}}
```

Рисунок 13 – содержимое файла output.txt после выполнения программы

На рисунке 13 показан результат выполнения программы. На рисунке 36 элементов, что корректно, так как  $3 \cdot 2 \cdot 3 \cdot 2 = 36$ , 3, 2, 3, 2 – мощности данных множеств.

## **6. Заключение**

В результате выполнения работы был реализован алгоритм, вычисляющий прямое произведение множеств  $A_1, A_2, A_3, A_4$  и проверенный на корректность.

## 7. Использованная литература

- 1) <https://ru.wikipedia.org> [Электронный ресурс]/ wikipedia.org Режим доступа: [https://ru.wikipedia.org/wiki/Прямое\\_произведение](https://ru.wikipedia.org/wiki/Прямое_произведение) , свободный (Дата обращения: 15.11.2020).
- 2) Кареева Е. Д., Кузьмин Д. А., Легалов А.И., Редькин А. В., Удалова Ю. В., Федоров Г. А. Средства разработки параллельных программ. Учебное пособие. Красноярск 2007

## Приложение

```
/*
 * Вычислить прямое произведение множеств A1, A2, A3, A4.
 * Входные данные: множества чисел A1, A2, A3, A4,
 * мощности множеств могут быть не равны между собой и мощность
 * каждого множества больше или равна 1. Количество потоков является входным
 * параметром. входные параметры: файл вида
 *
 * A1 A2 ... An
 * B1 B2 ... Bm
 * C1 C2 ... Ck
 * D1 D2 ... Dl
 * <num_of_threads>
 *
 * где
 * Ai - элементы первого множества, целые
 * Bi - элементы второго множества, целые
 * Ci - элементы третьего множества, целые
 * Di - элементы четвертого множества, целые
 * <num_of_threads> - количество потоков, целые
 */
#include <algorithm>
#include <exception>
#include <fstream>
#include <iostream>
#include <mutex>
#include <sstream>
#include <thread>
#include <vector>

using namespace std;
mutex mtx; //мьютекс для избегания ситуаций неуправляемого изменения одних и тех
           //же общих данных несколькими потоками
vector<vector<int>>> vecs; //вектор, состоящий из данных векторов A, B, C, D
vector<thread> threads;   //вектор потоков
int num_of_threads;       //количество потоков

/// Метод для вычисления заданного количества элементов прямого
/// произведения данных множеств, начиная с определенного момента
/// \param a - индекс множества A, с которого нужно начинать вычисление
/// \param b - индекс множества B, с которого нужно начинать вычисление
/// \param c - индекс множества C, с которого нужно начинать вычисление
/// \param d - индекс множества D, с которого нужно начинать вычисление
/// \param num - заданное количество элементов, которое нужно вычислить
/// \param result - вектор, в который нужно записать полученные элементы
void mult(int a, int b, int c, int d, int num, vector<vector<int>>> &result) {
    int ind1 = a, ind2 = b, ind3 = c, ind4 = d;
    for (int i = ind1; i < vecs[0].size(); ++i, ind2 = 0) {
        for (int j = ind2; j < vecs[1].size(); ++j, ind3 = 0) {
            for (int k = ind3; k < vecs[2].size(); ++k, ind4 = 0) {
                for (int l = ind4; l < vecs[3].size(); ++l) {
                    if (num == 0) //если num элементов вычислилось, то выход
                    {
                        return;
                    }
                    mtx.lock();
                    result.push_back({vecs[0][i], vecs[1][j], vecs[2][k], vecs[3][l]});
                    mtx.unlock();
                }
            }
        }
    }
}
```



```

        num--;
    }
}
}
}

/// Метод для вычисления индексов элементов множеств, начиная с которых нужно
/// начать вычисление элементов прямого произведения \param num - число
/// элементов, которое уже было вычислено \return - вектор длины 4 с индексами
vector<int> find_begin_inds(int num) {
    vector<int> new_num;
    new_num.push_back(num % vecs[3].size()); //индекс 4 множества
    num /= vecs[3].size();
    new_num.push_back(num % vecs[2].size()); //индекс 3 множества
    num /= vecs[2].size();
    new_num.push_back(num % vecs[1].size()); //индекс 2 множества
    num /= vecs[1].size();
    new_num.push_back(num % vecs[0].size()); //индекс 1 множества
    reverse(new_num.begin(),
            new_num.end()); //отзеркалить, чтобы индексы были от 1 до 4 множества
    return new_num;
}

/// Метод переводящий строку в число
/// \param line - строка
/// \param num - число, в которое нужно записать строку
/// \return - 1, если конвертация прошла успешно, иначе - 0
bool string_to_int(string line, int &num) {
    for (int i = 0; i < line.size(); ++i) {
        if (line[i] > '9' ||
            line[i] < '0') //если в строке что-то кроме цифр, то это не число
            if (line[i] == '-' && i == 0) //если число отрицательное
                continue;
            else
                return 0;
        num = atoi(line.c_str());
        return 1;
    }
}

/// Метод для считывания входных данных из файла
/// \param path - путь к файлу
/// \param vecs - вектор, в который нужно записать полученные множества
void read_data(string path, vector<vector<int>> &vecs) {
    try {
        ifstream in(path);
        string line{};
        string number{};
        int ind = -1; //индекс множества ([0;3])
        while (getline(in, line)) {
            stringstream strStream(line);
            ind++;
            while (getline(strStream, number, ' ')) { //разделение через пробел
                int num;
                if (!string_to_int(number, num)) { //если в файле есть другие символы
                                                    //кроме цифр, то он неверный
                    throw exception();
                }
                vecs[ind].push_back(atoi(number.c_str()));
            }
        }
        if (ind == 3) //если считано 4 множества
    }
}

```

```

        break;
    }
    for (int i = 0; i < vecs.size(); ++i) {
        if (vecs[i].size() == 0)
            throw exception();
    }

    string k;
    in >> k; //считывание количества потоков
    num_of_threads = atoi(k.c_str());
    in.close();
} catch (const std::exception &err) {
    num_of_threads = 0;
    vecs = vector<vector<int>>(0);
    cout << "Error while reading the file." << endl;
    cout << "Check the correctness of input data" << endl;
    cout << "input data in file: " << endl
        << "A1 A2 ... An" << endl
        << "B1 B2 ... Bm" << endl
        << "C1 C2 ... Ck" << endl
        << "D1 D2 ... Dl" << endl
        << "<num_of_threads>" << endl
        << "where" << endl
        << "Ai - elements of the first multiplicity (integer)" << endl
        << "Bi - elements of the second multiplicity (integer)" << endl
        << "Ci - elements of the third multiplicity (integer)" << endl
        << "Di - elements of the fourth multiplicity (integer)" << endl
        << "<num_of_threads> - number of threads (> 0)" << endl;
}
}

/// Метод для записи в файл результата выполнения программы
/// \param path - путь к файлу
/// \param result - прямое произведение множеств
void write_res(string path, vector<vector<int>> result) {
    try {
        ofstream out(path);
        if (out.is_open()) {
            out << "{";
            for (int i = 0; i < result.size(); ++i) {
                if (i != 0)
                    out << ",";
                out << "{" << result[i][0] << ", " << result[i][1] << ", "
                    << result[i][2] << ", " << result[i][3] << "}";
                if (i != result.size() - 1)
                    out << endl;
            }
            out << "}";
            out.close();
        }
    } catch (const exception &err) {
        cout << "Error while writing result into the file.";
    }
}

int main() {
    vector<vector<int>> result; //вектор для результата выполнения программы
    vecs = vector<vector<int>>(4); //вектор для хранения данных множеств
    string pathin; //путь для входных данных
    string pathout; //путь для выходных данных
    cout << "Type input file's path" << endl;

```

```

cin >> pathin; //считывание путей
cout << "Type output file's path" << endl;
cin >> pathout; //считывание путей
read_data(pathin, vecs);
//расчет количества элементов в прямом произведении множеств
int result_num =
    vecs[0].size() * vecs[1].size() * vecs[2].size() * vecs[3].size();
//расчет количества вычисляемых элементов прямого произведения для каждого
//потока
int operations_per_thread = result_num / num_of_threads;
//потоков не может быть больше чем элементов в декартовом произведении
if (num_of_threads > result_num) {
    cout << "Too many threads" << endl
        << "The program will use only " << result_num << " threads";
    num_of_threads = result_num;
}
if (num_of_threads <= 0) {
    cout << "Number of threads must be > 0";
    return 0;
}

//цикл с созданием потоков
for (int i = 0; i < num_of_threads; ++i) {
    vector<int> inds = find_begin_inds(operations_per_thread * i);
    try {
        if (i == num_of_threads - 1)
            threads.push_back(thread(mult, inds[0], inds[1], inds[2], inds[3],
                                     result_num - operations_per_thread * i,
                                     ref(result)));
        else
            threads.push_back(thread(mult, inds[0], inds[1], inds[2], inds[3],
                                     operations_per_thread, ref(result)));
    } catch (const std::exception &err) {
        cout << "ERROR";
    }
}
for (auto &th : threads) //ожидание окончания выполнения всех потоков
    th.join();

write_res(pathout, result); //вывод результата в файл

return 0;
}

```