Пояснительная записка к домашнему задания №3
НИУ ВШЭ Департамент программной инженерии
Формулировка задания: «Определить множество индексов i, для которых A[i] и B[i] не имеют общих делителей (единицу в роли делителя не рассматривать). Входные данные: массивы целых положительных чисел A и B, произвольной длины ≥ 1000. Количество потоков является входным параметром.»
Ловчикова Юлия Васильевна, БПИ192, вариант 14

1. Задание

Определить множество индексов i, для которых A[i] и B[i] не имеют общих делителей (единицу в роли делителя не рассматривать). Входные данные: массивы целых положительных чисел A и B, произвольной длины ≥ 1000. Количество потоков является входным параметром.

2. Математическое обоснование и алгоритм

I. Математическое обснование метода, выполняемого каждым потоком

Взаимно простыми называют два натуральных числа, наибольший общий делитель которых равен единице.

Известно множество способов определения наибольшего общего делителя двух натуральных чисел; для реализации был выбран один из классических – быстрый алгоритм Евклида, работающий рекуррентно (математическое обснование и описание алгоритма Евклида в записке не приводится, кроме как указания на соответствующий источник, поскольку считается довольно известным и простым фактом).

Примечание. gcf – greatest common factor (НОД)

$$gcf(x,y) = \begin{cases} x, if\ y = 0 \\ gcf(y, x\ mod\ y)\ otherwise \end{cases}$$

II. Обоснование выбора парадигмы и модели построения многопоточного приложения

Подробнее про все модели многопоточных приложений можно прочитать в источнике, и ссылкой на него была выбрана следующая, поскольку она наиболее близка к описанной задаче:

Взаимодействующие равные — последняя парадигма взаимодействия. Она встречается в распределенных программах, в которых несколько процессов для решения задачи выполняют один и тот же код и обмениваются сообщениями. Взаимодействующие равные используются для реализации распределенных параллельных программ, особенно при итеративном параллелизме и децентрализованном принятии решений в распределенных системах. Программа, основанная на модели взаимодействующих равных, модет быть реализована двумя путями, схематично проиллюстрированных ниже:

а) взаимодействие управляющий-рабочие б) круговой конвейер

Для описанной задачи была выбрана более классическая модель «управляющий рабочие», поскольку потокам не нужно обмениваться никакиеми данными, а только взаимодействовать с управляющим звеном.

Алгоритм координирующего процесса.

```
Для каждого i∈[0,size), где size – количество потоков:
            Отправить i-тому процессу массивы A, B;
            Отправить i-тому процессу index_start, index_finish, где:
                      index_start = i * (size / count_of_threads);
                      index_finish = index_start + (size / count_of_threads),
                                          если это не послежний поток, и
                      index_finish = size, если последний поток;
            Отправить i-тому процессу ref(vector<int> res), куда будет
            записываться результирующее множество.
Для каждого i∈[0,size), где size – количество потоков:
            Принять res.
```

3. Программа

3.1. Список методов:

Метод, реализующий быстрый алгоритм Евклида рекрусивно:

```
int find_gcf(int x, int y) {
	if (y == 0)
		return x;
	else find_gcf(y, x % y);
}
```

Метод, используемый каждым потоком, который меняет результирующий вектор res:

```
void count_coprime_numbers(std::vector<int> A, std::vector<int>B, int index_start, int
index_finish, std::vector<int>& res) {
	for (int i = index_start; i < index_finish; i++) {
		if (find_gcf(A[i], B[i]) == 1)
			res.push_back(i);
	}
}
```

Часть кода из main с реализацией ввода массива A (ввод массива B аналогичный):

```
	std::cout << "Введите массив A: " << "\n";
	std::vector<int> A;
	for (int i = 0; i < size; i++) {
		int val;
		do {
			std::cout << "Введите элемент массива с индексом " << i << ". Он
должен быть натуральным числом: ";
			std::cin >> val;
			if (std::cin.fail()) {
				std::cin.clear();
				std::cin.ignore(INT_MAX);
			}
		} while (val <= 0);
		A.push_back(val);
	}
```

Часть кода из main с реализацией распределения индексов по потокам и непосредственного их запуска:

```
for (int i = 0; i < count_of_threads; i++)
	{
		int capacity = size / count_of_threads;
		int index_start;
		int index_finish;
		if (i < count_of_threads - 1) {
			index_start = i * capacity;
```

```
                        index_finish = index_start + capacity;
                }
                else {
                        index_start = i * capacity;
                        index_finish = size;
                }
                std::thread th(count_coprime_numbers, A, B, index_start, index_finish,
std::ref(res));
                th.join();
        }
```

3.2. Формат выходных данных

Выводится множество индексов, удовлетворяющих условию, например:

```
Множество индексов, для которых A[i] и B[i] взаимно просты:
0
1
2
4
6
```

4. Код программы

```
#include <iostream>
#include <thread>
#include <string>
#include <vector>
#include <clocale>

int find_gcf(int x, int y) {
        if (y == 0)
                return x;
        else find_gcf(y, x % y);
}

void count_coprime_numbers(std::vector<int> A, std::vector<int>B, int index_start, int
index_finish, std::vector<int>& res) {
        for (int i = index_start; i < index_finish; i++) {
                if (find_gcf(A[i], B[i]) == 1)
                        res.push_back(i);
        }
}

int main(int argc, char* argv[]) {
        setlocale(LC_CTYPE, "rus");

        int size = rand() + 1000;

        std::cout << "Массивы A и B будут иметь длину " << size << "\n";
        std::cout << "Введите массив A: " << "\n";
        std::vector<int> A;
        for (int i = 0; i < size; i++) {
                int val;
                do {
                        std::cout << "Введите элемент массива с индексом " << i << ". Он
должен быть натуральным числом: ";
                        std::cin >> val;
                        if (std::cin.fail()) {
                                std::cin.clear();
                                std::cin.ignore(INT_MAX);
```

```
                        }
                } while (val <= 0);
                A.push_back(val);
        }

        std::cout << "Введите массив B: " << "\n";
        std::vector<int> B;
        for (int i = 0; i < size; i++) {
                int val;
                do {
                        std::cout << "Введите элемент массива с индексом " << i << ". Он
должен быть натуральным числом: ";
                        std::cin >> val;
                        if (std::cin.fail()) {
                                std::cin.clear();
                                std::cin.ignore(INT_MAX);
                        }
                        val = rand();
                } while (val <= 0);
                B.push_back(val);
        }

        std::cout << "Программа будет выполняться на основе парадигмы взаимодействующих
равных многопоточного программирования. " <<
                "Введите число потоков. ";
        int count_of_threads = -1;
        while (count_of_threads <= 0 || count_of_threads > size) {
                std::cout << "Число потоков - натуральное число, не превосходящее размер
массивов A и B, равный " << size << ": ";
                std::cin >> count_of_threads;
        }

        std::vector<int> res;
        for (int i = 0; i < count_of_threads; i++)
        {
                int capacity = size / count_of_threads;
                int index_start;
                int index_finish;
                if (i < count_of_threads - 1) {
                        index_start = i * capacity;
                        index_finish = index_start + capacity;
                }
                else {
                        index_start = i * capacity;
                        index_finish = size;
                }
                std::thread th(count_coprime_numbers, A, B, index_start, index_finish,
std::ref(res));
                th.join();
        }

        std::cout << "\n" << "Множество индексов, для которых A[i] и B[i] взаимно просты:
" << "\n";
        for (int i = 0; i < res.size(); i++)
                std::cout << res[i] << "\n";
}
```

5. Испытания

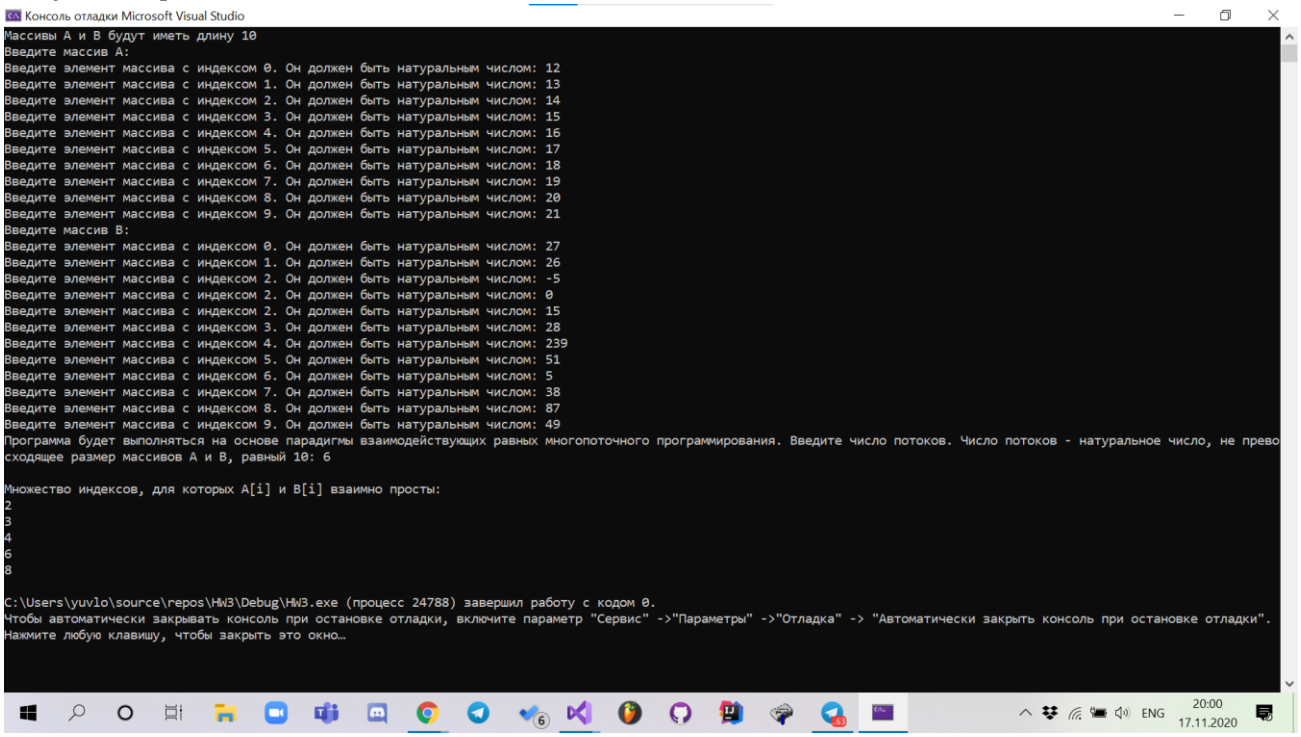
5.1. Запуск программы на данных, меньших заданной границы

Эта проверка была сделана с целью ручного просмотра правильности работы программы.

Была произведена следующая незначительная замена кода, **не влияющая на его работу**, а лишь на обработку входных данных:

```
22
23      int size = 10;//rand() + 1000;
24
```

Результат работы:

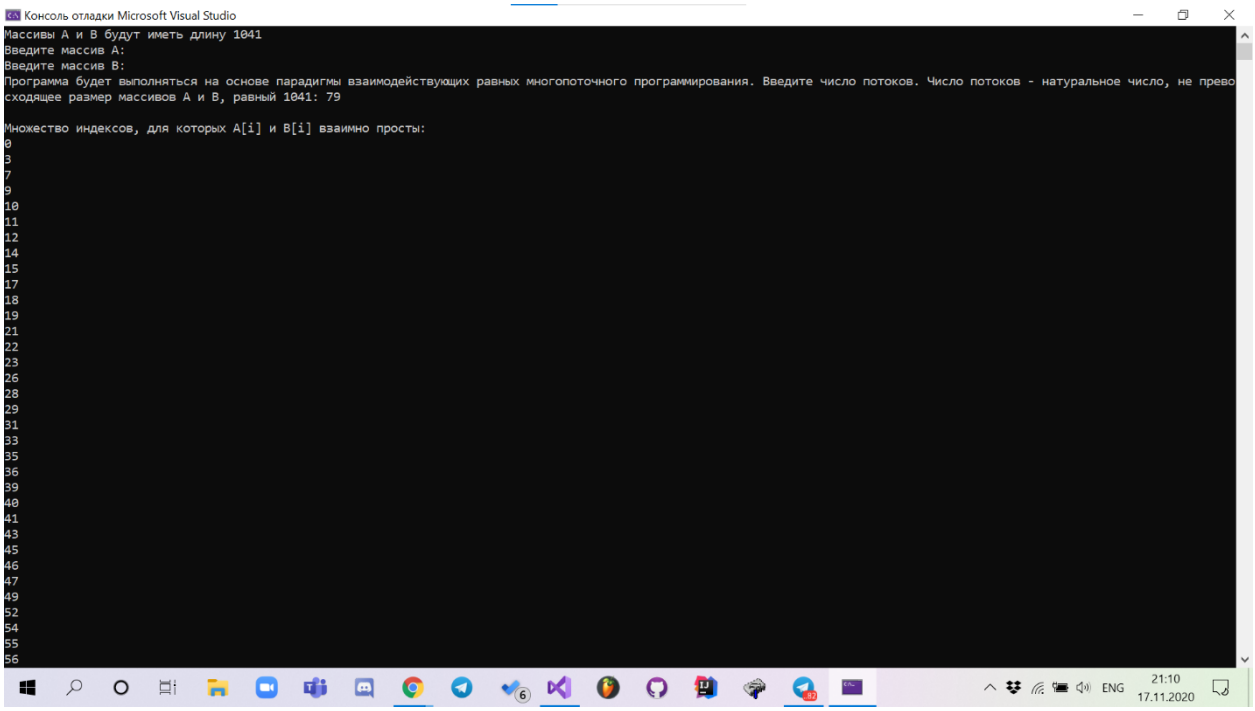


5.2. Запуск программы на исходных условиях со сгенерированными значениями

Была произведена следующая незначительная замена кода, **не влияющая на его работу**, а лишь на обработку входных данных (случайная генерация значений массивов):

```
std::cout << "Введите массив A: " << "\n";
std::vector<int> A;
for (int i = 0; i < size; i++) {
    /*
    int val;
    do {
        std::cout << "Введите элемент массива с индексом " << i << ". Он должен быть натуральным числом: ";
        std::cin >> val;
        if (std::cin.fail()) {
            std::cin.clear();
            std::cin.ignore(INT_MAX);
        }
    } while (val <= 0);
    A.push_back(val);
    */
    A.push_back(rand());
}
```

И аналогичная замена для ввода значений массива B.
Результат работы:



Поскольку разброс индексов довольно произвольный, можно сделать вывод, что программа работает корректно.

6. Репозиторий исходного кода

https://github.com/yuvlovchikova/ComputingArchitectureHW/tree/master/hw3

7. Источники

1) http://softcraft.ru/edu/comparch/tasks/t03/ - формулировка задания
2)https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%95%D0%B2%D0%BA%D0%BB%D0%B8%D0%B4%D0%B0 – алгоритм Евклида (и так его знала, но для полноты привожу как источник)
3) https://ravesli.com/urok-71-generatsiya-sluchajnyh-chisel-funktsii-srand-i-rand/#toc-0 – генерация случайного значения
4) https://docplayer.ru/48706922-Lekciya-5-paradigmy-parallelnogo-programmirovaniya.html – описание моделей и парадигм многопоточного программирования
5) http://www.williamspublishing.com/PDF/5-8459-0388-2/part.pdf - описание моделей и парадигм многопоточного программирования
6) https://m.habr.com/ru/post/182610/ - описаниие <thread> для C++
7) https://stepik.org/lesson/58810/step/6?unit=36391 – урок по многопоточному программированию на продолжающем курсе по C++