НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК

ДОМАШНЕЕ ЗАДАНИЕ №4

Пояснительная записка

Студент группы БПИ 197
Неугодников Сергей Игоревич

1. Текст задания

Текст 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». Используя OpenMP.

2. Методы вычисления

Использовали алгоритм Евклида для проверки наличия иного делителя кроме 1. Добавляем индекс в том случае, когда не нашли иных делителей.

```
bool evk(int a, int b) {
    while (a != 0 && b != 0) {
        if (a > b && b > 0)
            a %= b;
        else
            if (a > 0)
                b %= a;
    }
    return a == 1 || b == 1;
}
```

3. Список используемых источников

- Требование к сдачи дз (http://softcraft.ru/edu/comparch/tasks/t03/)
- Грегори Р. Эндрюс. Основы многопоточного, параллельного и распределенного программирования. - М.: Издательский дом "Вильямс", 2003.
- Примеры использования OpenMP на сайте SoftCraft (http://www.softcraft.ru/edu/comparch/practice/thread/03-openmp/)

4. Инструкция по работе с программой

На вход в консольное приложение пользователь должен ввести «максимально допустимое значение длинны массива», «максимальное значение переменной массива» и «количество потоков». Далее программа на основе полученных данных выполняет поставленную задачу и выводит ответ. (П.с. если я правильно понял задание, то можно было не только сгенерировать случайные переменные массива, но и сделать их длинны тоже случайными. Так что я предоставил доступ пользователю указать правую границу переменных.)

5. Пример работы программы

Пример работы программы предоставлен в файле Example.txt

6. Отличие от предыдущей домашней работы

Было:

```
std::vector<int> Threads(const std::vector<int>& A, const std::vector<int>& B, std::size_t start, std::size_t length) {
    //Массив индеков
    std::vector<int> answer;

    //Выделяем части массивов для проверки
    std::vector<int> a(A.begin() + std::min(start, A.size()), A.begin() + std::min(start + length + A.size(), A.size()));
    std::vector<int> b(B.begin() + std::min(start, B.size()), B.begin() + std::min(start + length + B.size(), B.size()));

    //Проверяем на наличие общих делитилей не считая 1
    for (size_t i = start; i < a.size() && i < b.size() && i < start + length; i++)
    {
        if (evk(a[i], b[i]))
            answer.push_back(i);
    }
    return answer;
}
```

Стало:

```
std::set<int> Threads(const std::vector<int>& A, const std::vector<int>& B, std::size_t thread_count, std::size_t length) {
    //Сет индеков
    std::set<int> answer;

    //Задаем количество потоков
    omp_set_num_threads(thread_count);

    //Начинаем работу с потоками
#pragma omp parallel
    {
#pragma omp for
        for (int i = 0; i < min(A.size(), B.size()); i++)
        {
            if (evk(A[i], B[i]))
#pragma omp critical
            {
                answer.insert(i);
            }
        }
    }
    return answer;
}
```

Было:

```
//Запускаем ассинхронно метод
for (int i = 0; i < thread_count - 1; i++)
    threads.push_back(async(Threads, A, B, parts * i, parts));

//Записываем значения в вектор
for (auto& thread : threads) {

    auto thr = thread.get();
    for (auto& i : thr)
        answer.push_back(i);
}
```

Стало:

```
//Получаем сеты с ответами
threads.insert(Threads(A, B, thread_count, parts));

//Записываем значения в вектор
for (auto& thread : threads) {
    for (auto& i : thread)
        answer.push_back(i);
}
```