ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

Пояснительная записка к третьему домашнему заданию по дисциплине “Архитектура вычислительных систем”

Преподаватель: Легалов Александр Иванович
Студент: Мкртумян Роберт Манвелович

Группа: ФКН БПИ194

Вариант: 12

Москва 2020

Текст задания

Определить индексы i, j, для которых существует наиболее длинная последовательность A[i] < A[i+1] < A[i+2] < A[i+3] < … < A[j]. Входные данные: массив чисел A, произвольной длины большей 1000. Количество потоков является входным параметром.

Используемая модель вычислений

При разработке использовалась итеративная модель построения многопоточных приложений, так как в данной задаче потоки равноправны и выполняют идентичные циклические задачи

Алгоритм работы программы

1) Разделяем исходный массив на интервалы по числу потоков

2) В каждом потоке(интервале) ищем максимальную по длине последовательность возрастающих чисел*

3) Ищем среди потоков максимальную последовательность

* если наступает конец интервала, а последовательность по прежнему возрастает, то считаем до окончания этой последовательности

Ввод/Вывод

Входные данные передаются в качестве аргументов командной строки в следующем формате:

<путь до файла с входными данными>_<путь до выходного файла>_<число потоков>

В входном файле

На первой строке записана длина массива, далее в каждой строке каждый элемент

В выходном файле

На первой строке - длина последовательности.

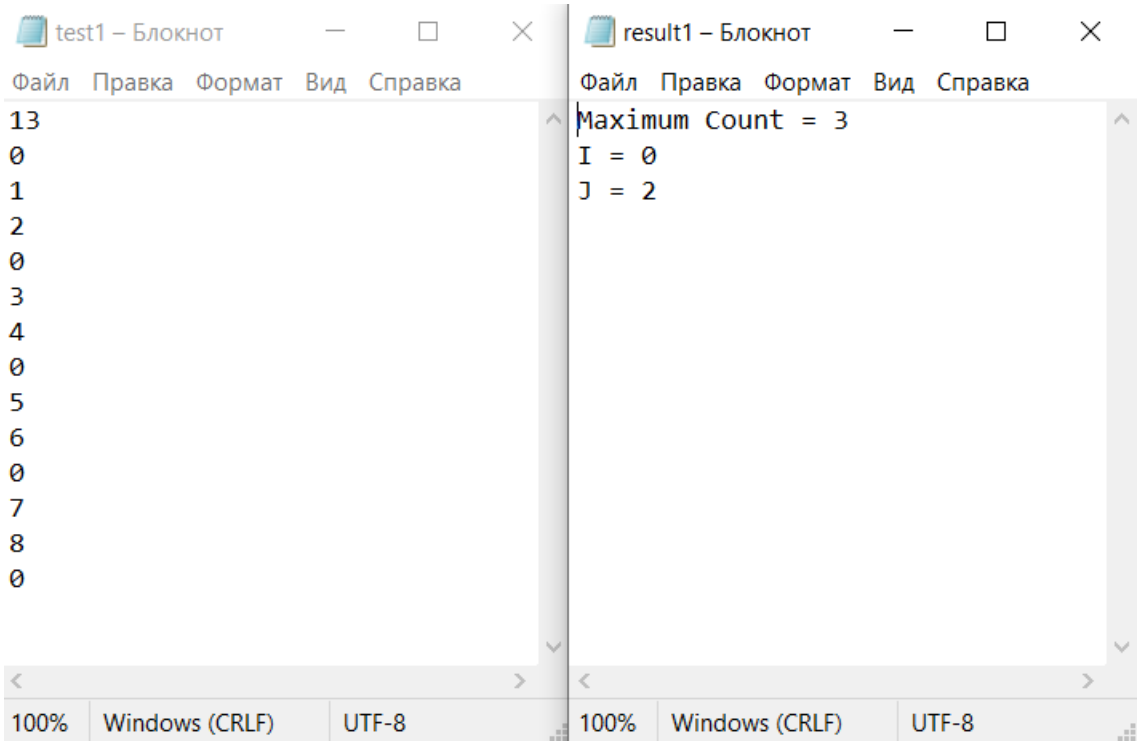
На второй строке - индекс начала последовательности.

На третьей строке - индекс конца последовательности.

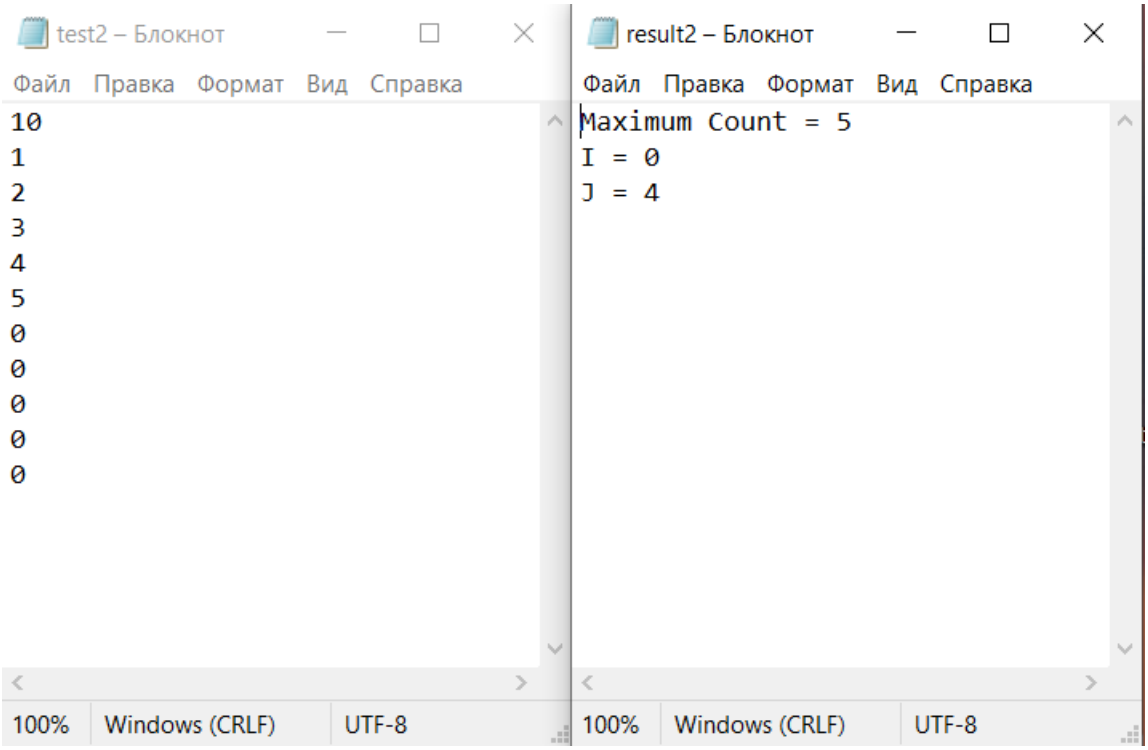
Тестирование

Тесты с массивами длины >1000 **можно найти** в репозитории в файле **test/input**.
Здесь для удобства рассмотрены тесты с небольшими входными данными.

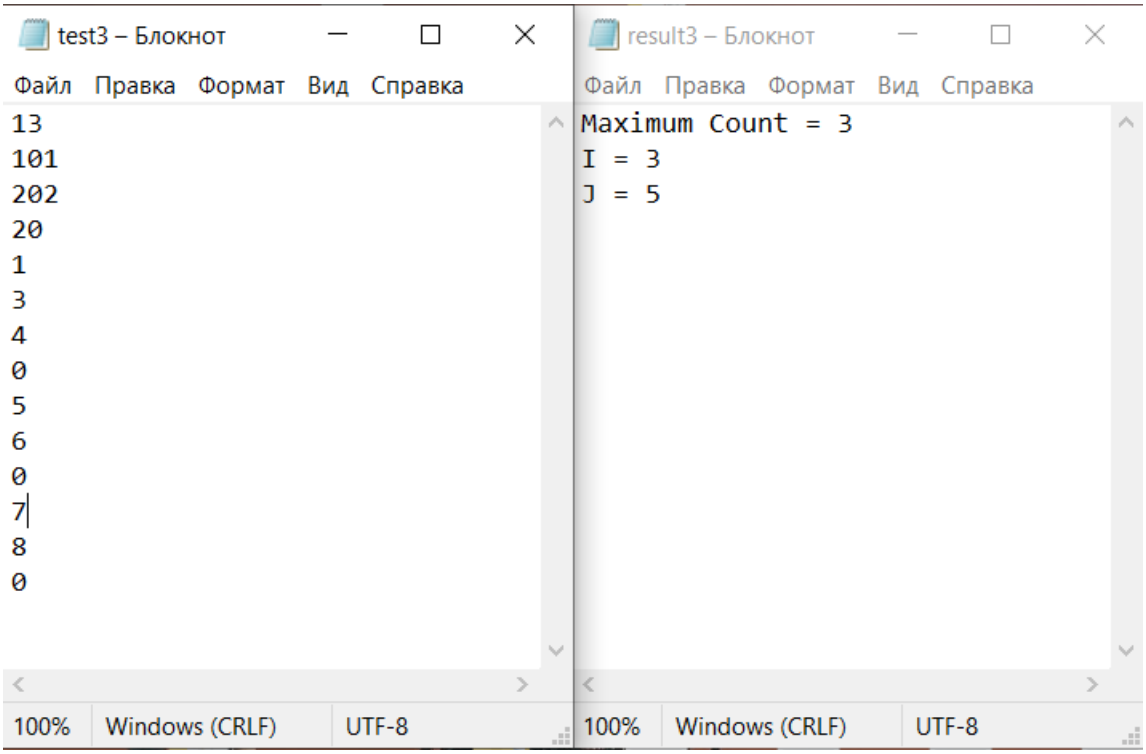
Тест 1



Тест 2



Тест 3



Список используемых источников

1. А.И. Легалов “Многопоточное программирование. Простые программы” (http://www.softcraft.ru/edu/comparch/practice/thread/01-simple/)
2. А.И. Легалов “Многопоточное программирование.

Синхронизация.”
(http://www.softcraft.ru/edu/comparch/practice/thread/02
-sync/)
3. А.И. Легалов “Архитектура параллельных
вычислительных систем. Многопоточность.”
(http://www.softcraft.ru/edu/comparch/lect/07-
parthread/)
4. Хабр. Pthreads : Потоки в русле POSIX
(https://habr.com/ru/post/326138/)

Текст программы

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <sstream>
#include <fstream>
#include <string>
#include <cmath>

using namespace std;

/// <summary>
/// Структура данных для потоков
/// </summary>
struct Package {
	int* array;					// Указатель на начало области обработки
	unsigned int threadNum;		// Номер потока
	unsigned int i;				// Индекс начала последовательности
	unsigned int j;				// Индекс конца последовательности
	unsigned int maxSize = 1;	// Максимальный размер последовательности
};

void* func(void* param);

int threadCount = 1; // Количество потоков(по умолчание программа однопоточная)
int arraySize = 0;		// Размер обрабатываемого массива
int* A;					// Указатель на начало обрабатываемого массива


int main(int argc, char** argv) {
	if (argc != 4) {	// Проверяем переданы ли все нужные параметры
		cout << "Incorrect input data" << endl;
		return 0;
	}

	string in_file = argv[1];	// Путь к файлу, с входными данными
	string out_file = argv[2];	// Путь к выходному файлу
	threadCount = stoi(string(argv[3]));

	ifstream fin; // Поток для работы с входным файлом
	fin.open(in_file);
	ofstream fout;		// Поток для работы с выходным файлом
	fout.open(out_file);
	if (!fin.is_open() || !fout.is_open()) {	// Проверяем нет ли ошибок при открытии
файлов
		cout << "File input/output error" << endl;
```

```
			return 0;
		}


		fin >> arraySize; // Читаем размер обрабатываемого массива
		A = new int[arraySize];     // Создаем массив заданной длины
		unsigned int index = 0;
		int num;
		while (!fin.eof() && index < arraySize)  // заполняем массив
		{
			fin >> num;
			A[index] = num;
			index++;
		}
		fin.close();  // закрываем поток входного файла

		pthread_t* thread = new pthread_t[threadCount]; // массив потоков
		Package* pack = new Package[threadCount];       // массив структур данных для работы
с потоками

		// Создаем дочерние потоки
		for (int i = 0; i < threadCount; i++) {
			// Создаем структуры для передачи потоку
			pack[i].array = A;
			pack[i].threadNum = i;
			pthread_create(&thread[i], nullptr, func, (void*)&pack[i]);
		}


		for (int i = 0; i < threadCount; i++) {    // ожидание завершения работы дочерних
потоков
			pthread_join(thread[i], nullptr);       // и получение результата их
вычислений
		}

		int maxElem = -1;    // Индекс структуры в которой находится максимальная по длине
последовательность
		int max = 0;         // Максимальная длина последовательности
		for (int i = 0; i < threadCount; i++) {  // Находим индекс структуры с максимальной
последовательностью
			if (max < pack[i].maxSize)
			{
				max = pack[i].maxSize;
				maxElem = i;
			}
		}

		// Записываем результат в выходной файл
		fout << "Maximum Count = " << pack[maxElem].maxSize << "\n";
		fout << "I = " << pack[maxElem].i << "\n";
		fout << "J = " << pack[maxElem].j << "\n";

		cout << "Main end..." << endl;
}


//стартовая функция для дочерних потоков
void* func(void* param) {
		Package* p = (Package*)param; // Восстанавливаем структуру
		unsigned int shift = arraySize / threadCount; // Смещение в потоке для начала массива
		int startIndex = p->threadNum * shift;    // Стартовый индекс обработки
		int endIndex = startIndex + shift;        // Конечный индекс обработки

		int max = 0;
		int start = 0;
		int end = 0;
		int index = startIndex;
		while (true)
```

```
	{	// Ищем максимальную последовательность в потоке
		start = index;
		int res = 1;
		while (A[index] < A[index+1] && index < arraySize)	// Пока последовательность возрастает, инкрементируем длину
		{
			index++;
			res++;
		}
		end = index;
		if (res > max)		// если очередная найденная длина максимальна, то перезаписываем длину
		{
			max = res;
			p->i = start;
			p->j = end;
			p->maxSize = res;
		}
		if (index >= endIndex)	// если вышли за границы нашево интервала, завершаем работу потока
		{
			break;
		}
		if (start == index)
		{
			index++;
		}
	}
	return nullptr;
}
```