**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |
| --- | --- |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **ОТЧЁТ К ДОМАШНЕМУ ЗАДАНИЮ № 3**  **ПО ДИСЦИПЛИНЕ**  **«АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ»**  **ВАРИАНТ 15** | | |
|  |  | |
| **Исполнитель**  студент группы БПИ191  Н.К. Игумнов | |
|  | | |
|  | |  |

**Москва 2020**

ЗАДАНИЕ

Вывести список всех целых чисел, содержащих от 4 до 9 значащих цифр, которые после умножения будут содержать все те же самые цифры в произвольной последовательности и в произвольном количестве. Входные данные: целое положительное число , больше единицы и меньше десяти. Количество потоков является входным параметром.

ИНТЕРПРЕТАЦИЯ

Пройдёмся по всем числам из промежутка , если после умножения на входной параметр все цифры числа содержатся в , то добавим его в итоговый ответ. Выведем список всех полученных чисел в консоль / файл.

РЕШЕНИЕ

Для проверки, содержатся ли все цифры нового числа в , было предложено пять вариантов:

1. Приведём изначальное целочисленное число в строковое методом , пройдёмся по всем символам в полученной строке и добавим их в <> (сделаем ). Повторим эту операцию для (только выделим для данного числа другой ). Если -ы совпадут, то условие выполняется (работает за ≈ **307.704** секунд для , , разрядов до 8, Debug).
2. Пройдёмся по всем цифрам числа и добавим их в <> (сделаем ). Повторим эту операцию для (только выделим для данного числа другой ). Если -ы совпадут, то условие выполняется (работает за ≈ **262.573** секунд для , , разрядов до 8, Debug).
3. Выделим для каждого числа ( и ) отдельный массив -ов размера 10 (изначально заполненный ), отвечающий за наличие / отсутствие конкретных цифр в числе. Пройдёмся по всем цифрам числа и обновим значение массива , соответствующее данной цифре (сделаем ). Повторим эту операцию для . Если массивы совпадут, то условие выполняется (работает за ≈ **15.06** секунд для , , разрядов до 8, Debug).
4. Выделим для каждого числа отдельный <10>, отвечающий за наличие / отсутствие конкретных цифр в числе. Пройдёмся по всем цифрам числа и обновим значение -а , соответствующее данной цифре (сделаем ). Повторим эту операцию для . Если -ы совпадут, то условие выполняется (работает за ≈ **20.87** секунд для , , разрядов до 8, Debug).
5. Выделим для каждого числа отдельное дополнительное целое число (), отвечающее за наличие / отсутствие конкретных цифр в числе. Пройдёмся по всем цифрам числа и обновим . Повторим эту операцию для . Если данные числа совпадут, то условие выполняется (работает за ≈ **12.436** секунд для , , разрядов до 8, Debug). – Почему это возможно? В худшем случае число состоит из всех цифр . Тогда значение этой функции будет равно . То есть это число помещается в .

Для того, чтобы программа сгенерировала входные числа, требуется раскомментировать 11-ую строчку: #define GENERATE.

Для того, чтобы программа выводила всю информацию в файл, требуется раскомментировать 12-ую строчку: #define FILE\_OUT.

Функция считывает входные данные: – входной множитель, – количество потоков и, если требуется, путь до выходного файла .

В данной задаче я применяю модель **итеративного паралеллизма**.

Функция находит ответ на задачу, вызывая потоков. Каждый из потоков описывается итеративной функцией для своего промежутка. В этой функции для каждого числа из данного промежутка будет проверятся условие задачи с помощью функции – возвращающей структуру данных, отвечающую за наличие / отсутствие конкретных цифр в числе (см. выше).

Функция выводит в консоль / выходной файл (в зависимости от FILE\_OUT) список всех полученных чисел.

ТЕСТИРОВАНИЕ ПРОГРАММЫ

1. Некорректный ввод ()

Изображение выглядит как текст

Автоматически созданное описаниеПрограмма отработала успешно.2. Некорректный ввод ()

Изображение выглядит как текст

Автоматически созданное описаниеПрограмма отработала успешно.

3.

Изображение выглядит как текст

Автоматически созданное описание

Программа отработала успешно.

4. (см. tests/answer.txt)

Изображение выглядит как текст

Автоматически созданное описаниеПрограмма отработала успешно.

5. (см. tests/answer2.txt)

Изображение выглядит как текст

Автоматически созданное описание

Программа отработала успешно

6. (см. tests/answer3.txt)

Изображение выглядит как текст

Автоматически созданное описание

Программа отработала успешно

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ошибки многопоточности в C++ [Электронный ресурс]. Режим доступа: https://otus.ru/nest/post/145/, свободный. (дата обращения: 17.11.2020)
2. Simple example of threading in C++ [Электронный ресурс]. Режим доступа: https://stackoverflow.com/questions/266168/simple-example-of-threading-in-c, свободный. (дата обращения: 17.11.2020)
3. std::thread::hardware\_concurrency [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/thread/thread/hardware\_concurrency, свободный. (дата обращения: 17.11.2020)
4. Measuring execution time of a fuction in C++ [Электронный ресурс]. Режим доступа: https://stackoverflow.com/questions/22387586/measuring-execution-time-of-a-function-in-c, свободный. (дата обращения: 17.11.2020)
5. Практические приёмы построения многопоточных приложений [Электронный ресурс]. Режим доступа: http://www.softcraft.ru/edu/comparch/tasks/t03/, свободный. (дата обращения: 17.11.2020)
6. Choosing the number of threads at runtime [Электронный ресурс]. Режим доступа: https://livebook.manning.com/book/c-plus-plus-concurrency-in-action/chapter-2/92/, свободный. (дата обращения: 17.11.2020)

КОД ПРОГРАММЫ

Смотрите multiThread.cpp

#include <iostream>  
#include <optional>  
#include <thread>  
#include <chrono>  
#include <random>  
#include <fstream>  
#include <list>  
#include <vector>  
#include <mutex>  
  
//#define GENERATE  
//#define FILE\_OUT  
  
constexpr int64\_t MIN\_VALUE = 1000;  
constexpr int64\_t MAX\_VALUE = 999'999'999ll;  
  
std::mutex mut;  
  
bool Read(int \*n, int \*thread\_number, std::string \*file\_name) { // NOLINT  
#ifdef GENERATE  
 std::mt19937 gen(std::chrono::high\_resolution\_clock::now().time\_since\_epoch().count());  
 std::uniform\_int\_distribution<int> dist\_n(2, 9);  
 \*n = dist\_n(gen);  
 std::uniform\_int\_distribution<int64\_t> dist\_t(1, MAX\_VALUE - MIN\_VALUE + 1);  
 \*thread\_number = dist\_t(gen);  
 \*file\_name = "answer.txt";  
 std::cout << "Number n: " << \*n << '\n';  
 std::cout << "Amount of threads: " << \*thread\_number << '\n';  
#else  
 std::cout << "Input your number n (in (1, 10)): ";  
 std::cin >> \*n;  
 if (\*n <= 1 || \*n >= 10) {  
 std::cout << "Incorrect number!\n";  
 return false;  
 }  
 // see https://livebook.manning.com/book/c-plus-plus-concurrency-in-action/chapter-2/92 for details  
 unsigned int hardware\_threads = std::thread::hardware\_concurrency();  
 std::cout << "Optimal number of threads for your machine: " << (hardware\_threads != 0 ? hardware\_threads : 2)  
 << '\n';  
 std::cout << "Input amount of threads (in [1, 10000]): ";  
 std::cin >> \*thread\_number;  
 if (\*thread\_number < 1 || \*thread\_number > 10000) {  
 std::cout << "Incorrect amount of threads!\n";  
 return false;  
 }  
#endif  
#ifdef FILE\_OUT  
 std::cout << "Input name of output file: ";  
 std::cin >> \*file\_name;  
 std::ofstream out(\*file\_name);  
 if (!out.is\_open()) {  
 std::cout << "Incorrect file name!\n";  
 out.close();  
 return false;  
 }  
 out.close();  
#endif  
 return true;  
}  
  
unsigned int Fun(int64\_t x) {  
 unsigned int y = 0; // [0, 1024)  
 while (x != 0) {  
 y |= 1u << unsigned(x % 10);  
 x /= 10;  
 }  
 return y;  
}  
  
void ComputeThread(int n, int64\_t from, int64\_t to, std::list<int64\_t> &numbers) {  
 for (int64\_t i = from; i <= to; ++i) {  
 if (Fun(i) == Fun(i \* n)) {  
 mut.lock();  
 numbers.push\_back(i);  
 mut.unlock();  
 }  
 }  
}  
  
void Compute(int n, int64\_t thread\_number, std::list<int64\_t> &numbers) {  
 int64\_t loop\_size = (MAX\_VALUE - MIN\_VALUE + 1) / thread\_number;  
 std::vector<std::thread> thr(thread\_number);  
 for (int64\_t i = 0; i < thread\_number; ++i) {  
 if (i != thread\_number - 1) {  
 thr[i] = std::thread{ComputeThread, n, MIN\_VALUE + loop\_size \* i, MIN\_VALUE + loop\_size \* (i + 1) - 1,  
 std::ref(numbers)};  
 } else {  
 thr[i] = std::thread{ComputeThread, n, MIN\_VALUE + loop\_size \* i, MAX\_VALUE, std::ref(numbers)};  
 }  
 }  
 for (std::thread &Thread : thr) {  
 Thread.join();  
 }  
}  
  
void Print(int n, int threads\_num, const std::string &file\_name, const std::list<int64\_t> &numbers) {  
#ifdef FILE\_OUT  
 std::ofstream out(file\_name);  
 out << "Number n: " << n << '\n';  
 out << "Amount of threads: " << threads\_num << '\n';  
 for (int64\_t x : numbers) {  
 out << x << '\n';  
 }  
 out.close();  
#else  
 for (int64\_t x : numbers) {  
 std::cout << x << '\n';  
 }  
#endif  
}  
  
int main() {  
 std::ios\_base::sync\_with\_stdio(false);  
 int n, thread\_number;  
 std::string file\_name;  
 if (!Read(&n, &thread\_number, &file\_name)) {  
 return 1;  
 }  
  
 std::list<int64\_t> numbers;  
 auto begin = std::chrono::steady\_clock::now();  
  
 Compute(n, thread\_number, numbers);  
  
 auto end = std::chrono::steady\_clock::now();  
 auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);  
  
 Print(n, thread\_number, file\_name, numbers);  
 std::cout << "Time (computation): " << elapsed\_ms.count() / 1000. << " sec\n";  
 return 0;  
}