ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

Домашнее задание №3
по дисциплине «Архитектура вычислительных систем»

Практические приемы построения многопоточных приложений

Исполнитель

студент группы БПИ196-1

Махнач Ф. О.

17.11.2020 г.

Оглавление

1. Текст задания

Вывести список всех целых чисел, содержащих от 4 до 9 значащих цифр, которые после умножения на n будут содержать все те же самые цифры в произвольной последовательности и в произвольном количестве.

Входные данные: целое положительное число $1 < n < 10$. Количество потоков является входным параметром

2. Однопоточный аналог приложения на языке Python

Для общего понимания сути задачи была написана примитивная версия программы на языке Python:

```python
n = int(input())
nums = set()
lower, upper = 10**3, 10**9
for i in range(lower, upper):
    digits_of_i = set([int(c) for c in str(i)])
    digits_of_i_mul_n = set([int(c) for c in str(i * n)])
    if digits_of_i == digits_of_i_mul_n:
        nums.add(-i)
        nums.add(i)
print(*nums)
```

Реализация алгоритма следующая: для каждого значения $\boldsymbol{i}$ из указанного диапазона

1) Создаём множество (контейнер из уникальных элементов) из всех цифр числа $\boldsymbol{i}$;
2) Создаём множество из всех цифр числа $\boldsymbol{i} \cdot \boldsymbol{n}$;
3) Если множества равны – добавляем $\boldsymbol{i}$ и $-\boldsymbol{i}$ в результирующее множество;

Здесь отсутствует:

1) Использование потоков;
2) Проверка на входной параметр n;
3) Вывод в файл;

Выводы, сделанные в результате реализации примитивной версии программы:

1) Для определения равенства множества цифр для чисел можно использовать `set` – контейнер, содержащий упорядоченное множество уникальных элементов. Такой способ неэффективен, однако он наиболее прост для восприятия и понимания (и первым пришёл на ум).

2) Из формулировки условия задачи считаю, что если число i является искомым, то число $-i$ также является искомым (подходит под критерии).

3) Для диапазона [1e3; 1e9) программа работает слишком долго. Вменяемый результат удалось получить лишь на [1e3; 1e6).

4) Для вывода результата в отсортированной форме также придётся использовать контейнер `set`, однако такой подход, скорее всего, потребует большого количества памяти в время исполнения, а также потребует синхронизации доступа к этому контейнеру разных потоков. Для 1e9 понадобится 4*1e9 байт≈475Мб. Конечно, на практике значений будет меньше, но не на порядок.

5) Проверка для каждого значения из диапазона независима. Из предложенных моделей построения многопоточных приложений наиболее подходящей мне кажется **итеративный параллелизм**.

3. Реализация алгоритма

3.1. Первая реализация алгоритма на языке C++

Первоначально идея использования контейнера `set` была перенесена реализацию на C++. Таким образом, основная часть алгоритма выглядела следующим образом:

```
// Создаёт множество цифр числа.
std::set<int> MakeSetOfDigits(int64_t num) {
	std::set<int> result;
	while (num != 0) {
		result.insert(num % 10);
		num /= 10;
	}
	return result;
}

// Проверяет, удовлетворяет ли число условию.
void CheckNum(int num) {
	std::set<int> num_digit_set = MakeSetOfDigits(num);
	std::set<int> prod_digit_set = MakeSetOfDigits(static_cast<int64_t>(num) * ::n);

	if (num_digit_set == num_digit_set) {
		result_set.insert(-num);
		result_set.insert(num);
	}
}

// Проверяет каждое число из диапазона.
void ProcessNums() {
	for (int i = LOWER_BOUND; i < UPPER_BOUND; ++i) {
		CheckNum(i);
	}
}
```

Здесь последовательно вызывается функция `CheckNum` для каждого числа из диапазона [1e3; 1e9) (числа из 4-9 цифр). В `CheckNum` создаются множества из цифр чисел и сравниваются на равенство. Если число проходит такую проверку, оно добавляется в `result_set` – контейнер, содержащий результат.
Часть кода, отвечающая за ввод-вывод не приведена для краткости (см. п. 4).
Конечно, это всё еще линейная программа. Необходимо добавить потоки выполнения и распределить между ними задачи.

3.2. Распараллеливание

Так как каждая проверка является независимой задачей, можем использовать итеративный параллелизм: имея `m` потоков (`std::thread`) в массиве, `i`-ый поток будет обрабатывать каждый `m`-ое значения начиная с `1000+i` (`1000` – нижняя граница, т.е. самое малое четырёхзначное число. Аналогичный способ распределения потоков был показан на семинарах (например, тут).
После добавления распараллеливания изменилась функция `ProcessNums` и добавилась функция `CalculateMultiThread`, запускающая потоки:

```
// Запускает проверку каждого num_of_threads числа (так каждый поток вычислит свою долю значений).
void ProcessNums(int thread_num) {
	for (int i =LOWER_BOUND + thread_num; i < UPPER_BOUND; i += num_of_threads) {
		CheckNum(i);
	}
}

// Запускает вычисляющие потоки.
void CalculateMultiThread() {
	std::thread* threads = new std::thread[num_of_threads];
	// Запускаем потоки.
	for (size_t i = 0; i < num_of_threads; ++i) {
		threads[i] = std::thread(ProcessNums, i);
	}
	// Ждём конца работы.
	for (size_t i = 0; i < num_of_threads; ++i) {
		threads[i].join();
	}
	delete[] threads;
}
```

Помимо этого, для безопасной работы с result_set был добавлен mutex:

```
void CheckNum(int num) {
	std::set<int> num_digit_set = MakeSetOfDigits(num);
	std::set<int> prod_digit_set = MakeSetOfDigits(static_cast<int64_t>(num) * ::n);

	if (num_digit_set == prod_digit_set) {
		glob::result_set_mutex.lock();
		glob::result_set.insert(-num);
		glob::result_set.insert(num);
		glob::result_set_mutex.unlock();
	}
}
```

Однако запуск программы показал, что она всё ещё недостаточно эффективна: диапазон [1e3; 1e6] обрабатывался >40 секунд.

3.3. Оптимизация проверки на равенство цифр десятичной записи

Очевидно, использование контейнера `set` в функции `CheckNum` несёт в себе издержки. На практике нам необходима лишь информация о 10 цифрах, т. е. для каждого числа `k` нам нужно знать, какие из 10 цифр используются в его десятичной записи, а какие нет. Эту информацию можно закодировать в 10 битах: если `i`-ый бит единица, то цифра `i` входит в десятичную запись числа. Тогда числа состоят из одних и тех же цифр тогда и только тогда, когда их 10-битные коды будут совпадать.

Это несложно реализовать, используя 16-битный целочисленный тип (за неимением лучшего). Функция нахождения 10-битного кода числа:

```
uint16_t GetBitmapOfDigits(int64_t num) {
	uint16_t bitmap = 0;
	while (num != 0) {
		// Отмечаем биты, соответствующие каждой цифре.
		bitmap |= glob::digit_masks[num % 10];
		// Проходим по каждой цифре числа.
		num /= 10;
	}
	return bitmap;
}
```

где `glob::digit_masks` – массив подготовленных масок с единственным битом на i-ой позиции, где `i` – индекс в массиве:

```
void PrecomputeDigitMasks() {
	for (size_t i = 0; i < 10; ++i) {
		glob::digit_masks[i] = 1 << i;
	}
}
```

Изменённая функция `CheckNum`:

```
void CheckNum(int num) {
	// Вычисляем маску цифр для num и для num * n.
	uint16_t num_map = GetBitmapOfDigits(num);
	uint16_t prod_map = GetBitmapOfDigits(static_cast<int64_t>(num) * glob::n);

	if (num_map == prod_map) {
		glob::result_set_mutex.lock();
		// Если num подходит, то -num также подходит.
		// Заносим -num и num в множество.
		glob::result_set.insert(-num);
		glob::result_set.insert(num);
		glob::result_set_mutex.unlock();
	}
}
```

Текущая версия уже может быть использована по назначению – на диапазоне [1e3; 1e8) программа работает 25 секунд, на [1e3; 1e9) – 6 минут (на моём компьютере). Безусловно, это довольно долго, но в то же время значительно лучше предыдущей версии.

4. Прочие части программы

Все глобальные переменные, использующиеся в программе, выделены в пространство имён `glob`:

```
namespace glob {
	// Нижняя граница диапазона проверяемых значений.
	const int LOWER_BOUND = 1e3;
	// Верхняя граница диапазона проверяемых значений.
	const int UPPER_BOUND = 1e9;
	// Входной параметр n -- то, на что мы умножаем каждое число из диапазона.
	int n;
	// Нижнее ограничение входного параметра n.
	const int n_lower = 1;
	// Верхнее ограничение входного параметра n.
	const int n_upper = 10;
	// Входной параметр -- число потоков.
	size_t num_of_threads;
	// Множество значений x таких, что {цифры x} = {цифры x * n}.
	std::set<int> result_set;
	// Мьютекс для контроля доступа к result_set.
	std::mutex result_set_mutex;
	// Битовые маски, соответствующие каждой цифре (напр 1 -> 0000000010, 7 ->
0010000000).
	uint16_t digit_masks[10];
}
```

Для ввода значений n и числа потоков используются соответствующие функции:

```
// Читает параметр N из консоли.
bool TryReadN() {
	std::cout << "Please, enter N: ";
	std::cin >> glob::n;
	return glob::n_lower < glob::n && glob::n < glob::n_upper;
}

// Читает число потоков из консоли.
bool TryReadNumberOfThreads() {
	std::cout << "Please, enter number of threads: ";
	std::cin >> glob::num_of_threads;
	return 0 < glob::num_of_threads &&
		glob::num_of_threads <= std::thread::hardware_concurrency();
}
```

Для вывода результата в файл используются следующая функция:

```
// Записывает множество чисел в файл.
void WriteSetToFile(const std::set<int>& set, const std::string& path) {
	std::ofstream out;
	out.open(path);
	if (out.is_open()) {
		for (int i : set) {
			out << i << std::endl;
		}
	}
	out.close();
}
```

Сама функция `main`:

```
int main() {
	// Считаем битовые маски, которые понадобятся в функции GetBitmapOfDigits.
	PrecomputeDigitMasks();
```

```
	bool reading_successful = TryReadN();
	if (reading_successful == false) {
		std::cerr << "Wrong input! n must be a integer in range [2; 9]." <<
std::endl;
		return 1;
	}
	reading_successful = TryReadNumberOfThreads();
	if(reading_successful == false){
		std::cerr << "Wrong input! Num of threads cannot be less than 1 or more
than " << std::thread::hardware_concurrency() << "!" << std::endl;
		return 1;
	}
	CalculateMultiThread();
	WriteSetToFile(glob::result_set, "out.txt");
	return 0;
}
```

5. Тестирование программы

Результаты вывода программы при различных входных параметрах приведены в папке tests репозитория с работой. Название файла output_N.txt обозначает, что это вывод программы при входных параметрах n = N и кол-во потоков = 4 (максимальное на моём компьютере), например, output_3.txt.

Также протестированы следующие случаи некорректного ввода:

1) n = 0

```
Please, enter N: 0
Wrong input! n must be a integer in range [2; 9].

C:\Users\fedya\source\repos\CPP_Fun\Debug\CPP_Fun.exe (process 7712) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

2) n = 10

```
Please, enter N: 10
Wrong input! n must be a integer in range [2; 9].

C:\Users\fedya\source\repos\CPP_Fun\Debug\CPP_Fun.exe (process 19328) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

3) n = 1

```
Please, enter N: 1
Wrong input! n must be a integer in range [2; 9].

C:\Users\fedya\source\repos\CPP_Fun\Debug\CPP_Fun.exe (process 19352) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

4) n = -1

```
Please, enter N: -1
Wrong input! n must be a integer in range [2; 9].

C:\Users\fedya\source\repos\CPP_Fun\Debug\CPP_Fun.exe (process 7816) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

5) n = -100

```
Please, enter N: -100
Wrong input! n must be a integer in range [2; 9].

C:\Users\fedya\source\repos\CPP_Fun\Debug\CPP_Fun.exe (process 20384) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

6) n = 5 num_of_threads = 1000

```
Please, enter N: 5
Please, enter number of threads: 1000
Wrong input! Num of threads cannot be less than 1 or more than 4!

C:\Users\fedya\source\repos\CPP_Fun\Debug\CPP_Fun.exe (process 19000) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the consc
le when debugging stops.
Press any key to close this window . . .
```

7) n = 7 num_of_threads = -15

```
Please, enter N: 7
Please, enter number of threads: -15
Wrong input! Num of threads cannot be less than 1 or more than 4!

C:\Users\fedya\source\repos\CPP_Fun\Debug\CPP_Fun.exe (process 9708) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the consc
le when debugging stops.
Press any key to close this window . . .
```