Лабораторная работа №8

Условие

Вова записал все натуральные числа в виде лестнички:

$$
\begin{array}{ccccc}
\mathbf{1} & & & & \\
\mathbf{2} & \mathbf{3} & & & \\
4 & \mathbf{5} & 6 & & \\
\mathbf{7} & 8 & 9 & 10 & \\
\mathbf{11} & 12 & \mathbf{13} & 14 & 15 \\
 & & \ldots & &
\end{array}
$$

Миша выяснил, что B(10000) = 950007619

Помогите ему вычислить B(5678027) + B(7208785). Реализуйте это программно, предложив наиболее оптимальный алгоритм.

Замечания

Число является подходящим (членом “магической тройки”) в двух случаях:

1. Представим число x и числа a, b, c, d, e, f, g, h, стоящие вокруг него в виде матрицы 3x3:

$$
\begin{pmatrix}
a & b & c \\
d & \underline{x} & e \\
f & g & h
\end{pmatrix}
$$

Тогда число x-подойдет нам, если число x - простое и среди чисел a, b, c, d, e, f, g, h есть два простых числа. Также заметим, что число d или e будет является простым только в частном случае:

$$
\begin{array}{ccc}
\mathbf{1} & & \\
\mathbf{2} & \mathbf{3} & \\
4 & \mathbf{5} & 6
\end{array}
$$

В остальных случаях числа, соседние с простым по горизонтали будут четными. Дальше мы их учитывать не будем.

2. Пусть для числа x, среди чисел a, b, c, d, e, f, g, h простое число лишь одно. Тогда для числа, являющимся в матрице 3x3 с x в центре помимо x единственным простым, построим аналогичную матрицу и будем проверять только те числа, которые не входят в матрицу от числа x. Если среди этих чисел найдется хоть одно простое, тогда число x является членом “магической тройки” и подходит.

Для n-ой строки возможно определить первое число, а для i-го числа - номер строки, в которой содержится это число.

Номер строки n, в которой содержится число i:

$$n = ceil(\frac{-1 + \sqrt{1 + 8i}}{2})$$

где ceil - округление вверх.

Первое число i, содержащееся в строке n:

$$i = \frac{n(n-1)}{2} + 1$$

В матрице для числа n, содержащегося в строке i:

$$\begin{pmatrix} i - 1_{n-1} & i_{n-1} & i + 1_{n-1} \\ ... & i_n & ... \\ i - 1_{n+1} & i_{n+1} & i + 1_{n+1} \end{pmatrix}$$

1. $i - 1_{n-1} = i_n - n$;
2. $i_{n-1} = i_n - n + 1$;
3. $i - 1_{n-1} = i_n - n + 2$;
4. $i - 1_{n+1} = i_n + n - 1$;
5. $i - 1_{n+1} = i_n + n$;
6. $i - 1_{n+1} = i_n + n + 1$;

1. Для первого числа в строке доступны числа:

- i_{n-1};
- $i + 1_{n-1}$;
- i_{n+1};
- $i + 1_{n+1}$.

2. Для предпоследнего числа в строке доступны числа:

- $i - 1_{n-1}$
- i_{n-1};
- $i - 1_{n+1}$;
- i_{n+1};
- $i + 1_{n+1}$.

3. Для последнего числа в строке доступны числа:

- $i - 1_{n-1}$
- $i - 1_{n+1}$;
- i_{n+1};
- $i + 1_{n+1}$.

Алгоритм

Найдем первое число i в строке $n - 2$ и последнее число j в строке $n + 2$ по формуле, указанной выше. После этого построим решето Эратосфена для чисел отрезка $[i; j]$. Переберем все числа n-ой строки и проверим каждое, является ли оно членом “магической тройки”.

Код

Метод *get_line_number*

Принимает на вход число и по формуле, указанной выше возвращает номер строки, в которой содержится полученное число.

```
int get_line_number(uint64_t n) {
        return std::ceil((-1 + std::sqrt(1 + 8 * n)) / 2);
}
```

Метод *get_first_number_in_line*

Принимает на вход номер строки и по формуле, указанной выше возвращает первое число в полученной строке.

```
uint64_t get_first_number_in_line(int c) {
        return c * (c - 1) / 2 + 1;
}
```

Метод *eratosthenes*

Принимает на вход числа n, k и в этих границах ищет простые числа путем перебора всех множителей чисел до $\sqrt{k}$.

```
std::vector<bool> eratosthenes(uint64_t n, uint64_t k) {
    std::vector<bool> primes(k - n, true);

  for (uint64_t p = 2; p * p <= k; p++) {
    for (uint64_t i = p * p; i <= k; i += p)
            if (i >= n) primes[i - n] = false;
  }

  return primes;
}
```

Метод *is_suitable*

Принимает на вход число n, массив простых чисел, нижнюю границу (для индексации по массиву простых чисел) и предыдущее число, если таковое имеется.

Сначала проверяем, является ли число простым. Если нет - одно нам не подходит. Дале проверим, является ли число первым, последним или последним в строке и

определим, какие числа доступны для n. После этого определим “дельты” - числа, которые надо отнять от заданного, чтобы получить числа, входящие в матрицу числа n.

Пройдем по массиву дельт и проверим каждое из чисел. Если число - простое, тогда сохраним дельту в переменную *delta_prev.* Если переменная *delta_prev* уже не равна нулю, следовательно это не первое простое число и значит, что число n является членом магической тройки. Если же мы прошли массив дельт до конца, проверим, обновили ли мы переменную *delta_prev:* если нет, значит в матрице числа n число n - единственное простое и оно нам не подходит. Иначе, возьмем единственное простое число x в матрице n помимо n и вызовем функцию еще раз, но от найденного числа и передадим туда n. Теперь, если мы найдем хотя бы одно простое число в матрице числа x и оно не будет равно n, значит число n является членом магической тройки и вернем *true*, иначе - *false.*

```
bool is_suitable(int n, std::vector<bool>& primes, uint64_t lower_border, uint64_t prev_n = 0) {
        if (!primes[n - lower_border]) return false;

        int line = get_line_number(n);
        bool is_first = n == get_first_number_in_line(line);
        bool is_last = n == (get_first_number_in_line(line + 1) - 1);
        bool is_near_last = n == (get_first_number_in_line(line + 1) - 2);
        int delta_prev = 0;

        std::vector<int> deltas;
        if (is_first) {
                deltas = {line - 1, line - 2, -line, -line - 1};
        } else if (is_near_last) {
                deltas = {line, line - 1, -line + 1, -line, -line - 1};
        } else if (is_last) {
                deltas = {line, -line + 1, -line, -line - 1};
        } else {
                deltas = {line, line - 1, line - 2, -line + 1, -line, -line - 1};
        }

        for (int delta: deltas) {
                if (primes[n - delta - lower_border] && n - delta != prev_n) {
                    if (delta_prev) return true;
                    if (prev_n) return true;
                    delta_prev = delta;
                    }
                }

        if (!delta_prev) return false;
        else return is_suitable(n - delta_prev, primes, lower_border, n);
}
```

Метод *B*

Инициализируем массив простых чисел через функцию *eratosthenes*. *И*терируемся по числам в строке n и проверяем каждое: если оно является членом магической тройки, прибавляем к *result.* После цикла возвращаем *result.*

```
uint64_t B(int n) {
    uint64_t result = 0;
  uint64_t lower_border = get_first_number_in_line(n - 2);
        uint64_t top_border = get_first_number_in_line(n + 3) - 1;
        std::vector<bool> primes = eratosthenes(lower_border, top_border);

        for (uint64_t i = get_first_number_in_line(n); i < get_first_number_in_line(n + 1); ++i) {
                if (is_suitable(i, primes, lower_border)) {
                        result += i;
                }
        }
    return result;
}
```

Метод *main*

Создаем заданные переменные $n1, n2$ и для каждой находим значение *B(n),* после чего складываем и выводим результат.

```
int main() {
    int n1 = 5678027;
    int n2 = 7208785;
    uint64_t b_n1 = B(n1);
    uint64_t b_n2 = B(n2);
    std::cout << b_n1 + b_n2 << std::endl;
    return 0;
}
```