Megaprime Numbers

locked

by _mfv_

Editorial by _mfv_

One of the fastest ways to find all prime numbers from 1 to n for n in the range 10^8 is the sieve of Eratosthenes. Let's try to tweak this method to make it work in this problem.

Remember that we need to calculate an only relatively small range of numbers (max 10^9), but it may be situated far away (max 10^{15}) from 1. Let's look at the original method.

```
std::vector<bool> isPrime(1 + max, true);
isPrime[0] = false;
isPrime[1] = false;
for (int i = 2; i * i <= max; i++) {
    if (isPrime[i]) {
        for (int j = i * i; j <= max; j += i) {
            isPrime[j] = false;
        }
    }
}
```

What elements of `isPrime` should we fill to get correct values in the range $[first, last]$? Only elements $[2, \lfloor\sqrt{last}\rfloor]$ to get prime numbers to remove their multiples from the range up to $last$ and elements $[first, last]$ to get the result. If there are elements greater than $\lfloor\sqrt{last}\rfloor$ and less than $first$ we do not need to fill them.

How can we modify the code to not fill these elements in between? One possible solution is to calculate prime numbers up to $\lfloor\sqrt{last}\rfloor$ beforehand with original method and then remove the multiples of that primes from $[first, last]$ range. For each prime p we need to find minimum number $\geq max(first, p^2)$ that is divisible by p and then go from this number with the step p assigning `false` to all array elements in the way while current position $\leq last$. One possible implementation of this procedure is following.

```
static std::vector<int> primes = getPrimesTill(MAX_ROOT);
// isPrime[i] <-> (i + first) is prime
std::vector<bool> isPrime(last - first + 1, true);
for (int p : primes) {
    long long p2 = 1LL * p * p;
    if (p2 > last) {
        break;
    }
    long long from = std::max(1LL * p, (first + p - 1) / p) * p;
    assert(from >= first);
    int fromShifted = (int)(from - first);
    int lastShifted = (int)(last - first);
    for (int i = fromShifted; i <= lastShifted; i += p) {
        isPrime[i] = false;
    }
}
```

So now we have an efficient algorithm for determining prime numbers in range. The next step is to research optimal parameters of the algorithm. For example, if we need to find prime numbers in the range $[1, 10^8]$ we can run this algorithm on the whole range or split it into parts, for example of length $2 \cdot 10^7$. Theoretically the greater the subrange the worse it fits into processor cache, and the less the subrange the worse its overhead for calculations required for each subrange and each prime number. So both too large subranges and too small subranges should be slow. My research shows that the optimal length of a subrange is about $2 \cdot 10^7$, of course, it depends on processor cache size, but it's good to have this estimate.

Let's return to the original problem. Unfortunately, the algorithm works several seconds or even tens of seconds for the range length up to 10^9, more than allowed time limit. But now we can remember that we calculate not primes but megaprimes, and there are large spaces that do not contain megaprime numbers at all in any given range of length 10^9. For example, there are no megaprime numbers from $37\,777\,777 + 1$ to $52\,222\,222 - 1$. So we split required range to subranges of length 10^7 and in each such subrange find only primes with the last 7 digits from $2\,222\,222$ to $7\,777\,777$. There are no more than $4^2 = 16$ such populated ranges (4 because there are 4 prime digits of 10 that we can place in the 8^{th} and 9^{th} place from the end). So the total length of subranges is $(7\,777\,777 - 2\,222\,222 + 1) \cdot 4^2 \approx 9 \cdot 10^7$ that fits nicely into the time limit. Also, we need to take care of megaprime numbers before 10^6 separately.

We check each found prime number naively if it consists of prime digits only.

One implementation note: C++ `vector<bool>` uses 1 bit per stored boolean value. It is faster for large arrays than to use 1 byte per value because it fits better into processor cache.

Statistics

Difficulty: Medium

Time Complexity: $\mathcal{O}(\sqrt{last} \cdot \log(\log(\sqrt{last})) + (last - first) \cdot \log(\log(last)) \cdot \frac{4}{10}^{\lg((last-first)/10^7)}$

Required Knowledge: Sieve of Eratosthenes

Publish Date: Feb 15 2017

Set by _mfv_

Problem Setter's code :

```
#pragma GCC diagnostic ignored "-Wunused-result"

#include <cstdio>
#include <vector>
#include <cassert>
#include <algorithm>

inline bool hasPrimeDigitsOnly(long long n) {
    while (n > 0) {
        int d = int(n % 10);
        if (d != 2 && d != 3 && d != 5 && d != 7) {
            return false;
        }
        n /= 10;
    }
    return true;
}

std::vector<int> getPrimesTill(int max) {
    assert(max >= 1);
    std::vector<int> primes;
    std::vector<bool> isPrime(1 + max, true);
    isPrime[0] = false;
    isPrime[1] = false;
    for (int i = 2; i * i <= max; i++) {
        if (isPrime[i]) {
            for (int j = i * i; j <= max; j += i) {
                isPrime[j] = false;
            }
        }
    }
    for (int i = 2; i < (int)isPrime.size(); i++) {
        if (isPrime[i]) {
            primes.push_back(i);
        }
    }
    return primes;
}

const int MAX_ROOT = (int)std::sqrt(1e15);

int countMegaPrimes(long long first, long long last) {
    assert(1 <= first && last <= (long long)MAX_ROOT * MAX_ROOT);
    if (first > last) {
        return 0;
    }
    static std::vector<int> primes = getPrimesTill(MAX_ROOT);
    std::vector<bool> isPrime(last - first + 1, true); // isPrime[i] <-> (i + first) is p
rime
    for (int p : primes) {
        long long p2 = 1LL * p * p;
        if (p2 > last) {
            break;
        }
        long long from = std::max(1LL * p, (first + p - 1) / p) * p;
        assert(from >= first);
        int fromShifted = (int)(from - first);
        int lastShifted = (int)(last - first);
        for (int i = fromShifted; i <= lastShifted; i += p) {
            isPrime[i] = false;
        }
    }
    int count = 0;
    for (int i = 0; i < (int)isPrime.size(); i++) {
        if (isPrime[i] && hasPrimeDigitsOnly(i + first)) {
            count++;
        }
    }
    return count;
}

int main() {
    long long first, last;
    scanf("%lld %lld", &first, &last);
    const int CHUNK = 10 * 1000 * 1000;
    const int CHUNK_FIRST = 2222222; // 7 digits
    const int CHUNK_LAST = 7777777; // 7 digits
    const int LAST_BEFORE_CHUNK = 777777; // 6 digits
    long long count = 0;
    if (last <= LAST_BEFORE_CHUNK) {
        count = countMegaPrimes(first, last);
    } else {
        assert(last > LAST_BEFORE_CHUNK);
        if (first <= LAST_BEFORE_CHUNK) {
            count = countMegaPrimes(first, LAST_BEFORE_CHUNK);
            first = LAST_BEFORE_CHUNK + 1;
        }
        assert(first <= last);
        for (long long partFirst = first / CHUNK * CHUNK + CHUNK_FIRST; partFirst <=
last; partFirst += CHUNK) {
            if (hasPrimeDigitsOnly(partFirst)) {
                long long partLast = partFirst - CHUNK_FIRST + CHUNK_LAST;
                count += countMegaPrimes(std::max(first, partFirst), std::min(last, partL
ast));
            }
        }
    }
    printf("%lld", count);
    return 0;
}
```

Join us on IRC at #hackerrank on freenode for hugs or bugs.

Contest Calendar | Interview Prep | Blog | Scoring | Environment | FAQ | About Us | Support | Careers | Terms Of Service | Privacy Policy | Request a Feature