Sequential Search on unsorted arrays: implementation

- 查找区间[1, N]中的所有素数（质数）

方法0：穷举法 ---- 誰もができる/everybody can do

方法1：筛选法 ---- 誰もが知っている/everybody knows

方法2：合数限定法 ---- 誰もが知らない/nobody knows

方法3：快速线性筛选法 ---- 神のみぞ知る/Only God knows

方法1：筛选法

- （思路）把数值2到N排列起来。标记2是最小的质数，然后删除2后面所有2的倍数（合数）。2后面第一个没删除的数是3，3就是下一个质数，再把3后面3的倍数都删除，找到质数5，再把5后面所有能被5整除的数都删除。这样一直做下去，就会把不超过N的全部合数都筛掉，留下的就是不超过N的全部质数。
- （关键数据结构）bool primes[1..N]

  primes[i]=true\false 标注 数值 i 是否质数

时间复杂度:

$$\Theta\left(n+\frac{n}{2}+\frac{n}{3}+\cdots\frac{n}{p}+\cdots\right)=\Theta(n\,\mathrm{loglog}(n))$$

方法1：筛选法

```
void searching_primes(const int N, Stack<int>& prime_set)
{
    bool* primes = new bool [N+1];
    memset(primes, true, (N+1)*sizeof(bool));  //初始化

    for(int p=2; p<=N; p++)
    {
        if(primes[p]) { //p是质数
                prime_set.push(p);          //入栈

            for(int i=p+p; i<=N; i+=p){
                primes[i] = false;          //排除p的倍数
            }
    }

    delete[] primes;
}
```

方法2：合数限定法

- （思路）筛选法的问题在于一个合数会被多次删除，造成时间浪费。如 6,12,18,36是质数2和3的倍数

  问：整数k会被删除几次？

  答：有多少个不同的素因子就被删除几次

  如12 = 2 x 2 x 3

为使每个合数只删除一次，不能简单地删除质数的所有倍数，而是删除由当前找到的质数合成的数。

方法2：合数限定法

- （关键数据结构）

(1) bool primes[1..N]

(2) Stack<int> BF

设当前找到k-1个质数: $p_1 < p_2 < \cdots < p_{k-1}$

用栈BF记录由这些质数合成的数值

即 BF[i] = $p_1^{n_1} p_2^{n_2} \ldots p_{k-1}^{n_{k-1}}$, $(\forall i \in [1, k) \colon n_i \geq 0)$

且 $\mathrm{BF[i]} < \dfrac{N}{p_{k-1}}$ ⇒ $\boxed{|BF| < \dfrac{N}{p_{k-1}}}$

$\boxed{p_{k-1} < p_k \wedge BF[i] * p_k \leq N}$

方法2：合数限定法

- （计算过程）

设当前找到k-1个质数: $p_1 < p_2 < \ldots < p_{k-1}$

用BF记录由这些质数合成的数值

对第k个素数p_k (primes[p_k] = true)

if (BF[i] * p_k^m <= N) (m = 1,2,…)

primes[BF[i] * p_k^m]= false //从未被删除过?

BF = BF + { BF[i] * p_k^{m-1} }

else

BF = BF - { BF[i] * p_k^{m-1} } //除去不可加大的数

方法2：合数限定法

```
void fast_searching_primes(const int N, Stack<int>& prime_set) {
    bool* primes = new bool [N+1];
    Stack<int> BF(N/2+1); //合数栈
    Stack<int> tBF(N/2+1); //辅助栈
    memset(primes, true, (N+1)*sizeof(bool));

    for(int p=2; p<=N; p++) {
        if(primes[p]) {
            prime_set.push(p);                        //新质数
            while(BF.length()) tBF.push(BF.pop());    //拷贝合数至辅助栈
            tBF.push(p);                              //存入新质数

            while(tBF.length()){
                int tt = tBF.pop();
                while( tt*p <= N){        //计算含质因子p的合数
                    BF.push(tt);          //存入小于或等于N/p的合数
                    tt *= p;
                    primes[tt] = false;
                }
            }
        }
    }
    delete[] primes;
}
```

方法3：快速线性筛选法

- （思路）考虑质数p的倍数 k * p（k>1）
- 合数k*p在筛选法中什么情况下会被多次删除？

$$k = q_1 q_2 \ldots q_j \text{ (素因子 } q_1 \le q_2 \le \cdots \le q_j)$$

$$\text{且 } \exists s \in [1, j]: q_s \ne p$$

- 第一次删除是在查找哪个素数的时候发生的？

$$\min(q_1, p)$$

$$\text{因为 } k * p = \min(q_1, p) * q_2 q_3 \ldots q_j * \max(q_1, p)$$

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

方法3：快速线性筛选法

```
void lineartime_searching_primes(const int N, Stack<int>& prime_set)
{
    bool* primes = new bool [N+1];
    memset(primes, true, (N+1)*sizeof(bool));

    for(int k=2; k<=N; k++)
    {
        if(primes[k]) prime_set.push(k);    //先判断k是否是质数

        for(int j=0; j<prime_set.length(); j++) {

            int p = prime_set[j] * k;        //再把k作为系数，与已找到的质数相乘
            if(p > N) break;
            primes[p] = false;                     //筛选
            if(k%prime_set[j] == 0) break;
                                 //如果第j个质数是k的因数，结束处理
        }

    }
    delete[] primes;
}
```

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	○		✖																	

k = 2

最小质因数：2

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	O	O	✖		✖			✖												

k = 3

最小质因数：3

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	○	○	✖		✖		✖	✖												

k = 4

最小质因数：2

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	○	○	✖	○	✖		✖	✖	✖					✖						

k = 5

最小质因数：5

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	○	○	✖	○	✖		✖	✖	✖		✖			✖						

k = 6

最小质因数：2

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	O	O	✖	O	✖	O	✖	✖	✖		✖		✖	✖						✖

k = 7

最小质因数：7

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	○	○	✖	○	✖	○	✖	✖	✖		✖		✖	✖	✖					✖

k = 8

最小质因数：2

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	○	○	✖	○	✖	○	✖	✖	✖		✖		✖	✖	✖		✖			✖

k = 9

最小质因数：3

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	○	○	✖	○	✖	○	✖	✖	✖		✖		✖	✖	✖		✖		✖	✖

k = 10

最小质因数：2

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	○	○	✖	○	✖	○	✖	✖	✖	○	✖		✖	✖	✖		✖		✖	✖

k = 11

最小质因数：11

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	○	○	✖	○	✖	○	✖	✖	✖	○	✖	○	✖	✖	✖	○	✖	○	✖	✖

k = 12,13,…,21

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

正确性：[1,n]中所有合数都被删除

（证明） 设合数$a \in [1, n]$，且$a = p_1 p_2 \ldots p_j$ (质因数$p_1 \leq p_2 \leq \cdots \leq p_j$)

↓

因为$j > 1$ (?)，设$k = p_2 \ldots p_j$，得到$a = k * p_1$

↓

由于整数k的最小质因数 $p_2 \geq p_1$，根据算法，一定和p_1相乘得a

↓

合数a被删除

方法3：快速线性筛选法

每个整数k只需与小于或等于其最小素因子的质数相乘!!!

Linear time: [1,n]中所有合数只被删除1次！

（反证） 设合数$a \in [1,n]$被删除两次，即存在整数 $k_1 > k_2$，使得

$a = k_1 * p_1 = k_2 * p_2$ （p_1和p_2为质数且 $p_1 < p_2$）

$\Downarrow$

因素分解：$k_1 = q_1 q_2 \ldots q_i \ (i > 0)$ 和 $k_2 = r_1 r_2 \ldots r_j \ (j > 0)$，满足

$p_1 \leq q_1 \leq \cdots \leq q_i$ 和 $p_2 \leq r_1 \leq \cdots \leq r_j$ (算法)

$\Downarrow$

$$a = p_1 * q_1 \ldots q_i = p_2 * r_1 \ldots r_s \ldots r_j$$

$\Downarrow$

$\exists s \in [1,j]$: $p_1 = r_s$ （因数分解唯一性）

矛盾

$\because p_1 < p_2 \wedge p_2 \leq r_1 \leq r_s$ $\therefore p_1 < r_s$