# 筛法

## Sieve

## 线性筛 / 欧拉筛 Linear Sieve

Idea：让每个合数都被最小质因数筛掉，由此保证每个合数只被筛一次。

ATT：线性筛的过程可以处理出每个数的最小质因数。

Complexity：$O(n)$

Code：

```
bool notP[N];
int pList[N], pID;
void Euler(int n){
    notP[0] = notP[1] = 1;
    for(int i = 1; i <= n; i++){
        if(!notP[i])    pList[++pID] = i;
        for(int j = 1; j <= pID; j++){
            if(1ll * i * pList[j] > n)  break;
            notP[i * pList[j]] = 1;
            if(i % pList[j] == 0)   break;
        }
    }
}
```

## 线性筛求欧拉函数

$$\varphi(n) = n \cdot \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right)$$

Idea：设 $p$ 是 $n$ 的最小质因子，那么在线性筛的过程中，$n$ 被 $p \times n'$ 筛掉（对应我的代码中，$p$ 是 `pList[j]`，$n'$ 是 `i`）。

由欧拉函数 $\varphi(n)$ 的定义和性质有：

- 当 $n$ 是质数时，$\varphi(n) = n - 1$
- 当 $p \mid n'$ 时，$n'$ 包含了所有 $n$ 的质因子，于是 $\varphi(n) = n \times \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right) = p \times n' \times \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right) = p \times \varphi(n')$
- 当 $p \nmid n'$ 时，$(p, n') = 1$，由积性：$\varphi(n) = \varphi(p) \times \varphi(n') = (p - 1) \times \varphi(n')$

Code：

```
int phi[N], pList[N], pID;
bool notP[N];
void Euler(int n){
    notP[0] = notP[1] = 1;
    phi[1] = 1;
    for(int i = 1; i <= n; i++){
        if(notP[i] == 0){
            pList[++pID] = i;
            phi[i] = i - 1;
        }
        for(int j = 1; j <= pID; j++){
            if(1ll * i * pList[j] > n)  break;
            notP[i * pList[j]] = 1;
            if(i % pList[j] == 0){
                phi[i * pList[j]] = phi[i] * pList[j];
                break;
            }
            else    phi[i * pList[j]] = phi[i] * (pList[j] - 1);
        }
    }
}
```

# 线性筛求莫比乌斯函数

Idea：由莫比乌斯函数 $\mu(n)$ 的定义和性质有：

- 当 $n$ 是质数时，$\mu(n)=-1$
- 当 $p \mid n'$ 时，$p^2 \mid n$，$\mu(n)=0$
- 当 $p \nmid n'$ 时，$(p,n')=1$，由定义：$\mu(n)=-\mu(n')$

Code：

```
int mu[N], pList[N], pID;
bool notP[N];
void Euler(int n){
    notP[0] = notP[1] = 1;
    mu[1] = 1;
    for(int i = 1; i <= n; i++){
        if(notP[i] == 0){
            pList[++pID] = i;
            mu[i] = -1;
        }
        for(int j = 1; j <= pID; j++){
            if(1ll * i * pList[j] > n)  break;
            notP[i * pList[j]] = 1;
            if(i % pList[j] == 0){
                mu[i * pList[j]] = 0;
                break;
            }
            else    mu[i * pList[j]] = -mu[i];
        }
    }
}
```

# 线性筛求约数个数函数

$$d(n)=\prod_{i=1}^{k}(r_i+1)$$

Idea：由约束个数函数 $d(n)$ 的定义和性质有：

- 当 $n$ 是质数时，$d(n)=2$
- 当 $p \mid n'$ 时，$p$ 在 $n$ 中出现次数比 $n'$ 多一，故 $d(n)=d(n') \times \frac{\text{num}(n')+2}{\text{num}(n')+1}$，其中 $\text{num}(n)$ 表示 $n$ 的最小质因数的次数。
- 当 $p \nmid n'$ 时，$(p,n')=1$，故 $d(n)=d(n') \times 2$

Code：

```
int d[N], num[N], pList[N], pID;
bool notP[N];
void Euler(int n){
    notP[0] = notP[1] = 1;
    d[1] = 1;
    for(int i = 1; i <= n; i++){
        if(notP[i] == 0){
            pList[++pID] = i;
            d[i] = 2, num[i] = 1;
        }
        for(int j = 1; j <= pID; j++){
            if(1ll * i * pList[j] > n)  break;
            notP[i * pList[j]] = 1;
            if(i % pList[j] == 0){
                d[i * pList[j]] = d[i] / (num[i] + 1) * (num[i] + 2);
                num[i * pList[j]] = num[i] + 1;
                break;
            }
            else    d[i * pList[j]] = d[i] * 2, num[i * pList[j]] = 1;
        }
    }
}
```

# 线性筛求约数和函数

$$\sigma(n)=\prod_{i=1}^{k}(1+p_i+{p_i}^2+\cdots+{p_1}^{r_i})=\prod_{i=1}^{k}\frac{1-{p_i}^{r_i+1}}{1-p_i}$$

Idea：由约数和函数 $\sigma(n)$ 的定义和性质有：

- 当 $n$ 是质数时，$\sigma(n)=n+1$
- 当 $p\mid n'$ 时，$p$ 在 $n$ 中出现次数比 $n'$ 多一，故 $\sigma(n)=\sigma(n')\times\frac{1+p+p^2+\cdots+p^{\text{num}(n')+1}}{1+p+p^2+\cdots+p^{\text{num}(n')}}$

  实现时，令 $g(n)=1+p+\cdots+p^{\text{num}(n)}$，则 $\sigma(n)=\sigma(n')\times\frac{p\cdot g(n')+1}{g(n')}$
- 当 $p\nmid n'$ 时，$(p,n')=1$，故 $\sigma(n)=\sigma(n')\times(1+p)$

Code：

```
int sigma[N], g[N], pList[N], pID;
bool notP[N];
void Euler(int n){
    notP[0] = notP[1] = 1;
    sigma[1] = 1, g[1] = 1;
    for(int i = 1; i <= n; i++){
        if(notP[i] == 0){
            pList[++pID] = i;
            sigma[i] = 1 + i, g[i] = 1 + i;
        }
        for(int j = 1; j <= pID; j++){
            if(1ll * i * pList[j] > n)  break;
            notP[i * pList[j]] = 1;
            if(i % pList[j] == 0){
                sigma[i * pList[j]] = sigma[i] / g[i] * (g[i] * pList[j] + 1);
                g[i * pList[j]] = g[i] * pList[j] + 1;
                break;
            }
            else    sigma[i * pList[j]] = sigma[i] * (1 + pList[j]), g[i * pList[j]] = 1 + pList[j];
        }
    }
}
```