Min_25筛

积性函数的前缀和

简介

求积性函数前缀和，线性筛需要把函数的每一位值都算出来，作了许多不必要的操作。
线性筛中，通过计算最小质因子幂的函数值，与之前已计算出的函数值相乘，得到新的函数值。
如果我们能批量执行上面的操作，即用最小质因子幂的函数值，与另一些函数值的和相乘，就可以得到更多的函数值的和。
我们还能发现，当我们要求前n项的和时，大于$\sqrt{n}$的质因子都不能用于计算任何其它的积性函数

于是就有了Min_25筛，将质数分为小于等于$\sqrt{n}$，和大于$\sqrt{n}$的。再预处理出所有质数函数值的前缀和，利用小于等于$\sqrt{n}$的质数进行转移，推出所有的函数值的和。

用途

对于积性函数$F(n)$
在$O(\frac{n^{\frac{3}{4}}}{log\ n})$的复杂度内求出$\sum_{i=1}^{n}F(i)$。
要求对于每个质数p，$F(p)$是简单的多项式，$F(p^k)$很容易算出。

一般题目$n=10^{10}$左右的数量级

过程

预处理

我们需要得到所有质数函数值得前缀和。

首先用线性筛处理出$\leq\sqrt{n}$的质数，$pr[i]$表示第i个质数。

设原积性函数为$F(n)$，即满足若$(a,b)=1$，则$F(ab)=F(a)F(b)$
设p为质数，$F(p)=f_0(p)+f_1(p)+\dots$（必须保证$f(p)=p^k$）。
即质数代入原函数中，得到的简单多项式，每一项都拆成一个函数f，下面的过程将对每一个f单独求和。
*注：即使p不是质数，也可以代入f（p）进行计算，但这并不是原来的积性函数的值。
设$g(i,j)$满足

$$g(i,j)=\sum_{k为质数，或k的最小质因子>pr[j]}^{i}f(k)$$

实际上可以发现，上式中的k，即为线性筛筛了j个质数后，剩余没有被筛掉的数。
因为线性筛用$\leq\sqrt{n}$的质数就可以筛掉n以内的合数，所以我们只预处理了$\leq\sqrt{n}$的质数。（$g(n,|pr|)$即为n以内所有质数的函数值之和，|pr|为$\sqrt{n}$以内的质数个数 ）

$g(i,j)$的转移：

当$pr[j]^2>i$时，显然线性筛中$pr[j]$筛不掉任何数，所以

$$g(i,j)=g(i,j-1)$$

当$pr[j]^2\leq i$时，将上一次的答案减去$pr[j]$筛掉的答案，即最小质因子为$pr[j]$的合数的函数值之和，将这些数的$pr[j]$提出来，剩下的和可以用g表示，公式如下：

$$g(i,j)=g(i,j-1)-f(pr[j])\left(g\left(\left\lfloor\frac{i}{pr[j]}\right\rfloor,j-1\right)-\sum_{k=1}^{j-1}f(pr[k])\right)$$

初始状态$g(i,0)$为$\sum_{k=2}^{i}f(k)$，不管k是不是质数。。（$f(1)$需单独处理）

在实现过程中，可以发现最后$g(i,j)$只需要用到$i=\lfloor\frac{n}{k}\rfloor,j=|pr|$的部分，总共只有$2\sqrt{n}$个位置，可以离散化处理，后面再讲。

递归求和

我们现在已经可以通过$g(i,|pr|)$得到所有质数的函数值前缀和，现在要把它扩展到所有数。
设$S(i,j)$，满足

$$S(i,j)=\sum_{k的最小质因子\geq pr[j]}^{i}F(k)$$

如果$i<pr[j]$，显然$S(i,j)=0$，现在考虑当$i\geq pr[j]$时
$S(i,j)$中质数部分很容易算出，为$g(i,|pr|)-\sum_{k=1}^{j-1}f(pr[k])$
合数部分，可以枚举合数的最小质因子，和最小质因子的指数，将规模缩小，递归求解。

$$S(i,j)=g(i,|pr|)-\sum_{k=1}^{j-1}f(pr[k])+\sum_{k=j}^{pr[k]^2\leq i}\sum_{e=1}^{pr[k]^{e+1}\leq i}F(pr[k]^e)\times S\left(\left\lfloor\frac{i}{pr[k]^e}\right\rfloor,k+1\right)+F(pr[k]^{e+1})$$

末状态为$S(1,j)=0$，和$S(i,j)=0\ (pr[j]>i)$
*注：因为实际上F(n)被拆成了多个fi函数，实际上式中对f(pr[k])求和部分要将每一个fi函数的值求和

结果

$$\sum_{i=1}^{n} F(i)=S(n,1)+F(1)$$

实现

观察发现我们要求的是$S(n,1)$，里面只会访问到$g(n,|pr|)$和$g(\left\lfloor\frac{n}{a_1}\right\rfloor,|pr|)$，$g(\left\lfloor\frac{\frac{n}{a_1}}{a_2}\right\rfloor,|pr|)\dots$
也就是实际上用到的值只有$g(\left\lfloor\frac{n}{i}\right\rfloor,|pr|)$，只有最多$2\sqrt{n}$个位置，预处理出这些位置，离散化如果$\left\lfloor\frac{n}{i}\right\rfloor\le\sqrt{n}$用$id1[\left\lfloor\frac{n}{i}\right\rfloor]$存储离散化之后的下标，如果$\left\lfloor\frac{n}{i}\right\rfloor>\sqrt{n}$用$id2[\left\lfloor\frac{n}{\left\lfloor\frac{n}{i}\right\rfloor}\right\rfloor]$存储它的下标。

因为最后只需要用到$g(\left\lfloor\frac{n}{i}\right\rfloor,|pr|)$，所以第二维不需要开数组存储，转移时之间用一维计算即可。

这样空间复杂度就降为了$O(\sqrt{n})$

后面求S没什么好说的，直接递归。

时间复杂度

不会证

例题：LOJ6053 简单的函数

定义积性函数$F(n)$
$F(p^c)=p\oplus c$

发现除了2以外的质数，$F(p)=p-1$，即可以把F拆成两个fi函数
$f_0(p)=p$
$f_1(p)=1$
然后分别求和计算，用$g(i,j)$表示f_0的和，$h(i,j)$表示f_1的和

在求$S(i,j)$时，如果$j=1$，将答案+2（用于特判p=2）

代码：

```
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
const int MOD=1000000007,MAXN=200005;

long long n,sqr;

bool npr[MAXN];
int pr[MAXN],sum[MAXN],cnt[MAXN],P;
int id1[MAXN],id2[MAXN],m;
long long w[MAXN];
int g[MAXN],h[MAXN];

void LinerSieve()
{
    npr[1]=true;
    for(int i=2;i<=sqr;i++)
    {
        if(!npr[i])
        {
            pr[++P]=i;
            sum[P]=(sum[P-1]+i)%MOD;
        }
        for(int j=1;j<=P&&1LL*i*pr[j]<=sqr;j++)
        {
            npr[i*pr[j]]=true;
            if(i%pr[j]==0)
                break;
        }
    }
}
void Calc_g_h()
{
    for(long long i=1,j;i<=n;i=j+1)
    {
        j=n/(n/i);
        w[++m]=n/i;
        if(w[m]<=sqr)
            id1[w[m]]=m;
```

```
        else
            id2[n/w[m]]=m;
        g[m]=(w[m]%MOD+2)*(w[m]%MOD-1)%MOD*(MOD+1)/2%MOD;
        h[m]=w[m]%MOD-1;
    }
    for(int j=1;j<=P;j++)
        for(int i=1;i<=m&&1LL*pr[j]*pr[j]<=w[i];i++)
        {
            long long k=w[i]/pr[j];
            k=(k<=sqr)?id1[k]:id2[n/k];
            g[i]=(g[i]-1LL*pr[j]*(g[k]-sum[j-1])%MOD+MOD)%MOD;
            h[i]=(h[i]-1LL*(h[k]-(j-1))%MOD+MOD)%MOD;
        }
}
int S(long long x,int j)
{
    if(x<=1||pr[j]>x)
        return 0;
    int k=(x<=sqr)?id1[x]:id2[n/x];
    int res=((g[k]-h[k]-sum[j-1]+(j-1))%MOD+MOD)%MOD;
    if(j==1)
        res=(res+2)%MOD;
    for(int k=j;k<=P&&1LL*pr[k]*pr[k]<=x;k++)
    {
        long long t=pr[k];
        for(int e=1;1LL*pr[k]*t<=x;e++,t*=pr[k])
            res=((res+1LL*(pr[k]^e)*S(x/t,k+1)%MOD)%MOD+(pr[k]^(e+1)))%MOD;
    }
    return res;
}

int main()
{
    scanf("%lld",&n);
    sqr=sqrt(n);
    LinerSieve();
    Calc_g_h();
    int ans=S(n,1)+1;
    printf("%d\n",ans);

    return 0;
}
```