

数论分块

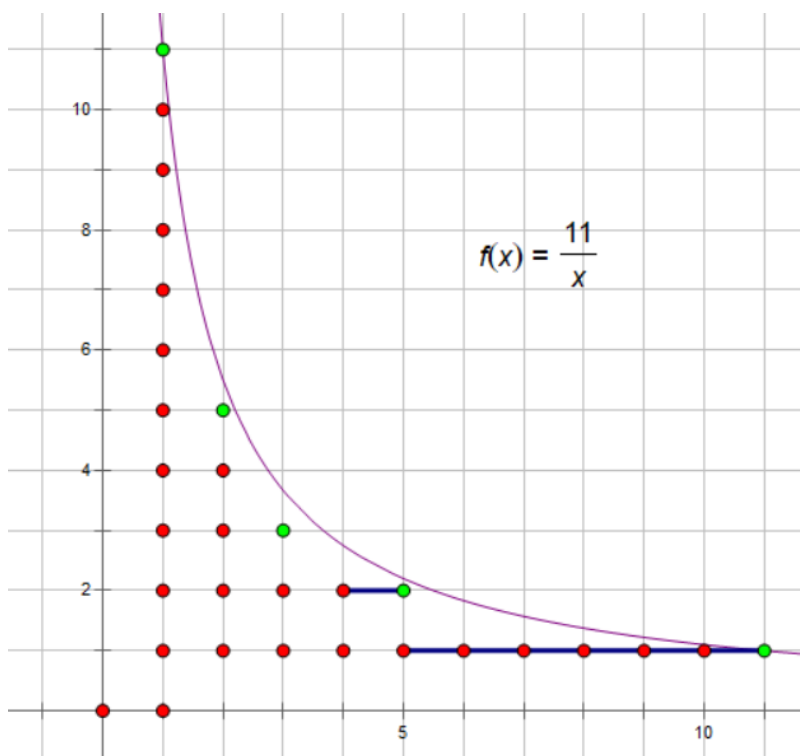
简介

数论分块的目的是：将有除法下取整的式子，从 $O(n)$ 优化到 $O(\sqrt{n})$

它就是换了一种计数顺序，从纵向计数改为横向计数（Fubini原理），将 n/d 相同的数打包同时计算。

（以下若未采用公式体写的 n/d 均代表C语言中带取整的整数除法，而不是数学意义上的除法。）

直观表示就是：我们看下面这个双曲线（的一支）图片，思考双曲线下整点的划分。



图中共分为了5块，这5块整点的最大纵坐标都相同。

数论分块的核心代码很简单：

```
int l=1,r;
while(l<=n)
{
    r=min(n,n/(n/l));
    //中间的部分要具体问题具体分析
    l=r+1;
}
```

这是因为C语言的整数除法，恰好全部都是向下取整。（这里用中括号表示）

因此，关键就在于表达式 $n/(n/l)$ 究竟是什么。即这个表达式：

$$\left\lfloor \frac{n}{\left\lfloor \frac{n}{l} \right\rfloor} \right\rfloor$$

在“分块”计算的时候，对于任意一个 d ，我们需要找到一个最大的 r 使得 $n/d = n/r$ 目的是确定 d 落入了哪一块。

我们指出：表达式 $n/(n/d)$ 恰好就是使得 n/d 不变的那个最大的 r

因此每次将 l 更新为 $r+1$ 就是下一个左端点。上面 $n/(n/l)$ 的式子就是为了寻找图中绿色的点，即每一块的右端点。

证明

首先 $n/(n/l)$ 不比给定的 l 小。这是显然的,把里面的取整符号放缩掉就行。

$$\left\lceil \frac{n}{\left\lfloor \frac{n}{l} \right\rfloor} \right\rceil \geq \frac{n}{\left\lfloor \frac{n}{l} \right\rfloor} = l$$

然后 $n/(n/l)$ 代入（迭代）原式，同理有：

$$\left\lceil \frac{n}{\left\lfloor \frac{n}{\left\lfloor \frac{n}{l} \right\rfloor} \right\rfloor} \right\rceil \geq \left\lfloor \frac{n}{l} \right\rfloor$$

但是由于没取整前，图形是双曲线 n/x 这个函数是单调不增的。对于不同的 x 大小关系，代入函数后大小关系相反。这只能表明：

$$\left\lceil \frac{n}{\left\lfloor \frac{n}{\left\lfloor \frac{n}{l} \right\rfloor} \right\rfloor} \right\rceil = \left\lfloor \frac{n}{l} \right\rfloor$$

这说明 l 和 $n/(n/l)$ 一定位于同一块中。

怎么说明 $n/(n/l)$ 是右端点？只要说明下一个邻居已经不落在区间里就行了。根据带余除法，有：

$$\begin{aligned} n &= x \left\lfloor \frac{n}{x} \right\rfloor + r_1 \\ &= x \left(1 + \left\lfloor \frac{n}{x} \right\rfloor \right) - (x - r_1) \\ &= \left(1 + \left\lfloor \frac{n}{x} \right\rfloor \right) n - (x - r_1) \end{aligned}$$

其中 r_2 非负 $x - r_1$ 是严格大于0的正整数。这样，我们就证明了：

$$x \left(1 + \left\lfloor \frac{n}{x} \right\rfloor \right) < \left(1 + \left\lfloor \frac{n}{x} \right\rfloor \right) \left\lfloor \frac{n}{1 + \left\lfloor \frac{n}{x} \right\rfloor} \right\rfloor$$

代入 x 为 n/l

$$\left\lfloor \frac{n}{l} \right\rfloor < \left\lfloor \frac{n}{1 + \left\lfloor \frac{n}{\left\lfloor \frac{n}{l} \right\rfloor} \right\rfloor} \right\rfloor$$

n/l 严格比 $n/(1+(n/(n/l)))$ 小，因此原命题也就证完了。

除法取整的突变问题

上面的讨论，解决了 n/d 在下方的 d 不断增加的情况下，什么时候函数值发生突变。并且，这种增加是单向的，计算时只能让 d 从小往大变化，因为采用这种方法无法找到左端点。

那么如果下方的 d 不变，上方的 n 变化，会发生什么？答案是变简单了。

考虑表达式 $\lfloor \frac{n}{d} \rfloor$ 和 $\lfloor \frac{n-1}{d} \rfloor$ 除以 d 的商取整之差。

$$\left\lfloor \frac{n}{d} \right\rfloor - \left\lfloor \frac{n-1}{d} \right\rfloor$$

根据带余除法的定义，有：

$$n = d \left\lfloor \frac{n}{d} \right\rfloor + r_1 \quad n-1 = d \left\lfloor \frac{n-1}{d} \right\rfloor + r_2$$

r_1 和 r_2 是余数，都在 0 到 $d-1$ 之间。因此这个表达式，仅当 d 整除 n 的时候相差 1 ，其他时候均为 0 。

一个常用引理

引理

一个狄利克雷卷积式的推广，本式有两个变量 n 和 a 。当 n 和 a 相等的时候，就是一个标准的狄利克雷卷积式。

$$\sum_{i=1}^n (i, a) = \sum_{d|a} \left\lfloor \frac{n}{d} \right\rfloor \varphi(d)$$

它的推论是：

$$\sum_{i=1}^r (i, a) = \sum_{d|a} \left\lfloor \frac{r}{d} \right\rfloor - \left\lfloor \frac{r-1}{d} \right\rfloor \varphi(d)$$

证明

由狄利克雷卷积 $\varphi * 1 = n$ 有：

$$(n, a) = \sum_{d|(a, n)} \varphi(d) = \sum_{d|a \ \&\& \ d|n} \varphi(d)$$

根据上面的讨论，有：

$$(n, a) = \sum_{d|a} \left(\left\lfloor \frac{n}{d} \right\rfloor - \left\lfloor \frac{n-1}{d} \right\rfloor \right) \varphi(d)$$

利用数学归纳法对 n 归纳，或两边同时计算部分和，就证明了原命题。

例题

题目

计算：

$$\sum_{i=1}^n (\lfloor \sqrt[3]{i} \rfloor, i) \pmod{998244353} \quad n \leq 10^{21}$$

题解

分析这个问题。完全立方数将 1 到 n 划分为许许多多左闭右开的整数区间，那么最后一个区间是不完全的。因此对立方数进行划分，并单独提取出最后一个不完全区间：

$$\sum_{i=1}^n (\left\lceil \sqrt[3]{i} \right\rceil, i) = \sum_{i=1}^{\left\lceil \sqrt[3]{n} \right\rceil - 1} \sum_{j=i^3}^{(i+1)^3 - 1} (i, j) + \sum_{i=\left\lceil \sqrt[3]{n} \right\rceil}^n (\left\lceil \sqrt[3]{n} \right\rceil, i)$$

根据引理，对于原式右半部分的内容我们便可以通过数论分块在 $O(\sqrt[6]{n})$ 的时间内解决。

$$\sum_{i=\left\lceil \sqrt[3]{n} \right\rceil}^n (\left\lceil \sqrt[3]{n} \right\rceil, i) = \sum_{d|\left\lceil \sqrt[3]{n} \right\rceil} \left(\left\lceil \frac{n}{d} \right\rceil - \left\lceil \frac{\left\lceil \sqrt[3]{n} \right\rceil^3 - 1}{d} \right) \varphi(d)$$

继续展开左边的式子。由求和式中d整除i 设变量x满足 $xd=i$ 交换求和次序并去取整号整理得：

$$\begin{aligned} & \sum_{i=1}^{\left\lceil \sqrt[3]{n} \right\rceil - 1} \sum_{j=i^3}^{(i+1)^3 - 1} (i, j) \\ &= \sum_{i=1}^{\left\lceil \sqrt[3]{n} \right\rceil - 1} \sum_{d|i} \left(\left\lceil \frac{(i+1)^3 - 1}{d} \right\rceil - \left\lceil \frac{i^3 - 1}{d} \right\rceil \right) \varphi(d) \\ &= \sum_{d=1}^{\left\lceil \sqrt[3]{n} \right\rceil - 1} \varphi(d) \sum_{x=1}^{\left\lceil \frac{\left\lceil \sqrt[3]{n} \right\rceil}{d} \right\rceil - 1} \left(\left\lceil \frac{(xd+1)^3 - 1}{d} \right\rceil - \left\lceil \frac{(xd)^3 - 1}{d} \right\rceil \right) \\ &= \sum_{d=1}^{\left\lceil \sqrt[3]{n} \right\rceil - 1} \varphi(d) \sum_{x=1}^{\left\lceil \frac{\left\lceil \sqrt[3]{n} \right\rceil}{d} \right\rceil - 1} \left(3dx^2 + 3x + 1 \right) \end{aligned}$$

接下来就是平方和公式和等差数列求和，设仅与d相关的y为：

$$y = \left\lceil \frac{\left\lceil \sqrt[3]{n} \right\rceil - 1}{d} \right\rceil$$

得：

$$\sum_{i=1}^{\left\lceil \sqrt[3]{n} \right\rceil - 1} \sum_{j=i^3}^{(i+1)^3 - 1} (i, j) = \sum_{d=1}^{\left\lceil \sqrt[3]{n} \right\rceil - 1} \varphi(d) d \frac{y(y+1)(2y+1)}{2} + \sum_{d=1}^{\left\lceil \sqrt[3]{n} \right\rceil - 1} \varphi(d) \left(\frac{y(y+1)}{2} + y \right)$$

y 也是一个除以d后取整的形式，故依旧可以用数论分块维护。总和式为：

$$\sum_{i=1}^n (\left\lceil \sqrt[3]{i} \right\rceil, i) = \sum_{d=1}^{\left\lceil \sqrt[3]{n} \right\rceil - 1} \varphi(d) d \frac{y(y+1)(2y+1)}{2} + \sum_{d=1}^{\left\lceil \sqrt[3]{n} \right\rceil - 1} \varphi(d) \left(\frac{y(y+1)}{2} + y \right) + \sum_{d|\left\lceil \sqrt[3]{n} \right\rceil} \left(\left\lceil \frac{n}{d} \right\rceil - \left\lceil \frac{\left\lceil \sqrt[3]{n} \right\rceil^3 - 1}{d} \right) \varphi(d)$$

通过 $O(\sqrt[3]{n})$ 预处理出 $\sum \varphi(i)$ 和 $\sum \varphi(i)$ ，就可以在 $O(\sqrt[6]{n})$ 的时间内处理每一组询问了。总时间复杂度 $O(\sqrt[3]{n} + \sqrt[6]{n} * T)$

注意，读入要用 `__int128`，但是在开数组的时候都要开 `int` 否则会爆空间。

代码

```
#include<bits/stdc++.h>

using namespace std;
```

```

inline __int128 read()
{
    __int128 x=0,f=1;
    char c=getchar();
    while(!isdigit(c))
    {
        if(c=='-')f=-1;
        c=getchar();
    }
    while(isdigit(c))
    {
        x=x*10+c-'0';
        c=getchar();
    }
    return x*f;
}

__int128 n,ans;
const int MOD=998244353,maxN=10000000;
int
T,prime[maxN+10],len,A,sqrt3N,phi[maxN+10],phii[maxN+10],pre[maxN+10],inv2=
499122177;
bool vis[maxN+10];

void calc()//线性筛计算欧拉函数
{
    phi[1]=1;
    phii[1]=1;
    int i;
    for(i=2;i<=maxN;i++)
    {
        if(!vis[i])
        {
            prime[++len]=i;
            phi[i]=i-1;
        }
        int j;
        for(j=1;j<=len&& i*prime[j]<=maxN;j++)
        {
            vis[i*prime[j]]=1;
            if(i%prime[j]==0)
            {
                phi[i*prime[j]]=phi[i]*prime[j];
                break;
            }
            phi[i*prime[j]]=phi[i]*(prime[j]-1);
        }
    }
    for(i=1;i<=maxN;i++)//欧拉函数乘自变量，其实也是个积性函数
    {
        phii[i]=((long long)i*(long long)phi[i])%MOD;
    }
}

```

```
    }
    for(i=1;i<=maxN;i++)//部分和
    {
        pre[i]=((long long)pre[i-1]+(long long)phi[i])%MOD;
        phii[i]=((long long)phii[i-1]+(long long)phii[i])%MOD;
    }
}

__int128 sqrt3(__int128 N)//二分
{
    __int128 l=0,r=1e9;
    while(r-l>1)
    {
        __int128 mid=(l+r)/2;
        if(mid*mid*mid<N)l=mid;
        else r=mid;
    }
    return (r*r*r<=N) ? r : l;
}

int main()
{
    calc();
    scanf("%d",&T);
    while(T--)
    {
        n=read();
        sqrt3N=(int)sqrt3(n);
        __int128 ans=0;
        for(int d=1;d*d<=sqrt3N;d++)
        {
            if(sqrt3N%d==0)
            {
                ans=(ans+(n/d-
                ((__int128)sqrt3N*((__int128)sqrt3N*((__int128)sqrt3N-1)/d)%MOD*phi[d])%MOD;
                if(d*d!=sqrt3N)
                {
                    int t=sqrt3N/d;
                    ans=(ans+(n/t-
                    ((__int128)sqrt3N*((__int128)sqrt3N*((__int128)sqrt3N-1)/t)%MOD*phi[t])%MOD;
                }
            }
        }
        for(int l=1,r=0;l<=sqrt3N-1;l=r+1)
        {
            int x=(sqrt3N-1)/l;
            r=min(sqrt3N-1,(sqrt3N-1)/((sqrt3N-1)/l));//分块操作
            long long tmp1=(phii[r]-phii[l-1]+MOD)%MOD,tmp2=(pre[r]-
            pre[l-1]+MOD)%MOD;
            ans=(ans+tmp1*(long long)inv2%MOD*x%MOD*(x+1)%MOD*(2*x+1))%MOD;
```

```
ans=((ans+tmp2*(x+1)%MOD*x%MOD*inv2%MOD*3%MOD)%MOD+x*tmp2%MOD)%MOD;
    }
    cout<<(long long)ans<<endl;
}
return 0;
}
```

From:

<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:

https://wiki.cvbbacm.com/doku.php?id=technique:number_theory_sqrt_decomposition&rev=1591369877 

Last update: 2020/06/05 23:11