

数论

整除

如果 k_1, k_2 互质, 则 $k_1 + k_2$ 与 $k_1 \times k_2$ 互质。

在自然数集中, 小于 n 的质数约有 $\frac{n}{\ln(n)}$ 个。

切比雪夫定理

1. 对整数 $n \geq 3$ 则至少存在一个质数 p 符合 $n \leq p \leq 2n-2$

2. 对任意自然数 $n > 6$ 至少存在一个 $4k+1$ 型和一个 $4k+3$ 型素数 p 使得 $n < p < 2n$

3. 对任意自然数 k 存在自然数 N 对任意自然数 $n > N$ 至少存在 k 个素数 p 使得 $n < p < 2n$

Miller-Rabin

Miller-Rabin 的复杂度是 $O(k \log n)$ 其中 k 是测试次数。

质数筛法

埃氏筛

思想：从小到大枚举分析每一个数，然后同时把当前这个数的所有（比自己大的）倍数记为合数，那么运行结束的时候没有被标记的数就是素数了。

```
int v[N];
void primes(int n) {
    memset(v, 0, sizeof v);
    for(int i = 2; i <= n; ++ i){
        if(v[i]) continue;
        for(int j = i; j <= n / i; ++ j) v[i * j] = 1;
    }
}
```

时间复杂度 $O(n \log_{10} \log_{10} n) \approx O(n)$ 所以它的时间复杂度其实是劣于线性筛的。这里补充自然数以及合数的和都是 $O(\log_{10} n)$ 质数为 $O(\log_{10} \log_{10} n)$

虽然其时间复杂度比较劣，但这种思想是很值得学习的。如果需要筛一个 $[L, R]$ 的区间内的素数，我们需要先看 $\sqrt{R}$ 的范围，然后预处理出这个范围内的素数。然后从小到大枚举素数，找到不小于 L 的最小 p 的倍数，且不能是 p 本身，然后按照这个筛法打标记，复杂度是 $O(R-L + \sqrt{R})$

代码如下：

```
memset(st, 0, sizeof st);
for (int i = 0; i < cnt; i++) {
    LL p = primes[i]; // 先筛一遍
    for (LL j = max(p * 2, (l + p - 1) / p * p); j <= r; j += p)
        st[j - l] = true;
}
```

线性筛（欧拉筛）

扫到一个数字 i 时，如果没有标记过，则为质数。

不然 i 为合数，考虑将质数数组中从小到大开始给 $i \times \text{prime}[j]$ 打标记，直到 $i \% \text{prime}[j] \neq 0$ 。这时我们找到了 i 的最小质因子 $\text{prime}[j]$ 。在此之前的质因子全都比 i 的最小质因子要小，所以打标记的数字，都是因为枚举到了这个数字的最小质因子才打的，之后的数字，因为都比最小质因子要大，所以不标记，直接 `break`。所以每个数字因为只有一个最小质因子，所以只枚举了一次，复杂度为 $O(n)$ 。并且我们在筛的同时，也拿到了 $1 \sim n$ 每个数字的最小质因子。

```
const int N = 10005;
int n, primes[N], cnt, min_prime[N];
bool vis[N];
inline void get_prime() {
    for (register int i = 2; i <= n; i++) {
        if (!vis[i]) primes[cnt++] = i;
        for (register int j = 1; j <= cnt && i * primes[j] <= n; ++j) {
            vis[i * primes[j]] = 1;
            if (i % primes[j] == 0) {
                min_prime[i] = primes[j]; // 最小质因子
                break;
            }
        }
    }
}
```

反素数

如果 n 是 $1 \dots n$ 中正约数个数最多的数，且唯一，也就是约数最多且最小，那么 n 就是反素数。

若 $N \leq 2^{31}$ ， $1 \dots N$ 中任何数的不同质因子都不会超过 10 个且所有质因子的质数都不会超过 30。因为光 2 乘到 31 这个数都比 N 大。所以反素数都可以表示为 $2^{c_1} \times 3^{c_2} \times 5^{c_3} \times 7^{c_4} \times 11^{c_5} \times 13^{c_6} \times 17^{c_7} \times 19^{c_8} \times 23^{c_9} \times 29^{c_{10}}$ 其中 c 数组递减。

所以我们可以直接 `dfs` 找到前 10 个质数

例 \$1\$

题目

给定一个正整数 n ，输出最小的整数，满足这个整数有 n 个因子，即求因子数一定的最小反素数。

题解

按上面的剪剪枝，乱写就行了。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;

int num[12]={2,3,5,7,11,13,17,19,23,29};
int cnt[12];
int n;
ull ans=9e18;
void dfs(int now,ull q,int pos) {
    if(now==n) {
        if(q<ans) ans=q;
        return;
    }
    if(pos>=10) return;
    ull tmp=q;
    for(int i=1;(i+1)*now<=n;i++) {
        if(pos!=0) {
            if(i>cnt[pos-1]) break;
        }
        tmp*=num[pos];
        if (tmp>2e18) break;
        if(n/now%(i+1)!=0) continue;
        cnt[pos] = i;
        dfs(now*(i+1),tmp,pos+1);
    }
}

int main() {
    scanf("%d",&n);
    dfs(1,1,0);
    printf("%llu",ans);
    return 0;
}
```

\$Pollard Rho\$

时间复杂度 $O(n^{\frac{1}{4}})$ 用来找到 n 的一个素因子 p 每次找到就一直除它，最不利的情况是每个素因子都是一次幂，所以全部分解的复杂度正常是 $O(n^{\frac{1}{4}} \log n)$ 的，但因为质因数越多时大小都不一样，正常的对数级别最后实际上也就是常数级别的影响，所以完全分解也可以看作是 $O(n^{\frac{1}{4}})$ 的。

例 2

题目

<https://nanti.jisuanke.com/t/42544>

给 t 组数据 $(t \leq 8)$ 每组数据给一个 $n, x, y, n \leq 10^5, 2 \leq x, y \leq 10^{18}$ 代表一个长度为 n 的数组 $a_i, a_i \leq 10^{18}$ 且保证 a_i 之和小于 y 现在定义 $Z = \{\prod_{i=1}^n a_i!\}$ 求最大的 i 使得 $(Z \times X^i) | Y!$

题解

显然对着一堆阶乘使劲是不可以的，显然 $Y!$ 可以约掉 Z 所有的因子。剩下看 X 都有什么因子，然后提前处理出 $Y!$ 和 Z 中这些因子的幂次，做一个差，然后取所有剩余幂次/一个 X 中有多少个幂次，就是能取多少个 X 然后取最小值就是答案。而 X 的质因数分解，显然需要 `pollard_rho`

竟然一遍过了 $\times$ 。

```
#include <bits/stdc++.h>
#define ll __int128
using namespace std;
ll maxv;
inline ll quick_mul(ll a, ll b, ll p) {
    unsigned long long c = (long double) a/p*b;
    ll ret = a*b - (unsigned long long) c*p;
    ret %= p;
    while (ret < 0) ret += p;
    return ret % p;
}
inline ll quick_power(ll a, ll b, ll p) {
    ll ret = 1;
    while (b) {
        if (b & 1) ret = quick_mul(ret, a, p);
        a = quick_mul(a, a, p);
        b >>= 1;
    }
    while (ret < 0) ret += p;
    return ret % p;
}
```

```
}
bool check(ll a,ll n,ll x,ll t) {
    ll ans=quick_power(a,x,n);
    ll aans=ans;
    for(int i=1; i<=t; i++) {
        ans=quick_mul(ans,ans,n);
        if(ans==1&&aans!=1&&aans!=n-1) return true;
        aans=ans;
    }
    if(ans!=1) return true;
    return false;
}
bool Miller_Rabin(ll n) {
    if(n<2) return false;
    if(n==2) return true;
    if(!(n&1)) return false;
    ll x=n-1,t=0;
    while(!(x&1)) {
        x>>=1;
        t++;
    }
    for(int i=0; i<8; i++) { //8为测试次数
        ll a=rand()%(n-1)+1;
        if(check(a,n,x,t)) return false;
    }
    return true;
}
ll factor[1010],num;
ll gcd(ll x,ll y) {
    if(!x) return y;
    if(!y) return x;
    if(x<0) x=-x;
    if(y<0) y=-y;
    ll t=__builtin_ctzll(x|y);
    x>>=__builtin_ctzll(x);
    y>>=__builtin_ctzll(y);
    if(x>y) swap(x,y);
    y-=x;
} while(y);
return x<<t;
}
ll pollard_rho(ll x,ll c) {
    ll ci=1,k=2;
    srand(time(NULL));
    ll x0=rand()%(x-1)+1;
    ll y=x0,t=1;
    while(1) {
        ci++;
        x0=(quick_mul(x0,x0,x)+c)%x;
        t=quick_mul(y-x0,t,x);
    }
```

```
        if(!t||!(y^x0)) return x;
        if(ci==k) {
            ll d=gcd(t,x);
            if(d!=1) return d;
            y=x0;
            k<<=1;
        }
    }
}

void findfac(ll n,ll k) {
    if(n==1) return;
    if(Miller_Rabin(n)) {
        factor[++num] = n;
        return;
    }
    ll p=n,c=k;
    while(p>=n) {
        p=pollard_rho(p,c--);
    }
    findfac(p,k);
    findfac(n/p,k);
}

inline void read(ll &X) {
    X = 0;
    int w=0;
    char ch=0;
    while(!isdigit(ch)) {
        w|=ch=='-';
        ch=getchar();
    }
    while(isdigit(ch)) X=(X<<3)+(X<<1)+(ch^48),ch=getchar();
    if (w) X = -X;
}

void print(ll x) {
    if (!x) return ;
    if (x < 0) putchar('-'),x = -x;
    print(x / 10);
    putchar(x % 10 + '0');
}

ll n,x,y,a[100100];
ll tmpfac[10010],tmps,tmpnum[10010];
ll js[100100][17],zs[17];
int main() {
    int t;
    scanf("%d",&t);
    while(t--) {
        num=0;
        tmps = 0;
        read(n);
```

```
read(x);
read(y);
for(int i=1; i<=n; i++) {
    read(a[i]);
}
findfac(x,324757);
sort(factor+1,factor+num+1);
ll las=factor[1];
tmpfac[++tmps] = las;
tmpnum[tmps]=1;
for(int i=2; i<=num; i++) {
    if(factor[i]==las) {
        tmpnum[tmps]++;
    } else {
        las = factor[i];
        tmpfac[++tmps] = factor[i];
        tmpnum[tmps] = 1;
    }
}
for(int i=1; i<=tmps; i++) {
    for(int j=1; j<=n; j++) {
        ll ansss=0;
        ll kkk = tmpfac[i];
        for(ll k=a[j]; k; k=k/kkk) {
            ansss+=k/kkk;
        }
        js[j][i]=ansss;
    }
}
for(int i=1; i<=tmps; i++) {
    for(int j=2; j<=n; j++) js[1][i] += js[j][i];
}
for(int i=1; i<=tmps; i++) {
    ll ansss=0;
    ll kkk = tmpfac[i];
    for(ll k=y; k; k=k/kkk) {
        ansss+=k/kkk;
    }
    zs[i] = ansss;
}
for(int i=1;i<=tmps;i++) zs[i]-=js[1][i];
ll maxv=1e18;
for(int i=1;i<=tmps;i++) {
    maxv=min(maxv,zs[i]/tmpnum[i]);
}
if(maxv) print(maxv);
else putchar('0');
putchar('\n');
}
return 0;
```

```
}
```

威尔逊定理

若 p 为素数，则 $(p-1)!+1 \equiv p \pmod p$

原根

如果 a 模 m 的阶等于 $\phi(m)$ 则称 a 为模 m 的一个原根。素数一定有原根，原根不唯一，部分合数也有原根。

1000000007 的原根为 5。998244353 的原根为 3。

最大公约数和最小公倍数

求 n 的正约数集合

复杂度是 $O(\sqrt{n})$ 的。

```
vector<int>factor;
int m;
int main() {
    scanf("%d", &n);
    for(int i = 1; i * i <= n; ++ i) {
        if(n % i == 0) {
            factor.push_back(i);
            if(i != n / i)
                factor.push_back(n / i);
        }
    }
    for(int i = 0; i < factor.size(); ++ i) {
        printf("%d\n", factor[i]);
    }
    return 0;
}
```

求 $1 \dots n$ 中每个数的正约数的集合

求 $1 \dots n$ 中每个数的正约数的集合，算法复杂度 $O(n \log n)$ 总数也是这个级别的。

方法倍数法：类似于埃氏筛的思路，枚举倍数，如果能除开一定会被发现，并且因为每轮倍数不同，所以只发现一次。


```

#include <bits/stdc++.h>
#define ll long long
using namespace std;

int n;
vector<int> factor[100100];

int main() {
    scanf("%d", &n);
    n=100000;
    for(int i = 1; i <= n; ++ i) {
        for(int j = 1 ; j * i <= n; ++ j) {
            factor[i * j].push_back(j);
        }
    }
    ll ans=0;
    for(int i=1;i<=n;i++) {
        ans+=factor[i].size();
    }
    printf("%lld\n",ans);

    return 0;
}

```

一些结论

$$\gcd(F(n), F(m)) = F(\gcd(n, m))$$

$$\gcd(a^m - 1, a^n - 1) = a^{\gcd(n, m)} - 1 \quad (a > 1, n > 0, m > 0)$$

更进一步，有：

$$\gcd(a^m - b^m, a^n - b^n) = a^{\gcd(n, m)} - b^{\gcd(n, m)} \quad (\gcd(a, b) = 1)$$

$$\text{设 } G = \gcd(C(n, 1), C(n, 2), \dots, C(n, n-1))$$

1. 若 n 为素数 $G = n$ 因为分母不含 p 分子有 p

2. n 非素且有只有一个素因子 p $G = p$ $n = p^a$ 显然分母幂次小于分子幂次，取 $C(n, p^{a-1})$ 可以证明分子只比分母多一次，所以 $G = p$

3. n 有多个素因子 $G = 1$ 将 n 做质因数分解，设 p_1 的幂次是 a_1 取 $C(n, p_1^{a_1})$ 可以证明没有 p_1 的幂次。同理也有没有 p_i 幂次的数，所以所有的最大公约数是 1。

$$(n+1) \mid \text{lcm}(C(n, 0), C(n, 1), \dots, C(n, n)) = \text{lcm}(1, 2, \dots, n+1)$$

$$\sum_{i=1}^n \gcd(i, n) = \sum_{d|n} d \cdot \phi\left(\frac{n}{d}\right)$$

斐波那契数列

前缀和 $\sum_{i=1}^n f_i = f_{n+2} - 1$ 数学归纳法。

奇数项前缀和 $\sum_{i=1}^n f_{2i-1} = f_{2n}$ 数学归纳法。

偶数项前缀和 $\sum_{i=1}^n f_{2i} = f_{2n+1} - 1$ 数学归纳法。

平方前缀和 $\sum_{i=1}^n f_i^2 = f_n f_{n+1}$ 数学归纳法

$$f_{n+m} = f_{n-1}f_{m-1} + f_n f_m$$

$$f_n^2 = (-1)^{n-1} + f_{n-1}f_{n+1}$$

$$f_{2n-1} = f_n^2 - f_{n-2}^2$$

$$f_n = \frac{f_{n+2} + f_{n-2}}{3}$$

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

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:%E7%8E%8B%E6%99%BA%E5%BD%AA:%E6%95%B0%E8%AE%BA

Last update: 2021/09/25 19:16