

辗转相除法

POJ 2429: GCD & LCM Inverse

显然 $\gcd(a,b) \mid \text{lcm}(a,b)$ 原因在于 $\text{lcm}(a,b)$ 质因数分解的结果必然包含 $\gcd(a,b)$ 的所有质因子。

因此 $\text{lcm}(a,b)$ 质因数分解的结果中不包含在 $\gcd(a,b)$ 的所有质因子中的部分可以进行调整。即需要分解 $\text{lcm}(a,b)/\gcd(a,b)$ 的质因子。然后爆搜这些质因子的组合方式。

(注意其中每种质因子只可能属于 a 或者 b ，因为如果 $\text{lcm}(a,b)/\gcd(a,b)$ 的一个质因子在 a 和 b 中均出现，则 $\gcd(a,b)$ 必然增大)

但是直接分解会报错，因为 $\text{lcm}(a,b)/\gcd(a,b)$ 太大，复杂度为 $O(\sqrt{n})$ 的朴素算法不行。

于是采用miller判断质数，再用pollard-rho来分解质因数，复杂度为 $O(n^{\frac{1}{4}})$ 。代码如下

```
/*
```

```
*/
```

```
#define method_1
```

```
#ifdef method_1
```

```
/*
```

显然 $\gcd(a,b) \mid \text{lcm}(a,b)$ 原因在于 $\text{lcm}(a,b)$ 质因数分解的结果必然包含 $\gcd(a,b)$ 的所有质因子。

因此 $\text{lcm}(a,b)$ 质因数分解的结果中不包含在 $\gcd(a,b)$ 的所有质因子中的部分可以进行调整。

即需要分解 $\text{lcm}(a,b)/\gcd(a,b)$ 的质因子。然后爆搜这些质因子的组合方式。

(注意其中每种质因子只可能属于 a 或者 b ，因为如果 $\text{lcm}(a,b)/\gcd(a,b)$ 的一个质因子在 a 和 b 中均出现，则 $\gcd(a,b)$ 必然增大)

但是直接分解会报错，因为 $\text{lcm}(a,b)/\gcd(a,b)$ 太大，复杂度为根号 n 的朴素算法不行。

于是采用miller判断质数，再用pollard-rho来分解质因数，复杂度为 n 的四分之一方。

```
*/
```

```
#include<iostream>
```

```
#include<cstdio>
```

```
#include<algorithm>
```

```
#include<cmath>
```

```
#include<set>
```

```
#include<map>
```

```
#include<queue>
```

```
#include<stack>
```

```
#include<vector>
```

```
#include<cstring>
```

```
#include<cstdlib>
```

```
#include<iomanip>
```

```

#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<"
#define E cout<<endl
using namespace std;
typedef unsigned long long ull;
typedef pair<int,int>pii;
const int maxn=128+5;
const int times=50;// 测试次数
const ull INF=0x3f3f3f3f3f3f3f3full*2;
ull x,y,a[maxn],ans1,ans2,maxx,pos,sz;//gcd(a,b)=x,lcm(a,b)=y
int b[maxn];
map<ull,int>mp;
ull mul(ull a,ull b,ull p){ //cal(a*b%p)
    ull ans=0;
    while(b){
        if(b&1) ans=(ans+a)%p;
        b>>=1;a=(a+a)%p;
    }
    return ans;
}
ull pow(ull a,ull b,ull p){ //cal(a^b%p)
    ull ans=1%p;
    while(b){
        if(b&1) ans=mul(ans,a,p);
        b>>=1;a=mul(a,a,p);
    }
    return ans;
}
bool Miller_Rabin(ull n, int repeat){ //n是测试的大数, repeat是测试重复次数
    if(n == 2 || n == 3)return true;//特判
    if(n % 2 == 0 || n == 1)return false;//偶数和1
    //将n-1分解成2^s*d
    ull d = n - 1;
    int s = 0;
    while(!(d & 1)) ++s, d >>= 1;
    for(int i = 0; i < repeat; i++)//重复repeat次
    {
        ull a = rand() % (n - 3) + 2;//取一个随机数,[2,n-1)
        ull x = pow(a, d, n);
        ull y = 0;
        for(int j = 0; j < s; j++)
        {
            y = mul(x, x, n);
            if(y == 1 && x != 1 && x != (n - 1))return false;
            x = y;
        }
    }
}

```

```

        if(y!=1)return false;//费马小定理
    }
    return true;
}
ull gcd(ull a, ull b)
{
    return !b?a:gcd(b,a%b);
}
ull pollard_rho(ull n, ull c)//找到n的一个因子
{
    ull x = rand() % (n - 2) + 1;
    ull y = x, i = 1, k = 2;
    while(1){
        i++;
        x = (mul(x, x, n) + c) + n;//不断调整x2
        ull d = gcd(y - x, n);
        if(1 < d && d < n)
            return d;//找到因子
        if(y == x)
            return n;//找到循环, 返回n, 重新来
        if(i == k)//一个优化
        {
            y = x;
            k <= 1;
        }
    }
}
void Find(ull n, ull c)
{
    if(n == 1)return;//递归出口
    if(Miller_Rabin(n, times)){//如果是素数, 就加入
        mp[n]++;
        return;
    }
    ull p = n;
    while(p >= n)
        p = pollard_rho(p, c--);//不断找因子, 直到找到为止, 返回n说明没找到
    Find(p, c);
    Find(n / p, c);
}
ull pow2(ull a,ull b){ //cal(a^b)
    ull ans=1;
    while(b){
        if(b&1) ans=ans*a;
        b>>=1;a=a*a;
    }
}

```

```

    }
    return ans;
}
void init(){
    mp.clear();maxx=INF;pos=0;
}
int main() {
    ios::sync_with_stdio(false);
    //freopen("GCD & LCM Inverse.in","r",stdin);
    // srand((unsigned)time(NULL));
    while(cin>>x>>y){
        init();
        if(x==y){cout<<x<<" "<<y<<endl;continue;}
        y/=x;//y=lcm(a,b)/gcd(a,b)
        Find(y,180);
        map<ull,int>::iterator it;
        for(it=mp.begin();it!=mp.end();it++) a[pos]=it->first,b[pos]=it-
>second,pos++;
        sz=mp.size();
        /*
        for(int i=0;i<sz;i++){
            D(a[i]);D(b[i]);E;
        }
        */
        for(ull i=0;i<(1<<sz);i++){
            ull tot=1;
            for(int j=0;j<sz;j++){
                if(i&(1<<j)){
                    tot*=pow2(a[j],b[j]);
                }
            }
            if(maxx>tot+y/tot){
                maxx=tot+y/tot;ans1=min(tot,y/tot),ans2=max(tot,y/tot);
            }
        }
        cout<<ans1*x<<" "<<ans2*x<<endl;
    }
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

```

#endif

[POJ 1930: Dead Fraction](https://www.cnblogs.com/akrusher/p/5335127.html) 题解链接 <https://www.cnblogs.com/akrusher/p/5335127.html>

代码如下

```
/*

*/

#define method_1
#ifdef method_1
/*
https://www.cnblogs.com/akrusher/p/5335127.html
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=+5;
const int INF=0x3f3f3f3f;
int gcd(int a,int b){
    return !b?a:gcd(b,a%b);
}
string s;
int num,len; //num记录小数部分 len记录小数部分的位数
int min_denominator,numerator; //分母 分子
void init(){
    num=0;
    min_denominator=INF;
}
int pow2(int a,int b){ //calc a^b
    int ans=1;
    for(int i=1;i<=b;i++) ans*=a;
```

```

        return ans;
    }
int main() {
    ios::sync_with_stdio(false);
    //freopen("Dead Fraction.in", "r", stdin);
    while(cin>>s){
        if(s=="0") break;
        init();
        for(int i=2;i<s.length();i++){
            if(s[i]=='.'){
                len=i-2;
                break;
            }
            num=num*10+s[i]-'0';
        }
        // D(num);D(len);E;
        int temp=num;
        for(int i=1;i<=len;i++){//i为循环节长度 len-i为非循环节长度
            temp/=10;
            int a=num-temp;
            int b=pow2(10,len)-pow2(10,len-i);
            int k=gcd(a,b);
            if(b/k<min_denominator){
                min_denominator=b/k;
                numerator=a/k;
            }
        }
        cout<<numerator<<"/"<<min_denominator<<endl;
    }
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

素数

[POJ 3126: Prime Path](#) 预处理出[1~10000]的质数，然后bfs即可。代码如下

```

/*

*/
#define method_1
#ifdef method_1
/*
预处理出[1~10000]的质数，然后bfs即可。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=10000+5;
const int INF=0x3f3f3f3f;
int T,m=0,v[maxn],p[maxn];
map<string,int>mp,d,prime;
string s,t;
queue<string>q;
string to_string(int x){
    string s="";
    while(x){
        s+=x%10+'0';
        x/=10;
    }
    reverse(s.begin(),s.end());
    return s;
}
void pre(int n){ //预处理出[1~10000]的质数
    for(int i=2;i<=n;i++){
        if(!v[i]){
            p[++m]=i;v[i]=i;
        }
        for(int j=1;j<=m;j++){
            if(p[j]>v[i]||p[j]>n/i) break;

```

```

        v[i*p[j]]=p[j];
    }
}
for(int i=1;i<=m;i++){
    prime[to_string(p[i))]=1;
}
}
void init(){
    mp.clear();d.clear();
    while(q.size()) q.pop();
}
int bfs(){
    q.push(s);mp[s]=1;d[s]=0;
    while(q.size()){
        string now=q.front();q.pop();
        if(now==t) return d[now];
        for(int i=0;i<=3;i++){ //依次尝试改变每一位
            if(i==0){
                for(int j=1;j<=9;j++){
                    if(now[i]==j+'0') continue;
                    string temp=now;
                    temp[i]=j+'0';
                    if(mp[temp]==1||prime[temp]!=1) continue;
                    d[temp]=d[now]+1;mp[temp]=1;
                    q.push(temp);
                }
            }
            else{
                for(int j=0;j<=9;j++){
                    if(now[i]==j+'0') continue;
                    string temp=now;
                    temp[i]=j+'0';
                    if(mp[temp]==1||prime[temp]!=1) continue;
                    d[temp]=d[now]+1;mp[temp]=1;
                    q.push(temp);
                }
            }
        }
    }
    return -1;
}
int main() {
    ios::sync_with_stdio(false);
    //freopen("Prime Path.in","r",stdin);
    pre(maxn-5);
    cin>>T;
    while(T--){
        init();
        cin>>s>>t;
    }
}

```

```

        int ans=bfs();
        ans==-1?cout<<"Impossible"<<endl:cout<<ans<<endl;
    }
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

[POJ 3421: X-factor Chains](#) 法一：找出所有约数，然后用类似LIS的dp方法即可。

法二：对x分解质因数，那么最长链的长度就是所有素因子的积。（即最优解就是将所有素因子连起来的结果）方案数就是素因子的全排列，注意相同的素因子之间没有顺序。

代码如下

```

/*

*/

#define method_1
#ifdef method_1
/*
找出所有约数，然后用类似LIS的dp方法即可。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "

```

```

#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=(1<<11)+5; //n的约束个数最大为2根号n
const int INF=0x3f3f3f3f;
int x,a[maxn],m,f[maxn],ans1;
ll g[maxn],ans2;
void init() {
    ans1=m=0;
    ans2=0ll;
    memset(a,0,sizeof(a));
    memset(f,0,sizeof(f));
    memset(g,0,sizeof(g));
    g[0]=1;
}
void pre(int n) {
    for(int i=1; i*i<=n; i++) {
        if(n%i==0) {
            a[++m]=i;
            if(n/i!=i) a[++m]=n/i;
        }
    }
    sort(a+1,a+m+1);
}
void dp() {
    for(int i=1; i<=m; i++) {
        for(int j=1; j<i; j++) { //j=0时a[j]=0 会除法RE
            if(a[i]%a[j]==0) {
                f[i]=max(f[i],f[j]+1);
            }
        }
    }
    ans1=f[m]; //因为必须要以x作为结尾
    // for(int i=1;i<=m;i++) D(f[i]);
    // E;
    for(int i=1; i<=m; i++) {
        for(int j=0; j<i; j++) {
            if(j==0) {
                if(f[i]==f[j]+1) {
                    g[i]+=g[j];
                }
            } else {
                if(f[i]==f[j]+1&& a[i]%a[j]==0) {
                    g[i]+=g[j];
                }
            }
        }
    }
}

```

```

    }
    // for(int i=1;i<=m;i++) D(g[i]);
    // E;
    ans2=g[m];
}
int main() {
    ios::sync_with_stdio(false);
    //freopen("X-factor Chains.in","r",stdin);
    while(cin>>x) {
        init();
        pre(x);
        dp();
        cout<<ans1<<" "<<ans2<<endl;
    //    for(int i=1;i<=m;i++) D(a[i]);
    }
    return 0;
}
#endif
#ifdef method_2
/*

```

对x分解质因数，那么最长链的长度就是所有素因子的积。（即最优解就是将所有素因子连起来的结果）

方案数就是素因子的全排列，注意相同的素因子之间没有顺序。

```

*/
#include<cstdio>
#include<cstring>
#include<iostream>
#include<cmath>
#include<algorithm>
typedef long long ll;
using namespace std;
int su[1025];
bool he[1025];
ll t[1025];
int cnt=0;
ll sum=0;
void Euler(int n) {
    for(int i=2; i<=n; i++) {
        if(he[i]==0) {
            cnt++;
            su[cnt]=i;
        }
        for(int j=1; j<=cnt&& i*su[j]<=n; j++) {
            he[su[j]*i]=1;
            if(i%su[j]==0)break;
        }
    }
    return;
}

```

```

}
void fen(ll x) {
    for(int i=1; i<=cnt&&su[i]*su[i]<=x; i++) {
        ll p=su[i];
        if(x%p==0) {
            while(x%p==0) {
                sum++;
                t[i]++;
                x/=p;
            }
        }
        if(x>1)sum++;
        return;
    }
}
ll jie(int k) {
    ll ans=1;
    for(int i=2; i<=k; i++) {
        ans*=i;
    }
    return ans;
}
int main() {
    //freopen("X-factor Chains.in","r",stdin);
    Euler(1024);
    ll x;
    while(scanf("%lld",&x)!=EOF) {
        if(x==0)break;
        memset(t,0,sizeof(t));
        sum=0;
        fen(x);
        printf("%lld ",sum);
        ll ans=jie(sum);
        for(int i=1; i<=cnt; i++) {
            ans/=jie(t[i]);
        }
        printf("%lld\n",ans);
    }
    return 0;
}
#endif
#ifdef method_3
/*

*/

#endif

```

[POJ 3292: Semi-prime H-numbers](#) 另类筛法。模仿埃式筛法。若 $i \times j$ 是H-semi-primes意味着 i 、 j 都是H-prime。因此枚举 i ，范围 $[5, \sqrt{maxn}]$ 。若 i 为H-prime，则枚举 $[i, maxn]$ 中的所有 $4n+1$ 数。用 $vis[i \times j] = vis[i] + vis[j] + 1$ 转移即可。筛出来 vis 等于1的都是H-semi-primes，0或者其他不是1的不是。代码如下

```
/*
*/
#define method_1
#ifdef method_1
/*
另类筛法。模仿埃式筛法。
若 $i \times j$ 是H-semi-primes意味着 $i$ 、 $j$ 都是H-prime。
因此枚举 $i$ ，范围 $[5, \sqrt{maxn}]$ 。若 $i$ 为H-prime，则枚举 $[i, maxn]$ 中的所有 $4n+1$ 数。
用 $vis[i \times j] = vis[i] + vis[j] + 1$ 转移即可。
筛出来 $vis$ 等于1的都是H-semi-primes，0或者其他不是1的不是。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=1000001+5;
const int INF=0x3f3f3f3f;
int h,sum[maxn]={0},vis[maxn]={0};
void pre(int n){ //筛出来vis等于1的都是H-semi-primes，0或者其他不是1的不是
    int m=sqrt(n+0.5);
    for(int i=5;i<=m;i+=4){ //1不算H-prime
        if(!vis[i]){
            for(int j=i;i*j<=n;j+=4) vis[i*j]=vis[i]+vis[j]+1; //i*j是否是H-
```

semi-primes 取决于i、j和本身

```
    }
}
// for(int i=1;i<=1000;i++) if(vis[i]==1) cout<<i<<endl;
for(int i=1;i<=n;i++) sum[i]=(vis[i]==1)+sum[i-1]; //预处理前缀和 方便输出
}
int main() {
    ios::sync_with_stdio(false);
    //freopen("Semi-prime H-numbers.in","r",stdin);
    pre(maxn-5);
    while(cin>>h){
        if(h==0) break;
        cout<<h<<" "<<sum[h]<<endl;
    }
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif
```

快速幂

[POJ 3641: Pseudoprime numbers](#) 代码如下

```
/*

*/
#define method_1
#ifdef method_1
/*
伪素数：满足①p不是素数②存在a > 1使得a^p = a (mod
p)的p是伪素数，给出p和a，判断p是否是伪素数。
直接快速幂即可。 注意可能会乘爆，所以要开longlong。
*/
#include<iostream>
#include<cstdio>
```

```

#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=+5;
const int INF=0x3f3f3f3f;
ll p,a;
bool check(ll x){
    int m=sqrt(x+0.5);
    for(int i=2;i<=m;i++){
        if(x%i==0) return false;
    }
    return true;
}
ll ksm(ll a,ll b,ll p){
    ll ans=1%p;
    while(b){
        if(b&1) ans=(ans*a)%p;
        b>>=1;a=(a*a)%p;
    }
    return ans;
}
int main() {
    ios::sync_with_stdio(false);
    //freopen("Pseudoprime numbers.in","r",stdin);
    while(cin>>p>>a){
        if(p==0&&a==0) break;
        if(check(p)){
            cout<<"no"<<endl;
            continue;
        }
        if(ksm(a,p,p)!=a%p) cout<<"no"<<endl;
        else cout<<"yes"<<endl;
    }
    return 0;
}
#endif

```

```

#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

POJ 1995: Raising Modulo Numbers 题解链接 <https://www.jianshu.com/p/dfb78e06c19d>

代码如下

```

/*

*/

#define method_1
#ifdef method_1
/*

*/

#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
using namespace std;
typedef long long ll;
const int maxn=45000+5;
const ll INF=0x3f3f3f3f3f3f3f3fll;
ll t,m,h,a,b;
ll solve(ll a,ll b,ll m){
    ll ans=1%m;
    while(b){
        if(b&1) ans=(ans*a)%m;
        a=(a*a)%m;
        b>>=1;
    }
    return ans;
}

```

```

int main() {
    ios::sync_with_stdio(false);
    //freopen("Raising Modulo Numbers.in", "r", stdin);
    cin>>t;
    while(t--){
        cin>>m>>h;
        ll ans=0;
        while(h--){
            cin>>a>>b;
            ans=(ans+solve(a,b,m))%m;
        }
        cout<<ans<<endl;
    }

    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```