$Min-Max$ 容斥

初听到这个算法的时候，你可能会想，min，max这两个运算甚至都没有逆元，怎么还能容斥？其实这个算法是针对集合的。

$Min-Max$ 容斥

直接上公式吧：

$max(S)=\sum_{T\subseteq S,T\neq\phi}(-1)^{|T|-1}min(T)$

是不是很神奇呀...

其实这里用到了一个很显然的性质（假设集合中的数两两不同，如果出现相同元素就按照下标赋予不同的优先级）：S 中的最大值，只有在 T 恰好只包含它的时候才会被作为最小值算一次。

还是严谨一点吧，证明一下（抄的）：

构造容斥系数 $f(x)$ 满足 $max(S)=\sum_{T\subseteq S,T\neq\phi}f(|T|)min(T)$

考虑排名为 $x+1$ 的元素被计算到的次数：

$\sum_{i=0}^{x}\binom{x}{i}f(x+1)$

也就是说首先将当前枚举的元素钦定进去，然后在比它大的元素中任选。

套定义：

$[x==0]=\sum_{i=0}^{x}\binom{x}{i}f(x+1)$

二项式反演：

$f(x+1)=\sum_{i=0}^{n}(-1)^{x-i}\binom{x}{i}[x==0]$

$f(x+1)=(-1)^{x}$

$f(x)=(-1)^{x-1}$

带入就可以得到一开始的式子了。

$kthMin-Max$ 容斥：

$kthMax(S)=\sum_{T\subseteq S\ ,\ T\neq\phi}(-1)^{|T|-k}\binom{|T|-1}{k-1}min(T)$

证明：

构造容斥系数 $f(x)$ 满足 $kthMax(S)=\sum_{T\subseteq S,\ T\neq\phi} f(|T|)min(T)$

第 $x+1$ 大的元素被计算到的次数：

$\sum_{i=0}^{x}\binom{x}{i}f(i+1)$

套定义：

$[x==k-1]=\sum_{i=0}^{x}\binom{x}{i}f(i+1)$

二项式反演：

$f(x+1)=\sum_{i=0}^{x}(-1)^{x-i}\binom{x}{i}[i==k-1]$

$f(x+1)=(-1)^{x-(k-1)}\binom{x}{k-1}$

$f(x)=(-1)^{x-k}\binom{x-1}{k-1}$

$problems$:

1、[HAOI2015]按位或：Link

刚开始你有一个数字 0，每一秒钟你会随机选择一个 $[0,2^n-1]$ 的数字，与你手上的数字进行或操作。选择数字i的概率是 p_i。保证 $0<=p_i<=1$，$\sum p_i=1$。问期望多少秒后，你手上的数字变成 2^n-1。$n<=20$

最简单的想法是将取到每一位的期望时间取个max，但是这样是不对的，因为期望时间不能直接取 $\max$。举个例子：有两种球，取到的概率相等（取完后放回），那么取到任意一种的期望时间都是2，但是都取到过的期望时间显然不是2（2是最优情况）。为什么会出现这样的情况呢？因为虽然后面要取两个球，但还是一个人在取，就得把期望放到一起去算了。

> min-max容斥真正暴力的地方在于我们就算根本没法进行大小比较，也可以仅通过加减法把 $\max$ 或者 $\min$ 以极其暴力的方式 $O(2^n)$ 的枚举子集容斥出来。---shadowice1984

比如说这道题，就是典型的不能直接进行大小比较。那么最大值不能求，最小值会不会好球一点呢？是的。最小值的含义就是集合中至少取到了一个的期望时间，设为 $E(T)$，通过常识可以知道 $E(T)=\frac{1}{P(T)}$ 而 $P(T)$ 表示的就是任取一个数字，与 T 有交集的概率...好像还是不大好求？那么补集转化一下，就是 $1-\sum_X P(X\cap T=\phi)$，也就是说要求出 T 的补集的子集和，高维前缀和就可以解决啦。虽然说了这么多，代码却非常短。

```
# include <cstdio>
# include <iostream>
# define R register int

using namespace std;

const int maxn=(1<<20)+5;
const double eps=1e-10;
int n,f[maxn]; //容斥系数
double p[maxn],vis[40],g[maxn];

int cal (int x)
```

```
{
    int ans=0;
    for (R i=0;i<n;++i) if(x&(1<<i)) ans++;
    return ans;
}

int main()
{
    scanf("%d",&n);
    for (R i=0;i<(1<<n);++i)
    {
        f[i]=cal(i);
        if(f[i]%2) f[i]=1;
        else f[i]=-1;
    }
    for (R i=0;i<(1<<n);++i)
    {
        scanf("%lf",&p[i]);
        for (R j=0;j<n;++j)
            if(i&(1<<j)) vis[j]+=p[i];
    }
    for (R i=0;i<n;++i)
        for (R j=0;j<(1<<n);++j)
            if(j&(1<<i)) p[j]+=p[j^(1<<i)];
    for (R i=0;i<n;++i)
        if(vis[i]<eps) { puts("INF"); return 0; }
    for (R i=1;i<(1<<n);++i)
        g[i]=1/(1-p[((1<<n)-1)^i]);
    double ans=0;
    for (R i=0;i<(1<<n);++i)
        ans+=f[i]*g[i];
    printf("%.10lf",ans);
    return 0;
}
```

2、斐波那契的最小公倍数

题意概述：给出 n 个数，求以它们为下标的斐波那契数的最小公倍数。$n <= 5000, a_i <= 1000000$

斐波那契数有一个性质，$gcd(f_i, f_j) = f_{gcd(i,j)}$ ，证明略，因为以前写过。

虽然 GCD 有这样的性质，但是 LCM 显然没有对吧...所以可以考虑将 LCM 转化成 GCD 来做。

GCD 相当于是对质因子的指数取 min ，而 LCM 则是取 max ，可以想到利用 $min-max$ 容斥来做。

对于每一个质因子单独考虑：(对于一个等式，两边同时扔到指数上显然还是正确的)

$d^{max(S)} = d^{\sum_{T\subseteq S,T\neq\phi}(-1)^{|T|-1}min(T)}$

稍微变一点：

$d^{max(S)} = \prod_{T\subseteq S,T\neq\phi} d^{min(T)(-1)^{|T|-1}}$

$LCM(S) = \prod_{T\subseteq S,T\neq\phi} GCD(T)^{(-1)^{|T|-1}}$

一直带着这个 $T \neq \phi$ 感觉挺烦人的，所以把它单独提出来吧（将空集的 GCD 设为 1 好像很科学，因为它没有任何质因子）：

$LCM(S)=\prod_{T\subseteq S} GCD(T)^{(-1)^{|T|-1}}$

现在用一下斐波那契公约数的那个性质：

$LCM(S)=\prod_{T\subseteq S} f(GCD(a_i))^{(-1)^{|T|-1}}$

嗯嗯，我们得到了一个 $2^{10^5}10^5log(10^5)$ 的优秀做法！~~可能比高精度算斐波那契再求公倍数都慢~~

虽然好像有点不同，但反演套路还是可以用的：

$LCM(S)=\prod_{d=1}^{1e6} f(d)^{\sum_{T\subseteq S}(-1)^{|T|-1}[gcd(T)==d]}$

发现这个 $1e6$ 即使枚举也是完全OK的，所以就不管他了，把指数拿下来作为一个新函数：

$h(x)=(-1)^{|T|-1}[gcd(T)==d]$

好像非常难做...这时候我突然想到了学反演之初看到过的公式：

$g(n)=\sum_{d|n} f(d) \Rightarrow f(n)=\sum_{n|d} \mu(\frac{d}{n})g(d)$

虽然做题做多了以后就很少用这个公式了，但是它还是很有用的：

$g(n)=\sum_{d|n} h(d)$

这个 $g(n)$ 看起来还挺好算的：

首先一个集合的公约数是 d 的倍数，那么集合中的每个数就应当都是 d 的倍数。假设在原集合中有 k 个数是 d 的倍数，那么就有：

$\sum_{i=1}^{k} \binom{k}{i}(-1)^{i-1}$

诶，这不就是二项式定理变了一小点？把下标改一改：

$\sum_{i=0}^{k} \binom{k}{i}(-1)^{i-1}1^{k-i+1}+1$

$=0+1=1$

所以说 g 恒等于一？差不多，不过还是要注意一下，如果集合中甚至没有数是 d 的倍数，那么答案就是0了。如何知道集合中是否存在 d 的倍数？开个桶，把 a 都塞进去，对于每个 d 枚举倍数，复杂度 $NlogN$。

知道了 g 如何求 f ？依旧采用同样的套路，对于每个 $g(d)$ 计算对 f 的贡献即可，复杂度 $NlogN$。

```
# include <cstdio>
# include <iostream>
# include <algorithm>
# define R register int
# define ll long long

using namespace std;

const int mod=1000000007;
const int maxn=200004;
const int maxv=1000006;
int a[maxn],f[maxv],n,b[maxv],v;
int pri[maxv],h,vis[maxv],mu[maxv],t[maxv];
```

```
//a 是给出的数字们，t就是一个桶，b[i]表示i的倍数有没有出现过

void init (int n)
{
    mu[1]=1;
    for (R i=2;i<=n;++i)
    {
        if(!vis[i]) pri[++h]=i,mu[i]=-1;
        for (R j=1;j<=h&&i*pri[j]<=n;++j)
        {
            vis[ i*pri[j] ]=1;
            if(i%pri[j]==0) break;
            mu[ i*pri[j] ]=-mu[i];
        }
    }
}

ll qui (ll a,ll b)
{
    ll s=1;
    while(b)
    {
        if(b&1) s=s*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return s;
}

int ask (int x)
{
    int ans=0;
    for (R i=x;i<=v;i+=x)
        ans+=b[i]*mu[i/x];
    return ans;
}

int main()
{
    scanf("%d",&n);
    for (R i=1;i<=n;++i) scanf("%d",&a[i]),t[ a[i] ]=1;
    sort(a+1,a+1+n);
    v=a[n];
    for (R i=1;i<=v;++i)
        for (R j=i;j<=v;j+=i)
            if(t[j]) b[i]=1;
    init(v);
    f[1]=1;
    for (R i=2;i<=v;++i) f[i]=(f[i-1]+f[i-2])%mod;
    ll ans=1,t;
    for (R i=1;i<=v;++i)
    {
        t=ask(i);
        if(t==0) continue;
        if(t>0)
            ans=ans*qui(f[i],t)%mod;
        else
```

```
            ans=ans*qui(qui(f[i],mod-2),-t)%mod;
    }
    printf("%lld",ans);
    return 0;
}
```