1322B - Present

$4e5$ 个数，问两两求和后的异或和。

可以按位考虑，考虑第 k 位时给数字模 2^{k+1} []这样第 k 位为 1 和只可能在 $[2^k,2^{k+1}-1],[2^{k+1}+2^k,2^{k+2}-2]$ 区间，对于每个数lower_bound找位置就好。

```
#include<bits/stdc++.h>
#define ll long long
#define pii pair<int,int>
#define mp make_pair
#define fi first
#define se second
#define pb push_back
using namespace std;
const int N=4e5+10;
int a[N],num[N];
int main()
{
    int n;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)scanf("%d",&a[i]);
    int ans=0;
    for(int k=1;k<=2e7;k<<=1)
    {
        ll tot=0,res=0;
        for(int i=1;i<=n;i++)num[++tot]=a[i]%(k<<1);
        sort(num+1,num+1+tot);
        for(int i=1;i<=tot;i++)
        {
            int l=k-num[i],r=(k<<1)-num[i];
            if(l<=r)res+=lower_bound(num+1,num+i,r)-
lower_bound(num+1,num+i,l);
            l=(k<<1)+k-num[i],r=(k<<2)-1-num[i];
            if(l<=r)res+=lower_bound(num+1,num+i,r)-
lower_bound(num+1,num+i,l);
        }
        if(res&1)ans|=k;
    }
    printf("%d\n",ans);
    return 0;
}
```

1322C - Instant Noodles

二分图，右边的点每个有一个点权，对于左边点的集合 S □ $N(S)$ 是所有与集合中点邻接的点□ $f(S)$ 是 $N(S)$ 的点权之和。问对与所有非空集合 S □ $f(S)$ 的 $\gcd$ □

如果某些右边的点邻接的点完全相同，则缩点，将权值定为它们的和，去除度数零的点，此时所有点权的 $\gcd$ 即答案。

证明：设此时的 $\gcd$ 为 g □全集 U □给所有 c_i 除以 g □若答案为 $k\times g(k > 1)$ □则 $k\mid f(U_L)$ □而必定存在 $k\nmid c_j$ □因为 $\gcd$ 已除），取度数最小的这样的 j □取集合 S' 为左边所有不与 j 相邻的点，则 $N(S')$ 仅不包含 j 与邻接点是 j 的子集的点，所有右边点的点权和被 k 整除，而这些点的点权和不被 k 整除，可以推出 $k\nmid f(S')$ □矛盾，故不存在这样的 k □

写法参考了别人的题解，第一次用iota还有vector比较大小什么的，很神秘。

```
#include<bits/stdc++.h>
#define ll long long
#define pii pair<int,int>
#define mp make_pair
#define fi first
#define se second
#define pb push_back
using namespace std;
const int N=5e5+10;
ll c[N];
vector<int>g[N],num;
int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        int n,m;
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;i++)vector<int>().swap(g[i]),scanf("%lld",&c[i]);
        for(int i=1;i<=m;i++)
        {
            int u,v;
            scanf("%d%d",&u,&v);
            g[v].pb(u);
        }
        for(int i=1;i<=n;i++)sort(g[i].begin(),g[i].end());
        vector<int>(n).swap(num);
        iota(num.begin(),num.end(),1);
        sort(num.begin(),num.end(),[&](int a,int b){return g[a]<g[b];});
        ll res=0;
        for(int i=0;i<n;i++)
        {
```

```
            if(g[num[i]].empty())continue;
            ll f=c[num[i]];
            while(i+1<n&&g[num[i+1]]==g[num[i]])i++,f+=c[num[i]];
            if(!res)res=f;else res=__gcd(res,f);
        }
        printf("%lld\n",res);
    }
    return 0;
}
```

1322D - Reality Show

n 个观众每个观众有攻击力 l_i ，邀请他花费 s_i ，并直接获得 c_{l_i} 的收益。当现有的两个观众攻击力相同，他们中一个会消失另一个攻击力变 l_i+1 并获得新的收益 c_{l_i+1} ，这个过程会不断进行直到选择的观众互不相同）。所选观众的攻击力要求是不上升的，最大化利润。

注意到观众打架消除的过程像二进制加法的进位，处理观众的顺序其实是不影响最终的获利的。由于进位是向高位进的，翻转序列搞一个不下降序列比较容易。我们可以用 $f[l_i][k]$ 表示当前选择的最大攻击力为 l_i ，且 l_i 有 k 个时的最大获利，则 $f[l_i][k]$ 可以向 $f[l_i+1][k/2]$ 转移（因为每次都除二复杂度是可接受的）。

```
#include<bits/stdc++.h>
#define ll long long
#define pii pair<int,int>
#define mp make_pair
#define fi first
#define se second
#define pb push_back
using namespace std;
const int N=5005;
const ll inf=(1LL<<60);
int l[N],s[N],c[N];
ll f[N][N],res;
int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)scanf("%d",&l[i]);
    for(int i=1;i<=n;i++)scanf("%d",&s[i]);
    for(int i=1;i<=n+m;i++)scanf("%d",&c[i]);
    for(int i=1;i<=n+m;i++)for(int j=1;j<=n;j++)f[i][j]=-inf;
    for(int i=n;i;i--)
    {
        for(int k=n;k;k--)
        {
            f[l[i]][k]=max(f[l[i]][k],f[l[i]][k-1]+c[l[i]]-s[i]);
```

```
            for(int j=l[i],cnt=k;j<n+m&&cnt;j++,cnt>>=1)
            f[j+1][cnt>>1]=max(f[j+1][cnt>>1],f[j][cnt]+(cnt>>1)*c[j+1]);
        }
        for(int j=l[i];j<n+m;j++)f[j+1][0]=max(f[j+1][0],f[j][0]);
    }
    for(int i=1;i<=n+m;i++)res=max(res,f[i][1]);
    printf("%lld\n",res);
    return 0;
}
```

1325E - Ehab's REAL Number Theory Problem

n 个因子不超过 7 个的数 a_i 问最少选多少个乘积会是完全平方数。

首先如果某个数字能被完全平方数整除就除掉它，不影响结果。之后由于因子不超过 7 个，所有 a_i 的质因子数不会多于 2 个。如果有 1 就直接选它结束，否则其余数字将会是 p 或 $p·q$ 的形式，前者给 $1,p$ 连边，后者给 p,q 连边，问题就转化成无向图找最小环。找环我们直接bfs由于每条边意味着两个顶点的数字相乘，最小环中必包含一个不大于 $\sqrt{\max_{a_i}}$ 的点，仅枚举这些点作为起点即可。

```
#include<bits/stdc++.h>
#define ll long long
#define pii pair<int,int>
#define mp make_pair
#define fi first
#define se second
#define pb push_back
using namespace std;
const int N=1e6+10;
int pri[N],tot1,a[N],head[N],cnt=-1,num[N],tot2,res=N,d[N],vis[N];
bool jud[N];
vector<int>g[N];
struct Node{int nxt,to;}edges[N];
void addedge(int u,int v){edges[++cnt]=Node{head[u],v},head[u]=cnt;}
void init()
{
    jud[1]=1;
    for(int i=2;i<=1000;i++)
    {
        if(!jud[i])pri[++tot1]=i;
        for(int j=1;j<=tot1&&pri[j]*i<=1000;j++)
        {
            jud[pri[j]*i]=1;
            if(i%pri[j]==0)break;
```

```
        }
    }
}
queue<int>q;
void bfs(int s)
{
    memset(d,0x3f,sizeof(d));
    memset(vis,0,sizeof(vis));
    queue<int>().swap(q);
    q.push(s),d[s]=0;
    while(!q.empty())
    {
        int u=q.front();q.pop();
        for(int i=head[u];~i;i=edges[i].nxt)
        {
            int v=edges[i].to;
            if(vis[i])continue;
            if(d[v]==0x3f3f3f3f)d[v]=d[u]+1,vis[i]=vis[i^1]=1,q.push(v);
            else {res=min(res,d[u]+d[v]+1);}
        }
    }
}
int main()
{
    int n;
    memset(head,-1,sizeof(head));
    scanf("%d",&n),init();
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
        for(int j=1;j<=tot1;j++)
        {
            while(a[i]%(pri[j]*pri[j])==0)a[i]/=pri[j]*pri[j];
if(a[i]%pri[j]==0)g[i].pb(pri[j]),a[i]/=pri[j],num[++tot2]=pri[j];
        }
        if(a[i]!=1)g[i].pb(a[i]),num[++tot2]=a[i];
        if(g[i].empty()){puts("1");return 0;}
        if(g[i].size()<2)g[i].pb(1),num[++tot2]=1;
    }
    sort(num+1,num+1+tot2);
    tot2=unique(num+1,num+1+tot2)-num-1;
    for(int i=1;i<=n;i++)
    {
        g[i][0]=lower_bound(num+1,num+1+tot2,g[i][0])-num;
        g[i][1]=lower_bound(num+1,num+1+tot2,g[i][1])-num;
        addedge(g[i][0],g[i][1]),addedge(g[i][1],g[i][0]);
    }
    for(int i=1;i<=tot2;i++)if(num[i]<=1000)bfs(i);
    printf("%d\n",res==N?-1:res);
    return 0;
```

```
}
```

1325F - Ehab's Last Theorem

给无重边自环的 n 个节点的无向图，要求构造至少 $\lceil\sqrt{n}\rceil$ 个节点的简单环，或 $\lceil\sqrt{n}\rceil$ 个节点的独立集。

dfs树上每个非树边 (u,v) 都代表了一个大小为 $|\mathrm{dep}_u-\mathrm{dep}_v|+1$ 的环（dfs树的一个性质是所有非树边连接一个点与它的某个祖先）。搞一个栈记录一下节点，如果环的大小满足 $\lceil\sqrt{n}\rceil$ 选择方案2直接输出。

而如果不存在 $|\mathrm{dep}_u-\mathrm{dep}_v|+1\ge\lceil\sqrt{n}\rceil$ 则鸽巢原理说明每个点最多连出 $\lceil\sqrt{n}\rceil-2$ 条非树边，最多限制 $\lceil\sqrt{n}\rceil-1$ 个点不可取，所以必能得到 $\lceil\sqrt{n}\rceil$ 大小独立集。从dfs树的叶子节点往上取即可。

```
#include<bits/stdc++.h>
#define ll long long
#define pii pair<int,int>
#define mp make_pair
#define fi first
#define se second
#define pb push_back
using namespace std;
const int N=2e5+10;
queue<int>q;
int head[N],cnt,st[N],top,vis[N],d[N],f[N],bloc,done=0,banned[N];
struct Node{int nxt,to;}edges[N*2];
void addedge(int u,int v){edges[++cnt]=Node{head[u],v},head[u]=cnt;}
void dfs1(int u)
{
    vis[u]=1,st[++top]=u;
    int isleaf=1;
    for(int i=head[u];~i&&!done;i=edges[i].nxt)
    {
        int v=edges[i].to;
        if(f[u]==v)continue;
        if(!vis[v])isleaf=0,f[v]=u,d[v]=d[u]+1,dfs1(v);
        else if(d[u]-d[v]+1>=bloc)
        {
            printf("2\n%d\n",d[u]-d[v]+1);
            for(int j=d[v];j<=d[u];j++)printf("%d ",st[j]);
            puts(""),done=1;
        }
    }
    if(isleaf)q.push(u);
```

```
    top--;
}
int main()
{
    int n,m;
    memset(head,-1,sizeof(head));
    scanf("%d%d",&n,&m),bloc=ceil(sqrt(n))+0.1;
    for(int i=1;i<=m;i++)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        addedge(u,v),addedge(v,u);
    }
    d[1]=1,dfs1(1);
    if(!done)
    {
        printf("1\n");
        while(!q.empty()&&bloc)
        {
            int u=q.front();q.pop();
            if(!banned[u])
            {
                printf("%d ",u),bloc--,banned[u]=1;
                for(int i=head[u];~i;i=edges[i].nxt)
                {
                    int v=edges[i].to;
                    banned[v]=1;
                }
            }
            if(f[u])q.push(f[u]);
        }
        assert(bloc==0);
        puts("");
    }
    return 0;
}
```

1326E - Bombs

给出长度为 n 的两个排列 p,q 按照顺序从 1 到 n 把 p_i 加入集合，如果位置 i 有炸弹则从集合中取出一个最大值，结果是最后集合中的最大值。第 i 个答案回答的是 $q_1,q_2,\ldots q_{i-1}$ 处有炸弹时的结果。

一个比较厉害的转换思维。我们观察到答案是单调不上升的，如果答案至多为 x 我们就需要让 $x+1,x+2,\ldots,n$ 都被炸掉，条件就是对于每个位置右边大于 x 的 p_i 的数量都不多于右边的炸弹数量。可以线段树维护 $右面不小于当前答案的p_i的数量-右面炸弹数量$，如果小于等于 0 则减小当前答案。

```cpp
#include<bits/stdc++.h>
#define ll long long
#define pii pair<int,int>
#define mp make_pair
#define fi first
#define se second
#define pb push_back
using namespace std;
const int N=3e5+10;
int p[N],q[N],maxn[N<<2],lz[N<<2],pos[N];
void pushdown(int idx)
{
    int t=lz[idx];
    lz[idx]=0,lz[idx<<1]+=t,lz[idx<<1|1]+=t;
    maxn[idx<<1]+=t,maxn[idx<<1|1]+=t;
}
void add(int idx,int l,int r,int a,int b,int x)
{
    if(a<=l&&b>=r){lz[idx]+=x,maxn[idx]+=x;return;}
    int mid=(l+r)>>1;
    pushdown(idx);
    if(b<=mid)add(idx<<1,l,mid,a,b,x);
    else if(a>mid)add(idx<<1|1,mid+1,r,a,b,x);
    else add(idx<<1,l,mid,a,b,x),add(idx<<1|1,mid+1,r,a,b,x);
    maxn[idx]=max(maxn[idx<<1],maxn[idx<<1|1]);
}
int main()
{
    int n;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)scanf("%d",&p[i]),pos[p[i]]=i;
    for(int i=1;i<=n;i++)scanf("%d",&q[i]);
    int res=n;
    add(1,1,n,1,pos[n],1);
    for(int i=1;i<=n;i++)
    {
        printf("%d ",res);
        if(i==n)break;
        add(1,1,n,1,q[i],-1);
        while(maxn[1]<=0)--res,add(1,1,n,1,pos[res],1);
    }
    puts("");
    return 0;
}
```

1373F - Network Coverage

$a_i $代表第 $i $处的需求量， $b_i $代表第 $i $处的资源， $i $处资源只可以供给 $i $和 $i+1 $， $n $则可以供给 $n $和 $1 $），问是否可以满足全部需求。

神秘的单调性与神秘的二分。我们发现只要确定某一处提供给自己的资源量就可以确定整个状态，这里我们选择 $1 $， $1 $提供给自己的量越少（即提供给 $2 $越多），后面 $2 $到 $n $的需求越可能得到满足。我们二分这个能使 $2 $到 $n $的需求得以满足的 $1 $能提供给自己最多的量，最后判断这种情况下首尾相接后 $1 $能否得到满足即可。

```
#include<bits/stdc++.h>
#define ll long long
#define pii pair<int,int>
#define mp make_pair
#define fi first
#define se second
#define pb push_back
using namespace std;
const int N=1e6+10;
ll read()
{
    ll x=0,f=1;char c=getchar();
    while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}
    while(c>='0'&&c<='9'){x=x*10+c-'0';c=getchar();}
    return x*f;
}
int n,a[N],b[N],c[N];       //a 需要 b 可分配
bool judge(int k)
{
    for(int i=0;i<n;i++)c[i]=0;
    c[0]=k,c[1]=b[0]-k;         //c 已经得到
    for(int i=1;i<n;i++)
    {
        int t=max(0,a[i]-c[i]);
        if(b[i]<t)return false;
        c[(i+1)%n]+=b[i]-t;
    }
    return true;
}
int main()
{
    int t=read();
    while(t--)
    {
        n=read();
        for(int i=0;i<n;i++)a[i]=read();
        for(int i=0;i<n;i++)b[i]=read();
        int l=0,r=b[0],res=-1;
        while(l<=r)
```

```
        {
            int mid=(l+r)>>1;
            if(judge(mid))l=mid+1,res=mid;
            else r=mid-1;
        }
        if(res!=-1&&judge(res)&&c[0]>=a[0])puts("YES");
        else puts("NO");
    }
    return 0;
}
```

1373E - Sum of Digits

$f(x) $是 $x $的数位和，给定 $n,k $构造 $x $使得 $\sum\limits_{i=0}^kf(x+i)=n $

数据范围很小甚至可以打表。观察到 $k\le9 $所以至多进位 $1 $次。事实上手玩一下发现是这样 $n $在进位情况下是 $x99\ldots998y $的形式，不进位情况下是 $x99\ldots999y $的形式，枚举 $x,y $和长度 $l $就好。

```
#include<bits/stdc++.h>
#define ll long long
#define pii pair<int,int>
#define mp make_pair
#define fi first
#define se second
#define pb push_back
using namespace std;
char s1[55],s2[55];
int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        int n,k,f=0;
        scanf("%d%d",&n,&k);
        for(int i=0;i<9;i++)
        for(int l=0;l<20;l++)
        for(int j=0;j<10;j++)
        {
            int tmp=(i+l*9+j)*(k+1)+(1+k)*k/2;
            if(j+k>=10)tmp-=9*(j+k-9);
            if(j+k>=10&&l)tmp-=k+1;
            if(tmp==n)
```

```
            {
                int cnt=0;
                if(i)s2[cnt++]=i+'0';
                for(int ii=1;ii<=l;ii++)
                {
                    if(ii==l&&j+k>=10)s2[cnt++]='8';
                    else s2[cnt++]='9';
                }
                s2[cnt++]=j+'0',s2[cnt]='\0';
if(!f||strlen(s1)>strlen(s2)||(strlen(s1)==strlen(s2)&&strcmp(s1,s2)>0))
                strcpy(s1,s2);
                f=1;
            }
        }
        if(!f)printf("%d\n",-1);
        else printf("%s\n",s1);
    }
    return 0;
}
```

1383D - Rearrange

给 $n\times m $的矩阵，其中元素是 $1 $到 $n\times m $的排列。现要求构造同样用 $1 $到 $n\times m $的排列填充的 $n\times m $矩阵，使得：

- 每一行、每一列都是单峰的。
- 行最大值的集合与给定矩阵相同，列最大值的集合与给定矩阵相同。

一个想法是数字由大到小填，这样当填下某个数字时就可以确定某行或某列的最大值。官方提供了这样一种构造方法：

初始待填充矩阵 $ 0 $行 $ 0 $列。

$\mathtt{for}\;num\;\mathtt{in}\;n\times m\;..\;1 $

如果有行最大值为 $num $增加行；有列最大值为 $num $增加列 $num $为某个最大值就直接填在右下角，如果是行最大值，将左边的一整行由近到远加入队列；是列最大值，将上边的一整列由近到远加入队列 $num $不是最大值则从队列中取一个位置出来放在该位置。

随便想一下就发现这样能够保证单峰性（峰值即那些行列最大值）。

```
#include<bits/stdc++.h>
#define ll long long
#define pii pair<int,int>
#define mp make_pair
#define fi first
```

```
#define se second
#define pb push_back
using namespace std;
const int N=255;
int a[N][N],b[N][N],r[N],c[N];
queue<pii>q;
int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=0;i<n;i++)for(int j=0;j<m;j++)
    scanf("%d",&a[i][j]),r[i]=max(r[i],a[i][j]),c[j]=max(c[j],a[i][j]);
    sort(r,r+n),sort(c,c+m);
    int x=0,y=0,i=n-1,j=m-1;
    for(int num=n*m;num;num--)
    {
        if(r[i]==num)x++;
        if(c[j]==num)y++;
        if(r[i]==num||c[j]==num)b[x][y]=num;
        else b[q.front().fi][q.front().se]=num,q.pop();
        if(r[i]==num)
        {
            for(int k=y-1;k;k--)q.push(mp(x,k));
            i--;
        }
        if(c[j]==num)
        {
            for(int k=x-1;k;k--)q.push(mp(k,y));
            j--;
        }
    }
    for(int i=1;i<=n;i++,puts(""))for(int j=1;j<=m;j++)
    printf("%d ",b[i][j]);
    return 0;
}
```

1388E - Uncle Bogdan and Projections

给 x 轴以上的若干水平线段。现可以指定一个向量让所有线段沿该方向投影到 x 轴上，投影不可以重叠，宽度定义为投影最右端横坐标减去最左端横坐标，问可能的最小宽度。

如果纵坐标全部相同，直接垂直投影即可。否则我们可以在使得某个投影与投影相切的时候取到最小宽度。对任意两个纵坐标不同的线段，我们可以算出两个投影相切的角度，从而得到一段不可行的区间。扫描线得出全局的可行投影角度区间。

而要在合理时间内得到取若干角度时投影的最大最小横坐标，可以用一个叫Convex Hull Trick的做法。设

θ 为投影线与 x 轴正方向的夹角 (x,y) 投影在横坐标 $x-\frac y{tan(\theta)}$ 的位置。以 $-\frac 1{tan(\theta)}$ 为自变量，则 y_i 为斜率 x_i 为截距。若干直线只会在一个凸包上取得最大值，我们先求出这个凸包，之后对于每个 $-\frac 1{tan(\theta_i)}$ 可以二分。最小值同理。

```
#include<bits/stdc++.h>
#define fi first
#define se second
#define mp make_pair
#define pb push_back
#define pii pair<int,int>
using namespace std;
const int N=2005;
const double pi=acos(-1.0);
const double eps=1e-10;
const double inf=1e20;
int l[N],r[N],y[N],tot1,tot2,top1,top2;
int dcmp(double x){return x<-eps?-1:x>eps;}
vector<double>v1,v2;
double num[N*N*2],res=inf,x1[N*2],x2[N*2];
struct line{int k,b;}s[N*2],st1[N*2],st2[N*2];
double getInt(line l1,line l2){return (l2.b-l1.b)*1.0/(l1.k-l2.k);}
void getMax(double x)
{
    int ll=1,rr=top1;double maxn;
    while(ll<=rr)
    {
        int mid=(ll+rr)>>1;
        double pl=x1[mid-1],pr=x1[mid];
        if(x>=pl&&x<=pr){maxn=st1[mid].k*x+st1[mid].b;break;}
        else if(x>pr)ll=mid+1;
        else rr=mid-1;
    }
    ll=1,rr=top2;double minn;
    while(ll<=rr)
    {
        int mid=(ll+rr)>>1;
        double pl=x2[mid-1],pr=x2[mid];
        if(x>=pl&&x<=pr){minn=st2[mid].k*x+st2[mid].b;break;}
        else if(x>pr)ll=mid+1;
        else rr=mid-1;
    }
    res=min(res,maxn-minn);
}
int read()
{
    int x=0,f=1;char c=getchar();
    while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}
    while(c>='0'&&c<='9'){x=x*10+c-'0';c=getchar();}
    return x*f;
}
```

```
int main()
{
    int n=read();
    double maxt=-inf,mint=inf;
    bool f=true;
    for(int i=1;i<=n;i++)
    {
        l[i]=read(),r[i]=read(),y[i]=read();
        maxt=max(maxt,1.0*r[i]),mint=min(mint,1.0*l[i]);
        s[++tot2]=line{y[i],l[i]};
        s[++tot2]=line{y[i],r[i]};
        for(int j=1;j<i;j++)
        if(y[i]!=y[j])
        {
            int a=i,b=j;f=false;
            if(y[a]<y[b])swap(a,b);
            int u=y[a]-y[b],p=r[a]-l[b],q=l[a]-r[b];
            int g1=abs(__gcd(u,p)),g2=abs(__gcd(u,q));
            double ang1=atan2(u/g1,p/g1);
            double ang2=atan2(u/g2,q/g2);
            v1.pb(ang1),v2.pb(ang2);
            num[++tot1]=ang1,num[++tot1]=ang2;
        }
    }
    if(f){printf("%.10lf\n",maxt-mint);return 0;}
sort(num+1,num+1+tot1),sort(v1.begin(),v1.end()),sort(v2.begin(),v2.end());
    sort(s+1,s+1+tot2,[&](line l1,line l2){return
l1.k==l2.k?l1.b>l2.b:l1.k<l2.k;});
    for(int i=1;i<=tot2;i++)
    {
        if(top1&&st1[top1].k==s[i].k)continue;
        while(top1>=2&&dcmp(getInt(s[i],st1[top1])-x1[top1-1])<=0)top1--;
        st1[++top1]=s[i];
        if(top1>=2)x1[top1-1]=getInt(st1[top1],st1[top1-1]);else
x1[top1-1]=-inf;
    }
    x1[top1]=inf;
    sort(s+1,s+1+tot2,[&](line l1,line l2){return
l1.k==l2.k?l1.b<l2.b:l1.k>l2.k;});
    for(int i=1;i<=tot2;i++)
    {
        if(top2&&st2[top2].k==s[i].k)continue;
        while(top2>=2&&dcmp(getInt(s[i],st2[top2])-x2[top2-1])<=0)top2--;
        st2[++top2]=s[i];
        if(top2>=2)x2[top2-1]=getInt(st2[top2],st2[top2-1]);else
x2[top2-1]=-inf;
    }
    x2[top2]=inf;
    for(int i=1,j=0,k=0,cnt=0;i<=tot1;i++)
    {
```

```
        while(k<v2.size()&&dcmp(v2[k]-num[i])<=0)k++,cnt--;
        if(!cnt)getMax(-1.0/tan(num[i]));
        while(j<v1.size()&&dcmp(v1[j]-num[i])<=0)j++,cnt++;
    }
    printf("%.10lf\n",res);
    return 0;
}
```

1332E - Height All the Same

$n\times m$ 的网格，位置 (i,j) 上有 $a_{i,j}$ 个方块，每次操作可以在某位置加两个方块，或者在两个相邻位置各加一个方块，目标是使得所有位置高度一致。问在 $a_{i,j}\in[l,r]$ 的条件下，能达成目标的方案有多少。

一个观察是重要的只是所有位置上方块数量的奇偶性，同一个位置加两个方块这一操作不改变奇偶性，而使得所有位置奇偶性一致后必能用这个操作达成目标。

另一个观察是，我们必能改变且仅改变任意一对格子的奇偶性（这是因为两个位置间必有一条路径，沿路径每次翻转相邻两个就好）。

当有偶数个奇数或偶数个偶数个时，可以把它们两两配对改变奇偶性达成目标，否则不可以。

- 当 $n\times m$ 为奇数时，必然是偶数个奇数奇数个偶数/偶数个偶数奇数个奇数，答案是总数 $\mathrm{total}=(r-l+1)^{nm}$
- 当 $n\times m$ 为偶数时，奇偶都是偶数个可以，奇偶都是奇数个不行pikmike提供了一个直觉法：

- 当 $r-l+1$ 为偶数时，答案是 $\frac{\mathrm{total}}2$
- 当 $r-l+1$ 为奇数时，可行将会比不可行多 1。这是因为 $[l,r]$ 中必定有 $k\in[l,r]$且 $k\oplus1\notin[l,r]$ 每一种方案，我们可以选第一个不为 k 的位置给 $a_{i,j}$ 异或 1，这样可以使可行方案与不可行方案两两配对，除了全部为 k 的一个方案可行。答案 $\frac{\mathrm{total}+1}{2}$

```
#include<bits/stdc++.h>
#define ll long long
#define fi first
#define se second
#define mp make_pair
#define pb push_back
#define pii pair<int,int>
using namespace std;
const ll mod=998244353;
ll n,m,l,r;
ll qpow(ll a,ll b)
{
    ll res=1;
    while(b)
    {
        if(b&1)res=res*a%mod;
```

```
        a=a*a%mod,b>>=1;
    }
    return res;
}
int main()
{
    scanf("%lld%lld%lld%lld",&n,&m,&l,&r);
    ll tot=qpow(r-l+1,n*m);
    if((n*m)&1)printf("%lld\n",tot);
    else if((r-l+1)&1)printf("%lld\n",(tot+1)%mod*qpow(2,mod-2)%mod);
    else printf("%lld\n",tot*qpow(2,mod-2)%mod);
    return 0;
}
```