A. Passing Ball

- 通过列式子，我们应该能得到一开始的问题对应的数列的递推式：
  $f(i) = (n-2) * f(i-1) + (n-1) * f(i-2)$
- 然后就可以通过特征方程求解数列的通项公式，得到
  $f(t) = ((n-1)^t + (n-1) * (-1)^t)/n = x(mod\ 998244353)$
- 这里我们发现，如果分t的奇偶性分类讨论进行移项，就会变成$(n-1)^{2a} = b$的形式，bsgs即可。

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
#define N 10000010
#define LL long long
#define MOD 998244353
LL T,n,x,base,B=sqrt(MOD);
unordered_map<LL,LL>mp;
LL qow(LL x,LL y){return y?(y&1?x*qow(x,y-1)%MOD:qow(x*x%MOD,y/2)):1;}
int main()
{
    scanf("%lld",&T);
    while(T--){
        scanf("%lld%lld",&n,&x);
        x=x*n%MOD*qow(n-1,MOD-2)%MOD;
        mp.clear();
        mp[1]=0;
        base=(n-1)*(n-1)%MOD;
        LL f1=(x+MOD-1)*(n-1)%MOD;
        LL f2=(x+1)%MOD;
        if(f1==1){
            printf("0\n");
            continue;
        }
        if(f2==1){
            printf("1\n");
            continue;
        }
        LL p=1,flg=0;
        for(LL i=1;i<=B;i++){
            p=p*base%MOD;
            mp[p]=i;
            if(f1==p){
                printf("%lld\n",2ll*i);
                flg=1;
                break;
            }
            if(f2==p){
                printf("%lld\n",2ll*i+1);
                flg=1;
                break;
            }
        }
```

```
        if(flg)continue;
        p=qow(p,MOD-2);
        for(LL k=1,kp=p;k*B<MOD;k++,kp=kp*p%MOD){
            if(mp.count(f1*kp%MOD)){
                printf("%lld\n",2ll*(k*B+mp[f1*kp%MOD]));
                flg=1;
                break;
            }
            if(mp.count(f2*kp%MOD)){
                printf("%lld\n",2ll*(k*B+mp[f2*kp%MOD])+1);
                flg=1;
                break;
            }
        }
        if(flg)continue;
        printf("-1\n");
    }
}
```

B. 2D Plane Game

两条直线存在公共点当且仅当它们重合或者它们斜率不同，因此Bob的最优策略一定是避开斜率出现次数最多的那些直线。Alice为了让Bob与尽量多的直线相交，最优策略就是最小化斜率出现次数的最大值，所以不断从每种斜率的直线中各选一种即可。

时间复杂度$O(nlogn)$。

参考代码：

```
#include<cstdio>
#include<algorithm>
using namespace std;
typedef pair<int,int>P;
const int N=100005;
int Case,n,i,j,k,f[N];
P a[N];
inline int abs(int x){return x>0?x:-x;}
int gcd(int a,int b){return b?gcd(b,a%b):a;}
int main(){
  scanf("%d",&Case);
  while(Case--){
    scanf("%d",&n);
    for(i=1;i<=n;i++){
      int x1,y1,x2,y2;
      scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
      int dx=x2-x1,dy=y2-y1;
      if(dx==0)dy=1;
      else if(dy==0)dx=1;
      else{
        if(dx<0)dx=-dx,dy=-dy;
        int d=gcd(abs(dx),abs(dy));
        dx/=d,dy/=d;
      }
      a[i]=P(dx,dy);
    }
    sort(a+1,a+n+1);
```

```
    for(i=1;i<=n;i++)f[i]=0;
    for(i=1;i<=n;i=j){
      for(j=i;j<=n&&a[i]==a[j];j++);
      for(k=1;k<=j-i;k++)f[k]++;
    }
    for(i=j=1;i<=n;i++){
      while(!f[j])j++;
      f[j]--;
      printf("%d\n",i-j);
    }
  }
}
```

C. Segment Tree

线段树上代表区间长度相同的节点的子树点数相同，且最多只有$O(logn)$种本质不同的区间长度，对区间长度记忆化搜索即可。

时间复杂度$O(lognloglogn)$。

参考代码：

```
#include<cstdio>
#include<map>
using namespace std;
typedef long long ll;
int Case;ll n,k;map<ll,ll>T;
ll build(ll n){
  if(T.find(n)!=T.end())return T[n];
  if(n<=k)return T[n]=1;
  return T[n]=build(n/2)+build(n-n/2)+1;
}
int main(){
  scanf("%d",&Case);
  while(Case--){
    scanf("%lld%lld",&n,&k);
    T.clear();
    printf("%lld\n",build(n));
  }
}
```

D. Number of Cycles

- 要形成若干个没有公共边的环，充要条件是每个点的度数都是偶数。将点的度数限制与格子的限制结合起来，将每条边是否存在作为变量，可以列出$2nm$个异或方程，使用高斯消元求解即可。
- 若常数过大可以用bitset优化。
- 时间复杂度：$O((nm)^3)$。

参考代码：

```
#include<bits/stdc++.h>
#define rep(i,x,y) for(auto i=(x);i<=(y);++i)
#define dep(i,x,y) for(auto i=(x);i>=(y);--i)
#define fr first
#define sc second
using namespace std;
```

```cpp
typedef long long ll;
typedef pair<int,int>pii;
const int inf=1e9;
const int N=19;
const int mo=998244353;
char s[N];
int idh[N][N],ids[N][N];
bitset<1000>a[1000];
void sol(){
    int h,w;
    scanf("%d%d",&h,&w);
    int n=0,m=0;
    rep(i,1,h)rep(j,1,w-1)idh[i][j]=++m;
    rep(i,1,h-1)rep(j,1,w)ids[i][j]=++m;
    ++m;
    rep(i,1,h)rep(j,1,w){
        a[++n].reset();
        if(i>1)a[n][ids[i-1][j]]=1;
        if(j>1)a[n][idh[i][j-1]]=1;
        if(i<h)a[n][ids[i][j]]=1;
        if(j<w)a[n][idh[i][j]]=1;
    }
    rep(i,1,h-1){
        scanf("%s",s+1);
        rep(j,1,w-1)if(s[j]!='.'){
            a[++n].reset();
            a[n][idh[i][j]]=a[n][idh[i+1][j]]=a[n][ids[i][j]]=a[n][ids[i]
[j+1]]=1;
            a[n][m]=s[j]-'0';
        }
    }
    int nw=1;
    rep(j,1,m-1){
        int x=0;
        rep(i,nw,n)if(a[i][j]){
            x=i;break;
        }
        if(!x)continue;
        if(x!=nw)swap(a[nw],a[x]);
        rep(i,1,n)if(i!=nw&&a[i][j])a[i]^=a[nw];
        ++nw;
    }
    rep(i,nw,n)if(a[i].any()){
        printf("0\n");return;
    }
    int ans=1;
    rep(i,nw,m-1)ans=ans*2%mo;
    printf("%d\n",ans);
}
int main(){
    int t;
    scanf("%d",&t);
    rep(i,1,t)sol();
}
```

E. Bitwise OR

- 当n为偶数时，设$m = n/2 - 1$。
- 当n为奇数时，设$m = (n-1)/2$。
- 可以发现，$n \ mod \ i \leq m$，且当$i \leq m$时，有$n \ mod \ (n-i) = i$。于是可以得出$n \ mod \ i$取到$0 \sim m$的所有整数，因此答案会是$2^k - 1$，k的具体值判断一下即可。

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
#define LL long long
LL T,n;
int main()
{
    scanf("%lld",&T);
    while(T--){
        scanf("%lld",&n);
        if((n&(-n))==n){
            printf("%lld\n",max(0ll,n/2-1));
            continue;
        }
        while((n&(-n))!=n) n-=(n&(-n));
        printf("%lld\n",n-1);
    }
}
```

F. Race

从1号点开始DFS整个图，并把出栈序列记下来，那么若x能到达y，显然x晚于y出栈。因为图是平面图，考虑最有代表性的两种遍历方式：顺时针遍历和逆时针遍历，那么可以得到两个出栈序列。设a_i表示顺时针遍历图时i点的出栈序，b_i表示逆时针遍历图时i点的出栈序，那么x能到达y当且仅当$a_x > a_y$且$b_x > b_y$。

按照题意，选出来的点应当满足两两不可达，因此把a_i看作横坐标，b_i看作纵坐标，那么选了(a_i, b_i)就不能选它左下角以及右上角的所有点，因此选出来的点一定满足从左往右a递增且b递减，问题转化为求价值和最大的下降子序列，可以用树状数组优化朴素DP在$O(nlogn)$
的时间内得到最优解。

对于字典序最小最优解的求解，有一个方法是在DP值里直接用长度为n的01串来记录当前方案里选了哪些点，转移的时候需要支持字典序大小的比较、拷贝以及把单点从0修改成1。因此考虑使用可持久线段树来记录方案，那么单点修改的时间复杂度为$O(logn)$，比较两个方案的字典序时根据左子树是否相同来决定往左还是往右递归比较，时间复杂度也为$O(logn)$。在这里，注意到每个点只会加入一次，因此如果两棵线段树的根节点的指针不同，那么表示的01串一定不同，不需要额外维护Hash值。

时间复杂度$O(nlog^2n)$。

参考代码：

```
#include<cstdio>
#include<algorithm>
#include<vector>
using namespace std;
typedef long long ll;
const int N=100005,M=N*21;
int Case,n,m,i,x,y,w[N],cnt,dfn1[N],dfn2[N],q[N],fin[N];bool vis[N];
```

```
vector<int>g[N];
struct P{
  int x,y;
  P(){}
  P(int _x,int _y){x=_x,y=_y;}
  P operator-(const P&b)const{return P(x-b.x,y-b.y);}
}a[N],pivot;
int tot,l[M],r[M];
struct E{
  ll sum;int root;
  E(){}
  E(ll _sum,int _root){sum=_sum,root=_root;}
}bit[N],tmp,ans;
inline ll cross(const P&a,const P&b){return 1LL*a.x*b.y-1LL*a.y*b.x;}
inline bool cmp(int x,int y){return cross(a[x]-pivot,a[y]-pivot)>0;}
int ins(int x,int a,int b,int c){
  int y=++tot;
  if(a==b)return y;
  int mid=(a+b)>>1;
  if(c<=mid)l[y]=ins(l[x],a,mid,c),r[y]=r[x];
  else l[y]=l[x],r[y]=ins(r[x],mid+1,b,c);
  return y;
}
inline bool smaller(int x,int y){
  if(x==y)return 0;
  int a=1,b=n,mid;
  while(a<b){
    mid=(a+b)>>1;
    if(l[x]==l[y]){
      a=mid+1;
      x=r[x];
      y=r[y];
    }else{
      b=mid;
      x=l[x];
      y=l[y];
    }
  }
  return x>y;
}
void go(int x,int a,int b){
  if(!x)return;
  if(a==b){
    fin[++cnt]=a;
    return;
  }
  int mid=(a+b)>>1;
  go(l[x],a,mid);
  go(r[x],mid+1,b);
}
inline void up(E&a,const E&b){
  if(a.sum>b.sum)return;
  if(a.sum<b.sum){a=b;return;}
  if(smaller(b.root,a.root))a.root=b.root;
}
```

```
void dfs1(int x){
  if(vis[x])return;
  vis[x]=1;
  for(int i=0;i<g[x].size();i++)dfs1(g[x][i]);
  dfn1[x]=++cnt;
}
void dfs2(int x){
  if(vis[x])return;
  vis[x]=1;
  for(int i=((int)g[x].size())-1;i>=0;i--)dfs2(g[x][i]);
  dfn2[x]=++cnt;
  q[cnt]=x;
}
inline void modify(int x,const E&p){for(;x<=n;x+=x&-x)up(bit[x],p);}
inline E ask(int x){E t(0,0);for(;x;x-=x&-x)up(t,bit[x]);return t;}
int main(){
  scanf("%d",&Case);
  while(Case--){
    scanf("%d%d",&n,&m);
    cnt=0;
    for(i=0;i<=n;i++){
      g[i].clear();
      vis[i]=0;
      bit[i]=E(0,0);
    }
    for(i=1;i<=n;i++)scanf("%d%d%d",&a[i].x,&a[i].y,&w[i]);
    while(m--)scanf("%d%d",&x,&y),g[x].push_back(y);
    for(i=1;i<=n;i++){
      pivot=a[i];
      sort(g[i].begin(),g[i].end(),cmp);
    }
    dfs1(1);
    for(cnt=0,i=1;i<=n;i++)vis[i]=0;
    dfs2(1);
    ans=E(0,0);
    for(i=n;i;i--){
      x=q[i];
      tmp=ask(dfn1[x]);
      tmp.sum+=w[x];
      tmp.root=ins(tmp.root,1,n,x);
      up(ans,tmp);
      modify(dfn1[x],tmp);
    }
    printf("%lld\n",ans.sum);
    cnt=0;
    go(ans.root,1,n);
    for(i=1;i<=cnt;i++)printf("%d%c",fin[i],i<cnt?' ':'\n');
    for(i=0;i<=tot;i++)l[i]=r[i]=0;
    tot=0;
  }
}
```

G. Price of Diamonds

设$f_{i,j,k}$表示从$(1,1)$走到(i,j)，一路上收集了k个钻石时，钻石的单价最高能涨到多少，则$ans=max(k\times f_{n,n,k})$。

对于固定的(i,j)来说，考虑两个状态$f_{i,j,x}$和$f_{i,j,y}$，其中$x<y$，如果$f_{i,j,x}\le f_{i,j,y}$，则状态$f_{i,j,x}$一定不可能发展为最优解，可以剔除。对于每个(i,j)，用列表按照k升序保存所有状态，并剔除不可能成为最优解的状态即可。

随机数据下当$n=100$时，单个(i,j)的有效状态的峰值k大约为几千。时间复杂度$O(n^2k)$。

参考代码：

```
#include<cstdio>
#include<algorithm>
#include<vector>
using namespace std;
typedef long long ll;
typedef pair<int,int>P;
typedef vector<P>V;
const int N=105;
int Case,n,m,i,j,k,a[N][N],b[N][N];ll ans;V f[N][N];P pool[1000005];
inline void ext(const P&t){
  while(m&&pool[m].second<=t.second)m--;
  if(!m||pool[m].first<t.first)pool[++m]=t;
}
inline void merge(const V&A,const V&B,V&C){
  int ca=A.size(),cb=B.size(),i=0,j=0;
  m=0;
  while(i<ca&&j<cb)ext(A[i].first<B[j].first?A[i++]:B[j++]);
  while(i<ca)ext(A[i++]);
  while(j<cb)ext(B[j++]);
  C.resize(m);
  for(i=0;i<m;i++)C[i]=pool[i+1];
}
int main(){
  scanf("%d",&Case);
  while(Case--){
    scanf("%d",&n);
    for(i=1;i<=n;i++)for(j=1;j<=n;j++)scanf("%d",&a[i][j]);
    for(i=1;i<=n;i++)for(j=1;j<=n;j++)scanf("%d",&b[i][j]);
    f[1][1].resize(1);
    f[1][1][0]=P(a[1][1],b[1][1]);
    for(i=1;i<=n;i++)for(j=1;j<=n;j++){
      if(i==1&&j==1)continue;
      if(i==1)f[i][j]=f[i][j-1];
      else if(j==1)f[i][j]=f[i-1][j];
      else merge(f[i-1][j],f[i][j-1],f[i][j]);
      for(k=0;k<f[i][j].size();k++){
        f[i][j][k].first+=a[i][j];
        f[i][j][k].second+=b[i][j];
      }
    }
    ans=0;
    for(i=0;i<f[n][n].size();i++)ans=max(ans,1LL*f[n][n][i].first*f[n][n]
[i].second);
    printf("%lld\n",ans);
  }
}
```

H. Calculus

注意到题中所给的所有函数均为发散。所以只需要检查是否所有的构成函数的系数均为0即可。

参考代码：

```
#include<bits/stdc++.h>
using namespace std;

#define fi first
#define se second
#define pb push_back

typedef pair<int,int> pii;
typedef vector<pii> Vpii;


char s[MAXN];
void solve()
{
    scanf("%s",s+1);
    int lst=1,n=LEN(s+1);
    Vpii vec;
    for(int i=1;i<=n;i++)if(s[i]=='+')
        vec.pb({lst,i-1}),lst=i+1;
    vec.pb({lst,n});
    for(auto j:vec)
    {
        bool f=0;
        for(int i=j.fi;i<=j.se;i++)
            if(s[i]=='0'&&!isdigit(s[i-1]))f=1;
        if(!f) {
            printf("NO\n");
            return;
        }
    }
    printf("YES\n");
}

int main()
{
    int T;
    scanf("%d",&T);
    for(int i=1; i<=T; i++)
        solve();
    return 0;
}
```

I. Rectangle Queries

- 首先来看一个子问题：给一个数组，初始所有位置全是0，每次单点加一或减一，询问区间有多少个位置不为0。也许你很快会给出一个修改$O(logn)$查询$O(logn)$的优秀做法。那么如果强制规定修改O(1)，查询能做到什么效率？我们去考虑分块，维护两个数组$num[i]$和$sum[i]$分别维护第i位置上的值，以及第i块内的值不为0的位置个数。那么每次单点修改只需要修改$num[i]$和$sum[i/block_size]$两个位置的值，然后查询的时候需要$O(\sqrt{n})$效率的去查询区间所覆盖的完整

块的值以及区间两端散块的值（散块值可以没有）。这显然是一个修改$O(1)$，查询$O(\sqrt{n})$的做法。

- 然后考虑这个题，我们发现可以用莫队去维护询问的区间，即x坐标。$y(fx)$维度单独拎出来就变成了上述子问题。我们会神奇的发现，这样的复杂度是：$O(n\cdot\sqrt{n}\cdot 1+m\cdot 1\cdot\sqrt{n})$的，莫队单次修改时$O(\sqrt{n})$的复杂度遇上了值域上$O(1)$的修改，莫队单次查询时$O(1)$的复杂度遇上了值域查询时$O(\sqrt{n})$的复杂度。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
const int maxn=1e5+10;
int T,n,m,k,a[maxn],num[maxn],sum[maxn],ans[maxn];
struct da{int l,r,L,R,id;}q[maxn];
bool cmp(da aa,da bb){
        if (aa.l/k==bb.l/k){
                if ((aa.l/k)&1) return aa.r<bb.r;else return aa.r>bb.r;
       }
        return aa.l/k<bb.l/k;
}
void add(int x) {if (++num[x]==1) sum[x/k]++;}
void dec(int x) {if (--num[x]==0) sum[x/k]--;}
int calc(int x){
        int now=0;
        for (int i=0;i<x/k;i++) now+=sum[i];
        for (int i=(x/k)*k;i<=x;i++) now+=(num[i]>=1);
        return now;
}
void solve(){
        memset(num,0,sizeof num); memset(sum,0,sizeof sum); scanf("%d%d",&n,&m);
k=313;
        for (int i=1;i<=n;i++) scanf("%d",&a[i]);
        for (int i=1;i<=m;i++)
scanf("%d%d%d%d",&q[i].l,&q[i].L,&q[i].r,&q[i].R),q[i].id=i;
        sort(q+1,q+m+1,cmp); int pl=q[1].l,pr=q[1].r;
        for (int i=pl;i<=pr;i++) add(a[i]);
        ans[q[1].id]=calc(q[1].R)-calc(q[1].L-1);
        for (int i=2;i<=m;i++){
                while (pl>q[i].l) pl--,add(a[pl]);
                while (pr<q[i].r) pr++,add(a[pr]);
                while (pl<q[i].l) dec(a[pl]),pl++;
                while (pr>q[i].r) dec(a[pr]),pr--;
                ans[q[i].id]=calc(q[i].R)-calc(q[i].L-1);
        }
        for (int i=1;i<=m;i++) printf("%d\n",ans[i]);
}
int main(){
        freopen("1.in","r",stdin);
        freopen("1.out","w",stdout);
        scanf("%d",&T); while (T--) solve();
return 0;
}
```

J. XOR Sum

- 对数列做前缀异或，将题面转化为找两个距离最近的数，使得他们的异或值不小于k。
- 枚举靠右的那个数，同时维护字母树，字母树每个节点保存范围内最靠右的点的位置。根据k来询问对应的log个节点，从而更新答案。
- 时间复杂度:$O(nlogn)$。

参考代码:

```
#include<bits/stdc++.h>
#define rep(i,x,y) for(auto i=(x);i<=(y);++i)
#define dep(i,x,y) for(auto i=(x);i>=(y);--i)
#define fr first
#define sc second
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int inf=1e9;
const int N=1e5+10;
const int M=3e6+10;
const int mo=998244353;
int p[M][2],mx[M],a[N];
void sol(){
    int n,k;
    scanf("%d%d",&n,&k);
    rep(i,1,n){
        scanf("%d",&a[i]);
        a[i]^=a[i-1];
    }
    int anl=-1,anr=n,tot=1;
    mx[1]=-1;
    p[1][0]=p[1][1]=0;
    rep(i,0,n){
        int x=1,res=-1;
        dep(j,29,0){
            int w=(a[i]>>j)&1;
            if(!((k>>j)&1)){
                if(p[x][w^1])res=max(res,mx[p[x][w^1]]);
                x=p[x][w];
            }else x=p[x][w^1];
            if(!x)break;
        }
        if(x)res=max(res,mx[x]);
        if(res>=0&&i-res<anr-anl)anl=res,anr=i;
        x=1;
        dep(j,29,0){
            int w=(a[i]>>j)&1;
            if(!p[x][w]){
                p[x][w]=++tot;mx[tot]=-1;
                p[tot][0]=p[tot][1]=0;
            }
            x=p[x][w];
            mx[x]=max(mx[x],i);
        }
    }
    if(anl>=0)printf("%d %d\n",anl+1,anr);
    else printf("-1\n");
```

```
}
int main(){int t;
    scanf("%d",&t);
    rep(i,1,t)sol();
}
```

K. Zombie Movements

考虑所有点的个数减去不能到达的点的个数，即为可以到达的点的个数。

根据题意，有地雷的地方是不可以到达的。由于僵尸只会向右和向下走，当某个点的左边和上方都不可达时，该点不可达，并会对自己右边的点和下方的点造成影响。

由于空间很大但地雷数有限，可以从上往下逐行对每一行的地雷排序后进行处理。对每个地雷，找到从自己的右上角点$(x-1,y+1)$开始的从左往右的连续不可达区域的范围，那么x这行的这个范围也不可达。可以用线段树来实现区间查询和区间覆盖。每一行处理完后查询该行不可达的点数，累加后用总点数减即得到答案。

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
#define ls (x<<1)
#define rs (x<<1|1)
const int N = 1e5 + 5;
const int inf = 0x3f3f3f3f;
vector<int>e[N];
int tr[2][N << 2], lz[2][N << 2];

void push_down(int f, int x, int l, int r, int mid) {
    if (lz[f][x] == -1)return;
    tr[f][ls] = lz[f][x] * (mid - l + 1);
    tr[f][rs] = lz[f][x] * (r - mid);
    lz[f][ls] = lz[f][rs] = lz[f][x];
    lz[f][x] = -1;
}
void update(int f,int x, int l, int r, int L, int R, int v) {
    if (L <= l && R >= r) {
        tr[f][x] = (r - l + 1) * v;
        lz[f][x] = v;
        return;
    }
    int mid = (l + r) >> 1;
    push_down(f, x, l, r, mid);
    if (R <= mid)update(f, ls, l, mid, L, R, v);
    else if (L > mid)update(f, rs, mid + 1, r, L, R, v);
    else {
        update(f, ls, l, mid, L, mid, v);
        update(f, rs, mid + 1, r, mid + 1, R, v);
    }
    tr[f][x] = tr[f][ls] + tr[f][rs];
}
int query(int f, int x, int l, int r, int L, int R) {
    if (!tr[f][x])return inf;
    if (l == r)return l;
    int mid = l + r >> 1;
    push_down(f, x, l, r, mid);
```

```
    if (L <= l && R >= r) {
        if (tr[f][ls] > 0) return query(f, ls, l, mid, l, mid);
        else return query(f, rs, mid + 1, r, mid + 1, r);
    }
    else {
        if (R <= mid)return query(f, ls, l, mid, L, R);
        else if (L > mid)return query(f, rs, mid + 1, r, L, R);
        else return min(query(f, ls, l, mid, L, mid), query(f, rs, mid + 1, r,
mid + 1, R));
    }
}
int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        int n, m, k;
        scanf("%d %d %d", &n, &m, &k);
        for (int i = 1; i <= n; ++i)e[i].clear();
        for (int i = 1; i <= (m << 2); ++i) {
            tr[0][i] = tr[1][i] = 0;
            lz[0][i] = lz[1][i] = -1;
        }
        for (int i = 0; i < k; ++i) {
            int x, y;
            scanf("%d %d", &x, &y);
            e[x].push_back(y);
        }
        long long ans = 0;
        update(0, 1, 1, m, 1, 1, 1);
        for (int x = 1; x <= n; ++x) {
            int l = 0;
            sort(e[x].begin(), e[x].end());
            for (auto& y : e[x]) {
                if (y - 1 >= l + 1) {
                    int pos = query((x & 1) ^ 1, 1, 1, m, l + 1, y - 1);
                    if (pos != inf)update(x & 1, 1, 1, m, pos, y - 1, 1);
                }
                l = y;
            }
            if (l + 1 <= m) {
                int pos = query((x & 1) ^ 1, 1, 1, m, l + 1, m);
                if (pos != inf)update(x & 1, 1, 1, m, pos, m, 1);
            }
            ans += tr[x & 1][1];
            update((x & 1) ^ 1, 1, 1, m, 1, m, 0);
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

L. K-th Smallest Energy Consumption

二分答案，然后使用任意一种后缀数据结构check即可。

如何 check?

- 后缀数组：所有后缀的所有前缀即所有子串。我们遍历后缀数组，对于每个后缀，其越长的前缀能耗越大，于是可以二分找到能耗小于等于要check的值的前缀的个数，再减去重复部分即可。而每个后缀被重复统计的部分就是$height$数组对应的值。
- 后缀自动机/后缀树：这两种后缀数据结构都是将本质不同的子串记录在其结点上。每个结点表示的子串都是形如$Suffix(T,i)+S$的串（即取T的一个后缀和S连接构成的串，在后缀树上则是再翻转一次的串）。显然我们取的T的后缀越长，其表示的子串能耗越大，于是可以二分这个长度来找到满足条件的子串的个数，每次check对每个结点做一次二分即可。

时间复杂度$O(nlognlogk)$。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;

const int N = 100000 + 9;

char s[N];
int n, c[26];
long long k;

template <int N, int sigma> struct Suffix_Automaton {
    int tot, last, nxt[N * 2][sigma], len[N * 2], link[N * 2];
    int sum[N], pos[N * 2];
    vector<int> ch[N * 2];
    void init(int n) {
        tot = last = 0; link[0] = -1;
        memset(nxt, 0, (n * 2 + 5) * sigma * sizeof(int));
        memset(pos, 0, (n * 2 + 5) * sizeof(int));
        for (int i = 0; i < n * 2 + 5; ++i) ch[i].clear();
    }
    void add_char(int c) {
        int p = last, cur = last = ++tot; len[cur] = len[p] + 1;
        while (~p && !nxt[p][c]) { nxt[p][c] = cur; p = link[p]; }
        if (p == -1) { link[cur] = 0; return; }
        int q = nxt[p][c];
        if (len[q] == len[p] + 1) { link[cur] = q; return; }
        int _q = ++tot; len[_q] = len[p] + 1;
        memcpy(nxt[_q], nxt[q], sigma * sizeof(int));
        link[_q] = link[q]; link[q] = link[cur] = _q;
        while (~p && nxt[p][c] == q) { nxt[p][c] = _q; p = link[p]; }
    }
    void dfs(int u) {
        for (int v : ch[u]) dfs(v);
        if (!pos[u]) pos[u] = pos[ch[u][0]];
    }
    void add_string(const char s[], int n) {
        for (int i = 1; i <= n; ++i) {
            sum[i] = sum[i - 1] + c[s[i] - 'a'];
            add_char(s[i] - 'a');
            pos[last] = i;
        }
        for (int i = 1; i <= tot; ++i) {
            ch[link[i]].push_back(i);
        }
```

```
        dfs(0);
    }
    bool check(int C) {
        long long cnt = 0;
        for (int i = 1; i <= tot; ++i) {
            int P = pos[i], L = P - len[i] + 1, R = P - len[link[i]];
            while (L < R) {
                int M = (L + R) >> 1;
                if (sum[P] - sum[M - 1] <= C) R = M;
                else L = M + 1;
            }
            if (sum[P] - sum[L - 1] <= C) cnt += P - len[link[i]] - L + 1;
        }
        return cnt >= k;
    }
};

Suffix_Automaton<N, 26> sam;

void solve() {
    scanf("%d %lld", &n, &k);
    scanf("%s", s + 1);
    for (int i = 0; i < 26; ++i) {
        scanf("%d", c + i);
    }
    sam.init(n);
    sam.add_string(s, n);
    int L = 1, R = sam.sum[n];
    while (L < R) {
        int M = (L + R) >> 1;
        if (sam.check(M)) R = M;
        else L = M + 1;
    }
    if (sam.check(L)) printf("%d\n", L);
    else printf("-1\n");
}

int main() {
    int T; scanf("%d", &T);
    while (T--) solve();
    return 0;
}
```

M. Rockets

- 每次添加一个点，求以该点为圆心半径为r的圆中的权值和。
- 圆是比较难进行维护的一种形状，std选择了使用kd tree近似暴力地来进行维护，在随机数据下可以良好的呈现近似$O(n\sqrt{n})$的复杂度（可以通过造数据卡掉）。std使用了离线做法，不需要对树进行重构，通过标记点是否已经被加入树来对查询进行剪枝优化。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
#define fr first
```

```
#define sc second
typedef long long ll;
const int K=2;
const int N=5e5+10;
const int mod=1e9+7;
const double alpha=0.75;
ll qpow(ll a,ll n){ll res=1;while(n)
{if(n&1)res=res*a%mod;a=a*a%mod;n>>=1;}return res;}
int now;
struct Point
{
    int num[K],val,id;
    Point(){};
    Point(int xx,int yy,int vall){
        num[0]=xx,num[1]=yy,val=vall;
    }
    friend bool operator <(Point a,Point b)
    {
        return a.num[now]<b.num[now];
    }
}p[N],p1[N];
struct tree
{
    int l,r,siz,minn[K],maxx[K],tf,fa;
    ll sum;
    int exist;
    Point p;
};
struct KDT
{
    tree t[N];
    int id[N];
    int tot,root;
    int cnt,rubbish[N];
    void init()
    {
        cnt=0,tot=0;
    }
    int newnode(){
        if(cnt)return rubbish[cnt--];
        return ++tot;
    }
    void up(int u)
    {
        for(int i=0;i<K;i++){
            t[u].minn[i]=t[u].maxx[i]=t[u].p.num[i];
            if(t[u].l){
                t[u].minn[i]=min(t[u].minn[i],t[t[u].l].minn[i]);
                t[u].maxx[i]=max(t[u].maxx[i],t[t[u].l].maxx[i]);
            }
            if(t[u].r){
                t[u].minn[i]=min(t[u].minn[i],t[t[u].r].minn[i]);
                t[u].maxx[i]=max(t[u].maxx[i],t[t[u].r].maxx[i]);
            }
        }
```

```
        if(t[u].l)t[t[u].l].fa=u;
        if(t[u].r)t[t[u].r].fa=u;
        t[u].sum=t[t[u].l].sum+t[t[u].r].sum+(t[u].exist?t[u].p.val:0);
        t[u].siz=t[t[u].l].siz+t[t[u].r].siz+t[u].exist;
    }
    int build(int l,int r,int d)
    {
        if(l>r)return 0;
        now=d;
        int mid=(l+r)>>1;
        int u=newnode();
        nth_element(p1+l,p1+mid,p1+r+1);
        t[u].exist=0;
        t[u].p=p1[mid];
        id[p1[mid].id]=u;
        t[u].l=build(l,mid-1,(d+1)%K);
        t[u].r=build(mid+1,r,(d+1)%K);
        up(u);
        return u;
    }
    void change(int x){
        x=id[x];
        t[x].exist=1;
        for(;x;x=t[x].fa)up(x);
    }
    bool inside(int x1,int y1,int r,int X1,int Y1,int X2,int Y2)
    {
        int cnt=0;
        if(1ll*(X1-x1)*(X1-x1)+1ll*(Y1-y1)*(Y1-y1)<=1ll*r*r)cnt++;
        if(1ll*(X2-x1)*(X2-x1)+1ll*(Y1-y1)*(Y1-y1)<=1ll*r*r)cnt++;
        if(1ll*(X1-x1)*(X1-x1)+1ll*(Y2-y1)*(Y2-y1)<=1ll*r*r)cnt++;
        if(1ll*(X2-x1)*(X2-x1)+1ll*(Y2-y1)*(Y2-y1)<=1ll*r*r)cnt++;
        if(cnt==4)return true;
        else return false;
    }
    bool outside(int x1,int y1,int r,int X1,int Y1,int X2,int Y2)
    {
        int x,y;
        if(x1<=X2&&x1>=X1)x=x1;
        else if(x1<X1)x=X1;
        else x=X2;
        if(y1<=Y2&&y1>=Y1)y=y1;
        else if(y1<Y1)y=Y1;
        else y=Y2;
        if(1ll*(x1-x)*(x1-x)+1ll*(y1-y)*(y1-y)<=1ll*r*r)return false;
        else return true;
    }
    ll query(int u,int x1,int y1,int r)
    {
        if(!u)return 0;
        if(t[u].siz==0)return 0;

if(inside(x1,y1,r,t[u].minn[0],t[u].minn[1],t[u].maxx[0],t[u].maxx[1]))return
t[u].sum;
```

```
if(outside(x1,y1,r,t[u].minn[0],t[u].minn[1],t[u].maxx[0],t[u].maxx[1]))return
0;
        ll ans=0;

if(t[u].exist&&inside(x1,y1,r,t[u].p.num[0],t[u].p.num[1],t[u].p.num[0],t[u].p.n
um[1]))ans+=t[u].p.val;
        ans+=query(t[u].l,x1,y1,r)+query(t[u].r,x1,y1,r);
        return ans;
    }
}T1;
int r[N];
void solve()
{
    int n;
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d%d%d%d",&p[i].num[0],&p[i].num[1],&p[i].val,&r[i]);
        p[i].id=i;
        p1[i]=p[i];
    }
    T1.init();
    T1.root=T1.build(1,n,0);
    for(int i=1;i<=n;i++){

 printf("%lld\n",T1.query(T1.root,p[i].num[0],p[i].num[1],r[i])+p[i].val);
        T1.change(i);
    }
}
int main()
{
    int T;
    scanf("%d",&T);
    while(T--){
        solve();
    }
}
```