比赛链接

补题情况

题目	蒋贤蒙	王赵安	王智彪
D	2	0	0
E	0	0	0
L	0	0	2

题解

D. Contest Strategy

题意

给定 n 道题以及完成每道题需要的时间。对固定读题顺序，一个队伍会先按顺序读 k 道题，然后完成读过的题中需要时间最小的题。

然后按顺序读下一道题(如果还有题没读的话)，然后再完成读过且未完成的题中需要时间最小的题，不断重复，直到完成所有题。

对所有的读题顺序 ($1\sim n$ 的排列)，问这个队伍的总罚时之和。

题解

首先假如按做题顺序每题的做题时间分别为 $t_1,t_2\cdots t_n$，则总罚时为 $\sum_{i=1}^n (n+1-i)t_i$。

将所有题按所需时间大小排序，同时设完成第 i 题需要的时间为 a_i。不难发现对于最后 $k-1$ 道题，第 i 题一定也是做题顺序中的第 i 题。

接下来只考虑前 $n-k+1$ 道题，设 $f(i,j)$ 表示第 i 道题在读完 j 题后已经被完成的总方案数。

不难发现第 i 道题在读完 j 题后已经被完成等价于读题顺序前 j 道题一定包含第 i 题且至少包含 $k-1$ 道需要时间大于 i 的题。

考虑枚举前 j 题中一定包含第 i 题且正好包含 p 道需要时间大于 i 的题的方案数，于是有

$$ f(i,j)=j!(n-j)!\sum_{p=k-1}^{j-1}{n-i\choose p}{i-1\choose j-p-1} $$

然后第 i 道题在读完 j 题后恰好被完成的方案就是 $f(i,j)-f(i,j-1)$，此时对答案的贡献为

$$ (n+k-j)a_i\times (f(i,j)-f(i,j-1)) $$

时间复杂度 $O\left(n^3\right)$。

```
const int mod=1e9+7,MAXN=305;
int a[MAXN],frac[MAXN],invf[MAXN];
```

```
int dp[MAXN][MAXN];
int quick_pow(int a,int k){
    int ans=1;
    while(k){
        if(k&1)ans=1LL*ans*a%mod;
        a=1LL*a*a%mod;
        k>>=1;
    }
    return ans;
}
int C(int n,int m){
    if(n<m)return 0;
    return 1LL*frac[n]*invf[m]%mod*invf[n-m]%mod;
}
int main(){
    int n=read_int(),k=read_int();
    _rep(i,1,n)a[i]=read_int();
    sort(a+1,a+n+1);
    frac[0]=1;
    _for(i,1,MAXN)
    frac[i]=1LL*frac[i-1]*i%mod;
    invf[MAXN-1]=quick_pow(frac[MAXN-1],mod-2);
    for(int i=MAXN-1;i;i--)
    invf[i-1]=1LL*invf[i]*i%mod;
    LL ans=0;
    _rep(i,1,n-k+1){
        _rep(j,k,n){
            _for(t,k-1,j)
            dp[i][j]=(dp[i][j]+1LL*C(n-i,t)*C(i-1,j-t-1))%mod;
            dp[i][j]=1LL*dp[i][j]*frac[j]%mod*frac[n-j]%mod;
            ans=(ans+1LL*(dp[i][j]-dp[i][j-1])*(n+k-j)%mod*a[i])%mod;
        }
    }
    _rep(i,n-k+2,n)
    ans=(ans+1LL*frac[n]*a[i]%mod*(n+1-i))%mod;
    enter(ans);
    return 0;
}
```

L. Windy Path

题意

给定 n 个点，并且给一个操作序列 str ，其中 $str[i]$ 可以为 L ，也可以为 R ，如果是 L ，则需要你给的点的序列满足 $p[i],p[i+1],p[i+2]$ 是逆时针排序， R 则需要为顺时针，请给出满足要求的操作序列。

题解

比赛的时候被这个题的题意吓到了，因为是打着计算几何幌子的其他题，结果真的是计算几何…，给我的队友们道歉 qwq 其实我们只需要满足每一个操作之前我们都准备好下一步一定能进行这样的操作就可以了，随便选第一个点，从第二个点开始，比如我下一步是 L 那我这一步就从一个点走到逆时针的第一个点（其实是沿着凸包走），这样我从这个方向出发，到剩下的任意一个点都是逆时针方向 R 的话就到顺时针的下一个点，这样再下一个操作走到任意一个位置都是顺时针。

然后第二个操作开始，就按照上面的规律，一直走下去就行了，这样是一定有解的。因为数据范围很小，时间很充裕，每一步都暴力求凸包就可以了。

```
char str[1001];
bool vis[1001];
int main() {
    scanf("%d",&PO.n);
    int nn=PO.n;
    for(int i=0; i<PO.n; i++) {
        scanf("%lf %lf",&PO.p[i].x,&PO.p[i].y);
        PO.p[i].id=i;
    }
    scanf("%s",str+1);
    PO.Graham(po);
    int pos1=po.p[0].id+1;
    printf("%d ",pos1);
    vis[pos1]=true;
    int laspos,lasid;
    for(int i=0; i<po.n; i++) {
        if(po.p[i].id==pos1-1) {
            laspos=i;
            lasid=pos1-1;
            break;
        }
    }
    for(int i=1; i<=nn-2; i++) {
//        for(int j=0; j<po.n; j++) {
//            po.p[j].output();
//        }
        int mmp=(str[i]=='L')?1:-1;
        int tid=po.p[(laspos+mmp+po.n)%po.n].id;
        printf("%d ",tid+1);
        vis[tid+1]=true;
        int tpos=0;
        for(int j=0; j<PO.n; j++) {
            if(PO.p[j].id==lasid) {
                tpos=j;
                break;
            }
        }
        PO.p[tpos]=PO.p[PO.n-1];
        PO.n--;
```

```
        PO.Graham(po);
        for(int j=0; j<po.n; j++) {
            if(po.p[j].id==tid) {
                laspos=j;
                lasid=tid;
                break;
            }
        }
    }
    for(int i=1; i<=nn; i++) {
        if(!vis[i]) {
            printf("%d\n",i);
            return 0;
        }
    }
    return 0;
}
```