A.D.E 签到题

B. Farmer John Solves 3SUM

仅为解法之一：基于区间长度的区间dp

令 dp[i][j] 表示所求的三元组 (a,b,c) 满足 $i \le a,b,c \le j$ 的组数

区间长度从小到大进行规划，则可知大区间可以由比其小的区间规划而来

所以可以将 dp[i][j] 中的 i 和 j 看作是代表的答案的左右区间

首先得到相邻的 dp[i][j-1]+dp[i+1][j]，即三元组均在左右区间为 [i,j-1] 以及 [i+1,j] 之内

发现这样计数时，中间的满足左右区间为 [i+1,j-1] 的三元组被重复计数，故需减去dp[i+1][j-1]

最后考虑到这样转移还需要加上表示当前状态的答案，即三元组中有两个固定为 i 和 j 时，满足题意的组数

既然固定了 i 和 j，表示固定了其中两个数，第三个数 a_k 可以由 $a_i+a_j+a_k=0$ 推得 $a_k=-(a_i+a_j)$

引入cnt数组动态表示当前 [i+1.j-1] 内各数字出现的次数，则对于状态转移方程，能得到

`dp[i][j]=dp[i][j-1]+dp[i+1][j]-dp[i+1][j-1]+cnt[-a[i]-a[j]];`

因为数组索引需要为非负数，又考虑到数据范围为 $-10^6 \le A_i \le 10^6$

所以可以将读入的数全部加上 $ave=10^6$（稍大一点），使其全部成为非负数，再用 cnt[0~2000000] 来表达

每个数都加上基准后，$a_i+a_j+a_k=ave*3$，故第三个数字 $a_k=ave*3-(a_i+a_j)$

并且注意在使用状态转移方程时判断 a_k 是否会越界（致RE）

三元组最小长度为3，故从3开始枚举长度

首先，将 [l+1,r-1] 的数加入cnt数组中

每次枚举，让左边界从1开始，右边界则从len开始，对于cnt即对应 [2,len-1]

每次左边界与右边界需要往右移动一格（窗口整体右移）

所以原本 A[i+1] 的位置会变成左边界，使其从cnt数组中减去

原本 A[j] 的位置会从此时的右边界变成界内元素，故将其加入cnt数组中

整段处理结束后（右边界越界时），需要将cnt数组清零，但直接memset或者遍历清零复杂度很高

所以考虑到最后一次的左右边界分别为 n-len+1 以及 n

所以此时 cnt 数组表示的范围为 [n-len+2,n-1]

又因为转移而使得表示范围变成 [n-len+3,n]，所以将这一段遍历清零即可

处理完dp数组，对于每个询问 l 与 r，输出 dp[l][r] 即可

```
#include<bits/stdc++.h>
```

```
using namespace std;
typedef long long ll;

const int ave=1000025,ave3=3000075; //读入数所需要加的基准
int A[5050];
ll dp[5050][5050];
int cnt[2000050];

void solve()
{
    int n,q,l,r,tmp;
    cin>>n>>q;
    for(int i=1;i<=n;i++)
        cin>>A[i],A[i]+=ave;
    for(int len=3;len<=n;len++)
    {
        for(int i=2;i<len;i++)
            cnt[A[i]]++;
        for(int i=1,j=len;j<=n;i++,j++)
        {
            tmp=ave3-(A[i]+A[j]);
            if(tmp>=0&&tmp<2000050)
                dp[i][j]=dp[i][j-1]+dp[i+1][j]-dp[i+1][j-1]+cnt[tmp];
            else
                dp[i][j]=dp[i][j-1]+dp[i+1][j]-dp[i+1][j-1];
            cnt[A[j]]++;
            cnt[A[i+1]]--;
        }
        for(int i=n-len+3;i<=n;i++)
            cnt[A[i]]--;
    }
    while(q--)
    {
        cin>>l>>r;
        cout<<dp[l][r]<<'\n';
    }
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    solve();
    return 0;
}
```

--wqt

C. Cave Paintings

题意:

给一个洞穴装水，'#'表示石头，水是有重力的，如果某一格有水的话，它所在的联通块（四联通）中所有不高于自己的格子也要有水，问有多少种装水方案

思路:

如果能求出每个联通块的方案数，那么只要把他们乘起来就是答案了。

考虑dp：

对于当前层i来说，假设他下面一层$i+1$的方案数已经求好了，比如：

```
#...#..#..#<-当前层
#.#.#.#####
###########
12345678901
```

那么存在三种情况:

1.低一层的几个水坑合并在一起了（2-5列）

2.没有合并，但继承了上一个水坑 （6-7列）

3.新的水坑 （9-10列）

方便起见，设$dp[i][j]$为第i行左数第j个水坑的方案数（不考虑比i小的行）

那么对于上面的情况，很容易手写出转移:

$$dp[1][1] = dp[2][1] * dp[2][2] + 1$$
$$dp[1][2] = dp[2][3] + 1$$
$$dp[1][3] = 1 + 1$$

其实就是这个水坑下面的方案总数+这个水坑装满水的情况（也就是1）

现在就只要知道这个水坑下面有那些水坑就行了，这部分用并查集之类的可以很方便的求出来

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll; typedef double db;
typedef pair<int, int> pii; typedef pair<ll, ll> pll;
typedef pair<int, ll> pil; typedef pair<ll, int> pli;
#define Fi first
#define Se second
#define _Out(a) cerr<<#a<<" = "<<(a)<<endl
const int INF = 0x3f3f3f3f, MAXN = 1e3 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1e9 + 7;
const db Pi = acos(-1), EPS = 1e-6;

int fa[MAXN * MAXN];
ll dp[MAXN * MAXN];
int fifa(int x) { return x == fa[x] ? x : fa[x] = fifa(fa[x]); }
int n, m;
char str[MAXN][MAXN];
int getid(int x, int y){return y + (x - 1) * m;}
void work()
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)scanf("%s", str[i] + 1);
    int cnt = 0;
    ll ans = 1;
    for (int i = n - 1; i > 1; i--)
    {
        set<int>st;
        for (int j = 2; j < m; j++){
            if (str[i][j] == '#')continue;
```

```
            fa[getid(i, j)] = getid(i, j);
            dp[getid(i, j)] = 1;
            if (str[i][j - 1] == '.')
            {
                int fx = fifa(getid(i, j)), fy = fifa(getid(i, j - 1));
                if (fx == fy)continue;
                dp[fy] = dp[fy] * dp[fx] % MOD;
                fa[fx] = fy;
            }
            if (str[i + 1][j] == '.')
            {
                int fx = fifa(getid(i, j)), fy = fifa(getid(i + 1, j));
                if (fx == fy)continue;
                dp[fy] = dp[fx] * dp[fy] % MOD;
                fa[fx] = fy;
            }
        }
        for (int j = 2; j < m; j++)
        {
            if (str[i][j] == '#')continue;
            st.insert(fifa(getid(i, j)));
        }
        ll QAQ = 1;
        for (auto x : st)
        {
            dp[x] = (dp[x] + 1) % MOD;
        }
    }
    for (int i = n - 1; i > 1; i--)
    {
        for (int j = 2; j < m; j++)
        {
            if (str[i][j] == '#')continue;
            if (fifa(getid(i, j)) == getid(i, j))
                ans = ans * dp[getid(i, j)] % MOD;
        }
    }
    printf("%lld\n", (ans) % MOD);
}
int main(){
    work();
}
/*
5 5
#####
#.#.#
##..#
#.#.#
#####

ans:16

6 6
######
#....#
#.#..#
######
#....#
```

```
######

ans:10
*/
```

--ztc

F.

题目大意

给出一张有向图,求出一条经过点1的回路,使得回路的权值减去$C*T^2$最大,T为回路的长度

若回路为$p_1-p_2-p_3...-p_k$,那么权值为$a[p_1]+a[p_2]+a[p_3]+...+a[p_k]$

考虑处理出所有可能对答案有贡献的回路,这些回路的长度显然不会太大

当$T>1000$时,由于$C>=1$,$a_i<=1000$,此时的回路权值$\sum a_i$减去$C*T^2$一定非正,可以不用考虑

那么只需要处理所有合法的长度不大于1000的回路

方便起见,将回路视为以1为起点,1终点的路径,问题转化为求所有合法路径最大权

用类似$Bellman-Ford$算法的方式求出每种长度的路径的最大权值

令$dp[i][j]$为以1为起点,长度为i,终点为j的路径的最大权值

对于边$x->y$,转移就是

$$dp[i+1][y]=max(dp[i+1][y],dp[i][x]+a[y])$$

最后将ans为$\max_{T=0}^{1000}dp[T][1]-C*T^2$

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;typedef double db;
typedef pair<int, int> pii;typedef pair<ll, ll> pll;
typedef pair<int,ll> pil;typedef pair<ll,int> pli;
#define Fi first
#define Se second
#define _Out(a) cerr<<#a<<" = "<<(a)<<endl
const int INF = 0x3f3f3f3f, MAXN = 1e3 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1004535809;
const db Pi = acos(-1), EPS = 1e-6;
ll dp[MAXN],tmp[MAXN],val[MAXN];
vector<int>to[MAXN];
void work()
{
    int n,m,c;scanf("%d%d%d",&n,&m,&c);
    ll maxv=0;
    for(int i=1;i<=n;i++)
    {
        scanf("%lld",val+i);
        maxv=max(maxv,val[i]);
    }
    for(int i=1;i<=m;i++)
    {
        int u,v;scanf("%d%d",&u,&v);
        to[u].push_back(v);
```

```
    }
    memset(dp,0xcf,sizeof dp);
    dp[1]=0;ll ans=0;
    for(ll day=1;;day++)
    {
        if(day*c>maxv)break;
        memset(tmp,0xcf,sizeof tmp);
        for(int i=1;i<=n;i++)
        {
            for(auto v:to[i])
            {
                tmp[v]=max(tmp[v],dp[i]+val[v]);
            }
        }
        memcpy(dp,tmp,sizeof tmp);
        ans=max(ans,dp[1]-c*day*day);
    }
    printf("%lld\n",ans);
}
int main{
        work();
}
```

G.Loan Repayment

题意：

你欠了N元，每天还$\max(M,\frac{N'}{X})$，其中N'是剩下还欠的钱，找出最大的X使得接下来K天内能还完这N元

题解：

有单调性，可以二分X，接下来要想办法check

性质1：每天还的钱的取值范围是$[M,\frac{N}{X}]$,即最多有$\frac{N}{X}-M+1$种取值，但X较小时复杂度还是很大

性质2：假设A_i为取值为i的天数，那么$\sum_{\text{所有可能的取值}i} A_i \times i \geq N$ 里i的取值是$\sqrt{N}$的，因为最坏情况下$A_i=1$,且$i=M,M+1,M+2...,M+q$，那么q只要满足$\frac{(M+M+q)\cdot(q+1)}{2} \geq N$即可，显然q是$O(\sqrt{N})$的

所以只要枚举还钱的种类就可以了，最终复杂度$O(\sqrt{N}\log N)$

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;typedef double db;
typedef pair<int, int> pii;typedef pair<ll, ll> pll;
typedef pair<int,ll> pil;typedef pair<ll,int> pli;
#define Fi first
#define Se second
#define _Out(a) cerr<<#a<<" = "<<(a)<<endl
const int INF = 0x3f3f3f3f, MAXN = 1e3 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1004535809;
const db Pi = acos(-1), EPS = 1e-6;
ll n, k, m;
bool check(ll x)
```

```
{
    ll now=n;
    ll tim=0;
    for(;now/x>m;)
    {
        ll tak=now/x;
        ll a=cede(now-tak*x+1,tak);
        tim+=a;
        now-=tak*a;
    }
    tim+=cede(now,m);
    return tim<=k;
}
void work()
{
    scanf("%lld%lld%lld",&n,&k,&m);
    ll l=1,r=n;
    while(l<r)
    {
        ll mid=(l+r+1)>>1;
        if(check(mid))l=mid;
        else r=mid-1;
    }
    printf("%lld\n",l);
}
int main(){
        work();
}
```

--ztc

H.Non-Decreasing Subsequences

题意：
现在有n个数，每个数的值<=k，每次询问你区间l到r内有多少个序列是非下降的。

题解：
我一开始以为是SOSDP，但是怎么想也想不出来，看了别人的题解才知道是CDQ，并且这个CDQ又是我以前没写过的那种。
$CDQ(l,r,vec)$表示当前处理的是l到r区间，并且这个区间内的询问在vec数组中。
由于有些区间没有询问，并且这个不需要进行偏序操作，所以先序询问。
然后我们要处理出来的东西是：$dp[i][j]$
1.在i>mid的时候，$dp[i][j]$表示起始的值是j，到位置i的时候的非降序序列的个数。
2.在i<=mid的时候，$dp[i][j]$表示最终的值是j，到位置i的时候的非降序序列的个数。
使用树状数组加速。
注意第二种请况是要从后往前去做的。
然后由于有空集可以存在，所以我们在做完dp之后需要+1。
然后查看每个询问是否跨越了mid，如果是的话，那么我们需要处理出一个后缀和sum：
sum[j]表示第二种dp数组中起始值>=j，到位置q[i].r的时候的情况数。
然后就枚举左半部分的终值去做出答案即可。

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

```
const int N=5e4+5,M=25;
const ll mod=1e9+7;
int a[N];
ll dp[N][M],sum[M],num[M];
struct node{
    int l,r;
    ll ans;
}q[N*5];
int lowbit(int x){return x&(-x);}
void clear(){memset(num,0,sizeof(num));}
void add(int x,int v){
    for(int i=x;i<M;i+=lowbit(i))
        num[i]=(num[i]+v)%mod;
}
ll query(int x){
    ll ans=0;
    for(int i=x;i;i-=lowbit(i))
        ans=(ans+num[i])%mod;
    return ans;
}
int n,k;
void CDQ(int l,int r,vector<int>vec){
    if(l==r){
        for(auto i:vec)
            q[i].ans=2;
        return ;
    }
    int mid=l+r>>1;
    for(int s=1;s<=k;s++){
        clear();
        for(int i=mid+1;i<=r;i++){
            dp[i][s]=query(a[i]);
            if(a[i]==s)dp[i][s]=(dp[i][s]+1)%mod;
            add(a[i],dp[i][s]);
        }
        for(int i=mid+2;i<=r;i++)
            dp[i][s]=(dp[i][s]+dp[i-1][s])%mod;
    }
    for(int s=1;s<=k;s++){
        clear();
        for(int i=mid;i>=l;i--){
            dp[i][s]=query(k+1-a[i]);
            if(a[i]==s)dp[i][s]=(dp[i][s]+1)%mod;
            add(k+1-a[i],dp[i][s]);
        }
        for(int i=mid-1;i>=l;i--)
            dp[i][s]=(dp[i][s]+dp[i+1][s])%mod;
    }
    for(int i=l;i<=mid;i++)
        dp[i][1]=(dp[i][1]+1)%mod;
    for(int i=mid+1;i<=r;i++)
        dp[i][k]=(dp[i][k]+1)%mod;
    vector<int>lef,rig;
    lef.clear(),rig.clear();
    for(auto i:vec){
        if(q[i].r<=mid)lef.push_back(i);
        else if(q[i].l>mid)rig.push_back(i);
        else{
```

```
            for(int j=k;j;j--)
                sum[j]=(sum[j+1]+dp[q[i].r][j])%mod;
            for(int j=1;j<=k;j++)
                q[i].ans=(q[i].ans+dp[q[i].l][j]*sum[j])%mod;
        }
    }
    if(lef.size())CDQ(l,mid,lef);
    if(rig.size())CDQ(mid+1,r,rig);
}
int main()
{
    scanf("%d%d",&n,&k);
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    int que;
    scanf("%d",&que);
    vector<int>vec;
    for(int i=1;i<=que;i++)
        scanf("%d%d",&q[i].l,&q[i].r),vec.push_back(i);
    CDQ(1,n,vec);
    for(int i=1;i<=que;i++)
        printf("%lld\n",q[i].ans);
    return 0;
}
```

--yf

J.Springboards

题意:

你要从(1,1)走到(n,n)，你每次可以往右或往上走一步，现在有一些跳板，当你走到左下角的时候，你就会跳到右上角，并且不算步数，问你需要走最少多少步。

题解:

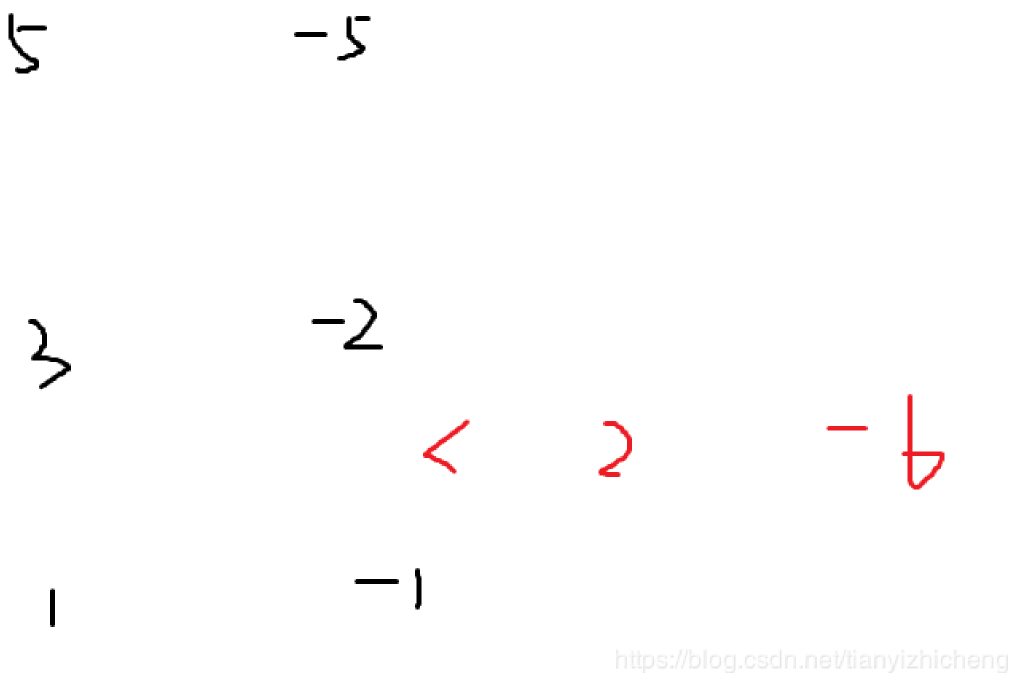
我感觉这道题CDQ分治可以做，但是写到一半的时候忘掉了。。。
我用ans数组表示到第i个跳板可以减掉的最长步数。
首先为了保持每个点位置的一致性，需要将跳板的起点和终点拆开进行分步运算。
那么对于每个跳板的起始位置，肯定是找到它左下角的最大答案进行转移，那么我们就需要按照x轴进行排序，然后查找的话可以用线段树，当然也可以用别的方法。
对于跳板的终点位置，我用一个类似单调栈的东西，也就是y从小到大的时候，值是从大到小的，这样查询的时候，它下面的第一个存在的位置就是最优解。比如说我现在有这样一个序列:
表示y=1的时候答案为-1,...

5 -5

3 -2

1 -1

然后新进来一个数，在y=2的时候，答案最小是-6



那么此时就要把3,5排出去，因为他们更劣。
用map一直维护这个东西就好了。

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
int ans[N];
struct node{
    int x,y,id,f;
    bool operator< (const node& a)const {
        if(x!=a.x)
            return x<a.x;
        return y<a.y;
    }
}e[N*2];
map<int,int>mp;
int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++){
        scanf("%d%d",&e[i*2-1].x,&e[i*2-1].y),e[i*2-1].id=i;
        scanf("%d%d",&e[i*2].x,&e[i*2].y),e[i*2].id=i,e[i*2].f=1;
    }
    sort(e+1,e+1+m*2);
    map<int,int>::iterator it;
    mp[0]=0;
    for(int i=1;i<=m*2;i++){
        if(!e[i].f){
            it=mp.upper_bound(e[i].y);
            it--;
            ans[e[i].id]=e[i].x+e[i].y+it->second;
        }
        else{
            it=mp.upper_bound(e[i].y);
```

```
            it--;
            int v=ans[e[i].id]-e[i].x-e[i].y;
            if(it->second<=v)continue;
            it++;
            while(it!=mp.end()&&it->second>v)
                mp.erase(it++);
            mp[e[i].y]=v;
        }
    }
    it=mp.end();
    it--;
    printf("%d\n",n*2+it->second);
    return 0;
}
```

--yf

K.Berry Picking

题意：

从N棵树上摘果子，有K个篮子，一个篮子只能装同一棵树上的果子，要求最小的$\frac{K}{2}$个篮子中的果子数量最多

题解：

如果某一次你的果子方案是$6,5,4,3$,那么$4,4,4,3$就更加可行一点，也就是前$\frac{K}{2}$大的篮子中的果子数应该要等于第$\frac{K}{2}+1$大的篮子中的果子数，于是可以枚举这个篮子中的果子数，然后贪心选剩下的果子。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;typedef double db;
typedef pair<int, int> pii;typedef pair<ll, ll> pll;
typedef pair<int,ll> pil;typedef pair<ll,int> pli;
#define Fi first
#define Se second
#define _Out(a) cerr<<#a<<" = "<<(a)<<endl
const int INF = 0x3f3f3f3f, MAXN = 1e3 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1004535809;
const db Pi = acos(-1), EPS = 1e-6;

int a[MAXN];
int lef[MAXN];
void work()
{
    int n,k;
    scanf("%d%d",&n,&k);
    for (int i=1;i<=n;i++)
    {
        scanf("%d",a+i);
    }
    int ans=0;
    for(int i=1;i<=1000;i++)
    {
        int cnt=0;
```

```
        for(int j=1;j<=n;j++)
        {
            cnt+=a[j]/i;
            lef[j]=a[j]%i;
        }
        if(cnt<k/2)break;
        if(cnt>=k)ans=max(ans,k/2*i);
        else
        {
            sort(lef+1,lef+1+n);
            int tmp=(cnt-k/2)*i;
            for(int j=n;cnt<k;cnt++,j--)
            {
                if(j<=0)break;
                tmp+=lef[j];
            }
            ans=max(ans,tmp);
        }

    }
    printf("%d\n",ans);
}
int main(){
        work();
}
```

--ztc