A.C.D.F 签到题

B.TOPOVI

我的思路是递推答案，解题过程就是画一个图，思考每一次操作对答案的影响，分为以下三类（以列为例子）：

- 列值在更新前>0，
  - 更新后>0，答案单纯与那些行值等于列值的情况有关，新旧的数值都要考虑。
  - 更新后变成0，本列相当于删除，和n有关，与行值等于列值的个数有关。
- 列值在更新前=0，
  - 更新后>0，相当于新建一条线，和n有关，与行值等于列值的个数有关。（相当于上种情况相反）
  - 更新后还是0，（不存在的，这说明棋子的权重要是0，数据范围不包括）忽略。

下面是对列值的操作函数，可以拷贝一份做转换就是行值的操作了。

`key`表示要更新的的列。

`val`表示这个棋子的权值。

其实不懂c++容器的细节的话，还挺难写对的，因为维护`SIZE & row_v`的细节挺需要斟酌的，下面代码也不长，留给读者思考。

```
// using ll = long long
// using pii = pair<ll,ll>
#define SIZE(x) ((long long)(x.size()))

map<ll, ll> row, col;
map<ll, ll> row_v, col_v;
map<pii, ll> a;
ll n, k, p;
ll ans;

// void upd_r(ll key, ll val) { ... }

void upd_c(ll key, ll val) {
    ll v_old, v_new;
    if (col.count(key) == 0) { // 防止[]运算符，创建map元素。
        col.emplace(key, val);
        v_old = 0;
        v_new = val;
    } else {
        v_old = col[key];
        col[key] ^= val;
        v_new = col[key];
        if (col[key] == 0) col.erase(key); // 优化map的规模，排除0列.
    }
    col_v[v_old]--;
    col_v[v_new]++;
    // 前面是更新和维护数值， 下面是推算答案的公式逻辑
    if (v_old > 0) {
```

```
        if (v_new > 0) {
            ans = ans + row_v[v_old] - row_v[v_new];
        } else {
            ans = ans - n;
            ans = ans + row_v[v_old] + SIZE(row);
        }
    } else {
        ans = ans + n;
        ans = ans - row_v[v_new] - SIZE(row);
    }
}
```

主函数:

```
{
    cin >> n >> k >> p;
    for (int i = 1; i <= k; ++i) {
        ll r, c, ai;
        cin >> r >> c >> ai;
        upd_r(r, ai);
        upd_c(c, ai);
        a[pii(r, c)] = ai;
    }
    for (int i = 1; i <= p; ++i) {
        ll r1, c1, r2, c2;
        cin >> r1 >> c1 >> r2 >> c2;
        ll val = a[pii(r1, c1)];
        a[pii(r2, c2)] = val;
        a.erase(pii(r1, c1));   // 优化map的规模

        upd_r(r1, val);
        upd_c(c1, val);
        upd_r(r2, val);
        upd_c(c2, val);
        cout << ans << endl;
    }
}
```

--wtw

E: RELATIVNOST

题意:

两种画（有颜色的和黑白的）。有N个客户会购买其中的一种（至少会买一种但不会两种都买），第i个客户最多买a_i幅有颜色的画，且最多b_i幅黑白的画。
老板希望至少有C个人买了有颜色的画，希望求出满足这样条件的购买情况数。
客户会不断修改需求，要求在每次修改需求后输出一次结果。

解题思路:

整体思路是考虑得到一个初始答案，然后在修改的时候进行。
至少有C个人买了有颜色的画的情况数$<=>$情况总数减去少于C个人买了有颜色的画的情况数。
情况总数就是$sum = \prod_{i=1}^{n}(a_i + b_i)$。
考虑没有修改的情况时，可以用 $dp_{i,j}$表示前i个物体，有j个人买了有颜色的画的情况数。

那么答案就是$ans = sum - \sum_{i=0}^{c-1} dp_{n,i}$
同时得到dp转移方程$dp_{i,j} = dp_{i-1,j-1} \times a_i + dp_{i-1,j} \times b_i$
因为dp转移时添加次序可以任意调换，不会影响结果，所以可以改写成

$$dp_{S\cap(a,b),j} = dp_{S,j-1} \times a + dp_{S,j} \times b$$

这样就得到了加入新顾客时的答案转移方程。要修改顾客的话，还需要一个删除顾客时的答案转移方程，就得到了下面的式子。

$$dp_{S,j} = (dp_{S\cap(a,b),j} - dp_{S,j-1} \times a) \times inv(b)$$

然后就可以在$o(c)$和$o(clog)$的复杂度内进行加入删除，也就是修改操作。
(情况的总数也很好维护)
理论上可行了，但是操作的时候会有数字大于模数的情况，也就意味着当数字为模数倍数的时候会没有逆元，就无法进行除法操作，要把这里处理一下，我这里是认定这样的数出现次数不多，就暴力存起来了。
复杂度$o((n+q)c\log + \text{感觉不大})$

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod=10007;
typedef pair<ll,ll> P;
ll fpow(ll a,ll n)
{
    ll sum=1,base=a;
    while(n!=0)
    {
        if(n%2)sum=sum*base%mod;
        base=base*base%mod;
        n/=2;
    }
    return sum;
}
ll inv(ll a)
{
    return fpow(a,mod-2);
}

ll a[100005],b[100005];
ll n,c;
ll dp[21],tmp[21],tmp2[21];
ll sum=1;
map<P,ll>mp;
void add(ll a,ll b)
{
    if((a+b)%mod==0||b%mod==0)
    {
        mp[P(a,b)]++;
        return ;
    }
    sum=sum*(a+b)%mod;
    tmp[0]=dp[0]*b%mod;
    for(ll i=1;i<=c;i++)
    {
        tmp[i]=(dp[i-1]*a+dp[i]*b)%mod;
    }
```

```
    for(ll i=0;i<=c;i++)dp[i]=tmp[i];
}

void del(ll a,ll b)
{
    if((a+b)%mod==0||b%mod==0)
    {
        mp[P(a,b)]--;
        return ;
    }
    sum=sum*inv(a+b)%mod;
    ll ivb=inv(b);
    tmp[0]=(dp[0]*ivb)%mod;
    for(ll i=1;i<=c;i++)
    {
        tmp[i]=(dp[i]-tmp[i-1]*a%mod+mod)%mod*ivb%mod;
    }
    for(ll i=0;i<=c;i++)dp[i]=tmp[i];
}

ll getAns()
{
    ll ans=sum;
    for(ll i=0;i<c;i++)tmp[i]=dp[i];
    for(auto x:mp)
    {
        ll a=x.first.first,b=x.first.second;
        for(ll i=1;i<=x.second;i++)
        {
            ans=ans*(a+b)%mod;
            tmp2[0]=tmp[0]*b%mod;
            for(ll i=1;i<=c;i++)
            {
                tmp2[i]=(tmp[i-1]*a+tmp[i]*b)%mod;
            }
            for(ll i=0;i<=c;i++)tmp[i]=tmp2[i];
        }
    }
    for(ll i=0;i<c;i++)ans=(ans-tmp[i]%mod+mod)%mod;
    return (ans%mod+mod)%mod;
}

int main()
{
    scanf("%lld%lld",&n,&c);
    for(ll i=1;i<=n;i++)scanf("%lld",&a[i]);
    for(ll i=1;i<=n;i++)scanf("%lld",&b[i]);
    dp[0]=1;
    for(ll i=1;i<=n;i++)add(a[i],b[i]);
    ll q;
    scanf("%lld",&q);
    while(q--)
    {
        ll id,aa,bb;
        scanf("%lld%lld%lld",&id,&aa,&bb);
        del(a[id],b[id]);
        a[id]=aa;b[id]=bb;
        add(a[id],b[id]);
```

```
        printf("%lld\n",getAns());
    }
    return 0;
}
```

--yjw

线段树做法:

　如果遇到过线段树上dp的题目，这道题就很简单。令$dp[now][i]$ 为now这个节点包含的区间中有i个人买了有颜色的画的方案数，lson，rson分别是左、右节点的下标，那么转移式就是：

$$dp[now][i] = \sum_{0 \le j \le i} dp[lson][j] \times dp[rson][i-j]$$

特别的，对于叶子节点，也就是只包含一个人的节点now:
$dp[now][0] = b_i, dp[now][1] = a_i, dp[now][\dots] = 0$

那么每次修改的时候就是把这个节点修改一下，然后重新算一下他祖先节点的答案就行了，每个节点花$O(C^2)$的时间转移，总共修改$O(M \log N)$次，最终复杂度$O(NC^2 + MC^2 \log N)$,勉强能过

```
#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 = 1e6 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD =10007;
inline ll qpow(ll a, ll b){return b?((b&1)?
a*qpow(a*a%MOD,b>>1)%MOD:qpow(a*a%MOD,b>>1))%MOD:1;}
inline ll qpow(ll a, ll b,ll c){return b?((b&1)?
a*qpow(a*a%c,b>>1)%c:qpow(a*a%c,b>>1)) %c:1;}
int C;
struct SegTree
{
#define nd snode[now]
#define lson (now<<1)
#define rson (now<<1|1)
#define lc snode[lson]
#define rc snode[rson]
#define mid (nd.l+nd.r>>1)
#define MAXS 800005
    struct SegNode
    {
        int l,r;
        int dp[21];
    }snode[MAXN];
    void pushup(int now){
        for(int i=0;i<=C;i++){
            nd.dp[i]=0;
            for(int j=0;j<=i;j++)nd.dp[i]=(nd.dp[i]+lc.dp[j]*rc.dp[i-
j]%MOD)%MOD;
        }
    }
    void build(int now,int l,int r,pii *A)
```

```
    {
        nd.l=l;nd.r=r;
        if(l==r)
        {
            nd.dp[0]=A[l].first;
            nd.dp[1]=A[r].second;
            return;
        }
        build(lson,l,mid,A);
        build(rson,mid+1,r,A);
        pushup(now);
    }
    void update(int now,int x,pii v)
    {
        if(nd.l==nd.r){
            nd.dp[0]=v.first;
            nd.dp[1]=v.second;return;
        }
        if(x<=mid)update(lson,x,v);
        else update(rson,x,v);
        pushup(now);
    }
}ST;
pii A[MAXN];
void work()
{
    int n=read(),m;C=read();C--;
    for(int i=1;i<=n;i++)A[i].second=read()%MOD;
    ll tot=1;
    for(int i=1;i<=n;i++)A[i].first=read()%MOD,tot=tot*
(A[i].first+A[i].second)%MOD;
    ST.build(1,1,n,A);m=read();
    for(int cas=1;cas<=m;cas++)
    {
        int pos=read(),a=read()%MOD,b=read()%MOD;
        tot=tot*qpow(A[pos].first+A[pos].second,MOD-2)%MOD;
        ST.update(1,pos,{b,a});
        A[pos]={b,a};
        tot=tot*(A[pos].first+A[pos].second)%MOD;
        ll tmp=0;
        for(int j=0;j<=C;j++)
        {
            tmp=(tmp+ST.snode[1].dp[j])%MOD;
        }
        printf("%lld\n",(tot-tmp+MOD)%MOD);
    }
}
int main(){
        work();
}
```

--ztc

H.NEKAMELEONI

题意:

给你n个数，每个数的值不超过k，接下来有m个询问，有两种操作：
1 x v表示将位置x的值变成v
2 输出包含1~k的所有值的区间

题解：

这个一看就是尺取，但是尺取的时间复杂度太大，然后又有单点更新，然后的话又可以将状态合并，可以用线段树优化。
线段树求答案的时候肯定是左子树的后缀加上右子树的前缀来搞，那么我们就需要维护前后缀上每个值的位置，但是如果单个值去维护的话，检查又要花掉时间。此时发现k=50，那么可以用longlong来维护所有的状态，这里不是说2^{50} 种状态都放到线段树里，而是前缀中包含50个数的最多50个状态。然后后缀也是。
线段树合并的时候，先将左子树的前缀赋给这个点，然后再枚举右子树的所有状态，看是否要进行合并，因为左子树的前缀一定是比右子树的前缀更优的，那么只有右子树有了左子树没有的值的时候才进行合并。后缀同样。
最后进行尺取，尺取肯定是左子树的后缀和右子树的前缀进行尺取。

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int N=1e5+5,M=55;
struct node{
    ll s;
    int p;
}pre[N*4][M],suf[N*4][M];
int ans[N*4],num[N*4],a[N];
ll mx;

void push_up(int root){
    ans[root]=min(ans[root<<1],ans[root<<1|1]);

    for(int i=1;i<=num[root<<1];i++)
        pre[root][i]=pre[root<<1][i];
    int top=num[root<<1];
    for(int i=1;i<=num[root<<1|1];i++)
        if((pre[root][top].s&pre[root<<1|1][i].s)!=pre[root<<1|1][i].s)
            pre[root][++top]={pre[root][top-1].s|pre[root<<1|1]
[i].s,pre[root<<1|1][i].p};

    num[root]=top;
    for(int i=1;i<=num[root<<1|1];i++)
        suf[root][i]=suf[root<<1|1][i];
    top=num[root<<1|1];
    for(int i=1;i<=num[root<<1];i++)
        if((suf[root][top].s&suf[root<<1][i].s)!=suf[root<<1][i].s)
            suf[root][++top]={suf[root][top-1].s|suf[root<<1][i].s,suf[root<<1]
[i].p};

    int p=1;
    for(int i=num[root<<1];i;i--){
        while(p<=num[root<<1|1]&&(suf[root<<1][i].s|pre[root<<1|1][p].s)<mx)p++;
        if(p<=num[root<<1|1])
            ans[root]=min(ans[root],pre[root<<1|1][p].p-suf[root<<1][i].p+1);
    }
}
void build(int l,int r,int root){
```

```
    if(l==r){
        pre[root][1]=suf[root][1]={1ll<<(a[l]-1),l};
        num[root]=1;
        ans[root]=1e9;
        return ;
    }
    int mid=l+r>>1;
    build(l,mid,root<<1);
    build(mid+1,r,root<<1|1);
    push_up(root);
}
void update(int l,int r,int root,int p,int v){
    if(l==r){
        pre[root][1]=suf[root][1]={1ll<<(v-1),l};
        return ;
    }
    int mid=l+r>>1;
    if(mid>=p)
        update(l,mid,root<<1,p,v);
    else
        update(mid+1,r,root<<1|1,p,v);
    push_up(root);
}
int main()
{
    int n,k,m;
    scanf("%d%d%d",&n,&k,&m);
    mx=(1ll<<k)-1;
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    build(1,n,1);
    while(m--){
        int op;
        scanf("%d",&op);
        if(op==1){
            int v,p;
            scanf("%d%d",&p,&v);
            update(1,n,1,p,v);
        }
        else
            printf("%d\n",ans[1]<=n?ans[1]:-1);
    }
    return 0;
}
```

--yf

J. DEATHSTAR

题意：

给了个$N \times N$ 的矩阵M,要你构造一个长度为N的数组A，满足$A_i \& A_j = M_{i,j}$

题解：

对于$M_{i,j}$的每一个二进制位，如果这一位为1，就意味着A_i和A_j的这一位必须都是1，令$b_i = M_{i,1}|M_{i,2}|\ldots|M_{i,N}$,就是$M$第$i$行的值都按位或，如果$b_i$的某一位为1，那么$A_i$的这一位必须是1，否则可以为0，也可以是1（但可能会导致答案错误），题目只需要输出任意解即可，于是b数组就是答案

```
#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;
const int INF = 0x3f3f3f3f, MAXN =1e3 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1e9+7;
int M[MAXN][MAXN];
int a[MAXN];
void work()
{
    int n;scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            int x;scanf("%d",&x);
            a[i]|=x;a[j]|=x;
        }
    }
    for(int i=1;i<=n;i++)printf("%d ",a[i]);
}
int main(){
    work();
}
```

--ztc

K.VUDU

题解:

一个数组，问有多少个子区间，他的平均数大于等于P

思路:

如果暴力枚举左右端点，用sum数组表示前缀和，那么答案就是$\sum_{i=0}^{n}\sum_{j=i+1}^{n}[\frac{sum[j]-sum[i]}{j-i} \geq P]$，($[一个表达式]$表示如果表达式为真，值为1，否则是0)，将里面的表达式化简一下，就变成了$sum[j] - P \cdot j \geq sum[i] - P \cdot i$，然后令$val[i] = sum[i] - P \cdot i$，那么最终的式子就是:

$$\sum_{i=0}^{n}\sum_{j=i+1}^{n}[val[i] \leq val[j]]$$

实际上就是val数组中每个数前面有多少个数小于等于自己，也就是有多少个“正序对”，可以用树状数组来求

```
#pragma comment(linker, "/STACK:142400000,142400000")
#include<bits/stdc++.h>
```

```cpp
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 =2e6 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1e9+7;
ll a[MAXN],sum[MAXN],tree[MAXN];
map<ll,int>mp;
int lowbit(int x){return x&-x;}
int cntS=0;
void add(int x,int val){
    while(x<=cntS)tree[x]+=val,x+=lowbit(x);
}
ll ask(int x){
    ll ret=0;
    while(x)ret+=tree[x],x-=lowbit(x);
    return ret;
}
void work()
{
    int n;scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%lld",a+i);
        sum[i]=sum[i-1]+a[i];
    }
    ll P;scanf("%lld",&P);
    for(int i=0;i<=n;i++)mp[sum[i]-P*i]=0;
    for(auto &x:mp)x.second=++cntS;
    ll ans=0;
    for(int i=0;i<=n;i++)
    {
        ans+=ask(mp[sum[i]-P*i]);
        add(mp[sum[i]-P*i],1);
    }
    printf("%lld\n",ans);
}
int main(){
    work();
}
```

--ztc

L.DOMINO

题目大意

有一个$n*n$的网格,每个格子里有一个非负的数,你有K张$1*2$的多米诺骨牌,需要将它们覆盖到网格上,问怎样覆盖使得露出的数最小

骨牌不能重叠也不能越界.$n <= 2000, K <= 8$

思路

让露出的数最小就是让盖住的数最大

问题转化为求最大覆盖

将每张牌盖着两个数视为一个匹配,匹配的权值为两个数的和

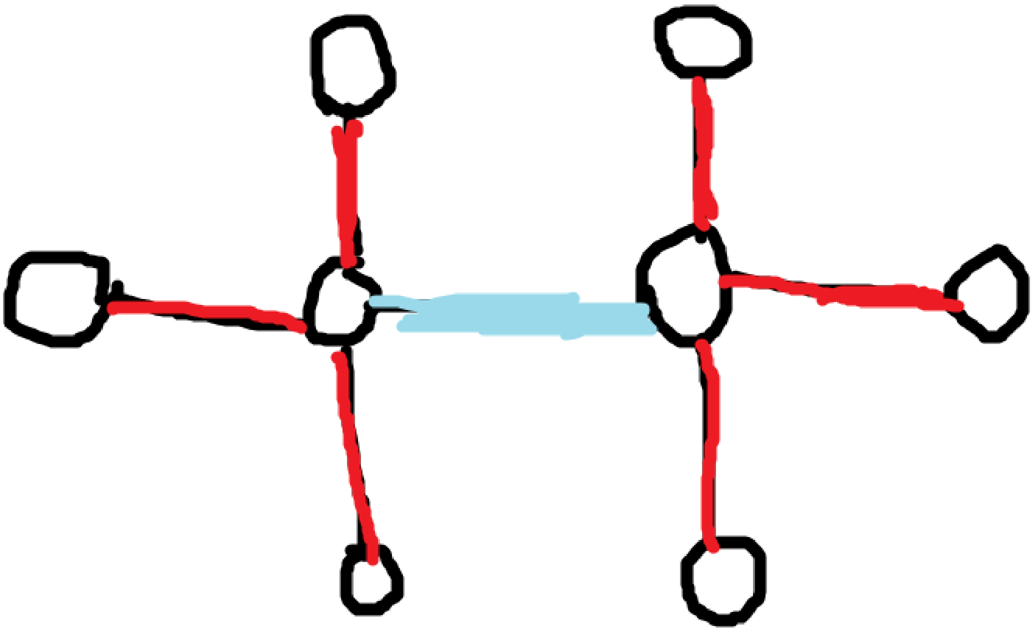
由于不能重叠,每个数最多只能在一个匹配中

每个点都和周围四个点相邻,那么n^2个点就差不多有$4n^2$个匹配

处理出$4n^2$个匹配后,问题转化为求最大的合法的K个匹配的和

因为K比较小,取的匹配不会太多,所以只要考虑权值最大的那些配对就可以了,权值小的肯定用不上

那么具体应该考虑最大的几个呢



如上图所示,如果取了蓝色的那个匹配,周围6个红色的匹配就都不能取了,相当于这一个匹配占了7个匹配的空间

我们要取K个合法匹配,所以理论上考虑前$7K$个匹配肯定是够的(其实好像$7K-6$就够了,~~当然只是好像~~)

因为$K <= 8$,所以考虑前56个匹配就可以了

用一个优先队列维护前$7K$大的匹配即可(如果把所有匹配存下来排序可能会MLE)

现在要做的就是56个匹配中取K个,并且要合法,权值还要尽量大

可能有人会想到最大权匹配,但是K的限制有点难处理

这种匹配的题一般可以用网络流/费用流解决

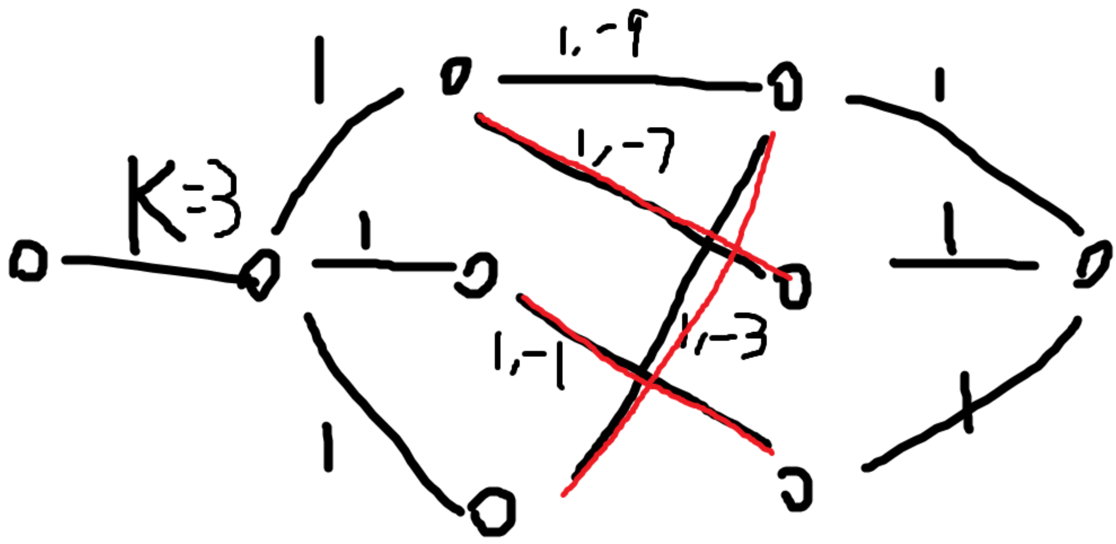
因为网格图是二分图,染色后让黑点与源点连边,白点与汇点连边,容量都为1,代表每个点最多在一个匹配中

多开一个真源点与源点连一条容量为K的边,代表最多取K个匹配

对于每个匹配,将匹配对应的黑点向白点连一条容量为1,费用为匹配的权值相反数的边

代表每个匹配最多取一次,如果取了就能得到相应的权值(取相反数是因为要跑最小费用最大流)

连完边后,图可能长这样



比如$K=3$的话,那三条红色的就是最优方案使用的三个匹配,最小费用是-11,权值和是11

每个匹配有没有用上可以通过流量来判断,所以输出方案也是可以的

连完之后跑一遍费用流,最小费用+网格中所有数的和就是答案(别忘了要求的是露出的数)

代码很丑,仅供参考,而且$spfa$这个假算法可能会出各种问题,不建议拿来当模板(~~当然如果有负权还是要靠它~~)

连边的时候有一点仓促,处理得不怎么好,建议自己连(当然我只是建议)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
const ll inf = (1LL<<60);
const int MOD = 998244353;
const int N = 100005;
ll power(const ll & x,const ll & mi)
{
    ll s1=1LL,s2=x,m=mi;
    while (m)
    {
        if (m&1) s1=s1*s2%MOD;
        s2=s2*s2%MOD;
        m>>=1;
    }
    return s1;
}
inline int read()
{
    char ch=getchar();
    int x=0;
    while (ch<'0' || ch>'9') ch=getchar();
    while ('0'<=ch && ch<='9') x=(x<<3)+(x<<1)+ch-'0',ch=getchar();
    return x;
}
struct Node_{
    int p,x,y,t;
```

```
};
bool operator > (Node_ x,Node_ y){return x.t<y.t;}
bool operator < (Node_ x,Node_ y){return x.t>y.t;}
std::priority_queue <Node_> heap;
int a[3005][3005];
int n,K;
int id(int x,int y){return (x-1)*n+y;}
void solve()
{
    return;
}
int s1[1005],t1,s2[1005],t2;
int S,T,SS;
int num,from[N+N],to[N+N],Next[N+N],Head[N],len[N+N],cost[N+N];
int que[N],inq[N];
int pre[N];
ll dis[N];
ll ans,flow;
int spfa()
{
    for (int i=1;i<=SS;++i) dis[i]=inf;
    dis[S]=0LL;
    int head=0,tail=1;
    que[1]=S;
    inq[S]=1;
    pre[S]=-1;
    while (head<tail)
    {
        int x=que[(++head)%N];
        for (int i=Head[x];i;i=Next[i])
        if (len[i] && dis[x]+cost[i]<dis[to[i]])
        {
            dis[to[i]]=dis[x]+cost[i];
            pre[to[i]]=i;
            if (!inq[to[i]])
            {
                inq[to[i]]=1;
                if (dis[to[i]]<dis[que[head%N]]) que[(head--)%N]=to[i];
                else que[(++tail)%N]=to[i];
            }
        }
        inq[x]=0;
    }
    return dis[T]!=inf;
}
void mcf()
{
    ll x=inf;
    for (int i=pre[T];i!=-1;i=pre[from[i]]) x=min(x,(ll)(len[i]));
    flow+=x;
    for (int i=pre[T];i!=-1;i=pre[from[i]])
        ans+=(ll)(x)*cost[i],len[i]-=x,len[i^1]+=x;
}
inline void add(int x,int y,int t,int c)
{
    from[num]=x;
    to[num]=y;
    Next[num]=Head[x];
```

```
    Head[x]=num;
    len[num]=t;
    cost[num]=c;
    ++num;
}
int flag[100005];
map <int,int> mp;int mpcnt=0;
int main()
{
    ll sum=0;
    num=2;
    int tot=0;
    cin>>n>>K;
    for (int i=1;i<=n;++i)
    {
        for (int j=1;j<=n;++j)
        {
            a[i][j]=read();
            sum+=a[i][j];
            if (j>1) heap.push((Node_){0,i,j-1,a[i][j-1]+a[i][j]}),++tot;
            if (i>1) heap.push((Node_){1,i-1,j,a[i-1][j]+a[i][j]}),++tot;
            while (tot>K*7) heap.pop(),--tot;
        }
    }
    while (!heap.empty())
    {
        Node_ t=heap.top();heap.pop();
        int x1,y1,x2,y2;
        x1=t.x;y1=t.y;
        if (t.p==0) x2=t.x,y2=t.y+1;else x2=t.x+1,y2=t.y;
        if ((x1+y1)&1) swap(x1,x2),swap(y1,y2);
        int id1=id(x1,y1),id2=id(x2,y2);
        if (!mp[id1]) mp[id1]=++mpcnt;
        if (!mp[id2]) mp[id2]=++mpcnt;
        s1[++t1]=mp[id1];
        s2[++t2]=mp[id2];
        add(s1[t1],s2[t2],1,-t.t);
        add(s2[t2],s1[t1],0,t.t);
    }
    S=mpcnt+1,T=S+1,SS=T+1;
    add(S,SS,K,0);
    add(SS,S,0,0);
    while (t1)
    {
        int tt=s1[t1--];
        if (flag[tt]) continue;
        flag[tt]=1;
        add(SS,tt,1,0);
        add(tt,SS,0,0);
    }
    while (t2)
    {
        int tt=s2[t2--];
        if (flag[tt]) continue;
        flag[tt]=1;
        add(tt,T,1,0);
        add(T,tt,0,0);
    }
```

```
    ans=flow=0;
    while (spfa()) mcf();
    cout<<sum+ans<<endl;
    return 0;
}
```

--yzh