/\*给你一个二分图，N左边的点，M右边的点的，K总共的边数，L，R

要求能否删除一些边使得图中所有点的度都在区间[L,R]

思路

构造一个网络流，这是关于网路流的上下界问题。增加四个点，分别是点超级源点s，超级汇点t,还有两个属于上下界问题的xx,yy节点。

题目这样就转化为了一个有汇源上下界限的可行流了。

\*/

const int maxn = 4050;

const int INF = 0x3f3f3f3f;

const ll mod = 1000000007;

int n, m, k, s, t, S, T;

int l, r, no,

a, b;

int head[maxn], level[maxn];

struct node

{

int to, nxt;

int f;

}e[maxn \* 7];

void add(int a, int b, int w)

{

e[no].to = b, e[no].nxt = head[a], e[no].f = w;

head[a] = no++;

e[no].to = a, e[no].nxt = head[b], e[no].f = 0;

head[b] = no++;

}

bool bfs(int s, int t)

{

memset(level, -1, sizeof(level));

level[s] = 0;

queue<int>q;

q.push(s);

while(!q.empty())

{

int u = q.front();

q.pop();

for(int i = head[u];i != -1;i = e[i].nxt)

{

int v = e[i].to;

if(level[v] == -1 && e[i].f > 0)

{

level[v] = level[u] + 1;

q.push(v);

if(v == t) return true;

}

}

}

return level[t] != -1;

}

int dfs(int s, int t, int f)

{

if(s == t || f == 0) return f;

int tmp, ret = 0;

for(int i = head[s];i != -1;i = e[i].nxt)

{

int v = e[i].to;

if(level[v] == level[s] + 1 && e[i].f > 0 && (tmp = dfs(v, t, min(f, e[i].f))))

{

e[i].f -= tmp;

e[i^1].f += tmp;

ret += tmp;

f -= tmp;

if(f == 0) break;

}

}

if(f > 0) level[s] = -1;

return ret;

}

int dinic(int s, int t)

{

int res = 0;

while(bfs(s, t))

{

int tmp = dfs(s, t, INF);

res += tmp;

}

return res;

}

int main()

{

int kase = 0;

while(scanf("%d%d%d", &n, &m, &k) != EOF)

{

no = 0;

memset(head, -1, sizeof(head));

s = n+m + 1, t = n+m + 2;

S = 0, T = n+m + 3;

scanf("%d%d", &l ,&r);

for(int i = 0;i < k;i++)

{

scanf("%d%d", &a, &b);

add(a, b + n, 1);

}

add(t, s, INF);

for(int i = 1;i <= n;i++)

{

add(s, i, r - l);

add(S, i, l), add(s, T, l);

}

for(int i = 1;i <= m;i++)

{

add(i + n, t, r - l);

add(i + n, T, l), add(S, t, l);

}

printf("Case %d: ", ++kase);

int ans = dinic(S, T);

if(ans == (n + m)\*l) printf("Yes\n");

else printf("No\n");

}

return 0;

}

/\*

假设一个试题库中有n道试题。每道试题都标明了所属类别。同一道题可能有多个类别属性。现要从题库中抽取m 道题组成试卷。并要求试卷包含指定类型的试题。试设计一个满足要求的组卷算法。

«编程任务：

对于给定的组卷要求，计算满足要求的组卷方案。

输入输出格式

输入格式：

第1行有2个正整数k和n (2 <=k<= 20, k<=n<= 1000)

k 表示题库中试题类型总数，n 表示题库中试题总数。第2 行有k 个正整数，第i 个正整数表示要选出的类型i的题数。这k个数相加就是要选出的总题数m。接下来的n行给出了题库中每个试题的类型信息。每行的第1 个正整数p表明该题可以属于p类，接着的p个数是该题所属的类型号。

输出格式：

第i 行输出 “i：”后接类型i的题号。如果有多个满足要求的方案，只要输出1个方案。如果问题无解，则输出“No Solution!”。

3 15

3 3 4

2 1 2

1 3

1 3

1 3

1 3

3 1 2 3

2 2 3

2 1 3

1 2

1 2

2 1 2

2 1 3

2 1 2

1 1

3 1 2 3

输出样例#1：

1: 1 6 8

2: 7 9 10

3: 2 3 4 5

解析

网络流24题之一orz。

二分图多重匹配。

每道题跟T连容量为1的边。

每类题跟S连容量为需要该类题数量的边。

每道题跟它所在的题的类别连容量为1的边。

跑最大流，如果结果小于题目总数，则无解。

否则，对每条连接题的类别和题目的边检查，

若满流，输出题目的编号即可。

\*/#include<iostream>

#include<cstdlib>

#include<cstring>

#include<cstdio>

#include<queue>

#define INF 2147483647

const int MAXN=1e7;

using namespace std;

struct Edge

{

int from,next,to,c,f;

}edge[MAXN];

int head[MAXN],cnt,level[MAXN],Num\_k[MAXN],n,m,p,s,t,k;

inline int read()

{

int x=0;

char ch=getchar();

while(ch<'0'||ch>'9') ch=getchar();

while(ch>='0'&&ch<='9') x=x\*10+ch-'0',ch=getchar();

return x;

}

inline void Add\_Edge(int u,int v,int cc,int ff)

{

edge[cnt]=(Edge){u,head[u],v,cc,ff};

head[u]=cnt++;

}

bool Bfs(int s,int t)

{

memset(level,0,sizeof(level));

level[s]=1;

queue<int> que;

que.push(s);

while(!que.empty())

{

int c=que.front();

que.pop();

for(int i=head[c];i!=-1;i=edge[i].next)

{

int v=edge[i].to;

if(level[v]==0&&edge[i].c>edge[i].f)

{

level[v]=level[c]+1;

que.push(v);

}

}

}

if(level[t]==0) return 0;

else return 1;

}

int Dfs(int u,int f)

{

if(u==t) return f;

int adf=0;

for(int i=head[u];i!=-1&&adf<f;i=edge[i].next)

{

int v=edge[i].to;

if(level[v]==level[u]+1&&edge[i].c>edge[i].f)

{

int temp=Dfs(v,min(f-adf,edge[i].c-edge[i].f));

edge[i].f+=temp;

edge[i^1].f-=temp;

adf+=temp;

}

}

return adf;

}

int Dinic(int s,int t)

{

int sumflow=0;

while(Bfs(s,t))

{

sumflow+=Dfs(s,INF);

}

return sumflow;

}

int main()

{

memset(head,-1,sizeof(head));

k=read(),n=read();

for(int i=1;i<=k;i++)

{

Num\_k[i]=read();

m+=Num\_k[i];

}

s=0,t=n+m+1;

for(int i=1;i<=k;i++)

{

Add\_Edge(i+n,t,Num\_k[i],0);

Add\_Edge(t,i+n,0,0);

}

for(int i=1;i<=n;i++)

{

Add\_Edge(s,i,1,0);

Add\_Edge(i,s,0,0);

p=read();

for(int j=1;j<=p;j++)

{

int x;

scanf("%d",&x);

Add\_Edge(i,x+n,1,0);

Add\_Edge(x+n,i,0,0);

}

}

int now=Dinic(s,t);

if(!now)

{

printf("No Solution!");

return 0;

}

else

{

for(int i=1;i<=k;i++)

{

printf("%d: ",i);

for(int j=0;j<=cnt;j++)

{

// cout << "edge[i].from： " <<edge[i].from << endl;

if(edge[j].to==i+n // 16 17 18 == j.to

&&edge[j].f>0 // j.f 存在流

&&edge[i].from>s) // i.from > 0 大概没必要吧？

{

printf("%d ",edge[j].from); // from

}

}

printf("\n");

}

}

return 0;

}

/\*有n个柱子，编号为1，2…….的小环，要将它们依次套在环上，要求直接接触的两小球的和为完全平方数，那么最多可以套几个小球。

做法

没错，就是这样，是不是马上想到了暴力dfs！！！n只有55，但是你无法确定球的个数，所以dfs暴枚肯定会很慢（wjh：不然怎么出在网络流24题里）所以，我们要往网络流方面去想，而我们又要运用到网络流的反悔性质，所以很明显就会想到拆点.

我们每一次把一个球拆成两个点，前面的点连（源点），后面的点连（汇点），权值都为1，然后中间的怎么连呢？

很明显我们回记起来还有一个条件没有用上，那就是同一个柱子中，两个相邻的球的编号之和为完全平方数，所以如果a+b为完全平方数，那么我们就把a的前面那个店和b后面那个点连接起来，权值为一（注意：这里可能会理解错误，因为有些人可能想问x后面可以跟y，y后面可以跟z，那么是不是要把y的后面那个点与y前面那个点连起来，这样跑出来的结果才会对。。错！！！因为这个网络流中的流量只有两种状态，无解和有解，无解时跑出来的结果为0，有解时跑出来的结果>0，这样才能实现一次装一个柱子，要不你一次装多个柱子，那你怎么加点。。对吧!？）

我们每次加一个点，然后跑一次网络流，如果有解，那么就继续加点，否则，就将它压在第二个柱子的最底下，也就是里面的head[]数组。

input = 4

11

1 3 6 10

2 7 9

4 5 11

8

\*/

int n;//共有柱子

int spend=0;//开始用第几根柱子

int tot\_ball=0;

int begin,end;

int head[60];//第i根柱子在柱底的是第几号球

struct edge{int y,next,c;};

edge s[400010];

int first[200010];

int match[50010];

int len=1;

int h[50010];

int f[100010];

void ins(int x,int y,int c)

{

len++;

s[len].y=y;s[len].c=c;

s[len].next=first[x];first[x]=len;

}

int min(int x,int y){

return x<y?x:y;

}

bool bfs()

{

int st=1,ed=2;

f[st]=begin;

memset(h,0,sizeof(h));

h[begin]=1;

while(st!=ed)

{

int x=f[st];

for(int i=first[x];i!=0;i=s[i].next)

{

int y=s[i].y;

if(h[y]==0 && s[i].c>0)

{

h[y]=h[x]+1;

f[ed]=y;

ed++;

if(ed==100000) ed=1;

}

}

st++;

if(st==100000) st=1;

}

return h[end];

}

int dfs(int x,int t)

{

if(x==end) return t;

int tot=0,flow=0;

for(int i=first[x];i!=0;i=s[i].next)

{

if(tot==t) return t;

int y=s[i].y;

if(s[i].c>0 && h[y]==h[x]+1)

{

flow=dfs(y,min(s[i].c,t-tot));

tot+=flow;s[i].c-=flow;s[i^1].c+=flow;

if(flow!=0 && y!=begin && y!=end && x%2==0)

match[x>>1]=y>>1;

}

}

if(tot==0) h[x]=0;

return tot;

}

int dinic()

{

int tot=0;

int dx;

while(bfs())

{

dx=dfs(begin,1e9);

while(dx!=0)

{

tot+=dx;

dx=dfs(begin,1e9);

}

}

return tot;

}

int main()

{

scanf("%d",&n);

begin=1;end=50000;

while(spend<=n)

{

tot\_ball++;

ins(begin,tot\_ball<<1,1);ins(tot\_ball<<1,begin,0);

ins((tot\_ball<<1)|1,end,1);ins(end,(tot\_ball<<1)|1,0);

for(int i=sqrt(tot\_ball)+1;i\*i<2\*tot\_ball;i++)

{

ins((i\*i-tot\_ball)<<1,(tot\_ball<<1)|1,1),ins((tot\_ball<<1)|1,(i\*i-tot\_ball)<<1,0);

}

int k=dinic();

if(k==0) {

//因为没有流量流到汇点（也就是说，这个球只能放到下一个柱子了）；

//所以我们把它设为下一个柱子的开头

spend++;

cout << spend << ' ' << tot\_ball << endl;

head[spend]=tot\_ball;

}

}

printf("%d\n",tot\_ball-1);

bool visit[50010];//第i号球是否用过

memset(visit,false,sizeof(visit));

for(int i=1;i<=n;i++)

{

if(visit[head[i]]) continue;

int now=head[i];

while(now!=0 && visit[now]==false)

{

printf("%d ",now);

visit[now]=true;

now=match[now];

}

printf("\n");

}

}

/\* http://www.itdaan.com/blog/2018/02/04/5a3d711eaf666eb5fdae5098802ee58a.html

给定正整数序列x1,...,xn 。

（1）计算其最长不下降子序列的长度s。

（2）计算从给定的序列中最多可取出多少个长度为s的不下降子序列。

（3）如果允许在取出的序列中多次使用x1和xn，则从给定序列中最多可取出多少个长度为s的不下降子序列。

思路与想法

看到这道题，我想到的只是推DP方程，只要保持n的平方就可以完成了（但是不行，最少n的三次方）。

网络流中的最大流可以解决这个东西，只要保证从原点流到汇点的流量满足s长度的最长非下降就可以了。怎么保证呢？

解法

我们把第一问的答案用n的平方（或者大佬可以用单调队列来维护这个max）。f[i]表示以i结尾的最长非下降序列长度是多少。(拆点，把 i 拆成 i 和 i’)

1.当f[i]==1时，那么表示i不能作为一个序列的结尾，只能作为开头，所以从begin（源点）到i连一条流量为1的边。

2.当f[i]==mmax时,表示i能作为一个最长非下降序列的结尾，所以从i’到end（汇点）连一条流量为1的边。

3.当a[i]<a[j] and(&&) f[j]=f[i]+1 的时候，那么从i到j‘连一条流量为1的边。

4.为了能使流量能连续流通，我们从i’到i连一条流量为1的边。

（注意，当mmax==1 时，要一定要特判，从i到i’连一条流量为1的边；因为不这样子的话，i到end就没有边可以通，一开始我就卡了两个点。。。）

第三个问，就干脆将begin到1点连一条INF的边，n’点到end连一条INF的边。

想看具体图片吗？。。

像下面这组样例

4

3 6 2 5

\*/

int n;

int a[510];

int f[510];

struct edge{int y,next,c;};

edge s[600010];

int bg,endd;

int first[10010];

int len=1;

int q[10010];

int h[10010];

bool tf[10010];

int mmax=0;

int st,ed;

int mmin(int x,int y)

{

return x<y?x:y;

}

void ins(int x,int y,int c)

{

len++;

s[len].y=y;s[len].c=c;

s[len].next=first[x];first[x]=len;

len++;

s[len].y=x;s[len].c=0;

s[len].next=first[y];first[y]=len;

}

void build\_edge(int x)

{

memset(first,0,sizeof(first));

len=1;

for(int i=1;i<=n;i++)

if(f[i]==1)

if(i==1)

ins(bg,i,x);

else

ins(bg,i,1);

for(int i=1;i<=n;i++)

for(int j=i;j<=n;j++)

if(a[i]<=a[j] && (f[j]==f[i]+1 || mmax==1))

ins(i,n+j,1);

for(int i=1;i<=n;i++) ins(i+n,i,1);

for(int i=1;i<=n;i++)

if(f[i]==mmax)

if(i==n)

ins(i+n,endd,x);

else

ins(i+n,endd,1);

}

bool bfs()

{

st=1,ed=2;

memset(h,0,sizeof(h));

h[bg]=1;

q[st]=bg;

while(st!=ed)

{

int x=q[st];

for(int i=first[x];i!=0;i=s[i].next)

{

int y=s[i].y;

if(h[y]==0 && s[i].c>0)

{

h[y]=h[x]+1;

q[ed]=y;

ed++;

if(ed==n+n+n) ed=1;

}

}

st++;

if(st==n+n+n) st=1;

}

if(h[endd]!=0) return true;

return false;

}

int dfs(int x,int t)

{

if(x==endd) return t;

int tot=0;

for(int i=first[x];i!=0;i=s[i].next)

{

if(t==tot) return t;

int y=s[i].y;

if(h[y]==h[x]+1)

{

int get=dfs(y,mmin(t-tot,s[i].c));

tot+=get;s[i].c-=get;s[i^1].c+=get;

}

}

if(tot==0) h[x]=0;

return tot;

}

int max\_flow()

{

int flow=0,now=0;

while(bfs())

{

now=dfs(bg,1e9);

while(now!=0)

{

flow+=now;

now=dfs(bg,1e9);

}

}

return flow;

}

int main()

{

scanf("%d",&n);

for(int i=1;i<=n;i++)

scanf("%d",&a[i]);

for(int i=1;i<=n;i++)

{

f[i]=1;

for(int j=1;j<i;j++)

if(a[j]<=a[i] && f[j]+1>f[i]) f[i]=f[j]+1;

if(f[i]>mmax) mmax=f[i];

}

printf("%d\n",mmax);

bg=0;

endd=2\*n+1;

build\_edge(1);

printf("%d\n",max\_flow());

build\_edge(1e9);

printf("%d\n",max\_flow());

}

/\*在一个有 m\*n 个方格的棋盘中，每个方格中有一个正整数。现要从方格中取数，使任意 2 个数所在方格没有公共边，且取出的数的总和最大。试设计一个满足要求的取数算法。对于给定的方格棋盘，按照取数要求编程找出总和最大的数。

输入输出格式

输入格式：

第 1 行有 2 个正整数 m 和 n，分别表示棋盘的行数和列数。接下来的 m 行，每行有 n 个正整数，表示棋盘方格中的数。

输出格式：

程序运行结束时，将取数的最大总和输出

输入输出样例

输入样例：

3 3

1 2 3

3 2 3

2 3 1

输出样例：

11

说明

m,n<=100

这是一道最小割题目，答案=全局和-最小割，由于最小割等于最大流，所以我们只要求出全局和-最大流即可。

我们先对棋盘进行染色，把行加列为偶数的格子染成黑色，其余为白色，有些人也许会想，ans=max(Σ

白,Σ

黑)，但为了让一个点与它不相邻且颜色不相同的点连边，我们要用到最小割。

构图

1. 建立超级源点与超级汇点

2. 从源点向每一个白色格子建一条边，流量为这个格子的值，从每一个黑色格子向汇点建一条边，流量也为这个格子的值

3. 从每一个白色的点向与它相临的黑色的点建边，流量为INF，这些边是要割掉的

然后跑最大流，由于最大流等于最小割，用所有格子的值的和减去最大流即为答案。

\*/

#include<cstdio>

#include<cstdlib>

#include<cstring>

const int INF=1e9;

struct node{int x,y,c,next,other;}a[200010];

int first[2510],last[2510],h[2510],q[2510];

int n,m,st,ed,len=0,ans=0;

int fx[4]={0,1,0,-1};

int fy[4]={1,0,-1,0};

void ins(int x,int y,int c)

{

int k1,k2;

len++;k1=len;

a[len].x=x;a[len].y=y;a[len].c=c;

a[len].next=first[x];first[x]=len;

len++;k2=len;

a[len].x=y;a[len].y=x;a[len].c=0;

a[len].next=first[y];first[y]=len;

a[k1].other=k2;a[k2].other=k1;

}

bool bfs()

{

int head=1,tail=2;q[1]=st;

memset(h,0,sizeof(h));h[st]=1;

while(head!=tail)

{

int x=q[head];

for(int i=first[x];i;i=a[i].next)

{

int y=a[i].y;

if(h[y]==0 && a[i].c>0)

{

q[tail++]=y;

h[y]=h[x]+1;

}

}

head++;

}

if(h[ed]==0) return false;

return true;

}

inline int mymin(int x,int y)

{

return x<y?x:y;

}

inline int dfs(int x,int flow)

{

if(x==ed) return flow;

int tt=0,minf=0;

for(int i=first[x];i;i=a[i].next)

{

int y=a[i].y;

if(a[i].c>0 && h[y]==(h[x]+1) && tt<=flow)

{

minf=dfs(y,mymin(flow-tt,a[i].c));

tt+=minf;

a[i].c-=minf;a[a[i].other].c+=minf;

}

}

if(tt==0) h[x]=0;

return tt;

}

inline int dinic()

{

int s=0;

while(bfs()) s+=dfs(st,INF);

return s;

}

inline int num(int i,int j){return i\*m+j+1;}

int main()

{

memset(first,0,sizeof(first));

scanf("%d %d",&n,&m);

st=n\*m+1;ed=st+1;

int x;

for(int i=0;i<n;i++)

{

for(int j=0;j<m;j++)

{

scanf("%d",&x);

ans+=x;

if((i+j)&1) ins(st,num(i,j),x);

else ins(num(i,j),ed,x);

}

}

for(int i=0;i<n;i++)

{

for(int j=0;j<m;j++)

{

if((i+j)&1)

for(int k=0;k<4;k++)

{

int xx=i+fx[k],yy=j+fy[k];

if(xx<0 || xx>=n || yy<0 || yy>=m) continue;

ins(num(i,j),num(xx,yy),INF);

}

}

}

int tt=dinic();

printf("%d",ans-tt);

}