/\* 给出一个n\*n的矩阵,每一格有一个非负整数Aij,(Aij <= 1000)现在从(1,1)出发,可以往右或者往下走,最后到达(n,n),每达到一格,把该格子的数取出来,该格子的数就变成0,这样一共走K次,现在要求K次所达到的方格的数的和最大

输入描述 Input Description

第一行两个数n,k（1<=n<=50, 0<=k<=10）

接下来n行,每行n个数,分别表示矩阵的每个格子的数

输出描述 Output Description

一个数,为最大和

样例输入 Sample Input

3 1

1 2 3

0 2 1

1 4 2

样例输出 Sample Output

11

\*/

#include<iostream>

#include<cstdio>

#include<cstring>

#define N 6001

#define M 1000010

#define INF 0x7fffffff

using namespace std;

int n,k,cnt=1;

long long ans;

struct data{int from,to,v,c,next;}e[M];

int dis[N],head[N],q[M],from[N];

bool inq[N];

void insert(int u,int v,int w,int c)

{

cnt++;

e[cnt].from=u;e[cnt].to=v;

e[cnt].v=w;e[cnt].c=c;

e[cnt].next=head[u];head[u]=cnt;

}

void ins(int u,int v,int w,int c)

{insert(u,v,w,c);insert(v,u,0,-c);}

////////////////////////////////////////

bool spfa()

{

int t=0,w=1,i,now;

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

q[0]=0;dis[0]=0;inq[0]=1;

while(t<w)

{

now=q[t];t++;i=head[now];

while(i)

{

if(e[i].v>0&&dis[now]+e[i].c>dis[e[i].to])

{

dis[e[i].to]=dis[now]+e[i].c;

from[e[i].to]=i;

if(!inq[e[i].to])

{q[w]=e[i].to;w++;inq[e[i].to]=1;}

}

i=e[i].next;

}

inq[now]=0;

}

if(dis[6000]==-1)return 0;

return 1;

}

void mincf()

{

int i,sum=INF;

i=from[6000];

while(i)

{

sum=min(sum,e[i].v);

i=from[e[i].from];

}

i=from[6000];

while(i)

{

e[i].v-=sum;

e[i^1].v+=sum;

ans+=sum\*e[i].c;

i=from[e[i].from];

}

}

// 走1次价值x

// 走k次价值0

int main()

{

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

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

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

{

int x;

scanf("%d",&x);

ins((i-1)\*n+j,(i-1)\*n+j+n\*n,1,x); // to : (i-1)\*n+j+n\*n 后面大

ins((i-1)\*n+j,(i-1)\*n+j+n\*n,k,0); // to : (i-1)\*n+j+n\*n 后面大

if(j<n) // 细节

ins((i-1)\*n+j+n\*n,(i-1)\*n+j+1,k,0); // from : (i-1)\*n+j+n\*n 前面大

if(i<n) // 细节

ins((i-1)\*n+j+n\*n,i\*n+j,k,0); // from : (i-1)\*n+j+n\*n 前面大

}

ins(0,1,k,0);

ins(n\*n\*2,6000,k,0);

while(spfa())mincf();

printf("%lld",ans);

return 0;

}

/\*

题目描述 Description

在一块梯形田地上，一群蚯蚓在做收集食物游戏。蚯蚓们把梯形田地上的食物堆积整理如下：

a(1,1) a(1,2)…a(1,m)

a(2,1) a(2,2) a(2,3)…a(2,m) a(2,m+1)

a(3,1) a (3,2) a(3,3)…a(3,m+1) a(3,m+2)

……

a(n,1) a(n,2) a(n,3)… a(n,m+n-1)

它们把食物分成n行，第1行有m堆的食物，每堆的食物量分别是a(1,1),a(1,2),…,a(1,m)；

第2行有m+1堆食物，每堆的食物量分别是a(2,1),a(2,2),…, a(2,m+1)；以下依次有m+2堆、m+3堆、…m+n-1堆食物。

现在蚯蚓们选择了k条蚯蚓来测试它们的合作能力（1≤ k ≤m）。测试法如下：第1条蚯蚓从第1行选择一堆食物，然后往左下或右下爬，并收集1堆食物，例如从a（1,2）只能爬向a(2,2) 或a(2,3)，而不能爬向其它地方。接下来再爬向下一行收集一堆食物，直到第n行收集一堆食物。第1条蚯蚓所收集到的食物量是它在每一行所收集的食物量之和；第2条蚯蚓也从第1行爬到第n行，每行收集一堆食物，爬的方法与第1条蚯蚓相类似，但不能碰到第1条蚯蚓所爬的轨迹；一般地，第i 条蚯蚓从第1行爬到第 n行，每行收集一堆食物，爬的方法与第1条蚯蚓类似，但不能碰到前 I-1 条蚯蚓所爬的轨迹。这k条蚯蚓应该如何合作，才能使它们所收集到的食物总量最多？收集到的食物总量可代表这k条蚯蚓的合作水平。

Ø编程任务：

给定上述梯形m、n和k的值（1≤k≤m≤30;1≤n≤30）以及梯形中每堆食物的量（小于10的非整数），编程计算这k条蚯蚓所能收集到的食物的最多总量。

输入描述 Input Description

输入数据由文件名为INPUT1.\*的文本文件提供，共有n+1行。每行的两个数据之间用一个空格隔开。

●第1行是n、m和k的值。

接下来的n行依次是梯形的每一行的食物量a(i,1)，a(i,2)，…，a(i,m+i-1)，i=1,2,…,n。

输出描述 Output Description

程序运行结束时，在屏幕上输出k蚯蚓条所能收集到的食物的最多总量。

样例输入 Sample Input

3 2 2

1 2

5 0 2

1 10 0 6

样例输出 Sample Output

26

\*/

#include<iostream>

#include<cstdio>

#include<cstring>

#define inf 0x7fffffff

#define X 1501

#define N 3001

#define M 50001

using namespace std;

int n,m,k,ans,cnt=1,tot;

struct data{int from,to,v,c,next;}e[M];

int head[N],inq[N],from[N],dis[N],q[N];

void ins(int u,int v,int w,int c)

{

cnt++;

e[cnt].to=v;e[cnt].from=u;

e[cnt].v=w;e[cnt].c=c;

e[cnt].next=head[u];head[u]=cnt;

}

void insert(int u,int v,int w,int c)

{ins(u,v,w,c);ins(v,u,0,-c);}

////////////////////////////////////////

bool spfa()

{

int t=0,w=1,i,now;

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

q[0]=dis[0]=0;inq[0]=1;

while(t!=w)

{

now=q[t];t++;i=head[now];

if(t==N)t=0;

while(i)

{

if(e[i].v>0&&dis[now]+e[i].c>dis[e[i].to])

{

dis[e[i].to]=dis[now]+e[i].c;

from[e[i].to]=i;

if(!inq[e[i].to])

{q[w++]=e[i].to;if(w==N)w=0;inq[e[i].to]=1;}

}

i=e[i].next;

}

inq[now]=0;

}

if(dis[N-1]==-1)return 0;

return 1;

}

void mcf()

{

int i,x=inf;

i=from[N-1];

while(i)

{

x=min(x,e[i].v);

i=from[e[i].from];

}

i=from[N-1];

while(i)

{

e[i].v-=x;e[i^1].v+=x;

ans+=x\*e[i].c;

i=from[e[i].from];

}

}

// 走1次价值x

// 走1次价值0

// insert( 1 , k ) ---> insert( 1 , 0 )insert( 1 , 0 )insert( 1 , 0 )insert( 1 , 0 )

int main()

{

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

int x;

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

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

{

tot++;

scanf("%d",&x); // #define X 1501

insert(tot,tot+X,1,x); // to : tot+X 后面大

if(i<n) // 细节

{

insert(tot+X,tot+i+m,1,0); // from : tot+X 前面大

insert(tot+X,tot+i+m-1,1,0); // from : tot+X 前面大

}

}

for(int i=1;i<=m;i++)insert(0,i,1,0);

for(int i=1;i<=m+n-1;i++)insert(tot-i+X+1,N-1,1,0);

for(int i=1;i<=k;i++)if(spfa())mcf();else break;

printf("%d",ans);

return 0;

}

/\*

Description

给定一张有向图，每条边都有一个容量C和一个扩容费用W。这里扩容费用是指将容量扩大1所需的费用。求： 1、 在不扩容的情况下，1到N的最大流； 2、 将1到N的最大流增加K所需的最小扩容费用。

Input

输入文件的第一行包含三个整数N,M,K，表示有向图的点数、边数以及所需要增加的流量。 接下来的M行每行包含四个整数u,v,C,W，表示一条从u到v，容量为C，扩容费用为W的边。

Output

输出文件一行包含两个整数，分别表示问题1和问题2的答案。

Sample Input

5 8 2

1 2 5 8

2 5 9 9

5 1 6 2

5 1 1 8

1 2 8 7

2 5 4 9

1 2 1 1

1 4 2 1

Sample Output

13 19

\*/

#include<iostream>

#include<cstdio>

#include<cstring>

#define inf 0x7fffffff

using namespace std;

int n,m,k,cnt=1,ans,head[1001],from[1001],q[1001],h[1001],dis[1001];

bool inq[1001];

struct data{int from,to,v,c,next,t;}e[50001];

void ins(int u,int v,int w,int c)//原图中的所有边费用都是0，但是要把扩容费用记下来

{

cnt++;

e[cnt].to=v;e[cnt].from=u;

e[cnt].v=w;e[cnt].t=c;//暂存扩容费用

e[cnt].next=head[u];head[u]=cnt;

}

void insert(int u,int v,int w,int c)

{ins(u,v,w,c);ins(v,u,0,-c);}

void ins2(int u,int v,int w,int c)

{

cnt++;

e[cnt].to=v;e[cnt].from=u;

e[cnt].v=w;e[cnt].c=c;

e[cnt].next=head[u];head[u]=cnt;

}

void insert2(int u,int v,int w,int c)

{

ins2(u,v,w,c);ins2(v,u,0,-c);}

void build()

{

int t=cnt;

for(int i=2;i<=t;i+=2)

if(i%2==0)

insert2(e[i].from,e[i].to,inf,e[i].t);

/\* 第一问很简单，按数据建图，然后一遍最大流算法即可。

第二问则需要用最小费用最大流算法，主要是建图，那么可以从第一问的残留网络上继续建图，

对残留网络上的每一条边建一条容量是inf∞费用是w的边（反向弧容量为0，费用为-w），

然后建一个超级源点，从超级源向1建一条容量为k，费用为0的边，对这个图进行最小费用最大流算法。

最小费用最大流操作：

1.首先要对于这道题的图来说，有的边需要花费费用，而有的又不用，而不用扩容的边费用为0，需要扩容的边费用为w，容量无限，这就是本题这样建图的原因。

2.对于残留网络进行费用最短路SPFA算法，不用扩容的边一定会选费用为0的边，然后记录路径，找最小容量对可行路进行增流，更新ans

\*/

}

bool bfs()

{

int t=0,w=1,i,now;

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

q[0]=1;h[1]=0;

while(t!=w)

{

now=q[t];t++;if(t==1000)t=0;i=head[now];

while(i)

{

if(e[i].v&&h[e[i].to]==-1)

{

h[e[i].to]=h[now]+1;

q[w++]=e[i].to;if(w==1000)w=0;

}

i=e[i].next;

}

}

if(h[n]==-1)return 0;

return 1;

}

int dfs(int x,int f)

{

if(x==n)return f;

int i=head[x],w,used=0;

while(i)

{

if(e[i].v>0&&h[e[i].to]==h[x]+1)

{

w=f-used;

w=dfs(e[i].to,min(w,e[i].v));

e[i].v-=w;

e[i^1].v+=w;

used+=w;

if(used==f)return f;

}

i=e[i].next;

}

if(!used)h[x]=-1;

return used;

}

void dinic(){while(bfs())ans+=dfs(1,inf);}

bool spfa()

{

int t=0,w=1,i,now;

for(int i=0;i<=n;i++)dis[i]=inf;

q[0]=dis[0]=0;inq[0]=1;

while(t!=w)

{

now=q[t];t++;i=head[now];

if(t==n)t=0;

while(i)

{

if(e[i].v>0&&dis[now]+e[i].c<dis[e[i].to])

{

dis[e[i].to]=dis[now]+e[i].c;

from[e[i].to]=i;

if(!inq[e[i].to])

{q[w++]=e[i].to;if(w==n)w=0;inq[e[i].to]=1;}

}

i=e[i].next;

}

inq[now]=0;

}

if(dis[n]==inf)return 0;

return 1;

}

void mcf()

{

int i,x=inf;

i=from[n];

while(i)

{

x=min(x,e[i].v);

i=from[e[i].from];

}

i=from[n];

while(i)

{

e[i].v-=x;

e[i^1].v+=x;

ans+=x\*e[i].c;

i=from[e[i].from];

}

}

int main()

{

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

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

{

int u,v,w,c;

scanf("%d%d%d%d",&u,&v,&w,&c);

insert(u,v,w,c);

}

dinic();

printf("%d ",ans);

ans=0;build();

ins(0,1,k,0);

while(spfa())mcf();

printf("%d",ans);

return 0;

}

/\* Description

纳米机器人可以从任意一个可以作为起点的块落出发进行修复，也可以在任意一个可以作为终点的块落结束修复（并不是到了某个终点就一定要停止）。春希希望所有的节点都能被修复(只要纳米机器人到过该点就算修复过)，这样才能让雪菜重获新生。

作为纳米机器人1号的你能帮助春希算算至少需要多少个机器人才能拯救雪菜吗？

当然，如果无论如何都无法使得春希的愿望被满足的话，请输出”no solution”（不包括引号）

Input

题目包含多组数据

第1行有一个正整数t，表示数据的组数。

第2行有两个正整数n、m，a，b，分别表示块落的数量、有向边的数量、起点的数量、终点的数量。

第3行有a个正整数，表示可以作为起点的块落。

第4行有b个正整数，表示可以作为终点的块落。

第5行至第m+4行，每行有两个正整数u、v，表示能从编号为u的块落到编号为v的块落。

之后以此类推。

Output

输出共有t行，每行输出对应数据的答案。

Sample Input

2

2 1 1 1

1

2

2 1

3 2 3 3

1 2 3

1 2 3

1 2

1 3

Sample Output

no solution

2

「数据规模和约定」

对于30%的数据，满足n <= 10, m <= 100。

对于60%的数据，满足n <= 200, m <= 5000。

对于100%的数据，满足t<=10,n <= 1000, m <= 10000。

这题……一眼缩点,缩成DAG,然后问题就是限制起点终点的可重复经过的最小路径覆盖

神奇的费用流:

对于一个点u,拆成u->u’ ,连两条边,一条cap=1 cost=1,另一条cap=inf cost=0

s->u cap=inf cost=0,u’ -> t cap=inf cost=0

对于原图的边<u,v>: u’->v cap=inf cost=0

跑最大费用流

有cost的边代表这个点经过一次

那么最大费用流每次会尽量走费用多的边,即走尽量多的点,正确性由网络流保证

增广次数即为答案

复杂度?

每次增广至少有1个cost,总共有n个cost,最坏增广n次

zky神犇告诉我，有向图60%的情况是缩成DAG

\*/

#include<iostream>

#include<cstdio>

#include<cstring>

#include<cstdlib>

#include<set>

#include<ctime>

#include<vector>

#include<cmath>

#include<algorithm>

#include<map>

#define inf 1000000000

#define ll long long

using namespace std;

inline int read()

{

int x=0,f=1;char ch=getchar();

while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}

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

return x\*f;

}

int S,T;

int n,m,A,B,cnt;

int ind,top,scc,cost,ans;

int dfn[1005],low[1005],bl[1005];

int a[1005],b[1005],last[2005],last2[2005];

int d[2005],q[2005];

bool mark[2005],inq[1005];

struct edge{

int to,next,v,c;

}e[100005],ed[100005];

void insert(int u,int v)

{

e[++cnt].to=v;e[cnt].next=last[u];last[u]=cnt;

}

void insert2(int u,int v,int w,int c)

{

ed[++cnt].to=v;ed[cnt].next=last2[u];last2[u]=cnt;ed[cnt].v=w;ed[cnt].c=c;

ed[++cnt].to=u;ed[cnt].next=last2[v];last2[v]=cnt;ed[cnt].v=0;ed[cnt].c=-c;

}

////////////////////////////////////////

void tarjan(int x)

{

low[x]=dfn[x]=++ind;

q[++top]=x;inq[x]=1;

for(int i=last[x];i;i=e[i].next)

if(!dfn[e[i].to])

{

tarjan(e[i].to);

low[x]=min(low[x],low[e[i].to]);

}

else if(inq[e[i].to])

low[x]=min(low[x],dfn[e[i].to]);

if(low[x]==dfn[x])

{

int now=0;scc++;

while(now!=x)

{

now=q[top--];inq[now]=0;

bl[now]=scc;

}

}

}

bool spfa()

{

memset(mark,0,sizeof(mark));

for(int i=0;i<=S;i++)d[i]=-1;

int head=0,tail=1;

q[0]=T;d[T]=0;mark[T]=1;

while(head!=tail)

{

int now=q[head];head++;if(head>2000)head=0;

mark[now]=0;

for(int i=last2[now];i;i=ed[i].next)

if(ed[i^1].v&&d[now]+ed[i^1].c>d[ed[i].to])

{

d[ed[i].to]=d[now]+ed[i^1].c;

if(!mark[ed[i].to])

{

mark[ed[i].to]=1;

q[tail++]=ed[i].to;

if(tail>2000)tail=0;

}

}

}

return d[S]!=-1;

}

int dfs(int x,int f)

{

mark[x]=1;

if(x==T)return f;

int w,used=0;

for(int i=last2[x];i;i=ed[i].next)

if(!mark[ed[i].to]&&ed[i].v&&d[x]-ed[i].c==d[ed[i].to])

{

w=dfs(ed[i].to,min(ed[i].v,f-used));

ed[i].v-=w;ed[i^1].v+=w;cost+=ed[i].c\*w;

used+=w;

if(used==f)return f;

}

return used;

}

void solve()

{

cnt=1;

T=2\*scc+1;S=T+1;

for(int i=1;i<=A;i++)insert2(0,bl[a[i]],inf,0); // inf 0

for(int i=1;i<=B;i++)insert2(bl[b[i]]+scc,T,inf,0); // inf 0

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

for(int i=last[x];i;i=e[i].next)

if(bl[x]!=bl[e[i].to])

insert2(bl[x]+scc,bl[e[i].to],inf,0); // inf 0

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

{

insert2(i,i+scc,1,1); // 1 1

insert2(i,i+scc,inf,0); // inf 0

}

insert2(S,0,1,0);

while(spfa())

{

int pre=cost;

mark[T]=1;

while(mark[T])

{

memset(mark,0,sizeof(mark));

dfs(S,inf);

}

insert2(S,0,1,0);

if(pre==cost)break;

else ans++; // \*\*\*\* //

}

/\*

while(spfa())

void mincf()

{

int i,sum=INF;

i=from[6000];

while(i)

{

sum=min(sum,e[i].v);

i=from[e[i].from];

}

i=from[6000];

while(i)

{

e[i].v-=sum;

e[i^1].v+=sum;

ans+=sum\*e[i].c; // \*\*\*\* //

i=from[e[i].from];

}

}

\*/

}

int main()

{

int T=read();

while(T--)

{

memset(last,0,sizeof(last));

memset(last2,0,sizeof(last2));

memset(dfn,0,sizeof(dfn));

ans=scc=cost=ind=cnt=0;

n=read();m=read();A=read();B=read();

for(int i=1;i<=A;i++)a[i]=read();

for(int i=1;i<=B;i++)b[i]=read();

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

{

int u=read(),v=read();

insert(u,v);

}

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

if(!dfn[i])tarjan(i);

solve();

if(cost!=scc)puts("no solution");

else printf("%d\n",ans);

}

return 0;

}

/\*

4

3 1

1 2 2

2 3 4

3 4 8

3 1

1 3 2

2 3 4

3 4 8

3 1

1 100000 100000

1 2 3

100 200 300

3 2

1 100000 100000

1 150 301

100 200 300

Sample Output

14

12

100000

100301

费用流，构图方法很巧妙。

我一开始的想法是区间放左边，点放右边，跑费用流。后来发现显然是错的，因为在最大流时源点流入区间的边不一定满流，也就是一个区间内的点没有全部覆盖，这显然是错误的。

下面是正确地做法：要保证每个点最多覆盖k次，可以把所有点串联起来，每个点向后一个点连一条容量k、费用0的边。(是不是很机智啊！！)然后对于区间[a,b]，从a到b连一条容量1、费用w的边就可以了。最后从源点到1点、从n点到汇点，分别连一条容量k、费用0的点。

至于这个方法的正确性如何证明呢？？我可以说只可意会不可言传吗=\_=……大家自己脑补吧……额

如此看来，构图真是一种艺术啊！

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <queue>

using namespace std;

#define N 1010

#define INF 0x3f3f3f3f

struct Edge

{

int from,to,next,cap,cost;///起点，终点，同起点下一条边，残余流量，费用

} edge[N\*N];

struct E

{

int u,v,w;

} e[N];

int cnt,head[N],sum[N];

int vis[N],d[N],pp[N],f[N];

int sumflow;///最大流量总和

void init()

{

cnt=0;

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

}

void addedge(int from,int to,int cap,int cost)

{

edge[cnt].from=from;

edge[cnt].to=to;

edge[cnt].cost=cost;

edge[cnt].cap=cap;

edge[cnt].next=head[from];

head[from]=cnt++;

edge[cnt].from=to;

edge[cnt].to=from;

edge[cnt].cost=-cost;

edge[cnt].cap=0;

edge[cnt].next=head[to];

head[to]=cnt++;///存反向边

}

int spfa(int s,int t,int n)

{

queue<int>q;

memset(vis,0,sizeof(vis));

memset(pp,-1,sizeof(pp));///pp[i]表示最短路径上以i为终点的边的编号

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

d[i]=-INF;

d[s]=0;

vis[s]=1;

q.push(s);

while(!q.empty())

{

int u=q.front();

q.pop();

vis[u]=0;

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

{

int v=edge[i].to;

if(edge[i].cap>0&&d[v]<d[u]+edge[i].cost)

{

d[v]=d[u]+edge[i].cost;

pp[v]=i;

if(!vis[v])

{

vis[v]=1;

q.push(v);

}

}

}

}

if(d[t]==-INF) return 0;///找不到一条到终点的路

return 1;

}

int MCMF(int s,int t,int n)

{

int mincost=0,minflow,flow=0;///最小费用，路径中最小流量，总流量

while(spfa(s,t,n))///找当前的最短路

{

minflow=INF+1;

for(int i=pp[t]; i!=-1; i=pp[edge[i].from])

minflow=min(minflow,edge[i].cap);///从路径中找最小的流量

flow+=minflow;///总流量加上最小流量

for(int i=pp[t]; i!=-1; i=pp[edge[i].from])

{

edge[i].cap-=minflow;///当前边减去最小流量

edge[i^1].cap+=minflow;///反向边加上最小流量

}

mincost+=d[t]\*minflow;///最小费用等于路径和\*每条路径的流量（经过多少次）

}

sumflow=flow;

return mincost;

}

int main()

{

int T,n,m,k;

scanf("%d",&T);

while(T--)

{

init();

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

int num=0;

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

{

scanf("%d %d %d",&e[i].u,&e[i].v,&e[i].w);

f[++num]=e[i].u;

f[++num]=e[i].v;

}

sort(f+1,f+1+num);

int tot=1;

for(int i=2; i<=num; i++)

if(f[i]!=f[i-1])

f[++tot]=f[i];

int s=0,t=tot+1;

addedge(s,1,m,0);

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

addedge(i,i+1,m,0);

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

{

int u=lower\_bound(f+1,f+1+tot,e[i].u)-f;

int v=lower\_bound(f+1,f+1+tot,e[i].v)-f;

addedge(u,v,1,e[i].w);

}

addedge(tot,t,m,0);

printf("%d\n",MCMF(s,t,t));

}

return 0;

}