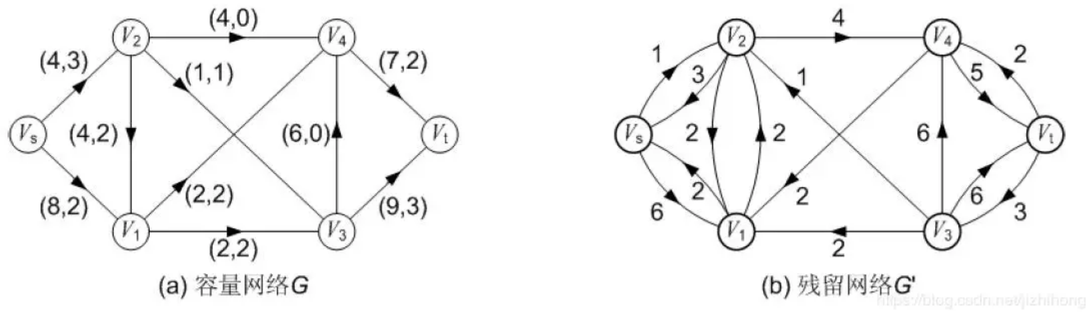
网络流算法

网络流图里，源点流出的量，等于汇点流入的量，除源汇外的任何点，其流入量之和等于流出两之和。

网络最大流量

残余网络

在一个网络流图上，找到一条源到汇的路径 （即找到了一个流量）后，对路径上所有的 边，其容量都减去此次找到的流量，对路径上所有的边，都添加一条反向边，其容量也等于此次找到的流量，这样得到的新图，就称为原图的“残余网络”。



图一.残余网络

添加一条反悔的边后，从源点流出，流入汇点的流量与添加反悔边之前相同，等于在原图的流量中再找一条路径，该路径与作增广图之前寻找到的路径不同， 所以可以求出多条从源点到汇点的流量路径。路径流量相加之和就是最大流量。

Ford-Fulkerson算法

求最大流的过程，就是不断找到一条源到汇的路径，然后构建残余网络，再在残余网络上寻找新的路径，使总流量增加，然后形成新的残余网络，再寻找新路径.....直到某个残余网络上找不到从源到汇的路径为止，最大流就算出来了。

每次寻找新流量并构造新残余网络的过程， 就叫做寻找流量的“增广路径”，也叫“增广”。假设每条边的容量都是整数，该算法每次都能将流至少增加 1 ，由于整个网络的流量最多不超过图中所有的边的容量和C，从而算法会结束。

时间复杂度：边数m，顶点数n，使用DFS算法寻找从源点到汇点的路径，设DFS最多运行C次。

$$\text{时间复杂度：} C*(m+n)=c*n^2$$

算法实现很容易，但是寻找路径时会多次调用DFS算法，使得作了无用功太多，消耗太多时间。

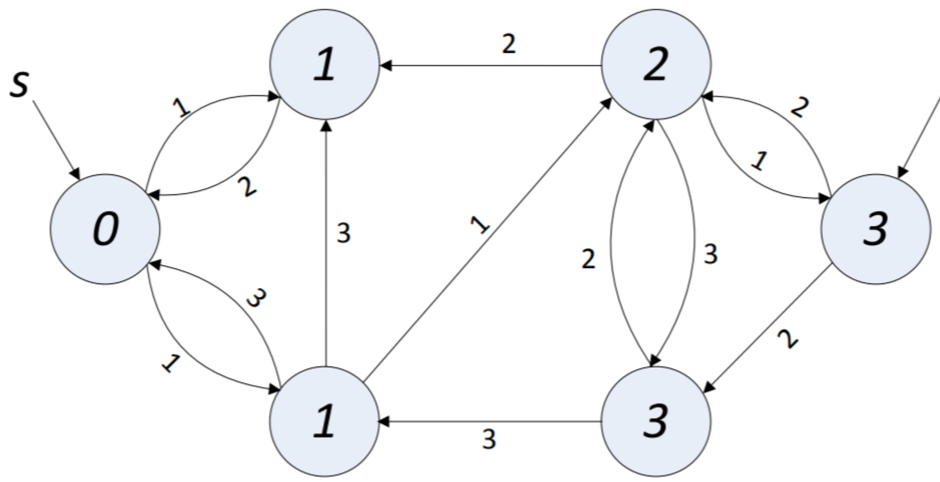
算法改进：每次选择增广的时候，选择从源到汇的具有最少边数的增广路径，即选择路径使用BFS算法，不用DFS算法——**Edmonds-Karp 最短增广路算法**，时间复杂度上限为 $o(nm^2)$ （n 是点数，m 是边数）

Dinic 算法

Edmonds-Karp算法还可以进一步提高：需要多次从源点到汇点调用BFS，可以设法减少调用次数，即使用一种代价较小的高效增广方法，可以在一次增广的过程中，寻找多条增广路径。

算法步骤：

1. 首先用BFS算法对图进行分层：**一个节点的“层”数，就是源点到它最少要经过的边数**（可以用弗洛伊德算法或迪杰特斯拉算法）。在分层时，只要进行到汇点的层次数被算出即可停止，因为按照该DFS的规则，和汇点同层或更下一层的节点，是不可能走到汇点的分完层后，利用DFS从前一层向后一层反复寻找增广路（即要求DFS的每一步都必须要走到下一层的节点）。



图二.流量图分层

2. DFS过程中，要是碰到了汇点，则说明找到了一条增广路径。此时要增加总流量的值，消减路径上各边的容量，并添加反向边，即所谓的进行增广。
3. DFS找到一条增广路径后，并不立即结束，而是回溯后继续DFS寻找下一个增广路径，回溯到的节点 u 满足以下条件：（1）DFS搜索树的树边 (u, v) 上的容量已经变成 0。即刚刚找到的增广路径上所增加的流量，等于 (u, v) 本次增广前的容量。（DFS的过程中，是从 u 走到更下层的 v）（2）u是满足条件（1）的最上层的节点。
4. 如果回溯到源点而且无法继续往下走了，DFS结束。
5. 一次DFS过程中，可以找到多条增广路径。 DFS结束后，对残余网络再次进行分层，然后再进行DFS。当残余网络的分层操作无法算出汇点的层次（即BFS到达不了汇点）时，算法结束，最大流求出。 一般用**栈**实现DFS,这样就能从栈中提取出增广路径。

时间复杂度：$n * n * m$ （n是点数，m是边数）

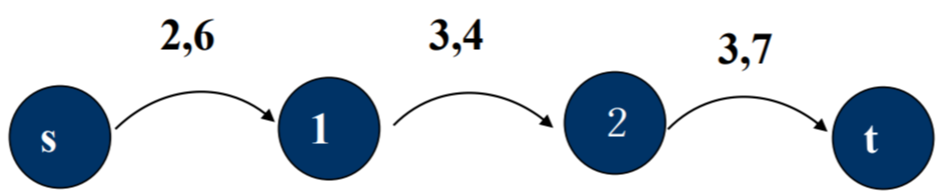
求每条边的最大流量：将原图备份，原图上的边的容量减去做完最大流的残余网络上的边的剩余容量，就是边的流量。

有流量下界的网络最大流

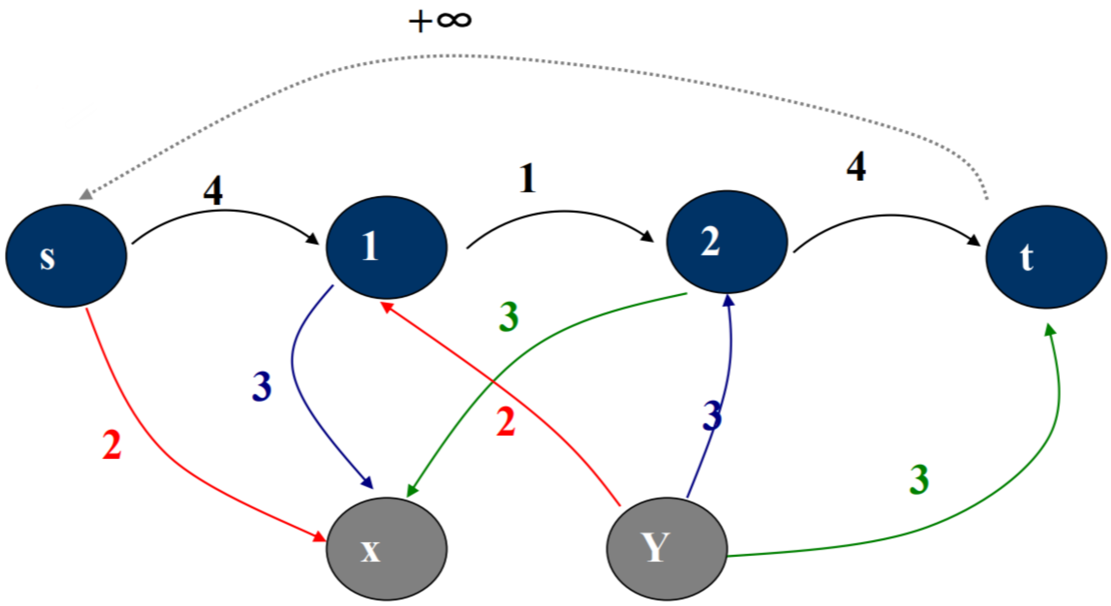
如果流网络中每条边 e 对应两个数字 $B(e)$ 和 $C(e)$, 分别表示该边上的流量至少要是 $B(e)$，最多 $C(e)$,那么，在这样的流网络上求最大流，就是有下界的最大流问题，但是这种网络不一定存在可行流。

将下界“分离”出去，使问题转换为下界为 0 的普通网络流问题：将有下界的边分为必要边和非必要边，将必要边拉出去成立两个新的节点，去除新的节点之间的连线，使这两个新的节点成为新的源点和汇点。若最大流图不能占满汇点的所有边，则该图无可行流。新图最大流若小于新图中x的流入量之和,则原问题无解。

在做过一遍最大流的新图的残余网络中，去掉 $t \to s$ 以及 $s \to t$ 的边，然后以 s 为源，t 为汇再做一次最大流， 此时得到的流量 sum2，则 sum1+sum2 就是在原图 上满足下界的最大流。



图三.有流量上下界的网络图



图四.分离必要边的图

最小费用最大流

设有一个网络图 $G(V,E)$ ，$V=\{s,a,b,c,\ldots,s'\}$ ，E 中的每条边 (i,j) 对应一个容量 $c(i,j)$ 与输送单位流量所需费用 $a(i,j)$ 。如有一个运输方案（可行流），流量为$f(i,j)$ ，则最小费用最大流就是求极值：

$$min_{f\in F}a(f)=min_{f\in F}\sum_{(i,j)\in E}a(i,j)f(i,j)$$

其中 F 为 G 的最大流的集合，即在最大流中寻找费用最小的最大流。

spfa算法

1. 求出从发点到收点的最小费用通路 $\mu(s,t)$
2. 对该通路$\mu(s,t)$分配最大可能的流量：$\overline{f}=min_{(i,j)\in\mu(s,t)}\{c(i,j)\}$并让通路上所有边的容量相应减少了$\overline{f}$。这时，对于通路上的饱和边，其单位流费用相应改为$\infty$
3. 作该通路$\mu(s,t)$上所有边 (i,j) 的反向边 (j,i) ,令 $c(j,i)=\overline{f},d(j,i)=-d(i,j)$ 。
4. 在这样构成的新网络中，重复上述步骤 $1,2,3$ 直到从发点到收点的全部流量等于 f_v 为止（或者再也找不到从 s 到 t 的最小费用通路）

反复用spfa算法做源到汇的最短路进行增广，边权值为边上单位费用。反向边上的单位费用是负的。 直到无法增广，即为找到最小费用最大流。

成立原因：每次增广时，每增加1个流量，所增加的费用都是最小的。

因为有负权边（取消流的时候产生的），所以不 能用迪杰斯特拉算法求最短路。

涉及数学概念

最大流量：网络流理论研究的一个基本问题，求网络中一个可行流 $f*$，使其流量 $v(f)$ 达到最大， 这种流 f 称为最大流，这个问题称为(网络)最大流问题。最大流问题是一个特殊的线性规划问题，就是在容量网络中，寻找流量最大的可行流。

建立如下形式的线性规划数学模型：

$$maxV=f^*$$

$$s.t\begin{cases}\sum f_{ij}-f_{ji}=0(i\neq s,t)\\ \sum f_{sj}-f_{js}=0(i=s)\\ \sum f_{ij}-f_{ji}=-v(f)(i=t)\end{cases}$$

割：设 C_i 为网络 N 中一些弧的集合，若从 N 中删去 C_i 中的所有弧能使得从源点 V_s 到汇点 V_t 的路集为空集时，称 C_i 为 V_s 和 V_t 间的一个割。通俗理解，一个图或网络的割，表示一个切面或切线，将图或网络分为分别包含源点和汇点的两个子集，该切线或切面与网络相交的楞或边的集合，称为图像的割。

最小割：最小割是边权值和最小的割。一个图或网络的割表示一个切面或切线，将图或网络分为分别包含源点和汇点的两个子集，该切线或切面与网络相交的楞或边的集合，称为图像的割。

算法练习

题目一

一个餐厅在相继的 N 天里,每天需用的餐巾数不尽相同。假设第 i 天需要r_i块餐巾(i=1,2,…,N)。餐厅可以购买新的餐巾,每块餐巾的费用为 p*p* 分;或者把旧餐巾送到快洗部,洗一块需 m 天,其费用为 f 分;或者送到慢洗部,洗一块需 n*n* 天(n>m*n*>*m*),其费用为 ss 分(s<fs<*f*)。

每天结束时,餐厅必须决定将多少块脏的餐巾送到快洗部,多少块餐巾送到慢洗部,以及多少块保存起来延期送洗。但是每天洗好的餐巾和购买的新餐巾数之和,要满足当天的需求量。

试设计一个算法为餐厅合理地安排好 N*N* 天中餐巾使用计划,使总的花费最小。编程找出一个最佳餐巾使用计划。

思考

将一天拆成晚上和早上，每天晚上会受到脏餐巾（来源：当天早上用完的餐巾，在这道题中可理解为从原点获得），每天早上又有干净的餐巾（来源：购买、快洗店、慢洗店）

构图：

1.从原点向每一天晚上连一条流量为当天所用餐巾x，费用为0的边，表示每天晚上从起点获得x条脏餐巾。

2.从每一天早上向汇点连一条流量为当天所用餐巾x，费用为0的边，每天白天,表示向汇点提供x条干净的餐巾,流满时表示第i天的餐巾够用

3.从每一天晚上向第二天晚上连一条流量为INF，费用为0的边，表示每天晚上可以将脏餐巾留到第二天晚上（注意不是早上，因为脏餐巾在早上不可以使用）。

4.从每一天晚上向这一天+快洗所用天数t1的那一天早上连一条流量为INF，费用为快洗所用钱数的边，表示每天晚上可以送去快洗部,在地i+t1天早上收到餐巾 。

5.同理，从每一天晚上向这一天+慢洗所用天数t2的那一天早上连一条流量为INF，费用为慢洗所用钱数的边，表示每天晚上可以送去慢洗部,在地i+t2天早上收到餐巾 。

6.从起点向每一天早上连一条流量为INF，费用为购买餐巾所用钱数的边，表示每天早上可以购买餐巾 。
注意，以上6点需要建反向边！3~6点需要做判断（即连向的边必须<=n）

代码

```
#include<cstdio>
#include<queue>
#include<cstring>
```

```
#include<queue>
#include<algorithm>
#define INF 2147483647
#define LL long long
using namespace std;
queue<int> f;
    int n,m,m1,t1,m2,t2,len=-1,st,ed;
    struct node{int x,y,c,d,next;} a[100000];
    int b[100000],last[100000],pre[100000],pos[100000],p[100000];
    LL dis[100000];
    bool bz[100000];
void ins(int x,int y,int c,int d)
{

a[++len].x=x;a[len].y=y;a[len].c=c;a[len].d=d;a[len].next=last[x];last[x]=len;
    a[++len].x=y;a[len].y=x;a[len].c=0;a[len].d=-
d;a[len].next=last[y];last[y]=len;
}
bool spfa()
{
    memset(bz,true,sizeof(bz));
    bz[st]=false;
    memset(dis,63,sizeof(dis));
    dis[st]=0;
    p[st]=INF;
    f.push(st);
    while(!f.empty())
    {
        int x=f.front();
        bz[x]=true;
        for(int i=last[x];i>-1;i=a[i].next)
        {
            int y=a[i].y;
            if(a[i].c>0&&dis[y]>dis[x]+a[i].d)
            {
                dis[y]=dis[x]+a[i].d;
                pos[y]=x;
                pre[y]=i;
                p[y]=min(p[x],a[i].c);
                if(bz[y])
                {
                    f.push(y);
                    bz[y]=false;
                }
            }
        }
        f.pop();
    }
    return dis[ed]<45574;
}
LL flow()
{
    LL ans=0;
    while(spfa())
    {
        ans+=p[ed]*dis[ed];
```

```
        for(int i=ed;i!=st;i=pos[i])
        {
            a[pre[i]].c-=p[ed];
            a[pre[i]^1].c+=p[ed];
        }
    }
    return ans;
}
int main()
{
    int x;
    scanf("%d",&n);
    st=0,ed=2*n+1;
    memset(last,-1,sizeof(last));
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&x);
        ins(st,i,x,0);//每天晚上从起点获得x条脏餐巾
        ins(i+n,ed,x,0);//每天白天,向汇点提供x条干净的餐巾,流满时表示第i天的餐巾够用
    }
    scanf("%d %d %d %d %d",&m,&t1,&m1,&t2,&m2);
    for(int i=1;i<=n;i++)
    {
        if(i+1<=n) ins(i,i+1,INF,0);//每天晚上可以将脏餐巾留到第二天晚上
        if(i+t1<=n) ins(i,i+n+t1,INF,m1);//每天晚上可以送去快洗部,在地i+t1天早上收到餐巾
        if(i+t2<=n) ins(i,i+n+t2,INF,m2);//每天晚上可以送去慢洗部,在地i+t2天早上收到餐巾
        ins(st,i+n,INF,m);//每天早上可以购买餐巾
    }
    printf("%lld",flow());
}
```

题目二

W 公司有 m 个仓库和 n 个零售商店。第 i 个仓库有 a_i 个单位的货物；第 j 个零售商店需要 b_j 个单位的货物。

货物供需平衡，即$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b + j$。

从第 i 个仓库运送每单位货物到第 j 个零售商店的费用为c_{ij} 。

试设计一个将仓库中所有货物运送到零售商店的运输方案，使总运输费用最少。

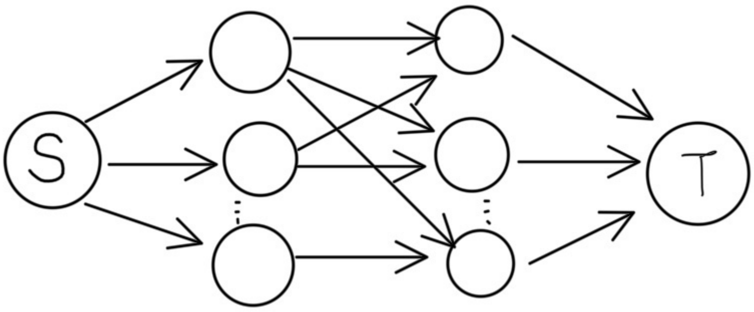
思考

由于每一个仓库只能流出定量的货物，但是又不能把每一个仓库看做源。

所以把所有货物都连到同一个源上，连到第 i 个仓库的边嘚的容量为A_i，费用为0。

每一家零售店又都连到一个汇上，从第i家零售店连出的边的容量为B_i ，费用为0。

中间从仓库到零售店的边就按照题目里的说的那样连，容量为$+\infty$。



图五.图中建模

代码

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int maxn=210,maxm=20205,inf=0x3F3F3F3F;
int m,n,S,T,tot,lnk[maxn],son[maxm],nxt[maxm],w[maxm],cap[maxm],que[maxn],lst[maxn],pre[maxn],dist[maxn],flow[maxn],ans;bool vis[maxn];
inline int read()
{
    int ret=0,f=1;char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-f;ch=getchar();}
    while(ch>='0'&&ch<='9'){ret=ret*10+ch-'0';ch=getchar();}
    return ret*f;
}
inline void add_e(int x,int y,int z,int c)
{tot++;son[tot]=y;w[tot]=z;cap[tot]=c;nxt[tot]=lnk[x];lnk[x]=tot;}
inline void MinCostMaxFlow(int flg)
{
    while(true)
    {
        if(flg==1) memset(dist,63,sizeof(dist));
        else memset(dist,192,sizeof(dist));
        memset(flow,63,sizeof(flow));
        int hed=0,til=1;
        que[1]=S;dist[S]=0;vis[S]=true;pre[T]=0;
        while(hed!=til)
        {
            hed=(hed+1)%maxn;vis[que[hed]]=false;
            for(int i=lnk[que[hed]];i;i=nxt[i])
            {
                if(cap[i]&&((flg==1&&dist[que[hed]]+w[i]<dist[son[i]])||(flg==-1&&dist[que[hed]]+w[i]>dist[son[i]])))
                {
                    dist[son[i]]=dist[que[hed]]+w[i];
                    pre[son[i]]=que[hed];
                    lst[son[i]]=i;
                    flow[son[i]]=min(flow[que[hed]],cap[i]);
                    if(!vis[son[i]])
                    {
                        vis[son[i]]=true;
```

```
                        til=(til+1)%maxn;
                        que[til]=son[i];
                    }
                }
            }
        }
        if(pre[T]==0) return;
        ans+=flow[T]*dist[T];
        int p=T;
        while(p!=S)
        {
            cap[lst[p]]-=flow[T];
            cap[(lst[p]&1)?lst[p]+1:lst[p]-1]+=flow[T];
            p=pre[p];
        }
    }
}
int main()
{
    m=read();n=read();S=1;T=m+n+2;
    for(int i=1;i<=m;i++)
    {
        int ai=read();
        add_e(S,i+1,0,ai);
        add_e(i+1,S,0,0);
    }
    for(int i=1;i<=n;i++)
    {
        int bi=read();
        add_e(i+m+1,T,0,bi);
        add_e(T,i+m+1,0,0);
    }
    for(int i=1;i<=m;i++)
    {
        for(int j=1;j<=n;j++)
        {
            int cij=read();
            add_e(i+1,j+m+1,cij,inf);
            add_e(j+m+1,i+1,-cij,0);
        }
    }
    MinCostMaxFlow(1);
    printf("%d\n",ans);
    for(int i=2;i<=tot;i+=2){cap[i-1]+=cap[i];cap[i]=0;}
    ans=0;
    MinCostMaxFlow(-1);
    printf("%d\n",ans);
    return 0;
}
```