网络流入门

算法简介

几个概念

容量

每条边(u,v)都有一个权值$c(u,v)$被称为容量，而当边不属于这个图时，容量为0

流

流满足以下几个条件：

容量限制：流经过边的流量不能超过该边的容量，即 $f(u,v)≤c(u,v)$

斜对称性：每条边的流量与其相反边的流量之和为0，即 $f(u,v)=-f(v,u)$

流守恒性：从源点流出的流量等于汇点流入的流量

剩余容量

表示一条边的容量与流量之差 $c_{f}(u,v)=c(u,v)-f(u,v)$

残量网络

对于流函数 f 残存网络 G_{f} 是网络 G 中所有结点和剩余容量大于0的边构成的子图。注意，剩余容量大于0的边可能不在原图中。可以理解为，残量网络中包括了那些还剩了流量空间的边构成的图，也包括虚边（即反向边）。

增广路

在原图中若一条从源点到汇点的路径上所有边的剩余容量都大于0，这条路叫做增广路。

对于网络流，现在主流的有 $EK,Dinic,SAP,ISAP$ 算法

算法思想

我们先设点数为 n 边数为 m

EK

使用 BFS 进行增广。

具体来说就是从源点一直 BFS 走来走去，碰到汇点就停，然后进行增广，要注意一下流量合不合法。

增广方式： 把找到的增广路再走一遍，走的时候把这条路的能够成的最大流量减下去，然后给答案加上最小流量。

增广的时候要注意建造反向边，原因是这条路不一定是最优的，这样子程序可以进行反悔。假如我们对这条路进行增广了，那么其中的每一条边的反向边的流量就是它的流量。

时间复杂度为 $O(nm^{2})$ 效率较低。

$Dinic$

每次增广前，我们先用 BFS 来将图分层。设源点的层数为 0，那么一个点的层数便是它离源点的最近距离。

何时停止：如果不存在到汇点的增广路（即汇点的层数不存在），我们即可停止增广。

优化：

多路增广：在一次 DFS 中找出多条增广路。

当前弧优化：因为一条边最多可以被增广一次，所以我们下一次进行增广的时候，就可以不必再走那些已经被增广过的边。

时间复杂度为 $O(n^{2}m)$ 在稀疏图上效率和 EK 算法相当，但在稠密图上效率要比EK算法高很多。在求解二分图最大匹配问题时Dinic 算法的时间复杂度是 $O(m\sqrt{n})$

我们要确保找到的增广路是最短的，所以我们每次找增广路的时候，都只找比当前点层数多1的点进行增广（这样就可以确保我们找到的增广路是最短的）。

```
void addEdge(int i,int a,int b,ll c,int d) {
    u[i]=a;
    v[i]=b;
    w[i]=c;
    rev[i]=d;
    nex[i]=first[a];
    first[a]=i;
}
bool bfs(ll s,ll t) {
    memset(d,0x7fffffff,sizeof(d));
    memset(vis,0,sizeof(vis));
    Q.push(s);
    vis[s]=1;
    d[s]=0;
    while(Q.size()) {
        int p=Q.front();
```

```
        Q.pop();
        for(int i=first[p]; i!=-1; i=nex[i])
            if(!vis[v[i]] && w[i]>0) {
                vis[v[i]]=1;
                d[v[i]]=d[p]+1;
                Q.push(v[i]);
            }
    }
    return vis[t];
}
ll dfs(ll x,ll t,ll a) {
    if(x==t || a==0)return a;
    ll flow=0,f;
    for(ll& i=cur[x]; i!=-1; i=nex[i])
        if(d[x]+1==d[v[i]] && (f=dfs(v[i],t,min(a,w[i])))>0) {
            w[i]-=f;
            w[rev[i]]+=f;
            a-=f;
            flow+=f;
            if(a==0)break;
        }
    return flow;
}
ll Dinic(int s,int t) {
    ll flow=0;
    while(bfs(s,t)) {
        for(int i=1; i<=n; i++)cur[i]=first[i];
        flow+=dfs(s,t,oo);
    }
    return flow;
}



        //加边操作

        for(int i=0; i<m; i++) {
            a=read();
            b=read();
            c=read();
            addEdge(2*i,a,b,c,2*i+1);
            addEdge(2*i+1,b,a,0,2*i);
        }
```

$ISAP$

$Dinic$ 算法中，每次求完增广路之后要跑一遍 BFS 分层ISAP的策略是在反图中，从 t 到 s 点进行 BFS

增广过程和 $Dinic$ 类似，选择比当前点层数少 1 的点来增广，所以也存在当前弧优化。

但不同的是 $ISAP$ 在找增广路的途中会完成下一步的分层，比如现在到了 i 号点，其层数为 d_{i} ，结束这个点的增广过程后，遍历残量网络上 i 的所有出边，找到层最小的出点 j ，然后令 $d_{i}=d_{j}+1$ ，当无出边时，则 $d_{i}=n$ ，则当 $d_{s}\geq n$ 时，图上不存在增广路，此时即可终止算法。

$ISAP$ 还存在 GAP 优化，记录层数为 i 的点的数量为 $num[i]$ ，每当将一个点的层数从 x 更新到 y 时，要同时更行 num 数组的值，当某次更新之后，如果 $num[x]==0$ 则图中出现断层，必然找不到增广路，则可以直接终止算法(将 d_{s} 标记为 n)。

时间复杂度与 $Dinic$ 等同，但是实际时间上优于 $Dinic$ 。

```
inline void addedge(int u,int v,int val){
    node[++cnt].v=v;
    node[cnt].val=val;
    node[cnt].next=head[u];
    head[u]=cnt;
}

void bfs(){
    memset(dep,-1,sizeof(dep));
    memset(gap,0,sizeof(gap));
    dep[t]=0;
    gap[0]=1;
    queue<ll>q;
    q.push(t);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=head[u];i;i=node[i].next) {
            int v=node[i].v;
            if(dep[v]!=-1) continue;
            q.push(v);
            dep[v]=dep[u]+1;
            gap[dep[v]]++;
        }
    }
    return;
}
long long maxflow;
ll dfs(ll u,ll flow){
    if(u==t){
        maxflow+=flow;
        return flow;
    }
    ll used=0;
    for(int i=cur[u];i;i=node[i].next){
        cur[u]=i;
        ll d=node[i].v;
        if(node[i].val&&dep[d]+1==dep[u]){
            int mi=dfs(d,min(node[i].val,flow-used));
```

```
            if(mi){
                node[i].val-=mi;
                node[i^1].val+=mi;
                used+=mi;
            }
            if(used==flow)return used;
        }
    }
    --gap[dep[u]];
    if(gap[dep[u]]==0)dep[s]=n+1;
    dep[u]++;
    gap[dep[u]]++;
    return used;
}
long long ISAP(){
    maxflow=0;
    bfs();
    while(dep[s]<n)    memcpy(cur,head,sizeof(head)),dfs(s,inf);
    return maxflow;
}



        //加边操作

        for(int i=1;i<=m;i++){
            u=Read();v=Read();w=Read();
            addedge(u,v,w);addedge(v,u,0);
        }
```

预流推进

思想：从源点疯狂“灌水”，水不停流向汇点，能流多少流多少，最后汇点的水的量，就是最大流。

余流：每个点当前有多少水。

步骤：

假装源点有无限的水，并向周围的点推流，推的流量不能超过自身的余流，也不能超过边的容量，并让周围的点入队(s 和 t 不能入队)。

不断取队首的元素，对队首元素进行推流。

队列为空的时候结束算法，此时汇点的余流即为最大流。

但是有可能会存在两个点之间来回推流，导致死循环。

于是我们需要给每个点设置一个高度，让水只从搞出往低处走，在算法进行的时候，不断地对有余流的点更改高度（更新为与它相邻（算反向边）且最低的点的高度+ 1），直到这些点全部没有余流为止。 而此时，因为两个点来回推流的时候，高度会不断上升，而超过 s 的高度之后，他们自动会把流还给 s 这样就不会死循环了，而 s 的初始高度我们设为 n 即可。

奇慢无比，但是是下一个算法的基础。

$HLPP$

步骤：

从 t 到 s 反向 BFS 使得每个点有一个初始高度。

从 s 开始推流，将有余流的点放进优先队列，

不断从优先队列中取出高度最高的点进行推流操作，

如果还有余流，更新高度，重新放入优先队列

优先队列为空时结束算法t的余流是最大流。

为什么相较于普通的预流推进变快了呢？首先我们用BFS预处理了高度，并且因为有优先队列，所以我们可以每次选用高度最高的点，这样大大减少了推流的次数。

优化：

GAP 优化，同 $ISAP$ 如果某个高度不存在，将所有比该高度高的节点标记为不可到达。（使它的高度为 $n+1$ 这样就会直接向 s 推流了）。

时间复杂度：理论上是 $O(n^{2}\sqrt{m})$ 但是有较大常数，实际状况下，一般 $ISAP$ 已经足够用。

```
inline void addEdge(const ll u,const ll v,const ll f) {
    a[u].push_back(edge(v,f,a[v].size()));
    a[v].push_back(edge(u,0,a[u].size()-1));
}
inline void relabel(ll n,ll t) {
    h.assign(n,n);
    h[t]=0;
    cnt.assign(n,0);
    que.clear();
    que.resize(n+1);
    ll qh=0,qt=0;
    for(que[qt++]=t; qh<qt;) {
        ll u=que[qh++],het=h[u]+1;
        for(Iterator p=a[u].begin(); p!=a[u].end(); ++p) {
            if(h[p->to]==n&&a[p->to][p->next].flow>0) {
                cnt[h[p->to]=het]++;
                que[qt++]=p->to;
            }
        }
    }
    for(ll i=0; i<=n; ++i) {
        llist[i].clear();
        dlist[i].clear();
    }
```

```
    for(ll u=0; u<n; ++u) {
        if(h[u]<n) {
            iter[u]=dlist[h[u]].insert(dlist[h[u]].begin(),u);
            if(e[u]>0)llist[h[u]].push_back(u);
        }
    }
    hst=(nowh=h[que[qt-1]]);
}
inline void push(ll u,edge &ed) {
    ll v=ed.to;
    ll df=min(e[u],ed.flow);
    ed.flow-=df;
    a[v][ed.next].flow+=df;
    e[u]-=df;
    e[v]+=df;
    if(0<e[v]&&e[v]<=df)llist[h[v]].push_back(v);
}
inline void push(ll n,ll u) {
    ll nh=n;
    for(Iterator p=a[u].begin(); p!=a[u].end(); ++p) {
        if(p->flow>0) {
            if(h[u]==h[p->to]+1) {
                push(u,*p);
                if(e[u]==0)return;
            } else nh=min(nh,h[p->to]+1);
        }
    }
    ll het=h[u];
    if(cnt[het]==1) {
        for(ll i=het; i<=hst; ++i) {
            for(List::iterator it=dlist[i].begin(); it!=dlist[i].end();
++it) {
                cnt[h[*it]]--;
                h[*it]=n;
            }
            dlist[i].clear();
        }
        hst=het-1;
    } else {
        cnt[het]--;
        iter[u]=dlist[het].erase(iter[u]);
        h[u]=nh;
        if(nh==n)return;
        cnt[nh]++;
        iter[u]=dlist[nh].insert(dlist[nh].begin(),u);
        hst=max(hst,nowh=nh);
        llist[nh].push_back(u);
    }
}
inline ll hlpp(ll n,ll s,ll t) {
    if(s==t)return 0;
```

```
    nowh=0;
    hst=0;
    h.assign(n,0);
    h[s]=n;
    iter.resize(n);
    for(ll i=0; i<n;
++i)if(i!=s)iter[i]=dlist[h[i]].insert(dlist[h[i]].begin(),i);
    cnt.assign(n,0);
    cnt[0]=n-1;
    e.assign(n,0);
    e[s]=INF;
    e[t]=-INF;
    for(ll i=0; i<(ll)a[s].size(); ++i)push(s,a[s][i]);
    relabel(n,t);
    for(ll u; nowh>=0;) {
        if(llist[nowh].empty()) {
            nowh--;
            continue;
        }
        u=llist[nowh].back();
        llist[nowh].pop_back();
        push(n,u);
    }
    return e[t]+INF;
}



        //加边操作

        for(register int i=m; i>0; --i) {
            u=read(),v=read(),f=read();
            addEdge(u,v,f);
        }
```

至此洛谷的两道模板题（基础版和加强版）结束

补充：最小割

由最大流最小割定理知，最小割等于最大流，如果求割边数量，则要将每条边的容量变为 1，重新跑最大流模板即可。

经典问题

现在有 n 个物品， 2 个集合 A 和 B ，如果将一个物品放入 A 集合产生了 a_{i} 的代价，放入 B 集合会产生 b_{i} 的代价，还有 m 个限制条件形如 u_{i},v_{i},w_{i} ，其表示如果 u_{i},v_{i} 不在同一集合，会产生 w_{i} 的代价，每个物品必须且只能属于一个集合，求总共最

小代价。

对于放在集合 A 产生的代价，我们可以从源点 s 向这个点连一条容量为 a_{i} 的边，对于放在集合 B 产生的代价，我们可以从这个点向汇点 t 连一条容量为 b_{i} 的边，于是利用最小割的思想，因为一个点只能属于一个集合，所以两条边至少要有一条被选中，不然会有从 s 到 t 的流。

而对于最后一种情况，不妨设 u_{i} 在 A 集合里，v_{i} 在 B 集合里，所以现在是 $s→u_{i}$ 和 $v_{i}→t$ 有边，而想要承担 w_{i} 的代价，我们只需要在 u_{i} , v_{i} 之间连一条 w_{i} 的边，因为另一种情况也有可能，所以这条边要是双向的，于是图就构建完毕了。

代码练习

1.https://www.luogu.com.cn/problem/P1345

给定点总数 n 边总数 m 源点 s 汇点 t 边是双向边，问最少割掉多少个点才能让源点汇点失去联系

很明显是一个最小割，但这里割的是点不是边，于是需要拆点（太长时间不做连基本操作都忘了的 wzb 是屑…），1 到 n 是出点，分别对应坐标 $+n$ 的 $n+1$ 到 $2×n$ 是入点，对于正常的 m 条边，我们从 u 到 $v+n$ 连一条权值为 inf 的边，反向权值为 0 。因为是双向的，所以 v 到 $u+n$ 的边也是 inf 的，反向权值为 0 。为 inf 的原因是我们不要割掉这种边，我们的目的是割掉入点和出点之间的边，表示割掉这个点。而源点和汇点显然不需要割掉，所以对于其他的点我们都从入点到出点连一条正向为 1 反向为 0 的边。

```
while(~scanf("%d %d %d %d",&n,&m,&s,&t)) {
        cnt=1;
        memset(head,0,sizeof(head));
        ll u,v;
        for(int i=1;i<=n;i++){
            if(i==s||i==t){
                addedge(i+n,i,inf);
                addedge(i,i+n,0);
            }else{
                addedge(i+n,i,1);
                addedge(i,i+n,0);
            }
        }
        for(int i=1; i<=m; i++) {
            u=Read();
            v=Read();
            addedge(u,v+n,inf);
            addedge(v+n,u,0);
            addedge(v,u+n,inf);
            addedge(u+n,v,0);
        }
        n<<=1;
        printf("%lld\n",ISAP());
    }
```

2.https://www.luogu.com.cn/problem/P2857

给定 n 只牛和 m 个牛棚 $(n≤1000,m≤20)$ □ n 只牛每只都有自己的想法，将 m 个牛棚按照喜欢程度进行排序，最后给出 m 个牛棚的容量，求在所有的分配方案中，让牛所居牛棚的座次最高与最低的跨度最小

每只牛看做一个点，每个牛棚也看做一个点，再构造一个源点和汇点。首先牛棚容量是牛棚的点连到汇点，容量为牛棚容量。源点连到各个牛，容量为 1，最后是牛和牛棚中间的边。注意到答案具有单调性，可以二分解决，最小为 1，最大为 m □对于每一个 mid □可以把从第 i 到第 $i+mid-1$ 的牛棚和所有的牛都连起来，容量为 1，如果跑出的最大流是 n □证明所有的牛都能分进去，二分继续。这样点数是 $O(n+m)$ □边数是 $O(n×m)$ □二分次数是 $O(logm)$ 的，每次要建 $O(m)$ 次图，但是由于各种玄学优化以及实际数据较水，所以个位数 ms 就通过了。

```
bool check(int v) {
    for(int i=1; i<=m-v+1; i++) {
        cnt=1;
        memset(head,0,sizeof(head));
        s=m+n+1,t=m+n+2;
        for(int j=1; j<=n; j++) {
            addedge(s,j,1);
            addedge(j,s,0);
        }
        for(int j=1; j<=m; j++) {
            addedge(j+n,t,room[j]);
            addedge(t,j+n,0);
        }
        for(int j=1; j<=n; j++) {
            for(int k=i; k<=i+v-1; k++) {
                addedge(j,bj[j][k]+n,1);
                addedge(bj[j][k]+n,j,0);
            }
        }
        ll anss=ISAP();
        if(anss==n) return true;
    }
    return false;
}
int main() {
    while(~scanf("%d %d",&n,&m)) {
        for(int i=1; i<=n; i++) {
            for(int j=1; j<=m; j++) {
                scanf("%d",&bj[i][j]);
            }
        }
        for(int i=1; i<=m; i++) {
            scanf("%lld",&room[i]);
        }
        int l=1,r=m,ans=-1;
        while(l<=r) {
            int mid=l+r>>1;
```

```
            if(check(mid)) {
                ans=mid;
                r=mid-1;
            } else {
                l=mid+1;
            }
        }
        if(ans==-1) printf("%d\n",m);
        else printf("%d\n",ans);
    }
    return 0;
}
```

3.https://www.luogu.com.cn/problem/CF387D

定义一个有趣的图是：

1.存在一个点 u 有自环

2.这个点和其他所有点v都有边 (u,v) 和边 (v,u)

3.其他所有点的入度和出度刚好为 2 。

4.没有重边

保证输入没有重边

现在可以进行增加一条边或者删除一条边的操作，问给定一个图最少操作多少次才能把它变成一张有趣的图。

首先我们遍历每一个点，让它作为结点 u 对于自环和所有的那两条边，缺少则添加它。对于其他的点，因为要保证这些点的入度出度都为 2 ，因为有 $n-1$ 个点，发现正好要 $n-1$ 条边。并且相当于每一个点都只有一条边进入和一条边发出，所以我们可以拆点并用网络流或者二分图来解决。假设网络流跑出的结果是 $tmpans$ 代表最多有这些边是有用的，其他边要删去，所以要删掉原来和 u 无关的所有边的数量 $-tmpans$ 还要补足 $n-1-tmpans$ 条边，并且补足这些边一定有一组解是满足题意的。三种答案加起来即为所求。复杂度 $O(mn^{\frac 3 2})$

```
int main() {
    while(~scanf("%d %d",&n,&m)) {
        for(int i=1,u,v; i<=m; i++) {
            u=Read();
            v=Read();
            mapp[u][v]++;
        }
        int final_ans=INT_MAX;
        for(int i=1; i<=n; i++) {
            int anss=0;
            if(!mapp[i][i]) anss++;
            for(int j=1; j<=n; j++) {
                if(j==i) continue;
                if(!mapp[j][i]) anss++;
                if(!mapp[i][j]) anss++;
```

```
            }
            memset(head,0,sizeof(head));
            cnt=1;
            int cntt=0;
            for(int j=1; j<=n; j++) {
                if(j==i) continue;
                for(int k=1; k<=n; k++) {
                    if(k==i) continue;
                    if(mapp[j][k]) {
                        cntt++;
                        addedge(j,k+n,1);
                        addedge(k+n,j,0);
                    }
                }
            }
            s=i,t=i+n;
            for(int j=1;j<=n;j++){
                addedge(s,j,1);
                addedge(j,s,0);
            }
            for(int j=1;j<=n;j++){
                addedge(j+n,t,1);
                addedge(t,j+n,0);
            }
            ll tmp_ans=ISAP();
            //printf("%lld\n",tmp_ans);
            int qwq=cntt-tmp_ans+n-1-tmp_ans+anss;
            if(qwq<final_ans){
                final_ans=qwq;
            }
            //printf("%d %d\n",qwq,final_ans);
        }
        printf("%d\n",final_ans);
    }
    return 0;
}
```

4.https://www.luogu.com.cn/problem/P1344

给定 n 个点， 1 是源点， n 是汇点，给出 m 条有向边，每条边有割掉的代价，我们想让代价最小，并且找到代价最小的前提下割掉最少的边。

割掉最少的边貌似之前有提过，将所有的边权改为 1，跑一遍网络流，但是这样不一定是最大流。于是我们把原来的容量乘一个很大的权值最后再加 1，因为权值很大，剩下的 1 不会改变最大流的值（除以原来的权值即可），而又通过最小割的理论得到结果必然是切掉最少的边的结果。注意数据范围把 inf 设大一点即可。

```
int main() {
    while(~scanf("%d %d",&n,&m)) {
        ll w;
```

```
        cnt=1;
        for(int i=1,u,v; i<=m; i++) {
            u=Read();
            v=Read();
            scanf("%lld",&w);
            addedge(u,v,(m+1)*w+1);
            addedge(v,u,0);
        }
        s=1,t=n;
        ll tmp=ISAP();
        printf("%lld %lld\n",tmp/(m+1),tmp%(m+1));
    }
    return 0;
}
```

5.https://www.luogu.com.cn/problem/P3153

舞会上有 n 个男生和 n 个女生，每首曲子需要所有男女配对跳舞，但是有一些男女互相喜欢，其他的都是互相不喜欢，要求每个男生和每个女生最多只能和 k 个不喜欢的异性跳舞，问最多能有多少首舞曲？

答案显然具有单调性，因此二分解之，比如二分到 v □

难点是如何限制只能和 k 个不喜欢的异性跳舞，这里显然要把一个人拆成喜欢和不喜欢两个点，不然无法限制不喜欢这个点。

然后是男女喜欢的话就男喜欢连向女喜欢，容量为 1。男女不喜欢是男不喜欢连女不喜欢，容量为 1。

最后解决源点汇点以及喜欢不喜欢之间边的问题，因为是对不喜欢有限制，而对喜欢不限制，所以是源点连男喜欢 v 的边，然后男喜欢连男不喜欢 k 的边，女那边同理，跑一遍最大流，如果等于 $n\times v$ □ 则说明可以跳 v 首，反之不可，于是此题可解。

```
string str;
int mapp[52][52];

bool check(int v) {
    cnt=1;
    memset(head,0,sizeof(head));
    for(int i=1; i<=n; i++) {
        addedge(s,i,v);
        addedge(i,s,0);
        addedge(i,i+n,m);
        addedge(i+n,i,0);
        addedge(i+3*n,t,v);
        addedge(t,i+3*n,0);
        addedge(i+2*n,i+3*n,m);
        addedge(i+3*n,i+2*n,0);
    }
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=n; j++) {
            if(mapp[i][j]) {
```

```
                addedge(i,j+3*n,1);
                addedge(j+3*n,i,0);
            } else {
                addedge(i+n,j+n*2,1);
                addedge(j+n*2,i+n,0);
            }
        }
    }
    ll tmpans=ISAP();
    if(tmpans==v*n) return true;
    return false;
}

int main() {
    while(~scanf("%d %d",&n,&m)) {
        for(int i=1; i<=n; i++) {
            cin>>str;
            for(int j=0; j<n; j++) {
                if(str[j]=='Y') mapp[i][j+1]=1;
            }
        }
        int l=0,r=1e9,ans=0;
        s=4*n+1,t=4*n+2;
        while(l<=r) {
            int mid=(l+r)>>1;
            if(check(mid)) {
                l=mid+1;
                ans=mid;
            } else {
                r=mid-1;
            }
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

6.https://www.luogu.com.cn/problem/P3872

给出 N 部电影和依赖总数 M 某人对每部电影都有一个体验值，依赖关系指的是对于一个有序对 (A,B) 和一个值 C 如果看了 A 没看 B 则体验值减少 C 求最大体验值。这里原始的体验值范围是 $[-1000,1000]$。电影数不超过 100。

有负数怎么建图呢？这里对于体验值正的电影，我们从源点连到这个电影代表的点，负数先不管，对于有序对 (A,B) 和值 C 我用直觉从 A 到 B 连了一个权值为 C 的边。

剩下是原始体验值为负的情况，貌似还没出现汇点，我尝试着把这些点连向汇点，权值为给定值的相反数。这时如果对这张图跑最大流，最小割貌似是将所有的体验值正的电影选定的情况下，为了保证最后体验值最大，舍弃的最少的情况。

具体地说，舍弃从源点开始的边表示退掉体验值为正的那部电影，舍弃电影之间的边表示就是看 A 不看

B 舍弃到汇点的边表示为了不舍弃 C 而看 B 所以用正的总量-最小割即为所求。

```
int main() {
    while(~scanf("%d %d",&n,&m)) {
        cnt=1;
        s=n+1,t=n+2;
        ll sum=0;
        for(int i=1,tmp; i<=n; i++) {
            tmp=Read();
            if(tmp>0) {
                addedge(s,i,tmp);
                addedge(i,s,0);
                sum+=tmp;
            } else {
                addedge(i,t,-tmp);
                addedge(t,i,0);
            }
        }
        for(int i=1,u,v,w;i<=m;i++){
            u=Read();v=Read();w=Read();
            addedge(u,v,w);
            addedge(v,u,0);
        }
        printf("%lld\n",sum-ISAP());
    }
    return 0;
}
```

7.https://www.luogu.com.cn/problem/P4313

经典问题。大意：一个班坐成 $n\times m$ 的矩阵，每个人选文科会获得一种满意值，选理科会获得另一种满意值，这个人和周围（上下左右）的人都选理科会额外获得第三种满意值，都选文科会额外获得第四种满意值，问如何让满意值最大。

仿照上一题，我们先将所有的满意值相加，再利用最小割减下去。对于一个点因为没有别的限制，所以不需要拆点，对于文科需要将源点连到这个点，理科是这个点连到汇点。

后两种额外的需要自己和周围的人都选同一科，是且的关系。并且由于我们建图的意义是割掉哪条边代表排除哪条边代表的那种情况，只有把这个奖励单独作为一个点，并且如果是文科，从源点连到这个点，权值为奖励值，奖励点分别和那几个座位代表的点相连，权值为 inf 才能使得如果有一个点不连源点则一定连了汇点，那么会有流从源点到奖励点再到那个点最后到汇点，只能将这个奖励的流割掉，所以满足题意。

```
int dx[5]={0,1,-1,0,0};
int dy[5]={0,0,0,1,-1};

bool check(int x,int y){
    return x>=1&&x<=n&&y>=1&&y<=m;
}
```

```
int main() {
    while(~scanf("%d %d",&n,&m)) {
        cnt=1;
        memset(head,0,sizeof(head));
        s=n*m*4+1,t=n*m*4+2;
        ll sum=0;
        for(int i=1,tmp; i<=n; i++) {
            for(int j=1; j<=m; j++) {
                tmp=Read();
                sum+=tmp;
                addedge(s,(i-1)*m+j,tmp);
                addedge((i-1)*m+j,s,0);
            }
        }
        for(int i=1,tmp; i<=n; i++) {
            for(int j=1; j<=m; j++) {
                tmp=Read();
                sum+=tmp;
                addedge((i-1)*m+j,t,tmp);
                addedge(t,(i-1)*m+j,0);
            }
        }
        for(int i=1,tmp; i<=n; i++) {
            for(int j=1; j<=m; j++) {
                tmp=Read();
                sum+=tmp;
                addedge(s,(i-1+n)*m+j,tmp);
                addedge((i-1+n)*m+j,s,0);
                for(int k=0;k<5;k++){
                    int xx=i+dx[k],yy=j+dy[k];
                    if(check(xx,yy)){
                        addedge((i-1+n)*m+j,(xx-1)*m+yy,inf);
                        addedge((xx-1)*m+yy,(i-1+n)*m+j,0);
                    }
                }
            }
        }
        for(int i=1,tmp; i<=n; i++) {
            for(int j=1; j<=m; j++) {
                tmp=Read();
                sum+=tmp;
                addedge((i-1+2*n)*m+j,t,tmp);
                addedge(t,(i-1+2*n)*m+j,0);
                for(int k=0;k<5;k++){
                    int xx=i+dx[k],yy=j+dy[k];
                    if(check(xx,yy)){
                        addedge((xx-1)*m+yy,(i-1+2*n)*m+j,inf);
                        addedge((i-1+2*n)*m+j,(xx-1)*m+yy,0);
                    }
                }
```

```
                }
            }
            printf("%lld\n",sum-ISAP());
        }
        return 0;
}
```