网络流

HaHeHyt

2023 年 12 月 22 日

网络（network）是指一个特殊的有向图 $G = (V, E)$，其与一般有向图的不同之处在于有容量和源汇点。
E 中的每条边 (u, v) 都有一个被称为容量（capacity）的权值，记作 $c(u, v)$。当 $(u, v) \notin E$ 时，可以假定 $c(u, v) = 0$。
V 中有两个特殊的点：源点（source）s 和汇点（sink）t（$s \neq t$）。
顾名思义，源点 s 初始有 $+\infty$ 水流，最终都要流到汇点 t，中间水管不能爆，接下来引入最大流等问题。

对于网络 $G=(V,E)$，流（flow）是一个从边集 E 到整数集或实数集的函数，其满足以下性质。
容量限制：对于每条边，流经该边的流量不得超过该边的容量，即 $0 \leq f(u,v) \leq c(u,v)$；
流守恒性：除源汇点外，任意结点 u 的净流量为 0。其中，我们定义 u 的净流量为 $f(u)=\sum_{x\in V} f(u,x)-\sum_{x\in V} f(x,u)$。
对于网络 $G=(V,E)$ 和其上的流 f，我们定义 f 的流量 $|f|$ 为 s 的净流量 $f(s)$。作为流守恒性的推论，这也等于 t 的净流量的相反数 $-f(t)$。
对于网络 $G=(V,E)$，如果 $\{S,T\}$ 是 V 的划分（即 $S\cup T=V$ 且 $S\cap T=\varnothing$），且满足 $s\in S, t\in T$，则我们称 $\{S,T\}$ 是 G 的一个 $s-t$ 割（cut）。我们定义 $s-t$ 割 $\{S,T\}$ 的容量为 $||S,T||=\sum_{u\in S}\sum_{v\in T} c(u,v)$。

常见的网络流问题包括但不限于以下类型问题。

- 最大流问题：对于网络 $G=(V,E)$，给每条边指定流量，得到合适的流 f，使得 f 的流量尽可能大。此时我们称 f 是 G 的最大流。
- 最小割问题：对于网络 $G=(V,E)$，找到合适的 $s-t$ 割 $\{S,T\}$，使得 $\{S,T\}$ 的容量尽可能小。此时我们称 $\{S,T\}$ 是 G 的最小割。
- 最小费用最大流问题：在网络 $G=(V,E)$ 上，对每条边给定一个权值 (u,v)，称为费用（cost），含义是单位流量通过 (u,v) 所花费的代价。对于 G 所有可能的最大流，我们称其中总费用最小的一者为最小费用最大流。
- 最大权闭合子图：闭合子图，指原图的一个点集，满足点集内的点只向点集内的点连边而不向外部连边。最大权闭合子图，指在点带权后，权值和最大的闭合子图。（这里不讲，大家自行学习）

问题回顾

- 对于网络 $G = (V, E)$，给每条边指定流量，得到合适的流 f，使得 f 的流量尽可能大。此时我们称 f 是 G 的最大流。
- 模板题：洛谷 P3376。

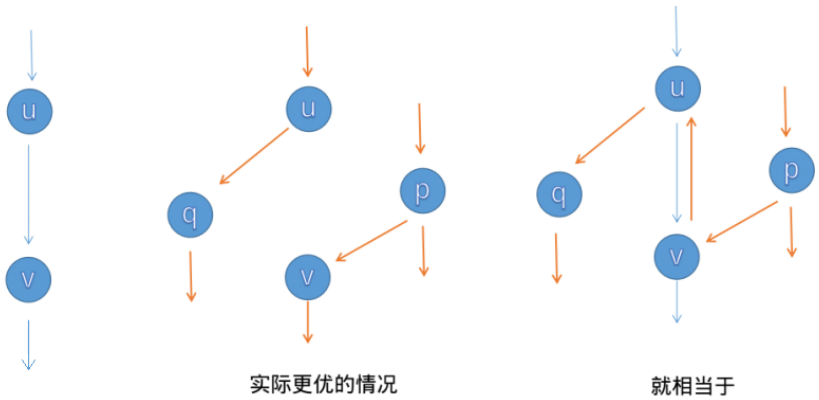
残量网络

- 给定网络 G 及 G 上的流 f，我们做如下定义。
- 对于边 (u, v)，我们将其容量与流量之差称为剩余容量 $c_f(u, v)$（Residual Capacity），即 $c_f(u, v) = c(u, v) - f(u, v)$。
- 我们将 G 中所有结点和剩余容量大于 0 的边构成的子图称为残量网络 G_f（Residual Network），即 $G_f = (V, E_f)$，其中 $E_f = \{(u, v) \mid c_f(u, v) > 0\}$。
- **注意：残量网络的边不一定在原图上。**

增广路

- 我们将 G_f 上一条从源点 s 到汇点 t 的路径称为增广路 (Augmenting Path)。
- 对于一条增广路，我们给每一条边 (u, v) 都加上等量的流量，以令整个网络的流量增加，这一过程被称为增广 (Augment)。
- 由此，最大流的求解可以被视为若干次增广分别得到的流的叠加。
- 此外，在 `Ford-Fulkerson` 增广的过程中，对于每条边 (u, v)，我们都新建一条反向边 (v, u)。我们约定 $f(u, v) = -f(v, u)$，这一性质可以通过在每次增广时引入退流操作来保证，即 $f(u, v)$ 增加时 $f(v, u)$ 应当减少同等的量。
- 提前提一嘴：在实现中建图时，用链式前向星。初始 $tot = 2$，这样设原边的 id 为 i，则反边的 id 为 id `xor` 1。

关于退流



图说的很清楚了，相当于如果你 **贪心**的找增广路，而且不**反悔**的话，那么会出现你没考虑到的更优情况。
既然允许反悔了，最终的一定是最优的。

EK 算法

考虑上文说的增广路以及退流方法。我们已经理解了这样为啥最优了，于是进行算法实现。采用 BFS。其具体流程如下：

- 如果在 G_f 上我们可以从 s 出发 BFS 到 t，则我们找到了新的增广路。
- 对于增广路 p，我们计算出 p 经过的边的剩余容量的最小值 $\Delta = \min_{(u,v)\in p} c_f(u,v)$。我们给 p 上的每条边都加上 Δ 流量，并给它们的反向边都退掉 Δ 流量，令最大流增加了 Δ。
- 因为我们修改了流量，所以我们得到新的 G_f，我们在新的 G_f 上重复上述过程，直至增广路不存在，则流量不再增加。

以上算法即 EK（Edmonds–Karp）算法。

EK 算法复杂度

下面统一规定：n 为点数，m 为边数。我不是很会证明，于是直接给出关键结论：

- 单轮 BFS 增广的时间复杂度是 $O(m)$。
- 增广总轮数的上界是 $O(nm)$。详细证明参见 OI-WiKi。

于是总复杂度 $O(nm^2)$。

EK 算法代码实现

```
const int maxn = 210, maxm = 100010;
long long int n, m, s, t;

struct Edge{
    long long int to, next, weight;
};
Edge edges[maxm];
long long int edge_cnt = 1, head[maxn];

void add(long long int x,long long  int y,long long int w){
    edges[++edge_cnt].next = head[x];
    edges[edge_cnt].to = y;
    edges[edge_cnt].weight = w;
    head[x] = edge_cnt;
}
```

```
long long int last[maxn], flow[maxn];
bool bfs(){
    memset(last, -1, sizeof(last));
    queue<int> q;
    q.push(s);
    flow[s] = INF;
    while (!q.empty()){
        long long int front = q.front();
        q.pop();
        if (front == t) break;
        for (int eg = head[front]; eg != 0; eg = edges[eg].next){
            long long int v = edges[eg].to, vol = edges[eg].weight;
            if (last[v] == -1 && vol > 0){
                last[v] = eg;
                flow[v] = min(flow[front], vol);
                q.push(v);
            }
        }
    }
    return (last[t] != -1);
}

long long int EK(){
    long long int max_flow = 0;
    while(bfs()){
        for (int i = t; i != s; i = edges[last[i] ^ 1].to){
            edges[last[i]].weight -= flow[t];
            edges[last[i] ^ 1].weight += flow[t];
        }
        max_flow += flow[t];
    }
    return max_flow;
}
```

dinic 算法-分层

- 一句话：BFS 后按到源点的最短距离分层。
- 考虑在增广前先对 G_f 做 BFS 分层，即根据结点 u 到源点 s 的距离 $d(u)$ 把结点分成若干层。令经过 u 的流量只能流向下一层的结点 v，即删除 u 向层数标号相等或更小的结点的出边，我们称 G_f 剩下的部分为层次图（Level Graph）。
- 形式化地，我们称 $G_L = (V, E_L)$ 是 $G_f = (V, E_f)$ 的层次图，其中 $E_L = \{(u, v) \mid (u, v) \in E_f, d(u) + 1 = d(v)\}$。

dinic 求最大流

如果我们在层次图 G_L 上找到一个最大的增广流 f_b，使得仅在 G_L 上是不可能找出更大的增广流的，则我们称 f_b 是 G_L 的阻塞流（Blocking Flow）。**简单的说：是一个极大的流。**
定义层次图和阻塞流后，Dinic 算法的流程如下：

- 在 G_f 上 BFS 出层次图 G_L。
- 在 G_L 上 DFS 出阻塞流 f_b。
- 将 f_b 并到原先的流 f 中，即 $f \leftarrow f + f_b$。
- 重复以上过程直到不存在从 s 到 t 的路径。

此时的 f 即为最大流。

dinic 当前弧优化

tips：优化前后多出的部分会再代码中标注。

在分析这一算法的复杂度之前，我们需要特别说明「在 G_L 上 DFS 出阻塞流 f_b」的过程。DFS 阻塞流的过程则稍需技巧——我们需要引入当前弧优化。

- 注意到在 G_L 上 DFS 的过程中，如果结点 u 同时具有大量入边和出边，并且 u 每次接受来自入边的流量时都遍历出边表来决定将流量传递给哪条出边，则 u 这个局部的时间复杂度最坏可达 $O(m^2)$。
- 为避免这一缺陷，如果某一时刻我们已经知道边 (u, v) 已经增广到极限，则 u 的流量没有必要再尝试流向出边 (u, v)。
- 据此，对于每个结点 u，我们维护 u 的出边表中第一条还有必要尝试的出边。习惯上，我们称维护的这个指针为当前弧，称这个做法为当前弧优化。

dinic 算法复杂度

我不是很会证明，于是直接给出关键结论：

- 单轮 DFS 增广求阻塞流的时间复杂度是 $O(nm)$，如果不用当前弧则可能退化到 $O(m^2)$，即最终和 EK 同复杂度。
- 增广总轮数的上界是 $O(n)$。详细证明参见 OI-WiKi。

于是总复杂度 $O(n^2m)$。远优于 EK，而且通常跑不满，于是一般网络流问题不用担心常数。
同时说明了当前弧优化是对复杂度的一个优化，并不是常数优化。

特殊情况下 dinic 复杂度

称边权位 $0,1$ 的图为单位容量的。我不是很会证明，于是直接给出关键结论：

- 在单位容量的网络中，Dinic 算法的单轮增广的时间复杂度为 $O(m)$。
- 在单位容量的网络中，Dinic 算法的增广轮数是 $O(\sqrt{m})$ 的。
- 在单位容量的网络中，Dinic 算法的增广轮数是 $O(n^{2/3})$ 的。

于是单位容量的网中网络流的复杂度为：$O(m\min(\sqrt{m}, n^{2/3}))$。于是二分图匹配就可以做到这个复杂度。

dinic 算法代码实现

```
#include<bits/stdc++.h>
#define LL long long
#define fr(x) freopen(#x".in","r",stdin);freopen(#x".out","w",stdout);
using namespace std;
const int N=205,M=5e3+5;
int n,m,tot=1,head[N],_head[N],d[N],S,T;LL ans;//上面说的tot=1(++tot的第一个是2)开始
struct edge{int to,nex;LL w;}e[M<<1];
inline void add(int u,int v,LL w)
{
    e[++tot]={v,head[u],w};head[u]=tot;
    e[++tot]={u,head[v],0};head[v]=tot;
}//建图
inline bool bfs()
{
    queue<int>q;memset(d,0,sizeof(d));d[S]=1;q.push(S);
    while(!q.empty())
    {
        int t=q.front();q.pop();
        for(int i=head[t];i;i=e[i].nex)
        {
            int to=e[i].to;
            if(!d[to]&&e[i].w>0) d[to]=d[t]+1,q.push(to);
        }
    }return d[T];
}//bfs分层
```

dinic 算法代码实现

```
LL dfs(int x,LL flow)
{
    if(x==T) return flow;LL old=flow;
    for(int &i=_head[x];i;i=e[i].nex)
    //注意这行&引用就是当指针了，进行当前弧优化，去掉&就是去掉这个优化
    {
        int to=e[i].to;
        if(d[to]==d[x]+1&&e[i].w>0)
        {
            LL nw=dfs(to,min((LL)e[i].w,flow));flow-=nw;
            e[i].w-=nw,e[i^1].w+=nw;//流，反向边加流
            if(!flow) break;
        }
    }return (flow==old)&&(d[x]=0),old-flow;
    //注意如果这个点没流了就之后都不流它了，配合前面的当前弧优化使用才能保证复杂度！！！
}//dfs增广
int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);cin>>n>>m>>S>>T;
    for(int i=1,u,v,w;i<=m;i++) cin>>u>>v>>w,add(u,v,w);
    while(bfs()) memcpy(_head,head,sizeof(head)),ans+=dfs(S,1e18);cout<<ans;
    //memcpy是为了方便指针扫一遍且不影响，而后跑dinic
    return 0;
}
```

例题

例 (洛谷 P3386/uoj #78)

求二分图最大匹配。$1 \leq n_0, n_1 \leq 500, m \leq 2.5 \times 10^5$。

例题

例 (洛谷 P3386/uoj #78)

求二分图最大匹配。$1 \le n_0, n_1 \le 500, m \le 2.5 \times 10^5$。

- 建立源点 S，汇点 T。把 S 向所有左侧点连边（或者说所有男生），同理 T 向所有右侧点连边。
- 左右（男女）之间的连边依然在网络中保留。
- 跑 $S \to T$ 的最大流就是答案。
- 好好理解一下是不难的。
- 代码。

问题回顾

- 对于网络 $G=(V,E)$，如果 $\{S,T\}$ 是 V 的划分（即 $S\cup T=V$ 且 $S\cap T=\varnothing$），且满足 $s\in S,t\in T$，则我们称 $\{S,T\}$ 是 G 的一个 $s-t$ 割（cut）。我们定义 $s-t$ 割 $\{S,T\}$ 的容量为 $||S,T||=\sum_{u\in S}\sum_{v\in T}c(u,v)$。
- 最小割问题：对于网络 $G=(V,E)$，找到合适的 $s-t$ 割 $\{S,T\}$，使得 $\{S,T\}$ 的容量尽可能小。此时我们称 $\{S,T\}$ 是 G 的最小割。

最大流最小割定理

最大流最小割定理：对于任意网络 $G = (V, E)$，其上的最大流 f 和最小割 $\{S, T\}$ 总是满足 $|f| = ||S, T||$。
看起来很假的样子，$\min$ 和 $\max$ 相等，其实本质上是线性规划对偶。
如果真要学这个，建议：线性规划学习，线性规划对偶证明最大流最小割定理。
下面给出一种不高深的证明：

最大流最小割定理证明

- 显然有 $|f| \geq ||S, T||$。因为最大流其实是阻塞流（或称为阻塞流的并），而且一条边只有流满/空流两种情况。
- 于是把最大流流满的边全割了一定是一个割，于是不小于最小割。

最大流最小割定理证明

另一方面，

$$
\begin{aligned}
|f| = f(s) &= \sum_{u\in S} f(u) = \sum_{u\in S}\left(\sum_{v\in V} f(u,v) - \sum_{v\in V} f(v,u)\right) \\
&= \sum_{u\in S}\left(\sum_{v\in T} f(u,v) + \sum_{v\in S} f(u,v) - \sum_{v\in T} f(v,u) - \sum_{v\in S} f(v,u)\right) \\
&= \sum_{u\in S}\left(\sum_{v\in T} f(u,v) - \sum_{v\in T} f(v,u)\right) + \sum_{u\in S}\sum_{v\in S} f(u,v) - \sum_{u\in S}\sum_{v\in S} f(v,u) \\
&= \sum_{u\in S}\left(\sum_{v\in T} f(u,v) - \sum_{v\in T} f(v,u)\right) \\
&\le \sum_{u\in S}\sum_{v\in T} f(u,v) \le \sum_{u\in S}\sum_{v\in T} c(u,v) = ||S,T||
\end{aligned}
$$

为了取等，第一个 $\le$ 需要 $\{(u,v) \mid u\in T, v\in S\}$ 的所有边均空流，第二个 $\le$ 需要 $\{(v,u) \mid u\in S, v\in T\}$ 的所有边均满流。于是就相等了。

例题

例 (洛谷 P2774)

有一个 m 行 n 列的方格图，每个方格中都有一个正整数。现要从方格中取数，使任意两个数所在方格没有公共边，且取出的数的总和最大，请求出最大的和。
$1 \leq n, m \leq 100, 1 \leq a_{i,j} \leq 10^5$。

例题

例 (洛谷 P2774)

有一个 m 行 n 列的方格图，每个方格中都有一个正整数。现要从方格中取数，使任意两个数所在方格没有公共边，且取出的数的总和最大，请求出最大的和。
$1 \le n, m \le 100, 1 \le a_{i,j} \le 10^5$。

- 我们对棋盘进行黑白染色（**横纵坐标和为奇数** 的点设为黑点），可以发现，若取一个黑格的点，受到影响的就是周围的白点。
- 于是我们可以建一个二分图。
- 然后可以发现这是一个最小割的套路题，假设所有的点都取，然后去掉最小割，就是答案了。
- 建模：$S \to$ 所有黑点，容量为点权。所有白点 $\to T$，容量为点权。每一个黑点 $\to$ 取该黑点会受到影响的白点，容量为 $+\infty$。表示不可割。
- 总和减最大流就是答案。

代码（洛谷 P2774）

```
#include<bits/stdc++.h>
#define LL long long
#define W(i,j) ((i-1)*m+j)
#define fr(x) freopen(#x".in","r",stdin);freopen(#x".out","w",stdout);
using namespace std;
const int N=1e4+5,M=1e5+5,dx[]={1,-1,0,0},dy[]={0,0,1,-1};
int n,m,s,t,tot=1,head[N],_head[N],d[N],ans;
struct edge{int to,nex,w;}e[M];
//最大流模板
int main()
{
    scanf("%d%d",&n,&m);t=n*m+1;int x;
    for(int i=1;i<=n;i++) for(int j=1;j<=m;j++) scanf("%d",&x),ans+=x,(i+j)&1?add(s,W(i,j),x):add(
    for(int i=1;i<=n;i++) for(int j=(i&1)+1;j<=m;j+=2) for(int k=0;k<4;k++)
    {
        int nx=i+dx[k],ny=j+dy[k];
        if(nx<1||nx>n||ny<1||ny>m) continue;
        add(W(i,j),W(nx,ny),2e9);
    }
    while(bfs()) memcpy(_head,head,sizeof(head)),ans-=dfs(s,2e9);
    printf("%d",ans);
    return 0;
}
```

例题

例 (洛谷 P2598)

给定一个 $n \times m$ 的矩阵，矩阵上每一个点可能是狼、空地或者羊，你要在某些点的某几个边界方篱笆使得任意狼、羊不能互通。$1 \leq n, m \leq 100$。

例题

例 (洛谷 P2598)

给定一个 $n \times m$ 的矩阵，矩阵上每一个点可能是狼、空地或者羊，你要在某些点的某几个边界方篱笆使得任意狼、羊不能互通。$1 \leq n, m \leq 100$。

- 源点向所有狼连流量 $+\infty$ 的边
- 所有羊向汇点连流量 $+\infty$ 的边
- 所有点向四周连流量为 1 的边。
- 正确性：所有狼和羊之间的边都被割掉了，相当于修了栅栏，所以是对的。

问题回顾

- 最小费用最大流问题：在网络 $G = (V, E)$ 上，对每条边给定一个权值 (u, v)，称为费用（cost），含义是单位流量通过 (u, v) 所花费的代价。对于 G 所有可能的最大流，我们称其中总费用最小的一者为最小费用最大流。
- **注意：要在先满足最大流的情况下，最小化费用。**
- 模板题：洛谷 P3381。

SSP 算法

- SSP（Successive Shortest Path）算法是一个贪心的算法。它的思路是每次寻找单位费用最小的增广路进行增广，直到图上不存在增广路为止。
- 如果图上存在单位费用为负的圈，SSP 算法无法正确求出该网络的最小费用最大流。此时需要先使用消圈算法消去图上的负圈。

SSP 算法证明

证明

我们考虑使用数学归纳法和反证法来证明 SSP 算法的正确性。

设流量为 i 的时候最小费用为 f_i。我们假设最初的网络上 **没有负圈**，这种情况下 $f_0 = 0$。

假设用 SSP 算法求出的 f_i 是最小费用，我们在 f_i 的基础上，找到一条最短的增广路，从而求出 f_{i+1}。这时 $f_{i+1} - f_i$ 是这条最短增广路的长度。

假设存在更小的 f_{i+1}，设它为 f'_{i+1}。因为 $f_{i+1} - f_i$ 已经是最短增广路了，所以 $f'_{i+1} - f_i$ 一定对应一个经过 **至少一个负圈** 的增广路。

这时候矛盾就出现了：既然存在一条经过至少一个负圈的增广路，那么 f_i 就不是最小费用了。因为只要给这个负圈添加流量，就可以在不增加 s 流出的流量的前提下，使 f_i 对应的费用更小。

综上，SSP 算法可以正确求出无负圈网络的最小费用最大流。

最小费用最大流

- 按照如上方法增广单位费用的最短路即可。每次用 spfa 找完带负权最短路，用 EK/dinic 增广即可。
- 这时两者复杂度一样，设该网络的最大流为 f，则复杂的为 $O(nmf)$。

都介绍完了复杂度，讲下著名的两大原则（所有网络流的题基本都满足，是为了防止毒瘤出题人的约定俗成）：

- 普通网络流不卡 dinic 原则，除了 HLPP 模板，其他正常的网络流题不卡 dinic，但是可以卡 EK。
- 费用流题除开个别模板题不卡 spfa+EK 的实现，但是下面会讲费用流的复杂度优化方法。

代码实现

```
#include<bits/stdc++.h>
#define fr(x) freopen(#x".in","r",stdin);freopen(#x".out","w",stdout);
using namespace std;
const int N=5005,M=50005;
struct edge{int to,nex,w,w1;}e[M<<1];
int n,m,S,T,tot=1,head[N],d[N],fl[N],pre[N],_pre[N],ans,ans1;bool v[N];
inline void add(int u,int v,int w,int w1){e[++tot]={v,head[u],w,w1};head[u]=tot;}
inline bool spfa()
{
    memset(d,0x3f,sizeof(d));memset(v,0,sizeof(v));
    v[S]=1;d[S]=0;fl[S]=2e9;queue<int>q;q.push(S);
    while(!q.empty())
    {
        int t=q.front();q.pop();v[t]=0;
        for(int i=head[t];i;i=e[i].nex)
        {
            int to=e[i].to;
            if(e[i].w>0&&d[to]>d[t]+e[i].w1)
            {
                d[to]=d[t]+e[i].w1;fl[to]=min(fl[t],e[i].w);
                pre[to]=t;_pre[to]=i;(!v[to])&&(q.push(to),v[to]=1);
            }
        }
    }
    return d[T]!=d[0];
}//spfa找单位费用的最短路
```

代码实现

```
int main()
{
    scanf("%d%d%d%d",&n,&m,&S,&T);int u,v,w,w1;
    while(m--) scanf("%d%d%d%d",&u,&v,&w,&w1),add(u,v,w,w1),add(v,u,0,-w1);
    while(spfa())
    {
        for(int i=T,t;i!=S;i=pre[i]) t=_pre[i],e[t].w-=fl[T],e[t^1].w+=fl[T];
        ans+=fl[T];ans1+=fl[T]*d[T];//EK算法增广,很短啊,记得反向边要减
    }
    printf("%d %d",ans,ans1);
    return 0;
}
```

费用流复杂度优化

注：大部分费用流题不需要优化，于是这可以选择不学。

- Primal-Dual 原始对偶算法：OI-WiKi，blog1，blog2，复杂度 $O(nm + m \log mf)$。
- 弱多项式复杂度做法：blog，例题。
  复杂度 $O(m^2 \log m \log U)$，U 为边的最大容量。

例题

例 (洛谷 P4015)

有 m 个仓库，有 n 个销售点，第 i 个仓库有 a_i 件货，第 j 个销售点必须销售 b_j 件货，第 i 个仓库到第 j 个销售点要 $c_{i,j}$ 元，问完成销售最少/最多要多少元?

$1 \leq n, m \leq 100$，所有数都在 int 范围内。

例题

例 (洛谷 P4015)

有 m 个仓库，有 n 个销售点，第 i 个仓库有 a_i 件货，第 j 个销售点必须销售 b_j 件货，第 i 个仓库到第 j 个销售点要 $c_{i,j}$ 元，问完成销售最少/最多要多少元?

$1 \leq n, m \leq 100$，所有数都在 int 范围内。

- 源点向每一个仓库连接一条流量为 a_i，费用为 0 的边。
- 仓库 i 向销售点 j 连接一条流量为无穷大，费用为 $c_{i,j}$。
- 销售点 j 向汇点连接一条流量为 b_j，费用为 0 的边
- 跑一个最小费用最大流，轻松完成第一个任务。
- 第二个任务只要清空仓库销售点之间的边，然后重新连接为费用是 $-c_{i,j}$ 的边即可。
- 第二问最后跑个最小费用最大流，然后取反即可。

例题

例 (洛谷 P2045)

给出一个 $n \times n$ 的矩阵，每一格有一个非负整数 $A_{i,j}$，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。
$0 \leq A_{i,j} \leq 10^3, 1 \leq n \leq 50, 0 \leq K \leq 10$。

例题

例 (洛谷 P2045)

给出一个 $n \times n$ 的矩阵，每一格有一个非负整数 $A_{i,j}$，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。
$0 \le A_{i,j} \le 10^3, 1 \le n \le 50, 0 \le K \le 10$。

- 点边转化：把每个格子 (i,j) 拆成一个入点一个出点。
- 从每个入点向对应的出点连两条有向边：一条容量为 1 ，费用为格子 (i,j) 中的数；另一条容量为 $k-1$，费用为 0 。
- 从 (i,j) 的出点到 $(i+1,j)$ 和 $(i,j+1)$ 的入点连有向边，容量为 k，费用为 0。
- 以 $(1,1)$ 的入点为源点，(n,n) 的出点为汇点，求最大费用最大流。

题单：

- 题单 1，到 洛谷 P4313 前建议板刷，后面选自己合适的做。
- 题单 2，题单 3，题单 4，没啥特别要板刷的，看自己能力吧。

题目（可能与题单有重合）：

- 洛谷 P4722，预流推进。
- 洛谷 P2764，洛谷 P2766，两个经典问题。
- 洛谷 P4001，奇妙 trick。神必题。CF1404E，难的思维题。洛谷 P5039，神秘思维题。
- 洛谷 P1231，拆点。
- 最大权闭合子图相关：洛谷 P4313，CF1082G/洛谷 P4174，CF1615H（一般图保序回归问题）

题单：题单，没啥特别要板刷的，看这比较经典/新奇的题目名称可以刷刷，积攒套路。题目（可能与题单有重合）：

- 洛谷 P7730，中等。
- 洛谷 SP371，中等。
- 洛谷 P4003，困难。