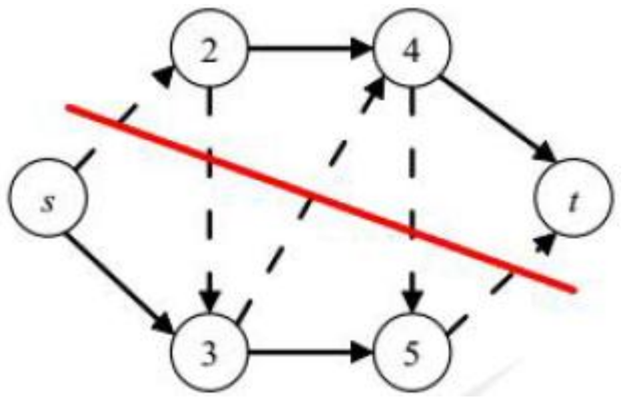
最小割问题

石室中学：梁德伟

最小割的定义



在网络G=(V,E)中，对于割[S,T]将点集V分为S,T(V-S)两部分，使得源s∈S,汇t ∈T。割[S,T]代表一个边的集合{<u,v>|<u,v> ∈E , u ∈S, v ∈T}。穿过该割的净流设为F(S,T)，割的容量记为C[S,T]

一个网络中最小割也就是该网络中容量最小的割。

如上图，红线下的点构成点集S，其他点构成点集T，则割[S,T]={<1,2>,<3,4>,<5,6>},割[T,S]={<2,3>,<4,5>}

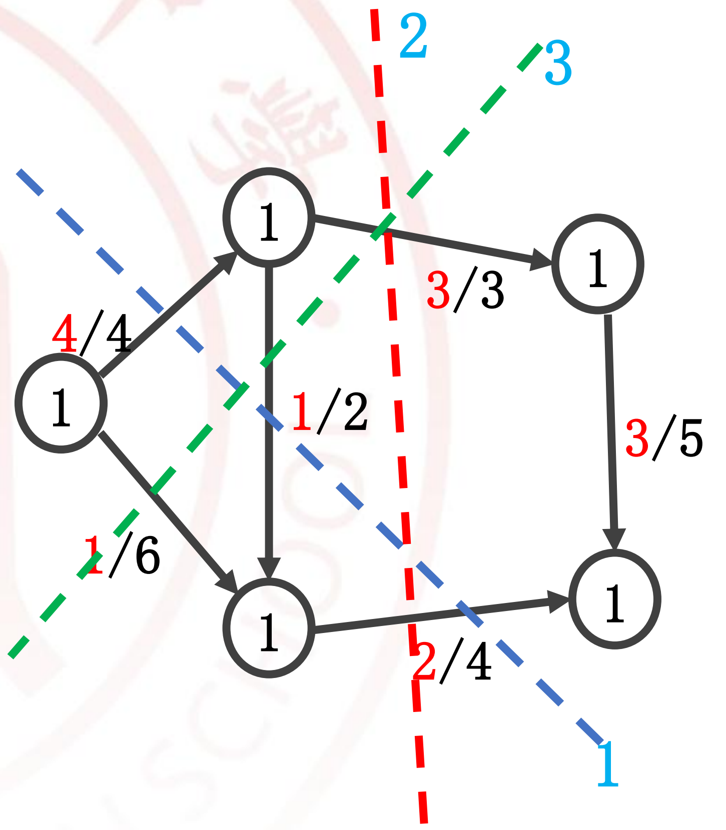
注意：边<2,3>,<4,5>的流量要算在割[S,T]的净流F[S,T]中，当然可能是负值。但他们的容量不能算在C[S,T]中

割与流的关系

网络流量：5

割：

割	正	反
1	6	1
2	5	0
3	5	0



割与流的关系　　引理1

- 在一个流网络G=(V, E)中，设其任意一个流为F，且[S, T]为G的一个割。则通过割的净流F(S, T)=|f|

证明：

$$
\begin{aligned}
f(S,T) &= f(S,V) - f(S,S) && \text{根据引理(1.1)(3)} \\
&= f(S,V) && \text{根据引理(1.1)(1)} \\
&= f(s,V) + f(S-\{s\},V) && \text{根据引理(1.1)(3)} \\
&= f(s,V) = |f| && \text{根据流守恒性，} f(S-\{s\},V)=0
\end{aligned}
$$

通俗理解：图的任意一个S, T割净流就是流量

割与流的关系　推论：对偶问题性质

- 在一个流网络G=(V, E)中，设其任意一个流为F，任意一个割为[S, T]，必有|f|<=C[S, T]

证明： 由引理 1.3 和容量限制，有

$$|f| = f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) \le \sum_{u \in S} \sum_{v \in T} c(u,v) = c[S,T]$$

通俗理解：网络中最大流必定不超过此网络流最小割的容量

定理：最大流最小割定理

- 如果f是具有源s和汇t的流网络G=(V,E)中的一个流，则下列条件是等价的：

  1、f是G的一个最大流

  2、残余网络G_f不包含增广路

  3、对G的某个割[S,T]，有|f|=c[S,T]

通俗理解：最小割=最大流

证明

设F是网络G=(V,E)的最大流

1、先证明F是G的割：f是G的最大流，根据最大流原理，G_f的残量网络中不存在S->T的增广路，即S,T分属两个连通分量=>f是G的割

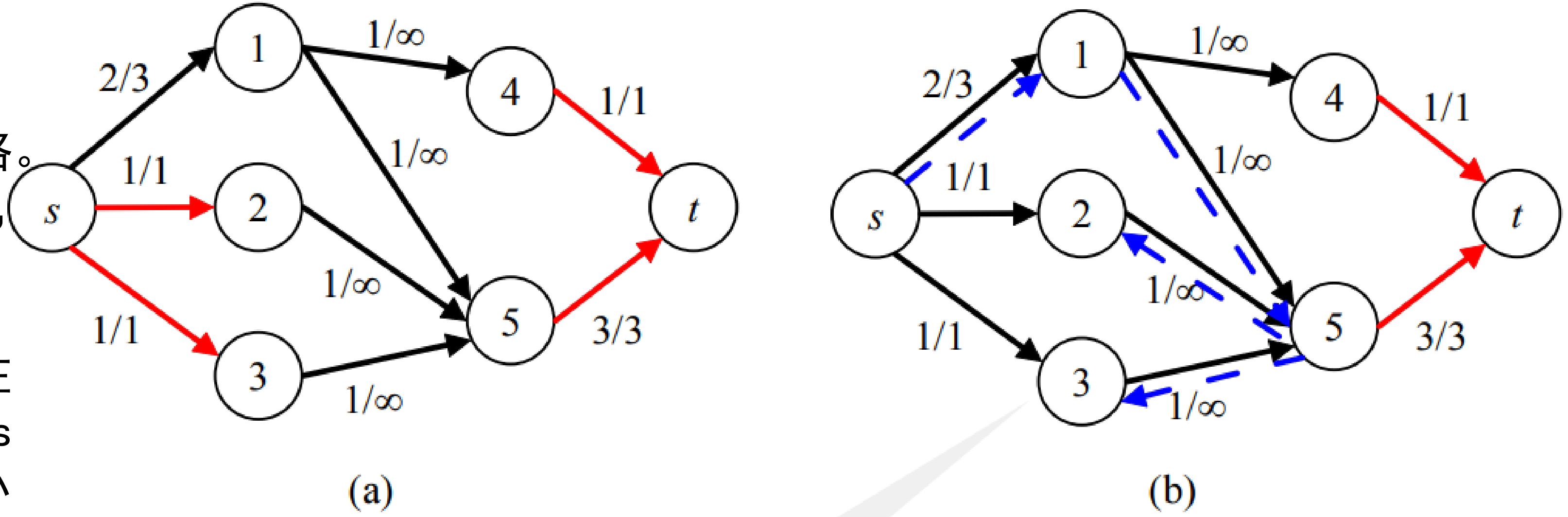
2、再证明f是G的最小割：反证法。设|f1|<|f|,且f1是G的割，由于|f1|<|f|，即在残量网络G_f1中有S->T的增广路，即G_f1的S,T不属于两个连通分量，与假设的f1是G的割不符。得证。

求割集

先求最大流。在得到最大流f后的残量网络G_f中，从s开始DFS，所有能遍历到的点构成点集S。没有搜索到的构成点集T，两集合间的边构成最小割边集。

注意：虽然 最小割[S,T]的边都是满流边，但是**满流边不一定是最小割边集**。如下面的二分图的例子

图(a)给出了一个基于二分图构造的流网络。由于从X部到Y部都是容量均为正无限的边，都不可能是最小割中的边，有人便会错误地认为与源或汇关联的满流边便组成了最小割（图(a)的红色边）。然而实际上，在该网络的残留网络中，结点2与3应该与源s是连通的（图(b)的蓝色路径），所以最小割应该是图(b)中的红色边。



Poj3204 求最小割边集

- 一个由n个点，m条边构 成的有向图，每条边都有一定的流量。现在求存在多少条边，在增加这些边的流量后从1点到n的总流量会增加。

```
int f1[maxn],f2[maxn];
void dfs(int u,int f[],int d){
    f[u]=1  ;
    for(int i=first[u];i;i=e[i].nxt){
        int v=e[i].v;
        if(e[i^d].w&&!f[v])dfs(v,f,d);
    }
}
```

```
int main(){
    n=in;m=in;
    for(int i=1;i<=m;i++){
        int u=in,v=in,w=in;
        ins(u,v,w);
    }
    D::dinic(0,n-1);
    dfs(0,f1,0);//源点出发走正向边
    dfs(n-1,f2,1);//汇点出发走反向边
    int ans=0;
    for(int i=2;i<=cnt;i+=2)
        if(f1[e[i].u]&&f2[e[i].v] && e[i].w==0)ans++;
    cout<<ans;
    return 0;
}
```

2、判断最小割是否唯一

方法：先计算最大流。在残量网络中分别对S,T做一次DFS，如果能搜索到所有的点，那么最小割唯一，否则不唯一。

解读：最大流后，割边一定满流，减少一条割边后，网络流减少。

如：1、(图1)正向搜索集合为S，反向搜索集合为T，cnt1和cnt2都是最小割边。很显然M还存在着最小割边，因为从M到T的残余网络也没有流向T的容量了。

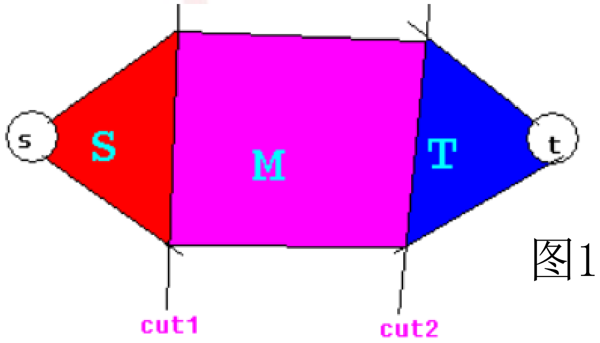


图1

2、(图2)增加E1这部分边的容量将直接导致网络最大流量增加。

增加E2和E3则不然。同样M中存在并上E1后为割边的边集。

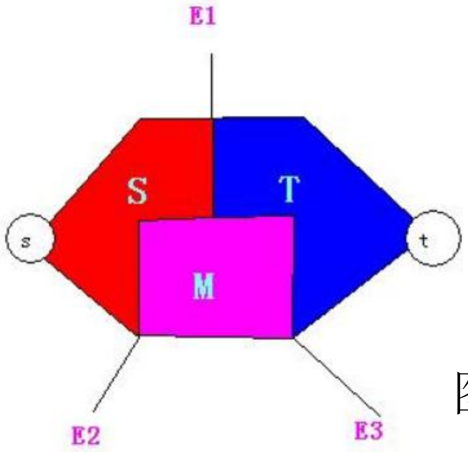


图2

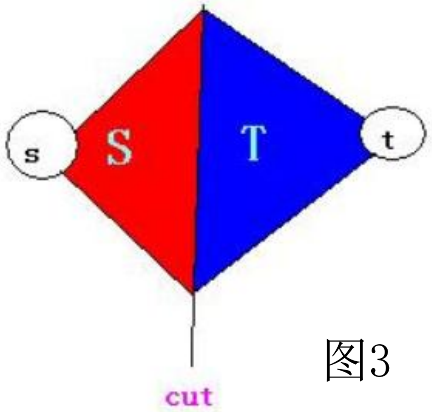


图3

3、两条割边完全重合为一条，此时网络中的割边唯一。

zoj2587判断最小割是否唯一

给定一个无向图的源点、汇点，要求判定分离两个点的最小割是否唯一。

在求出最大流的后，在残余网络中，从源点进行一次搜索，搜索按照未饱和的边进行，得到顶点子集S的顶点个数；再从汇点反向搜索未饱和的边，得到子集T的顶点个数，判定顶点数相加是否等于总共的顶点数。

如果能到所有顶点，则是唯一的，否则不是唯一

3、求字典序最小的最小割集
1791 奶牛的电信

对于第一个问，显然的一个拆点技巧，将一个点i拆分为2个点i,i’,i->i’,容量为1原图的边<u,v>，容量设为inf，因为每个点拆分为2个点，所以建边<u’ ,v ,inf>,<v’ , u ,inf>然后求出最大流即最小割

对于第二问，因为最大流后求出的最小割是任意最小割。所以在这里，我们可以采取依次枚举每个点，将它拆开后的边容量变为0，然后跑最大流，看最大流是否改变，如果改变了则输出。

```
int main(){
	n=in;m=in;S=in;T=in;
	for(int i=1;i<=n;i++)
		ins(i,i+n,1);//拆点
	for(int i=1;i<=m;i++){
		int u=in,v=in ;
		ins(u+n,v,inf);ins(v+n,u,inf);
	}
	memcpy(ee,e,sizeof(e));
	S+=n;
	int ans=D::dinic(S,T);
	printf("%d\n",ans);
	for(int i=2;i<=2*n;i+=2){
		ee[i].w=0;
		memcpy(e,ee,sizeof(e));
		int tmp=D::dinic(S,T);
		if(tmp==ans) //如果最大流没有改变，则不是割边
			ee[i].w=1;
		else
			printf("%d ",i/2),ans=tmp;//最大流改变，这条边就是在割集中
	}
	return 0;
}
```

4、与边数有关的最小割1673

求最小割前提下的最少边数

方法：

首先要满足第一个条件是最小割，再次满足是边数最少。

对每条边进行改造，在原来基础上*M+1，即ci’=ci*M+1

M取足够大（>=SUM(Ci)）

这样最大流maxflow=Cmin*M+最少割边数

原来最大流=maxflow/M

最小割边数=maxflow%M

二分图的最小点权覆盖集

- 在带点权无向图G中，点权之和最小的点覆盖集。

二分图最大匹配转为最大流

- 加入了额外的源和汇t，将匹配以一条条s-u-v-t形式的流路径“串联”起来。匹配的限制是在点上，恰当地利用了流的容量限制。

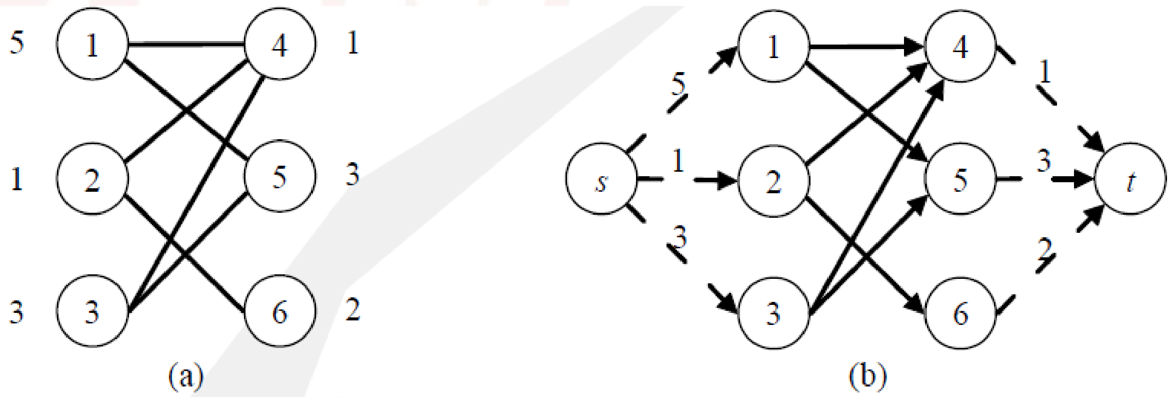
- 点覆盖集的限制在边上，最小割是最大流的对偶问题，对偶往往是将问题的性质从点转边，从边转点。可以尝试着转化到最小割模型。
- 建图方式：建立一个源s，向X部每个点连边。建立汇t，Y部每个点向t连边，边权都为1。原图的边<u,v>边权为inf。

二分图的最小点权覆盖集

- 回到问题，考虑刚才的建图方式。
- 割的性质是**不存在一条从s到t的路径。**故3条边<s, u>, <u, v>, <v, t>中至少一条边在割中，设<u, v>边权为inf，则<s, u>, <v, t>至少有一条边在割中，正好与点覆盖集限制条件的形式符合。目标是最小化点权之和，也恰好是最小割的优化目标
- 建图方式：$V_N = V \cup \{s,t\}$

$$E_N = E \cup \{\langle s,u \rangle \mid u \in X\} \cup \{\langle v,t \rangle \mid v \in Y\}$$

$$\begin{cases} c(u,v) = \infty & \langle u,v \rangle \in E \\ c(s,u) = w_u & u \in X \\ c(v,t) = w_v & v \in Y \end{cases}$$



二分图的最小点权覆盖集算法的例子

二分图的最大点权独立集2625

- （覆盖集与独立集互补定理）若$\overline{V'}$为不含孤立点的任意图的一个点覆盖集当且仅当V’是该图的一个点独立集

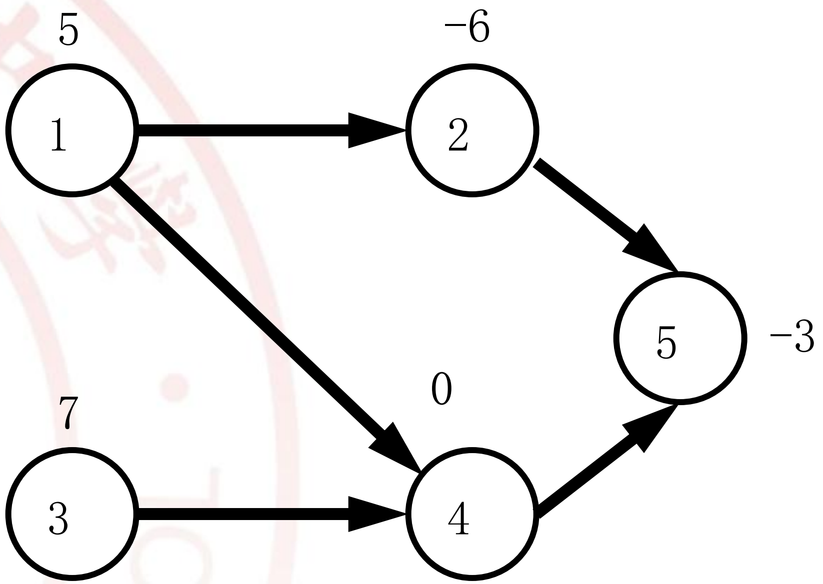
```
inline int idx(int x,int y){
    return (x-1)*n+y ;
}
int main(){
    m=in;n=in;S=0,T=m*n+1;
    int sum=0;
    for(int i=1;i<=m;i++)
    for(int j=1;j<=n;j++){
        int x=in;
        sum+=x;
        if((i+j)&1){
            ins(S,idx(i,j),x);
            for(int k=0;k<4;k++){
                int tx=i+dx[k],ty=j+dy[k];
                if(tx<1 || ty<1 || tx>m || ty>n)continue;
                ins(idx(i,j),idx(tx,ty),inf);
            }
        }
        else
            ins(idx(i,j),T,x);
    }
    cout<<sum-D::dinic(S,T);
    return 0;
}
```

最大权闭合图

定义：

- 有向图的**闭合图**（closure）：
  闭合图内任意点的任意后继也一定还在闭合图中。
  - 物理意义
    事物间依赖关系：一个事件要发生，它需要的所有**前提**也都一定要发生。

- **最大权闭合图**是点权之和最大的闭合图。

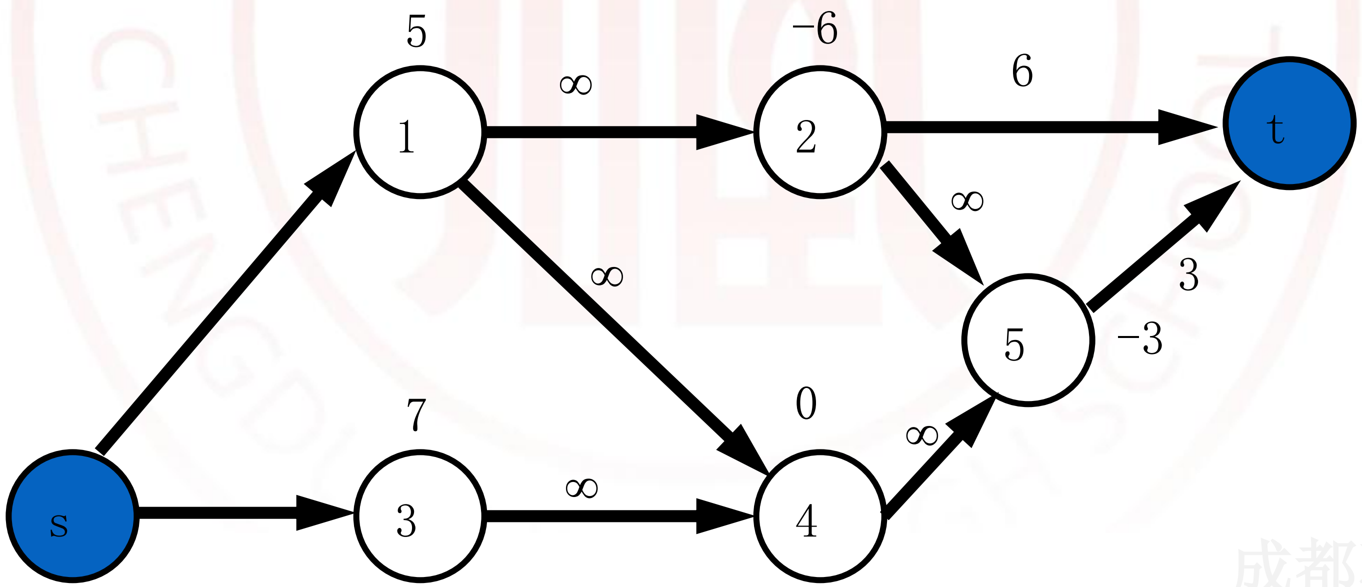


其中，{3, 4, 5}是一个闭合图。3的后继是4，4的后继是5，都在闭合图中。

而{1, 4, 5}不是一个闭合图，因为点1的后继2不在闭合图中。

最大权闭合图
构图

1. 增加源s汇t
2. 源s连接原图的正权点，容量为相应点权
3. 原图的负权点连接汇t，容量为相应点权的相反数
4. 原图边的容量为正无限.



最大权闭合图
解决

- 闭合图方案*V'*与不含正无限容量的割[*S*, *T*]一一对应

$$S = V' \cup \{s\}$$

- 闭合图*V'*的权为正权点总和减去对应割的容量

$$w(V_1) = \sum_{v \in V^+} w_v - c[S,T]$$

- 割[*S*, *T*]取最小时，闭合图权取最大。

复杂度为 $O(\mathrm{MaxFlow}(n, n+m))$