最大流算法及其应用

南京外国语学校 贾志鹏

【关键词】网络流、最大流问题、最小割问题

【摘要】本文介绍了一种特殊的图——流网络及其相关问题，主要介绍的是最大流问题和最小割问题，并提出了解决方案。最后介绍了一些网络流相关的建模问题。

【目录】

一、网络流相关的一些概念

（1）流网络
（2）流
（3）割
（4）残留网络
（5）增广路径
（6）增广

二、最大流和最小割问题

（1）最大流问题
（2）最小割问题
（3）最大流算法

三、最大流算法的应用

（1）最大流模型
（2）最小割模型

四、总结

一、网络流相关的一些概念

1. 流网络（Flow Network）

流网络 $G=(V,E)$ 是一个有向图，其中每条边 $(u,v)\in E$ 均有一非负容量 $c(u,v)\geq 0$。如果 $(u,v)\notin E$，则假定 $c(u,v)=0$。流网络中有两个特别的顶点：**源点 *s*** 和**汇点 *t***。

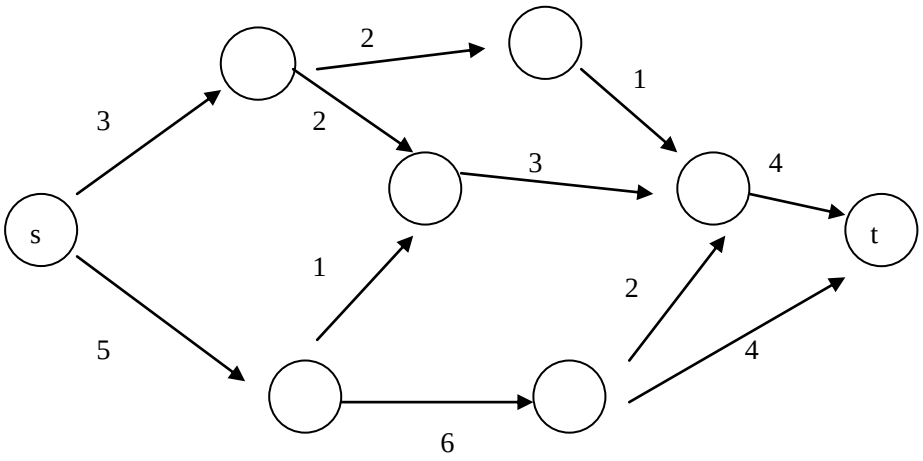

图 1　一个流网络的例子（边上的数字表示该弧的容量）

2. 流（Flow）

G 的流是一个实值函数 f，$f(u,v)$ 表示顶点 u 到顶点 v 的流，它可以为正，为零，也可以为负，且满足下列三个性质：

容量限制：对所有 $u,v\in V$，要求 $f(u,v)\leq c(u,v)$。

反对称性：对所有 $u,v\in V$，要求 $f(u,v)=-f(v,u)$。

流守恒性：对所有 $u\in V-\{s,t\}$，要求 $\sum_{v\in V} f(u,v)=0$。

整个流网络 G 的流量 $|f|=\sum_{v\in V} f(s,v)$ 或 $|f|=\sum_{u\in V} f(u,t)$。

3. 割（Cut）

流网络 $G=(V,E)$ 的割 (S,T) 将 V 划分成 S 和 $T=V-S$ 两部分，使得 $s\in S$，$t\in T$。定义割 (S,T) 的容量为 $c(S,T)$，则：

$$c(S,T)=\sum_{u\in S,v\in T}c(u,v)$$

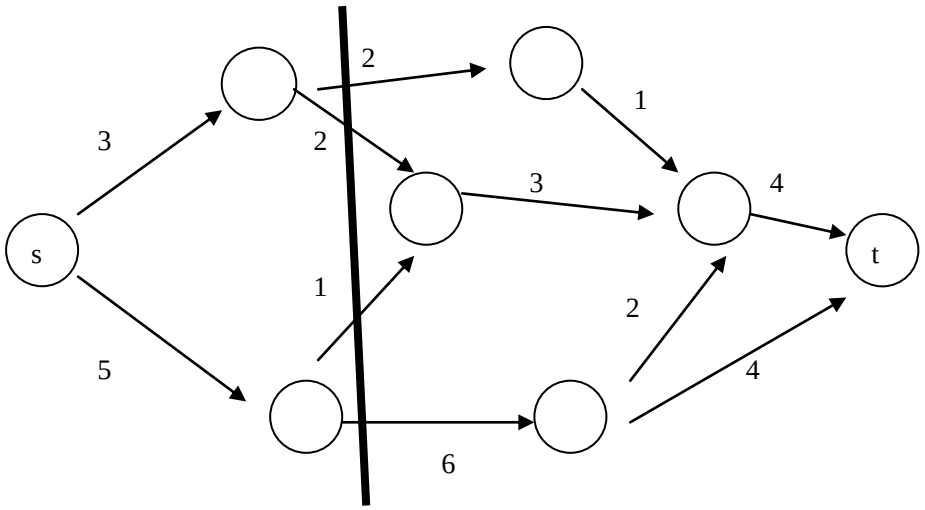

图 2　一个割的例子（边上的数字表示该弧的容量，割的容量即为 2+2+1+6=11）

4. 残留网络（Residual Network）

给定一个流网络 $G=(V,E)$ 和流 f，由 f 压得的 G 的残留网络 $G_f=(V,E_f)$，定义 $c_f(u,v)$ 为残留网络 G_f 中边 (u,v) 的容量。如果弧 $(u,v)\in E$ 或弧 $(v,u)\in E$，则弧 $(u,v)\in E_f$，且 $c_f(u,v)=c(u,v)-f(u,v)$。在下面的各种概念和方法中，我们只考虑残留网络中容量大于 0 的弧，但是编程时为了方便还是保留了。

5. 增广路径（Augmenting Path）

对于残留网络 G_f 中的一条 *s-t* 路径 p 称其为增广路径，定义增广路径 p 的残留容量为 p 上弧容量的最小值。后面求最大流要用到增广路径这个概念。

6. 增广（Augment）

对于残留网络 G_f 中的一条增广路径 p，增广的意思就是对于每一条属于 p 的弧 (u,v)，将 $f(u,v)$ 增加 p 的残留容量，将 $f(v,u)$ 减少 p 的残留容量。这样的话，新的流 f 仍然满足流的三条性质，并且原流网络的流量 $|f|$ 增加了。一般来说，增广过后就会修改残留网络，易得对于每一条属于 p 的弧 (u,v)，要将 $c_f(u,v)$

减去 p 的残留容量，$c_f(v,u)$ 加上 p 的残留容量。（程序实现基本都是通过直接修改残留网络来实现增广的）

二、最大流和最小割问题

1. 最大流问题

对于一个流网络 $G=(V,E)$，其流量 $|f|$ 的最大值称为最大流，最大流问题就是求一个流网络的最大流。

增广路定理：当且仅当由当前的流 f 压得的残留网络 G_f 中不存在增广路径时，流 f 的流量 $|f|$ 达到最大。（证明在此略去，可以参见相关书籍）

根据增广路定理，我们可以设计出最基本的求最大流的方法，一开始将流网络 $G=(V,E)$ 的流 f 置为零流，即对于 $(u,v)\in E$ 时，$f(u,v)=0$。然后构建残留网络，寻找增广路径增广，再修改残留网络，重复此过程，直到无法找到增广路径。此方法（之所以不是算法，是因为实现方法很多）称为 *Ford-Fulkerson* 方法。

伪代码如下：

```
FORD-FULKERSON-METHORD (G, s, t)
1   initialize flow f to 0
2   while there exists an augmenting path p
3       do augment flow f along p
4   return f
```

2. 最小割问题

最小割是指流网络中容量最小的割。

***Ford-Fulkerson* 定理（最小割最大流定理）**：在流网络中，最小割的容量等于最大流的流量。（证明也在此略去）

根据这个定理，我们就可以通过求流网络的最大流来得到最小割。

3. 最大流算法

前面所讲的只是求最大流的一种方法，但怎样高效地实现还是一个问题。

这个方法的最大问题就在于怎样快速地找到一条增广路径。当然我们可以用最基本的搜索（DFS 或 BFS），但是这种方法肯定不够高效，这时我们就需要更高效的算法。

本文将重点介绍一种高效且实用的最大流算法：**SAP 算法**（最短增广路算法）。

最短增广路算法(Shortest Augmenting Path Algorithm)，即每次寻找包含弧的个数最少的增广路进行增广，可以证明，此算法最多只需要进行 $\frac{|V||E|}{2}$ 次

增广。并且引入距离标号的概念，可以在$O(|V|)$的时间里找到一条最短增广路。最终的时间复杂度为$O(|V|^2|E|)$，但在实践中，时间复杂度远远小于理论值（特别是加了优化之后），因此还是很实用的。

距离标号：对于每个顶点 u 赋予一个非负整数值 $d(u)$ 来描述 u 到 t 的“距离”远近，称它为距离标号，并且满足以下两个条件：

（1）$d(t)=0$。

（2）对于残留网络 G_f 中的一条弧 (u,v)，$d(u)\le d(v)+1$。

如果残留网络 G_f 中的一条弧 (u,v) 满足 $d(u)=d(v)+1$，我们称 (u,v) 是允许弧，由允许弧组成的一条 *s*-*t* 路径是允许路。显然，允许路是残留网络 G_f 中的一条最短增广路。当找不到允许路的时候，我们需要修改某些点的 d 值。

实现中我们用一个 DFS 实现这个寻找最短增广的过程，如果无法继续向前，那么我们就修改当前结点的距离标号。

算法伪代码如下（此程序用非递归实现 DFS）：

```
SAP-ALGORITHM (G, s, t)
1   initialize flow f to 0
2   do BFS to calculate d of each vertex
    // In fact, we can initialize d to 0.
3   i ← s
4   while d(s) < n      // n is the amount of vertexes in G
5       if there exists j that (i,j) ∈E_f and d(i)=d(j)+1
6       then do add vertex i into augmenting path p
7            i ← j
8            if  i=t
9            then do augment flow f along p
10                i ← s
11      else if there exists j that (i,j) ∈E_f
12           then d(i) ← min{ d(j)+1 | (i,j) ∈E_f }
13           else d(i) ← n
14           if i ≠s then do delete i in augmenting path p
15                           i ← last vertex in p
16  return f
```

SAP 算法有两个重要的优化：**Gap 优化**和**当前弧优化**。

Gap 优化：我们可以注意到由于残留网络的修改只会使 $d(u)$ 越来越大（因为修改前 $d(u)<d(v)+1$，而修改后会存在 $d(u)=d(v)+1$，因此 $d(u)$ 变大了），所以说 d 函数是单调递增的，这就提示我们，如果 d 函数出现了“断层”，即没有

$d(u)=k$，而有 $d(v)=k\pm1$，这时候必定无法再找到增广路径。我们可以这么想，现在的 i 满足 $d(u)=k+1$，发现没有一个 $d(v)$ 为 k，因此就会尝试去调整 $d(u)$，但是 $d(u)$ 是单调递增的，只会越来越大，所以 k 这个空缺便永远不会被补上，也就是说无法再找到增广路径。

所以我们加一个 *count* 数组，统计一下 d 函数的每个值各有多少个，如果某次修改后出现了 *count[k]=0*，则说明出现了“断层”，可以直接结束程序。

当前弧优化：可以注意到一个事实：如果说在某次迭代中从 i 出发的弧 (u,v) 不是允许弧，则在顶点 i 的标号修改之前 (u,v) 都不可能是允许弧。（因为 $d(u)$ 不变，$d(v)$ 单调不降且 $d(u)<d(v)+1$）这样，在查找允许弧的时候只需要从上一次找到的允许弧开始找。所以我们增加“当前弧”这个数据结构，记录当前顶点找到的允许弧，只有在修改这个顶点标号时才会更改这个顶点的当前弧。

加了以上两个优化，SAP 的效率是十分高的（可以见后面的测试），并且需要增加的代码量都很少，也没有增加什么额外的判断时间，因此是十分有用的。

三、最大流算法的应用

在现在的信息学竞赛中，一般不会考裸的最大流问题，而是把问题转化为最大流或最小割模型，这使得题目有了难度。

1. 最大流模型

一个典型的最大流模型就是二分图的最大二分匹配。

二分图 $G=(X,Y,E)$，其中 X 和 Y 是两个不相交的点集，并且对于每对 $(u,v)\in E$，$u\in X$ 且 $v\in Y$。二分图的最大二分匹配问题就是从 E 中选择一些边，使得每个点最多在选择的边里出现一次，问最多能选多少条边。

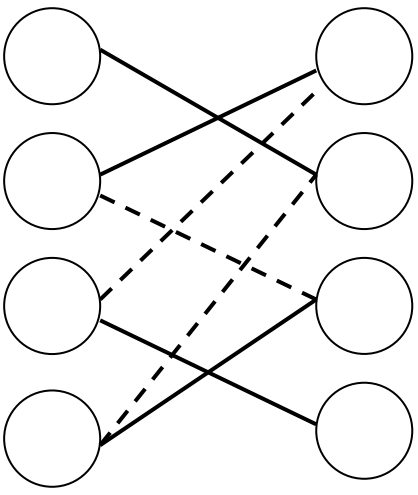

图 3　一个二分图的例子及其最大匹配（实线表示选中的边，虚线表示未选中的边）

由于每个点只能在选择的边里出现一次，就此我们可以联想到了流的容量限

制 $f(u,v) \le c(u,v)$，因此我们控制经过每个点的流量不超过 1 即可保证每个点只被使用一次。这样我们就得到了一个思路：增加源点和汇点，从源点到左边的每个点连边，从右边的每个点到汇点连边，容量都为 1。两边点中间的边保持不变，只是改成有向边，从左边指向右边。

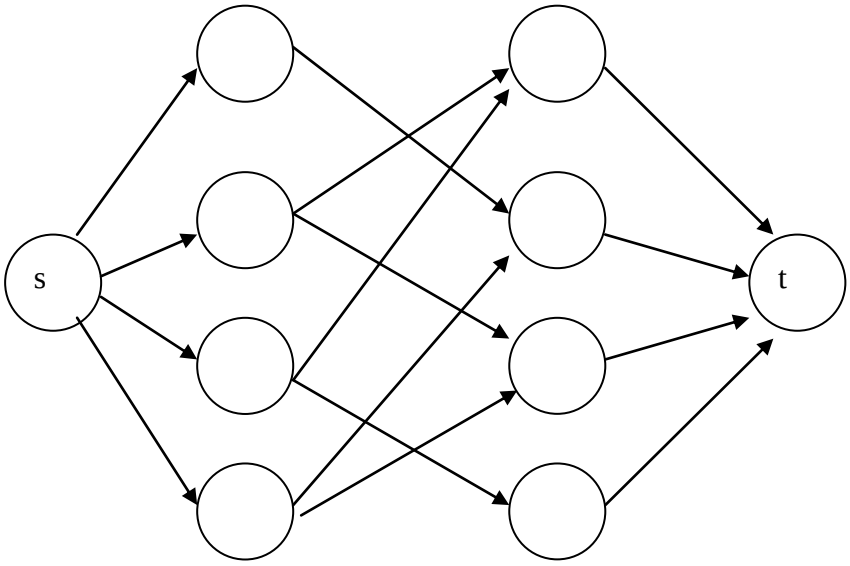

图 4　新建的流网络（图中弧的容量均为 1）

对于每个左边的点，进去的流量最多只有 1；对于每个右边的点，出去的流量最多只有 1，所以每个点最多在选中的边里最多出现一次（选中的边即为中间流量为 1 的弧）。又因为流最大，所以结果就是原二分图的最大二分匹配。

关于二分图的最大二分匹配还有另外一种算法：匈牙利算法，由于其可以更简单地实现，所以在竞赛中往往不使用最大流，但这种建模方式值得借鉴。

再来看一道例题：POJ 3281 Dining (USACO 2007 Open Gold)

题目意思比较简单，就是说现在有 N 只奶牛，F 种食物和 D 种饮料，每只奶牛喜欢其中的一些食物和饮料。现在每种食物和饮料只能分给一只奶牛，每只奶牛也只能吃一种食物和一种饮料，问最多能使多少奶牛既吃到食物又喝到饮料。

这个题和二分图匹配有相似之处，但又不完全相同，我们可以沿着二分图匹配的建模方式继续思考。

由于有 N 只奶牛、F 种食物和 D 种饮料，因此我们可以将这些东西抽象成图中的点。为了方便，我们将食物放在左边，奶牛放在中间，饮料放在右边。沿用前面的建模方式，由于食物和饮料的使用限制，我们从源点向每种食物连一条边，从每种饮料向汇点连一条边，容量都为 1。而每只奶牛都有喜欢的食物和饮料，因此将每只奶牛喜欢的食物连到这只奶牛，从这只奶牛连到每种它喜欢的饮料。

但这样是否就对了呢？实际还是有问题的，因为经过每只奶牛的食物可能超过一种，这就意味着每只奶牛可能会吃超过一组的食物和饮料，而这在题目中是不允许的。

怎么解决这个问题呢？我们又回到了流的基本性质：容量限制 $f(u,v) \le c(u,v)$。因此我们将每只奶牛拆成两个点，同一只奶牛的两个点之间连边，容量为 1。这样我们就能保证通过每只奶牛的流量为 1 了。

每个流对应每种方案，最大流即为最佳方案。

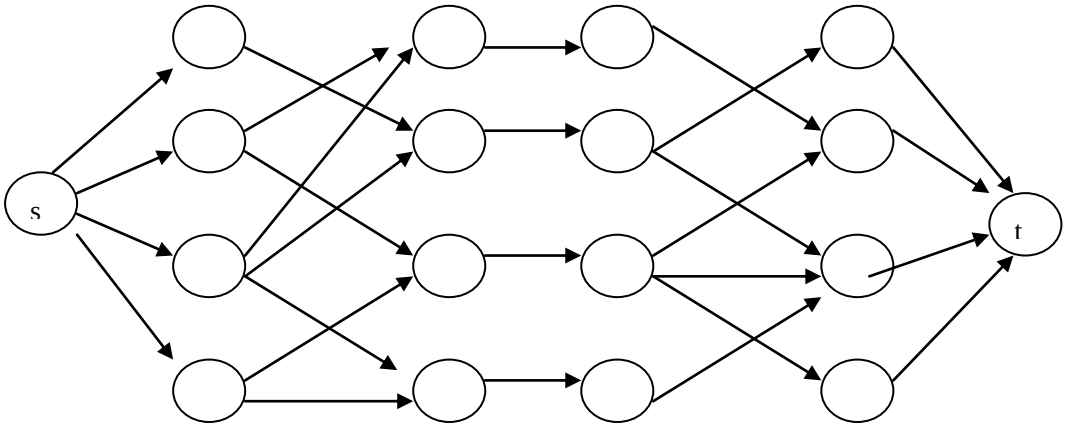

图 5　一个例子（图中弧的容量均为 1）

可见最大流模型的一般建模思路是运用流的容量限制，使得题目中的约束得以满足，有时还需使用一些特殊的方法（如上题中的拆点）来满足题目的特别约束。

2. 最小割模型

最小割问题是网络流建模里的一个难点，由于最小割模型在原问题中往往很隐蔽，有时确实需要凭感觉。

看一道比较有难度的例题《最大获利》（NOI2006 第二试）：

【题目简述】

现在有 N 个中转站和 M 个用户群，建设每一个中转站需要投入一定的成本。对于每个用户群，如果其所要求的两个中转站都被建立，那么即可得到一定的获利。问最大的获利为多少。（净获利 = 获益之和 - 投入成本之和）

【输入格式】

输入文件中第一行有两个正整数 N 和 M 。第二行中有 N 个整数描述每一个通讯中转站的建立成本，依次为 P1, P2, …, PN 。以下 M 行，第(i + 2)行的三个数 Ai, Bi 和 Ci 描述第 i 个用户群的信息。（Ai 和 Bi 表示要求用户群的编号，Ci 表示获利）

【输出格式】

一个整数，表示最大获利。

【输入样例】

5 5
1 2 3 4 5
1 2 3
2 3 4
1 3 3
1 4 2
4 5 3

【输出样例】

4

【数据范围】

N≤5000, M≤50000

看到了“最大”这个字眼，很多人便以为此题与最小割没有关系，但实际上不是。由于：

净获利 = 获益之和 - 投入成本之和

= 所有用户群的获益 -（损失用户群的获益 + 建设中转站的代价）

要使净获利最大，即使（损失用户群的获益 + 建设中转站的代价）最小，这就将“最大”变成了“最小”。

最小的出现为最小割提供了先决条件，同是割是将顶点分成两个集合，也能表示出某个中转站的建设与否。但是怎样满足题目中一个用户群要想获利就必须建设其要求的中转站这个约束条件呢？这时候我们就要运用最小割容量最小的性质。因为容量最小，因此得出结论：容量为∞的弧不会出现在割上。运用这个神奇的性质，我们就能使题目中的约束条件得到保证。

以每一个中转站和用户群为顶点，并增加源和汇。连接源到每个中转站，容量为建设该中转站的代价；连接每个用户群到汇，容量为该用户群的收益；对于每个用户群，连接它的两个相关中转站到这个用户群，容量为∞。

设最小割分得的两个集合是 S 和 T。则我们认为：

$S = \{s,$ *放弃的用户群, 未选择建设的中转站*$\}$

$T = \{t,$ *获益的用户群, 选择建设的中转站*$\}$

因为中间的弧的容量为∞，因此不会出现某个用户群在 T 中而其相关的中转站在 S 中。然后可以看出割的容量为（损失用户群的获益 + 需要建设的中转站的代价），又因为是最小割，所以结果最优。

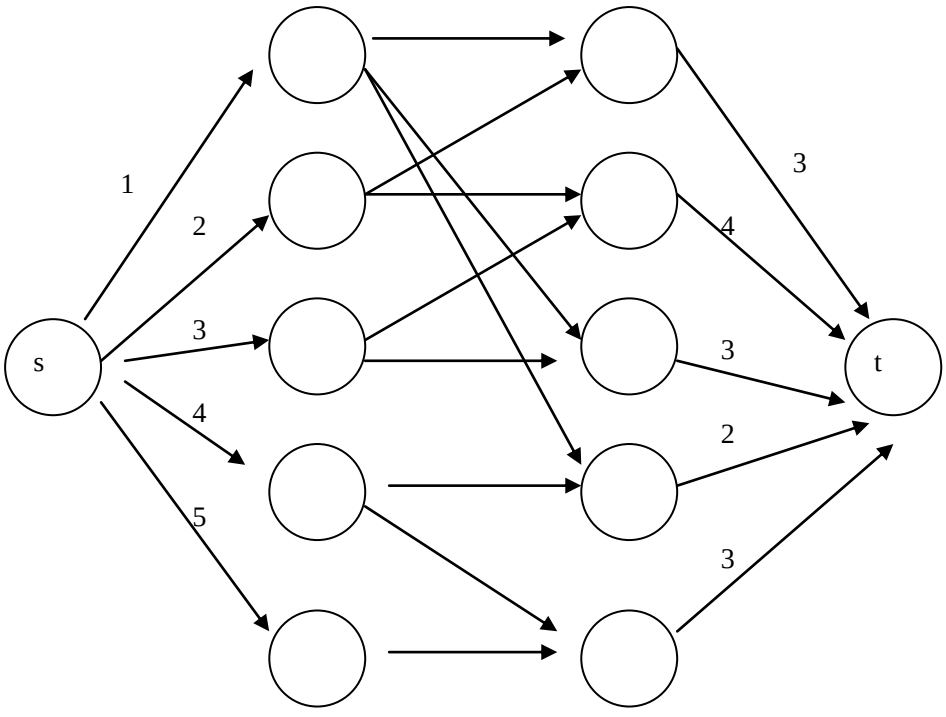

图 6　样例建成的流网络（中间弧的容量均为∞）

最后再观察数据范围，可得流网络的点数最多会有 55002（5000+50000+2）个，一般的最大流算法基本都是会超时的（虽然时限有 2s），但是用邻接表的 SAP

加上 Gap 和当前弧优化是可以轻松 AC 的（由于流网络是稀疏的）。

总结最小割的建模思想，可以发现基本都是运用割将点集分为两个集合的性质（也可以认为去除割上的弧后使得源点到汇点没有路径），加上最小割不会包含容量为∞的弧来满足题目的约束条件。

四、总结

网络流算法越来越在信息学竞赛中广泛应用，而且网络流的建模一直是一个难点。在平时的练习中，我们还是要多做题，多见一些常见的模型，这样才能在竞赛时很快想起来网络流的模型和算法。

总之，在学习中多练习和总结还是最重要的，只有这样才能更快地提高水平。

【参考书目】

1. 《算法艺术与信息学竞赛》 刘汝佳 黄亮 著
2. 《算法导论》 Thomas H. Cormen Charles E. Leiserson
Ronald L. Rivest Clifford Stein 著

【附录】

测试环境：Core 2 Duo E4600 @ 2.4 GHz / 2GB

《最大获利》的测试情况（单位：s）：
（程序 1：SAP；程序 2：SAP+Gap；程序 3：SAP+当前弧；程序 4：SAP+Gap+当前弧。所有程序都使用了数组模拟指针的邻接表）

	测试点 1	*测试点 2*	*测试点 3*	*测试点 4*	*测试点 5*	*测试点 6*	*测试点 7*	*测试点 8*	*测试点 9*	*测试点 10*
程序 1	0.04	0.01	0.02	0.02	0.02	0.02	0.02	0.03	25.87	25.70
程序 2	0.03	0.02	0.02	0.01	0.02	0.01	0.02	0.02	1.30	1.35
程序 3	0.01	0.01	0.02	0.01	0.01	0.02	0.02	0.02	5.94	6.40
程序 4	0.02	0.01	0.01	0.02	0.01	0.01	0.01	0.02	0.12	0.12

从上述测试可以看出，SAP 的高效完全是由于两个优化在起作用，因此只有加了优化的 SAP 才能算是一种高效的算法。

SAP 与其他最大流算法比较（单位：s）
（SAP 程序加了 Gap 和当前弧优化，HLPP 和 Dinic 的程序来自网络。数据均为稠密图，点数从 100 到 1000 递增）

	测试点 1	*测试点 2*	*测试点 3*	*测试点 4*	*测试点 5*	*测试点 6*	*测试点 7*	*测试点 8*	*测试点 9*	*测试点 10*
HLPP	0.06	0.06	0.08	0.08	0.11	0.20	0.18	0.23	0.28	0.32
SAP	0.02	0.02	0.08	0.08	0.13	0.17	0.24	0.31	0.39	0.49
Dinic	0.01	0.02	0.05	0.08	0.11	0.16	0.19	0.25	0.31	0.54

可见，由于 SAP 较低的编程复杂度和较低的理解难度，并且拥有不错的效率，因此在信息学竞赛中还是很实用的。

【程序】

USACO 4.2.1 Drainage Ditches 参考程序（SAP+Gap+当前弧）：

```
// Program  :  USACO 4.2.1 ditch (Max Flow)

#include <iostream>
#include <climits>
#include <cstring>

const int maxn = 200, maxm = 200;

using namespace std;

struct node
  {
    int data, weight;
    node *next, *anti;
  } *ge[maxn+1], *di[maxn+1], *path[maxn+1], data[maxm*2+1];
```

```
int dist[maxn+1], count_dist[maxn+1], his[maxn+1], pre[maxn+1], n;
int top = 0;

node *Add_Edge(int a, int b, int w)
  {
    node *p = &data[++ top];
    p->data = b;
    p->weight = w;
    p->next = ge[a];
    ge[a] = p;
    return p;
  }

void Ins_Edge(int a, int b, int w)
  {
    node *p1 = Add_Edge(a, b, w), *p2 = Add_Edge(b, a, 0);
    p1->anti = p2;
    p2->anti = p1;
  }

int Max_Flow(int s, int t)
  {
    int i, now_flow, total, min_dist;
    node *p, *locate;
    bool flag;
    memset(dist, 0, sizeof(dist));
    memset(count_dist, 0, sizeof(count_dist));
    count_dist[0] = n;
    for (i = 1; i <= n; i ++) di[i] = ge[i];
    for (total = 0, now_flow = INT_MAX, i = s; dist[s] < n; )
      {
        his[i] = now_flow;
        for (flag = false, p = di[i]; p != NULL; p = p->next)
          if ((p->weight > 0) && (dist[i] == dist[p->data]+1))
            {
              if (p->weight < now_flow) now_flow = p->weight;
              pre[p->data] = i;
              path[p->data] = p;
              di[i] = p;
              i = p->data;
              if (i == t)
                {
                  for (total += now_flow; i != s; i = pre[i])
                    {
                      path[i]->weight -= now_flow;
```

```
                path[i]->anti->weight += now_flow;
              }
            now_flow = INT_MAX;
          }
        flag = true;
        break;
      }
    if (! flag)
      {
      for (min_dist = n-1, p = ge[i]; p != NULL; p = p->next)
        if ((p->weight > 0) && (dist[p->data] < min_dist))
          {
          min_dist = dist[p->data];
          locate = p;
          }
      di[i] = locate;
      count_dist[dist[i]] --;
      if (count_dist[dist[i]] == 0) break;
      dist[i] = min_dist+1;
      count_dist[dist[i]] ++;
      if (i != s)
        {
        i = pre[i];
        now_flow = his[i];
        }
      }
  }
  return total;
}

int main()
{
  int i, m, a, b, w;
  freopen("ditch.in", "r", stdin);
  freopen("ditch.out", "w", stdout);
  scanf("%d%d", &m, &n);
  for (i = 1; i <= n; i ++) ge[i] = NULL;
  for (i = 1; i <= m; i ++)
    {
    scanf("%d%d%d", &a, &b, &w);
    Ins_Edge(a, b, w);
    }
  printf("%d\n", Max_Flow(1, n));
  return 0;
}
```