浅谈基于分层思想的网络流算法

上海市延安中学 王欣上

[关键字]

层次图 网络流 基本算法 应用

MPLA Dinic MPM

[摘要]

本文详细地介绍了基于层次图概念的三种算法,并通过例题来说明Dinic算法在信息学竞赛中的优越性。

[目录]

[正文]

一、引言

图论这门古老而又年轻的学科[①]在信息学竞赛中占据了相当大的比重。其中，网络流算法经常在题目中出现。网络流涵盖的知识非常丰富，从基本的最小割最大流定理到网络的许多变形再到最高标号预流推进的六个优化等等，同学们在平时需要多多涉猎这方面的知识，不断积累，才能应对题目的各种变化。

随着信息学竞赛的不断发展，其题目的难度以及考察范围都不断增大。现在，对于一些新出现的题目，仅仅掌握最朴素的网络流算法并不足以解决问题。本文针对一些数据规模比较大的网络流题目详细介绍了基于分层思想的 3 个网络流算法，并通过列举和比较说明了其在解题中的应用，而对一些基础的知识，如最小割最大流定理等，没有作具体阐释，大家可以在许多其他网络流资料中找到。

二、预备概念[②]

2.1 剩余图的概念

给定一个流量网络 $G_1=(E_1,V_1)$、源点 s、汇点 t、容量函数 c，以及其上的流量函数 f。我们这样定义对应的剩余图 $G_2=(E_2,V_2)$：剩余图中的点集与流量网络中的点集相同，即 $V_2=V_1$。对于流量网络中的任一条边[③] $(u,v)\in E_1$，若 $f(u,v)<c(u,v)$，那么边 $(u,v)\in E_2$，这条边在剩余图中的权值为

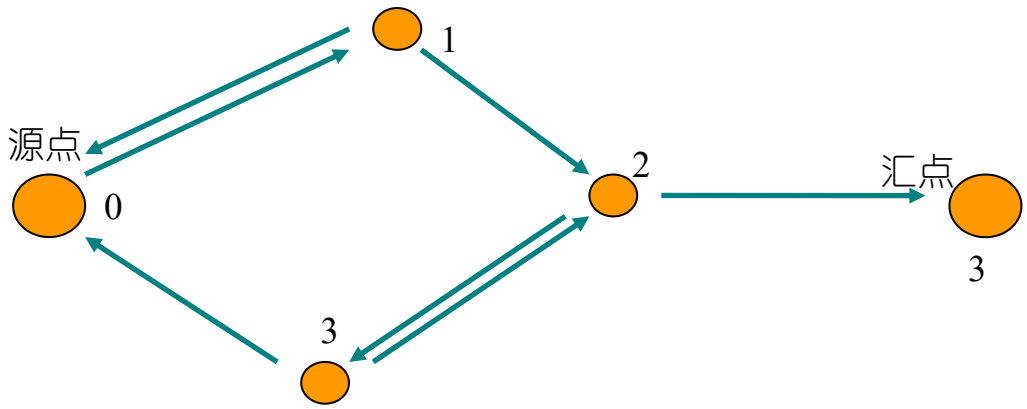
[①] 图论这门学科的诞生始于 18 世纪欧拉证明了七桥问题，发表《依据几何位置的解题方法》一文。但图论的真正发展是从 20 世纪五六十年代开始的。所以说，图论是一门既古老又年轻的学科。
[②] 本文对一些基本的理论，如最大流最小割定理等，不做阐述，读者可以参阅相关网络流资料。
[③] 本文中所有涉及到的边若无指明均为有向边。

$g(u,v)=c(u,v)-f(u,v)$；同时，若 $f(u,v)>0$ 那么边 $(v,u)\in E_2$，这条边在剩余图中的权值为 $g(v,u)=f(u,v)$。

我们可以发现，流量网络中的每条边在剩余图中都化作一条或二条边。剩余图中的每条边都表示在原流量网络中能沿其方向增广。剩余图的权值函数 $g(a,b)$ 表示在流量网络中能够沿着 a 到 b 的方向增广大小为 $g(a,b)$ 的流量。所以在剩余图中，从源点到汇点的任意一条简单路径[①]都对应着一条增广路，路径上每条边的权值的最小值即为能够一次增广的最大流量。

2.2 顶点的层次

在剩余图中，我们把从源点到点 u 的最短路径长度称作点 u 的层次，记为 $level(u)$。源点的层次为 0。在下面这张剩余图中：



每个点旁边的数字即表示该点在图中的层次。

2.3 层次图的概念

我们这样定义层次图 $G_3=(V_3,E_3)$：对于剩余图 $G_2=(V_2,E_2)$ 中的一条边 (u,v)，当且仅当 $level(u)+1=level(v)$ 时，边 $(u,v)\in E_3$；$V_3=\{u\mid E_3$中有边与u相连$\}$

[①] 简单路径：路径中不存在重复的顶点或边

直观地讲，层次图是建立在剩余图基础之上的一张“最短路图”。从源点开始，在层次图中沿着边不管怎么走，经过的路径一定是终点在剩余图中的最短路。

2.4 阻塞流的概念

在流量网络中存在一可行流 f，当该网络的层次图 G_3 中不存在增广路时，我们称流函数 f 为层次图 G_3 的阻塞流。

三、最短路径增值算法(MPLA)的步骤及复杂度分析

3.1 算法步骤

之前我们讲到的层次图将被应用在最短路径增值算法中。首先，我们看一下最短路径增值算法的步骤：

1、初始化流量，计算出剩余图
2、根据剩余图计算层次图。若汇点不在层次图内，则算法结束
3、在层次图内不断用 bfs 增广，直到层次图内没有增广路为止
4、转步骤 2

算法中，2、3 步被循环执行，我们将执行 2、3 步的一次循环称为一个**阶段**。每个阶段中，我们首先根据剩余图建立层次图，然后不断用 bfs 在层次图内增广，寻找阻塞流。增广完毕后，进入下一个阶段。这样不断重复，直到汇点不在层次图内出现为止。汇点不在层次图内意味着在剩余图中不存在一条从源点到汇点的

路径，即没有增广路。

在程序实现的时候，层次图并不用被“建”出来，我们只需对每个顶点标记层次，增广的时候，判断边是否满足 $level(u)+1=level(v)$ 这一约束即可。

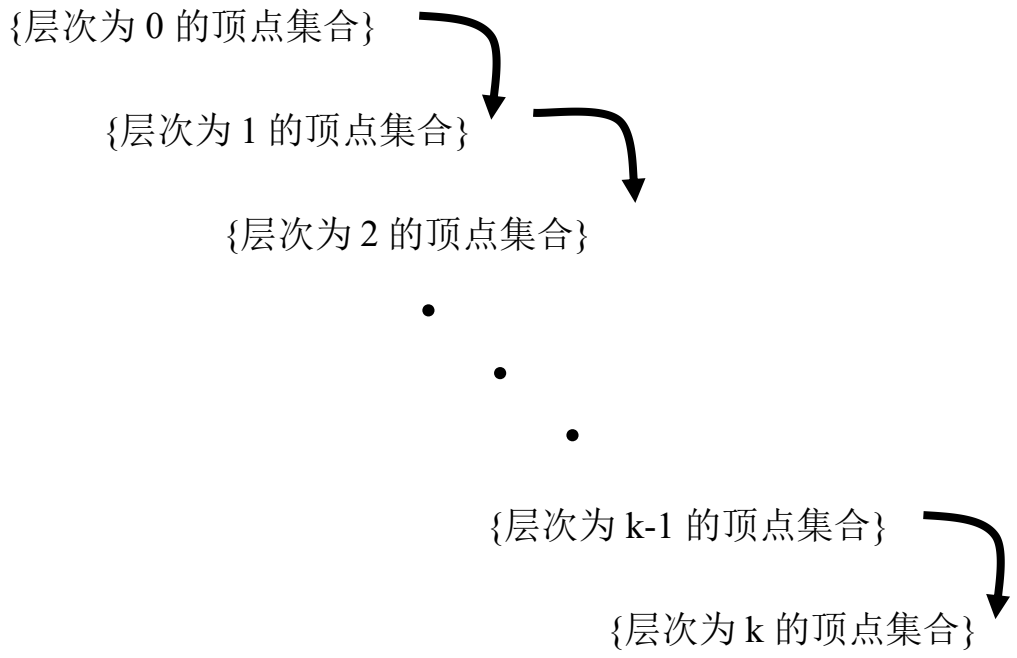
3.2 定理的证明

定理：对于有 n 个点的流量网络，在最短路径增值算法中，最多有 n 个阶段。

也就是说，在算法中层次图最多被建立 n 次。证明这个定理有助于我们进行算法复杂度分析。

证明：

在建立完层次图以后，假设从源点到汇点的最短路径长度为 k，我们将层次图中所有的点分到 k+1 个集合中，第 i 个集合为 $\{\text{顶点}u \mid level(u)=i-1\}$，如下图所示：



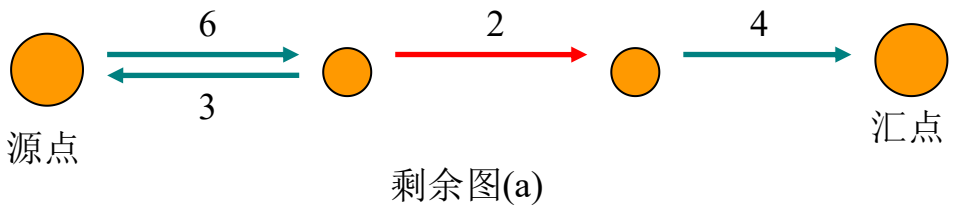
在剩余图中，存在着 2 类边。

第一类：从第 i 个集合中的顶点连到第 $i+1(1 \le i \le k)$ 个集合中的顶点

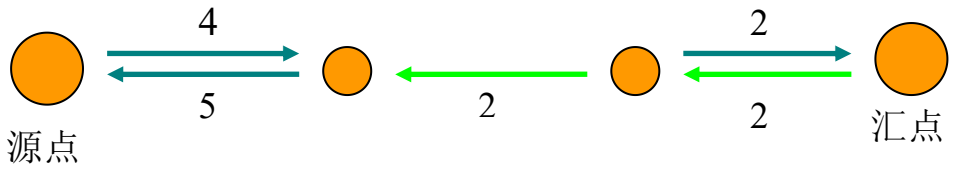
第二类：从第 $i(1 \le i \le k+1)$ 个集合中的顶点连到第 $j(1 \le j \le i)$ 个集合中的顶点

在层次图中，只存在第一类边，这是由层次图的性质决定的。我们所要找的增广路中的边也必定是第一类边。

当我们对一条增广路径增广后，会删除一条或多条增广路中的饱和边，也就是第一类边；而同时会在剩余图中加入一些与增广路径中的边反向的边。这些新加入的边一定是第二类边。如下图所示，在剩余图(a)中，找到一条从左向右的增广路径，能够增广的流量大小为 2。增广后的结果是剩余图(b)。可以发现，在剩余图(a)里面，中间一条红色第一类边在增广后饱和而被删除了，同时，在剩余图(b)中，新增了 2 条绿色的第二类边。



剩余图(a)



剩余图(b)

当我们在层次图中找到阻塞流之后，层次图中就不存在从第一个集合一步一步往下走，最后达到第 k+1 个集合的长为 k 的路径了。而此时不在层次图中的边都是第二类边。我们可以发现，这个时候在剩余图中的最短路径一定是这样：从

源点开始，往下一步一步走，走到某个集合后沿着第二类边向上退至某个集合，再继续一步一步向下走，到某个集合又向上退……直到走到汇点。

因为必然会经过第二类边，而经过的第一类边的数量>=k，所以路径总长度一定大于 k。这即是下一个阶段的最短路径长度。

由此，我们得出了一个结论：

结论：层次图中增广路径长度随阶段而严格递增。

因为增广路径长度最短是 1，最长是 n-1 ，再算上汇点不在层次图内的最后一次，层次图最多被建造 n 次，所以最短路径增值算法最多有 n 个阶段。

证毕。

3.3 复杂度分析

3.3.1 建造层次图的复杂度分析

我们将复杂度分析分为建层次图和找增广路两部分讨论。

前面已经证明了，在最短路径增值算法中，最多建 n 个层次图，每个层次图用 bfs 一次遍历即可得到。一次 bfs 遍历的复杂度为 $O(m)$，所以建层次图的总复杂度为 $O(n*m)$。

3.3.2 增广复杂度分析

我们首先分析在每一阶段中找增广路的复杂度。

注意到每增广一次，层次图中必定有一条边会被删除。层次图中最多有 m 条边，所以可以认为最多增广 m 次。在最短路径增广中，我们用 bfs 来增广。一次增广的复杂度为 $O(m+n)$。其中 $O(m)$ 为 bfs 的花费，$O(n)$ 为修改流量的花费。

所以在每一阶段的复杂度为 $O(m*(n+m))=O(m^2)$

这样，得到找增广路总的复杂度为 $O(n*m^2)$

最短路径增值算法的总复杂度即为建层次图的总复杂度与找增广路的总复杂度之和，为 $O(n*m^2)$。

四、Dinic 算法的步骤以及复杂度分析

4.1 算法步骤

Dinic 算法的思想也是分阶段地在层次图中增广。它与最短路径增值算法不同之处是：在 Dinic 算法中，我们用**一个 dfs** 过程代替多次 bfs 来寻找阻塞流。下面给出其算法步骤：

1、初始化流量，计算出剩余图
2、根据剩余图计算层次图。若汇点不在层次图内，则算法结束
3、在层次图内用一次 dfs 过程增广
4、转步骤 2

在 Dinic 的算法步骤中，只有第三步与最短路径增值算法不同。之后我们会发现，dfs 过程将会使算法的效率较之 MPLA 有非常大的提高。

下面是 dfs 的过程：

```
p←s;
While outdegree(s)>0
    u←p.top;
    if u<>t
        if outdegree(u)>0
            设(u,v)为层次图中的一条边;
            p←p,v;
        else
            从 p 和层次图中删除点 u,
            以及和 u 连接的所有边;
    else
        增广 p（删除了 p 中的饱和边）;
        令 p.top 为 p 中从 s 可到达的最后顶点;
end while
```

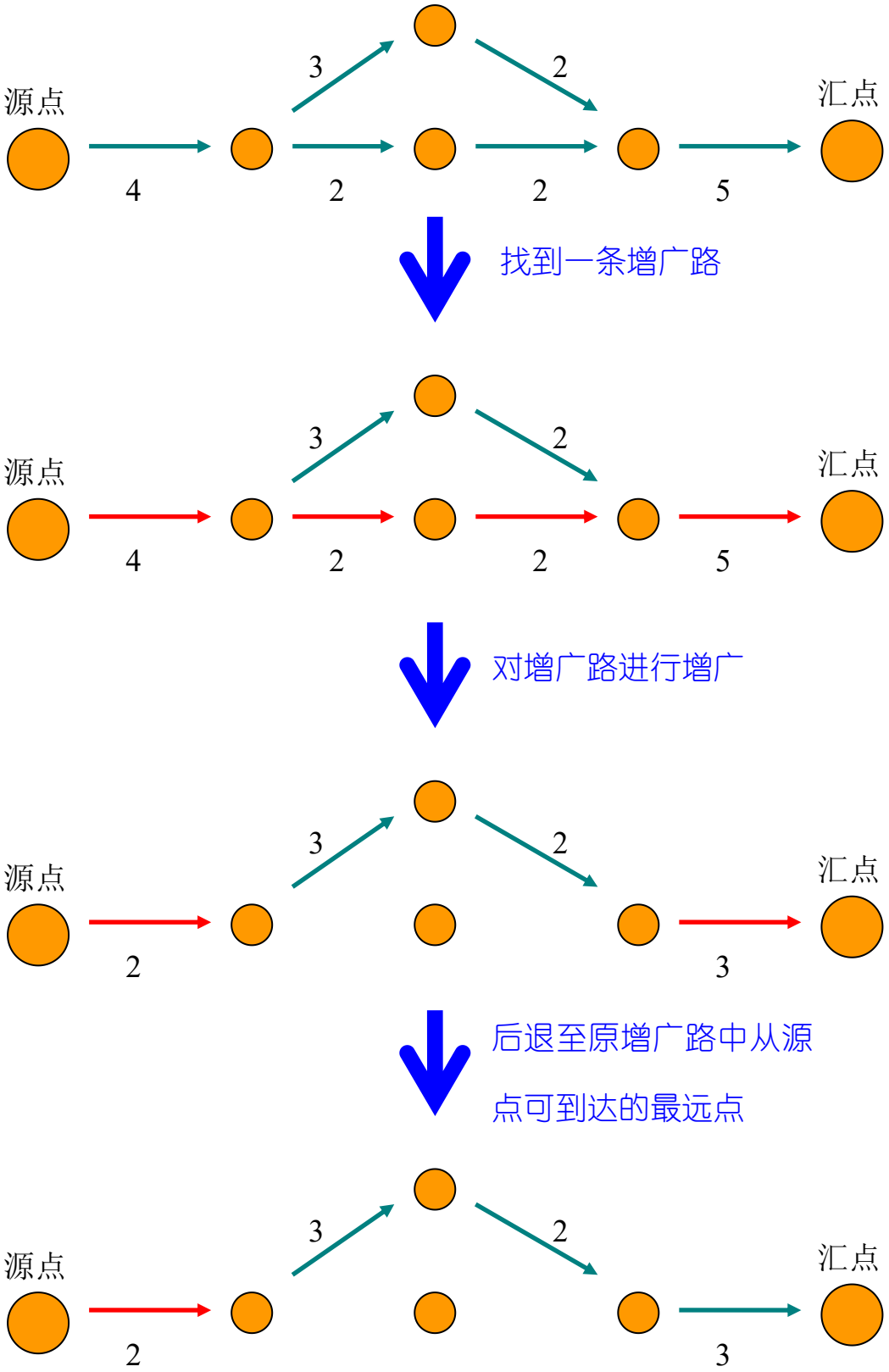
在程序里，p 表示找到的增广路径，p.top 为路径中的最后一个顶点。一开始，p 中只有源点。

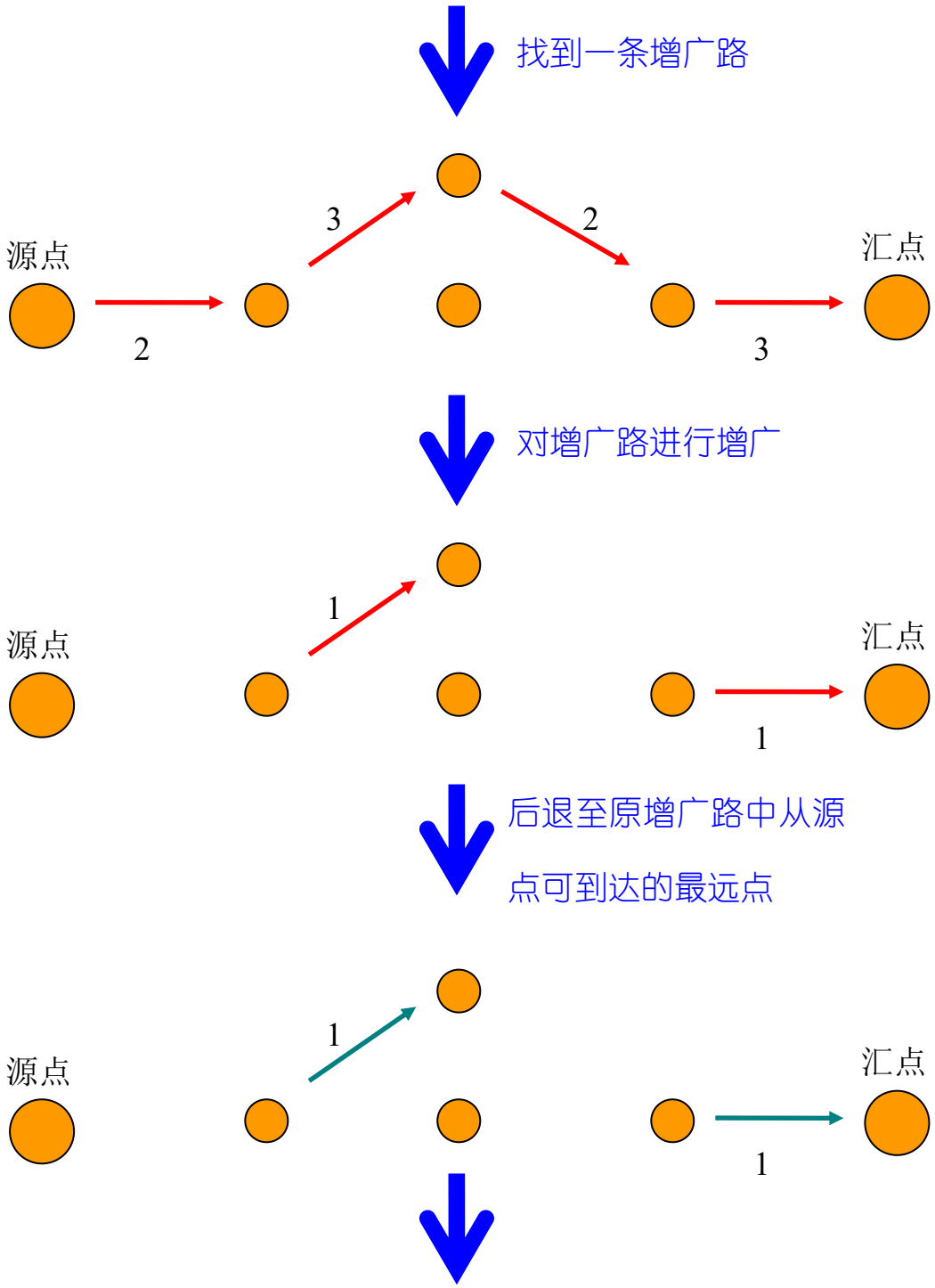
整个 While 循环分为 2 个操作。如果 p 的最后一个顶点为汇点，也就是说找到了增广路，那么对 p 增广，注意到增广后一定有一条或多条 p 中的边被删除了。这时，我们使增广路径**后退至 p 中从源点可到达的最后一个顶点。**

如果 p 的最后一个顶点不为汇点，那么观察最后那个的顶点 u 。若在层次图中存在从 u 连出的一条边，比如(u,v)，我们就将顶点 v 放入路径 p 中，继续 dfs 遍历；否则，点 u 对之后的 dfs 遍历就没有用了，我们将点 u 以及层次图中连到 u 的所有边删除，并且在 p 中后退一个点。

Dfs 过程将会不断重复这 2 个操作，直到从源点连出的边全部被删除为止。

下面给出一个 dfs 的图例，图中，红边代表找到的增广路 p 中的边。


源点
汇点
3
2
4
2
2
5
找到一条增广路
源点
汇点
3
2
4
2
2
5
对增广路进行增广
源点
汇点
3
2
2
3
后退至原增广路中从源
点可到达的最远点
源点
汇点
3
2
2
3


找到一条增广路
源点
汇点
3
2
2
3
对增广路进行增广
源点
汇点
1
1
后退至原增广路中从源
点可到达的最远点
源点
汇点
1
1

增广完毕

4.2 复杂度分析

和在最短路径增值算法中的证明一样，Dinic 算法最多被分为 n 个阶段。

这样首先可以得到 Dinic 算法中建立层次图的总复杂度仍是 $O(n*m)$。

我们再来分析 dfs 过程的总复杂度。在每一阶段，将 dfs 分成 2 部分分析：

```
p←s;
While outdegree(s)>0
    u←p.top;
    if u<>t
        if outdegree(u)>0
            设(u,v)为层次图中的一条边;
            p←p,v;
        else
            从 p 和层次图中删除点 u,
            以及和 u 连接的所有边;

    else
        增广 p（删除了 p 中的饱和边）;
        令 p.top 为 p 中从 s 可到达的最后顶点;
end while
```

（1） 修改增广路的流量并后退的花费：（即为代码中红框对应的部分）

在 3.3.2 小节中我们讲到过，在每一阶段，最多增广 m 次，每次修改流量的费用为 $O(n)$。而一次增广后在增广路中后退的费用也为 $O(n)$。所以在每一阶段，修改增广路以及后退的复杂度为 $O(m*(n+n))=O(m*n)$。

（2） Dfs 遍历时的前进与后退[①]：（即为代码中篮框对应的部分）

我们将用到 3.1 小节中的知识。

在 dfs 遍历时，如果当前路径的最后一个顶点能够继续扩展，则一定是沿着第一类边向汇点前进了一步。因为增广路径长度最长为 n，所以最多连续前进 n 步后就会遇到汇点。在前进的过程中，可能会遇到没有边能够沿着继续前进的情况，这时，我们将路径中的最后一个点在层次图中删除并出栈[②]。

注意到每后退一次必定会删除一个点，所以后退的次数最多为 n 次。在每一阶段中，后退的复杂度为 $O(n)$。

假设在最坏情况下，所有的点最后均被删除，一共后退了 n 次，这也就意味着，有 n 次的前进被“无情”地退了回来，这 n 次前进操作都打了水漂。除去这 n 次前进和 n 次后退，其余的前进都对最后找到增广路作了贡献。增广路最多找 m 次，每次最多前进 n 个点。所以所有前进操作最多为 $n+m*n$ 次，复杂度为 $O(m*n)$。

于是我们得到：在每一阶段中，dfs 遍历时前进与后退的花费为 $O(n)+O(m*n)=O(m*n)$。

综合以上二点，一次 dfs 的复杂度为 $O(m*n)$。因为最多进行 n 次 dfs，所以在 Dinic 算法中找增广路的总复杂度为 $O(m*n^2)$，这也是 Dinic 算法的总复杂度。

[①] 这里指的 dfs 遍历时的后退操作不是修改流量后进行的后退操作
[②] Dfs 遍历是基于栈这一数据结构的

五、Dinic 算法在信息学竞赛中的应用

例题 1 最大获利(profit)[①]

题目大意：一共有 N 个通讯信号中转站，建立第 i 个通讯中转站需要的成本为 Pi($1 \leqslant i \leqslant N$)。

另有 M 个用户群，第 i 个用户群会使用中转站 Ai 和中转站 Bi 进行通讯，公司可以获益 Ci。（$1 \leqslant i \leqslant M$, $1 \leqslant$ Ai, Bi $\leqslant N$）。

要求选择建立一些中转站，使得净收益最大。

数据范围：N<=5000　　M<=50000

初步分析：

这是一道比较经典的网络流题目。

按照题意，我们建一个图：把第 i 个通讯信号中转站用顶点 i 表示，第 i 个顶点的权值为 $-Pi$，用负号来表示亏损。对于使用中转站 Aj 和中转站 Bj 的用户群 j，我们把它用一条连结顶点 Aj 和 Bj 的边表示，这条边的权值为 Cj，用正数表示盈利。

我们的目标是：寻找一个点集，使得两个连结点都在这个集合中的边的权值和与点集中所有点的权值和之和最大。

我们可以这样想象：假设你准备建立所有的中转站，并且为每个用户群提供服务。就是说，你预计支出 $\sum Pi$ 的费用来得到 $\sum Ci$ 的收益。按照比较直观的经营想法：我们可以将每条边的权值分配一些到其连结的顶点上，来“填补”顶点的支出。当把顶点的权值“中和”为 0 时，边上剩下的权值即为我们的净收益。

假设每条边都尽可能多地分配权值到其连结的顶点上，使权值被中和为 0 的顶点尽可能多。最后可能有些边剩下一些权值，同时有些顶点的权值仍是负的。注意到与那些负权顶点相连的边现在的权值一定为 0 了，这就意味着，我们把一些用户群的收益全部投入建造中转站，最后还要自己贴钱进去，这当然是不划算的。所以，我们需要退掉一些用户，不建这个亏本的中转站。

我们删除负权顶点和与其相连的边（收回边对顶点的投入），这样一来，又

[①] 题目来源：Noi2006

可能会有一些顶点的权值变为负的（原来对其有贡献的边被删除了）。但与那些新增的负权顶点相连的边不可能存在正权值，否则的话，我们在之前就可以对边权的分配进行调整，从而将那些权值“转移”到早些被删除的点上去，违背了“尽可能多”的假设。

那些新增的负权顶点原本处于不亏不赚的状态，与其相连的边的收益已经全部被投入填补亏损，这种顶点删除后对最后的净收益没有影响。所以我们不断删除新增的负权顶点以及与其相连的边（注意到我们始终不可能删除正权边）。

这样，去除了所有亏本的生意，剩下的顶点要么不亏不赚，要么净赚，我们都用不着删它们。最后所有边上剩下的权值和即为最大净收益。

最大净收益的计算式为：

$$\sum Ci - \sum Pi + \text{所有删除的顶点建造成本} - \text{所有删除的边的初始收益}$$

$$= \sum Ci - \text{所有剩余顶点的成本} - \text{所有删除边的初始收益}$$

这样一来，问题的关键就是如何实现两个“尽可能多”。网络流模型可以形象地解决这个问题。我们建立一个二分图：把第 j 个用户群作为点 Xj，从源点 S 至 Xj 连一条边，容量为 Cj。把第 i 个中转站作为点 Yi，从 Yi 到汇点 T 连一条边，容量为 Pi。对第 j 个用户群，从 Xj 至 YAj、Xj 至 YBj 分别连 2 一条容量为无穷大的边。这样一来，两个“尽可能多”便可由求网络的最大流来实现。

我们又发现：

网络最小割的值恰好等于：所有剩余顶点的成本+所有删除边的初始收益

于是，最后答案为：$\sum Ci - \text{maxflow}$

深入分析

确定用网络流算法后，我们发现，这道题目的范围非常大。普通的最短路径增值算法一定不能通过所有的测试点。

	Test 1~8	Test 9	Test 10
最短路径增值算法	<0.1s	>30s	>30s

这时，有一种解题方法是：贪心初始流。

在算法的一开始，先根据图的某种特点，按照某种贪心思想构造一个初始流，

在这基础之上，使用最短路径增值算法。

注意到，普通的预先增广和预流增广[①]等对程序速度几乎不会有影响，因为主程序花同样多的时间也能达到同样的效果。所以，贪心初始流必须抓住题目和图的特点，这样才能达到事半功倍的目的。

在这题中，有一种可行的方法是：先贪心度数少的顶点[②]。这种方法在二分图匹配中被广泛应用到，但本题并不是匹配问题，中间边的权值大小均为无穷大，在没有写程序试验过实际效果以前，不敢主观判断这种方法在本题的效率。实际上，这种方法的效果的确不错，大家可以尝试一下，本文不作展开。

正当大家在比赛中犹豫是否贪心初始流时，简洁而又高效的 Dinic 算法粉墨登场了……

写 Dinic 的时间不比写 MPLA 的时间多，而且不容易出错，所以，在比赛中将 Dinic 代替 MPLA 作为首选算法是很划算的一件事。

写完 Dinic，用自己写的数据测试一下后，大家立刻发现：不需要再贪初始流了！Dinic 算法能够很快地通过最大数据[③]。

	Test 1~8	Test 9	Test 10
Dinic	<0.03s	0.40s	0.37s

至此，Dinic 秒掉了 profit。

除了 Dinic 算法，还有一些高效的算法，比如预流推进等等，它们也能很快地通过 profit 的所有数据：

	Test 1~8	Test 9	Test 10
预流推进	<0.03s	0.53s	0.51s

小结

在本题的二分图中，Dinic 的速度比预流推进略快一些。而对于一般随机图，Dinic 的速度远胜于无优化的预流推进，大家可以自己尝试测试一下。

但这并不意味着 Dinic 对于所有网络流图都比预流推进快，我们也可以构造出使 Dinic 比预流推进慢的情况。两种算法在速度上各有特点、各有优势，并不

[①] 贪心的预流增广：2 次 bfs 过程。第一次从源点向汇点推尽可能多的流量；第二次从汇点向源点把顶点积压的流量推回去。

[②] 具体做法可以参考周源写的 ahtsc2006 解题报告

[③] 在本文中出现的所有程序时间均为作者所写程序的测试结果，可取标准差σ =0.02s

能一概而论哪种算法速度更快。

但就编程实现这点而言，我认为 Dinic 相较其他高效算法更容易理解、更容易实现。这才是 Dinic 在信息学竞赛中的的最大优势所在。

例题 2 矩阵游戏[①]

题目大意：对于一个 n 行、m 列的 0-1 矩阵，规定在第 i 行中 1 的个数恰为 Ri 个（1<=i<=n）；在第 j 列中 1 的个数恰为 Cj 个（1<=j<=m）。

每一行、每一列最多可以有一个格子指定为 0。

问是否存在一种满足条件的 0-1 矩阵。

数据范围：n,m<=1000　　每个测试点最多 10 组数据

分析 1

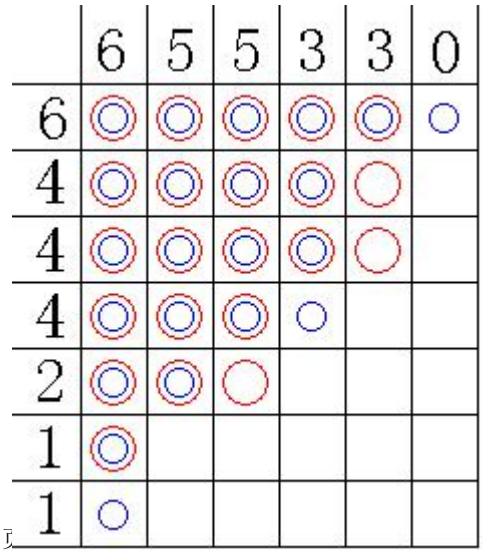
稍经分析，我们很容易建立起一个二分图网络流模型：把第 i 行作为顶点 Xi，从源点 S 至 Xi 连一条容量为 Ri 的边；把第 j 列作为顶点 Yj，从 Yj 至汇点 T 连一条容量为 Cj 的边；若第 i 行、第 j 列不指定为 0，那么从 Xi 至 Yj 连一条容量为 1 的边。

若边(Xi,Yj)的流量为 1，表示在第 i 行第 j 列的格子为 1。

我们只需求出网络的最大流，并判断最大流值是否等于总棋子数即可。

但是这样构图使得边数达到了 $O(n*m)$，如果使用 MPLA 算法，可以拿到 60%的分数；如果使用 Dinic 算法，可以拿到 80%的分数。

分析 2



我们不妨从贪心的角度考虑。

由于那几个指定的 0 过于妨碍我们思考，我们暂且不管它们，首先来考虑只有规定每行每列 1 的个数时，问题的特点。

考虑到将某两行或某两列整体交换后，不

[①] 题目来源：2006 年江苏省信息学竞赛选拔赛试题

影响问题的求解，我们将{Rn}和{Cm}分别从大到小排序，如图所示：上方的数字表示 Ci，左边的数字表示 Ri。

我们观察第一列。第一列需要有 6 个 1，而一共有 7 行可以提供 1，这时，我们将最后一行的 1 放到之后使用。

第二列正好每行都提供一个 1。

第三列需要 5 个 1，但只有 4 行能够提供，这时，我们将第一列储存着的 1 放到第三列。

这样一直下去，直到第五列，我们发现即使用完所有储存的 1，仍然不够（第五列右边可能还会发现剩余的 1，但由于那一行在第五列已经放了 1，所以不能被使用）。这就意味着，不存在满足条件的矩阵。

我们可以直接输出无解而不用对其作网络流了。

加入这个比较简单的判断后，100%的数据使用 Dinic 算法均可以通过了，但若使用 MPLA 仍只能过 60%的数据。

分析 3

其实，这题的标准做法是贪心，本文对其不再叙述，问题留给读者思考。

小结

Dinic 的高效性使得我们在解题时能够走“捷径”来获得高分。在比赛中，大部分时间往往都花在问题的思考、证明上，而在这题中，Dinic 算法省去了我们一大半思考时间，可见其在竞赛中的作用之大。

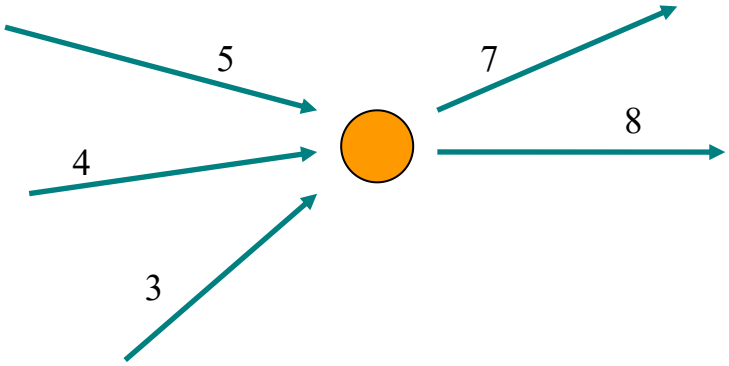
六、MPM 的算法步骤以及复杂度分析

6.1 算法步骤

MPM 算法的思想仍然是分阶段在层次图中寻找阻塞流。

首先，我们解释一个概念。顶点 u 的通过量：在层次图中，顶点 u 的入边权

和与出边权和的较小值。记为 throughput(u)。通过量是定义在非源、汇顶点之上的。



在上图中，顶点的入边权和为 12，出边权和为 15，所以通过量为 12。

每一阶段寻找阻塞流的流程：

1、	在层次图中寻找一个通过量最小的点 u
2、	在层次图中从点 u 向汇点“推”大小为 throughput(u)的流量。
3、	在层次图中从点 u 向源点“拉”大小为 throughput(u)的流量。
4、	删除点 u，若层次图中只剩下源、汇点，算法结束，否则转至第一步。

考虑到点 u 的通过量是其它所有点中最小的，所以在推流和拉流的过程中，不可能遇到阻碍。在推、拉流量的过程中，我们尽量使增广流量的边饱和，每次最多剩余一条半饱和的边。

6.2 复杂度分析

注意到上一节的流程每进行一次就删除一个点，一共进行了 n 次。每次找一个通过量最小的点花费 $O(n)$；一次推流和拉流的过程最多计算了 n 条半饱和边。在整个找阻塞流过程中，删除了最多 $O(m)$ 条边。

由此得到找阻塞流的复杂度为 $O(n*(n+n)+m)=O(n^2)$

MPM 算法总的复杂度为$O(n^3)$

七、总结

MPLA 算法易于理解，实现简单，适合中小规模数据；

Dinic 算法易于理解，实现简单，适合较大规模数据；

MPM 算法易于理解，实现繁琐，适合中小规模数据……

相较之下，Dinic 是竞赛中的首选网络流算法。在许多问题里，它是打开胜利之门的钥匙，是指引胜利彼岸的灯塔，是飞上成功蓝天的翅膀，是浇灌成功之花的甘露。

[感谢]

感谢 余晓清 老师 提供题目以及数据；

感谢 李天翼 同学 为本文的修改提出建议；

感谢 胡伯涛 同学 为我的程序提出改进意见；

感谢 胡刚 同学 为本文的修改提出建议；

感谢 寿鹤鸣 同学 为本文的修改提出建议；

感谢 刘汝佳 教练 为本文的修改提出建议。

[参考资料]

[1] 《Introduction to Algorithms》 Thomas H. Cormen
Charles E. Leiserson
Ronald L. Rivest
Clifford Stein

[2] 《Algorithms Design Techniques and Analysis》 M. H. Alsuwaiyel

[3] 《算法艺术与信息学竞赛》 刘汝佳、黄亮

[4] 《计算机算法设计技巧与分析》 王晓东

[5] 《A Friendly Introduction to Graph Theory》 Fred Buckley

Marty Lewinter

[6] 2006 年安徽省信息学竞赛选拔赛解题报告　周源

[附录]

最大获利

题目来源：NOI2006

【问题描述】

新的技术正冲击着手机通讯市场，对于各大运营商来说，这既是机遇，更是挑战。THU 集团旗下的CS&T 通讯公司在新一代通讯技术血战的前夜，需要做太多的准备工作，仅就站址选择一项，就需要完成前期市场研究、站址勘测、最优化等项目。

在前期市场调查和站址勘测之后，公司得到了一共N 个可以作为通讯信号中转站的地址，而由于这些地址的地理位置差异，在不同的地方建造通讯中转站需要投入的成本也是不一样的，所幸在前期调查之后这些都是已知数据：建立第i个通讯中转站需要的成本为Pi（$1 \leqslant i \leqslant N$）。

另外公司调查得出了所有期望中的用户群，一共M 个。关于第i 个用户群的信息概括为Ai, Bi 和Ci：这些用户会使用中转站Ai 和中转站Bi 进行通讯，公司可以获益Ci。（$1 \leqslant i \leqslant M$, $1 \leqslant$ Ai, Bi $\leqslant N$）

THU 集团的CS&T 公司可以有选择的建立一些中转站（投入成本），为一些用户提供服务并获得收益（获益之和）。那么如何选择最终建立的中转站才能让公司的净获利最大呢？（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。

所有变量的含义可以参见题目描述。

【输出格式】

你的程序只要向输出文件输出一个整数，表示公司可以得到的最大净获利。

【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

【样例说明】

选择建立1、2、3 号中转站，则需要投入成本6，获利为10，因此得到最大收益4。

【评分方法】

本题没有部分分，你的程序的输出只有和我们的答案完全一致才能获得满分，否则不得分。

【数据规模和约定】

80%的数据中：$N \leqslant 200$，$M \leqslant 1\,000$。

100%的数据中：$N \leqslant 5\,000$，$M \leqslant 50\,000$，$0 \leqslant C_i \leqslant 100$，$0 \leqslant P_i \leqslant 100$。

矩阵游戏

题目来源：JSOI2006 江苏省青少年信息学奥林匹克代表队组队选拔赛

【题目描述】

给定一个 0-1 矩阵，我们可对其各行、各列中“1”的个数进行统计。例如下面这个 3×4 的矩阵：

1	0	0	0	1
0	1	1	1	3
1	0	1	0	2
2	1	2	1	

其中各行包含“1”的个数分别是 1，3，2；各列包含“1”的个数分别是 2，1，

2，1。

对于一个的0-1矩阵，给定各行包含“1”的个数 r_1，r_2，...，r_n，以及各列包含“1”的个数 c_1，c_2，…，c_m。同时，指定一些矩阵中的一些元素**必须为0**，题目保证每行，每列至多有一个必须为0的元素。你的任务是判断是否存在满足上述要求的0-1矩阵。

【输入】

每组输入文件的第一行包含一个自然数 T（$1 \leqslant T \leqslant 10$），表示该组所包含的测试点个数。

每个测试点的第一行包含两个正整数 n 和 m（$1 \leqslant n, m \leqslant 1000$）。

第二行包含 n 个自然数 r_1，r_2，...，r_n，满足 $0 \leqslant r_i \leqslant m$，$i = 1,2, \cdots, n$。

第三行包含 m 个自然数 c_1，c_2，…，c_m，满足 $0 \leqslant c_j \leqslant n$，$j = 1,2, \cdots, m$。

第四行包含一个自然数 t，表示被指定为0的元素个数。接下来的 t 行每行包含两个自然数，表示被指定为0的元素所在的行和列。保证每行、每列至多有一个元素被指定为0。行、列的编号分别是1，2，…，n 和1，2，…，m。

每个测试点之间有一个空行分隔。

【输出】:

对于每个测试点，如果存在满足要求的矩阵，则输出“Yes”，否则输出“No”。

【样例输入输出】

matrix.in	matrix.out
2 3 4 1 3 2 2 1 2 1 2 1 2 2 1 1 1 1	Yes No

1 1 1 1	

【数据规模说明】

- 测试数据的前 6 个测试点满足 1 ≤ *n*, *m* ≤ 100
- 测试数据的最后 4 个测试点满足 1 ≤ *n*, *m* ≤ 1000