深 圳 大 学 实 验 报 告

课程名称：算法设计与分析

实验项目名称：图论——最大流应用问题

学院：计算机与软件学院

专业：计算机科学与技术

指导教师：刘刚

报告人：林宪亮 学号：2022150130

实验时间：2024 年 6 月 18 日—2024 年 6 月 20 日

实验报告提交时间：2024 年 6 月 20 日

教务部制

一、实验目的：

（1）掌握最大流算法思想。
（2）学会用最大流算法求解应用问题。

二、实验内容：

棒球赛问题：

TUESDAY, SEPTEMBER 10, 1996

San Francisco Chronicle

The Gate
Sports Online
► http://www.sfgate.com

SPORTING G

49ers, Young Get Big Bre

Quarterback m

By Gary Swan
Chronicle Staff Writer

The bye week has come at a perfect time for the 49ers and quarterback Steve Young. If they had a game next Sunday, there's a good chance Young would not play.

Giants Officially Leave the NL West Race

By Nancy Gay
Chronicle Staff Writer

With the smack of another National League West bat 500 miles away, the Giants' run at the division title ended last night, just as they were handing the visiting St. Louis Cardinals an even bigger lead in the NL Central.

CARDINALS 6
GIANTS 2

In San Diego, Greg Vaughn's three-run homer in the eighth pushed the Padres over the Pirates and officially shoved the rest of the Giants' season into the background. On the heels of their tedious 6-2 loss before an announced crowd of 10,307 at Candlestick Park, the Giants fell 19½ games off the lead.

As it is, the worst the Padres (80-65) can finish is 80-82. The Giants have fallen to 59-83 with 20 games left; they cannot win 80 games. Coming off a miserable 2-8 mark on a three-city road trip that saw their road record drop to 27-47, the Giants were hoping to get off on the right foot in their longest homestand of the year (15 games, 14 days).

Financing in Place For Giants' New Stadium
SEE PAGE B1, MAIN NEWS

"Where we are, you're going to be eliminated sooner or later," Baker said quietly. "But it doesn't alter the fact that we've still got to play ball. You've still got to play hard, the fans come out to watch you play. You've got to play for the fact of loving to play, no matter where you are in the standings.

"You've got to play the role of spoiler, to not make it easier on

GIANTS: Page B5 Col. 2

1996 年 9 月 10 日，《旧金山纪事报》的体育版上登载了《巨人队正式告别 NL 西区比赛》一文，宣布了旧金山巨人队输掉比赛的消息。当时，圣地亚哥教士队凭借 80 场胜利暂列西区比赛第一，旧金山巨人队只赢得了 59 场比赛，要想追上圣地亚哥教士队，至少还得再赢 21 场比赛才行。然而，根据赛程安排，巨人队只剩下 20 场比赛没打了，因而彻底与冠军无缘（摘自http://www.matrix67.com/blog/archives/5190）。

有趣的是，报社可能没有发现，其实在两天以前，也就是 1996 年 9 月 8 日，巨人队就已经没有夺冠的可能了。那一天，圣地亚哥教士队还只有 78 场胜利，与洛杉矶道奇队暂时并列第一。此时的巨人队仍然是 59 场胜利，但还有 22 场比赛没打。因而，表面上看起来，巨人队似乎仍有夺冠的可能。然而，根据赛程安排，圣地亚哥教士队和洛杉矶道奇队互相之间还有 7 场比赛要打，其中必有一方会获得至少 4 场胜利，从而拿到 82 胜的总分；即使巨人队剩下的 22 场比赛全胜，也只能得到 81 胜。由此可见，巨人队再怎么努力，也不能获得冠军了。

在美国职业棒球的例行赛中，每个球队都要打 162 场比赛（对手包括但不限于同一分区里的其他队伍，和同一队伍也往往会有多次交手），所胜场数最多者为该分区的冠军；如果有并列第一的情况，则用加赛决出冠军。在比赛过程中，如果我们发现，某支球队无论如何都已经不可能以第一名或者并列第一名的成绩结束比赛，那么这支球队就

提前被淘汰了（虽然它还要继续打下去）。从上面的例子中可以看出，发现并且证明一个球队已经告败，有时并不是一件容易的事。

关于这个事情有一个有趣的故事，下面是一段对话：

“看到上周报纸上关于爱因斯坦的那篇文章了吗？……有记者请他算出三角比赛的数学公式。你知道，一支球队赢得了很多剩余的比赛，其他球队则赢这个赢了那个。这个比赛到底有多少种可能性？哪个球队更有优势？”

“他到底知道吗？”

“显然他知道的也不多。上周五他选择道奇队没有选巨人队。”

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}			
				Atl	Phi	NY	Mon
Atlanta	83	71	8	-	1	6	1
Philly	80	79	3	1	-	0	2
New York	78	78	6	6	0	-	0
Montreal	77	82	3	1	2	0	-

上面的表是四个球队的比赛情况，现在的问题是哪些球队有机会以最多的胜利结束这个赛季？可以看到蒙特利尔队因最多只能取得 80 场胜利而被淘汰，但亚特兰大队已经取得 83 场胜利，蒙特利尔队因为wi + ri < wj 而被淘汰。费城队可以赢83场，但仍然会被淘汰。 。 。如果亚特兰大输掉一场比赛，那么其他球队就会赢一场。所以答案不仅取决于已经赢了多少场比赛，还取决于他们的对手是谁。

请利用最大流算法给出上面这个棒球问题的求解方法。

1. 解释流网络的构造原理。
2. 解释为什么最大流能解决这个问题。
3. 给出上面四个球队的求解结果。
4. 尽可能实验优化的最大流算法。

四、实验内容及过程：

1. 解释流网络的构造原理。

（1） 流网络介绍：

• 定义

流网络（Flow Network）是一个有向图，其中每条边都有一个非负容量。流网络用于解决许多实际问题，例如运输、通信和最大流问题。流网络的关键概念包括节点、边、容量、流量、源节点（source）和汇节点（sink）。

• 基本构造

节点：表示网络中的点，通常用 V 表示。

特殊节点：源节点 s 和汇节点 t。

边：表示节点之间的连接，通常用 E 表示。

每条边 (u,v) 具有一个非负容量 c(u,v)，表示边 (u,v) 上允许的最大流量。
容量：容量是指边 (u,v) 上可以容纳的最大流量，记作 c(u,v)。
流量：流量是指实际在边 (u,v) 上流动的量，记作 f(u,v)。必须满足流入等于流出的守恒条件，除非该节点是源或汇节点。

• 流网络中的基本性质
容量约束：每条边上的流量不能超过其容量。
流量守恒：对于除源节点 s 和汇节点 t 以外的每个节点 v，其流入量等于流出量。

（2） 构造原理
流网络的构造原理主要包括以下步骤：（以当前题目为例子）
• 节点的设置：
每个球队对应一个节点。
比赛对手对 (i, j) 也对应一个节点。
一个源节点 S 和一个汇节点 T。

• 边的设置：
从源节点 S 到每个比赛对手节点 (i, j) 连接一条边，容量为该对手对的比赛场次。
从每个比赛对手节点 (i, j) 到其两个对应的球队节点 i 和 j 各连一条边，容量为无穷大（实际上可以设置为一个非常大的值）。
从每个球队节点到汇节点 T 连接一条边，容量为该球队在理论上还能赢的最大比赛场次。

• 容量设置：
源节点 S 到比赛对手节点 (i, j) 的边容量为两队之间的剩余比赛场次。
比赛对手节点到球队节点的边容量设置为无穷大，以确保这些比赛结果只受源节点容量的限制。
球队节点到汇节点 T 的边容量设置为该球队的最大可能胜场数减去当前已经胜出的比赛场次。

2. 解释为什么最大流能解决这个问题。

（1） 重新定义问题
棒球比赛问题可以简化为：
目标：确定是否存在一种比赛结果分配，使得某支球队能达到或超过特定的胜场数（即确认这支球队是否仍有可能成为分区冠军）。
比赛和胜场数限制：每场比赛的结果必须分配给一支参赛的球队，每支球队的胜场数不能超过其剩余的最大可能胜场数。

（2） 转换为最大流问题
为了实现上述目标，我将问题转换为一个流网络问题：
比赛结果分配模型：将比赛结果分配建模为流网络中的流动，流量代表比赛的胜利被分配给特定的球队。

流网络的结构：

源节点 S 表示比赛的开始。

比赛节点（i，j）表示两支球队 i 和 j 之间的剩余比赛。

球队节点表示每支球队的胜场数。

汇节点 T 表示比赛的结束。

边和容量的设置：

边容量限制表示每场比赛只能有一个胜者。

每个比赛节点（i，j）到其对应的球队节点 i 和 j 的边容量为无穷大（代表比赛胜利可以分配给任意一支球队）。

每个球队节点到汇节点 T 的边容量表示该球队还能赢得的剩余比赛场数。

最大流算法的作用：

通过运行最大流算法，我能解决以下问题：

流量分配：

最大流算法在流网络中寻找一条从源到汇的最大流路径。每条路径中的流量代表了一场比赛的结果分配。

如果源节点到汇节点的最大流等于所有比赛场次的总和，则表示存在一种可能的比赛结果分配方案，使得球队能达到其最大可能的胜场数。

可行性验证：

如果最大流小于所有比赛场次的总和，则表示不存在一种比赛结果分配方案，导致球队无法达到或超过其所需的胜场数。

由此可以得出结论最大流可以解决这个问题。

3．给出上面四个球队的求解结果。

（1） 问题分析

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}			
				Atl	Phi	NY	Mon
Atlanta	83	71	8	-	1	6	1
Philly	80	79	3	1	-	0	2
New York	78	78	6	6	0	-	0
Montreal	77	82	3	1	2	0	-

图 1 数据表

• 问题描述：

如图 1，共有 4 支队伍，队伍 i 已经赢下了 w[i]场比赛，输掉了 l[i]场比赛，还剩下 r[i]场比赛尚未进行。同时，g[i][j]表示队伍 i 和队伍 j 之间还需要进行的比赛场数。我们的目标是计算哪些队伍在数学上已经被淘汰了。我们假设没有比赛会以平局结束，并且所有比赛都会如期完成。

• 简单分析：

如图 1，Montreal 在数学上已经被淘汰，因为即使赢下所有剩余比赛，它最多也只能赢得 80 场，而 Atlanta 已经赢了 83 场，这是最简单的一种淘汰情形。

但是，还有更复杂的情形。例如，Philly 同样在数学上已经被淘汰。Philly 最多可以赢得 83 场，这足够与 Atlanta 并列第一。但前提是 Atlanta 输掉所有剩余比赛，包括和 New York 的 6 场比赛，这样的话 New York 至少会赢 84 场。因此 Philly 也会被淘汰。

而反观 New York，虽然它当前的胜场数没有 Philadelly 多，但在数学上并没有被完全淘汰。

• 转为最大流问题：

为了判断某支队伍 x 是否已经被淘汰，我们可以通过最大流问题来解决。需要考虑以下两种情况：

平凡淘汰：如果队伍 x 能赢下的最大场数比其他某支队伍已经赢下的场数还要小，那么队伍 x 一定会被淘汰。即，如果 w[x]+r[x]<w[i]，那么队伍 x 一定会被淘汰。

非平凡淘汰：如果不是平凡淘汰的情况，我们需要构建一个流网络，并解决这个网络中的最大流问题。在这个网络中，每条流代表剩余比赛的结果。顶点对应所有队伍（不包括队伍 x）和剩余的比赛（不包括有队伍 x 的比赛）。

（2） 网络图构造

以对队伍 Philly 为例：

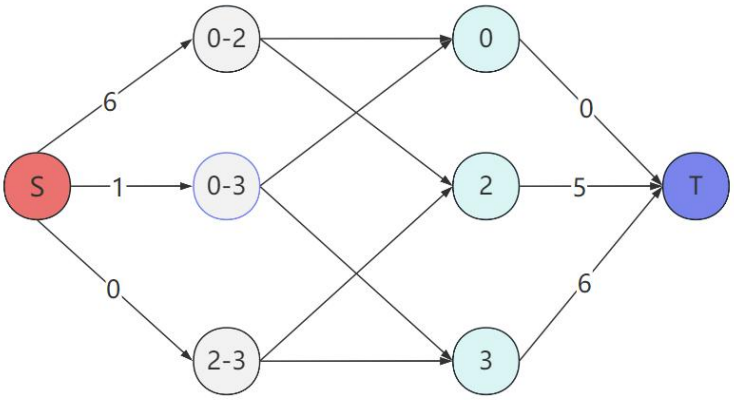

图 2 网络图

该图有四个部分：源点 S，比赛点，队伍点，汇点 T 组成。

源点 S：虚拟点一个

比赛点：比赛点由剩余 3 队伍两两比赛
队伍点：有 3 个队伍点
汇点 T：虚拟点

源点 S 到 比赛点 权值：g[i][j]
比赛点 到 队伍点 权值：无穷大
队伍点 到 汇点 T 权值：w[x]+r[w]-w[i],对于 Philly（x）,有 w[x]+r[w]=80+3=83; 83-w[0]=0; 83-w[2]=5;83-w[3]=6。

（3） 使用 Ford - Fulkerson 算法求解

算法基本思想：
Ford-Fulkerson 算法通过不断寻找增广路径并增加路径上的流量来增加网络的总体流量。增广路径是指从源节点到汇节点的路径，其上的边的剩余容量大于零。通过不断寻找增广路径并增加流量，直到无法找到增广路径为止，就可以得到最大流。

步骤：
• 初始化流量：初始化正向边的流量，初始化反向边的流量为 0。
• 寻找增广路径：在图中找到一条从源节点 s 到汇节点 t 的增广路径。可以使用深度优先搜索（DFS）或广度优先搜索（BFS）来寻找这条路径。
• 确定增加的流量：如果存在增广路径，则找出路径上边的最小剩余容量，这个最小值将成为路径上增加的流量。
• 更新流量：沿着增广路径更新每条边的流量。具体来说，与流量方向相同的边流量增加，与流量方向相反的边流量减少。
• 重复步骤 2-4：继续寻找新的增广路径，并更新流量，直到找不到任何增广路径为止。
• 计算最大流量：最终，所有增广路径的流量总和就是从源节点到汇节点的最大流量。

示例：

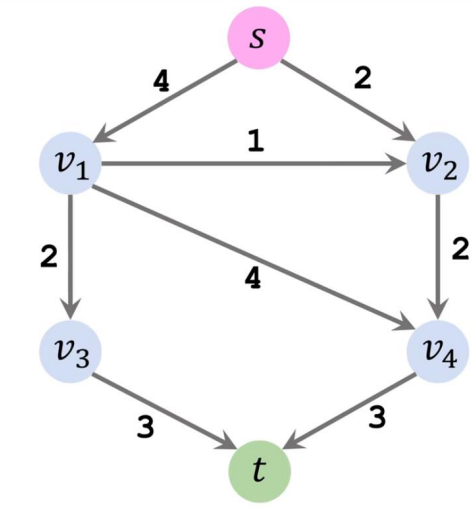

图 3 示例原图

图 3 为示例有向图。边上的数字代表可以流过的最大流量。

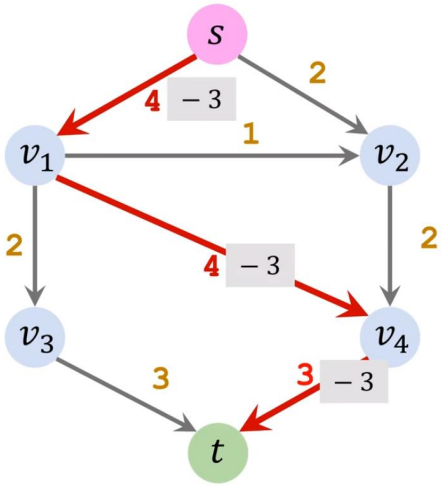

图 4 寻找增广路径

如图 4，我先寻找一条从源点到汇点的增广路径，然后计算这条路径上可以流过的最大流量，显然最大流量为 3，那么所有红色边的数值就需要减 3，代表可以流过的流量减少 3。

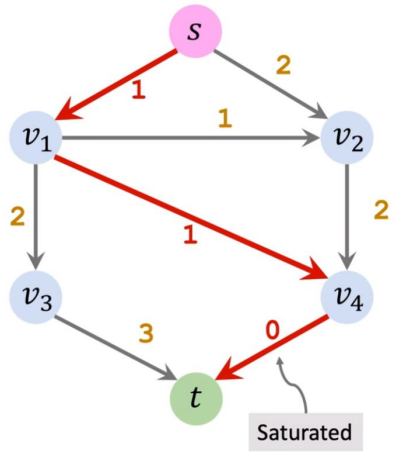

图 5 边饱和

如图 5，边（V4，T）的可用流量为 0，代表这条边不能再流经任何流量，因此将这条边删除。

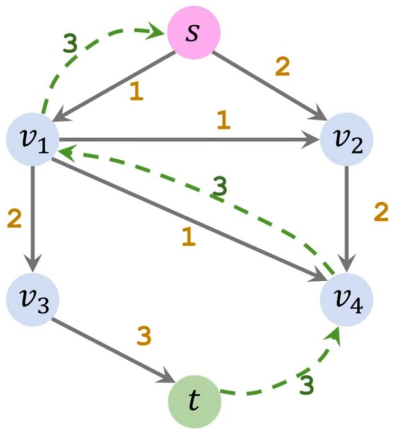

图 6 增加反向边

如图 6，对于刚刚流经的路径，我们增加一条反向路径，路径上边的数值大小为刚刚流经的流量大小，这是 Ford - Folkerson 算法的精髓，可以让流量回流，达到一个重新分配流量路径的效果，从而可以找到最大流量。

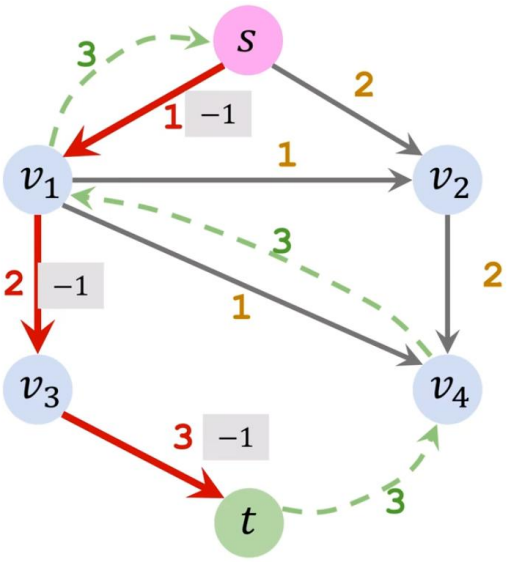

图 7 寻找增广路径

如图 7，和图五的步骤一样，我再寻找一条新的增广路径。然后根据路径上边的权值确定流量的大小。如图，流量的大小为 1。

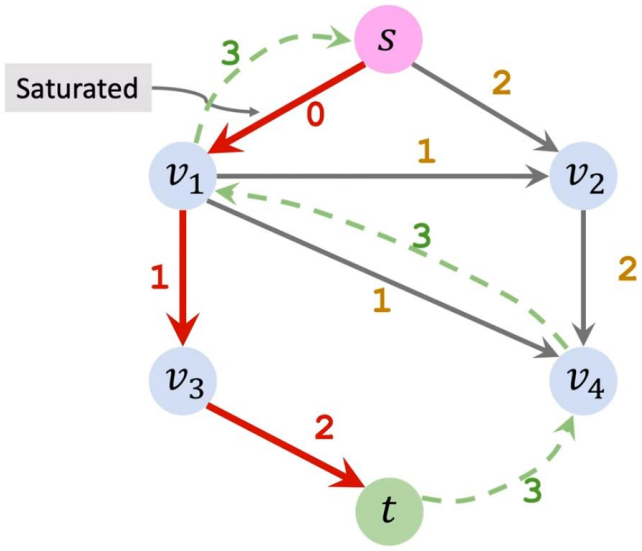

图 8 删除边

如图 8，对于权值为 0 的边，我将其删除。后续过程和前面的一致，都是重复而已，就不再赘述。

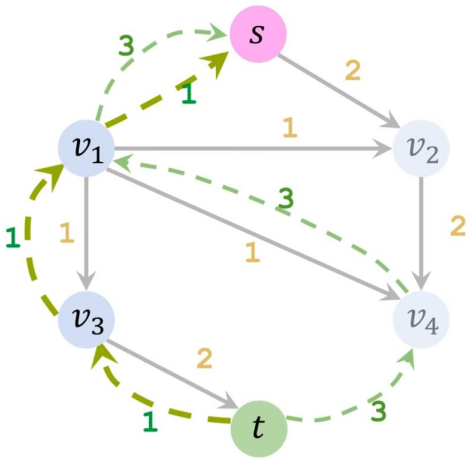

图 9　反向边

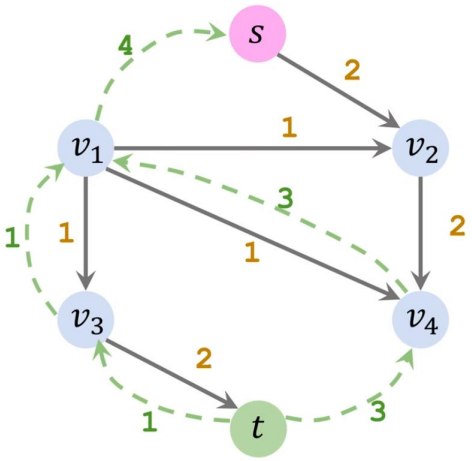

图 10　重合边

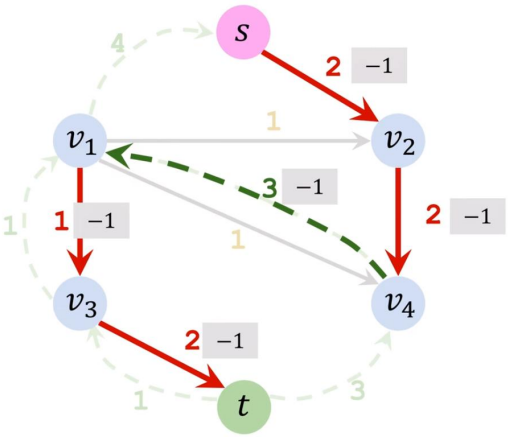

图 11 增广路径

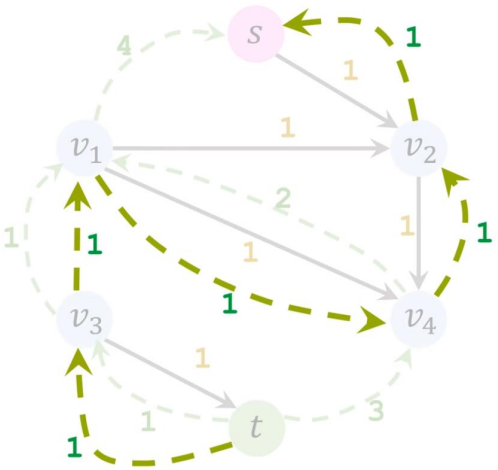

图 12 加反向边

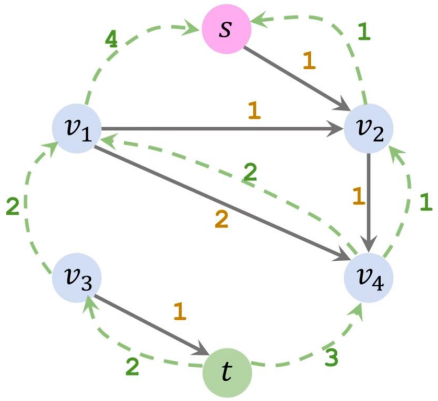

图 13 重合边合并

如图 10 和图 12，需要把一些重合的边进行合并。

最后我就能得出这个图的最大流为 5。因为本来从源点流出的两条边一条已经流了 4 份

流量，另一条流了 1 份流量，加起来为 5 份流量。

伪代码：

Algorithm 1：Ford - Folkerson

```
Input：S  V  Graph
Output: max_flow
1. Initial the Graph
2. While True:
3.     Parent, flow = DFS(G, S, T)
4.     If not path:
5.         Break
6.     V=T
7.     While V！=T：
8.         U=parent[V]
9.         Flow(U,V) - = flow
10.        Flow(V,U)+ = flow
11.        V=U
12. Max_temp=0
13. For each edge(S, V) in G:
14.     Max_temp+=Flow(S, V)
15. Max_Flow=Max_Flow-Max_temp
16. Return Max_Flow
```

代码解释：
先初始化图，如图 2 所示，然后寻找从源点到汇点的路径，找出路径上的最大可续流量，然后更新路径上边的权重并增加反向边。之后计算就可以得到最大流。那么如何判断一个队伍是否被淘汰呢，就只需要判断 Max_temp 是不是为 0 即可，此时的最大流为初始化的最大流。

时间复杂度分析：
Ford - Folkerson 算法的时间复杂度为 O（f×m），其中 f 为最大流，m 为图中边的数量。因为在最差的情况下，Ford - Folkerson 算法需要运行的次数等于最大流 f，并且每一次寻找从源点到汇点的路径的时间复杂度为 O（m），于是就有时间复杂度为 O（f×m）。

淘汰结果;
Atlanta，New York 可能夺冠，Philly 和 Montreal 被淘汰。

4. 最大流算法优化
（1）Edmonds - Karp 算法
Edmonds - Karp 算法是 Ford - Fulkerson 算法的一种改进版本，用于解决最大流问题。与 Ford-Fulkerson 算法不同的是，Edmonds-Karp 算法使用广度优先搜索（BFS）来选择增广路径，从而保证每次找到的增广路径是最短路径。

算法步骤

• 初始化流量：初始化正向边的流量，初始化反向边的流量为 0。

• 寻找增广路径：使用广度优先搜索（BFS）寻找增广路径：BFS 会按照节点的层级顺序搜索，确保找到的路径是最短路径。

• 确定增加的流量：如果存在增广路径，则找出路径上边的最小剩余容量，这个最小值将成为路径上增加的流量。

• 更新流量：沿着增广路径更新每条边的流量。具体来说，与流量方向相同的边流量增加，与流量方向相反的边流量减少。

• 重复步骤 2-4：继续寻找新的增广路径，并更新流量，直到找不到任何增广路径为止。

• 计算最大流量：最终，所有增广路径的流量总和就是从源节点到汇节点的最大流量。

算法关键定理证明：

① 增广路定理：

在残留网络中，任何一条从 s 到 t 的有向通路都对应原图中的一条增广路。只需找到该路径上所有残量的最小值 d_min，并将其增量添加到路径的每一条对应边的流量上，这个过程称为增广。显然，网络流中的流量可以增加，当且仅当当前网络中存在增广路。因此，当残留网络中不存在增广路时，当前流量即为最大流。这就是著名的增广路定理，它提供了解决最大流问题的思路，Edmonds-Karp 算法和 Dinic 算法都是基于增广路方法的具体实现。

② 证明增广路径长度递增：

设 p 是第一次 BFS 找到的增广路，则 p 必为残留网络 (s, t) 中的最短路径。在增广操作后，p 上的关键边 (u, v) 将从残留网络中删去并增加反向边 (v, u)。假设第一次找到的增广路径是 p = (s → u → v → t)，关键边为 (u, v)。此时进行增广操作后，将从残留网络中删去正向边 (u, v)，并增加反向边 (v, u)。

为证明增广路径长度递增，假设下一次 BFS 搜索出现了更短的路径 p'，则一定出现反向边 (v, u)，且 p' = (s → v → u → t)。由于 BFS 性质保证第一次搜索中 dis(s, u) 和 dis(v, t) 都是最短的，此时有： dis(s,t)=dis(s,u)+dis(v,t)+1

若出现了更短的路径，则有： $|p'|=dis'(s,v)+dis'(u,t)+1<|p|$

而根据 BFS 性质： $dis(s,v)\leqslant dis'(s,v), dis(u,t)\leqslant dis'(u,t)$

与假设矛盾，因此不存在长度更短的路径 p'，即 BFS 找到的增广路径长度是递增的。

③ 证明 Edmonds-Karp 算法迭代具有有穷上界

对于任一残留网络，在沿任一条增广路径增加流量后，该路径上的所有关键边都将从残留网络中消失。并且任意一条增广路径上都至少存在一条关键边。因此，只需证明 |E|

中的每条边成为关键边的次数是有限且为定值。具体结论如下：|E| 中的每条边成为关键边的次数最多为 |V| / 2 次。

设 u 和 v 为集合 V 中的两个节点，且它们由 E 中的一条有向边连接。由于增广路径都是最短路径，因此当边 (u, v) 第一次成为关键边时，有： $\delta f(s,v)=\delta f(s,u)+1$

在对流进行增量后，边 (u, v) 将从残留网络中删除，直到从 u 到 v 的网络流减小后为止，并且只有当 (u, v) 出现在增广路径上时，这种情况才会发生。此时，若 f' 是 G 的流，则有： $\delta f'(s,u)=\delta f'(s,v)+1$

根据之前的证明： $\delta f(s,v)\leqslant\delta f'(s,v)$

联合得出： $\delta f'(s,u)=\delta f'(s,v)+1\geqslant\delta f(s,v)+1=\delta f(s,u)+2$\

因此，从边 (u, v) 成为关键边到下一次再成为关键边，源点 s 到节点 u 的距离至少增加两个单位，而从源点 s 到 u 的最初距离至少为 0，从 s 到 u 的最短路径上的中间节点不可能包括节点 s、u 或 t（因为边 (u, v) 处于一条增广路径上意味着 u 与 t 不同）。因此，在节点 u 成为不可达节点前，其距离最多为 |V|-2。因此，在边 (u, v) 第一次成为关键边时，它还可以最多再成为 |V|/2-1 次关键边，即边 (u, v) 成为关键边的总次数为 |V| / 2。由于一共有 O(E) 对节点可以在一个残留网络中有边彼此连接，因此 Edmonds-Karp 算法执行的全部过程中，关键边的总数为 O(VE)。每条路径至少有一条关键边，原命题得证。

在使用 BFS 寻找增广路径时，每次迭代在 O(E) 时间内实现，因此 Edmonds-Karp 算法的时间复杂度为 $O(VE^2)$。

时间复杂度分析：
Edmonds-Karp 算法的时间复杂度为 $O(V\cdot E^2)$，其中 V 是节点数，E 是边数。

算法优势
最优解：通过使用广度优先搜索来选择最短增广路径，Edmonds-Karp 算法保证了每次迭代增加的流量是最优的，从而保证了算法能够找到最大流的最优解。
适中规模：由于使用了广度优先搜索，Edmonds-Karp 算法的运行时间相对较短，适用于中等规模的问题。

伪代码：

代码解释：
先初始化图，如图 2 所示，然后使用 BFS 算法找出从源点到汇点的最短路径，找出路径上的最大可续流量，然后更新路径上边的权重并增加反向边。之后计算就可以得到最大流。那么如何判断一个队伍是否被淘汰呢，就只需要判断 Max_temp 是不是为 0 即可，此时的最大流为初始化的最大流。

Algorithm 2：Edmonds-Karp

Input：S　V　Graph

Output: max_flow

```
1. Initial the Graph
2. While True:
3.     Parent, flow = BFS(G, S, T)
4.     If not path:
5.         Break
6.     V=T
7.     While V！=T：
8.         U=parent[V]
9.         Flow(U,V) - = flow
10.        Flow(V,U)+ = flow
11.        V=U
12. Max_temp=0
13. For each edge(S, V) in G:
14.     Max_temp+=Flow(S, V)
15. Max_Flow=Max_Flow-Max_temp
16. Return Max_Flow
```

（2）Dinic 算法

算法思想：

Dinic 算法的思想是通过分阶段在层次网络中进行增广，与 Edmonds-Karp 算法不同。Edmonds-Karp 算法在每个阶段执行完一次 BFS 增广后，需要重新从源点开始进行另一轮 BFS 来寻找增广路；而在 Dinic 算法中，只需通过一次 DFS 过程即可实现多次增广。

Dinic 算法的 DFS 与普通的 DFS 不同，不是只要存在可行边就递归，而是需要逐层进行。因此，除了残留网络，Dinic 算法还需要一个层次网络，即为每个节点定义一个层次，表示该节点到源点的最短距离。这一步操作实际上与 BFS 寻找最短路径类似，确保了每次 DFS 找到的增广路径的长度是相同的。同时，这也保证了每次更新残留网络后，下一次找到的增广路径长度必定是递增的，从而确保了算法的有穷性。

算法流程：

使用 BFS 建立分层图：注意，分层图是基于当前的残留网络建立的，因此需要重复建立分层图。

使用 DFS 寻找增广路径：在分层图中，用 DFS 方法寻找一条从源点到汇点的路径，获得这条路径的流量 Va （即路径上残量最小的边的残量）。根据这条路径修改整个图，将所经之处的正向边流量减少 Va，反向边流量增加 Va 。注意， Va 是非负数。

重复步骤 2：持续使用 DFS 寻找新的增广路径，直到无法找到新的路径为止。

重复步骤 1：在图无法再进行分层之前，重复建立分层图和寻找增广路径的过程。

示例：

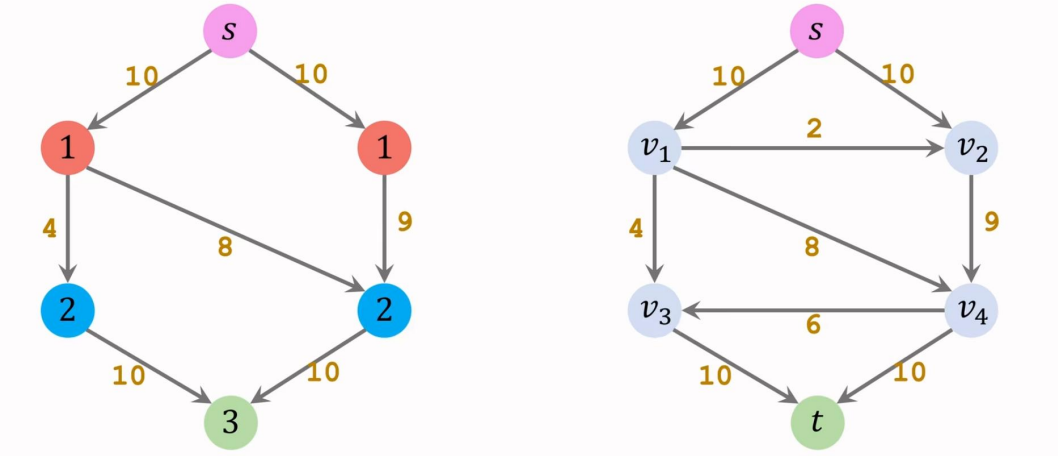

图 14 Dinic 示例

如图 14，右边为流网络原图，左边为根据右图使用 BFS 算法创建的层次图。

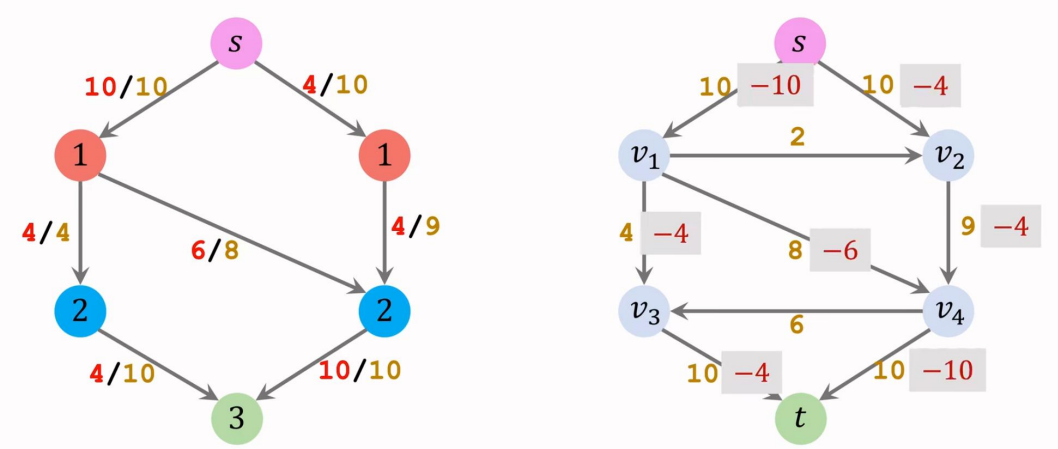

图 15 寻找阻塞流

如图 15， 我在层次网络上使用 DFS 算法寻找阻塞流，然后对右边网络原图上的相应边的权值进行更新。

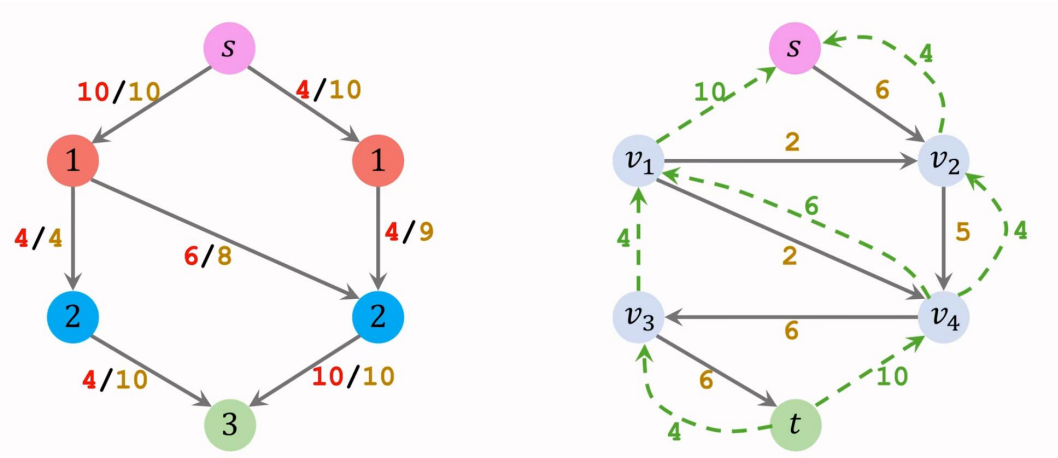

图 16 添加反向边

如图 16，我添加反向边，这与 FF 算法和 EK 算法的思路一致。

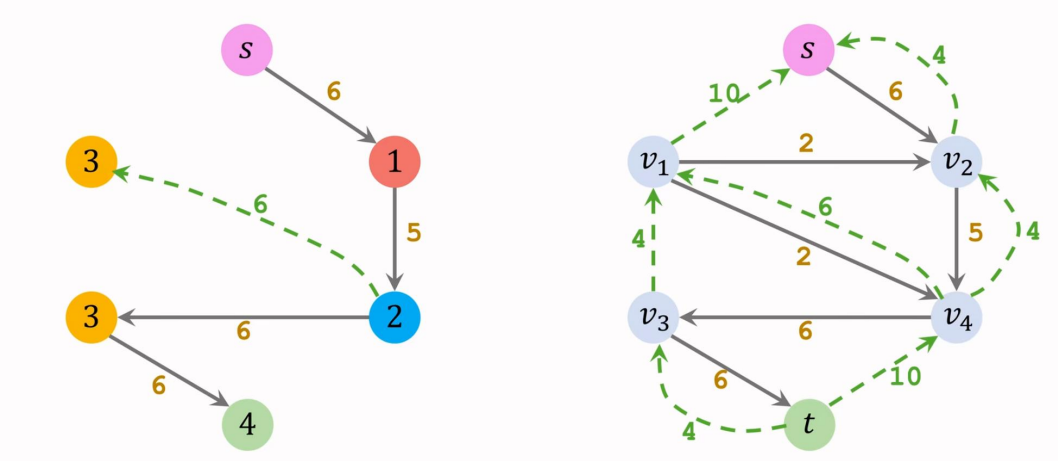

图 17 重复前面步骤

如图 17，我对更新后的网络重新创建一个层次图，注意反向边也是可以使用的。

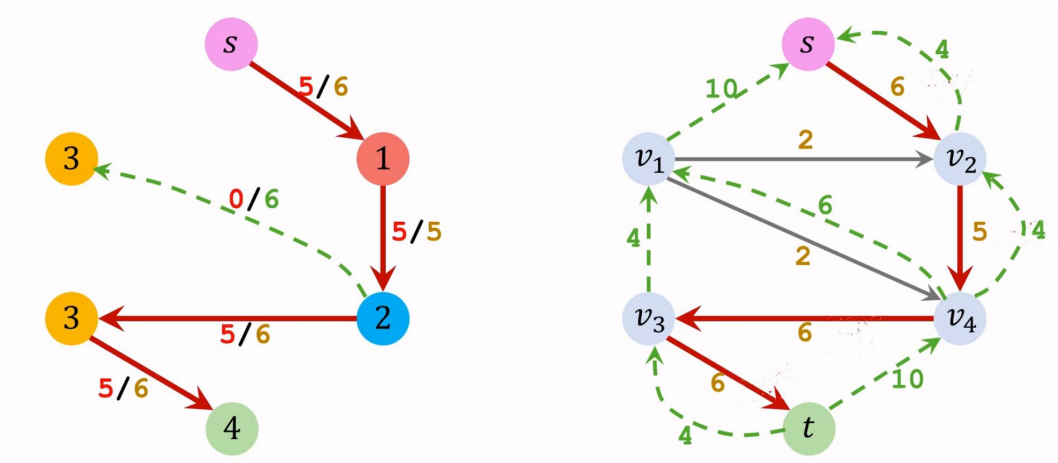

图 18 寻找阻塞流

与前面的步骤相同，再次使用 DFS 寻找阻塞流，然后更新原图。

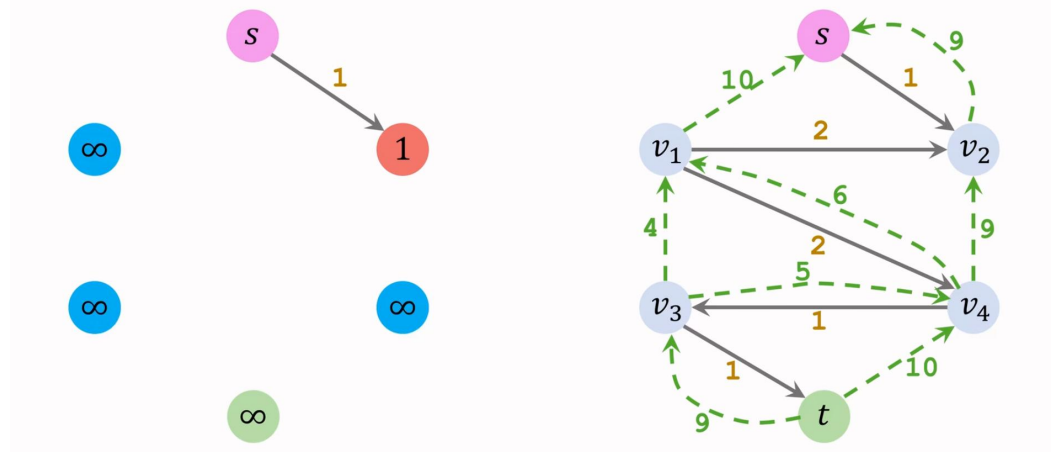

图 19 结束循环

如图 19，我们已经没有办法在层次图上找到阻塞流了，于是程序终止。

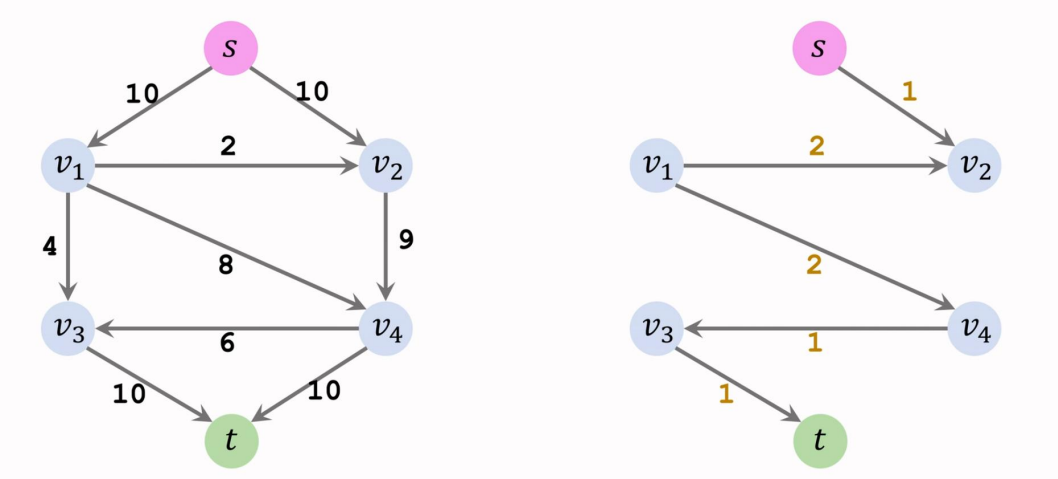

图 20 计算最大流

最大流的计算和 FF 算法以及 EF 算法是一致的，只需要使用 10+10-1，即使用原图中从 S 流出的总流量减去更新后网络图中从 S 流出的总流量即可。

伪代码：

Algorithm 3：Dinic

Input：S V Graph
Output: max_flow

```
1. Initial the Graph
2. While True:
3.     Level_Graph = BFS(Graph,S,T)
4.     If Level_Graph not exist:
5.         Break
6.     While True:
7.         Max_Block_flow = DFS(Level_Graph,S,T)
8.         If Max_Block_flow not exist:
9.             Break
10. For each edge(S, V) in G:
11.     Max_temp+=Flow(S, V)
12. Max_Flow=Max_Flow-Max_temp
13. Return Max_Flow
```

代码说明：

先初始化网络图，之后使用 BFS 算法进行构建层次图，对于层次图，使用 DFS 算法寻找阻塞流，同时更新网络图，循环直到不能找到阻塞流，最后使用原图中从 S 流出的总流量减去更新后网络图中从 S 流出的总流量即可计算出最大流。

时间复杂度分析：

设残留网络中顶点数为 V，边数为 E。

BFS 建立层网络：耗时 O(E)（与 Edmonds-Karp 算法中的 BFS 时间消耗一致）。
DFS 寻找增广路径：在一次 DFS 中，搜索出的增广路径包含关键边，且关键边的个数为 O(E)。因此，一次 DFS 可能搜索出 O(E) 条增广路径，对残留网络进行修改需要 O(V) 的时间，所以一次 DFS 的时间消耗为 O(VE)。
增广路径长度递增：与 Edmonds-Karp 算法相似，Dinic 算法中层网络的限制保证了每次 DFS 找到的增广路径长度是递增的，且最大长度为 O(V)。
综上所述，Dinic 算法的时间复杂度为： $O(V)\times(O(E)+O(VE))=O(V)\times O(VE)=O(V^2E)$。

5. 效率分析

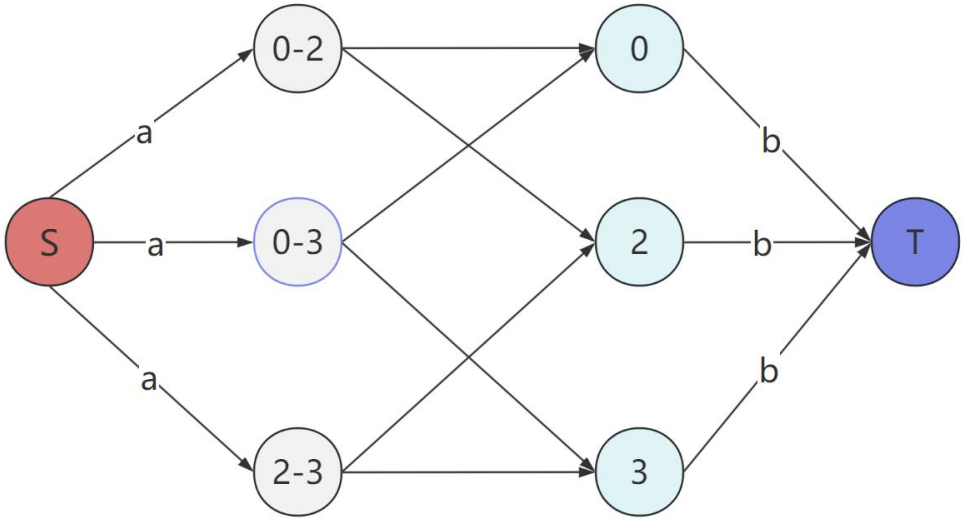

图 21 示例图

以图 21 为例，设置四个参数，分别为
a：代表从 S 流出的边的权重
b: 代表流入 T 点的边的权重
m：代表与 S 点直接相连的点的个数
n：代表与 T 点直接相连的点的个数

（1）设置 a=5，b=25，n=500，改变 m 的值

表一：时间随 m 值变化结果（ms）

m	EK	Dinic
100	43.83	1.76
200	161.41	1.12
300	406.00	1.22
400	784.12	1.74
500	1241.85	2.02

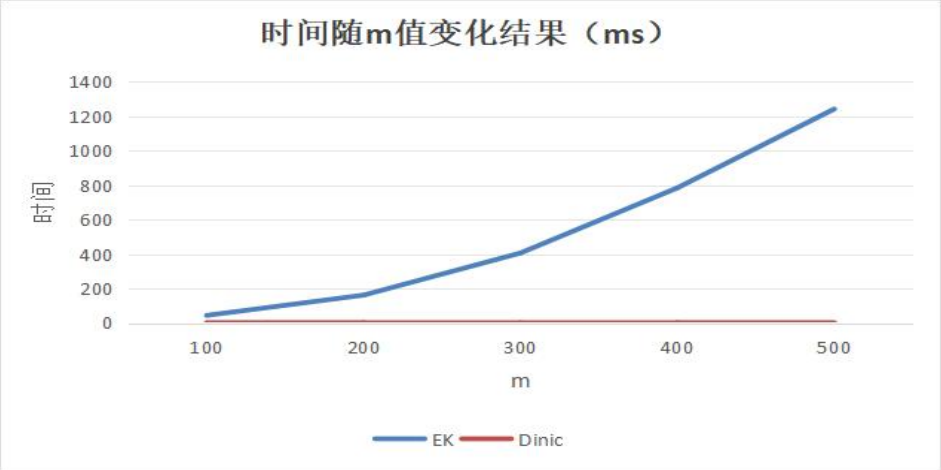

图 22 时间随 m 值变化结果

可以看出，Dinic 的时间效率比起 EK 算法高了很多。

（2）设置 a=5，b=25，m=500，改变 n 的值

表二：时间随 n 值变化结果（ms）

n	EK	Dinic
100	599.51	2.42
200	751.12	1.52
300	956.45	1.65
400	1075.95	2.24
500	1294.86	2.95

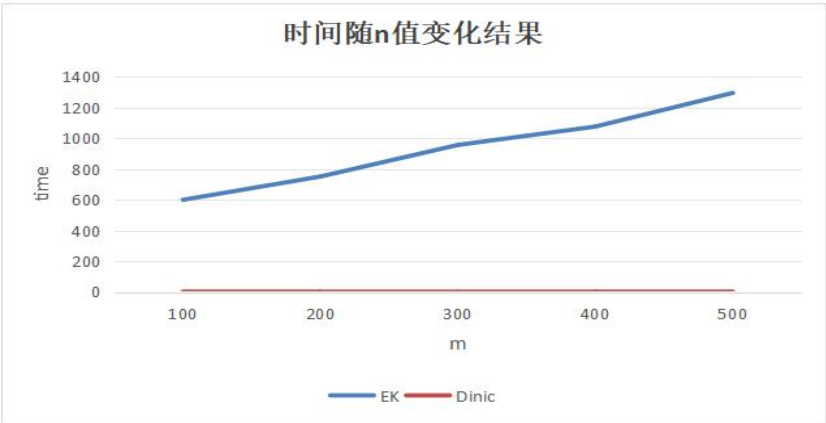

图 23 时间随 n 值变化结果

可以看出，Dinic 的时间效率比起 EK 算法高了很多。

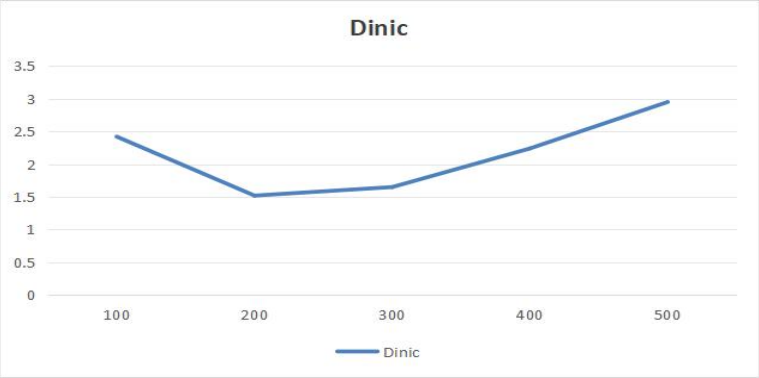

图 24 Dinic 时间随 n 值变化结果

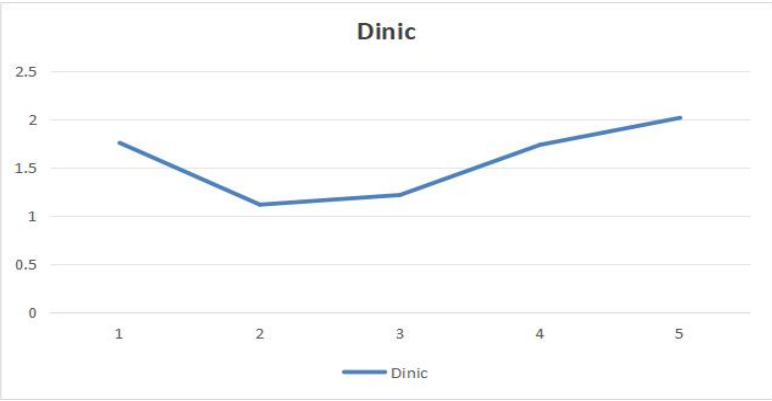

图 25 Dinic 时间随 n 值变化结果

我们可以发现，当 n 或者 m 的值比较小的时候，Dinic 算法的时间反而要长一点，这可能是：

• 初始化和辅助结构构建：对于较小规模的图，Dinic 算法构建的层次图和其他辅助数据结构可能会相对较大，导致初始化和构建的开销较高。而在大规模图上，这些开销相对较小，因为较大规模的图通常有更多的边和顶点，从而使得这些辅助结构更高效。

• 迭代次数：在较小规模的图上，Dinic 算法可能需要较多的迭代次数才能找到最大流，因为较小规模的图可能具有更多的路径选择和调整，从而增加了算法的运行时间。

• 算法本身特性：Dinic 算法的性能与图的结构密切相关。一些特定类型的图可能会导致 Dinic 算法在较小规模上表现不佳，例如具有大量短路径或较小容量边的图。

五、实验结果及分析：

我使用了 Ford - Folkerson（FF）算法，Edmonds - krap（EK）算法和 Dinic 算法解决棒球比赛淘汰问题。都可以得出结果：Atlanta，New York 可能夺冠，Philly 和 Montreal 被淘汰，三种算法都是正确的。由于图的规模太小，三种算法在图上的时间没有差别，都是接近 0ms。

但是相比 EK 算法以及 FF 算法，Dinic 算法有其优势：

时间复杂度更优：Dinic 算法的时间复杂度为 $O(V^2 * E)$，而 Edmonds-Karp 算法的时间复杂度为 O(V * E^2)。在密集图（边数接近顶点数的平方）中，Dinic 算法通常比 Edmonds-Karp 算法更快。这是因为 Dinic 算法利用层次图和 BFS 构建增广路径，从而减

少了不必要的搜索和迭代次数。
增广路径选择更优：Dinic 算法在每次迭代中选择残余容量最大的边作为增广路径的一部分，因此在每次迭代中增加更多的流量。这使得 Dinic 算法的收敛速度更快，可以更快地找到最大流。
不会出现返流：Edmonds-Karp 算法在每次迭代中使用 BFS 选择增广路径，可能导致残余图中出现返流边，需要对返流边进行反复搜索和调整。而 Dinic 算法使用层次图构建增广路径，保证了每次迭代中不会出现返流边，从而减少了不必要的操作。
空间效率更高：Dinic 算法使用层次图作为辅助数据结构，而不需要维护整个残余图。这使得 Dinic 算法的空间复杂度较低，只需额外 O(V)的空间。

因此，Dinic 为更加高效的算法。
后续，我也通过实验证实了，Dinic 算法确实是更加高效的算法。

六、实验结论：
本次实验，我使用了 Ford - Folkerson 算法，Edmonds - krap 算法和 Dinic 算法解决棒球比赛淘汰问题。
棒球比赛问题可以简化为：
目标：确定是否存在一种比赛结果分配，使得某支球队能达到或超过特定的胜场数（即确认这支球队是否仍有可能成为分区冠军）。
比赛和胜场数限制：每场比赛的结果必须分配给一支参赛的球队，每支球队的胜场数不能超过其剩余的最大可能胜场数。
转换为最大流问题
比赛结果分配模型：将比赛结果分配建模为流网络中的流动，流量代表比赛的胜利被分配给特定的球队。

将问题原型转换成最大流问题之后，就可以使用上诉的算进行解决了。
Ford - Fulkerson 算法通过不断寻找增广路径并增加路径上的流量来增加网络的总体流量。增广路径是指从源节点到汇节点的路径，其上的边的剩余容量大于零。通过不断寻找增广路径并增加流量，直到无法找到增广路径为止，就可以得到最大流。Ford - Folkerson 算法的时间复杂度为 O（f×m），其中 f 为最大流，m 为图中边的数量。因此在最差的情况下，Ford - Folkerson 算法需要运行的次数等于最大流 f，并且每一次寻找从源点到汇点的路径的时间复杂度为 O（m），于是就有时间复杂度为 O（f×m）。
Edmonds - Karp 算法是 Ford - Fulkerson 算法的一种改进版本，用于解决最大流问题。与 Ford - Fulkerson 算法不同的是，Edmonds - Karp 算法使用广度优先搜索（BFS）来选择增广路径，从而保证每次找到的增广路径是最短路径。Edmonds - Karp 算法的时间复杂度为 $O(V\cdot E^2)$，其中 V 是节点数，E 是边数。
Dinic 算法的思想是通过分阶段在层次网络中进行增广，与 Edmonds - Karp 算法不同。Edmonds - Karp 算法在每个阶段执行完一次 BFS 增广后，需要重新从源点开始进行另一轮 BFS 来寻找增广路；而在 Dinic 算法中，只需通过一次 DFS 过程即可实现多次增广。Dinic 算法的时间复杂度为： $O(V)\times(O(E)+O(VE))=O(V)\times O(VE)=O(V^2E)$。

最后测试结果为 Atlanta，New York 可能夺冠，Philly 和 Montreal 被淘汰，后续通过对于不同规模网络图的测试也可以证明 Dinic 算法的优势。

指导教师批阅意见： **成绩评定：** **指导教师签字：** **2024 年　　月　日**
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。