**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与分析**

**实验名称： 实验6 最大流应用（棒球比赛问题**）

**学院： 计算机与软件学院 专业：计算机科学与技术（创新班）**

**报告人： XXX 学号： XXXXXXXXXX 班级： 高性能班**

**同组人：**

**指导教师： 杨烜**

**实验时间： 2024.6.10——2024.6.15**

**实验报告提交时间： 2024.6.15**

**教务处制**

**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、内容**



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。

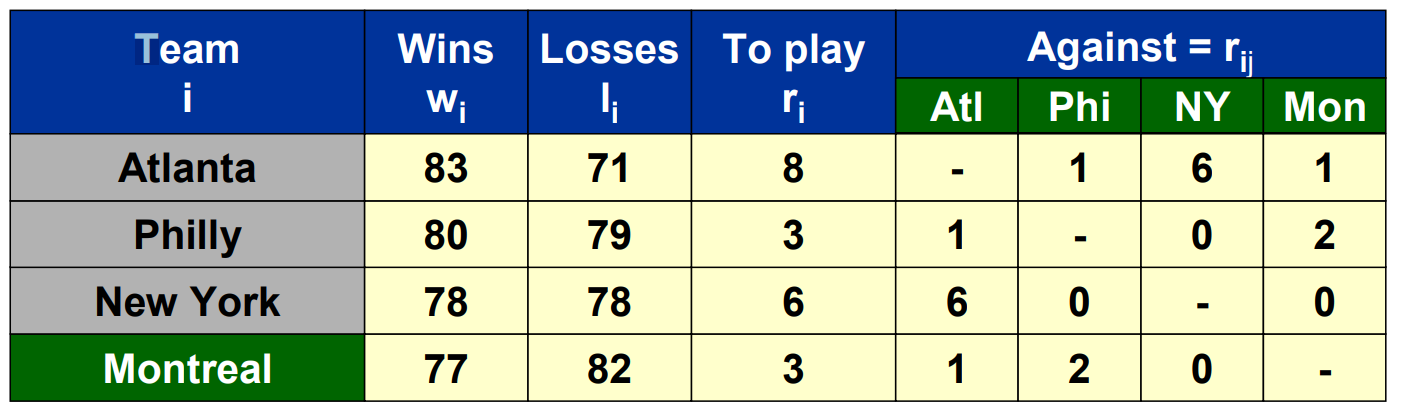
在美国职业棒球的例行赛中，每个球队都要打 162 场比赛（对手包括但不限于同一分区里的其他队伍，和同一队伍也往往会有多次交手），所胜场数最多者为该分区的冠军；如果有并列第一的情况，则用加赛决出冠军。在比赛过程中，如果我们发现，某支球队无论如何都已经不可能以第一名或者并列第一名的成绩结束比赛，那么这支球队就提前被淘汰了（虽然它还要继续打下去）。从上面的例子中可以看出，发现并且证明一个球队已经告败，有时并不是一件容易的事。

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。”



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。

**三、实验要求**

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、求解问题的算法原理描述**

**1、如何判断球队被淘汰**

* 简单淘汰

若球队的最大可能胜利场数小于球队已经取得的胜利场数，即，则球队必然没有夺冠的可能，可以直接被淘汰。

以上面四个球队的比赛情况为例，Montreal最大可能胜利场数为77+3=80，小于Atlanta已经取得的胜利场数83，则Montreal胜利场数一定不超过Atlanta，直接失去夺冠的可能。

* 非简单淘汰

不管剩余比赛场数的输赢结果如何（也可以理解为不管剩余比赛场数中的胜利场数如何分配到现有队伍中），队伍最大可能胜利场数都会小于某个队伍最终胜利场数，则队伍被淘汰。反之，若存在一种胜利场数分配方案使得队伍的最大可能胜利场数不被任何一个球队超越，则队伍存在夺冠可能，所以可以不被淘汰。

以上面四个球队的比赛情况为例，如果Philly在其剩余的3场比赛均取得胜利，则最终Philly的胜场数为83，此时Atlanta胜场数为83，New York的胜场数为78，且这两者间还剩6场比赛要进行。

假如6场比赛中Atlanta一旦有一场比赛胜过New York，Atlanta的胜场数就达到84，大于Philly的胜场数83，Philly的胜场数必然不可能再超越Atlanta，故被淘汰。

假如6场比赛中Atlanta全部输于New York，则New York的胜场数达到84，大于Philly的胜场数83，Philly的胜场数必然不可能再超越New York，故被淘汰。

上面描述，可以理解为6场比赛中一定会在这两个球队中产生6个胜场，而我们需要对这6个胜场在Atlanta和New York这两个球队中进行分配，如果将大于等于1个胜场数分配给Atlanta，则Atlanta的胜场数就大于等于84，大于Philly的胜场数83，Philly被淘汰；

如果将全部胜场数分配给New York，则New York的胜场数就达到84，大于Philly的胜场数83，Philly同样被淘汰。

因此对于Philly来说，尽管它的胜利场数达到最大，但由于不管如何分配胜利场数到剩下的队伍中，都存在队伍的最终胜场数大于Philly，所以Philly就失去了夺冠的可能，最终被淘汰。

综上看来，简单淘汰很容易实现，但非简单淘汰则要考虑各球队间比赛剩余场数、输赢的可能、目前的胜利场数等等因素，来进行剩余胜利场数的分配（剩余胜利场数等于剩余比赛场数，剩余胜利场数必然从剩余比赛场数中产生），对于这个复杂的问题，有什么算法可以用于求解？网络流结构可以很好地适配这个剩余胜利场数分配问题。

**2、流网络的构造原理**

1）流网络的4种节点

* 源节点：表示起点，用于连接所有的比赛节点。
* 比赛节点：对两个球队间未进行的比赛设一个节点，表示该比赛
* 球队节点：对每个球队设一个节点，表示该球队。
* 汇节点 ：表示终点，用于连接所有的球队节点。

以上面四个球队的比赛情况为例，会产生以下10个节点，其中1个节点为源节点s，1个节点为汇节点t，4个比赛节点Atlanta-Philly、Atlanta-New York、Atlanta-Montreal、Philly -Montreal，4个比赛节点Atlanta、Philly、New York、Montreal。

图示

描述已自动生成

2）流网络的3种连接关系

源节点到比赛节点、比赛节点到球队节点、球队节点到汇节点间的连接关系各不同。

图示

描述已自动生成

从源节点到比赛节点：每条边代表未进行的比赛数，也代表需要分配的剩余胜利场数，边的容量为比赛节点队伍和队伍之间剩余比赛场数。

例如，Atlanta和Philly之间还有1场比赛未进行，所以源节点s到比赛节点Atlanta-Philly的容量为1。

图示

描述已自动生成

从比赛节点到球队节点：表示两个球队间比赛的胜负结果会影响这两支球队，也可以表示这两个球队间比赛的胜场会分配到这两支球队，边的容量为无穷大（表示胜场可以自由流动，不限制比赛结果）。

图示

描述已自动生成

从球队节点到汇节点：代表保证球队不被淘汰下球队最大剩余胜利场数，边的容量为(假设)。

图示

描述已自动生成

以Philly为例子，希望知道Philly在达到最大胜利场数的情况下是否有夺冠的可能，则，所以Atlanta、New York、Montreal最大剩余胜利场数分别为0、5、6。

图示

描述已自动生成

**3、为什么网络流结构适合该问题**

在前面的流网络构造中，我们能看到，流网络中的流相当于剩余胜场数。从源节点到比赛节点边的容量，相当于有多少剩余胜场数需要分配；从比赛节点到球队节点，相当于将胜场分配给哪个球队；从球队节点到汇节点边的容量，相当于对前面胜场分配的限制，保证球队的最终胜场数不超过球队的最大可能胜利场数。因此，剩余胜场数从源节点流出，若能到达汇节点，说明该场胜利可以分配到某支球队且能保证球队𝑖最大可能胜利场数不被其他球队超越，即球队𝑖不被淘汰。

接下来，用网络流来判断Philly是否被淘汰，进而展示网络流的工作过程。

如Atlanta和Philly之间存在1场比赛，两个球队间必然会有一个球队获得1场胜利，在保证Philly不淘汰的情况下，这场胜利只能分配给Philly。

这是因为该场胜利从源节点出发，由于球队节点Atlanta到汇节点边的容量为0，所以该场胜利无法从源节点出发经过球队节点Atlanta到汇节点，即无法在保证Philly不被淘汰的情况下，将该场胜利分配给Atlanta；又由于Philly到汇节点边的容量为3，所以该场胜利可以从汇节点经过球队节点Philly到达汇节点，即可以在保证Philly不被淘汰的情况下，将该场胜利分配给Philly。

手机屏幕截图

描述已自动生成

又如Atlanta和New York之间存在6场比赛，则说明有6场剩余胜利场数需要分配，可以发现在保证Philly不淘汰的情况下，只能分配好5场比赛，即全部分配给New York。

这是因为这6场胜利从源节点出发，由于球队节点Atlanta到汇节点边的容量为0，所以6场胜利中的任意1场胜利无法从源节点出发经过球队节点Atlanta到汇节点，即无法在保证Philly不被淘汰的情况下，将这6场胜利中的任意1场胜利分配给Atlanta；由于球队节点New York到汇节点边的容量为5，所以6场胜利中的任意5场胜利可以从汇节点经过球队节点New York到达汇节点，即可以在保证Philly不被淘汰的情况下，将6场胜利中的任意5场胜利分配给New York。

将5场胜利分配给New York后，球队节点New York到汇节点边的剩余容量为0，无法再将胜利经过球队节点New York送到汇节点，说明此时剩余胜利场数无法从再经过Atlanta和New York两个节点到达汇节点。

显然，这6场胜利只有5场胜利可以分配给球队且保证Philly不被淘汰，剩余的1场胜利无论如何都到达不了汇节点，即剩余的1场胜利无论分配给Atlanta还是New York，都会导致Philly被淘汰，这和我们前面非简单淘汰的推断是一致的。

图示

描述已自动生成

若存在剩余胜利场数无法完全分配的情况，则说明不管剩余比赛输赢结果如何，都不能保证Philly被淘汰，换句话说，Philly必然被淘汰。

综合上述过程，我们能确定，将棒球淘汰问题应用到网络流结构中：

* 网络中的流 = 待分配的剩余胜利场数。
* 待分配的剩余胜利场数 = 从源节点出发边的容量和。
* 从源节点出发到汇节点的流的总量 = 在保证球队不被淘汰的情况下，可分配给各球队的剩余胜利场数。
* 求该网络流的最大流，实际是在保证球队𝑖不被淘汰的情况下，求最大的可分配给各球队的剩余胜利场数。
* 如果**网络的最大流（最大的可分配给各球队的剩余胜利场数） = 剩余比赛场数**，说明存在一种方案，使得全部剩余胜利场数分配给各球队后，能保证球队𝑖不被淘汰
* 如果**网络的最大流（最大的可合理分配的剩余胜利场数） < 剩余比赛场数**，说明不存在任何方案，使得全部剩余胜利场数分配给各球队后，能保证球队𝑖不被淘汰，即球队𝑖必然被淘汰

1. **求解最大流问题的算法**

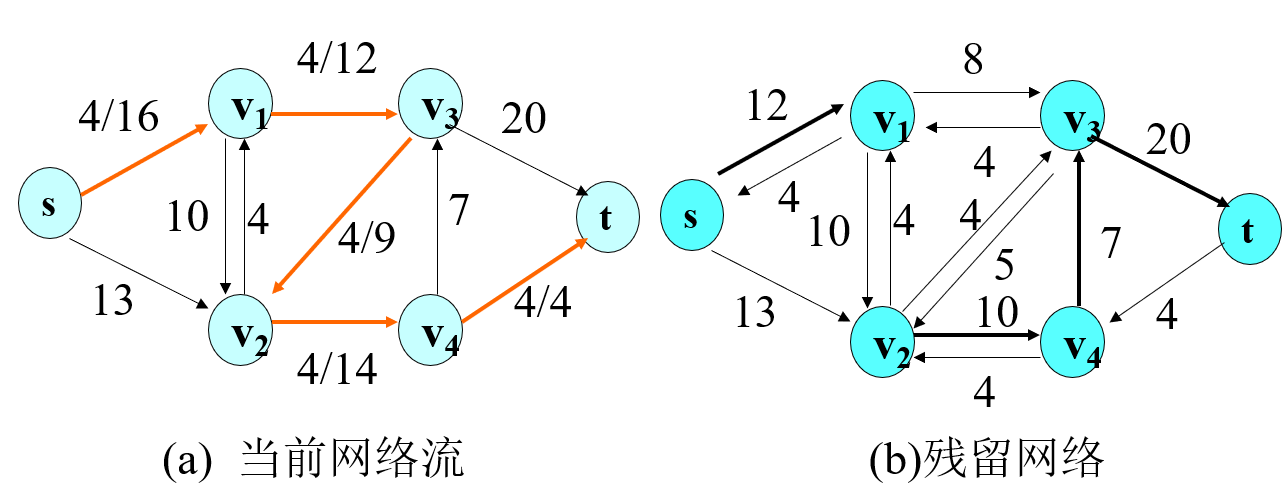
**1）Ford-Fulkerson**

**算法原理：**

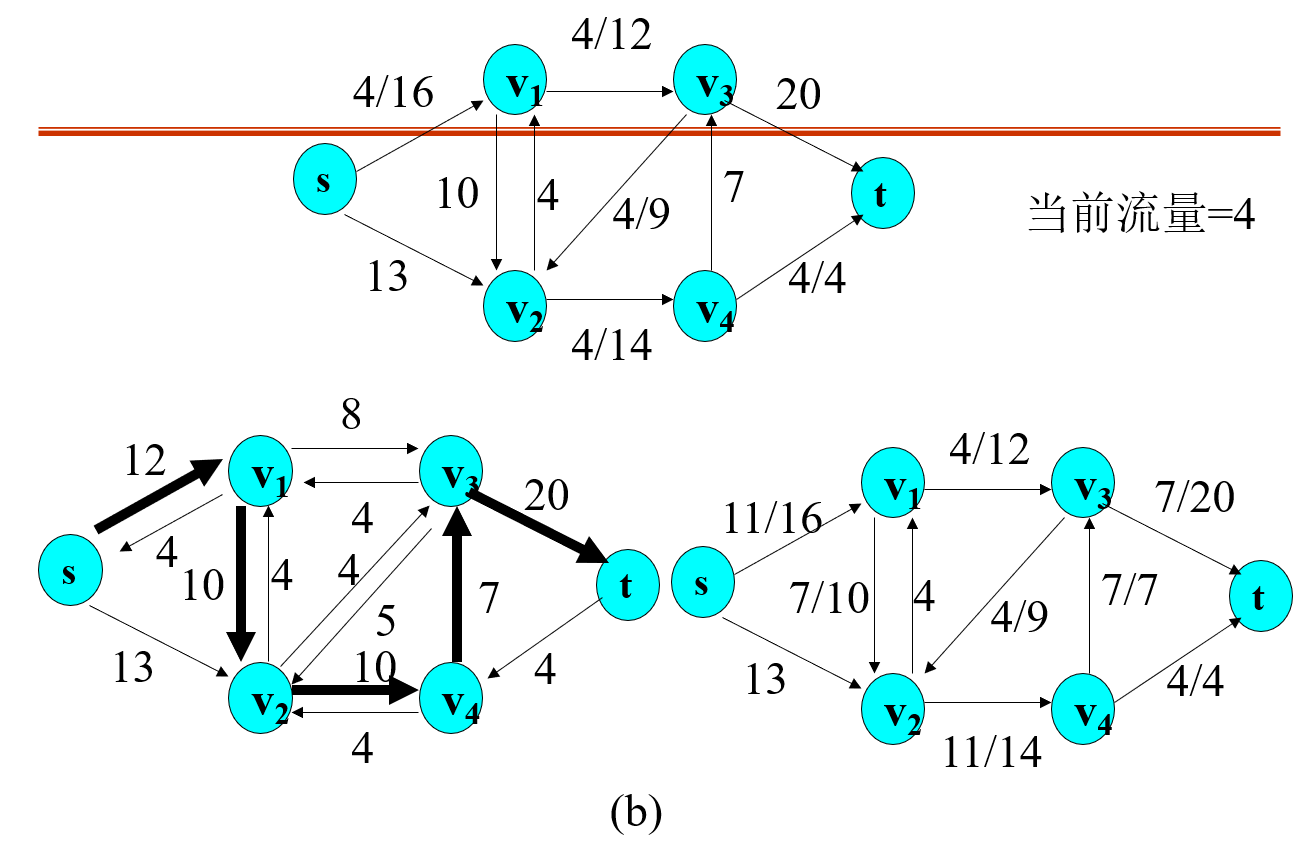
Ford-Fulkerson算法通过逐步调整流量，寻找增广路径，直到没有剩余增广路径，从而计算出最大流。

增广路径的找寻方法：利用深度优先搜索（DFS），从源节点出发，沿着当前容量大于0的边进行递归搜索，直到找到一个通往汇节点的路径。如果找到路径，则记录路径中的最小边容量作为可增量，并沿路径调整各边的流量；如果没有找到，则回溯并尝试其他路径。

Ford-Fulkerson算法在找寻增广路径前，需要将网络流转化为残留网络，这是因为残留网络反映了当前流量状态下各边的剩余容量和反向容量，从而支持对流量的调整。残留网络中的每条正向边表示从一个节点到另一个节点的剩余可用容量，每条反向边表示能够减少流量的容量。



增广路径是指在流网络中从源节点到汇节点之间的路径，这条路径上的每条边都有正的剩余容量（即在这条路径上还可以增加流量）。用DFS在残留网络中寻找从源节点到汇节点的路径，并且确保找到的路径上每条边的剩余容量都大于零。接下来沿着找到的增广路径增加流量，根据路径上的瓶颈容量（路径上最小的剩余容量）调整流量，更新路径上的正向和反向边容量。继续在更新后的残留网络中寻找增广路径，直到找不到新的增广路径为止，表示已达到最大流量。



**割的相关概念：**

图示

描述已自动生成

* 割：指的是一种边的集合，如果移除这个集合的全部边，就会断开源点和汇点的连接。
* 割的容量：s-t割C=(S,T)将图完全划分为和两部分，使得，组成割的边集为，在割中的边的容量用表示，则s-t割的容量 = 。
* 最小割：割容量最小的割。

手表的卡通人物

描述已自动生成

**最大流最小割定理：**

在一个流网络中，从源节点到汇节点的最大可实现流量等于从源点到汇点的最小割的容量，即最大流=最小割。

简单理解：

从源节点到汇节点需要经过割中的边，如果把割中的边看作是管道，希望从S到T的流尽可能大，就需要把所有从S到T的通道灌满，且所有从T到S的通道不通流量，而灌满所有从S到T的通道的流量则是当前割的通过的最大流量。如下图，割将图分成S和T两部分，其中源节点包含在S，汇节点包含在T，割包含四条边，从S到T的边的总容量为，从T到S的边总容量为，为了确保源节点到汇节点的流尽可能大，相当于要确保S到T的流尽可能大，故在当前割下从S到T的流量要达到，从T到S的流量为0。

门上写着字

中度可信度描述已自动生成

但割的方式有多种，每种割能通过的最大流量不同，为了保证图中的各通道不会因为流量而超过其容量限制，故图的最大流量只能取每种割能通过的最大流量的最小值。例如，下图中能够得到三条割（当然图中的割不止这三条），每条割的容量分别为、、，并且。为了确保从源节点出发的流能全部到达汇节点并希望流尽可能大，故流量最大取。

图示, 示意图

描述已自动生成

数学证明：

文本

低可信度描述已自动生成

文本, 信件

描述已自动生成

文本, 信件

描述已自动生成

**时间复杂度分析：**

* 路径查找：使用DFS查找增广路径时，每次搜索的时间复杂度是 ，其中是网络流图中边的数量。
* 迭代次数：最多需要增加的流量与路径有关。在每次找到增广路径时，路径上的最小剩余容量（瓶颈容量）至少为1。假设最大流的总值为（流量总和），每次找到一条增广路径至少可以增加1单位流量。因此，最多需要次增广路径查找。
* 所以，Ford-Fulkerson算法的时间复杂度。
* 假设球队数为n，则需要进行n次Ford-Fulkerson算法，所以算法整体复杂度为。

**代码解释：**

初始化残留网络：

文本

描述已自动生成

rGraph是残留网络的表示，用于追踪每条边上的剩余容量。通过遍历所有节点对 (u, v)，将输入的流图graph的容量复制到rGraph中。

寻找增广路径并更新最大流：

图形用户界面, 应用程序, Word

描述已自动生成

使用memset将visisted数组重置为0，以便在每次搜索增广路径时重新标记节点。调用dfs函数在残留网络中从源点s到汇点t寻找增广路径。dfs函数返回找到的增广路径的流量pathFlow。如果pathFlow为0，表示无法找到增广路径，最大流量计算结束，返回 maxFlow。如果找到增广路径，将pathFlow累加到 maxFlow 中，并继续寻找新的增广路径。

更新残留网络：

图形用户界面, 应用程序, Word

描述已自动生成

在dfs回溯时，如果找到有效路径，减少边u->v的容量，增加边v->u的回退流量。

1. **Edmond-Karp**

**算法原理：**

Edmond-Karp算法通过逐步调整流量，寻找增广路径，直到没有剩余增广路径，从而计算出最大流。

增广路径的找寻方法：利用广度优先搜索（BFS）寻找增广路径的方法是从源节点开始，按层次遍历图中的边，寻找所有可能的路径中容量大于0的边，直到找到通往汇节点的路径。BFS保证找到的增广路径是层次最浅的，即边数最少的，通过不断更新边的流量，层层推进，最终找到从源到汇的增广路径。

相对于Ford-Fulkerson算法，Edmond-Karp算法在某些复杂网络拓扑寻找增广路径方面具有显著优势。

例如下面这张图，可以看到，只要找到S -> B -> E -> T这条增广路径，就能找到该网路的最大流99。假设使用Ford-Fulkerson算法找网络最大流，则会使用DFS来找增广路径，那么首先会找到S -> A -> C ->……->D -> E -> T这条增广路径，增广1的流量，因为在DFS搜索时出现分支，DFS会优先选择搜索索引靠前的节点，故在S节点处往下搜索时，会往A节点处向下搜索，而错失了S -> B -> E -> T这条高容量且距离短的路径，增加了不必要的计算。而如果使用Edmond-Karp算法找网络最大流，则会使用BFS来找增广路径，首先找到的增广路径将会是距离最短的S -> B -> E -> T这条增广路径，直接增广99的流量。

因此，**Edmonds-Karp**的BFS保证了每次增广路径的最短性，这在某些网络结构中，可以避免因低容量路径导致的低效流量调整。

钟表的特写

描述已自动生成

**时间复杂度分析：**

* 路径查找：使用BFS查找增广路径时，每次搜索的时间复杂度是 ，其中是网络流图中边的数量， 是网络流图中顶点的数量。
* 增广路径的查找总次数：由于每条最短路径的流量增加会使得至少一条边的剩余容量减少（或从正变为零），这意味着同一条路径不会被多次使用，所以在每次查找增广路径时，从源到汇的路径长度至少增加1，最坏情况下，从源到汇的路径长度最多增加次，即经过全部节点。总路径长度从1增加到，每个长度最多进行次查找。所以增广路径的查找总次数为。
* 所以，Edmond-Karp算法的时间复杂度。
* 假设球队数为n，则需要进行n次Edmond-Karp算法，所以算法整体复杂度为。

**代码解释：**

记录增广路径：

文本

中度可信度描述已自动生成

由于BFS找增广路径无法向DFS找到增广路径后通过回溯更新残留网络，所以我们需要用一个parent数组，记录增广路径上所有节点的父节点，如此将增广路径记录下来。

循环寻找增广路径并更新网络：

图片包含 文本

描述已自动生成

调用bfs函数寻找增广路径，bfs返回是否存在从s到t的路径，并将路径存储在parent 数组中。path\_flow初始化为无穷大，用于在后续步骤中找出路径上最小的剩余容量。

找到增广路径上的最小流量：

图形用户界面

低可信度描述已自动生成

通过parent数组回溯从t到s的路径，找到该路径上各边的最小剩余容量path\_flow。该值表示该路径上可以增加的最大流量。

更新残留网络和最大流：

文本

描述已自动生成

根据找到的path\_flow更新残留网络：从u到v的剩余容量rGraph[u][v]减少path\_flow。从v到u的回退流量rGraph[v][u]增加path\_flow。同时将path\_flow加到max\_flow，累加到当前的最大流量中。

3）**Dinic**

**算法原理：**

Dinic算法通过构造层次网络，将流网络分解为多个层次，每层节点只连接到下一层，从源到汇递进搜索所有增广路径，同时在层次网络中计算最大流。

首先使用广度优先搜索（BFS）构建层次图，再用深度优先搜索（DFS）寻找所有可行的增广路径，每次搜索只能从当前层向下一层搜索，并调整路径上的流量，通过反复构建和调整，直至没有增广路径时，完成最大流计算。

以下图为例，展示Dinic算法的解题过程。

图示

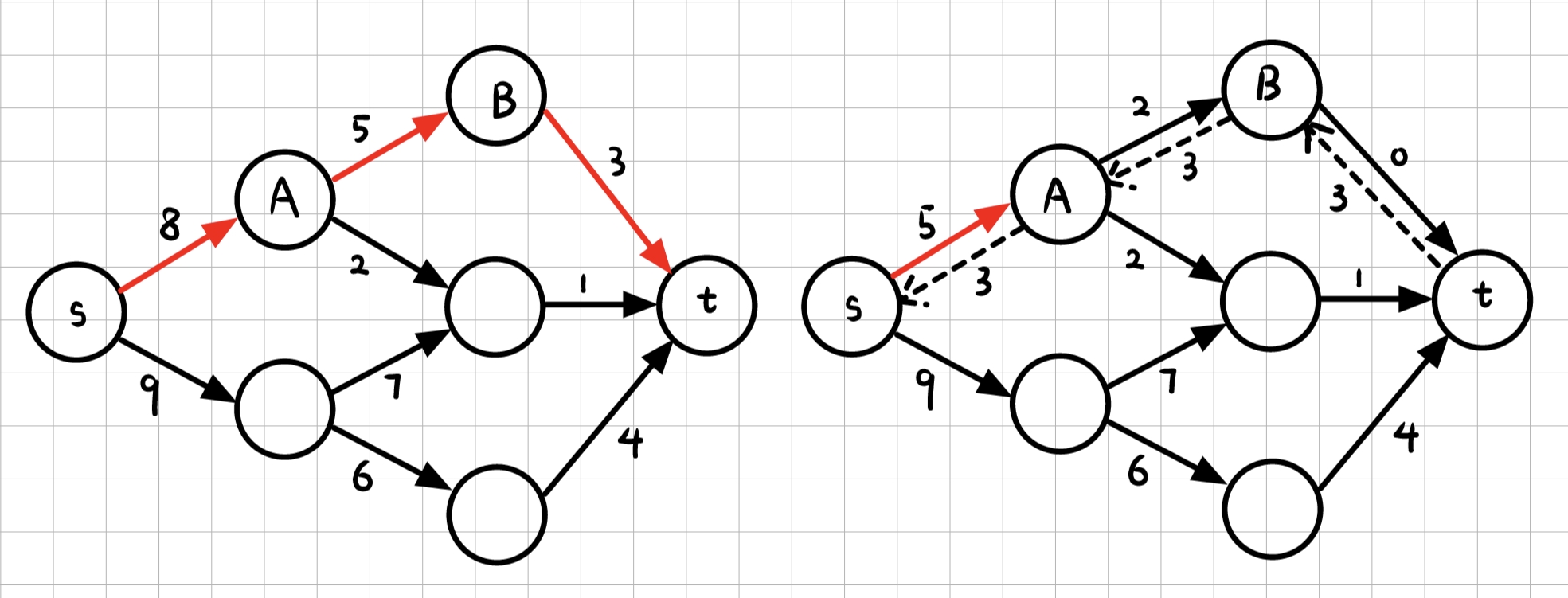
描述已自动生成

首先，使用BFS构建层次图，可以看到，下图被分成4个层次。

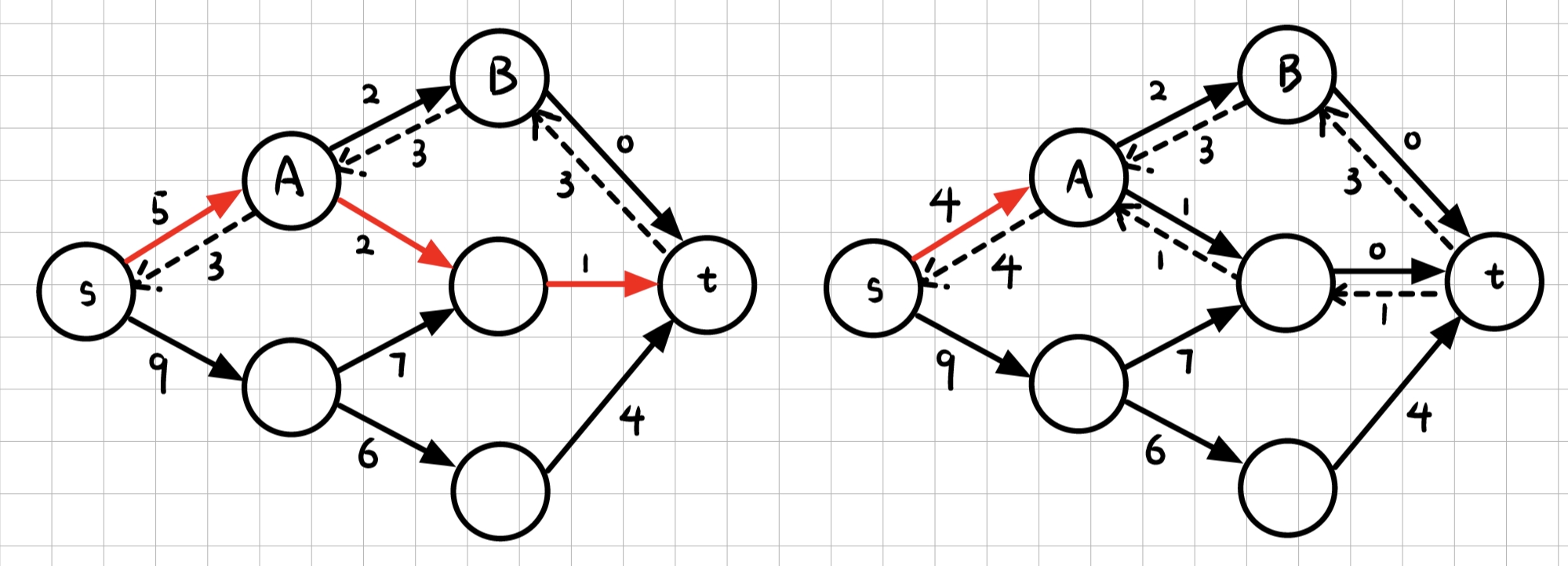
图示

描述已自动生成

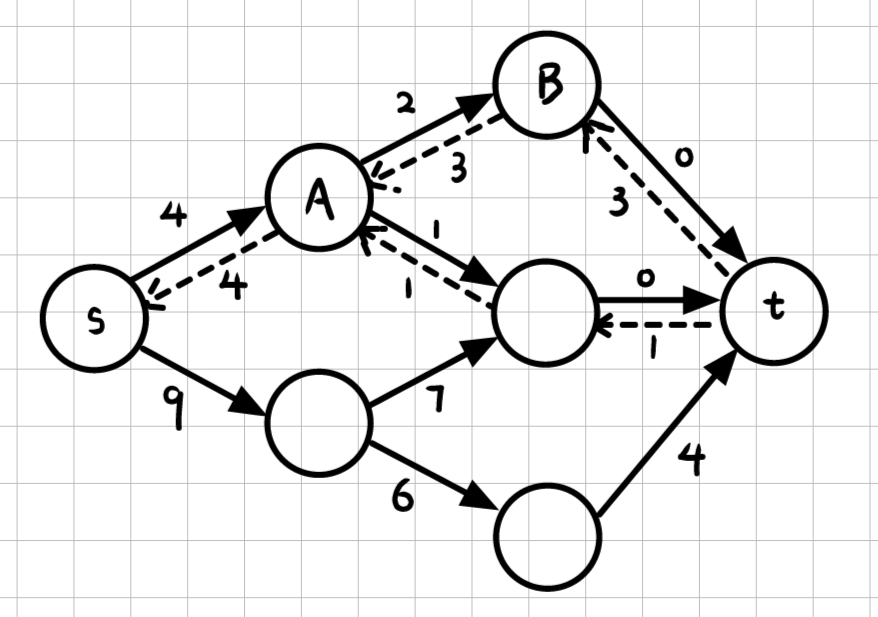
接下来，用DFS寻找增广路径，可以看到，路径上经过的节点的层次均不同，这便是Dinic算法中DFS找增广路径时的性质——只从当前层次向下一个层次搜索。找到源节点后回溯，更新路径上边的容量，回溯至产生分支的节点A。由于刚刚路径上经过的最小容量为3，所以路径上所有正向边的容量减3，反向边的容量加3。回溯到节点A并更新完路径容量后，查看到节点A的边容量是否大于0，若等于0，则继续回溯，若大于0，则继续往下搜索。可以看到，到节点A的边容量为5，大于0，所以继续向下搜索。



从节点A继续向下一个分支且下一个层次搜索，找到源节点后回溯，同样的，回溯过程中更新路径上边的容量。回溯到节点A，节点A的所有分支已经搜索完毕，故回溯。



从源节点出发到节点A的分支的增广路径搜索完毕，并且搜索过程中，所有边的容量均得到更新，接下来，会从源节点出发到下一个分支进行增广路径的搜索。



**与Edmond-Karp比较：**

由于其通过BFS构建层次图，搜索时也只从当前层向下一层进行搜索，所以它的搜索路径为最短路。

但它与Edmond-Karp算法不同之处是：最短增广路每个阶段执行完一次BFS增广后，要重新启动BFS从源点开始寻找另一条增广路;而在Dinic算法中，只需**一**次DFS过程就可以实现多次增广，这是Dinic算法的巧妙之处

**时间复杂度分析：**

* 构建分层网络：使用BFS构建分层网络时，时间复杂度是 ，其中是网络流图中边的数量， 是网络流图中顶点的数量。
* 增广路径查找：使用DFS寻找增广路径，DFS的时间复杂度为 。
* 构建分层网络次数：从源到汇的路径长度最多增加次，所以次数为
* 所以，Dinic算法的时间复杂度
* 假设球队数为n，则需要进行n次Edmond-Karp算法，所以算法整体复杂度为

**代码解释：**

循环构建分层网络并查找阻塞流：

图形用户界面, 文本, 应用程序

描述已自动生成

bfs函数用于构建分层网络，判断是否存在从源点s到汇点t的增广路径。在构建分层网络后，dfs函数查找阻塞流并计算可增加的流量tf（阻塞流是指从源点到汇点的路径，在这条路径上，不同层次的节点只按顺序访问且沿途边的剩余容量最大限度的减少）。当tf不为0时，增加tf到max\_flow，并更新残留网络中的容量。最终，当bfs找不到新的增广路径时，返回max\_flow作为最终的最大流量。

**4）二分+参数化最大流算法**

本方法参考于论文

**A NEW PROPERTY AND A FASTER ALGORITHM FOR BASEBALL ELIMINATION**

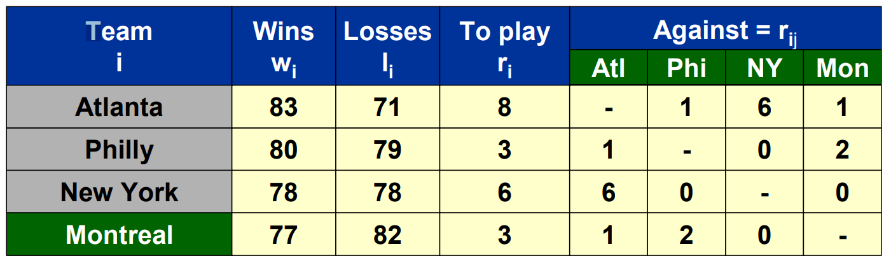
论文出处为<https://www.cs.princeton.edu/~wayne/papers/baseball.pdf>。

**前置知识：**

对于任意球队子集T为全部球队的集合，令表示子集中球队已经赢得的比赛总数，令 表示子集中球队之间剩余的比赛总数（这些比赛在内进行，因此最终这些比赛必须由内的某个球队赢得）。我们定义 ，其中给出了球队子集的平均必须赢得的比赛数的下限。

例如我们之前讨论的四个球队淘汰问题，若 ， ，

中的球队最终胜利总场数有两种情况：都等于，或有的球队大于，有的球队小于。



根据上面的性质，我们可以得到以下定理。

定理：设，且如果存在，那么球队被淘汰。

证明：如果球队赢得了它所有的剩余比赛，那么它将以胜场结束赛季，但因为，则球队子集中至少有一支球队的胜场数会超过球队，故球队被淘汰。

对于上面的定理，如果存在，则我们说将球队淘汰。

**论文提出的新性质：**

如果球队被淘汰，那么可能总胜场数（胜利场数+剩余比赛场数）小于等于球队可能总胜场数，即的球队也被淘汰。

证明：因为球队被淘汰，根据上面的定理可以确定，存在一个球队子集，有

假设球队的可能总胜场数小于等于球队可能总胜场数，则有

如果，则将球队淘汰；如果，显然，令表示球队子集将球队剔除，则

所以，，将球队淘汰。

根据上面的性质，我们能进一步得到推论：

存在一个球队被淘汰，使得所有的球队也被淘汰，所有球队淘汰。

**参数化最大流算法：**

根据上面得到的推论，我们可以优化我们的最大流算法。

在流网络的图中，将球队节点到汇节点的边权值设为W为参数，表示球队的最大胜利场数。

如果该流网络的最大流不等于剩余比赛数，则说明最大胜利场数小于等于W的球队要被淘汰（结合前面的推论）

如此，只需要二分查找，找到最大的W（临界值），使得所有最大胜利场数小于等于W的球队被淘汰，所有最大胜利场数大于W的球队不被淘汰，则可根据每个球队的，球队有可能夺冠， 球队必然被淘汰。

以我们之前讨论的四个球队为例，通过二分查找，可以找到W的临界值为83，故最大胜利场数分别为83和80的Philly和Montreal被淘汰，最大胜利场数分别为91和84的Atlanta和New York有可能夺冠。

图示

描述已自动生成

**时间复杂度分析：**

由于只需要二分查找临界值，不需要对每个球队进行最大流算法，所以判断所有球队是否被淘汰的时间复杂度从变成，其中。

**代码解释：**

初始化：



根据每支球队的最大可能胜利场数从小到大进行排序，并且初始化l为每支球队的最大可能胜利场数的最小值，r为每支球队的最大可能胜利场数的最大值。

二分查找临界值W：

文本

描述已自动生成

Build\_Graph(mid)根据当前的mid(即W)构建残留网络graph，然后Ford\_Fulkerson(graph, 1, 2 + (n \* (n - 1) / 2) + n)调用Ford-Fulkerson算法从源点到汇点计算最大流量。

如果计算得到最大流量等于剩余比赛数量，说明当前W大于临界值，则向左压缩查找范围；若计算得到最大流量小于剩余比赛数量，说明当前W小于或等于临界值，记录当前W后，向右压缩查找范围。最终找到临界值W后退出循环。

**五、算法测试结果及效率分析**

对于四个球队的比赛情况，它们淘汰结果如下。

表格

描述已自动生成

文本

描述已自动生成

下面从普林斯顿大学课程网站获取5个测试数据，对上面提出的算法效率进行测试。

文本

描述已自动生成

各最大流算法的测试结果如下。

|  |  |  |  |
| --- | --- | --- | --- |
|  | Ford-Fulkerson | Edmond-Karp | Dinic |
| Team32 | 0.486400 | 1.488000 | 0.388500 |
| Team36 | 0.745900 | 2.691800 | 0.415100 |
| Team42 | 0.665300 | 2.525900 | 0.500000 |
| Team48 | 0.834100 | 4.057400 | 0.616600 |
| Team54 | 1.288900 | 8.233900 | 0.816900 |

图表, 条形图

描述已自动生成

可以看到，三种最大流算法的运行效率从高到低为Dinic、Ford-Fulkerson、Edmond-Karp，对于数据规模最大的数据team54，三种最大流算法的运行时间分别为1.288900s、8.233900s、0.816900s。

各最大流算法结合参数化的测试结果如下。

|  |  |  |  |
| --- | --- | --- | --- |
|  | Ford-Fulkerson | Edmond-Karp | Dinic |
| Team32 | 0.070300 | 0.212400 | 0.061100 |
| Team36 | 0.113100 | 0.424000 | 0.068000 |
| Team42 | 0.064800 | 0.260600 | 0.049400 |
| Team48 | 0.082900 | 0.379400 | 0.059600 |
| Team54 | 0.109400 | 0.690300 | 0.076700 |

图表, 条形图

描述已自动生成

可以看到，三种最大流算法结合参数化的运行效率从高到低为Dinic、Ford-Fulkerson、Edmond-Karp，对于数据规模最大的数据team54，三种最大流算法结合参数化的运行时间分别为0.109400s、0.690300s、0.076700s。很显然，二分+参数化最大流算法对三种最大流算法的提升是巨大的，并且该提升会随着数据规模的增大而提高。

**六、经验总结**

这次实验主要考察我们对最大流算法的应用，除了Dinic、Ford-Fulkerson、Edmond-Karp这三个算法外，我自己还去找论文看看有没有进一步优化的算法，最终才了解到二分+参数化最大流这个算法。论文是全英的，阅读起来对我来说有点困难，不过我也学会了借助ChatGPT来辅助我阅读论文。此外，为了测试算法效率，我还搜到了普林斯顿大学专门为该问题开设了实验，且提供了一些测试样例，于是我就拿这些测试样例对算法的效率进行测试。所以，这次实验给我的最大的收获是学会搜索、学会使用工具、学会阅读文献。

|  |
| --- |
| 指导教师批阅意见：    成绩评定：  指导教师签字：  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。