**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计**

**实验名称： 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（创新班）**

**报告人： 何泽锋 学号： 2022150221 班级： 高性能特色班**

**指导教师： 杨烜**

**实验时间： 2024年6月14日-2024年6月21日**

**实验报告提交时间： 2024年6月23日**

**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．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。

**三．实验原理**

**1.流网络的构造原理：**

流网络的构造原理是将实际问题抽象为一个图论问题，其中每条边代表实体之间的关系，并赋予一个非负的容量值。

**①节点和边**：在流网络中，实体被抽象为图的顶点，而实体之间的关系被抽象为图的边。

**②容量**：对于每条边，根据问题的具体情况分配一个非负的容量值。容量表示实体之间的某种限制

**③源点和汇点**：源点是流量的起点，而汇点是流量的终点。所有的流量都必须从源点流向汇点。

**④构建网络**：使用节点和边构建一个有向图，其中边的方向表示流量的流向，边的容量表示流量的限制。

**2.为什么最大流能解决这个问题？**

（1）分析题目：

棒球赛求解目标：依次判断哪个队伍可以获胜（并列最高也视为胜利），比赛从开始到全部结束是单向的，因此可以视作为网络流

抽象为图，依次判断每个队伍：

**①源点和汇点**：剩余比赛数量总数是要流出的量，因此可以将比赛的总场数视为源点，比赛进行的最终数量可以视为流入汇点的量

**②流量：**每个队伍都有需要进行的比赛，因此可以将两个队伍的比赛视作一个节点，源点流向该节点，流量即为这两队还需比赛的场数。从比赛节点再指向队伍节点，流量为比赛中 胜利的场数。队伍节点指向汇点，流量是队伍从剩下比赛中胜利的场数

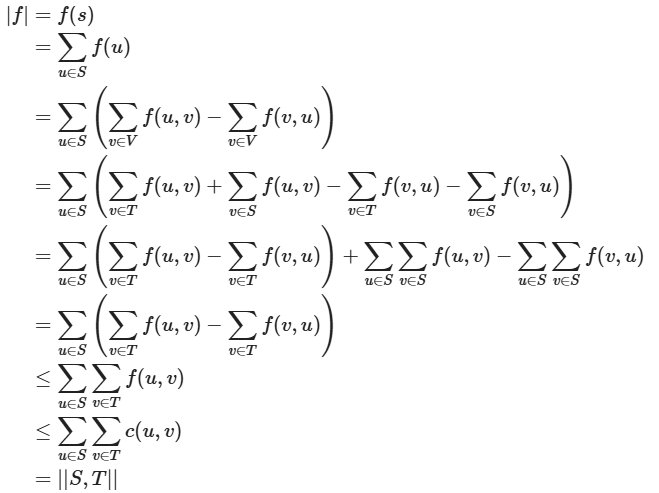
**③最大流的意义**：最终队伍流向会汇点的量即为满足条件情况下可以进行的比赛数量，若最大流等于所有还需进行的比赛数量，说明在进行完所有比赛后判断的队伍仍能获胜

（2）增广与最大流最小割的关系：**结论:当找不到增广路时，图中的流量即为最大流**

**1）证明最大流最小割定理**：

先从一个引理出发：对于网络*G = (V, E)* ，任取一个流*f*和一个割 *{S, T}* ，总是有 *|f| ≤||S, T||* ，其中等号成立当且仅当 *{(u, v)|u∈S, v∈T}* 的所有边均满流，且 *{(u, v)|u∈T, v∈S}* 的所有边均空流。

数学证明如下：



**2）证明增广停止时得到最大流*f***

假设某一轮增广后，我们得到流*f*使得上不存在增广路，即上不存在s到t的路径。此时我们记从s出发可以到达的结点组成的点集为S，并记*T = V \ S*。

显然，*{S, T}* 是的一个割，且 *||S, T|| = = 0*。由于剩余容量是非负的，这也意味着对于任意*uS*, *vT*, *(u, v)*，均有 *= 0*。以下我们将这些边分为存在于原图中的边和反向边两种情况讨论：

①*(u, v)*：此时，*=c(u, v) - f(u, v) = 0*，因此有*c(u, v) = f(u, v)*，即 *{(u, v)|u∈S, v∈T}* 的所有边均满流；

②*(v, u)*：此时，*=c(u, v) - f(u, v) = 0- f(u, v) = f(v, u) = 0*，即 *{(v, u)|u∈S, v∈T}* 的所有边均空流。

因此，增广停止后，上述流 *f* 满足取等条件。根据引理指出的大小关系，自然地，*f* 是 *G* 的一个最大流，*{S, T}* 是 *G* 的一个最小割。

**3.淘汰判断**

淘汰有两种判断方式：**平凡淘汰**和**非平凡淘汰**

**（1）平凡淘汰**

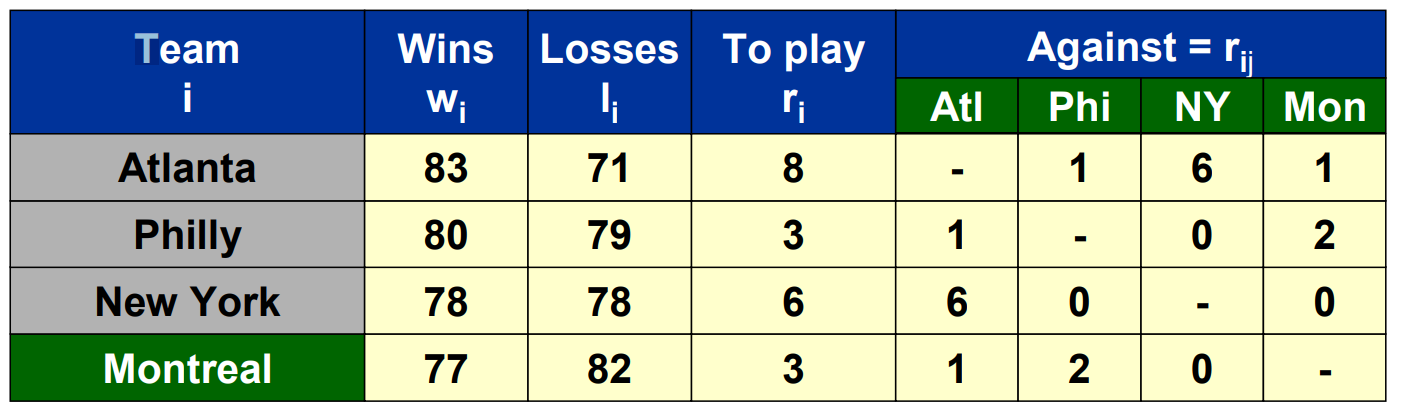
平凡淘汰采用直接判断的方式。如果一支队伍的已获胜次数加上剩余比赛的次数（最大获胜次数）小于其他队伍i的已获胜次数，则该队伍x必定会被淘汰。  
用数学表达，即是对于x队伍和其余队伍i，若，则x必会被淘汰。

**（2）非平凡淘汰**

非平凡淘汰（Untrivial elimination）采用最大流进行判断，若最终最大流不等于还需要进行的比赛数量则被淘汰。

**4.样例解释：**

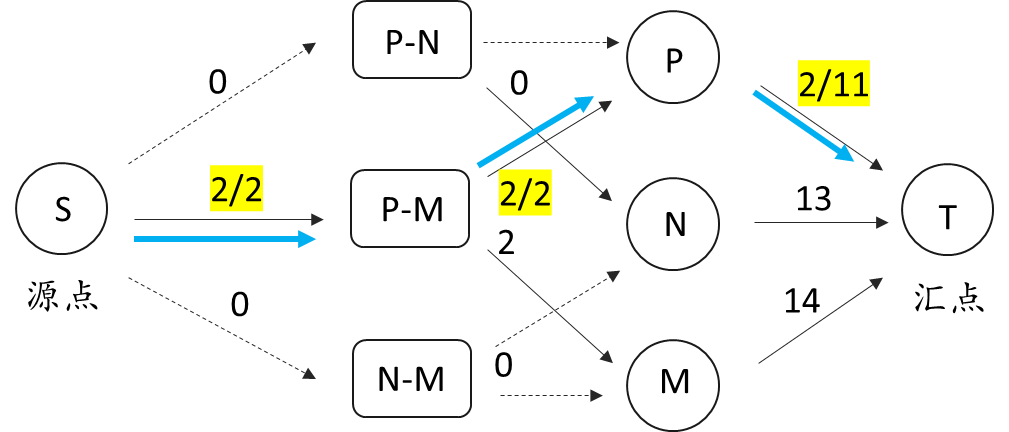
已知如下的四支队伍，表格中分别记录他们已经获胜场数、失败场数、剩余比赛场数以及具体与哪支队伍的比赛信息



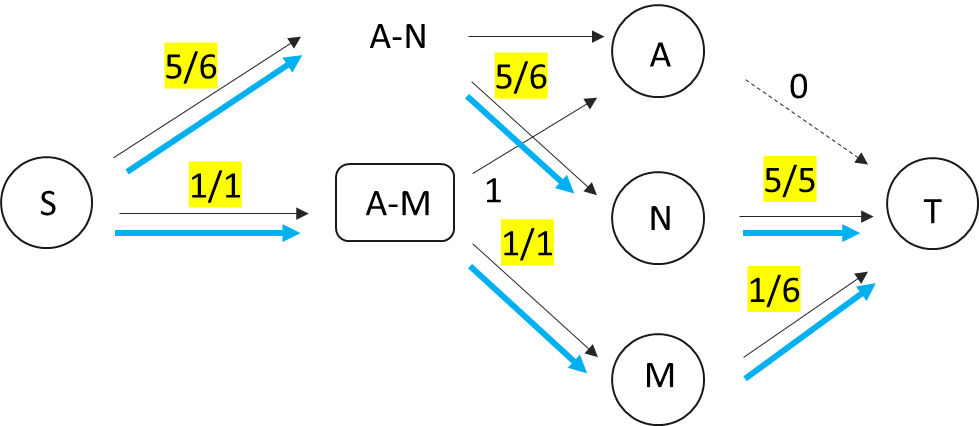
按照网络流的逻辑将表格抽象为图，其中源点为还未开始比赛，汇点为比赛结束，第一层的节点是两两比赛的队伍，第二层的节点是具体队伍，边的容量是队伍需要进行的比赛场数、

流量为实际比赛场数或获胜场数（流向队伍节点）

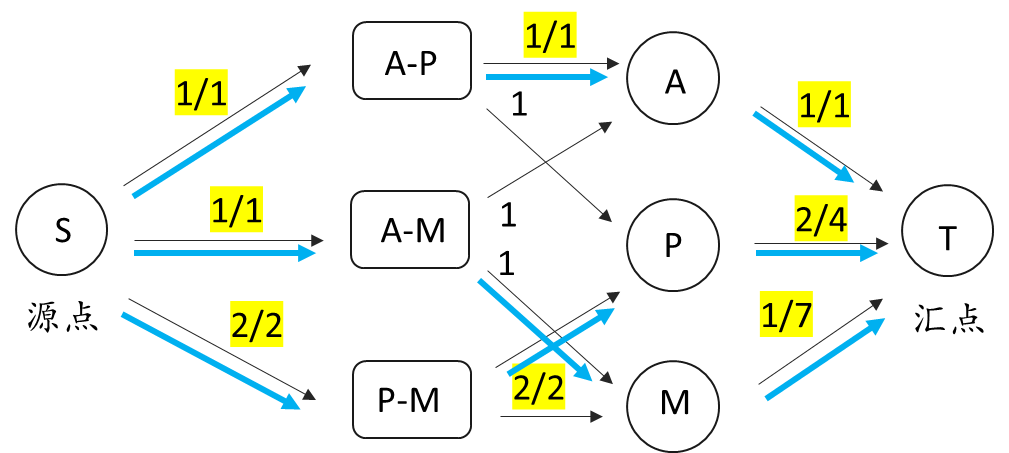
①Atlanta全胜情况下最多91胜，因此其他队伍最终胜场都不能超过91，构建如下网络，实际存图时如图中的虚线即容量为0的边是不存在的，没有比赛的队伍节点、比赛节点也都不存在图中。边上的信息为（实际流量/容量），蓝色箭头为流的方向。在没有超过流量限制（w[x]+re[x]-w[i]）的情况下，最终的源点流出量等于流入汇点的量，因此说明Atlanta队可以获胜



②Philly全胜情况下最多83胜，因此其他队伍最终胜场都不能超过83，构建如下网络并得到最大流，可以看到汇点的总流量为6，即最大流为6，但需要进行的比赛总数是7，因此无法在全部比赛打完的情况下获胜，Philly被非平凡淘汰



③New York全胜情况下最多84胜，因此其他队伍最终胜场都不能超过84，构建如下网络，可以看到源点流出量等于流入汇点的量，结果说明在限制条件下所有队伍都能完成比赛，因此New York队可以获胜

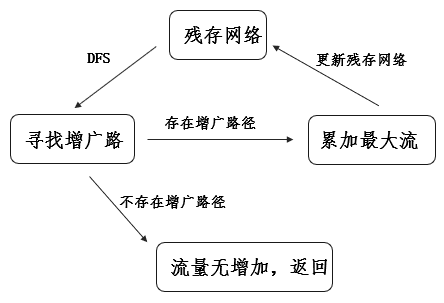


④Montreal全胜情况下最多80胜，但此时Atlanta队伍已经获得了83胜，因此Montreal队是不可能获胜的，被平凡淘汰

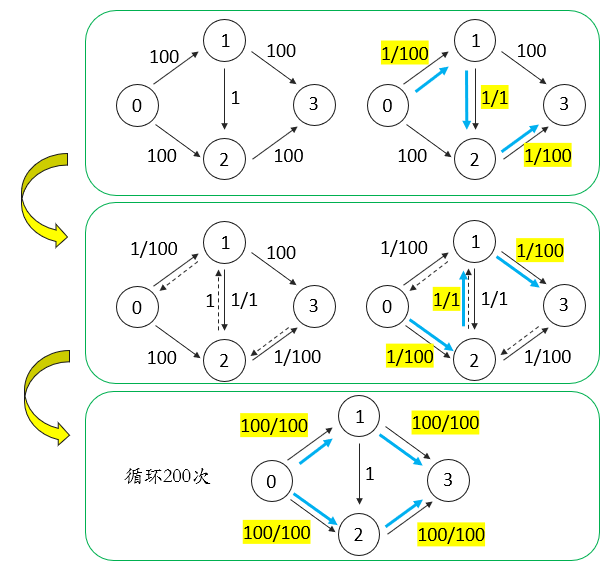
**四．算法分析**

**1.Ford- Fulkerson算法**

（1）FF算法的基本思路：在残存网络中不断寻找增广路径，每找到一条增广路径，就递增最大流f，并更新残存网络，直到残存网络中不存在增广路径，则此时f即为最终的最大流。



（2）样例解释：如图所示，从源点0出发通过DFS的方式寻找增广路径，当使用了一条边的容量后要在图中添加等大方向的边，在此案例中中间容量为1的边被频繁使用和退流，最终通过200次循环才得到最大流



（3）复杂度分析：

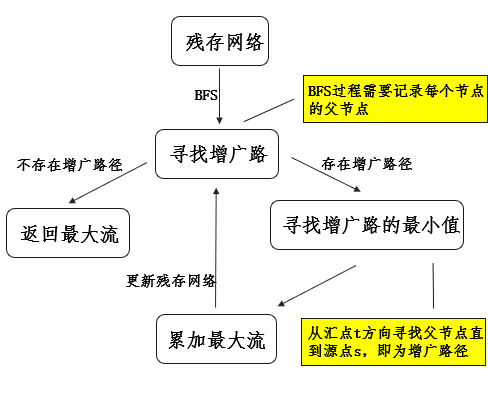
①图中有E条边，最坏情况下找到一条增广路径需要搜索m次，即

②假设最大流为f，最坏情况下每次都只增加1的流量，因此需要寻找f次，结合搜索的时间复杂度可知，最坏的时间复杂度为

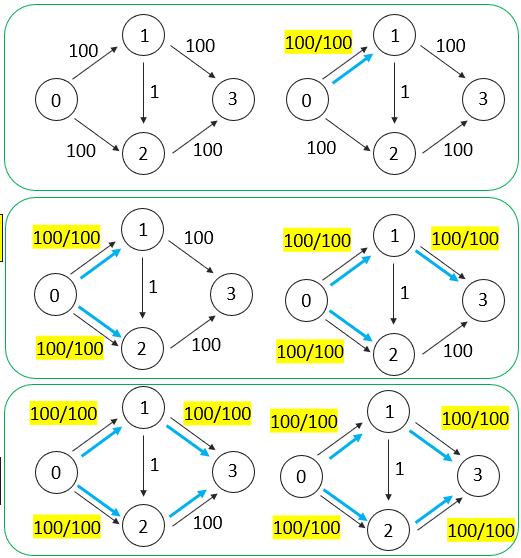
因此FF算法的时间复杂度依赖于最大流的大小，实际情况下可能会很慢，但在本次实验中，因为每个队伍剩余场数和队伍数都比较小，因此最大流不会很大，实际时间复杂度并不会很高，这也是为什么最终数据测得FF算法比EK算法要快的原因之一

**2.Edmons-Karp算法**

（1）EK算法的基本思路：基本思路与FF算法相同，即不断寻找增广路径，而与DFS的过程不同，BFS需要先找到整条路径，再进行残存网络的更新，而DFS可以在递归的过程中把残存网络的更新完成。引入父亲节点数组，在BFS过程中，记录下每个节点的父亲节点。这样子当BFS结束后，若存在增广路径，则对汇点t进行反向寻父流程，直到找到源点s，则中途经过的节点都是增广路径上的节点。



（2）样例解释：如图所示从源点出发，使用BFS遍历图，首次遍历一个点时记录其父节点为上一节点，当再次遍历到该点时则直接跳过（如图中容量为1的边，上下两个节点父节点都已为源点，因此不会经过此边）。



（3）复杂度分析：

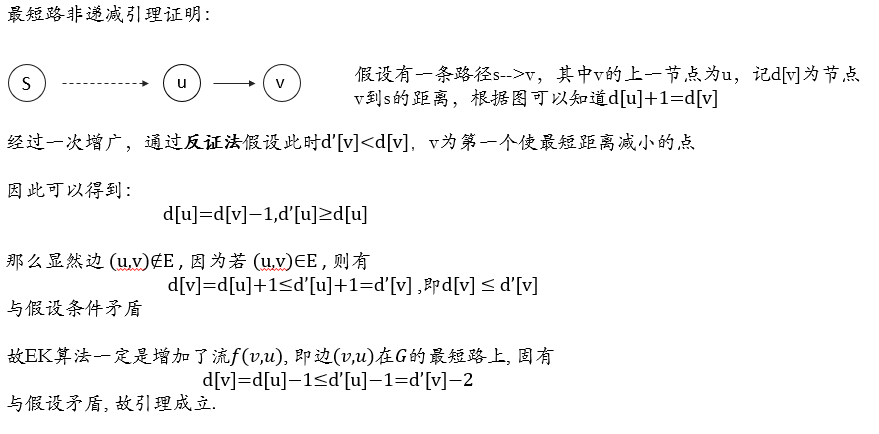
①图中有E条边，那么逆流图中最多含有2E条边，因此通过BFS找到每次最短的增广路径时间复杂度为

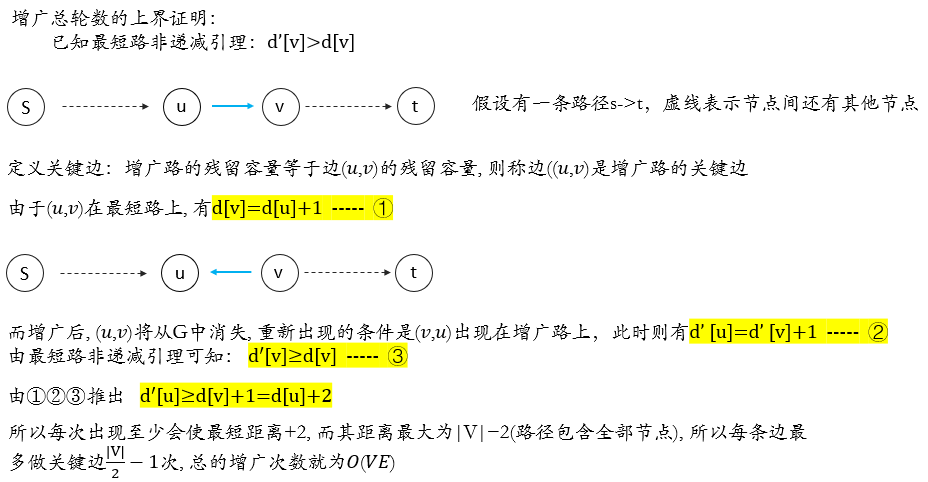
②图中有V个节点，增广总轮数的上界是

③因此时间复杂度为

**增广总轮数证明：**

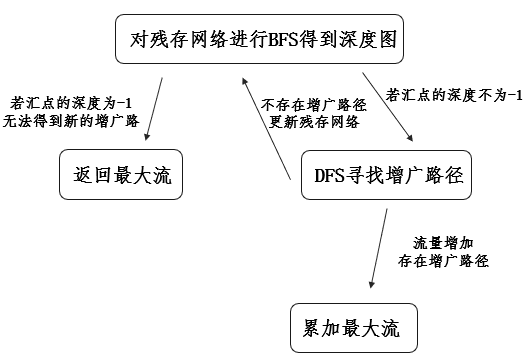
首先引入**最短路非递减引理**，通过该引理可以确保每次增广操作都不会使得源点s到任一节点u的最短路径长度变短，即





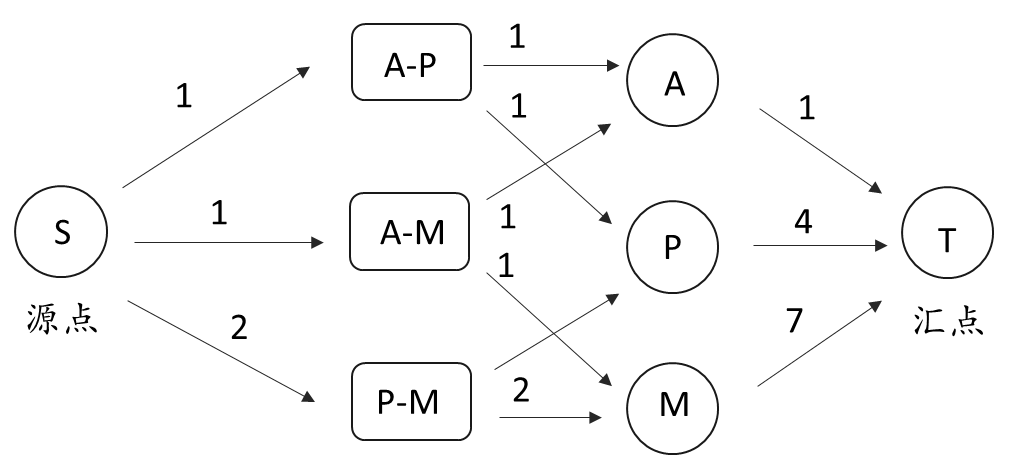
**3.Dinic算法**

（1）Dinic算法基本思路：构建初始的残存网络。接着进行BFS构建深度图（实际是在一张图），用一个深度数组记录每个节点的遍历深度。然后再对源点s不断进行DFS寻找增广路径，同时修改残留网络。改用深度数组进行递归，每次的深度都要+1才进行遍历，对于同深度或者低于当前节点的深度的节点，则不进行遍历。这么做的原因是，汇点t的深度一定是所有节点的深度中最大的，而增广路径的每个节点的深度是不断递增的，所以比起常规DFS的随意递归，该方法的递归更有针对性，即递归的每一步都是在靠着汇点t前进的。如果当前深度图进行DFS找不到增广路径，则重新进行一次BFS，创建新的深度图，继续寻找增广路径。直到汇点的深度为-1（初始值），说明无法再有路径到达汇点则返回最大流

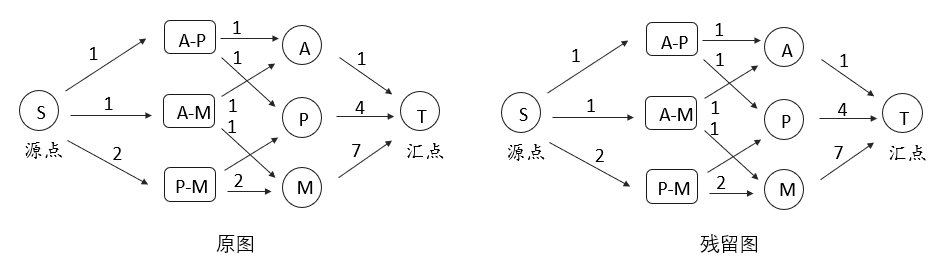


（2）样例解释：

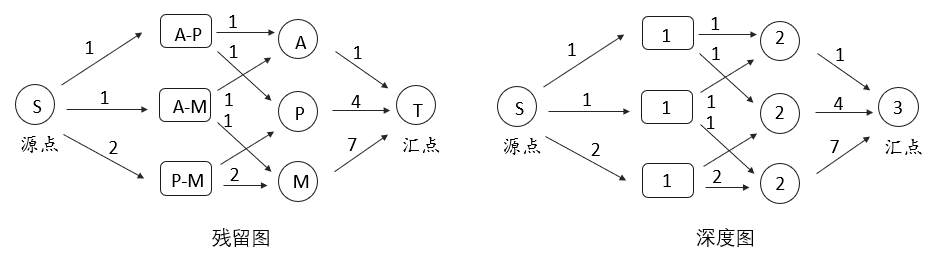
假设有如下一个网络图，各节点和边的信息都如图所示



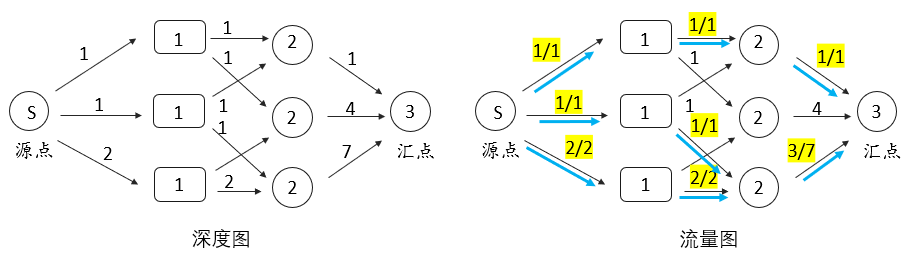
①根据原图获得残留图（初始状态一样）



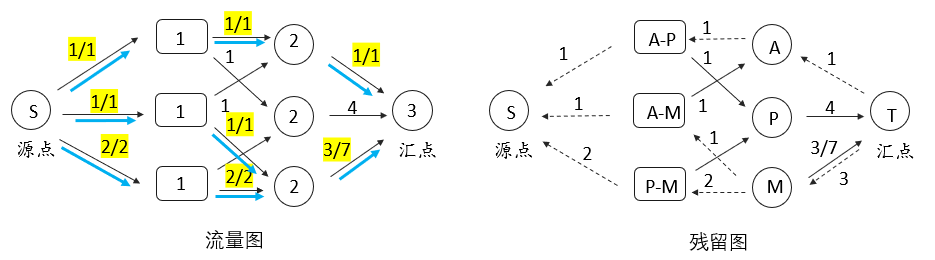
②根据残留图BFS获得深度图，从源点开始每次增加一层，若能直达则保留边，实际上深度图是残留图的一部分，只需通过一个深度数组记录深度信息即可，对于需要用到的边则判断两节点的深度关系便可确定



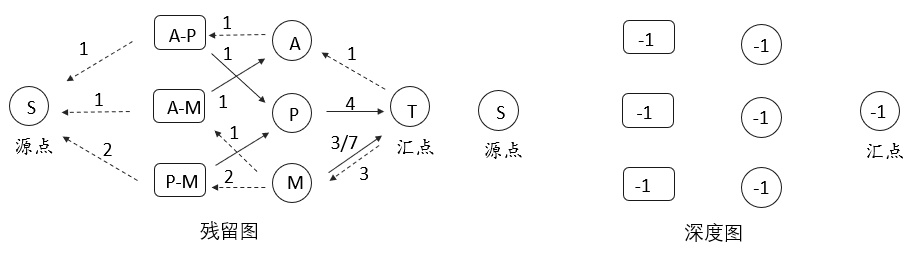
③在深度图中通过DFS寻找增广路径，可能找到多条，直到流量不再增大（无增广路径）



④在DFS的过程中要实时更新残留图的流量信息，添加对应的反向边



⑤根据残留图更新深度图，发现没有容量使得从源点流向各节点，因此都默认深度为-1，汇点深度为-1时直接返回



（3）复杂度证明：

时间复杂度的组成主要有两部分，分别是**单轮增广的时间复杂度**和**增广轮数**

**①单轮增广的时间复杂度**：Dinic算法采用了当前弧优化，即记录一个点上一次DFS所到之处，避免了重复访问之前走过的边。考虑阻塞流f中的每条增广路，它们都是在 G上每次沿当前弧跳转而得到的结果，其中每条增广路经历的跳转次数不可能多于|V|。

每找到一条增广路就有一条饱和边消失（剩余容量清零）。考虑阻塞流f中的每条增广路，我们将被它们清零的饱和边形成的边集记作E1。考虑到G分层的性质，饱和边消失后其反向边不可能在同一轮增广内被其他增广路经过，因此， E1是E的子集

此外，对于沿当前弧跳转但由于某个位置阻塞所以没有成功得到增广路的情形，我们将这些不完整的路径上的最后一条边形成的边集记作E2。 E2的成员不饱和，所以 E1与E2不交，且E1∪E2仍是E的子集

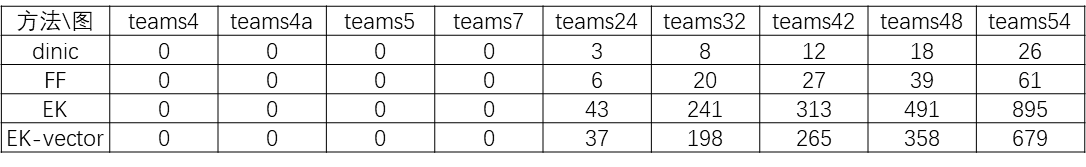
由于每条边都不会花费超|V|次跳转，因此单轮增广的时间复杂度为

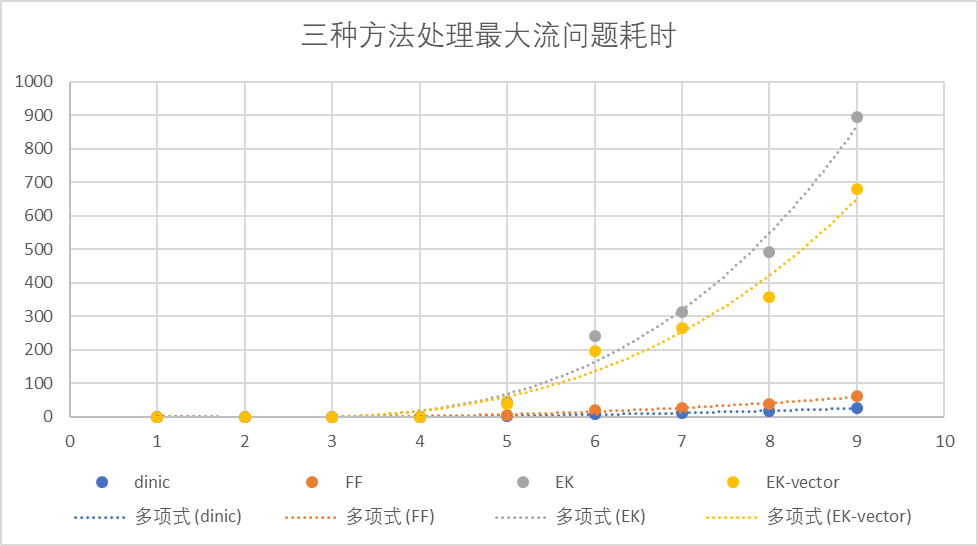
**②增广轮数**：采用BFS生成了深度图，其深度显然不可能超过|V|，我们可以证明知道在每次增广过程中深度图的深度都是严格递增的，因此可以知道增广的轮数是

因此总的时间复杂度为

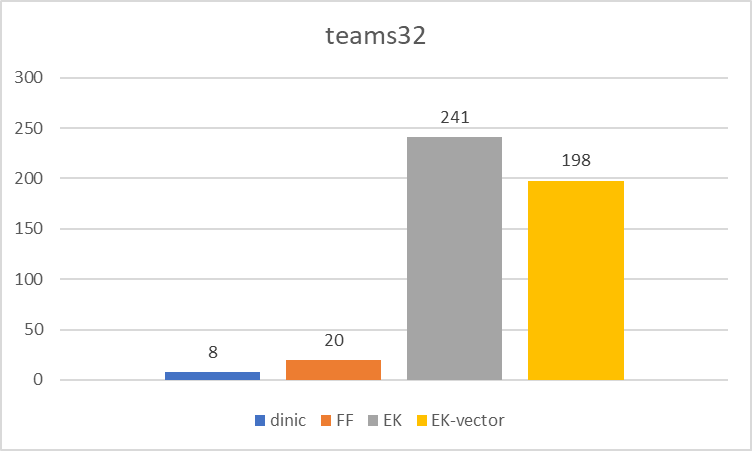
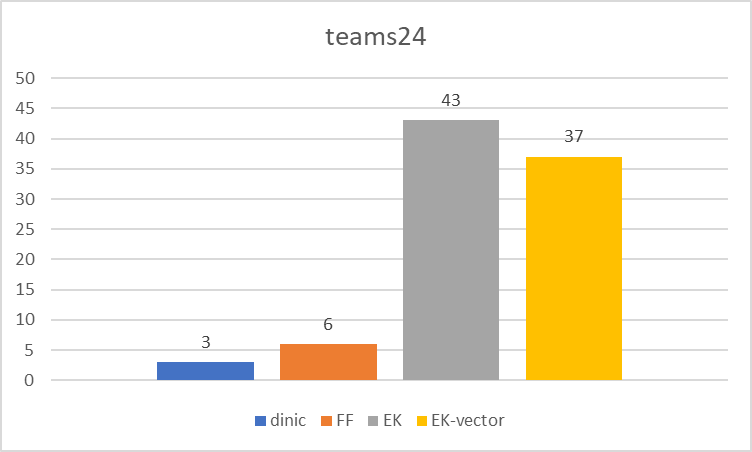
**五．数据分析**

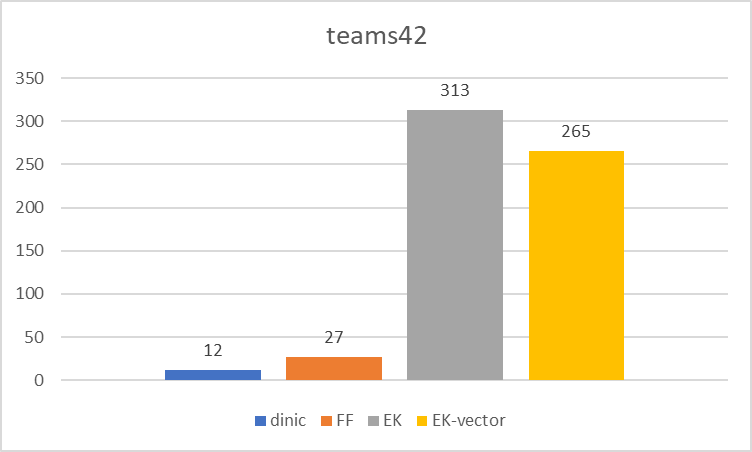
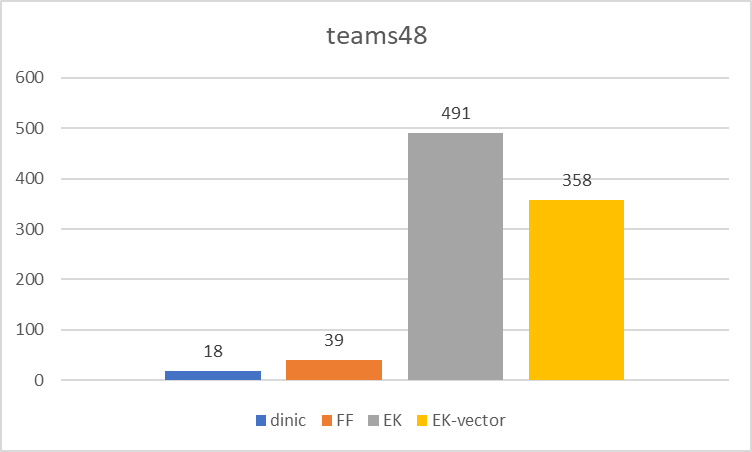
运行论文中的部分样例，输出非平凡淘汰的队伍名称，记录三种方法的运行时间

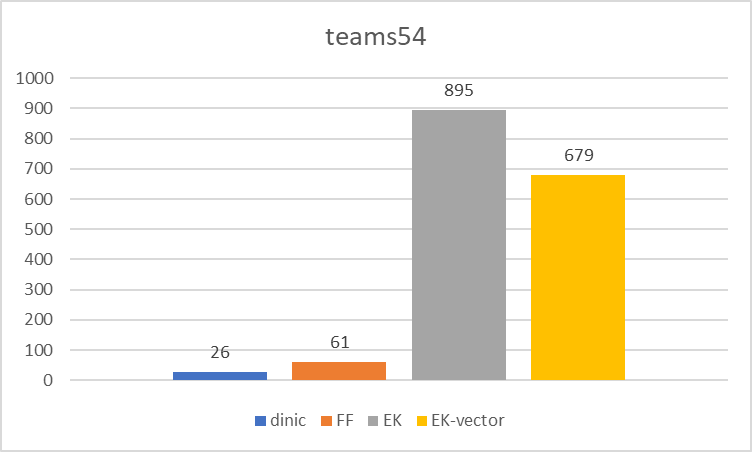




使用柱形图显示每组的比较信息，此处比较非0ms时间下的组，可以看到在多组数据下其效率顺序都是一样的即dinic> FF > EK-vector > EK。此处的EK-vector是在EK的方法上，通过将数组存图改为用vector存图，无需遍历每一个点之间是否有边，仅需判断有边的点。





**六．实验结论**

通过本次实验，了解了网路流的基本实现逻辑，学习了FF、EK以及dinic三种处理网络流问题的算法，经过实验可以知道，在处理棒球赛问题上三种算法的效率排序是dinic > FF > EK。对于算法的时间复杂度证明花了较多的时间进行理解，以及对最大流算法的合理性证明，需要从最大流最小割定理开始，再引申到增广路的可行性上。

|  |
| --- |
| 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。