姓名：马琦

学号：2011600

专业：计算机科学与技术

南開大學
Nankai University

计 算 机 学 院

算法导论课程大作业

对最大流问题的解决与优化探索

2023 年 4 月 16 日

摘要

本文主要探究的是网络流中经典的最大流问题。课程讲解中主要介绍了 Ford-Fulkerson 算法，但是该算法的运行效率并不是特别高。本文主要应用基于增广路求解的 Dinic 算法、Edmonds-Karp 算法，ISAP 算法以及基于预流推进的最高标号预流推进算法，并给出算法的正确性及终止性证明，分析算法的时空复杂度。代码在压缩包中 EK.cpp，Dinic.cpp,ISAP.cpp，HLPP.cpp。作者将链式前向星、gap 优化、bfs 过程中高度优化等技巧与代码结合，减小时间复杂度常数，在洛谷上的 P3376(中小规模数据)、P4722(大规模数据) 取得了较好的运行效果。

关键字：最大流问题　增广路　预流推进　Dinic 算法　EK 算法　ISAP 算法　HLPP 算法

目录

一、 最大流问题的介绍

(一) 问题的形式化定义及约束

对一个有向图 $G = (V,\ E)$, 每条边 $(u,\ v)\ \in\ E$，都有一个权值 $c(u,\ v)$, 称之为**容量**，当 $(u,\ v) \notin E$ 时有 $c(u,\ v)\ =\ 0$。其中有两个特殊的点：源点 $S \in V$ 和汇点 $T \in V\ \ (S \neq T)$

设 $f(u,\ v)$ 定义在二元组 $(u\ \in\ V,\ v\ \in\ V)$ 上的实数函数且满足

- **容量限制**：对于每条边，流经该边的流量不得超过该边的容量，即 $0 \leq f(u,\ v) \leq c(u,\ v)$
- **流守恒性**：从源点流出的流量等于汇点流入的流量，即

  对于 $\forall x \in\ V - \{S, T\}$,

$$\sum_{(u,x)\in E} f(u,x) = \sum_{(x,v)\in E} f(x,v)$$

那么 f 称为网络 G 的流函数，也就是该网络的**可行流**。对于 $(u,\ v)\ \in\ E$，$f(u,\ v)$ 称为边的流量，$c(u,\ v)\ -\ f(u,\ v)$ 称为边的**剩余容量**。整个网络的流量为 $\sum_{(s,\ v)\in E} f(s,\ v)$，即从源点发出的所有流量之和。流函数的完整定义式子如下：

$$f(u,v) = \begin{cases} f(u,v), & (u,v) \in E \\ -f(v,u), & (v,u) \in E \\ 0, & (u,v) \notin E, (v,u) \notin E \end{cases} \tag{1}$$

借由前面的定义，所求的问题就是**求从源点 S 流向汇点 T 的最大流量** (可以有很多条路到达汇点)，这也就是我们的最大流问题。

(二) 问题描述

给定一个包含 n 个点 m 条边的有向图，并给定每条边的容量，边的容量非负。图中可能存在重边和自环。求从点 S 到点 T 的最大流。

(三) 输入格式

第一行包含四个整数 n, m, S, T。

接下来 m 行，每行三个整数 u, v, c，表示从点 u 到点 v 存在一条有向边，容量为 c。

点的编号从 1 到 n。

(四) 输出格式

输出点 S 到点 T 的最大流。

如果从点 S 无法到达点 T 则输出 0。

二、 问题的核心解决思想及其证明

(一) 思想零：后悔策略的实现

我们是无法在单独的某一次运算中知道当满足最大流条件时某一边的流量的，尤其当网络复杂时，我们发现之前做过的策略并不是最优的，这时候我们就需要提供一种反悔策略来保证算法

的正确性，这种策略就是反向边。下面给出流网络 $G = V, E$ 中反向边集 E' 的形式定义。

$$E' = \{(u, v) \mid (v, u) \in E\}$$

反向边的流量初始化为 0。

当我们发现之前的流量安排不是最优时，可以通过反向边将流量推会原先的点。

(二) 思想一：基于增广路进行求解

第一种解决该问题的主要方法是利用残留网络、增广路、反向边。我们定义残留网络 G_f 是网络 G 中所有结点和所有的正向边与反向边。形式化的定义，即

$$G_f = (V_f = V, E_f = E \cup E')$$

残留网络边的容量为

$$c'(u, v) = \begin{cases} c(u, v) - f(u, v), & (u, v) \in E \\ f(v, u), & (v, u) \in E \end{cases} \tag{2}$$

在原图中若一条从源点到汇点的路径上所有边的剩余容量都大于 0，这条路被称为增广路。或者说，在残量网络中，一条从源点到汇点的路径被称为增广路。下面我们给出利用增广路解决最大流问题的证明。

引理 1： 假设 f 为原网络的可行流，f' 为残留网络的可行流，那么 $f + f'$ 依然是**原流网络**的可行流。且 $|f + f'| = |f| + |f'|$。($f = \{(u, v, f(u, v)) \mid (u,\ v) \in E\},\ f' = \{(u, v, f'(u, v)) \mid (u,\ v) \in E\}$)

说明：可行流加法只对边的流量生效，满足同向相加反向相减的原则，形式定义如下:

$$f + f' = \{(u, v, f(u, v)\ + f'(u, v)) \mid (u, v) \in E\}$$

|f| 指从 S 发出最终流入 T 的流量。

证明：只需证明 $f + f'$ 中流量满足可行流的两个性质即可。

性质一：即证明 $f(u, v)\ + f'(u, v)$ 满足**原图**的流量限制。易得

$$0 \le f(u, v) \le c(u, v)$$

$$0 \le f'(u, v) \le c'(u, v)$$

由 $c'(u, v)$ 定义式和 f(u,v) 的定义，当残留网络中的可行流与原网络中可行流方向相同时，

$$0 \le f'(u, v) + f(u, v) \le c(u, v)$$

当残留网络中的可行流与原网络中可行流方向不同时，

$$0 \le f'(u, v) + f(u, v) = f(u, v) - f'(v, u) \le f(u, v) - f(u, v) \le f(u, v) \le c(u, v)$$

可见性质一是符合的

性质二：

$$\sum_{(u,x) \in E} f(u, x) = \sum_{(x,v) \in E} f(x, v)$$

$$\sum_{(u,x) \in E_f} f(u, x) = \sum_{(x,v) \in E_f} f(x, v)$$

两式相加等号依然成立，所以满足性质二。
综上，$f+f'$ 是原网络的可行流。对于 $|f+f'|=|f|+|f'|$ 是很好证明的。我们可以将流网络看作 $S \to other\ points \to T$。其流量表示如下

$$\sum_{(S,x)\in E} f(S,x) + \sum_{(S,v)\in E_f} f(S,v)$$

没有方向的原因在于源点 S 只有流出没有流入，且整个流网络流量守恒，从 S 流出的一定会流入 T。
引理 2：假设 f 为原网络的可行流，f' 为残留网络的可行流，当 f' 不存在时，则该情况下流入 T 的流量即为所求最大流。
证明本引理需要用到**最大流最小割定理**
最大流最小割定理: $F(s,t)_{max}=C(s,t)_{min}$。其中 $F(s,t)_{max}$ 为两个割之间的最大流量,$C(s,t)_{min}$ 为割的容量。源点 $S\in s$，汇点 $T\in t$。
其中，流网络点集 V 的割 s 和 t 应该满足如下性质：

- $s\bigcup t=V$，$s\bigcap t=\phi$
- $\sum_{u\in s,v\in t} f(u,v)-\sum_{u\in t,v\in s} f(u,v)\le \sum_{u\in s,v\in t} c(u,v)$
- 可行流 f 应该满足 $|f|\le C(s,t)$，$|f|=F(s,t)$

在该定理的基础上，我们只需要证明

$$f'\subset \phi \Longleftrightarrow f \text{ 是最大可行流} \Longleftrightarrow \exists s,t,|f|=C(s,t)$$

证明：我们先证明 f 是最大可行流 $\Longrightarrow f'\subset\phi$。利用反证法，假设存在 f'，那么由我们之前证明的结论 $|f|<|f+f'|=|f|+|f'|$ 可知，当前 f 不是最大可行流，与条件矛盾，所以原假设成立。之后我们再证明 $\exists s,t,|f|=C(s,t)\Longrightarrow f$ 是最大可行流。我们假设所求最大可行流为 F，首先有 $|F|\le C(s,t)|min$，又有 $|f|\le|F|$，由条件有 $|F|\le C(s,t)|min\le|f|$。所以 $f=F$。即 f 就是最大流。最后再证明 $f'\subset\phi\Longrightarrow\exists s,t,|f|=C(s,t)$, 我们不妨对集合做如下划分，令

$$s'=\{u|\text{源点 S 以及 S 可以通过容量大于 0 的边到达的点}\}$$

$$t'=V-s$$

因为残留网络不存在增广路经，所以 S 和 T 一定在不同的集合中。我们取点 a,b,x,y 且满足 $a,x\in s',b,y\in t'$, 且 $(a,b)\in E$, $(y,x)\in E$, 那么我们有如下假设: $f(x,y)=0$ 且 $f(a,b)=c(a,b)$。
证明：均采用反证法，如果 $f(a,b)\ne c(a,b)$ 那么在残留网络里该处的剩余容量 $c'(u,v)\ne 0$，这与我们一开始对 s' 的假设矛盾，说明 S 是可达 v 的，所以 $f(a,b)=c(a,b)$。同理可以证明，$f(x,y)\ne 0$，剩余容量 >0，从而与点集划分矛盾，所以综上，两个假设成立。
通过两个假设，我们有

$$|f|=\sum_{u\in s,v\in t} f(u,v)-\sum_{u\in t,v\in s} f(u,v)=\sum_{u\in s,v\in t} f(u,v)=\sum_{u\in s,v\in t} c(u,v)=C(s'.t')$$

通过以上证明，这三个命题构成了一个论证的循环，也就是知道任一命题都能得出另外两个命题，从而证明了等价关系。
终止性：证明该算法的终止性，只要证明在算法迭代中流网络的残流网络的可行流会在某一次迭代中不存在。我们通过直觉想也是不对的。利用引理 1，我们知道 $|f+f'|=|f|+|f'|$，如果 $|f'|$ 一直存在大于 0，那么最终网络的可行流流量将会无穷尽地增大，这显然不符合常理。所以尽管有些迭代中会出现流量沿反向边退回去的情况，但是对于整体网络来说流量是变大的，所以算法

是一定会终止的。
综上，我们就可以将求最大流的问题转化为求使得原网络残留网络不存在增广路经的流量分布，这就是我们的思想核心所在。具体的算法不尽相同，但是主要的框架主要包括：

- 通过 bfs 寻找增广路径。
- 将增广路径加入到当前的可行流。
- 当不存在增广路径时，流量即为最大可行流。

(三) 思想二：基于预流推进进行求解

本思想所基于的想法看起来可能比之前的增广路径简洁一些，描述为：预流推进算法将每个点看做一个可以存储流的“水池”，其中存有流的点称为活动节点; 对于每个非 s 或 t 的点，流入该点的流只可能有两个去向：流入汇点 t，流回源点 s; 预流推进算法从源点开始沿边向其它点推流，之后每次选一个活动节点通过推流，试图使其变得不活动。当所有节点都是不活动节点时，算法就结束了。
下面给出该思想需要用到的一些定义：

- f：预流。
- x_f：每个点在预流为 f 时的滞留流量，也叫超额流，定义为

$$x_f(u) = \sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v)$$

- c_f：每个边在预流为 f 时的剩余容量，定义为

$$c_f(e) = c(e) - f(e)$$

- 标记函数 $l(u), (u \in V)$, 用来标记距离 (也可以称作高度)，该函数应该满足
  - **有效标记** $l(u) \leq l(v) + 1, \forall u, v, (u,v) \in E_f$
  - **源点条件：** $l(S) = |V|$
  - **汇点条件：** $l(T) = 0$
- $E_f \subset E$, 包含所有 $f < c$ 的边
- $G_f(V, E_f)$ 是当预流为 f 时的残流网络。
- 其余定义和之前相同

预流推进的主要操作为：

- 初始化网络：该算法首先创建一个残流网络，将预流值初始化为零，并对离开源的残流弧执行一组饱和推送操作,(s,v)，其中 $v \in V - \{S\}$ 类似地，标签被初始化，使得源点处的标签是图中节点的数量，$l(s) = |\mathrm{V}|$ ，并且所有其他节点都被赋予 0。初始化完成后，算法重复对活动节点执行推送或重新标记操作，直到无法执行适用的操作。
- 推流，将活动节点 u 的流量沿着边 (u,v) 推给另一个点，推的流的大小为

$$\delta = min(x_f(u), c(u,v) - f(u,v))$$

  如果 u 还是活动结点，重标记，继续重复此步骤。

- 重标记，将对应的 $l(u)$ 标记为

$$l(u) = 1 + min(l(v)), \forall v, c_f(u, v) > 0$$

引理 3：当所有节点都是不活动节点时，该网络的残流网络不存在增广路径.
证明：如果每个结点都是不活动结点时，那么该点的余流一定是已经都被分走的。我们不妨从入边和出边的角度分析。只考虑某个点，如果有退回流量给之前的结点，那说明从该点出发的所有边剩余容量均为 0，那么是不可能存在过该点的增广路径的。如果全部分发给其他的节点，那么说明之前的节点所能给的流量是有限的，之前的边的容量是有限的，这样也是不存在过该点的增广路径。综上，结论成立。
终止性：特殊的情况比如两个点之间反复传流，我们通过重标记的方法解决了这个问题，使得算法最终可以形成所有点都为不活动节点的情景，
由引理 3 以及思路一的结论我们也可以得出，该算法解出的流量即为最大可行流。

三、 几种算法的时空复杂度分析

(一) 增广路算法

1. EK 算法

EK 算法的步骤:

- 利用 BFS 求解残流网络的增广路
- 将增广路的流量加到当前可行流的流量中，并更新逆向残流网络的边权
- 当不存在增广路时直接退出

算法复杂度:
因为我们每次都采用 bfs 的方式寻找增广路，所以每一个点在不同迭代被找到时其深度也是会越来越大的。我们假设在相邻的两次迭代中点 u 被连续搜索，那么它的搜索深度一定会增大的，并且每个点的深度又是不大于 $|V|$ 的，进而我们可以得到总的增广次数为 $O(VE)$, 进而整个算法的复杂度就是 $O(VE^2)$

2. Dinic 算法

Dinic 算法的步骤:

- 利用 BFS 求解残流网络的增广路
- 如果存在增广路径，则由 DFS 直接一次性求出多条增广路径，进而将这些流量加入当前的可行流中。
- 当不存在增广路时直接退出

算法复杂度:
因为在 Dinic 的执行过程中，每次重新分层，汇点所在的层次是严格递增的，而 V 个点的层次图最多有 V 层，所以最多重新分层 V 次。在同一个层次图中，因为每条增广路都有一个瓶颈，而两次增广的瓶颈不可能相同，所以增广路最多 E 条。搜索每一条增广路时，前进和回溯都最多 V 次，所以这两者造成的时间复杂度是 $O(VE)$；而沿着同一条边 (i, j) 不可能枚举两次，因为第一次枚举时要么这条边的容量已经用尽，要么点 j 到汇不存在通路从而可将其从这一层次图中删除。综上所述，Dinic 算法时间复杂度的理论上界是 $O(V^2E)$。

3. 更优的算法——ISAP

ISAP 算法的步骤:

- 只进行一次逆向 BFS，并且标记所有点的层次，同时记录不同深度点的个数
- 进行深度优先搜索，并且不断更新点的深度进行 +1 操作
- 如果源点 S 的深度超过 n，说明不存在增广路，退出循环

算法复杂度:
该算法和 Dinic 算法的复杂度应该是相同的，只是对 Dinic 的 bfs 次数做了优化，故其时间复杂度为 $O(V^2E)$。(网上许多人的博客是错误的，误认为和 EK 是相同的，这显然是不对的)

4. 一个可能的优化——结合动态树 LCT

网上有论文提出可以结合 LCT 对 Dinic 进行优化，但是当下这种做法非常罕见，对于该优化需要极其特殊的流网络场景，而大部分小规模情况下该算法比其他任何算法都弱。另外作者鉴于水平有限，也没能成功复现该算法。

(二) 预推流最强算法——HLPP 算法

HLPP 算法的步骤:

- 初始化 (基于预流推进算法);
- 选择溢出结点中高度最高的结点 u，并对它所有可以推送的边进行推送;
- 如果 u 仍溢出，对它重贴标签，回到步骤 2;
- 如果没有溢出的结点，算法结束。

算法复杂度:
提出者通过一系列推演已经证明是 $O(V^2\sqrt{E})$。我们可以大致意会一下, 对于一个流网络, 我们推流时的复杂度为 $O(V\sqrt{E})$, 而每个点都要进行一次推流的操作, 所以最终的复杂度为 $O(V^2\sqrt{E})$。

(三) 基于 C++ 语言的优点

本问题中未涉及到特殊包或者链表的一些特殊的调用，采用了最基本的迭代与循环等基础操作，这种情况下 Python 和 Java 的运行效率就会相对较低，编写难度反而会提高，所以采用 C++ 编写该问题相对来说是比较合适的。

四、 有效性

在小规模数据范围下 Dinic 和 ISAP 较优，但在大规模数据集下只有 HLPP 能够在限定时间内跑出正确答案。

图 1: 在小规模数据上各个算法的效率

图 2: 在大规模数据下 HLPP 算法

五、 总结

以上算法为网络流中最大流问题目前可见的所有主流解法。除了 EK 之外，其余算法的效率都是非常优秀的，在解决我们所熟知的一些问题，比如二分图匹配、多源汇最大流问题等，均具有相当高的效率，例如在解决二分图匹配时，Dinic 算法的时间复杂度可以降低到 $O(E\sqrt{V})$。很多复杂的图论问题都可以通过合理的建图转化为网络流问题并且高效地解决