云南大学数学与统计学院
《运筹学通论实验》上机实践报告

课程名称： 运筹学实验	**年级：** 2015 级	**上机实践成绩：**
指导教师： 李建平	**姓名：** 刘鹏	**专业：** 信息与计算科学
上机实践名称： 求网络最大流问题	**学号：** 20151910042	**上机实践日期：** 2018-07-07
上机实践编号： 6	**组号：**	

一、 实验目的

加深对网络最大流算法的理解。

二、 实验内容

用 MATALB 风格的伪代码写出求网络最大流[1]的算法；
用 C 语言[2]编程实现 Edmond karp 算法。

三、 实验平台

Microsoft Windows 10 Pro Workstation 1803；
Cygwin GCC 与 Python2-dev 编译环境；
Python 2

四、 算法设计 [1]

4.1 背景介绍

4.1.1 网络流的定义与性质

流网络$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$是一个有向图，图中每条边$(u, v) \in \boldsymbol{E}$有一个非负的容量值$c(u, v) \geq 0$。而且，如果边集合$\boldsymbol{E}$包含一条边$(u, v)$，则图中不存在反方向的边$(v, u)$。如果$(u, v) \notin \boldsymbol{E}$，为方便起见，定义$c(u, v) = 0$，并且在图中不允许有自循环。在流网络的所有节点中，我们特别分辨出两个特殊的节点：源节点s和汇点t。假定每个节点都在从源节点到汇点的某条路径上。因此，流网络图是联通的，并且由于除了源节点之外的每个节点都至少有一条进入的边，我们有$|\boldsymbol{E}| \geq |\boldsymbol{V}| - 1$.

网络$\boldsymbol{G}$中的流是一个实值函数f: $\boldsymbol{V} \times \boldsymbol{V} \to \mathbb{R}$，它满足下面两条性质：

1. 流量限制：对于所有的节点$u, v \in \boldsymbol{V}$，要求$0 \leq f(u, v) \leq c(u, v)$;
2. 流量守恒：对于所有的节点$u \in \boldsymbol{V} - \{s, t\}$，要求

1 此处的伪代码中，矩阵运算符的意义均与 MATLAB 语言一致，如矩阵的左除、右除和点除等。

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

我们称非负数值$f(u,\ v)$为从节点u到节点v的流。一个流的值$|f|$的定义如下：

$$|f| = \sum_{v\in \boldsymbol{V}} f(s,\ v) - \sum_{v\in \boldsymbol{V}} f(v,\ s)$$

也就是说，流f的值是从源节点流出的总流量减去流入源节点的总流量。一般而言流入源节点的流量为零。

4.1.2 Ford-Fulkerson 方法

准确来讲这是一个方法而不是算法，因为它的运行效率由实现方法决定。Ford-Fulkerson 方法依赖于三种重要的思想：残存网络、增广路径和切割。其中最大流最小割定理是精髓，该定理以流网络的切割来表述最大流的值。当 Ford-Fulkerson 方法中的找寻增广路径步骤采用广度优先拓扑排序的时候，这个方法被称为 Edmonds-Karp 算法。

Algorithm	朴素的 **FORD-FULKERSON(**$\boldsymbol{G},\ s,\ t$**)**
Input:	网络流图$\boldsymbol{G}$
	源节点s
	汇点t
Output	最大流f
Begin	
Step 1	生成$\boldsymbol{G}$上的一个零流f
Step 2	**if** 还能从图$\boldsymbol{G}$的基于流f的残存网络图中找到一条增广路径p
	利用这条增广路p为f增加流量
Step 3	输出f，GOTO **End**
End	

4.1.3 残存网络

从直观上看，给定流网络$\boldsymbol{G}$和流量f，残存网络$\boldsymbol{G}_f$由那些仍有空间对流量进行调整的边构成。流网络的一条边可以允许的额外流量等于该边的容量减去该边上的流量。如果该差值为正，则将这条边至于图$\boldsymbol{G}_f$中，并将其残存容量设置为$c_f(u,\ v) = c(u,\ v) - f(u,\ v)$。对流网络$\boldsymbol{G}$而言，只有能够允许额外流量的边才能加入到图$\boldsymbol{G}_f$中。

残存网络$\boldsymbol{G}_{\mathrm{f}}$还可能包含图$\boldsymbol{G}$中不存在的边。算法对总流量进行操作，目标是增加总流量，因此算法可能对某些特定边上的流量进行缩减。为了表示一个正流量$f(u,\ v)$的缩减，我们将边$(v,\ u)$加入到图$\boldsymbol{G}_f$中，并将其残存容量设置为$c_f(v,\ u) = f(u,\ v)$。这就是说一条边所能允许的反向流量最多将其正向流量抵消。残存网络中的这些反向边允许算法将已经发送出来的流量发送回去。

更加形式化地说，假定有一个流网络$\boldsymbol{G}=(\boldsymbol{V},\ \boldsymbol{E})$，源节点为$s$，汇点为$t$。设$f$为图$\boldsymbol{G}$中的一个流，考虑节点对$u,\ v\in\boldsymbol{V}$，定义残存容量$c_f(u,\ v)$如下：

$$c_f(u,\ v)\begin{cases}c(u,\ v)-f(u,v) & (u,\ v)\in\boldsymbol{E}\\ f(v,\ u) & (v,\ u)\in\boldsymbol{E}\\ 0 & \text{else}\end{cases}$$

给定一个流网络$\boldsymbol{G}=(\boldsymbol{V},\ \boldsymbol{E})$和一个流$f$，则由$f$所诱导的图$\boldsymbol{G}$的残存网络为$\boldsymbol{G}_f(\boldsymbol{V},\ \boldsymbol{E}_f)$，其中

$$\boldsymbol{E}_f=\{(u,\ v)\in\boldsymbol{V}\colon c_f(u,\ v)>0\}$$

残存网络并不是一个流网络，因为边和其反向边同时存在。除此之外，这个残存网络与原流网络具有完全相同的性质。

4.1.4 增广路径

给定流网络$\boldsymbol{G}=(\boldsymbol{V},\ \boldsymbol{E})$和一个流$f$，增广路径$p$是残存网络$\boldsymbol{G}_f(\boldsymbol{V},\ \boldsymbol{E}_f)$中一条从源节点$s$到汇点$t$的简单路径。我们称在一条增广路径$p$上能够为每条边增加的流量的最大值为$p$的残余容量，该容量由下面的表达式给出：

$$c_f(p)=\min\{c_f(u,\ v)\colon(u,\ v)\in p\}$$

4.1.5 流网络的切割

最大流最小割定理告诉我们，一个流是最大流当且仅当其残存网络中不包含任何增广路径。

流网络$\boldsymbol{G}=(\boldsymbol{V},\ \boldsymbol{E})$中的一个切割$(\boldsymbol{S},\ \boldsymbol{T})$将节点集合$\boldsymbol{V}$划分为$\boldsymbol{S}$和$\boldsymbol{T}=\boldsymbol{V}-\boldsymbol{S}$两个集合，使得源节点$s\in\boldsymbol{S}$，汇点$t\in\boldsymbol{T}$。若$f$是一个流，则定义横跨切割$(\boldsymbol{S},\ \boldsymbol{T})$的净流量$f(\boldsymbol{S},\ \boldsymbol{T})$定义如下：

$$f(\boldsymbol{S},\ \boldsymbol{T})=\sum_{u\in\boldsymbol{S}}\sum_{v\in\boldsymbol{T}}f(u,\ v)-\sum_{u\in\boldsymbol{S}}\sum_{v\in\boldsymbol{T}}f(v,\ u)$$

切割$(\boldsymbol{S},\ \boldsymbol{T})$的容量是：

$$c(\boldsymbol{S},\ \boldsymbol{T})=\sum_{u\in\boldsymbol{S}}\sum_{v\in\mathbf{T}}c(\mathbf{u},\ \mathbf{v})$$

一个网络的最小切割是整个网络中容量最小的切割。

可以证明，对于给定流f，横跨任何切割的净流量都相同且等于$|f|$。同样可以证明，网络中的任意流的值不超过任意切割的容量。这个通过不等式放缩容易得到。

最大流最小切割定理：设f为网络$\boldsymbol{G}=(\boldsymbol{V},\ \boldsymbol{E})$的一个流，该流网络的源节点为$s$，汇点为$t$，则下面的条件是等价的：

1. f是$\boldsymbol{G}$的一个最大流
2. 残存网络中不包括任何增广路径；
3. $|f|=(\boldsymbol{S},\ \boldsymbol{T})$，其中$(\boldsymbol{S},\ \boldsymbol{T})$是流网络$\boldsymbol{G}$的某个切割。

有了上面的铺垫，可以在假定存在有选择增广路径的基础上给出这个算法。

Algorithm **FORD-FULKERSON(G, s, t)**
Input: 网络流图G
源节点s
汇点t
Output 最大流f
Begin
Step 1 G上每一条边的流量f都设为0，GOTO **Step 2**

Step 2 **if** 还能从图G的基于流f的残存网络图G_f中找到一条增广路径p
let $c_f(p) = \min\{c_f(u, v): (u, v) \in p\}$
对于路径p中的任意一条边(u, v)，若$(u, v) \in G.E$
let $(u, v).f = (u, v).f + c_f(p)$
else $(u, v).f = (u, v).f - c_f(p)$
输出f，GOTO **End**

End

这个算法的核心十分简单，在残存网络中的增广路径上的边，要么是原来图中的边（权重不一样，但是起始点一样），要么是原来边的反向边（如果原来权重为满，即边上流量小于边的容量，就会频繁出现这种情况）。这时就要进行调整，原来有的那种要增加，反向的要减少一点。

五、 程序代码

5.1 程序描述

以下是程序中的核心片段，求最大流。其他构建器之类的函数在代码文件夹中给出。

```
1   float Graph_maxflow(FlowGraph g)
2   {
3       int n = g->numVertices, i;
4       struct MaxFlowInfo mfi;
5       struct Vertex *v;
6       struct Edge *e;
7       float increment, maxflowVal = 0.0;
8       mfi.visited = (int *) malloc(n * sizeof(int));
9       mfi.reverseEdges = (struct Edge **) malloc(g->numEdges * sizeof(struct Edge *));
10      mfi.numReverseEdges = 0;
11      mfi.queue = (struct Vertex **) malloc((n+2) * sizeof(struct Vertex *));
12
13      /* While there exists an augmenting path, increment the flow along
14         this path. */
15      while(findPath(g, &mfi)) {
16          /* Determine the amount by which we can increment the flow. */
17          increment = INFINITY;
18          v = g->sink;
```

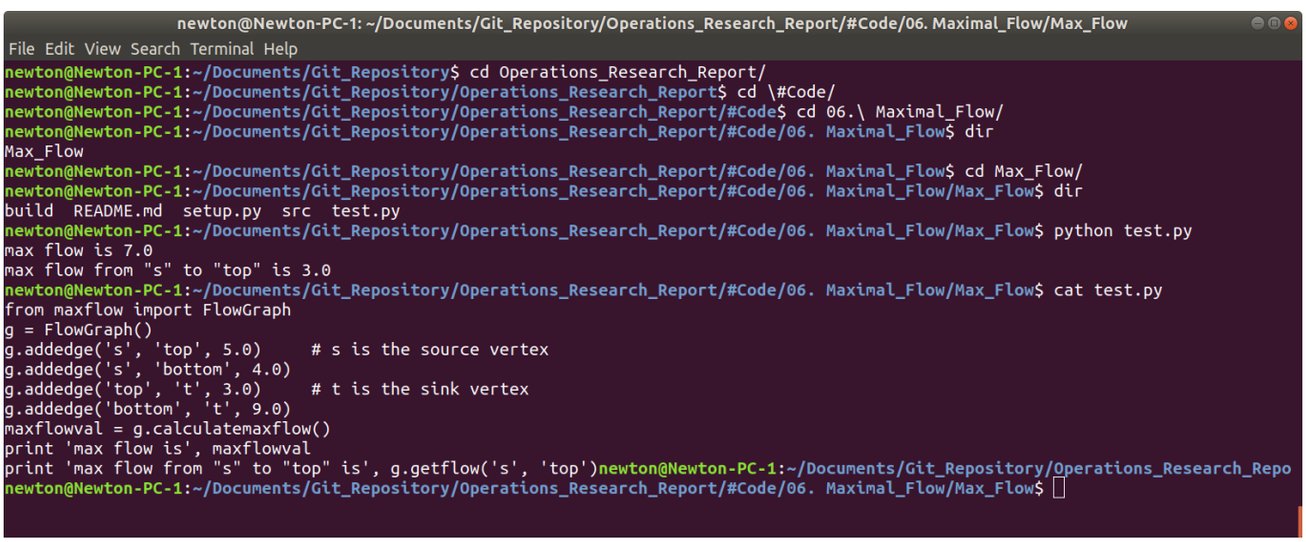
```
        while(v != g->source) {
            increment = MIN(increment, v->predEdge->capacity - v->predEdge->flow);
            v = v->predEdge->from;
        }
        /* Now increment the flow. */
        v = g->sink;
        while(v != g->source) {
            addFlow(&mfi, v->predEdge, increment);
            v = v->predEdge->from;
        }
        maxflowVal += increment;
    }

    free(mfi.visited);
    free(mfi.queue);
    for(i = 0; i < mfi.numReverseEdges; i++) {
        e = mfi.reverseEdges[i];
        e->reverseEdge->reverseEdge = NULL;
        e->from->degree--;
        free(e);
    }
    free(mfi.reverseEdges);
    return maxflowVal;
}
```

程序代码 1

六、 运行结果

6.1 代码分析

社区代码贡献者把这个做成了 Python2 的包，通过在 Ubuntu 18.04 上运行，有如下结果：



运行结果 1

七、 实验体会

这个算法比较简单，最大流最小割定理指出该具体如何实现。感谢开源社区 Github[3]提供代码让我在有限的时间里能得到高质量的代码进行修改与测试。

感谢开源社区[3]提供的高质量代码。

八、 参考文献

[1] HILLIER F S, LIEBERMAN G J. 运筹学导论 [M]. 9th ed. 北京: 清华大学出版社, 2010.

[2] 林锐. 高质量 C++/C 编程指南 [M]. 1.0 ed., 2001.

[3] https://github.com/webblue/maxflow