

# 第一章 图论高级算法

## 1.1 最大流

最大流算法分成两大类：增广路（augmenting path）算法与预流推进（preflow-push）算法。这一节介绍的三个算法，都属于增广路算法。下面给出几个定义。

### 流网络

网络流的研究对象是流网络。流网络  $G = (V, E, c, s, t)$  是一个有向图， $V$ 、 $E$  是其点集与边集，点和边的数目分别记作  $n$ 、 $m$ 。  $c$  是容量函数，每条边  $((u, v) \in E)$  都有一容量  $c(u, v) \in \mathbb{N}$ 。  $s$  和  $t$  是网络中的两个特殊点，称作源点和汇点。为简便计，流网络简称「网络」或「图」，简记作  $G = (V, E)$ 。

自环在网络中无意义，我们规定图  $G$  中不含自环。下文在论述、证明关于网络流的原理、性质或定理时，为了表示上的方便，我们对流网络做出两条限定：

1. 图中不存在重边；
2. 图中不存在反向边，即若  $(u, v) \in E$ ，则  $(v, u) \notin E$ 。

这两条限定都不妨碍一般性。我们可以通过将容量相加将重边合为一条边，反向边可以通过新增一个节点来消除。请读者注意，上文所谓「表示上的方便」是指一条边可以通过两个端点唯一确定。下文我们要介绍的算法和代码可以处理含有重边或反向边的图，这两条限定都不是根本性的，仅仅是为了方便表述而已。

## 流

流是满足下述两个性质的实值函数  $f: V \times V \rightarrow \mathbb{R}$ :

**容量限制**: 对任意  $u, v \in V$ , 有  $0 \leq f(u, v) \leq c(u, v)$ 。

**流守恒**: 对任意  $u \in V - \{s, t\}$ , 有  $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$

$f(u, v)$  即边  $(u, v)$  上的流量, 若  $(u, v) \notin E$ ,  $f(u, v) = 0$ 。从源点  $s$  到汇点  $t$  的总流量称作流  $f$  的值, 记作  $|f|$ , 不难得出

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

最大流问题即求给定的网络  $G$  中的一个值最大的流。

### 1.1.1 增广路方法

增广路方法是求解最大流问题的一种方法。本章要介绍的三个最大流算法都是基于增广路方法的。增广路算法涉及三个重要概念: 残余网络, 增广路, 割。

#### 残量网络

给定流网络  $G = (V, E)$  和  $G$  上的一个流  $f$ , 残量网络  $G_f$  是由  $G$  和  $f$  所导出的一个网络。首先定义残余容量  $c_f$ :

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{若 } (u, v) \in E, \\ f(v, u) & \text{若 } (v, u) \in E, \\ 0 & \text{其他情况.} \end{cases}$$

这里需要指出我们提出限制 2 的用意。 $(u, v) \in E$  和  $(v, u) \in E$  同时成立会给  $c_f$  的定义带来形式上的不便。残量网络  $G_f$  定义为  $G_f = (V, E_f)$ , 其中  $E_f = \{(u, v) \in V \times V: c_f(u, v) > 0\}$ 。除了可能含有反向边, 残量网络也符合流网络的定义; 而我们已经指出「不含反向边」并非根本性的要求, 我们可以用残余容量  $c_f$  类似地定义残量网络上的流, 称作残量流。

我们考虑残量流的原因在于, 借助残量流  $f'$ , 可以将原网络  $G$  上的流  $f$  修改成一个值更大的流  $f \uparrow f'$ 。用  $f'$  增广  $f$ , 这正是「增广」二字含义所

在。增广方法为:

$$(f \uparrow f')(u, v) = \begin{cases} f(u, v) + f'(u, v) - f'(v, u) & \text{若 } (u, v) \in E, \\ 0 & \text{其他情况.} \end{cases}$$

不难证明  $|f \uparrow f'| = |f| + |f'|$ 。

### 增广路

增广路是残量网络  $G_f$  上从  $s$  到  $t$  的一条简单路径。有了增广路  $p$ , 很容易得到一个残量流  $f_p$ 。

$$f_p(u, v) = \begin{cases} c_f(p) & \text{若边 } (u, v) \text{ 在路径 } p \text{ 上,} \\ 0 & \text{其他情况.} \end{cases}$$

其中  $c_f(p) = \min\{c_f(u, v) : (u, v) \text{ 在路径 } p \text{ 上}\}$ ,  $c_f(p)$  称作路径  $p$  的残余容量。易见,  $|f_p| = c_f(p) > 0$ 。

增广路方法即, 从图  $G$  上的某个初始流  $f$  (比如零流) 开始, 在  $G_f$  找一条增广路  $p$ ; 沿着  $p$  增广, 更新  $f$  和  $G_f$ ; 如此循环, 直到  $G_f$  上找不到增广路。此时  $f$  便是  $G$  上的一个最大流。

### 流网络的割

为了给出最大流最小割定理, 我们先介绍割的概念。将流网络  $G = (V, E)$  的点集  $V$  划分成两个子集  $S$  和  $T = V - S$  使得  $s \in S$  且  $t \in T$ ,  $(S, T)$  称作  $G$  的一个割。令  $f$  为  $G$  上的一个流, 割  $(S, T)$  之间的净流  $f(S, T)$  定义为

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

不难证明, 对  $G$  的任意一个割  $(S, T)$  都有  $f(S, T) = |f|$ 。割  $(S, T)$  的容量  $c(S, T)$  定义为

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

网络的最小割即所有割之中容量最小者。显然, 对于  $G$  上的任意一个流  $f$  和  $G$  的任意一个割  $(S, T)$  都有  $|f| \leq c(S, T)$ 。

**定理 1** (最大流最小割定理). 若  $f$  是流网络  $G = (V, E, c, s, t)$  上的一个流, 则下列三个命题等价:

1.  $f$  是  $G$  上的一个最大流。
2. 残量网络  $G_f$  上无增广路。
3. 存在某个割  $(S, T)$  满足  $|f| = c(S, T)$ 。

证明. (1) $\Rightarrow$ (2): 显然。

(2) $\Rightarrow$ (3): 假设  $G_f$  中无增广路, 即  $G_f$  上不存在从  $s$  到  $t$  的路径。令  $S = \{v \in V: G_f \text{ 上有从 } s \text{ 到 } v \text{ 的路径}\}, T = V - S$ , 易见  $t \notin S$ , 则  $(S, T)$  是一个割。考虑点对  $u \in S$  和  $v \in T$ 。若  $(u, v) \in E$ , 则必有  $f(u, v) = c(u, v)$ ; 因为若不然则有  $(u, v) \in E_f$ , 即  $v \in S$ 。若  $(v, u) \in E$ , 则必有  $f(v, u) = 0$ ; 因为若不然则有  $c_f(u, v) = f(v, u) > 0$ , 即  $(u, v) \in E_f$ , 仍有  $v \in S$ 。若  $(u, v) \notin E$  且  $(v, u) \notin E$ , 则  $f(u, v) = f(v, u) = 0$ 。因此我们有

$$\begin{aligned} f(S, T) &= \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{v \in T} \sum_{u \in S} f(v, u) \\ &= \sum_{u \in S} \sum_{v \in T} c(u, v) - \sum_{v \in T} \sum_{u \in S} 0 \\ &= c(S, T) \end{aligned}$$

所以  $|f| = f(S, T) = c(S, T)$ 。(3) $\Rightarrow$ (1): 由于对任意割  $(S, T)$  都有  $|f| \leq c(S, T)$ ,  $|f| = c(S, T)$  蕴含着  $f$  是一个最大流。□

不难看出, 高效地实现增广路方法应从两个方面考虑:

1. 如何快速地在残量网络  $G_f$  上找一条增广路。
2. 如何减少增广的次数。

我们已经知道, 通过深度优先搜索 (DFS) 或宽度优先搜索 (BFS) 可在线性时间内找到一条增广路。在下一小节中我们将证明, 如果每次都沿着最短增广路 (shortest augmenting path, SAP) 增广, 那么增广次数是  $O(VE)$  的。沿着最短增广路增广的算法统称为最短增广路算法。下面三个小节中要介绍的算法都属于最短增广路算法。

### 1.1.2 Edmonds-Karp 算法

Edmonds-Karp 是 SAP 算法的朴素实现。代码如下：

---

```
#include <climits>
#include <queue>
const int N = 1e5 + 5, M = 1e5 + 5;
struct Edge{
    int v, rc, next;    //rc: residual capacity
}E[M*2];
int head[N], sz;
void add_edge(int u, int v, int c){
    E[sz] = {v, c, head[u]};
    head[u] = sz++;
    E[sz] = {u, 0, head[v]};
    head[v] = sz++;
}
void init(){
    sz = 0;
    memset(head, -1, sizeof head);
}
std::queue<int> que;
int pre[N];
int ek(int s, int t){
    for(int ans = 0; ; ){
        memset(pre, -1, sizeof pre); que.push(s);
        while(que.size()){
            int u = que.front(); que.pop();
            for(int i = 0; i != -1; i = E[i].next){
                if(E[i].rc > 0 && pre[E[i].v] == -1)
                    if(E[i] == t){
                        int cp = INT_MAX;
                        for(int j = i; j != -1; j = pre[E[j ^ 1].v])
                            cp = min(cp, E[j].rc);
                        for(int j = i; j != -1; j = pre[E[j ^ 1].v])
```

```

        E[j].rc -= cp, E[j ^ 1].rc += cp;
        ans += cp;
        for(; que.size(); que.pop());
        break;
    }
    else{
        pre[E[i].v] = i;
        que.push(E[i].v);
    }
}
}
if(pre[t] == -1) return ans;
}
}

```

---

说明:

1. 用链式前项星存图。
2. 对任意边  $e \in E$ ,  $e$  在边数组中的下标  $\text{idx}(e)$  为偶数,  $e$  的反向边  $e'$  的下标  $\text{idx}(e') = \text{idx}(e) + 1$ 。

下面我们来分析 Edmonds-Karp 算法的复杂度。用  $\delta_f(u, v)$  表示残量网络  $G_f$  上从  $u$  到  $v$  的距离,  $G_f$  中边的长度都为 1。

**引理 1.** 用 Edmonds-Karp 算法求流网络  $G = (V, E, c, s, t)$  的最大流的过程中, 对任意节点  $v \in V - \{s, t\}$ , 残量网络  $G_f$  上从源点  $s$  到  $v$  的距离  $\delta_f(s, v)$  随着每次增广而递增。

**证明.** 假设此命题不成立。设  $v$  是  $V - \{s, t\}$  中一点; 令  $f$  为「 $s$  到  $v$  的距离减小」首次出现之前的流, 令  $f'$  为  $f$  增广之后的流。再令  $v$  为满足  $\delta_f(s, v) > \delta_{f'}(s, v)$  的点中  $\delta_{f'}(s, v)$  最小的一个点。令  $p = s \rightsquigarrow u \rightarrow v$  为  $G_{f'}$  中从  $s$  到  $v$  的一条最短路, 因而有  $(u, v) \in E_{f'}$  且

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) - 1. \quad (1.1)$$

又由于  $v$  是满足  $\delta_f(s, v) > \delta_{f'}(s, v)$  的点中  $\delta_{f'}(s, v)$  最小者, 我们有

$$\delta_{f'}(s, u) \geq \delta_f(s, u). \quad (1.2)$$



1.1.3 Dinic 算法

1.1.4 ISAP 算法

1.1.5 网络流的建图

1.2 费用流

1.3 二分图

1.3.1 最大流和二分图

1.3.2 匈牙利算法

1.3.3 二分图模型应用

1.4 图的连通

1.4.1 强连通-Tarjan 算法

1.4.2 双连通

1.4.3 2-SAT 问题