Chương 9. Luồng cực đại trên mạng

9.1. Các khái niệm và bài toán

9.1.1. Mạng

Mạng (*flow network*) là một bộ năm $G = (V, E, c, s, t)$, trong đó:

V và E lần lượt là tập đỉnh và tập cung của một đồ thị có hướng không có khuyên (cung nối từ một đỉnh đến chính nó).

s và t là hai đỉnh phân biệt thuộc V, s gọi là *đỉnh phát* (*source*) và t gọi là *đỉnh thu* (*sink*).

c là một hàm xác định trên tập cung E:

$$c: E \longrightarrow [0, +\infty)$$
$$e \longmapsto c(e)$$

gán cho mỗi cung $e \in E$ một số không âm gọi là *sức chứa (capacity)*[*] $c(e) \geq 0$.

Bằng cách thêm vào mạng một số cung có sức chứa 0, ta có thể giả thiết rằng mỗi cung $e = (u, v) \in E$ luôn có tương ứng duy nhất một cung ngược chiều, ký hiệu – $e = (v, u) \in E$, gọi là *cung đối* của cung e, ta cũng coi e là cung đối của cung $-e$ (tức là – $(-e) = e$). Có thể thấy rằng số cung cần thêm vào mạng là một đại lượng $O(E)$.

Chú ý rằng **mạng là một đa đồ thị**, tức là giữa hai đỉnh có thể có nhiều cung nối.

Để thuận tiện cho việc trình bày, ta quy ước các ký hiệu sau:

Với X, Y là hai tập con của tập đỉnh V và $f: E \to \mathbb{R}$ là một hàm xác định trên tập cạnh E:

Ký hiệu $\{X \to Y\}$ là tập các cung nối một từ một đỉnh thuộc X tới một đỉnh thuộc Y:

$$\{X \to Y\} = \{e = (u, v) \in E: u \in X, y \in Y\}$$

Ký hiệu $f(X, Y)$ là tổng các giá trị hàm f trên các cung $e \in \{X \to Y\}$:

$$f(X, Y) = \sum_{e \in X \to Y} f(e)$$

9.1.2. Luồng

Luồng (*flow*) trên mạng G là một hàm:

$$f: E \longrightarrow \mathbb{R}$$
$$e \longmapsto f(e)$$

gán cho mỗi cung e một số thực $f(e)$, gọi là luồng trên cung e, thỏa mãn ba ràng buộc sau đây:

- Ràng buộc về sức chứa (*Capacity constraint*): Luồng trên mỗi cung không được vượt quá sức chứa của cung đó: $\forall e \in E: f(e) \leq c(e)$.

---

[*] Từ này còn có thể dịch là "khả năng thông qua" hay "lưu lượng"

- Ràng buộc về tính đối xứng lệch (*Skew symmetry*): Với $\forall e \in E$, luồng trên cung e và luồng trên cung đối $-e$ có cùng giá trị tuyệt đối nhưng trái dấu nhau: $\forall e \in E: f(e) = -f(-e)$.
- Ràng buộc về tính bảo tồn (*Flow conservation*): Với mỗi đỉnh v không phải đỉnh phát và cũng không phải đỉnh thu, tổng luồng trên các cung đi ra khỏi v bằng 0: $\forall v \in V - \{s,t\}: f(\{v\},V) = 0$.

Từ ràng buộc về tính đối xứng lệch và tính bảo tồn, ta suy ra được: Với mọi đỉnh $v \in V - \{s,t\}$, tổng luồng trên các cung đi vào v bằng 0: $f(V,\{v\}) = 0$.

Giá trị của luồng f trên mạng G được định nghĩa bằng tổng luồng trên các cung đi ra khỏi đỉnh phát:

$$|f| = f(\{s\},V) \tag{9.1}$$

Bài toán luồng cực đại trên mạng (*maximum-flow problem*): Cho một mạng G với đỉnh phát s và đỉnh thu t, hàm sức chứa c, hãy tìm một luồng có giá trị lớn nhất trên mạng G.

9.1.3. Luồng dương

Luồng dương (*positive flow*) trên mạng G là một hàm

$$\varphi: E \longrightarrow \mathbb{R}_{\geq 0}$$
$$e \longmapsto \varphi(e)$$

gán cho mỗi cung e một số thực không âm $\varphi(e)$ gọi là luồng dương trên cung e thỏa mãn hai ràng buộc sau đây:

- Ràng buộc về sức chứa (*Capacity constraint*): Luồng dương trên mỗi cung không được vượt quá sức chứa của cung đó: $\forall e \in E: 0 \leq \varphi(e) \leq c(e)$.
- Ràng buộc về tính bảo tồn (*Flow conservation*): Với mỗi đỉnh v không phải đỉnh phát và cũng không phải đỉnh thu, tổng luồng dương trên các cung đi vào v bằng tổng luồng dương trên các cung đi ra khỏi v: $\forall v \in V - \{s,t\}: \varphi(V,\{v\}) = \varphi(\{v\},V)$.

Giá trị của một luồng dương được định nghĩa bằng tổng luồng dương trên các cung đi ra khỏi đỉnh phát trừ đi tổng luồng dương trên các cung đi vào đỉnh phát*:

$$|\varphi| = \varphi(\{s\},V) - \varphi(V,\{s\}) \tag{9.2}$$

9.1.4. Mối quan hệ giữa luồng và luồng dương

Bổ đề 9-1

Cho $f: E \to \mathbb{R}$ là một luồng trên mạng $G = (V,E,c,s,t)$. Khi đó hàm:

---

* Một số tài liệu khác đưa vào thêm ràng buộc: đỉnh phát s không có cung đi vào và đỉnh thu t không có cung đi ra. Khi đó giá trị luồng dương bằng tổng luồng dương trên các cung đi ra khỏi đỉnh phát. Cách hiểu này có thể quy về một trường hợp riêng của định nghĩa.

$$\varphi: E \longrightarrow \mathbb{R}_{\geq 0}$$
$$e \longmapsto \varphi(e) = \begin{cases} f(e), \text{nếu } f(e) \geq 0 \\ 0, \text{nếu } f(e) < 0 \end{cases}$$

là một luồng dương trên mạng và $|\varphi| = |f|$

Chứng minh

Ta chứng minh φ thỏa mãn tất cả các tính chất của luồng dương:

Ràng buộc về sức chứa:

$\forall e \in E^+$, ta có $\varphi(e) \geq 0$ theo cách xây dựng φ.

Nếu $f(e) \geq 0$, từ ràng buộc về sức chứa đối với luồng $f(e) \leq c(e)$. Ta suy ra $\varphi(e) = f(e) \leq c(e)$.

Nếu $f(e) < 0$, hiển nhiên $\varphi(e) = 0 \leq c(e)$

Vậy $\forall e \in E: 0 \leq \varphi(e) \leq c(e)$.

Ràng buộc về tính bảo tồn:

Xét một đỉnh v bất kỳ, ta có:

$$\begin{aligned} f(\{v\}, V) &= \sum_{e \in \{\{v\} \to V\}} f(e) \\ &= \sum_{\substack{e \in \{\{v\} \to V\} \\ f(e) \geq 0}} f(e) + \sum_{\substack{e \in \{\{v\} \to V\} \\ f(e) < 0}} f(e) \\ &= \underbrace{\sum_{\substack{e \in \{\{v\} \to V\} \\ f(e) \geq 0}} f(e)}_{\varphi(\{v\}, V)} - \underbrace{\sum_{\substack{-e \in \{V \to \{v\}\} \\ f(-e) \geq 0}} f(-e)}_{\varphi(V, \{v\})} \end{aligned}$$

Điều này có thể hiểu như sau: Tổng luồng f ra khỏi v bằng tổng luồng f trên các cung có $f(e) \geq 0$ cộng với tổng luồng f trên những cung có $f(e) < 0$ xét trên những cung e ra khỏi v. Những cung ra khỏi v mang luồng âm $(f(e) < 0)$ lại có một cung đối đi vào v mang luồng dương với cùng giá trị tuyệt đối $(f(-e) = -f(e) > 0)$. Vì vậy cộng luồng trên các cung mang luồng âm ra khỏi v tương đương với việc trừ đi tổng luồng trên các cung mang luồng dương đi vào v. Tổng hợp lại ta được: Tổng luồng f ra khỏi v bằng tổng luồng dương φ ra khỏi v trừ đi tổng luồng dương φ đi vào v.

$$f(\{v\}, V) = \varphi(\{v\}, V) - \varphi(V, \{v\})$$

Nếu v không phải đỉnh phát cũng không phải đỉnh thu. Tính bảo tồn luồng cho ta $f(\{v\}, V) = 0$ hay $\varphi(\{v\}, V) = \varphi(V, \{v\})$, tức là tổng luồng dương φ ra khỏi v bằng tổng luồng dương φ đi vào v.

c) Cũng từ chứng minh trên thay v bởi s, ta có:

$$|\varphi| = \varphi(\{s\}, V) - \varphi(V, \{s\}) = f(\{s\}, V) = |f|$$

Bổ đề 9-2

Cho $\varphi: E \to \mathbb{R}_{\geq 0}$ là một luồng dương trên mạng $G = (V, E, c, s, t)$. Khi đó hàm

$$\begin{aligned} f: E &\longrightarrow \mathbb{R} \\ e &\longmapsto f(e) = \varphi(e) - \varphi(-e) \end{aligned}$$

là một luồng trên mạng G và $|f| = |\varphi|$

Chứng minh

Trước hết ta chứng minh f thỏa mãn tất cả các ràng buộc về luồng:

Ràng buộc về sức chứa: Với $\forall e \in E$:

$$f(e) = \varphi(e) - \underbrace{\varphi(-e)}_{\geq 0} \leq \varphi(e) \leq c(e)$$

Ràng buộc về tính đối xứng lệch: Với $\forall e \in E$

$$\begin{aligned} f(e) &= \varphi(e) - \varphi(-e) \\ &= -\big(\varphi(-e) - \varphi(e)\big) \\ &= -f(-e) \end{aligned}$$

Ràng buộc về tính bảo tồn: $\forall v \in V$, ta có:

$$\begin{aligned} f(\{v\}, V) &= \sum_{e \in \{\{v\} \to V\}} \big(\varphi(e) - \varphi(-e)\big) \\ &= \left(\sum_{e \in \{\{v\} \to V\}} \varphi(e)\right) - \left(\sum_{e \in \{\{v\} \to V\}} \varphi(-e)\right) \\ &= \left(\sum_{e \in \{\{v\} \to V\}} \varphi(e)\right) - \left(\sum_{-e \in \{V \to \{v\}\}} \varphi(-e)\right) \\ &= \varphi(\{v\}, V) - \varphi(V, \{v\}) \end{aligned}$$

Nếu $v \neq s$ và $v \neq t$, ta có:

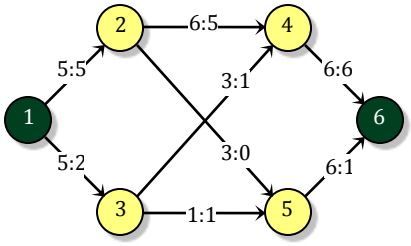
$$f(\{v\}, V) = \varphi(\{v\}, V) - \varphi(V, \{v\}) = 0$$

(Do tổng luồng dương đi ra khỏi v bằng tổng luồng dương đi vào v)

Nếu $v = s$, xét giá trị luồng f

$$|f| = f(\{s\}, V) = \varphi((\{s\}, V) - \varphi(V, \{s\}) = |\varphi|$$

Bổ đề 9-1 và Bổ đề 9-2 cho ta một mối tương quan giữa luồng và luồng dương. Khái niệm về luồng dương dễ hình dung hơn so với khái niệm luồng, tuy nhiên những định nghĩa về luồng tổng quát lại thích hợp hơn cho việc trình bày và chứng minh các thuật toán trong bài. Ta sẽ **sử dụng luồng dương trong các hình vẽ và output** (chỉ quan tâm tới các giá trị luồng dương $\varphi(e)$), còn các khái niệm về luồng sẽ được dùng để diễn giải các thuật toán.



Hình 9-1. Ví dụ về một mạng với đỉnh phát 1, đỉnh thu 6. Số trên các cung là sức chứa: luồng dương trên cung

Mô hình trực quan nhất của luồng dương là các đường ống dẫn nước từ trạm nguồn s tới nơi trạm đích t. Các đỉnh khác của mạng là các trạm trung chuyển và các cung là đường ống dẫn nước một chiều từ một trạm tới một trạm khác. Sức chứa của đường ống là tiết diện của ống, tỉ lệ thuận với lưu lượng nước tối đa có thể chảy qua trong một đơn vị thời gian. Trạm trung gian không có khả năng chứa nước, do vậy lượng nước chảy vào trạm trung gian phải được phát tán toàn bộ và ngay lập tức sang trạm khác qua các đường ống. Ta có ngay các ràng buộc: Trong mỗi đơn vị thời gian, lượng nước lưu thông qua mỗi đường ống không được vượt quá lưu lượng tối đa của đường ống và bao nhiêu nước đi vào một trạm trung chuyển thì cũng phải có bấy nhiêu nước đi ra khỏi trạm trung chuyển đó. Vấn đề đặt ra là điều khiển lượng nước vào các đường ống một cách hợp lý để lượng nước chuyển đi trong mỗi đơn vị thời gian là lớn nhất. Các vấn đề như phân luồng giao thông để tránh tắc nghẽn, lắp đặt mạch điện để đảm bảo không gặp phải sự cố quá tải đường dây đều có thể quy về bài toán luồng cực đại trên mạng.

Trong quá trình cài đặt thuật toán, các hàm c và f sẽ được xác định bởi tập các giá trị $\{c[e]\}_{e\in E}$ và $\{f[e]\}_{e\in E}$ nên ta có thể dùng lẫn các ký hiệu $c(e), f(e)$ (nếu muốn đề cập tới giá trị hàm) hoặc $c[e], f[e]$ (nếu muốn đề cập tới các biến số).

9.1.5. Một số tính chất cơ bản

Cho mạng $G = (V, E, c, s, t)$ và một luồng f trên G. Gọi $c(X, Y)$ là *lưu lượng* từ X sang Y và $f(X, Y)$ là giá trị luồng từ X sang Y.

Định lý 9-3

Cho f là một luồng trên mạng $G = (V, E, c, s, t)$, khi đó:

a) $\forall X \subseteq V$, ta có $f(X, X) = 0$.

b) $\forall X, Y \subseteq V$, ta có $f(X, Y) = -f(Y, X)$.

c) $\forall X, Y, Z \subseteq V$ và $X \cap Y = \emptyset$, ta có $f(X, Z) + f(Y, Z) = F(X \cup Y, Z)$.

d) $\forall X \subseteq V - \{s, t\}$, ta có $f(X, V) = 0$.

Chứng minh

a) $\forall X \subseteq V$, ta có:

$$f(X,X) = \sum_{e \in \{X \to X\}} f(e)$$

như vậy $f(e)$ xuất hiện trong tổng nếu và chỉ nếu $f(-e)$ cũng xuất hiện trong tổng. Theo tính đối xứng lệch của luồng: $f(e) = -f(-e)$, ta có $f(X,X) = 0$.

b) $\forall X, Y \subseteq V$, ta có :

$$f(X,Y) = \sum_{e \in \{X \to Y\}} f(e) = - \sum_{-e \in \{Y \to X\}} f(-e) = -f(Y,X)$$

c) $\forall X, Y, Z \subseteq V$ và $X \cap Y = \emptyset$, ta có:

$$f(X \cup Y, Z) = \sum_{e \in \{X \cup Y \to Z\}} f(e)$$

$$= \underbrace{\sum_{e \in \{X \to Z\}} f(e)}_{f(X,Z)} + \underbrace{\sum_{e \in \{Y \to Z\}} f(e)}_{f(Y,Z)}$$

d) $\forall X \subseteq V - \{s,t\}$, do

$$X = \bigcup_{u \in X} \{u\}$$

Nên theo chứng minh phần c):

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

Mỗi hạng tử của tổng: $f(\{u\},V)$ chính là tổng luồng trên các cung đi ra khỏi đỉnh u, do tính bảo tồn luồng và u không phải đỉnh phát cũng không phải đỉnh thu, hạng tử này phải bằng 0, suy ra $f(X,V) = 0$. Từ chứng minh phần b), ta còn suy ra $f(V,X) = 0$ nữa.

Định lý 9-4

Giá trị luồng trên mạng bằng tổng luồng trên các cung đi vào đỉnh thu

Chứng minh

Giả sử f là một luồng trên mạng $G = (V,E,c,s,t)$, ta có:

$$\begin{aligned} |f| &= f(\{s\},V) \\ &= f(V,V) - f(V - \{s\},V) \\ &= -f(V - \{s\},V) \\ &= f(V, V - \{s\}) \\ &= f(V,\{t\}) + f(V, V - \{s,t\}) \\ &= f(V,\{t\}) \end{aligned}$$

Hệ quả

Giá trị luồng dương trên mạng bằng tổng luồng dương đi vào đỉnh thu trừ tổng luồng dương ra khỏi đỉnh thu.

9.1.6. Mạng thặng dư

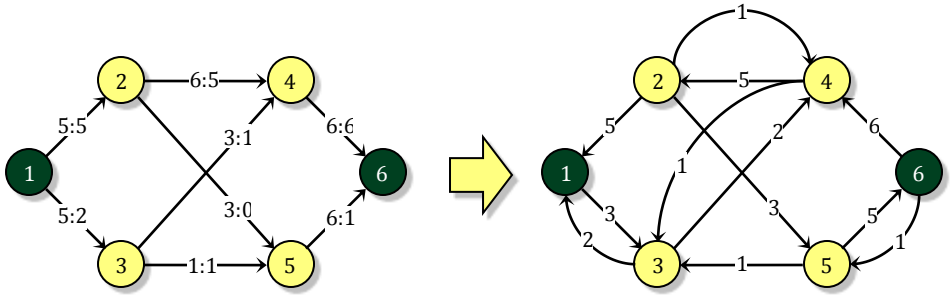
Với f là một luồng trên mạng $G = (V, E, c, s, t)$. Ta xét mạng G_f cũng là mạng G nhưng với hàm sức chứa mới cho bởi:

$$\begin{aligned} c_f &: E \longrightarrow [0, +\infty) \\ e &\longmapsto c_f(e) = c(e) - f(e) \end{aligned} \tag{9.3}$$

Mạng G_f xây dựng như vậy được gọi là *mạng thặng dư* (*residual network*) của mạng G sinh ra bởi luồng f. Sức chứa $c_f(e)$, còn gọi là *dư lượng* (*residual capacity*) của cung e, thực chất là lượng luồng tối đa chúng ta có thể đẩy thêm vào luồng $f(e)$ mà không làm vượt quá sức chứa $c(e)$.

Một cung trên G gọi là *cung bão hòa (saturated edge)* nếu dư lượng của cung đó bằng 0, ngược lại cung đó gọi là *cung thặng dư (residual edge)*. Ký hiệu E_f là tập các cung thặng dư trên mạng thặng dư G_f. Một đường đi chỉ qua các cung thặng dư trên G_f gọi là *đường thặng dư (residual path)*.

Các cung bão hòa của mạng G có dư lượng 0, cung này ít có ý nghĩa trong thuật toán nên chúng ta sẽ chỉ vẽ các cung thặng dư ($\in E_f$) trong các hình vẽ.



Hình 9-2. Một luồng trên mạng (số ghi trên các cung là: sức chứa:luồng dương) và mạng thặng dư tương ứng.

Hình 9-2 là một ví dụ về mạng thặng dư. Như đã quy ước, chúng ta chỉ vẽ các luồng dương. Đồ thị có cung $(2,4)$ với sức chứa $c(2,4) = 6$, tức là phải có cung đối $(4,2)$ với $c(4,2) = 0$. Luồng dương trên cung $(2,4)$ là $f^+(2,4) = f(2,4) = 5$, điều này cũng cho biết luồng trên cung $(4,2)$ là $f(4,2) = -5$ theo tính đối xứng lệch. Vậy trên mạng thặng dư, ta có cung $(2,4)$ với sức chứa $c(2,4) - f(2,4) = 6 - 5 = 1$ đồng thời có cung $(4,2)$ với sức chứa $c(4,2) - f(4,2) = 0 - (-5) = 5$.

Định lý 9-5

Cho f là một luồng trên mạng $G = (V, E, c, s, t)$. Khi đó nếu f' là một luồng trên G_f thì hàm:

$$f + f' : E \longrightarrow \mathbb{R}$$
$$e \longmapsto (f + f')(e) = f(e) + f'(e)$$

là một luồng trên mạng G với giá trị luồng $|f + f'| = |f| + |f'|$.

Chứng minh

Ta chứng minh $(f + f')$ thỏa mãn ba tính chất của luồng:

Ràng buộc về sức chứa: Với $\forall e \in E$:

$$\begin{aligned}(f + f')(e) &= f(e) + f'(e)\\ &\leq f(e) + \big(c(e) - f(e)\big)\\ &= c(e)\end{aligned}$$

Tính đối xứng lệch: Với $\forall e \in E$:

$$\begin{aligned}(f + f')(e) &= f(e) + f'(e)\\ &= -f(-e) - f'(-e)\\ &= -\big(f(-e) + f'(-e)\big)\\ &= -(f + f')(-e)\end{aligned}$$

Tính bảo tồn: Với $\forall u \in V$, tổng luồng $f + f'$ đi ra khỏi u bằng:

$$\begin{aligned}(f + f')(\{u\}, V) &= \sum_{e \in \{\{u\} \to V\}} \big(f(e) + f'(e)\big)\\ &= \sum_{e \in \{\{u\} \to V\}} f(e) + \sum_{e \in \{\{u\} \to V\}} f'(e)\\ &= f(\{u\}, V) + f'(\{u\}, V)\end{aligned}$$

Nếu $u \neq s$ và $u \neq t$, ta có $(f + f')(\{u\}, V) = f(\{u\}, V) + f'(\{u\}, V) = 0$.

Thay $u = s$, ta có

$$|f + f'| = (f + f')(\{s\}, V) = f(\{s\}, V) + f'(\{s\}, V) = |f| + |f'|$$

Định lý 9-6

Cho f và f' là hai luồng trên mạng $G = (V, E, c, s, t)$ khi đó hàm:

$$f' - f : E \longrightarrow \mathbb{R}$$
$$e \longmapsto (f' - f)(e) = f'(e) - f(e)$$

là một luồng trên mạng thặng dư G_f với giá trị luồng $|f' - f| = |f'| - |f|$.

Chứng minh

Ta chứng minh rằng $f' - f$ thỏa mãn ba tính chất của luồng

Ràng buộc về sức chứa: Với $\forall e \in E$:

$$\begin{aligned}(f' - f)(e) &= f'(e) - f(e)\\ &\leq c(e) - f(e)\end{aligned}$$

$$= c_f(e)$$

Tính đối xứng lệch: Với $\forall e \in E$:

$$\begin{aligned}(f' - f)(e) &= f'(e) - f(e) \\ &= -\big(f'(-e) - f(-e)\big) \\ &= -(f' - f)(-e)\end{aligned}$$

Tính bảo tồn: Với $\forall v \in V$

$$\begin{aligned}(f' - f)(\{v\}, V) &= \sum_{e \in \{\{v\} \to V\}} \big(f'(e) - f(e)\big) \\ &= \sum_{e \in \{\{v\} \to V\}} f'(e) - \sum_{e \in \{\{v\} \to V\}} f(e) \\ &= f'(\{v\}, V) - f(\{v\}, V)\end{aligned}$$

Nếu $u \neq s$ và $u \neq t$, ta có $(f' - f)(\{v\}, V) = f'(\{v\}, V) - f(\{v\}, V) = 0$.

Thay $u = s$, ta có

$$|f' - f| = (f' - f)(\{s\}, V) = f'(\{s\}, V) - f(\{s\}, V) = |f'| - |f|$$

9.2. Thuật toán Ford-Fulkerson

9.2.1. Đường tăng luồng

Với f là một luồng trên mạng $G = (V, E, c, s, t)$. Gọi P là một đường đi đơn từ s tới t trên mạng thặng dư G_f. *Giá trị thặng dư* (*residual capacity*) của đường P, ký hiệu Δ_P, được định nghĩa bằng dư lượng nhỏ nhất của các cung dọc trên đường P:

$$\Delta_P = \min\{c_f(e) : (e) \text{ nằm trên } P\}$$

Vì các dư lượng $c_f(e)$ là số không âm nên Δ_P luôn là số không âm. Nếu $\Delta_P > 0$ tức là đường đi P là một đường thặng dư, khi đó đường đi P gọi là một *đường tăng luồng (augmenting path)* tương ứng với luồng f.

Định lý 9-7

Cho f là một luồng trên mạng $G = (V, E, c, s, t)$, P là một đường tăng luồng trên G_f. Khi đó hàm $f_P : E \to \mathbb{R}$ định nghĩa như sau:

$$f_P(e) = \begin{cases} +\Delta_P, \text{ nếu } e \in P \\ -\Delta_p, \text{ nếu } -e \in P \\ 0, \text{ trường hợp khác} \end{cases} \qquad (9.4)$$

là một luồng trên G_f với giá trị luồng $|f_P| = \Delta_P > 0$.

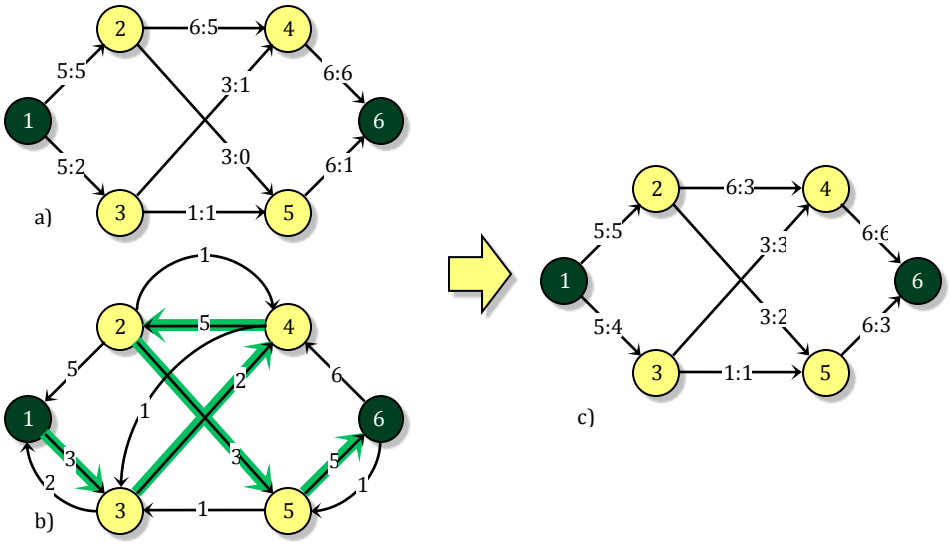
Chứng minh

Chúng ta sẽ không chứng minh cụ thể vì việc kiểm chứng f_P thỏa mãn ba tính chất của luồng khá dễ dàng. Bản chất của luồng f_P là đẩy một giá trị luồng Δ_P từ s tới t dọc theo các cung trên đường P, đồng thời kéo một giá trị luồng $-\Delta_P$ từ t về s theo hướng ngược lại*.

Định lý 9-5 và Định lý 9-7 cho ta một hệ quả sau:

Hệ quả 9-8

Cho f là một luồng trên mạng $G = (V, E, c, s, t)$ và P là một đường tăng luồng trên G_f, gọi f_P là luồng trên G_f định nghĩa như trong công thức (9.4). Khi đó $f + f_P$ là một luồng mới trên G với giá trị $|f + f_P| = |f| + |f_P| = |f| + \Delta_P$.



Hình 9-3. Tăng luồng dọc đường tăng luồng.

Hình 9-3 là ví dụ về cơ chế tăng luồng trên mạng với đỉnh phát 1, đỉnh thu 6 và luồng f giá trị 7 (hình a) (chú ý rằng ta chỉ vẽ các luồng dương cho đỡ rối). Với mạng thặng dư G_f (hình b), giả sử ta chọn đường đi $P = \langle 1,3,4,2,5,6 \rangle$ làm đường tăng luồng, giá trị thặng dư của P bằng $\Delta_P = 2$ (sức chứa của cung (3,4)). Luồng f_P trên G_f sẽ có các giá trị sau:

$$f_P(1,3) = f_P(3,4) = f_P(4,2) = f_P(2,5) = f_P(5,6) = 2$$
$$f_P(3,1) = f_P(4,3) = f_P(2,4) = f_P(5,2) = f_P(6,5) = -2$$

* Có thể hình dung cơ chế này như một quá trình điện phân: Bao nhiêu ion dương (cation) chuyển đến cực âm (catot) t thì cũng phải có bấy nhiêu ion âm (anion) chuyển đến cực dương (anot) s.

Cộng các giá trị này vào luồng f đang có, ta sẽ được một luồng mới trên G với giá trị 9 (hình c).

Cơ chế cộng luồng f_P vào luồng f hiện có gọi là *tăng luồng dọc theo đường tăng luồng P*.

9.2.2. Thuật toán Ford-Fulkerson

Thuật toán Ford-Fulkerson (Ford & Fulkerson, Flows in Networks, 1962) để tìm luồng cực đại trên mạng dựa trên cơ chế tăng luồng dọc theo đường tăng luồng. Bắt đầu từ một luồng f bất kỳ trên mạng (chẳng hạn luồng trên mọi cung đều bằng 0), thuật toán tìm đường tăng luồng P trên mạng thặng dư, gán $f := f + f_P$ để tăng giá trị luồng f và lặp lại cho tới khi không tìm được đường tăng luồng nữa.

```
f := «Một luồng bất kỳ»;
while «Tìm được đường tăng luồng P» do
   f := f + fP;
Output ← f;
```

9.2.3. Cài đặt

Chúng ta sẽ cái đặt thuật toán Ford-Fulkerson để tìm luồng cực đại trên mạng với khuôn dạng Input/Output như sau:

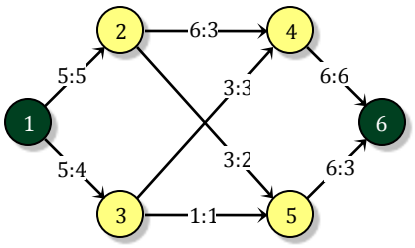
Input

- Dòng 1 chứa số đỉnh $n \leq 10^3$, số cung $m \leq 10^5$ của mạng, đỉnh phát s, đỉnh thu t.
- m dòng tiếp theo, mỗi dòng chứa ba số nguyên dương u, v, c tương ứng với một cung nối từ u tới v với sức chứa $c \leq 10^4$.

Output

Luồng cực đại trên mạng (như đã quy ước, chỉ đưa ra các luồng dương trên các cung).

Sample Input	Sample Output
6 8 1 6 5 6 6 4 6 6 3 5 1 3 4 3 2 5 3 2 4 6 1 3 5 1 2 5	Maximum flow: e[1] = (5, 6): c = 6, f = 3 e[2] = (4, 6): c = 6, f = 6 e[3] = (3, 5): c = 1, f = 1 e[4] = (3, 4): c = 3, f = 3 e[5] = (2, 5): c = 3, f = 2 e[6] = (2, 4): c = 6, f = 3 e[7] = (1, 3): c = 5, f = 4 e[8] = (1, 2): c = 5, f = 5 Value of flow: 9



Để cài đặt thuật toán được hiệu quả cần có một cơ chế tổ chức dữ liệu hợp lý. Chúng ta cần lưu trữ luồng f trên các cung, tìm đường tăng luồng P trên G_f và cộng luồng f_P vào luồng f hiện có. Việc tìm đường tăng luồng P trên G_f sẽ được thực hiện bằng một thuật toán tìm kiếm trên đồ thị còn việc tăng luồng dọc trên đường P đòi hỏi phải tăng giá trị luồng trên các cung

dọc trên đường đi đồng thời giảm giá trị luồng trên các cung đối. Vậy cấu trúc dữ liệu cần tổ chức để tạo điều kiện thuận lợi cho thuật toán tìm đường tăng luồng cũng như dễ dàng chỉ ra cung đối của một cung cho trước.

Đồ thị được biểu diễn bởi danh sách liên thuộc. Tất cả m cung của mạng được chứa trong danh sách $e[1 \dots m]$. Ngoài ra ta thêm m cung đối của chúng với sức chứa 0. Các cung đối này được lưu trữ trong danh sách $e[-m \dots -1]$, cung đối của cung $e[i]$ là cung $e[-i]$, cung $e[0]$ được sử dụng với vai trò phần tử cầm canh và không được tính đến.

Mỗi phần tử của danh sách e là một bản ghi gồm 4 trường (x, y, c, f) trong đó x, y là đỉnh đầu và đỉnh cuối của cung, c là sức chứa và f là luồng trên cung. Danh sách liên thuộc được xây dựng bởi hai mảng $head[1 \dots n]$ và $link[-m \dots m]$, trong đó:

- $head[u]$ là chỉ số cung đầu tiên trong danh sách liên thuộc các cung đi ra khỏi u, trường hợp u không có cung đi ra, $head[u]$ được gán bằng 0.
- $link[i]$ là chỉ số cung kế tiếp cung $e[i]$ trong cùng danh sách liên thuộc các cung đi ra khỏi một đỉnh. Trường hợp $e[i]$ là cung cuối cùng của một danh sách liên thuộc, $link[i]$ được gán bằng 0

Việc duyệt các cung đi ra khỏi đỉnh u sẽ được thực hiện theo cách sau:

```
i := head[u];  //i là chỉ số cung đầu tiên trong danh sách liên thuộc các cung ra khỏi u
while i ≠ 0 do //Chừng nào chưa duyệt qua cung cuối danh sách liên thuộc
  begin
    «Xử lý cung e[i]»;
    i := link[i];  //Nhảy sang xét cung kế tiếp trong danh sách liên thuộc
  end;
```

Tại mỗi bước, ta dùng thuật toán BFS để tìm đường đi từ s tới t trên G_f, mỗi đỉnh v trên đường đi được lưu vết $trace[v]$ là chỉ số cung đi vào v trên đường đi P tìm được. Dựa vào vết này, ta sẽ liệt kê được tất cả các cung trên đường đi, tăng luồng trên các cung này lên Δ_P đồng thời giảm luồng trên các cung đối đi Δ_P.

Edmonds và Karp (Edmonds & Karp, 1972) đã đề xuất mô hình cài đặt thuật toán Ford-Fulkerson trong đó thuật toán BFS được sử dụng để tìm đường tăng luồng nên người ta còn gọi thuật toán Ford-Fulkerson với kỹ thuật sử dụng BFS tìm đường tăng luồng là thuật toán Edmonds-Karp.

💻 EDMONDSKARP.PAS ✓ Thuật toán Edmonds-Karp

```
{$MODE OBJFPC}
program MaximumFlow;
const
  maxN = 1000;
  maxM = 100000;
  maxC = 10000;
type
  TEdge = record  //Cấu trúc một cung
    x, y: Integer;  //Hai đỉnh đầu mút
```

```
    c, f: Integer;  //Sức chứa và luồng
  end;
  TQueue = record  //Hàng đợi dùng cho BFS
    items: array[1..maxN] of Integer;
    front, rear: Integer;
  end;
var
  e: array[-maxM..maxM] of TEdge;  //Danh sách các cung
  link: array[-maxM..maxM] of Integer;  //Móc nối trong danh sách liên thuộc
  head: array[1..maxN] of Integer;  //head[u]: Chỉ số cung đầu tiên trong danh sách liên thuộc các cung ra khỏi u
  trace: array[1..maxN] of Integer;  //Vết đường đi
  n, m, s, t: Integer;
  FlowValue: Integer;
  Queue: TQueue;

procedure Enter;  //Nhập dữ liệu
var
  i: Integer;
  u, v, capacity: Integer;
begin
  ReadLn(n, m, s, t);
  FillChar(head[1], n * SizeOf(head[1]), 0);
  for i := 1 to m do
    begin
      ReadLn(u, v, capacity);
      with e[i] do  //Thêm cung e[i] = (u, v) vào danh sách liên thuộc của u
        begin
          x := u; y := v; c := capacity;
          link[i] := head[u]; head[u] := i;
        end;
      with e[-i] do  //Thêm cung e[-i] = (v, u) vào danh sách liên thuộc của v
        begin
          x := v; y := u; c := 0;
          link[-i] := head[v]; head[v] := -i;
        end;
    end;
end;

procedure InitZeroFlow;  //Khởi tạo luồng 0
var
  i: Integer;
begin
  for i := -m to m do e[i].f := 0;
  FlowValue := 0;
end;

function FindPath: Boolean;  //Tìm đường tăng luồng bằng BFS
var
  u, v: Integer;
  i: Integer;
begin
  FillChar(trace[1], n * SizeOf(trace[1]), 0);
  trace[s] := 1;  //trace[s] ≠ 0: đỉnh đã thăm, có thể dùng bất cứ hằng số nào khác 0
  with Queue do
    begin
      items[1] := s; front := 1; rear := 1;  //Hàng đợi chỉ gồm đỉnh s
```

```
      repeat
        u := items[front];  Inc(front);  //Lấy một đỉnh u khỏi hàng đợi
        i := head[u];
        while i <> 0 do //Duyệt danh sách liên thuộc của u
          begin
            v := e[i].y; //nút e[i] chứa một cung đi từ u tới v
            if (trace[v] = 0) and (e[i].f < e[i].c) then //v chưa thăm và e[i] là cung thặng dư
              begin
                trace[v] := i; //Lưu vết
                if v = t then Exit(True); //Tìm thấy đường tăng luồng, thoát
                Inc(rear); items[rear] := v; //Đẩy v vào hàng đợi
              end;
            i := link[i]; //Nhảy sang nút kế tiếp trong danh sách liên thuộc
          end;
      until front > rear;
      Result := False; //Không tìm thấy đường tăng luồng
    end;
end;

procedure AugmentFlow; //Tăng luồng dọc đường một tăng luồng
var
  Delta: Integer;
  v, i: Integer;
begin
  //Trước hết xác định Delta bằng dư lượng nhỏ nhất của các cung trên đường tăng luồng
  v := t; //Bắt đầu từ t
  Delta := maxC;
  repeat
    i := trace[v]; // e[i] là một cung trên đường tăng luồng với sức chứa e[i].c - e[i].f
    if e[i].c - e[i].f < Delta then
      Delta := e[i].c - e[i].f;
    v := e[i].x; //Đi dần về s
  until v = s;
  //Tăng luồng thêm Delta
  v := t; //Bắt đầu từ t
  repeat
    i := trace[v]; // e[i] là một cung trên đường tăng luồng
    Inc(e[i].f, Delta); //Tăng luồng trên e[i] lên Delta
    Dec(e[-i].f, Delta); //Giảm luồng trên cung đối tương ứng đi Delta
    v := e[i].x; //Đi dần về s
  until v = s;
  Inc(FlowValue, Delta); //Giá trị luồng f được tăng lên Delta
end;

procedure PrintResult; //In kết quả
var
  i: Integer;
begin
  WriteLn('Maximum flow: ');
  for i := 1 to m do
    with e[i] do
      if f > 0 then //Chỉ cần in ra các cung có luồng > 0
        WriteLn('e[', i, '] = (', x, ', ', y, '): c = ', c, ', f = ', f);
  WriteLn('Value of flow: ', FlowValue);
end;
```

```
begin
  Enter;  //Nhập dữ liệu
  InitZeroFlow;  //Khởi tạo luồng 0
  while FindPath do //Thuật toán Ford-Fulkerson
    AugmentFlow;
  PrintResult;  //In kết quả
end.
```

9.2.4. Tính đúng của thuật toán

Trước hết dễ thấy rằng thuật toán Ford-Fulkerson trả về một luồng, tức là kết quả mà thuật toán trả về thỏa mãn các tính chất của luồng. Việc chứng minh luồng đó là cực đại đã xây dựng một định lý quan trọng về mối quan hệ giữa luồng cực đại và lát cắt hẹp nhất.

Ta gọi một lát cắt (X,Y) là một cách phân hoạch tập đỉnh V làm hai tập rời nhau: $X \cap Y = \emptyset$ và $X \cup Y = V$. Lát cắt có $s \in X$ và $t \in Y$ được gọi là một lát cắt $s-t$.

Lưu lượng từ X sang Y $(c(X,Y))$ và luồng từ X sang Y $(f(X,Y))$ được gọi là lưu lượng và luồng thông qua lát cắt.

Bổ đề 9-9

Với f là một luồng trên mạng $G = (V,E,c,s,t)$. Khi đó luồng thông qua một lát cắt $s-t$ bất kỳ bằng $|f|$.

Chứng minh

Với $V = X \cup Y$ là một lát cắt $s-t$ bất kỳ, theo Định lý 9-3

$$f(X,Y) = f(X,V) - f(X,V-Y) = f(X,V) - f(X,X) = f(X,V)$$

Cũng theo định lý này ta có:

$$f(X,V) = f(s,V) + \underbrace{f(X-\{s\},V)}_{0} = f(s,V) = |f|$$

Bổ đề 9-10

Với f là một luồng trên mạng $G = (V,E,c,s,t)$. Khi đó luồng thông qua một lát cắt $s-t$ bất kỳ không vượt quá lưu lượng của lát cắt đó.

Chứng minh

Với $V = X \cup Y$ là một lát cắt $s-t$ bất kỳ ta có

$$f(X,Y) = \sum_{e \in \{X \to Y\}} f(e) \le \sum_{e \in \{X \to Y\}} c(e) = c(X,Y)$$

Định lý 9-11 (mối quan hệ giữa luồng cực đại, đường tăng luồng và lát cắt hẹp nhất)

Nếu f là một luồng trên mạng $G = (V,E,c,s,t)$, khi đó ba mệnh đề sau là tương đương:

a) f là luồng cực đại trên mạng G.

b) Mạng thặng dư G_f không có đường tăng luồng.

c) Tồn tại $V = X \cup Y$ là một lát cắt $s - t$ để $f(X,Y) = c(X,Y)$

Chứng minh

"a⇒b"

Giả sử phản chứng rằng mạng thặng dư G_f có đường tăng luồng P thì $f + f_P$ cũng là một luồng trên G với giá trị luồng lớn hơn f, trái giả thiết f là luồng cực đại trên mạng.

"b⇒c"

Nếu G_f không tồn tại đường tăng luồng thì ta đặt X là tập các đỉnh đến được từ s bằng một đường thặng dư và Y là tập các đỉnh còn lại:

$$X = \{v: \exists \text{ đường thặng dư } s \rightsquigarrow v\}; Y = V - X$$

Rõ ràng $X \cap Y = \emptyset$, $X \cup Y = V$ và $s \in X$, $t \in Y$ (t không thể đến được từ s bởi một đường thặng dư bởi nếu không thì đường đi đó sẽ là một đường tăng luồng).

Các cung $e \in \{X \to Y\}$ chắc chắn phải là cung bão hòa, bởi nếu có cung thặng dư $e = (u,v) \in \{X \to Y\}$ thì từ s sẽ tới được v bằng một đường thặng dư. Tức là $v \in X$, trái với cách xây dựng lát cắt. Từ $f(e) = c(e)$ với $\forall e \in \{X \to Y\}$, ta có

$$f(X,Y) = \sum_{e \in \{X \to Y\}} f(e) = \sum_{e \in \{X \to Y\}} c(e) = c(X,Y)$$

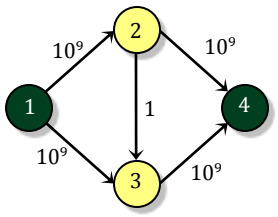
"c⇒a"

Bổ đề 9-9 và Bổ đề 9-10 cho thấy giá trị của một luồng trên mạng không thể vượt quá lưu lượng của một lát cắt $s - t$ bất kỳ. Nếu tồn tại một lát cắt $s - t$ mà luồng thông qua lát cắt đúng bằng lưu lượng thì luồng đó chắc chắn phải là luồng cực đại.

Lát cắt $s - t$ có lưu lượng nhỏ nhất (bằng giá trị luồng cực đại trên mạng) gọi là *Lát cắt $s - t$ hẹp nhất* của mạng G.

9.2.5. Tính dừng của thuật toán

Thuật toán Ford-Fulkerson có thời gian thực hiện phụ thuộc vào thuật toán tìm đường tăng luồng tại mỗi bước. Có thể chỉ ra được ví dụ mà nếu dùng DFS để tìm đường tăng luồng thì thời gian thực hiện giải thuật không bị chặn bởi một hàm đa thức của số đỉnh và số cạnh. Thêm nữa, nếu sức chứa của các cung là số thực, người ta còn chỉ ra được ví dụ mà với thuật toán tìm đường tăng luồng không tốt, giá trị luồng sau mỗi bước vẫn tăng nhưng **không bao giờ đạt luồng cực đại**. Tức là nếu có thể cài đặt chương trình tính toán số thực với độ chính xác tuyệt đối, thuật toán sẽ chạy mãi không dừng.



Hình 9-4. Mạng với 4 đỉnh (1 phát, 4 thu), thuật toán Ford-Fulkerson có thể mất 2 tỉ lần tìm đường tăng luồng nếu luân phiên chọn hai đường ⟨1,2,3,4⟩ và ⟨1,3,2,4⟩ làm đường tăng luồng, mỗi lần tăng giá trị luồng lên 1 đơn vị.

Chính vì vậy nên trong một số tài liệu người ta gọi là "phương pháp Ford-Fulkerson" để chỉ một cách tiếp cận chung, còn từ "thuật toán" được dùng để chỉ một cách cài đặt phương pháp Ford-Fulkerson trên một cấu trúc dữ liệu cụ thể, với một thuật toán tìm đường tăng luồng cụ thể. Ví dụ phương pháp Ford-Fulkerson cài đặt với thuật toán tìm đường tăng luồng bằng BFS như trên được gọi là thuật toán Edmonds-Karp. Tính dừng của thuật toán Edmonds-Karp sẽ được chỉ ra khi chúng ta đánh giá thời gian thực hiện giải thuật.

Xét G_f là mạng thặng dư của một mạng G ứng với luồng f nào đó, ta gán trọng số 1 cho các cung thặng dư của G_f và gán trọng số $+\infty$ cho các cung bão hòa của G_f. Dễ thấy rằng thuật toán tìm đường tăng luồng bằng BFS sẽ trả về một đường đi ngắn nhất từ s tới t tương ứng với hàm trọng số đã cho. Ký hiệu $\delta_f(u,v)$ là độ dài đường đi ngắn nhất từ u tới v (khoảng cách từ u tới v) trên mạng thặng dư.

Bổ đề 9-12

Nếu ta khởi tạo luồng 0 và thực hiện thuật toán Edmonds-Karp trên mạng $G = (V, E)$ có đỉnh phát s và đỉnh thu t. Khi đó với mọi đỉnh $v \in V$, khoảng cách từ s tới v trên mạng thặng dư không giảm sau mỗi bước tăng luồng.

Chứng minh

Khi $v = s$, rõ ràng khoảng cách từ s tới chính nó luôn bằng 0 từ khi bắt đầu tới khi kết thúc thuật toán. Ta chỉ cần chứng minh bổ đề đúng với những đỉnh $v \neq s$.

Giả sử phản chứng rằng tồn tại một đỉnh $v \in V - \{s\}$ mà khi thuật toán Edmonds-Karp tăng luồng f lên thành f' sẽ làm cho $\delta_{f'}(s,v)$ nhỏ hơn $\delta_f(s,v)$. Nếu có nhiều đỉnh v như vậy ta chọn đỉnh v có $\delta_{f'}(s,v)$ nhỏ nhất. Gọi $P = s \rightsquigarrow u \rightarrow v$ là đường đi ngắn nhất từ s tới v trên $G_{f'}$, ta có (u,v) là cung thặng dư trên $G_{f'}$ và

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

Bởi cách chọn đỉnh v, độ dài đường đi ngắn nhất từ s tới u không thể bị giảm đi sau phép tăng luồng, tức là

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

Ta chứng minh rằng (u, v) phải là cung bão hòa trên G_f. Thật vậy, nếu (u, v) là cung thặng dư (có trọng số 1) trên G_f thì:

$$\begin{aligned}\delta_f(s, v) &\leq \delta_f(s, u) + 1 \text{ (bất đẳng thức tam giác)} \\ &\leq \delta_{f'}(s, u) + 1 \text{ (khoảng cách từ } s \text{ tới } u \text{ không giảm)} \\ &= \delta_{f'}(s, v)\end{aligned}$$

Trái với giả thiết rằng khoảng cách từ s tới v phải giảm đi sau phép tăng luồng.

Làm thế nào để (u, v) là cung bão hòa trên G_f nhưng lại là cung thặng dư trên $G_{f'}$? Câu trả lời duy nhất là do phép tăng luồng từ f lên f' làm giảm luồng trên cung (u, v), tức là cung đối (v, u) phải là một cung trên đường tăng luồng tìm được. Vì đường tăng luồng tại mỗi bước luôn là đường đi ngắn nhất nên (v, u) phải là cung cuối cùng trên đường đi ngắn nhất từ s tới u của G_f. Từ đó suy ra:

$$\begin{aligned}\delta_f(s, v) &= \delta_f(s, u) - 1 \\ &\leq \delta_{f'}(s, u) - 1 \text{ (khoảng cách từ } s \text{ tới } u \text{ không giảm)} \\ &= \delta_{f'}(s, v) - 2 \text{ (theo cách chọn } u \text{ và } v)\end{aligned}$$

Mâu thuẫn với giả thuyết khoảng cách từ s tới v phải giảm đi sau khi tăng luồng. Ta có điều phải chứng minh: Với $\forall v \in V$, khoảng cách từ s tới v trên mạng thặng dư không giảm sau mỗi bước tăng luồng.

Bổ đề 9-13

Nếu thuật toán Edmonds-Karp thực hiện trên mạng $G = (V, E, c, s, t)$ với luồng khởi tạo là luồng 0 thì số lượt tăng luồng được sử dụng trong thuật toán là $O(|V||E|)$.

Chứng minh

Ta chia quá trình thực hiện thuật toán Edmonds-Karp thành các pha. Mỗi pha tìm một đường tăng luồng P và tăng luồng thêm một giá trị thặng dư Δ_P. Giá trị thặng dư này theo định nghĩa sẽ phải bằng sức chứa của một cung thặng dư e nào đó trên đường P:

$$\exists e \in P: \Delta_P = c(e) - f(e)$$

Khi tăng luồng dọc trên được P thì cung e sẽ trở thành bão hòa. Những cung thặng dư trở nên bão hòa sau khi tăng luồng gọi là *cung tới hạn* (*critical edge*) tại mỗi pha. Mỗi pha có ít nhất một cung tới hạn.

Ta đánh giá xem mỗi cung của mạng có thể trở thành cung tới hạn bao nhiêu lần. Với một cung $e = (u, v)$, ta xét pha A đầu tiên làm e trở thành cung tới hạn và f_A là luồng khi bắt đầu pha A. Do e nằm trên đường tăng luồng ngắn nhất trên G_{f_A} nên khi pha này bắt đầu:

$$\delta_{f_A}(s, u) + 1 = \delta_{f_A}(s, v)$$

Pha A sau khi tăng luồng sẽ làm cung e sẽ trở nên bão hòa.

Để e có thể trở thành cung tới hạn một lần nữa thì tiếp theo pha A phải có một pha B giảm luồng trên cung e để biến e thành cung thặng dư, tức là cung $-e = (v,u)$ phải là một cung trên đường tăng luồng của pha B. Gọi f_B là luồng khi pha B bắt đầu, cũng vì tính chất của đường đi ngắn nhất, ta có

$$\delta_{f_B}(s,v) + 1 = \delta_{f_B}(s,u)$$

Bổ đề 9-12 đã chứng minh rằng khoảng cách từ s tới v trên mạng thặng dư không giảm đi sau mỗi pha, nên $\delta_{f_B}(s,v) \geq \delta_{f_A}(s,v)$. Suy ra:

$$\begin{aligned}\delta_{f_B}(s,u) &= \delta_{f_B}(s,v) + 1 \\ &\geq \delta_{f_A}(s,v) + 1 \\ &= \delta_{f_A}(s,u) + 2\end{aligned}$$

Như vậy nếu một cung (u,v) là cung tới hạn trong k pha thì khi pha thứ k bắt đầu, khoảng cách từ s tới u trên mạng thặng dư đã tăng lên ít nhất $2(k-1)$ đơn vị so với thời điểm trước pha thứ nhất. Khoảng cách $\delta_f(s,u)$ ban đầu là số không âm và chừng nào còn đường thặng dư đi từ s tới u, khoảng cách $\delta_f(s,u)$ không thể vượt quá $|V|-1$. Điều đó cho thấy $k \leq \frac{|V|+1}{2} = O(|V|)$.

Tổng hợp lại, ta có:

- Mạng có tất cả $|E|$ cung.
- Mỗi pha có ít nhất một cung tới hạn
- Một cung có thể trở thành tới hạn trong $O(|V|)$ pha

Vậy tổng số pha được thực hiện trong thuật toán Edmonds-Karp là một đại lượng $O(|V||E|)$

Định lý 9-14

Có thể cài đặt thuật toán Edmonds-Karp để tìm luồng cực đại trên mạng $G = (V,E,c,s,t)$ trong thời gian $O(|V||E|^2)$.

Chứng minh

Bổ đề 9-15 đã chứng minh rằng thuật toán Edmonds-Karp cần thực hiện $O(|V||E|)$ lượt tăng luồng. Tại mỗi lượt thuật toán tìm đường tăng luồng bằng BFS và tăng luồng dọc đường này có thời gian thực hiện $O(|E|)$. Suy ra thời gian thực hiện giải thuật Edmonds-Karp là $O(|V||E|^2)$.

Nếu khả năng thông qua trên các cung của mạng là số nguyên thì còn có một cách đánh giá khác dựa trên giá trị luồng cực đại, nếu ta khởi tạo luồng 0 thì sau mỗi lượt tăng luồng, giá trị luồng được tăng lên ít nhất 1 đơn vị. Suy ra thời gian thực hiện giải thuật khi đó là $O(|f||E|)$ với $|f|$ là giá trị luồng cực đại.

9.3. Thuật toán đẩy tiền luồng

Thuật toán Ford-Fulkerson không những là một cách tiếp cận thông minh mà việc chứng minh tính đúng đắn của nó cho ta nhiều kết quả thú vị về mối liên hệ giữa luồng cực đại và lát cắt hẹp nhất. Tuy vậy với những đồ thị kích thước rất lớn thì tốc độ của chương trình tương đối chậm.

Trong phần này ta sẽ trình bày một lớp các thuật toán nhanh nhất cho tới nay để giải bài toán luồng cực đại, tên chung của các thuật toán này là thuật toán *đẩy tiền luồng* (*preflow-push*).

Hãy hình dung mạng như một hệ thống đường ống dẫn nước từ với điểm phát s tới điểm thu t, các cung là các đường ống, sức chứa là lưu lượng đường ống có thể tải. Nước chảy theo nguyên tắc từ chỗ cao về chỗ thấp. Với một lượng nước lớn phát ra từ s tới một đỉnh v, nếu có cách chuyển lượng nước đó sang địa điểm khác thì không có vấn đề gì, nếu không thì có hiện tượng "tràn" xảy ra tại v, ta "dâng cao" điểm v để lượng nước đó đổ sang điểm khác (có thể đổ ngược về s). Cứ tiếp tục quá trình như vậy cho tới khi không còn hiện tượng tràn ở bất cứ điểm nào. Cách tiếp cận này hoàn toàn khác với thuật toán Ford-Fulkerson: thuật toán Ford-Fulkerson cố gắng tìm một dòng chảy phụ từ s tới t và thêm dòng chảy này vào luồng hiện có đến khi không còn dòng chảy phụ nữa.

9.3.1. Tiền luồng

Cho một mạng $G = (V, E, c, s, t)$. Một *tiền luồng (preflow)* trên G là một hàm:

$$f: E \longrightarrow \mathbb{R}$$
$$e \longmapsto f(e)$$

gán cho mỗi cung $e \in E$ một số thực $f(e)$ thỏa mãn ba ràng buộc:

- Ràng buộc về sức chứa (capacity constraint): tiền luồng trên mỗi cung không được vượt quá sức chứa của cung đó: $\forall e \in E: f(e) \leq c(e)$.
- Ràng buộc về tính đối xứng lệch (skew symmetry): Với $\forall e \in E$, tiền luồng trên cung e và cung đối – e có cùng giá trị tuyệt đối nhưng trái dấu nhau: $f(e) = -f(-e)$.
- Ràng buộc về tính dư: Với mọi đỉnh không phải đỉnh phát, tổng tiền luồng trên các cung đi vào đỉnh đó là số không âm: $\forall v \in V - \{s\}$: $f(V, \{v\}) = \sum_{e \in \{V \to \{v\}\}} f(e) \geq 0$.

Với $\forall v \in V$, ta gọi lượng tràn tại v, ký hiệu $excess[v]$, là tổng tiền luồng trên các cung đi vào đỉnh v:

$$excess[v] = f(V, \{v\}) = \sum_{e \in \{V \to \{v\}\}} f(e)$$

Đỉnh $v \in V - \{s, t\}$ gọi là *đỉnh tràn (overflowing vertex)* nếu $excess[v] > 0$. Khái niệm đỉnh tràn chỉ có nghĩa với các đỉnh không phải đỉnh phát cũng không phải đỉnh thu.

```
function Overflow(v∈V): Boolean;
begin
```

```
  Result := (v ≠ s) and (v ≠ t) and (excess[u] > 0);
end;
```

Định nghĩa về tiền luồng tương tự như định nghĩa luồng, chỉ khác nhau ở ràng buộc thứ ba. Vì vậy chúng ta cũng có khái niệm mạng thặng dư, cung thặng dư, đường thặng dư… ứng với tiền luồng tương tự như đối với luồng.

9.3.2. Khởi tạo

Cho f là một tiền luồng trên mạng $G = (V, E, c, s, t)$. Ta gọi $h: V \to \mathbb{N}$ là một hàm độ cao ứng với f nếu h gán cho mỗi đỉnh $v \in V$ một số tự nhiên $h(v)$ thỏa mãn ba điều kiện:

- $h(s) = |V|$.
- $h(t) = 0$.
- $h(u) \leq h(v) + 1$ với mọi cung thặng dư (u, v).

Những ràng buộc này gọi là **ràng buộc độ cao**.

Hàm độ cao h khi cài đặt sẽ được xác định bởi tập các giá trị $\{h[v]\}_{v \in V}$ nên tùy theo từng trường hợp, ta có thể sử dụng ký hiệu $h(v)$ (nếu muốn nói tới giá trị hàm) hoặc $h[v]$ (nếu muốn nói tới một biến số).

Thao tác khởi tạo $Init$ chịu trách nhiệm khởi tạo một tiền luồng và một hàm độ cao tương ứng. Một cách khởi tạo là đặt tiền luồng trên mỗi cung e đi ra khỏi s đúng bằng sức chứa $c(e)$ của cung đó (dĩ nhiên sẽ phải đặt cả tiền luồng trên cung đối – e bằng – $c(e)$ để thỏa mãn tính đối xứng lệch), còn tiền luồng trên các cung khác bằng 0. Khi đó tất cả các cung đi ra khỏi s là bão hòa.

$$f(e) = \begin{cases} c(e), \text{nếu } e \in E^+(s) \\ -c(e), \text{nếu } -e \in E^+(s) \\ 0, \text{trường hợp khác} \end{cases}$$

Ta khởi tạo hàm độ cao $h: V \to \mathbb{N}$ như sau:

$$h(v) = \begin{cases} |V|, \text{nếu } v = s \\ 0, \text{nếu } v = t \\ 1, \text{nếu } v \neq \{s, t\} \end{cases}$$

Rõ ràng mọi cung thặng dư (u, v) không thể là cung đi ra khỏi s $(u \neq s)$ nên ta có $h(u) \leq 1 \leq h(v) + 1$. Hàm độ cao trên là thích ứng với tiền luồng f.

Việc cuối cùng là khởi tạo các giá trị $excess[.]$ ứng với tiền luồng f.

```
procedure Init;
begin
  //Khởi tạo tiền luồng
  for ∀e∈E do f[e] := 0;
  for ∀v∈V do excess[v] := 0;
  for ∀e=(s,v)∈E+(s) do
    begin
      f[e] := c(e); f[-e] := -c(e);
      excess[v] := excess[v] + c(e);
    end;
  //Khởi tạo hàm độ cao
  for ∀v∈V do h[v] := 1;
  h[s] := |V|; h[t] := 0;
end;
```

9.3.3. Phép đẩy luồng

Phép đẩy luồng $Push(e)$ có thể thực hiện trên cung $e = (u, v)$ nếu các điều kiện sau được thỏa mãn:

- u là đỉnh tràn: $u \in V - \{s, t\}$ và $excess[u] > 0$
- e là cung thặng dư trên G_f: $c_f(e) = c(e) - f(e) > 0$
- u cao hơn v: $h(u) > h(v)$

Ràng buộc $h(u) > h(v)$ kết hợp với ràng buộc độ cao: $h(u) \leq h(v) + 1$ có thể viết thành $h(u) = h(v) + 1$.

Phép $Push\big(e = (u, v)\big)$ sẽ tính lượng luồng tối đa có thể thêm vào theo cung e: $\Delta = \min\{excess[u], c_f(e)\}$, thêm lượng luồng này vào cung e và bớt một lượng luồng Δ từ v về u theo cung $-e$ để giữ tính đối xứng lệch của tiền luồng. Việc cuối cùng là cập nhật lại $excess[u]$ và $excess[v]$ theo tiền luồng mới. Bản chất của phép $Push\big(e = (u, v)\big)$ là chuyển một lượng luồng tràn Δ từ đỉnh u sang đỉnh v. Dễ thấy rằng các tính chất của tiền luồng vẫn được duy trì sau phép $Push$:

```
procedure Push(e = (u,v));
begin
  Δ := min(excess[u], cf(u, v)); //Tính lượng luồng tối đa có thể đẩy
  f[e] := f[e] + Δ; f[-e] := f[-e] - Δ; //Đẩy luồng
  excess[u] := excess[u] - Δ; excess[v] := excess[v] + Δ; //Cập nhật mức tràn
end;
```

Phép $Push$ bảo tồn tính chất của hàm độ cao. Thật vậy, khi thao tác $Push\big(e = (u, v)\big)$ được thực hiện, nó chỉ có thể sinh ra thêm một cung thặng dư $-e = (v, u)$ mà thôi. Phép $Push$ không làm thay đổi các độ cao, tức là trước khi $Push$, $h[u] > h[v]$ thì sau khi $Push$, $h[v]$ vẫn nhỏ hơn $h[u]$, tức là ràng buộc độ cao $h[v] \leq h[u] + 1$ vẫn được duy trì trên cung thặng dư $-e = (v, u)$.

Phép $Push\big(e=(u,v)\big)$ đẩy một lượng luồng $\Delta = \min\{excess[u], c_f(e)\}$ tràn từ u sang v. Nếu Δ đúng bằng $c_f(e) = c(e) - f(e)$, có nghĩa là khi phép $Push$ tăng $f(e)$ lên Δ thì cung e sẽ bão hòa và không còn là cung thặng dư trên G_f nữa, ta gọi phép đẩy luồng này là *đẩy bão hòa (saturating push)*, ngược lại phép đẩy luồng đó gọi là *đẩy không bão hòa (non-saturating push)*, sau phép đẩy không bão hòa thì $excess[u] = 0$, tức là u không còn là đỉnh tràn nữa.

9.3.4. Phép nâng

Phép nâng $Lift(u)$ thực hiện trên đỉnh u nếu các điều kiện sau được thỏa mãn:

- u là đỉnh tràn: $(u \neq s)$, $(u \neq t)$ và $excess[u] > 0$.
- u không chuyển được luồng xuống nơi nào thấp hơn: Với mọi cung thặng dư $e = (u,v) \in E_f$: $h(u) \leq h(v)$.

Khi đó phép $Lift(u)$ nâng đỉnh u lên bằng cách đặt $h[u]$ bằng độ cao thấp nhất của một đỉnh v nó có thể chuyển tải sang cộng thêm 1:

$$h[u] := \min\{h[v] : \exists (u,v) \in E_f\} + 1$$

```
procedure Lift(u∈V);
begin
  minH := +∞;
  for ∀v:(u,v)∈Ef do
    if h[v] < minH then minH := h[v];
  h[u] := minH + 1;
end;
```

Nếu u là đỉnh tràn thì ít nhất phải có một cung thặng dư đi ra khỏi u, điều này đảm bảo cho phép lấy $\min\{h[v] : (u,v) \in E_f\}$ được thực hiện trên một tập khác rỗng. Thật vậy, do u là đỉnh tràn, ta có $excess[u] = \sum_{e \in \{V \to \{u\}\}} f(e) > 0$ tức là ít nhất có một cung $e \in \{V \to \{u\}\}$ để $f(e) > 0$. Cung đối – e chắc chắn là một cung thặng dư đi ra khỏi u bởi:

$$c_f(-e) = c(-e) - f(-e) = c(-e) + f(e) > 0$$

Phép $Lift$ không động chạm gì đến tiền luồng f. Ngoài ra phép $Lift$ **chỉ tăng độ cao của một đỉnh** và bảo tồn ràng buộc độ cao: Với một cung thặng dư (v,u) đi vào u, ràng buộc độ cao $h(v) \leq h(u) + 1$ không bị vi phạm nếu ta nâng độ cao $h(u)$ của đỉnh u. Mặt khác, với một cung thặng dư (u,v) đi ra khỏi u thì việc đặt $h[u] := \min\{h[v] : \exists (u,v) \in E_f\} + 1$ cũng đảm bảo rằng $h(u) \leq h(v) + 1$.

9.3.5. Mô hình chung và thuật toán FIFO Preflow-Push

❑ Mô hình chung

Thuật toán đẩy tiền luồng có mô hình cài đặt chung khá đơn giản: Khởi tạo tiền luồng f và hàm độ cao, sau đó nếu thấy phép nâng ($Lift$) hay đẩy luồng ($Push$) nào thực hiện được thì

thực hiện ngay… Cho tới khi không còn phép nâng hay đẩy nào có thể thực hiện được nữa thì tiền luồng f sẽ trở thành luồng cực đại trên mạng.

Chính vì thứ tự các phép $Push$ và $Lift$ được thực hiện không ảnh hưởng tới tính đúng đắng của thuật toán nên người ta đã đề xuất rất nhiều cơ chế chọn thứ tự thực hiện nhằm giảm thời gian thực hiện giải thuật.

Bổ đề 9-15

Cho mạng $G = (V, E, c, s, t)$ có tiền luồng f và hàm độ cao h. Với một đỉnh tràn u, luôn có thể thực hiện được thao tác $Push(e)$ trên một cung e đi ra khỏi u hoặc thực hiện được thao tác $Lift(u)$

Chứng minh

Nếu thao tác $Push$ không thể áp dụng được cho cung thặng dư nào đi ra khỏi u tức là với mọi cung thặng dư $(u, v) \in E_f$, $h(u)$ không cao hơn $h(v)$, điều đó chính là điều kiện hợp lệ để thực hiện thao tác $Lift(u)$.

❑ Thuật toán FIFO Preflow-Push

Định lý 9-17 là cơ sở cho thuật toán FIFO Preflow-Push. Thuật toán được Goldberg đề xuất (Goldberg, Efficient graph algorithms for sequential and parallel computers, 1987) dựa trên cơ chế xử lý đỉnh tràn lấy ra từ một hàng đợi.

Tại thao tác khởi tạo, các đỉnh tràn sẽ được lưu trữ trong một hàng đợi $Queue$ hỗ trợ hai thao tác: $PushToQueue(v)$ để đẩy một đỉnh tràn v vào hàng đợi và $PopFromQueue$ để lấy một đỉnh tràn khỏi hàng đợi. Thuật toán sẽ xử lý từng đỉnh tràn z lấy ra khỏi hàng đợi theo cách sau: Trước hết cố gắng đẩy luồng trên các cung thặng dư đi ra khỏi z bằng phép $Push$. Nếu đẩy được hết lượng tràn ($excess[z] = 0$) thì xong, nếu không ta dâng cao đỉnh z bằng phép $Lift(z)$ và đẩy lại z vào hàng đợi chờ xử lý sau. Thuật toán sẽ tiếp tục với đỉnh tràn tiếp theo trong hàng đợi và kết thúc khi hàng đợi rỗng, bởi khi mạng không còn đỉnh tràn thì không còn thao tác $Push$ hay $Lift$ nào có thể thực hiện được nữa.

Giả sử rằng chúng ta có một đỉnh tràn u và một cung $e = (u, v)$ không thể đẩy luồng được, tức là ít nhất một trong hai điều kiện sau đây được thỏa mãn:

- (u, v) là cung bão hòa $c(e) = f(e)$.
- u không cao hơn v: $h(u) \leq h(v)$.

Khi đó:

- Sau bất kỳ phép $Push$ nào, chúng ta vẫn không thể đẩy luồng được trên cung $e = (u, v)$. Thật vậy, nếu u không cao hơn v, phép $Push$ không làm thay đổi hàm độ cao nên sau phép $Push$ thì u vẫn không cao hơn v. Nếu u cao hơn v thì e phải là cung bão hòa, lệnh $Push$ duy nhất có thể biến nó thành cung thặng dư là lệnh $Push(-e)$ làm giảm $f(e)$. Nhưng lệnh $Push(-e)$ không thể thực hiện được vì cung – $e = (v, u)$ có v thấp hơn u.

- Sau bất kỳ phép $Lift$ nào ngoại trừ $Lift(u)$, chúng ta cũng không thể đẩy luồng được trên cung $e = (u, v)$. Bởi phép $Lift$ không làm thay đổi tiền luồng trên các cung, dư lượng của các cung được giữ nguyên. Như vậy nếu (u, v) đang bão hòa thì sau phép $Lift$ nó vẫn bão hòa và không thể đẩy luồng được. Nếu (u, v) là cung thặng dư thì u đang không cao hơn v, lệnh $Lift$ duy nhất có thể khiến u cao hơn v là lệnh $Lift(u)$.

Hai nhận định trên cho phép ta xây dựng một cấu trúc dữ liệu hiệu quả để cài đặt thuật toán:

Tương tự như chương trình cài đặt thuật toán Edmonds-Karp, ta sử dụng mảng $e[-m \dots m]$ chứa các cung, mảng $link[m \dots m]$ chứa móc nối trong danh sách liên thuộc và mảng $head[1 \dots n]$ chứa chỉ số cung đầu tiên của các danh sách liên thuộc. Ngoài ra thuật toán duy trì một mảng chỉ số $current[1 \dots n]$, ở đây $current[v]$ là chỉ số của một cung nào đó trong danh sách liên thuộc các cung đi ra khỏi v, ban đầu $current[v]$ được gán bằng $head[v]$ với mọi đỉnh $v \in V$.

```
type
  TEdge = record  //Cấu trúc một cung
    x, y: Integer;  //Hai đỉnh đầu mút
    c, f: Integer;  //Sức chứa và luồng
  end;
var
  e: array[-maxM..maxM]  of TEdge;  //Danh sách các cung
  link: array[-maxM..maxM]  of Integer;  //Móc nối trong danh sách liên thuộc
  head, current: array[1..maxN] of Integer;
```

Trên cấu trúc dữ liệu này, danh sách móc nối các nút chứa các cung đi ra khỏi z là:

$$e[i_1], e[i_2], e[i_3], \dots$$

Trong đó $i_1 = head[z]$, $i_2 = link[i_1]$, $i_3 = link[i_2]$,...

Thuật toán FIFO Preflow-Push sẽ xử lý lần lượt từng đỉnh tràn lấy ra khỏi hàng đợi. Với mỗi đỉnh tràn z lấy khỏi hàng đợi, cung $e\big[current[z]\big]$ là một cung đi ra khỏi z, giả sử cung đó là (z, v). Nếu phép đẩy luồng ($Push$) trên cung đó có thể thực hiện được thì thực hiện ngay, đồng thời đẩy v vào hàng đợi nếu v chưa có trong hàng đợi. Nếu phép đẩy luồng này làm z hết tràn thì chuyển sang xử lý đỉnh tràn kế tiếp trong hàng đợi, ngược lại nếu z vẫn còn là đỉnh tràn (tức là không thể đẩy luồng trên cung $e\big[current[z]\big]$ nữa), ta dịch chỉ số $current[z]$ sang cung kế tiếp trong danh sách liên thuộc ($current[z] := link\big[current[z]\big]$) để chuyển sang xét một cung khác…Khi dịch chỉ số $current[z]$ đến hết danh sách liên thuộc mà z vẫn tràn, đỉnh z sẽ được nâng lên bằng phép $Lift(z)$, chỉ số $current[x]$ được đặt trở lại bằng $head[z]$ để nó trỏ lại về đầu danh sách liên thuộc, đỉnh z sau đó được đẩy lại vào hàng đợi chờ xử lý sau…

Tính hợp lý của thuật toán nằm ở chỗ : khi đỉnh tràn z bắt đầu được xử lý, tất cả những cung đứng trước cung $e\big[current[z]\big]$ đều không thể đẩy luồng được. Tức là nếu muốn đẩy luồng ra khỏi z thì chỉ cần xét các cung từ $e\big[current[z]\big]$ trở đi là đủ, không cần duyệt từ đầu danh sách liên thuộc.

```
procedure FIFOPreflowPush;
begin
  Init; //Khởi tạo tiền luồng, độ cao, hàng đợi Queue chứa các đỉnh tràn
  while Queue ≠ ∅ do
    begin
      z := PopFromQueue; //Xử lý đỉnh tràn x lấy ra từ hàng đợi
      while current[z] <> 0 do //Cố gắng đẩy luồng khỏi z
        begin //Xét cung (z, v) chứa trong nút e[current[z]]
          v := e[current[z]].y;
          if «Có thể đẩy luồng trên cung (z, v)» then
            begin
              NeedQueue := (v ≠ s) and (v ≠ t) and (excess[v] = 0);
              Push(z, v); //Đẩy luồng
              if NeedQueue then //Sau phép đẩy, v đang không tràn trở thành tràn
                PushToQueue(v); //Đẩy v vào hàng đợi chờ xử lý
              if excess[z] = 0 then Break; //Sau phép đẩy mà z hết tràn thì dừng đẩy
            end;
          current[z] := link[current[z]]; //z chưa hết tràn, chuyển sang xét cung liên thuộc tiếp theo
        end;
      if excess[z] > 0 then //Duyệt hết danh sách liên thuộc mà x vẫn tràn
        begin
          Lift(z); //Dâng cao z
          current[z] := head[z]; //Đặt lại chỉ số current[z] về nút đầu danh sách liên thuộc
          PushToQueue(z); //Đẩy z vào hàng đợi chờ xử lý sau
        end;
    end;
end;
```

Từ nhận xét trên, có thể nhận thấy rằng những phép $Push$ và $Lift$ trong mô hình cài đặt đảm bảo được gọi tại những thời điểm mà những điều kiện cần để thực thi chúng được thỏa mãn.

9.3.6. Tính đúng của thuật toán

Sau mỗi bước của vòng lặp chính, hàng đợi $Queue$ luôn chứa danh sách các đỉnh tràn và thuật toán sẽ kết thúc khi không còn đỉnh tràn nào trên mạng. Với $\forall v \in V - \{s,t\}$, ta có:

$$f(\{v\},V) = -f(V,\{v\}) = -excess[v] = 0$$

Tức là với $\forall v \in V - \{s,t\}$ thì tổng luồng trên các cung đi ra khỏi v bằng 0, điều này chỉ ra rằng khi thuật toán kết thúc, tiền luồng chúng ta duy trì trên mạng trở thành một luồng.

Định lý 9-16

Cho f là một tiền luồng trên mạng $G = (V, E, c, s, t)$, nếu tồn tại một hàm độ cao $h: V \to \mathbb{N}$ ứng với f thì mạng thặng dư G_f không có đường tăng luồng.

Chứng minh

Nhắc lại về ràng buộc độ cao: $h(s) = |V|$, $h(t) = 0$ và với mọi cung thặng dư (u,v) thì $h(u) \leq h(v) + 1$. Giả sử phản chứng rằng có đường tăng luồng $\langle s = v_0, v_1, \dots, v_k = t\rangle$ trên mạng thặng dư G_f đi qua k cung thặng dư. Khi đó:

$$h(v_0) \leq h(v_1) + 1 \leq h(v_2) + 2 \leq \cdots \leq h(v_k) + k$$

hay

$$\underbrace{h(s)}_{|V|} \leq \underbrace{h(t)}_{0} + k$$

Ta có $|V| \leq k$, nhưng đường tăng luồng phải là đường đi đơn, tức là qua không quá $|V| - 1$ cạnh, vậy $k \leq |V| - 1$. Điều này mâu thuẫn, nghĩa là không thể tồn tại đường tăng luồng trên G_f.

Định lý 9-17 và Định lý 9-11 (mối quan hệ giữa luồng cực đại, đường tăng luồng và lát cắt hẹp nhất) chỉ ra rằng: thuật toán đẩy tiền luồng trả về một luồng và một hàm độ cao ứng với luồng đó nên luồng trả về chắc chắn là luồng cực đại.

9.3.7. Tính dừng của thuật toán

Tính dừng của thuật toán đẩy tiền luồng ở trên sẽ được suy ra khi chúng ta phân tích thời gian thực hiện giải thuật. Tương tự như thuật toán Ford-Fulkerson, chúng ta sẽ không phân tích thời gian thực hiện trên mô hình tổng quát mà chỉ phân tích thời gian thực hiện giải thuật FIFO Preflow-Push mà thôi.

Định lý 9-17

Cho f là một tiền luồng trên mạng $G = (V, E, c, s, t)$, khi đó với mọi đỉnh tràn u, tồn tại một đường thặng dư đi từ u tới s.

Chứng minh

Với một đỉnh tràn u bất kỳ, xét tập X là tập các đỉnh có thể đến được từ u bằng một đường thặng dư. Đặt $Y = V - X$ là tập những đỉnh nằm ngoài X. Trước hết ta chỉ ra rằng tiền luồng trên các cung thuộc $\{Y \to X\}$ không thể là số dương. Thật vậy nếu có $e \in \{Y \to X\}$ mà $f(e) > 0$ thì $-e \in \{X \to Y\}$ và $f(-e) < 0$. Suy ra có cung thặng dư $-e$ nối một đỉnh thuộc X với một đỉnh y nào đó thuộc Y. Theo cách xây dựng tập X, y sẽ phải là đỉnh thuộc X. Mâu thuẫn.

Tiền luồng trên các cung thuộc $\{Y \to X\}$ không thể là số dương thì $f(Y, X) \leq 0$. Ta xét tổng mức tràn của các đỉnh $\in X$:

$$excess(X) = f(V, X) = \underbrace{f(X, X)}_{0} + \underbrace{f(Y, X)}_{\leq 0} \leq 0$$

Lượng tràn tại mỗi đỉnh không phải đỉnh phát đều là số không âm, ngoài ra u là đỉnh tràn $\in X$ nên $excess[u] > 0$, điều này cho thấy chắc chắn đỉnh phát s phải thuộc X để $excess(X) \leq 0$. Nói cách khác từ u đến được s bằng một đường thặng dư.

Hệ quả

Cho mạng $G = (V, E, c, s, t)$. Giả sử chúng ta thực hiện thuật toán đẩy tiền luồng với hàm độ cao $h: V \to \mathbb{N}$ thì độ cao của các đỉnh trong quá trình thực hiện giải thuật không vượt quá $2|V| - 1$.

Chứng minh

Mạng phải có ít nhất một đỉnh phát và một đỉnh thu nên $|V| \geq 2$. Ban đầu, $h(s) = |V|$, $h(t) = 0$ và $h(v) = 1, \forall v \notin \{s, t\}$ nên độ cao của các đỉnh đều nhỏ hơn $2|V| - 1$.

Độ cao của s và t không bao giờ bị thay đổi và với mỗi đỉnh $u \in V - \{s, t\}$ thì chỉ phép $Lift(u)$ có thể làm tăng độ cao của đỉnh u. Trước và sau phép $Lift(u)$, u phải là đỉnh tràn. Áp dụng kết quả của Định lý 9-17, tồn tại đường đi đơn từ u tới s $\langle u = v_0, v_1, \ldots, v_k = s \rangle$ chỉ đi qua k cung thặng dư (v_0, v_1),(v_1, v_2),...,(v_{k-1}, v_k). Từ ràng buộc độ cao ta có:

$$h(u) = h(v_0) \leq h(v_1) + 1 \leq h(v_2) + 2 \leq \cdots \leq h(v_k) + k \leq |V| + k$$

Đường đi đơn thì không qua nhiều hơn $|V| - 1$ cạnh nên ta có $k \leq |V| - 1$, kết hợp lại có $h(u) \leq 2|V| - 1$. ĐPCM.

Định lý 9-18 (thời gian thực hiện giải thuật FIFO Preflow-Push)

Có thể cài đặt giải thuật FIFO Preflow-Push để tìm luồng cực đại trên mạng $G = (V, E, c, s, t)$ trong thời gian $O(|V|^3 + |V||E|)$.

Chứng minh

Ta sẽ chứng minh mô hình cài đặt thuật toán FIFO Preflow-Push ở trên có thời gian thực hiện là $O(|V|^3 + |V||E|)$. Vòng lặp chính của thuật toán mỗi lượt lấy một đỉnh tràn z khỏi hàng đợi và cố gắng tháo luồng cho đỉnh z bằng các phép $Push$ theo các cung đi ra khỏi z. Nếu z chưa hết tràn thì thực hiện phép $Lift(z)$ và đẩy lại z vào hàng đợi. Như vậy thuật toán FIFO Preflow-Push sẽ thực hiện một dãy các phép $Lift$ và $Push$:

$$Lift(.), Push(.), Push(.) \ldots, Push(.), Lift(.), Push(.), \ldots$$

Trước hết ta chứng minh rằng số phép $Lift$ trong dãy thao tác trên là $O(|V|^2)$ và tổng thời gian thực hiện chúng là $O(|V||E|)$. Thật vậy, Mỗi phép $Lift$ sẽ nâng độ cao của một đỉnh lên ít nhất 1, ngoài ra độ cao của mỗi đỉnh không vượt quá $2|V| - 1$ (theo hệ quả của định lý Định lý 9-17). Cứ cho là mọi đỉnh $\in V - \{s, t\}$ khi kết thúc thuật toán đều có độ cao $2|V| - 1$ đi nữa thì do chúng được khởi tạo bằng 1, tổng số phép $Lift$ cần thực hiện cũng không vượt quá:

$$(|V| - 2)(2|V| - 2) = O(|V|^2)$$

Mỗi cung (u, v) sẽ được xét đến đúng một lần trong phép $Lift(u)$, phép $Lift(u)$ lại được gọi không quá $2|V| - 2$ lần. Vậy tổng cộng trong tất cả các phép $Lift$ thì mỗi cung sẽ được xét không quá $2|V| - 2$ lần, mạng có $|E|$ cung suy ra tổng thời gian thực hiện của các phép $Lift$ trong giải thuật là $|E|(2|V| - 2) = O(|V||E|)$.

Tiếp theo ta chứng minh rằng số phép đẩy bão hòa cũng như tổng thời gian thực hiện chúng là $O(|V||E|)$. Sau phép đẩy bão hòa $Push(e)$, nếu muốn thực hiện tiếp phép đẩy $Push(e)$ nữa thì trước đó chắc chắn phải có phép đẩy $Push(-e)$ để làm giảm $f(e)$ và biến e trở lại thành cung thặng dư. Giả sử $e = (u, v)$ và $-e = (v, u)$ thì để thực hiện phép $Push(e)$, ta phải có $h(u) > h(v)$. Để thực hiện $Push(-e)$, ta phải có $h(v) > h(u)$ và để thực hiện tiếp $Push(e)$ nữa ta lại phải có $h(u) > h(v)$. Bởi độ cao của các đỉnh không bao giờ giảm đi nên sau phép $Push(e)$ thứ hai, độ cao $h(u)$ lớn hơn ít nhất 2 đơn vị so với $h(u)$ ở phép $Push(e)$ thứ nhất. Vậy nếu một cung $e = (u, v)$ của mạng được đẩy bão hòa k lần thì độ cao của đỉnh u sẽ tăng lên ít nhất là $2(k - 1)$. Vì độ cao của các đỉnh không vượt quá $2|V| - 1$ nên số phép đẩy bão hòa trên mỗi cung e là $k \leq |V|$. Mạng có $|E|$ cung và thời gian thực hiện phép $Push$ là $O(1)$ nên số phép đẩy bão hòa là $O(|V||E|)$ và thời gian thực hiện chúng cũng là $O(|V||E|)$.

Đối với các phép đẩy không bão hòa, việc đánh giá thời gian thực hiện giải thuật được thực hiện bằng *hàm thế* (*potential function*). Định nghĩa hàm thế Φ là độ cao lớn nhất của các đỉnh tràn:

$$\Phi = \max\{h[v] : v \text{ là đỉnh tràn}\}$$

Trong trường hợp mạng không còn đỉnh tràn thì ta quy ước $\Phi = 0$. Vậy $\Phi \leq 1$ khi khởi tạo tiền luồng và trở lại bằng 0 khi thuật toán kết thúc.

Chia dãy các thao tác $Lift$ và $Push$ làm các pha liên tiếp. Pha thứ nhất bắt đầu khi hàng đợi được khởi tạo gồm các đỉnh tràn và kết thúc khi tất cả các đỉnh đó (và chỉ những đỉnh đó thôi) đã được lấy ra khỏi hàng đợi và xử lý. Pha thứ hai tiếp tục với hàng đợi gồm những đỉnh được đẩy vào trong pha thứ nhất và kết thúc khi tất cả các đỉnh này được lấy ra khỏi hàng đợi và xử lý, pha thứ ba, thứ tư… được chia ra theo cách tương tự như vậy.

Nhận xét rằng phép $Push$ chỉ đẩy luồng từ đỉnh cao xuống đỉnh thấp, vậy nên những đỉnh được đẩy vào hàng đợi sau phép $Push$ luôn thấp hơn đỉnh đang xét vừa lấy ra khỏi hàng đợi. Suy ra nếu một pha chỉ chứa phép $Push$ thì giá trị hàm thế Φ sau pha đó giảm đi ít nhất 1 đơn vị.

Giá trị hàm thế Φ chỉ **có thể** tăng lên sau một pha nếu pha đó có chứa phép $Lift$ và giá trị Φ tăng lên phải bằng một độ cao của một đỉnh v nào đó sau phép $Lift(v)$ trong pha. Xét mức tăng của Φ sau pha đang xét:

$$\Phi_{\text{mới}} - \Phi_{\text{cũ}} = h(v)_{\text{mới}} - \Phi_{\text{cũ}} \leq h(v)_{\text{mới}} - h(v)_{\text{cũ}}$$

Tức là sau mỗi pha làm Φ tăng lên, luôn tồn tại một đỉnh v mà mức tăng độ cao của v lớn hơn mức tăng của Φ. Xét trên toàn bộ giải thuật, độ cao của mỗi đỉnh $v \in V - \{s, t\}$ được khởi tạo bằng 0 và được nâng lên tối đa bằng $2|V| - 1$ nên tổng toàn bộ mức tăng của các đỉnh không vượt quá $(|V| - 2)(2|V| - 1) = O(|V|^2)$.

Vậy nếu ta xét các pha làm Φ tăng thì tổng mức tăng của Φ trên các pha này là $O(|V|^2)$, tức là số các pha làm Φ giảm cũng phải là $O(|V|^2)$. Nói cách khác, sẽ chỉ có $O(|V|^2)$ pha có chứa phép

$Lift$ và $O(|V|^2)$ pha không chứa phép $Lift$. Cộng lại ta có số pha cần thực hiện trong toàn bộ giải thuật là $O(|V|^2)$.

Một pha sẽ phải lấy khỏi hàng đợi tối đa $|V| - 2$ đỉnh để xử lý. Với mỗi đỉnh lấy từ hàng đợi, việc tháo luồng sẽ chỉ sử dụng tối đa 1 phép đẩy không bão hòa vì sau phép đẩy này thì đỉnh sẽ hết tràn và quá trình xử lý sẽ chuyển sang đỉnh tiếp theo trong hàng đợi. Vậy trong mỗi pha có không quá $|V| - 2$ phép đẩy không bão hòa. Vì tổng số pha là $O(|V|^2)$, ta có số phép đẩy không bão hòa trong cả giải thuật là $O(|V|^3)$ và tổng thời gian thực hiện chúng cũng là $O(|V|^3)$.

Cuối cùng, ta đánh giá thời gian thực hiện những thao tác duyệt danh sách liên thuộc bằng các chỉ số $current[.]$ trong thuật toán FIFO Preflow-Push. Với mỗi đỉnh z, chỉ số $current[z]$ ban đầu sẽ ứng với nút đầu danh sách liên thuộc và chuyển dần đến hết danh sách gồm $\deg^+(z)$ nút. Khi duyệt hết danh sách liên thuộc mà z vẫn tràn thì sẽ có một phép $Lift(z)$ và con trỏ $current[z]$ được đặt lại về đầu danh sách liên thuộc. Số phép $Lift(z)$ trong toàn bộ giải thuật không vượt quá $2|V| - 1$, nên số lượt dịch chỉ số $current[z]$ không vượt quá $2|V| \deg^+(z)$. Suy ra nếu xét tổng thể, số phép dịch các chỉ số $current[.]$ trên tất cả các danh sách liên thuộc phải nhỏ hơn:

$$(2|V|)\sum_{z\in V} \deg^+(z) = 2|V||E| = O(|V||E|)$$

Kết luận:

Tổng thời gian thực hiện các phép nâng: $O(|V||E|)$.

Tổng thời gian thực hiện các phép đẩy bão hòa: $O(|V||E|)$.

Tổng thời gian thực hiện các phép đẩy không bão hòa: $O(|V|^3)$.

Tổng thời gian thực hiện các phép duyệt danh sách liên thuộc bằng chỉ số $current[.]$: $O(|V||E|)$.

Thời gian thực hiện giải thuật FIFO Preflow-Push: $O(|V|^3 + |V||E|)$.

9.3.8. Một số kỹ thuật tăng tốc độ giải thuật

Ta đã chứng minh rằng thuật toán Edmonds-Karp có thời gian thực hiện $O(|V||E|^2)$ và thuật toán FIFO Preflow-Push có thời gian thực hiện $O(|V|^3 + |V||E|)$. Những đại lượng này thoạt nhìn làm chúng ta có cảm giác như thuật toán FIFO Preflow-Push thực hiện nhanh hơn thuật toán Edmonds-Karp, đặc biệt trong trường hợp đồ thị dày ($|E| \gg |V|$).

Tuy vậy, những đánh giá này chỉ là cận trên của thời gian thực hiện giải thuật trong trường hợp xấu nhất. Hiện tại chưa có các đánh giá chặt về cận trên và cận dưới trong trường hợp trung bình. Những thử nghiệm bằng chương trình cụ thể cũng cho thấy rằng các thuật toán đẩy tiền luồng như FIFO Preflow-Push, Lift-to-Front Preflow-Push, Highest-Label Preflow-Push… không có cải thiện gì về tốc độ so với thuật toán Edmonds-Karp (thậm chí còn chậm hơn) nếu không sử dụng những mẹo cài đặt (*heuristics*).

Chưa có đánh giá lý thuyết chặt chẽ nào về tác động của những mẹo cài đặt lên thời gian thực hiện giải thuật nhưng hầu hết các thử nghiệm đều cho thấy việc sử dụng những mẹo cài đặt trên thực tế gần như là bắt buộc đối với các thuật toán đẩy tiền luồng.

❑ Bản chất của hàm độ cao

Nhắc lại về ràng buộc độ cao: Xét một tiền luồng f trên mạng $G = (V, E, c, s, t)$, hàm độ cao $h: V \to \mathbb{N}$ gọi là tương ứng với tiền luồng f nếu $h(s) = |V|$; $h(t) = 0$; và với mọi cung thặng dư (u, v) thì $h(u) \leq h(v) + 1$.

Nếu ta gán trọng số cho các cung của mạng thặng dư G_f theo quy tắc: Cung thặng dư có trọng số 1 và cung bão hòa có trọng số $+\infty$. Ký hiệu $\delta_f(u, v)$ là độ dài đường đi ngắn nhất từ u tới v trên G_f với cách gán trọng số này. Khi đó không khó khăn kiểm chứng được rằng với $\forall v \in V - \{s, t\}$:

- $h(v) \leq \delta_f(v, t)$, tức là $h(v)$ luôn là cận dưới của độ dài đường đi ngắn nhất từ v tới đỉnh thu.
- Trong trường hợp $h(v) > |V|$, từ v chắc chắn không có đường thặng dư đi tới t và $h(v) - |V| \leq \delta_f(v, s)$, tức là $h(v) - |V|$ trong trường hợp này là cận dưới của độ dài đường đi ngắn nhất từ v về đỉnh phát.

Những mẹo cài đặt dưới đây nhằm đẩy nhanh các độ cao $h(v)$ trong tiến trình thực hiện giải thuật dựa vào những nhận xét trên.

❑ Gán nhãn lại toàn bộ

Nội dung của phương pháp gán nhãn lại toàn bộ (*global relabeling heuristic*) được tóm tắt như sau: Xét lát cắt chia tập V làm hai tập rời nhau X và Y: Tập Y gồm những đỉnh đến được t bằng một đường thặng dư và tập X gồm những đỉnh còn lại. Chắc chắn không có cung thặng dư nối từ X sang Y, ta có $s \in X, t \in Y$. Phép gán nhãn lại toàn bộ sẽ đặt:

- Với $\forall v \in Y$, ta gán lại độ cao $h[v] := \delta_f(v, t)$.
- Với $\forall u \in X$ và $\delta_f(u, s) < +\infty$, ta gán lại độ cao $h[u] := |V| + \delta_f(u, s)$
- Với $\forall u \in X$ và $\delta_f(u, s) = +\infty$, ta gán lại độ cao $h[u] := 2|V| - 1$

Không khó khăn để kiểm chứng tính hợp lý của hàm độ cao mới. Có thể thấy rằng các độ cao mới ít ra là không thấp hơn các độ cao cũ.

Các giá trị $\delta_f(v, t)$ cũng như $\delta_f(u, s)$ có thể được xác định bằng hai lượt thực hiện thuật toán BFS từ t và s. Bởi ta cần thời gian $O(|E|)$ cho hai lượt BFS và gán lại các độ cao, nên phép gán nhãn lại toàn bộ thường được gọi thực hiện sau một loạt chỉ thị sơ cấp để không làm ảnh hưởng tới đánh giá O lớn của thời gian thực hiện giải thuật (chẳng hạn sau mỗi $|V|$ phép $Lift$). Chú ý là khi nâng độ cao $h[z]$ của một đỉnh z nào đó, cần cập nhật lại $current[z] := head[z]$.

❑ Đẩy nhãn theo khe

Phép đẩy nhãn theo khe (*gap heuristic*) được thực hiện nhờ quan sát sau:

Giả sử ta có một số nguyên $0 < gap < |V|$ mà không đỉnh nào có độ cao gap (số nguyên gap này được gọi là "khe"), khi đó mọi đỉnh z có $h[z] > gap$ đều không có đường thặng dư đi đến t.

Nhận định trên có thể chứng minh bằng phản chứng: Giả sử từ z có đường thặng dư đi đến t, với một cung (u, v) trên đường đi ta có $h(u) \leq h(v) + 1$, tức là trên đường đi này, từ một đỉnh u ta chỉ có thể đi sang một đỉnh v không thấp hơn hoặc thấp hơn u đúng một đơn vị. Từ $h(z) > gap > 0$ và $h(t) = 0$, chắc chắn trên đường thặng dư từ x tới t phải có một đỉnh độ cao gap. Mâu thuẫn với giả thuyết phản chứng.

Phép đẩy nhãn theo khe nếu phát hiện khe $0 < gap < |V|$ sẽ xét tất cả những đỉnh $z \in V - \{s\}$ có $gap < h(z) \leq |V|$ và đặt lại $h[z] \coloneqq |V| + 1$.

Ta sẽ chỉ ra rằng phép đẩy độ cao này vẫn đảm bảo ràng buộc độ cao của hàm h. Độ cao của đỉnh phát và đỉnh thu không bị động chạm đến, tức là $h[s] = |V|$ và $h[t] = 0$. Trước khi thực hiện phép đẩy theo khe, ta chia tập đỉnh V thành hai tập rời nhau: Tập X gồm những đỉnh cao hơn gap và tập Y gồm những đỉnh thấp hơn gap. Do ràng buộc độ cao $h(u) \leq h(v) + 1$ với mọi cung thặng dư (u, v), không tồn tại cung thặng dư nối từ X tới Y. Phép đẩy theo khe chỉ tăng độ cao của một vài đỉnh $x \in X$ và như vậy ràng buộc độ cao nếu bị vi phạm thì chỉ bị vi phạm trên những cung thặng dư đi ra khỏi x. Như lập luận trên, cung thặng dư đi ra khỏi x chắc chắn phải đi vào một đỉnh $x' \in X$ có độ cao ít nhất là $|V|$ sau phép đẩy theo khe. Từ $h(x) = |V| + 1$ ta có $h(x) \leq h(x') + 1$.

Phép đẩy nhãn theo khe sử dụng mảng $count[0 \dots 2|V| - 1]$ để đếm $count[k]$ là số đỉnh có độ cao k. Mỗi khi có sự thay đổi độ cao, ta phải đồng bộ lại mảng $count$ theo tình trạng hàm độ cao mới. Sau mỗi phép $Lift(u)$, độ cao cũ của đỉnh u được lưu trữ lại trong biến $OldH$ và phép $Lift$ thực hiện như bình thường. Sau đó nếu $0 < OldH < |V|$ và $count[OldH] = 0$, phép đẩy theo khe $OldH$ sẽ được gọi và thực hiện trong thời gian $O(|V|)$. Bởi số phép $Lift$ cần thực hiện trong toàn bộ giải thuật là $O(|V|^2)$, tổng thời gian thực hiện các phép đẩy theo khe sẽ là $O(|V|^3)$ nên không ảnh hưởng tới đánh giá O-lớn của thời gian thực hiện giải thuật FIFO Preflow-Push.

Dưới đây là bảng so sánh tốc độ của các chương trình cài đặt cụ thể trên một số bộ dữ liệu. Với một cặp số n, m, 100 đồ thị với n đỉnh, m cung được sinh ngẫu nhiên với sức chứa là số nguyên trong khoảng từ 0 tới 10^4. Có 4 chương trình được thử nghiệm: A: Thuật toán Edmonds-Karp, B: thuật toán FIFO Preflow-Push, C: thuật toán FIFO Preflow-Push với phép gán nhãn lại toàn bộ và D: thuật toán FIFO Preflow-Push với phép đẩy nhãn theo khe. Mỗi chương trình được thử trên cả 100 đồ thị và đo thời gian thực hiện trung bình (tính bằng giây):

	$n = 100$ $m = 10000$	$n = 200$ $m = 30000$	$n = 500$ $m = 40000$	$n = 800$ $m = 90000$	$n = 1000$ $m = 100000$
A	0.0688	0.5925	0.6598	1.7158	2.7629
B	0.0983	0.7395	3.4377	9.9014	25.0723
C	0.0313	0.0624	0.0857	0.1809	0.2433
D	0.0282	0.0577	0.0828	0.1575	0.1889

❑ Cài đặt

Dưới đây là chương trình cài đặt thuật toán FIFO Preflow-Push kết hợp với kỹ thuật đẩy nhãn theo khe, việc cài đặt và đánh giá hiệu suất của phép gán nhãn lại toàn bộ chúng ta coi như bài tập. Các bạn có thể thử cài đặt kết hợp cả hai kỹ thuật tăng tốc này để xác định xem việc đó có thực sự cần thiết không.

Input/Output có khuôn dạng giống như ở chương trình cài đặt thuật toán Edmonds-Karp. Hàng đợi chứa các đỉnh tràn được tổ chức dưới dạng danh sách vòng: Các chỉ số đầu/cuối hàng đợi sẽ chạy xuôi trong một mảng và khi chạy đến hết mảng sẽ tự động quay về đầu mảng.

💻 FIFOPREFLOWPUSH.PAS ✓ Thuật toán FIFO Preflow-Push

```
{$MODE OBJFPC}
program MaximumFlow;
const
  maxN = 1000;
  maxM = 100000;
  maxC = 10000;
type
  TEdge = record //Cấu trúc một cung
    x, y: Integer;  //Hai đỉnh đầu mút
    c, f: Integer;  //Sức chứa và luồng
  end;
  TQueue = record //Cấu trúc hàng đợi
    items: array[0..maxN - 1] of Integer;  //Danh sách vòng
    front, rear, nItems: Integer;
  end;
var
  e: array[-maxM..maxM] of TEdge;  //Mảng chứa các cung
  link: array[-maxM..maxM] of Integer;  //Móc nối trong danh sách liên thuộc
  head, current: array[1..maxN] of Integer;  //con trỏ tới đầu và vị trí hiện tại của danh sách liên thuộc
  excess: array[1..maxN] of Integer;  //mức tràn của các đỉnh
  h: array[1..maxN] of Integer;  //hàm độ cao
  count: array[0..2 * maxN - 1] of Integer;  //count[k] = số đỉnh có độ cao k
  Queue: TQueue;  //Hàng đợi chứa các đỉnh tràn
  n, m, s, t: Integer;
  FlowValue: Integer;

procedure Enter;  //Nhập dữ liệu
var
  i: Integer;
  u, v, capacity: Integer;
```

```
begin
  ReadLn(n, m, s, t);
  FillChar(head[1], n * SizeOf(head[1]), 0);
  for i := 1 to m do
    begin
      ReadLn(u, v, capacity);
      with e[i] do //Thêm cung e[i] = (u, v) vào danh sách liên thuộc của u
        begin
          x := u; y := v; c := capacity;
          link[i] := head[u]; head[u] := i;
        end;
      with e[-i] do //Thêm cung e[-i] = (v, u) vào danh sách liên thuộc của v
        begin
          x := v; y := u; c := 0;
          link[-i] := head[v]; head[v] := -i;
        end;
    end;
  for v := 1 to n do current[v] := head[v];
end;

procedure PushToQueue(v: Integer); //Đẩy một đỉnh v vào hàng đợi
begin
  with Queue do
    begin
      rear := (rear + 1) mod maxN; //Dịch chỉ số cuối hàng đợi, rear = maxN - 1 sẽ trở lại thành 0
      items[rear] := v; //Đặt v vào vị trí cuối hàng đợi
      Inc(nItems); //Tăng biến đếm số phần tử trong hàng đợi
    end;
end;

function PopFromQueue: Integer; //Lấy một đỉnh khỏi hàng đợi
begin
  with Queue do
    begin
      Result := items[front]; //Trả về phần tử ở đầu hàng đợi
      front := (front + 1) mod maxN; //Dịch chỉ số đầu hàng đợi, front = maxN - 1 sẽ trở lại thành 0
      Dec(nItems); //Giảm biến đếm số phần tử trong hàng đợi
    end;
end;

procedure Init; //Khởi tạo
var
  v: Integer;
  sf: Integer;
  i: Integer;
begin
  //Khởi tạo tiền luồng
  for i := -m to m do e[i].f := 0;
  FillChar(excess[1], n * SizeOf(excess[1]), 0);
  i := head[s];
  while i <> 0 do //Duyệt các cung đi ra khỏi đỉnh phát và đẩy bão hòa các cung đó, cập nhật các mức tràn excess[.]
    begin
      sf := e[i].c;
      e[i].f := sf; e[-i].f := -sf;
      Inc(excess[e[i].y], sf); Dec(excess[s], sf);
      i := link[i];
    end;
```

```
  //Khởi tạo hàm độ cao
  for v := 1 to n do h[v] := 1;
  h[s] := n; h[t] := 0;
  //Khởi tạo các biến đếm: count[k] là số đỉnh có độ cao k
  FillChar(count[0], (2 * n) * SizeOf(count[0]), 0);
  count[n] := 1; count[0] := 1; count[1] := n - 2;
  //Khởi tạo hàng đợi chứa các đỉnh tràn
  Queue.front := 0; Queue.rear := -1; Queue.nItems := 0; //Hàng đợi rỗng
  for v := 1 to n do //Duyệt tập đỉnh
    if (v <> s) and (v <> t) and (excess[v] > 0) then //v tràn
      PushToQueue(v); //đẩy v vào hàng đợi
end;

procedure Push(i: Integer); //Phép đẩy luồng theo cung e[i]
var
  Delta: Integer;
begin
  with e[i] do
    if excess[x] < c - f then Delta := excess[x]
    else Delta := c - f;
  Inc(e[i].f, Delta); Dec(e[-i].f, Delta);
  with e[i] do
    begin
      Dec(excess[x], Delta); Inc(excess[y], Delta);
    end;
end;

procedure SetH(u: Integer; NewH: Integer); //Đặt độ cao của u thành NewH, đồng bộ hóa mảng count
begin
  Dec(count[h[u]]);
  h[u] := NewH;
  Inc(count[NewH]);
end;

procedure PerformGapHeuristic(gap: Integer); //Đẩy nhãn theo khe gap
var
  v: Integer;
begin
  if (0 < gap) and (gap < n) and (count[gap] = 0) then //gap đúng là khe thật
    for v := 1 to n do
      if (v <> s) and (gap < h[v]) and (h[v] <= n) then
        begin
          SetH(v, n + 1);
          current[v] := head[v]; //Nâng độ cao của v cần phải cập nhật lại con trỏ current[v]
        end;
end;

procedure Lift(u: Integer); //Phép nâng đỉnh u
var
  minH, OldH: Integer;
  i: Integer;
begin
  minH := 2 * maxN;
  i := head[u];
  while i <> 0 do //Duyệt các cung đi ra khỏi u
    begin
      with e[i] do
```

```
        if (c > f) and (h[y] < minH) then //Gặp cung thặng dư (u, v), ghi nhận đỉnh v thấp nhất
          minH := h[y];
      i := link[i];
    end;
  OldH := h[u]; //Nhớ lại h[u] cũ
  SetH(u, minH + 1); //nâng cao đỉnh u
  PerformGapHeuristic(OldH); //Có thể tạo ra khe OldH, đẩy nhãn theo khe
end;

procedure FIFOPreflowPush; //Thuật toán FIFO Preflow-Push
var
  NeedQueue: Boolean;
  z: Integer;
begin
  while Queue.nItems > 0 do //Chừng nào hàng đợi vẫn còn đỉnh tràn
    begin
      z := PopFromQueue; //Lấy một đỉnh tràn x khỏi hàng đợi
      while current[z] <> 0 do //Xét một cung đi ra khỏi x
        begin
          with e[current[z]] do
            begin
              if (c > f) and (h[x] > h[y]) then //Nếu có thể đẩy luồng được theo cung (u, v)
                begin
                  NeedQueue := (y <> s) and (y <> t) and (excess[y] = 0);
                  Push(current[z]); //Đẩy luồng luôn
                  if NeedQueue then //v đang không tràn sau phép đẩy trở thành tràn
                    PushToQueue(y); //Đẩy v vào hàng đợi
                  if excess[z] = 0 then Break; //x hết tràn thì chuyển qua xét đỉnh khác ngay
                end;
            end;
          current[z] := link[current[z]]; //x chưa hết tràn thì chuyển sang xét cung liên thuộc tiếp theo
        end;
      if excess[z] > 0 then //Duyệt hết danh sách liên thuộc mà x vẫn tràn
        begin
          Lift(z); //Nâng cao x
          current[z] := head[z]; //Đặt con trỏ current[x] trỏ lại về đầu danh sách liên thuộc
          PushToQueue(z); //Đẩy lại x vào hàng đợi chờ xử lý sau
        end;
    end;
  FlowValue := excess[t]; //Thuật toán kết thúc, giá trị luồng bằng tổng luồng đi vào đỉnh thu (= - excess[s])
end;

procedure PrintResult; //In kết quả
var
  i: Integer;
begin
  WriteLn('Maximum flow: ');
  for i := 1 to m do
    with e[i] do
      if f > 0 then //Chỉ cần in ra các cung có luồng > 0
        WriteLn('e[', i, '] = (', x, ', ', y, '): c = ', c, ', f = ', f);
  WriteLn('Value of flow: ', FlowValue);
end;

begin
  Enter; //Nhập dữ liệu
```

```
  Init;  //Khởi tạo
  FIFOPreflowPush;   //Thực hiện thuật toán đẩy tiền luồng
  PrintResult;   //In kết quả
end.
```

Định lý 9-19 (định lý về tính nguyên)

Nếu tất cả các sức chứa là số nguyên thì thuật toán Ford-Fulkerson cũng như thuật toán đẩy tiền luồng luôn tìm được luồng cực đại với luồng trên cung là các số nguyên.

Chứng minh

Đối với thuật toán Ford-Fulkerson, ban đầu ta khởi tạo luồng 0 thì luồng trên các cung là nguyên. Mỗi lần tăng luồng dọc theo đường tăng luồng P, luồng trên mỗi cung hoặc giữ nguyên, hoặc tăng/giảm một lượng Δ_P cũng là số nguyên. Vậy nên cuối cùng luồng cực đại phải có giá trị nguyên trên tất cả các cung.

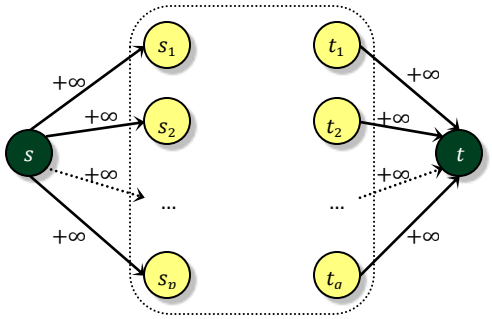
Đối với thuật toán đẩy tiền luồng, ban đầu ta khởi tạo một tiền luồng trên các cung là số nguyên. Phép $Lift$ và $Push$ không làm thay đổi tính nguyên của tiền luồng trên các cung. Vậy nên khi thuật toán kết thúc, tiền luồng trở thành luồng cực đại với giá trị luồng trên các cung là số nguyên.

9.4. Một số mở rộng và ứng dụng của luồng

9.4.1. Mạng với nhiều đỉnh phát và nhiều đỉnh thu

Ta mở rộng khái niệm mạng bằng cách cho phép mạng G có p đỉnh phát: $s_1, s_2, \dots, s_p$ và q đỉnh thu $t_1, t_2, \dots, t_q$, các đỉnh phát và các đỉnh thu hoàn toàn phân biệt. Hàm sức chứa và luồng trên mạng được định nghĩa tương tự như trong trường hợp mạng có một đỉnh phát và một đỉnh thu. Giá trị của luồng được định nghĩa bằng tổng luồng trên các cung đi ra khỏi các đỉnh phát. Bài toán đặt ra là tìm luồng cực đại trên mạng có nhiều đỉnh phát và nhiều đỉnh thu.

Thêm vào mạng hai đỉnh: một siêu đỉnh phát s và siêu đỉnh thu t. Thêm các cung nối từ s tới các đỉnh s_i có sức chứa $+\infty$, thêm các cung nối từ các đỉnh t_j tới t với sức chứa $+\infty$. Ta được một mạng mới $G' = (V, E')$ (Hình 9-5).



Hình 9-5. Mạng với nhiều đỉnh phát và nhiều đỉnh thu

Có thể thấy rằng nếu f là một luồng cực đại trên G', thì f hạn chế trên G cũng là luồng cực đại trên G. Vậy để tìm luồng cực đại trên G, ta sẽ tìm luồng cực đại trên G' rồi loại bỏ siêu đỉnh phát s, siêu đỉnh thu t và tất cả những cung giả mới thêm vào.

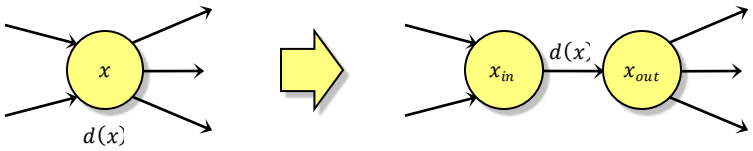
Một cách khác có thể thực hiện để tìm luồng trên mạng có nhiều đỉnh phát và nhiều đỉnh thu là loại bỏ tất các các cung đi vào các đỉnh phát cũng như các cung đi ra khỏi các đỉnh thu. Chập tất cả các đỉnh phát thành một siêu đỉnh s và chập tất cả các đỉnh thu lại thành một siêu đỉnh thu t, mạng không còn các đỉnh $s_1, s_2, .., s_p$ và $t_1, t_2, \dots, t_q$ nữa mà chỉ có thêm đỉnh phát s và đỉnh thu t. Trên mạng ban đầu, mỗi cung đi vào/ra s_i được chỉnh lại đầu mút để nó đi vào/ra đỉnh s, mỗi cung đi vào/ra t_j cũng được chỉnh lại đầu mút để nó đi vào/ra đỉnh t, ta được một mạng mới G''.

Khi đó ta có thể tìm f là một luồng cực đại trên G'' và khôi phục lại đầu mút của các cung như cũ để f trở thành luồng cực đại trên G.

9.4.2. Mạng với sức chứa trên cả các đỉnh và các cung

Cho mạng $G = (V, E, c, s, t)$, mỗi đỉnh $v \in V - \{s, t\}$ được gán một số không âm $d(v)$ gọi là sức chứa của đỉnh đó. Luồng dương φ trên mạng này được định nghĩa với tất cả các ràng buộc của luồng dương và thêm một điều kiện: Tổng luồng dương trên các cung đi vào mỗi đỉnh $v \in V - \{s, t\}$ không được vượt quá $d(v)$: $\sum_{e \in E^-(v)} \varphi(e) \leq d(v)$. Bài toán đặt ra là tìm luồng dương cực đại trên mạng có ràng buộc sức chứa trên cả các đỉnh và các cung.

Tách mỗi đỉnh $x \in V - \{s, t\}$ thành 2 đỉnh mới x_{in}, x_{out} và một cung (x_{in}, x_{out}) với sức chứa $d(x)$. Các cung đi vào x được chỉnh lại đầu mút để đi vào x_{in} và các cung đi ra khỏi x được chỉnh lại đầu mút để đi ra khỏi x_{out} (Hình 9-6). Ta xây dựng được mạng $G' = (V', E')$ với đỉnh phát s và đỉnh thu t.



Hình 9-6. Tách đỉnh

Khi đó việc tìm luồng dương cực đại trên mạng G có thể thực hiện bằng cách tìm luồng dương cực đại trên mạng G', sau đó chập tất cả các cặp (x_{in}, x_{out}) trở lại thành đỉnh x ($\forall x \in V - \{s, t\}$) để khôi phục lại mạng G ban đầu.

9.4.3. Mạng với ràng buộc luồng dương bị chặn hai phía

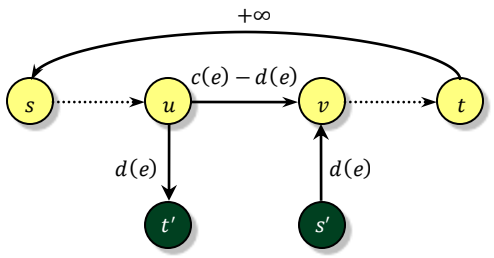
Cho mạng $G = (V, E, c, s, t)$ trong đó mỗi cung $e \in E$ ngoài sức chứa (lưu lượng) tối đa $c(e)$ còn được gán một số không âm $d(e) \leq c(e)$ gọi là lưu lượng tối thiểu. Một *luồng dương tương thích* φ trên G được định nghĩa với tất cả các ràng buộc của luồng dương và thêm một điều kiện: Luồng dương trên mỗi cung $e \in E$ không được nhỏ hơn sức chứa tối thiểu của cung đó:

$$d(e) \le \varphi(e) \le c(e)$$

Bài toán đặt ra là kiểm chứng sự tồn tại của luồng dương tương thích trên mạng với ràng buộc luồng dương bị chặn hai phía.

Xây dựng một mạng $G' = (V', E')$ từ mạng G theo quy tắc:

- Tập đỉnh V' có được từ tập V thêm vào đỉnh phát giả s' và đỉnh thu giả t': $V' = V + \{s', t'\}$.
- Mỗi cung $e = (u, v) \in E$ sẽ tương ứng với ba cung trên E': cung $e_1 = (u, v)$ có sức chứa $c(e) - d(e)$, cung $e_2 = (s', v)$ và cung $e_3 = (u, t')$ có sức chứa $d(e)$. Ngoài ra thêm vào cung $(t, s) \in E'$ với sức chứa $+\infty$



Hình 9-7.

Gọi $D = \sum_{e \in E} d(e)$ là tổng sức chứa tối thiểu của các cung trên mạng G. Khi đó trên mạng G', tổng sức chứa các cung đi ra khỏi s' cũng như tổng sức chứa các cung đi vào t' bằng D. Vì vậy với mọi luồng dương trên G' thì giá trị luồng đó không thể vượt quá D.

Từ đó suy ra rằng nếu tồn tại một luồng dương φ' trên G' có giá trị luồng $|\varphi'| = D$ thì φ' bắt buộc là luồng dương cực đại trên G'.

Bổ đề 9-20

Điều kiện cần và đủ để tồn tại luồng dương tương thích φ trên mạng G là tồn tại luồng dương cực đại φ' trên G' với giá trị luồng $|\varphi'| = D$.

Chứng minh

Giả sử luồng dương cực đại φ' trên G' có $|\varphi| = D = \sum_{e \in E} d(e)$. Ta xây dựng luồng φ trên G bằng cách cộng thêm vào luồng φ' trên mỗi cung e một lượng $d(e)$:

$$\varphi: E \to [0, +\infty)$$
$$e \to \varphi(e) = \varphi'(e) + d(e)$$

Khi đó có thể dễ dàng kiểm chứng được φ thỏa mãn tất cả các ràng buộc của luồng dương tương thích trên mạng G.

Ngược lại nếu φ là một luồng dương tương thích trên G. Ta xây dựng luồng dương φ' trên G' bằng cách trừ luồng φ trên mỗi cung e đi một lượng $d(e)$, đồng thời đặt luồng φ' trên các cung đi ra khỏi s' cũng như trên các cung đi vào t' đúng bằng sức chứa của cung đó. Khi đó cũng dễ dàng kiểm chứng được φ' là luồng dương cực đại và $|\varphi| = D$.

9.4.4. Mạng với sức chứa âm

Cho mạng $G = (V, E, c, w, s, t)$ trong đó ta mở rộng khái niệm sức chứa bằng cách cho phép cả những sức chứa âm trên một số cung. Khái niệm luồng được định nghĩa như bình thường.

Nếu như có thể khởi tạo được một luồng thì thuật toán Ford-Fulkerson vẫn hoạt động đúng để tìm luồng cực đại trên mạng có sức chứa âm. Vấn đề khởi tạo một luồng bất kỳ trên mạng không phải đơn giản vì chúng ta không thể khởi tạo bằng luồng 0, bởi nếu như vậy, ràng buộc sức chứa tối đa sẽ bị vi phạm trên các cung có sức chứa âm.

Giả sử một cung $e \in E$ có sức chứa $c(e) < 0$. Theo tính đối xứng lệch của luồng $f(e) = -f(-e)$ và ràng buộc sức chứa tối đa $f(e) \leq c(e)$, ta có:

$$f(-e) = -f(e) \geq -c(e) > 0 \qquad (9.5)$$

Như vậy ràng buộc sức chứa tối đa $f(e) \leq c(e)$ tương đương với ràng buộc về sức chứa tối thiểu $-c(e)$ trên cung đối $-e$. Việc chỉ ra một luồng bất kỳ trên G có thể thực hiện bằng cách tìm luồng dương trên mạng với ràng buộc luồng dương bị chặn hai phía sau đó biến đổi luồng dương này thành luồng cần tìm.

9.4.5. Lát cắt hẹp nhất

Ta quan tâm tới đồ thị vô hướng liên thông $G = (V, E)$ với hàm trọng số (hay lưu lượng) $c: E \to [0, +\infty)$. Giả sử $|V| \geq 2$, người ta muốn bỏ đi một số cạnh để đồ thị mất tính liên thông và yêu cầu tìm phương án sao cho tổng trọng số các cạnh bị loại bỏ là nhỏ nhất.

Bài toán cũng có thể phát biểu dưới dạng: hãy phân hoạch tập đỉnh V thành hai tập khác rỗng rời nhau X và Y sao cho tổng lưu lượng các cạnh nối giữa X và Y là nhỏ nhất có thể. Cách phân hoạch này gọi là lát cắt tổng quát hẹp nhất của G, ký hiệu $MinCut(G)$.

$$c(X, Y) \to \min$$

$$X \neq \emptyset; Y \neq \emptyset; X \cap Y = \emptyset; X \cup Y = V;$$

Một cách tệ nhất có thể thực hiện là thử tất cả các cặp đỉnh s, t. Với mỗi lần thử ta cho s làm đỉnh phát và t làm đỉnh thu trên mạng G, sau đó tìm luồng cực đại và lát cắt $s - t$ hẹp nhất. Cuối cùng là chọn lát cắt $s - t$ có lưu lượng nhỏ nhất trong tất cả các lần thử. Phương pháp này cần $\binom{n}{2} = \frac{n \times (n-1)}{2}$ lần tìm luồng cực đại, có tốc độ chậm và không khả thi với dữ liệu lớn.

Bổ đề 9-21

Với s và t là hai đỉnh bất kỳ. Từ đồ thị G, ta xây dựng đồ thị G_{st} bằng cách chập hai đỉnh s và t thành một đỉnh duy nhất, ký hiệu st, các cạnh nối s với t bị hủy bỏ, các cạnh liên thuộc với chỉ s hoặc t được chỉnh lại đầu mút để trở thành cạnh liên thuộc với st. Khi đó $MinCut(G)$ có thể thu được bằng lấy lát cắt có lưu lượng nhỏ nhất trong hai lát cắt:

- Lát cắt $s - t$ hẹp nhất: Coi s là đỉnh phát và t là đỉnh thu, lát cắt $s - t$ hẹp nhất có thể xác định bằng việc giải quyết bài toán luồng cực đại trên mạng G.

- Lát cắt tổng quát hẹp nhất trên G_{st}: $MinCut(G_{st})$.

Chứng minh

Xét lát cắt tổng quát hẹp nhất trên G có thể đưa s và t vào hai thành phần liên thông khác nhau hoặc đưa chúng vào cùng một thành phần liên thông. Trong trường hợp thứ nhất, $MinCut(G)$ là lát cắt $s-t$ hẹp nhất. Trong trường hợp thứ hai, $MinCut(G)$ là $MinCut(G_{st})$.

Bổ đề 9-21 cho phép chúng ta xây dựng một thuật toán tốt hơn: Nếu đồ thị chỉ gồm 2 đỉnh thì chỉ việc cắt rời hai đỉnh vào hai tập. Nếu không, ta chọn hai đỉnh bất kỳ s, t làm đỉnh phát và đỉnh thu, tìm luồng cực đại và ghi nhận lát cắt $s-t$ hẹp nhất. Tiếp theo ta chập hai đỉnh s, t thành một đỉnh st và lặp lại với đồ thị G_{st}... Cuối cùng là chỉ ra lát cắt $s-t$ hẹp nhất trong số tất cả các lát cắt được ghi nhận. Phương pháp này đòi hỏi phải thực hiện $|V|-1$ lần tìm luồng cực đại, tuy đã có sự cải thiện về tốc độ nhưng chưa phải thật tốt.

Nhận xét rằng tại mỗi bước của cách giải trên, chúng ta có thể chọn hai đỉnh s, t bất kỳ miễn sao $s \neq t$. Vì vậy người ta muốn tìm một cách chọn cặp đỉnh s, t một cách hợp lý tại mỗi bước để có thể chỉ ra ngay lát cắt $s-t$ hẹp nhất mà không cần tìm luồng cực đại. Thuật toán dưới đây (Stoer & Wagner, 1997) là một trong những thuật toán hiệu quả dựa trên ý tưởng đó.

Với A là một tập con của tập đỉnh V và x là một đỉnh không thuộc A. Định nghĩa *lực hút* của A đối với x là tổng trọng số các cạnh nối x với các đỉnh thuộc A:

$$c(A,\{x\}) = \sum_{\substack{e=(x,y)\in E \\ y\in A}} c(e)$$

Bổ đề 9-22

Bắt đầu từ tập A chỉ gồm một đỉnh bất kỳ $a \in V$, ta cứ tìm một đỉnh bị A hút chặt nhất kết nạp thêm vào A cho tới khi $A = V$. Gọi s và t là hai đỉnh được kết nạp cuối cùng theo cách này. Khi đó lát cắt $(V-\{t\},\{t\})$ là lát cắt $s-t$ hẹp nhất.

Chứng minh

Xét một lát cắt $s-t$ bất kỳ κ, ta sẽ chứng minh rằng lưu lượng của lát cắt $(V-\{t\},\{t\})$ không lớn hơn lưu lượng của lát cắt κ.

Một đỉnh v được gọi là *đỉnh hoạt tính* nếu v và đỉnh được đưa vào A liền trước v bị rơi vào hai phía của lát cắt κ. Gọi A_v là tập các đỉnh được kết nạp vào A trước đỉnh v, κ_v là lát cắt κ hạn chế trên $A_v \cup \{v\}$ (Lát cắt κ_v dùng đúng cách phân hoạch của lát cắt κ nhưng chỉ quan tâm tới tập đỉnh $A_v \cup \{v\}$). Gọi $c(\kappa)$ là lưu lượng của lát cắt κ, $c(\kappa_v)$ là lưu lượng của lát cắt κ_v.

Trước hết ta sử dụng phép quy nạp để chỉ ra rằng nếu u là đỉnh hoạt tính thì:

$$c(A_u,\{u\}) \leq c(\kappa_u) \qquad (9.6)$$

Nếu u là đỉnh hoạt tính đầu tiên được kết nạp vào A, lát cắt κ_u sẽ chia tập $A_u \cup \{u\}$ làm hai tập, một tập là A_u và một tập là $\{u\}$, khi đó ta có $c(A_u,\{u\})$ cũng chính là $c(\kappa_u)$. Giả thiết rằng bất

đẳng thức (9.6) đúng với đỉnh hoạt tính u, ta sẽ chứng nó cũng đúng với những đỉnh hoạt tính v được kết nạp vào A sau u. Thật vậy,

$$c(A_v,\{v\}) = c(A_u,\{v\}) + c(A_v - A_u,\{v\}) \tag{9.7}$$

Do A_u phải hút u mạnh hơn v, kết hợp với giả thiết quy nạp, ta có:

$$c(A_u,\{v\}) \le c(A_u,\{u\}) \le c(\kappa_u)$$

Hạng tử $c(A_v - A_u,\{v\})$ là tổng trọng số các cạnh nối giữa v và $A_v - A_u$. Do u và v là hai đỉnh hoạt tính liên tiếp, các cạnh này sẽ nối giữa hai phía của lát cắt κ_v và có đóng góp trong phép tính $c(\kappa_v)$, mặt khác do $v \notin A_u \cup \{u\}$ nên những cạnh này không đóng góp trong phép tính $c(\kappa_u)$. Vậy từ công thức (9.7), ta suy ra:

$$\begin{aligned} c(A_v,\{v\}) &= c(A_u,\{v\}) + c(A_v - A_u,\{v\}) \\ &\le c(\kappa_u) + c(A_v - A_u,\{v\}) \\ &\le c(\kappa_v) \end{aligned} \tag{9.8}$$

Vì κ là một lát cắt $s-t$ nên chắc chắn s và t nằm ở hai phía khác nhau của lát cắt κ, hay nói cách khác, t là đỉnh hoạt tính. Bất đẳng thức (9.6) chứng minh ở trên cho ta kết quả:

$$\begin{aligned} c(V - \{t\},\{t\}) &= c(A_t,\{t\}) \\ &\le c(\kappa_t) \\ &= c(\kappa) \end{aligned} \tag{9.9}$$

Ta chứng minh được lát cắt $(V - \{t\},\{t\})$ là lát cắt $s-t$ hẹp nhất.

Định lý 9-23

Việc tìm lát cắt tổng quát trên đồ thị vô hướng liên thông với hàm trọng số không âm có thể được thực hiện bằng thuật toán trong thời gian $O(|V|^2 \log|V| + |V||E|)$.

Chứng minh

Bắt đầu từ tập A chỉ gồm một đỉnh bất kỳ, ta mở rộng A bằng cách lần lượt kết nạp vào A đỉnh bị hút chặt nhất cho tới khi $A = V$. Việc này được thực hiện với kỹ thuật tương tự như thuật toán Prim: với $\forall v \notin A$, ta ký hiệu nhãn $d[v]$ là lực hút của A đối với đỉnh v. khi A được kết nạp thêm một u thì các nhãn lực hút của những đỉnh v khác sẽ được cập nhật lại theo công thức:

$$d[v]_{\text{mới}} := d[v]_{\text{cũ}} + c(u,v), \forall (u,v) \in E$$

Bằng việc tổ chức các đỉnh ngoài A trong một hàng đợi ưu tiên dạng Fibonacci Heap, việc mở rộng tập A cho tới khi $A = V$ được thực hiện trong thời gian $O(|V|\log|V| + |E|)$. Trong quá trình đó, s và t là hai đỉnh cuối cùng được kết nạp vào A cũng được xác định và $MinCut(G)$ được cập nhật theo lát cắt $s-t$ hẹp nhất. Sau đó hai đỉnh s, t được chập vào và thuật toán lặp lại với đồ thị G_{st}. Tổng cộng ta có $|V| - 1$ lần lặp, suy ra lát cắt tổng quát hẹp nhất có thể tìm được trong thời gian $O(|V|^2 \log|V| + |V||E|)$.

Mặc dù tính đúng đắng của thuật toán được chứng minh dựa vào lý thuyết về luồng cực đại và lát cắt hẹp nhất, việc cài đặt thuật toán lại khá đơn giản và không động chạm gì đến luồng cực đại.

Bài tập 9-1

Cho f_1 và f_2 là hai luồng trên mạng $G = (V, E, c, s, t)$ và α là một số thực nằm trong đoạn $[0,1]$. Xét ánh xạ:

$$f_\alpha: E \to \mathbb{R}$$
$$e \mapsto f_\alpha(e) = \alpha f_1(e) + (1-\alpha) f_2(e)$$

Chứng minh rằng f_α cũng là một luồng trên mạng G với giá trị luồng:

$$|f_\alpha| = \alpha |f_1| + (1-\alpha)|f_2|$$

Bài tập 9-2

Cho f là một luồng trên mạng $G = (V, E, c, s, t)$, chứng minh rằng với $\forall e \in E$, ta có:

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

Bài tập 9-3

Cho f là luồng cực đại trên mạng $G = (V, E, c, s, t)$, gọi Y là tập các đỉnh đến được t bằng một đường thặng dư trên G_f và $X = V - Y$. Chứng minh rằng (X, Y) là lát cắt $s - t$ hẹp nhất của mạng G.

Bài tập 9-4

Viết chương trình nhận vào một đồ thị có hướng $G = (V, E)$ với hai đỉnh phân biệt s và t và tìm một tập gồm nhiều đường đi nhất từ s tới t sao cho các đường đi trong tập này đôi một không có cạnh chung.

Gợi ý

Coi s là đỉnh phát và t là đỉnh thu, các cung đều có sức chứa 1 và tìm luồng cực đại trên mạng, theo Định lý 9-19 (định lý về tính nguyên), luồng trên các cung chỉ có thể là 0 hoặc 1. Loại bỏ các cung có luồng 0 và chỉ giữ lại các cung có luồng 1. Tiếp theo ta tìm một đường đi từ s tới t, chọn đường đi này vào tập hợp, loại bỏ tất cả các cung dọc trên đường đi này khỏi đồ thị và lặp lại…, thuật toán sẽ kết thúc khi đồ thị không còn cạnh nào (không còn đường đi từ s tới t).

Về kỹ thuật cài đặt, ta có thể tìm một đường đi từ s tới t trên đồ thị G, đảo chiều tất cả các cung trên đường đi này và lặp lại cho tới khi không còn đường đi từ s tới t nữa. Có thể thấy rằng đồ thị G tại mỗi bước chính là đồ thị các cung thặng dư và đường đi tìm được ở mỗi bước chính là đường tăng luồng.

Đồ thị G giờ đây không còn đường đi từ s tới t, ta tìm một đường đi từ t về s, kết nạp đường đi theo chiều ngược lại (từ s tới t) vào tập hợp, xóa bỏ tất cả các cung trên đường đi và cứ tiếp tục như vậy cho tới khi không còn đường đi từ t về s nữa.

Bài tập 9-5

Tương tự như Bài tập 9-4 nhưng yêu cầu thực hiện trên đồ thị vô hướng.

Bài tập 9-6 (Hệ đại diện phân biệt)

Một lớp học có n bạn nam và n bạn nữ. Nhân ngày 8/3, lớp có mua m món quà để các bạn nam tặng các bạn nữ. Mỗi món quà có thể thuộc sở thích của một số bạn trong lớp.

Hãy lập chương trình tìm cách phân công tặng quà thỏa mãn:

- Mỗi bạn nam phải tặng quà cho đúng một bạn nữ và mỗi bạn nữ phải nhận quà của đúng một bạn nam. Món quà được tặng phải thuộc sở thích của cả hai người.
- Món quà nào đã được một bạn nam chọn để tặng thì bạn nam khác không được chọn nữa.

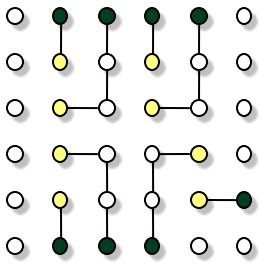
Gợi ý: Xây dựng một mạng trong đó tập đỉnh V gồm 3 lớp đỉnh S, X và T:

- Lớp đỉnh phát $S = \{s_1, s_2, \dots, s_n\}$, mỗi đỉnh tương ứng với một bạn nam.
- Lớp đỉnh $X = (x_1, x_2, \dots, x_n)$ mỗi đỉnh tương ứng với một món quà.
- Lớp đỉnh thu $T = \{t_1, t_2, \dots, t_n\}$ mỗi đỉnh tương ứng với một bạn nữ.

Nếu bạn nam i thích món quà k, ta cho cung nối từ s_i tới x_k, nếu bạn nữ j thích món quà k, ta cho cung nối từ x_k tới t_j. Sức chứa của các cung đặt bằng 1 và sức chứa của các đỉnh $v_1, v_2, \dots, v_n$ cũng đặt bằng 1. Tìm luồng nguyên cực đại trên mạng G có n đỉnh phát, n đỉnh thu, đồng thời có cả ràng buộc sức chứa trên các đỉnh, những cung có luồng 1 sẽ nối giữa một món quà và người tặng/nhận tương ứng.

Bài tập 9-7

Cho mạng điện gồm $m \times n$ điểm nằm trên một lưới m hàng, n cột. Một số điểm nằm trên biên của lưới là nguồn điện, một số điểm trên lưới là các thiết bị sử dụng điện. Người ta chỉ cho phép nối dây điện giữa hai điểm nằm cùng hàng hoặc cùng cột. Hãy tìm cách đặt các dây điện nối các thiết bị sử dụng điện với nguồn điện sao cho hai đường dây bất kỳ nối hai thiết bị sử dụng điện với nguồn điện tương ứng của chúng không được có điểm chung.



Bài tập 9-8 (Kỹ thuật giãn sức chứa)

Cho mạng $G = (V, E, c, w, s, t)$ với sức chứa nguyên: $c: E \to \mathbb{N}$. Gọi $C := \max_{e \in E} c(e)$.

a) Chứng minh rằng lát cắt $s - t$ hẹp nhất của G có lưu lượng không vượt quá $C|E|$

b) Với một số nguyên k, tìm thuật toán xác định đường tăng luồng có giá trị thặng dư $\geq k$ trong thời gian $O(|E|)$.

c) Chứng minh rằng thuật toán sau đây tìm được luồng cực đại trên mạng G:

```
procedure MaxFlowByScaling;
begin
  f := «Luồng 0»;
  k := C; //k là sức chứa lớn nhất của một cung trong E
  while k ≥ 1 do
    begin
      while «Tìm được đường tăng luồng P có giá trị thặng dư ≥ k» do
        «Tăng luồng dọc theo đường P»;
      k := k div 2;
    end;
end;
```

d) Chứng minh rằng khi bước vào mỗi lượt lặp của vòng lặp:

```
while k ≥ 1 do...
```

Lưu lượng của lát cắt hẹp nhất trên mạng thặng dư G_f không vượt quá $2k|E|$.

e) Chứng minh rằng trong mỗi lượt lặp của vòng lặp:

```
while k ≥ 1 do...
```

Vòng lặp while bên trong thực hiện $O(E)$ lần với mỗi giá trị của k.

f) Chứng minh rằng thuật toán trên (*maximum flow by scaling*) có thể cài đặt để tìm luồng cực đại trên G trong thời gian $O(|E|^2 \log C)$.