Chương 29. Luồng cực đại trên mạng

29.1. Các khái niệm và bài toán

29.1.1. 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
- 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 số xác định trên tập cung E

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

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

Để thuận tiện cho việc trình bày, ta quy ước các ký hiệu sau: Với X, Y là hai tập đỉnh và $f: E \to \mathbb{R}$ là một hàm xác định trên tập cung E:

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

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

Trong trường hợp X chỉ gồm một đỉnh x, ta dùng ký hiệu $\{x \to Y\}$ thay cho $\big\{\{x\} \to Y\big\}$, tương tự như vậy trong trường hợp Y chỉ gồm một đỉnh y, ta dùng ký hiệu $\{X \to y\}$ thay cho $\big\{X \to \{y\}\big\}$

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

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

Bổ đề 29-1

Cho $f: E \to \mathbb{R}$ là hàm xác định trên tập cung e.

Với $\forall X, Y, Z \subseteq V, X \cap Y = \emptyset$, ta có:

$$f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$$
$$f(Z, X \cup Y) = f(Z, X) + f(Z, Y)$$

Chứng minh

Phép chứng minh đơn giản dùng định nghĩa:

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

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

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

Bổ đề 29-2

Cho $f: E \to \mathbb{R}$ và $g: E \to \mathbb{R}$ là các hàm xác định trên tập cung e. Xét hàm $f + g$ và $f - g$ xác định như sau:

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

Khi đó $\forall X, Y \subseteq V$ ta có:

$$\begin{aligned} (f+g)(X,Y) &= f(X,Y) + g(X,Y) \\ (f-g)(X,Y) &= f(X,Y) - g(X,Y) \end{aligned}$$

Chứng minh

Phép chứng minh đơn giản dùng định nghĩa:

$$\begin{aligned} (f \pm g)(X,Y) &= \sum_{\forall e \in \{X \to Y\}} (f \pm g)(e) = \sum_{\forall e \in \{X \to Y\}} \big(f(e) \pm g(e)\big) \\ &= \sum_{\forall e \in \{X \to Y\}} f(e) \pm \sum_{\forall e \in \{X \to Y\}} g(e) = f(X,Y) \pm g(X,Y) \end{aligned}$$

29.1.2. Luồng dương

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

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

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:

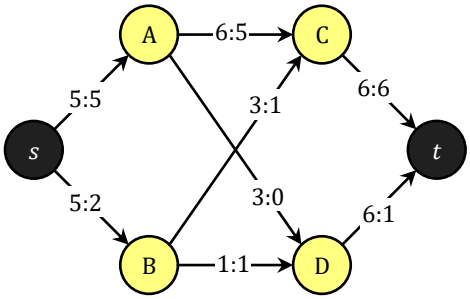
- Ràng buộc về sức chứa (*Capacity constraint*): Luồng dương trên mỗi cung không được vượt quá sức chứa của cung đó:
$$\forall e \in E: 0 \leq \varphi(e) \leq c(e)$$
- Ràng buộc về tính bảo tồn (*Flow conservation*): Với mỗi đỉnh u không phải đỉnh phát và cũng không phải đỉnh thu, tổng luồng dương trên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

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*:

$$|\varphi| = \varphi(s,V) - \varphi(V,s) \qquad (29.1)$$

* 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.

Chú ý rằng luồng dương φ trên các cung là số không âm nhưng giá trị luồng $|\varphi|$ hoàn toàn có thể là số âm.



Hình 29-1. Ví dụ về một mạng, nhãn ghi trên cung là sức chứa:luồng dương

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 dương có giá trị lớn nhất trên mạng G.

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 là những ví dụ thực tế của bài toán luồng cực đại trên mạng.

29.1.3. Mở rộng khái niệm luồng

Định nghĩa về luồng dương khá trực quan và dễ hiểu, tuy vậy định nghĩa này không thích hợp lắm trong việc chứng minh và cài đặt các giải thuật. Ta mở rộng khái niệm luồng bằng các thêm các cung giả và cho phép luồng trên các cung giả này có thể âm.

Với mỗi cung $e = (u, v)$ của mạng, ta thêm vào mạng một cung giả đi ngược chiều, ký hiệu $-e = (v, u)$, gọi là cung đối của cung e, ta cũng coi e là cung đối của cung $-e$ (tức là $e = -(-e)$).

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: $\forall e \in E: f(e) \leq c(e)$
- Ràng buộc về tính phản đối xứng (*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 u không phải đỉnh phát và cũng không phải đỉnh thu, tổng luồng trên các cung đi ra khỏi u bằng 0: $\forall u \in V - \{s, t\}: f(u, V) = 0$

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

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

$$|f| = f(u, V) \tag{29.2}$$

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.

29.1.4. Mối quan hệ giữa luồng và luồng dương

Bổ đề 29-3 (Phép xây dựng luồng dương từ luồng)

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

$$\varphi: E \longrightarrow \mathbb{R}_{\geq 0}$$
$$e \longmapsto \varphi(e) = \max\{f(e), 0\} \tag{29.3}$$

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

Bổ đề 29-3 cho thấy luồng dương φ được tạo thành từ luồng f bằng cách bỏ đi tất cả các cung giả ta đã thêm vào mạng. Thật vậy, những cung giả có sức chứa 0 nên luồng trên cung ≤ 0. Cách xây dựng đặt luồng dương φ trên các cung đó bằng 0 nên việc có hay không những cung đó trên mạng không ảnh hưởng tới luồng dương φ.

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. Ngoài ra $c(e) \geq 0$ theo quy định mạng và $c(e) \geq f(e)$ theo ràng buộc về sức chứa của luồng f nên $c(e) \geq \max\{f(e), 0)\} = \varphi(e)$.

Ràng buộc về tính bảo tồn: Xét một đỉnh u bất kỳ, ta có:

$$f(u,V) = \sum_{\forall e \in \{u \to V\}} f(e) = \sum_{\substack{\forall e \in \{u \to V\} \\ f(e) \geq 0}} f(e) + \sum_{\substack{\forall e \in \{u \to V\} \\ f(e) < 0}} f(e)$$

$$= \underbrace{\sum_{\substack{\forall e \in \{u \to V\} \\ f(e) \geq 0}} f(e)}_{\varphi(u,V)} - \underbrace{\sum_{\substack{\forall -e \in \{V \to u\} \\ f(-e) \geq 0}} f(-e)}_{\varphi(V,u)}$$

Điều này có thể hiểu như sau: Tổng luồng f ra khỏi u 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 u. Những cung ra khỏi u mang luồng âm lại có một cung đối đi vào u mang luồng dương trái dấu. Vì vậy cộng luồng trên các cung mang luồng âm ra khỏi u tương đương với việc trừ đi tổng luồng trên các cung mang luồng dương đi vào u. Tổng hợp lại ta được: Tổng luồng f ra khỏi u bằng tổng luồng dương φ ra khỏi u trừ đi tổng luồng dương φ đi vào u.

$$f(u,V) = \varphi(u,V) - \varphi(V,u)$$

Nếu u không phải đỉnh phát cũng không phải đỉnh thu. Từ bảo tồn của luồng f cho ta $f(u,V) = 0$ hay $\varphi(u,V) = \varphi(V,u)$, tức là hàm φ thỏa mãn ràng buộc về tính bảo tồn.

Về giá trị luồng φ, cũng từ chứng minh trên, thay u bởi s, ta có:

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

Bổ đề 29-4 (Phép xây dựng luồng từ luồng dương)

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

$$\begin{aligned} f: E &\longrightarrow \mathbb{R} \\ e &\longmapsto f(e) = \varphi(e) - \varphi(-e) \end{aligned} \tag{29.4}$$

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 phản đối xứng: Với $\forall e \in E$:

$$f(e) = \varphi(e) - \varphi(-e) = -\big(\varphi(-e) - \varphi(e)\big) = -f(-e)$$

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

$$f(u,V) = \sum_{\forall e \in \{u \to V\}} \big(\varphi(e) - \varphi(-e)\big)$$

$$= \left(\sum_{\forall e \in \{u \to V\}} \varphi(e)\right) - \left(\sum_{\forall e \in \{u \to V\}} \varphi(-e)\right)$$

$$= \left(\sum_{\forall e \in \{u \to V\}} \varphi(e) \right) - \left(\sum_{\forall -e \in \{V \to u\}} \varphi(-e) \right)$$
$$= \varphi(u, V) - \varphi(V, u)$$

Nếu $u \notin \{s, t\}$, ta có:

$$f(u, V) = \varphi(u, V) - \varphi(V, u) = 0$$

(Do tính bảo tồn của luồng dương φ)

Nếu $u = s$, xét giá trị luồng f

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

Bổ đề 29-3 và Bổ đề 29-4 cho ta một mối tương quan giữa luồng và luồng dương. Khái niệm về luồng dương trực quan 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 cũng như dễ dàng mở rộng cho các bài toán khác như luồng chi phí cực tiểu. Ta sẽ **sử dụng luồng dương trong các hình vẽ và output** (chỉ quan tâm tới các giá trị luồng dương $\varphi(e)$, những cung giả thêm vào không cần quan tâm), còn các khái niệm về luồng sẽ được dùng để diễn giải các thuật toán.

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).

29.1.5. Một số tính chất cơ bản

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

Bổ đề 29-5

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) $\forall X \subseteq V - \{s, t\}$, 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 phản đối xứng của luồng: $f(e) = -f(-e)$, ta có $f(X, X) = 0$.

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

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

c) $\forall X \subseteq V - \{s, t\}$, theo Bổ đề 29-1:

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

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 29-6

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.

Chứng minh

Suy ra trực tiếp từ Định lý 29-6 và Bổ đề 29-4 (Phép xây dựng luồng từ luồng dương).

29.1.6. Mạng dư lượng

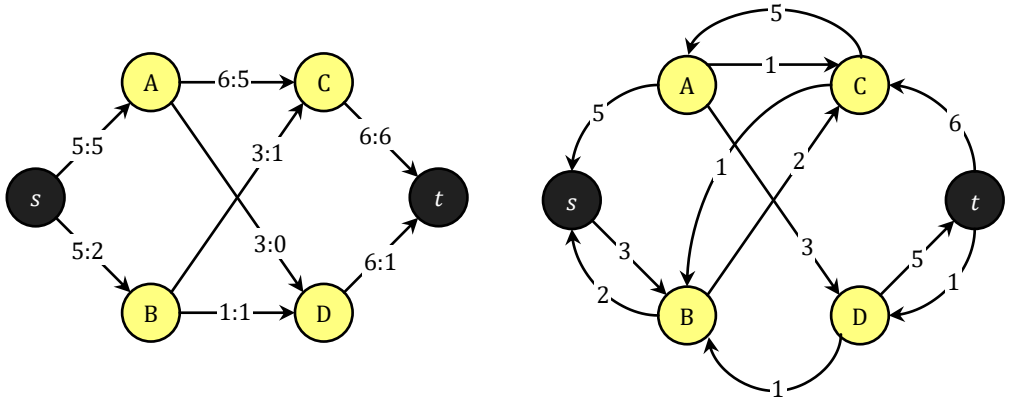
Với f là một luồng trên mạng $G = (V,E,c,s,t)$. Ta xét mạng G_f cũng là mạng G nhưng với hàm sức chứa mới cho bởi:

$$\begin{aligned} c_f &: E \longrightarrow \mathbb{R}_{\geq 0} \\ e &\longmapsto c_f(e) = c(e) - f(e) \end{aligned} \tag{29.5}$$

Mạng G_f xây dựng như vậy được gọi là *mạng dư lượng* (*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 gọi là *cung bão hòa (saturated edge)* nếu dư lượng của cung đó bằng 0, ngược lại cung đó gọi là *cung dư (residual edge)*. Ký hiệu E_f là tập các cung dư trên mạng dư lượng G_f. Một đường dư (*residual path*) trên G_f là một đường đi chỉ chứa các cung dư ($\in E_f$).

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 dư ($\in E_f$) trong các hình vẽ.



Hình 29-2. Một luồng trên mạng và mạng dư lượng tương ứng

Hình 29-2 là một ví dụ về mạng dư lượng ứng với một luồng. Để đỡ rối hình, trên mạng ban đầu ta không vẽ các cung giả, tuy nhiên ta cần hiểu rằng mạng có cung (A, C) với sức chứa 6 và luồng 5 thì cũng có cung giả (C, A) với sức chứa 0 và luồng -5. Vậy nên trên mạng dư lượng, cung (A, C) có sức chứa $6 - 5 = 1$ và cung (C, A) với sức chứa $0 - (-5) = 5$.

Định lý 29-7 (Tổng luồng)

Cho f là một luồng trên mạng $G = (V, E, c, s, t)$ và d là một luồng trên G_f thì hàm:

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

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

Chứng minh

Về bản chất có thể coi luồng f trên mạng G là luồng chính và luồng d trên mạng dư lượng G_f là luồng phụ mà ta muốn gộp vào luồng chính. Ta chứng minh $f + d$ thỏa mãn các ràng buộc của một luồng trên mạng G.

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

$$(f + d)(e) = f(e) + d(e) \leq f(e) + c_f(e) = f(e) + c(e) - f(e) = c(e)$$

Ràng buộc về tính phản đối xứng: $\forall e \in E$,

$$(f + d)(e) = f(e) + d(e) = -f(-e) - d(-e) = -(f + d)(-e)$$

Ràng buộc về tính bảo tồn: $\forall u \in V$, xét tổng luồng $f + d$ đi ra khỏi u, theo Bổ đề 29-2:

$$(f + d)(u, V) = f(u, V) + d(u, V)$$

Nếu $u \notin \{s, t\}$, do tính bảo tồn của luồng f trên G và luồng d trên G_f, ta có $f(u, V)$ và $d(u, V)$ đều bằng 0, từ đó suy ra $(f + d)(u, V) = 0$.

Về giá trị luồng thay $u = s$, ta có:

$$|f + d| = (f + d)(s, V) = f(s, V) + d(s, V) = |f| + |d|$$

Định lý 29-8 (Hiệu luồng)

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

$$g - f: E \longrightarrow \mathbb{R}$$

$$e \longmapsto (g-f)(e) = g(e) - f(e)$$

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

Chứng minh

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

Ràng buộc về sức chứa:

$$(g-f)(e) = g(e) - f(e) \le c(e) - f(e) = c_f(e)$$

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

$$(g-f)(e) = g(e) - f(e) = -\big(g(-e) - f(-e)\big) = -(g-f)(-e)$$

Tính bảo tồn: Với $\forall u \in V$, theo Bổ đề 29-2:

$$(g-f)(u,V) = g(u,V) - f(u,V)$$

Nếu $u \notin \{s,t\}$, từ tính bảo tồn của luồng g cũng như luồng f, ta có $g(u,V) = f(u,V) = 0$, hay $(g-f)(u,V) = 0$.

Về giá trị luồng, thay $u = s$, ta có:

$$|g-f| = (g-f)(s,V) = g(s,V) - f(s,V) = |g| - |f|$$

29.2. Thuật toán Ford-Fulkerson

29.2.1. 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 dư lượng G_f. *Dư lượng* (*residual capacity*) của đường P, ký hiệu Δ_P, được định nghĩa bằng dư lượng nhỏ nhất của các cung dọc trên đường P:

$$\Delta_P = \min\{c_f(e) : e \in P\}$$

Chú ý là $c_f(e) \ge 0$ nên Δ_P luôn ≥ 0. Nếu $\Delta_P > 0$ đường đi P gọi là một *đường tăng luồng (augmenting path)* tương ứng với luồng f.

Định lý 29-9

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:

$$f_P(e) = \begin{cases} +\Delta_P, \text{ nếu } e \in P \\ -\Delta_p, \text{ nếu } -e \in P \\ 0, \text{ trường hợp khác} \end{cases} \qquad (29.6)$$

là một luồng trên mạng dư lượng G_f

Chứng minh

Bản chất của luồng f_P là đẩy một giá trị luồng Δ_P từ s tới t trên mạng dư lượng dọc theo đường tăng luồng.

Ràng buộc về sức chứa khá hiển nhiên: rõ ràng $f_P(e) \le \Delta_P \le c_f(e)$, với $\forall e \in P$ và với $\forall e \notin P$ thì $f_P(e) = 0 \le c_f(e)$ do dư lượng luôn là số không âm.

Ràng buộc về tính phản đối xứng cũng hiển nhiên, với một cặp cung đối nhau e và $-e$ thì chỉ có tối đa một cung e hoặc $-e$ thuộc đường P do P là đường đi đơn, luồng trên một

cung đặt bằng Δ_P thì luồng trên cung đối đặt bằng $-\Delta_P$, đảm bảo $f(e) = -f(-e)$. Trường hợp cả e và $-e$ đều không thuộc đường P thì $f(e) = -f(-e) = 0$.

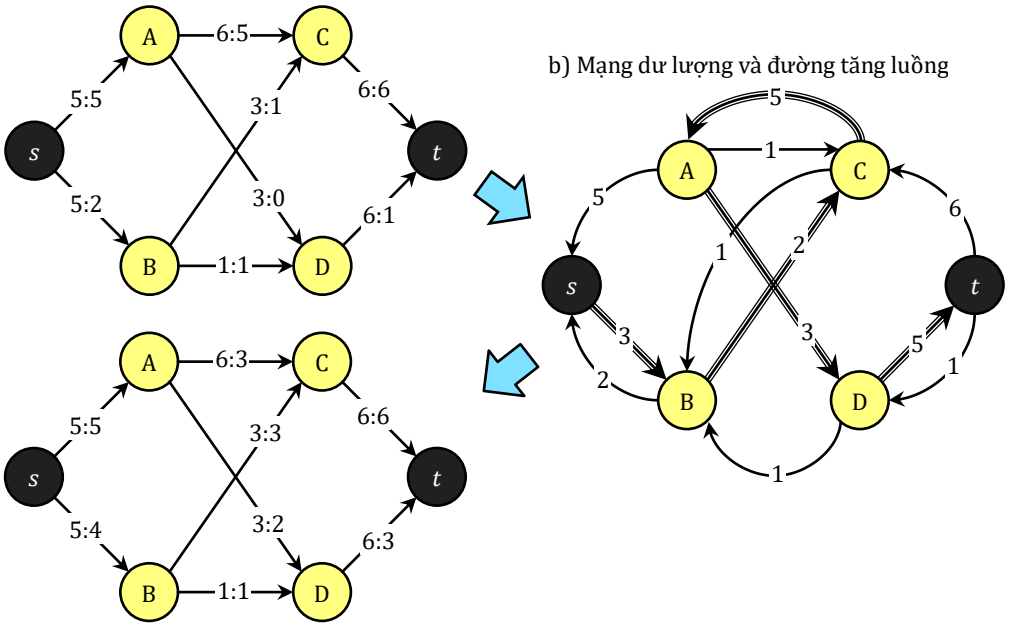
Ràng buộc về tính bảo tồn: $\forall u \notin \{s,t\}$ nằm trên đường P thì có một cung $e \in P$ đi vào u và một cung $e' \in P$ ra khỏi u, tức là trên mạng dư lượng có 2 cung mang luồng f_P khác 0 ra khỏi u: Cung $-e$ mang luồng $-\Delta_P$ và cung e' mang luồng Δ_P, tổng luồng trên hai cung này bằng 0 tức là $f_P(u,V) = 0$. Dĩ nhiên nếu $u \notin P$ thì $f_P(u,V) = 0$ theo cách xây dựng luồng f_P

Riêng với đỉnh s, chỉ có một cung mang luồng khác 0 ($= \Delta_P$) đi ra khỏi s, cung này là cung đầu tiên trên đường P, từ đó suy ra $|f_P| = \Delta_P$.

Định lý 29-9 vừa chứng minh và Định lý 29-7 (Tổng luồng) cho ta một hệ quả sau:

Hệ quả 29-10

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



Hình 29-3. Tăng luồng dọc đường tăng luồng.

Hình 29-3 là ví dụ về cơ chế tăng luồng trên mạng với đỉnh phát s, đỉnh thu t và luồng f giá trị 7 (hình a, chú ý là ta không vẽ các cung giả đỡ rối). Với mạng dư lượng G_f (hình b, ta chỉ vẽ các cung dư), đường đi $P = \langle s,B,C,A,D,t\rangle$ được chọn làm đường tăng luồng, dư lượng của P bằng $\Delta_P = 2$ (dư lượng của cung (B,C)).

Luồng f_P trên G_f sẽ có các giá trị sau:

$$f_P(s,B) = f_P(B,C) = f_P(C,A) = f_P(A,D) = f_P(D,t) = 2$$
$$f_P(t,D) = f_P(D,A) = f_P(A,C) = f_P(C,B) = f_P(B,s) = -2$$

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

29.2.2. Thuật toán Ford-Fulkerson

Thuật toán Ford-Fulkerson [24] 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 dư lượng, 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»)
   f = f + fP;
Output ← f;
```

29.2.3. Cài đặt

Thuật toán Ford-Fulkerson được cài đặt để tìm luồng cực đại trên mạng với khuôn dạng Input/Output như sau:

Input

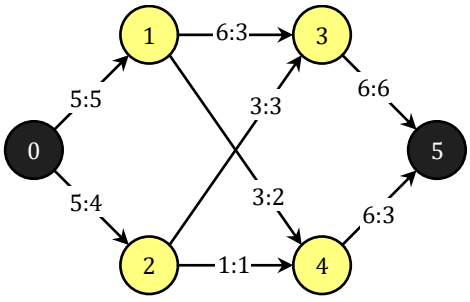
Mạng với n đỉnh, m cung. Các đỉnh đánh số từ 0 tới $n-1$, các cung đánh số từ 0 tới $m-1$

- Dòng 1 chứa số đỉnh $n \leq 10^5$, 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 (chỉ đưa ra luồng trên các cung đã cho).

Sample Input	Sample Output
6 8 0 5 0 1 5 0 2 5 1 3 6 1 4 3 2 3 3 2 4 1 3 5 6 4 5 6	e[0] = (0, 1): c = 5, f = 5 e[1] = (0, 2): c = 5, f = 4 e[2] = (1, 3): c = 6, f = 3 e[3] = (1, 4): c = 3, f = 2 e[4] = (2, 3): c = 3, f = 3 e[5] = (2, 4): c = 1, f = 1 e[6] = (3, 5): c = 6, f = 6 e[7] = (4, 5): c = 6, f = 3 Value of flow: 9



❋ Tổ chức các cặp cung đối nhau

Để duy trì thông tin về các cặp cung đối nhau, tất cả m cung của mạng và cung đối của chúng được chứa trong danh sách $e[0 \ldots 2m-1]$. Các cung ban đầu được lưu trữ ở các vị trí chẵn, các cung giả cho thêm làm cung đối của chúng được lưu ở vị trí lẻ liền sau. Tức là các cặp cung đối được bố trí theo thứ tự:

$$e[0] = -e[1]$$
$$e[2] = -e[3]$$
$$\ldots$$
$$e[2m-2] = -e[2m-1]$$

Khi đó với một cung $e[i]$, ta có thể dễ dàng chỉ ra được cung đối của nó bằng công thức $e[i\text{^}1]$ trong đó toán tử ^ là phép toán XOR (eXclusive OR). Danh sách liên thuộc $L[u]$ chứa chỉ số các cung trong mảng e đi ra khỏi u.

Các cung hoàn toàn có thể tổ chức trong một cấu trúc dữ liệu khác, miễn là ngoài các thông tin về hai đầu mút, sức chứa, luồng, ta cần lưu con trỏ tới cung đối nữa. Tuy vậy đối với bài toán này, phương pháp tổ chức danh sách cung như trên tỏ ra tiết kiệm bộ nhớ và có thể tận dụng được tốt cơ chế cache của CPU.

❋ Duy trì thông tin về dư lượng

Việc duy trì thông tin về sức chứa, luồng, dư lượng trên mỗi cung sẽ trở nên khá cồng kềnh khi có sự thay đổi về luồng. Vì vậy ta chỉ duy trì thông tin về dư lượng, cụ thể là mỗi phần tử của danh sách e chứa 3 trường (x, y, c_f) trong đó x, y là đỉnh đầu và đỉnh cuối của cung, c_f là dư lượng trên cung. Ta không duy trì thông tin về sức chứa và luồng trên cung.

Với luồng khởi tạo là luồng 0, dư lượng của các cung được cho bằng chính sức chứa cung đó và dư lượng trên các cung đối ta thêm vào được khởi tạo bằng 0. Trong thuật toán, khi luồng $f(e)$ trên cung e tăng lên bao nhiêu thì $c_f(e)$ $(= c(e) - f(e))$ sẽ được duy trì bằng cách giảm đi bấy nhiêu.

Kết thúc thuật toán, với mỗi cung e trên mạng ban đầu, sức chứa và luồng dương trên cung đó sẽ được suy ra như sau:

Tổng dư lượng 2 cung đối nhau bằng sức chứa của một cung đã cho ban đầu:

$$\begin{aligned} c_f(e) + c_f(-e) &= c(e) - f(e) + \underbrace{c(-e)}_{=0} - f(-e) \\ &= c(e) - \underbrace{\big(f(e) + f(-e)\big)}_{=0} \\ &= \boxed{c(e)} \end{aligned}$$

Dư lượng của cung giả bằng luồng dương trên cung đã cho tương ứng:

$$\begin{aligned} c_f(-e) &= \underbrace{c(-e)}_{=0} - f(-e) \\ &= -f(-e) \\ &= \boxed{f(e)} \end{aligned}$$

❋ Lưu vết và dò đường tăng luồng

Tại mỗi bước, thuật toán BFS được dùng 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, giảm dư lượng các cung này đi Δ_P đồng thời tăng dư lượng các cung đối lên Δ_P.

Edmonds và Karp [25] 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.CPP ✓ Thuật toán Edmonds-Karp

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
const int maxN = 1e5;
const int maxM = 1e5;
const int maxC = 1e4;
struct TEdge //Cấu trúc một cung
{
    int x, y; //Hai đỉnh đầu mút
    int cf; //dư lượng
};

int n, m, s, t, FlowVal;
TEdge e[2 * maxM];
vector<int> L[maxN]; //Danh sách liên thuộc
int trace[maxN];

void Enter()
{
    cin >> n >> m >> s >> t;
    for (int i = 0; i < m; i++)
    {
        int u, v, cap;
        cin >> u >> v >> cap; //Đọc một cung (u, v) với sức chứa cap, luồng khởi tạo = 0
        e[2 * i] = {u, v, cap}; //Thêm cung e[2 * i] = (u, v) với dư lượng cap
        L[u].push_back(2 * i); //Đưa cung 2 * i vào danh sách liên thuộc của u
        e[2 * i + 1] = {v, u, 0}; //Thêm cung đối e[2 * i + 1] = (v, u) với dư lượng 0
        L[v].push_back(2 * i + 1); //Đưa cung 2 * i vào danh sách liên thuộc của v
    }
}

bool FindPath() //Tìm đường tăng luồng bằng BFS
{
    fill(trace, trace + n, -1); //trace[.] ≠ -1: đỉnh chưa thăm
    trace[s] = -2; //s đã thăm, trace[s] = -2 để tiện truy vết
    queue<int> Q;
    Q.push(s);
    do //Thuật toán BFS
    {
        int u = Q.front(), v;
        Q.pop();
        for (int i: L[u]) //Quét các cung e[i] ra khỏi u
            if (v = e[i].y, e[i].cf > 0 && trace[v] == -1) //Gặp cung dư (u, v), v chưa thăm
            {
                trace[v] = i; //Lưu vết
                if (v == t) return true; //Nếu v = t, tìm ra đường tăng luồng
                Q.push(v); //Đẩy v vào hàng đợi chờ xử lý
            }
    }
    while (!Q.empty());
    return false; //Không còn đường tăng luồng
}

void AugmentFlow() //Tăng luồng dọc đường tăng luồng
{
    int u, i;
    int Delta = maxC;
    //Tìm Delta = dư lượng nhỏ nhất của các cung trên đường tăng luồng
    for (u = t; (i = trace[u]) >= 0; u = e[i].x)
        Delta = min(Delta, e[i].cf);
```

```
    //Tăng luồng dọc đường tăng luồng
    for (u = t; (i = trace[u]) >= 0; u = e[i].x)
    {
        e[i].cf -= Delta; //Tăng luồng trên cung e[i] lên Delta
        e[i ^ 1].cf += Delta; //Giảm luồng trên cung đối tương ứng đi Delta
    }
    FlowVal += Delta; //Giá trị luồng f được tăng lên Delta
}

void Print() //In kết quả
{
    //Chỉ ra sức chứa và luồng dương trên các cung đã cho chỉ bằng thông tin về dư lượng
    for (int i = 0; i < 2 * m; i += 2) //Cung đã cho mang chỉ số chẵn trong e
        cout << "e[" << i / 2 << "] = "
             << "(" << e[i].x << ", " << e[i].y << "): "
             << "c = " << e[i].cf + e[i ^ 1].cf << ", "
             << "f = " << e[i ^ 1].cf << '\n';
    cout << "Value of flow: " << FlowVal;
}

int main()
{
    Enter(); //Nhập dữ liệu
    FlowVal = 0;
    while (FindPath()) //Thuật toán Ford-Fulkerson
        AugmentFlow();
    Print(); //In kết quả
}
```

29.2.4. 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 ($X \cup Y = V$ và $X \cap Y = \emptyset$). Hai tập X, Y được gọi là hai phía của lát cắt. 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 29-11

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. Bổ đề này rất dễ hiểu về mặt trực quan: Bao nhiêu đơn vị luồng phát ra từ s tới t thì bấy nhiêu đơn vị luồng phải chạy từ X sang Y. Phép chứng minh bằng công thức cũng khá đơn giản:

$$\begin{aligned} f(X, Y) &= f(X, V) - f(X, V - Y) \\ &= f(X, V) - \underbrace{f(X, X)}_{=0} \\ &= f(X, V) \end{aligned}$$

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

Bổ đề 29-12

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

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

Định lý 29-13 (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 dư lượng 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 dư lượng 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 dư và Y là tập các đỉnh còn lại:

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

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 dư bởi nếu không thì tồn tại đường tăng luồng).

Các cung $e \in \{X \to Y\}$ chắc chắn phải là cung bão hòa, bởi nếu có cung dư $e = (u,v) \in \{X \to Y\}$ thì từ s sẽ tới được v bằng một đường 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 $f(X,Y) = c(X,Y)$

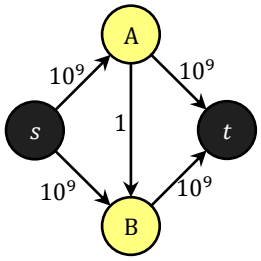
"c⇒a"

Bổ đề 29-11 và Bổ đề 29-12 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.

29.2.5. 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 29-4. Mạng với 4 đỉnh (s phát, t thu), thuật toán Ford-Fulkerson có thể mất 2 tỉ lần tìm đường tăng luồng nếu luân phiên chọn hai đường $\langle s$,A,B,$t\rangle$ và $\langle s$,B,A,$t\rangle$ 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 dư lượng của một mạng G ứng với luồng f nào đó, ta gán trọng số 1 cho các cung 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 dư lượng ứng với luồng f.

Bổ đề 29-14

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 dư lượng 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 luồng 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 dư trên $G_{f'}$ và

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

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 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 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 trong 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 dư lượng không giảm sau mỗi bước tăng luồng.

Bổ đề 29-15

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 bằng Δ_P. Dư lượng Δ_P này theo định nghĩa sẽ phải bằng sức chứa của một cung dư e nào đó trên đường P:

$$\exists e \in P: \Delta_P = c_f(e)$$

Khi tăng luồng dọc trên đường P thì cung e sẽ trở thành bão hòa. Những cung 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 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 29-14 đã chứng minh rằng khoảng cách từ s tới v trên mạng dư lượng 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 dư lượng 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 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 29-16

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 29-15 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 sức chứa 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.

29.3. Thuật toán Dinic

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.

Dinitz phát kiến ra một kỹ thuật mới gọi là *khóa luồng* (*blocking flow*) [26], một kỹ thuật rất tốt để cải thiện tốc độ của phương pháp Ford-Fulkerson. Kỹ thuật này được ông tìm ra trong một bài tập của lớp học thuật toán và vào thời điểm đó ông không hề biết rằng mình là người đầu tiên tìm ra một thuật toán đa thức cho bài toán luồng cực đại*.

Ý tưởng của thuật toán Dinic là tại mỗi bước, xây dựng một cơ chế phân tầng các đỉnh dựa trên độ dài đường đi ngắn nhất tới đỉnh thu trên mạng dư lượng, sau đó tìm một luồng trên mạng dư lượng để hợp vào luồng chính. Khác với thuật toán Edmonds-Karp (trong đó luồng trên mạng dư lượng chỉ dựa vào 1 đường tăng luồng), thuật toán Dinic tăng luồng theo tất cả các đường tăng luồng ngắn nhất bằng cách "khóa luồng" hay "chặn dòng": chỉ cho phép luồng đổ từ đỉnh tầng trên xuống đỉnh tầng liền dưới.

29.3.1. Mạng phân tầng và cơ chế tăng luồng

Cho f là một luồng trên mạng $G = (V, E, c, s, t)$. Với mỗi đỉnh u, gọi $h[u]$ là độ dài đường dư ngắn nhất (tính bằng số cung) từ u tới t trên mạng dư lượng. Nếu u không tới được t bằng đường dư, ta coi như $h[u] = +\infty$ ứng với việc đỉnh u không được phân tầng.

Việc tính các giá trị $h[.]$ có thể thực hiện bằng thuật toán BFS với đỉnh xuất phát là t, đi ngược chiều các cung. Nhãn $h[u]$ được gọi là tầng của u.

Nếu s không đến được t (s không được phân tầng) thì f là luồng cực đại. Nếu không, thuật toán lần lượt tìm các đường tăng luồng ngắn nhất từ s tới t (qua đúng $h[s]$ cung dư) và tăng luồng dọc trên các đường tăng luồng này. Pha này được xử lý khéo léo bằng thuật toán DFS trên mạng dư lượng đã phân tầng.

Khi không còn đường tăng luồng độ dài $h[s]$ nữa, thuật toán Dinic sẽ phân tầng lại và tiếp tục quá trình tương tự…

Trong khâu tìm đường đi từ s tới t trên mạng đã phân tầng, những cung dư nối từ một tầng tới tầng liền dưới được gọi là cung được thừa nhận (*admissible)*, những cung khác

---

* Thuật toán của Dinitz được công bố năm 1970 trên một tạp chí bằng tiếng Nga. Năm 1974, Shimon Even và Alon Itai đọc được bài báo và bị hấp dẫn với ý tưởng này nên đưa vào một bài giảng "Thuật toán Dinic" nhưng lại đánh máy sai tên tác giả khi phổ biến nó. Tuy nhiên hai tác giả này cũng góp phần vào thuật toán bằng cách kết hợp giữa BFS và DFS trở thành phiên bản hiện tại của thuật toán này. Hiện nay ta có thể gọi dưới tên Dinitz hay Dinic đều được. Mặc dù năm 2006 Dinitz có viết một bài báo phân định rõ ràng ý tưởng của mình và những bổ sung của Even nhưng thuật toán vẫn được gọi dưới cái tên Dinic.

Thuật toán Dinic (1970) có trước thuật toán Edmonds-Karp (1972), mục đích của Dinitz chỉ là tìm một thuật toán đa thức cho phương pháp Ford-Fulkerson chứ không phải cải tiến thuật toán Edmonds-Karp. Người ta nhận ra kỹ thuật khóa luồng có thể tăng tốc thuật toán Edmons-Karp chính là qua bài giảng của Even (1974).

được coi là bị khóa (*blocked*). Ý tưởng của thuật toán Dinic sẽ được trình bày kỹ hơn trong phần cài đặt

29.3.2. Cài đặt

Thuật toán Dinic sẽ được cài đặt với khuôn dạng Input/Output như trong chương trình cài đặt thuật toán Edmonds-Karp.

Việc tổ chức dữ liệu không có nhiều khác biệt: Với mỗi đỉnh u, ta gắn với nó một danh sách $L[u]$ chứa chỉ số các cung đi ra khỏi u trên mạng dư lượng.

❋ Kỹ thuật phân tầng

Ta cần tính các giá trị $h[v]$ là độ dài đường dư ngắn nhất từ v tới t (tính bằng số cạnh), những đỉnh không có đường dư đến v có $h[.] = n$. Điều này có thể thực hiện bằng thuật toán BFS từ t với các cung của mạng dư lượng đảo chiều.

Tuy nhiên khi cài đặt ta không cần đảo chiều các cung, với mỗi đỉnh u, để quét tất cả các cung đi vào u, ta có thể quét danh sách $L[u]$ để duyệt các cung e ra khỏi u, rồi thay vì xử lý cung e, ta sẽ xử lý cung $-e$ là cung đi vào u.

❋ Tăng luồng dọc các đường tăng luồng ngắn nhất

Dễ thấy rằng một đường tăng luồng ngắn nhất (độ dài $h[s]$) sẽ chỉ đi qua các cung được thừa nhận (là những cung dư (u, v) có $h[u] = h[v] + 1$).

Phép tăng luồng dọc các đường tăng luồng ngắn nhất được thực hiện bằng thuật toán tìm kiếm theo chiều sâu (DFS), ở đây là mô hình DFS theo cung chứ không phải DFS theo đỉnh: Bắt đầu với các cung dư nối từ s (ở tầng $h[s]$) sang tầng $h[s] - 1$, thuật toán duyệt từ một cung sang cung dư nối tiếp ở tầng dưới cho tới khi tới được t, trong quá trình tiến sâu từ tầng $h[s]$ xuống tầng 0, thuật toán DFS duy trì dư lượng nhỏ nhất của các cung đi qua để tăng luồng dọc theo đường tăng luồng, khi quá trình đệ quy của DFS lùi về, những cung trở thành bão hòa sẽ bị loại trước khi chuẩn bị tiến sâu theo một hướng khác nhằm tìm đường dư tới t. Chi tiết kỹ thuật được giải thích như sau:

Ta sẽ viết một hàm $DFS(u, flowIn)$ với ý nghĩa là nếu đổ một lượng luồng $flowIn$ vào đỉnh u sau đó phát tán qua các đỉnh khác ở tầng dưới thì lượng luồng đó phát tán tối đa tới t được bao nhiêu. Lượng luồng này được trả về trong kết quả hàm. Lượng luồng không phát tán hết sẽ dồn trả về u khi quá trình đệ quy của DFS lùi về. Phép tăng luồng lần lượt theo tất cả các đường tăng luồng độ dài ngắn nhất sẽ thực hiện bằng lời gọi $DFS(s, +\infty)$.

Hàm $DFS(u, flowIn)$ thực hiện như sau:

Nếu $u = t$, hàm trả về $flowIn$ (luồng phát tán hết tới t).

Nếu $u \neq t$, xét tất cả các cung dư (u, v) mà $h[u] = h[v] + 1$:

- Trên mạng dư lượng, thử đẩy một lượng luồng tối đa $\min\{flowIn, c_f(u, v)\}$ theo cung (u, v): Tính $q = DFS(v, \min\{flowIn, c_f(u, v)\})$ là lượng luồng phát tán theo cung (u, v) mà tới được t.
- Đẩy thêm q đơn vị luồng qua cung (u, v) trên mạng dư lượng, q đơn vị luồng này sẽ phát tán hết tới t. Dĩ nhiên ta phải giảm luồng trên cung đối (v, u) đi q để đảm bảo tính phản đối xứng.
- Tính lại $flowIn$ xem lượng luồng còn lại cần phát tán là bao nhiêu trước khi xét một cung (u, v') khác để phát tán theo hướng mới.

Chú ý rằng việc đẩy thêm một lượng luồng q qua cung trên mạng dư lượng sẽ làm dư lượng cung đó giảm đi q và làm dư lượng cung đối tăng lên q, điều này ta đã phân tích trong phần cài đặt thuật toán Edmonds-Karp: Chỉ duy trì thông tin về dư lượng, những thông tin về sức chứa và luồng trên các cung sẽ được suy ra khi in kết quả.

❋ Con trỏ tới cung còn hiệu lực

Để thuật toán hiệu quả hơn, mỗi đỉnh u sẽ được kèm theo một con trỏ $current[u]$ tới cung đang xét trong danh sách $L[u]$. Ngay sau khi phân tầng các đỉnh và trước khi thực hiện phát tán luồng từ s, con trỏ $current[u]$ được đặt vào đầu danh sách $L[u]$. Trong thuật toán DFS, khi xét danh sách các cung đi ra khỏi u ta sẽ xét con trỏ $current[u]$, nếu cung này là cung bão hòa hay cung không nối tới đỉnh ở tầng liền dưới, con trỏ $current[u]$ sẽ được dịch sang cung kế tiếp. Bởi thuật toán DFS ở đây có thể thăm một đỉnh nhiều lần, con trỏ $current[.]$ cho phép ta không cần phải duyệt lại từ đầu danh sách liên thuộc chứa những cung không còn ý nghĩa nữa.

DINIC.CPP ✓

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
const int maxN = 1e5;
const int maxM = 1e5;
const int maxC = 1e4;
const int inftyFlow = maxM * maxC;
struct TEdge //Cấu trúc một cung
{
    int x, y; //Hai đỉnh đầu mút
    int cf; //dư lượng
};
int n, m, s, t;
TEdge e[2 * maxM];
vector<int> L[maxN]; //Danh sách liên thuộc
vector<int>::iterator current[maxN]; //con trỏ tới cung đang xét trong danh sách liên thuộc
int h[maxN];
int FlowVal;

void Enter()
{
```

```
    cin >> n >> m >> s >> t;
    for (int i = 0; i < m; i++)
    {
        int u, v, cap;
        cin >> u >> v >> cap; //Đọc một cung (u, v) với sức chứa cap, luồng khởi tạo = 0
        e[2 * i] = {u, v, cap}; //Thêm cung e[2 * i] = (u, v) với dư lượng cap
        L[u].push_back(2 * i); //Đưa cung 2 * i vào danh sách liên thuộc của u
        e[2 * i + 1] = {v, u, 0}; //Thêm cung đối e[2 * i + 1] = (v, u) với dư lượng 0
        L[v].push_back(2 * i + 1); //Đưa cung 2 * i + 1 vào danh sách liên thuộc của v
    }
}

void BFS() //Thuật toán phân tầng bằng BFS
{
    queue<int> q;
    fill(h, h + n, n); //Các đỉnh đều có nhãn h[.] = n
    h[t] = 0; //Riêng đỉnh thu có h[t] = 0
    q.push(t); //hàng đợi dùng cho BFS
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (int i: L[u]) //Quét các cung e[i] = (u, v)
        {
            i ^= 1; //Lấy cung đối (v, u)
            int v = e[i].x;
            if (e[i].cf == 0 || h[v] < n) //gặp cung bão hòa hoặc đỉnh đã phân tầng
                continue; //bỏ qua
            h[v] = h[u] + 1; //gán nhãn tầng cho đỉnh v, cũng là đánh dấu h[v] < n
            if (v == s) return; //s đã được phân tầng, dừng ngay
            q.push(v);
        }
    }
}

//Nếu phát tán flowIn luồng từ u thì tới t tối đa được bao nhiêu?
int DFS(int u, int flowIn) //Thuật toán DFS trên mạng phân tầng
{
    if (u == t) return flowIn; //u == t coi như phát tán hết
    int flowOut = 0;
    for (; current[u] != L[u].end(); ++current[u]) //Xét các cung từ current[u] trở đi
    {
        int i = *current[u];
        int v = e[i].y;
        if (e[i].cf == 0 || h[v] != h[u] - 1)
            continue; //Gặp cung bão hòa hoặc cung không sang tầng liền dưới, bỏ qua
        int q =  DFS(v, min(flowIn, e[i].cf));
        flowOut += q;
        flowIn -= q;
        e[i].cf -= q; //Đẩy lượng luồng q qua cung e[i] để phát tán hết tới t
        e[i ^ 1].cf += q; //Giảm luồng trên cung đối tương ứng
        if (flowIn == 0) break; //Phát tán hết, dừng
    }
    return flowOut; //Trả về lượng luồng phát được tới t
}

void BlockingFlow() //Tăng luồng theo tất cả đường tăng luồng ngắn nhất
{
    for (int u = 0; u < n; ++u)
        current[u] = L[u].begin(); //Đặt con trỏ current vào đầu danh sách L[u]
    FlowVal += DFS(s, inftyFlow); //Thử phát tán lượng luồng +∞ từ s
}
```

```
void Print() //In kết quả, không khác gì trước
{
    for (int i = 0; i < 2 * m; i += 2) //Chỉ in luồng dương trên các cung đã cho
        cout << "e[" << i / 2 << "] = "
             << "(" << e[i].x << ", " << e[i].y << "): "
             << "c = " << e[i].cf + e[i ^ 1].cf << ", "
             << "f = " << e[i ^ 1].cf << '\n';
    cout << "Value of flow: " << FlowVal;
}

int main()
{
    Enter(); //Nhập dữ liệu
    FlowVal = 0;
    while (BFS(), h[s] < n) //Chừng nào từ s có đường tăng luồng tới t: h[s] < n
        BlockingFlow();
    Print(); //In kết quả
}
```

29.3.3. Thời gian thực hiện giải thuật

Xét vòng lặp chính của thuật toán, tại mỗi bước lặp nếu độ dài đường dư ngắn nhất từ s tới t là $h[s]$ thì sau khi gọi hàm $BlockingFlow()$ sẽ không còn đường tăng luồng độ dài $h[s]$ trên mạng dư lượng nữa. Tương tự như trong chứng minh thuật toán Edmonds-Karp, điều này cho thấy giá trị $h[s]$ tăng ngặt sau mỗi lượt lặp, và vì $h[s] \leq |V| - 1$, số bước lặp là một đại lượng $O(|V|)$.

Trong mỗi bước lặp, thuật toán BFS có thời gian thực hiện $O(|E|)$.

Xét số lần gọi hàm DFS trong pha $BlockingFlow()$, giả sử hàm $DFS(u, FlowIn)$ gọi hàm $DFS(v, .)$ sau đó tính lại $FlowIn$ mới...

- Nếu giá trị $FlowIn$ trở thành 0, hàm $DFS(u, FlowIn)$ thoát, quá trình DFS lùi lại.
- Nếu giá trị $FlowIn$ vẫn > 0, con trỏ $current[u]$ dịch đi một vị trí đồng nghĩa với việc cung (u, v) không bao giờ bị xét đến nữa.

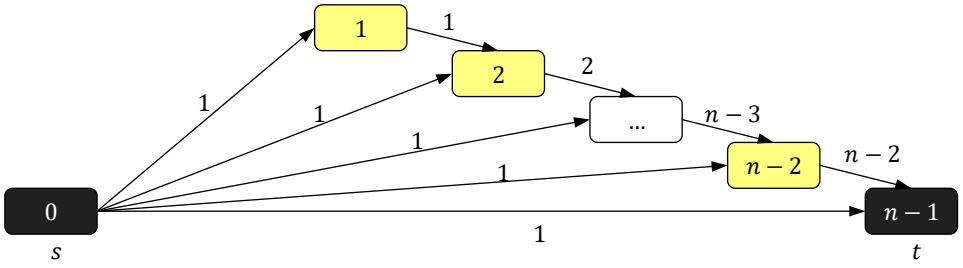
Quá trình DFS tìm đường đi từ s tới t qua tất cả các tầng của mạng từ tầng $h[s]$ tới tầng 0. Vì vậy dây chuyền đệ quy không thể lùi về liên tục quá $h[s]$ bước, tức là trong $h[s]$ lần hàm $DFS(.,.)$ thoát, chắc chắn có một cung bị loại. Số cung là một đại lượng $O(|E|)$ suy ra số lần thoát hàm $DFS(.,.)$ (cũng là số lần gọi hàm) là $O(|E|.(h[s])) = O(|E||V|)$.

Còn lại là tổng thời gian duyệt danh sách liên thuộc dựa vào con trỏ $current[.]$, mỗi lần con trỏ $current[.]$ được xét sẽ là một lời gọi hàm DFS hoặc một phép dịch con trỏ $current$: Vậy số lần xét con trỏ $current[.]$ sẽ là $O(|E||V| + |E|) = O(|E||V|)$

Vậy hàm $BlockingFlow()$ thực hiện trong thời gian $O(|V||E|)$ và thuật toán Dinic có thời gian thực hiện là $O(|V|^2|E|)$.

29.4. Thuật toán đẩy tiền luồng

Mặc dù thuật toán Dinic có tốc độ tốt trong trường hợp các cung của mạng được phân bố ngẫu nhiên, cũng có trường hợp xấu khiến cho tại mỗi pha $BlockingFlow()$ thuật toán chỉ tìm được một đường tăng luồng duy nhất, nghĩa là cách hoạt động của thuật toán Dinic không khác gì thuật toán Edmonds-Karp, lại mất thêm thời gian xây dựng mạng phân tầng.



Hình 29-5. Ví dụ xấu của thuật toán Dinic, số ghi trên các cung là sức chứa, cũng là luồng trên cung ứng với luồng cực đại

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.

29.4.1. 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 phản đối xứng (*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: $\forall u \in V - \{s\}: f(V, u) \geq 0$*.

Tổng tiền luồng trên các cung đi vào đỉnh u ($f(V, u)$) được gọi là lượng tràn tại u, ký hiệu $excess[u]$†. Đỉnh $v \in V - \{s, t\}$ gọi là *đỉnh tràn (overflowing vertex)* nếu $excess[v] > 0$.

* Nếu phát biểu theo khái niệm tiền luồng dương thì ràng buộc về tính dư này tương đương với: Tổng tiền luồng dương đi vào u lớn hơn hoặc bằng tổng tiền luồng dương ra khỏi u ($\varphi(V, u) \geq \varphi(u, V)$)

† Nếu phát biểu theo khái niệm tiền luồng dương thì $excess[u]$ là hiệu số: tổng tiền luồng dương đổ vào u trừ tổng tiền luồng dương thoát khỏi u.

Khái niệm đỉnh tràn chỉ có nghĩa với các đỉnh không phải đỉnh phát cũng không phải đỉnh thu.

```
bool Overflow(u∈V)
{
    return u != s && u != t && excess[u] > 0;
}
```

Chú ý rằng một luồng bất kỳ luôn là tiền luồng nhưng một tiền luồng chỉ là luồng khi trên mạng không có đỉnh tràn. Ta cũng có khái niệm tiền luồng dương, mạng dư lượng, cung dư, đường dư... ứng với tiền luồng tương tự như định nghĩa đối với luồng.

29.4.2. 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 gọi là **ràng buộc độ cao**:

$$\begin{cases} h(s) = |V| \\ h(t) = 0 \\ h(u) \le h(v) + 1, \text{với mọi cung dư } (u, v) \in E_f \end{cases}$$

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 $h[v]$ khi 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 phản đối xứng) còn tiền luồng trên các cung khác bằng 0. Khi đó tất cả các cung đi ra khỏi s là bão hòa.

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

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

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

Rõ ràng mọi cung dư (u, v) không thể là cung đi ra khỏi s $(u \ne s)$ nên ta có $h(u) \le 1 \le h(v) + 1$. 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.

```
void Init()
{
    //Khởi tạo tiền luồng
    for (∀e ∈ E) f[e] = 0;
    for (∀v ∈ V) excess[v] = 0;
    for (∀e = (u, v) ∈ E(s, V))
    {
        f[e] = c(e);
        f[-e] = -c(e);
        excess[v] += f[e];
    }
    //Khởi tạo hàm độ cao
    for (∀v ∈ V) h[v] = 1;
    h[s] = |V|; h[t] = 0;
}
```

29.4.3. Phép đẩy luồng

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

- u là đỉnh tràn: $u \in V - \{s, t\}$ và $excess[u] > 0$
- e là cung dư: $c_f(e) > 0$
- 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 $Push\big(e = (u, v)\big)$ tính $\Delta = \min\{excess[u], c_f(e)\}$, đây là lượng luồng có thể lấy từ lượng tràn tài u mà có thể đẩy qua cung e, thêm lượng luồng này vào cung e và bớt một lượng luồng Δ từ v về u theo cung $-e$ để giữ tính phản đối xứng của tiền luồng. Có thể hiểu là phép $Push(e)$ tháo bớt một lượng tối đa luồng tràn tại u chảy qua cung e. Phép $Push\big(e = (u, v)\big)$ sai đó cập nhật lại $excess[u]$ và $excess[v]$ theo tiền luồng mới.

```
void Push(e = (u,v))
{
    Δ = min(excess[u], cf(e)); //Tính lượng luồng tối đa có thể đẩy
    f[e] += Δ; //Đẩy Δ luồng thêm vào cung e
    f[-e] -= Δ; //Giữ tính phản đối xứng của tiền luồn
    excess[u] -= Δ; //Cập nhật mức tràn tại u
    excess[v] += Δ; //Cập nhật mức tràn tại v
}
```

Dễ dàng kiểm tra được các tính chất của tiền luồng vẫn được thỏa mãn sau phép $Push$. Ngoài ra 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 dư $-e = (v, u)$ mà thôi. Phép $Push$ không làm thay đổi các độ cao, tức là trước và sau khi $Push$, $h[u]$ vẫn lớn hơn $h[v]$, ràng buộc độ cao $h[v] \leq h[u] + 1$ vẫn được duy trì trên cung dư $-e = (v, u)$.

Khi phép $Push(e = (u, v))$ đẩy một lượng luồng $\Delta = \min\{excess[u], c_f(e)\}$ tràn từ u sang v:

- Nếu $\Delta = c_f(e)$ thì sau phép $Push(e)$ lẽ làm cho $c_f(e)$ sẽ trở thành 0, ta gọi phép đẩy luồng này là *đẩy bão hòa (saturating push)*,
- Nếu $\Delta < c_f(e)$ tức là $\Delta = excess[u]$, phép đẩy luồng này 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.

29.4.4. 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 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:

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

```
void Lift(u∈V)
{
    h[u] = +∞;
    for (∀(u,v) ∈ Ef)
        h[u] = min(h[u], h[v]);
    ++h[u];
}
```

Nếu u là đỉnh tràn, ta có $excess[u] = f(V, u) > 0$ tức là ít nhất có một cung $e \in \{V \to u\}$ để $f(e) > 0$. Cung đối $-e \in \{u \to V\}$ có sức chứa bằng $c(-e) - f(-e) = c(-e) + f(e) > 0$ nên đây là một cung dư ra khỏi u, điều này đảm bảo cho phép lấy $\min_{\forall v:(u,v)\in E_f} \{h[v]\}$ được thực hiện trên một tập khác rỗng.

Phép $Lift(u)$ không động chạm gì đến tiền luồng f. Ngoài ra phép $Lift(u)$ chỉ tăng độ cao của một đỉnh u và bảo tồn ràng buộc độ cao:

- Với một cung 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.
- Với một cung dư (u, v) đi ra khỏi u thì việc đặt:

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

  cũng đảm bảo rằng $h(u) \leq h(v) + 1$ với mọi cung dư $(u, v) \in E_f$.

29.4.5. 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 đắn 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.

29.4.6. Thuật toán FIFO Preflow-Push

❋ Ý tưởng của thuật toán

Cho mạng $G = (V, E, c, s, t)$ có tiền luồng f và hàm độ cao h. Khi đó 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 $Lift(u)$. Thật vậy, nếu thao tác $Push$ không thể áp dụng được cho cung dư nào đi ra khỏi u tức là với mọi cung 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 $Lift(u)$. Đây chính là cơ sở cho thuật toán do Goldberg đề xuất [27] 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, 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 dư đi ra khỏi z bằng phép $Push$. Những đỉnh tràn mới do các phép $Push$ này tạo ra được đưa vào hàng đợi.
- Nếu không đẩy được hết lượng tràn ($excess[z]$ vẫn > 0) thì ta nâng 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.

❋ Tổ chức dữ liệu

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 một số phép $Push$ nào, chúng ta vẫn không thể đẩy luồng được trên cung e. Thật vậy, nếu u không cao hơn v, thì sau bao nhiêu phép $Push()$, 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 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. Bởi phép $Lift$ không làm thay đổi tiền luồng trên các cung, dư lượng của các cung được giữ nguyên. Nếu e đang bão hòa thì sau phép $Lift$ nó vẫn bão hòa và không thể đẩy luồng được. Nếu e là cung dư thì u đang không cao hơn v, lệnh $Lift$ duy nhất có thể khiến u cao hơn v là lệnh $Lift(u)$.

Nhận xét trên cho phép thiết kế một cấu trúc dữ liệu phù hợp cho thuật toán: Các cung đi ra khỏi u sẽ được lưu trữ trong một danh sách $L[u]$. Ngoài ra có một con trỏ $current[u]$ trỏ tới cung đang xét trong danh sách $L[u]$. Ban đầu $current[u]$ trỏ tới đầu danh sách $L[u]$.

Với mỗi đỉnh tràn u lấy khỏi hàng đợi, ta xét cung ứng với con trỏ $current[u]$. Chừng nào phép $Push()$ không thể thực hiện được trên cung đó ta dịch con trỏ $current[u]$ đi một vị trí để xét cung kế tiếp. Cho tới khi:

- Hoặc ta xét tới một cung (u, v) có thể thực hiện được phép $Push()$, phép $Push()$ đó được thực hiện ngay và đẩy v vào hàng đợi nếu v là đỉnh tràn chưa có trong hàng đợi.
- Hoặc con trỏ $current[u]$ đã duyệt hết danh sách $L[u]$ mà u vẫn tràn, đỉnh u sẽ được nâng lên bằng phép $Lift(u)$ và con trỏ $current[u]$ được đặt lại vào đầu danh sách $L[u]$, đỉnh u sau đó được đẩy lại vào hàng đợi chờ xử lý sau…

29.4.7. Tính đúng của thuật toán

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

$$f(u, V) = -f(V, u) = -excess[u] = 0$$

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

Bổ đề 29-17

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 dư lượng 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 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 dư lượng G_f đi qua k cung dư. Khi đó:

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

hay:

$$h(s) \leq h(t) + k$$

Nhưng ta biết rằng $h(s) = |V|, h(t) = 0$ và $k < |V|$ do đường tăng luồng phải là đường đi đơn, sự mâu thuẫn này cho thấy không thể tồn tại đường tăng luồng trên G_f.

Bổ đề 29-17 và Định lý 29-13 (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.

29.4.8. 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. 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.

Bổ đề 29-18

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 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 dư và $Y = V - X$.

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 dư $-e$ nối từ X sang Y. Trái với cách xây dựng hai tập X, Y.

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 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. Hơn nữa u phải là đỉnh tràn trước và sau phép $Lift(u)$. Bổ đề 29-18 cho biết tồn tại đường đi đơn từ u tới s, $\langle u = v_0, v_1, \dots, v_k = s \rangle$, chỉ đi qua các cung dư. Từ ràng buộc độ cao ta có:

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

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

Định lý 29-19 (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

Để chứng minh mô hình cài đặt thuật toán FIFO Preflow-Push ở trên có thời gian thực hiện là $O(|V|^3 + |V||E|)$, có thể coi thuật toán gồm một dãy các phép $Lift$ và $Push$:

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

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 đơn vị, độ cao của mỗi đỉnh không vượt quá $2|V| - 1$ (theo hệ quả của Bổ đề 29-18). 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 $(u, .)$ 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 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 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 (hoặc bằng) 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 con trỏ $current[.]$. Với mỗi đỉnh u, con trỏ $current[z]$ ban đầu được đặt vào đầu danh sách $L[u]$ chuyển dần về phía cuối danh sách. Khi duyệt hết danh sách liên thuộc mà u vẫn tràn thì sẽ có một phép $Lift(u)$ được thực hiện và con trỏ $current[z]$ được đặt lại về đầu danh sách $L[u]$. Số phép $Lift(u)$ trong toàn bộ giải thuật không vượt quá $2|V| - 1$, nên số lượt dịch con trỏ $current[u]$ không vượt quá $2|V||L[u]|$. Suy ra nếu xét tổng thể, số phép dịch các con trỏ $current[.]$ trên tất cả các danh sách liên thuộc phải nhỏ hơn:

$$(2|V|)\sum_{u \in V}|L[u]| = 2|V||E| = O(|V||E|)$$

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 con trỏ $current[.]$: $O(|V||E|)$.

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

29.4.9. Mẹo giải tăng tốc độ chương trình

Ta đã chứng minh 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 ($O(|V||E|^2)$) và thuật toán Dinic ($O(|V|^2|E|)$).

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... sẽ thực hiện rất chậm 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 dư (u, v) thì $h(u) \leq h(v) + 1$.

Ký hiệu $\delta(u, v)$ là độ dài đường dư ngắn nhất (tính bằng số cung) từ u tới v. Nếu không tồn tại đường dư $u \rightsquigarrow v$ ta coi như $\delta(u, v) = +\infty$.

Nếu $k = \delta(u, v)$ và $\langle u = x_0, x_1, \dots, x_k = v \rangle$ là đường dư ngắn nhất $u \rightsquigarrow v$. Từ ràng buộc độ cao ta có:

$$h(u) = h(x_0) \leq h(x_1) + 1 \leq h(x_2) + 2 \leq \cdots \leq h(x_k) + k = h(v) + k$$

Vậy với mọi đỉnh u:

- Nếu u tới được t bởi một đường dư thì $h(u) \leq h(t) + \delta(u, t) = \delta(u, t)$
- Nếu u tới được s bởi một đường dư thì $h(u) \leq h(s) + \delta(u, s) = |V| + \delta(u, s)$

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 dư và tập X gồm những đỉnh còn lại. Bởi không thể có cung dư nối từ X sang Y, ta có $s \in X, t \in Y$. Sau đó:

- Với $\forall u \in Y$ đặt lại độ cao $h[u] = \delta(u, t)$
- Với $\forall u \in X$ đến được s bằng một đường dư, đặt lại độ cao $h[u] = |V| + \delta(u, s)$
- Với những đỉnh còn lại $\in X$ không đến được s bằng một đường dư, độ cao của chúng được gán bằng $2|V| - 1$ (giá trị lớn nhất của hàm độ cao). Thực ra những đỉnh này có thể loại bỏ luôn khỏi thuật toán.

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 $\delta(v, t)$ cũng như $\delta(u, s)$ 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 $h[u]$ của một đỉnh u nào đó, cần cập nhật lại $current[u]$ trỏ vào đầu danh sách $L[u]$.

Để ý rằng nếu như ta không cài đặt phép $Lift()$ mà thay vào đó, mỗi khi thấy không thể thực hiện được bất kỳ phép $Push()$ thì gán nhãn lại toàn bộ độ cao theo thuật toán trên. Khi đó thuật toán FIFO Preflow-push có cách thức làm việc khá giống thuật toán Dinic với độ phức tạp $O(|V|^2|E|)$.

❋ 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 gap mà không đỉnh nào có độ cao gap (số nguyên gap này được gọi là "khe"), khi đó từ mọi đỉnh z có $h[z] > gap$ đều không có đường dư đi đến t. Nhận định này có thể chứng minh bằng phản chứng: Giả sử từ z có đường 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 dư từ z 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 $h[z] = |V| + 1$. Có thể chỉ ra rằng ràng buộc độ cao vẫn được đảm bảo sau phép đặt này.

Mặc dù mẹo đẩy nhãn theo khe này rất tốt trên các bộ dữ liệu ngẫu nhiên, vẫn có những trường hợp xấu khiến cho mẹo giải này trở nên không hiệu quả. Vì vậy ta chỉ nên sử dụng nó khi kết hợp cùng với phép gán nhãn lại toàn bộ. Khi cần quan tâm tới tính đơn giản của chương trình, phép gán nhãn lại toàn bộ có thể coi là đủ tốt cho thuật toán FIFO Preflow-push nên không cần cài đặt thêm phép đẩy nhãn theo khe nữa.

29.4.10. Cài đặt

Dưới đây là chương trình cài đặt thuật toán FIFO Preflow-Push kết hợp với kỹ thuật gán nhãn lại toàn bộ, việc cài đặt và đánh giá hiệu suất của phép đẩy nhãn theo khe 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. Tương tự như trong cài đặt thuật toán Edmonds-Karp và thuật toán Dinic, ta sẽ chỉ duy trì dư lượng trên các cung.

FIFOPREFLOWPUSH.cpp ✓ Thuật toán đẩy tiền luồng

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
const int maxN = 1e5;
const int maxM = 1e5;
const int maxC = 1e4;
const int inftyFlow = maxM * maxC;
struct TEdge //Cấu trúc một cung
```

```
{
    int x, y; //Hai đỉnh đầu mút
    int cf; //dư lượng
};
int n, m, s, t;
TEdge e[2 * maxM];
vector<int> L[maxN]; //Danh sách liên thuộc
vector<int>::iterator current[maxN]; //con trỏ tới cung đang xét trong danh sách liên thuộc
int h[maxN]; //độ cao các đỉnh
int excess[maxN]; //lượng tràn ở các đỉnh
queue<int> OverflowQueue; //hàng đợi chứa các đỉnh tràn
bool InQueue[maxN]; //mảng đánh dấu một đỉnh có trong OverflowQueue không
int FlowVal;


void Enter()
{
    cin >> n >> m >> s >> t;
    for (int i = 0; i < m; i++)
    {
        int u, v, cap;
        cin >> u >> v >> cap; //Đọc một cung (u, v) với sức chứa cap, luồng khởi tạo = 0
        e[2 * i] = {u, v, cap}; //Thêm cung e[2 * i] = (u, v) với dư lượng cap
        L[u].push_back(2 * i); //Đưa cung 2 * i vào danh sách liên thuộc của u
        e[2 * i + 1] = {v, u, 0}; //Thêm cung đối e[2 * i + 1] = (v, u) với dư lượng 0
        L[v].push_back(2 * i + 1); //Đưa cung 2 * i vào danh sách liên thuộc của v
    }
}

void BFS(int Finish) //BFS để đo h[v] = độ dài đường dư ngắn nhất từ v tới Finish
{
    queue<int> q; //Hàng đợi dùng cho BFS
    q.push(Finish);
    while (!q.empty())
    {
        int u, v;
        u = q.front(); q.pop(); //Lấy u khỏi q
        for (int i: L[u]) //Xét các cung e[i] = (u, v) có cung đối dư và h[v] == 2 * n - 1
            if (v = e[i].y, e[i ^ 1].cf > 0 && h[v] == 2 * n - 1)
            {
                h[v] = h[u] + 1; //Đặt nhãn h[v]
                q.push(v); //Xếp v vào q
            }
    }
}

void GlobalRelabel() //Thuật toán gán nhãn lại toàn bộ
{
    fill(h, h + n, 2 * n - 1); //Các đỉnh đều được khởi tạo nhãn 2 * n - 1
    h[t] = 0;
    BFS(t); //BFS từ t: h[v] = độ dài đường dư ngắn nhất từ v tới t
    h[s] = n;
    BFS(s); //BFS từ s: h[v] = n + độ dài đường đi ngắn nhất từ v tới s
    for (int u = 0; u < n; ++u) //Chỉnh lại các con trỏ current[.]
        current[u] = L[u].begin();
}

inline bool Overflow(int u) //u có phải đỉnh tràn không?
{
    return u != s && u != t && excess[u] > 0;
}

void Init() //Khởi tạo tiền luồng
```

```
{
    fill(excess, excess + n, 0);
    for (int i: L[s]) //Xét các cung e[i] = (s, .)
    {
        int fval = e[i].cf; //Đặt luồng fval lên cung e[i] và -fval lên cung đối
        e[i].cf -= fval;  //e[i] trở thành bão hòa, e[i].cf = 0
        e[i ^ 1].cf += fval;
        excess[s] -= fval; //excess[e[i].x] giảm đi fval
        excess[e[i].y] += fval; //excess[e[i].y] tăng lên fval
    }
    GlobalRelabel(); //Dùng GlobalRelabel() khởi tạo độ cao luôn
    //Xây dựng OverflowQueue chứa các đỉnh tràn
    fill(InQueue, InQueue + n, false);
    for (int u = 0; u < n; ++u)
        if (Overflow(u))
        {
            OverflowQueue.push(u);
            InQueue[u] = true;
        }
}

void Push(int i) //Phép đẩy luồng trên cung e[i] = (u, v)
{
    int u = e[i].x, v = e[i].y;
    int Delta = min(excess[u], e[i].cf); //Lượng luồng tối đa có thể đẩy
    e[i].cf -= Delta; //Tăng luồng trên e[i] lên Delta
    e[i ^ 1].cf += Delta; //Giảm luồng trên cung đối đi Delta
    excess[u] -= Delta; //Mức tràn tại u giảm Delta
    excess[v] += Delta; //Mức tràn tại v tăng Delta
}

void Lift(int u) //Phép nâng
{
    h[u] = 2 * n - 1;
    for (int i: L[u])
        if (e[i].cf > 0) h[u] = min(h[u], h[e[i].y]);
    ++h[u];
}

inline int PopFromQueue() //Lấy một đỉnh tràn khỏi OverflowQueue, trả về trong kết quả hàm
{
    int u = OverflowQueue.front();
    OverflowQueue.pop();
    InQueue[u] = false;
    return u;
}

inline void PushToQueue(int u) //Đẩy một đỉnh tràn vào OverflowQueue
{
    if (InQueue[u]) return; //u đã có trong hàng đợi, bỏ qua
    OverflowQueue.push(u);
    InQueue[u] = true;
}

void FifoPreflowPush() //Thuật toán FIFO Preflow-Push
{
    int LiftCnt = 0;
    while (!OverflowQueue.empty())
    {
        int u = PopFromQueue(); //Lấy một đỉnh tràn u khỏi hàng đợi
        for (; excess[u] > 0 && current[u] != L[u].end(); ++current[u])
        {
```

```
            int i = *current[u]; //Xét cung e[i] ứng với con trỏ current[u]
            int v = e[i].y;
            if (e[i].cf > 0 && h[u] > h[v]) //Phép Push() thực hiện được trên cung e[i]
            {
                Push(i); //Thực hiện luôn
                if (Overflow(v)) PushToQueue(v); //Đẩy v vào hàng đợi nếu v là đỉnh tràn mới
            }
        }
        if (excess[u] > 0) //Sau khi xét hết L[u] mà u vẫn tràn.
        {
            if (LiftCnt == n) //Số lần Lift đã đạt n lần
            {
                GlobalRelabel(); //Gán nhãn lại toàn bộ
                LiftCnt = 0; //Khởi tạo lại bộ đếm
            }
            else
            {
                Lift(u); //Thực hiện phép Lift(u)
                ++LiftCnt; //Đếm số phép Lift()
                current[u] = L[u].begin(); //Đặt lại con trỏ current[u] vào đầu L[u]
            }
            PushToQueue(u); //Đẩy u vào hàng đợi chờ xử lý sau
        }
    }
}

void Print() //In kết quả
{
    FlowVal = 0;
    for (int i = 0; i < 2 * m; i += 2) //Chỉ in luồng dương trên các cung đã cho
    {
        cout << "e[" << i / 2 << "] = "
             << "(" << e[i].x << ", " << e[i].y << "): "
             << "c = " << e[i].cf + e[i ^ 1].cf << ", "
             << "f = " << e[i ^ 1].cf << '\n';
        if (e[i].x == s) FlowVal += e[i ^ 1].cf; //Cộng luồng dương trên các cung ra khỏi s
        if (e[i].y == s) FlowVal -= e[i ^ 1].cf; //Trừ luồng dương trên các cung đi vào s
    }
    cout << "Value of flow: " << FlowVal;
}

int main()
{
    Enter(); //Nhập dữ liệu
    Init();
    FifoPreflowPush();
    Print(); //In kết quả
}
```

Định lý 29-20 (Định lý về tính nguyên)

Nếu tất cả các sức chứa là số nguyên, đồng thời luồng/tiền luồng khởi tạo là các số nguyên, thì thuật toán Edmonds-Karp, thuật toán Dinic, 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 Edmonds-Karp, ban đầu ta khởi tạo luồng 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. Tương tự với thuật toán Dinic.

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.

29.5. So sánh về hiệu suất thực tế

Chính vì các kết quả lý thuyết về thời gian thực hiện giải thuật chỉ là đánh giá trong trường hợp xấu nhất, khá lệch so với thực tế gồm những dữ liệu cụ thể và ngẫu nhiên, ta sẽ thử nghiệm các thuật toán đã trình bày trong bài với các bộ dữ liệu ngẫu nhiên nhằm đưa ra nhận định nên dùng thuật toán nào trong trường hợp nào.

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. Có 4 chương trình được thử nghiệm: A: Thuật toán Edmonds-Karp, B: thuật toán Dinic, C: thuật toán FIFO Preflow-Push và D: Thuật toán FIFO Preflow-Push với phép gán nhãn lại toàn bộ. Mỗi chương trình được thử trên cả 100 đồ thị và đo thời gian thực hiện trung bình (tính bằng giây):

Số đỉnh (n)	Số cung (m)	A	B	C	D
100	10000	0.0705	0.0221	0.0331	0.0226
200	30000	0.5004	0.0541	0.1282	0.0556
500	40000	1.0985	0.0709	0.2770	0.0722
1000	10000	0.1551	0.0237	0.1343	0.0239
1000	100000	6.5447	0.1704	1.3981	0.1729
10000	100000	19.0872	0.1857	13.6925	0.1864

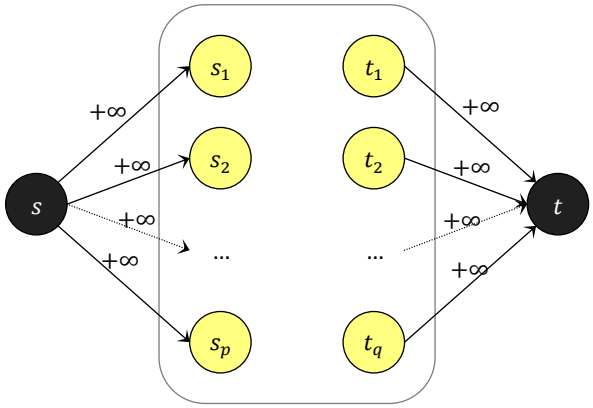
Những con số về thời gian thực thi chương trình cho thấy thuật toán A và C không thích hợp khi xử lý với mạng kích thước lớn. Thuật toán (B) và (D) có tốc độ tốt trong trường hợp mạng ngẫu nhiên, tuy nhiên phải nhấn mạnh rằng tuy thuật toán B (Dinic) có tốc độ tốt nhất trong các mạng ngẫu nhiên, có những cấu hình mạng khiến nó rơi vào trường hợp xấu như ta đã chỉ ra trong bài. Vì vậy khi triển khai trên các mạng kích thước lớn, sự lựa chọn an toàn vẫn là thuật toán FIFO Preflow-push với mẹo giải gán nhãn lại toàn bộ sau mỗi $|V|$ phép $Lift()$.

29.6. Một số mở rộng và ứng dụng của luồng

29.6.1. 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 29-6).



Hình 29-6. 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.

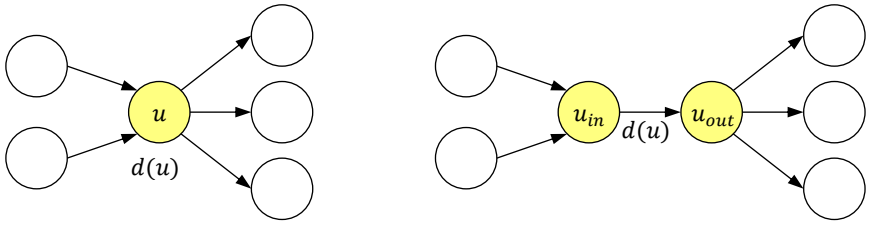
Một cách khác có thể thực hiện để tìm luồng trên mạng có nhiều đỉnh phát và nhiều đỉnh thu là loại bỏ tất các các cung đi vào các đỉnh phát cũng như các cung đi ra khỏi các đỉnh thu. Chập tất cả các đỉnh phát thành một siêu đỉnh s và chập tất cả các đỉnh thu lại thành một siêu đỉnh thu t, mạng không còn các đỉnh $s_1, s_2, .., s_p$ và $t_1, t_2, ..., t_q$ 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.

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

Cho mạng $G = (V, E, c, s, t)$, mỗi đỉnh $u \in V - \{s, t\}$ được gán một số không âm $d(u)$ 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/đi ra mỗi đỉnh $u \in V - \{s, t\}$ không được vượt quá $d(u)$: $\varphi(V, u) = \varphi(u, V) \leq d(u)$. Bài toán đặt ra là tìm luồng cực dương đại trên mạng có ràng buộc sức chứa trên cả các đỉnh và các cung.

Tách mỗi đỉnh $u \in V - \{s, t\}$ thành 2 đỉnh mới u_{in}, u_{out} và một cung (u_{in}, u_{out}) với sức chứa $d(u)$. Các cung đi vào u được chỉnh lại đầu mút để đi vào u_{in} và các cung đi ra khỏi u được chỉnh lại đầu mút để đi ra khỏi u_{out} (Hình 22-4). Ta xây dựng được mạng $G' = (V', E')$ với đỉnh phát s và đỉnh thu t.



Hình 29-7. 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 (u_{in}, u_{out}) trở lại thành đỉnh u để khôi phục lại mạng G ban đầu.

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

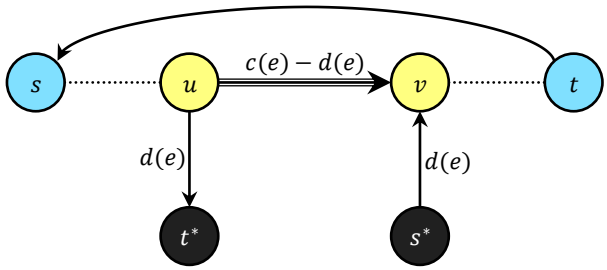
Cho mạng $G = (V, E, c, s, t)$ trong đó mỗi cung $e \in E$ ngoài sức chứa (lưu lượng) tối đa $c(e)$ còn được gán một số không âm $d(e) \leq c(e)$ gọi là lưu lượng tối thiểu. Một *luồng dương tương thích* φ trên G được định nghĩa với tất cả các ràng buộc của luồng dương và thêm một điều kiện: Luồng dương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

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

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

Xây dựng một mạng G' 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^*.
- Mỗi cung $e = (u, v)$ trên G sẽ tương ứng với ba cung trên G': 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) trên G' với sức chứa $+\infty$



Hình 29-8. Biến đổi mạng mới từ mạng với sức chứa các cung bị chặn hai phía

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 dương đó không thể vượt quá D.

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

Bổ đề 29-21

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 $|\varphi'| = D$.

Chứng minh

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

29.6.4. 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.

Để chỉ ra một luồng trên G, xét một cung $e \in E$ có sức chứa $c(e) < 0$. Theo tính phản đối xứng của luồng $f(e) = -f(-e)$ và ràng buộc sức chứa tối đa $f(e) \leq c(e)$, ta có:

$$f(-e) = -f(e) \geq -c(e) > 0 \qquad (29.7)$$

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 khởi tạo trên G có thể thực hiện bằng cách tìm luồng dương với ràng buộc luồng dương bị chặn hai phía:

$$-c(e) \leq f(-e) \leq c(-e)$$

Khi đã có luồng khởi tạo f:

- Để tìm luồng cực đại trên G, ta có thể tìm luồng cực đại d trên G_f (có sức chứa không âm) rồi trả về luồng $f + d$
- Để tìm luồng cực tiểu trên G, ta đảo vai trò đỉnh phát và đỉnh thu trên G_f (t phát, s thu) và tìm luồng cực đại d trên G_f (có sức chứa không âm) rồi trả về luồng $f + d$

Kỹ thuật này cũng có thể sử dụng để tìm luồng dương khả thi cực đại/cực tiểu trên mạng với luồng dương bị chặn hai phía (Mục 29.6.3).

29.6.5. 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 29-22

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 29-22 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 [28] 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: $c(A,x)$

Bổ đề 29-23

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 (X, Y), ta sẽ chứng minh rằng lưu lượng của lát cắt $(V - \{t\}, \{t\})$ không lớn hơn lưu lượng của lát cắt (X, Y), tức là:

$$c(V - \{t\}, t) \leq c(X, Y)$$

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 (X, Y). Xét thời điểm sau khi v được kết nạp vào A, gọi:

- A_v bằng $A - \{v\}$
- X_v là các đỉnh $\in X$ được kết nạp vào A
- Y_v là các đỉnh $\in Y$ được kết nạp vào A

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

$$c(A_u, u) \leq c(X_u, Y_u) \tag{29.8}$$

Nếu u là đỉnh hoạt tính đầu tiên được kết nạp vào A, ta có u nằm ở một phía của lát cắt (X, Y) và A_u chứa các đỉnh nằm ở phía còn lại. Tức là cặp $(A_u, \{u\})$ cũng là cặp (X_u, Y_u) nên $c(A_u, u) = c(X_u, Y_u)$.

Giả thiết rằng bất đẳng thức (29.8) đúng với đỉnh hoạt tính u, ta sẽ chứng minh nó cũng đúng với những đỉnh hoạt tính v được kết nạp vào A sau u. Thật vậy:

$$c(A_v, v) = c(A_u, v) + c(A_v - A_u, v) \tag{29.9}$$

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

$$c(A_u, v) \leq c(A_u, u) \leq c(X_u, Y_u)$$

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 tập X_v, Y_v. Mặt khác khi u được kết nạp vào A thì v chưa được kết nạp nên những cạnh này không nối giữa hai tập X_u, Y_u. Tổng hợp lại ta có: sức chứa của những cạnh nối giữa v và $A_v - A_u$ có mặt trong phép tính:

$$c(X_v, Y_v) = \sum_{\forall e \in \{X_v \to Y_v\}} c(e)$$

nhưng không có mặt trong phép tính:

$$c(X_u, Y_u) = \sum_{\forall e \in \{X_u, Y_u\}} c(e)$$

Vậy:

$$\begin{aligned} c(A_v, v) &= c(A_u, v) + c(A_v - A_u, v) \\ &\leq c(X_u, Y_u) + c(A_v - A_u, v) \\ &\leq c(X_v, Y_v) \end{aligned} \tag{29.10}$$

Vì (X, Y) là một lát cắt $s - t$ nên s và t nằm ở hai phía khác nhau của lát cắt (X, Y). Ngoài ra s và t là hai đỉnh được kết nạp vào A sau cùng nên t là đỉnh hoạt tính. Bất đẳng thức (29.10) chứng minh ở trên cho ta kết quả:

$$c(V - \{t\}, t) = c(A_t, t) \leq c(X_t, Y_t) = c(X, Y)$$

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

Định lý 29-24

Lát cắt tổng quát trên đồ thị vô hướng liên thông với hàm trọng số không âm có thể tìm được 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 gán nhãn lực hút: 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 đỉnh 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 đắn 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 29-1

Chỉ ra rằng nếu giữa 2 đỉnh (u, v) có nhiều cung song song, ta có thể thay thế các cung song song này bằng một cung (u, v) duy nhất có sức chứa bằng tổng sức chứa các cung song song ban đầu mà không làm ảnh hưởng tới giá trị luồng cực đại. Từ đó tìm cách cài đặt thuật toán giải quyết bài toán luồng cực đại trên mạng vô hướng mà không cần thêm vào các cung đối.

Bài tập 29-2

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 29-3

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 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 29-4

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 29-20 (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).

Bài tập 29-5

Tương tự như Bài tập 29-4 nhưng yêu cầu thực hiện trên đồ thị vô hướng.

Bài tập 29-6 (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:

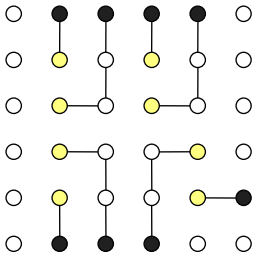
- Lớp đỉnh phát $S = \{s_1, s_2, \dots, s_n\}$, mỗi đỉnh tương ứng với một bạn nam.
- Lớp đỉnh $X = (x_1, x_2, \dots, x_n)$ mỗi đỉnh tương ứng với một món quà.
- Lớp đỉnh thu $T = \{t_1, t_2, \dots, t_n\}$ mỗi đỉnh tương ứng với một bạn nữ.

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 từ một người tặng tới một món quà và từ một món quà với người nhận tương ứng.

Bài tập 29-7

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 29-8 (Kỹ thuật giãn sức chứa)

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

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

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

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

```
void MaxFlowByScaling()
{
    f = «Luồng 0»;
    for (k = C; k ≥ 1; k = ⌊k / 2⌋)
        while («Tìm được đường tăng luồng P có dư lượng ≥ k»)
            «Tăng luồng dọc đường P»;
}
```

d) Khi bước vào mỗi lượt lặp của vòng lặp for..., gọi (X, Y) là lát cắt $s - t$ hẹp nhất trên mạng dư lượng G_f. Chứng minh rằng $c_f(X, Y) \leq 2k|E|$.

e) Chứng minh rằng trong mỗi lượt lặp của vòng lặp for, vòng lặp while bên trong thực hiện $O(|E|)$ lần

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 $O(|E|^2 \log C)$.

Bài tập 29-9 (Chọn dự án)

Cho n dự án đánh số từ 1 tới n và m máy đánh số từ 1 tới m. Dự án thứ i khi được chọn thực hiện sẽ thu được khoản tiền là r_i, máy thứ j khi mua sẽ phải chi ra khoản tiền là c_j. Một dự án muốn thực hiện có thể cần mua một vài máy và khi một máy đã mua có thể dùng trong nhiều dự án khác nhau. Hãy chọn ra các máy cần mua và các dự án cần thực hiện để số tiền lợi nhuận (sau khi cân đối thu chi) là lớn nhất.

Gợi ý:

Trong một phương án bất kỳ, gọi P là tập các dự án không được làm và Q là tập các máy được mua, khi đó ta cần tìm phương án để cực đại hóa lợi nhuận:

$$\sum_{i=1}^{n} r_i - \sum_{\forall i \in P} r_i - \sum_{\forall j \in Q} c_j \longrightarrow \max$$

Vì giá trị $\sum_{i=1}^{n} r_i$ không phụ thuộc phương án, ta có thể đổi hàm mục tiêu thành:

$$\sum_{\forall i \in P} p_i + \sum_{\forall j \in Q} q_j \longrightarrow \min$$

Dựng mạng trong đó mỗi dự án i ứng với một đỉnh p_i, mỗi máy j ứng với một đỉnh q_j, thêm một đỉnh phát s và đỉnh thu t.

- Với mọi dự án i, thêm cung (s, p_i) với sức chứa r_i $(\forall i: 1 \le i \le n)$
- Với mọi máy j, thêm cung (q_j, t) với sức chứa c_j $(\forall j: 1 \le j \le m)$
- Nếu dự án i cần máy j, thêm cung (p_i, q_j) với sức chứa $+\infty$ $(\forall i, j: 1 \le i \le n; 1 \le j \le m)$

Tìm luồng cực đại trên mạng và xác định một lát cắt $s - t$ hẹp nhất (X, Y)

a) Chỉ ra rằng những cung $\in \{X \to Y\}$ chỉ thuộc một trong hai dạng: cung (s, p_i) hoặc cung (q_j, t)

b) Với một phương án bất kỳ, gọi P là tập các dự án không được làm và Q là tập các máy được mua. Chứng minh điều kiện cần và đủ để phương án P hợp lệ là: với mọi cung (p_i, q_j) thì hoặc $i \in P$ hoặc $j \in Q$.

c) Xét những cung $\in \{X \to Y\}$, đặt:

$$P = \{i: (s, p_i) \in (X, Y)\}$$
$$Q = \{j: (q_j, t) \in (X, Y)\}$$

Chỉ ra rằng phương án tối ưu sẽ là phương án mà nhưng dự án $\in P$ không được làm và những máy $\in Q$ được mua.

Bài tập 29-10 (Chọn dự án 2)

Có n dự án đánh số từ 1 tới n, dự án thứ i khi làm xong sẽ thu được lợi nhuận là r_i (r_i có thể âm nếu số tiền chi ra để đầu tư ban đầu lớn hơn số tiền thu lại sau khi hoàn thành dự án). Có m mối quan hệ, mỗi quan hệ có dạng (i, j) cho biết muốn chọn dự án i thì bắt buộc phải chọn cả dự án j. Hãy chọn một số dự án để làm để tổng lợi nhuận thu được là lớn nhất.

Gợi ý:

Dựng mạng với n đỉnh tương ứng với n dự án, thêm một đỉnh phát $s = 0$ và một đỉnh thu $t = n + 1$, coi như $r_0 = r_{n+1} = 0$. Các cung của mạng được xây dựng như sau:

- Với mỗi quan hệ (i, j) (nếu muốn chọn dự án i thì phải chọn dự án j), ta thêm cung (i, j) với sức chứa $+\infty$.
- Với mỗi dự án i có lợi nhuận $r_i \ge 0$, thêm cung (s, i) với sức chứa r_i $(\forall i: 1 \le i \le n)$
- Với mỗi dự án j có lợi nhuận $r_j < 0$, thêm cung (j, t) với sức chứa $-r_j$ $(\forall j: 1 \le j \le n)$

Với một phương án hợp lệ để chọn dự án, gọi Y là tập gồm đỉnh t và các dự án được chọn, tập X gồm dự án không được chọn và đỉnh s. Ta xây dựng được (X, Y) là lát cắt $s - t$ ứng với một phương án chọn dự án.

a) Chứng minh rằng $c(X, Y) < +\infty$, tức là không tồn tại cung với sức chứa $+\infty$ nối từ X sang Y. Nói cách khác, những cung nối từ X sang Y hoặc là cung dạng (s, i), hoặc là cung dạng (j, t).

b) Xét biểu thức của $c(X,Y)$

$$\begin{aligned} c(X,Y) &= \sum_{\forall i \in Y: r_i \geq 0} c(s,i) + \sum_{\forall j \in X: r_j < 0} c(j,t) \\ &= \sum_{\forall i \in X: r_i \geq 0} r_i + \sum_{\forall j \in Y: r_j < 0} (-r_j) \\ &= \underbrace{\sum_{\forall i: r_i \geq 0} r_i}_{\text{Hằng số } C} - \sum_{\forall i \in Y: r_i \geq 0} r_i - \sum_{\forall j \in Y: r_j < 0} r_j \\ &= C - \sum_{\forall i \in Y} r_i \end{aligned}$$

Chỉ ra rằng nếu lưu lượng của lát cắt (X,Y) càng nhỏ thì lợi nhuận của phương án tương ứng càng lớn.

c) Xây dựng thuật toán giải quyết bài toán này thông qua mô hình luồng cực đại và lát cắt hẹp nhất.