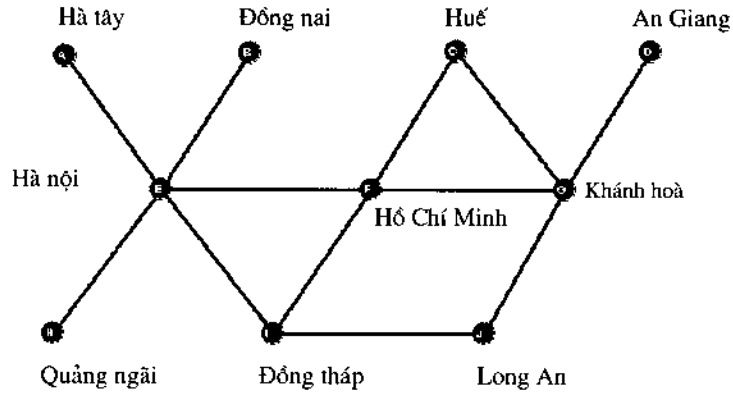
PHẦN II

LÝ THUYẾT ĐỒ THỊ

các máy tính và các kênh điện thoại (gọi tắt là kênh thoại) nối các máy tính này. Chúng ta có thể biểu diễn các vị trí đặt máy tính bởi các điểm và các kênh thoại nối chúng bởi các đoạn nối, xem hình 1.

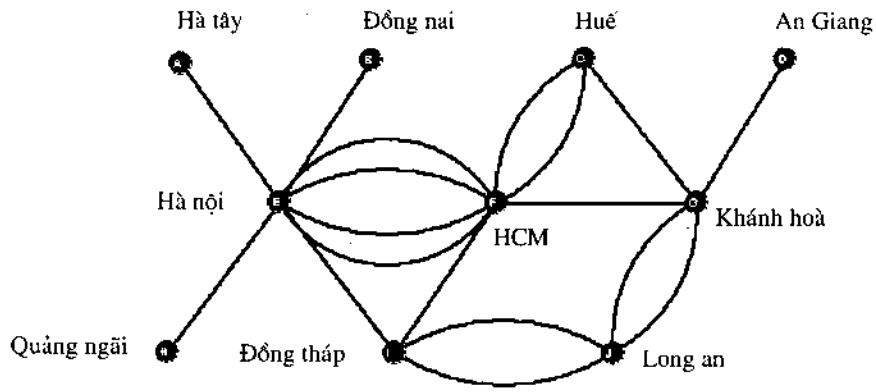


Hình 1. Sơ đồ mạng máy tính

Nhận thấy rằng trong mạng ở hình 1, giữa hai máy bất kỳ chỉ có nhiều nhất là một kênh thoại nối chúng, kênh thoại này cho phép liên lạc cả hai chiều và không có máy tính nào lại được nối với chính nó. Sơ đồ mạng máy tính cho trong hình 1 được gọi là *đơn đồ thị vô hướng*. Ta đi đến định nghĩa sau

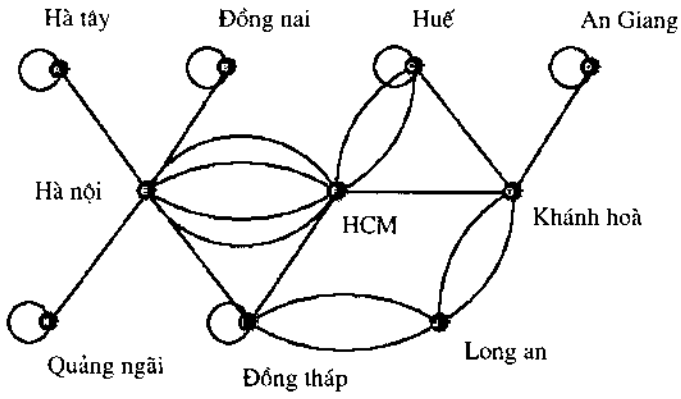
Định nghĩa 1. *Đơn đồ thị vô hướng $G = (V,E)$ bao gồm V là tập các đỉnh, và E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh.*

Trong trường hợp giữa hai máy tính nào đó thường xuyên phải truyền tải nhiều thông tin người ta phải nối hai máy này bởi nhiều kênh thoại. Mạng với đa kênh thoại giữa các máy được cho trong hình 2.



Hình 2. Sơ đồ mạng máy tính với đa kênh thoại

Định nghĩa 2. *Đa đồ thị vô hướng $G = (V,E)$ bao gồm V là tập các đỉnh, và E là họ các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh. Hai cạnh e_1 và e_2 được gọi là cạnh lặp nếu chúng cùng tương ứng với một cặp đỉnh.*



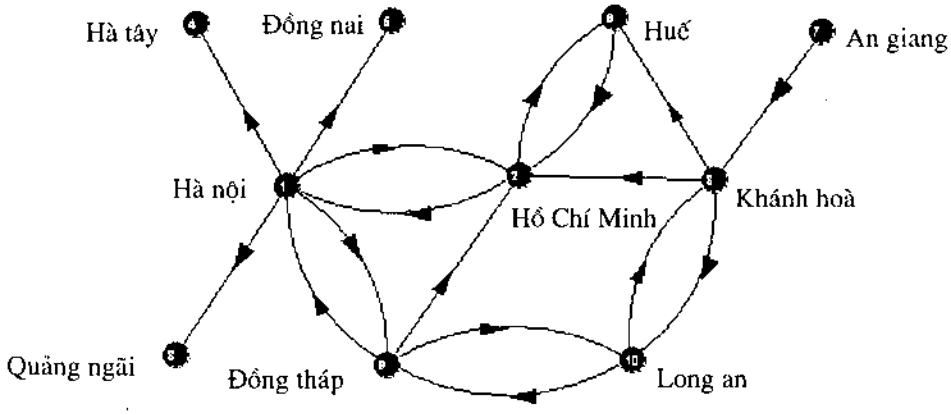
Hình 3. Sơ đồ mạng máy tính với kênh thông báo

Rõ ràng mỗi đơn đồ thị đều là đa đồ thị, nhưng không phải đa đồ thị nào cũng là đơn đồ thị, vì trong đa đồ thị có thể có hai (hoặc nhiều hơn) cạnh nối một cặp đỉnh nào đó.

Trong mạng máy tính có thể có những kênh thoại nối một máy nào đó với chính nó (chẳng hạn với mục đích thông báo). Mạng như vậy được cho trong hình 3. Khi đó đa đồ thị không thể mô tả được mạng như vậy, bởi vì có những *khuyên* (cạnh nối một đỉnh với chính nó). Trong trường hợp này chúng ta cần sử dụng đến khái niệm *giả đồ thị vô hướng*, được định nghĩa như sau

Định nghĩa 3. *Giả đồ thị vô hướng $G = (V,E)$ bao gồm V là tập các đỉnh, và E là họ các cặp không có thứ tự gồm hai phần tử (không nhất thiết phải khác nhau) của V gọi là các cạnh. Cạnh e được gọi là khuyên nếu nó có dạng $e=(u,u)$.*

Các kênh thoại trong mạng máy tính có thể chỉ cho phép truyền tin theo một chiều. Chẳng hạn, trong hình 4 máy chủ ở Hà nội chỉ có thể nhận tin từ các máy ở địa phương, có một số máy chỉ có thể gửi tin đi, còn các kênh thoại cho phép truyền tin theo cả hai chiều được thay thế bởi hai cạnh có hướng ngược chiều nhau.



Hình 4. Mạng máy với các kênh thoại một chiều

Thí dụ 3. Xét đồ thị cho trong hình 2. Ta có

$deg^-(A) = 2, deg^-(B) = 3, deg^-(C) = 1, deg^-(D) = 2, deg^-(E) = 2.$

$deg^+(A) = 3, deg^+(B) = 2, deg^+(C) = 2, deg^+(D) = 2, deg^+(E) = 1.$

Do mỗi cung (u,v) sẽ được tính một lần trong bán bậc vào của đỉnh v và một lần trong bán bậc ra của đỉnh u nên ta có:

Định lý 2. *Giả sử $G=(V,E)$ là đồ thị có hướng. Khi đó*

$$\sum_{v \in V} \deg^+(v) = \sum_{v \in V} \deg^-(v) = |E|$$

Rất nhiều tính chất của đồ thị có hướng không phụ thuộc vào hướng trên các cung của nó. Vì vậy, trong nhiều trường hợp sẽ thuận tiện hơn nếu ta bỏ qua hướng trên các cung của đồ thị. Đồ thị vô hướng thu được bằng cách bỏ qua hướng trên các cung được gọi là *đồ thị vô hướng tương ứng* với đồ thị có hướng đã cho.

1.3. Đường đi, Chu trình. Đồ thị liên thông

Định nghĩa 1. ***Đường đi độ dài n*** *từ đỉnh u đến đỉnh v, trong đó n là số nguyên dương, trên đồ thị vô hướng $G=(V,E)$ là dãy*

$$x_0, x_1, \ldots, x_{n-1}, x_n$$

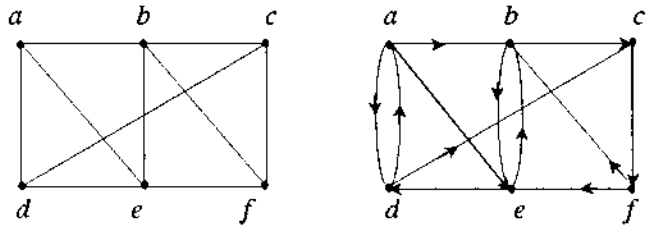
trong đó $u = x_0$, $v = x_n$, $(x_i, x_{i+1}) \in E$, $i = 0, 1, 2, \ldots, n-1$.

Đường đi nói trên còn có thể biểu diễn dưới dạng dãy các cạnh:

$$(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n).$$

Đỉnh u gọi là đỉnh đầu, còn đỉnh v gọi là đỉnh cuối của đường đi. Đường đi có đỉnh đầu trùng với đỉnh cuối (tức là $u = v$) được gọi là ***chu trình****. Đường đi hay chu trình được gọi là* ***đơn*** *nếu như không có cạnh nào bị lặp lại.*

Thí dụ 1. Xét đồ thị vô hướng cho trong hình 1:



Hình 1. Đường đi trên đồ thị

Ta có: *a, d, c, f, e* là đường đi đơn độ dài 4. Còn *d, e, c, a* không là dường đi, do (*e, c*) không phải là cạnh của đồ thị. Dãy *b, c, f, e, b* là chu trình độ dài 4. Đường đi *a, b, e, d, a, b* có độ dài là 5 không phải là đường đi đơn, do cạnh (*a, b*) có mặt trong nó hai lần.

Khái niệm đường đi và chu trình trên đồ thị có hướng được định nghĩa hoàn toàn tương tự như trường hợp đồ thị vô hướng, chỉ khác là ta có chú ý đến hướng trên các cung.

Định nghĩa 2. ***Đường đi độ dài n*** *từ đỉnh u đến đỉnh v, trong đó n là số nguyên dương, trên đồ thị có hướng G=(V,A) là dãy*

$$x_0, x_1, \ldots, x_{n-1}, x_n$$

trong đó $u = x_0$, $v = x_n$, $(x_i, x_{i+1}) \in A$, $i = 0, 1, 2, \ldots, n-1$.

Đường đi nói trên còn có thể biểu diễn dưới dạng dãy các cung:

$$(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n).$$

Đỉnh u gọi là đỉnh đầu, còn đỉnh v gọi là đỉnh cuối của đường đi. Đường đi có đỉnh đầu trùng với đỉnh cuối (tức là u = v) được gọi là ***chu trình****. Đường đi hay chu trình được gọi là* ***đơn*** *nếu như không có cung nào bị lặp lại.*

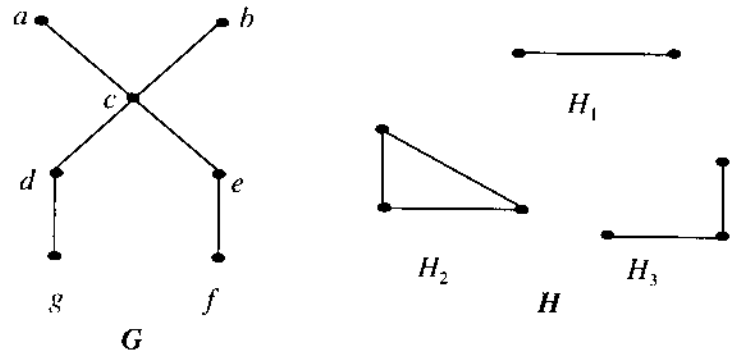
Thí dụ 2. Trên đồ thị có hướng cho trong hình 1: *a, d, c, f, e* là đường đi đơn độ dài 4. Còn *d, e, c, a* không là đường đi, do (*e,c*) không phải là cung của đồ thị. Dãy *b, c, f, e, b* là chu trình độ dài 4. Đường đi *a, b, e, d, a, b* có độ dài là 5 không phải là đường đi đơn, do cung (*a,b*) có mặt trong nó hai lần.

Xét một mạng máy tính. Một câu hỏi đặt ra là hai máy tính bất kỳ trong mạng này có thể trao đổi thông tin được với nhau hoặc là trực tiếp qua kênh nối chúng hoặc thông qua một hoặc vài máy tính trung gian trong mạng? Nếu sử dụng đồ thị để biểu diễn mạng máy tính này (trong đó các đỉnh của đồ thị tương ứng với các máy tính, còn các cạnh tương ứng với các kênh nối) câu hỏi đó được phát biểu trong ngôn ngữ đồ thị như sau: Tồn tại hay chăng đường đi giữa mọi cặp đỉnh của đồ thị?

Định nghĩa 3. *Đồ thị vô hướng G= (V,E) được gọi là liên thông nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.*

Như vậy hai máy tính bất kỳ trong mạng có thể trao đổi thông tin được với nhau khi và chỉ khi đồ thị tương ứng với mạng này là đồ thị liên thông.

Thí dụ 3. Trong hình 2: Đồ thị *G* là liên thông, còn đồ thị *H* là không liên thông.



Hình 2. Đồ thị liên thông G và đồ thị H gồm 3 thành phần liên thông H_1, H_2, H_3.

Định nghĩa 4. *Ta gọi đồ thị con của đồ thị $G = (V,E)$ là đồ thị $H = (W, F)$, trong đó $W \subseteq V$ và $F \subseteq E$.*

Trong trường hợp đồ thị là không liên thông, nó sẽ rã ra thành một số đồ thị con liên thông đôi một không có đỉnh chung. Những đồ thị con liên thông như vậy ta sẽ gọi là các ***thành phần liên thông*** của đồ thị.

Thí dụ 4. Đồ thị H trong hình 2 gồm 3 thành phần liên thông H_1, H_2, H_3.

Trong mạng máy tính có thể có những máy (những kênh nối) mà sự hỏng hóc của nó sẽ ảnh hưởng đến việc trao đổi thông tin trong mạng. Các khái niệm tương ứng với tình huống này được đưa ra trong định nghĩa sau.

Định nghĩa 5. *Đỉnh v được gọi là **đỉnh rẽ nhánh** nếu việc loại bỏ v cùng với các cạnh liên thuộc với nó khỏi đồ thị làm tăng số thành phần liên thông của đồ thị. Cạnh e được gọi là **cầu** nếu việc loại bỏ nó khỏi đồ thị làm tăng số thành phần liên thông của đồ thị.*

Thí dụ 5. Trong đồ thị G ở hình 2, đỉnh d và e là đỉnh rẽ nhánh, còn các cạnh (d,g) và (e,f) là cầu.

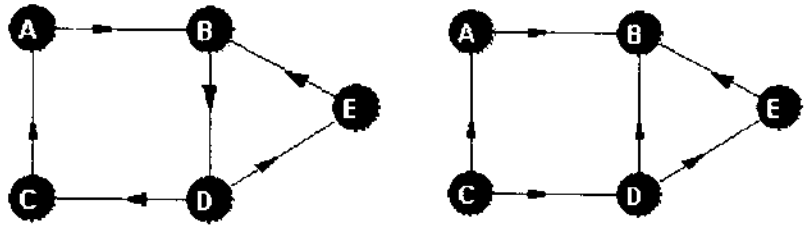
Đối với đồ thị có hướng có hai khái niệm liên thông phụ thuộc vào việc ta có xét đến hướng trên các cung hay không.

Định nghĩa 6. *Đồ thị có hướng $G= (V,A)$ được gọi là liên thông mạnh nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.*

Định nghĩa 7. *Đồ thị có hướng $G= (V,A)$ được gọi là liên thông yếu nếu đồ thị vô hướng tương ứng với nó là đồ thị vô hướng liên thông.*

Rõ ràng nếu đồ thị là liên thông mạnh thì nó cũng là liên thông yếu, nhưng điều ngược lại là không luôn đúng, như chỉ ra trong thí dụ dưới đây.

Thí dụ 6. Trong hình 3 đồ thị G là liên thông mạnh, còn H là liên thông yếu nhưng không là liên thông mạnh.



G H

Hình 3. Đồ thị liên thông mạnh G và đồ thị liên thông yếu H

Một câu hỏi đặt ra là khi nào có thể định hướng các cạnh của một đồ thị vô hướng liên thông để có thể thu được đồ thị có hướng liên thông mạnh? Ta sẽ gọi đồ thị như vậy là đồ thị định hướng được. Định lý dưới đây cho ta tiêu chuẩn nhận biết một đồ thị có là định hướng được hay không.

Định lý 1. *Đồ thị vô hướng liên thông là định hướng được khi và chỉ khi mỗi cạnh của nó nằm trên ít nhất một chu trình.*

Chứng minh. ***Điều kiện cần***. Giả sử (u,v) là một cạnh của đồ thị. Từ sự tồn tại đường đi có hướng từ u đến v và ngược lại suy ra (u,v) phải nằm trên ít nhất một chu trình.

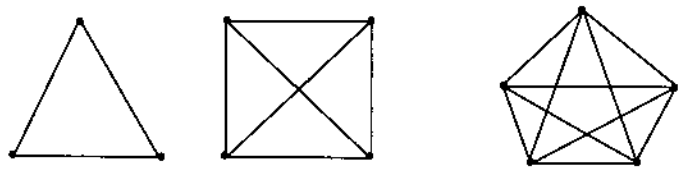
Điều kiện đủ. Thủ tục sau đây cho phép định hướng các cạnh của đồ thị để thu được đồ thị có hướng liên thông mạnh. Giả sử C là một chu trình nào đó trong đồ thị. Định hướng các cạnh trên chu trình này theo một hướng đi vòng theo nó. Nếu tất cả các cạnh của đồ thị là đã được định hướng thì kết thúc thủ tục. Ngược lại, chọn e là một cạnh chưa định hướng có chung đỉnh với ít nhất một trong số các cạnh đã định hướng. Theo giả thiết tìm được chu trình C' chứa cạnh e. Định hướng các cạnh chưa được định hướng của C' theo một hướng dọc theo chu trình này (không định hướng lại các cạnh đã có hướng). Thủ tục trên sẽ được lặp lại cho đến khi tất cả các cạnh của đồ thị được định hướng. Khi đó ta thu được đồ thị có hướng liên thông mạnh.

1.4. Một số dạng đồ thị đặc biệt

Trong mục này ta xét một số dạng đơn đồ thị vô hướng đặc biệt xuất hiện trong nhiều vấn đề ứng dụng thực tế.

Đồ thị đầy đủ. Đồ thị đầy đủ n đỉnh, ký hiệu bởi K_n, là đơn đồ thị vô hướng mà giữa hai đỉnh bất kỳ của nó luôn có cạnh nối.

Các đồ thị K_3, K_4, K_5 cho trong hình 1 dưới đây.



K_3 K_4 K_5

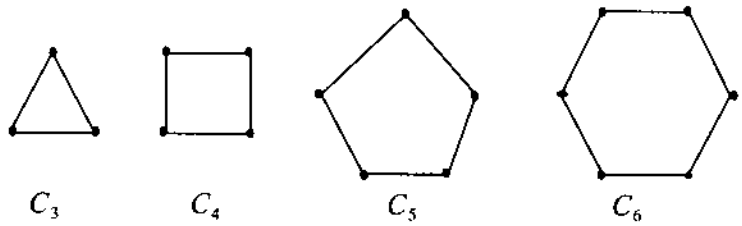
Hình 1. Đồ thị đầy đủ

Đồ thị đầy đủ K_n có tất cả $n(n\text{-}1)/2$ cạnh, nó là đơn đồ thị có nhiều cạnh nhất.

Đồ thị vòng. Đồ thị vòng C_n , $n \geq 3$, gồm n đỉnh $v_1, v_2, \ldots, v_n$ và các cạnh

$$(v_1, v_2),\ (v_2, v_3),\ \ldots,\ (v_{n-1}, v_n),\ (v_n, v_1).$$

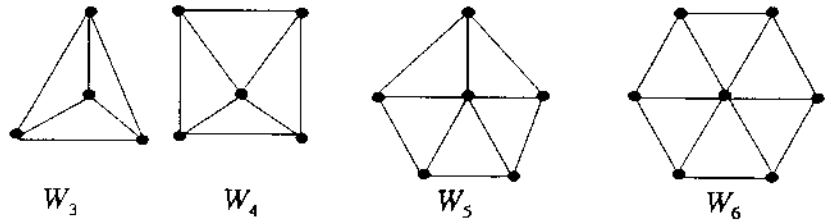
Đồ thị vòng C_3, C_4, C_5, C_6 cho trong hình 2.



C_3 C_4 C_5 C_6

Hình 2. Đồ thị vòng C_3 , C_4 , C_5, C_6

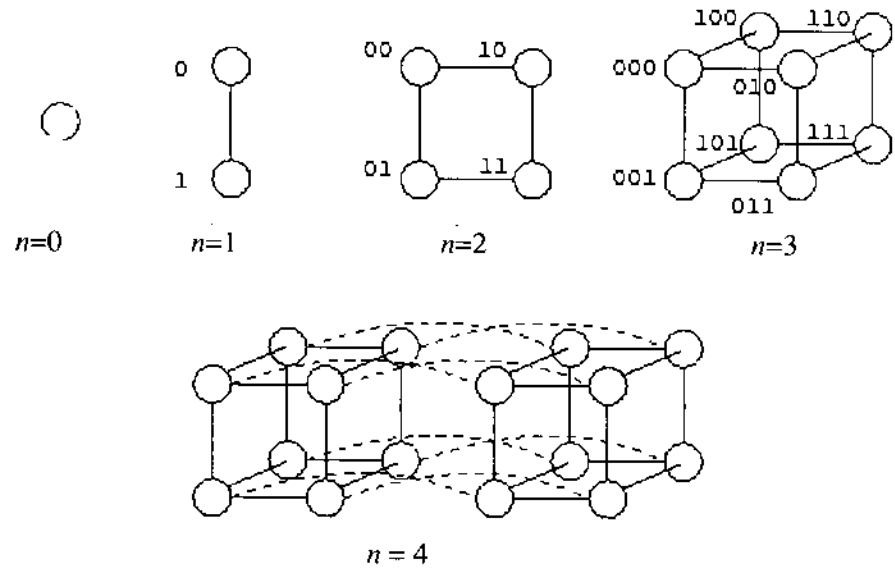
Đồ thị bánh xe. Đồ thị W_n thu được từ C_n bằng cách bổ sung vào một đỉnh mới nối với tất cả các đỉnh của C_n (xem hình 3).



W_3 W_4 W_5 W_6

Hình 3. Đồ thị bánh xe W_3, W_4, W_5, W_6

Đồ thị lập phương. Đồ thị lập phương n đỉnh Q_n là đồ thị với các đỉnh biểu diễn 2^n xâu nhị phân độ dài n. Hai đỉnh của nó là kề nhau nếu như hai xâu nhị phân tương ứng chỉ khác nhau 1 bit. Hình 4 cho thấy Q_n với $n = 0,1, 2, 3, 4$.



Hình 4. Đồ thị lập phương Q_n

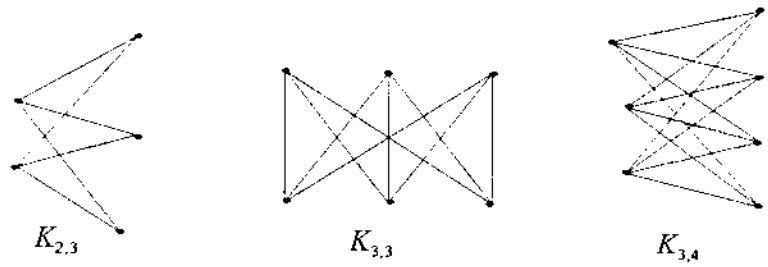
Đồ thị hai phía. Đơn đồ thị $G = (V,E)$ được gọi là *hai phía* nếu như tập đỉnh V của nó có thể phân hoạch thành hai tập X và Y sao cho mỗi cạnh của đồ thị chỉ nối một đỉnh nào đó trong X với một đỉnh nào đó trong Y. Khi đó ta sẽ sử dụng ký hiệu $G=(X \cup Y, E)$ để chỉ đồ thị hai phía với tập đỉnh $X \cup Y$.

Định lý sau đây cho phép nhận biết một đơn đồ thị có phải là hai phía hay không.

Định lý 1. *Đơn đồ thị là đồ thị hai phía khi và chỉ khi nó không chứa chu trình độ dài lẻ.*

Để kiểm tra xem một đồ thị liên thông có phải là hai phía hay không có thể áp dụng thủ tục sau. Chọn v là một đỉnh bất kỳ của đồ thị. Đặt $X = \{v\}$, còn Y là tập các đỉnh kề của v. Khi đó các đỉnh kề của các đỉnh trong Y phải thuộc vào X. Ký hiệu tập các đỉnh như vậy là T. Vì thế nếu phát hiện $T \cap Y \neq \varnothing$ thì đồ thị không phải là hai phía, kết thúc. Ngược lại, đặt $X = X \cup T$. Tiếp tục xét như vậy đối với T' là tập các đỉnh kề của T,...

Đồ thị hai phía $G = (X \cup Y, E)$ với $|X| = m$, $|Y| = n$ được gọi là ***đồ thị hai phía đầy đủ*** và ký hiệu là $K_{m,n}$ nếu mỗi đỉnh trong tập X được nối với mỗi đỉnh trong Y. Các đồ thị $K_{2,3}$, $K_{3,3}$, $K_{3,4}$ được cho trong hình 5.



Hình 5. Đồ thị hai phía

Đồ thị phẳng. Đồ thị được gọi là *đồ thị phẳng* nếu ta có thể vẽ nó trên mặt phẳng sao cho các cạnh của nó không cắt nhau ngoài ở đỉnh. Cách vẽ như vậy sẽ được gọi là *biểu diễn phẳng* của đồ thị.

Thí dụ đồ thị K_4 là phẳng, vì có thể vẽ nó trên mặt phẳng sao cho các cạnh của nó không cắt nhau ngoài ở đỉnh (xem hình 6).

Hình 6. Đồ thị K_4 là đồ thị phẳng

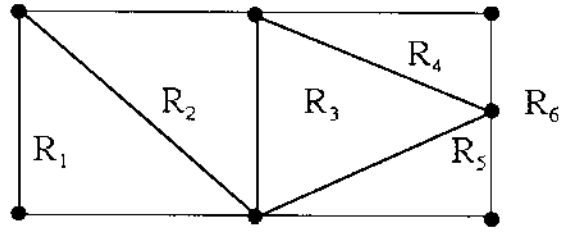
Một điểm đáng lưu ý là nếu đồ thị là phẳng thì luôn có thể vẽ nó trên mặt phẳng với các cạnh nối là các đoạn thẳng không cắt nhau ngoài ở đỉnh (ví dụ xem cách vẽ K_4 trong hình 3).

Để nhận biết xem một đồ thị có phải là đồ thị phẳng có thể sử dụng định lý Kuratovski, mà để phát biểu nó ta cần một số khái niệm sau: Ta gọi một ***phép chia cạnh*** (u,v) của đồ thị là việc loại bỏ cạnh này khỏi đồ thị và thêm vào đồ thị một đỉnh mới w cùng với hai cạnh (u,w), (w,u). Hai đồ thị $G = (V, E)$ và $H = (W, F)$ được gọi là ***đồng cấu*** nếu chúng có thể thu được từ cùng một đồ thị nào đó nhờ các phép chia cạnh.

Định lý 2 (Kuratovski). *Đồ thị là phẳng khi và chỉ khi nó không chứa đồ thị con đồng cấu với $K_{3,3}$ hoặc K_5.*

Trong trường hợp riêng, đồ thị $K_{3,3}$ và K_5 không phải là đồ thị phẳng. Bài toán về tính phẳng của đồ thị $K_{3,3}$ là bài toán đố nổi tiếng về *ba căn hộ và ba hệ thống cung cấp năng lượng* cho chúng: Cần xây dựng hệ thống đường cung cấp điện, hơi đốt và nước cho ba căn hộ, nối mỗi một trong ba nguồn cung cấp năng lượng với mỗi một căn hộ nói trên sao cho chúng không cắt nhau.

Đồ thị phẳng còn tìm được những ứng dụng quan trọng trong công nghệ chế tạo mạch in. Biểu diễn phẳng của đồ thị sẽ chia mặt phẳng ra thành các miền, trong đó có thể có cả miền không bị chặn. Thí dụ, biểu diễn phẳng của đồ thị cho trong hình 7 chia mặt phẳng ra thành 6 miền $R_1, R_2, \ldots, R_6$.



Hình 7. Các miền tương ứng với biểu diễn phẳng của đồ thị

Euler đã chứng minh được rằng các cách biểu diễn phẳng khác nhau của một đồ thị đều chia mặt phẳng ra thành cùng một số miền. Để chứng minh điều đó, Euler đã tìm được mối liên hệ giữa số miền, số đỉnh của đồ thị và số cạnh của đồ thị phẳng sau đây.

Định lý 3 (Công thức Euler). Giả sử G là đồ thị phẳng liên thông với n đỉnh, m cạnh. Gọi r là số miền của mặt phẳng bị chia bởi biểu diễn phẳng của G. Khi đó

$$r = m - n + 2.$$

Có thể chứng minh định lý bằng qui nạp. Xét thí dụ minh hoạ cho áp dụng công thức Euler.

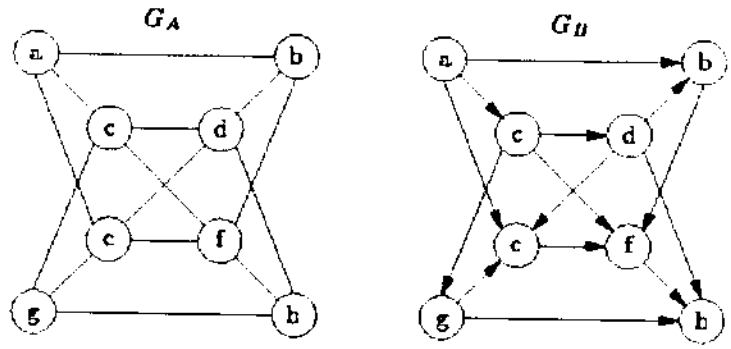
Thí dụ. Cho G là đồ thị phẳng liên thông với 20 đỉnh, mỗi đỉnh đều có bậc là 3. Hỏi mặt phẳng bị chia làm bao nhiêu phần bởi biểu diễn phẳng của đồ thị G?

Giải. Do mỗi đỉnh của đồ thị đều có bậc là 3, nên tổng bậc của các đỉnh là $3 \times 20 = 60$. Từ đó suy ra số cạnh của đồ thị $m = 60/2 = 30$. Vì vậy, theo công thức Euler, số miền cần tìm là

$$r = 30 - 20 + 2 = 12.$$

Bài tập

1. Xác định bậc của các đỉnh trong đồ thị G_A. Xác định bán bậc ra và bán bậc vào của các đỉnh của đồ thị G_B.



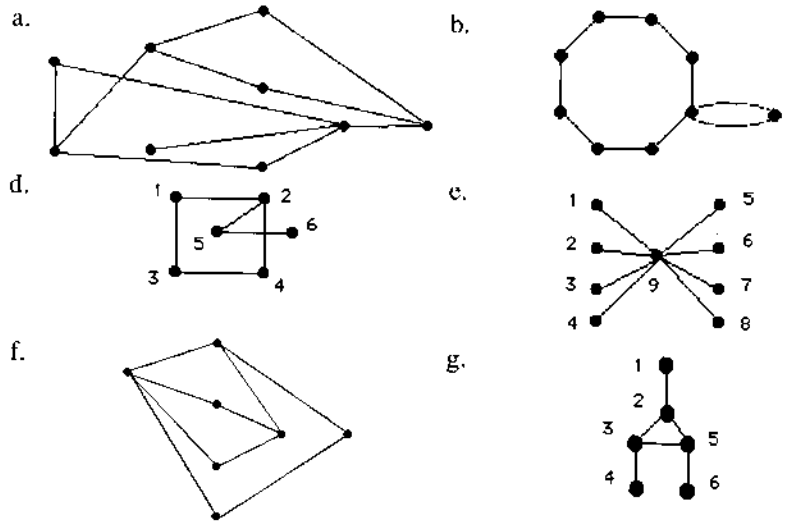
2. Vẽ đồ thị vô hướng $G = (V,E)$ cho bởi:

$$V = \{A, B, C, D, E, F\}$$

và

$$E = \{(E,G), (B,F), (D,C), (D,F), (F,B), (C,F), (A,F), (E,D)\}$$

3. Với mỗi đồ thị trong các đồ thị sau đây hãy cho biết nó có là đồ thị hai phía hay không? Nếu câu trả lời là khẳng định, hãy chỉ rõ cách phân hoạch tập đỉnh thành hai tập đỉnh sao cho cạnh nối chỉ có giữa hai đỉnh thuộc hai tập khác nhau.



3. Cho đơn đồ thị vô hướng liên thông $G = (V, E)$ với n đỉnh.

a) Chứng minh rằng luôn tồn tại đường đi đơn nối hai đỉnh u, v bất kỳ của đồ thị.

(*Gợi ý: Đường đi cần tìm là đường đi ngắn nhất theo số cạnh*)

b) Chứng minh rằng luôn tồn tại đường đi qua không quá n đỉnh nối hai đỉnh u, v bất kỳ của đồ thị.

(Gợi ý: Đường đi cần tìm là đường đi ngắn nhất theo số cạnh).

4. Cho G là đơn đồ thị vô hướng với n đỉnh, m cạnh, k thành phần liên thông. Chứng minh rằng:

$$n-k \leq (n-k)(n-k+1)/2.$$

Từ đó suy ra đồ thị n đỉnh với số cạnh lớn hơn $(n-1)(n-2)/2$ là liên thông.

(Gợi ý: Chứng minh bằng qui nạp theo số cạnh của đồ thị).

5. Chứng minh rằng trong đơn đồ thị với $n > 1$ đỉnh luôn tìm được hai đỉnh không là đỉnh rẽ nhánh.

6. Chứng minh rằng đỉnh u trong đơn đồ thị liên thông G là đỉnh rẽ nhánh khi và chỉ khi tìm được hai đỉnh v và w $(v,w \neq u)$ sao cho mọi đường đi nối v và w đều đi qua đỉnh u.

7. Chứng minh rằng cạnh e trong đơn đồ thị G là cầu khi và chỉ khi nó không thuộc bất cứ chu trình nào trong G.

8. Cho G là đồ thị hai phía với n đỉnh và m cạnh. Chứng minh rằng $m \leq n^2/4$.

9. Cho G là đồ thị hai phía với n đỉnh và m cạnh. Gọi K và k là bậc lớn nhất và nhỏ nhất của các đỉnh của G. Chứng minh rằng

$$m \leq 2m/n \leq M.$$

10. Đồ thị vô hướng được gọi là **chính qui bậc *k*** nếu tất cả các đỉnh của nó đều có bậc là k. Với giá trị nào của n đồ thị sau là chính qui?

a) K_n

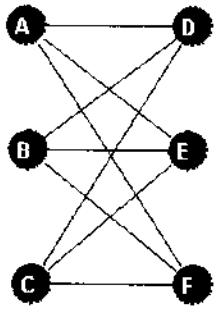
b) C_n

c) W_n

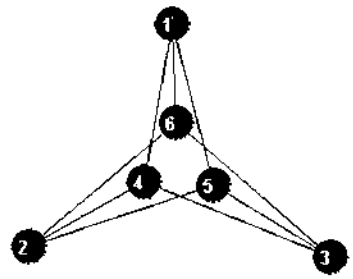
d) Q_n

11. Ta gọi đồ thị G' là **đồ thị bù** của đơn đồ thị G nếu các đỉnh của nó là đỉnh của đồ thị G và hai đỉnh của G' là kề nhau khi và chỉ khi chúng là không kề nhau trên G. Hãy vẽ các đồ thị bù của K_n, $K_{m,n}$, C_n, Q_n.

12. Hai đơn đồ thị vô hướng $G_1 = (V_1, E_1)$ và $G_2 = (V_2, E_2)$ được gọi là ***đẳng cấu*** nếu tồn tại một song ánh f: $V_1 \to V_2$ sao cho $(u,v) \in E_1$ khi và chỉ khi $(f(u), f(v)) \in E_2$. Thí dụ, hai đồ thị G_1 và G_2 cho trong hình dưới đây là đẳng cấu



G_1

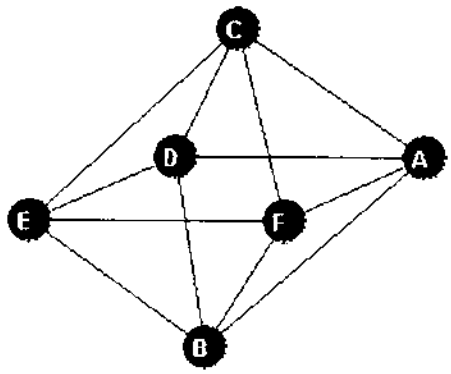


G_2

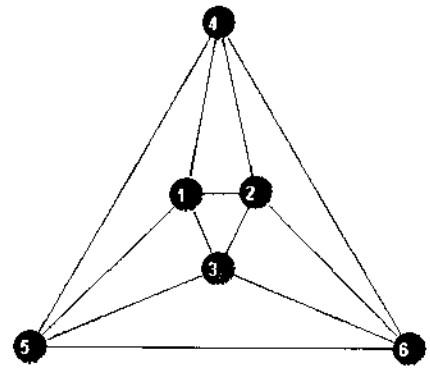
Song ánh f được xác định như sau: f(A)=1, f(B)=2, f(C)=3; f(D)=4, f(E)=5, f(F)=6.

Hỏi hai đồ thị sau đây có đẳng cấu hay không?

a)

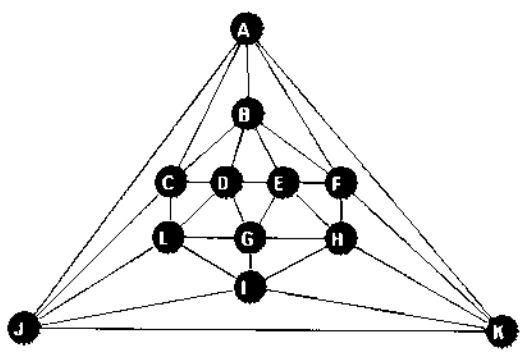


G_1

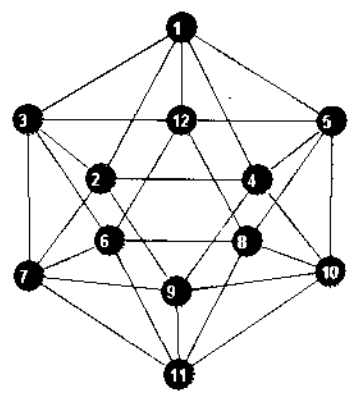


G_2

b)

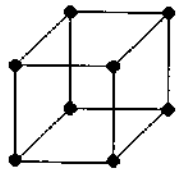


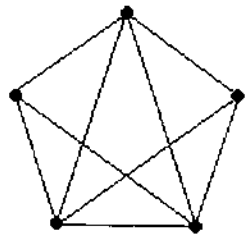
H_1

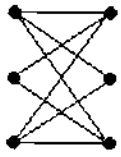


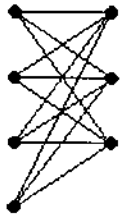
H_2

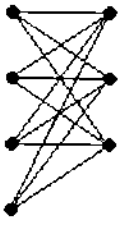
13. Với mỗi đồ thị trong các đồ thị sau đây hãy cho biết nó có là đồ thị phẳng hay không. Nếu câu trả lời là khẳng định hãy trình bày cách vẽ đồ thị sao cho các cạnh không cắt nhau ngoài ở đỉnh:

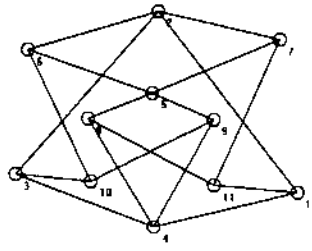
a) 

b) 

c) 

d) 

e) 

f) 

14. Hỏi rằng đồ thị Q_3 có phải là phẳng không? Trong trường hợp câu trả lời là khẳng định hãy vẽ nó trên mặt phẳng sao cho không có cạnh nào cắt nhau.

15. Cho G là đơn đồ thị phẳng liên thông với 20 đỉnh và mỗi đỉnh của nó đều có bậc là 3. Hỏi rằng khi vẽ G trên mặt phẳng thì mặt phẳng bị chia làm bao nhiêu phần ?

16. Bài toán tô màu đồ thị: Cho đơn đồ thị vô hướng $G = (V,E)$. Hãy tìm cách gán cho mỗi đỉnh của đồ thị một màu sao cho hai đỉnh kề nhau không bị tô bởi cùng một màu. Một phép gán màu cho các đỉnh như vậy được gọi là một phép tô màu đồ thị. Bài toán tô màu đòi hỏi tìm phép tô màu với số màu phải sử dụng là ít nhất.

Ta gọi sắc số của đồ thi G, ký hiệu là $\chi(G)$, là số màu ít nhất cần dùng để tô màu đồ thị. Dưới đây là một số kết quả liên quan đến tô màu đồ thị

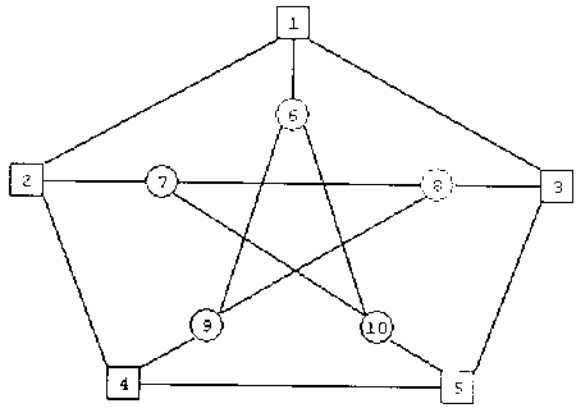
a) Định lý 4 màu: "Mọi đồ thị phẳng đều có thể tô bởi 4 mầu".

b) Đơn đồ thị G là hai phía khi và chỉ khi $\chi(G) = 2$.

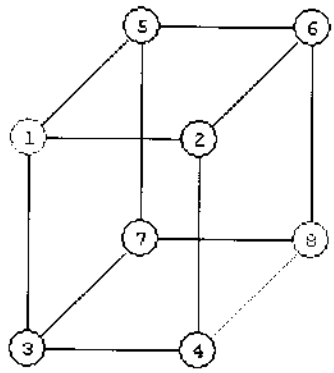
Hãy chứng minh mệnh đề b).

17. Hãy tính sắc số của các đồ thị cho trong các hình vẽ sau

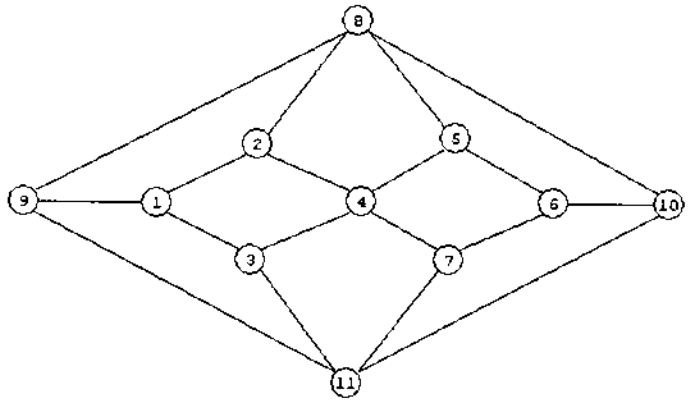
a) Đồ thị Petersen



b) Đồ thị lập phương



c) Đồ thị Herschel



2

BIỂU DIỄN ĐỒ THỊ TRÊN MÁY TÍNH

Để lưu trữ đồ thị và thực hiện các thuật toán khác nhau với đồ thị trên máy tính cần phải tìm những cấu trúc dữ liệu thích hợp để mô tả đồ thị. Việc chọn cấu trúc dữ liệu nào để biểu diễn đồ thị có tác động rất lớn đến hiệu quả của thuật toán. Vì vậy, việc chọn lựa cấu trúc dữ liệu để biểu diễn đồ thị phụ thuộc vào từng tình huống cụ thể (bài toán và thuật toán cụ thể). Trong mục này chúng ta sẽ xét một số phương pháp cơ bản được sử dụng để biểu diễn đồ thị trên máy tính, đồng thời cũng phân tích một cách ngắn gọn những ưu điểm cũng như những nhược điểm của chúng.

2.1. Ma trận kề. Ma trận trọng số

Xét đơn đồ thị vô hướng $G = (V, E)$, với tập đỉnh $V = \{1, 2,..., n\}$, tập cạnh $E=\{e_1, e_2,..., e_m\}$. Ta gọi ma trận kề của đồ thị G là (0,1)-ma trận

$$A = \{a_{ij} : i, j = 1, 2,..., n\}$$

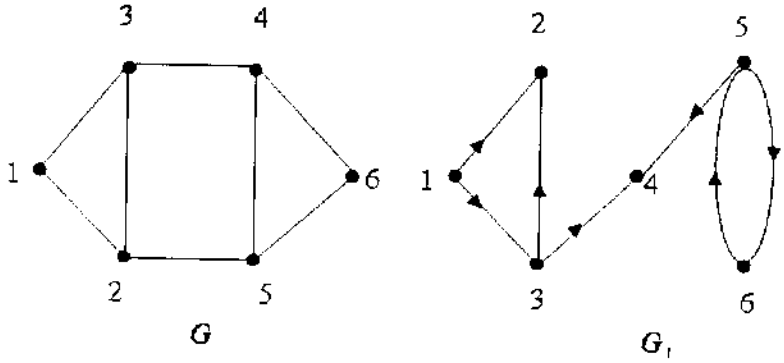
với các phần tử được xác định theo quy tắc sau đây:

$$a_{ij} = 0, \text{ nếu } (i,j) \notin E \quad \text{và} \quad a_{ij} = 1, \text{ nếu } (i,j) \in E,$$

$i, j = 1,2,..., n.$

Thí dụ 1. Ma trận kề của đồ thị vô hướng G cho trong hình 1 là

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$



Hình 1. Đồ thị vô hướng G và Đồ thị có hướng G_1

Các tính chất của ma trận kề:

1) Rõ ràng ma trận kề của đồ thị vô hướng là ma trận đối xứng, tức là

$$a[i, j] = a[j, i], \quad i, j = 1, 2,..., n.$$

Ngược lại, mỗi (0,1)-ma trận đối xứng cấp n sẽ tương ứng, chính xác đến cách đánh số đỉnh (còn nói là: *chính xác đến đẳng cấu*) , *với một đơn đồ thị vô hướng n đỉnh.*

2) Tổng các phần tử trên dòng i (cột j) của ma trận kề chính bằng bậc của đỉnh i (đỉnh *j*).

3) Nếu ký hiệu

$$a_{ij}^{p}, \ i, j = 1, 2,..., n$$

là các phần tử của ma trận tích

$$A^p = \underbrace{A.A\ldots A}_{p}$$

Khi đó

$$a_{ij}^p\ ,\ i, j = 1,\ 2,\ldots,\ n$$

cho ta số đường đi khác nhau từ đỉnh i đến đỉnh j qua p-1 đỉnh trung gian.

Ma trận kề của đồ thị có hướng được định nghĩa một cách hoàn toàn tương tự.

Thí dụ 2. Đồ thị có hướng G_1 cho trong hình 1 có ma trận kề là ma trận sau

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

Lưu ý rằng ma trận kề của đồ thị có hướng không phải là ma trận đối xứng.

Chú ý: Trên đây chúng ta chỉ xét đơn đồ thị. Ma trận kề của đa đồ thị có thể xây dựng hoàn toàn tương tự, chỉ khác là thay vì ghi 1 vào vị trí $a[i,j]$ nếu (i, j) là cạnh của đồ thị, chúng ta sẽ ghi k là số cạnh nối hai đỉnh i và j.

Trong rất nhiều vấn đề ứng dụng của lý thuyết đồ thị, mỗi cạnh $e = (u, v)$ của đồ thị được gán với một con số $c(e)$ (còn viết là $c(u,v)$) gọi là trọng số của cạnh e. Đồ thị trong trường hợp như vậy được gọi là đồ thị có trọng số. Trong trường hợp đồ thị có trọng số, thay vì ma trận kề, để biểu diễn đồ thị ta sử dụng ma trận trọng số

$$C = c[i, j],\ i, j = 1, 2,\ldots, n,$$

với

$$c[i, j] = c(i, j), \qquad \text{nếu } (i, j) \in E$$

và

$$c[i, j] = \theta, \qquad \text{nếu } (i, j) \notin E,$$

trong đó số 0 , tuỳ từng trường hợp cụ thể, có thể được đặt bằng một trong các giá trị sau: $0, +\infty, -\infty$.

Ưu điểm lớn nhất của phương pháp biểu diễn đồ thị bằng ma trận kề (hoặc ma trận trọng số) là để trả lời câu hỏi: Hai đỉnh u, v có kề nhau trên đồ thị hay không, chúng ta chỉ phải thực hiện một phép so sánh. Nhược điểm lớn nhất của phương pháp này là:

không phụ thuộc vào số cạnh của đồ thị, ta luôn phải sử dụng n^2 đơn vị bộ nhớ để lưu trữ ma trận kề của nó.

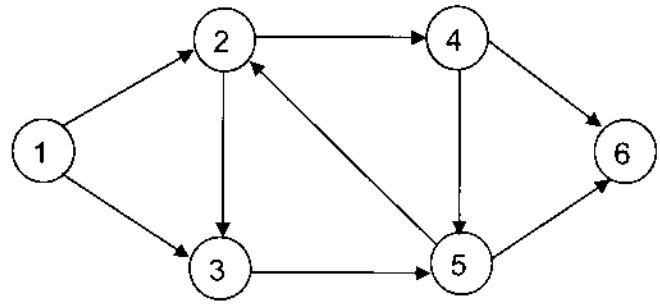
2.2. Ma trận liên thuộc đỉnh-cạnh

Xét $G = (V, E)$, ($V = \{1, 2, ..., n\}$, $E = \{e_1, e_2, ..., e_m\}$), là đơn đồ thị có hướng. Xây dựng ma trận $A = (a_{ij}: i = 1, 2, ..., n; j = 1, 2, ..., m)$, trong đó

$$a_{ij} = \begin{cases} 1, & \text{nếu đỉnh } i \text{ là đỉnh đầu của cung } e_j \\ -1, & \text{nếu đỉnh } i \text{ là đỉnh cuối của cung } e_j \\ 0, & \text{nếu đỉnh } i \text{ không là đầu mút của cung } e_j \end{cases}$$

Ma trận A xây dựng theo qui tắc vừa nêu được gọi là ma trận liên thuộc đỉnh-cạnh.

Ví dụ. Xét đồ thị cho trên hình 3



$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccccc} (1,2) & (1,3) & (2,3) & (2,4) & (3,5) & (4,5) & (4,6) & (5,2) & (5,6) \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \end{bmatrix} \end{array}$$

Hình 2. Đồ thị có hướng và ma trận liên thuộc đỉnh cạnh

Ma trận liên thuộc đỉnh-cạnh là một trong những cách biểu diễn rất hay được sử dụng trong các bài toán liên quan đến đồ thị có hướng mà trong đó phải xử lý các cung của đồ thị.

2.3. Danh sách cạnh (cung)

Trong trường hợp đồ thị thưa (đồ thị có số cạnh m thoả mãn bất đẳng thức: $m < 6n$) người ta thường dùng cách biểu diễn đồ thị dưới dạng *danh sách cạnh.*

Trong cách biểu diễn đồ thị bởi danh sách cạnh (cung) chúng ta sẽ lưu trữ danh sách tất cả các cạnh (cung) của đồ thị vô hướng (có hướng). Mỗi cạnh (cung) $e = (x, y)$ của đồ thị sẽ tương ứng với hai biến Dau[e], Cuoi[e]. Như vậy, để lưu trữ đồ thị ta cần sử dụng $2m$ đơn vị bộ nhớ. Nhược điểm của cách biểu diễn này là để xác định những đỉnh nào của đồ thị là kề với một đỉnh cho trước chúng ta phải làm cỡ m phép so sánh (khi duyệt qua danh sách tất cả các cạnh của đồ thị).

Chú ý: Trong trường hợp đồ thị có trọng số ta cần thêm m đơn vị bộ nhớ để lưu trữ trọng số của các cạnh.

Thí dụ 3. Danh sách cạnh (cung) của đồ thị G (G_1) cho trong hình 1 là:

Dau	Cuoi
1	2
1	3
1	5
2	3
2	5
3	4
4	5
4	6
5	6

Danh sách cạnh của G

Dau	Cuoi
1	2
1	3
3	2
3	4
5	4
5	6
6	5

Danh sách cung của G_1

2.4. Danh sách kề

Trong rất nhiều vấn đề ứng dụng của lý thuyết đồ thị, cách biểu diễn đồ thị dưới dạng *danh sách kề* là cách biểu diễn thích hợp nhất được sử dụng.

Trong cách biểu diễn này, với mỗi đỉnh v của đồ thị chúng ta lưu trữ danh sách các đỉnh kề với nó, mà ta sẽ ký hiệu là $Ke(v)$, tức là

$$Ke(v) = \{ u \in V: (v,u) \in E \}.$$

Khi đó vòng lặp thực hiện với mỗi một phần tử trong danh sách này theo thứ tự các phần tử được xắp xếp trong nó sẽ được viết như sau:

for $u \in Ke(v)$ **do...**

Chẳng hạn, trên PASCAL có thể mô tả danh sách này như sau (Gọi là cấu trúc **Forward Star**):

```
Const
    m = 1000; { m - số cạnh}
    n =  100;  { n -số đỉnh}
var
    Ke : array[1..m] of  integer;
    Tro: array[1..n+1] of  integer;
```

trong đó Tro[i] ghi nhận vị trí bắt đầu của danh sách kề của đỉnh i, i=1,2,...,n, Tro[n+1] = 2m+1.

Khi đó dòng lệnh qui ước

```
for u ∈ Ke(v) do
begin
    .......
end;
```

có thể thay thế bởi cấu trúc lệnh cụ thể trên PASCAL sau

```
for i:= Tro[v] to Tro[v+1]-1 do
begin
   u:=Ke[i];
   . . . . . .
end;
```

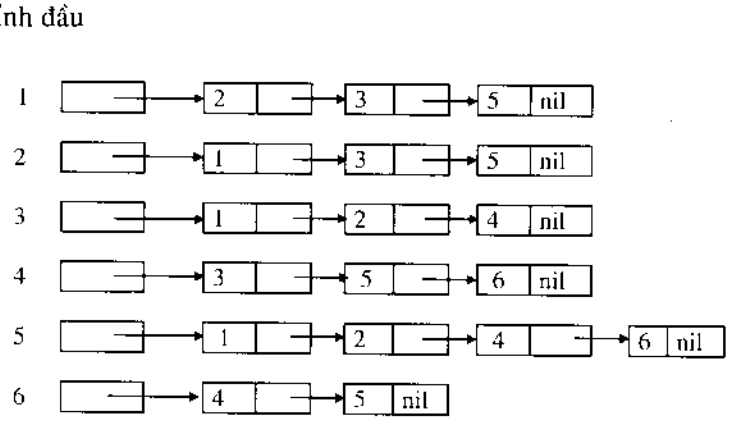
Trong rất nhiều thuật toán làm việc với đồ thị chúng ta thường xuyên phải thực hiện các thao tác: Thêm hoặc bớt một số cạnh. Trong trường hợp này cấu trúc dữ liệu dùng ở trên là không thuận tiện. Khi đó nên chuyển sang sử dụng danh sách kề liên kết (***Linked Adjancency List***) như mô tả trong chương trình nhập danh sách kề của đồ thị từ bàn phím và đưa danh sách đó ra màn hình sau đây:

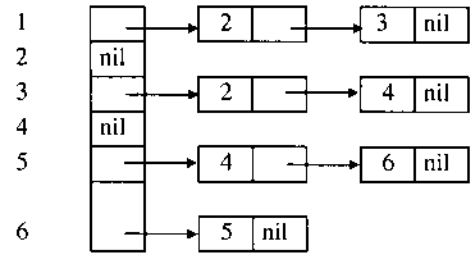
```
program AdjList;
const
     maxV = 100;
type
    link = ^node;
    node = record
                v    : integer;
                next : link
           end;
```

```
var
    j, x, y, m, n, u, v : integer;
    t                   : link;
    Ke                  : array[1..maxV] of link;

BEGIN
    write('Cho số cạnh và đỉnh của đồ thị: '); readln( m,n);
    (*  Khởi tạo  *)
    for  j:=1 to n do Ke[j]:=nil;
    for j:=1 to m do
    begin
        write('Cho đỉnh đầu và cuối của cạnh ',j, ': ');
        readln(x,y);
       new(t); t^.v:=x; t^.next:=Ke[y]; Ke[y]:=t;
       new(t); t^.v:=y; t^.next:=Ke[x]; Ke[x]:=t;
    end;
    writeln('Danh sách kề của các  đỉnh của đồ thị: ');
    for j:=1 to m do
    begin
        writeln('Danh sách các đỉnh kề của đỉnh ',j, ': ');
        t:=Ke[j];
        while t^.next < > nil do
        begin
            write(t^.v:4);
            t:=t^.next;
        end;
    end;
    readln;
END.
```

Thí dụ 4. Danh sách kề của các đồ thị trong hình 1 được mô tả trong hình sau:





Hình 2. Danh sách kề của đồ thị vô hướng G
và có hướng G_1 cho trong hình 1.

Để ý rằng trong cách biểu diễn này chúng ta cần phải sử dụng cỡ $m+n$ đơn vị bộ nhớ.

Trong các thuật toán mô tả ở các phần tiếp theo hai cấu trúc danh sách kề và ma trận trọng số được sử dụng thường xuyên.

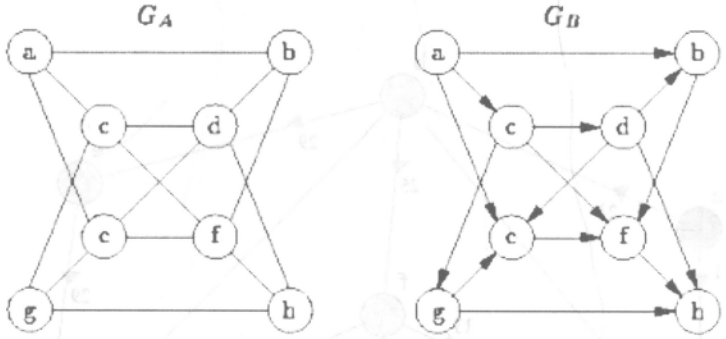
Bài tập

1. Lập trình nhập đồ thị với các cấu trúc dữ liệu đã mô tả.

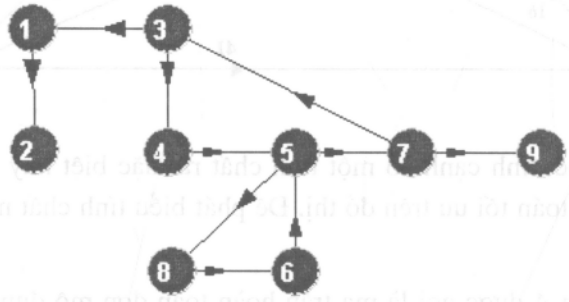
2. Lập trình cho phép chuyển đổi từ cấu trúc dữ liệu biểu diễn đồ thị dưới dạng ma trận kề sang danh sách kề và ngược lại.

3. Hãy xây dựng ma trận kề, danh sách kề của các đồ thị cho trong các hình vẽ sau đây

a)

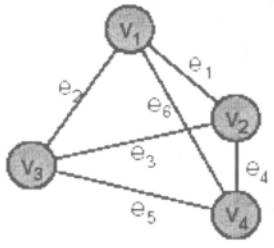


b)

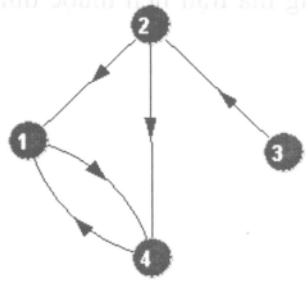


4. Hãy xây dựng ma trận liên thuộc đỉnh cạnh của các đồ thị cho trong các hình vẽ sau

a)

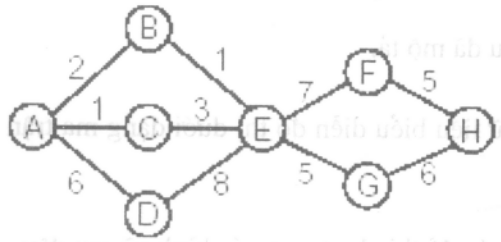


b)

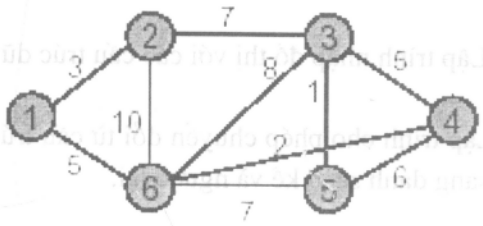


5. Hãy xây dựng ma trận trọng số của các đồ thị cho trong các hình vẽ sau

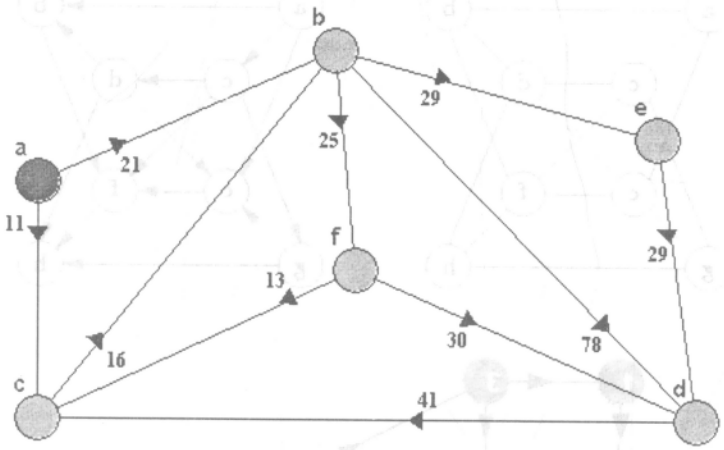
a)



b)



c)



6. Ma trận liên thuộc đỉnh cạnh có một tính chất rất đặc biệt hay được sử dụng trong việc phát triển thuật toán tối ưu trên đồ thị. Để phát biểu tính chất này ta cần khái niệm sau.

Định nghĩa. Ma trận A được gọi là ma trận hoàn toàn đơn mô đun, nếu mọi định thức con khác không của nó đều có trị tuyệt đối là bằng 1.

Chứng minh rằng ma trận liên thuộc đỉnh cạnh của đơn đồ thị có hướng là hoàn toàn đơn mô đun.

3

CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ VÀ ỨNG DỤNG

Rất nhiều thuật toán trên đồ thị được xây dựng dựa trên cơ sở duyệt tất cả các đỉnh của đồ thị sao cho mỗi đỉnh của nó được viếng thăm đúng một lần. Vì vậy, việc xây dựng những thuật toán cho phép duyệt một cách hệ thống tất cả các đỉnh của đồ thị là một vấn đề quan trọng thu hút sự quan tâm nghiên cứu của nhiều tác giả. Những thuật toán như vậy chúng ta sẽ gọi là thuật toán tìm kiếm trên đồ thị. Các thuật toán này giữ một vai trò quan trọng trong việc thiết kế các thuật toán trên đồ thị. Trong mục này chúng ta sẽ giới thiệu hai thuật toán tìm kiếm cơ bản trên đồ thị: ***Thuật toán tìm kiếm theo chiều sâu*** (Depth First Search) và ***Thuật toán tìm kiếm theo chiều rộng*** (Breadth First Search) và ứng dụng của chúng vào việc giải một số bài toán trên đồ thị.

Trong mục này chúng ta sẽ xét đồ thị vô hướng $G = (V, E)$, với n đỉnh và m cạnh. Chúng ta sẽ quan tâm đến việc đánh giá hiệu quả của các thuật toán trên đồ thị, mà một trong những đặc trưng quan trọng nhất là *độ phức tạp tính toán*, tức là *số phép toán mà thuật toán cần phải thực hiện trong tình huống xấu nhất* được biểu diễn như là *hàm của kích thước đầu vào* của bài toán. Trong các thuật toán trên đồ thị, đầu vào là đồ thị $G = (V, E)$, vì vậy, kích thước của bài toán là số đỉnh n và số cạnh m của đồ thị. Khi đó độ phức tạp tính toán của thuật toán sẽ được biểu diễn như là hàm của hai biến số $f(n,m)$ là số phép toán nhiều nhất cần phải thực hiện theo thuật toán đối với mọi đồ thị với n đỉnh

và m cạnh. Khi so sánh tốc độ tăng của hai hàm nhận giá trị không âm $f(n)$ và $g(n)$ chúng ta sẽ sử dụng ký hiệu sau:

$f(n) = O(g(n))$

$\Leftrightarrow$ tìm được các hằng số $C, N > 0$ sao cho

$f(n) \le C\, g(n)$ với mọi $n \ge N$.

Tương tự như vậy nếu $f(n_1, n_2,..., n_k)$, $g(n_1, n_2,...,n_k)$ là các hàm nhiều biến, ta viết

$f(n_1, n_2,...,n_k) = O(g(n_1, n_2,...,n_k))$

$\Leftrightarrow$ tìm được các hằng số $C, N > 0$ sao cho

$f(n_1, n_2,...,n_k) \le C\, g(n_1, n_2,...,n_k)$ với mọi $n_1, n_2,...,n_k \ge N$.

Nếu độ phức tạp tính toán của thuật toán là $O(g(n))$ thì ta sẽ còn nói là nó đòi hỏi thời gian tính cỡ $O(g(n))$.

3.1. Tìm kiếm theo chiều sâu trên đồ thị

Ý tưởng chính của thuật toán có thể trình bày như sau. Ta sẽ bắt đầu tìm kiếm từ một đỉnh v_0 nào đó của đồ thị. Sau đó chọn u là một đỉnh tuỳ ý kề với v_0 và lặp lại quá trình đối với u. ở bước tổng quát, giả sử ta đang xét đỉnh v. Nếu như trong số các đỉnh kề với v tìm được đỉnh w là chưa được xét thì ta sẽ xét đỉnh này (nó sẽ trở thành đã xét) và bắt đầu từ nó ta sẽ tiếp tục quá trình tìm kiếm. Còn nếu như không còn đỉnh nào kề với v là chưa xét thì ta sẽ nói rằng đỉnh này là đã duyệt xong và quay trở lại tiếp tục tìm kiếm từ đỉnh mà trước đó ta đến được đỉnh v (nếu $v = v_0$ thì kết thúc tìm kiếm). Có thể nói nôm na là tìm kiếm theo chiều sâu bắt đầu từ đỉnh v được thực hiện trên cơ sở tìm kiếm theo chiều sâu từ tất cả các đỉnh chưa xét kề với v. Quá trình này có thể mô tả bởi thủ tục đệ qui sau đây.

```
procedure DFS(v);
(*  Tìm kiếm theo chiều sâu bắt đầu từ đỉnh v;
    Các biến Chuaxet, Ke là biến toàn cục  *)
begin
     Thăm_đỉnh(v);
     Chuaxet[v]:=false;
     for  u ∈ Ke(v)  do
              if   Chuaxet[u]  then  DFS(u);
end;  (* đỉnh v là đã duyệt xong *)
```

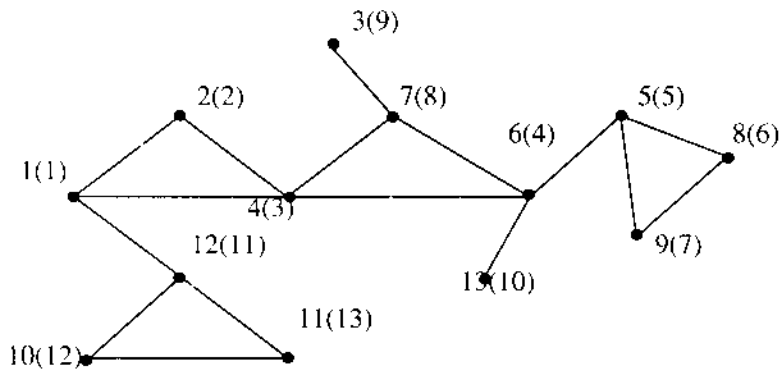
Khi đó, Tìm kiếm theo chiều sâu trên đồ thị được thực hiện nhờ thuật toán sau:

```
BEGIN
   (*   Initialization  *)
   for  v ∈ V  do Chuaxet[v]:=true;
   for  v ∈ V  do
          if   Chuaxet[v] then  DFS(v);
END.
```

Rõ ràng lệnh gọi DFS(v) sẽ cho phép đến thăm tất cả các đỉnh thuộc cùng thành phần liên thông với đỉnh v, bởi vì sau khi thăm đỉnh là lệnh gọi đến thủ tục DFS đối với tất cả các đỉnh kề với nó. Mặt khác, do mỗi khi thăm đỉnh v xong, biến Chuaxet[v] được đặt lại giá trị *false* nên mỗi đỉnh sẽ được thăm đúng một lần. Thuật toán lần lượt sẽ tiến hành tìm kiếm từ các đỉnh chưa được thăm, vì vậy, nó sẽ xét qua tất cả các đỉnh của đồ thị (không nhất thiết phải là liên thông).

Để đánh giá độ phức tạp tính toán của thủ tục, trước hết nhận thấy rằng số phép toán cần thực hiện trong hai chu trình của thuật toán (hai vòng for ở chương trình chính) là cỡ n. Thủ tục DFS phải thực hiện không quá n lần. Tổng số phép toán cần phải thực hiện trong các thủ tục này là $O(n+m)$, do trong các thủ tục này ta phải xét qua tất cả các cạnh và các đỉnh của đồ thị. Vậy độ phức tạp tính toán của thuật toán là $O(n+m)$.

Thí dụ 1. Xét đồ thị cho trong hình 1. Các đỉnh của nó được đánh số lại theo thứ tự chúng được thăm theo thủ tục tìm kiếm theo chiều sâu mô tả ở trên. Giả thiết rằng các đỉnh trong danh sách kề của đỉnh v (Ke(v)) được sắp xếp theo thứ tự tăng dần của chỉ số.



Hình 1. Chỉ số mới (trong ngoặc) của các đỉnh được đánh lại theo thứ tự chúng được thăm trong thuật toán tìm kiếm theo chiều sâu

Thuật toán tìm kiếm theo chiều sâu trên đồ thị vô hướng trình bày ở trên dễ dàng có thể mô tả lại cho đồ thị có hướng. Trong trường hợp đồ thị có hướng, thủ tục DFS(v) sẽ cho phép thăm tất cả các đỉnh u nào mà từ v có đường đi đến u. Độ phức tạp tính toán của thuật toán là $O(n+m)$.

3.2. Tìm kiếm theo chiều rộng trên đồ thị

Để ý rằng trong thuật toán tìm kiếm theo chiều sâu đỉnh được thăm càng muộn sẽ càng sớm trở thành đã duyệt xong. Điều đó là hệ quả tất yếu của việc các đỉnh được thăm sẽ

được kết nạp vào trong ngăn xếp (STACK). Tìm kiếm theo chiều rộng trên đồ thị, nếu nói một cách ngắn gọn, được xây dựng dựa trên cơ sở thay thế ngăn xếp (STACK) bởi hàng đợi (QUEUE). Với sự cải biên như vậy, đỉnh được thăm càng sớm sẽ càng sớm trở thành đã duyệt xong (tức là càng sớm dời khỏi hàng đợi). Một đỉnh sẽ trở thành đã duyệt xong ngay sau khi ta xét xong tất cả các đỉnh kề (chưa được thăm) với nó. Thủ tục có thể mô tả như sau:

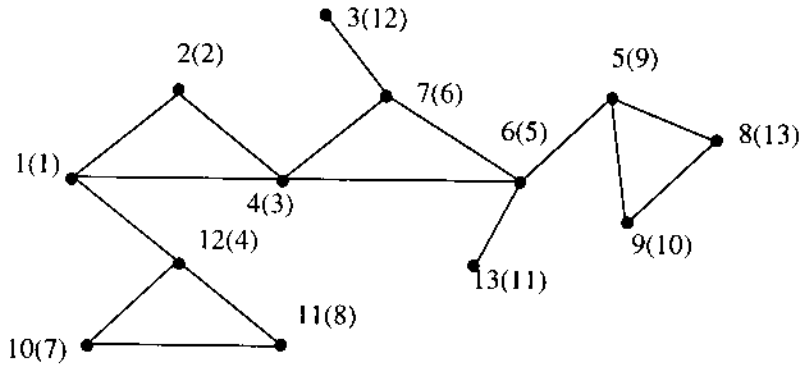
```
procedure BFS(v);
(*  Tìm kiếm theo chiều rộng bắt đầu từ đỉnh v;
    Các biến Chuaxet, Ke là biến toàn cục  *)
begin
     QUEUE:=∅;
     QUEUE ⇐ v; (* Kết nạp v vào QUEUE *)
    Chuaxet[v]:=false;
    while QUEUE ≠ ∅ do
   begin
      p ⇐ QUEUE; (* Lấy p từ QUEUE *)
     Thăm_đỉnh(p);
    for u ∈ Ke(p) do
       if  Chuaxet[u] then
      begin
         QUEUE ⇐ u; Chuaxet[u]:=false;
     end;
   end;
end;
```

Khi đó, Tìm kiếm theo chiều rộng trên đồ thị được thực hiện nhờ thuật toán sau:

```
BEGIN
   (*  Initialization *)
    for v ∈ V do Chuaxet[v]:=true;
    for v ∈ V do
          if  Chuaxet[v] then BFS(v);
END.
```

Lập luận tương tự như trong thủ tục tìm kiếm theo chiều sâu, có thể chỉ ra được rằng lệnh gọi BFS(v) sẽ cho phép đến thăm tất cả các đỉnh thuộc cùng thành phần liên thông với đỉnh v, và mỗi đỉnh của đồ thị sẽ được thăm đúng một lần. Độ phức tạp tính toán của thuật toán là $O(n+m)$.

Thí dụ 2. Xét đồ thị trong hình 2. Thứ tự thăm đỉnh của đồ thị này theo thuật toán tìm kiếm theo chiều rộng được ghi trong ngoặc.



Hình 2. Chỉ số mới (trong ngoặc) của các đỉnh được đánh lại theo thứ tự chúng được thăm trong thuật toán tìm kiếm theo chiều sâu

3.3. Tìm đường đi và kiểm tra tính liên thông

Trong mục này ta xét ứng dụng các thuật toán tìm kiếm mô tả trong các mục trước vào việc giải hai bài toán cơ bản trên đồ thị: bài toán tìm đường đi và bài toán về xác định các thành phần liên thông của đồ thị.

a) Bài toán tìm đường đi giữa hai đỉnh: *Giả sử s và t là hai đỉnh nào đó của đồ thị. Hãy tìm đường đi từ s đến t.*

Như trên đã phân tích, thủ tục DFS(s) (BFS(s)) sẽ cho phép thăm tất cả các đỉnh thuộc cùng một thành phần liên thông với s. Vì vậy, sau khi thực hiện xong thủ tục, nếu Chuaxet[t]=true, thì điều đó có nghĩa là không có đường đi từ s đến t, còn nếu Chua xet[t]=false thì t thuộc cùng thành phần liên thông với s, hay nói một cách khác: tồn tại đường đi từ s đến t. Trong trường hợp tồn tại đường đi, để ghi nhận đường đi, ta dùng thêm biến Truoc[v] để ghi nhận đỉnh đi trước đỉnh v trong đường đi tìm kiếm từ s đến v. Khi đó, đối với thủ tục DFS(v) cần sửa sửa đổi câu lệnh **if** trong nó như sau:

```
if Chuaxet[u] then
begin
   Truoc[u]:=v;
   DFS(u);
end;
```

Còn đối với thủ tục BFS(v) cần sửa sửa đổi câu lệnh **if** trong nó như sau:

```
if Chuaxet[u] then
begin
   QUEUE ⇐ u; Chuaxet[u]:=false;
   Truoc[u]:=p;
end;
```

Đường đi cần tìm sẽ được khôi phục theo quy tắc sau:

$$t \leftarrow p1:=Truoc[t] \leftarrow p2:=Truoc[p1] \leftarrow \ldots \leftarrow s .$$

Chú ý: Đường đi tìm được theo thuật toán tìm kiếm theo chiều rộng là ***đường đi ngắn nhất*** (theo số cạnh) từ đỉnh *s* đến đỉnh *t*. Điều này suy trực tiếp từ thứ tự thăm đỉnh theo thuật toán tìm kiếm theo chiều rộng.

b) Tìm các thành phần liên thông của đồ thị: *Hãy cho biết đồ thị gồm bao nhiêu thành phần liên thông và từng thành phần liên thông của nó là gồm những đỉnh nào.*

Do thủ tục DFS(s) (BFS(s)) cho phép thăm tất cả các đỉnh thuộc cùng một thành phần liên thông với *s*, nên số thành phần liên thông của đồ thị chính bằng số lần gọi đến thủ tục này. Vấn đề còn lại là cách ghi nhận các đỉnh trong từng thành phần liên thông. Ta dùng thêm biến *Index[v]* để ghi nhận chỉ số của thành phần liên thông chứa đỉnh *v*, và dùng thêm biến *Inconnect* để đếm số thành phần liên thông (biến này cần được khởi tạo giá trị là 0). Thủ tục Thăm_đỉnh(v) trong các thủ tục DFS(v) và BFS(v) có nhiệm vụ gán: *Index[v]:= Inconnect*, còn câu lệnh **if** trong các chương trình chính gọi đến các thủ tục này cần được sửa lại như sau

```
Inconnect:=0;
if  Chuaxet[v]  then
begin
    Inconnect:=Inconnect+1;
     DFS(v);  (* BFS(v) *)
end;
```

Kết thúc vòng lặp thứ hai trong chương trình chính, Inconect cho số thành phần liên thông của đồ thị, còn biến mảng Index[v], v $\in V$ cho phép liệt kê các đỉnh thuộc cùng một thành phần liên thông.

Chương trình trên PASCAL giải hai bài toán nói trên có thể viết như sau.

```
{ CHUONG TRINH TIM DUONG DI VA KIEM TRA TINH LIEN THONG
      THEO CAC THUAT TOAN TIM KIEM TREN DO THI
      ========================================          }
uses crt;
var
  a                    : array[1..20,1..20] of byte;
  QUEUE, Chuaxet, Truoc  : array[1..20] of byte;
  i,j,n,Solt,k,s,t     : integer;
  stop                 : boolean;
  ch                   : char;
```

```
procedure Nhapsolieu;
begin
  write('Cho so dinh cua do thi:');readln(n);
  writeln('Nhap so lieu ma tran ke:');
  for i:=1 to n do
  begin
    for j:= i+1 to n do
    begin
      write(' a[',i,',',j,']='); read(a[i,j]); a[j,i]:=a[i,j];
    end;
     a[i,i]:=0; writeln;
  end;
end;

procedure ReadFile;
var   f:text;  fn:string;
begin
  write('Cho ten file du lieu:');readln(fn);
  assign(f,fn);reset(f); readln(f,n);
  writeln('Nhap so lieu ma tran ke:');
  for i:=1 to n do
    for j:=1 to n do read(f,a[i,j]);
  close(f);
end;

procedure Insolieu;
begin
  writeln('Ma tran ke:');
  for i:=1 to n do
  begin
     for j:=1 to n do write(a[i,j]:3); writeln;
  end;
end;

procedure Ketqualienthong;
begin
   Insolieu;
   if  Solt=1 then writeln('Do thi la lien thong')
   else
   begin
      writeln('So thanh phan lien thong cua do thi la: ',Solt);
```

```
      for i:=1 to Solt do
      begin
        writeln('Thanh phan lien thong thu ',i, ' gom cac dinh:');
        for j:=1 to n do if Chuaxet[j]=i then write(j:3); writeln;
      end;
    end;
    write('Go Enter de tiep tuc...'#7);readln;
end;

procedure BFS(i:integer);
(* Tìm kiếm theo chiều rộng bắt đầu từ đỉnh i *)
var u, dauQ, cuoiQ: integer;
begin
  dauQ:=1; cuoiQ:=1;
  QUEUE[cuoiQ]:=i; Chuaxet[i]:=Solt;
  while dauQ <= cuoiQ do
  begin
    u:=QUEUE[dauQ]; dauQ:=dauQ+1;
    for j:=1 to n do
     if (a[u,j]=1)and(Chuaxet[j]=0) then
     begin
       cuoiQ:=cuoiQ+1; QUEUE[cuoiQ]:=j;
       Chuaxet[j]:=Solt;  Truoc[j]:=u;
     end;
  end;
end; { of procedure BFS }

procedure DFS(v:integer);
(* Tìm kiếm theo chiều sâu bắt đầu từ đỉnh v *)
var u:integer;
begin
  Chuaxet[v]:=Solt;
  for u:=1 to n do
   if (a[v,u]=1)and(Chuaxet[u]=0) then
   begin
     Truoc[u]:=v; DFS(u);
   end;
end;

procedure Lienthong;
begin
  { Khoi tao so lieu }
```

```
  for j:=1 to n do Chuaxet[j]:=0; Solt:=0;
  for i:=1 to n do
    if Chuaxet[i]=0 then
    begin
      Solt:=Solt+1;
      { BFS(i);} DFS(i);
    end;
  Ketqualienthong;
end;

procedure Ketquaduongdi;
begin
  if Truoc[t]=0 then writeln('Khong co duong di tu ',s,' den ',t)
  else
  begin
    writeln('Duong di tu ',s,' den ',t,' la:');
    j:=t;
    write(t,' <== ');
    while Truoc[j]<>s do
    begin
      write(Truoc[j],' <== ');
      j:=Truoc[j];
    end;
    writeln(s);
  end;
  write('Go Enter de tiep tuc...'#7);readln;
end;

procedure Duongdi;
begin
  Insolieu;
  write(' Tim duong di tu dinh: '); readln(s);
  write('          den dinh: '); readln(t);
  for j:=1 to n do   { Khoi tao so lieu }
  begin
    Truoc[j]:=0; Chuaxet[j]:=0;
  end;
  Solt:=1; BFS(s); { DFS(s); }
  Ketquaduongdi;
end;
```

```
procedure Menu;
begin
  clrscr;
  writeln('          TIM DUONG DI VA KIEM TRA TINH LIEN THONG');
  writeln('  CUA DO THI THEO THUAT TOAN TIM KIEM TREN DO THI');
  writeln('     ==========================================');
  writeln('     1. Nhap so lieu tu ban phim.');
  writeln('     2. Nhap so lieu tu file.');
  writeln('     3. Kiem tra tinh lien thong.');
  writeln('     4. Tim duong di giua hai dinh.');
  writeln('     5. Thoat.');
  writeln('-------------------------------------------------------');
  write  ('    Hay go phim so de chon chuc nang...'#7);
  ch:=readkey;writeln(ch);
end;

{ Main program }
BEGIN
  repeat
    Menu;
    case ch of
      '1' : Nhapsolieu;
      '2' : ReadFile;
      '3' : Lienthong;
      '4' : Duongdi;
    end;
  until (ch='5') or (upcase(ch)='Q');
END.
```

Bài tập

1. Giả sử đồ thị $G = (V,E)$ được cho bởi danh sách kề. Hãy viết thủ tục loại bỏ cạnh (u,v), thêm cạnh (x,y) vào đồ thị.

2. Thủ tục sau đây cho phép duyệt qua tất cả các đỉnh của đồ thị được cho bởi danh sách kề:

```
procedure Find(x);
begin
      Thamdinh(x);  Chuatham[x]:= Đã thăm;
      while (Trong V còn đỉnh đã thăm) do
       begin
            v:= đỉnh đã thăm nào đó;
            if  (Tìm được đỉnh chưa thăm  u ∈ Ke(v) ) then
            begin
                 u:= đỉnh chưa thăm đầu tiên trong Ke(v);
                 Thamdinh(u);   Chuatham[u]:= Đã thăm;
            end
            else   Chuatham[v] := Đã duyệt xong;
       end;
end;

BEGIN
      for   v ∈ V do  Chuatham[v]:=Chưa thăm;
      for   v ∈ V do
             if   (Chuatham[v]= Chưa thăm)  then  Find(v);
END.
```

Hãy chỉ ra rằng cả hai thủ tục tìm kiếm theo chiều sâu và tìm kiếm theo chiều rộng đều có thể xem như trường hợp riêng của thủ tục này. Viết chương trình trên PASCAL thực hiện thuật toán mô tả ở trên.

3. Áp dụng thủ tục tìm kiếm theo chiều sâu tìm tất cả các cầu trên đồ thị vô hướng. (Cầu là cạnh mà việc loại bỏ nó làm tăng số thành phần liên thông của đồ thị).

4. Áp dụng thủ tục tìm kiếm theo chiều sâu kiểm tra xem đồ thị có hướng $G = (V,A)$ có chứa chu trình hay không?

5. Cho một bảng ô vuông gồm $m \times n$ ô, ô nằm trên dòng i cột j gọi là ô (i,j), $i = 1, 2,..., m$, $j = 1, 2,..., n$. Trong mỗi ô (i, j) của nó ta viết một số a[i,j] $\in \{0,1\}$. Từ một ô nào đó ta chỉ có thể di chuyển sang ô chứa số 1 có chung cạnh với nó . Giả sử cho ô (p, q) là ô

xuất phát, hãy viết chương trình tìm xem có cách đi chuyển từ ô này ra một ô ở mép của bảng hay không? (Ô (u, v) gọi là ô ở mép bảng nếu hoặc là $u = 1$, hoặc là $u = m$, hoặc là $v = 1$, hoặc là $v = n$).

6. Cho một bảng ô vuông gồm $m \times n$ ô, ô nằm trên dòng i cột j gọi là ô (i,j), $i = 1, 2,..., m$, $j = 1, 2,..., n$. Trong mỗi ô (i, j) của nó ta viết một số a[i,j] $\in \{0,1\}$. Hãy viết chương trình đếm số *miền con toàn 0* của bảng. Ví dụ số *miền con toàn 0* của bảng kích thước 5x5 được chỉ ra trong hình sau đây

1	0	1	0	0
1	1	1	1	0
0	0	0	1	0
1	0	1	1	0
1	0	1	1	0

7. Có N người khách (N < 100) mang số hiệu từ 1 đến N, được mời đến dự tiệc. Giữa họ có một số người quen biết nhau. Dữ liệu về mối quan hệ quen biết này được cho trong một file văn bản có tên là KHACH.DAT có cấu trúc như sau: Dòng đầu tiên của nó chứa số lượng khách mời N. Mỗi dòng thứ i trong số N dòng tiếp theo chứa số hiệu của các người quen của khách i, các số hiệu được ghi cách nhau bởi ít nhất một dấu cách.

Người ta muốn xếp các khách này vào các phòng tiệc, sao cho hai khách ở trong cùng một phòng hoặc là quen biết nhau hoặc là có thể làm quen nhau thông qua những người quen biết trung gian của họ.

Viết chương trình nhập dữ liệu vào từ file, sau đó tìm cách phân khách vào các phòng tiệc sao cho số phòng phải sử dụng là ít nhất. Kết quả phân công đưa ra một file văn bản, dòng đầu tiên chứa số phòng tiệc cần sử dụng K, mỗi dòng thứ i trong số K dòng tiếp theo ghi số hiệu của khách xếp vào phòng tiệc i.

8. Bản đồ giao thông được cho bởi n nút giao thông đánh số là 1, 2,..., n và hệ thống gồm m đoạn đường $E_1, E_2,..., E_m$ mỗi đoạn nối 2 trong số n nút giao thông nói trên. Bản đồ giao thông được gọi là liên thông đơn nếu giữa hai nút giao thông bất kỳ chỉ có đúng một đường đi nối chúng. Dữ liệu cho trong file văn bản có tên GT.DAT, dòng đầu tiên chứa các số n, m, hai số ghi cách nhau bởi dấu cách. Mỗi dòng i trong số m dòng tiếp theo chứa số hiệu hai nút giao thông được nối bởi đoạn đường E_i.

Viết chương trình nhập dữ liệu vào từ file sau đó thông báo xem bản đồ có phải là liên thông đơn hay không. Trong trường hợp bản đồ là không liên thông đơn hãy tìm cách xây dựng bổ sung và loại bỏ một số đoạn đường để cho bản đồ là liên thông đơn và sao cho số đoạn đường phải xây dựng bổ sung là ít nhất. Các kết quả đưa ra màn hình.

4

ĐỒ THỊ EULER VÀ ĐỒ THỊ HAMILTON

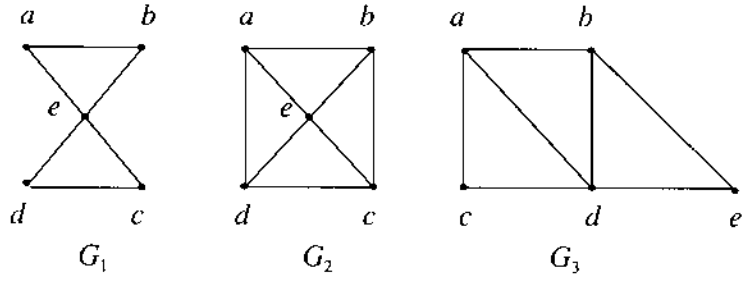
Trong chương này chúng ta sẽ nghiên cứu hai dạng đồ thị đặc biệt là đồ thị Euler và đồ thị Hamilton. Dưới đây, nếu không có giải thích bổ sung, thuật ngữ đồ thị được dùng để chỉ chung đa đồ thị vô hướng và có hướng, và thuật ngữ cạnh sẽ dùng để chỉ chung cạnh của đồ thị vô hướng cũng như cung của đồ thị có hướng.

4.1. Đồ thị Euler

Định nghĩa 1. *Chu trình đơn trong G đi qua mỗi cạnh của nó một lần được gọi là chu trình Euler. Đường đi đơn trong G đi qua mỗi cạnh của nó một lần được gọi là đường đi Euler. Đồ thị được gọi là đồ thị Euler nếu nó có chu trình Euler, và gọi là đồ thị nửa Euler nếu nó có đường đi Euler.*

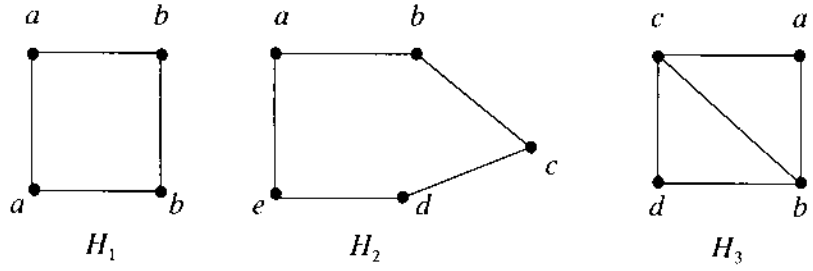
Rõ ràng mọi đồ thị Euler luôn là nửa Euler, nhưng điều ngược lại không luôn đúng.

Thí dụ 1. Đồ thị G_1 trong hình 1 là đồ thị Euler vì nó có chu trình Euler *a, e, c, d, e, b, a*. Đồ thị G_3 không có chu trình Euler nhưng nó có đường đi Euler *a, c, d, e, b, d, a, b*, vì thế G_3 là đồ thị nửa Euler. Đồ thị G_2 không có chu trình cũng như đường đi Euler.



Hình 1. Đồ thị G_1, G_2, G_3

Thí dụ 2. Đồ thị H_2 trong hình 2 là đồ thị Euler vì nó có chu trình Euler *a, b, c, d, e, a*. Đồ thị H_3 không có chu trình Euler nhưng nó có đường đi Euler *c, a, b, c, d, b* vì thế H_3 là đồ thị nửa Euler. Đồ thị H_1 không có chu trình cũng như đường đi Euler.



Hình 2. Đồ thị H_1, H_2, H_3

Điều kiện cần và đủ để một đồ thị là một đồ thị Euler được Euler tìm ra vào năm 1736 khi ông giải quyết bài toán hóc búa nổi tiếng thời đó về bảy cái cầu ở thành phố Konigsberg, và đây là định lý đầu tiên của lý thuyết đồ thị.

Định lý 1 (Euler). *Đồ thị vô hướng liên thông G là đồ thị Euler khi và chỉ khi mọi đỉnh của G đều có bậc chẵn.*

Để chứng minh định lý trước hết ta chứng minh bổ đề:

Bổ đề. *Nếu bậc của mỗi đỉnh của đồ thị G không nhỏ hơn 2 thì G chứa chu trình.*

Chứng minh. Nếu G có cạnh lặp thì khẳng định của bổ đề là hiển nhiên. Vì vậy giả sử G là đơn đồ thị. Gọi v là một đỉnh nào đó của G. Ta sẽ xây dựng theo quy nạp đường đi

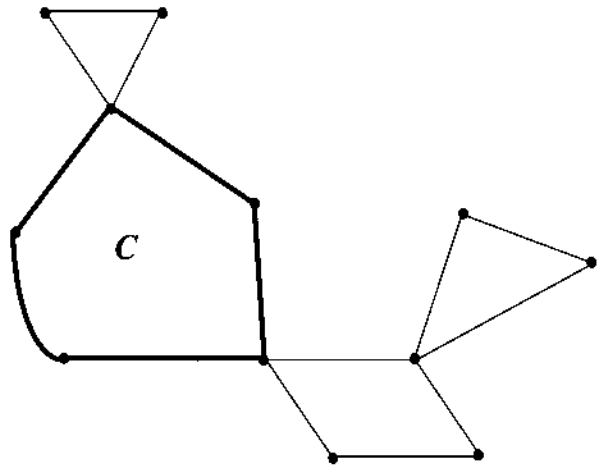
$$v \to v_1 \to v_2 \to \dots$$

trong đó v_1 là đỉnh kề với v, còn với $i \geq 1$ chọn v_{i+1} là kề với v_i và $v_{i+1} \neq v_{i-1}$ (có thể chọn v_{i+1} như vậy là vì $deg(v_i) \geq 2$). Do tập đỉnh của G là hữu hạn, nên sau một số hữu hạn bước ta phải quay lại một đỉnh đã xuất hiện trước đó. Gọi đỉnh đầu tiên như thế là v_k. Khi đó, đoạn của đường đi xây dựng nằm giữa hai đỉnh v_k là 1 chu trình cần tìm.

Chứng minh định lý.

Cần. Giả sử G là đồ thị Euler tức là tồn tại chu trình Euler P trong G. Khi đó cứ mỗi lần chu trình P đi qua 1 đỉnh nào đó của G thì bậc của đỉnh đó tăng lên 2. Mặt khác mỗi cạnh của đồ thị xuất hiện trong P đúng 1 lần, suy ra mỗi đỉnh của đồ thị đều có bậc chẵn.

Đủ. Quy nạp theo số cạnh của G. Do G liên thông và $deg(v)$ là số chẵn nên bậc của mỗi đỉnh của nó không nhỏ hơn 2. Từ đó theo bổ đề G phải chứa chu trình C. Nếu C đi qua tất cả các cạnh của G thì nó chính là chu trình Euler. Giả sử C không đi qua tất cả các cạnh của G. Khi đó loại bỏ khỏi G tất cả các cạnh thuộc C ta thu được 1 đồ thị mới H (không nhất thiết là liên thông). Số cạnh trong H nhỏ hơn trong G và rõ ràng mỗi đỉnh của H vẫn có bậc là chẵn. Theo giả thiết quy nạp trong mỗi thành phần liên thông của H đều tìm được chu trình Euler. Do G là liên thông nên mỗi thành phần trong H có ít nhất 1 đỉnh chung với chu trình C. Vì vậy, ta có thể xây dựng chu trình Euler trong G như sau: Bắt đầu từ một đỉnh nào đó của chu trình C, đi theo các cạnh của chu trình C chừng nào chưa gặp phải đỉnh không cô lập của H. Nếu gặp phải đỉnh như vậy thì ta đi theo chu trình Euler của thành phần liên thông của H chứa đỉnh đó. Sau đó lại tiếp tục đi theo cạnh của C cho đến khi gặp phải đỉnh không cô lập của H thì lại theo chu trình Euler của thành phần liên thông tương ứng trong H v.v... (xem hình 3). Quá trình sẽ kết thúc khi ta trở về đỉnh xuất phát, tức là thu được chu trình đi qua mỗi cạnh của đồ thị đúng một lần.



Hình 3. Minh hoạ cho chứng minh định lý 1

Hệ quả 2. *Đồ thị vô hướng liên thông G là nửa Euler khi và chỉ khi nó có không quá 2 đỉnh bậc lẻ.*

Chứng minh. Thực vậy nếu G có không quá 2 đỉnh bậc lẻ thì số đỉnh bậc lẻ của nó chỉ có thể là 0 hoặc 2. Nếu G không có đỉnh bậc lẻ thì theo định lý 1 nó là đồ thị Euler. Giả sử G có hai đỉnh bậc lẻ là u và v. Gọi H là đồ thị thu được từ G bằng cách thêm vào G một đỉnh mới w và hai cạnh (w,u) và (w,v). Khi đó tất cả các đỉnh của H đều có bậc là chẵn, vì thế theo định lý 1 nó có chu trình Euler C. Xoá bỏ khỏi chu trình này đỉnh w và hai cạnh kề nó ta thu được đường đi Euler trong đồ thị G.

Giả sử G là đồ thị Euler, từ chứng minh định lý ta có thủ tục sau để tìm chu trình Euler trong G.

```
procedure Euler_Cycle;
begin
    STACK := ∅ ;  CE := ∅ ;
    Chọn  u  là một đỉnh nào đó của đồ thị;
    STACK ⇐ u;
    while STACK ≠ ∅ do
     begin
          x:= top(STACK);  (*   x là phần tử  ở đầu STACK *)
          if   Ke(x) ≠ ∅   then
          begin
              y:= đỉnh đầu tiên trong danh sách Ke(x);
              STACK   ⇐ y;
              (*  Loại bỏ cạnh (x,y) khỏi đồ thị  *)
               Ke(x) := Ke(x) \ {y};  Ke(y) := Ke(y) \ {x};
           end  else
           begin   x  ⇐  STACK;  CE ⇐ x;  end;
    end;
end;
```

Giả sử G là đồ thị Euler, thuật toán đơn giản sau đây cho phép xác định chu trình Euler khi làm bằng tay.

Thuật toán Flor

Xuất phát từ 1 đỉnh u nào đó của G ta đi theo các cạnh của nó 1 cách tuỳ ý chỉ cần tuân thủ 2 quy tắc sau:

i) Xoá bỏ cạnh đã đi qua và đồng thời xoá cả những đỉnh cô lập tạo thành.

ii) ở mỗi bước ta chỉ đi qua cầu khi không còn cách lựa chọn nào khác.

Chứng minh tính đúng đắn của thuật toán. Trước tiên ta chỉ ra rằng thủ tục trên có thể thực hiện được ở mỗi bước. Giả sử ta đi đến một đỉnh v nào đó, khi đó nếu $v \neq u$ thì đồ thị con còn lại H là liên thông và chứa đúng hai đỉnh bậc lẻ là v và u. Theo hệ quả trong H có đường đi Euler P từ v đến u. Do việc xoá bỏ cạnh đầu tiên của đường đi P không làm mất tính liên thông của H, từ đó suy ra thủ tục có thể thực hiện ở mỗi bước. Nếu $v = u$ thì lập luận ở trên sẽ vẫn đúng chừng nào vẫn còn cạnh kề với u.

Như vậy chỉ còn phải chỉ ra là thủ tục trên dẫn đến đường đi Euler. Thực vậy trong G không thể còn cạnh chưa đi qua khi mà ta sử dụng cạnh cuối cùng kề với u (trong trường hợp ngược lại, việc loại bỏ 1 cạnh nào đó kề với 1 trong số những cạnh còn lại chưa đi qua sẽ dẫn đến một đồ thị không liên thông, và điều đó là mâu thuẫn với giả thiết ii)).

Chứng minh tương tự như trong định lý 1 ta thu được kết quả sau đây cho đồ thị có hướng.

Định lý 2. *Đồ thị có hướng liên thông mạnh là đồ thị Euler khi và chỉ khi*

$$\deg^+(v) = \deg^-(v), \quad \forall v \in V.$$

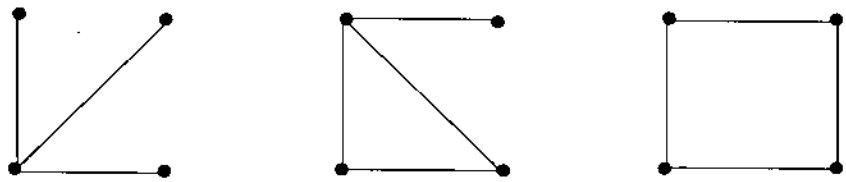
4.2. Đồ thị Hamilton

Trong mục này chúng ta xét bài toán tương tự như trong mục trước chỉ khác là ta quan tâm đến đường đi qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần. Sự thay đổi tưởng chừng như là không đáng kể này trên thực tế đã dẫn đến sự phức tạp hoá vấn đề cần giải quyết.

Định nghĩa 2. *Đường đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần được gọi là đường đi Hamilton. Chu trình bắt đầu từ một đỉnh v nào đó qua tất cả các đỉnh còn lại mỗi đỉnh đúng một lần rồi quay trở về v được gọi là chu trình Hamilton. Đồ thị G được gọi là đồ thị Hamilton nếu nó chứa chu trình Hamilton, và gọi là nửa Hamilton nếu nó chứa đường đi Hamilton.*

Rõ ràng đồ thị Hamilton là nửa Hamilton, nhưng điều ngược lại không luôn đúng.

Thí dụ 3. Trong hình 3: G_3 là Hamilton, G_2 là nửa Hamilton còn G_1 không là nửa Hamilton.



Hình 3. Đồ thị Hamilton G_3, nửa Hamilton G_2, và G_1

Cho đến nay việc tìm một tiêu chuẩn nhận biết đồ thị Hamilton vẫn còn là mở, mặc dù đây là một vấn đề trung tâm của lý thuyết đồ thị. Hơn thế nữa, cho đến hiện nay cũng chưa có thuật toán hiệu quả để kiểm tra một đồ thị có là Hamilton hay không. Các kết quả thu được phần lớn là các điều kiện đủ để một đồ thị là đồ thị Hamilton. Phần lớn chúng đều có dạng *"nếu G có số cạnh đủ lớn thì G là Hamilton"*. Một kết quả như vậy được phát biểu trong định lý sau đây:

Định lý 3 (Dirak 1952). *Đơn đồ thị vô hướng G với $n > 2$ đỉnh, mỗi đỉnh có bậc không nhỏ hơn $n/2$ là đồ thị Hamilton.*

Chứng minh. Thêm vào đồ thị G k đỉnh mới và nối chúng với tất cả các đỉnh của G. Giả sử k là số nhỏ nhất các đỉnh cần thêm vào để cho đồ thị thu được G' là đồ thị Hamilton. Ta sẽ chỉ ra rằng $k = 0$. Thực vậy, giả sử ngược lại là $k > 0$. Ký hiệu

$$v,\ p,\ w,\ ....,\ v$$

là chu trình Hamilton trong G', trong đó v, w là đỉnh của G còn p là một trong số các đỉnh mới. Khi đó w không kề với v vì nếu ngược lại, ta không cần sử dụng p và điều đó là mâu thuẫn với giả thiết k nhỏ nhất. Hơn thế nữa đỉnh (w'chẳng hạn) kề với w không thể đi liền sau đỉnh v' (kề với v) vì rằng khi đó có thể thay

$$v \to p \to w \to ... \to v' \to w' \to ... \to v$$

bởi

$$v \to v' \to ... \to w \to w' ... \to v$$

bằng cách đảo ngược đoạn của chu trình nằm giữa w và v'. Từ đó suy ra là số đỉnh của đồ thị G' không kề với w là không nhỏ hơn số đỉnh kề với nó (tức là ít nhất cũng là bằng $n/2 + k$), đồng thời số đỉnh của G' kề với w cũng ít ra là phải bằng $n/2 + k$. Do không có đỉnh nào của G' vừa không kề, lại vừa kề với w, cho nên tổng số đỉnh của đồ thị G' (G' có $n + k$ đỉnh) không ít hơn $n + 2k$. Mâu thuẫn thu được đã chứng minh định lý.

Định lý sau là tổng quát hoá của định lý Dirak cho đồ thị có hướng:

Định lý 4. *Giả sử G là đồ thị có hướng liên thông mạnh với n đỉnh. Nếu*

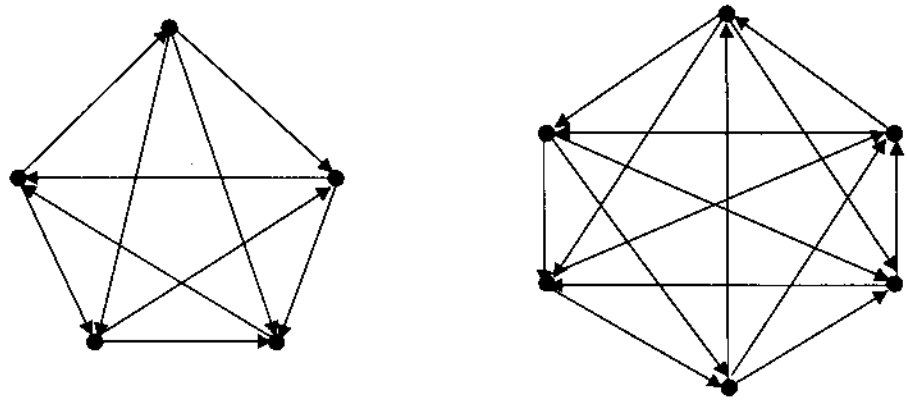
$$\deg^+(v) \geq n/2, \quad \deg^-(v) \geq n/2, \quad \forall v$$

thì G là Hamilton.

Có một số dạng đồ thị mà ta có thể biết khi nào nó là đồ thị Hamilton. Một ví dụ như vậy là đồ thị đấu loại. *Đồ thị đấu loại* là đồ thị có hướng mà trong đó 2 đỉnh bất kỳ của nó được nối với nhau bởi đúng một cung. Tên gọi *đấu loại* xuất hiện vì đồ thị như vậy có thể dùng để biểu diễn kết quả thi đấu bóng chuyền, bóng bàn hay bất cứ một trò chơi nào mà không cho phép hoà. Ta có định lý sau.

Định lý 5. *i) Mọi đồ thị đấu loại là nửa Hamilton;*
ii) Mọi đồ thị đấu loại liên thông mạnh là Hamilton.

Thí dụ 4. Đồ thị đấu loại D_5, D_6 được cho trong hình 4.



D_5 D_6

Hình 4. Đồ thị đấu loại D_5, đấu loại liên thông mạnh D_6

Thuật toán liệt kê tất cả các chu trình Hamilton của đồ thị.

Thuật toán sau đây được xây dựng dựa trên cơ sở thuật toán quay lui cho phép liệt kê tất cả các chu trình Hamilton của đồ thị.

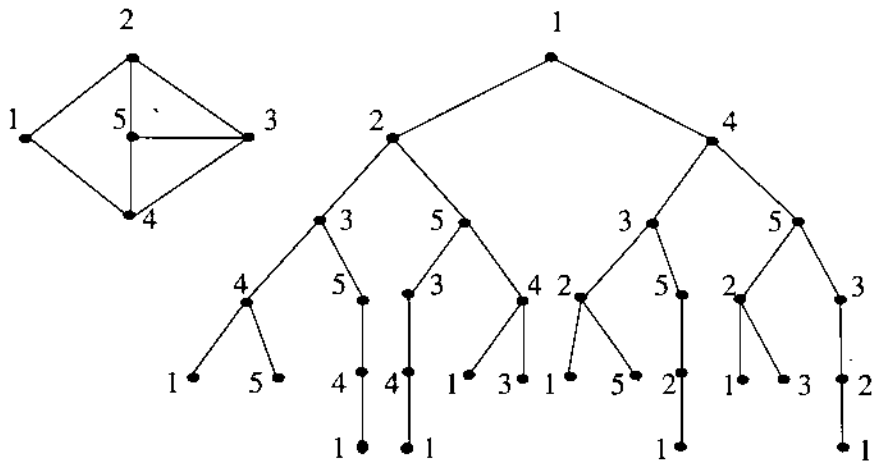
procedure Hamilton(k);

(* *Liệt kê các chu trình Hamilton thu được bằng việc*
phát triển dãy đỉnh (X[1], . . ., X[k-1])
của đồ thị G=(V, E) *cho bởi danh sách kề:* Ke(v), v ∈ V
*)

```
begin
    for  y ∈ Ke(X[k-1]) do
        if  ( k = n+1 )  and ( y = v0 ) then  Ghinhan(X[1],...,X[n],v0)
        else
        if  Chuaxet[y]  then
        begin
            X[k] := y;
            Chuaxet[y] := false;
            Hamilton(k+1);
            Chuaxet[y] := true;
        end;
end;

(*   Main Program  *)
BEGIN
    for  v ∈ V  do  Chuaxet[v] := true;
    X[1] := v0 ;  (*  v0  là một đỉnh nào đó của đồ thị   *)
    Chuaxet[v0] := false ;
    Hamilton(2);
END.
```

Ví dụ 5. Hình 5 dưới đây mô tả cây tìm kiếm theo thuật toán vừa mô tả.



Hình 5. Đồ thị và cây liệt kê chu trình Hamilton của nó theo thuật toán quay lui

Trong trường hợp đồ thị có không quá nhiều cạnh thuật toán trên có thể sử dụng để kiểm tra xem đồ thị có phải là Hamilton hay không.

Bài tập

1. Chứng minh rằng đa đồ thị có hướng có đường đi Euler khi và chỉ khi nó là liên thông yếu và bán bậc vào và bán bậc ra của tất cả các đỉnh là bằng nhau, ngoại trừ hai đỉnh đặc biệt u, v ta có

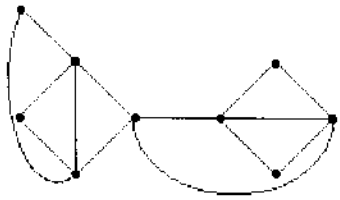
$$\deg^+(u) = \deg^-(u)+1,\ \deg^-(v) = \deg^+(v)+1$$

2. a) Với những giá trị nào của m và n đồ thị hai phía đầy đủ $K_{m,n}$ là đồ thị Euler, đồ thị nửa Euler?

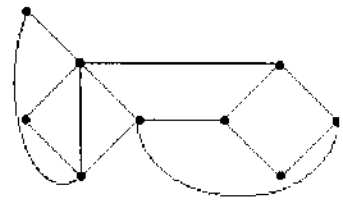
b) Chứng minh rằng đồ thị Q_n luôn có chu trình Hamilton.

3. Với mỗi đồ thị sau sử dụng thuật toán Fleury để đưa ra chu trình Euler của đồ thị hoặc chỉ ra rằng đồ thị không phải là đồ thị Euler:

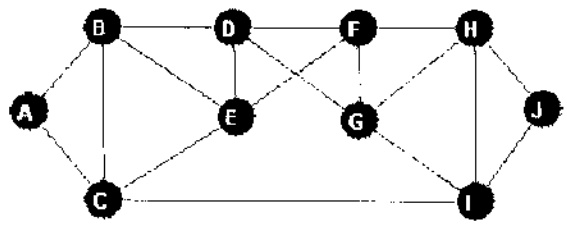
a.



b.



c)



4. Viết chương trình kiểm tra xem một đồ thị có là đồ thị Euler, đồ thị nửa Euler, và trong trường hợp câu trả lời là khẳng định, hãy tìm chu trình Euler hoặc đường đi Euler trong đồ thị theo thuật toán mô tả trong 4.1.

5. Viết chương trình kiểm tra xem một đồ thị có phải là đồ thị Hamilton, đồ thị nửa Hamilton hay không? (Sử dụng thuật toán mô tả trong 4.2)

6. Trong giải vô địch bóng chuyền có n đội đánh số từ 1 đến n thi đấu vòng tròn một lượt. Kết quả thi đấu được cho bởi một bảng số(a[i,j], i, j = 1, 2, ..., n), trong đó a[i, j]=1 nếu đội i thắng trong trận gặp đội j và a[i,j]=0, trong trường hợp đội i thua trong trận gặp đội j. Dữ liệu vào được cho trong một file văn bản có tên là KQTD.INP, dòng đầu tiên chứa số n, các dòng tiếp theo chứa các số a[i,j], i, j =1, 2,..., n, các số cách nhau

bởi dấu cách hoặc dấu xuống dòng. Kết thúc giải, ban tổ chức muốn mời các đội trưởng của đội bóng ra xếp thành một hàng ngang để chụp ảnh. Hãy tìm cách xếp các đội trưởng thành một hàng ngang sao ngoại trừ hai người đứng ở hai mép của hàng mỗi người trong hàng đều đứng cạnh một đội trưởng của đội thắng, một đội trưởng của đội thua đội mình ở trận đấu của giải. Kết quả đưa ra màn hình: thứ tự đứng các đội trưởng của các đội trong hàng ngang tìm được.

7. Tìm cách viết 9 số 1, 9 số 2, 9 số 3 thành dãy

$$a_1, a_2, \ldots, a_{27},$$

sao cho 27 số có 3 chữ số

$$\overline{a_1a_2a_3}, \overline{a_2a_3a_4}, \ldots, \overline{a_{25}a_{26}a_{27}}, \overline{a_{26}a_{27}a_1}, \overline{a_{27}a_1a_2}$$

là đôi một khác nhau.

8. Cho n xâu ký tự S_1, S_2,..., S_n, mỗi xâu có độ dài không quá 80 ký tự. Các xâu này được ghi trong một file văn bản có tên là XAU.INP, mỗi dòng của nó chứa một xâu ký tự nói trên. Giả sử P và Q là hai xâu nào đó trong các xâu đã cho. Ta nói P có thể nối được với Q nếu ký tự đầu tiên của P là trùng với ký tự cuối cùng của Q. Lập trình kiểm tra xem có thể nối n xâu đã cho thành một xâu theo quy tắc nêu trên hay không.

9. Một mạng máy tính gồm n máy tính đánh số từ 1 đến n, trong đó có một máy gọi là máy chủ được đánh số là 1. Hai máy trong mạng có thể được nối với nhau bởi nhiều hơn một kênh truyền tin. Tất cả có m kênh truyền tin giữa các máy được đánh số từ 1, 2,..., m. Dữ liệu nối mạng được cho trong một file văn bản có tên là NET.INP, mỗi dòng của nó chứa thông tin về một kênh truyền tin trong mạng gồm hai số d_i, c_i là chỉ số của hai máy được nối bởi kênh truyên tin thứ i trong mạng (i = 1, 2,..., m). Để kiểm tra các kênh truyền tin, người ta gửi một thông điệp từ máy chủ trong mạng, thông điệp này cần phải lần lượt thông qua các máy trong mạng truyền đi qua tất cả các kênh truyền tin trong mạng mỗi kênh đúng một lần rồi lại quay trở về máy chủ. Hãy lập trình nhập dữ liệu vào từ file, sau đó cho biết có cách truyền tin như vừa mô tả ở trên hay không. Trong trường hợp câu trả lời là khẳng định hãy đưa ra màn hình trình tự các kênh trong mạng mà thông điệp cần lần lượt được truyền qua.

10. Có 17 người bạn gặp nhau ở một lớp bồi dưỡng nâng cao trình độ. Trong suốt thời gian của đợt học tập họ cùng nhau ăn tối ở một nhà hàng có nhiều món ăn hợp khẩu vị với họ. Biết rằng trong tất cả các bữa ăn tối đó, họ đều ngồi quanh một cái bàn tròn và mỗi cặp chỉ ngồi cạnh nhau đúng một lần. Hỏi rằng đợt học tập đó kéo dài nhiều nhất bao nhiêu ngày? Hãy lập trình đưa ra cách xếp chỗ ngồi của họ trong những ngày đó.

5

CÂY VÀ CÂY KHUNG CỦA ĐỒ THỊ

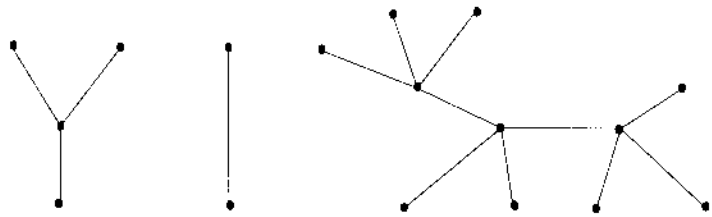
Đồ thị vô hướng liên thông không có chu trình được gọi là cây. Khái niệm cây lần đầu tiên được Cayley đưa ra vào năm 1857, khi ông sử dụng chúng để đếm một số dạng cấu trúc phân tử của các hợp chất hoá học trong hoá học hữu cơ. Cây còn được sử dụng rộng rãi trong rất nhiều lĩnh vực khác nhau, đặc biệt trong tin học, cây được sử dụng để xây dựng các thuật toán tổ chức các thư mục, các thuật toán cất giữ, truyền dữ liệu và tìm kiếm ...

5.1. Cây và các tính chất cơ bản của cây

Định nghĩa 1. *Ta gọi* ***cây*** *là đồ thị vô hướng liên thông không có chu trình. Đồ thị không có chu trình được gọi là* ***rừng.***

Như vậy, rừng là đồ thị mà mỗi thành phần liên thông của nó là một cây.

Thí dụ 1. Trong hình 1 là một rừng gồm 3 cây T_1, T_2, T_3.



Hình 1. Rừng gồm ba cây T_1, T_2, T_3

Có thể nói cây là đồ thị vô hướng đơn giản nhất. Định lý sau đây cho ta một số tính chất của cây.

Định lý 1. *Giả sử $T=(V,E)$ là đồ thị vô hướng n đỉnh. Khi đó các mệnh đề sau đây là tương đương:*

(1) T là cây;

(2) T không chứa chu trình và có n-1 cạnh;

(3) T liên thông và có n-1 cạnh;

(4) T liên thông và mỗi cạnh của nó đều là cầu;

(5) Hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi đơn;

(6) T không chứa chu trình nhưng hễ cứ thêm vào nó một cạnh ta thu được đúng một chu trình.

Chứng minh. Ta sẽ chứng minh định lý theo sơ đồ sau:

$$(1) \Rightarrow (2) \Rightarrow (3) \Rightarrow (4) \Rightarrow (5) \rightarrow (6) \Rightarrow (1).$$

(1) ⇒ (2) Theo định nghĩa T không chứa chu trình. Ta sẽ chứng minh bằng quy nạp theo số đỉnh n khẳng định: Số cạnh của cây với n đỉnh là n-1. Rõ ràng khẳng định đúng với $n = 1$. Giả sử $n > 1$. Trước hết nhận thấy rằng trong mọi cây T có n đỉnh đều tìm được ít nhất một đỉnh là đỉnh treo (tức là đỉnh có bậc là 1). Thực vậy, gọi $v_1, v_2, \ldots, v_k$ là đường đi dài nhất (theo số cạnh) trong T. Khi đó rõ ràng v_2 và v_k là các đỉnh treo, vì từ v_1 (v_k) không có cạnh nối tới bất cứ đỉnh nào trong số các đỉnh $v_2, v_3, \ldots, v_k$ (do đồ thị không chứa chu trình), cũng như tới bất cứ đỉnh nào khác của đồ thị (do đường đi đang xét là dài nhất). Loại bỏ v_1 và cạnh (v_1, v_2) khỏi T ta thu được cây T_1 với n-1 đỉnh, mà theo giả thiết qui nạp có n-2 cạnh. Vậy cây T có n-2+1 = n-1 cạnh.

(2) ⇒ (3) Ta chứng minh bằng phản chứng. Giả sử T không liên thông. Khi đó T phân rã thành $k \geq 2$ thành phần liên thông $T_1, T_2,...,T_k$. Do T không chứa chu trình nên mỗi T_i ($i = 1, 2,..., k$) cũng không chứa chu trình, vì thế mỗi T_i là cây. Do đó nếu gọi $n(T_i)$ và $e(T_i)$ theo thứ tự là số đỉnh và cạnh của T_i, ta có

$$e(T_i) = n(T_i) - 1, \quad i = 1, 2, \ldots, k,$$

suy ra

$$\begin{aligned} n - 1 = e(T) &= e(T_1) + \ldots + e(T_k) \\ &= n(T_1) + \ldots + n(T_k) - k \\ &= n(T) - k < n - 1 \text{ ?!} \end{aligned}$$

Mâu thuẫn thu được chứng tỏ T là liên thông.

(3) ⇒ (4) Việc loại bỏ một cạnh bất kỳ khỏi T dẫn đến đồ thị với n đỉnh và n-2 cạnh rõ ràng là đồ thị không liên thông. Vậy mọi cạnh trong T đều là cầu.

(4) ⇒ (5) Do T là liên thông nên hai đỉnh bất kỳ của nó được nối với nhau bởi một đường đi đơn. Nếu có cặp đỉnh nào của T có hai đường đi đơn khác nhau nối chúng, thì từ đó suy ra đồ thị chứa chu trình, và vì thế các cạnh trên chu trình này không phải là cầu ?!

(5) ⇒ (6) T không chứa chu trình, bởi vì nếu có chu trình thì hoá ra tìm được cặp đỉnh của T được nối với nhau bởi hai đường đi đơn. Bây giờ, nếu thêm vào T một cạnh e nối hai đỉnh u và v nào đó của T. Khi đó cạnh này cùng với đường đi đơn nối u với v sẽ tạo thành chu trình trong T. Chu trình thu được này là duy nhất, vì nếu thu được nhiều hơn một chu trình thì suy ra trong T trước đó phải có sẵn chu trình.

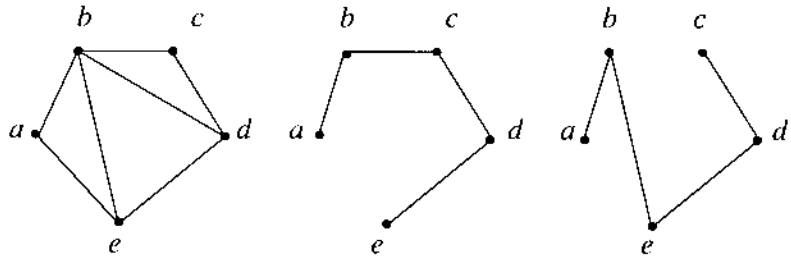
(6) ⇒ (1) Giả sử T không liên thông. Khi đó nó gồm ít ra là 2 thành phần liên thông. Vì vậy, nếu thêm vào T một cạnh nối hai đỉnh thuộc hai thành phần liên thông khác nhau ta không thu được thêm một chu trình nào cả. Điều đó mâu thuẫn với giả thiết (6).

Định lý được chứng minh.

5.2. Cây khung của đồ thị

Định nghĩa 2. *Giả sử $G=(V,E)$ là đồ thị vô hướng liên thông. Cây $T=(V,F)$ với $F \subset E$ được gọi là* ***cây khung*** *của đồ thị G.*

Thí dụ 2. Đồ thị G và cây khung của nó được cho trong hình 2



Hình 2. Đồ thị và các cây khung của nó

Định lý sau đây cho biết số lượng cây khung của đồ thị đầy đủ K_n:

Định lý 2 (Cayley). *Số cây khung của đồ thị* K_n *là* n^{n-2}.

Định lý 2 cho thấy số lượng cây khung của một đồ thị là một số rất lớn. Bây giờ ta xét áp dụng của thuật toán tìm kiếm theo chiều sâu và theo chiều rộng trên đồ thị để xây dựng cây khung của đồ thị vô hướng liên thông. Trong cả hai trường hợp mỗi khi ta đến được đỉnh mới u (tức Chuaxet[u] = true) từ đỉnh v thì cạnh (v, u) sẽ được kết nạp vào cây khung. Hai thuật toán tương ứng được trình bày trong hai thủ tục sau đây.

```
procedure STREE_DFS(v);
(* Tìm kiếm theo chiều sâu áp dụng vào tìm tập cạnh của cây khung T
   của đồ thị vô hướng liên thông G cho bởi danh sách kề.
   Các biến Chuaxet, Ke, T là toàn cục                              *)
begin
      Chuaxet[v] := false ;
      for  u ∈ Ke(v)  do
           if  Chuaxet[u]  then
           begin
                T:= T ∪ (v,u);
                STREE_DFS(u);
           end;
end;
(*     Main Program    *)
BEGIN
      (* Initialiation  *)
      for  u ∈ V  do    Chuaxet[u]:=true;
      T := ∅;              (*  T là tập cạnh của cây khung  *)
      STREE_DFS(root); (*  root   là đỉnh nào đó của đồ thị   *)
END.
```

```
procedure STRE_BFS(r);
(*  Tìm kiếm theo chiều rộng áp dụng tìm tập cạnh của cây khung T
    của đồ thị vô hướng liên thông G cho bởi danh sách Ke      *)
begin
      QUEUE :=∅ ;
      QUEUE ⇐ r ;
      Chuaxet[r]:=false;
      while QUEUE ≠ ∅  do
      begin
           v ⇐ QUEUE;
           for u ∈ Ke(v) do
               if  Chuaxet[u] then
               begin
                    QUEUE ⇐ u; Chuaxet[u]:=false;
                    T:= T ∪ (v,u);
               end;
      end;
end;

(*  Main Program  *)
BEGIN
      for  u ∈ V  do   Chuaxet[u]:=true;
      T := ∅;          (*  T là tập cạnh của cây khung  *)
       STREE_BFS (root); (*  root là một đỉnh tuỳ ý của đồ thị  *)
END.
```

Chú ý:

1. Lập luận tương tự như trong phần trước có thể chỉ ra được rằng các thuật toán mô tả ở trên có độ phức tạp tính toán $O(n+m)$.
2. Cây khung tìm được theo thủ tục STREE_BFS(r) là cây đường đi ngắn nhất từ gốc r đến tất cả các đỉnh còn lại của đồ thị.

5.3. Xây dựng tập các chu trình cơ bản của đồ thị

Bài toán xây dựng cây khung của đồ thị liên quan chặt chẽ với một bài toán ứng dụng khác của lý thuyết đồ thị: Bài toán xây dựng tập các chu trình cơ bản của đồ thị mà ta sẽ xét trong mục này.

Giả sử $G = (V, E)$ là đơn đồ thị vô hướng liên thông, $H=(V,T)$ là cây khung của nó. Các cạnh của đồ thị thuộc cây khung ta sẽ gọi là các cạnh trong, còn các cạnh còn lại sẽ gọi là cạnh ngoài.

Định nghĩa 3. *Nếu thêm một cạnh ngoài $e \in E \setminus T$ vào cây khung H chúng ta sẽ thu được đúng một chu trình trong H, ký hiệu chu trình này là C_e. Tập các chu trình*

$$\Omega = \{ C_e : e \in E \setminus T \}$$

được gọi là ***tập các chu trình cơ bản*** *của đồ thị G.*

Giả sử A và B là hai tập hợp, ta đưa vào phép toán sau

$$A \oplus B = (A \cup B) \setminus (A \cap B).$$

Tập $A \oplus B$ được gọi là ***hiệu đối xứng*** của hai tập hợp A và B.

Tên gọi chu trình cơ bản gắn liền với sự kiện là mỗi chu trình của đồ thị đều có thể thu được từ các chu trình cơ bản như chỉ ra trong định lý sau đây:

Định lý 3. *Giả sử $G=(V,E)$ là đồ thị vô hướng liên thông, $H=(V,T)$ là cây khung của nó. Khi đó mọi chu trình của đồ thị G đều có thể biểu diễn như là hiệu đối xứng của một số các chu trình cơ bản.*

Việc tìm tập các chu trình cơ bản giữ một vai trò quan trọng trong vấn đề giải tích mạng điện. Cụ thể hơn, theo mỗi chu trình cơ bản của đồ thị tương ứng với mạng điện cần phân tích ta sẽ thiết lập được một phương trình tuyến tính theo định luật Kirchoff: *Tổng hiệu điện thế dọc theo một mạch vòng là bằng không.* Hệ thống phương trình tuyến tính thu được cho phép tính toán hiệu điện thế trên mọi đoạn đường dây của lưới điện.

Ta sẽ mô tả thuật toán xây dựng tập các chu trình cơ bản dựa trên thủ tục tìm kiếm theo chiều sâu trên đồ thị. Thuật toán có cấu trúc tương tự như thuật toán xây dựng cây khung theo thủ tục tìm kiếm theo chiều sâu mô tả trong mục trước.

Thuật toán xây dựng tập các chu trình cơ bản.

Giả thiết rằng đồ thị $G = (V, E)$ được mô tả bằng danh sách kề $Ke(v)$, $v \in V$.

```
procedure Cycle(v);
(* Tìm tập các chu trình cơ bản của thành phần liên thông chứa đỉnh v;
   Các biến d, num, STACK, Index là toàn cục    *)
begin
    d:=d+1;
    STACK[d] := v;
    num := num+1;
```

```
    Index[v] := num;
    for  u ∈ Ke(v)  do
        if  Index[u]=0 then Cycle(u)
        else
        if  (u ≠ STACK[d-1]) and (Index[v] > Index[u]) then
            < Ghi nhận chu trình với các đỉnh:
            STACK[d], STACK[d-1], ... , STACK[c], với STACK[c]=u >;
        d := d-1;
end;

(*  Main Program  *)
BEGIN
     for  v ∈ V  do  Index[v] := 0;
      num := 0;  d := 0; STACK[0] := 0;
      for  v ∈ V  do
          if  Index[v] = 0  then  Cycle(v);
END.
```

Chú ý: Độ phức tạp tính toán của thuật toán vừa mô tả là $O(|E||V|)$.

5.4. Bài toán cây khung nhỏ nhất

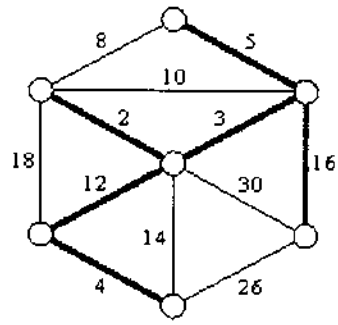
Bài toán cây khung nhỏ nhất của đồ thị là một trong số những bài toán tối ưu trên đồ thị tìm được ứng dụng trong nhiều lĩnh vực khác nhau của đời sống. Trong mục này chúng ta sẽ trình bày những thuật toán cơ bản để giải bài toán này. Trước hết chúng ta phát biểu nội dung của bài toán.

Cho $G = (V, E)$ là đồ thị vô hướng liên thông với tập đỉnh $V = \{1, 2,..., n\}$ và tập cạnh E gồm m cạnh. Mỗi cạnh e của đồ thị G được gán với một số thực $c(e)$, gọi là độ dài của nó. Giả sử $H = (V,T)$ là cây khung của đồ thị G. Ta gọi độ dài $c(H)$ của cây khung H là tổng độ dài của các cạnh của nó:

$$c(H) = \sum_{e \in T} c(e).$$

Bài toán đặt ra là trong số tất cả các cây khung của đồ thị G hãy tìm cây khung với độ dài nhỏ nhất. Cây khung như vậy được gọi là cây khung nhỏ nhất của đồ thị và bài toán đặt ra được gọi là bài toán cây khung nhỏ nhất.

Thí dụ 3. Hình 3 cho một thí dụ về đồ thị có trọng số trên cạnh và cây khung nhỏ nhất của đồ thị được chỉ ra bởi các cạnh tô đậm.



Hình 3. Cây khung nhỏ nhất của đồ thị

Để minh hoạ cho những ứng dụng của bài toán cây khung nhỏ nhất, dưới đây, ta phát biểu hai mô hình thực tế tiêu biểu của nó.

Bài toán xây dựng hệ thống đường sắt. Giả sử ta muốn xây dựng một hệ thống đường sắt nối n thành phố sao cho hành khách có thể đi từ bất cứ một thành phố nào đến bất kỳ một trong số các thành phố còn lại. Mặt khác trên quan điểm kinh tế đòi hỏi là chi phí về xây dựng hệ thống đường phải là nhỏ nhất. Rõ ràng là đồ thị mà đỉnh là các thành phố còn các cạnh là các tuyến đường sắt nối các thành phố tương ứng với phương án xây dựng tối ưu phải là cây. Vì vậy, bài toán đặt ra dẫn về bài toán tìm cây khung nhỏ nhất trên đồ thị đầy đủ n đỉnh, mỗi đỉnh tương ứng với một thành phố, với độ dài trên các cạnh chính là chi phí xây dựng đường ray nối hai thành phố tương ứng (chú ý là trong bài toán này ta giả thiết là không được xây dựng tuyến đường sắt có các nhà ga phân tuyến nằm ngoài các thành phố).

Bài toán nối mạng máy tính. Cần nối mạng một hệ thống gồm n máy vi tính đánh số từ 1 đến n. Biết chi phí nối máy i với máy j là $c[i, j]$, $i, j = 1, 2,..., n$ (thông thường chi phí này phụ thuộc vào độ dài cáp nối cần sử dụng). Hãy tìm cách nối mạng sao cho tổng chi phí nối mạng là nhỏ nhất.

Để giải bài toán cây khung nhỏ nhất, tất nhiên có thể liệt kê tất cả các cây khung của đồ thị và chọn trong số chúng cây khung nhỏ nhất. Phương pháp như vậy, trong trường hợp đồ thị đầy đủ, sẽ đòi hỏi thời gian cỡ n^{n-2}, và rõ ràng không thể thực hiện được ngay cả với những đồ thị với số đỉnh cỡ hàng chục. Rất may là đối với bài toán cây khung nhỏ nhất chúng ta đã có những thuật toán rất hiệu quả để giải chúng. Chúng ta sẽ xét hai trong số những thuật toán như vậy: Thuật toán Kruskal và Thuật toán Prim.

5.4.1. Thuật toán Kruskal

Thuật toán sẽ xây dựng tập cạnh T của cây khung nhỏ nhất $H = (V,T)$ theo từng bước. Trước hết xắp xếp các cạnh của đồ thị G theo thứ tự không giảm của độ dài. Bắt đầu từ tập $T = \varnothing$, ở mỗi bước ta sẽ lần lượt duyệt trong danh sách cạnh đã xắp xếp, từ cạnh có độ dài nhỏ đến cạnh có độ dài lớn hơn, để tìm ra cạnh mà việc bổ sung nó vào tập T không tạo thành chu trình trong tập này. Thuật toán sẽ kết thúc khi ta thu được tập T gồm n-1 cạnh. Cụ thể, thuật toán có thể mô tả như sau:

```
procedure  Kruskal;
begin
      T := ∅;
       while  | T | < (n-1) and ( E ≠ ∅ ) do
       begin
           Chọn e là cạnh có độ dài nhỏ nhất trong E;
          E := E \ {e};
          if  ( T ∪ {e}  không chứa chu trình )  then  T := T ∪ {e};
       end;
       if  ( | T | < n-1 )  then   Đồ thị không liên thông;
end;
```

Thí dụ 4. Tìm cây khung nhỏ nhất của đồ thị cho trong hình 4.

Bước khởi tạo. Đặt $T := \varnothing$. Sắp xếp các cạnh của đồ thị theo thứ tự không giảm của độ dài ta có dãy:

(3,5), (4,6), (4,5), (5,6), (3,4), (1,3), (2,3), (2,4), (1,2)

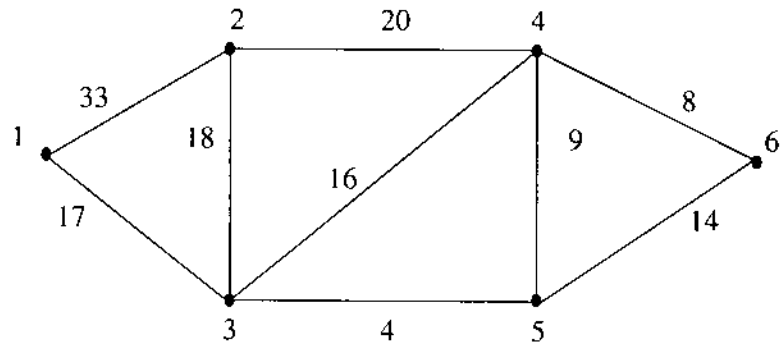
dãy độ dài tương ứng của chúng

4, 8, 9, 14, 16, 17, 18, 20, 23.

ở ba lần lặp đầu tiên ta lần lượt bổ sung vào tập T các cạnh (3,5), (4,6), (4,5). Rõ ràng nếu thêm cạnh (5,6) vào T thì nó sẽ tạo thành với 2 cạnh (4,5), (4,6) đã có trong T chu trình. Tình huống tương tự cũng xảy ra đối với cạnh (3,4) là cạnh tiếp theo trong dãy. Tiếp theo ta bổ sung cạnh (1,3), (2,3) vào T và thu được tập T gồm 5 cạnh:

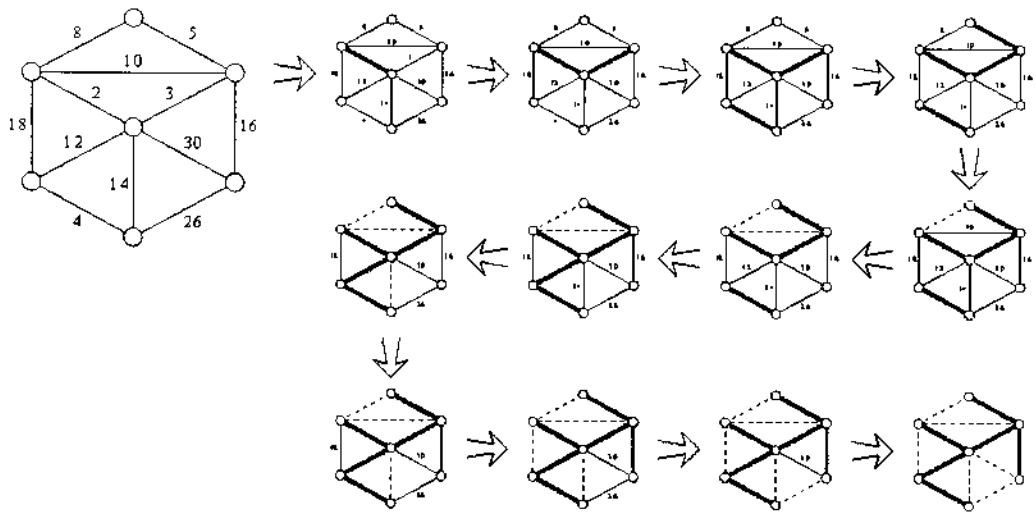
$$T = \{(3,5), (4,6), (4,5), (1,3), (2,3)\}$$

chính là tập cạnh của cây khung nhỏ nhất cần tìm.



Hình 4. Đồ thị có trọng số

Thí dụ 5. Hình 5 minh hoạ quá trình tìm cây khung nhỏ nhất của đồ thị cho trong hình 3: Các cạnh đậm là các cạnh được chọn vào cây khung, các cạnh đứt nét là các cạnh được bỏ qua trong quá trình duyệt qua các cạnh.



Hình 5. Tìm cây khung nhỏ nhất theo thuật toán Kruscal

Chứng minh tính đúng đắn của thuật toán.

Rõ ràng đồ thị thu được theo thuật toán có n-1 cạnh và không có chu trình, vì vậy theo định lý 1 nó là cây khung của đồ thị G. Như vậy, chỉ còn phải chỉ ra rằng T có độ dài nhỏ nhất. Giả sử tồn tại cây S của đồ thị G mà $c(S) < c(T)$. Ký hiệu e_k là cạnh đầu tiên trong dãy các cạnh của T xây dựng theo thuật toán vừa mô tả không thuộc S. Khi đó đồ thị con của G sinh bởi cây S được bổ sung cạnh e_k sẽ chứa 1 chu trình duy nhất C đi qua e_k. Do chu trình C phải chứa cạnh e thuộc S nhưng không thuộc T nên đồ thị con thu

được từ S bằng cách thay cạnh e của nó bởi e_k (ký hiệu đồ thị này là S') sẽ là cây khung. Theo cách xây dựng $c(e_k) \le c(e)$ do đó $c(S') \le c(S)$, đồng thời số cạnh chung của S' và T đã tăng thêm một so với số cạnh chung của S và T. Lặp lại quá trình trên từng bước một ta có thể biến đổi S thành T và trong mỗi bước tổng độ dài không tăng, tức là $c(T) \le c(S)$. Mâu thuẫn thu được chúng tỏ T là cây khung nhỏ nhất.

Về việc lập trình thực hiện thuật toán.

Khối lượng tính toán nhiều nhất của thuật toán chính là ở bước sắp xếp các cạnh của đồ thị theo thứ tự không giảm của độ dài để lựa chọn cạnh bổ sung. Đối với đồ thị có m cạnh cần phải thực hiện cỡ $m \log m$ phép toán để sắp xếp các cạnh của đồ thị thành dãy không giảm theo độ dài. Tuy nhiên, để xây dựng cây khung nhỏ nhất với n-1 cạnh, nói chung, ta không cần phải sắp thứ tự toàn bộ các cạnh mà chỉ cần xét phần trên của dãy đó chứa $r < m$ cạnh. Để làm việc đó ta có thể sử dụng các thủ tục sắp xếp dạng Vun đống (Heap Sort). Trong thủ tục này, để tạo đống đầu tiên ta mất cỡ $O(m)$ phép toán, mỗi phần tử lớn tiếp theo trong đống có thể tìm sau thời gian $O(\log m)$. Vì vậy, với cải tiến này thuật toán sẽ mất thời gian cỡ $O(m+p \log m)$ cho việc sắp xếp các cạnh. Trong thực tế tính toán số p nhỏ hơn rất nhiều so với m.

Vấn đề thứ hai trong việc thể hiện thuật toán Kruskal là việc lựa chọn cạnh để bổ sung đòi hỏi phải có một thủ tục hiệu quả kiểm tra tập cạnh $T \cup \{e\}$ có chứa chu trình hay không. Để ý rằng, các cạnh trong T ở các bước lặp trung gian sẽ tạo thành một rừng. Cạnh e cần khảo sát sẽ tạo thành chu trình với các cạnh trong T khi và chỉ khi cả hai đỉnh đầu của nó thuộc vào cùng một cây con của rừng nói trên. Do đó, nếu cạnh e không tạo thành chu trình với các cạnh trong T, thì nó phải nối hai cây khác nhau trong T. Vì thế, để kiểm tra xem có thể bổ sung cạnh e vào T ta chỉ cần kiểm tra xem nó có nối hai cây khác nhau trong T hay không. Một trong các phương pháp hiệu quả để thực hiện việc kiểm tra này là ta sẽ phân hoạch tập các đỉnh của đồ thị ra thành các tập con không giao nhau, mỗi tập xác định bởi một cây con trong T (được hình thành ở các bước do việc bổ sung cạnh vào T). Chẳng hạn, đối với đồ thị trong ví dụ 3, đầu tiên ta có sáu tập con 1 phần tử:{1}, {2}, {3}, {4}, {5}, {6}. Sau khi bổ sung cạnh (3, 5), ta có các tập con {1}, {2}, {3, 5}, {4}, {6}. Tiếp theo, sau khi cạnh (4,6) được chọn, ta có các tập con {1}, {2}, {3, 5}, {4, 6}. ở bước thứ 3, ta chọn cạnh (4,5), khi đó hai tập con được nối lại và danh sách các tập con là {1}, {2}, {3, 4, 5, 6}. Cạnh có độ dài tiếp theo là (4, 6), do hai đầu của nó thuộc vào cùng một tập con {3, 4, 5, 6}, nên nó sẽ tạo thành chu trình trong tập này. Vì vậy cạnh này không được chọn. Và thuật toán sẽ tiếp tục chọn cạnh tiếp theo để khảo sát ...

Như vậy, để giải quyết vấn đề thứ hai này ta phải xây dựng hai thủ tục: Kiểm tra xem hai đầu u, v của cạnh $e=(u, v)$ có thuộc vào hai tập con khác nhau hay không, và trong trường hợp câu trả lời là khẳng định, nối hai tập con tương ứng thành một tập.

Chú ý rằng mỗi tập con trong phân hoạch có thể lưu trữ như là một cây có gốc, và khi đó mỗi gốc sẽ được sử dụng làm nhãn nhận biết tập con tương ứng.

Chương trình trên Pascal thực hiện thuật toán Kruskal với những nhận xét vừa nêu có thể viết như sau:

```
(*   Tìm cây khung nhỏ nhất theo thuật toán Kruskal
                của đồ thị cho bởi  danh sách cạnh        *)
uses crt;
type
   arrn = array[1..50] of integer;
   arrm = array[1..5000] of integer;
var
    n, m, MinL       : integer;
    Dau, Cuoi, W     : arrm;
    DauT, CuoiT, Father: arrn;
    Connect          : boolean;

procedure Nhapdl;
var     i : integer;
   fname: string;
   f      : text;
begin
     write('Cho Tên file dữ liệu: '); readln(fname);
     assign(f, fname);
     reset(f);
     readln(f,n,m);
     for i := 1 to m do readln( f, Dau[i], Cuoi[i], W[i] );
     close(f);
end;

procedure  Indulieu;
var  i: integer;
begin
     writeln('Số đỉnh: ',n,'.  Số cạnh: ',m);
     writeln(' Đỉnh đầu  Đỉnh cuối   Độ dài');
     for  i:=1 to m do
        writeln(Dau[i]:4, Cuoi[i]:10, W[i]:12);
end;
```

```
procedure  Heap(First, Last:integer);
var    j, k, t1, t2, t3: integer;
begin
    j := first;
    while (j <= trunc(last/2) ) do
    begin
        if (2*j < last) and (W[2*j+1] < W[2*j]) then
          k := 2*j+1
        else
          k := 2*j;
        if W[k] < W[j] then
        begin
           t1 := Dau[j];   t2 := Cuoi[j];   t3 := W[j];
           Dau[j] := Dau[k];  Cuoi[j] := Cuoi[k];  W[j] := W[k];
           Dau[k] := t1;  Cuoi[k] := t2;  W[k] := t3;
           j := k;
        end
        else j := Last;
    end;
end;

function   Find(i: integer) : integer;
var    Tro : integer;
begin
     Tro := i;
     while   Father[Tro] > 0 do
                 Tro:= Father[Tro];
     Find:=Tro;
end;

procedure  Union(i,j : integer);
var    x: integer;
begin
     x := Father[i] + Father[j];
     if Father[i]> Father[j] then
     begin
        Father[i] := j;
        Father[j] := x;
     end
     else
```

```
    begin
       Father[j] := i;
       Father[i] := x;
    end;
end;

procedure   Kruskal;
var  i, Last, u, v, r1, r2, Ncanh, Ndinh : integer;
begin
    (*  Khởi tạo mảng Father đánh dấu cây con
        và khởi tạo Heap   *)
    for  i:=1  to  n  do Father[i]:=-1;
    for  i:= trunc(m/2)  downto  1  do  Heap(i,m);
    last:=m;  Ncanh := 0; Ndinh := 0;
    MinL := 0;
    Connect := true;
    while (Ndinh < n-1) and (Ncanh < m) do
    begin
       Ncanh := Ncanh+1;
        u := Dau[1]; v := Cuoi[1];
       (*  Kiểm tra u và v có thuộc cùng một cây con?  *)
       r1 := Find(u);
       r2 :=  Find(v);
       if   r1 <> r2 then
       begin
        (*   Kết nạp cạnh (u,v) vào cây khung   *)
          Ndinh:=Ndinh+1;  Union(r1,r2);
          DauT[Ndinh]:=u;
          CuoiT[Ndinh]:=v;
          MinL:=MinL+W[1];;
       end;
      (*   Tổ chức lại Heap   *)
       Dau[1] := Dau[Last];
       Cuoi[1] := Cuoi[Last];
       W[1] := W[Last];
       Last := Last-1;
       Heap(1,Last);
    end;
    if   Ndinh < > n-1  then  Connect := false;
end;
```

```
procedure Inketqua;
var    i: integer;
begin
     writeln('*************************************');
     writeln('***              Kết quả tính toán              ****');
     writeln('*************************************');
     writeln('Độ dài của cây khung nhỏ nhất: ',MinL);
     writeln('Các cạnh của cây khung nhỏ nhất:');
     for   i := 1   to  n-1  do
        writeln('(',DauT[i]:2,', ',CuoiT[i]:2,') ');
     writeln('*************************************');
end;

BEGIN
    clrscr;
    Nhapdl;
    Indulieu;
    Kruskal;
    if Connect then Inketqua
    else
       writeln('Đồ thị không liên thông');
    readln;
END.
```

File dữ liệu của bài toán trong thí dụ 4 có dạng sau:

```
6   9
3   5   4
4   6   8
4   5    9
5   6  14
3   4   16
1   3   17
2   3   18
2  4    20
1  2    23
```

Kết quả tìm được: Độ dài của cây khung nhỏ nhất và tập cạnh của cây khung nhỏ nhất được đưa ra màn hình

5.4.2. Thuật toán Prim

Thuật toán Kruskal làm việc kém hiệu quả đối với những đồ thị **dày** (đồ thị với số cạnh $m \approx n(n-1)/2$). Trong trường hợp đó thuật toán Prim tỏ ra hiệu quả hơn. Thuật toán Prim còn được gọi là phương pháp lân cận gần nhất. Trong phương pháp này, bắt đầu từ một đỉnh tuỳ ý của đồ thị s, đầu tiên ta nối s với đỉnh lân cận gần nó nhất, chẳng hạn là đỉnh y. Nghĩa là trong số các cạnh kề của đỉnh s, cạnh (s, y) có độ dài nhỏ nhất. Tiếp theo, trong số các cạnh kề với hai đỉnh s hoặc y ta tìm cạnh có độ dài nhỏ nhất, cạnh này dẫn đến đỉnh thứ ba z, và ta thu được cây bộ phận gồm ba đỉnh và hai cạnh. Quá trình này sẽ được tiếp tục cho đến khi ta thu được cây gồm n đỉnh và n-1 cạnh sẽ chính là cây khung nhỏ nhất cần tìm.

Giả sử đồ thị cho bởi ma trận trọng số $C = \{c[i,j], i, j = 1, 2,..., n\}$. Trong quá trình thực hiện thuật toán, ở mỗi bước để có thể nhanh chóng chọn đỉnh và cạnh cần bổ sung vào cây khung, các đỉnh của đồ thị sẽ được gán cho các nhãn. Nhãn của một đỉnh v sẽ gồm hai phần và có dạng [$d[v]$, $near[v]$], trong đó $d[v]$ dùng để ghi nhận độ dài của cạnh có độ dài nhỏ nhất trong số các cạnh nối đỉnh v với các đỉnh của cây khung đang xây dựng (ta sẽ gọi là khoảng cách từ đỉnh v đến tập đỉnh của cây khung), nói một cách chính xác

$$d[v] := \min\{ c[v, w] : w \in V_H \} \ (= c[v, z]),$$

còn $near[v]$ ghi nhận đỉnh của cây khung gần v nhất ($near[v] := z$).

Thuật toán Prim được mô tả đầy đủ trong thủ tục sau:

```
procedure Prim;
begin
    (*      Bước khởi tạo       *)
    Chọn  s  là một đỉnh nào đó của đồ thị ;
    VH := { s }; T := ∅ ;
    d[s] := 0; near[s] := s.
    for  v ∈ V \ VH  do
    begin
        d[v] := c[s,v];
        near[v] := s;
    end;
    (*      Bước lặp        *)
    Stop := false;
    while  not Stop  do
    begin
        Tìm  u ∈ V \ VH  thoả mãn:  d[u] = min { d[v] : u ∈ V \ VH };
```

```
        VH := VH ∪ { u };   T := T ∪ { ( u, near[u] ) } ;
        if    | VH | = n  then
        begin
              H = ( VH , T )  là cây khung nhỏ nhất của đồ thị ;
              Stop := true;
        end
        else
        for  v∈ V \ VH  do
            if   d[v] > c[u,v]  then
            begin
                d[v] := c[u,v] ;
                near[v] := u;
            end;
    end;
end;
```

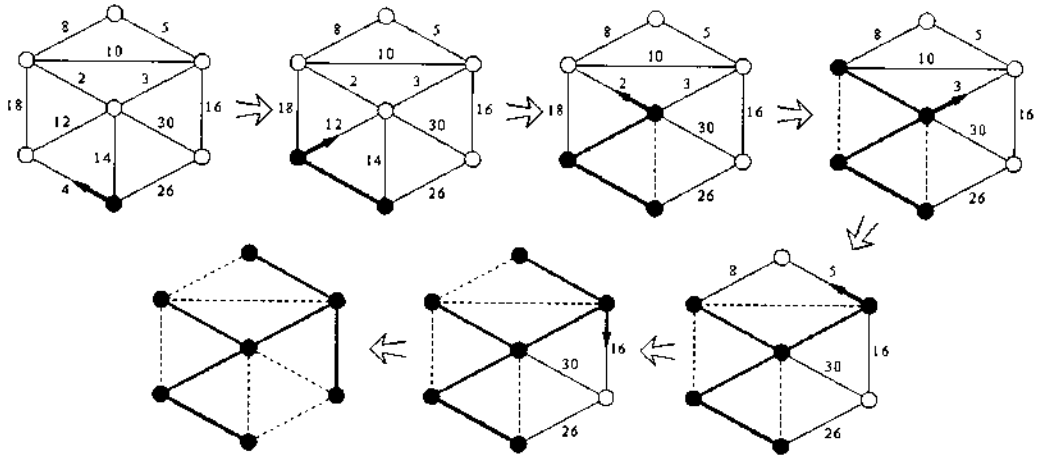
Thí dụ 6. Tìm cây khung nhỏ nhất cho đồ thị xét trong thí dụ 4 theo thuật toán Prim. Ma trận trọng số của đồ thị có dạng

$$C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 33 & 17 & \infty & \infty & \infty \\ 33 & 0 & 18 & 20 & \infty & \infty \\ 17 & 18 & 0 & 16 & 4 & \infty \\ \infty & 20 & 16 & 0 & 9 & 8 \\ \infty & \infty & 4 & 9 & 0 & 14 \\ \infty & \infty & \infty & 8 & 14 & 0 \end{bmatrix} \end{array}$$

Bảng dưới đây ghi nhãn của các đỉnh trong các bước lặp của thuật toán, đỉnh đánh dấu * là đỉnh được chọn để bổ sung vào cây khung (khi đó nhãn của nó không còn bị biến đổi trong các bước lặp tiếp theo, vì vậy ta đánh dấu - để ghi nhận điều đó) :

Bước lặp	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6	V_H	T
Khởi tạo	[0, 1]	[33, 1]	[17, 1]*	[∞, 1]	[∞, 1]	[∞, 1]	1	∅
1	-	[18, 3]	-	[16, 3]	[4, 3]*	[∞, 1]	1, 3	(3,1)
2	-	[18, 3]	-	[9,5]*	-	[14, 5]	1, 3, 5	(3,1), (5,3)
3	-	[18, 3]	-	-	-	[8, 4]*	1, 3, 5, 4	(3,1), (5,3), (4,5)
4	-	[18, 3]*	-	-	-	-	1, 3, 5, 4, 6	(3,1), (5,3), (4,5), (6,4)
5	-	-	-	-	-	-	1, 3, 5, 4, 6, 2	(3,1), (5,3), (4,5), (6,4), (2,3)

Thí dụ 6. Hình 6 minh hoạ quá trình tìm cây khung nhỏ nhất của đồ thị cho trong hình 3: Các cạnh đậm là các cạnh được chọn vào cây khung, Mũi tên chỉ hướng chọn đỉnh tiếp theo, Cạnh đứt nét không cần phải xét trong các bước tiếp theo.



Hình 6. Tìm cây khung theo thuật toán Prim

5.4.3. Một số bài toán dẫn về bài toán cây khung nhỏ nhất

Trong mục này ta nêu hai bài toán có thể dẫn về bài toán cây khung nhỏ nhất, và vì thế có thể giải được nhờ các thuật toán mô tả trong mục trước

a) Bài toán cây khung lớn nhất. Dễ dàng nhận thấy rằng trong các thuật toán trên ta không cần sử dụng đến đòi hỏi về dấu của độ dài. Vì vậy có thể áp dụng chúng đối với đồ thị có các cạnh với độ dài có dấu tuỳ ý. Vì vậy, giả sử ta phải tìm cây lớn nhất (tức là có độ dài $c(H)$ lớn nhất) thì chỉ cần đổi dấu tất cả các độ đo và áp dụng một trong hai thuật toán vừa mô tả.

b) Bài toán tìm mạng điện với độ tin cậy lớn nhất. Cho lưới điện có n nút. Đường dây nối nút i với nút j có độ tin cậy $1 > p[i, j] > 0$, $i, j = 1, 2,..., n$. Gọi $G = (V, E)$ là đồ thị tương ứng với lưới điện này. Hãy tìm cây khung H của đồ thị G với độ tin cậy

$$\prod_{e \in H} p(e)$$

lớn nhất.

Bài toán này dẫn về bài toán tìm cây khung với tổng độ dài nhỏ nhất trên đồ thị G với độ dài của mỗi cạnh $e \in E$ là $-\log p(e)$. Thực vậy, giả sử H là cây khung nhỏ nhất trên đồ thị với độ dài $-\log p(e)$, ta có

$$-\prod_{e\in H}\log p(e) \quad \leq \quad -\prod_{e\in H'}\log p(e)$$

với mọi cây khung H' của đồ thị G. Từ đó suy ra

$$\sum_{e\in H}\log p(e) \quad \geq \quad \sum_{e\in H'}\log p(e)$$

do đó

$$\log\prod_{e\in H} p(e) \quad \geq \quad \log\prod_{e\in H'} p(e)$$

hay là

$$\prod_{e\in H} p(e) \quad \geq \quad \prod_{e\in H'} p(e)$$

với mọi cây khung H'. Vậy H là cây khung có độ tin cậy lớn nhất.

Bài tập

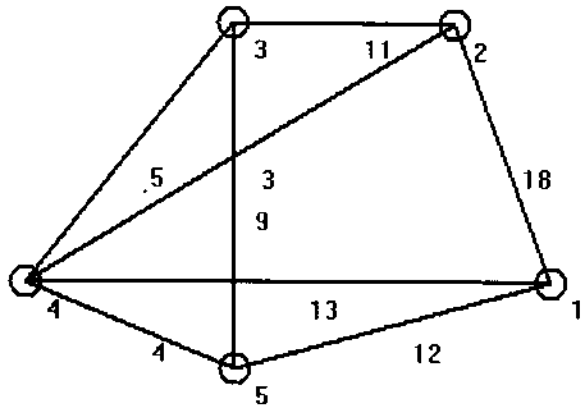
1. Hãy viết các chương trình trên Pascal thực hiện các thuật toán:

a) Xây dựng cây khung của đồ thị;

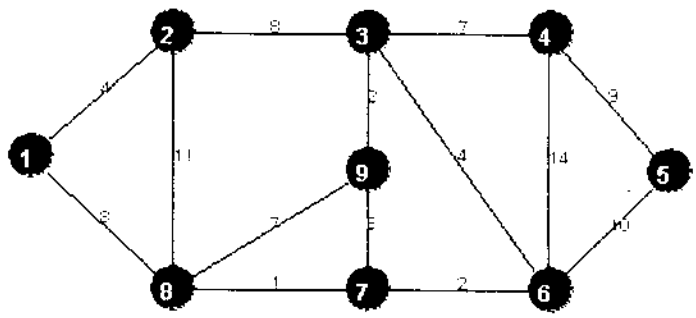
b) Xây dựng tập các chu trình cơ bản của đồ thị;

c) Xây dựng cây khung nhỏ nhất theo thuật toán Prim.

2. Áp dụng thuật toán Kruscal tìm cây khung ngắn nhất cho đồ thị sau đây

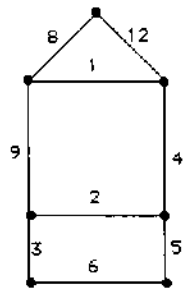


3. Áp dụng thuật toán Prim tìm cây khung ngắn nhất cho đồ thị sau đây, lấy đỉnh xuất phát là đỉnh 1

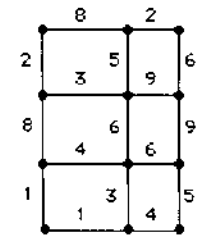


4. Tìm cây khung ngắn nhất cho đồ thị có trọng số cho trong hình vẽ dưới đây theo thuật toán Kruscal và thuật toán Prim

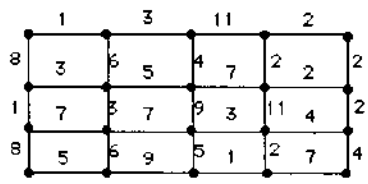
a)



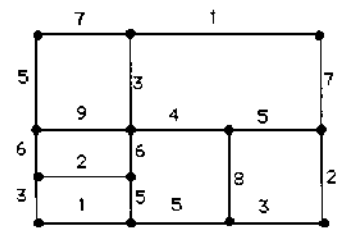
b)



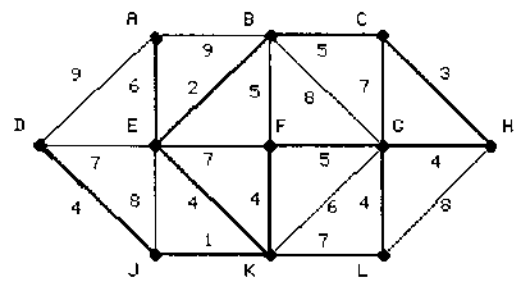
c)



d)



e)



5. Xét đồ thị gồm 6 đỉnh A, B, C, D, E, F cho bởi ma trận trọng số

	A	B	C	D	E	F
A	0	33	17	85	85	85
B	33	0	18	20	85	85
C	17	18	0	16	4	85
D	85	20	16	0	9	8
E	85	85	4	9	0	14
F	85	85	85	8	14	0

Tìm cây khung ngắn nhất theo thuật toán Prim. Yêu cầu viết rõ kết quả trung gian trong từng bước. Kết quả cuối cùng: Tập cạnh của cây khung nhỏ nhất và độ dài của nó.

6. Một công ty lập một dự án mắc điện thoại cho n (n<100) nhân viên của mình bằng một lưới các đoạn nối từ máy của một người đến máy của một số người khác (không phải hai người nào cũng được nối với nhau). Dự án phải đảm bảo yêu cầu sau (gọi là yêu cầu về ***tính thông suốt*** của mạng): ***Trên lưới điện thoại đó mỗi nhân viên của Công ty đều có thể nhắn tin cho bất cứ một nhân viên khác hoặc trực tiếp hoặc***

thông qua một số nhân viên trung gian. Dữ liệu về dự án được ghi trong file văn bản có tên NETW.INP có cấu trúc như sau:

- Dòng đầu tiên chứa số n;
- Dòng tiếp theo chứa số đường nối;
- Trong các dòng tiếp theo chứa các đường nối của dự án: mỗi dòng gồm 3 số nguyên theo thứ tự là chỉ số của hai máy được nối và chi phí nối hai máy này.

Hãy lập trình nhập dữ liệu vào tù file, sau đó:

a) Kiểm tra xem dự án có đáp ứng yêu cầu về tính thông suốt hay không?

b) Trong trường hợp dự án đáp ứng yêu cầu thông suốt, hãy tìm cách loại bỏ một số đường nối sao cho mạng vẫn là thông suốt đồng thời giảm đến mức tối thiểu chi phí nối mạng.

7. Người ta định tổ chức một mạng máy tính gồm $m \times n$ máy được bố trí theo một mạng ô vuông gồm $m \times n$ nút. Nút của lưới ở hàng i cột j gọi là nút (i,j), $i=1,2,...,m$; $j=1,2,...,n$. Mỗi máy sẽ được đặt ở một nút và các máy được nối với nhau theo cạnh của các ô vuông (gọi là các đường nối của mạng). Trong khi mạng còn đang trong quá trình xây dựng (mới chỉ có một số đường nối nào đó được xây dựng xong), vì phải đưa mạng vào phục vụ kịp thời kỳ thi Olimpic Tin học Quốc tế, người ta muốn kiểm tra xem tất cả các máy đã trao đổi thông tin được với nhau hay chưa (hai máy bất kỳ trong mạng gọi là trao đổi thông tin được với nhau nếu như chúng có thể trao đổi thông tin được với nhau hoặc là trực tiếp, hoặc thông qua các máy trung gian). Trong trường hợp câu trả lời là phủ định, cần xác định số đường nối ít nhất cần phải gấp rút xây dựng trước sao cho tất cả các máy đều trao đổi thông tin được với nhau, nhằm phục vụ kịp thời nhiệm vụ đột xuất nói trên. Dữ liệu về các đường nối đã xây dựng xong được cho trong một file văn bản có tên COMNET.INP có cấu trúc như sau

- Dòng đầu tiên chứa hai số m, n;
- Trong các dòng tiếp theo, mỗi dòng chứa thông tin về một đường nối đã xây dựng xong trong thời điểm hiện tại gồm 4 số nguyên dương xác định toạ độ của hai máy được nối với nhau bởi đường nối này.

Hãy lập trình nhập dữ liệu vào từ file, sau đó thực hiện các yêu cầu nói trên.

6

BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT

Trong các ứng dụng thực tế, Bài toán tìm đường đi ngắn nhất giữa hai đỉnh của một đồ thị liên thông có một ý nghĩa to lớn. Có thể dẫn về bài toán như vậy nhiều bài toán thực tế quan trọng. Ví dụ, Bài toán chọn một hành trình tiết kiệm nhất (theo tiêu chuẩn khoảng cách hoặc thời gian hoặc chi phí) trên một mạng giao thông đường bộ, đường thuỷ hoặc đường không; Bài toán chọn một phương pháp tiết kiệm nhất để đưa một hệ động lực lực từ trạng thái xuất phát đến một trạng thái đích, Bài toán lập lịch thi công các công đoạn trong một công trình thi công lớn, Bài toán lựa chọn đường truyền tin với chi phí nhỏ nhất trong mạng thông tin, v..v... Hiện nay có rất nhiều phương pháp để giải các bài toán như vậy. Thế nhưng, thông thường, các thuật toán được xây dựng dựa trên cơ sở lý thuyết đồ thị tỏ ra là các thuật toán có hiệu quả cao nhất. Trong chương này chúng ta sẽ xét một số thuật toán như vậy.

6.1. Các khái niệm mở đầu

Trong chương này chúng ta chỉ xét đồ thị có hướng $G = (V, E)$, $|V|=n$, $|E|=m$ với các cung được gán trọng số, nghĩa là, mỗi cung $(u, v) \in E$ của nó được đặt tương ứng với một số thực $a(u,v)$ gọi là trọng số của nó. Chúng ta sẽ đặt $a(u, v) = \infty$, nếu $(u, v) \notin E$. Nếu dãy

$$v_0, v_1, \ldots, v_p$$

là một đường đi trên G, thì độ dài của nó được định nghĩa là tổng sau

$$\sum_{i=1}^{p} a(v_{i-1}, v_i).$$

tức là, độ dài của đường đi chính là tổng các trọng số trên các cung của nó. (Chú ý rằng nếu chúng ta gán trọng số cho tất cả các cung đều bằng 1, thì ta thu được định nghĩa độ dài của đường đi như là số cung của đường đi giống như trong các chương trước đã xét).

Bài toán tìm đường đi ngắn nhất trên đồ thị dưới dạng tổng quát có thể phát biểu như sau: Tìm đường đi có độ dài nhỏ nhất từ một đỉnh xuất phát $s \in V$ đến đỉnh cuối (đích) $t \in V$. Đường đi như vậy ta sẽ gọi là *đường đi ngắn nhất từ s đến t* còn độ dài của nó ta sẽ ký hiệu là $d(s,t)$ và còn gọi là *khoảng cách* từ s đến t (khoảng cách định nghĩa như vậy có thể là số âm). Nếu như không tồn tại đường đi từ s đến t thì ta sẽ đặt $d(s,t) = \infty$. Rõ ràng, nếu như mỗi chu trình trong đồ thị đều có độ dài dương, thì trong đường đi ngắn nhất không có đỉnh nào bị lặp lại (đường đi không có đỉnh lặp lại sẽ được gọi là *đường đi cơ bản*). Mặt khác, nếu trong đồ thị có chu trình với độ dài âm (chu trình như vậy, để ngắn gọn, ta sẽ gọi là *chu trình âm*) thì khoảng cách giữa một số cặp đỉnh nào đó của đồ thị có thể là không xác định, bởi vì, bằng cách đi vòng theo chu trình này một số đủ lớn lần, ta có thể chỉ ra đường đi giữa các đỉnh này có độ dài nhỏ hơn bất cứ số thực cho trước nào. Trong những trường hợp như vậy, có thể đặt vấn đề tìm đường đi cơ bản ngắn nhất, tuy nhiên bài toán đặt ra sẽ trở nên phức tạp hơn rất nhiều, bởi vì nó chứa bài toán xét sự tồn tại đường đi Hamilton trong đồ thị như là một trường hợp riêng. Một tính chất của đường đi ngắn nhất nữa có thể phát biểu một cách không hình thức như sau: Mọi đoạn đường con của đường đi ngắn nhất cũng là đường đi ngắn nhất. Tính chất tuy rất hiển nhiên này nhưng lại hàm chứa một nội dung rất sâu sắc, và người ta thường gọi nó là nguyên lý tối ưu. Việc chứng minh tính đúng đắn của hầu hết các thuật toán tìm đường đi ngắn nhất đều được xây dựng dựa vào nguyên lý này.

Trước hết cần chú ý rằng nếu biết khoảng cách từ s đến t, thì đường đi ngắn nhất từ s đến t, trong trường hợp trọng số không âm, có thể tìm được một cách dễ dàng. Để tìm

đường đi, chỉ cần để ý là đối với cặp đỉnh $s, t \in V$ tuỳ ý ($s \neq t$) luôn tìm được đỉnh v sao cho

$$d(s,t) = d(s,v) + a(v,t).$$

Thực vậy, đỉnh v như vậy chính là đỉnh đi trước đỉnh t trong đường đi ngắn nhất từ s đến t. Tiếp theo ta lại có thể tìm được đỉnh u sao cho $d(s,v) = d(s,u) + a(u,v)$, ... Từ giả thiết về tính không âm của các trọng số dễ dàng suy ra rằng dãy $t, v, u, ...$ không chứa đỉnh lặp lại và kết thúc ở đỉnh s. Rõ ràng dãy thu được xác định (nếu lật ngược thứ tự các đỉnh trong nó) đường đi ngắn nhất từ s đến t. Từ đó ta có thuật toán sau đây để tìm đường đi ngắn nhất từ s đến t khi biết độ dài của nó.

```
procedure Find_Path;
(*
Đầu vào:
    d[v]    - khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại v ∈ V;
    t       - đỉnh đích;
    a[u,v] , u, v ∈ V - ma trận trọng số trên các cung.
Đầu ra:
    Mảng STACK chứa dãy đỉnh xác định đường đi ngắn nhất từ s đến t
*)
begin
   STACK := ∅ ;  STACK ⇐ t ;   v := t ;
   while  v ≠ s do
   begin
       u:= đỉnh thoả mãn  d[v] = d[u] + a[u,v] ;
       STACK ⇐ u ;
       v := u ;
   end;
end;
```

Chú ý rằng độ phức tạp tính toán của thuật toán là $O(n^2)$, do để tìm đỉnh u ta phải xét qua tất cả các đỉnh của đồ thị. Tất nhiên, ta cũng có thể sử dụng kỹ thuật ghi nhận đường đi đã trình bày trong chương 3: dùng biến mảng Truoc[v], $v \in V$, để ghi nhớ đỉnh đi trước v trong đường đi tìm kiếm.

Cũng cần lưu ý thêm là trong trường hợp trọng số trên các cạnh là không âm, bài toán tìm đường đi ngắn nhất trên đồ thị vô hướng có thể dẫn về bài toán trên đồ thị có hướng, bằng cách thay mỗi cạnh của nó bởi hai cung có hướng ngược chiều nhau với cùng trọng số là trọng số của cạnh tương ứng. Tuy nhiên, trong trường hợp có trọng số âm, việc thay như vậy có thể dẫn đến chu trình âm.

6.2. Đường đi ngắn nhất xuất phát từ một đỉnh

Phần lớn các thuật toán tìm khoảng cách giữa hai đỉnh s và t được xây dựng nhờ kỹ thuật tính toán mà ta có thể mô tả đại thể như sau: từ ma trận trọng số $a[u,v]$, $u, v \in V$, ta tính cận trên $d[v]$ của khoảng cách từ s đến tất cả các đỉnh $v \in V$. Mỗi khi phát hiện

$$d[u] + a[u,v] < d[v] \qquad (1)$$

cận trên $d[v]$ sẽ được là tốt lên: $d[v] := d[u] + a[u,v]$.

Quá trình đó sẽ kết thúc khi nào chúng ta không làm tốt thêm được bất cứ cận trên nào. Khi đó, rõ ràng giá trị của mỗi $d[v]$ sẽ cho ta khoảng cách từ đỉnh s đến đỉnh v. Khi thể hiện kỹ thuật tính toán này trên máy tính, cận trên $d[v]$ sẽ được gọi là nhãn của đỉnh v, còn việc tính lại các cận trên này sẽ gọi là phép gán nhãn cho đồ thị và toàn bộ thủ tục thường gọi là thủ tục gán nhãn. Nhận thấy rằng để tính khoảng cách từ s đến t, ở đây, ta phải tính khoảng cách từ s đến tất cả các đỉnh còn lại của đồ thị. Hiện nay vẫn chưa biết thuật toán nào cho phép tìm đường đi ngắn nhất giữa hai đỉnh làm việc thực sự hiệu quả hơn những thuật toán tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh còn lại.

Sơ đồ tính toán mà ta vừa mô tả còn chưa là xác định, bởi vì còn phải chỉ ra thứ tự chọn các đỉnh u và v để kiểm tra điều kiện (1). Thứ tự chọn này có ảnh hưởng rất lớn đến hiệu quả của thuật toán.

Bây giờ ta sẽ mô tả thuật toán Ford - Bellman tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại của đồ thị. Thuật toán làm việc trong trường hợp trọng số của các cung là tuỳ ý, nhưng giả thiết rằng trong đồ thị không có chu trình âm.

```
procedure Ford_Bellman;
(*
Đầu vào: Đồ thị có hướng G=(V,E) với n đỉnh,
         s ∈ V là đỉnh xuất phát,
         a[u,v], u,v ∈ V, ma trận trọng số;
Đầu ra:  Khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại d[v], v ∈ V.
         Truoc[v], v ∈ V, ghi nhận đỉnh đi trước v trong đường đi ngắn
         nhất từ s đến v
Giả thiết: Đồ thị không có chu trình âm.
*)

begin
```

```
(*  Khởi tạo  *)
for  v ∈ V  do
begin
     d[v] := a[s,v] ;
     Truoc[v]:=s;
end;
d[s]:=0;
for k := 1  to  n-2  do
     for  v ∈ V \ {s}  do
          for  u ∈ V do
               if  d[v] > d[u] + a[u,v]  then
               begin
                    d[v] := d[u] + a[u,v] ;
                    Truoc[v] := u ;
               end;
end;
```

Tính đúng đắn của thuật toán có thể chứng minh trên cơ sở nguyên lý tối ưu của quy hoạch động. Rõ ràng là độ phức tạp tính toán của thuật toán là $O(n^3)$.

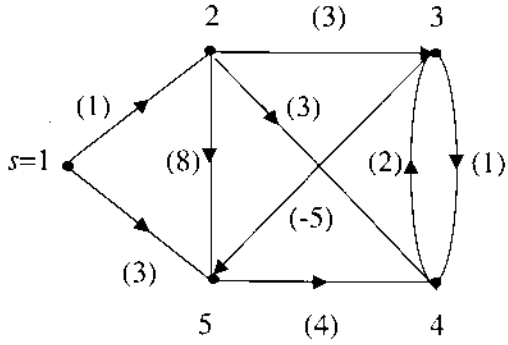
Lưu ý rằng chúng ta có thể chấm dứt vòng lặp theo k khi phát hiện trong quá trình thực hiện hai vòng lặp trong không có biến $d[v]$ nào bị đổi giá trị. Việc này có thể xảy ra đối với $k < n$-2, và điều đó làm tăng hiệu quả của thuật toán trong việc giải các bài toán thực tế. Tuy nhiên, cải tiến đó không thực sự cải thiện được đánh giá độ phức tạp của bản thân thuật toán.

Đối với đồ thị thưa tốt hơn là sử dụng danh sách kề Ke(v), $v \in V$, để biểu diễn đồ thị, khi đó vòng lặp theo u cần viết lại dưới dạng

```
for  u ∈ Ke(v)  do
     if  d[v] > d[u] + a[u,v]  then
     begin
          d[v] := d[u] + a[u,v] ;
          Truoc[v] := u ;
     end;
```

Trong trường hợp này ta thu được thuật toán với độ phức tạp $O(n.m)$.

Thí dụ 1. Xét đồ thị cho trong hình 1. Các kết quả tính toán theo thuật toán được mô tả trong bảng dưới đây



$$A = \begin{bmatrix} \infty & 1 & \infty & \infty & 3 \\ \infty & \infty & 3 & 3 & 8 \\ \infty & \infty & \infty & 1 & -5 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty \end{bmatrix}$$

Hình 1. Minh hoạ cho thuật toán Ford - Bellman

k	d[1], Truoc[1]	d[2], Truoc[2]	d[3], Truoc[3]	d[4], Truoc[4]	d[5], Truoc[5]
	0, 1	1, 1	∞, 1	∞, 1	3, 1
1	0, 1	1, 1	4, 2	4, 2	-1, 3
2	0, 1	1, 1	4, 2	3, 5	-1, 3
3	0, 1	1, 1	4, 2	3, 5	-1, 3

Bảng kết quả tính toán theo thuật toán Ford - Bellman

Trong các mục tiếp theo chúng ta sẽ xét một số trường hợp riêng của bài toán tìm đường đi ngắn nhất mà để giải chúng có thể xây dựng những thuật toán hiệu quả hơn thuật toán Ford - Bellman. Đó là khi trọng số của tất cả các cung là các số không âm hoặc là khi đồ thị không có chu trình.

6.3. Trường hợp ma trận trọng số không âm. Thuật toán Dijkstra

Trong trường hợp trọng số trên các cung là không âm thuật toán do Dijkstra đề nghị để giải bài toán tìm đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại của đồ thị làm việc hữu hiệu hơn rất nhiều so với thuật toán trình bày trong mục trước. Thuật toán được xây dựng dựa trên cơ sở gán cho các đỉnh các nhãn tạm thời. Nhãn của mỗi đỉnh cho biết cận trên của độ dài đường đi ngắn nhất từ s đến nó. Các nhãn này sẽ được biến đổi theo một thủ tục lặp, mà ở mỗi một bước lặp có một nhãn tạm thời trở thành nhãn cố định. Nếu nhãn của một đỉnh nào đó trở thành cố định thì nó sẽ cho ta không phải là cận trên mà là độ dài của đường đi ngắn nhất từ đỉnh s đến nó. Thuật toán được mô tả cụ thể như sau.

```
procedure Dijkstra;
(*  Đầu vào: Đồ thị có hướng G=(V,E) với n đỉnh,
            s ∈ V là đỉnh xuất phát,  a[u,v], u,v ∈ V, ma trận trọng số;
    Giả thiết: a[u,v] ≥ 0,  u, v ∈ V.
    Đầu ra: Khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại d[v], v∈ V.
    Truoc[v], v∈ V, ghi nhận đỉnh đi trước v trong đường đi ngắn nhất từ s đến v.
*)
begin
   (*  Khởi tạo  *)
   for  v ∈ V  do
   begin
       d[v] := a[s,v] ;
       Truoc[v]:=s;
   end;
   d[s] := 0;  T := V \ {s};   (*  T là tập các đỉnh có nhãn tạm thời  *)
   (*    Bước lặp  *)
   while  T ≠ ∅  do
    begin
        Tìm đỉnh  u  ∈  T thoả mãn  d[u] = min{ d[z] :  z  ∈  T};
        T :=  T\{u};   (* Cố định nhãn của đỉnh u *)
        for  v ∈ T  do  (* Gán nhãn lại cho các đỉnh trong T *)
            if  d[v] > d[u] + a[u,v]  then
            begin
                d[v] := d[u] + a[u,v] ;
                Truoc[v] := u ;
            end;
    end;
end;
```

Định lý 1. *Thuật toán Dijkstra tìm được đường đi ngắn nhất trên đồ thị sau thời gian cỡ* $O(n^2)$.

Chứng minh. Trước hết ta chứng minh là thuật toán tìm được đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại của đồ thị. Giả sử rằng ở một bước lặp nào đó các nhãn cố định cho ta độ dài các đường đi ngắn nhất từ s đến các đỉnh có nhãn cố định, ta sẽ chứng minh rằng ở lần lặp tiếp theo nếu đỉnh u^* thu được nhãn cố định thì $d[u^*]$ chính là độ dài đường đi ngắn nhất từ s đến u^*.

Ký hiệu S_1 là tập các đỉnh có nhãn cố định còn S_2 là tập các đỉnh có nhãn tạm thời ở bước lặp đang xét. Kết thúc mỗi bước lặp nhãn tạm thời $d[v]$ cho ta độ dài của đường đi ngắn nhất từ s đến v chỉ qua những đỉnh nằm hoàn toàn trong tập S_1. Giả sử rằng đường đi ngắn nhất từ s đến u^* không nằm trọn trong tập S_1, tức là nó đi qua ít nhất một đỉnh của tập S_2. Gọi $z \in S_2$ là đỉnh đầu tiên như vậy trên đường đi này. Do trọng số trên các cung là không âm, nên đoạn đường từ z đến u^* có độ dài $L > 0$ và

$$d[z] < d[u^*] - L < d[u^*].$$

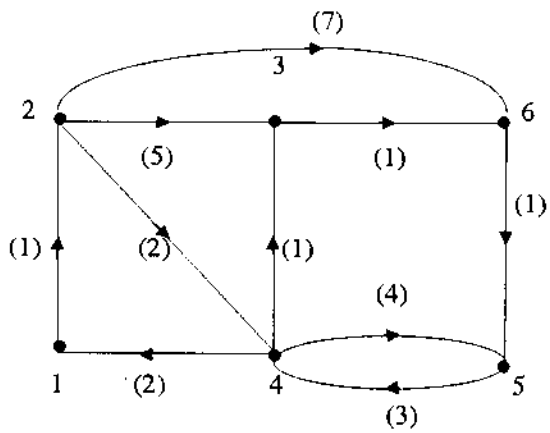
Bất đẳng thức này là mâu thuẫn với cách xác định đỉnh u^* là đỉnh có nhãn tạm thời nhỏ nhất. Vậy đường đi ngắn nhất từ s đến u^* phải nằm trọn trong S_1, và vì thế $d[u^*]$ là độ dài của nó. Do ở lần lặp đầu tiên $S_1 = \{s\}$ và sau mỗi lần lặp ta chỉ thêm vào S_1 một đỉnh u^* nên giả thiết là $d[v]$ cho độ dài đường đi ngắn nhất từ s đến v với mọi $v \in S_1$ là đúng với bước lặp đầu tiên. Theo qui nạp suy ra thuật toán cho ta đường đi ngắn nhất từ s đến mọi đỉnh của đồ thị.

Bây giờ sẽ đánh giá số phép toán cần thực hiện theo thuật toán. Ở mỗi bước lặp để tìm ra đỉnh u cần phải thực hiện $O(n)$ phép toán, và để gán nhãn lại cũng cần phải thực hiện một số lượng phép toán cũng là $O(n)$. Thuật toán phải thực hiện n-1 bước lặp, vậy thời gian tính toán của thuật toán là cỡ $O(n^2)$.

Định lý được chứng minh.

Khi đã tìm được độ dài của đường đi ngắn nhất $d[v]$ thì đường đi này có thể tìm dựa vào nhãn $Truoc[v]$, $v \in V$, theo qui tắc giống như chúng ta đã xét trong chương 3.

Thí dụ 2. Tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị ở hình 2.



Hình 2. Minh hoạ thuật toán Dijkstra

Kết quả tính toán theo thuật toán được trình bày trong bảng dưới đây. Qui ước viết hai thành phần của nhãn theo thứ tự: $d[v]$, $Truoc[v]$. Đỉnh được đánh dấu '*' là đỉnh

được chọn để cố định nhãn ở bước lặp đang xét, nhãn của nó không biến đổi ở các bước tiếp theo, vì thế ta đánh dấu -.

Bước lặp	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6
Khởi tạo	0, 1	1, 1 *	∞, 1	∞, 1	∞, 1	∞, 1
1	-	-	6, 2	3, 2 *	∞, 1	8, 2
2	-	-	4, 4 *	-	7, 4	8, 2
3	-	-	-	-	7, 4	5, 3 *
4	-	-	-	-	6, 6 *	-
5						

Bảng kết quả tính toán theo thuật toán Dijkstra

Chú ý:

1) Nếu chỉ cần tìm đường đi ngắn nhất từ s đến một đỉnh t nào đó thì có thể kết thúc thuật toán khi đỉnh t trở thành có nhãn cố định.

2) Tương tự như trong mục 6.2, dễ dàng mô tả lại thuật toán cho trường hợp đồ thị cho bởi danh sách kề. Để có thể giảm bớt khối lượng tính toán trong việc xác định đỉnh u ở mỗi bước lặp, có thể sử dụng thuật toán Heapsort (tương tự như trong chương 5 khi thể hiện thuật toán Kruskal). Khi đó có thể thu được thuật toán với độ phức tạp tính toán là $O(m \log n)$.

6.4. Đường đi trong đồ thị không có chu trình

Bây giờ ta xét trường hợp riêng thứ hai của bài toán đường đi ngắn nhất, mà để giải nó có thể xây dựng thuật toán với độ phức tạp tính toán $O(n^2)$, đó là khi đồ thị không có chu trình (còn trọng số trên các cung có thể là các số thực tuỳ ý). Trước hết ta chứng minh định lý sau.

Định lý 2. *Giả sử G là đồ thị không có chu trình. Khi đó các đỉnh của nó có thể đánh số sao cho mỗi cung của đồ thị chỉ hướng từ đỉnh có chỉ số nhỏ hơn đến đỉnh có chỉ số lớn hơn, nghĩa là mỗi cung của nó có thể biểu diễn dưới dạng $(v[i], v[j])$, trong đó $i < j$.*

Để chứng minh định lý ta mô tả thuật toán sau đây, cho phép tìm ra cách đánh số thoả mãn điều kiện định lý.

procedure Numbering;

(* **Đầu vào:** *Đồ thị có hướng $G=(V,E)$ với n đỉnh không chứa chu trình được cho bởi danh sách kề $Ke(v)$, $v \in V$.*

Đầu ra: *Với mỗi đỉnh $v \in V$ chỉ số $NR[v]$ thoả mãn:*
*Với mọi cung (u,v) của đồ thị ta đều có $NR[u] < NR[v]$. *)*

```
begin
    for  v ∈ V  do  Vao[v] := 0;
    (*  Tính Vao[v] = deg-(v)  *)
    for  u ∈ V  do
       for  v ∈ Ke(u)  do  Vao[v] := Vao[v] + 1 ;
    QUEUE := ∅ ;
    for  v ∈ V  do
        if  Vao[v] = 0  then  QUEUE ⇐ v ;
    num := 0;
    while  QUEUE ≠ ∅  do
    begin
        u ⇐ QUEUE ;
        num := num + 1 ;  NR[u] := num ;
        for  v ∈ Ke(u)  do
        begin
            Vao[v] := Vao[v] - 1 ;
            if  Vao[v] = 0  then  QUEUE ⇐ v ;
        end;
    end;
end;
```

Thuật toán được xây dựng dựa trên ý tưởng rất đơn giản sau: Rõ ràng trong đồ thị không có chu trình bao giờ cũng tìm được đỉnh có bán bậc vào bằng 0 (không có cung đi vào). Thực vậy, bắt đầu từ đỉnh v_1 nếu có cung đi vào nó từ v_2 thì ta lại chuyển sang xét đỉnh v_2. Nếu có cung từ v_3 đi vào v_2, thì ta lại chuyển sang xét v_3, ... Do đồ thị là không có chu trình nên sau một số hữu hạn lần chuyển như vậy ta phải đi đến đỉnh không có cung đi vào. Thoạt tiên, tìm các đỉnh như vậy của đồ thị. Rõ ràng ta có thể đánh số chúng theo một thứ tự tuỳ ý bắt đầu từ 1. Tiếp theo, loại bỏ khỏi đồ thị những đỉnh đã được đánh số cùng các cung đi ra khỏi chúng, ta thu được đồ thị mới cũng không có chu trình, và thủ tục được lặp lại với đồ thị mới này. Quá trình đó sẽ được tiếp tục cho đến khi tất cả các đỉnh của đồ thị được đánh số.

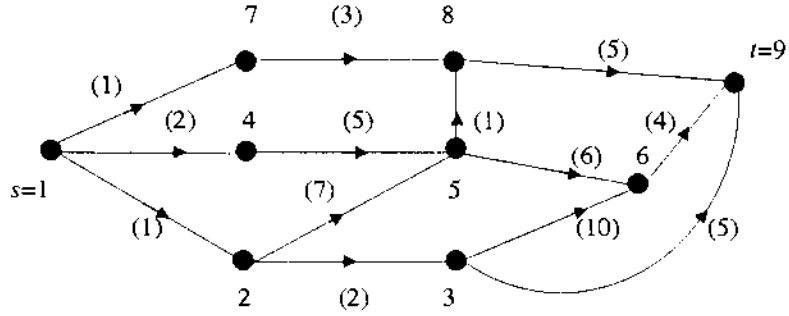
Chú ý:

1) Rõ ràng trong bước khởi tạo ta phải duyệt qua tất cả các cung của đồ thị khi tính bán bậc vào của các đỉnh, vì vậy ở đó ta tốn cỡ $O(m)$ phép toán, trong đó m là số cung của đồ thị. Tiếp theo, mỗi lần đánh số một đỉnh, để thực hiện việc loại bỏ đỉnh đã đánh số cùng với các cung đi ra khỏi nó, chúng ta lại duyệt qua tất cả các cung này. Suy ra để

đánh số tất cả các đỉnh của đồ thị chúng ta sẽ phải duyệt qua tất cả các cung của đồ thị một lần nữa. Vậy độ phức tạp của thuật toán là O(m).

2) Thuật toán có thể áp dụng để kiểm tra xem đồ thị có chứa chu trình hay không? Thực vậy, nếu kết thúc thuật toán vẫn còn có đỉnh chưa được đánh số ($num < n$) thì điều đó có nghĩa là đồ thị chứa chu trình.

Thí dụ 3. Đồ thị trong hình 3 có các đỉnh được đánh số thoả mãn điều kiện nêu trong định lý.



Hình 3. Đồ thị không có chu trình

Do có thuật toán đánh số trên, nên khi xét đồ thị không có chu trình ta có thể giả thiết là các đỉnh của nó được đánh số sao cho mỗi cung chỉ đi từ đỉnh có chỉ số nhỏ đến đỉnh có chỉ số lớn hơn. Thuật toán tìm đường đi ngắn nhất trên đồ thị không có chu trình được mô tả trong sơ đồ sau đây.

```
procedure  Critical_Path;
(*  Tìm đường đi ngắn nhất từ đỉnh nguồn đến tất cả
    các đỉnh còn lại trênđồ thị không có chu trình

Đầu vào:  Đồ thị G = (V, E), trong đó  V = { v[1], v[2], ... , v[n] }.
          Đối với mỗi cung  (v[i], v[j])  ∈  E, ta có  i < j.
          Đồ thị được cho bởi danh sách kề  Ke(v) ,  v ∈ V.

Đầu ra :  Khoảng cách từ v[1] đến tất cả các đỉnh còn lại
          được ghi trong mảng d[v[i]],  i = 2, 3, ..., n          *)
begin
     d[v[1]] := 0;
     for  j:= 2  to  n do
          d[v[j]] := a[v[1], v[j] ] ;
     for  j:= 2  to  n  do
          for  v ∈ Ke[v[j]]  do
               d[v] :=  min ( d[v],  d[v[j]] + a[v[j], v] ) ;
end;
```

Độ phức tạp tính toán của thuật toán là O(m), do mỗi cung của đồ thị phải xét qua đúng một lần.

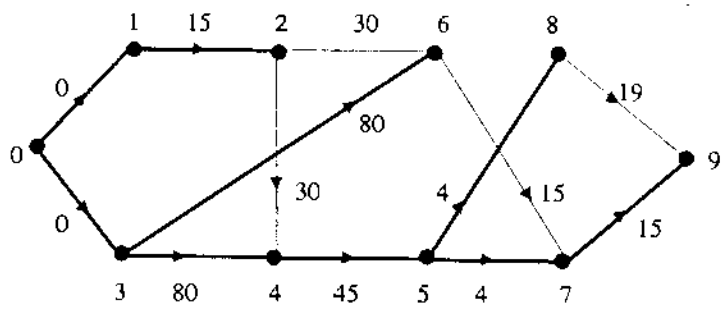
Các thuật toán mô tả ở trên thường được ứng dụng vào việc xây dựng những phương pháp giải bài toán điều khiển việc thực hiện những dự án lớn, gọi tắt là PERT (Project Evaluation and Review Technique) hay CDM (Critical path Method). Một thí dụ đơn giản cho ứng dụng này được mô tả trong thí dụ dưới đây.

Thí dụ 4. Việc thi công một công trình lớn được chia ra làm n công đoạn, đánh số từ 1 đến n. Có một số công đoạn mà việc thực hiện nó chỉ được tiến hành sau khi một số công đoạn nào đó đã hoàn thành. Đối với mỗi công đoạn i biết $t[i]$ là thời gian cần thiết để hoàn thành nó (i = 1, 2,..., n). Các dữ liệu với n = 8 được cho trong bảng sau đây

Công đoạn	t[i]	Các công đoạn phải được hoàn thành trước nó
1	15	không có
2	30	1
3	80	không có
4	45	2 , 3
5	124	4
6	15	2 , 3
7	15	5 , 6
8	19	5

Giả sử thời điểm bắt đầu tiến hành thi công công trình là 0. Hãy tìm tiến độ thi công công trình (chỉ rõ mỗi công đoạn phải được bắt đầu thực hiện vào thời điểm nào) để cho công trình được hoàn thành xong trong thời điểm sớm nhất có thể được.

Ta có thể xây dựng đồ thị có hướng n đỉnh biểu diễn hạn chế về trình tự thực hiện các công việc như sau: Mỗi đỉnh của đồ thị tương ứng với một công việc, nếu công việc i phải được thực hiện trước công đoạn j thì trên đồ thị có cung (i,j), trọng số trên cung này được gán bằng t[i], xem hình 4 dưới đây.



Hình 4. Đồ thị minh hoạ PERT

Thêm vào đồ thị 2 đỉnh 0 và $n+1$ tương ứng với hai sự kiện đặc biệt: đỉnh số 0 tương ứng với công đoạn *Lễ khởi công*, nó phải được thực hiện trước tất cả các công đoạn khác, và đỉnh $n+1$ tương ứng với công đoạn *Cắt băng khánh thành công trình*, nó phải thực hiện sau tất cả các công đoạn, với $t[0] = t[n+1] = 0$ (trên thực tế chỉ cần nối đỉnh 0 với tất cả các đỉnh có bán bậc vào bằng 0 và nối tất cả các đỉnh có bán bậc ra bằng 0 với đỉnh $n+1$). Gọi đồ thị thu được là G. Rõ ràng bài toán đặt ra dẫn về bài toán tìm đường đi dài nhất từ đỉnh 0 đến tất cả các đỉnh còn lại trên đồ thị G. Do đồ thị G rõ ràng là không chứa chu trình, nên để giải bài toán đặt ra có thể áp dụng các thuật toán mô tả ở trên, chỉ cần đổi dấu tất cả các trọng số trên các cung thành dấu ngược lại, hoặc đơn giản hơn chỉ cần đổi toán tử ***min*** trong thuật toán Critical_Path thành toán tử ***max***. Kết thúc thuật toán, chúng ta thu được $d[v]$ là độ dài đường đi dài nhất từ đỉnh 0 đến đỉnh v. Khi đó $d[v]$ cho ta thời điểm sớm nhất có thể bắt đầu thực hiện công đoạn v, nói riêng $d[n+1]$ là thời điểm sớm nhất có thể cắt băng khánh thành, tức là thời điểm sớm nhất có thể hoàn thành toàn bộ công trình.

Cây đường đi dài nhất của bài toán trong thí dụ 4 tìm được theo thuật toán được chỉ ra trong hình 4.

6.5. Đường đi ngắn nhất giữa tất cả các cặp đỉnh

Rõ ràng ta có thể giải bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị bằng cách sử dụng n lần thuật toán mô tả ở mục trước, trong đó ta sẽ chọn s lần lượt là các đỉnh của đồ thị. Rõ ràng, khi đó ta thu được thuật toán với độ phức tạp là $O(n^4)$ (nếu sử dụng thuật toán Ford - Bellman) hoặc $O(n^3)$ đối với trường hợp trọng số không âm hoặc đồ thị không có chu trình. Trong trường hợp tổng quát, sử dụng thuật toán Ford - Bellman n lần không phải là cách làm tốt nhất. ở đây ta sẽ mô tả một thuật toán giải bài toán trên với độ phức tạp tính toán $O(n^3)$: Thuật toán Floyd. Thuật toán được mô tả trong thủ tục dưới đây.

procedure Floyd;

(* ***Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh***

- **Đầu vào:** *Đồ thị cho bởi ma trận trọng số* $a[i,j]$, $i, j = 1,2,...,n$.
- **Đầu ra :**

Ma trận đường đi ngắn nhất giữa các cặp đỉnh

$d[i,j], i, j = 1,2,...,n,$

trong đó $d[i,j]$ *cho độ dài đường đi ngắn nhất từ* i *đến* j.

Ma trận ghi nhận đường đi

$p[i,j], i, j = 1,2,...,n,$

trong đó $p[i,j]$ *ghi nhận đỉnh đi trước đỉnh* j *trong đường đi ngắn nhất từ* i *đến* j.

*)

```
begin
      (*   Khởi tạo   *)
      for i := 1   to   n   do
          for j := 1   to   n   do
          begin
              d[i,j]  := a[i,j];
              p[i,j]  := i;
          end;
      (*   Bước lặp   *)
      for   k:= 1   to   n   do
         for  i := 1   to   n   do
             for j := 1   to   n   do
                 if   d[i,j]  >  d[i,k] +  d[k,j]   then
                 begin
                     d[i,j]  :=  d[i,k] +  d[k,j] ;
                     p[i,j]  :=  p[k,j];
                 end;
end;
```

Rõ ràng độ phức tạp tính toán của thuật toán là $O(n^3)$.

Kết thúc chương này chúng ta trình bày một cách thể hiện thuật toán Dijkstra trên ngôn ngữ PASCAL:

```
(* Chương trình tìm đường đi ngắn nhất từ đỉnh s đến đỉnh t theo thuật toán Dijkstra *)
uses crt;
const
  max = 50;
var
   n, s, t : integer;
   chon : char;
   Truoc : array [1..max] of  byte;
   d       : array [1..max] of  integer;
   a       : array [1..max,1..max] of  integer;
   final   : array [1..max] of  boolean;

procedure NhapSoLieu;
var
   f       : text;
   fname : string;
```

```
  i,j     : integer;
begin
  write ('Vao ten file du lieu can doc: ');
  readln(fname);
  assign(f,fname);
  reset(f);
  readln(f,n);
  for i:= 1 to n do
    for j:= 1 to n do read(f,a[i,j]);
  close(f);
end;

procedure InSoLieu;
var
  i, j : integer;
begin
  writeln(' So dinh cua do thi : ',n);
  writeln(' Ma tran khoang cach:');
  for i := 1 to n do
  begin
    for j:= 1 to n do write(a[i,j]:3,' ');
    writeln;
  end;
end;

procedure InKetQua;
var
  i, j : integer;
begin
  writeln (' DUONG DI NGAN NHAT TU ',s,' DEN ',t);
  write(t,' <= ');
  i:=Truoc[t];
  while i<>s do
  begin
    write(i,' <= ');
    i:=Truoc[i];
  end;
  writeln(s);
  writeln('Do dai cua duong di la: ',d[t]);
end;
```

```
procedure Dijkstra;
var  v, u, minp : integer;
begin
  write('Tim duong di tu s = ');
  readln(s);
  write('          den t = ');
  readln(t);
  for v:=1 to n do
  begin   (* Khởi tạo nhãn *)
     d[v]:=a[s,v];
     Truoc[v]:=s;
     final[v]:=false;
  end;
  Truoc[s]:=0;
  d[s]:=0;
  final[s]:=true;
  while not final[t] do   (***** Bước lặp *****)
  begin
    { Tìm u là đỉnh có nhãn tạm thời nhỏ nhất }
    minp:= maxint;
    for v:=1 to n do
      if (not final[v]) and (minp > d[v]) then
      begin
         u:= v;
         minp:= d[v];
      end;
    final[u]:= true;
    if not final[t] then
       for v:=1 to n do
          if ( not final[v] ) and ( d[u] + a[u,v] < d[v] ) then
             begin
                d[v]:= d[u] + a[u,v];
                Truoc[v]:= u;
             end;
          end;
  end;
end;
```

```
procedure Menu;
begin
  clrscr;
  writeln('  CHUONG TRINH TIM  DUONG DI NGAN NHAT TU s DEN t  ');
  writeln('          THEO THUAT TOAN DIJKSTRA');
  writeln('       ******************************');
  writeln('          1. Nhap so lieu tu file.');
  writeln('          2. Giai bai toan.');
  writeln('          3. Ket thuc.');
  writeln('-------------------------------------------------------------');
  write('       Hay chon chuc nang: '#7);
end;

BEGIN  (*  Chương trình chính  *)
  repeat
     Menu;  chon:=readkey;  writeln(chon);
     case chon of
        '1': NhapSoLieu;
        '2': begin
               InSoLieu;  Dijkstra;  InKetQua;
             end;
        '3': Exit;
     end;
  until false;
END.
```

Bài tập

1. Lập trình thực hiện các thuật toán mô tả trong chương 6:

a) Thuật toán Ford - Bellman;

b) Thuật toán Dijkstra;

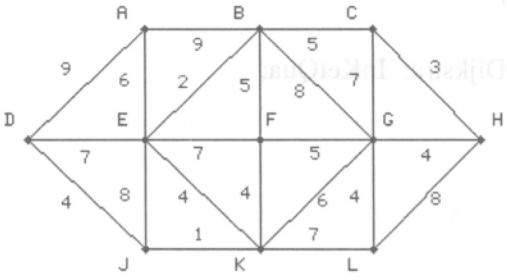
c) Thuật toán Floyd;

d) Thuật toán tìm đường đi dài nhất trên đồ thị không có chu trình (PERT).

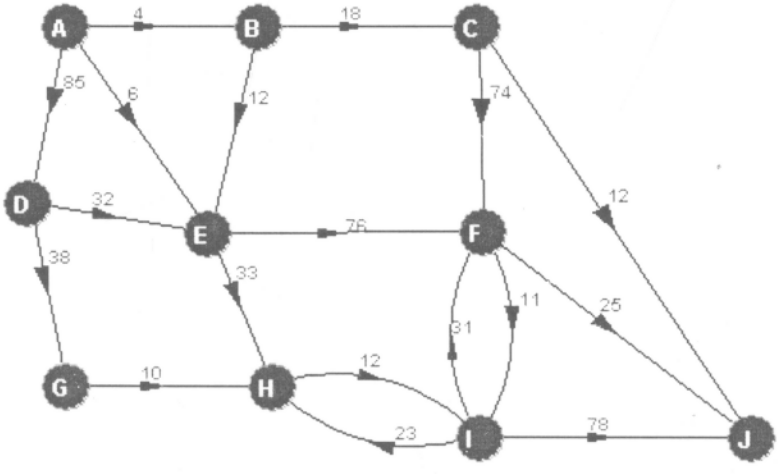
2. Trong trường hợp đồ thị thưa, các thuật toán Ford - Bellman, Dijkstra, Floyd sẽ làm việc hiệu quả hơn nếu ta sử dụng danh sách kề để biểu diễn đồ thị. Hãy mô tả lại các thuật toán trên cho trường hợp này, và đánh giá độ phức tạp tính toán của chúng.

3. Tìm đường đi ngắn nhất theo thuật toán Dijkstra từ đỉnh A đến tất cả các đỉnh còn lại trên đồ thị cho trong các hình vẽ dưới đây:

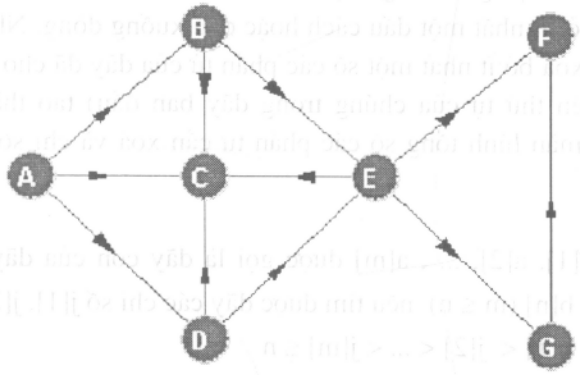
a)



b)



4. Áp dụng thuật toán đánh số đỉnh để kiểm tra đồ thị sau đây có chứa chu trình hay không



5. Xét đồ thị gồm 7 đỉnh A, B, C, D, E, F, G cho bởi ma trận trọng số

	A	B	C	D	E	F	G
A	0	11	65	17	65	65	65
B	65	0	12	65	65	10	16
C	65	65	0	13	14	65	19
D	65	65	65	0	65	65	18
E	65	65	65	65	0	65	15
F	65	13	18	65	65	0	10
G	65	65	65	65	65	65	0

Tìm đường đi ngắn nhất từ đỉnh A đến đỉnh G theo thuật toán Dijkstra. Yêu cầu viết rõ kết quả trung gian trong từng bước. Kết quả cuối cùng: Đường đi ngắn nhất từ A đến G và độ dài của nó.

6. Cho một bảng gồm $m \times n$ ô. Nếu một ô nằm ở hàng i cột j của bảng thì ta sẽ nói ô này có toạ độ (i,j) (i=1,2,...,m; j=1,2,...,n). Trong mỗi ô viết một số nguyên (gọi là giá trị của ô). Từ một ô bất kỳ của bảng chỉ có thể di chuyển đến những ô ở bên phải có chung cạng hoặc đỉnh với nó. Dữ liệu về bảng nói trên được cho trong một file văn bản có tên DATA3.INP, dòng đầu tiên của nó chứa hai số m, n được ghi cách nhau bởi dấu cách. Mỗi dòng thứ i trong số m dòng tiếp theo chứa n số ghi trong n ô của dòng thứ i của bảng, các số được ghi cách nhau bởi dấu cách (i=1,2,...,m). Hãy lập trình nhập dữ liệu vào từ file, sau đó tìm dãy các di chuyển từ mép trái của bảng sang mép phải của bảng sao cho tổng số các giá trị của các ô đi qua là nhỏ nhất. Yêu cầu kết quả đưa ra dưới dạng dãy toạ độ của các ô cần di chuyển qua bắt đầu từ một ô xuất phát ở mép trái và kết thúc ở một ô nào đó ở mép phải, và tổng số các giá trị của các ô này.

7. Cho dãy n số nguyên $x[1], x[2],..., x[n]$, trong đó $1<n<1000$ và $1<x[i]<200$, $i = 1, 2,...,n$. Dãy số đã cho được ghi trong một file văn bản có tên là DATA4.INP, các số được ghi cách nhau bởi ít nhất một dấu cách hoặc dấu xuống dòng. Nhập dữ liệu vào từ file, sau đó tìm cách xoá bỏ ít nhất một số các phần tử của dãy đã cho sao cho các phần tử còn lại (giữ nguyên thứ tự của chúng trong dãy ban đầu) tạo thành một dãy con không giảm. Đưa ra màn hình tổng số các phần tử cần xoá và chỉ số của chúng trong dãy đã cho.

8. Dãy số nguyên a[1], a[2], ... , a[m] được gọi là dãy con của dãy con của dãy số nguyên b[1], b[2],..., b[n] (m < n) nếu tìm được dãy các chỉ số j[1], j[2],...,j[m] sao cho

$$1 \le j[1] < j[2] < ... < j[m] \le n$$

và

$$a[i] = b[j[i]], i=1,2,...,m. \quad (1)$$

Cho hai dãy số nguyên

$$c[1], c[2], ... , c[p], \quad (2)$$

$$d[1], d[2], ... , d[q]. \quad (3)$$

Dữ liệu về hai dãy số được ghi trong file văn bản có tên là DATA5.INP, dòng đầu tiên chứa hai số p, q được ghi cách nhau bởi dấu cách. Trong cách dòng tiếp theo ghi lần lượt các phần tử của dãy (2) rồi tiếp đến các phần tử của dãy (3), các số ghi cách nhau bởi ít nhất một dấu cách hoặc dấu xuống dòng. Hãy lập trình nhập dữ liệu vào từ file, sau đó tìm dãy con chung có số phần tử là lớn nhất của hai dãy đã cho. Kết quả đưa ra file có tên KQUA5.OUT: dòng đầu tiên chứa số phần của dãy con chung tìm được, trong các dòng tiếp theo lần lượt ghi các phần tử của hai dãy chỉ số tương ứng chỉ rõ vị trí của dãy con này trong hai dãy ban đầu (các số ghi cách nhau bởi ít nhất một dấu cách hoặc dấu xuống dòng.

9. Cho bàn cờ quốc tế gồm $n.n$ ô ($n < 50$). Trong mỗi ô (i,j) của nó ta điền một số nguyên dương $a[i,j] <100$, $i, j = 1,2,...,n$. Từ một ô bất kỳ (i,j) của bảng ta có thể di chuyển sang ô có cùng mầu với nó nếu như $a[i,j]$ là số chẵn, và sang ô khác màu với nó nếu $a[i,j]$ là số lẻ. Dữ liệu được cho trong một file văn bản có tên là DATA6.INP, dòng đầu tiên chứa số n. Trong các dòng tiếp theo chứa các phần tử $a[1,1]$, $a[1,2]$, ... $a[1,n]$, ... $a[n,1]$, $a[n,2]$, ... $a[n,n]$, các số được lần lượt ghi theo thứ tự vừa nêu và cách nhau bởi ít nhất một dấu cách hoặc dấu xuống dòng.

Yêu cầu: Tìm một cách di chuyển một ô nào đó ở cột thứ nhất sang một ô nào đó ở cột thứ n của bàn cờ sao cho tổng các số ghi trong các ô di chuyển qua là nhỏ nhất. Đưa ra màn hình kết quả dưới dạng ba số nguyên x, y, L, trong đó x là toạ độ dòng của ô xuất phát, y - toạ độ dòng của ô kết thúc, L - tổng các số ghi trong các ô cần di chuyển qua.

7

BÀI TOÁN LUỒNG CỰC ĐẠI TRONG MẠNG

Bài toán luồng cực đại trong mạng là một trong số những bài toán tối ưu trên đồ thị tìm được những ứng dụng rộng rãi trong thực tế cũng như những ứng dụng thú vị trong lý thuyết tổ hợp. Bài toán được đề xuất vào đầu những năm 1950, và gắn liền với tên tuổi của hai nhà toán học Mỹ là Ford và Fulkerson. Trong chương này chúng ta sẽ trình bày thuật toán của Ford và Fulkerson để giải bài toán đặt ra và nêu một số ứng dụng của bài toán.

7.1. Mạng. Luồng trong mạng. Bài toán luồng cực đại

Định nghĩa 1. *Ta gọi mạng là đồ thị có hướng $G = (V, E)$, trong đó có duy nhất một đỉnh s không có cung đi vào gọi là điểm phát, duy nhất một đỉnh t không có cung đi ra gọi là điểm thu và mỗi cung $e = (v,w) \in E$ được gán với một số không âm $c(e) = c(v,w)$ gọi là khả năng thông qua của cung e.*

Để thuận tiện cho việc trình bày ta sẽ qui ước rằng nếu không có cung (v,w) thì khả năng thông qua $c(v,w)$ được gán bằng 0.

Định nghĩa 2. *Giả sử cho mạng $G=(V, E)$. Ta gọi luồng f trong mạng $G = (V, E)$ là ánh xạ $f : E \to R_+$ gán cho mỗi cung $e = (v,w) \in E$ một số thực không âm $f(e) = f(v,w)$, gọi là luồng trên cung e, thoả mãn các điều kiện sau:*

1) *Luồng trên mỗi cung $e \in E$ không vượt quá khả năng thông qua của nó:*

$$0 \le f(e) \le c(e),$$

2) *Điều kiện cân bằng luồng trên mỗi đỉnh của mạng: Tổng luồng trên các cung đi vào đỉnh v bằng tổng luồng trên các cung đi ra khỏi đỉnh v, nếu $v \neq s, t$:*

$$Div_f(v) = \sum_{w \in \Gamma^-(v)} f(w,v) - \sum_{w \in \Gamma^+(v)} f(v,w).$$

trong đó $\Gamma^-(v)$ - tập các đỉnh của mạng mà từ đó có cung đến v, $\Gamma^+(v)$ - tập các đỉnh của mạng mà từ v có cung đến nó:

$$\Gamma^-(v) = \{w \in V: (w, v) \in E\}, \quad \Gamma^+(v) = \{w \in V: (v, w) \in E\}.$$

3) *Ta gọi giá trị của luồng f là số*

$$val(f) = \sum_{w \in \Gamma^-(s)} f(s,w) = \sum_{w \in \Gamma^-(t)} f(w,t).$$

Chú ý là đẳng thức thứ hai trong định nghĩa giá trị luồng là hệ quả của điều kiện cân bằng luồng.

Bài toán luồng cực đại trong mạng: Cho mạng $G= (V, E)$. Hãy tìm luồng f^* trong mạng với giá trị luồng $val(f^*)$ là lớn nhất. Luồng như vậy ta sẽ gọi là luồng cực đại trong mạng.

Bài toán như vậy có thể xuất hiện trong rất nhiều ứng dụng thực tế. Chẳng hạn khi cần xác định cường độ lớn nhất của dòng vận tải giữa 2 nút của một bản đồ giao thông. Trong ví dụ này lời giải của bài toán luồng cực đại sẽ chỉ cho ta các đoạn đường đông xe nhất và chúng tạo thành "chỗ hẹp" tương ứng với dòng giao thông xét theo 2 nút được chọn. Một ví dụ khác là nếu xét đồ thị tương ứng với một hệ thống đường ống dẫn dầu. Trong đó các ống tương ứng với các cung, điểm phát có thể coi là tàu chở dầu, điểm thu là bể chứa, còn những điểm nối giữa các ống là các nút của đồ thị. Khả năng thông qua của các cung tương ứng với tiết diện các ống. Cần phải tìm luồng dầu lớn nhất có thể bơm từ tàu chở dầu vào bể chứa.

7.2. Lát cắt. Đường tăng luồng. Định lý Ford - Fulkerson

Định nghĩa 3. *Ta gọi lát cắt (X, X^*) là một cách phân hoạch tập đỉnh V của mạng ra thành hai tập X và $X^* = V \setminus X$, trong đó $s \in X$ và $t \in X^*$. Khả năng thông qua của lát cắt (X, X^*) là số*

$$c(X, X^*) = \sum_{\substack{v \in X \\ w \in X^*}} c(v, w).$$

Lát cắt với khả năng thông qua nhỏ nhất được gọi là ***lát cắt hẹp nhất.***

Bổ đề 1. *Giá trị của mọi luồng f trong mạng luôn nhỏ hơn hoặc bằng khả năng thông qua của lát cắt (X, X^*) bất kỳ trong nó: $val(f) \le c(X, X^*)$.*

Chứng minh. Cộng các điều kiện cân bằng luồng $Div_f(v) = 0$ với mọi $v \in X$. Khi đó ta có

$$\sum_{v \in X} \Big(\sum_{w \in \Gamma^-(v)} f(w, v) - \sum_{w \in \Gamma^+(v)} f(v, w) \Big) = -val(f)$$

tổng này sẽ gồm các số hạng dạng $f(u,v)$ với dấu cộng hoặc dấu trừ mà trong đó có ít nhất một trong hai đỉnh u, v phải thuộc tập X. Nếu cả hai đỉnh u, v đều trong tập X, thì $f(u,v)$ xuất hiện với dấu cộng trong $Div_f(v)$ và với dấu trừ trong $Div_f(u)$, vì thế, chúng triệt tiêu lẫn nhau. Do đó, sau khi giản ước các số hạng như vậy ở vế trái, ta thu được

$$-\sum_{\substack{v \in X \\ w \in X^*}} f(v, w) + \sum_{\substack{v \in X^* \\ w \in X}} f(v, w) = -val(f),$$

hay là

$$val(f) = \sum_{\substack{v \in X \\ w \in X^*}} f(v, w) - \sum_{\substack{v \in X^* \\ w \in X}} f(v, w).$$

Mặt khác, từ điều kiện 1 rõ ràng là

$$\sum_{\substack{v \in X \\ w \in X^*}} f(v, w) \le \sum_{\substack{v \in X \\ w \in X^*}} c(v, w),$$

còn

$$-\sum_{\substack{v \in X^* \\ w \in X}} f(v, w) \le 0.$$

suy ra $val(f) \le c(X, X^*)$. Bổ đề được chứng minh.

Từ bổ đề 1 suy ra

Hệ quả 1. *Giá trị luồng cực đại trong mạng không vượt quá khả năng thông qua của lát cắt hẹp nhất trong mạng.*

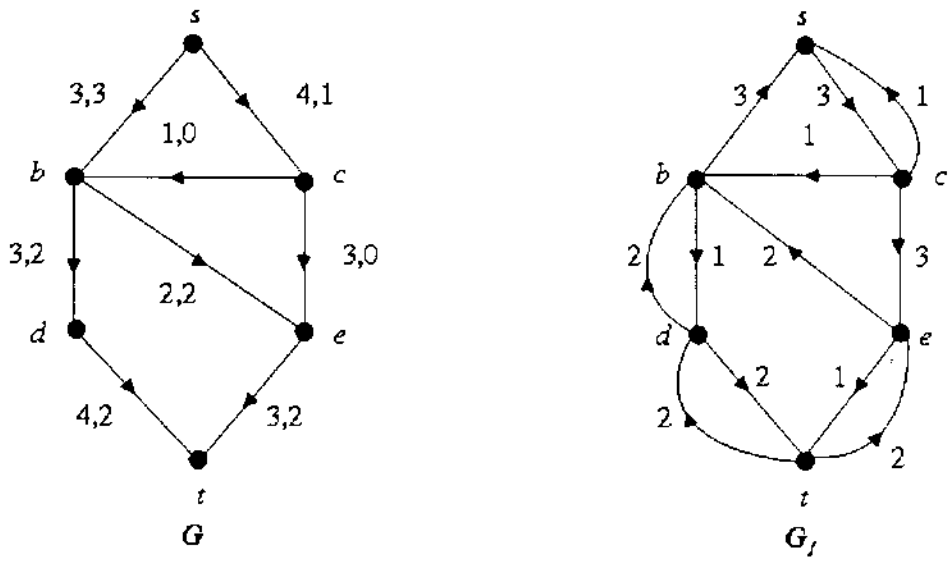
Ford và Fulkerson đã chứng minh rằng giá trị luồng cực đại trong mạng đúng bằng khả năng thông qua của lát cắt hẹp nhất. Để có thể phát biểu và chứng minh kết quả này chúng ta sẽ cần thêm một số khái niệm.

Giả sử f là một luồng trong mạng $G = (V, E)$. Từ mạng $G = (V, E)$ ta xây dựng đồ thị có trọng số trên cung $G_f = (V, E_f)$, với tập cung E_f và trọng số trên các cung được xác định theo quy tắc sau:

1) Nếu $e = (v,w) \in E$ với $f(v,w) = 0$, thì $(v,w) \in E_f$ với trọng số $c(v,w)$;
2) Nếu $e = (v,w) \in E$ với $f(v,w) = c(v,w)$, thì $(w,v) \in E_f$ với trọng số $f(v,w)$;
3) Nếu $e = (v,w) \in E$ với $0 < f(v,w) < c(v,w)$, thì $(v,w) \in E_f$ với trọng số $c(v,w) - f(v,w)$ và $(w,v) \in E_f$ với trọng số $f(v,w)$.

Các cung của G_f đồng thời cũng là cung của G được gọi là ***cung thuận***, các cung còn lại được gọi là ***cung nghịch.*** Đồ thị G_f được gọi là ***đồ thị tăng luồng.***

Thí dụ: Các số viết cạnh các cung của G ở hình 1 theo thứ tự là khả năng thông qua và luồng trên cung.



Hình 1. Mạng G và luồng f. Đồ thị có trọng số G_f tương ứng.

Giả sử $P = (s = v_0, v_1, v_2, \ldots, v_k = t)$ là một đường đi từ s đến t trên đồ thị tăng luồng G_f. Gọi δ là giá trị nhỏ nhất của các trọng số của các cung trên đường đi P. Xây dựng luồng f' trên mạng G theo quy tắc sau:

$$f'(u,v) = \begin{cases} f(u,v) + \delta, & \text{nếu } (u,v) \in P \text{ là cung thuận,} \\ f(u,v) - \delta, & \text{nếu } (u,v) \in P \text{ là cung nghịch,} \\ f(u,v), & \text{nếu } (u,v) \notin P. \end{cases}$$

Dễ dàng kiểm tra được rằng f' được xây dựng như trên là luồng trong mạng và $val(f') = val(f) + \delta$. Ta sẽ gọi thủ tục biến đổi luồng vừa nêu là ***tăng luồng dọc theo đường P***.

Định nghĩa 4. *Ta gọi đường tăng luồng f là mọi đường đi từ s đến t trên đồ thị tăng luồng $G(f)$.*

Định lý dưới đây cho mối liên hệ giữa luồng cực đại, đường tăng luồng và lát cắt.

Định lý 1. *Các mệnh đề dưới đây là tương đương:*

(i) f là luồng cực đại trong mạng;

(ii) Không tìm được đường tăng luồng f;

(iii) $val(f) = c(X,X^*)$ với một lát cắt (X, X^*) nào đó.

Chứng minh.

(i) $\Rightarrow$ *(ii)*. Giả sử ngược lại, tìm được đường tăng luồng P. Khi đó ta có thể tăng giá trị luồng bằng cách tăng luồng dọc theo đường P. Điều đó mâu thuẫn với tính cực đại của luồng f.

(ii) $\Rightarrow$ *(iii)*. Giả sử không tìm được đường tăng luồng. Ký hiệu X là tập tất cả các đỉnh có thể đến được từ đỉnh s trong đồ thị G_f, và đặt $X^* = V \setminus X$. Khi đó (X, X^*) là lát cắt, và $f(v,w) = 0$ với mọi $v \in X^*$, $w \in X$ nên

$$val(f) = \sum_{\substack{v \in X \\ w \in X^*}} f(v,w) - \sum_{\substack{v \in X^* \\ w \in X}} f(v,w) = \sum_{\substack{v \in X \\ w \in X^*}} f(v,w)$$

Với $v \in X$, $w \in X^*$, do $(v, w) \notin G_f$, nên $f(v, w) = c(v, w)$. Vậy

$$val(f) = \sum_{\substack{v \in X \\ w \in X^*}} f(v,w) = \sum_{\substack{v \in X \\ w \in X^*}} c(v,w) = c(X, X^*).$$

(iii) $\Rightarrow$ *(i)*. Theo Bổ đề 1, $val(f) \le c(X, X^*)$ với mọi luồng f và với mọi lát cắt (X, X^*). Vì vậy, từ đẳng thức $val(f) = c(X, X^*)$ suy ra luồng f là luồng cực đại trong mạng.

Định lý được chứng minh.

7.3. Thuật toán tìm luồng cực đại trong mạng

Định lý 1 là cơ sở để xây dựng thuật toán lặp sau đây để tìm luồng cực đại trong mạng: Bắt đầu từ luồng với luồng trên tất cả các cung bằng 0 (ta sẽ gọi luồng như vậy là luồng không), và lặp lại bước lặp sau đây cho đến khi thu được luồng mà đối với nó không còn đường tăng:

Bước lặp tăng luồng (Ford - Fulkerson): Tìm đường tăng P đối với luồng hiện có. Tăng luồng dọc theo đường P.

Khi đã có luồng cực đại, lát cắt hẹp nhất có thể tìm theo thủ tục mô tả trong chứng minh định lý 1. Sơ đồ của thuật toán Ford - Fulkerson có thể mô tả trong thủ tục sau đây:

```
procedure Max_Flow;
(* Thuật toán Ford - Fulkerson *)
begin
    (* Khởi tạo: Bắt đầu từ luồng với giá trị 0 *)
    for u ∈ V do
        for v ∈ V do f(u,v):=0;
    Stop:=false;
    while not Stop do
        if <Tìm được đường tăng luồng P> then <Tăng luồng dọc theo P>
        else Stop:=true;
end;
```

Để tìm đường tăng luồng trong G_f có thể sử dụng thuật toán tìm kiếm theo chiều rộng (hay tìm kiếm theo chiều sâu) bắt đầu từ đỉnh s, trong đó không cần xây dựng tường minh đồ thị G_f. Ford - Fulkerson đề nghị thuật toán gán nhãn chi tiết sau đây để giải bài toán luồng cực đại trong mạng. Thuật toán bắt đầu từ luồng chấp nhận được nào đó trong mạng (có thể bắt đầu từ luồng không), sau đó ta sẽ tăng luồng bằng cách tìm các đường tăng luồng. Để tìm đường tăng luồng ta sẽ áp dụng phương pháp gán nhãn cho các đỉnh. Mỗi đỉnh trong quá trình thực hiện thuật toán sẽ ở một trong 3 trạng thái: chưa có nhãn, có nhãn chưa xét, có nhãn đã xét. Nhãn của một đỉnh v gồm 2 phần và có một trong 2 dạng sau: $[+p(v),\varepsilon(v)]$ hoặc $[-p(v),\varepsilon(v)]$. Phần thứ nhất $+p(v)$ (hoặc $-p(v)$) chỉ ra là cần tăng (hoặc giảm) luồng theo cung $(p(v), v)$ (cung $(v, p(v))$ còn phần thứ hai $\varepsilon(v)$ chỉ ra lượng lớn nhất có thể tăng hoặc giảm luồng theo cung này. Đầu tiên chỉ có đỉnh s được khởi tạo nhãn và nhãn của nó là chưa xét, còn tất cả các đỉnh còn lại đều chưa có nhãn. Từ s ta gán nhãn cho tất cả các đỉnh kề với nó và nhãn của đỉnh s sẽ trở thành đã xét. Tiếp theo, từ mỗi đỉnh v có nhãn chưa xét ta lại gán nhãn cho tất cả các

đỉnh chưa có nhãn kề với nó và nhãn của đỉnh v trở thành đã xét. Quá trình sẽ được lặp lại cho đến khi hoặc là đỉnh t trở thành có nhãn hoặc là nhãn của tất cả các đỉnh có nhãn đều là đã xét nhưng đỉnh t vẫn không có nhãn. Trong trường hợp thứ nhất ta tìm được đường tăng luồng, còn trong trường hợp thứ hai đối với luồng đang xét không tồn tại đường tăng luồng (tức là luồng đã là cực đại). Mỗi khi tìm được đường tăng luồng, ta lại tăng luồng theo đường tìm được, sau đó xoá tất cả các nhãn và đối với luồng mới thu được lại sử dụng phép gán nhãn các đỉnh để tìm đường tăng luồng. Thuật toán sẽ kết thúc khi nào đối với luồng đang có trong mạng không tìm được đường tăng luồng.

Hai thủ tục Tìm đường tăng luồng và Tăng luồng có thể mô tả như sau.

```
procedure Find_Path;
(*  Thủ tục gán nhãn tìm đường tăng luồng
    p[v], ε[v] là nhãn của đỉnh v;
    V_T - danh sách các đỉnh có nhãn nhưng chưa xét;
    c[u,v] - khả năng thông qua của cung (u, v), u, v ∈ V;
    f[u,v] - luồng trên cung (u, v), (u, v ∈ V)    *)
begin
    p[s] := s;
    ε[s]:= +∞;
    V_T = V\{s};
    PathFound:=true;
    while  V_T ≠ ∅  do
    begin
        u ⇐ V_T ;  (*  Lấy  u  từ V_T *)
        for  v ∈ V\V_T do
        begin
            if  (c[u, v] > 0) and (f[u, v] < c[u,v]) then
            begin
                p[v]:= u;
                ε[v]:= min { ε[u], c[u,v] - f[u,v];
                V_T = V_T ∪ {v}; (* Nạp v vào danh sách đỉnh có nhãn *)
                if  v = t then  exit;
            end;
            if  (c[v,u] > 0) and (f[v, u] > 0) then
            begin
                p[v]:= -u;
                ε[v]:= min { ε[u], f[v,u]};
                V_T = V_T ∪ {v}; (* Nạp v vào danh sách đỉnh có nhãn *)
```

```
                if v = t then exit;
            end;
    end;
    PathFound:=false;
end;

procedure Inc_Flow;
(* Tăng luồng theo đường tăng *)
begin
    v := p[t]; u:=t; tang:= ε[t];
    while u ≠ s do
    begin
        if v > 0 then f[v, u] := f[v, u] + tang
        esle
        begin
            v:= -v;
            f[u,v]:= f[u,v] - tang;
        end;
        u:=v; v:=p[u];
    end;
end;
```

Thuật toán Ford - Fulkerson được thực hiện nhờ thủ tục:

```
procedure Max_Flow;
(* Thuật toán Ford - Fulkerson *)
begin
    (* Khởi tạo: Bắt đầu từ luồng với giá trị 0 *)
    for u ∈ V do
        for v ∈ V do f[u,v]:=0;
    Stop:=false;
    while not Stop do
    begin
        Find_Path;
        if PathFound then Inc_Flow
        else Stop:=true;
    end;
    < Luồng cực đại trong mạng là f[u,v], u, v ∈ V >
    < Lát cắt hẹp nhất là (V_T, V\V_T) >
end;
```

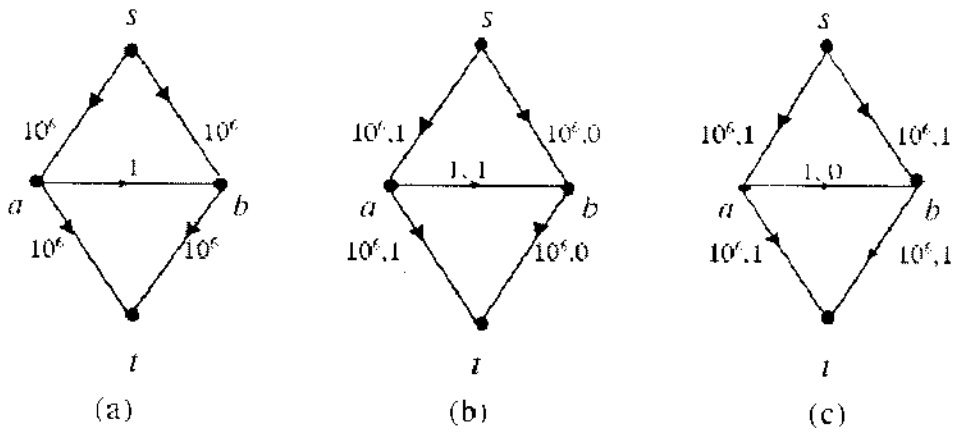
Giả sử là khả năng thông qua của tất cả các cung của đồ thị là các số nguyên. Khi đó sau mỗi lần tăng luồng, giá trị luồng sẽ tăng lên ít nhất là 1. Từ đó suy ra thuật toán Ford - Fulkerson sẽ dừng sau không quá $val(f^*)$ lần tăng luồng và cho ta luồng cực đại trong mạng. Đồng thời, rõ ràng $f(u,v)$ sẽ là số nguyên đối với mỗi cung $(u, v) \in E$. Từ đó ta có các kết quả sau:

Định lý 2 ***(Định lý về luồng cực đại trong mạng và lát cắt hẹp nhất).*** *Luồng cực đại trong mạng bằng khả năng thông qua của lát cắt hẹp nhất.*

Định lý 3. (Định lý về tính nguyên). *Nếu tất cả các khả năng thông qua là các số nguyên thì luôn tìm được luồng cực đại với luồng trên các cung là các số nguyên.*

Hai định lý vừa nêu được sử dụng trong việc ứng dụng luồng cực đại vào việc giải quyết nhiều bài toán trong tổ hợp.

Tuy nhiên, nếu các khả năng thông qua là các số rất lớn thì giá trị của luồng cực đại cũng có thể là rất lớn và khi đó thuật toán mô tả ở trên sẽ đòi hỏi thực hiện rất nhiều bước tăng luồng. Thí dụ trong hình 2 sẽ minh hoạ cho điều này. Hình 2(a) mô tả mạng cần xét với các khả năng thông qua trên các cung. Hình 2(b) mô tả luồng trên các cung (số thứ hai bên cạnh cung) sau khi thực hiện tăng luồng dọc theo đường tăng luồng (s, a, b, t). Hình 2(c) mô tả luồng trên các cung sau khi thực hiện tăng luồng dọc theo đường tăng luồng (s, b, a, t). Rõ ràng, sau 2.10^6 lần tăng luồng theo đường (s, a, b, t) và (s, b, a, t) một cách luân phiên ta thu được luồng cực đại.



Hình 2. Ví dụ tồi tệ đối với thuật toán Ford - Fulkerson.

Hơn thế nữa, nếu các khả năng thông qua là các số vô tỷ, người ta còn xây dựng được ví dụ để cho thuật toán không dừng, và tệ hơn là nếu dãy các giá trị luồng xây dựng theo thuật toán hội tụ thì nó còn không hội tụ đến giá trị luồng cực đại. Như vậy,

muốn thuật toán làm việc hiệu quả, việc lựa chọn đường tăng luồng cần được tiến hành hết sức cẩn thận.

Edmonds và Karp chỉ ra rằng nếu đường tăng luồng được chọn là đường ngắn nhất từ s đến t trên đồ thị G_f. Điều đó có thể thực hiện, nếu trong thủ tục tìm đường tăng Find_Path mô tả ở trên, danh sách V_f được tổ chức dưới dạng QUEUE (nghĩa là ta thực hiện tìm đường tăng bởi thủ tục tìm kiếm theo chiều rộng) thì thuật toán sẽ kết thúc sau không quá $nm/2$ lần sử dụng đường tăng luồng. Nếu để ý rằng, tìm kiếm theo chiều rộng trên đồ thị đòi hỏi thời gian $O(n+m)$, thì thuật toán thu được sẽ có độ phức tạp tính toán là $O(nm^2)$.

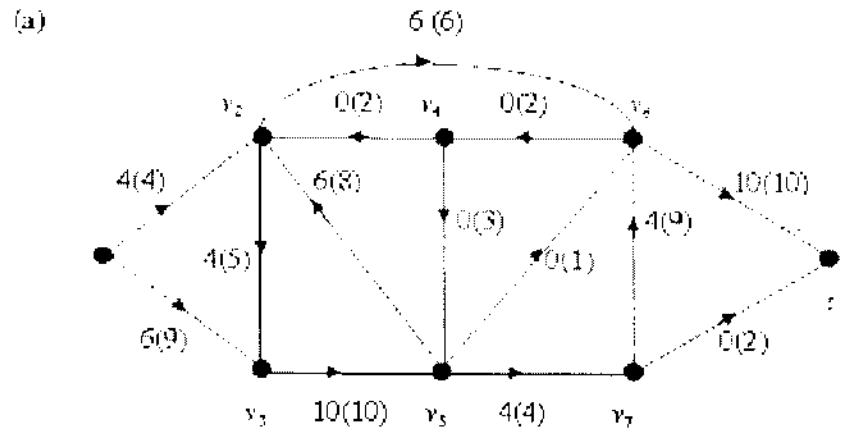
Nhờ cách tổ chức tìm đường tăng khéo léo hơn, người ta đã xây dựng được thuật toán với độ phức tạp tính toán tốt hơn như: $O(n^2m)$ (Dinic, 1970), $O(n^3)$ (Karzanov, 1974), $O(n^2m^{1/2})$ (Cherkasky, 1977), $O(nm \log n)$ (Sleator -Tarjan, 1980).

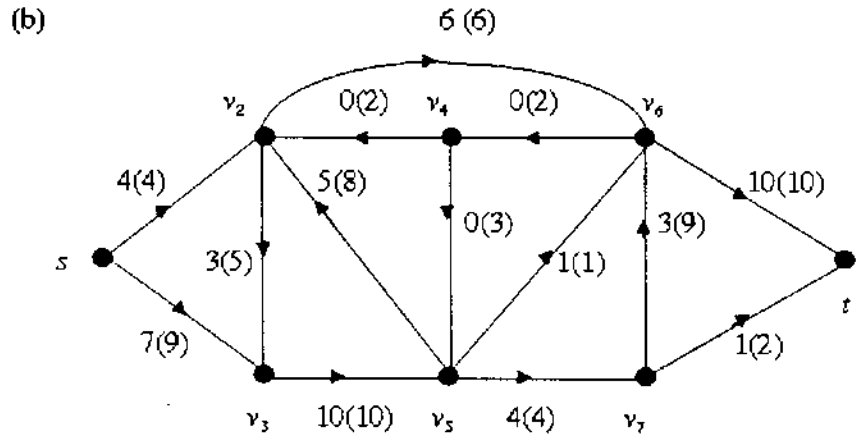
Ta kết thúc mục này bởi ví dụ minh hoạ cho thuật toán Ford - Fulkerson sau đây. Hình 3 (a) cho mạng G cùng với khả năng thông qua của các cung và luồng giá trị 10 trong nó. Hai số viết bên cạnh mỗi cung là khả năng thông qua của cung (số trong ngoặc) và luồng trên cung. Đường tăng luồng có dạng $(s, v_2, v_3, v_5, v_6, v_7, t)$. Ta tính được $\varepsilon(t) = 1$, giá trị luồng tăng từ 10 lên 11. Hình 3 (b) cho luồng thu được sau khi tăng luồng theo đường tăng tìm được.

Luồng trong hình 3 (b) đã là cực đại. Lát cắt hẹp nhất

$$X = \{s, v_2, v_3, v_5\}, \bar{X} = \{v_4, v_6, v_7, t\}.$$

Giá trị luồng cực đại là 11.





Hình 3. Tăng luồng dọc theo đường tăng

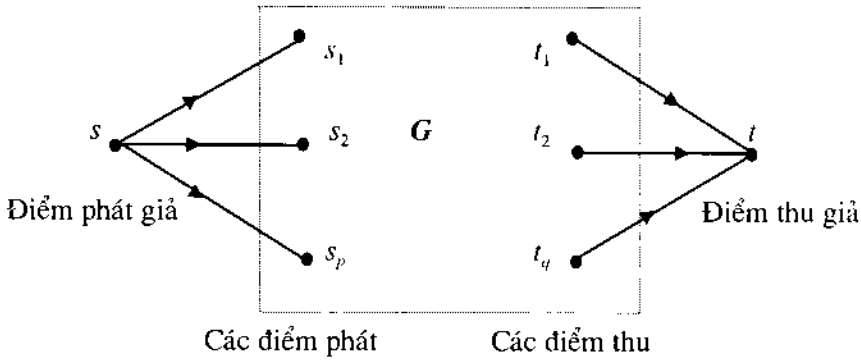
7.4. Một số bài toán luồng tổng quát

Trong phần này ta nêu ra một số dạng bài toán về luồng tổng quát mà việc giải chúng có thể dẫn về bài toán luồng cực đại trình bày ở trên.

a) Mạng với nhiều điểm phát và điểm thu.

Xét mạng G với p điểm phát $s_1, s_2,..., s_p$ và q điểm thu $t_1, t_2,..., t_q$. Giả sử rằng luồng có thể đi từ một điểm phát bất kỳ đến tất cả các điểm thu. Bài toán tìm luồng cực đại từ các điểm phát đến các điểm thu có thể đưa về bài toán với một điểm phát và một điểm thu bằng cách đưa vào một điểm phát giả s và một điểm thu giả t và các cạnh nối s với tất cả các điểm phát và các cạnh nối các điểm thu với t.

Hình 4 minh hoạ cho cách đưa mạng với nhiều điểm phát và nhiều điểm thu về mạng chỉ có một điểm phát và một điểm thu. Khả năng thông qua của cung nối s với điểm phát s_k sẽ bằng $+\infty$ nếu không có hạn chế về lượng phát của điểm phát s_k, và nếu lượng phát của s_k bị hạn chế bởi b_k thì cung (s, s_k) có khả năng thông qua là b_k. Cũng như vậy, đối với các cung nối t_k với điểm thu t, giá trị khả năng thông qua của (t_k, t) sẽ là giới hạn hoặc không giới hạn tuỳ theo lượng thu của điểm thu này có bị giới hạn hay không.



Hình 4. Mạng với nhiều điểm phát và thu

Trường hợp một số điểm thu chỉ nhận "hàng" từ một số điểm phát ta có bài toán nhiều luồng là một bài toán phức tạp hơn rất nhiều so với bài toán luồng cực đại giữa điểm phát s và điểm thu t.

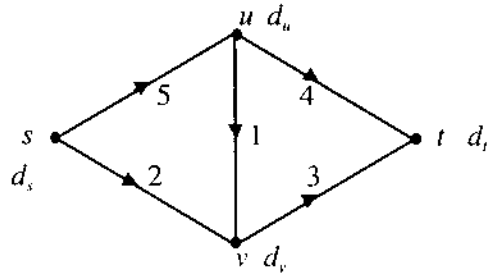
b) Bài toán với khả năng thông qua của các cung và các đỉnh.

Giả sử trong đồ thị G, ngoài khả năng thông qua của các cung $c(u, v)$, ở mỗi đỉnh $v \in V$ còn có khả năng thông qua của đỉnh là $d(v)$, và đòi hỏi tổng luồng đi vào đỉnh v không được vượt quá $d(v)$, tức là

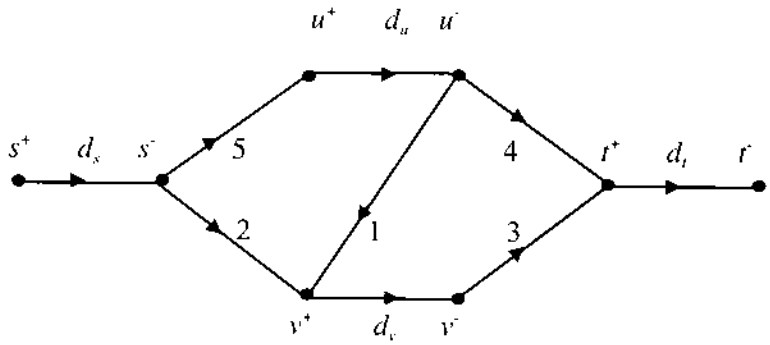
$$\sum_{w \in V} f(w,v) \leq d(v).$$

Cần phải tìm luồng cực đại giữa s và t trong mạng như vậy.

Xây dựng một mạng G' sao cho: mỗi đỉnh v của G tương ứng với 2 đỉnh v^+, v^- trong G', mỗi cung (u, v) trong G ứng với cung (u^-, v^+) trong G', mỗi cung (v,w) trong G ứng với cung (v^-, w^+) trong G'. Ngoài ra, mỗi cung (v^+, v^-) trong G' có khả năng thông qua là $d(v)$, tức là bằng khả năng thông qua của đỉnh v trong G.



Hình 5a. Mạng G với khả năng thông qua ở cung và đỉnh.



Hình 5.b. Mạng G' tương ứng chỉ có khả năng thông qua trên các cung.

Hình 5. Mạng với khả năng thông qua trên các đỉnh và các cung.

Do luồng đi vào đỉnh v^+ phải đi qua cung (v^+, v^-) với khả năng thông qua $d(v)$, nên luồng cực đại trong G' sẽ bằng luồng cực đại trong G với khả năng thông qua của các cung và các đỉnh.

c) Mạng trong đó khả năng thông qua của mỗi cung bị chặn hai phía.

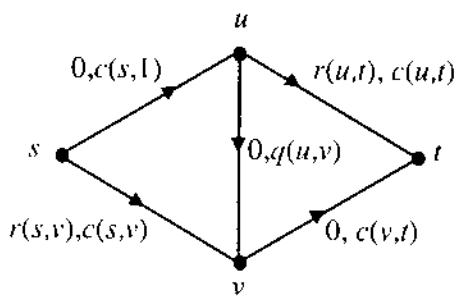
Xét mạng G mà trong đó mỗi cung (u, v) có khả năng thông qua (cận trên của luồng trên cung) $c(u, v)$ và cận dưới của luồng là $d(u, v)$. Bài toán đặt ra là liệu có tồn tại luồng tương thích từ s đến t, tức là luồng $\{f(u, v): u, v \in V\}$ thoả mãn thêm ràng buộc

$$d(u, v) \le f(u, v) \le c(u, v), \quad \forall (u, v) \in E,$$

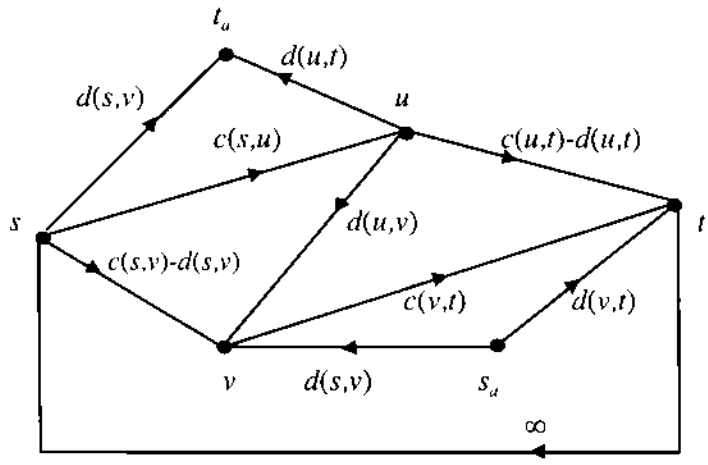
hay không?

Đưa vào mạng G đỉnh phát giả s_a và đỉnh thu giả t_a và xây dựng mạng G_a theo qui tắc: Mỗi cung (u, v) mà $d(u, v) \neq 0$ sẽ tương ứng với 2 cung (s_a, v) và (u, t_a) với khả năng thông qua là $d(u, v)$. Giảm $c(u, v)$ đi $d(u, v)$ tức là thay khả năng thông qua của cung (u, v) bởi $c(u, v) - d(u, v)$ còn cận dưới cuả nó đặt bằng 0. Ngoài ra thêm vào cung (t, s) với $c(t, s) = \infty$.

Hình 6a cho ví dụ mạng G với khả năng thông qua của các cung bị chặn cả hai phía. Đồ thị G_a tương ứng được cho trong Hình 6(b).



Hình 6a



Hình 6b

Hình 6. Mạng với khả năng thông qua bị chặn hai phía.

Ký hiệu

$$d^* = \sum_{d(u,v)\neq 0} d(u,v).$$

Ta có kết quả sau đây.

Định lý 4. *1) Nếu luồng lớn nhất trong mạng G_a từ s_a đến t_a bằng d^* thì tồn tại luồng tương thích trong G.*

2) Nếu luồng lớn nhất trong mạng G_a từ s_a đến t_a là khác d^* thì không tồn tại luồng tương thích trong G.

7.5. Một số ứng dụng trong tổ hợp

Bài toán luồng cực đại có rất nhiều ứng dụng trong việc giải nhiều bài toán tổ hợp. Khó khăn chính ở đây là phải xây dựng mạng tương ứng sao cho việc tìm luồng cực đại trong nó sẽ tương đương với việc giải bài toán tổ hợp đặt ra. Mục này sẽ giới thiệu một số bài toán như vậy.

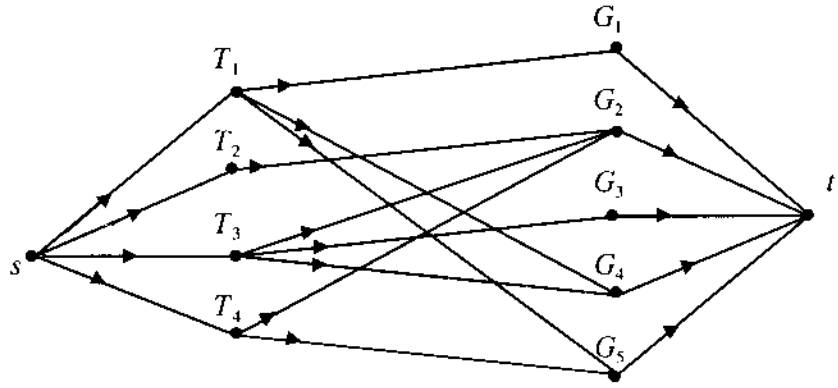
a) Bài toán đám cưới vùng quê. Có *m* chàng trai ở một làng quê nọ. Đối với mỗi chàng trai ta biết các cô gái mà anh ta vừa ý. Hỏi khi nào thì có thể tổ chức các đám cưới trong đó chàng trai nào cũng sánh duyên với cô gái mà mình vừa ý.

Ta có thể xây dựng đồ thị với các đỉnh biểu thị các chàng trai và các cô gái, còn các cung biểu thị sự vừa ý của các chàng trai đối với các cô gái. Khi đó ta thu được một đồ thị hai phía.

Thí dụ. Có 4 chàng trai $\{T_1, T_2, T_3, T_4\}$ và 5 cô gái $\{G_1, G_2, G_3, G_4, G_5\}$. Sự vừa ý cho trong bảng sau

Chàng trai	Các cô gái mà chàng trai ưng ý
T_1	G_1, G_4, G_5
T_2	G_2
T_3	G_2, G_3, G_4
T_4	G_2, G_4

Đồ thị tương ứng được cho trong hình 7.



Hình 7. Mạng tương ứng với Bài toán đám cưới vùng quê

Đưa vào điểm phát s và điểm thu t. Nối s với tất cả các đỉnh biểu thị các chàng trai, và nối t với tất cả các đỉnh biểu thị các cô gái. Tất cả các cung của đồ thị đều có khả năng thông qua bằng 1. Bắt đầu từ luồng 0, ta tìm luồng cực đại trong mạng xây dựng được theo thuật toán Ford - Fulkerson. Từ định lý về tính nguyên, luồng trên các cung là các số 0 hoặc 1. Rõ ràng là nếu luồng cực đại trong đồ thị có giá trị $V_{max} = m$, thì bài toán có lời giải, và các cung với luồng bằng 1 sẽ chỉ ra cách tổ chức đám cưới thoả mãn điều kiện đặt ra. Ngược lại, nếu bài toán có lời giải thì $V_{max} = m$. Bài toán về các đám cưới vùng quê là một trường hợp riêng của bài toán về cặp ghép trên đồ thị hai phía mà để giải nó có thể xây dựng thuật toán hiệu quả hơn.

b) Bài toán về hệ thống đại diện chung.

Cho tập m phần tử $X = \{z_1, z_2, ..., z_m\}$. Giả sử $<A_1, A_2, ..., A_n>$ và $<B_1, B_2, ..., B_n>$ là hai dãy các tập con của tập X. Dãy gồm n phần tử khác nhau của X: $<a_1, a_2, ..., a_n>$ được gọi là hệ

thống các đại diện chung của hai dãy đã cho nếu như tìm được một hoán vị σ của tập{ 1, 2,..., n} sao cho <a_1, a_2,..., a_n> là hệ thống các đại diện phân biệt của hai dãy <A_1, A_2,..., A_n> và <$B_{\sigma(1)}$, $B_{\sigma(2)}$,..., $B_{\sigma(n)}$>, tức là điều kiện sau được thoả mãn: $a_i \in A_i \cap B_{\sigma(i)}$, i = 1, 2,..., n. Xây dựng mạng $G = (V, E)$ với tập đỉnh

$$V = \{s, t\} \cup \{x_1, x_2,..., x_n\} \cup \{u_1, u_2,..., u_m\}$$
$$\cup \{v_1, v_2,..., v_m\} \cup \{y_1, y_2,..., y_n\},$$

trong đó đỉnh x_i tương ứng với tập A_i, đỉnh y_i tương ứng với tập B_i, các phần tử u_j, v_j tương ứng với phần tử z_j. Tập các cung của mạng G được xác định như sau

$$E = \{ (s,x_i): 1 \le i \le n\} \cup \{(x_i,u_j): \text{với } z_j \in A_i, 1 \le i \le n, 1 \le j \le m\} \cup$$
$$\cup \{(u_j, v_j): 1 \le j \le m\} \cup$$
$$\cup \{(v_j, y_i): \text{với } z_j \in B_i, 1 \le i \le n, 1 \le j \le m\} \cup \{(y_i, t): 1 \le i \le n\}.$$

Khả năng thông qua của tất cả các cung được đặt bằng 1. Dễ dàng thấy rằng *hệ thống đại diện chung của hai dãy <A_1, A_2,..., A_n> và <B_1, B_2,..., B_n> tồn tại khi và chỉ khi trong mạng $G = (V, E)$ tìm được luồng với giá trị n.* Để xét sự tồn tại của luồng như vậy có thể sử dụng thuật toán tìm luồng cực đại từ s đến t trong mạng $G = (V, E)$.

c) Về một bài toán tối ưu rời rạc.

Trong mục này ta sẽ trình bày thuật toán được xây dựng dựa trên thuật toán tìm luồng cực đại để giải một bài toán tối ưu rời rạc là mô hình toán học cho một số bài toán tối ưu tổ hợp.

Xét bài toán tối ưu rời rạc:

$$f(x_1, x_2,..., x_n) = \max_{1 \le i \le m} \sum_{j=1}^{n} x_{ij} \to \min \qquad (1)$$

với điều kiện

$$\sum_{j=1}^{n} a_{ij} x_{ij} = p_i, \quad i = 1,2,...,m, \qquad (2)$$

$$x_{ij} = 0 \text{ hoặc } 1, \ j = 1, 2,..., n \qquad (3)$$

trong đó $a_{ij} \in \{0,1\}$, i = 1, 2, ..., m; j = 1, 2, ..., n, p_i -nguyên dương, i = 1, 2, ..., m.

Bài toán (1)-(3) là mô hình toán học cho nhiều bài toán tối ưu tổ hợp thực tế. Dưới đây ta dẫn ra một vài ví dụ điển hình.

Bài toán phân nhóm sinh hoạt. Có m sinh viên và n nhóm sinh hoạt chuyên đề. Với mỗi sinh viên i, biết

a_{ij} = 1, nếu sinh viên i có nguyện vọng tham gia vào nhóm j,
a_{ij} = 0, nếu ngược lại,

và p_i là số lượng nhóm chuyên đề mà sinh viên i phải tham dự, $i = 1, 2, ..., m$; $j = 1, 2, ..., n$.

Trong số các cách phân các sinh viên vào các nhóm chuyên đề mà họ có nguyện vọng tham gia và đảm bảo mỗi sinh viên i phải tham gia đúng p_i nhóm, hãy tìm cách phân phối với số người trong nhóm có nhiều sinh viên tham gia nhất là nhỏ nhất có thể được.

Đưa vào biến số

$x_{ij} = 1$, nếu sinh viên i tham gia vào nhóm j,

$x_{ij} = 0$, nếu ngược lại,

$i = 1, 2, ..., m$, $j = 1, 2, ..., n$, khi đó dễ thấy mô hình toán học cho bài toán đặt ra chính là bài toán (1) - (3).

Bài toán lập lịch cho hội nghị. Một hội nghị có m tiểu ban, mỗi tiểu ban cần sinh hoạt trong một ngày tại phòng họp phù hợp với nó. Có n phòng họp dành cho việc sinh hoạt của các tiểu ban. Biết

$a_{ij} = 1$, nếu phòng họp i là thích hợp với tiểu ban j,

$a_{ij} = 0$, nếu ngược lại,

$i = 1, 2, ..., m$; $j = 1, 2, ..., n$. Hãy bố trí các phòng họp cho các tiểu ban sao cho hội nghị kết thúc sau ít ngày làm việc nhất.

Đưa vào biến số

$x_{ij} = 1$, nếu bố trí tiểu ban i làm việc ở phòng j,

$x_{ij} = 0$, nếu ngược lại,

$i = 1, 2, ..., m$, $j = 1, 2, ..., n$, khi đó dễ thấy mô hình toán học cho bài toán đặt ra chính là bài toán (1) - (3), trong đó $p_i = 1$, $i = 1, 2, ..., m$.

Bổ đề 2. *Bài toán (1)-(3) có phương án tối ưu khi và chỉ khi*

$$\sum_{j=1}^{n} a_{ij} \geq p_i, \quad i = 1,2,...,m, \tag{4}$$

Chứng minh. Điều kiện cần của bổ đề là hiển nhiên vì từ sự tồn tại phương án của bài toán suy ra các bất đẳng thức trong (4) được thực hiện ít nhất dưới dạng dấu đẳng thức. Để chứng minh điều kiện đủ, chỉ cần chỉ ra rằng nếu điều kiện (4) được thực hiện thì bài toán luôn có phương án. Thực vậy, giả sử điều kiện (4) được thực hiện. Khi đó nếu ký hiệu

$$I_{+i} = \{ 1 \leq j \leq n : a_{ij} = 1 \},$$

thì $|I_{+i}| \geq p_i$, $i = 1, 2, ..., m$. Do đó nếu gọi

$I_i \subset I_{+i}$, $|I_i| = p_i$, $i = 1, 2, ..., m$,

thì $X^* = (x^*_{ij})_{m \times n}$ với các thành phần được xác định theo công thức

$$x^*_{ij} = 1, \ j \in I_i, \ x^*_{ij} = 0, \ j \notin I_i, \ i = 1, 2, ..., m, \tag{5}$$

là phương án của bài toán (1) - (3). Bổ đề được chứng minh.

Do (4) là điều kiện cần để bài toán (1)-(3) có phương án, nên trong trong phần tiếp theo ta sẽ luôn giả thiết rằng điều kiện này được thực hiện.

Bây giờ ta sẽ chỉ ra rằng việc giải bài toán (1)-(3) có thể dẫn về việc giải một số hữu hạn bài toán luồng cực đại trong mạng. Trước hết, với mỗi số nguyên dương k, xây dựng mạng $G(k) = (V,E)$ với tập đỉnh

$$V = \{s\} \cup \{ u_i : i = 1, 2, ..., m \} \cup \{ w_j : j = 1, 2,..., n \} \cup \{t\},$$

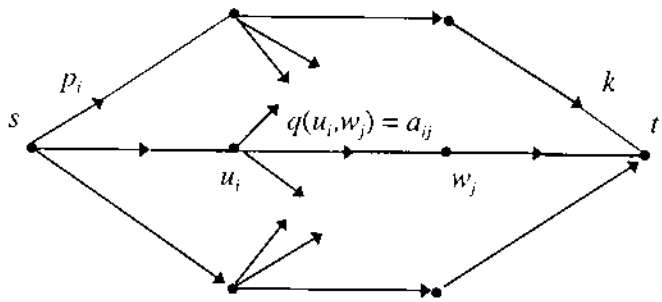
trong đó s là điểm phát, t là điểm thu, và tập cung

$$E = \{(s,u_i) : i=1, 2,..., m\} \cup \{(u_i,w_j) : i=1, 2,..., m; j = 1, 2,..., n \}$$
$$\cup \{ (w_j,t) : j =1, 2,..., n \}.$$

Mỗi cung $e \in E$ được gán với khả năng thông qua $q(e)$ theo qui tắc sau:

$q(s,u_i) = p_i$, $i= 1, 2,..., m$,
$q(u_i,w_j) = a_{ij}$, $i = 1, 2,..., m; j = 1, 2,..., n$;
$q(w_j, t) = k, j = 1, 2,..., n$.

Hình 8 chỉ ra cách xây dựng mạng $G(k)$.



Hình 8. Mạng $G(k)$.

Ký hiệu: $\sigma = \sum_{i=1}^{m} p_i$.

Bổ đề sau đây cho thấy mối liên hệ giữa luồng cực đại trong mạng $G(k)$ và phương án của bài toán (1)-(3).

Bổ đề 3. *Giả sử đối với số nguyên dương k nào đó, luồng cực đại nguyên ξ^* trong mạng G(k) có giá trị là σ. Khi đó $X^* = (x^*_{ij})_{m\times n}$ với các thành phần được xác định theo công thức*

$$x^*_{ij} = \xi^*(u_i,w_j),\ i = 1,2,...,m;\ j=1,2,...,n,$$

là phương án của bài toán (1)-(3).

Chứng minh. Thực vậy, do luồng cực đại trong mạng có giá trị là σ và là luồng nguyên nên

$$\xi^*(s,u_i) = p_i,\ i = 1, 2,..., m,$$

$$\xi^*(u_i,w_j) \in \{0, 1\},\ i = 1, 2,..., m;\ j = 1, 2,..., n,$$

từ đó suy ra

$$\sum_{j=1}^{n} x^*_{ij} = \sum_{j=1}^{n} \xi^*(u_i,v_j) = \xi^*(s,u_i) = p_i,\ i = 1,2,...,m.$$

Vậy X^* là phương án của bài toán (1)-(3). Bổ đề được chứng minh.

Bổ đề 4. Giả sử $X^* = (x^*_{ij})$ *là phương án tối ưu và* k^* *là giá trị tối ưu của bài toán* (1)-(3). *Khi đó luồng cực đại trong mạng* $G(k^*)$ *có giá trị là* σ.

Chứng minh. Do giá trị của luồng cực đại trong mạng $G(k^*)$ không vượt quá σ, nên để chứng minh bổ đề ta chỉ cần chỉ ra luồng với giá trị σ trong mạng $G(k^*)$. Xây dựng luồng ξ^* theo công thức sau:

$$\xi^*(s,u_i) = p_i,\quad \xi^*(u_i,w_j) = x^*_{ij},$$

$$\xi^*(w_j,t) = \sum_{i=1}^{m} x^*_{ij},\ i = 1,2,...,m,\ j = 1,2,...,n.$$

Dễ dàng kiểm tra được rằng ξ^* là luồng trong mạng $G(m)$ có giá trị σ. Bổ đề được chứng minh.

Bổ đề 5. *Nếu k = m thì luồng cực đại trong mạng G(m) có giá trị là* σ.

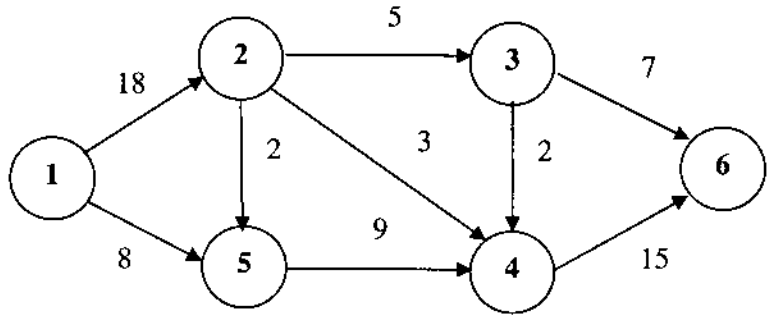
Chứng minh. Lập luận tương tự như trong Bổ đề 4, ta chỉ cần chỉ ra luồng với giá trị σ trong mạng $G(m)$. Thực vậy, giả sử $X^* = (x^*_{ij})_{m\times n}$ là phương án của bài toán (1)-(3) xây dựng theo công thức (5). Xây dựng luồng ξ^* theo công thức giống như trong chứng minh bổ đề 4, ta có luồng với giá trị σ. Bổ đề được chứng minh.

Từ bổ đề 3 và 4 suy ra việc giải bài toán (1) - (3) dẫn về việc tìm giá trị k^* nguyên dương nhỏ nhất sao cho luồng cực đại trong mạng $G(k^*)$ có giá trị σ. Bổ đề 5 cho thấy giá trị $k^* \in [1, m]$. Vì vậy để giải bài toán (1) - (3) ta có thể áp dụng phương pháp tìm kiếm nhị phân trên đoạn $[1, m]$ để tìm giá trị k^*, trong đó ở mỗi bước cần giải một bài toán luồng cực đại. Để giải bài toán tìm luồng cực đại trong mạng có thể sử dụng các thuật toán đa thức như đã nói ở trên. Từ đó suy ra kết quả sau

Định lý 5. *Bài toán* (1)-(3) *giải được nhờ thuật toán đa thức với thời gian tính là* $\log_2 m \times O_{NF}$, *trong đó* O_{NF} *là thời gian tính của thuật toán giải bài toán tìm luồng cực đại trong mạng* $G(k)$.

Bài tập

1. Xét mạng cho trong hình vẽ sau, các số bên cạnh các cung là khả năng thông qua của nó



Thực hiện thuật toán Ford-Fulkerson tìm luồng cực đại trên mạng đã cho bắt đầu từ luồng 0. Trình bày các kết quả tính toán trong mỗi bước lặp bao gồm:

- Đồ thị tăng luồng,
- Đường tăng luồng tìm được theo tìm kiếm theo chiều rộng và khả năng thông qua của nó (Giả thiết khi duyệt các đỉnh kề của một đỉnh ta duyệt theo thứ tự tăng dần của chỉ số),
- Mạng cùng luồng thu được sau khi tăng luồng.

Kết quả cuối cùng: Cần đưa ra giá trị của luồng cực đại và lát cắt hẹp nhất

2. Cho $G = (V, E)$ đồ thị có hướng trong đó không có cung (s, t). Chứng minh rằng số đường đi cơ bản nối hai đỉnh s và t là bằng số ít nhất các đỉnh của đồ thị cần loại bỏ để trong đồ thị không còn đường đi nối s với t.

3. Xây dựng thuật toán tìm tập E_1 tất cả các cung của đồ thị mà việc tăng khả năng thông qua của bất kỳ cung nào trong E đều dẫn đến tăng giá trị của luồng cực đại trong mạng.

4. Cho hai dãy số nguyên dương $\{p_i, i=1, 2,..., m\}$ và $\{q_j, j=1, 2,..., n\}$. Hãy xây dựng ma trận $A = \{a_{ij} : i=1, 2,..., m; j=1, 2,..., n\}$ với các phần tử $a_{ij} \in \{0, 1\}$ có tổng các phần tử trên dòng i là p_i, tổng các phần tử trên cột j là q_j.

5. Có m chàng trai, n cô gái và k bà mối. Mỗi bà mối p ($p = 1, 2,..., k$) có một danh sách L_p một số chàng trai và cô gái trong số các chàng trai và cô gái nói trên là khách hàng của bà ta. Bà mối p có thể se duyên cho bất cứ cặp trai-gái nào là khách hàng của bà ta, nhưng không đủ sức tổ chức quá d_p đám cưới. Hãy xây dựng thuật toán căn cứ vào danh sách L_p, b_p, $p = 1, 2,..., k$, đưa ra cách tổ chức nhiều nhất các đám cưới giữa m chàng trai và n cô gái với sự giúp đỡ của các bà mối.

6. Bài toán phủ đỉnh nhỏ nhất: Cho đồ thị vô hướng $G = (V, E)$ với trọng số trên các đỉnh $c(v)$, $v \in V$. Một tập con các đỉnh của đồ thị S được gọi là phủ đỉnh nếu như mỗi cạnh của đồ thị có ít nhất một đầu mút trong S. Trọng lượng của phủ đỉnh S là tổng các trọng số của các đỉnh trong nó. Bài toán đặt ra là tìm phủ đỉnh có trọng lượng nhỏ nhất (mà ta sẽ gọi là phủ đỉnh nhỏ nhất). Hãy chỉ ra cách qui dẫn bài toán tìm phủ đỉnh nhỏ nhất trên đồ thị hai phía về bài toán luồng cực đại.

7. Cho đơn đồ thị vô hướng $G = (V, E)$. Giả sử s, t là hai đỉnh của đồ thị. Sử dụng lý thuyết luồng cực đại trong mạng chứng minh khẳng định sau: "*Số lượng lớn nhất các đường đi đôi một không có cạnh chung nối hai đỉnh s và t là bằng số cạnh ít nhất cần loại bỏ khỏi đồ thị G để không còn đường đi nối s và t*".

8. Bài toán ghép cặp trên đồ thị hai phía: Cho đồ thị hai phía $G = (X \cup Y, E)$. Một tập con các cạnh của đồ thị được gọi là một cặp ghép nếu như hai cạnh bất kỳ trong nó không có đỉnh chung. Bài toán đặt ra là tìm cặp ghép có lực lượng lớn nhất của G. Hãy chỉ ra cách qui dẫn bài toán đặt ra về bài toán luồng cực đại trong mạng.

9. Xây dựng hình chữ nhật latinh. Hãy chỉ ra cách xây dựng hình chữ nhật la tinh cấp $m \times n$ ($m \le n$) nhờ sử dụng thuật toán tìm cặp ghép lớn nhất trên đồ thị hai phía.

PHẦN III

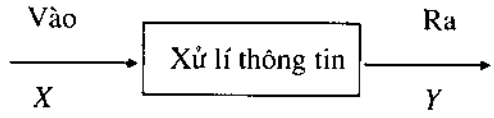
HÀM ĐẠI SỐ LÔGIC

1

MỞ ĐẦU

1.1. Mô hình xử lí thông tin và hàm đại số lôgic

Trong nhiều vấn đề, ta thường gặp một mô hình xử lý thông tin dạng đơn giản, gồm một đầu vào và một đầu ra:



Đầu vào và đầu ra lấy giá trị từ những tập hữu hạn X và Y. Ta xem mô hình hoạt động trên một trục thời gian được phân chia thành những thời điểm rời rạc. Tại thời điểm t, tín hiệu vào là $x_t \in X$ và tín hiệu ra là $y_t \in Y$. Giả thiết rằng mô hình hoạt động không nhớ, tức là tín hiệu ra tại thời điểm t chỉ phụ thuộc vào tín hiệu vào tại thời điểm đó. Như vậy, có thể mô tả sự hoạt động của mô hình bằng một hàm $f: X \to Y$, trong đó $y = f(x)$ là tín hiệu ra tương ứng với tín hiệu vào x.

Trong kỹ thuật điều khiển tự động, các tín hiệu thường được truyền đi và chế biến dưới dạng chỉ có 2 giá trị, mà ta sẽ ký hiệu là 0 và 1. Vì vậy có thể xem X như tập hợp E^n và Y như tập hợp E^m, trong đó $E = \{0, 1\}$ (với n và m được chọn thích hợp). Mỗi

phần tử $x \in X$ có dạng $x = (x_1, x_2, ..., x_n)$ và mỗi phần tử $y \in Y$ có dạng $y = (y_1, y_2, ..., y_m)$ trong đó $x_i, y_i \in E$.

Một hàm $f: X \to Y$ được phân tích thành một hệ m hàm

$$y_1 = f_1(x_1, x_2, ..., x_n)$$
$$y_2 = f_2(x_1, x_2, ..., x_n)$$
$$\cdots\cdots\cdots\cdots$$
$$y_m = f_m(x_1, x_2, ..., x_n)$$

Mỗi hàm $y_i = f_i(x_1, x_2, ..., x_n)$ có các đối số nhận giá trị 0, 1 và hàm cũng nhận các giá trị 0,1. Một hàm như thế được gọi là một hàm đại số lôgic.

Định nghĩa. *Một hàm đại số lôgic của n đối số là một ánh xạ* $E^n \to E$.

Một hàm đại số lôgic thường được xác định bằng bảng giá trị của nó, nghĩa là trong đó liệt kê tất cả các ảnh của các bộ đối số có thể có. Thí dụ bảng dưới đây xác định một hàm đại số lôgic 3 đối số:

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Hình 1

Vì $|E^n| = 2^n$ và $|E| = 2$ nên số các hàm đại số lôgic của n đối số là 2^{2^n}, chẳng hạn với $n = 3$, con số này là $2^8 = 256$.

Một hàm đại số lôgic $y = f(x_1, ..., x_n)$ được gọi là phụ thuộc thực sự vào đối số x_i nếu có 2 bộ giá trị của đối số chỉ khác nhau một thành phần thứ i mà giá trị của hàm tại 2 bộ này là khác nhau.

Không phải mọi hàm đại số lôgic của n đối số đều phụ thuộc thực sự vào cả n đối số của nó. Chẳng hạn các hàm hằng $y = 0$ hay $y = 1$ không phụ thuộc thực sự vào đối số nào cả. Gọi A_n là số tất cả các hàm đại số lôgic phụ thuộc thực sự vào n đối số. Khi đó dễ dàng tính được A_n theo hệ thức truy hồi dưới đây:

$A_0 = 2$,

$A_n = 2^{2^n} - C_n^{n-1} A_{n-1} - ... - C_n^1 A_1 - A_0$.

Thông thường, việc mã hoá nhị phân tập X bị dư thừa, nghĩa là có một số phần tử của X tại đó hàm f không xác định (chẳng hạn nếu X có 6 phần tử thì khi mã hoá nó bằng tập E^3, ta thừa 2 phần tử). Một hàm $f : M \to E$, với M là một tập con thực sự của E^n được gọi là một hàm đại số lôgic không đầy đủ và tập con M được gọi là miền xác định của nó. Rõ ràng tại những nơi mà f không xác định, ta có thể cho nó một giá trị (0 hoặc 1) tuỳ ý để được một hàm đại số lôgic đầy đủ. Nói cách khác, với mọi hàm đại số lôgic không đầy đủ f có miền xác định M, luôn xây dựng được một hàm đại số lôgic đầy đủ F để $F(x_1,...,x_n) = f(x_1,...,x_n)$ với mọi $(x_1,...,x_n) \in M$. Hàm F như vậy được gọi là hàm phủ của hàm f. Dễ thấy rằng, số lượng các hàm phủ f bằng 2^k, trong đó $k = 2^n - |M|$.

1.2. Các hàm đại số lôgic sơ cấp

Dưới đây ta sẽ xét cụ thể các hàm đại số lôgic phụ thuộc thực sự vào n đối số với $n \le 2$. Các hàm này được gọi là các hàm sơ cấp.

a) $n = 0$: có 2 hàm là hàm hằng 0 và hàm hằng 1.

b) $n = 1$: có 2 hàm cho bởi bảng

x	f_1	f_2
0	0	1
1	1	0

Hàm $f_2(x)$ được gọi là phủ định của x và được ký hiệu là $\neg x$ hay $\bar{x}$

c) $n = 2$: có 10 hàm cho bởi bảng

x	y	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
0	0	0	0	0	1	1	1	1	1	0	0
0	1	0	1	1	1	0	0	1	0	1	0
1	0	0	1	1	0	0	0	1	1	0	1
1	1	1	1	0	1	1	0	0	1	0	0

trong đó có một số hàm thông dụng như sau:

- hàm $g_1(x, y)$ được gọi là hội (hay tích) của x và y, ký hiệu là $x \,\&\, y$ hay $x.y$, xy
- hàm $g_2(x, y)$ được gọi là tuyển của x và y, ký hiệu $x \vee y$
- hàm $g_3(x, y)$ được gọi là tổng (theo modun 2) của x và y, ký hiệu là $x + y$
- hàm $g_4(x, y)$ được gọi là hàm kéo theo, ký hiệu $x \to y$

- hàm $g_5(x, y)$ được gọi là hàm tương đương, ký hiệu $x \Leftrightarrow y$
- hàm $g_6(x, y)$ được gọi là hàmVebb, ký hiệu $x \circ y$
- hàm $g_7(x, y)$ được gọi là hàm Sheffer, ký hiệu $x \mid y$

Quan trọng nhất là các hàm tuyển, hội và phủ định. Chúng tương ứng với các phép toán "hoặc", "và", "không" trong đại số mệnh đề, trong đó giá trị 1 ứng với "đúng" và giá trị 0 ứng với "sai". Dễ thấy rằng các tính chất dưới đây của các hàm tuyển, hội phủ định được suy từ các tính chất quen thuộc của đại số mệnh đề (hoặc cũng có thể suy từ bảng giá trị):

- *Kết hợp*

$$x \vee (y \vee z) = (x \vee y) \vee z = x \vee y \vee z,$$
$$x \,\&\, (y \,\&\, z) = (x \,\&\, y) \,\&\, z = x \,\&\, y \,\&\, z.$$

- *Giao hoán*

$$x \vee y = y \vee x,$$
$$x \,\&\, y = y \,\&\, x.$$

- *Phân bố*

$$x \vee (y \,\&\, z) = (x \vee y) \,\&\, (x \vee z),$$
$$x \,\&\, (y \vee z) = (x \,\&\, y) \vee (x \,\&\, z).$$

- *Đối ngẫu*

$$\overline{x \vee y} = \bar{x} \,\&\, \bar{y},$$
$$\overline{x \,\&\, y} = \bar{x} \vee \bar{y}.$$

Trong kỹ thuật, các hàm tuyển, hội, phủ định được thực hiện nhờ những linh kiện đơn giản. Mục trình bày dưới đây khẳng định rằng, mọi hàm đại số lôgic đều có thể được xây dựng từ những linh kiện đơn giản này.

1.3. Biểu diễn các hàm đại số lôgic qua hệ tuyển, hội và phủ định

Để tiện trình bày, ta đưa vào quy ước sau đây: giả sử x là một biến và $\sigma \in \{0, 1\}$. Khi đó

$$x^{\sigma} = \begin{cases} x, & \text{nếu } \sigma = 1, \\ \bar{x}, & \text{nếu } \sigma = 0. \end{cases}$$

Từ định nghĩa trên dễ thấy

$$x^{\sigma} = 1 \quad \Leftrightarrow \quad x = \sigma.$$

Giả sử $\{f_i : i \in I\}$ là một họ các hàm đại số lôgic. Ta sẽ dùng ký hiệu

$$\bigvee_{i \in I} f_i \qquad \text{và} \qquad \&_{i \in I} f_i$$

để chỉ tuyển và hội của tất cả các hàm này, nói riêng nếu $I = \{1, 2, ..., m\}$, thì ta sẽ dùng các ký hiệu

$$\bigvee_{i=1}^{m} f_i \qquad \text{và} \qquad \&_{i=1}^{m} f_i$$

để thay thế. Ta cũng quy ước rằng nếu I là tập rỗng thì

$$\bigvee_{i \in I} f_i = 0 \qquad \text{và} \qquad \&_{i \in I} f_i = 1.$$

Với mỗi hàm $f(x_1, ..., x_n)$, ký hiệu T_f là tập

$$T_f = \{(x_1, ..., x_n) \in E^n : f(x_1, ..., x_n) = 1\}$$

và gọi nó là tập đặc trưng của f. Dễ dàng chứng minh tương ứng giữa hàm đại số lôgic với tập đặc trưng của nó là tương ứng 1-1 và có các tính chất:

$$T_{\bar{f}} = \bar{T}_f, \quad T_{f \vee g} = T_f \cup T_g, \quad T_{f \& g} = T_f \cap T_g.$$

Điều này cho phép chuyển các chứng minh trên đại số lôgic sang các chứng minh tương ứng trên đại số tập hợp.

Định lý. *Mọi hàm đại số lôgic $f(x_1, ..., x_n)$ đều có thể được biểu diễn dưới dạng:*

$$f(x_1, ..., x_n) = \bigvee_{(\sigma_1, ..., \sigma_i) \in E^i} x_1^{\sigma_1} x_2^{\sigma_2} ... x_i^{\sigma_i} f(\sigma_1, \sigma_2, ..., \sigma_i, x_{i+1}, ..., x_n)$$

trong đó i là số tự nhiên bất kỳ, $1 \le i \le n$.

Chứng minh. Giả sử $(x_1, ..., x_n) \in T_f$, khi đó số hạng ứng với bộ giá trị $\sigma_1 = x_1, ..., \sigma_i = x_i$ trong tuyển vế phải

$$x_1^{\sigma_1} \cdot \cdot \cdot x_i^{\sigma_i} f(\sigma_1, ..., \sigma_i, x_{i+1}, ..., x_n)$$

sẽ bằng 1, điều này kéo theo toàn bộ vế phải sẽ bằng 1. Ngược lại, nếu vế phải bằng 1, thì phải xảy ra bằng 1 tại một số hạng nào đó, chẳng hạn tại số hạng ứng với bộ giá trị $(\sigma_1, ..., \sigma_i)$. Khi đó $x_1 = \sigma_1, ..., x_i = \sigma_i$ và $(x_1, ..., x_n) \in T_f$. Định lý được chứng minh.

Cho $i = 1$ trong định lý và nhận xét rằng vai trò của các biến x_i là như nhau, ta được:

Hệ quả 1. *Hàm đại số lôgic $f(x_1,...,x_n)$ có thể được khai triển theo một đối số x_i:*

$$f(x_1,...,x_n) = \bar{x}_i f(x_1,..., x_{i-1}, 0, x_{i+1},..., x_n) \vee x_i f(x_1,..., x_{i-1}, 1, x_{i+1},..., x_n)$$

Cho $i = n$ trong định lý và bỏ đi các nhân tử bằng 1 trong một tích, ta được:

Hệ quả 2. *Hàm đại số lôgic $f(x_1,...,x_n)$ có thể được khai triển dưới dạng:*

$$f(x_1,...,x_n) = \bigvee_{(\sigma_1,...,\sigma_n)\in T_f} x_1^{\sigma_1} \ldots x_n^{\sigma_n}$$

Công thức khai triển này còn được gọi là dạng tuyển chuẩn tắc hoàn toàn của f và mỗi số hạng của nó được gọi là một cấu tạo đơn vị của f.

Thí dụ. Dạng tuyển chuẩn tắc hoàn toàn của hàm f cho bởi bảng ở hình 1 là

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 \bar{x}_3 \vee \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 x_3.$$

Từ hệ quả 2, ta nhận được khẳng định sau đây: *Mọi hàm đại số lôgic đều có thể xây dựng từ các biến nhờ các hàm tuyển, hội và phủ định.*

Bằng luật đối ngẫu, ta có thể chứng minh một kết quả tương tự bằng cách thay phép tuyển bằng phép hội và ngược lại, từ đó dẫn đến việc biểu diễn f qua một hội các tuyển. Biểu diễn này được gọi là dạng hội chuẩn tắc hoàn toàn của f. Chẳng hạn, trong thí dụ vừa nêu, dạng hội chuẩn tắc hoàn toàn của f là

$$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \,\&\, (x_1 \vee x_2 \vee \bar{x}_3) \,\&\, (\bar{x}_1 \vee \bar{x}_2 \vee x_3).$$

Ngoài hệ tuyển, hội và phủ định, tồn tại nhiều hệ khác cũng có tính chất mọi hàm đại số lôgic đều được biểu diễn qua các thành viên của hệ. Một hệ hàm như vậy được gọi là một hệ đầy đủ. Chẳng hạn có thể chứng minh các hệ

$$\{0, 1, x + y, x \,\&\, y\},\ \{x, x \vee y\},\ \{x, x \,\&\, y\},\ \{x \mid y\},\ \{x \circ y\}$$

đều là những hệ hàm đầy đủ.

Việc nghiên cứu tính đầy đủ của một hệ hàm có một ý nghĩa thực tiễn quan trọng, nó trả lời câu hỏi có thể xây dựng mọi hàm lôgic từ một số hàm đơn giản chọn trước hay không? Về vấn đề này, Post đã thực hiện một cách hệ thống từ năm 1921, bạn đọc có thể tìm hiểu một cách chi tiết qua cuốn sách tiếng Nga "Hàm đại số logic và các lớp Post" xuất bản năm 1966 của các tác giả S.B.Iablonski, G.P. Gavrilov, V.B.Kudriavcev.

1.4. Biểu diễn tối thiểu của hàm đại số lôgic

Biểu diễn một hàm đại số lôgic f qua một hệ hàm đầy đủ H là không duy nhất. Thí dụ hàm Sheffer, khi biểu diễn qua hệ tuyển, hội và phủ định, có thể có các cách

$$x \mid y = \bar{x}\,\bar{y} \vee \bar{x}\,y \vee x\,\bar{y} = \bar{x} \vee \bar{y}.$$

Mỗi một biểu diễn f tương ứng với một cách "ghép" các thành viên của H (mà ta gọi là các yếu tố cơ bản) để thu được f. Hiển nhiên, một vấn đề có ý nghĩa thực tế quan trọng là, cần tìm một biểu diễn sao cho việc ghép như thế là tốn ít yếu tố cơ bản nhất. Theo một nghĩa nào đó, điều này dẫn về việc tìm một công thức trên hệ H biểu diễn hàm f với số ký hiệu các yếu tố này là ít nhất. Một công thức như vậy, được gọi là một biểu diễn tối thiểu của hàm f trong hệ H.

Về nguyên tắc, số công thức biểu diễn f là hữu hạn, nên bằng cách duyệt tất cả các khả năng, ta luôn tìm được biểu diễn tối thiểu của f. Tuy nhiên, số khả năng này là rất lớn và việc duyệt nó đòi hỏi một khối lượng tính toán khổng lồ, do đó trên thực tế khó mà thực hiện được dù rằng ngay cả với những siêu máy tính.

Việc xây dựng những thuật toán hữu hiệu tìm biểu diễn tối thiểu của các hàm đại số lôgic, vì thế càng trở nên cấp bách nhưng đồng thời nó cũng là bài toán rất khó. Cho đến nay, bài toán này vẫn chưa được giải quyết thoả đáng ngay cả trong một số trường hợp đơn giản và còn đang được tiếp tục nghiên cứu.

Một hệ đầy đủ được nghiên cứu nhiều nhất là hệ tuyển, hội và phủ định. Bài toán tìm biểu diễn tối thiểu của các hàm đại số lôgic trong hệ này đã được nghiên cứu nhiều trong vài chục năm gần đây. Tuy nhiên, các kết quả đạt được thường mới chỉ đề cập đến một dạng biểu diễn riêng biệt trong hệ, đó là dạng tuyển chuẩn tắc của các hàm đại số lôgic mà ta sẽ xét kỹ trong chương sau.

Bài tập

1. Xét các hàm đại số lôgic 3 đối số $f(x, y, z)$. Cho tập đặc trưng T_f. Xác định dạng tuyển chuẩn tắc hoàn toàn của f:

a) $T_f = \{(0, 1, 1), (1, 0, 0), (1, 1, 0)\}$

b) $T_f = \{(0, 0, 1), (0, 1, 1), (1. 0, 0), (1, 0, 1), (1, 1, 1)\}$

c) $T_f = \{(0, 0, 0), (1, 1, 1), (0, 1, 1), (1, 0, 0)\}$

2. Có bao nhiêu hàm đại số lôgic 3 đối số mà tập đặc trưng T_f của nó có 5 phần tử? Khái quát hoá: có bao nhiêu hàm đại số lôgic n đối số mà tập đặc trưng T_f của nó có k ($k \le 2^n$) phần tử?

3. Tìm dạng tuyển chuẩn tắc hoàn toàn của các hàm dưới đây:

a) $f(x, y) = x + y$ (hàm cộng modun 2)

b) $f(x, y) = x \circ y$ (hàm Vebb)

c) $f(x, y) = x \mid y$ (hàm Sheffer)

d) $f(x, y, z) = (x \vee y) \,\&\, \bar{z}$

e) $f(x, y, z) = x \vee y \vee z$

f) $f(x, y, z) = (x \,\&\, y) \vee (x \,\&\, z) \vee (y \,\&\, z)$

g) $f(x, y, z) = (x \mid y) \circ z$

4. Hàm đại số lôgic $f(x, y, z)$ nhận giá trị 0 tại (0, 0, 0). Nếu thay đổi giá trị bất cứ đối số nào của nó thì giá trị của hàm cũng thay đổi (so với giá trị trước đấy). Tìm dạng tuyển chuẩn tắc hoàn toàn của f.

5. Xét xem các hàm f, g dưới đây khác nhau hay bằng nhau:

a) $f(x, y, z) = (x \,\&\, y) \vee \bar{z}$

$g(x, y, z) = (\bar{x} \vee y \vee \bar{z}) \,\&\, (x \vee \bar{y} \vee \bar{z}) \,\&\, (x \vee y \vee \bar{z})$

b) $f(x, y, z) = (x \mid y) \circ z$

$g(x, y, z) = x \,\&\, (y + z)$

6. Chứng minh các hệ hàm sau đây là hệ đầy đủ:

$$\{0, 1, x + y, x \,\&\, y\},\ \{\bar{x}, x \vee y\},\ \{\bar{x}, x \,\&\, y\},\ \{x \mid y\},\ \{x \circ y\}$$

7. Biểu diễn hàm $f(x, y) = x \vee \bar{y}$ qua các hệ hàm:

a) $\{\bar{x}, x \,\&\, y\}$

b) $\{x \mid y\}$

c) $\{x \circ y\}$

2

DẠNG TUYỂN CHUẨN TẮC CỦA HÀM ĐẠI SỐ LÔGIC

2.1. Các khái niệm cơ bản

Giả sử cho n biến $x_1, x_2, ..., x_n$. Một biểu thức dạng

$$x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} ... x_{i_j}^{\sigma_j}$$

trong đó $\sigma_1,..., \sigma_j \in \{0, 1\}$, $1 \le i_1, ..., i_j \le n$ và $i_t \ne i_s$ nếu $t \ne s$ được gọi là một hội sơ cấp của n biến $x_1, ..., x_n$.

Số các biến xuất hiện trong một hội sơ cấp được gọi là hạng của hội sơ cấp đó. Chẳng hạn, hội sơ cấp $x_1 \bar{x}_3 x_4$ có hạng 3, hội sơ cấp $\bar{x}_2 x_3 x_5 \bar{x}_7$ có hạng 4.

Giả sử $f(x_1,..., x_n)$ là một hàm đại số lôgic. Một công thức biểu diễn f dưới dạng tuyển của một số hội sơ cấp khác nhau của các biến $x_1, ..., x_n$ được gọi là một dạng tuyển chuẩn tắc của hàm đó.

Thí dụ 1.

$\bar{x}y \vee x\bar{y}$ là một dạng tuyển chuẩn tắc của hàm $x + y$.

$\bar{x} \vee \bar{y}$ và $\bar{x}\bar{y} \vee \bar{x}y \vee x\bar{y}$ là các dạng tuyển chuẩn tắc của hàm Sheffer $x \mid y$.

Dễ thấy rằng, dạng tuyển chuẩn tắc hoàn toàn của f là dạng chuẩn tắc duy nhất của f mà trong đó mọi hội sơ cấp đều có hạng n. Mỗi hội sơ cấp (hạng n) trong dạng tuyển chuẩn tắc hoàn toàn của f được gọi là một cấu tạo đơn vị của f.

Ta gọi độ phức tạp của một dạng chuẩn tắc là số các ký hiệu biến xuất hiện trong dạng chuẩn tắc đó.

Thí dụ 2.

$\bar{x}\bar{y} \vee \bar{x}y \vee x\bar{y}$ có độ phức tạp là 6,

$\bar{x} \vee \bar{y}$ có độ phức tạp là 2.

Dạng tuyển chuẩn tắc của f có độ phức tạp bé nhất được gọi là dạng tuyển chuẩn tắc tối thiểu của f. Dạng tuyển chuẩn tắc tối thiểu, theo một nghĩa nào đó, tương ứng với một sơ đồ tối thiểu (thuộc một loại nhất định) thực hiện f. Những phần tiếp theo của chương này, giới thiệu một số phương pháp tìm dạng tuyển chuẩn tắc tối thiểu của một hàm đại số lôgic.

Ta đưa thêm một khái niệm quan trọng sau đây.

Giả sử $f(x_1,..., x_n)$ là một hàm đại số lôgic. Một hàm $g(x_1,..., x_n)$ được gọi là một nguyên nhân (implicant) của f nếu T_g là tập con của T_f, nói cách khác, nếu $(g \to f) = 1$ (vì thế có tên gọi là nguyên nhân).

Dễ dàng thấy rằng mỗi hội sơ cấp trong một dạng tuyển chuẩn tắc của f là một nguyên nhân của f.

Xét một hội sơ cấp A. Việc xoá bớt một biến trong nó làm tập đặc trưng T_A của nó "nở" ra. Điều đó khiến cho, nếu trước đó A là một nguyên nhân của f, thì sau khi xoá một biến, A có thể không phải là nguyên nhân của f nữa. Một hội sơ cấp A được gọi là một nguyên nhân nguyên tố của f, nếu A là một nguyên nhân của f, sao cho không thể xoá đi bất cứ biến nào trong nó (cùng với dấu phủ định nếu có) để A vẫn còn là nguyên nhân của f.

Thí dụ 3.

$\bar{x}y$ và $x\bar{y}$ là các nguyên nhân nguyên tố của hàm $x + y$.

$\bar{x}$ và $\bar{y}$ là các nguyên nhân nguyên tố của hàm $x \mid y$ nhưng các hội $\bar{x}\bar{y}$, $\bar{x}y$, $x\bar{y}$ không phải là các nguyên nhân nguyên tố của nó.

2.2. Dạng tuyển chuẩn tắc thu gọn

Định lý 1. *Tuyển của một số bất kỳ các nguyên nhân của hàm f cũng là một nguyên nhân của hàm đó.*

Chứng minh. Tập đặc trưng của tuyển này là hợp của các tập đặc trưng của các nguyên nhân đang xét, vì thế nó cũng là tập con của tập đặc trưng của hàm f.

Định lý được chứng minh.

Giả sử S là một hệ các nguyên nhân của f. Ta nói rằng hệ S là đầy đủ nếu với mọi giá trị $\sigma \in T_f$ luôn tìm được một nguyên nhân g thuộc hệ để $\sigma \in T_g$.

Định lý 2. *Giả sử S là một hệ đầy đủ các nguyên nhân của hàm f. Khi đó tuyển của tất cả các nguyên nhân trong S sẽ trùng với f (ta cũng nói tuyển này thực hiện f).*

Chứng minh. Từ định nghĩa của hệ đầy đủ, ta nhận được

$$\bigcup_{g \in S} T_g = T_f$$

nhưng vế trái của đẳng thức trên chính là tập đặc trưng của tuyển đang xét. Từ đó nhận được định lý.

Định lý 3. *Tuyển của tất cả các nguyên nhân nguyên tố của hàm f là thực hiện f. Nói khác đi, hệ các nguyên nhân nguyên tố của f là một hệ đầy đủ.*

Chứng minh. Để chứng minh định lý, ta sẽ chứng minh rằng

$$\bigcup_{g \in S} T_g = T_f$$

trong đó S là hệ các nguyên nhân nguyên tố của f.

Vì S gồm các nguyên nhân của f nên vế trái của đẳng thức trên là tập con của vế phải. Ta chỉ cần chứng tỏ điều ngược lại. Thật vậy, gọi S' là hệ các cấu tạo đơn vị của f, ta có (dạng tuyển chuẩn tắc hoàn toàn của f):

$$\bigcup_{g' \in S'} T_{g'} = T_f$$

Xét $g' \in S'$, nếu g' không phải là nguyên nhân nguyên tố của f, thì bằng cách xoá bớt một số biến (cùng với dấu phủ định nếu có) trong g', ta thu được một nguyên nhân nguyên tố của f. Gọi nguyên nhân này là g ($g \in S$). Rõ ràng $T_{g'}$ là tập con của T_g. Từ đó nhận được

$$\bigcup T_{g'} = T_f \subset \bigcup T_g$$

$$g' \in S' \qquad\qquad g \in S$$

hay

$$T_f \subset \bigcup_{g \in S} T_g$$

và định lý được chứng minh.

Tuyển của tất cả các nguyên nhân nguyên tố của f được gọi là dạng tuyển chuẩn tắc thu gọn của f. Vì tập hợp các nguyên nhân nguyên tố của f là hoàn toàn xác định, nên dạng tuyển chuẩn tắc thu gọn của nó là duy nhất.

Chú ý rằng, dạng tuyển chuẩn tắc thu gọn của một hàm đại số lôgic, nói chung còn khác xa với dạng tuyển chuẩn tắc tối thiểu của hàm đó. Chẳng hạn, hàm $(x + y) \vee z\ (x \vee y)$ có dạng tuyển chuẩn tắc thu gọn là

$$\bar{x}y \vee x\bar{y} \vee xz \vee yz,$$

nhưng một dạng chuẩn tắc tối thiểu của nó là

$$\bar{x}y \vee x\bar{y} \vee xz.$$

Mặc dù dạng tuyển chuẩn tắc thu gọn chưa phải là tối thiểu, nhưng việc tìm nó, như trong những phương pháp trình bày dưới đây, là một bước trung gian quan trọng trong quá trình tìm dạng tuyển chuẩn tắc tối thiểu.

2.3. Dạng tuyển chuẩn tắc nghẽn và dạng tuyển chuẩn tắc tối thiểu

Một hệ nguyên nhân nguyên tố của hàm f được gọi là một hệ nghẽn, nếu nó đầy đủ và không một hệ con thực sự nào của nó là đầy đủ. Nếu S là một hệ nghẽn của f thì tuyển của các thành viên trong S sẽ thực hiện f. Tuyển này được gọi là dạng tuyển chuẩn tắc nghẽn của f. Nói khác đi dạng tuyển chuẩn tắc nghẽn của f là một dạng tuyển chuẩn tắc gồm các nguyên nhân nguyên tố của f, thực hiện f mà không thể bỏ bớt đi một số hạng nào để vẫn thực hiện được f.

Chú ý rằng, dạng tuyển chuẩn tắc nghẽn của một hàm đại số lôgic là không duy nhất, chẳng hạn hàm

$$f(x, y, z) = \bar{x}y \vee x\bar{y} \vee xz \vee yz$$

có hai dạng tuyển chuẩn tắc nghẽn là

$$\bar{x}y \vee x\bar{y} \vee xz \quad \text{và} \quad \bar{x}y \vee x\bar{y} \vee yz.$$

Định lý 4. *Mọi dạng tuyển chuẩn tắc tối thiểu của một hàm đại số lôgic f đều là một dạng tuyển chuẩn tắc nghẽn của hàm đó.*

Chứng minh. Để chứng minh định lý, ta chỉ cần chứng tỏ rằng nếu p là một hội sơ cấp của một dạng tuyển chuẩn tắc tối thiểu của f, thì p phải là một nguyên nhân nguyên tố của f.

Giả sử p không phải là nguyên nhân nguyên tố của f. Khi đó, bằng cách bỏ đi một số biến (cùng với dấu phủ định nếu có), ta luôn tìm được một phần con thực sự q của p, để q là một nguyên nhân nguyên tố của f. Gọi $p \vee P$ là dạng tuyển chuẩn tắc tối thiểu của f (P là một tuyển của một số hội sơ cấp nào đó). Thay p bởi q trong tuyển này, ta nhận được $q \vee P$ cũng là một dạng tuyển chuẩn tắc của f. Điều này mâu thuẫn với tính tối thiểu của $p \vee P$, và định lý được chứng minh.

Chú ý rằng, điều ngược lại của định lý 4 là không đúng: có những dạng tuyển chuẩn tắc nghẽn mà không phải là dạng tuyển chuẩn tắc tối thiểu, thí dụ hàm

$$f(x, y, z) = (x + y) \vee (x + z)$$

có một dạng chuẩn tắc nghẽn là

$$x\,\bar{z} \vee x\,\bar{y} \vee \bar{x}y \vee \bar{x}z,$$

nhưng đó không phải là tối thiểu, một dạng như thế của f là

$$x\,\bar{z} \vee \bar{x}y \vee \bar{y}z.$$

Định lý 4 cho thấy rằng, để tìm dạng tuyển chuẩn tắc tối thiểu, ta chỉ cần xét các dạng tuyển chuẩn tắc nghẽn. Mà để tìm các dạng tuyển chuẩn tắc nghẽn thì cần biết dạng tuyển chuẩn tắc thu gọn. Đó cũng là cơ sở cho cách tìm dạng tuyển chuẩn tắc tối thiểu trình bày trong chương sau.

Bài tập

1. Cho hàm đại số lôgic $f(x, y, z)$ xác định bởi bảng:

x	y	z	$f(x, y, z)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

a) Liệt kê tất cả các nguyên nhân của f, trong đó mỗi nguyên nhân được viết dưới dạng tuyển của một số hội sơ cấp. Số lượng của chúng là bao nhiêu (kể cả tuyển rỗng và tuyển chuẩn tắc hoàn toàn)?

b) Xác định các nguyên nhân nguyên tố trong các nguyên nhân đã tìm.

2. Cho hàm đại số lôgic n đối số. Biết dạng tuyển chuẩn tắc hoàn toàn của nó gồm k hội sơ cấp. Đếm tất cả các nguyên nhân của hàm đã cho.

3

THUẬT TOÁN TÌM DẠNG TUYỂN CHUẨN TẮC TỐI THIỂU

3.1. Chú ý mở đầu

Từ những cơ sở lý thuyết đã trình bày trong chương 2, ta có thể chia quá trình tìm dạng tuyển chuẩn tắc tối thiểu của một hàm đại số lôgic $f(x_1, ..., x_n)$ thành 2 giai đoạn:

a) Xuất phát từ dạng tuyển chuẩn tắc hoàn toàn của f, tìm dạng tuyển chuẩn tắc thu gọn của f.

b) Xuất phát từ dạng tuyển chuẩn tắc thu gọn của f, tìm các dạng tuyển chuẩn tắc nghẽn của f và lựa chọn từ các dạng này để được các dạng tuyển chuẩn tắc tối thiểu của f.

Vì dạng tuyển chuẩn tắc thu gọn của f là duy nhất, nên giai đoạn a) hoàn toàn xác định. Trái lại, giai đoạn b) cần phải xây dựng các dạng nghẽn từ dạng rút gọn, sau đó lựa chọn trong chúng để được dạng tối thiểu. Số các dạng nghẽn của một hàm đại số

lôgic có thể rất lớn, điều này làm cho việc lựa chọn trở nên khó khăn, nhiều khi không khác mấy so với việc "lựa chọn toàn bộ". Iablonski đã xây dựng được những hàm đại số lôgic $f(x_1,..., x_n)$ mà số các nguyên nhân nguyên tố của nó lớn gấp $2^{n/2}$ lần số các hội sơ cấp trong dạng tuyển chuẩn tắc hoàn toàn của f. Glagolev chứng minh rằng, đối với hầu hết các hàm đại số lôgic f của n đối số, số các dạng tuyển chuẩn tắc nghẽn của f là vượt quá $2^{2^{n(1-e)}}$, trong đó $e \to 0$ khi $n \to \infty$. Juravliev đưa ra một thí dụ đơn giản sau đây: khi $n \geq 3$, hàm

$$f(x_1,..., x_n) = [(x_1 + x_2) \vee (x_2 + x_3)] (x_1 + ... + x_n)$$

có $5^{2^{n-4}}$ dạng tuyển chuẩn tắc nghẽn.

Như vậy, về nguyên tắc, việc tìm dạng tuyển chuẩn tắc tối thiểu từ dạng tuyển chuẩn tắc thu gọn đòi hỏi phải tiến hành những sự lựa chọn phức tạp. Tuy nhiên, các phương pháp được trình bày dưới đây, mặc dù nói chung, chúng không tránh được những khó khăn và phức tạp nói trên, trong nhiều trường hợp cụ thể, chúng cũng tỏ ra có hiệu quả.

Dưới đây là một số phép toán mà ta sẽ dùng nhiều lần trong quá trình biến đổi

(i) phép nuốt sơ cấp

$$A B \vee A = A$$

(ii) phép dán

$$A x \vee A \bar{x} = A$$

(iii) phép dán không đầy đủ

$$A x \vee A \bar{x} = A \vee A x \vee A \bar{x}$$

(iv) phép dán mở rộng

$$A C \vee B \bar{C} = A C \vee B \bar{C} \vee A B$$

Trong các phép toán đó A, B và C là các biểu thức bất kỳ, x là biến.

3.2. Tìm dạng tuyển chuẩn tắc thu gọn

Mỗi hội sơ cấp hạng k

$$x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} ... x_{i_j}^{\sigma_j}$$

được gọi là một nguyên nhân hạng k của hàm f, nếu nó là một nguyên nhân của f. Một cấu tạo đơn vị của f là một nguyên nhân hạng n của nó.

Định lý 1.

a) Một nguyên nhân hạng k của hàm f ($k < n$) là kết quả của phép dán hai nguyên nhân hạng k+1 của hàm f đó.

b) Một nguyên nhân hạng k của hàm f ($k \le n$) là nguyên nhân nguyên tố của f nếu không thể dán được với bất kỳ một nguyên nhân hạng k nào của f.

Chứng minh.

a) Giả sử A là một nguyên nhân hạng k ($k < n$) của f. Tìm được biến x không có mặt trong A. Khi đó các hội $A\,x$ và $A\,\bar{x}$ sẽ là các nguyên nhân hạng $k+1$ của f, và A nhận được từ phép dán hai nguyên nhân này.

b) Giả sử A là một nguyên nhân hạng k ($k \le n$) của f. Nếu A không phải là nguyên nhân nguyên tố thì luôn tìm được một biến x trong A để sau khi xoá nó (cùng với dấu phủ định nếu có) khỏi A, ta được phần con B là một nguyên nhân của f. Có thể giả thiết $A = B\,x$. Khi đó $A' = B\,\bar{x}$ cũng là một nguyên nhân hạng k của f và A có thể dán với A'.

Từ định lý 1, ta có thuật toán Quine sau đây để tìm dạng tuyển chuẩn tắc thu gọn của hàm đại số lôgic $f(x_1,..., x_n)$:

Bước 1. Tìm dạng tuyển chuẩn tắc hoàn toàn của f, ký hiệu f_0.

Bước 2. Từ f_i xây dựng f_{i+1} bằng cách trong f, thực hiện tất cả các phép dán không đầy đủ đối với các hội sơ cấp hạng n-i, sau đó xoá bỏ tất cả các hội sơ cấp hạng n-i có thể được bằng phép nuốt sơ cấp.

Bước 3. Lặp lại bước 2 cho đến khi thu được $f_{k+1} = f_k$. Khi đó f_k sẽ là dạng tuyển chuẩn tắc thu gọn của f.

Từ cách xây dựng f_i, có thể chứng minh quy nạp điều khẳng định: trongmỗi f_i, có mặt tất cả các nguyên nhân nguyên tố hạng không nhỏ hơn n-k+1 và tất cả các nguyên nhân hạng n-k của f, và chỉ có chúng.

Nếu $f_{k+1} = f_k$ thì theo phần b) của định lý 1, mọi nguyên nhân hạng n-k của f trong f_k đều là nguyên tố và như vậy f_k chứa tất cả các nguyên nhân nguyên tố của f, do đó nó là dạng tuyển chuẩn tắc thu gọn của f.

Thí dụ 1. Tìm dạng tuyển chuẩn tắc thu gọn của hàm

$$f = x\,y\,z \vee \bar{x}\,y\,z \vee x\,\bar{y}\,z \vee x\,y\,\bar{z} \vee \bar{x}\,\bar{y}\,\bar{z}.$$

Ta có $f_0 = f$. Dùng các phép dán không đầy đủ đối với các hội sơ cấp hạng 3, ta được

$$y\,z \vee x\,z \vee x\,y \vee x\,y\,z \vee \bar{x}\,y\,z \vee x\,\bar{y}\,z \vee x\,y\,\bar{z} \vee \bar{x}\,\bar{y}\,\bar{z}.$$

Sau đó dùng các phép nuốt sơ cấp ta được

$$f_1 = yz \vee xz \vee xy \vee \bar{x}\,\bar{y}\,\bar{z}.$$

Đến đây các hội sơ cấp hạng 2 không dán được với nhau, tức là $f_2 = f_1$ và f_1 là dạng tuyển chuẩn tắc thu gọn của f.

Thuật toán Quine không chỉ rõ việc tìm cho hết các phép dán có thể có. Vì thế McCluskey đề nghị bổ sung thuật toán Quine một thủ tục hình thức như trình bày dưới đây.

Giả sử f là hàm của n biến $x_1,..., x_n$. Mỗi hội sơ cấp của n biến đó được biểu diễn bằng một dãy n ký hiệu trong bảng {0, 1, -} theo quy ước: ký tự thứ i là 1 hay 0 nếu x_i có mặt trong hội là bình thường hay với dấu phủ định, còn nếu x_i không có mặt thì ký tự này là -. Chẳng hạn hội sơ cấp của 5 biến $x_1,..., x_5$: $x_1\,\bar{x}_3\,x_5$ được biểu diễn bởi 1-0-1. Hai hội sơ cấp được gọi là lân cận nhau, nếu các biểu diễn nói trên của chúng chỉ khác nhau ở một vị trí. Rõ ràng các hội sơ cấp chỉ có thể dán được với nhau nếu chúng là lân cận nhau.

Thuật toán Quine - McCluskey.

Thuật toán Quine, theo thủ tục McCluskey, được tiến hành như sau: Lập một bảng gồm nhiều cột để ghi kết quả các phép dán. Sau đó lần lượt thực hiện các bước:

Bước 1. Viết vào cột thứ nhất, các biểu diễn của các nguyên nhân hạng n (tức là các cấu tạo đơn vị) của hàm f. Các biểu diễn được chia thành từng nhóm, các biểu diễn trong mỗi nhóm có số các ký hiệu 1 bằng nhau và các nhóm xếp theo thứ tự số các ký hiệu 1 tăng dần (các nhóm được đánh số từ 1).

Bước 2. Lần lượt thực hiện tất cả các phép dán các biểu diễn trong nhóm i với các biểu diễn trong nhóm $i+1$ (i = 1, 2, ...). Biểu diễn nào tham gia ít nhất một phép dán sẽ được ghi nhận một dấu * bên cạnh. Kết quả dán được ghi vào cột tiếp theo.

Bước 3. Lặp lại bước 2 cho cột kế tiếp cho đến khi không thu thêm được cột nào mới (tức là tại cột hiện hành, không thực hiện được một phép dán nào cả). Khi đó, tất cả các biểu diễn không có dấu * sẽ cho ta tất cả các nguyên nhân nguyên tố của f.

Thí dụ 2. Tìm dạng tuyển chuẩn tắc thu gọn của hàm

$$f = \bar{x}\,\bar{y}\,\bar{z}u \vee \bar{x}y\,\bar{z}u \vee \bar{x}\,\bar{y}zu \vee x\,\bar{y}\,\bar{z}u \vee x\,\bar{y}zu \vee \bar{x}yzu \vee xyzu.$$

Thủ tục McCluskey được tiến hành bởi bảng sau đây

0001 *	0-01 *	0--1
0101 *	00-1 *	-0-1
0011 *	-001 *	--11
1001 *	-011 *	
1011 *	10-1 *	
0111 *	01-1 *	
1111 *	0-11 *	
	1-11 *	
	-111 *	

Dạng tuyển chuẩn tắc thu gọn của hàm f là

$$f = \bar{x}\,u \vee \bar{y}\,u \vee z\,u.$$

Thí dụ 3. Tìm dạng tuyển chuẩn tắc thu gọn của hàm

$$f = \bar{x}_1\,\bar{x}_2\,x_3\,\bar{x}_4 \vee \bar{x}_1\,\bar{x}_2\,x_3\,x_4 \vee x_1\,\bar{x}_2\,x_3\,x_4 \vee x_1\,x_2\,\bar{x}_3\,\bar{x}_4$$
$$\vee\, x_1\,x_2\,\bar{x}_3\,x_4 \vee x_1\,x_2\,x_3\,\bar{x}_4 \vee x_1\,x_2\,x_3\,x_4$$

Thủ tục McCluskey được tiến hành bởi bảng sau đây.

0010 *	001-	11--
0011 *	-011	
1100 *	110- *	
1011 *	11-0 *	
1101 *	1-11	
1110 *	11-1 *	
1111 *	111- *	

Dạng tuyển chuẩn tắc thu gọn của hàm f là

$$f = x_1\,x_2 \vee \bar{x}_1\,\bar{x}_2\,x_3 \vee \bar{x}_2\,x_3\,x_4 \vee x_1\,x_2\,x_3\,.$$

Thuật toán Quine cùng với sự cải tiến của McCluskey gọi chung là thuật toán Quine -McCluskey. Thuật toán này cho phép tìm dạng tuyển chuẩn tắc thu gọn từ dạng tuyển chuẩn tắc hoàn toàn.

Dưới đây trình bày thêm phương pháp Blake - Poreski, cho phép tìm dạng tuyển chuẩn tắc thu gọn từ một dạng tuyển chuẩn tắc tuỳ ý. Cơ sở của phương pháp này là định lý sau đây:

Định lý 2. *Nếu trong một dạng tuyển chuẩn tắc tuỳ ý của hàm đại số lôgic $f(x_1, ..., x_n)$ ta liên tiếp thực hiện tất cả các phép dán mở rộng có thể có được rồi sau đó thực hiện tất cả các phép nuốt sơ cấp, thì sẽ thu được dạng tuyển chuẩn tắc thu gọn của hàm f.*

Có thể chứng minh định lý này bằng quy nạp theo số các đối số của hàm *f*. Tuy nhiên chúng tôi không trình bày chi tiết ở đây.

Thí dụ 4. Tìm dạng tuyển chuẩn tắc thu gọn của hàm

$$f = x y \bar{z} \vee z x \vee \bar{x} y.$$

Thực hiện liên tiếp các phép dán mở rộng và các phép nuốt sơ cấp ta được

$$\begin{aligned} f &= x y \bar{z} \vee z x \vee \bar{x} y \\ &= x y \bar{z} \vee z x \vee \bar{x} y \vee x y \vee y \bar{z} \vee y z \\ &= x z \vee \bar{x} y \vee x y \vee y \bar{z} \vee y z \vee y \\ &= x z \vee y. \end{aligned}$$

Vậy $x z \vee y$ là dạng tuyển chuẩn tắc thu gọn của hàm *f*.

Thí dụ 5. Tìm dạng tuyển chuẩn tắc thu gọn của hàm

$$f = (x + y) \vee (y + z).$$

Ta có

$$\begin{aligned} f &= \bar{x} y \vee x \bar{y} \vee \bar{y} z \vee y \bar{z} \\ &= \bar{x} y \vee x \bar{y} \vee \bar{y} z \vee y \bar{z} \vee \bar{x} z \vee x \bar{z} \end{aligned}$$

Đó là dạng tuyển chuẩn tắc thu gọn của hàm *f*. Thí dụ 5 chứng tỏ rằng, dạng tuyển chuẩn tắc thu gọn của một hàm *f* có thể "dài" hơn dạng nguyên thuỷ của nó.

3.3. Tìm dạng tuyển chuẩn tắc tối thiểu

Sau khi tìm được dạng tuyển chuẩn tắc thu gọn của *f*, nghĩa là tìm được tất cả các nguyên nhân nguyên tố của nó, ta tiếp tục phương pháp Quine tìm dạng tuyển chuẩn tắc tối thiểu của f như sau: Lập một bảng chữ nhật, mỗi cột ứng với một cấu tạo đơn vị của *f* và mỗi dòng ứng với một nguyên nhân nguyên tố của *f*. Tại ô (i, j), ta đánh dấu + nếu nguyên nhân nguyên tố ở dòng *i* là một phần con của cấu tạo đơn vị ở cột *j*. Ta cũng nói rằng khi đó, nguyên nhân nguyên tố *i* là phủ cấu tạo đơn vị *j*. Một hệ *S* các nguyên nhân nguyên tố của *f* được gọi là phủ hàm *f*, nếu mọi cấu tạo đơn vị của *f* đều được phủ ít

nhất bởi một thành viên thuộc hệ. Dễ thấy rằng, nếu hệ S là phủ hàm f thì nó là đầy đủ, nghĩa là tuyển của các thành viên trong S là thực hiện f.

Một nguyên nhân nguyên tố được gọi là cốt yếu, nếu thiếu nó thì một hệ các nguyên nhân nguyên tố không thể phủ hàm f. Các nguyên nhân nguyên tố cốt yếu được tìm như sau: tại những cột chỉ có duy nhất một dấu +, xem dấu + đó thuộc dòng nào thì dòng đó ứng với một nguyên nhân nguyên tố cốt yếu.

Việc lựa chọn các nguyên nhân nguyên tố trên bảng đã đánh dấu, để được một dạng tuyển chuẩn tắc tối thiểu, có thể tiến hành theo các bước sau đây:

Bước 1. Phát hiện tất cả các nguyên nhân nguyên tố cốt yếu.

Bước 2. Xoá tất cả các cột được phủ bởi các nguyên nhân nguyên tố cốt yếu, tức là tất cả các cột có ít nhất một dấu + tại những dòng ứng với các nguyên nhân nguyên tố cốt yếu.

Bước 3. Trong bảng còn lại, xoá nốt những dòng không còn dấu + và sau đó nếu có hai cột giống nhau thì xoá bớt một cột.

Bước 4. Sau các bước trên, tìm một hệ S các nguyên nhân nguyên tố với số biến ít nhất phủ tất cả các cột còn lại.

Tuyển của các nguyên nhân nguyên tố cốt yếu và các nguyên nhân trong hệ S sẽ là dạng tuyển chuẩn tắc tối thiểu của hàm f.

Các bước 1, 2, 3 có tác dụng rút gọn bảng trước khi lựa chọn. Độ phức tạp chủ yếu nằm ở bước 4. Tình huống tốt nhất là mọi nguyên nhân nguyên tố đều là cốt yếu. Trường hợp này không phải lựa chọn gì và hàm f có duy nhất một dạng tuyển chuẩn tắc tối thiểu cũng chính là dạng tuyển chuẩn tắc thu gọn. Tình huống xấu nhất là không có nguyên nhân nguyên tố nào là cốt yếu. Trường hợp này ta phải lựa chọn toàn bộ bảng. Các thí dụ sau đây minh hoạ các tình huống có thể xảy ra.

Thí dụ 1. Tìm dạng tuyển chuẩn tắc tối thiểu của hàm f cho trong thí dụ 1, mục 2. Bảng sau khi đã đánh dấu + có dạng

	xyz	$\bar{x}yz$	$x\bar{y}z$	$xy\bar{z}$	$\bar{x}\bar{y}\bar{z}$
xy	+			+	
xz	+		+		
yz	+	+			
$\bar{x}\bar{y}\bar{z}$					+

Trong trường hợp này mọi nguyên nhân nguyên tố đều là cốt yếu. Hàm f có một dạng tuyển chuẩn tắc tối thiểu, đồng thời cũng là dạng tuyển chuẩn tắc thu gọn:

$$f = xy \vee xz \vee yz \vee \bar{x}\,\bar{y}\,\bar{z}.$$

Thí dụ 2. Tìm dạng tuyển chuẩn tắc tối thiểu của hàm f cho trong thí dụ 3, mục 2. Sau khi đánh dấu + , bảng có dạng

	$\bar{x}_1\,\bar{x}_2$ $x_3\,\bar{x}_4$	$\bar{x}_1\,\bar{x}_2$ $x_3\,x_4$	$x_1\,\bar{x}_2$ $x_3\,x_4$	x_1x_2 $\bar{x}_3\,\bar{x}_4$	$x_1\,x_2$ $\bar{x}_3\,x_4$	$x_1\,x_2$ $x_3\,\bar{x}_4$	$x_1\,x_2$ $x_3\,x_4$
$x_1\,x_2$				+	+	+	+
$\bar{x}_1\,\bar{x}_2x_3$	+	+					
$\bar{x}_2\,x_3\,x_4$		+	+				
$x_1\,x_3\,x_4$			+				+

Có hai nguyên nhân nguyên tố cốt yếu nằm ở dòng 1 và 2. Sau khi rút gọn, bảng còn hai dòng 3, 4 và một cột 3. Việc chọn S khá đơn giản: có thể chọn một trong hai nguyên nhân nguyên tố còn lại. Vì vậy ta được hai dạng tuyển chuẩn tắc tối thiểu là

$$f = x_1x_2 \vee \bar{x}_1\,\bar{x}_2x_3 \vee \bar{x}_2x_3x_4,$$
$$f = x_1x_2 \vee \bar{x}_1\,\bar{x}_2x_3 \vee x_2x_3x_4.$$

Thí dụ 3. Tiếp tục thí dụ 5, mục 2. Dạng tuyển chuẩn tắc hoàn toàn của hàm f là

$$f = \bar{x}\,\bar{y}z \vee \bar{x}y\,\bar{z} \vee \bar{x}yz \vee x\,\bar{y}\,\bar{z} \vee x\,\bar{y}z \vee xy\,\bar{z}.$$

Ta lập bảng

	$\bar{x}\,\bar{y}z$	$\bar{x}y\,\bar{z}$	$\bar{x}yz$	$x\,\bar{y}\,\bar{z}$	$x\,\bar{y}z$	$xy\,\bar{z}$
$\bar{x}y$		+	+			
$x\,\bar{y}$				+	+	
$\bar{y}z$	+				+	
$y\,\bar{z}$		+				+
$\bar{x}z$	+		+			
$x\,\bar{z}$				+		+

Không có nguyên nhân nào là nguyên nhân nguyên tố cốt yếu. Trường hợp này phải lựa chọn toàn bộ. Có hai hệ phủ hàm f với số biến ít nhất, chúng tương ứng với hai dạng tuyển chuẩn tắc tối thiểu:

$$f = \bar{x}y \vee \bar{y}z \vee x\bar{z},$$

$$f = x\bar{y} \vee y\bar{z} \vee \bar{x}z.$$

Hệ $\{ \bar{x}y, x\bar{y}, \bar{y}z, y\bar{z} \}$ phủ f không thừa, nó cho ta một dạng tuyển chuẩn tắc nghẽn

$$f = \bar{x}y \vee x\bar{y} \vee \bar{y}z \vee y\bar{z}$$

nhưng không phải là tối thiểu.

3.4. Sơ đồ tối thiểu

Dạng tuyển chuẩn tắc của một hàm đại số lôgic là một loại biểu diễn đơn giản, nó tương ứng với một loại sơ đồ nào đó thực hiện hàm đã cho. Trong ứng dụng kỹ thuật, vấn đề là với một hàm đại số lôgic cho trước, làm thế nào để xây dựng một sơ đồ thực hiện được hàm đó, với ít yếu tố cơ bản nhất. Sơ đồ như vậy được gọi là sơ đồ tối thiểu của hàm đang xét. Nói chung, sơ đồ ứng với dạng tuyển chuẩn tắc tối thiểu mà ta tìm trong mục trước, chưa phải đã là sơ đồ tối thiểu.

Thí dụ. Sơ đồ thực hiện hàm f theo dạng tuyển chuẩn tắc tối thiểu

$$f = xy \vee xz \vee yz \vee \bar{x}\bar{y}\bar{z}$$

cần dùng 8 yếu tố & (hội) và $\vee$ (tuyển), trong khi đó sơ đồ theo biểu diễn

$$f = x(y \vee z) \vee yz \vee \bar{x}\bar{y}\bar{z}$$

tốn tất cả là 7 yếu tố & và $\vee$.

Vấn đề tìm sơ đồ tối thiểu (không nhất thiết là dạng tuyển chuẩn tắc) của một hàm đại số lôgic (trong hệ $\vee$, &, -) là một vấn đề phức tạp. Hiện nay chưa có một thuật toán

hữu hiệu nào để giải quyết vấn đề này. Thông thường, người ta tìm dạng tuyển chuẩn tắc tối thiểu, rồi sau đó dùng các luật phân bố để rút bớt số các chữ biến. Làm như vậy, tuy chưa hẳn đạt được một biểu diễn tối thiểu, nhưng nói chung, cũng thu được một biểu diễn cho phép xây dựng một sơ đồ tương đối tiết kiệm.

Rõ ràng là vấn đề còn trở nên phức tạp hơn nhiều, nếu xét việc biểu diễn trên một hệ đầy đủ bất kỳ. Ngoài ra, trên thực tế, ta thường gặp việc xây dựng một sơ đồ thực hiện đồng thời một hệ hàm đại số lôgic. Có thể xây dựng riêng rẽ sơ đồ tối thiểu cho từng hàm, rồi ghép chúng lại, với chú ý là có thể lợi dụng những phần chung trong chúng. Tuy nhiên, cần xét vấn đề này một cách căn bản và có hệ thống hơn.

Một vấn đề nữa là, trên thực tế, ta lại hay gặp những hàm đại số lôgic xác định không đầy đủ (do việc mã hoá nhị phân các tín hiệu rời rạc). Việc chọn những giá trị cho hàm tại những nơi không xác định, càng làm tăng thêm các khả năng được xét.

Tóm lại, đối với việc tổng hợp các sơ đồ thực hiện các hàm đại số lôgic còn rất nhiều bài toán khó khăn và lý thú. Giáo trình này chỉ đề cập đến một vài khía cạnh cơ bản nhất. Bạn đọc nào quan tâm, cần tham khảo thêm các tài liệu khác.

Bài tập

1. Tìm tất cả các nguyên nhân nguyên tố của các hàm dưới đây theo thuật toán *Quine - McCluskey*:

a) $f(x, y, z) = x\,y\,\bar{z} \vee x\,\bar{y}\,\bar{z} \vee \bar{x}\,y\,z \vee \bar{x}\,\bar{y}\,\bar{z}$

b) $f(x, y, z) = x\,\bar{y}\,z \vee x\,\bar{y}\,\bar{z} \vee \bar{x}\,y\,z \vee \bar{x}\,\bar{y}\,z \vee \bar{x}\,\bar{y}\,\bar{z}$

c) $f(x, y, z) = x\,y\,z \vee x\,\bar{y}\,z \vee \bar{x}\,y\,z \vee \bar{x}\,\bar{y}\,z \vee \bar{x}\,\bar{y}\,\bar{z}$

d) $f(x, y, z) = (x \mid y) \mid z$

e) $f(x, y, z) = x \mid (y + z)$

f) $f(x, y, z) = x \circ (y \mid z)$

g) $f(x, y, z) = (x \circ y) \mid z$

h) $f(x, y, z) = (x + y) \circ z$

2. Tìm dạng tuyển chuẩn tắc thu gọn của các hàm cho trong bài **1**.

3. Viết chương trình tìm dạng tuyển chuẩn tắc thu gọn của một hàm đại số lôgic n đối số dựa trên dạng tuyển chuẩn tắc hoàn toàn của nó.

Dữ liệu vào là file văn bản có dạng:

- dòng đầu ghi giá trị n (là số đối số của hàm)
- dòng sau ghi giá trị k (là số hội sơ cấp trong dạng tuyển chuẩn tắc hoàn toàn của hàm)
- k dòng tiếp, mỗi dòng ghi một hội sơ cấp tương ứng dưới dạng một dãy n ký tự viết liền nhau, trong đó ký tự thứ i mô tả trạng thái của đối thứ i theo quy ước: 0 nếu đối có dấu phủ định, 1 nếu đối không có dấu phủ định.

Kết quả ghi ra file văn bản có dạng (nếu có nhiều dạng tối thiểu thì chỉ ghi một):

- dòng đầu ghi giá trị m (là số hội sơ cấp trong dạng tuyển chuẩn tắc tối thiểu của hàm)
- m dòng sau, mỗi dòng ghi một hội sơ cấp tương ứng dưới dạng một dãy n ký tự viết liền nhau, trong đó ký tự thứ i mô tả trạng thái của đối thứ i theo quy ước: - nếu đối không có mặt, 0 nếu đối có dấu phủ định, 1 nếu đối không có dấu phủ định.

Thí dụ: File dữ liệu vào cho hàm (thí dụ 3, mục 3)

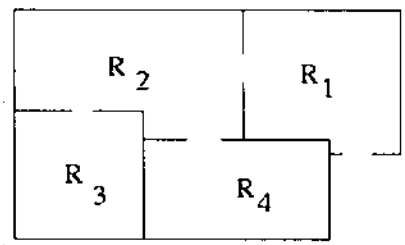
$$f(x, y, z) = \bar{x}\,\bar{y}\,z \vee \bar{x}\,y\,\bar{z} \vee \bar{x}\,y\,z \vee x\,\bar{y}\,\bar{z} \vee x\,\bar{y}\,z \vee x\,y\,\bar{z},$$

và file kết quả tương ứng với dạng tuyển chuẩn tắc tối thiểu $\bar{x}\,y \vee \bar{y}\,z \vee x\,\bar{z}$ có nội dung:

File dữ liệu vào	File kết quả
3 6 001 010 011 100 101 110	3 01- -01 1-0

4. *Hệ thống cảnh báo tự động.*

Người ta cần thiết kế một mạch logic điều khiển hệ thống cảnh báo tự động trong một ngôi nhà. Trong hệ thống cảnh báo này người ta sử dụng hai dạng bộ cảm biến (sensor): tích cực và thụ động. Bộ cảm biến tích cực sẽ phát tín hiệu 1 khi mọi việc đều bình thường và tín hiệu 0 nếu phát hiện có sự cố (cửa ra vào hoặc của sổ bị mở). Bộ cảm biến thụ động sẽ phát tín hiệu 0 nếu mọi việc bình thường và tín hiệu 1 nếu phát hiện sự cố (làm chuyển động hoặc va đập vào nó). Ngôi nhà có 4 phòng R_1, R_2, R_3, R_4. Trong mỗi phòng R_i có một bộ cảm biến tích cực phát tín hiệu x_i và một bộ cảm biến thụ động phát tín hiệu y_i. Sơ đồ của ngôi nhà được mô tả trong hình 1 (các đoạn đứt là các cửa thông phòng hoặc ra vào).



Hình 1

Mạch logic điều khiển phải phát ra tín hiệu 0 khi không có sự cố nào xảy ra và phát tín hiệu 1 để kích còi báo động. Ta cần kích còi báo động nếu xảy ra ít nhất một trong các tình huống sau đây:

- Trong phòng R_1 có ít nhất một trong hai bộ cảm biến x_1 hoặc y_1 phát hiện sự cố;
- Trong bất cứ phòng R_i, $i > 1$, cả hai bộ cảm biến x_i và y_i đều phát hiện sự cố;
- Trong hai phòng có cửa thông nhau có hai bộ cảm biến phát hiện sự cố.

a) Hãy xây dựng biểu thức cho hàm biến bun $f(x_1,x_2,x_3,x_4,y_1,y_2,y_3,y_4)$ nhận giá trị 1 khi và chỉ khi cần kích còi báo động.

b) Tìm dạng tuyển chuẩn tắc hoàn toàn của f;

c) Áp dụng thuật toán Quine-McCluskey tìm dạng tuyển chuẩn tắc thu gọn của f.

Tài liệu tham khảo

1. Hall M. *Combinatorial Theory*. Blaisdell Publishing Company, London, 1967.
2. Kaufman A. *Introduction à la combinatorique en vue des applications*. Dunod, Paris, 1968.
3. Aho A.W., Hopcroft J.E., Ultman J.D. *The design and analysis of computer algorithms*. Addison - Wesley Publishing Co., Inc., 1974.
4. Reingold E.M., Nievergelt J., Deo N. *Combinatorial Algorithms. Theory and Practice*. Prentice-Hall Inc. Englewood Cliff, New Jersey, 1977.
5. Papadmitriou C.H., Steiglitz K. *Combinatorial Optimization*. Prentice Hall Inc., N. J., 1982.
6. Rubnikov K.A. *Introduction in combinatorial analysis*. Nauka, Moscow, 1972. (in Russ.)
7. Sachkov V.N. *Combinatorial Methods of Discrete Mathematics*. Nauka, Moscow, 1977. (in Russ.)
8. Conway R.W., Maxwell W.L., Miller L.W. *Theory of Scheduling*. Addison - Wesley Reading, Mass., 1967.
9. Tanaev V.C., Skurba V.V. *Introduction to Scheduling Theory*. Nauka, Moscow, 1975. (in Russ.)
10. Kovaliev M.M. *Discrete Optimization*. Izd.BGU, Minsk, 1977 (in Russ.)
11. Berg C. *Theorie des Graphes et ses Applications*. Dunod, Paris, 1958.
12. Ore O. *Theory of Graphs*. American Mathematical Society, 1962.
13. Ford L.R., Fulkerson D.R. *Flows in Networks*. Princeton Univ. Press, Princeton, N.J., 1962.
14. Busacker R.G., Saaty T.L. *Finite Graphs and Networks. An Introduction with Application*. McGraw Hill, N.Y., 1965.
15. Harary F. *Graph Theory*. Addison Wesley Publishing Academic Press, N.Y., 1973.

16. Christofides N. Graph Theory. *An algorithmic approach*. Academic Press, N.Y., 1975.

17. Iablonski S.V. *Introduction to Discrete Mathematics*. Nauka, Moscow, 1979. (in Russ.).

18. Liu C.L. *Elements of Discrete Mathematics*. McGraw - Hill Book Company, 1985.

19. Rosen K.H. *Discrete Mathematics and its Applications*. McGraw - Hill Book Company, 1991.

20. Johnsonbaugh R. *Discrete Mathematics*. Prentice Hall Inc., N. J., 1997.

21. Hoàng Tuy. *Đồ thị hữu hạn và các ứng dụng trong vận trù học*, NXB Khoa học, Hà nội, 1964.

22. Phan đình Diệu. *Lý thuyết ôtômat hữu hạn và thuật toán*. NXB ĐHTHCN, Hà nội, 1977.

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

16 Hàng Chuối - Hai Bà Trưng - Hà Nội
Điện thoại: (04) 9718312; (04) 7547936. Fax: (04) 9714899
E-mail: nxb@ vnu.edu.vn

★ ★ ★

Chịu trách nhiệm xuất bản:

Giám đốc: PHÙNG QUỐC BẢO

Tổng biên tập: PHẠM THÀNH HƯNG

Người nhận xét: TS PHẠM PHÚ TRIÊM

Biên tập: TS PHẠM PHÚ TRIÊM

Biên tập tái bản: HỒ ĐỒNG, ĐỖ PHÚ

Trình bày bìa: VĂN SÁNG

TOÁN RỜI RẠC

Mã số: 1L-83 ĐH2006
In 1.000 cuốn, khổ 19 x 27 cm tại Trung tâm in Tranh Tuyên truyền Cổ động
Số xuất bản: 85 - 2006/CXB/71 - 01/ĐHQGHN, ngày 24/01/2006
Quyết định xuất bản số: 217 LK/XB
In xong và nộp lưu chiểu quý III năm 2006.

GIÁ: 37.000đ