
LÝ THUYẾT ĐỒ THỊ

Graph Theory

(Tham khảo chính từ bộ Slide bài giảng của PGS Nguyễn Đức Nghĩa)

Chương 1.

CÁC KHÁI NIỆM CƠ BẢN

Đồ thị là gì?

- Là một cấu trúc rời rạc bao gồm các đỉnh và các cạnh nối các đỉnh này.

Các ứng dụng thực tế của đồ thị

- Có tiềm năng ứng dụng trong nhiều lĩnh vực,
- Ứng dụng trong mạng máy tính, mạng giao thông, mạng cung cấp nước, mạng điện, lập lịch, tối ưu hóa luồng, thiết kế mạch, quy hoạch phát triển,...
- Các ứng dụng khác: Phân tích gen, trò chơi máy tính, chương trình dịch, thiết kế hướng đối tượng, ...

Đồ thị vô hướng

(Undirected Graphs)

Định nghĩa. Đơn (đa) đồ thị vô hướng $G = (V, E)$ là cặp gồm:

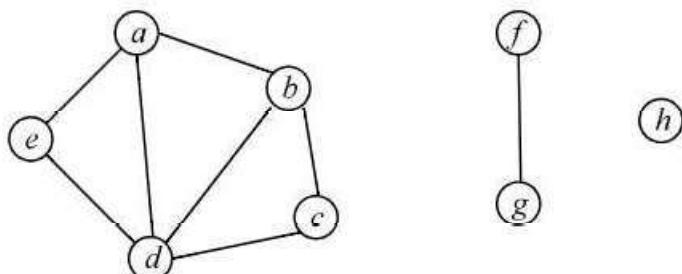
- Tập đỉnh V là tập hữu hạn phân tử, các phân tử gọi là các **đỉnh**
- Tập cạnh E là tập (họ) các bộ không có thứ tự dạng

$$(u, v), u, v \in V, u \neq v$$

Đơn đồ thị vô hướng

(Simple Graph)

- **Ví dụ:** Đơn đồ thị $G_1 = (V_1, E_1)$, trong đó
 $V_1 = \{a, b, c, d, e, f, g, h\}$,
 $E_1 = \{(a, b), (b, c), (c, d), (a, d), (d, e), (a, e), (d, b), (f, g)\}$.



Đồ thị G_1

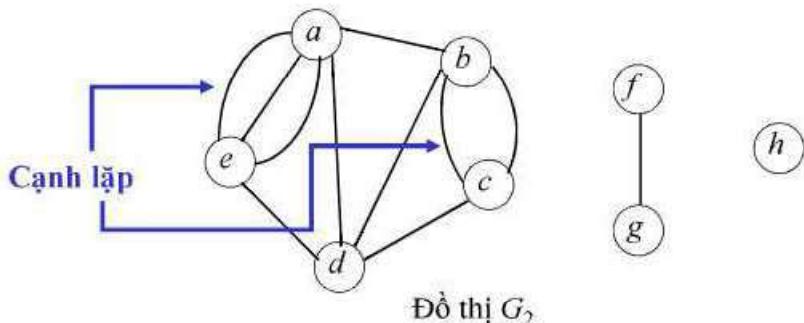
Đa đồ thị vô hướng

(Multi Graphs)

- **Ví dụ:** Đa đồ thị $G_2 = (V_2, E_2)$, trong đó

$$V_2 = \{a, b, c, d, e, f, g, h\},$$

$$E_2 = \{(a,b), (b,c), (b,c), (c,d), (a,d), (d,e), (a,e), (a,e), (d,b), (f,g)\}.$$



Đồ thị có hướng

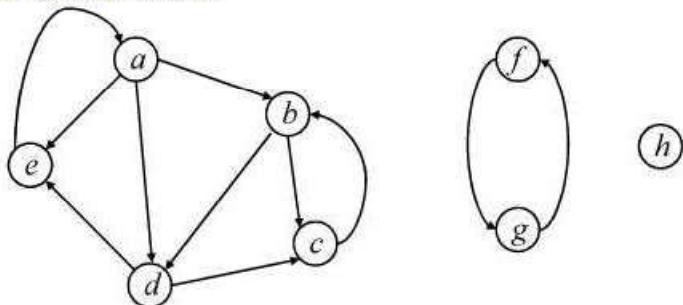
(Directed Graph)

Định nghĩa. Đơn (**đa**) đồ thị có hướng $G = (V, E)$ là cặp gồm:

- Tập đỉnh V là tập hữu hạn phần tử, các phần tử gọi là các **đỉnh**
- Tập cung E là tập (**họ**) các bộ có thứ tự dạng (u, v) , $u, v \in V$, $u \neq v$

Đơn đồ thị có hướng (Simple digraph)

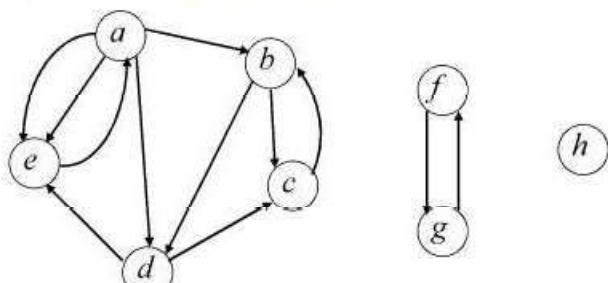
- Ví dụ:** Đơn đồ thị có hướng $G_3 = (V_3, E_3)$, trong đó
 $V_3 = \{a, b, c, d, e, f, g, h\}$,
 $E_3 = \{(a,b), (b,c), (c,b), (d,c), (a,d), (b, d), (a,e), (d,e), (e,a), (f,g), (g,f)\}$



Đồ thị G_3

Đa đồ thị có hướng (Multi Graphs)

- Ví dụ:** Đa đồ thị có hướng $G_4 = (V_4, E_4)$, trong đó
 $V_4 = \{a, b, c, d, e, f, g, h\}$,
 $E_4 = \{(a,b), (b,c), (c,b), (d,c), (a,d), (b, d), (a,e), (a,e), (d,e), (e,a), (f,g), (g,f)\}$



Đồ thị G_4

Các loại đồ thị: Tóm tắt

| Loại | Kiểu cạnh | Có cạnh lặp? |
|---------------------|-----------|--------------|
| Đơn đồ thị vô hướng | Vô hướng | Không |
| Đa đồ thị vô hướng | Vô hướng | Có |
| Đơn đồ thị có hướng | Có hướng | Không |
| Đa đồ thị có hướng | Có hướng | Có |

- Chú ý:

- Một dạng đồ thị ít sử dụng hơn, đó là giả đồ thị. **Giả đồ thị** là đa đồ thị mà trong đó có các **khuyên** (cạnh nối 1 đỉnh với chính nó).
- Cách phân loại đồ thị dùng ở đây chưa chắc đã được chấp nhận trong các tài liệu khác...

Khuyên (loop)



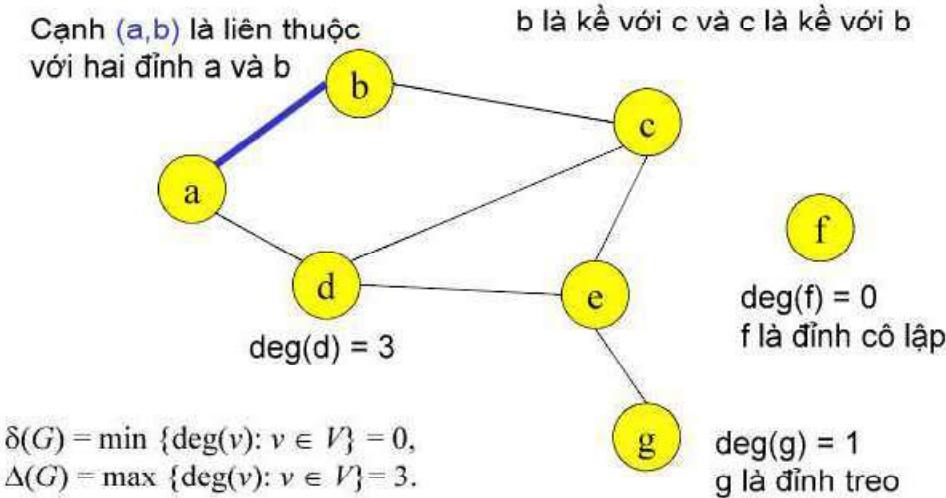
Bậc của đỉnh (Degree of a Vertex)

- Giả sử G là đồ thị vô hướng, $v \in V$ là một đỉnh nào đó.
- Bậc* của đỉnh v , $\deg(v)$, là số cạnh kề với nó.
- Đỉnh bậc 0 được gọi là *đỉnh cô lập* (*isolated*).
- Đỉnh bậc 1 được gọi là *đỉnh treo* (*pendant*).
- Các ký hiệu thường dùng:

$$\delta(G) = \min \{\deg(v): v \in V\},$$

$$\Delta(G) = \max \{\deg(v): v \in V\}.$$

Ví dụ



Định lý về các cái bắt tay (Handshaking Theorem)

- **Định lý.** Giả sử G là đồ thị vô hướng (đơn hoặc đa) với tập đỉnh V và tập cạnh E . Khi đó

$$\sum_{v \in V} \deg(v) = 2|E|$$

- **Hệ quả:** Trong một đồ thị vô hướng bất kỳ, số lượng đỉnh bậc lẻ (đỉnh có bậc là số lẻ) bao giờ cũng là số chẵn.

Ví dụ.

Biết rằng mỗi đỉnh của đồ thị vô hướng $G=(V,E)$ với 14 đỉnh và 25 cạnh đều có bậc là 3 hoặc 5.

Hỏi G có bao nhiêu đỉnh bậc 3?

Giải. Giả sử G có x đỉnh bậc 3.

Khi đó có $14-x$ đỉnh bậc 5.

Do $|E| = 25$, nên tổng tất cả các bậc là 50.

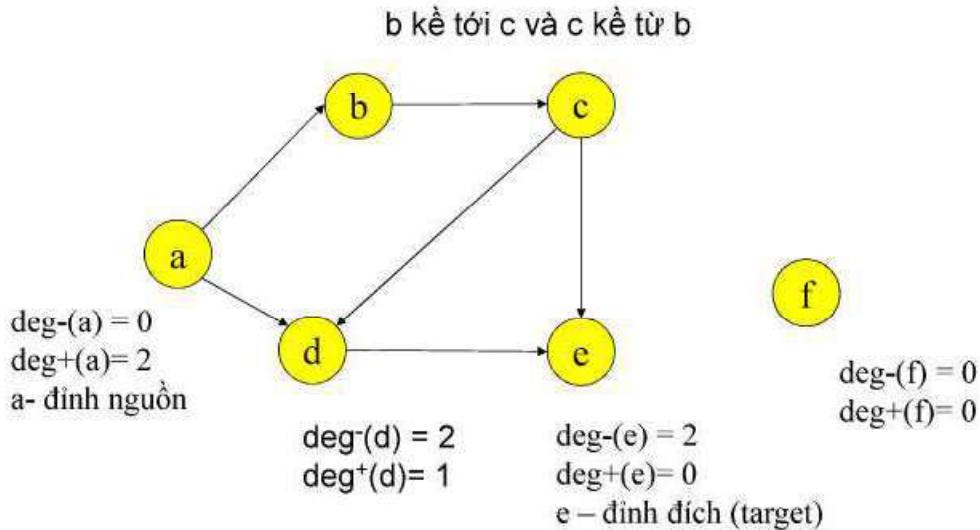
Từ đó, $3x + 5(14-x) = 50$

Suy ra $x = 10$.

Bậc của đỉnh của đồ thị có hướng

- Cho G là đồ thị có hướng, v là đỉnh của G .
 - *Bán bậc vào (in-degree)* của v , $\deg^-(v)$, là số cạnh đi vào v .
 - *Bán bậc ra (out-degree)* của v , $\deg^+(v)$, là số cạnh đi ra khỏi v .
 - *Bậc* của v , $\deg(v) \equiv \deg^-(v) + \deg^+(v)$, là tổng của bán bậc vào và bán bậc ra của v .

Ví dụ



Định lý về các cái bắt tay có hướng

Directed Handshaking Theorem

- **Định lý.** Giả sử G là đồ thị có hướng (có thể là đơn hoặc đa) với tập đỉnh V và tập cạnh E . Khi đó:

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = \frac{1}{2} \sum_{v \in V} \deg(v) = |E|$$

- Chú ý là khái niệm bậc của đỉnh là không thay đổi cho dù ta xét đồ thị vô hướng hay có hướng.

Đường đi, Chu trình

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

$P: \quad x_0, x_1, \dots, x_{n-1}, x_n$
trong đó $u = x_0$, $v = x_n$, $(x_i, x_{i+1}) \in E$, $i = 0, 1, 2, \dots, n-1$.
Đường đi nói trên còn có thể biểu diễn dưới dạng dây các cạnh:

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

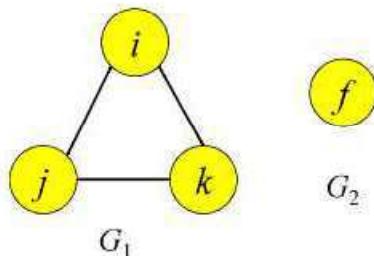
Đỉnh u gọi là **đỉnh đầu**, còn đỉnh v gọi là **đỉnh cuối** của đường đi.

Đường đi, Chu trình

- Đường đi gọi là **đường đi đơn** nếu không có đỉnh nào bị lặp lại trên nó.
- Đường đi gọi là **đường đi cơ bản** nếu không có cạnh nào bị lặp lại trên nó.
- Nếu có đường đi từ u đến v thì ta nói đỉnh v **đạt đến được** từ đỉnh u. Ta quan niệm rằng một đỉnh v luôn đạt đến được từ chính nó.
- Đường đi cơ bản có đỉnh đầu trùng với đỉnh cuối (tức là $u = v$) được gọi là **chu trình**.

Tính liên thông (Connectedness)

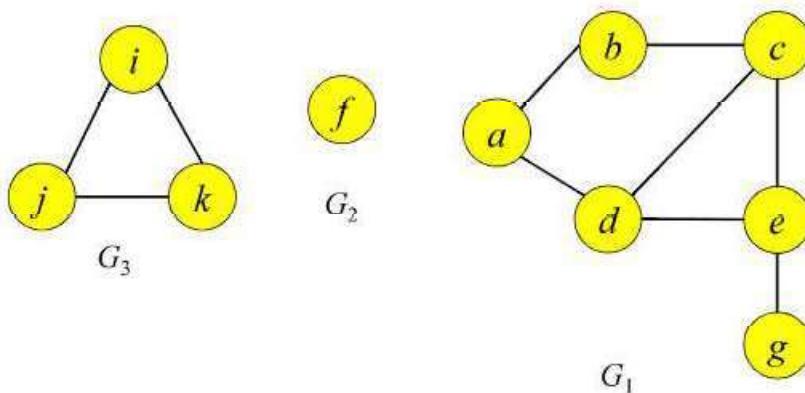
- Đồ thị vô hướng được gọi là *liên thông* nếu luôn tìm được đường đi nối hai đỉnh bất kỳ của nó.
- Ví dụ**



- G_1 và G_2 là các đồ thị liên thông
- Đồ thị G bao gồm G_1 và G_2 không là đồ thị liên thông

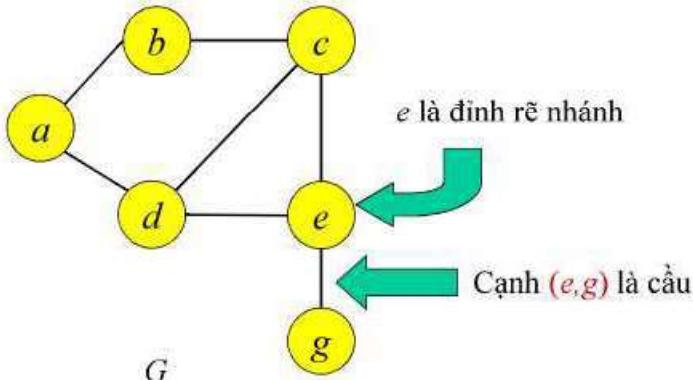
Tính liên thông (Connectedness)

- Thành phần liên thông (Connected component):* Đồ thị con liên thông cực đại của đồ thị vô hướng G được gọi là thành phần liên thông của nó.
- Ví dụ:** Đồ thị G có 3 thành phần liên thông G_1 , G_2 , G_3



Định rõ nhánh và cầu (Connectedness)

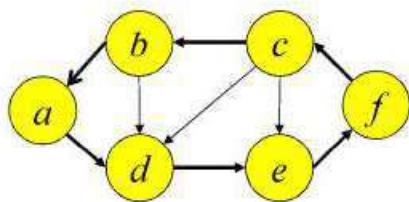
- *Định rõ nhánh (cut vertex):* là đỉnh mà việc loại bỏ nó làm tăng số thành phần liên thông của đồ thị
- *Cầu (bridge):* Cạnh mà việc loại bỏ nó làm tăng số thành phần liên thông của đồ thị .
- **Ví dụ:**



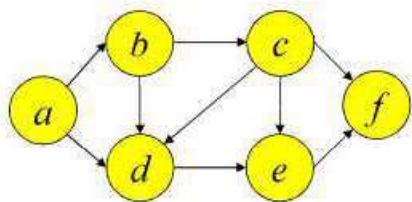
Tính liên thông của Đồ thị có hướng

- Đồ thị có hướng được gọi là *liên thông mạnh (strongly connected)* nếu như luôn tìm được đường đi nối hai đỉnh bất kỳ của nó.
- Đồ thị có hướng được gọi là *liên thông yếu (weakly connected)* nếu như đồ thị vô hướng thu được từ nó bởi việc bỏ qua hướng của tất cả các cạnh của nó là đồ thị vô hướng liên thông.
- Để thấy là nếu G là liên thông mạnh thì nó cũng là liên thông yếu, nhưng điều ngược lại không luôn đúng.

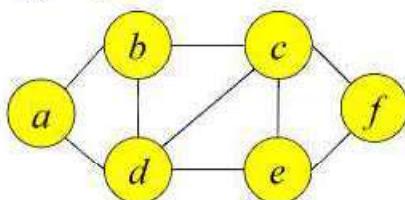
Ví dụ



- Đồ thị liên thông mạnh



- Đồ thị liên thông yếu

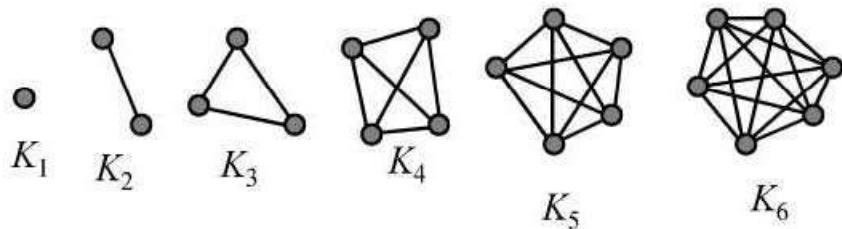


Một số dạng đồ thị đặc biệt-đồ thị phẳng

- Đồ thị đầy đủ (Complete graphs) K_n
- Đồ thị hai phía (Bipartite graphs)
- Đồ thị phẳng

Đồ thị đầy đủ Complete Graphs

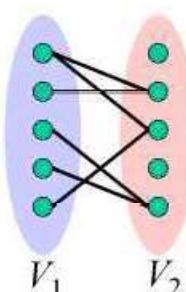
- Với $n \in \mathbb{N}$, đồ thị đầy đủ n đỉnh, K_n , là đơn đồ thị vô hướng với n đỉnh trong đó giữa hai đỉnh bất kỳ luôn có cạnh nối: $\forall u, v \in V: u \neq v \Leftrightarrow (u, v) \in E$.



Để ý là K_n có $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ cạnh.

Đồ thị hai phía (Bipartite Graphs)

- Định nghĩa.** Đồ thị $G=(V,E)$ là hai phía nếu và chỉ nếu $V = V_1 \cup V_2$ với $V_1 \cap V_2 = \emptyset$ và $\forall e \in E: \exists v_1 \in V_1, v_2 \in V_2: e = (v_1, v_2)$.
- Bằng lời:** Có thể phân hoạch tập đỉnh thành hai tập sao cho mỗi cạnh nối hai đỉnh thuộc hai tập khác nhau.

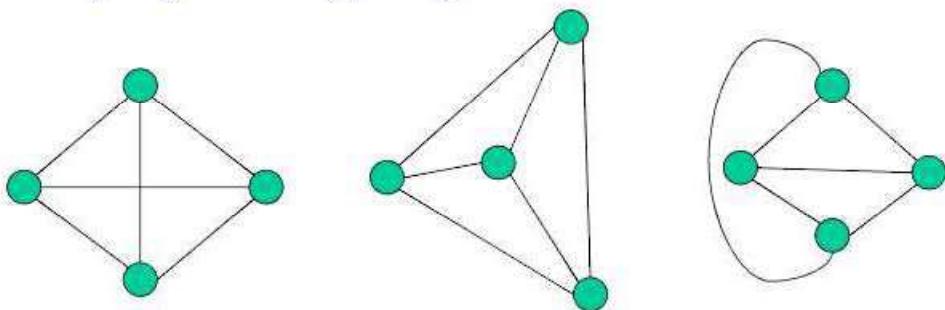


Định nghĩa này là chung cho cả đơn lẩn đa đồ thị vô hướng, có hướng.

Đồ thị phẳng

(Planar Graphs)

- **Định nghĩa.** Đồ thị vô hướng G được gọi là đồ thị phẳng nếu như có thể vẽ nó trên mặt phẳng sao cho không có hai cạnh nào cắt nhau ngoài ở đỉnh.
- **Ví dụ:** K_4 là đồ thị phẳng?



K_4 là đồ thị phẳng!

Công thức Euler

Euler's Formula

- Nếu G là đồ thị phẳng, thì mọi cách vẽ phẳng G đều chia mặt phẳng ra thành các vùng mà ta sẽ gọi là các **diện** (faces).
- Một trong các diện này là không bị chặn và nó được gọi là **diện vô hạn**.
- **Theorem (Euler's Formula)** Let G be a connected planar graph, and let n , m and f denote, respectively, the numbers of vertices, edges, and faces in a plane drawing of G . Then $n - m + f = 2$.



Leonhard Euler

1707-1783

Bài toán tô màu

Bài toán tô màu:

Cho n thành phố, hãy tô màu các thành phố này sao cho không có bất kỳ hai thành phố nào kề nhau được tô cùng một màu và số màu được tô là ít nhất có thể.

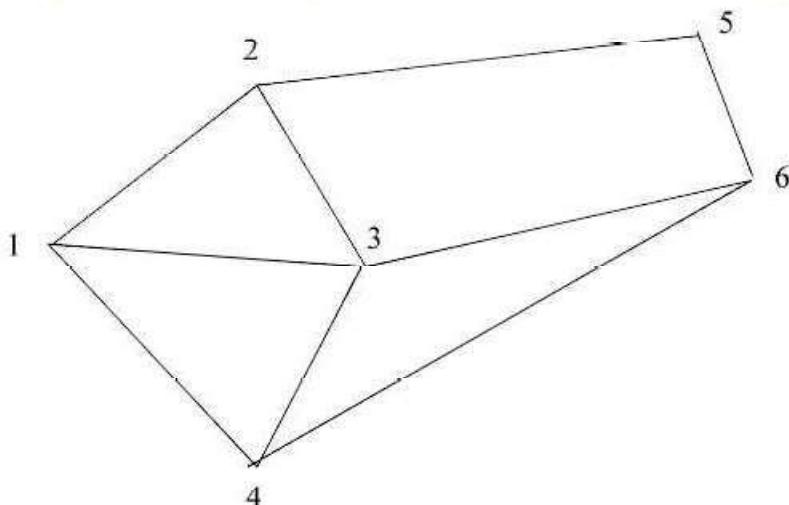
Định nghĩa: Giả sử c là số nguyên dương. Đơn đồ thị vô hướng được gọi là *tô được bởi c màu* nếu có thể tô các đỉnh của nó bởi c màu sao cho không có hai đỉnh kề nhau nào bị tô bởi cùng một màu.

Sắc số (**chromatic number**) của đồ thị G , ký hiệu bởi $\chi(G)$, là số c nhỏ nhất sao cho có thể tô được G bởi c màu.

Định lý. Mọi đồ thị phẳng đều tô được tối đa bởi 4 màu.

Ví dụ

Hãy tô màu đồ thị sau, cho biết sắc số tương ứng



Một thuật toán tham lam

(thuật toán Welch-Powell)

- *Bước 1:Sắp xếp các đỉnh theo bậc giảm dần.*
- *Bước 2:Dùng màu thứ nhất tô cho đỉnh có bậc cao nhất và các đỉnh khác có thể tô còn lại.*
- *Bước 3:Dùng màu thứ hai tô cho đỉnh có bậc cao thứ nhất (còn lại) và các đỉnh khác có thể tô còn lại*
- *Bước 4:Và cứ như thế... cho đến khi tất cả các đỉnh được tô màu hết*
- **Chú ý:** Kết quả thực hiện thuật toán là phụ thuộc vào trình tự sắp xếp các đỉnh của đồ thị.

BÀI TẬP

1-1.Vẽ các đồ thị vô hướng (đơn hoặc đa đồ thị):

- a. Vẽ một đồ thị có 4 đỉnh với bậc các đỉnh là 3, 2, 2, 1.
- b. Vẽ các đồ thị mà mọi đỉnh của nó đều có bậc là lần lượt là k ($1 \leq k \leq 5$)
- c. Vẽ các đồ thị mà mọi đỉnh của nó đều có bậc là 3 và có số đỉnh lần lượt là:4,6,8.

1-2.Vẽ đồ thị phẳng liên thông (đơn hoặc đa đồ thị):

- a. có 6 cạnh và 3 miền.
- b. có 4 đỉnh và 5 miền.
- c. có 6 đỉnh và 7 cạnh.

1-3.

- a. Một đồ thị phẳng liên thông có 8 đỉnh, các đỉnh lân lượt có bậc là 2, 2, 3, 3, 3, 3, 4, 6. Hỏi đồ thị có bao nhiêu cạnh ?
- b. Một đồ thị phẳng liên thông có 10 mặt, tất cả các đỉnh đều có bậc 4. Tìm số đỉnh của đồ thị.
- c. Xét một đơn đồ thị liên thông có 8 đỉnh bậc 3. Hỏi biểu diễn phẳng của đồ thị này sẽ chia mặt phẳng thành mấy miền.
- d. Đồ thị phẳng liên thông G có 9 đỉnh, bậc các đỉnh là 2,2,2,3,3,3,4,4,5. Tìm số cạnh và số mặt của G.
- e. Cho G là một đồ thị phẳng liên thông gồm 16 đỉnh; trong đó có 4 đỉnh bậc 2, 6 đỉnh bậc 4, 4 đỉnh bậc 5 và 2 đỉnh bậc 7. Hỏi đồ thị G có bao nhiêu cạnh và biểu diễn phẳng của đồ thị G sẽ chia mặt phẳng làm bao nhiêu phần ?
- f. Giả sử G là một đồ thị vô hướng có 22 cạnh. Tính số đỉnh của G biết đồ thị có 5 đỉnh bậc 4 và các đỉnh còn lại đều có bậc 3.
- g. Giả sử G là một đồ thị vô hướng có 22 cạnh. Tính số đỉnh nhiều nhất mà G có thể có nếu tất cả các đỉnh của G đều có bậc lớn hơn hoặc bằng 3
- h. Trong một đồ thị phẳng liên thông luôn tồn tại ít nhất một đỉnh có bậc không vượt quá 5.

1-4.

- a. Một đồ thị có 19 cạnh và mỗi đỉnh đều có bậc ≥ 3 , hỏi đồ thị này có tối đa bao nhiêu đỉnh?
- b. Cho một đồ thị vô hướng có n đỉnh. Hỏi đồ thị này có thể có tối đa bao nhiêu cạnh. Trong trường hợp số cạnh là tối đa thì mỗi đỉnh sẽ có bậc là bao nhiêu ?
- c. Chứng minh rằng trong một đơn đồ thị vô hướng nếu không chứa chu trình thì sẽ luôn tồn tại ít nhất là hai đỉnh treo.

1-5.

-
- a. Xét đồ thị vô hướng đơn có số đỉnh $n \geq 2$. Chứng minh rằng đồ thị có ít nhất 2 đỉnh cùng bậc với nhau.
 - b. Cho 1 đồ thị G có chứa đúng 2 đỉnh bậc lẻ (các đỉnh khác nếu có phải bậc chẵn). Chứng minh rằng 2 đỉnh này liên thông với nhau.
 - c. Xét đồ thị vô hướng đơn có số đỉnh $n \geq 2$. Giả sử đồ thị không có đỉnh nào có bậc nhỏ hơn $(n-1)/2$. Chứng minh rằng đồ thị này liên thông.
 - d. Đồ thị G có n đỉnh, 97 cạnh, mọi đỉnh đều có bậc p. Nếu $p > 2$ thì G có liên thông không?
 - e. Đồ thị G có n đỉnh, 323 cạnh, mọi đỉnh đều có bậc p. Nếu $p > 38$ thì G có liên thông không?
 - f. Một đồ thị đơn có 10 đỉnh, 37 cạnh thì có liên thông hay không?

1-6.

Giả sử có 6 cuộc mitting A,B,C,D,E,F cần được tổ chức. Mỗi cuộc mitting được tổ chức trong một buổi. Các cuộc mitting sau không được diễn ra đồng thời: BEF, CEF, ABE, CD, AD.

Hãy bố trí các cuộc mitting vào các buổi sao cho số buổi diễn ra là ít nhất.

Chương 2.

BIỂU DIỄN ĐỒ THỊ

Representations of Graphs

(Từ chương này ta chỉ xét đơn đồ thị; nếu là đa đồ thị thì sẽ nói rõ)

Biểu diễn đồ thị

- Có nhiều cách biểu diễn đồ thị. Việc lựa chọn cách biểu diễn phụ thuộc vào từng bài toán cụ thể, thuật toán cụ thể cần cài đặt.
- Có hai vấn đề chính cần quan tâm khi lựa chọn cách biểu diễn:
 - Bộ nhớ mà cách biểu diễn đó đòi hỏi
 - Thời gian cần thiết để trả lời các truy vấn thường xuyên đối với đồ thị trong quá trình xử lý đồ thị:
 - **Chẳng hạn:**
 - Có cạnh nối hai đỉnh u, v ?
 - Liệt kê các đỉnh kề của đỉnh v ?

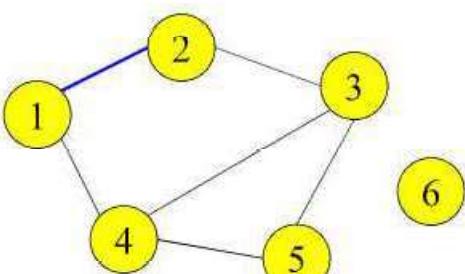
Ma trận kè (Adjacency Matrix)

- $|V| \times |V|$ ma trận A .
- Các đỉnh được đánh số từ 1 đến $|V|$ theo 1 thứ tự nào đó.
- A xác định bởi:

$$A[i, j] = a_{ij} = \begin{cases} 1 & \text{nếu } (i, j) \in E \\ 0 & \text{nếu trái lại} \end{cases}$$

- $n = |V|$; $m = |E|$

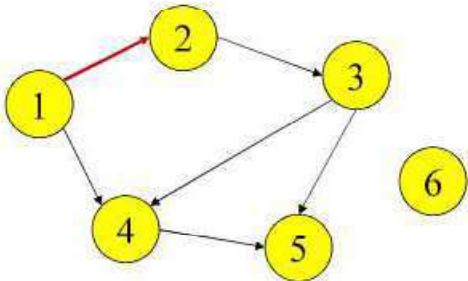
Ma trận kè của đồ thị vô hướng



$$A[u, v] = \begin{cases} 1 & \text{nếu } (u, v) \in E \\ 0 & \text{nếu trái lại} \end{cases}$$

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 1 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 1 | 0 | 1 | 0 |
| 5 | 0 | 0 | 1 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 |

Ma trận kè của đồ thị có hướng



$$A[u,v] = \begin{cases} 1 & \text{nếu } (u,v) \in E \\ 0 & \text{nếu trái lại} \end{cases}$$

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 |

Tính chất của ma trận kè

– Gọi A là ma trận kè của đồ thị vô hướng:

- A là ma trận đối xứng: $a_{ij} = a_{ji}$
- $\deg(v) = \text{Tổng các phần tử trên dòng } v \text{ của } A$

Phân tích chi phí

- Bộ nhớ (Space)
 - $|V|^2$ bits
- Thời gian trả lời các truy vấn
 - Hai đỉnh i và j có kề nhau? $O(1)$
 - Bổ sung hoặc loại bỏ cạnh $O(1)$
 - Bổ sung đỉnh: tăng kích thước ma trận
 - Liệt kê các đỉnh kề của v $O(|V|)$ (ngay cả khi v là đỉnh cô lập).

Ma trận trọng số

- Trong trường hợp đồ thị có trọng số trên cạnh, thay vì ma trận kề để biểu diễn đồ thị ta sử dụng **ma trận trọng số**

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

với

$$c[i, j] = \begin{cases} c(i, j), & \text{nếu } (i, j) \in E \\ \theta, & \text{nếu } (i, j) \notin E, \end{cases}$$

trong đó θ là giá trị đặc biệt để chỉ ra một cặp (i, j) không là cạnh, tùy từng trường hợp cụ thể, có thể được đặt bằng một trong các giá trị sau: $0, +\infty, -\infty$ (nhưng thường sử dụng giá trị 0).

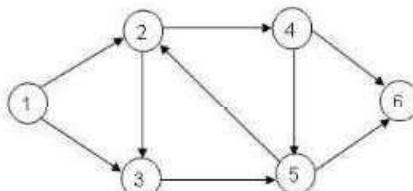
Ma trận liên thuộc đỉnh cạnh

- Xét $G = (V, E)$, ($V = \{1, 2, \dots, n\}$, $E = \{e_1, e_2, \dots, e_m\}$), là đồ thị có hướng.
- Ma trận liên thuộc đỉnh cạnh $A = (a_{ij} : i = 1, 2, \dots, n, j = 1, 2, \dots, m)$, với

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

- Ma trận 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.

Ví dụ



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

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \end{bmatrix}$$

Danh sách cạnh (cung)

6

1 2 // hai đỉnh kề với cạnh e_{12}

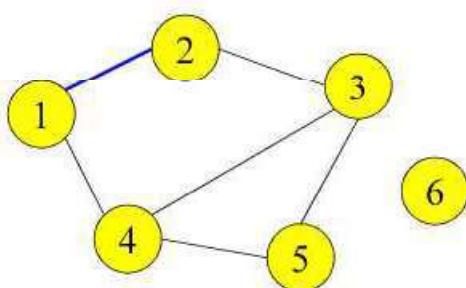
1 4

2 3

3 4

3 5

4 5



Lưu ý: Nếu là đồ thị có trọng số thì thêm cột trọng số

Danh sách cạnh(cung)

6

1 2// đỉnh đầu, đỉnh cuối

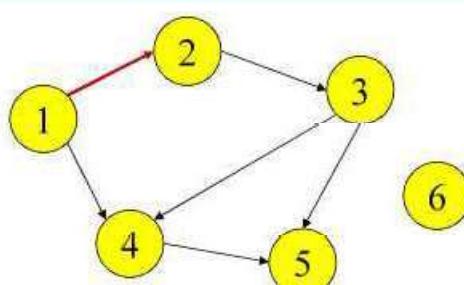
1 4

2 3

3 4

3 5

4 5



• Lưu ý: Nếu là đồ thị có trọng số thì thêm cột trọng số

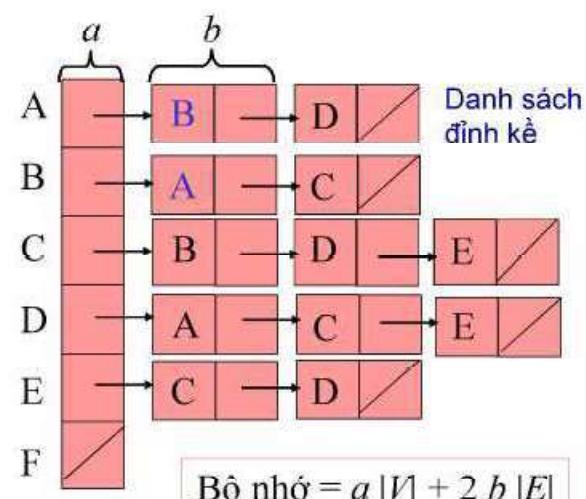
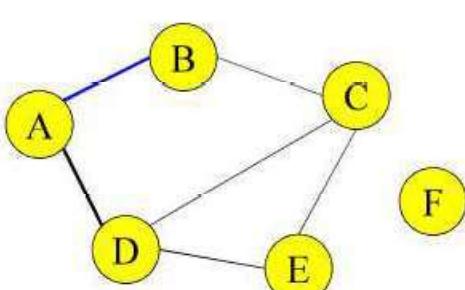
• Cách biểu diễn này phù hợp với đồ thị thưa.

Danh sách kè

- **Danh sách kè** (Adjacency Lists): Với mỗi đỉnh v cất giữ danh sách các đỉnh kè của nó.
 - Mỗi đỉnh có một danh sách: Với mỗi $u \in V$, $Ke[u]$ bao gồm tất cả các đỉnh kè của u .

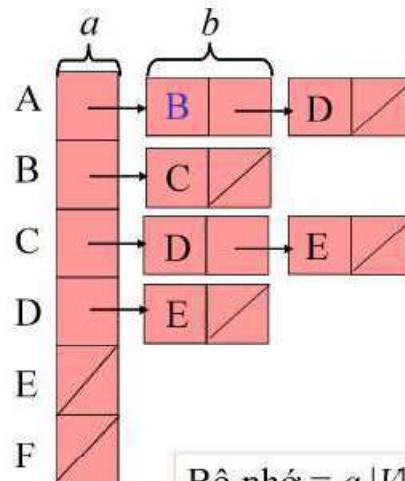
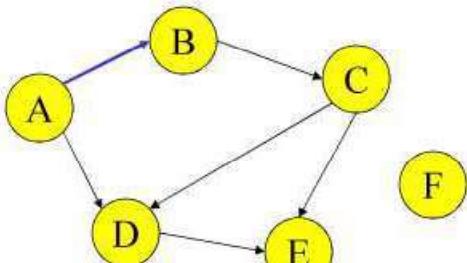
Danh sách kè của đồ thị vô hướng

Với mỗi $v \in V$, $Ke(v)$ = danh sách các đỉnh u : $(v, u) \in E$



Danh sách kè của đồ thị có hướng

Với mỗi $v \in V$, $\text{Ke}(v) = \{ u : (v, u) \in E \}$



Yêu cầu bộ nhớ

- Tổng cộng bộ nhớ: $\Theta(|V|+|E|)$
- Thường là nhỏ hơn nhiều so với $|V|^2$, nhất là đối với đồ thị thưa (sparse graph).
- Đồ thị thưa là đồ thị mà $|E| = k |V|$ với $k < 10$.
- **Chú ý:**
 - Phần lớn các đồ thị trong thực tế ứng dụng là đồ thị thưa!
 - Cách biểu diễn này được sử dụng nhiều nhất trong ứng dụng

- Thời gian trả lời các truy vấn:
 - Thêm cạnh $O(1)$
 - Xoá cạnh Duyệt qua danh sách kè của mỗi đầu mút.
 - Thêm đỉnh Phụ thuộc vào cài đặt.
 - Liệt kê các đỉnh kè của v : $O(<\text{số đỉnh kè}>)$ (tốt hơn ma trận kè)
 - Hai đỉnh i, j có kè nhau?
 - Tìm kiếm trên danh sách: $\Theta(\text{degree}(i))$. Đánh giá trong tình huống tồi nhất là $O(|V|) \Rightarrow$ không hiệu quả (tốt hơn ma trận kè)

Bài tập

2-1. Chuyển đổi cấu trúc dữ liệu biểu diễn đồ thị dưới dạng ma trận trọng số qua dạng danh sách cạnh và ngược lại.

2-2. Chuyển đổi cấu trúc dữ liệu biểu diễn đồ thị dưới dạng ma trận kè qua dạng danh sách kè và ngược lại.

2-3. Cho một đơn đồ thị. Hãy viết các hàm thực hiện các yêu cầu sau:

a. Đồ thị là có hướng hay vô hướng ?

b. Tính bậc của mỗi đỉnh.

Chương 3.

Các thuật toán duyệt đồ thị

(Graph Searching, Graph Traversal)

(Chương này đề cập đến đồ thị vô hướng)

Các thuật toán duyệt đồ thị

- Duyệt đồ thị: Graph Searching hoặc Graph Traversal
 - Duyệt qua mỗi đỉnh và mỗi cạnh của đồ thị
- Ứng dụng:
 - Cần để khảo sát các tính chất của đồ thị
 - Là thành phần cơ bản của nhiều thuật toán trên đồ thị
- Hai thuật toán duyệt cơ bản:
 - Tìm kiếm theo chiều rộng (Breadth First Search – BFS)
 - Tìm kiếm theo chiều sâu (Depth First Search – DFS)

Tìm kiếm theo chiều rộng

Breadth-first Search (BFS)

Ý tưởng

- Ta sẽ bắt đầu tìm kiếm từ một đỉnh v_o nào đó của đồ thị; sau đó duyệt qua tất cả các đỉnh kề với đỉnh v_o ; chọn u là đỉnh đầu tiên kề với v_o và lặp lại quá trình tìm kiếm.
- Ở bước tổng quát, giả sử ta đang xét đỉnh v . thì ta duyệt hết tất cả các đỉnh kề với đỉnh v .
- Còn nếu như không còn đỉnh nào kề với v là chưa xét thì ta nói đỉnh này đã duyệt xong và quay trở lại tiếp tục tìm kiếm với đỉnh được duyệt sau khi duyệt đến đỉnh v .

```

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_dịnh(p);
      for u ∈ Ke(p) do
        if Chuaxet[u] then
          begin
            QUEUE ← u; Chuaxet[u]:=false;
          end;
      end;
    end;

```

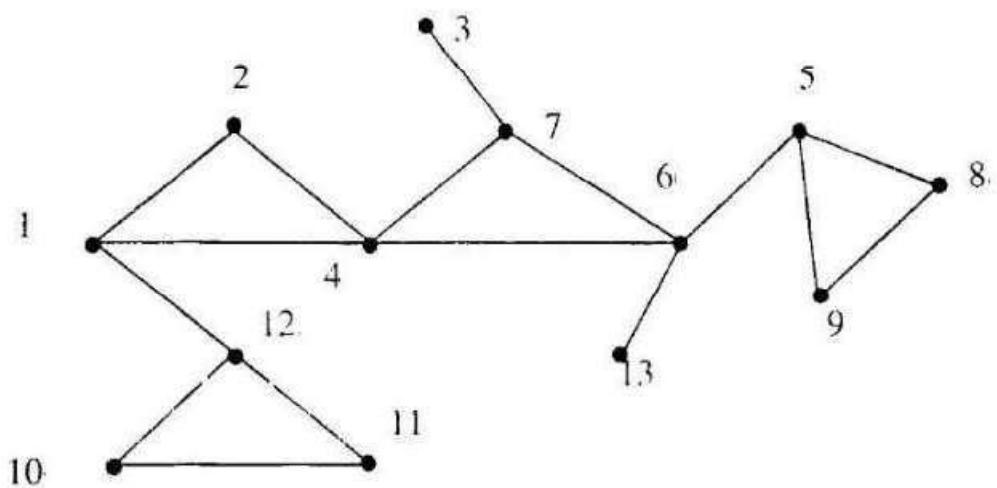
Khi đó, Tìm kiếm theo chiều rộng trên đồ thị được thực hiện nhờ thuật toán sau:

```

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

```

Ví dụ (BFS)



Phân tích BFS

- Việc khởi tạo đòn hỏi $O(|V|)$.
- Vòng lặp duyệt
 - Mỗi đỉnh được nạp vào và loại ra khỏi hàng đợi một lần, mỗi thao tác đòn hỏi thời gian $O(1)$. Như vậy tổng thời gian làm việc với hàng đợi là $O(V)$.
 - Danh sách kè của mỗi đỉnh được duyệt qua đúng một lần. Tổng độ dài của tất cả các danh sách kè là $\mathcal{O}(|E|)$.
- Tổng cộng ta có thời gian tính của $\text{BFS}(s)$ là $O(|V| + |E|)$, là tuyến tính theo kích thước của danh sách kè biểu diễn đồ thị.

CÁC ỨNG DỤNG CỦA BFS

- Sử dụng BFS để kiểm tra tính liên thông của đồ thị vô hướng:
 - Mỗi lần gọi đến BFS ở trong chương trình chính sẽ sinh ra một thành phần liên thông
- Xét sự tồn tại đường đi từ đỉnh s đến đỉnh t :
 - Thực hiện BFS(s).
 - Nếu $\pi[t] = \text{NIL}$ thì không có đường đi, trái lại ta có đường đi $t \leftarrow \pi[t] \leftarrow \pi[\pi[t]] \leftarrow \dots \leftarrow s$
- *Chú ý: BFS tìm được đường đi ngắn nhất theo số cạnh.
- Cây BFS

Tìm kiếm theo chiều sâu

Depth-first Search (DFS)

Ý tưởng của tìm kiếm theo chiều sâu

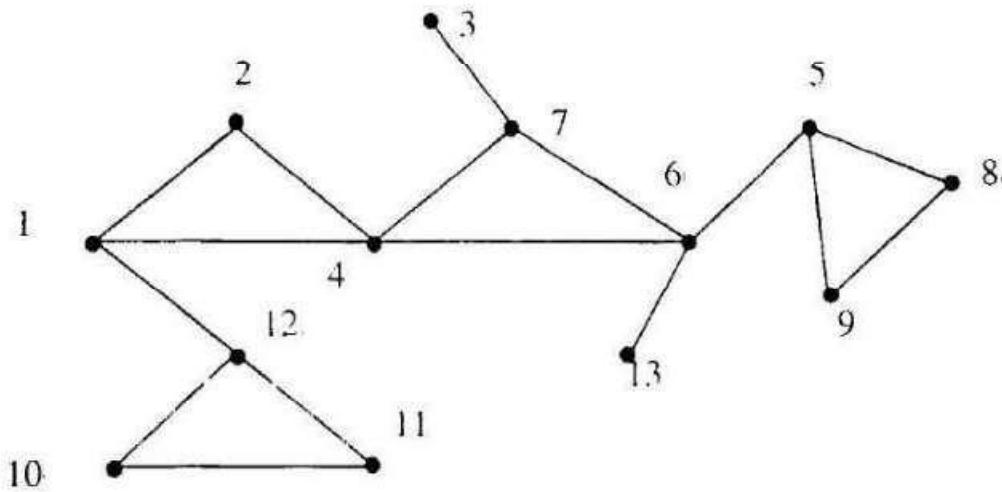
- Ta sẽ bắt đầu tìm kiếm từ một đỉnh v_0 nào đó của đồ thị; sau đó chọn u là một đỉnh nào đó kề với v_0 và lặp lại quá trình đối với u .
- Ở bước tổng quát, giả sử ta đang xét đỉnh v . Nếu như trong số các đỉnh kề với v ta tìm được đỉnh w là chưa được xét thì ta sẽ xét đỉnh này (w trở thành đỉnh đã xét) 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 nói đỉnh này 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 quá trình tìm kiếm).

```
procedure DFS(v);
(* Tìm kiếm theo chiều sâu bắt đầu từ đỉnh v;
   Các biến Chuaxet, Ke là biến toàn cục *)
begin
    Tham_dinh(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.
```

Ví dụ (DFS)



Phân tích thuật toán DFS

- Mỗi đỉnh được thăm đúng 1 lần, việc thăm mỗi đỉnh đòi hỏi chi phí thời gian $O(1)$, suy ra thao tác thăm đỉnh đòi hỏi thời gian $O(|V|)$.
- Vòng lặp trong $\text{DFS}(u)$ thực hiện việc duyệt cạnh của đồ thị
 - Mỗi cạnh được duyệt qua đúng một lần nếu đồ thị là có hướng và 2 lần nếu đồ thị là vô hướng
 - Như vậy tổng số lần lặp là $O(|E|)$.
- Vậy, thuật toán có thời gian $O(|V|+|E|)$
- Đối với đồ thị, thuật toán có đánh giá như vậy gọi là **thuật toán thời gian tuyến tính**

DFS: Các loại cạnh

- DFS tạo ra một cách phân loại các cạnh của đồ thị đã cho:
 - *Cạnh của cây (Tree edge)*: là cạnh mà theo đó từ một đỉnh ta đến thăm một đỉnh mới (cạnh đi vào đỉnh trắng)
 - *Cạnh ngược (Back edge)*: đi từ con cháu (descendent) đến tổ tiên (ancestor)
 - *Cạnh tới (Forward edge)*: đi từ tổ tiên đến con cháu
 - *Cạnh vòng (Cross edge)*: cạnh nối hai đỉnh không có quan hệ họ hàng
 - Không là cạnh của cây, và giống như cạnh vòng cũng
 - Đi từ đỉnh xám đến đỉnh đen

Phân tích DFS

- Do đó thời gian của DFS là $\Theta(|V| + |E|)$.
- Thuật toán trên đồ thị có đánh giá thời gian như trên gọi là thuật toán thời gian tuyến tính

CÁC ỨNG DỤNG CỦA DFS

- Kiểm tra tính liên thông của đồ thị
- Tìm đường đi từ s đến t
- Tìm cây DFS
- Phát hiện chu trình

Bài toán về tính liên thông

- **Bài toán:** Cho đồ thị vô hướng $G = (V,E)$. Hỏi đồ thị gồm bao nhiêu thành phần liên thông, và từng thành phần liên thông gồm các đỉnh nào?
- Giải: Sử dụng DFS (BFS) :
 - Mỗi lần gọi đến DFS (BFS) ở trong chương trình chính sẽ sinh ra một thành phần liên thông

Tìm đường đi

- Bài toán tìm đường đi
 - Input: Đồ thị $G = (V,E)$ xác định bởi danh sách kề và hai đỉnh s, t .
 - Đầu ra: Đường đi từ đỉnh s đến đỉnh t , hoặc khẳng định không tồn tại đường đi từ s đến t .
- Thuật toán: Thực hiện DFS(s) (hoặc BFS(s)).
 - Nếu $\pi[t] = \text{NIL}$ thì không có đường đi, trái lại ta có đường đi $t \leftarrow \pi[t] \leftarrow \pi[\pi[t]] \leftarrow \dots \leftarrow s$



DFS và Chu trình

- **Bài toán:** Cho đồ thị $G=(V,E)$. Hỏi G có chứa chu trình hay không?
- **Định lý:** *Đồ thị G là không chứa chu trình khi và chỉ khi trong quá trình thực hiện DFS ta không phát hiện ra cạnh ngược.*

DFS và chu trình

- **Câu hỏi:** Thời gian tính là bao nhiêu?
- **Trả lời:** Chính là thời gian thực hiện DFS: $O(|V|+|E|)$.
- **Câu hỏi:** Nếu G là đồ thị vô hướng thì có thể đánh giá thời gian tính sát hơn nữa được không?
- **Trả lời:** Thuật toán có thời gian tính $O(|V|)$, bởi vì:
 - Trong một rừng (đồ thị không chứa chu trình) $|E| \leq |V| - 1$
 - Vì vậy nếu đồ thị có $|V|$ cạnh thì chắc chắn nó chứa chu trình, và thuật toán kết thúc.

Kiểm tra tính liên thông mạnh

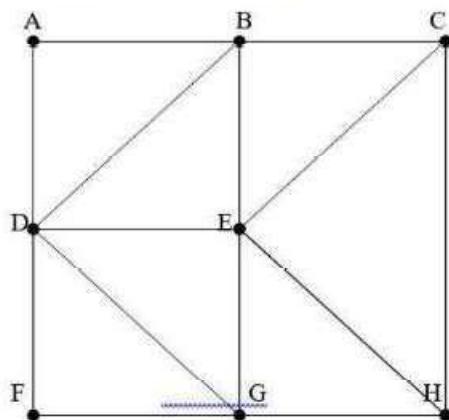
- **Bài toán:** Hỏi đồ thị có hướng G có là liên thông mạnh?
- **Mệnh đề:** *Đồ thị có hướng $G=(V,E)$ là liên thông mạnh khi và chỉ khi luôn tìm được đường đi từ một đỉnh v đến tất cả các đỉnh còn lại và luôn tìm được đường đi từ tất cả các đỉnh thuộc $V \setminus \{v\}$ đến v .*

Thuật toán kiểm tra tính liên thông mạnh

- Thuật toán.
 - Chọn $v \in V$ là một đỉnh tùy ý
 - Thực hiện DFS(v) trên G . Nếu tồn tại đỉnh u không được thăm thì G không liên thông mạnh và thuật toán kết thúc. Trái lại thực hiện tiếp
 - Thực hiện DFS(v) trên $G^T = (V, E^T)$, với E^T thu được từ E bởi việc đảo ngược hướng các cung. Nếu tồn tại đỉnh u không được thăm thì G không liên thông mạnh, nếu trái lại G là liên thông mạnh.
- Thời gian tính: $O(|V|+|E|)$

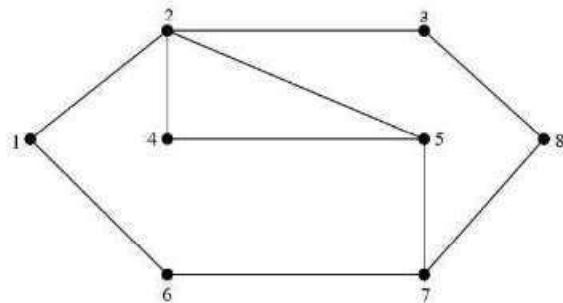
BÀI TẬP

3-1. Hãy liệt kê các đỉnh của đồ thị được duyệt theo phương pháp tìm kiếm theo chiều sâu, tìm kiếm theo chiều rộng. Tìm đường đi từ đỉnh A đến đỉnh H.



3-2. Cho đồ thị vô hướng liên thông G như hình vẽ bên.

- Hãy liệt kê danh sách các đỉnh của G theo thuật toán tìm kiếm theo chiều sâu (DFS), theo thuật toán tìm kiếm theo chiều rộng (BFS) bắt đầu từ đỉnh 1.
- Hãy tìm một đường đi từ đỉnh 1 đến đỉnh 6 trên G theo thuật toán DFS và từ đỉnh 1 đến đỉnh 7 theo thuật toán BFS.



3-3. Cài đặt các chương trình cho thuật toán duyệt đồ thị DFS, BFS.

3-4. Cài đặt các chương trình cho các ứng dụng của các thuật toán duyệt đồ thị (tìm đường đi giữa hai đỉnh, tìm các thành phần liên thông của đồ thị).

Chương 4. **Đồ thị Euler và Đồ thị Hamilton**

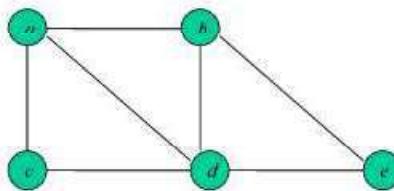
Chương này đề cập đến đơn đồ thị vô hướng/có hướng

Đồ thị Euler

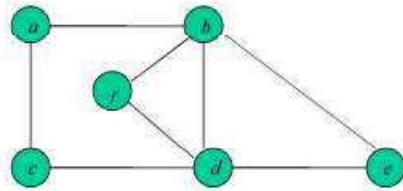
Định nghĩa:

- **Chu trình Euler** trong đồ thị G là chu trình đi qua mỗi cạnh của G đúng một lần.
 - **Đường đi Euler** trong đồ thị G là đường đi qua mỗi cạnh của G đúng một lần.
 - Đồ thị có chu trình Euler được gọi là **đồ thị Euler**.
 - Đồ thị có đường đi Euler được gọi là **đồ thị nửa Euler**.
- ❖ Rõ ràng mọi đồ thị Euler đều là nửa Euler.

Ví dụ



Đồ thị nửa Euler
a, c, d, b, e, d, a, b



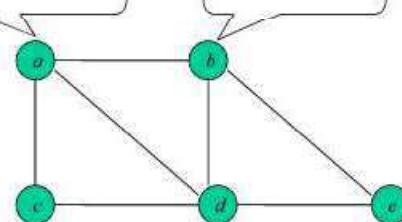
Đồ thị Euler
a, c, d, e, b, d, f, b, a

Định lý Euler

- **Định lý:** Đa đồ thị vô hướng liên thông có chu trình Euler khi và chỉ khi nó không có đỉnh bậc lẻ.
- **Hệ quả:** Đồ thị vô hướng liên thông G là nửa Euler khi và chỉ khi nó có không quá 2 đỉnh bậc lẻ; (và khi đó đường đi Euler có 2 đầu mút tại các đỉnh bậc lẻ đó).

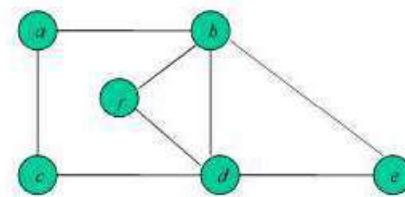
Ví dụ

Định bậc lẻ



Đồ thị nửa Euler

Định bậc lẻ



Đồ thị Euler

Thuật toán Flor tìm chu trình Euler

Xuất phát từ một đỉnh u nào đó của đồ thị G ta đi theo các cạnh của nó một cách tùy ý và chỉ cần tuân thủ 2 nguyên tắc sau :

- Xóa bỏ cạnh đã đi qua và đồng thời xóa cả những đỉnh cô lập tạo thành.
- Ở mỗi bước ta chỉ đi qua cầu khi không còn cách lựa chọn nào khác.

Thuật toán tìm chu trình Euler (thuật toán mở rộng chu trình Euler)

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

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

Thuật toán tìm 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 một 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 một đỉnh chung với chu trình C .

Thuật toán tìm chu trình Euler

Vì vậy, ta có thể xây dựng chu trình Euler trong G như sau:

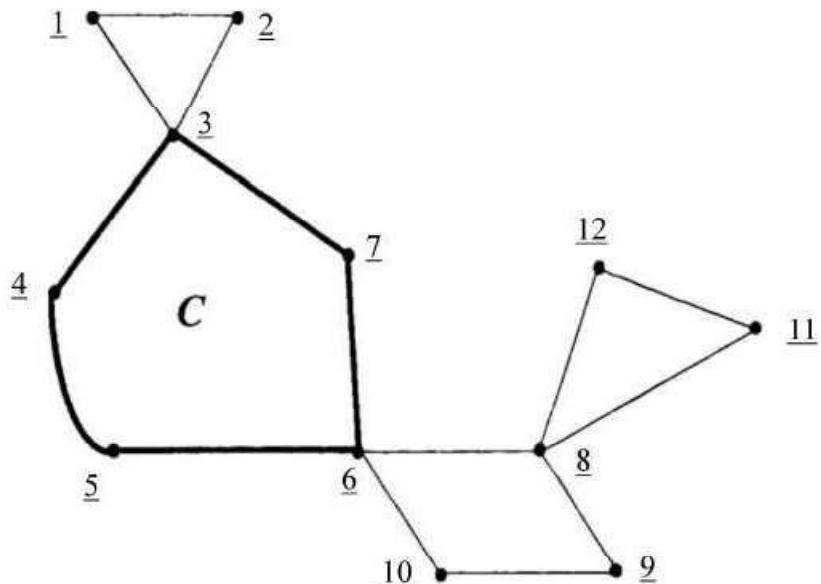
- Bắt đầu từ một đỉnh nào đó của chu trình C ,
- Đi theo các cạnh của chu trình C chừng nào chưa gặp phải đỉnh không cô lập của H . Nếu gặp phải đỉnh như vậy thì ta đi theo chu trình Euler của thành phần liên thông của H chứa đỉnh đó. Sau đó lại tiếp tục đi theo cạnh của C cho đến khi gặp phải đỉnh không cô lập của H thì lại theo chu trình Euler của thành phần liên thông tương ứng trong H
- 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

Thuật toán tìm chu trình Euler

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;
```

Minh họa thuật toán



Định lý

- Đồ thị có hướng liên thông mạnh là đồ thị Euler khi và chỉ khi:

$$\deg^+(v) = \deg(v), \text{ mọi } v \in V.$$

Đồ thị Hamilton

Định nghĩa:

- **Chu trình Hamilton** trong đồ thị G là chu trình đi qua mỗi đỉnh của G đúng một lần.
 - **Đường đi Hamilton** trong đồ thị G là đường đi qua mỗi đỉnh của G đúng một lần.
 - Đồ thị có chu trình Hamilton được gọi là **đồ thị Hamilton**.
 - Đồ thị có đường đi Hamilton được gọi là **đồ thị nửa Hamilton**.
- ❖Rõ ràng mọi đồ thị Hamilton đều là nửa Hamilton

Bài toán chu trình Hamilton

Cho đơn đồ thị vô hướng G , hỏi G có chứa chu trình Hamilton hay không?

Bài toán chu trình Hamilton là bài toán thuộc lớp NP-đầy đủ

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

- **Định lý Dirac:** Nếu G là đơn đồ thị vô hướng liên thông với $n \geq 3$ đỉnh, và $\forall v \deg(v) \geq n/2$, thì G có chu trình Hamilton.
- **Định lý Ore:** Nếu G đơn đồ thị vô hướng liên thông với $n \geq 3$ đỉnh, và $\deg(u) + \deg(v) \geq n$ với mọi cặp đỉnh không kề nhau u, v , thì G có chu trình Hamilton.
- **Định lý:** Giả sử G là đồ thị có hướng liên thông mạnh với n đỉnh. Nếu $\deg^+(v) \geq n/2$ và $\deg^-(v) \geq n/2$ với mọi v thì G là đồ thị Hamilton.

Thuật toán tìm tất cả các chu trình Hamilton

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 dây dinh (X[1], ..., X[k-1])
của đồ thi G=(V, E) cho bối cảnh gách kẽ 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 thi *)
        Chuaxet[v0] := false ;
        Hamilton(2);
    END.
```

Bài tập

4-1.a. Cho đơn đồ thi G. Hãy cài đặt chương trình tìm một chu trình Euler của đồ thi bắt đầu tại một đỉnh u.

b. Cho đơn đồ thi G. Hãy cài đặt chương trình tìm tất cả chu trình Euler của đồ thi bắt đầu tại một đỉnh u.

4-2. Cho đơn đồ thi có trọng số. Cài đặt chương trình tìm một chu trình Hamilton của đồ thi có trọng số có tổng trọng số các cạnh là nhỏ nhất.

Chương 5

Bài toán cây khung nhỏ nhất

The Minimum Spanning Tree Problem

Chương này đề cập đơn đồ thị vô hướng liên thông có trọng số

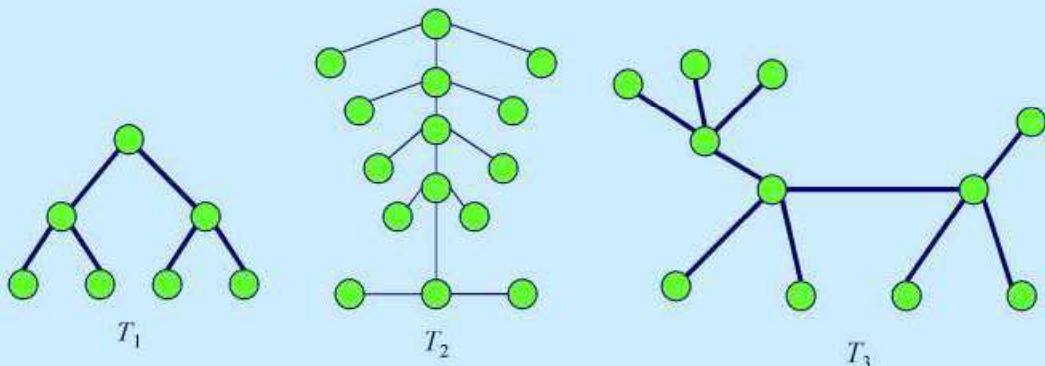
Nội dung

5.1. Cây và các tính chất cơ bản của cây

- 5.2. Cây khung của đồ thị
- 5.3. Bài toán cây khung nhỏ nhất
- 5.4. Thuật toán tìm khung nhỏ nhất

Cây và rừng (Tree and Forest)

- ◆ 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.



Rừng F gồm 3 cây T_1 , T_2 , T_3

Các tính chất cơ bản 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 liên thông và không chứa chu trình;
 - (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 nếu thêm vào nó một cạnh ta thu được đúng một chu trình.

Nội dung

5.1. Cây và các tính chất cơ bản của cây

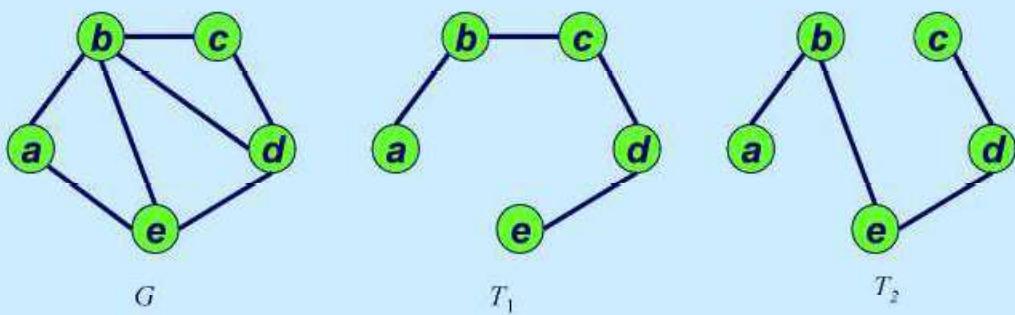
5.2. Cây khung của đồ thị

5.3. Bài toán cây khung nhỏ nhất

5.4. Thuật toán tìm khung nhỏ nhất

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 .



Đồ thị G và 2 cây khung T_1 và T_2 của nó

Số lượng cây khung của đồ thị



Arthur Cayley
(1821 – 1895)

- ◆ **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} .*
- ◆ **Thuật toán tìm cây khung của đồ thị:** BFS, DFS

Nội dung

5.1. Cây và các tính chất cơ bản của cây

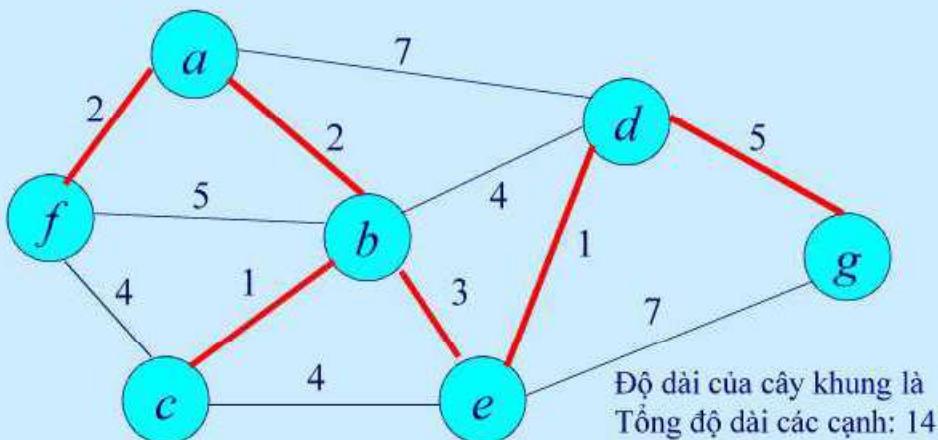
5.2. Cây khung của đồ thị

5.3. Bài toán cây khung nhỏ nhất

5.4. Thuật toán tìm khung nhỏ nhất

Bài toán CKNN

Bài toán: Cho đồ thị vô hướng liên thông $G=(V,E)$ với trọng số $c(e)$, $e \in E$. Độ dài của cây khung là tổng trọng số trên các cạnh của nó. Cần tìm cây khung có độ dài nhỏ nhất.



Bài toán cây khung nhỏ nhất

- ◆ Có thể phát biểu dưới dạng bài toán tối ưu tổ hợp:
Tìm cực tiểu

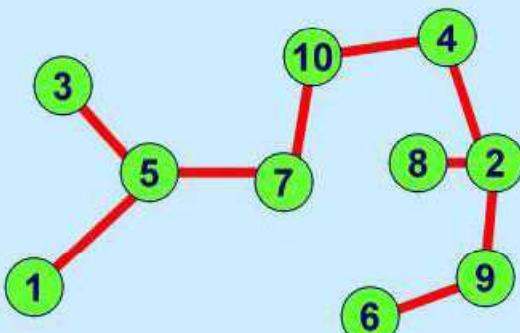
$$c(H) = \sum_{e \in T} c(e) \rightarrow \min,$$

với điều kiện $H=(V,T)$ là cây khung của G .

Do số lượng cây khung của G là rất lớn (xem định lý Cayley), nên không thể giải nhờ duyệt toàn bộ

Ứng dụng thực tế: Mạng truyền thông

- ◆ Công ty truyền thông AT&T cần xây dựng mạng truyền thông kết nối n khách hàng. Chi phí thực hiện kênh nối i và j là c_{ij} . Hỏi chi phí nhỏ nhất để thực hiện việc kết nối tất cả các khách hàng là bao nhiêu?

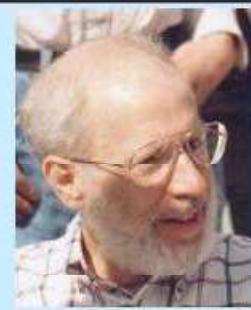


Giả thiết là: Chỉ có cách kết nối duy nhất là đặt kênh nối trực tiếp giữa hai nút.

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 lại giữa hai thành phố bất kỳ đồng thời tổng chi phí xây dựng phải là nhỏ nhất.
- ◆ Rõ ràng là đồ thị mà định 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: 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.

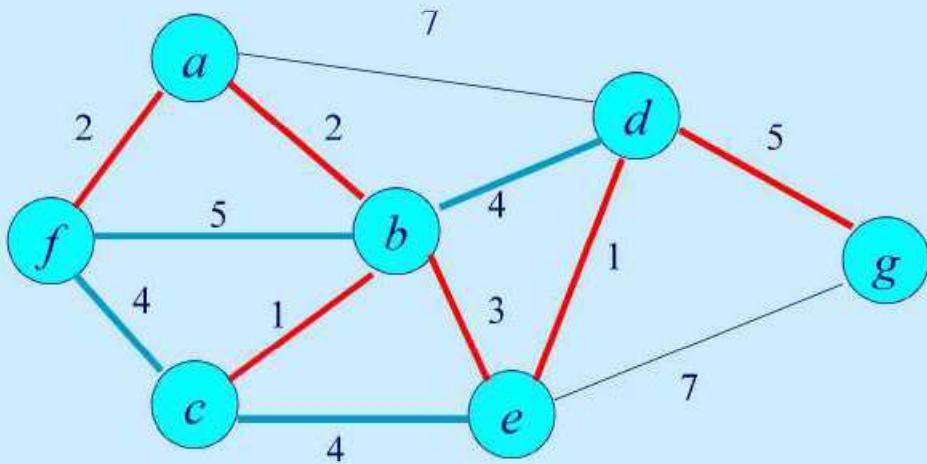
Mô tả thuật toán Kruskal



```
procedure Kruskal;  
begin  
    sắp xếp các cạnh  $e_1, \dots, e_m$  theo thứ tự không giảm của độ dài;  
     $T = \emptyset$ ; (*  $T$  – tập cạnh của CKNN *)  
    for  $i = 1$  to  $m$  do  
        if  $T \cup \{e_i\}$  không chứa chu trình then  $T := T \cup \{e_i\}$ ;  
end
```

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

Thuật toán Kruskal – Ví dụ



Độ dài của CKNN: 14

Thời gian tính

Bước 1. Sắp xếp dãy độ dài cạnh.

$$O(m \log n)$$

Bước lặp: Xác định xem $T \cup \{ e_i \}$ có chứa chu trình hay không?

Có thể sử dụng DFS để kiểm tra với thời gian $O(n)$.

Tổng cộng: $O(m \log n + mn)$

Mô tả thuật toán Prim

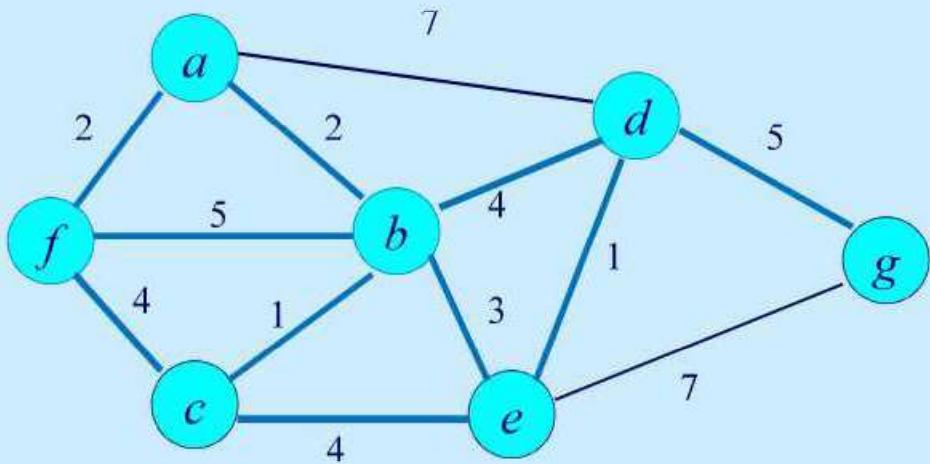


```
procedure Prim;
begin
    Chọn đỉnh tuỳ ý  $r \in V$ ;
    Khởi tạo cây  $T = (V(T), E(T))$  với  $V(T) = \{ r \}$  và  $E(T) = \emptyset$ ;
    while  $|V(T)| < n$  đỉnh do
        begin
            Gọi  $(u, v)$  là cạnh nhỏ nhất với  $u \in V(T)$  và  $v \in V(G) - V(T)$ 
             $E(T) \leftarrow E(T) \cup \{ (u, v) \}$ ;
             $V(T) \leftarrow V(T) \cup \{ v \}$ ;
        end
    end;
```

Thời gian tính: $O(|V|^2)$

```
procedure Prim;
begin
    (* Bước khởi tạo *)
    Chọn  $s$  là một đỉnh nào đó của đồ thị;
     $V_H := \{ s \}$ ;  $T := \emptyset$ ;
     $d[s] := 0$ ;  $near[s] := s$ ;
    for  $v \in V \setminus V_H$  do
        begin
             $d[v] := c[s, v]$ ;
             $near[v] := s$ ;
        end;
    (* Bước lặp *)
    Stop := false;
    while not Stop do
        begin
            Tim  $u \in V \setminus V_H$  thoả mãn:  $d[u] = \min \{ d[v] : v \in V \setminus V_H \}$ ;
             $V_H := V_H \cup \{ u \}$ ;  $T := T \cup \{ (u, near[u]) \}$ ;
            if  $|V_H| = n$  then
                begin
                     $H = (V_H, T)$  là cây khung nhỏ nhất của đồ thị;
                    Stop := true;
                end
            else
                for  $v \in V \setminus V_H$  do
                    if  $d[v] > c[u, v]$  then
                        begin
                             $d[v] := c[u, v]$ ;
                             $near[v] := u$ ;
                        end;
                end;
        end;
end;
```

Thuật toán Prim – Ví dụ



Độ dài của CKNN: 14

Thuật toán Prim – Ví dụ

- Ví dụ: Tìm CKNN cho đồ thị cho bởi ma trận trọng số

| | | 1 | 2 | 3 | 4 | 5 | 6 |
|-----|---|----------|----------|----------|----------|----------|----------|
| C = | 1 | 0 | 33 | 17 | ∞ | ∞ | ∞ |
| | 2 | 33 | 0 | 18 | 20 | ∞ | ∞ |
| | 3 | 17 | 18 | 0 | 16 | 4 | ∞ |
| | 4 | ∞ | 20 | 16 | 0 | 9 | 8 |
| | 5 | ∞ | ∞ | 4 | 9 | 0 | 14 |
| | 6 | ∞ | ∞ | ∞ | 8 | 14 | 0 |

Thuật toán Prim: Ví dụ

| | Đỉnh 1 | Đỉnh 2 | Đỉnh 3 | Đỉnh 4 | Đỉnh 5 | Đỉnh 6 | S |
|----------|--------|----------|----------|-------------|-------------|-------------|------------------|
| Khởi tạo | [0, 1] | [33, 1] | [17, 1]* | [\infty, 1] | [\infty, 1] | [\infty, 1] | 1 |
| 1 | - | [18, 3] | - | [16, 3] | [4, 3]* | [\infty, 1] | 1, 3 |
| 2 | - | [18, 3] | - | [9, 5]* | - | [14, 5] | 1, 3, 5 |
| 3 | - | [18, 3] | - | - | - | [8, 4]* | 1, 3, 5, 4 |
| 4 | - | [18, 3]* | - | - | - | - | 1, 3, 5, 4, 6 |
| 5 | - | - | - | - | - | - | 1, 3, 5, 4, 6, 2 |

Độ dài của CKNN : $18 + 17 + 9 + 4 + 8 = 56$

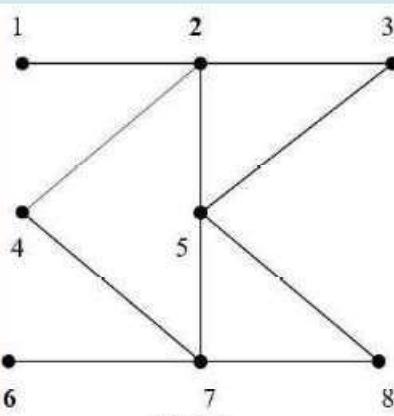
Tập cạnh của CKNN: $\{(2,3), (3,1), (4,5), (5,3), (6,4)\}$

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|----------|----------|----------|----------|----------|----------|
| 1 | 0 | 33 | 17 | ∞ | ∞ | ∞ |
| 2 | 33 | 0 | 18 | 20 | ∞ | ∞ |
| 3 | 17 | 18 | 0 | 16 | 4 | ∞ |
| 4 | ∞ | 20 | 16 | 0 | 9 | 8 |
| 5 | ∞ | ∞ | 4 | 9 | 0 | 14 |
| 6 | ∞ | ∞ | ∞ | 8 | 14 | 0 |

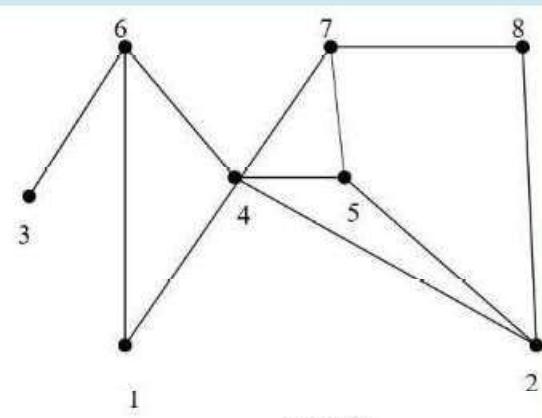
BÀI TẬP

5-1. Giả sử đồ thị G liên thông, có 13 đỉnh và 20 cạnh. Hỏi cây khung của G có bao nhiêu đỉnh ? có bao nhiêu cạnh ?

5-2. Tìm cây khung của đồ thị (hình 1,2) sau theo phương pháp DFS, BFS (chọn đỉnh 1 làm gốc)

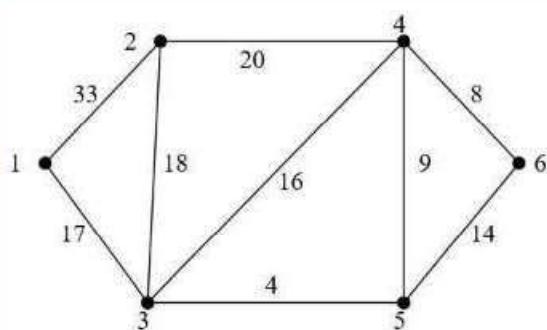


Hình 1

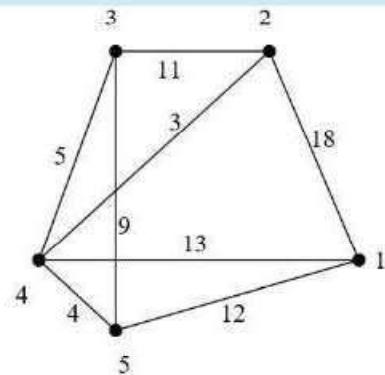


Hình 2

5-3.Tìm cây khung bé nhất của các đồ thị sau (hình 3,4) theo phương pháp KrusKal, Prim

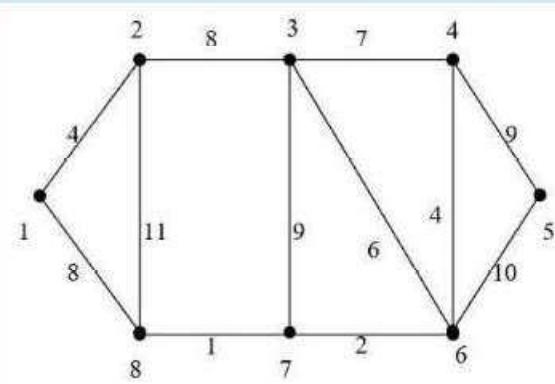


Hình 3

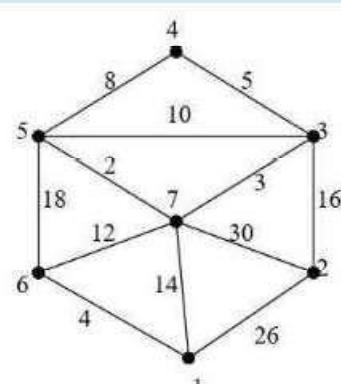


Hình 4

5-4.Tìm cây khung bé nhất của các đồ thị sau (hình 5,6) theo phương pháp KrusKal, Prim

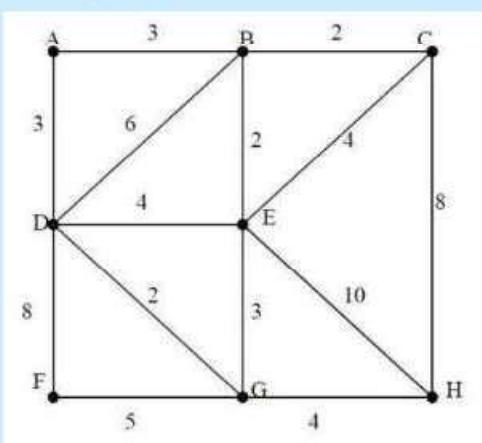


Hình 5



Hình 6

5-5.Tìm cây khung bé nhất của các đồ thị sau (hình 7) theo các phương pháp KrusKal,Prim



Hình 7

5-6. Cài đặt thuật toán Kruskal

5-7. Cài đặt thuật toán Prim

5-8.Cho đồ thị $G=(V,E)$. $W \subset V$. Tìm cây T nhỏ nhất chứa ít nhất là tất cả các đỉnh thuộc W và có tổng trọng số các cạnh là nhỏ nhất (T có thể chứa các đỉnh thuộc V mà không thuộc W , cây T tìm được gọi là cây steiner nhỏ nhất của W).

Chương 6

BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT

Chương này đề cập đơn đồ thị vô hướng/có hướng có trọng số

Nguyễn Đức Nghĩa

Nội dung

- 6.1. Bài toán đường đi ngắn nhất (ĐĐNN)**
- 6.2. Tính chất của ĐĐNN, Giảm cận trên
- 6.3. Thuật toán Bellman-Ford
- 6.4. Thuật toán Dijkstra
- 6.5. Thuật toán Floyd-Warshall

Nguyễn Đức Nghĩa

6.1. Bài toán đường đi ngắn nhất

- ❖ Cho đơn đồ thị có hướng $G = (V, E)$ với hàm trọng số $w: E \rightarrow R$ ($w(e)$ được gọi là độ dài hay trọng số của cạnh e)
- ❖ Độ dài của đường đi $P = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$ là số
$$w(P) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$
- ❖ Đường đi ngắn nhất từ đỉnh u đến đỉnh v là đường đi có độ dài ngắn nhất trong số các đường đi nối u với v .
- ❖ Độ dài của đường đi ngắn nhất từ u đến v còn được gọi là **khoảng cách** từ u tới v và ký hiệu là $\delta(u, v)$.

Nguyễn Đức Nghĩa

Các ứng dụng thực tế

- ❖ Giao thông (Transportation)
- ❖ Truyền tin trên mạng (Network routing) (cần hướng các gói tin đến đích trên mạng theo đường nào?)
- ❖ Truyền thông (Telecommunications)
- ❖ Speech interpretation (best interpretation of a spoken sentence)
- ❖ Điều khiển robot (Robot path planning)
- ❖ Medical imaging
- ❖ Giải các bài toán phức tạp hơn trên mạng
- ❖ ...

Nguyễn Đức Nghĩa

Giả thiết cơ bản

- ❖ Nếu đồ thị có chu trình âm thì độ dài đường đi giữa hai đỉnh nào đó có thể làm nhỏ tuỳ ý:

Giả thiết:

Đồ thị không chứa chu trình độ dài âm (gọi tắt là chu trình âm)

Nguyễn Đức Nghĩa

6.2. Tính chất của ĐĐNN

- ❖ **Tính chất 1.** Đường đi ngắn nhất luôn có thể tìm trong số các đường đi đơn.
- ❖ **Tính chất 2.** Mọi đường đi ngắn nhất trong đồ thị G đều đi qua không quá $n-1$ cạnh, trong đó n là số đỉnh.

Tính chất 3: Giả sử $P = \langle v_1, v_2, \dots, v_k \rangle$ là đđnn từ v_1 đến v_k . Khi đó, $P_{ij} = \langle v_i, v_{i+1}, \dots, v_j \rangle$ là đđnn từ v_i đến v_j , với $1 \leq i \leq j \leq k$.

Nguyễn Đức Nghĩa

Các tính chất của ĐĐNN

Ký hiệu: $\delta(u, v) = \text{độ dài đđnn từ } u \text{ đến } v$ (gọi là khoảng cách từ u đến v)

Hệ quả: Giả sử P là đđnn từ s tới v , trong đó $P = s \xrightarrow{p'} u \rightarrow v$.
Khi đó $\delta(s, v) = \delta(s, u) + w(u, v)$.

Tính chất 4: Giả sử $s \in V$. Đối với mỗi cạnh $(u, v) \in E$, ta có
 $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Nguyễn Đức Nghĩa

Đường đi ngắn nhất xuất phát từ một đỉnh
Single-Source Shortest Paths

Nguyễn Đức Nghĩa

Biểu diễn đường đi ngắn nhất

Các thuật toán tìm đường đi ngắn nhất làm việc với hai mảng:

- ❖ $d(v)$ = độ dài đường đi từ s đến v ngắn nhất hiện biết
(cận trên cho độ dài đường đi ngắn nhất thực sự).
- ❖ $p(v)$ = đỉnh đi trước v trong đường đi nói trên
(sẽ sử dụng để truy ngược đường đi từ s đến v).

Khởi tạo (Initialization)

```
for  $v \in V(G)$ 
    do  $d[v] \leftarrow \infty$ 
         $p[v] \leftarrow \text{NIL}$ 
     $d[s] \leftarrow 0$ 
```

Nguyễn Đức Nghĩa

Thuật toán Ford-Bellman



Richard Bellman
1920-1984



Lester R. Ford, Jr.
1927~

Nguyễn Đức Nghĩa

6.3.Thuật toán Ford-Bellman

- ❖ 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 và giả thiết rằng trong đồ thị không có chu trình âm.

Nguyễn Đức Nghĩa

```
procedure Ford_Bellman;
(*
Đầu vào: Đồ thị có hướng  $G=(V,E)$  với  $n$  đỉnh,
 $s \in V$  là đỉnh xuất phát,
 $a[u,v]$ ,  $u,v \in V$ , ma trận trọng số;

Đầu ra: Khoảng cách từ đỉnh  $s$  đến tất cả các đỉnh còn lại  $d[v]$ ,  $v \in V$ .
Truoc[v],  $v \in V$ , ghi nhận đỉnh đi trước  $v$  trong đường đi ngắn
nhất từ  $s$  đến  $v$ 

Giả thiết: Đồ thị không có chu trình âm.
*)

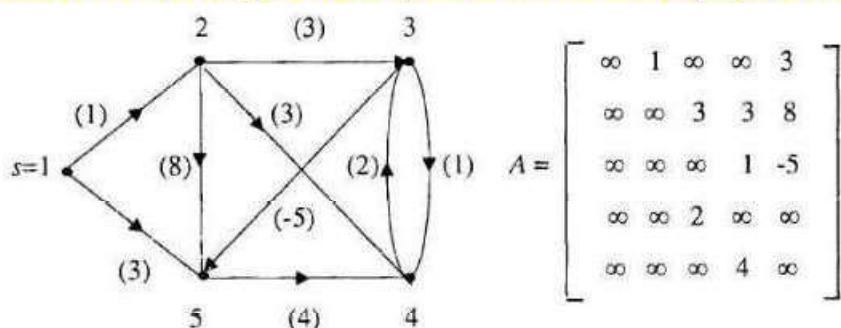
begin
(* Khởi tạo *)
for  $v \in V$  do
begin
     $d[v] := a[s,v]$  ;
    Truoc[v]:=s;
end;
 $d[s]:=0$ ;
for  $k := 1$  to  $n-2$  do
    for  $v \in V \setminus \{s\}$  do
        for  $u \in V$  do
            if  $d[v] > d[u] + a[u,v]$  then
                begin
                     $d[v] := d[u]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

Nhận xét

- ❖ Tính đúng đắn của thuật toán có thể chứng minh trên cơ sở nguyên lý tối ưu của quy hoạch động.
- ❖ Độ phức tạp tính toán của thuật toán là $O(n^3)$.
- ❖ Có thể chấm dứt vòng lặp theo k khi phát hiện trong quá trình thực hiện hai vòng lặp trong không có biến $d[i]$ 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.

Nguyễn Đức Nghĩa

VD Minh họa thuật toán Ford_Bellman



Hình 1. Minh họa 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

Nhận xét

- ❖ Đối với đồ thị tha 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

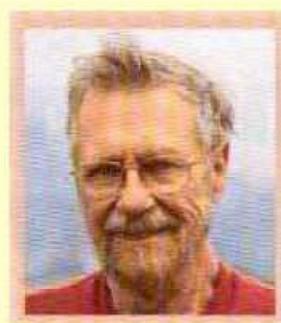
```
for u ∈ Ke-(v) do
    if d[v] > d[u] + w[u,v] then
        begin
            d[v] := d[u] + w[u,v];
            p[v] := u;
        end;
```

- ❖ Thuật toán có độ phức tạp $O(n.m)$.

Nguyễn Đức Nghĩa

6.4.Thuật toán Dijkstra

- ❖ Trong trường hợp trọng số trên các cung là không âm, thuật toán do Dijkstra đề nghị hữu hiệu hơn rất nhiều so với thuật toán Ford-Bellman.
- ❖ Thuật toán được xây dựng dựa trên thủ tục gán nhãn. Đầu tiên nhãn của các đỉnh là tạm thời. Ở mỗi một bước lặp có một nhãn tạm thời trở thành nhãn cố định. Nếu nhãn của một đỉnh u trở thành cố định thì $d[u]$ sẽ cho ta độ dài của đường đi ngắn nhất từ đỉnh s đến u . Thuật toán kết thúc khi nhãn của tất cả các đỉnh trở thành cố định.



Edsger W.Dijkstra
(1930-2002)



Nguyễn Đức Nghĩa

```

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;
   Giá 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
      Tim 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;

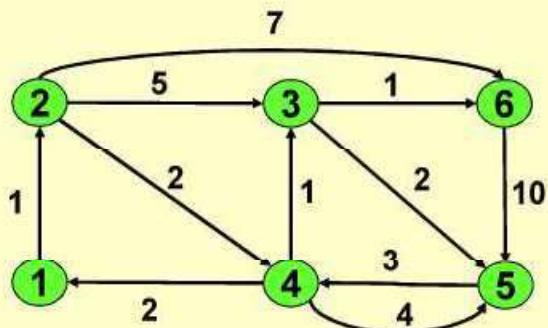
```

Thuật toán Dijkstra

- ❖ Nếu chỉ cần tìm đường đi ngắn nhất từ s đến t thì có thể chấm dứt thuật toán khi đỉnh t trở thành có nhãn cố định.
- ❖ Thuật toán Dijkstra tìm được đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại trên đồ thị sau thời gian $O(n^2)$.

Ví dụ

Tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh còn lại



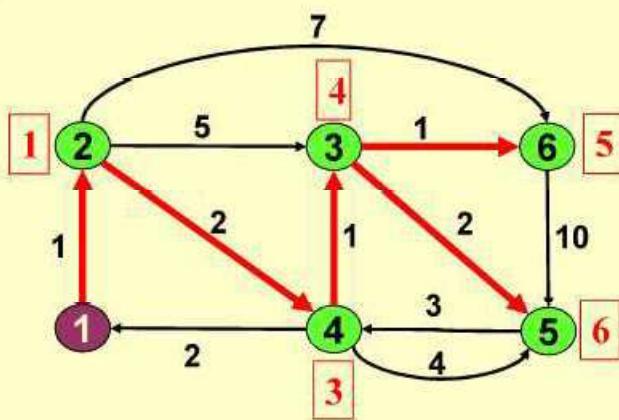
| | Đỉnh 1 | Đỉnh 2 | Đỉnh 3 | Đỉnh 4 | Đỉnh 5 | Đỉnh 6 |
|----------|--------|---------|-------------|-------------|-------------|-------------|
| Khởi tạo | [0, 1] | [1, 1]* | [\infty, 1] | [\infty, 1] | [\infty, 1] | [\infty, 1] |
| 1 | - | - | [6, 2] | [3, 2]* | [\infty, 1] | [8, 2] |
| 2 | - | - | [4, 4]* | - | [7, 4] | [8, 2] |
| 3 | - | - | - | - | [6, 3] | [5, 3]* |
| 4 | - | - | - | - | [6, 3]* | - |
| 5 | - | - | - | - | - | - |

Nguyễn Đức Nghĩa

Cây đường đi ngắn nhất

- ❖ Tập cạnh $\{(p(v), v) : v \in V \setminus \{s\}\}$ tạo thành cây có gốc tại đỉnh nguồn s được gọi là cây đđnn xuất phát từ đỉnh s .

- Các cạnh màu đỏ tạo thành cây đđnn xuất phát từ đỉnh 1
- Số màu đỏ viết bên cạnh mỗi đỉnh là độ dài đường đi ngắn nhất từ 1 đến nó.



Nguyễn Đức Nghĩa

ĐƯỜNG ĐI NGẮN NHẤT GIỮA MỌI CẶP ĐỈNH

All-Pairs Shortest Paths

Nguyễn Đức Nghĩa

6.5. Đường đi ngắn nhất giữa mọi cặp đỉnh

Bài toán Cho đồ thị $G = (V, E)$, với trọng số trên cạnh e là $w(e)$, đối với mỗi cặp đỉnh u, v trong V , tìm đường đi ngắn nhất từ u đến v .

- ✿ Đầu vào: *ma trận trọng số*.
- ✿ Đầu ra *ma trận*: phần tử ở dòng u cột v là độ dài đường đi ngắn nhất từ u đến v .
- ✿ Cho phép có trọng số âm
- ✿ **Giả thiết: Đồ thị không có chu trình âm.**

Thuật toán Floyd-Warshall

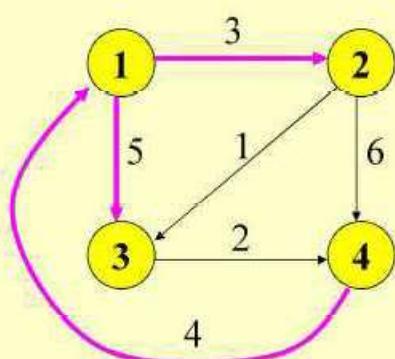
Nguyễn Đức Nghĩa

```

procedure Floyd;
(* Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh
• Đầu vào: Đồ thị稠密 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;

```

Ví dụ



$$D^{(0)} \begin{pmatrix} 0 & 3 & 5 & \infty \\ \infty & 0 & 1 & 6 \\ \infty & \infty & 0 & 2 \\ 4 & \infty & \infty & 0 \end{pmatrix}$$

$$P^{(0)} \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} \\ \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & \text{NIL} & \text{NIL} & 3 \\ 4 & \text{NIL} & \text{NIL} & \text{NIL} \end{pmatrix}$$

Có thể sử dụng 1 là đỉnh trung gian:

$$D^{(1)} \begin{pmatrix} 0 & 3 & 5 & \infty \\ \infty & 0 & 1 & 6 \\ \infty & \infty & 0 & 2 \\ 4 & 7 & 9 & 0 \end{pmatrix}$$

$$P^{(1)} \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} \\ \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & \text{NIL} & \text{NIL} & 3 \\ 4 & 1 & 1 & \text{NIL} \end{pmatrix}$$

Ví dụ (tiếp)

$$D^{(2)} \begin{pmatrix} 0 & 3 & 4 & 9 \\ \infty & 0 & 1 & 6 \\ \infty & \infty & 0 & 2 \\ 4 & 7 & 8 & 0 \end{pmatrix}$$

$$P^{(2)} \begin{pmatrix} \text{NIL} & 1 & 2 & 2 \\ \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & \text{NIL} & \text{NIL} & 3 \\ 4 & 1 & 2 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} \begin{pmatrix} 0 & 3 & 4 & 6 \\ \infty & 0 & 1 & 3 \\ \infty & \infty & 0 & 2 \\ 4 & 7 & 8 & 0 \end{pmatrix}$$

$$P^{(3)} \begin{pmatrix} \text{NIL} & 1 & 2 & 3 \\ \text{NIL} & \text{NIL} & 2 & 3 \\ \text{NIL} & \text{NIL} & \text{NIL} & 3 \\ 4 & 1 & 2 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} \begin{pmatrix} 0 & 3 & 4 & 6 \\ 7 & 0 & 1 & 3 \\ 6 & 9 & 0 & 2 \\ 4 & 7 & 8 & 0 \end{pmatrix}$$

$$P^{(4)} \begin{pmatrix} \text{NIL} & 1 & 2 & 3 \\ 4 & \text{NIL} & 2 & 3 \\ 4 & 1 & \text{NIL} & 3 \\ 4 & 1 & 2 & \text{NIL} \end{pmatrix}$$

Nguyễn Đức Nghĩa

Robert W. Floyd, 1936-2001



- ❖ Born in New York, Floyd finished school at age 14. At the University of Chicago, he received a Bachelor's degree in liberal arts in 1953 (when still only 17) and a second Bachelor's degree in physics in 1958.
- ❖ Becoming a computer operator in the early 1960s, he began publishing many noteworthy papers and was appointed an associate professor at Carnegie Mellon University by the time he was 27 and became a full professor at Stanford University six years later. He obtained this position without a Ph.D.
- ❖ Turing Award, 1978.

Nguyễn Đức Nghĩa

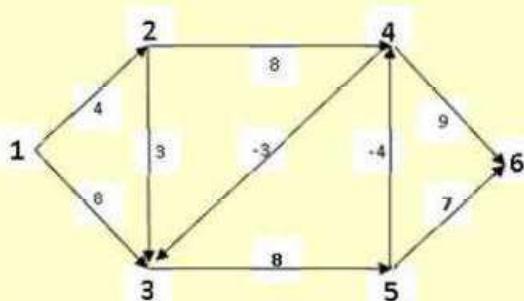
Stephen Warshall



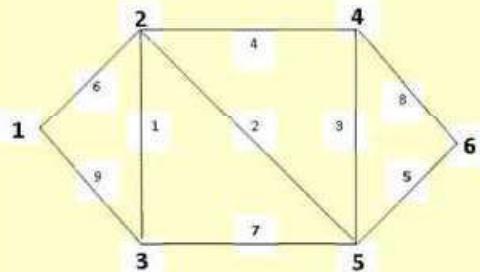
- ❖ 1935 – 2006
- ❖ Proving the correctness of the transitive closure algorithm for boolean circuit.
 - (Wikipedia) There is an interesting anecdote about his proof that the transitive closure algorithm, now known as Warshall's algorithm, is correct. He and a colleague at Technical Operations bet a bottle of rum on who first could determine whether this algorithm always works. Warshall came up with his proof overnight, winning the bet and the rum, which he shared with the loser of the bet. Because Warshall did not like sitting at a desk, he did much of his creative work in unconventional places such as on a sailboat in the Indian Ocean or in a Greek lemon orchard.

Nguyễn Đức Nghĩa

6-1. Hãy tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị G (hình bên phải) theo thuật toán *Ford-Bellman* (lập bảng tính toán chi tiết); từ đó hãy chỉ ra một đường đi ngắn nhất từ đỉnh 1 đến đỉnh 6 và cho biết độ dài của đường đi này.



6-2.Hãy tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị G (hình bên phải) theo thuật toán *Dijkstra* (lập bảng tính toán chi tiết). Hãy chỉ ra một đường đi ngắn nhất từ đỉnh 1 đến đỉnh 6 và khi đó hãy cho biết độ dài của đường đi này.



6-3.Hãy tìm bảng khoảng cách ngắn nhất (d) và bảng đường đi giữa tất cả các cặp đỉnh của đồ thị G (hình trên)

6-4. Cài đặt thuật toán Ford - Bellman

6-5. Cài đặt thuật toán Dijkstra

6-6. Cài đặt thuật toán Floyd-Warshal

6-7. Cài đặt thuật toán tìm một cây steiner nhỏ nhất của đồ thị.

6-8. Cài đặt thuật toán tìm tất cả các cây đường đi ngắn nhất của đồ thị. Và tìm ra một cây đường đi có tổng trọng số nhỏ nhất.

Chương 7

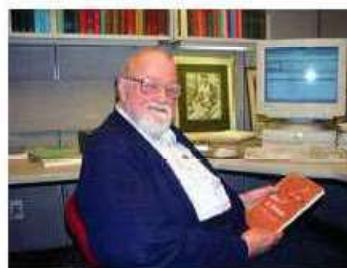
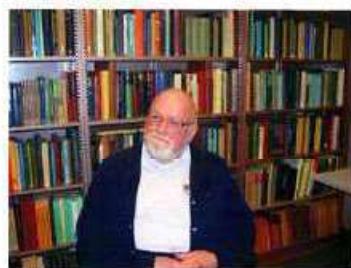


Bài toán luồng cực đại (Maximum Flow Problem)

Chương này đề cập đơn đồ thị có hướng có trọng số

BM Khoa học Máy tính • TOÁN RỜI RẠC • Fall 2005 • Nguyễn Đức Nghĩa

Lester Randolph Ford, Jr (1927 ~)



Lester Randolph Ford, Jr. (born September 23, 1927), son of Lester R. Ford, Sr., is an American mathematician specializing in network flow programming. His 1956 paper with D. R. Fulkerson on the maximum flow problem established the maxflow-mincut theorem.

Delbert Ray Fulkerson

(August 14, 1924 - January 10, 1976)



Delbert Ray Fulkerson was a mathematician who co-developed the Ford-Fulkerson algorithm, one of the most used algorithms to compute maximal flows in networks.

- ❖ Ph.D, Univ. of Wisconsin-Madison, 1951.
- ❖ In 1956, he published his famous paper on the Ford-Fulkerson algorithm together with Lester Randolph Ford.
- ❖ In 1979, the renowned **Fulkerson Prize** was established which is now awarded every three years for outstanding papers in discrete mathematics jointly by the Mathematical Programming Society and the American Mathematical Society.

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 .

Định nghĩa 2. Giả sử cho mạng $G = (V, E)$. Ta gọi luồng f trong mạng $G = (V, E)$ là ánh xạ $f : E \rightarrow \mathbb{R}$, 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:

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

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 $v \neq s, t$:

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

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

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

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

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

Định nghĩa 3. Ta gọi lát cắt (X, X') là một cách phân hoạch tiếp 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.

Bố 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: $\text{val}(f) \leq c(X, X')$.

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

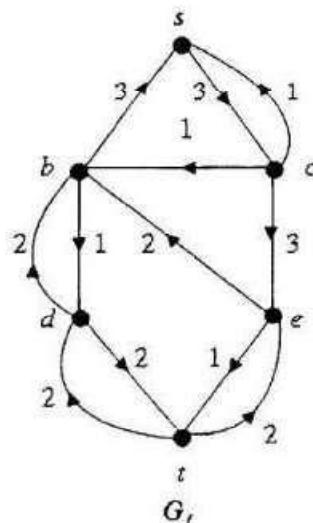
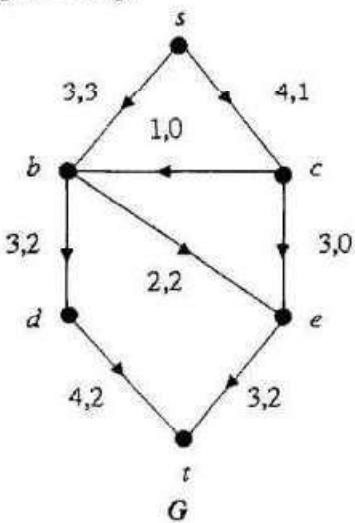
Ford và Fulkerson đã chứng minh rằng giá trị luồng cực đại trong mạng đúng bằng khả năng thông qua của lát cắt hẹp nhất. Để có thể phát biểu và chứng minh kết quả này chúng ta sẽ cần thêm một số khái niệm.

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.



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

Giả sử $P = (s = v_0, v_1, v_2, \dots, v_k = t)$ 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 . 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) $\text{val}(f) = c(X, \bar{X})$ với một lát cắt (X, \bar{X}) nào đó.

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  
        begin  
            Find_Path;  
            if PathFound then Inc_Flow  
            else Stop:=true;  
        end;  
        < Luồng cực đại trong mạng là  $f[u, v]$ ,  $u, v \in V$  >  
        < Lát cắt hẹp nhất là  $(V_T, V \setminus V_T)$  >  
    end;
```

```

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

```

```

procedure Find_Path;
(* Thủ tục gán nhãn tìm đường tăng luồng
p[v], c[v] là nhãn của đỉnh v;
Vt - 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;
    c[s]:=+∞;
    VT = V \ {s};
    PathFound:=true;
    while VT ≠ ∅ do
    begin
        u ∈ VT; (* Lấy u từ VT *)
        for v ∈ V \ VT do
        begin
            if (c[u, v] > 0) and ((l[u, v] < c[u, v])) then
            begin
                p[v]:=u;
                c[v]:= min { c[u], c[u,v] - l[u,v] };
                VT = VT ∪ {v}; (* Nộp v vào danh sách đỉnh có nhãn *)
                if v = t then exit;
            end;
            if (c[v,u] > 0) and (l[v, u] > 0) then
            begin
                p[v]:= -u;
                c[v]:= min { c[u], l[v,u] };
                VT = VT ∪ {v}; (* Nộp v vào danh sách đỉnh có nhãn *)
            end;
        end;
    end;

```

```

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

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

```

BÀI TẬP

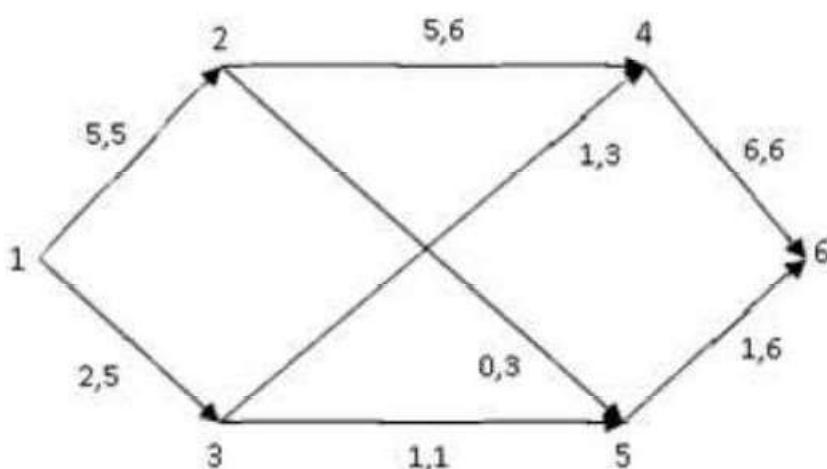
Thực hiện thuật toán Ford-Fulkerson tìm luồng cực đại trong mạng của các hình vẽ sau.

Trình bày các kết quả tính toán trong mỗi bước lặp bao gồm :

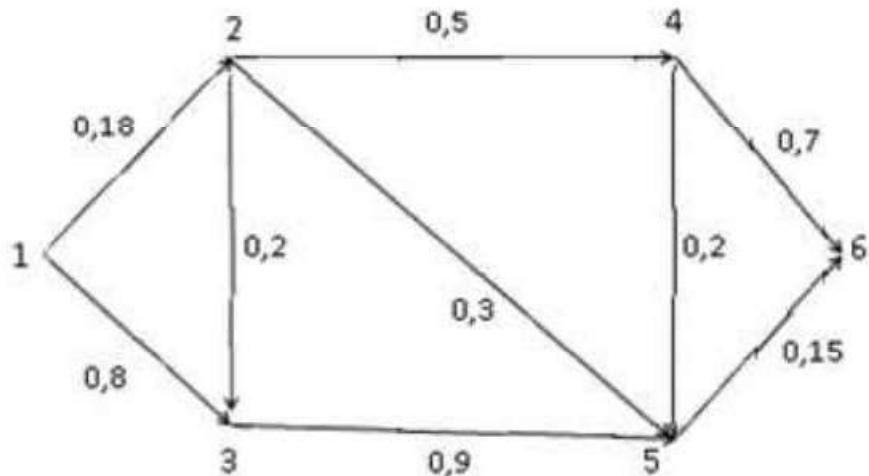
- Đồ thị tăng luồng (mạng thặng dư)
- Đường tăng luồng (đường đi bổ sung) tìm được theo tìm kiếm theo chiều rộng và khả năng thông qua của nó (giả thiết khi duyệt các đỉnh kè của một đỉnh ta duyệt theo thứ tự tăng dần của chỉ số).
- Mạng cùng luồng tìm được sau khi tăng luồng.

Kết quả cuối cùng: Đưa ra giá trị luồng cực đại.

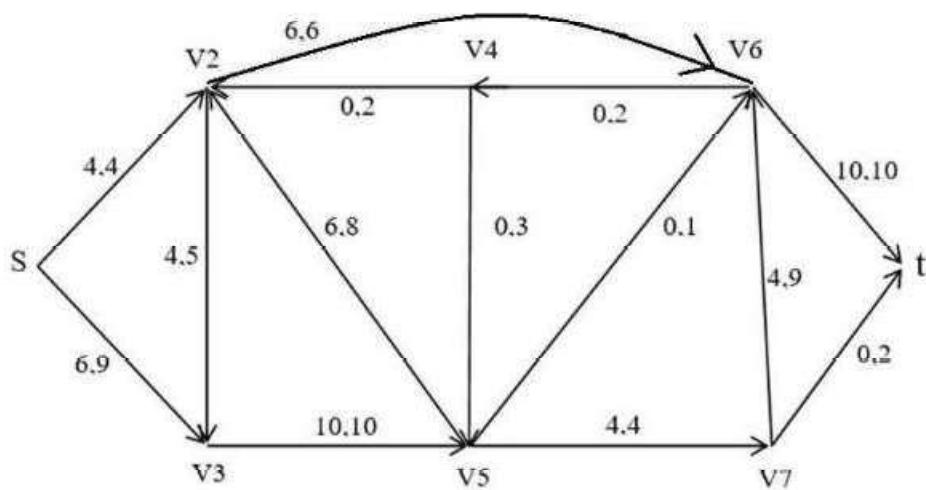
BT7-1



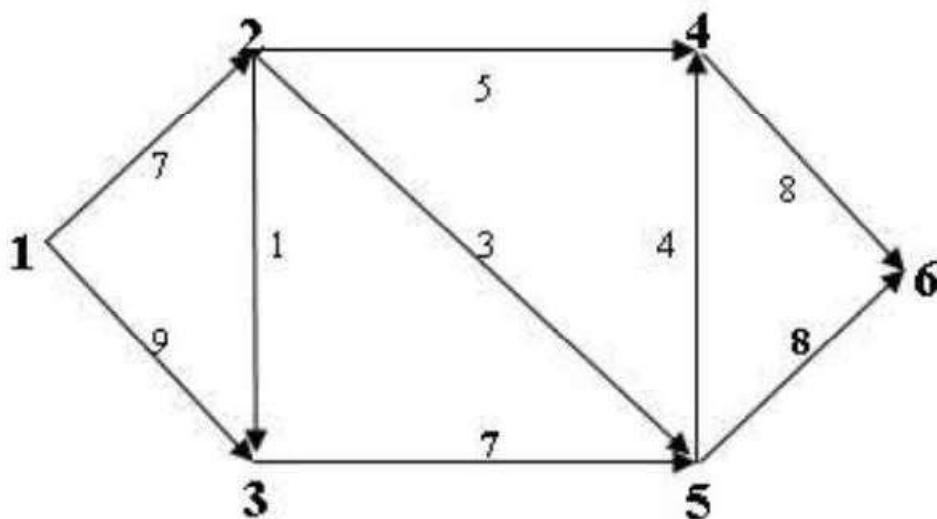
BT7-2



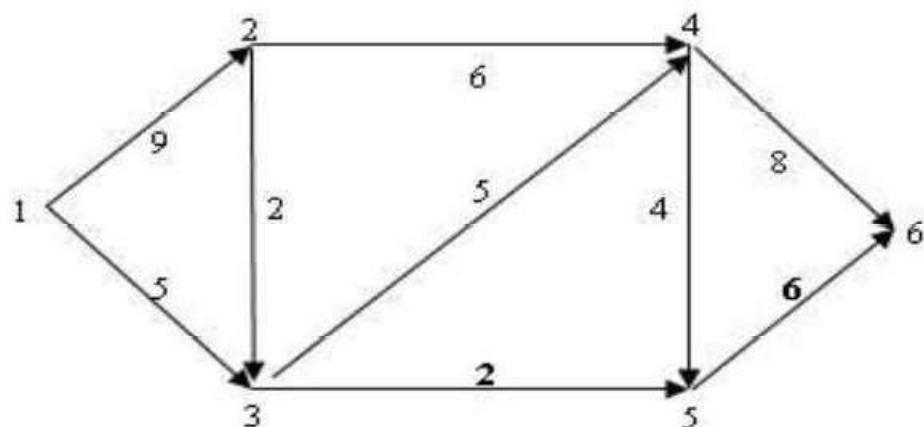
BT7-3



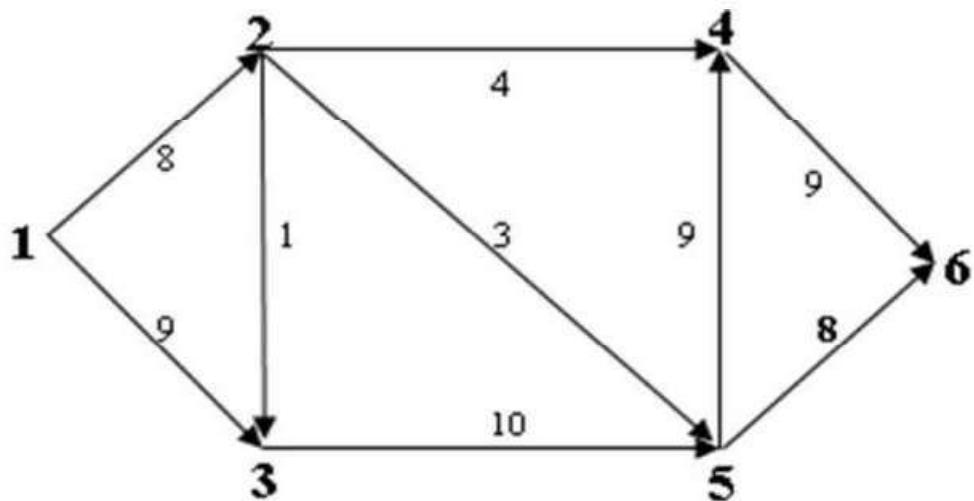
BT7-4



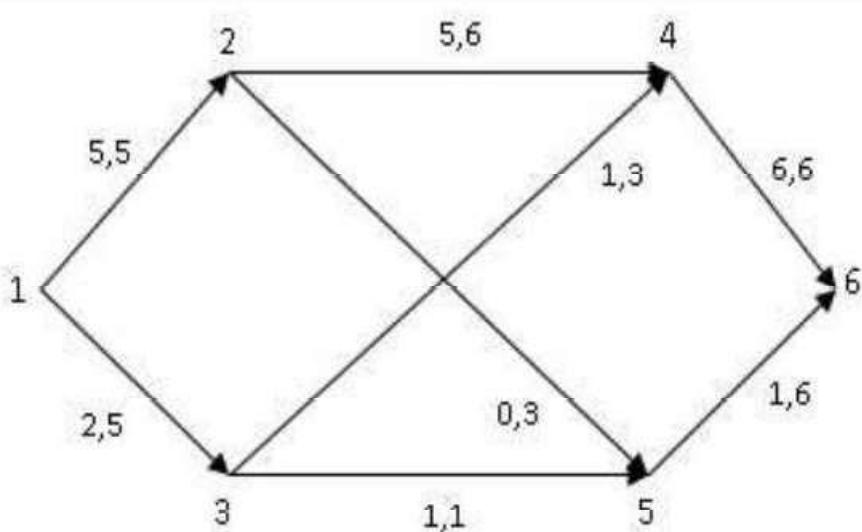
BT7-5



BT7-6



BT7-7



BT7.8.Cài đặt thuật toán tìm luồng cực đại trong mạng theo thuật toán Ford-Fulkerson.

Bài toán ghép cặp

Graph Matching

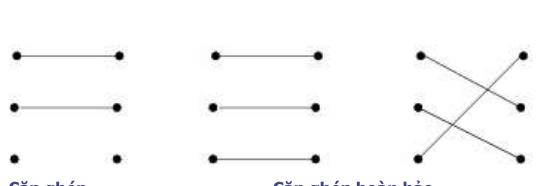
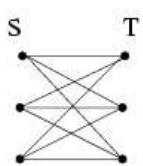
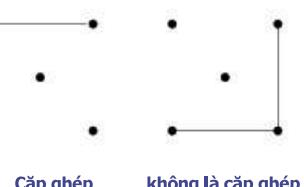
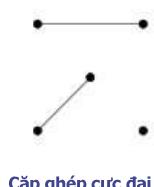
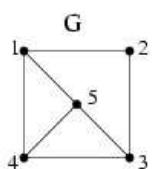
Bài toán ghép cặp trên đồ thị

- ◆ Giả sử $G=(V,E)$ là đồ thị vô hướng, trong đó mỗi cạnh (v,w) được gán với một số thực $c(v,w)$ gọi là trọng số của nó.
- ◆ **Sinh nghĩa.** Cặp ghép M trên đồ thị G là tập các cạnh của đồ thị trong đó không có hai cạnh nào có đỉnh chung.
 - Số cạnh trong M - **kích thước**,
 - Tổng trọng số của các cạnh trong M - **trọng lượng** của cặp ghép.
 - Cặp ghép với kích thước lớn nhất được gọi là **cặp ghép cực đại**.
 - Cặp ghép với trọng lượng lớn nhất được gọi là **cặp ghép lớn nhất**.
 - Cặp ghép được gọi là **đầy đủ (hoàn hảo)** nếu mỗi đỉnh của đồ thị là đầu mút của ít nhất một cạnh trong cặp ghép.

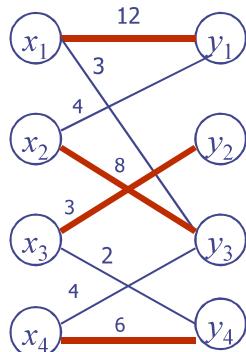
Hai bài toán

- ◆ **Bài toán cặp ghép cực đại:** *Tìm cặp ghép với kích thước lớn nhất trong đồ thị G .*
- ◆ **Bài toán cặp ghép lớn nhất:** *Tìm cặp ghép với trọng lượng lớn nhất trong đồ thị G .*
- ◆ *Ta hạn chế xét các bài toán đặt ra trên đồ thị hai phía $G = (X \cup Y, E)$.*

Ví dụ



Ví dụ



◆ Cặp ghép lớn nhất:

$$M = \{(x_1, y_1), (x_2, y_3), (x_3, y_2), (x_4, y_4)\}$$

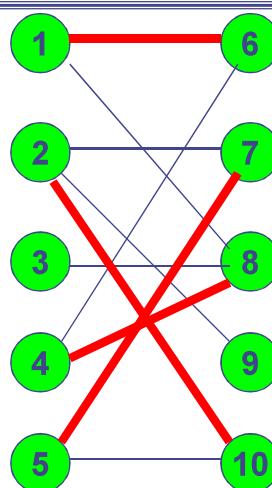
Có trọng lượng 29.

Bài toán cặp ghép cực đại trên đồ thị hai phía

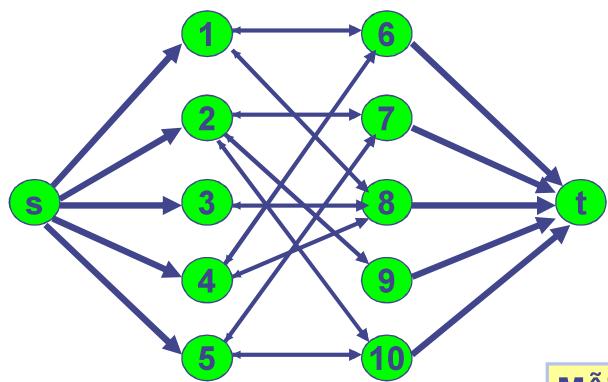
Xét đồ thị hai phía
 $G = (X \cup Y, E)$.

Cặp ghép là tập cạnh mà
không có hai cạnh nào có
chung đỉnh

Bài toán: Tìm cặp ghép
kích thước lớn nhất



Qui về Bài toán luồng cực đại

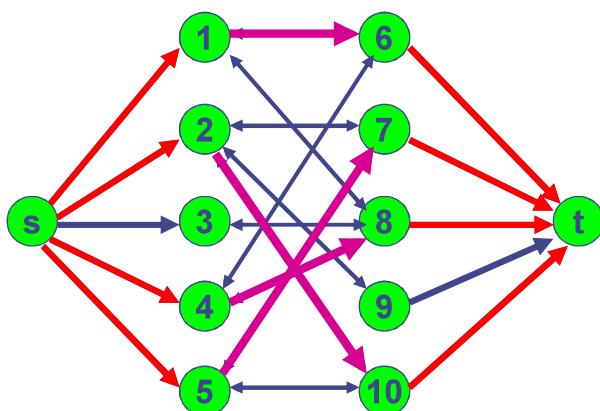


Mỗi cung (s, i) có kntq 1.

Mỗi cung (j, t) có kntq 1.

Mỗi cạnh được
thay thế bởi
cung có kntq 1.

Tìm luồng cực đại



Luồng cực đại từ $s \rightarrow t$ có giá trị 4.

Cặp ghép cực đại có kích thước 4.

Bài toán cặp ghép cực đại trên đồ thị hai phía

- ◆ Giả sử M là một cặp ghép trên G .
- ◆ Nếu cạnh $e = (x, y) \in M$, ta nói e là cạnh của cặp ghép (hay cạnh đậm) và các đỉnh x, y là các đỉnh đậm (hay không tự do).
- ◆ Nếu cạnh $e = (x, y) \notin M$, thì ta nói e là cạnh nhạt còn các đỉnh x, y là các đỉnh nhạt (hay tự do).

Đường tăng cặp ghép

- ◆ Một đường đi trên đồ thị G mà trong đó hai cạnh liên tiếp là không cùng đậm hay nhạt sẽ được gọi là **đường đi luân phiên** đậm/nhạt (hay gọi ngắn gọn là **đường đi luân phiên**).
- ◆ Đường đi luân phiên bắt đầu từ một đỉnh tự do thuộc tập X và kết thúc ở một đỉnh tự do thuộc tập Y được gọi là **đường tăng cặp ghép**.

Định lý Berge

- ◆ **Định lý 1 (Berge C).** Cặp ghép M là cực đại khi và chỉ khi không tìm được đường tăng cặp ghép.

Thuật toán tìm cặp ghép cực đại

- ◆ **Đầu vào:** Đồ thị vô hướng $G = (V, E)$.
- ◆ **Bước khởi tạo.** Xây dựng cặp ghép M trong đồ thị G (có thể bắt đầu từ $M = \emptyset$).
- ◆ **Bước lắp.**
- Kiểm tra tiêu chuẩn tối ưu: Nếu đồ thị G không chứa đường tăng cặp ghép thì M là cặp ghép cực đại, thuật toán kết thúc.
 - Ngược lại, gọi P là một đường tăng cặp ghép xuất phát từ đỉnh tự do $x_0 \in X$, kết thúc ở đỉnh tự do $y_0 \in Y$. Tăng cặp ghép theo qui tắc
$$M := (M \cup P) \setminus (M \cap P),$$
rồi lắp lại bước lắp.

Tìm đường tăng

- ◆ Từ đồ thị G ta xây dựng đồ thị có hướng $G_M = (X \cup Y, E_M)$ với tập cung E_M được bằng cách định hướng lại các cạnh của G theo quy tắc sau:
 - i) Nếu $(x,y) \in M \cap E$, thì $(y,x) \in E_M$;
 - ii) Nếu $(x,y) \in E \setminus M$, thì $(x,y) \in E_M$.Đồ thị G_M sẽ được gọi là **đồ thị tăng cắp ghép**.
- ◆ Dễ thấy:
 - Đường tăng cắp ghép tương ứng với một đường đi xuất phát từ một đỉnh tự do $x_0 \in X$ kết thúc tại một đỉnh tự do $y_0 \in Y$ trên đồ thị G_M .
 - Ngược lại, một đường đi trên đồ thị G_M xuất phát từ một đỉnh tự do $x_0 \in X$ kết thúc tại một đỉnh tự do $y_0 \in Y$ sẽ tương ứng với một đường tăng cắp ghép trên đồ thị G .
- ◆ Vì vậy, để xét xem đồ thị G có chứa đường tăng cắp ghép hay không, có thể thực hiện thuật toán tìm kiếm theo chiều rộng trên đồ thị G_M bắt đầu từ các đỉnh tự do thuộc tập X .

Thuật toán

- ◆ Sử dụng cách tìm đường tăng cắp ghép theo nhận xét vừa nêu, từ sơ đồ tổng quát dễ dàng xây dựng thuật toán để giải bài toán tìm cắp ghép cực đại trên đồ thị hai phía với thời gian tính $O(n^3)$, trong đó $n = \max(|X|, |Y|)$.

Bài toán phân công

Có n công việc và n thợ. Mỗi thợ đều có khả năng thực hiện tất cả các công việc. Biết w_{ij} - hiệu quả phân công thợ i làm việc j ,
($i, j = 1, 2, \dots, n$).

Cần tìm cách phân công thợ thực hiện các công việc sao cho mỗi thợ chỉ thực hiện một việc và mỗi việc chỉ do một thợ thực hiện, đồng thời tổng hiệu quả thực hiện các công việc là lớn nhất.

Qui về bài toán cặp ghép lớn nhất

Xây dựng đồ thị hai phía đầy đủ $G = (X \cup Y, E)$

- $X = \{x_1, x_2, \dots, x_n\}$ tương ứng với các thợ,
- $Y = \{y_1, y_2, \dots, y_n\}$ - tương ứng với các công việc.

Mỗi cạnh (x_i, y_j) được gán cho trọng số $w(x_i, y_j) = w_{ij}$.

Khi đó trong ngôn ngữ đồ thị, bài toán phân công có thể phát biểu như sau: Tìm trong đồ thị G cặp ghép đầy đủ có tổng trọng số là lớn nhất. Cặp ghép nhau vậy được gọi là cặp ghép tối ưu.

Cơ sở thuật toán

Ta gọi một phép gán nhãn chấp nhận được cho các đỉnh của đồ thị $G=(X \cup Y, E)$ là một hàm số f xác định trên tập đỉnh $X \cup Y$: $f: X \cup Y \rightarrow R$, thoả mãn

$$f(x) + f(y) \geq w(x, y), \forall x \in X, \forall y \in Y.$$

Một phép gán nhãn chấp nhận được nh vậy dễ dàng có thể tìm được, chẳng hạn phép gán nhãn sau đây là chấp nhận được

$$f(x) = \max \{ w(x, y) : y \in Y \}, \quad x \in X,$$

$$f(y) = 0, \quad y \in Y.$$

Đồ thị cân bằng

◆ Giả sử có f là một phép gán nhãn chấp nhận được, ký hiệu

$$E_f = \{(x, y) \in E : f(x) + f(y) = w(x, y)\}.$$

◆ Ký hiệu G_f là đồ thị con của G sinh bởi tập đỉnh $X \cup Y$ và tập cạnh E_f . Ta sẽ gọi G_f là **đồ thị cân bằng**.

Tiêu chuẩn tối ưu

Định lý 2. *Giả sử f là phép gán nhãn chấp nhận được. Nếu G_f chứa cặp ghép đầy đủ M^* , thì M^* là cặp ghép tối ưu.*

Sơ đồ thuật toán

- ◆ Ta sẽ bắt đầu từ một phép gán nhãn chấp nhận được f . Xây dựng đồ thị G_f . Bắt đầu từ một cặp ghép M nào đó trong G_f ta xây dựng cặp ghép đầy đủ trong G_f . Nếu tìm được cặp ghép đầy đủ M^* , thì nó chính là cặp ghép tối ưu. Ngược lại, ta sẽ tìm được cặp ghép cực đại không đầy đủ M' . Từ M' ta sẽ tìm cách sửa phép gán nhãn thành f' sao cho M' vẫn là cặp ghép của G'_f và có thể tiếp tục phát triển M' trong G'_f ., v.v...
- ◆ Quá trình được tiếp tục cho đến khi thu được cặp ghép đầy đủ trong đồ thị cân bằng.

Điều chỉnh nhãn

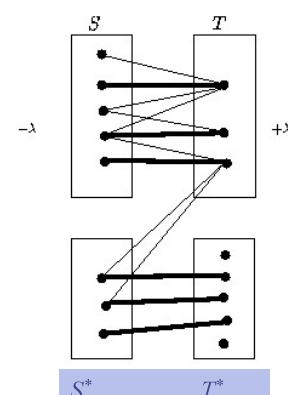
- Giả sử M là cặp ghép cực đại trong đồ thị G_f và M cha là cặp ghép đầy đủ của G . Ta cần tìm cách điều chỉnh phép gán nhãn f thoả mãn các yêu cầu đặt ra.
- Thực hiện tìm kiếm theo chiều rộng từ các đỉnh tự do trong X . Gọi S là các đỉnh được thăm trong X , còn T là các đỉnh được thăm trong Y trong quá trình thực hiện tìm kiếm. Ký hiệu

$$S^* = X \setminus S; \quad T^* = Y \setminus T.$$

$|S| > |T|$ (do mỗi đỉnh trong T đạt được từ một đỉnh nào đó trong S).

Điều chỉnh nhãn

- Từ tính chất của thuật toán tìm kiếm theo chiều rộng, rõ ràng, không có cạnh nào từ S đến T^* . Để sửa chữa nhãn, chúng ta sẽ tiến hành giảm đồng loạt các nhãn trong S đi cùng một giá trị λ nào đó, và đồng thời sẽ tăng đồng loạt nhãn của các đỉnh trong T lên λ . Điều đó đảm bảo các cạnh từ S sang T (nghĩa là những cạnh mà một đầu mút thuộc S còn một đầu mút thuộc T) không bị loại bỏ khỏi đồ thị cân bằng



Các tập S và T trong thực hiện thuật toán. Chỉ vẽ các cạnh trong G_f

Điều chỉnh nhãn

- ◆ Khi các nhãn trong S bị giảm, các cạnh trong G từ S sang T^* sẽ có khả năng gia nhập vào đồ thị cân bằng G_f . Ta sẽ tăng λ đến khi có thêm ít nhất một cạnh mới gia nhập đồ thị cân bằng. Có hai khả năng:
 - Nếu cạnh mới gia nhập đồ thị cân bằng giúp ta thăm được một đỉnh không tự do $y \in T^*$ thì từ nó ta sẽ thăm được một đỉnh được ghép với nó trong cặp ghép $x \in S^*$, và cả hai đỉnh này được bổ sung vào S và T tương ứng, và như vậy việc tìm kiếm đường tăng sẽ được tiếp tục mở rộng.
 - Nếu cạnh mới gia nhập đồ thị cân bằng cho phép thăm được một đỉnh tự do $y \in T^*$ thì ta tìm được đường tăng cặp ghép, và kết thúc một pha điều chỉnh nhãn.

Điều chỉnh nhãn

Ta gọi *một pha điều chỉnh* là tất cả các lần sửa nhãn cần thiết để tăng được kích thước của cặp ghép M .

- ◆ Vì sau mỗi pha điều chỉnh kích thước của cặp ghép tăng lên 1, nên ta phải thực hiện nhiều nhất n pha điều chỉnh.
- ◆ Trong mỗi pha điều chỉnh, do sau mỗi lần sửa nhãn có ít nhất hai đỉnh mới được bổ sung vào danh sách các đỉnh được thăm, nên ta phải thực hiện việc sửa nhãn không quá n lần. Mặt khác, trong thời gian $O(n^2)$ ta có thể xác định được cạnh nào từ S sang T^* là cạnh gia nhập đồ thị cân bằng (bằng việc duyệt hết các cạnh). Từ đó suy ra đánh giá thời gian tính của thuật toán là $O(n^4)$.

Thuật toán

- ◆ **Bước 0:** Tìm một phép gán nhãn chấp nhận được f .
- ◆ **Bước 1:** Xây dựng đồ thị cân bằng G_f .
- ◆ **Bước 2:** Tìm cặp ghép cực đại M trong G_f .
- ◆ **Bước 3:** Nếu M là cặp ghép đầy đủ thì nó là cặp ghép lớn nhất cần tìm. Thuật toán kết thúc.
- ◆ **Bước 4:** Gọi S là tập các đỉnh tự do trong X . Thực hiện tìm kiếm từ các đỉnh trong S . Gọi T là tập các đỉnh của Y được thăm trong quá trình tìm kiếm. Bổ sung các đỉnh trong X được thăm trong quá trình tìm kiếm vào S .
- ◆ **Bước 5:** Tiến hành điều chỉnh nhãn f ta sẽ bổ sung được các cạnh vào G_f cho đến khi tìm được đường tăng, bổ sung các đỉnh mới được thăm vào S và T tương ứng như đã mô tả ở trên. Tăng cặp ghép M và quay lại bước 3.

BÀI TẬP

- ◆ Cài đặt các thuật toán ghép cặp trên đồ thị hai phía (có trọng và không có trọng)

§12. BÀI TOÁN TÌM BỘ GHÉP CỰC ĐẠI VỚI TRỌNG SỐ CỰC TIỀU TRÊN ĐỒ THỊ HAI PHÍA - THUẬT TOÁN HUNGARI

I. BÀI TOÁN PHÂN CÔNG

- Đây là một dạng bài toán phát biểu như sau: Có m người (đánh số 1, 2, ..., m) và n công việc (đánh số 1, 2, ..., n), mỗi người có khả năng thực hiện một số công việc nào đó. Để giao cho người i thực hiện công việc j cần một chi phí là $c[i, j] \geq 0$. Cần phân cho mỗi thợ một việc và mỗi việc chỉ do một thợ thực hiện sao cho số công việc có thể thực hiện được là nhiều nhất và nếu có ≥ 2 phương án đều thực hiện được nhiều công việc nhất thì chỉ ra phương án chi phí ít nhất.
- Dựng đồ thị hai phía $G = (X \cup Y, E)$ với X là tập m người, Y là tập n việc và $(u, v) \in E$ với trọng số $c[u, v]$ nếu như người u làm được công việc v . Bài toán đưa về tìm bộ ghép nhiều cạnh nhất của G có trọng số nhỏ nhất.
- Gọi $k = \max(m, n)$. Bổ sung vào tập X và Y một số đỉnh giả để $|X| = |Y| = k$.
- Gọi M là một số dương đủ lớn hơn chi phí của mọi phép phân công có thể. Với mỗi cặp đỉnh (u, v) : $u \in X$ và $v \in Y$. Nếu $(u, v) \notin E$ thì ta bổ sung cạnh (u, v) vào E với trọng số là M .
- Khi đó ta được G là một **đồ thị hai phía đầy đủ** (Đồ thị hai phía mà giữa một đỉnh bất kỳ của X và một đỉnh bất kỳ của Y đều có cạnh nối). Và nếu như ta **tìm được bộ ghép đầy đủ k cạnh mang trọng số nhỏ nhất** thì ta chỉ cần **loại bỏ khỏi bộ ghép đó những cạnh mang trọng số M vừa thêm vào** thì sẽ được kế hoạch phân công 1 người \leftrightarrow 1 việc cần tìm. Điều này dễ hiểu bởi bộ ghép đầy đủ mang trọng số nhỏ nhất tức là phải ít cạnh trọng số M nhất, tức là số phép phân công là nhiều nhất, và tất nhiên trong số các phương án ghép ít cạnh trọng số M nhất thì đây là phương án trọng số nhỏ nhất, tức là tổng chi phí trên các phép phân công là ít nhất.

II. PHÂN TÍCH

- Vào: Đồ thị hai phía đầy đủ $G = (X \cup Y, E)$; $|X| = |Y| = k$. Được cho bởi ma trận vuông C cỡ $k \times k$, $c[i, j] =$ trọng số cạnh nối đỉnh X_i với Y_j . Giả thiết $c[i, j] \geq 0$. với mọi i, j .
- Ra: Bộ ghép đầy đủ trọng số nhỏ nhất.

Hai định lý sau đây tuy rất đơn giản nhưng là những định lý quan trọng tạo cơ sở cho thuật toán sẽ trình bày:

Định lý 1: Loại bỏ khỏi G những cạnh trọng số > 0 . Nếu những cạnh trọng số 0 còn lại tạo ra bộ ghép k cạnh trong G thì đây là bộ ghép cần tìm.

Chứng minh: Theo giả thiết, các cạnh của G mang trọng số không âm nên bất kỳ bộ ghép nào trong G cũng có trọng số không âm, mà bộ ghép ở trên mang trọng số 0, nên tất nhiên đó là bộ ghép đầy đủ trọng số nhỏ nhất.

Định lý 2: Với đỉnh X_i , nếu ta cộng thêm một số Δ (dương hay âm) vào tất cả những cạnh liên thuộc với X_i (tương đương với việc cộng thêm Δ vào tất cả các phần tử thuộc hàng i của ma trận C) thì không ảnh hưởng tới bộ ghép đầy đủ trọng số nhỏ nhất.

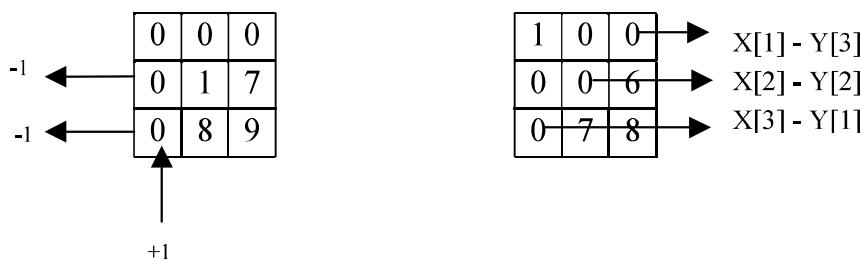
Chứng minh: Với một bộ ghép đầy đủ bất kỳ thì có một và chỉ một cạnh ghép với $X[i]$. Nên việc cộng thêm Δ vào tất cả các cạnh liên thuộc với $X[i]$ sẽ làm tăng trọng số bộ ghép đó lên Δ . Vì vậy

nếu như ban đầu, M là bộ ghép đầy đủ trọng số nhỏ nhất thì sau thao tác trên, M vẫn là bộ ghép đầy đủ trọng số nhỏ nhất.

Hệ quả: Với đỉnh $Y[j]$, nếu ta cộng thêm một số Δ (dương hay âm) vào tất cả những cạnh liên thuộc với $Y[j]$ (tương đương với việc cộng thêm Δ vào tất cả các phần tử thuộc cột j của ma trận C) thì không ảnh hưởng tới bộ ghép đầy đủ trọng số nhỏ nhất.

Từ đây có thể nhận ra tư tưởng của thuật toán: Từ đồ thị G , ta tìm chiến lược cộng / trừ một cách hợp lý trọng số của các cạnh liên thuộc với một đỉnh nào đó để được một đồ thị mới vẫn có các cạnh trọng số không âm, mà các cạnh trọng số 0 của đồ thị mới đó chứa một bộ ghép đầy đủ k cạnh.

Ví dụ: Biến đổi ma trận trọng số của đồ thị hai phía 3 đỉnh trái, 3 đỉnh phải:



III. THUẬT TOÁN

1. Các khái niệm:

Để cho gọn, ta gọi những cạnh trọng số 0 của G là những 0_cạnh.

Xét một bộ ghép M chỉ gồm những 0_cạnh.

- Những đỉnh $\in M$ gọi là những đỉnh đã ghép, những đỉnh còn lại gọi là những đỉnh chưa ghép.
- Những 0_cạnh $\in M$ gọi là những 0_cạnh đã ghép, những 0_cạnh còn lại là những 0_cạnh chưa ghép.

Nếu ta định hướng lại các 0_cạnh như sau: Những 0_cạnh chưa ghép cho hướng từ tập X sang tập Y , những 0_cạnh đã ghép cho hướng từ tập Y về tập X . Khi đó:

- Đường pha (Alternating Path) là một đường đi cơ bản xuất phát từ một X _đỉnh chưa ghép đi theo các 0_cạnh đã định hướng ở trên. Như vậy dọc trên đường pha, các 0_cạnh chưa ghép và những 0_cạnh đã ghép xen kẽ nhau. Vì đường pha chỉ là đường đi cơ bản trên đồ thị định hướng nên việc xác định những đỉnh nào có thể đến được từ $x \in X$ bằng một đường pha có thể sử dụng các thuật toán tìm kiếm trên đồ thị (BFS hoặc DFS). Những đỉnh và những cạnh được duyệt qua tạo thành một cây pha gốc x
- Một đường mở (Augmenting Path) là một đường pha đi từ một X _đỉnh chưa ghép tới một Y _đỉnh chưa ghép. Như vậy:
 - Đường đi trực tiếp từ một X _đỉnh chưa ghép tới một Y _đỉnh chưa ghép qua một 0_cạnh chưa ghép cũng là một đường mở.
 - Dọc trên đường mở, số 0_cạnh chưa ghép nhiều hơn số 0_cạnh đã ghép đúng 1 cạnh.

2. Thuật toán Hungari

Bước 1: Khởi tạo:

- Một bộ ghép $M := \emptyset$

Bước 2: Với mọi đỉnh $x^* \in X$, ta tìm cách ghép x^* như sau.

Bắt đầu từ đỉnh x^* chưa ghép, thử tìm đường mở bắt đầu ở x^* bằng thuật toán tìm kiếm trên đồ thị (BFS hoặc DFS - thông thường nên dùng BFS để tìm đường qua ít cạnh nhất) có hai khả năng xảy ra:

- Hoặc tìm được đường mở thì dọc theo đường mở, ta loại bỏ những cạnh đã ghép khỏi M và thêm vào M những cạnh chưa ghép, ta được một **bộ ghép mới nhiều hơn bộ ghép cũ 1 cạnh và đỉnh x^* trở thành đã ghép.**
- Hoặc không tìm được đường mở thì do ta sử dụng thuật toán tìm kiếm trên đồ thị nên có thể xác định được hai tập:
 - ❖ VisitedX = {Tập những X_đỉnh có thể đến được từ x^* bằng một đường pha}
 - ❖ VisitedY = {Tập những Y_đỉnh có thể đến được từ x^* bằng một đường pha}
 - ❖ Gọi Δ là trọng số nhỏ nhất của các cạnh nối giữa một đỉnh thuộc VisitedX với một đỉnh không thuộc VisitedY. Để thấy $\Delta > 0$ bởi nếu $\Delta = 0$ thì tồn tại một 0_cạnh (x, y) với $x \in \text{VisitedX}$ và $y \notin \text{VisitedY}$. Vì x^* đến được x bằng một đường pha và (x, y) là một 0_cạnh nên x^* cũng đến được y bằng một đường pha, dẫn tới $y \in \text{VisitedY}$, điều này vô lý.
 - ❖ Biến đổi đồ thị G như sau: Với $\forall x \in \text{VisitedX}$, trừ Δ vào trọng số những cạnh liên thuộc với x, Với $\forall y \in \text{VisitedY}$, cộng Δ vào trọng số những cạnh liên thuộc với y.
 - ❖ Lặp lại thủ tục tìm kiếm trên đồ thị thử tìm đường mở xuất phát ở x^* cho tới khi tìm ra đường mở.

Bước 3: Sau bước 2 thì mọi X_đỉnh đều được ghép, in kết quả về bộ ghép tìm được.

Mô hình cài đặt của thuật toán có thể viết như sau:

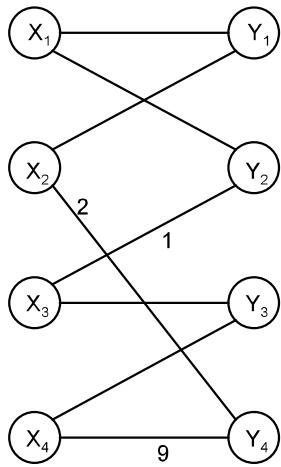
```

<Khởi tạo: M := Ø ...>;
for (x* ∈ X) do
begin
  repeat
    <Tìm đường mở xuất phát ở x*>;
    if <Không tìm thấy đường mở> then <Biến đổi đồ thị G: Chọn Δ := ...>;
    until <Tìm thấy đường mở>;
    <Đọc theo đường mở, loại bỏ những cạnh đã ghép khỏi M
      và thêm vào M những cạnh chưa ghép>;
  end;
<Kết quả>;

```

Ví dụ minh họa:

Để không bị rối hình, ta hiểu những cạnh không ghi trọng số là những 0_cạnh, những cạnh không vẽ mang trọng số rất lớn trong trường hợp này không cần thiết phải tính đến. Những cạnh nét đậm là những cạnh đã ghép, những cạnh nét thanh là những cạnh chưa ghép.

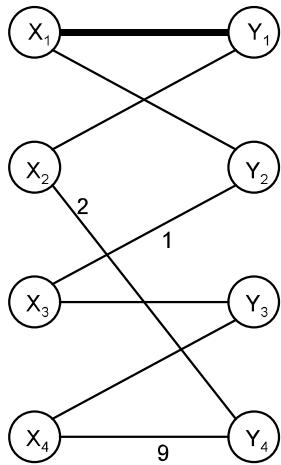
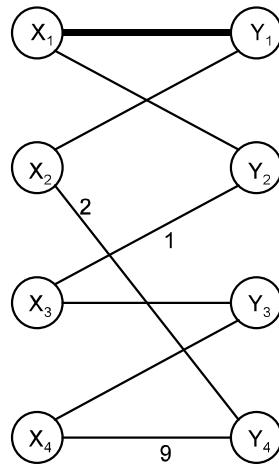


$x^* = X_1$

Tìm được đường mở:

$X_1 \rightarrow Y_1$

Tăng cắp

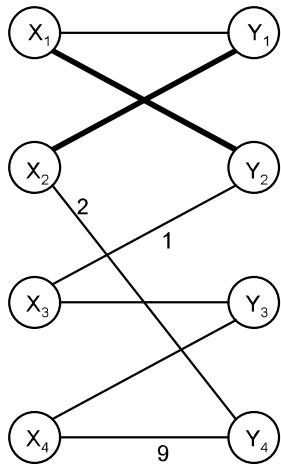
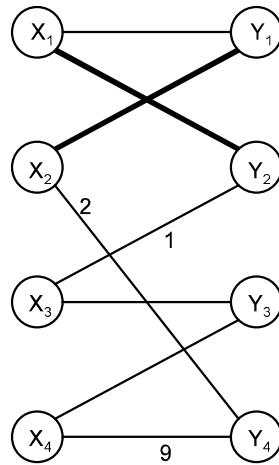


$x^* = X_2$

Tìm được đường mở:

$X_2 \rightarrow Y_1 \rightarrow X_1 \rightarrow Y_2$

Tăng cắp

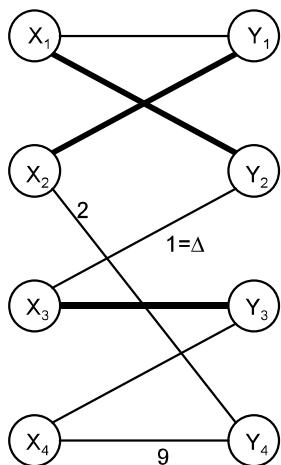
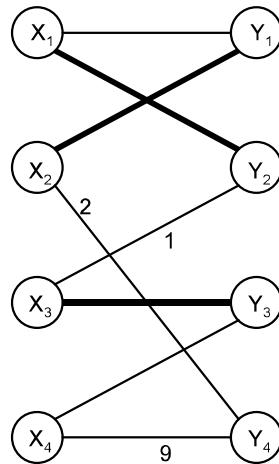


$x^* = X_3$

Tìm được đường mở:

$X_3 \rightarrow Y_3$

Tăng cắp



$x^* = X_4$

Không tìm được đường mở:

Tập những X_i đỉnh đến được từ X_4 bằng một đường pha: $\{X_3, X_4\}$

Tập những Y_j đỉnh đến được từ X_4 bằng một đường pha: $\{Y_3\}$

Giá trị xoay $\Delta = 1$ (Cạnh X_3-Y_2)

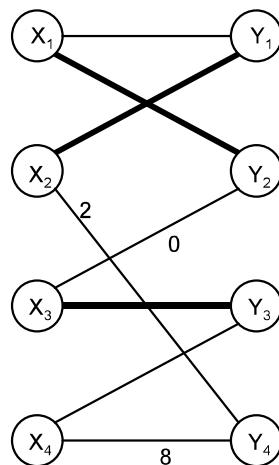
Trừ tất cả trọng số những cạnh liên

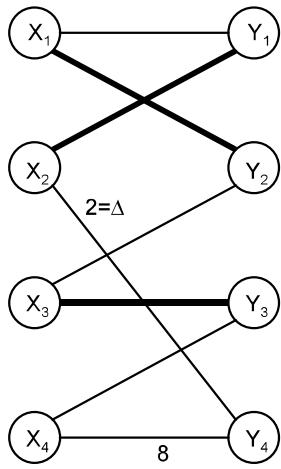
thuộc với $\{X_3, X_4\}$ đi 1

Cộng tất cả trọng số những cạnh liên

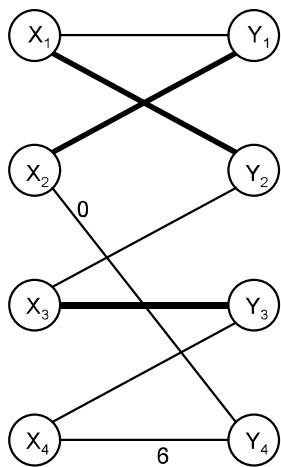
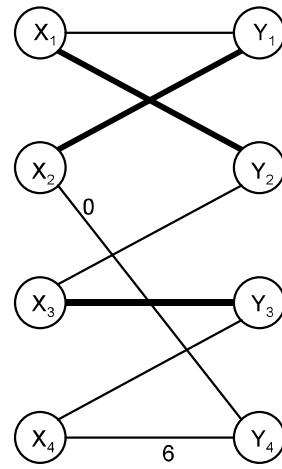
thuộc với Y_3 lên 1

106

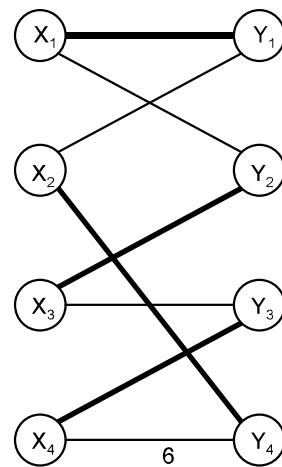




$x^* = X_4$
 Vẫn không tìm được đường mở:
 Tập những X _đỉnh đến được từ X_4
 bằng một đường pha:
 $\{X_1, X_2, X_3, X_4\}$
 Tập những Y _đỉnh đến được từ X_4
 bằng một đường pha:
 $\{Y_1, Y_2, Y_3\}$
 Giá trị xoay $\Delta = 2$ (Cạnh X_2-Y_4)
 Trừ tất cả trọng số những cạnh liên
 thuộc với $\{X_1, X_2, X_3, X_4\}$ đi 2
 Cộng tất cả trọng số những cạnh liên
 thuộc với $\{Y_1, Y_2, Y_3\}$ lên 2



$x^* = X_4$
 Tìm được đường mở:
 $X_4 \rightarrow Y_3 \rightarrow X_3 \rightarrow Y_2 \rightarrow X_1 \rightarrow Y_1 \rightarrow X_2 \rightarrow Y_4$
 Tăng cặp
 Xong



Để ý rằng nếu như không tìm thấy đường mở xuất phát ở x^* thì quá trình tìm kiếm trên đồ thị sẽ cho ta một cây pha gốc x^* . Giá trị xoay Δ thực chất là trọng số nhỏ nhất của cạnh nối một X _đỉnh trong cây pha với một Y _đỉnh ngoài cây pha (cạnh ngoài). Việc trừ Δ vào những cạnh liên thuộc với X _đỉnh trong cây pha và cộng Δ vào những cạnh liên thuộc với Y _đỉnh trong cây pha sẽ làm cho cạnh ngoài nói trên trở thành 0_cạnh, các cạnh khác vẫn có trọng số ≥ 0 . Nhưng quan trọng hơn là tất cả những cạnh trong cây pha vẫn cứ là 0_cạnh. Điều đó đảm bảo cho quá trình tìm kiếm trên đồ thị lần sau sẽ xây dựng được cây pha mới lớn hơn cây pha cũ (Thể hiện ở chỗ: tập VisitedY sẽ rộng hơn trước ít nhất 1 phần tử). Vì tập các Y _đỉnh đã ghép là hữu hạn nên sau không quá k bước, sẽ có một Y _đỉnh chưa ghép \in VisitedY, tức là tìm ra đường mở

Trên thực tế, để chương trình hoạt động nhanh hơn, trong bước khởi tạo, người ta có thể thêm một thao tác:

Với mỗi đỉnh $x \in X$, xác định trọng số nhỏ nhất của các cạnh liên thuộc với x , sau đó trừ tất cả trọng số các cạnh liên thuộc với x đi trọng số nhỏ nhất đó. Làm tương tự như vậy với các Y _đỉnh. Điều này tương đương với việc trừ tất cả các phần tử trên mỗi hàng của ma trận C đi giá trị nhỏ nhất trên hàng đó, rồi lại trừ tất cả các phần tử trên mỗi cột của ma trận C đi phần tử nhỏ nhất trên cột đó. Khi đó số 0_cạnh của đồ thị là khá nhiều, có thể chứa ngay bộ ghép đầy đủ hoặc chỉ cần qua ít bước biến đổi là sẽ chứa bộ ghép đầy đủ k_cạnh.

Để tưởng nhớ hai nhà toán học König và Egervary, những người đã đặt cơ sở lý thuyết đầu tiên cho phương pháp, người ta đã lấy tên của đất nước sinh ra hai nhà toán học này để đặt tên cho thuật

toán. Mặc dù sau này có một số cải tiến nhưng tên gọi Thuật toán Hungari (Hungarian Algorithm) vẫn được dùng phổ biến.

IV. CÀI ĐẶT

1. Phương pháp đối ngẫu Kuhn-Munkres (Không làm biến đổi ma trận C ban đầu)

Phương pháp Kuhn-Munkres đi tìm hai dãy số $Fx[1..k]$ và $Fy[1..k]$ thoả mãn:

- $c[i, j] - Fx[i] - Fy[j] \geq 0$
- Tập các cạnh $(X[i], Y[j])$ thoả mãn $c[i, j] - Fx[i] - Fy[j] = 0$ chứa trọn một bộ ghép đầy đủ k cạnh, đây chính là bộ ghép cần tìm.

Chứng minh:

Nếu tìm được hai dãy số thoả mãn trên thì ta chỉ việc thực hiện hai thao tác:

Với mỗi đỉnh $X[i]$, trừ tất cả trọng số của những cạnh liên thuộc với $X[i]$ đi $Fx[i]$

Với mỗi đỉnh $Y[j]$, trừ tất cả trọng số của những cạnh liên thuộc với $Y[j]$ đi $Fy[j]$

(Hai thao tác này tương đương với việc trừ tất cả trọng số của các cạnh $(X[i], Y[j])$ đi một lượng $Fx[i] + Fy[j]$ tức là $c[i, j] := c[i, j] - Fx[i] - Fy[j]$)

Thì dễ thấy đồ thị mới tạo thành sẽ gồm có các cạnh trọng số không âm và những 0_cạnh của đồ thị chứa trọn một bộ ghép đầy đủ.

| | 1 | 2 | 3 | 4 | |
|---|--------------|--------------|--------------|-------------|-------------|
| 1 | 0 | 0 | M | M | $Fx[1] = 2$ |
| 2 | 0 | M | M | 2 | $Fx[2] = 2$ |
| 3 | M | 1 | 0 | M | $Fx[3] = 3$ |
| 4 | M | M | 0 | 9 | $Fx[4] = 3$ |
| | $Fy[1] = -2$ | $Fy[2] = -2$ | $Fy[3] = -3$ | $Fy[4] = 0$ | |

(Có nhiều phương án khác: $Fx = (0, 0, 1, 1)$; $Fy = (0, 0, -1, 2)$ cũng đúng)

Vậy phương pháp Kuhn-Munkres đưa việc biến đổi đồ thị G (biến đổi ma trận C) về việc biến đổi hay dãy số Fx và Fy . Việc trừ Δ vào trọng số tất cả những cạnh liên thuộc với $X[i]$ tương đương với việc tăng $Fx[i]$ lên Δ . Việc cộng Δ vào trọng số tất cả những cạnh liên thuộc với $Y[j]$ tương đương với giảm $Fy[j]$ đi Δ . Khi cần biết trọng số cạnh $(X[i], Y[j])$ là bao nhiêu sau các bước biến đổi, thay vì viết $c[i, j]$, ta viết $c[i, j] - Fx[i] - Fy[j]$.

Ví dụ: Thủ tục tìm đường mòn trong thuật toán Hungari đòi hỏi phải xác định được cạnh nào là 0_cạnh, khi cài đặt bằng phương pháp Kuhn-Munkres, việc xác định cạnh nào là 0_cạnh có thể kiểm tra bằng đẳng thức: $c[i, j] - Fx[i] - Fy[j] = 0$ hay $c[i, j] = Fx[i] + Fy[j]$.

Sơ đồ cài đặt phương pháp Kuhn-Munkres có thể viết như sau:

Bước 1: Khởi tạo:

$$M := \emptyset;$$

Việc khởi tạo các Fx , Fy có thể có nhiều cách chẳng hạn $Fx[i] := 0$; $Fy[j] := 0$ với $\forall i, j$.

Hoặc: $Fx[i] := \min_{1 \leq j \leq k}(c[i, j])$ với $\forall i$. Sau đó đặt $Fy[j] := \min_{1 \leq i \leq k}(c[i, j] - Fx[i])$ với $\forall j$.

(Miễn sao $c[i, j] - Fx[i] - Fy[j] \geq 0$)

Bước 2: Với mọi đỉnh $x^* \in X$, ta tìm cách ghép x^* như sau:

Bắt đầu từ đỉnh x^* , thử tìm đường mở bắt đầu ở x^* bằng thuật toán tìm kiếm trên đồ thị (BFS hoặc DFS). Lưu ý rằng $0_{\text{cạnh}}$ là cạnh thoả mãn $c[i, j] = Fx[i] + Fy[j]$. Có hai khả năng xảy ra:

- Hoặc tìm được đường mở thì dọc theo đường mở, ta loại bỏ những cạnh đã ghép khỏi M và thêm vào M những cạnh chưa ghép.
- Hoặc không tìm được đường mở thì xác định được hai tập:
 - ❖ VisitedX = {Tập những X_đỉnh có thể đến được từ x^* bằng một đường pha}
 - ❖ VisitedY = {Tập những Y_đỉnh có thể đến được từ x^* bằng một đường pha}
 - ❖ Đặt $\Delta := \min\{c[i, j] - Fx[i] - Fy[j] \mid \forall X[i] \in \text{VisitedX}; \forall Y[j] \notin \text{VisitedY}\}$
 - ❖ Với $\forall X[i] \in \text{VisitedX}: Fx[i] := Fx[i] + \Delta;$
 - ❖ Với $\forall Y[j] \in \text{VisitedY}: Fy[j] := Fy[j] - \Delta;$
 - ❖ Lặp lại thủ tục tìm đường mở xuất phát tại x^* cho tới khi tìm ra đường mở.

Đáng lưu ý ở phương pháp Kuhn-Munkres là nó không làm thay đổi ma trận C ban đầu. Điều đó thực sự hữu ích trong trường hợp trọng số của cạnh ($X[i], Y[j]$) không được cho một cách tường minh bằng giá trị $C[i, j]$ mà lại cho bằng hàm $c(i, j)$: trong trường hợp này, việc trừ hàng/cộng cột trực tiếp trên ma trận chi phí C là không thể thực hiện được.

2. Dưới đây ta sẽ cài đặt chương trình giải bài toán phân công bằng thuật toán Hungari với phương pháp đối ngẫu Kuhn-Munkres:

a) Biểu diễn bộ ghép

Để biểu diễn bộ ghép, ta sử dụng hai mảng: $\text{matchX}[1..k]$ và $\text{matchY}[1..k]$.

- $\text{matchX}[i]$ là đỉnh thuộc tập Y ghép với đỉnh $X[i]$
- $\text{matchY}[j]$ là đỉnh thuộc tập X ghép với đỉnh $Y[j]$.

Tức là nếu như cạnh ($X[i], Y[j]$) thuộc bộ ghép thì $\text{matchX}[i] = j$ và $\text{matchY}[j] = i$.

Quy ước rằng:

- Nếu như $X[i]$ chưa ghép với đỉnh nào của tập Y thì $\text{matchX}[i] = 0$
- Nếu như $Y[j]$ chưa ghép với đỉnh nào của tập X thì $\text{matchY}[j] = 0$.
- Để thêm một cạnh ($X[i], Y[j]$) vào bộ ghép thì chỉ việc đặt $\text{matchX}[i] := j$ và $\text{matchY}[j] := i$;
- Để loại một cạnh ($X[i], Y[j]$) khỏi bộ ghép thì chỉ việc đặt $\text{matchX}[i] := 0$ và $\text{matchY}[j] := 0$;

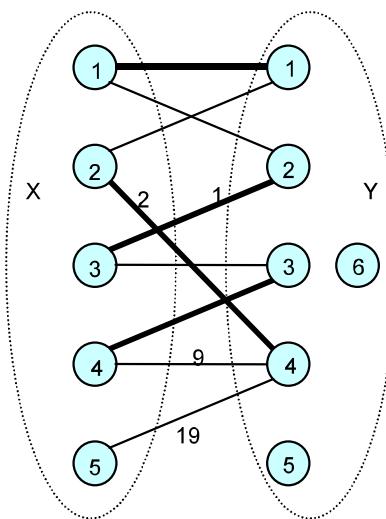
b) Tìm đường mở như thế nào

Ta sẽ tìm đường mở và xây dựng hai tập VisitedX và VisitedY bằng thuật toán tìm kiếm theo chiều rộng chỉ xét tới những đỉnh và những $0_{\text{cạnh}}$ đã định hướng như đã nói trong phần đầu:

Khởi tạo một hàng đợi (Queue) ban đầu chỉ có một đỉnh x^* . Thuật toán tìm kiếm theo chiều rộng làm việc theo nguyên tắc lấy một đỉnh v khỏi Queue và lại đẩy Queue những nối từ v chưa được thăm. Như vậy nếu thăm tới một $Y_{\text{đỉnh}}$ chưa ghép thì tức là ta tìm đường mở kết thúc ở $Y_{\text{đỉnh}}$ chưa ghép đó, quá trình tìm kiếm dừng ngay. Còn nếu ta thăm tới một đỉnh $y \in Y$ đã ghép, dựa vào sự kiện: **từ y chỉ có thể tới được $\text{matchY}[y]$** theo duy nhất một $0_{\text{cạnh}}$ định hướng, nên ta có thể **đánh dấu thăm y, thăm luôn cả $\text{matchY}[y]$, và đẩy vào Queue phần tử $\text{matchY}[y] \in X$** .

3. Nhập dữ liệu từ file văn bản ASSIGN.INP

- Dòng 1: Ghi hai số m, n theo thứ tự là số thợ và số việc cách nhau 1 dấu cách ($m, n \leq 100$)
- Các dòng tiếp theo, mỗi dòng ghi ba số $i, j, c[i, j]$ cách nhau 1 dấu cách thể hiện thợ i làm được việc j và chi phí để làm là $c[i, j]$ ($1 \leq i \leq m; 1 \leq j \leq n; 0 \leq c[i, j] \leq 100$).



| ASSIGN.INP | ASSIGN.OUT |
|--|--|
| <pre> 5 6 1 1 0 1 2 0 2 1 0 2 4 2 3 2 1 3 3 0 4 3 0 4 4 9 5 4 9 </pre> | <p>Optimal assignment:</p> <ol style="list-style-type: none"> 1) $X[1] - Y[1]$ 0 2) $X[2] - Y[4]$ 2 3) $X[3] - Y[2]$ 1 4) $X[4] - Y[3]$ 0 <p>Cost: 3</p> |

PROG12_1.PAS * Thuật toán Hungari

```

program AssignmentProblemSolve;
const
  max = 100;
  maxC = 10001;
var
  c: array[1..max, 1..max] of Integer;
  Fx, Fy, matchX, matchY, Trace: array[1..max] of Integer;
  m, n, k, start, finish: Integer; {đường mở sẽ bắt đầu từ start ∈ X và kết thúc ở finish ∈ Y}

procedure Enter; {Nhập dữ liệu từ thiết bị nhập chuẩn (Input)}
var
  i, j: Integer;
begin
  ReadLn(m, n);
  if m > n then k := m else k := n;
  for i := 1 to k do
    for j := 1 to k do c[i, j] := maxC;
  while not SeekEof do ReadLn(i, j, c[i, j]);
end;

procedure Init; {Khởi tạo}
var
  i, j: Integer;
begin
  {Bộ ghép rỗng}
  FillChar(matchX, SizeOf(matchX), 0);
  FillChar(matchY, SizeOf(matchY), 0);
  {Fx[i]:= Trọng số nhỏ nhất của các cạnh liên thuộc với X[i]}
  for i := 1 to k do
    begin
      Fx[i] := maxC;
      for j := 1 to k do
        if c[i, j] < Fx[i] then Fx[i] := c[i, j];
    end;
  {Fy[j]:= Trọng số nhỏ nhất của các cạnh liên thuộc với Y[j]}
  for j := 1 to k do
    begin
      Fy[j] := maxC;
      for i := 1 to k do {Lưu ý là trọng số cạnh (x[i], y[j]) bây giờ là c[i, j] - Fx[i] chứ không còn là c[i, j] nữa}
        if c[i, j] - Fx[i] < Fy[j] then Fy[j] := c[i, j] - Fx[i];
    end;
  {Việc khởi tạo các Fx và Fy như thế này chỉ đơn giản là để cho số 0_cạnh trở nên càng nhiều càng tốt mà thôi}
  {Ta hoàn toàn có thể khởi gán các Fx và Fy bằng giá trị 0}

```

```

end;
{Hàm cho biết trọng số cạnh (X[i], Y[j])}
function GetC(i, j: Integer): Integer;
begin
  GetC := c[i, j] - Fx[i] - Fy[j];
end;

procedure FindAugmentingPath; {Tìm đường mở bắt đầu ở start}
var
  Queue: array[1..max] of Integer;
  i, j, first, last: Integer;
begin
  FillChar(Trace, SizeOf(Trace), 0); {Trace[j] = X_đỉnh liền trước Y[j] trên đường mở}
  {Thuật toán BFS}
  Queue[1] := start; {Đẩy start vào hàng đợi}
  first := 1; last := 1;
  repeat
    i := Queue[first]; Inc(first); {Lấy một đỉnh X[i] khỏi hàng đợi}
    for j := 1 to k do {Duyệt những Y_đỉnh chưa thăm kè với X[i] qua một 0_cạnh chưa ghép}
      if (Trace[j] = 0) and (GetC(i, j) = 0) then
        begin
          Trace[j] := i; {Lưu vết đường đi, cùng với việc đánh dấu (=0) luôn}
          if matchY[j] = 0 then {Nếu j chưa ghép thì ghi nhận nơi kết thúc đường mở và thoát luôn}
            begin
              finish := j;
              Exit;
            end;
          Inc(last); Queue[last] := matchY[j]; {Đẩy luôn matchY[j] vào Queue}
        end;
    until first > last; {Hàng đợi rỗng}
  end;

procedure SubX_Ady; {Xoay các trọng số cạnh}
var
  i, j, t, Delta: Integer;
  VisitedX, VisitedY: set of Byte;
begin
  (* Đέ ý rằng:
   VisitedY = {y | Trace[y] ≠ 0}
   VisitedX = {start} ∪ match(VisitedY) = {start} ∪ {matchY[y] | Trace[y] ≠ 0}
   *)
  VisitedX := [start];
  VisitedY := [];
  for j := 1 to k do
    if Trace[j] <> 0 then
      begin
        Include(VisitedX, matchY[j]);
        Include(VisitedY, j);
      end;
  {Sau khi xác định được VisitedX và VisitedY, ta tìm Δ là trọng số nhỏ nhất của cạnh nối từ VisitedX ra Y\VisitedY}
  Delta := maxC;
  for i := 1 to k do
    if i in VisitedX then
      for j := 1 to k do
        if not (j in VisitedY) and (GetC(i, j) < Delta) then
          Delta := GetC(i, j);
  {Xoay trọng số cạnh}
  for t := 1 to k do
    begin
      {Trừ trọng số những cạnh liên thuộc với VisitedX đi Delta}
      if t in VisitedX then Fx[t] := Fx[t] + Delta;
      {Cộng trọng số những cạnh liên thuộc với VisitedY lên Delta}
      if t in VisitedY then Fy[t] := Fy[t] - Delta;
    end;
end;

```

```

end;
{Nói rộng bộ ghép bởi đường mỏ tìm được}
procedure Enlarge;
var
  x, next: Integer;
begin
  repeat
    x := Trace[finish];
    next := matchX[x];
    matchX[x] := finish;
    matchY[finish] := x;
    finish := Next;
  until finish = 0;
end;

procedure Solve; {Thuật toán Hungari}
var
  x, y: Integer;
begin
  for x := 1 to k do
    begin
      start := x; finish := 0; {Khởi gán nơi xuất phát đường mỏ, finish = 0 nghĩa là chưa tìm thấy đường mỏ}
      repeat
        FindAugmentingPath; {Thử tìm đường mỏ}
        if finish = 0 then SubX_AddY; {Nếu không thấy thì xoay các trọng số cạnh và lặp lại}
      until finish <> 0; {Cho tới khi tìm thấy đường mỏ}
      Enlarge; {Tăng cặp dựa trên đường mỏ tìm được}
    end;
end;

procedure Result;
var
  x, y, Count, W: Integer;
begin
  WriteLn('Optimal assignment:');
  W := 0; Count := 0;
  for x := 1 to m do {In ra phép phân công thì chỉ cần xét đến m, không cần xét đến k}
    begin
      y := matchX[x];
      {Những cạnh có trọng số maxC tương ứng với một thợ không được giao việc và một việc không được phân công}
      if c[x, y] < maxC then
        begin
          Inc(Count);
          WriteLn(Count:5, ' ', x, ' - ', y, ' ', c[x, y]);
          W := W + c[x, y];
        end;
    end;
  WriteLn('Cost: ', W);
end;

```

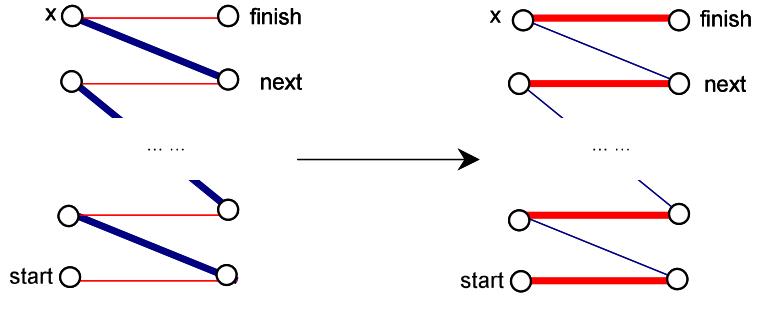
```

begin
  Assign(Input, 'ASSIGN.INP'); Reset(Input);
  Assign(Output, 'ASSIGN.OUT'); Rewrite(Output);
  Enter;
  Init;
  Solve;
  Result;
  Close(Input);
  Close(Output);
end.

```

Nhận xét:

1. Nếu cài đặt như trên thì cho dù đồ thị đã 2aphê mang trọng số âm, chương trình vẫn tìm được bộ ghép cực đại với trọng số cực tiểu. Lý do: Ban đầu, ta trừ tất cả các phần tử trên mỗi hàng



của ma trận C đi giá trị nhỏ nhất trên hàng đó, rồi lại trừ tất cả các phần tử trên mỗi cột của ma trận C đi giá trị nhỏ nhất trên cột đó (Phép trừ ở đây làm gián tiếp qua các F_x , F_y chứ không phải trừ trực tiếp trên ma trận C). Nên sau bước này, tất cả các cạnh của đồ thị sẽ có trọng số không âm bởi phần tử nhỏ nhất trên mỗi cột của C chắc chắn là 0.

2. Sau khi kết thúc thuật toán, tổng tất cả các phần tử ở hai dãy F_x , F_y bằng trọng số cực tiểu của bộ ghép đầy đủ tìm được trên đồ thị ban đầu.
3. Một vấn đề nữa phải hết sức cẩn thận trong việc ước lượng độ lớn của các phần tử F_x và F_y . Nếu như giả thiết cho các trọng số không quá 500 thì ta không thể dựa vào bất đẳng thức $F_x(x) + F_y(y) \leq c(x, y)$ mà khẳng định các phần tử trong F_x và F_y cũng ≤ 500 . Hãy tự tìm ví dụ để hiểu rõ hơn bản chất thuật toán.

V. BÀI TOÁN TÌM BỘ GHÉP CỰC ĐẠI VỚI TRỌNG SỐ CỰC ĐẠI TRÊN ĐỒ THỊ HAI PHÍA

Bài toán tìm bộ ghép cực đại với trọng số cực đại cũng có thể giải nhờ phương pháp Hungari bằng cách đổi dấu tất cả các phần tử ma trận chi phí (Nhờ nhận xét 1).

Khi cài đặt, ta có thể sửa lại đôi chút trong chương trình trên để giải bài toán tìm bộ ghép cực đại với trọng số cực đại mà không cần đổi dấu trọng số. Cụ thể như sau:

Bước 1: Khởi tạo:

- $M := \emptyset$;
- Khởi tạo hai dãy F_x và F_y thỏa mãn: $\forall i, j: F_x[i] + F_y[j] \geq c[i, j]$; Chẳng hạn ta có thể đặt $F_x[i] :=$ Phần tử lớn nhất trên dòng i của ma trận C và đặt các $F_y[j] := 0$.

Bước 2: Với mọi đỉnh $x^* \in X$, ta tìm cách ghép x^* như sau:

Với cách hiểu 0_cạnh là cạnh thoả mãn $c[i, j] = F_x[i] + F_y[j]$. Bắt đầu từ đỉnh x^* , thử tìm đường mở bắt đầu ở x^* . Có hai khả năng xảy ra:

- Hoặc tìm được đường mở thì dọc theo đường mở, ta loại bỏ những cạnh đã ghép khỏi M và thêm vào M những cạnh chưa ghép.
- Hoặc không tìm được đường mở thì xác định được hai tập:
 - ❖ VisitedX = {Tập những X_đỉnh có thể đến được từ x^* bằng một đường pha}
 - ❖ VisitedY = {Tập những Y_đỉnh có thể đến được từ x^* bằng một đường pha}
 - ❖ Đặt $\Delta := \min\{F_x[i] + F_y[j] - c[i, j] \mid \forall X[i] \in VisitedX; \forall Y[j] \notin VisitedY\}$
 - ❖ Với $\forall X[i] \in VisitedX: F_x[i] := F_x[i] - \Delta$;
 - ❖ Với $\forall Y[j] \in VisitedY: F_y[j] := F_y[j] + \Delta$;
 - ❖ Lặp lại thủ tục tìm đường mở xuất phát tại x^* cho tới khi tìm ra đường mở.

Bước 3: Sau bước 2 thì mọi X_đỉnh đều đã ghép, ta được một bộ ghép đầy đủ k cạnh với trọng số lớn nhất.

Dễ dàng chứng minh được tính đúng đắn của phương pháp, bởi nếu ta đặt:

$$c'[i, j] = -c[i, j]; F'x[i] := -F_x[i]; F'y[j] = -F_y[j].$$

Thì bài toán trở thành tìm cặp ghép đầy đủ trọng số cực tiểu trên đồ thị hai phía với ma trận trọng số $c'[1..k, 1..k]$. Bài toán này được giải quyết bằng cách tính hai dãy đối ngẫu $F'x$ và $F'y$. Từ đó bằng những biến đổi đại số cơ bản, ta có thể kiểm chứng được tính tương đương giữa các bước của phương pháp nêu trên với các bước của phương pháp Kuhn-Munkres ở mục trước.

VI. ĐỘ PHÚC TẠP TÍNH TOÁN

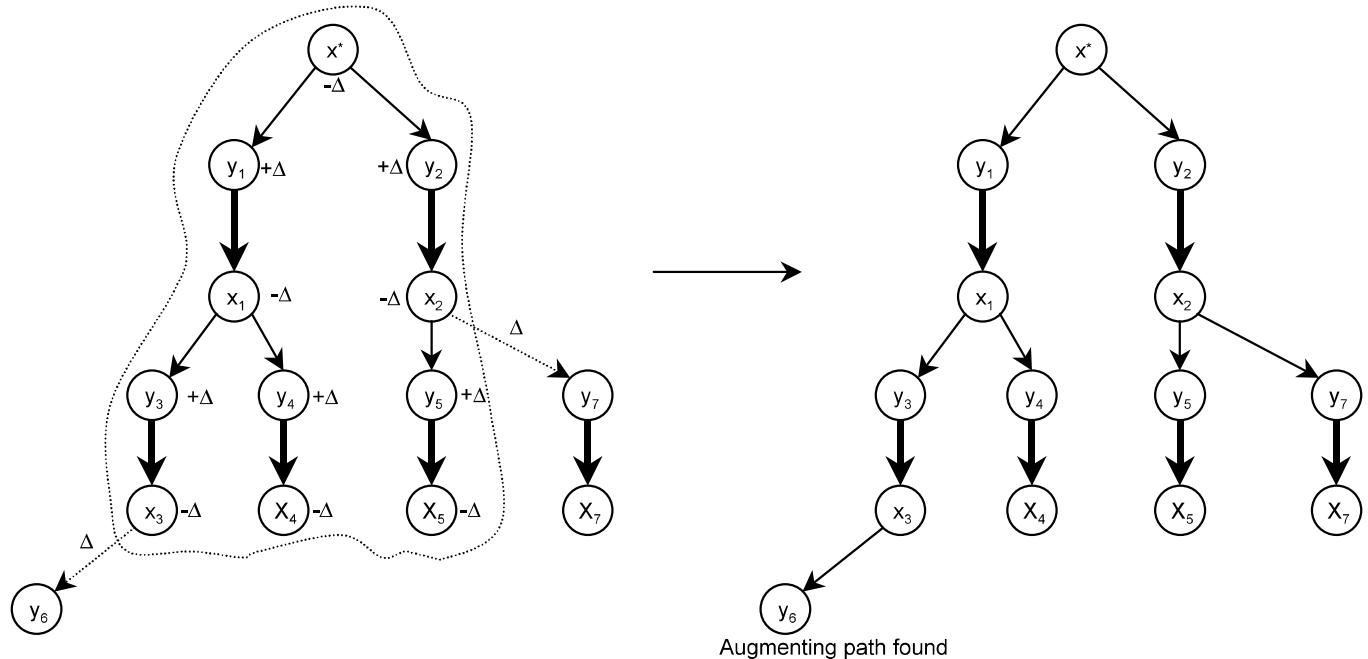
Dựa vào mô hình cài đặt thuật toán Kuhn-Munkres ở trên, ta có thể đánh giá về độ phức tạp tính toán lý thuyết của cách cài đặt này:

Thuật toán tìm kiếm theo chiều rộng được sử dụng để tìm đường mở có độ phức tạp $O(k^2)$, mỗi lần xoay trọng số cạnh mất một chi phí thời gian cỡ $O(k^2)$. Vậy mỗi lần tăng cặp, cần tối đa k lần dò đường và k lần xoay trọng số cạnh, mất một chi phí thời gian cỡ $O(k^3)$. Thuật toán cần k lần tăng cặp nên độ phức tạp tính toán trên lý thuyết của phương pháp này cỡ $O(k^4)$.

Có thể cải tiến mô hình cài đặt để được một thuật toán với độ phức tạp $O(k^3)$ dựa trên những nhận xét sau:

Nhận xét 1:

Quá trình tìm kiếm theo chiều rộng bắt đầu từ một đỉnh x^* chưa ghép cho ta một cây pha gốc x^* . Nếu tìm được đường mở thì dừng lại và tăng cặp ngay, nếu không thì xoay trọng số cạnh và bắt đầu tìm kiếm lại để được một cây pha mới lớn hơn cây pha cũ:



Hình 23: Cây pha "mọc" lớn hơn sau mỗi lần xoay trọng số cạnh và tìm đường

Nhận xét 2:

Việc xác định trọng số nhỏ nhất của cạnh nối một $X_{\text{đỉnh}}$ trong cây pha với một $Y_{\text{đỉnh}}$ ngoài cây pha có thể kết hợp ngay trong bước dựng cây pha mà không làm tăng cấp phức tạp tính toán. Để thực hiện điều này, ta sử dụng kỹ thuật như trong thuật toán Prim:

Với mọi $y \in Y$, gọi $d[y] :=$ khoảng cách từ y đến cây pha gốc x^* . Ban đầu $d[y]$ được khởi tạo bằng trọng số cạnh $(x^*, y) = c[x^*, y] - Fx[x^*] - Fy[y]$ (cây pha ban đầu chỉ có đúng một đỉnh x^*).

Trong bước tìm đường bằng BFS, mỗi lần rút một đỉnh x ra khỏi Queue, ta xét những đỉnh $y \in Y$ chưa thăm và đặt lại $d[y]_{\text{mới}} := \min(d[y]_{\text{cũ}}, \text{trọng số cạnh } (x, y))$ sau đó mới kiểm tra xem (x, y) có phải là 0_cạnh hay không để tiếp tục các thao tác như trước. Nếu quá trình BFS không tìm ra đường mở thì giá trị xoay Δ chính là giá trị nhỏ nhất trong các $d[y]$ dương. Ta bót được một đoạn chương trình tìm giá trị xoay có độ phức tạp $O(k^2)$. Công việc tại mỗi bước xoay chỉ là tìm giá trị nhỏ nhất trong các $d[y]$ dương và thực hiện phép cộng, trừ trên hai dãy đối ngẫu Fx và Fy , nó có độ phức tạp tính toán $O(k)$, tối đa có k lần xoay để tìm đường mở nên tổng chi phí thời gian thực hiện các lần xoay cho tới khi tìm ra đường mở cỡ $O(k^2)$. Lưu ý rằng đồ thị đang xét là đồ thị hai chiều đủ nêu sau khi xoay các trọng số cạnh bằng giá trị xoay Δ , tất cả các cạnh nối từ $X_{\text{đỉnh}}$ trong cây pha tới

$Y_{\text{đỉnh}}$ ngoài cây pha đều bị giảm trọng số đi Δ , chính vì vậy ta phải trừ tất cả các $d[y] > 0$ đi Δ để giữ được tính hợp lý của các $d[y]$.

Nhận xét 3:

Ta có thể tận dụng kết quả của quá trình tìm kiếm theo chiều rộng ở bước trước để nói rộng cây pha cho bước sau (grow alternating tree) mà không phải tìm lại từ đầu (BFS lại bắt đầu từ x^*).

Khi không tìm thấy đường mở, quá trình tìm kiếm theo chiều rộng sẽ đánh dấu được những đỉnh đã thăm (thuộc cây pha) và hàng đợi các $X_{\text{đỉnh}}$ trong quá trình tìm kiếm trở thành rỗng. Tiếp theo là phải xác định được $\Delta = \text{trọng số nhỏ nhất của cạnh nối một } X_{\text{đỉnh}} \text{ đã thăm với một } Y_{\text{đỉnh}} \text{ chưa thăm và xoay các trọng số cạnh để những cạnh này trở thành 0_cạnh}$. Tại đây ta sẽ dùng kỹ thuật sau: Thăm luôn những đỉnh $y \in Y$ chưa thăm tạo với một $X_{\text{đỉnh}}$ đã thăm một 0_cạnh (những $Y_{\text{đỉnh}}$ chưa thăm có $d[y] = 0$), nếu tìm thấy đường mở thì dừng ngay, nếu không thấy thì đầy tiếp những đỉnh $\text{match}_Y[y]$ vào hàng đợi và lặp lại thuật toán tìm kiếm theo chiều rộng bắt đầu từ những đỉnh này. Vậy nếu xét tổng thể, mỗi lần tăng cặp ta chỉ thực hiện một lần dựng cây pha, tức là tổng chi phí thời gian của những lần thực hiện giải thuật tìm kiếm trên đồ thị sau mỗi lần tăng cặp chỉ còn là $O(k^2)$.

Nhận xét 4:

Thủ tục tăng cặp dựa trên đường mở (Enlarge) có độ phức tạp $O(k)$

Từ 3 nhận xét trên, phương pháp đối ngẫu Kuhn-Munkres có thể cài đặt bằng một chương trình có độ phức tạp tính toán $O(k^3)$ bởi nó cần k lần tăng cặp và chi phí cho mỗi lần là $O(k^2)$.

PROG12_2.PAS * Cài đặt phương pháp Kuhn-Munkres $O(n^3)$

```
program AssignmentProblemSolve;
const
  max = 100;
  maxC = 10001;
var
  c: array[1..max, 1..max] of Integer;
  Fx, Fy, matchX, matchY: array[1..max] of Integer;
  Trace, Queue, d, arg: array[1..max] of Integer;
  first, last: Integer;
  start, finish: Integer;
  m, n, k: Integer;

procedure Enter;           {Nhập dữ liệu}
var
  i, j: Integer;
begin
  ReadLn(m, n);
  if m > n then k := m else k := n;
  for i := 1 to k do
    for j := 1 to k do c[i, j] := maxC;
  while not SeekEof do ReadLn(i, j, c[i, j]);
end;

procedure Init;           {Khởi tạo bộ ghép rỗng và hai dãy đối ngẫu Fx, Fy}
var
  i, j: Integer;
begin
  FillChar(matchX, SizeOf(matchX), 0);
  FillChar(matchY, SizeOf(matchY), 0);
  for i := 1 to k do
    begin
      Fx[i] := maxC;
      for j := 1 to k do
        if c[i, j] < Fx[i] then Fx[i] := c[i, j];
    end;
end;
```

```

    end;
  for j := 1 to k do
    begin
      Fy[j] := maxC;
      for i := 1 to k do
        if c[i, j] - Fx[i] < Fy[j] then Fy[j] := c[i, j] - Fx[i];
    end;
  end;

function GetC(i, j: Integer);           {Hàm trả về trọng số cạnh (X[i], Y[j])}
begin
  GetC := c[i, j] - Fx[i] - Fy[j];
end;

procedure InitBFS;          {Thủ tục khởi tạo trước khi tìm cách ghép start∈X}
var
  y: Integer;
begin
  {Hàng đợi chỉ gồm mỗi một đỉnh Start ⇔ cây pha khởi tạo chỉ có 1 đỉnh start}
  first := 1; last := 1;
  Queue[1] := start;
  {Khởi tạo các Y_đỉnh đều chưa thăm ⇔ Trace[y] = 0, ∀y}
  FillChar(Trace, SizeOf(Trace), 0);
  {Khởi tạo các d[y]}
  for y := 1 to k do
    begin
      d[y] := GetC(start, y);           {d[y] là khoảng cách từ y tới cây pha gốc start}
      arg[y] := start;                  {arg[y] là X_đỉnh thuộc cây pha tạo ra khoảng cách đó}
    end;
  finish := 0;
end;

procedure Push(v: Integer);         {Đẩy một đỉnh v∈X vào hàng đợi}
begin
  Inc(last); Queue[last] := v;
end;

function Pop: Integer;             {Rút một X_đỉnh khỏi hàng đợi, trả về trong kết quả hàm}
begin
  Pop := Queue[first]; Inc(first);
end;

procedure FindAugmentingPath;       {Thủ tục tìm đường mở}
var
  i, j, w: Integer;
begin
  repeat
    i := Pop;                         {Rút một đỉnh X[i] khỏi hàng đợi}
    for j := 1 to k do                {Quét những Y_đỉnh chưa thăm}
      if Trace[j] = 0 then
        begin
          w := GetC(i, j);            {xét cạnh (X[i], Y[j])}
          if w = 0 then              {Nếu là 0_cạnh}
            begin
              Trace[j] := i;          {Lưu vết đường đi}
              if matchY[j] = 0 then   {Nếu j chưa ghép thì ghi nhận nơi kết thúc đường mở và thoát}
                begin
                  finish := j;
                  Exit;
                end;
              Push(matchY[j]);        {Nếu j đã ghép thì đẩy tiếp matchY[j] vào hàng đợi}
            end;
        end;
    end;
    if d[j] > w then    {Cập nhật lại khoảng cách d[j] nếu thấy cạnh (X[i], Y[j]) ngắn hơn khoảng cách này}
      begin

```

```

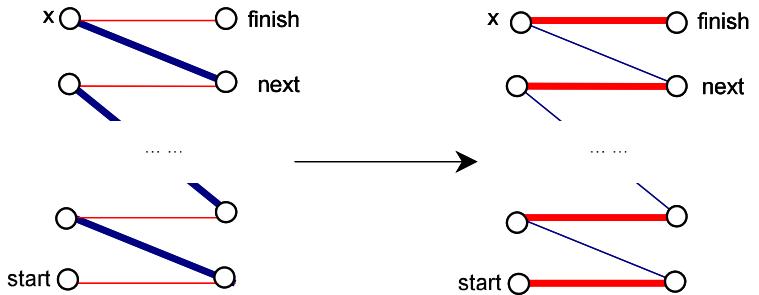
        d[j] := w;
        arg[j] := i;
    end;
end;
until first > last;
end;

{Xoay các trọng số cạnh}
procedure SubX_AddY;
var
    Delta: Integer;
    x, y: Integer;
begin
    {Trước hết tính Δ = giá trị nhỏ nhất trọng số các d[y], với y ∈ Y chưa thăm (y không thuộc cây pha)}
    Delta := maxC;
    for y := 1 to k do
        if (Trace[y] = 0) and (d[y] < Delta) then Delta := d[y];
    {Trừ trọng số những cạnh liên thuộc với start ∈ X đi Δ}
    Fx[start] := Fx[start] + Delta;
    for y := 1 to k do          {Xét các đỉnh y ∈ Y}
        if Trace[y] <> 0 then   {Nếu y thuộc cây pha}
            begin
                x := matchY[y];      {Thì x = matchY[y] cũng phải thuộc cây pha}
                Fy[y] := Fy[y] - Delta;    {Cộng trọng số những cạnh liên thuộc với y lên Δ}
                Fx[x] := Fx[x] + Delta;    {Trừ trọng số những cạnh liên thuộc với x đi Δ}
            end
        else
            d[y] := d[y] - Delta; {Nếu y ∉ cây pha thì sau bước xoay, khoảng cách từ y đến cây pha sẽ giảm Δ}
    {Chuẩn bị tiếp tục BFS}
    for y := 1 to k do
        if (Trace[y] = 0) and (d[y] = 0) then {Thăm luôn những đỉnh y ∈ Y tạo với cây pha một 0_cạnh}
            begin
                Trace[y] := arg[y];      {Lưu vết đường đi}
                if matchY[y] = 0 then     {Nếu y chưa ghép thì ghi nhận đỉnh kết thúc đường mở và thoát ngay}
                    begin
                        finish := y;
                        Exit;
                    end;
                Push(matchY[y]);         {Nếu y đã ghép thì đẩy luôn matchY[y] vào hàng đợi để chờ loang tiếp}
            end;
    end;

procedure Enlarge; {Nối rộng bộ ghép bằng đường mở kết thúc ở finish}
var
    x, next: Integer;
begin
repeat
    x := Trace[finish];
    next := matchX[x];
    matchX[x] := finish;
    matchY[finish] := x;
    finish := Next;
until finish = 0;
end;

procedure Solve;
var
    x, y: Integer;
begin
    for x := 1 to k do      {Với mỗi X_đỉnh: }
        begin
            start := x;      {Đặt nơi khởi đầu đường mở}
            InitBFS;          {Khởi tạo cây pha}
            repeat

```



```

    FindAugmentingPath;           {Tim đường mở}
    if finish = 0 then SubX_Addy;   {Nếu không thấy thì xoay các trọng số cạnh ...}
    until finish <> 0;           {Cho tới khi tìm ra đường mở}
    Enlarge;          {Nới rộng bộ ghép bởi đường mở tìm được}
  end;
end;

procedure Result;
var
  x, y, Count, W: Integer;
begin
  WriteLn('Optimal assignment:');
  W := 0; Count := 0;
  for x := 1 to m do      {Với mỗi X_dịnh, xét cặp ghép tương ứng}
    begin
      y := matchX[x];
      if c[x, y] < maxC then    {Chỉ quan tâm đến những cặp ghép có trọng số < maxC}
        begin
          Inc(Count);
          WriteLn(Count:5, ' ', x, ' - ', y, ' ', c[x, y]);
          W := W + c[x, y];
        end;
    end;
  WriteLn('Cost: ', W);
end;

begin
  Assign(Input, 'ASSIGN.INP'); Reset(Input);
  Assign(Output, 'ASSIGN.OUT'); Rewrite(Output);
  Enter;
  Init;
  Solve;
  Result;
  Close(Input);
  Close(Output);
end.

```

§13. BÀI TOÁN TÌM BỘ GHÉP CỰC ĐẠI TRÊN ĐỒ THỊ

I. CÁC KHÁI NIỆM

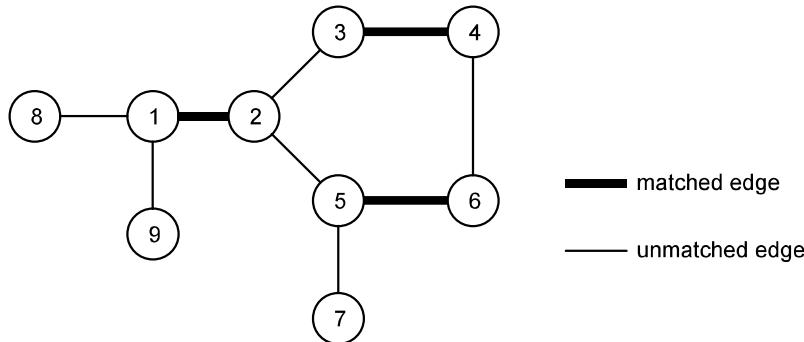
Xét đồ thị $G = (V, E)$, một bộ ghép trên đồ thị G là một tập các cạnh đôi một không có đỉnh chung. Bài toán tìm bộ ghép cực đại trên đồ thị tổng quát phát biểu như sau:

Cho một đồ thị G , phải tìm một bộ ghép cực đại trên G (bộ ghép có nhiều cạnh nhất).

Với một bộ ghép M của đồ thị G , ta gọi:

- Những cạnh thuộc M được gọi là cạnh đã ghép hay **cạnh đậm**
- Những cạnh không thuộc M được gọi là cạnh chưa ghép hay **cạnh nhạt**
- Những đỉnh đầu mút của các cạnh đậm được gọi là **đỉnh đã ghép**, những đỉnh còn lại gọi là **đỉnh chưa ghép**
- Một đường đi cơ bản (đường đi không có đỉnh lặp lại) được gọi là **đường pha** nếu nó bắt đầu bằng một cạnh nhạt và tiếp theo là các cạnh đậm, nhạt nằm nối tiếp xen kẽ nhau.
- Một chu trình cơ bản (chu trình không có đỉnh trong lặp lại) được gọi là một **Blossom** nếu nó đi qua ít nhất 3 đỉnh, bắt đầu và kết thúc bằng cạnh nhạt và dọc trên chu trình, các cạnh đậm, nhạt nằm nối tiếp xen kẽ nhau. Đỉnh xuất phát của chu trình (cũng là đỉnh kết thúc) được gọi là **đỉnh cơ sở** (base) của Blossom.
- **Đường mở** là một đường pha bắt đầu ở một đỉnh chưa ghép và kết thúc ở một đỉnh chưa ghép.

Ví dụ: Với đồ thị G và bộ ghép M dưới đây:



Hình 24: Đồ thị G và một bộ ghép M

- Đường $(8, 1, 2, 5, 6, 4)$ là một đường pha
- Chu trình $(2, 3, 4, 6, 5, 2)$ là một Blossom
- Đường $(8, 1, 2, 3, 4, 6, 5, 7)$ là một đường mở
- Đường $(8, 1, 2, 3, 4, 6, 5, 2, 1, 9)$ tuy có các cạnh đậm/nhạt xen kẽ nhưng không phải đường pha (và tất nhiên không phải đường mở) vì đây không phải là đường đi cơ bản.

Ta dễ dàng suy ra được các tính chất sau

- Đường mở cũng như Blossom đều là đường đi độ dài lẻ với số cạnh nhạt nhiều hơn số cạnh đậm đúng 1 cạnh.
- Trong mỗi Blossom, những đỉnh không phải đỉnh cơ sở đều là đỉnh đã ghép và đỉnh ghép với đỉnh đó cũng phải thuộc Blossom.
- Vì Blossom là một chu trình nên trong mỗi Blossom, những đỉnh không phải đỉnh cơ sở đều tồn tại hai đường pha từ đỉnh cơ sở đi đến nó, một đường kết thúc bằng cạnh đậm và một đường kết thúc bằng cạnh nhạt, hai đường pha này được hình thành bằng cách đi dọc theo chu trình theo hai hướng ngược nhau. Như ví dụ trên, đỉnh 4 có hai đường pha đi đỉnh cơ sở 2 đi tới: $(2, 3, 4)$ là đường pha kết thúc bằng cạnh đậm và $(2, 5, 6, 4)$ là đường pha kết thúc bằng cạnh nhạt

II. THUẬT TOÁN EDMONDS (1965)

Cơ sở của thuật toán là định lý (C.Berge): Một bộ ghép M của đồ thị G là cực đại khi và chỉ khi không tồn tại đường mở đối với M.

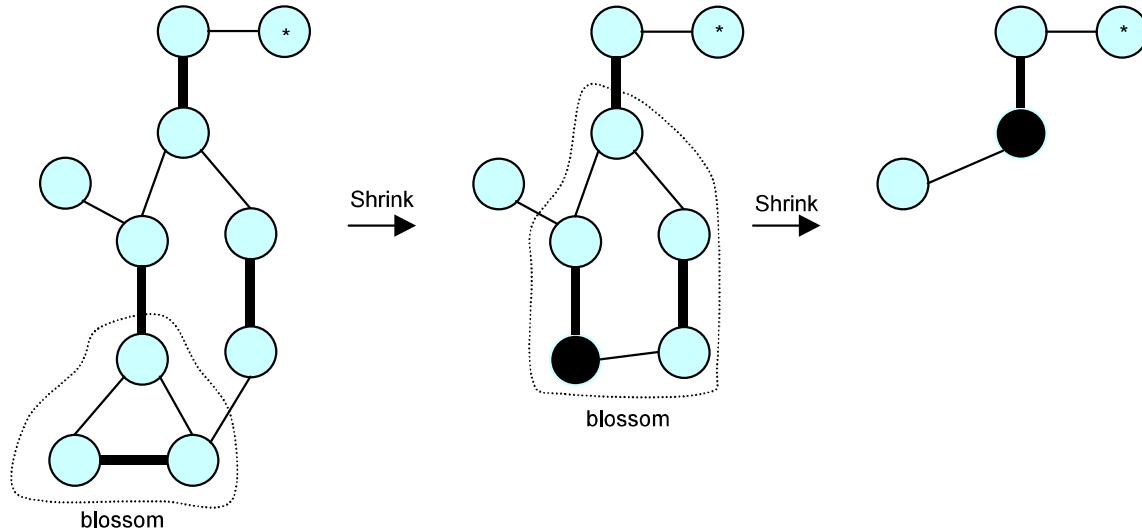
Thuật toán Edmonds:

```
M := Ø;
for (Và đỉnh u chưa ghép) do
    if <Tìm đường mở xuất phát từ u> then
        <
        Đọc trên đường mở:
        Loại bỏ những cạnh đậm khỏi M;
        Thêm vào M những cạnh nhạt;
        >
Result: M là bộ ghép cực đại trên G
```

Điều khó nhất trong thuật toán Edmonds là phải xây dựng thuật toán tìm đường mở xuất phát từ một đỉnh chưa ghép. Thuật toán đó được xây dựng bằng cách kết hợp một thuật toán tìm kiếm trên đồ thị với phép chập Blossom.

Xét những đường pha xuất phát từ một đỉnh x chưa ghép. Những đỉnh có thể đến được từ x bằng một đường pha kết thúc là cạnh nhạt được gán nhãn "nhạt", những đỉnh có thể đến được từ x bằng một đường pha kết thúc là cạnh đậm được gán nhãn "đậm".

Với một Blossom, ta định nghĩa phép chập (shrink) là phép thay thế các đỉnh trong Blossom bằng một đỉnh duy nhất. Những cạnh nối giữa một đỉnh thuộc Blossom tới một đỉnh v nào đó không thuộc Blossom được thay thế bằng cạnh nối giữa đỉnh chập này với v và giữ nguyên tính đậm/nhạt. Để thấy rằng sau mỗi phép chập, các cạnh đậm vẫn được đảm bảo là bộ ghép trên đồ thị mới:



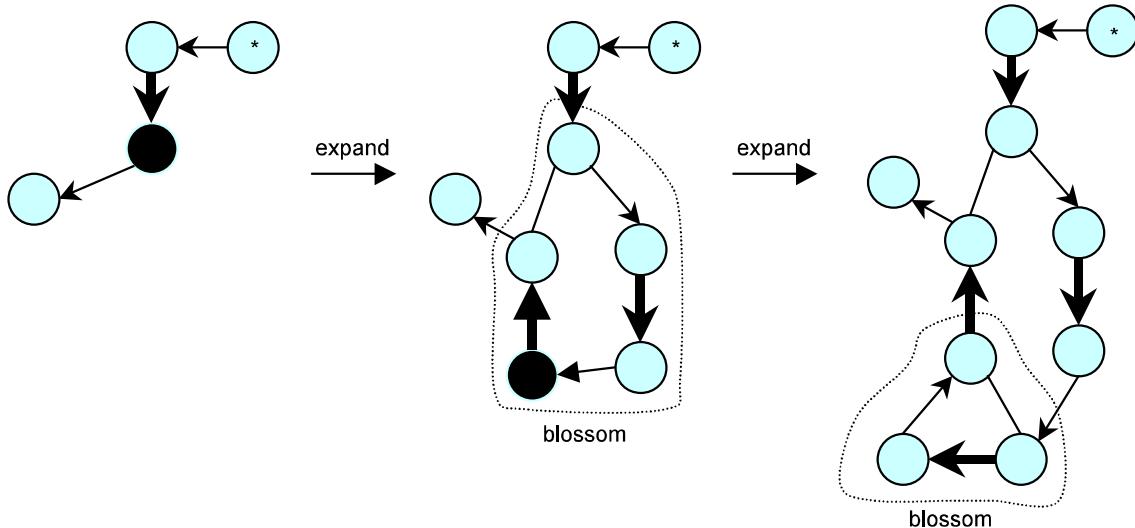
Hình 25: Phép chập Blossom

Thuật toán tìm đường mở có thể phát biểu như sau.

- Trước hết đỉnh xuất phát x được gán nhãn đậm.
- Tiếp theo là thuật toán tìm kiếm trên đồ thị bắt đầu từ x, theo nguyên tắc: từ đỉnh đậm chỉ được phép đi tiếp theo cạnh nhạt và từ đỉnh nhạt chỉ được đi tiếp theo cạnh đậm. Mỗi khi thăm tới một đỉnh, ta gán nhãn đậm/nhạt cho đỉnh đó và tiếp tục thao tác tìm kiếm trên đồ thị như bình thường. Cũng trong quá trình tìm kiếm, mỗi khi phát hiện thấy một cạnh nhạt nối hai đỉnh đậm, ta dừng lại ngay vì nếu gán nhãn tiếp sẽ gặp tình trạng một đỉnh có cả hai nhãn đậm/nhạt, trong trường hợp này, Blossom được phát hiện (xem tính chất của Blossom) và bị chập thành một

đỉnh, thuật toán được bắt đầu lại với đồ thị mới cho tới khi trả lời được câu hỏi: "có tồn tại đường mở xuất phát từ x hay không?"

- Nếu đường mở tìm được không đi qua đỉnh chập nào thì ta chỉ việc tăng cắp dọc theo đường mở. Nếu đường mở có đi qua một đỉnh chập thì ta lại nở đỉnh chập đó ra thành Blossom để thay đỉnh chập này trên đường mở bằng một đoạn đường xuyên qua Blossom:



Hình 26: Nở Blossom để dò đường xuyên qua Blossom

Lưu ý rằng không phải Blossom nào cũng bị chập, chỉ những Blossom ảnh hưởng tới quá trình tìm đường mở mới phải chập để đảm bảo rằng đường mở tìm được là đường đi cơ bản. Tuy nhiên việc cài đặt trực tiếp các phép chập Blossom và nở đỉnh khá rắc rối, đòi hỏi một chương trình với độ phức tạp $O(n^4)$.

Dưới đây ta sẽ trình bày một phương pháp cài đặt hiệu quả hơn với độ phức tạp $O(n^3)$, phương pháp này cài đặt không phức tạp, nhưng yêu cầu phải hiểu rất rõ bản chất thuật toán.

III. PHƯƠNG PHÁP LAWLER (1973)

Trong phương pháp Edmonds, sau khi chập mỗi Blossom thành một đỉnh thì đỉnh đó hoàn toàn lại có thể nằm trên một Blossom mới và bị chập tiếp. Phương pháp Lawler chỉ quan tâm đến đỉnh chập cuối cùng, đại diện cho Blossom ngoài nhất (Outermost Blossom), đỉnh chập cuối cùng này được định danh (đánh số) bằng đỉnh cơ sở của Blossom ngoài nhất.

Cũng chính vì thao tác chập/nở nói trên mà ta cần mở rộng khái niệm Blossom, có thể **coi một Blossom là một tập đỉnh nở ra từ một đỉnh chập** chứ không đơn thuần chỉ là một chu trình pha cơ bản nữa.

Xét một Blossom B có đỉnh cơ sở là đỉnh r . Với $\forall v \in B, v \neq r$, ta lưu lại hai đường pha từ r tới v , một đường kết thúc bằng cạnh đậm và một đường kết thúc bằng cạnh nhạt, như vậy có hai loại vết gãy cho mỗi đỉnh v :

- $S[v]$ là đỉnh liền trước v trên đường pha kết thúc bằng cạnh đậm, nếu không tồn tại đường pha loại này thì $S[v] = 0$.
- $T[v]$ là đỉnh liền trước v trên đường pha kết thúc bằng cạnh nhạt, nếu không tồn tại đường pha loại này thì $T[v] = 0$.

Bên cạnh hai nhãn S và T , mỗi đỉnh v còn có thêm

- Nhãn $b[v]$ là đỉnh cơ sở của Blossom chứa v . Hai đỉnh u và v thuộc cùng một Blossom $\Leftrightarrow b[u] = b[v]$.
- Nhãn $match[v]$ là đỉnh ghép với đỉnh v . Nếu v chưa ghép thì $match[v] = 0$.

Khi đó thuật toán tìm đường mở bắt đầu từ đỉnh x chưa ghép có thể phát biểu như sau:

Bước 1: (Init)

- Hàng đợi Queue dùng để chứa những đỉnh đậm chờ duyệt, ban đầu chỉ gồm một đỉnh đậm x.
- Với mọi đỉnh u, khởi gán $b[u] = u$ và $\text{match}[u] = 0$ với $\forall u$.
- Gán $S[x] \neq 0$; Với $\forall u \neq x$, gán $S[u] = 0$; Với $\forall v$: gán $T[v] = 0$

Bước 2: (BFS)

Lặp lại các bước sau cho tới khi hàng đợi rỗng:

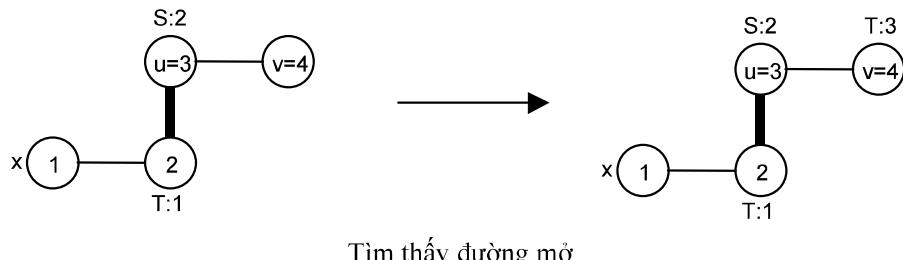
Với mỗi đỉnh đậm u lấy ra từ Queue, xét những cạnh nhạt (u, v):

- Nếu v chưa thăm:
 - ◆ Nếu v là đỉnh chưa ghép \Rightarrow Tìm thấy đường mở kết thúc ở v, dừng
 - ◆ Nếu v là đỉnh đã ghép \Rightarrow thăm v \Rightarrow thăm luôn $\text{match}[v]$ và đẩy $\text{match}[v]$ vào Queue.
- Sau mỗi lần thăm, chú ý việc lưu vết (hai nhãn S và T)
- Nếu v đã thăm
 - ◆ Nếu v là đỉnh nhạt hoặc $b[v] = b[u] \Rightarrow$ bỏ qua
 - ◆ Nếu v là đỉnh đậm và $b[v] \neq b[u]$ ta phát hiện được blossom mới chứa u và v, khi đó:
 - **Phát hiện đỉnh cơ sở:** Truy vết đường đi ngược từ hai đỉnh đậm u và v theo hai đường pha về nút gốc, chọn lấy đỉnh a là đỉnh đậm chung gấp đầu tiên trong quá trình truy vết ngược. Khi đó Blossom mới phát hiện sẽ có đỉnh cơ sở là a.
 - **Gán lại vết:** Gọi $(a = i_1, i_2, \dots, i_p = u)$ và $(a = j_1, j_2, \dots, j_q = v)$ lần lượt là hai đường pha dẫn từ a tới u và v. Khi đó $(a = i_1, i_2, \dots, i_p = u, j_q = v, j_{q-1}, \dots, j_1 = a)$ là một chu trình pha đi từ a tới u và v rồi quay trở về a. Bằng cách đi dọc theo chu trình này theo hai hướng ngược nhau, ta có thể gán lại tất cả các nhãn S và T của những đỉnh trên chu trình. Lưu ý rằng **không được gán lại nhãn S và T** cho những đỉnh k mà $b[k] = a$, và với những đỉnh k có $b[k] \neq a$ thì **bắt buộc phải gán lại nhãn S và T** theo chu trình này bắt kể $S[k]$ và $T[k]$ trước đó đã có hay chưa.
 - **Chập Blossom:** Xét những đỉnh v mà $b[v] \in \{b[i_1], b[i_2], \dots, b[i_p], b[j_1], b[j_2], \dots, b[j_q]\}$, gán lại $b[v] = a$. Nếu v là đỉnh đậm (có nhãn $S[v] \neq 0$) mà chưa được duyệt tới (chưa bao giờ được đẩy vào Queue) thì đẩy v vào Queue chờ duyệt tiếp tại những bước sau.

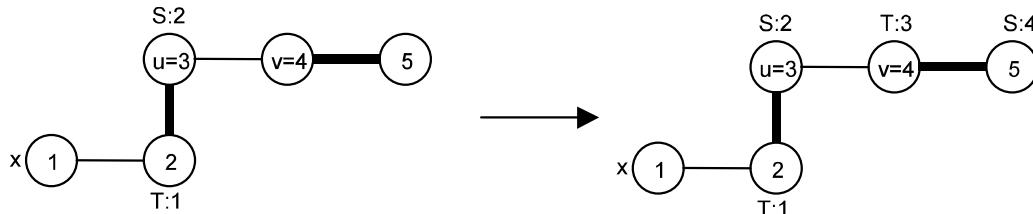
Nếu quá trình này chỉ thoát khi hàng đợi rỗng thì tức là không tồn tại đường mở bắt đầu từ x.

Sau đây là một số ví dụ về các trường hợp từ đỉnh đậm u xét cạnh nhạt (u, v):

Trường hợp 1: v chưa thăm và chưa ghép:

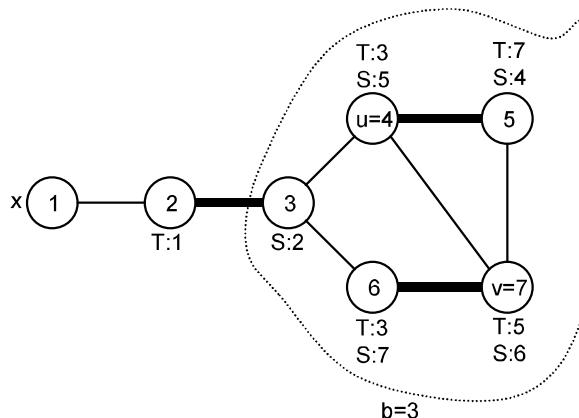


Trường hợp 2: v chưa thăm và đã ghép



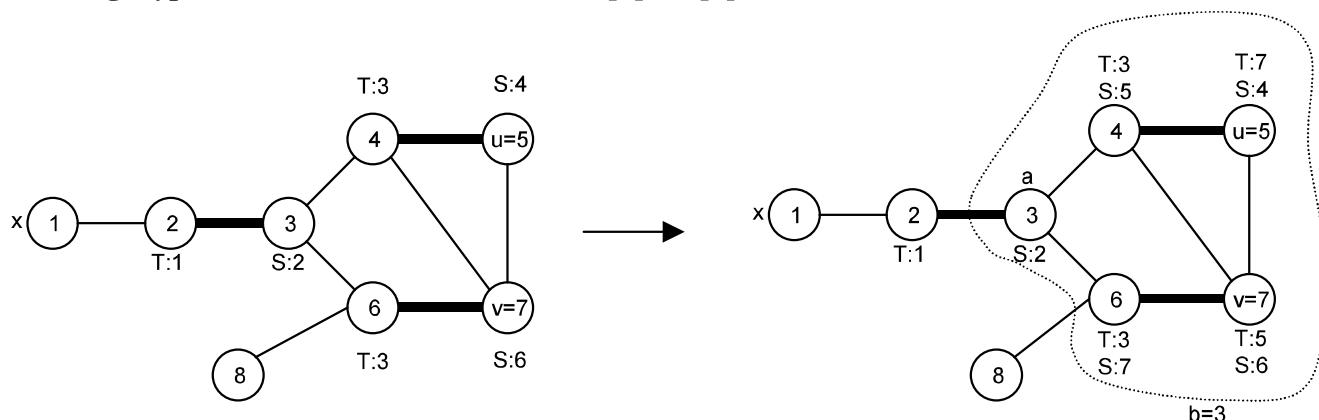
Thăm cả v lần $\text{match}[v]$, gán nhãn $T[v]$ và $S[\text{match}[v]]$

Trường hợp 3: v đã thăm, là đỉnh đậm thuộc cùng blossom với u



Không xét, bỏ qua

Trường hợp 4: v đã thăm, là đỉnh đậm và $b[u] \neq b[v]$



Tìm đỉnh cơ sở $a = 3$, gán lại nhãn S và T dọc chu trình pha, chập Blossom.

Đây hai đỉnh đậm mới 4, 6 vào hàng đợi, Tại những bước sau,

khi duyệt tới đỉnh 6, sẽ tìm thấy đường mở kết thúc ở 8,
truy vết theo nhãn S và T tìm được đường $(1, 2, 3, 4, 5, 7, 6, 8)$

Tư tưởng chính của phương pháp Lawler là dùng các nhãn $b[v]$ thay cho thao tác chập trực tiếp Blossom, dùng các nhãn S và T để truy vết tìm đường mở, tránh thao tác nổ Blossom. Phương pháp này dựa trên một nhận xét: Mỗi khi tìm ra đường mở, nếu đường mở đó xuyên qua một Blossom ngoài nhất thì chắc chắn nó phải đi vào Blossom này từ nút cơ sở và thoát ra khỏi Blossom bằng một cạnh nhạt.

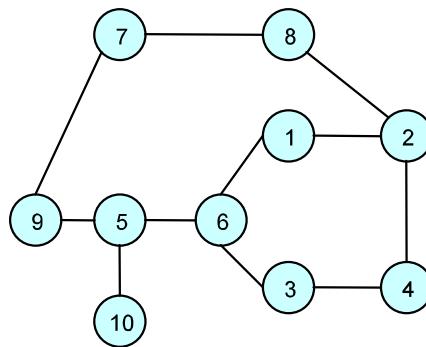
IV. CÀI ĐẶT

Ta sẽ cài đặt phương pháp Lawler với khuôn dạng Input/Output như sau:

Input: file văn bản GMATCH.INP

- Dòng 1: Chứa hai số n, m lần lượt là số cạnh và số đỉnh của đồ thị cách nhau ít nhất một dấu cách ($n \leq 100$)
- m dòng tiếp theo, mỗi dòng chứa hai số u, v ¹²³ tượng trưng cho một cạnh (u, v) của đồ thị

Output: file văn bản GMATCH.OUT chứa bộ ghép cực đại tìm được



| GMATCH.INP | GMATCH.OUT |
|------------|----------------------------|
| 10 11 | 1 6 |
| 1 2 | 2 8 |
| 1 6 | 3 4 |
| 2 4 | 5 10 |
| 2 8 | 7 9 |
| 3 4 | Number of matched edges: 5 |
| 3 6 | |
| 5 6 | |
| 5 9 | |
| 5 10 | |
| 7 8 | |
| 7 9 | |

Chương trình này sửa đổi một chút mô hình cài đặt trên dựa vào nhận xét:

- v là một đỉnh đậm $\Leftrightarrow v = x$ hoặc $\text{match}[v]$ là một đỉnh nhạt
- Nếu v là đỉnh đậm thì $S[v] = \text{match}[v]$

Vậy thì ta không cần phải sử dụng riêng một mảng nhãn $S[v]$, tại mỗi bước sửa vết, ta chỉ cần sửa nhãn vết $T[v]$ mà thôi. Để kiểm tra một đỉnh $v \neq x$ có phải đỉnh đậm hay không, ta có thể kiểm tra bằng điều kiện: $\text{match}[v]$ có là đỉnh nhạt hay không, hay $T[\text{match}[v]]$ có khác 0 hay không.

Chương trình sử dụng các biến với vai trò như sau:

- $\text{match}[v]$ là đỉnh ghép với đỉnh v
- $b[v]$ là đỉnh cơ sở của Blossom chứa v
- $T[v]$ là đỉnh liền trước v trên đường pha từ đỉnh xuất phát tới v kết thúc bằng cạnh nhạt, $T[v] = 0$ nếu quá trình BFS chưa xét tới đỉnh nhạt v .
- $\text{InQueue}[v]$ là biến Boolean, $\text{InQueue}[v] = \text{True} \Leftrightarrow v$ là đỉnh đậm đã được đẩy vào Queue để chờ duyệt.
- start và finish: Nơi bắt đầu và kết thúc đường mở.

PROG13_1.PAS * Phương pháp Lawler áp dụng cho thuật toán Edmonds

```

program MatchingInGeneralGraph;
const
  max = 100;
var
  a: array[1..max, 1..max] of Boolean;
  match, Queue, b, T: array[1..max] of Integer;
  InQueue: array[1..max] of Boolean;
  n, first, last, start, finish: Integer;

procedure Enter; {Nhập dữ liệu, từ thiết bị nhập chuẩn (Input)}
var
  i, m, u, v: Integer;
begin
  FillChar(a, SizeOf(a), 0);
  ReadLn(n, m);
  for i := 1 to m do
    begin
      ReadLn(u, v);
      a[u, v] := True;
      a[v, u] := True;
    end;
end;

procedure Init; {Khởi tạo bộ ghép rỗng}

```

```

begin
  FillChar(match, SizeOf(match), 0);
end;

procedure InitBFS; {Thủ tục này được gọi để khởi tạo trước khi tìm đường mỏ xuất phát từ start}
var
  i: Integer;
begin
  {Hàng đợi chỉ gồm một đỉnh đậm start}
  first := 1; last := 1;
  Queue[1] := start;
  FillChar(InQueue, SizeOf(InQueue), False);
  InQueue[start] := True;
  {Các nhãn T được khởi gán = 0}
  FillChar(T, SizeOF(T), 0);
  {Nút cơ sở của outermost blossom chứa i chính là i}
  for i := 1 to n do b[i] := i;
  finish := 0; {finish = 0 nghĩa là chưa tìm thấy đường mỏ}
end;

procedure Push(v: Integer); {Đẩy một đỉnh đậm v vào hàng đợi}
begin
  Inc(last);
  Queue[last] := v;
  InQueue[v] := True;
end;

function Pop: Integer; {Lấy một đỉnh đậm khỏi hàng đợi, trả về trong kết quả hàm}
begin
  Pop := Queue[first];
  Inc(first);
end;

{Khó nhất của phương pháp Lawler là thủ tục này: Thủ tục xử lý khi gặp cạnh nhạt nối hai đỉnh đậm p, q}
procedure BlossomShrink(p, q: Integer);
var
  i, NewBase: Integer;
  Mark: array[1..max] of Boolean;

{Thủ tục tìm nút cơ sở bằng cách truy vết ngược theo đường pha từ p và q}
function FindCommonAncestor(p, q: Integer): Integer;
var
  InPath: array[1..max] of Boolean;
begin
  FillChar(InPath, SizeOf(Inpath), False);
  repeat {Truy vết từ p}
    p := b[p]; {Nhảy tới nút cơ sở của Blossom chứa p, phép nhảy này để tăng tốc độ truy vết}
    Inpath[p] := True; {Đánh dấu nút đó}
    if p = start then Break; {Nếu đã truy về đến nơi xuất phát thì dừng}
    p := T[match[p]]; {Nếu chưa về đến start thì truy lùi tiếp hai bước, theo cạnh đậm rồi theo cạnh nhạt}
  until False;
  repeat {Truy vết từ q, tương tự như đối với p}
    q := b[q];
    if InPath[q] then Break; {Tuy nhiên nếu chạm vào đường pha của p thì dừng ngay}
    q := T[match[q]];
  until False;
  FindCommonAncestor := q; {Ghi nhận đỉnh cơ sở mới}
end;

procedure ResetTrace(x: Integer); {Gán lại nhãn vết dọc trên đường pha từ start tới x}
var
  u, v: Integer;
begin
  v := x;

```

```

while b[v] <> NewBase do {Truy vết đường pha từ start tới đỉnh đậm x}
begin
  u := match[v];
  Mark[b[v]] := True; {Đánh dấu nhãn blossom của các đỉnh trên đường đi}
  Mark[b[u]] := True;
  v := T[u];
  if b[v] <> NewBase then T[v] := u; {Chỉ đặt lại vết T[v] nếu b[v] không phải nút cơ sở mới}
end;
end;

begin {BlossomShrink}
FillChar(Mark, SizeOf(Mark), False); {Tất cả các nhãn b[v] đều chưa bị đánh dấu}
NewBase := FindCommonAncestor(p, q); {xác định nút cơ sở}
{Gán lại nhãn}
ResetTrace(p); ResetTrace(q);
if b[p] <> NewBase then T[p] := q;
if b[q] <> NewBase then T[q] := p;
{Chập blossom  $\Leftrightarrow$  gán lại các nhãn b[i] nếu blossom b[i] bị đánh dấu}
for i := 1 to n do
  if Mark[b[i]] then b[i] := NewBase;
{Xét những đỉnh đậm i chưa được đưa vào Queue nằm trong Blossom mới, đẩy i và Queue để chờ duyệt tiếp tại các bước sau}
for i := 1 to n do
  if not InQueue[i] and (b[i] = NewBase) then
    Push(i);
end;

{Thủ tục tìm đường mở}
procedure FindAugmentingPath;
var
  u, v: Integer;
begin
  InitBFS; {Khởi tạo}
  repeat {BFS}
    u := Pop;
    {Xét những đỉnh v chưa duyệt, kè với u, không nằm cùng Blossom với u, dĩ nhiên T[v] = 0 thì (u, v) là cạnh nhạt rồi}
    for v := 1 to n do
      if (T[v] = 0) and (a[u, v]) and (b[u] <> b[v]) then
        begin
          if match[v] = 0 then {Nếu v chưa ghép thì ghi nhận đỉnh kết thúc đường mở và thoát ngay}
            begin
              T[v] := u;
              finish := v;
              Exit;
            end;
          end;
        {Nếu v là đỉnh đậm thì gán lại vết, chập Blossom ...}
        if (v = start) or (T[match[v]] <> 0) then
          BlossomShrink(u, v)
        else {Nếu không thì ghi vết đường đi, thăm v, thăm luôn cả match[v] và đẩy tiếp match[v] vào Queue}
          begin
            T[v] := u;
            Push(match[v]);
          end;
        end;
      until first > last;
  end;

procedure Enlarge; {Nối rộng bộ ghép bởi đường mở bắt đầu từ start, kết thúc ở finish}
var
  v, next: Integer;
begin
  repeat
    v := T[finish];
    next := match[v];
    match[v] := finish;

```

```

        match[finish] := v;
        finish := next;
        until finish = 0;
end;

procedure Solve;      {Thuật toán Edmonds}
var
    u: Integer;
begin
    for u := 1 to n do
        if match[u] = 0 then
            begin
                start := u;           {Với mỗi đỉnh chưa ghép start}
                FindAugmentingPath; {Tìm đường mở bắt đầu từ start}
                if finish <> 0 then Enlarge; {Nếu thấy thì nói rộng bộ ghép theo đường mở này}
            end;
    end;

procedure Result;          {In bộ ghép tìm được}
var
    u, count: Integer;
begin
    count := 0;
    for u := 1 to n do
        if match[u] > u then {Vừa tránh sự trùng lặp (u, v) và (v, u), vừa loại những đỉnh không ghép được (match=0)}
            begin
                Inc(count);
                WriteLn(u, ' ', match[u]);
            end;
    WriteLn('Number of matched edges: ', count);
end;

begin
    Assign(Input, 'GMATCH.INP'); Reset(Input);
    Assign(Output, 'GMATCH.OUT'); Rewrite(Output);
    Enter;
    Init;
    Solve;
    Result;
    Close(Input);
    Close(Output);
end.

```

V. ĐỘ PHÚC TẠP TÍNH TOÁN

- Thủ tục BlossomShrink có độ phức tạp $O(n)$.
- Thủ tục FindAugmentingPath cần không quá n lần gọi thủ tục BlossomShrink, cộng thêm chi phí của thuật toán tìm kiếm theo chiều rộng, có độ phức tạp $O(n^2)$
- Phương pháp Lawler cần không quá n lần gọi thủ tục FindAugmentingPath nên có độ phức tạp tính toán là $O(n^3)$

TỔNG HỢP MỘT SỐ BÀI TẬP THỰC HÀNH MÔN LÝ THUYẾT ĐỒ THỊ

BÀI 1.

Cho đồ thị vô hướng liên thông G có n đỉnh. Hỏi G có bao nhiêu thành phần liên thông?

Dữ liệu vào:

- Dòng đầu ghi số n
- Trong n dòng tiếp theo mỗi dòng ghi n số (ma trận kề)

Dữ liệu ra: Số thành phần liên thông của G

BÀI 2.

Cho đồ thị vô hướng liên thông G có n đỉnh. Hỏi đường đi ngắn nhất từ đỉnh u đến đỉnh v của G đi qua ít nhất bao nhiêu cạnh?

Dữ liệu vào:

- Dòng đầu ghi 3 số: n , đỉnh u , đỉnh v
- Trong n dòng tiếp theo mỗi dòng ghi n số (ma trận kề)

Dữ liệu ra: Số cạnh của đường đi từ đỉnh u đến đỉnh v .

BÀI 3.

Cho đồ thị vô hướng liên thông có trọng số G có n đỉnh, m cạnh. Tìm cây khung nhỏ nhất của G bằng thuật toán Kruskal.

Dữ liệu vào:

- Dòng đầu ghi 2 số n, m là số đỉnh và số cạnh của đồ thị
- Trong m dòng tiếp theo, mỗi dòng ghi 3 số: là đỉnh đầu, đỉnh cuối và trọng số của cạnh tương ứng (danh sách cạnh).

Dữ liệu ra: Giá trị của cây khung nhỏ nhất tìm được

BÀI 4.

Cho đồ thị vô hướng liên thông có trọng số G có n đỉnh. Tìm cây khung nhỏ nhất của G bằng thuật toán Prim.

Dữ liệu vào:

- Dòng đầu ghi số n
- Trong n dòng tiếp theo, mỗi dòng ghi n số (ma trận trọng số).

Dữ liệu ra: Giá trị của cây khung nhỏ nhất tìm được

BÀI 5.

Cho đồ thị (có hướng/vô hướng) liên thông có trọng số G có n đỉnh. Tìm đường đi ngắn nhất từ đỉnh u đến đỉnh v của G bằng thuật toán Ford-Bellman.

Dữ liệu vào: //có thể có trọng số âm nhưng không có chu trình âm

Dòng đầu ghi 3 số: Số đỉnh n , đỉnh u , đỉnh v

Trong n dòng tiếp theo mỗi dòng ghi n số (ma trận trọng số).

Dữ liệu ra: Giá trị của đường đi ngắn nhất từ u đến v

BÀI 6.

Cho đồ thị (có hướng/vô hướng) liên thông có trọng số G có n đỉnh. Tìm đường đi ngắn nhất từ đỉnh u đến đỉnh v của G bằng thuật toán Dijkstra.

Dữ liệu vào: // trọng số không âm

Dòng đầu ghi 3 số: Số đỉnh n , đỉnh u , đỉnh v .

Trong n dòng tiếp theo mỗi dòng ghi n số (ma trận trọng số)

Dữ liệu ra: Giá trị của đường đi ngắn nhất từ u đến v

BÀI 7.

Cho đồ thị (có hướng/vô hướng) liên thông có trọng số G có n đỉnh. Tìm đường đi ngắn nhất từ đỉnh u đến đỉnh v của G bằng thuật toán Floyd.

Dữ liệu vào: //có thể có trọng số âm nhưng không có chu trình âm

-Dòng đầu ghi số n

-Trong n dòng tiếp theo mỗi dòng ghi n số (ma trận trọng số).

Dữ liệu ra: Ma trận đường đi P và ma trận chi phí ngắn nhất D (cuối cùng).

BÀI 8.

Cho mạng G có n đỉnh, m cạnh, luồng ban đầu trên các cung được cho bằng 0. Tìm luồng cực đại trong mạng G .

Dữ liệu vào:

-Dòng đầu ghi 4 số: n, m , đỉnh phát, đỉnh thu

Trong m dòng tiếp theo mỗi dòng ghi 3 số là đỉnh đầu, đỉnh cuối và khả năng thông qua của cung đó.

Dữ liệu ra: Giá trị luồng cực đại tìm được.

BÀI 9.

Cho đồ thị vô hướng liên thông có trọng số G có n đỉnh. Tìm một chu trình/hoặc đường đi Euler của G (nếu có).

Dữ liệu vào:

- Dòng đầu ghi số n
- Trong n dòng mỗi dòng ghi n số (ma trận kề)

Dữ liệu ra: các đỉnh trên chu trình/đường đi; hoặc 0 nếu không tồn tại.

BÀI 10.

Cho đồ thị vô hướng liên thông có trọng số G có n đỉnh. Tìm một chu trình/hoặc đường đi Hamilton của G (nếu có).

Dữ liệu vào:

- Dòng đầu ghi số n .
- Trong n dòng mỗi dòng ghi n số (ma trận kề).

Dữ liệu ra: các đỉnh trên chu trình/đường đi; hoặc 0 nếu không tồn tại.

HẾT

TIỂU LUẬN MÔN LÝ THUYẾT ĐỒ THỊ

(thay đổi hàng năm)

1. Bài toán max clique
2. Bài toán cây steiner nhỏ nhất
3. Bài toán ghép cặp (trên đồ thị hai phía và trên đồ thị tổng quát)
4. Bài toán luồng trong mạng (luồng cực đại, luồng với chi phí tối thiểu,...)

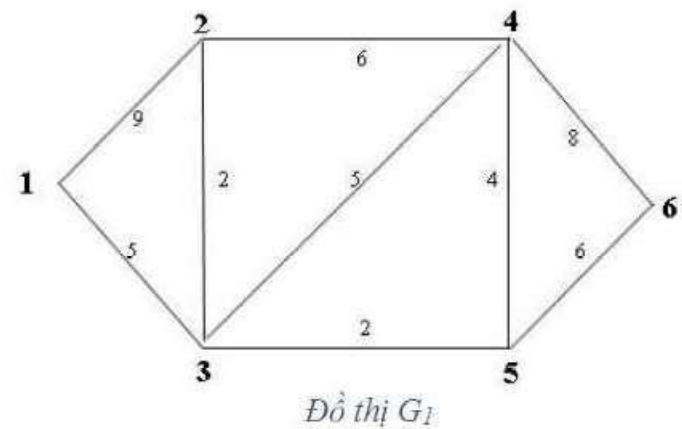
Sinh viên không sử dụng tài liệu – Cán bộ coi thi không giải thích gì đẽ.

Câu 1 (2 điểm).

- a. Cho đồ thị vô hướng liên thông $G = (V, E)$ có n đỉnh, m cạnh. Hãy phát biểu định lý biểu diễn mối liên hệ giữa bậc của đỉnh và số cạnh của đồ thị.
- Áp dụng: Cho đồ thị vô hướng $G = (V, E)$ có 14 đỉnh và 25 cạnh; biết rằng mỗi đỉnh của G đều có bậc là 3 hoặc 5. Hỏi G có bao nhiêu đỉnh bậc 3 ?
- b. Cho đồ thị vô hướng liên thông có trọng số không âm $G = (V, E)$; trong đó tập $|V| = n$, $|E| = m$.
Phát biểu các định nghĩa: cây, cây khung, cây khung nhỏ nhất của đồ thị.
Phát biểu các tính chất cơ bản của cây.

Câu 2 (3 điểm).

- a. Cho đồ thị vô hướng liên thông có trọng số không âm $G = (V, E)$; trong đó tập $|V| = n$, $|E| = m$.
Viết thuật toán Prim tìm cây khung nhỏ nhất của đồ thị.
- b. Cho đồ thị G_1 như hình vẽ bên. Hãy minh họa quá trình tìm cây khung nhỏ nhất (T) của G_1 theo thuật toán Prim.
Vẽ cây khung T và cho biết tổng trọng số các cạnh của cây khung T .

**Câu 3 (3 điểm).**

- a. Cho đồ thị có trọng số không âm $G = (V, E)$; trong đó tập $|V| = n$, $|E| = m$. Viết thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị G .
- b. Hãy minh họa quá trình tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị G_1 (hình vẽ ở câu 2) theo thuật toán Dijkstra; từ đó hãy chỉ ra đường đi ngắn nhất từ đỉnh 1 đến đỉnh 6 và cho biết độ dài của đường đi này.

Câu 4 (2 điểm).

Cho mạng như đồ thị G_2 ở hình bên (1 là đỉnh phát, 6 là đỉnh thu), trọng số trên các cung là khả năng thông qua của nó, luồng ban đầu trên các cung được cho bằng 0.

Hãy minh họa quá trình tìm luồng cực đại trên mạng G_2 bằng thuật toán Ford-Fulkerson và đồng thời cho biết giá trị luồng trên các cung tương ứng.

