GIỚI THIỆU MÔN HỌC

- Tên môn học: Lý thuyết đồ thị
- Số tiết: 30 LT
- Hình thức đánh giá:
  - *Thi giữa kỳ: 20%*
  - *Bài tập lớn: 30%*
  - *Thi cuối kỳ: 50%*

Giáo viên: Nguyễn Văn Lễ

Nội dung

CHƯƠNG 1: CÁC KHÁI NIỆM CƠ BẢN

CHƯƠNG 2: BIỂU DIỄN ĐỒ THỊ TRÊN MÁY TÍNH

CHƯƠNG 3: CÁC THUẬT TOÁN DUYỆT ĐỒ THỊ

CHƯƠNG 4: ĐỒ THỊ EULER VÀ ĐỒ THỊ HAMILTON

CHƯƠNG 5: CÂY

CHƯƠNG 6: BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT

CHƯƠNG 1: CÁC KHÁI NIỆM CƠ BẢN

Định nghĩa đồ thị:

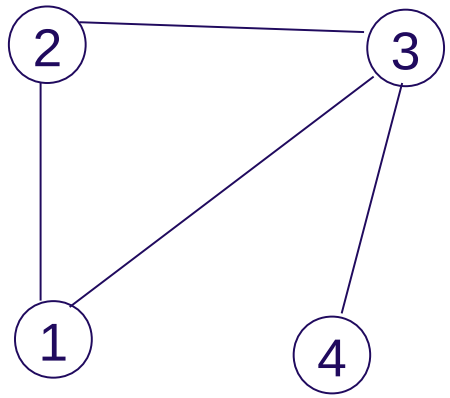
- Một đồ thị ký hiệu là G=(V,E), trong đó

  V: tập đỉnh

  E={(u,v) | u,v∈V}: tập cạnh

  n=|V| gọi là cấp của đồ thị

- **Đồ thị vô hướng:** Là đồ thị gồm các cạnh vô hướng (không thứ tự): (u,v) ∈ E; (v,u) ∈ E

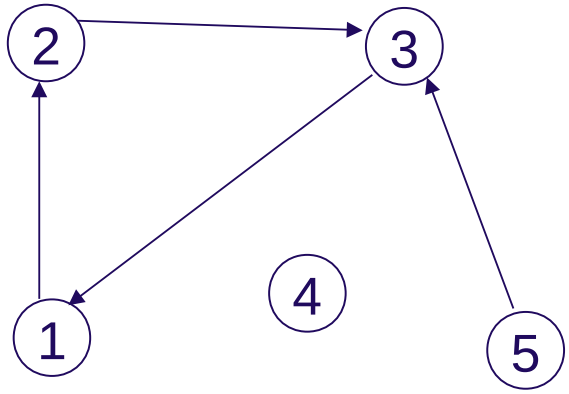


V={1,2,3,4}

E={(1,2), (1,3), (2,3), (3,4)}

Định nghĩa đồ thị

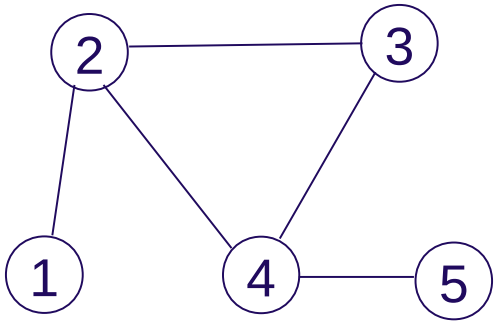
- **Đồ thị có hướng:** là đồ thị gồm các cạnh có thứ tự được gọi là cung.

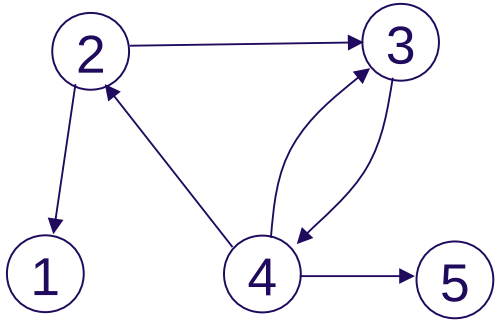


V={1,2,3,4}

E={(1,2),(2,3),(3,1),(5,3)}

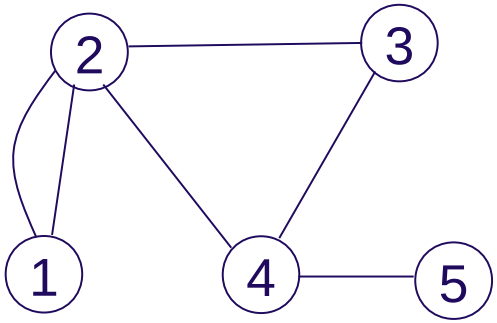
- **Đơn đồ thị:** Mỗi cặp đỉnh chỉ có duy nhất một cạnh (cung)

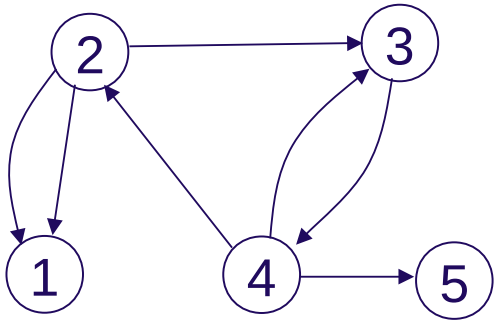




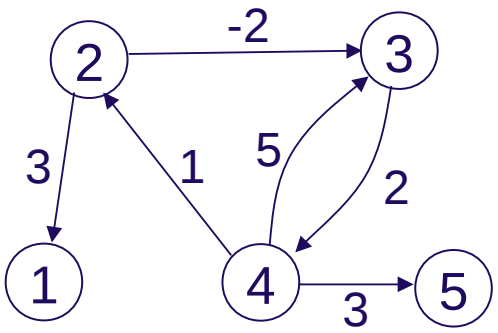
Định nghĩa đồ thị

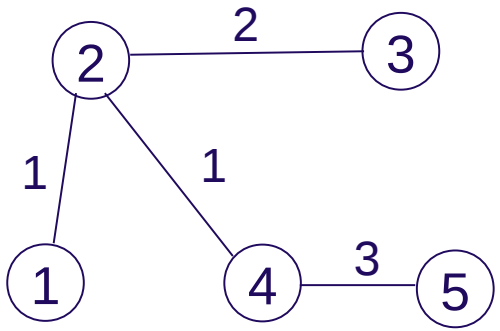
- **Đa đồ thị:** mỗi cặp đỉnh có thể có một hay nhiều cạnh (cung)





- **Đồ thị có trọng số:** trên mỗi cạnh (cung) được gắn một giá trị gọi là trọng số



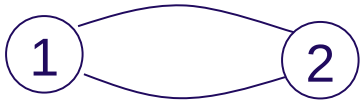


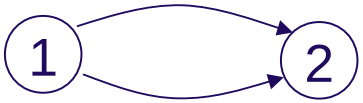
Một số khái niệm

Một số khái niệm:

- **Khuyên:** cạnh (cung) gọi là khuyên nếu đỉnh đầu trùng với đỉnh cuối.

- **Cạnh (cung) lặp:** là hai cạnh (cung) cùng tương ứng với một cặp đỉnh.

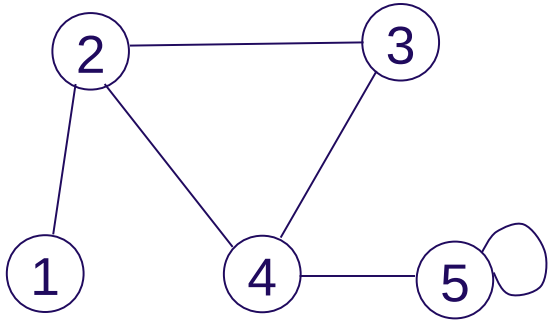




- **Đỉnh kề:** nếu (u,v) là cạnh (cung) của đồ thị thì v gọi là kề của u. Trong đồ thị vô hướng nếu v kề u thì u cũng kề v.

Một số khái niệm

- **Cạnh liên thuộc:** cạnh e=(u,v) gọi là cạnh liên thuộc với hai đỉnh u, v.
- **Bậc của đỉnh:** số cạnh liên thuộc với v gọi là bậc của đỉnh v, kí hiệu là d(v). Bậc của đỉnh có khuyên được cộng thêm 2 cho mỗi khuyên.



d(1)=1

d(2)=3

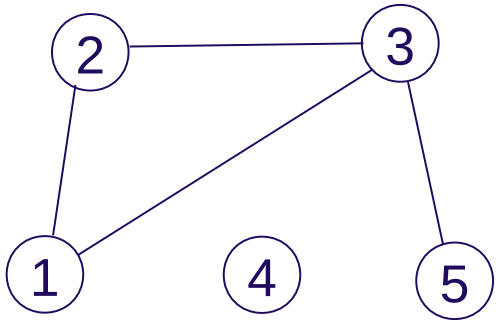
d(3)=2

d(4)=3

d(5)=3

Một số khái niệm

- **Đỉnh cô lập, đỉnh treo:** Đỉnh bậc 0 gọi là đỉnh cô lập, đỉnh bậc 1 gọi là đỉnh treo.



Đỉnh cô lập: 4

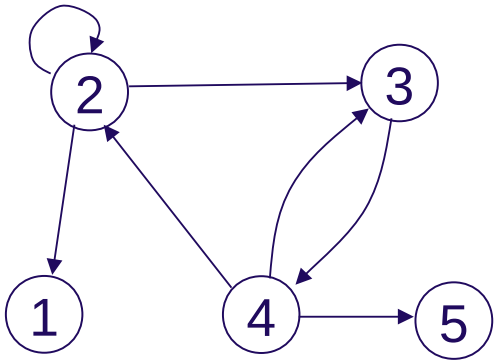
Đỉnh treo: 5

- **Cung vào, ra:** cung e=(u,v) gọi là cung ra khỏi u và là cung vào v.

Cung (1,2) là cung ra của 1 và là cung vào của 2

Một số khái niệm

- **Bán bậc của đỉnh:**
  - Số cung vào của đỉnh v gọi là bán bậc vào của v, kí hiệu $d^-(v)$
  - Số cung ra của đỉnh v gọi là bán bậc ra của v, kí hiệu $d^+(v)$



$d^-(1)=1;\ d^+(1)=0$

$d^-(2)=2;\ d^+(2)=3$

$d^-(3)=2;\ d^+(3)=1$

$d^-(4)=1;\ d^+(4)=3$

$d^-(5)=1;\ d^+(5)=0$

Một số khái niệm

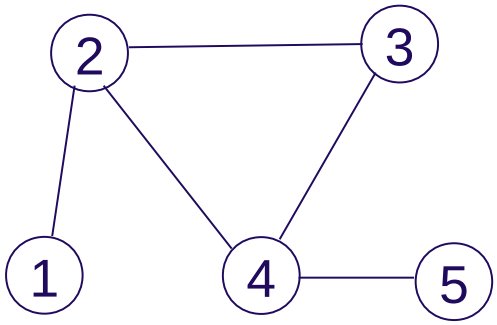
Định lý: Trong đồ thị vô hướng:

Tổng bậc các đỉnh = 2 lần số cạnh.

Chứng minh:

Gọi m là số cạnh, thì cần chứng minh $\sum_{v \in V} d(v) = 2m$

Mỗi cạnh e=(u,v) được tính một lần trong d(u) và một lần trong d(v) trong tổng bậc của các đỉnh, mỗi cạnh được tính hai lần tổng bậc bằng 2m.



Số cạnh: 5

Tổng bậc các đỉnh: 10

Một số khái niệm

Hệ quả: Trong đồ thị vô hướng thì:

Số đỉnh bậc lẻ là một số chẵn

Chứng minh:

Gọi O là tập các đỉnh có bậc là số lẻ, và U là tập các đỉnh có bậc là số chẵn.

Ta có:

$$\sum_{v \in V} d(v) = \sum_{v \in O} d(v) + \sum_{v \in U} d(v) = 2m$$

Do $\forall v \in$ U, deg(v) chẵn nên $\sum_{v \in U} d(v)$ chẵn $\Rightarrow$ $\sum_{v \in O} d(v)$ chẵn

Do $\forall v \in$ O,deg(v) lẻ mà tổng $\sum_{v \in O} d(v)$ chẵn, nên tổng này phải gồm một số chẵn các số hạng

$\Rightarrow$số đỉnh có bậc lẻ là một số chẵn (đpcm).

Một số khái niệm

Định lý 2: Trong đồ thị có hướng:

Tổng bán bậc ra = tổng bán bậc vào = số cung

Chứng minh:

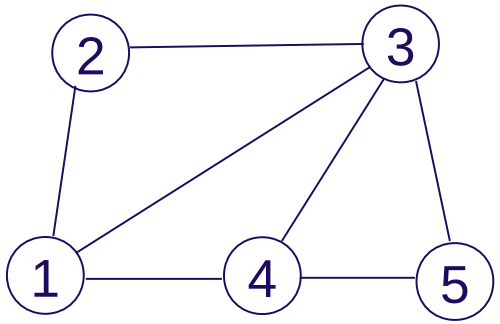
Gọi m là số cung thì cần cm: $\sum_{v \in V} d^-(v) = \sum_{v \in V} d^+(v) = m$

Hiển nhiên vì mỗi cung (u,v) ra ở đỉnh u và vào ở đỉnh v nên được tính một lần trong bậc ra của u và một lần trong bậc vào của v nên suy ra đpcm.

Một số khái niệm

Đường đi, chu trình, liên thông:

- **Đường đi:** Đường đi có độ dài n từ đỉnh v_0 đến đỉnh v_n là dãy $v_0, v_1, \ldots, v_{n-1}, v_n$; với $(v_i, v_{i+1}) \in E$, i=0,…,n-1. Đường đi có thể biểu diễn bằng một dãy n cạnh (cung): (v_0,v_1), (v_1,v_2),…, (v_{n-1}, v_n). Đỉnh v_0 gọi là đỉnh đầu, đỉnh v_n gọi là đỉnh cuối của đường đi.



Dãy các đỉnh sau là đường đi:

1,3,4,5,3,2

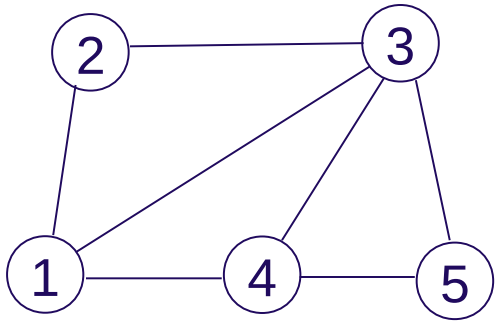
5,3,4,1,2

2,3,1,4,5,3

…

Một số khái niệm

- **Chu trình:** là đường đi có đỉnh đầu trùng với đỉnh cuối. Đường đi (hay chu trình) gọi là đơn nếu không có cạnh (cung) bị lặp lại; gọi là sơ cấp nếu không có đỉnh nào bị lặp lại

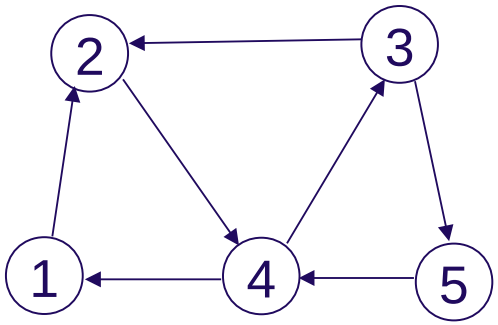


Dãy các đỉnh trên đồ thị vô hướng sau đây là các chu trình:

1,2,3,5,4,3,1 (chu trình đơn)

2,3,4,1,2 (chu trình sơ cấp)

1,3,4,1,3,2,1 (không đơn)



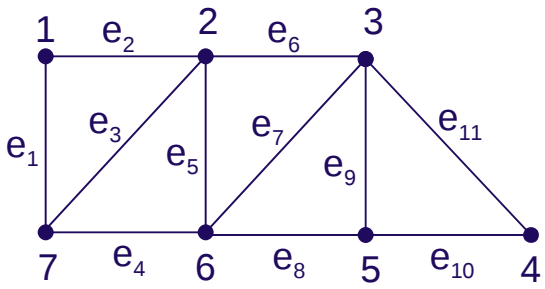
Dãy các đỉnh trên đồ thị có hướng sau đây là các chu trình:

1,2,4,3,2,4,1 (không đơn)

1,2,4,3,5,4,1 (chu trình đơn)

Một số khái niệm

- **Đối chu trình:** Cho G=(V,E) và A⊂V, đối chu trình xác định bởi A được định nghĩa là:
  w(A)={e ∈ E | e có một đỉnh ở trong A}
- **Đối chu trình sơ cấp:** Cho G liên thông đối chu trình w=w(A) được gọi là sơ cấp (hay tập cắt) nếu:
  - G – w không liên thông và
  - ∀ w'⊂ w thì G - w' liên thông



A={2,7} thì w(A)={e_1,e_2, e_4, e_5, e_6} không sơ cấp

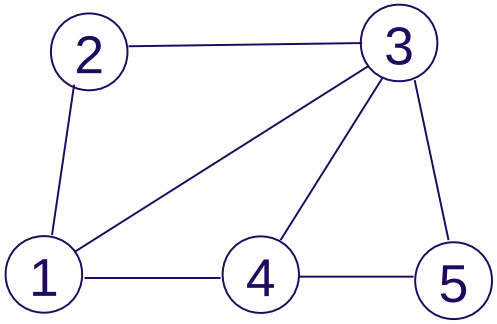
A={1,7} thì w(A)={e_2, e_3, e_4} sơ cấp

A={3,5,6}, w(A)=?

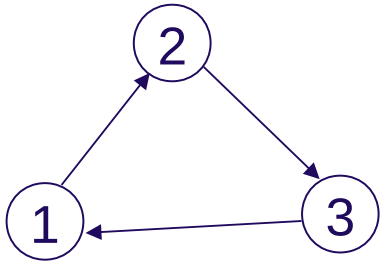
A={2,5}, w(A)=?

Một số khái niệm

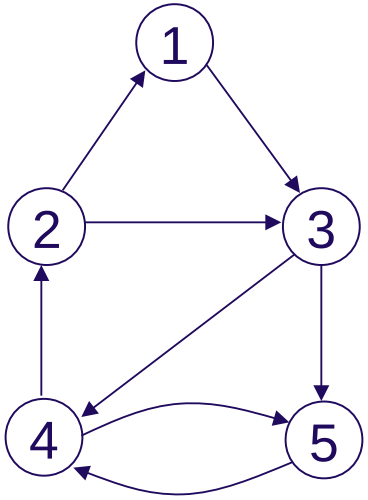
- **Đồ thị liên thông:** Một đồ thị được gọi là liên thông nếu hai đỉnh bất kỳ luôn có đường đi.



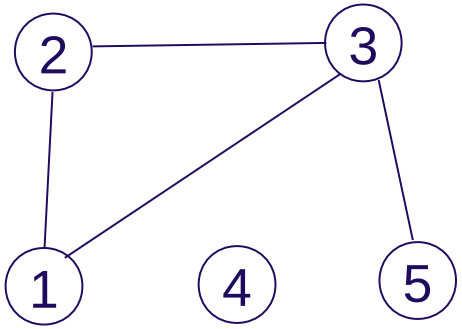
Liên thông



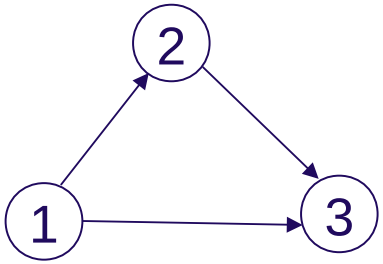
Liên thông



Liên thông



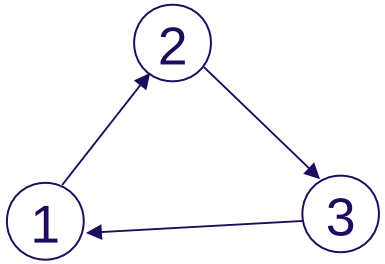
Không liên thông



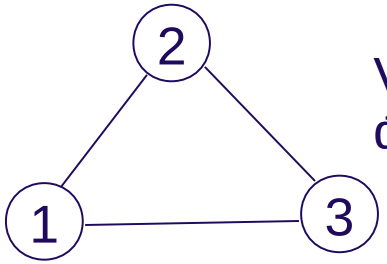
Không liên thông

Một số khái niệm

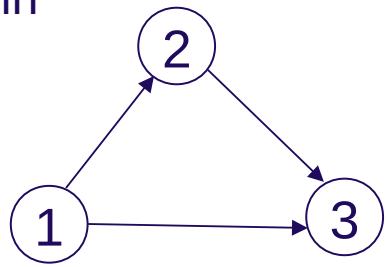
- **Đồ thị liên thông mạnh:** là đồ thị có hướng liên thông
- **Đồ thị liên thông yếu:** là đồ thị có hướng không liên thông, nhưng đồ thị vô hướng tương ứng liên thông
- **Đồ thị vô hướng liên thông gọi là định hướng được:** nếu có thể định hướng các cạnh để thu được đồ thị có hướng liên thông.



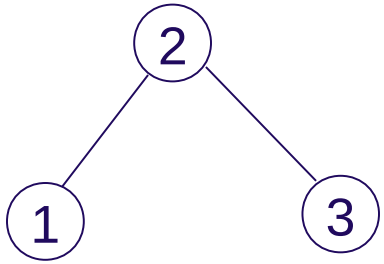
Liên thông mạnh



Vô hướng liên thông định hướng được



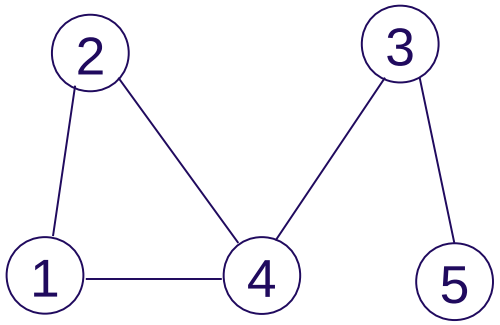
Liên thông yếu



Vô hướng liên thông không định hướng được

Một số khái niệm

- **Đỉnh rẽ nhánh:** Đỉnh v gọi là đỉnh rẽ nhánh nếu việc loại bỏ v cùng với các cạnh liên thuộc với nó làm tăng số thành phần liên thông.
- **Cạnh cầu:** Cạnh e gọi là cầu nếu việc loại bỏ e làm tăng số thành phần liên thông.

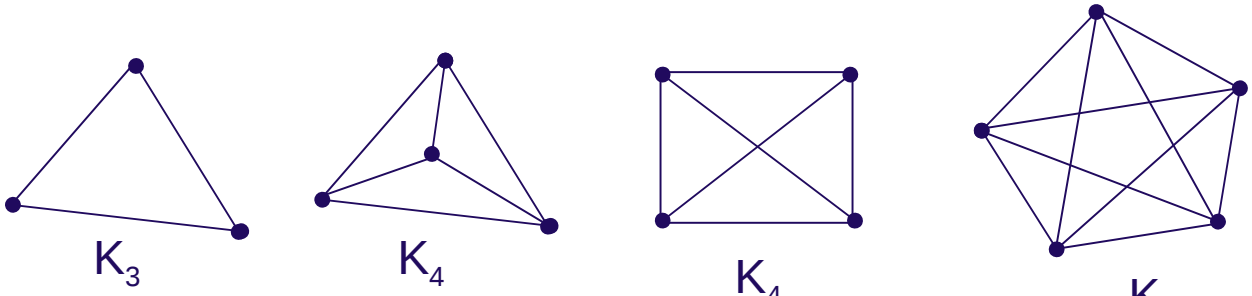


Đỉnh 3,4 gọi là đỉnh rẽ nhánh

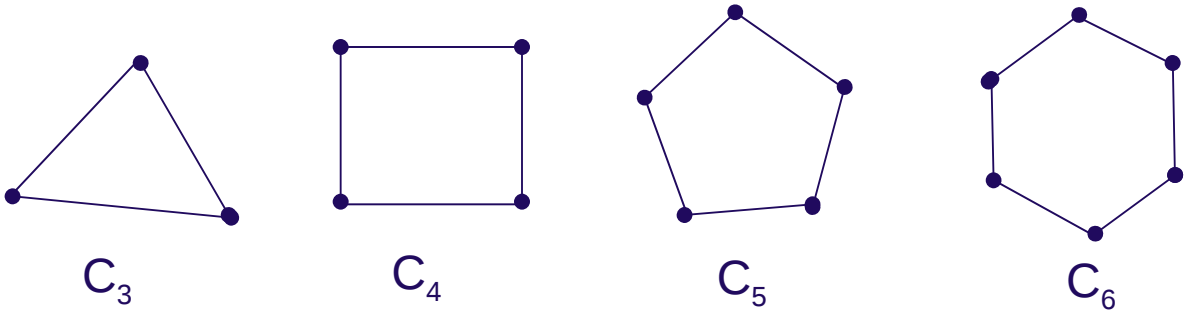
Cạnh (3,4), (3,5) gọi là cạnh cầu

Một số đồ thị đặc biệt

- **Đồ thị đủ cấp n:** Là đơn đồ thị vô hướng có n đỉnh, ký hiệu bởi $\mathbf{K_n}$, mà giữa hai đỉnh bất kỳ của nó luôn có cạnh nối. K_n có số cạnh là: n(n-1)/2

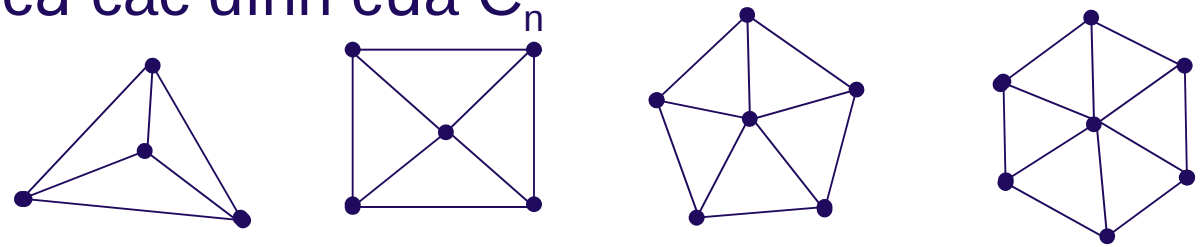


- **Đồ thị vòng:** Đồ thị vòng C_n,n≥3 gồm n đỉnh $v_1,v_2,\ldots,v_n$ và các cạnh (v_1,v_2), (v_2,v_3) . . . (v_{n-1},v_n), (v_n,v_1).



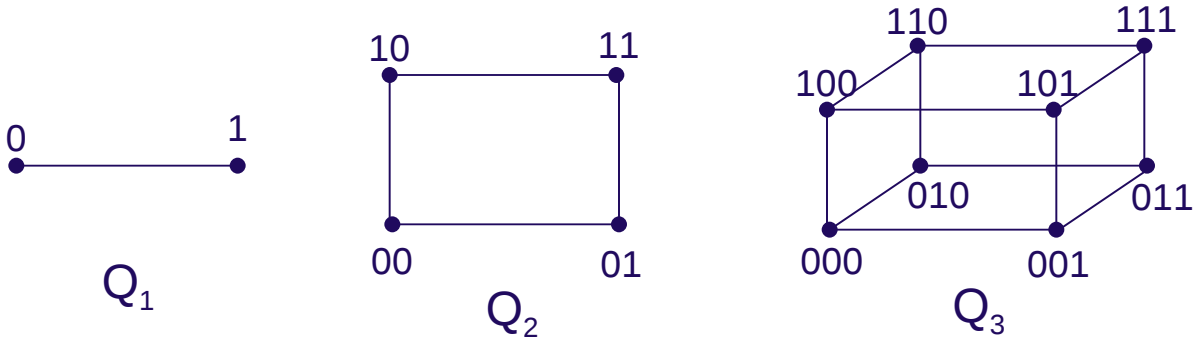
Một số đồ thị đặc biệt

- **Đồ thị bánh xe:** Đồ thị bánh xe $\mathbf{W_n}$ thu được từ đồ thị vòng C_n bằng cách bổ sung vào một đỉnh mới nối với tất cả các đỉnh của C_n



W_3 W_4 W_5 W_6

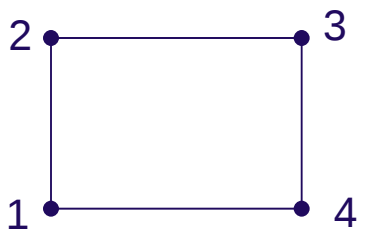
- **Đồ thị lập phương:** Đồ thị lập phương $\mathbf{Q_n}$ là đồ thị với các đỉnh biểu diễn 2^n xâu nhị phân độ dài n.

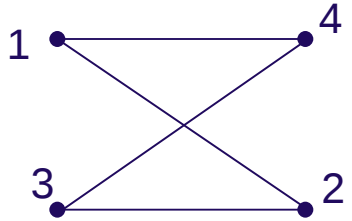


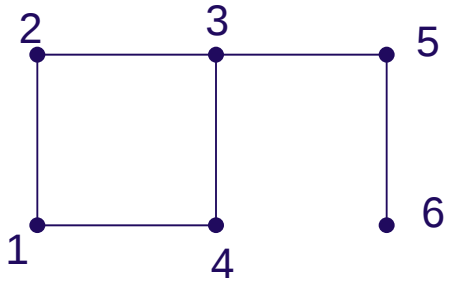
Q_1 Q_2 Q_3

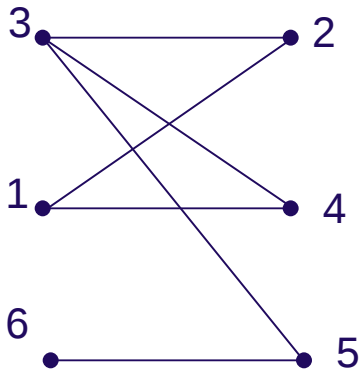
Một số đồ thị đặc biệt

- **Đồ thị lưỡng phân(hai phía):** Đơn đồ thị G=(V,E) được gọi là lưỡng phân(hai phía) nếu như tập đỉnh V của nó có thể phân hoạch thành hai tập X và Y sao cho mỗi cạnh của đồ thị chỉ nối một đỉnh trong X với một đỉnh trong Y. Ký hiệu G=(X∪Y, E)



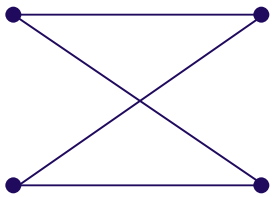




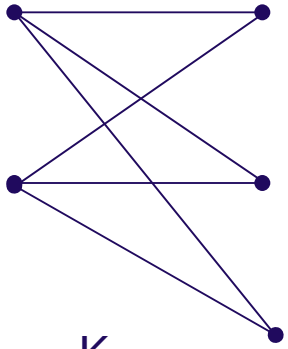


Một số đồ thị đặc biệt

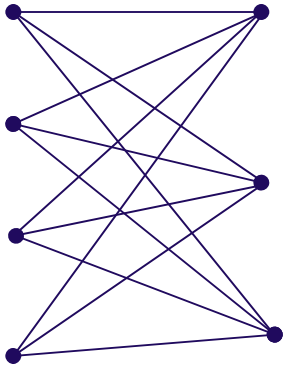
- **Đồ thị lưỡng phân đủ:** Đồ thị lưỡng phân G=(X,Y, E) với |X|= m, |Y| = n được gọi là đồ thị lưỡng phân đủ, ký hiệu là $K_{m,n}$ nếu mỗi đỉnh trong tập X được nối với tất cả các đỉnh trong tập Y.



$K_{2,2}$



$K_{2,3}$

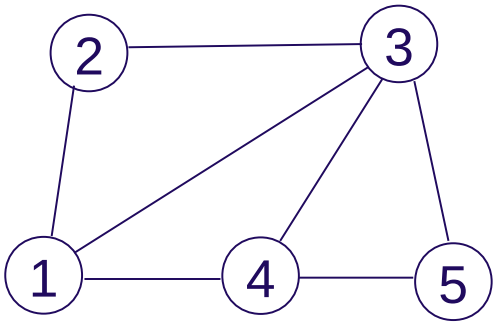


$K_{4,3}$

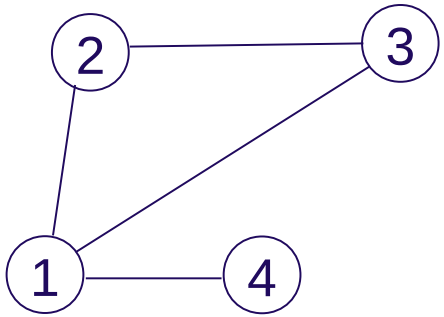
Định lý: G là đồ thị lưỡng phân nếu G không có chu trình độ dài lẻ

Một số đồ thị đặc biệt

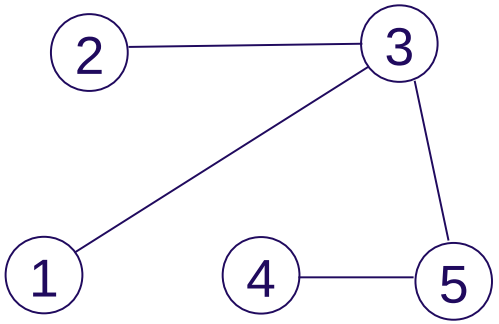
- **Đồ thị con:** Cho hai đồ thị G=(V,E) và G'(V', E'). G' là đồ thị con của G nếu V'$\subseteq$ V và E'$\subseteq$ E. Nếu V'=V thì G' gọi là đồ thị bộ phận hay đồ thị khung của G.



G



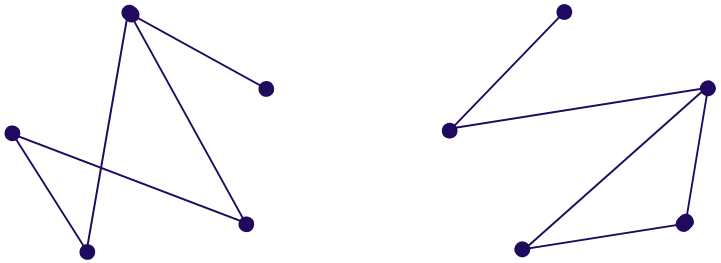
Đồ thị con của G



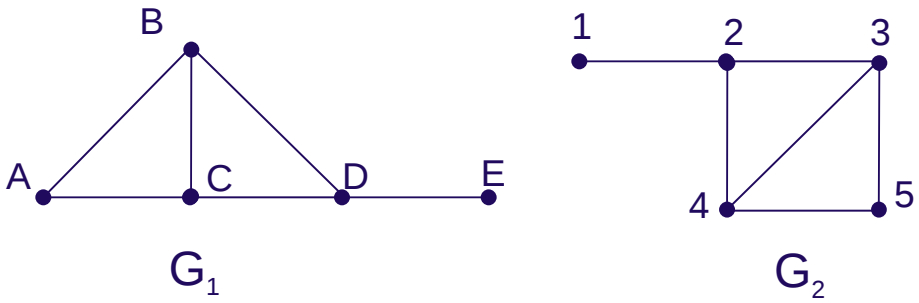
Đồ thị bộ phận của G

Một số đồ thị đặc biệt

- **Đồ thị bù:** Cho $K_n=(V,E)$ và $G=(V,E_1)$ là đồ thị khung của K_n. $\overline{G}=(V,E_2)$ gọi là đồ thị bù của G nếu $E_2=E-E_1$



- **Đồ thị đẳng cấu:** Hai đồ thị đơn vô hướng $G_1=(V_1,E_1)$ và $G_2(V_2,E_2)$ được gọi là đẳng cấu nếu có một song ánh $f: V_1 \rightarrow V_2$ sao cho với $(u,v) \in E_1 \Leftrightarrow (f(u),f(v)) \in E_2$

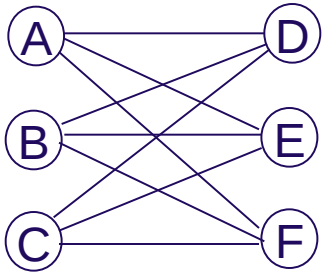


Song ánh f:

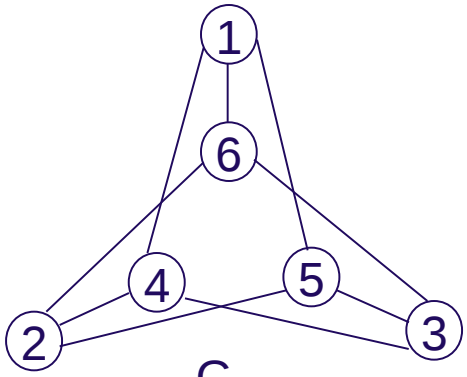
f(A)=5; f(B)=4; f(C)=3;
f(D)=2; f(E)=1

Một số đồ thị đặc biệt

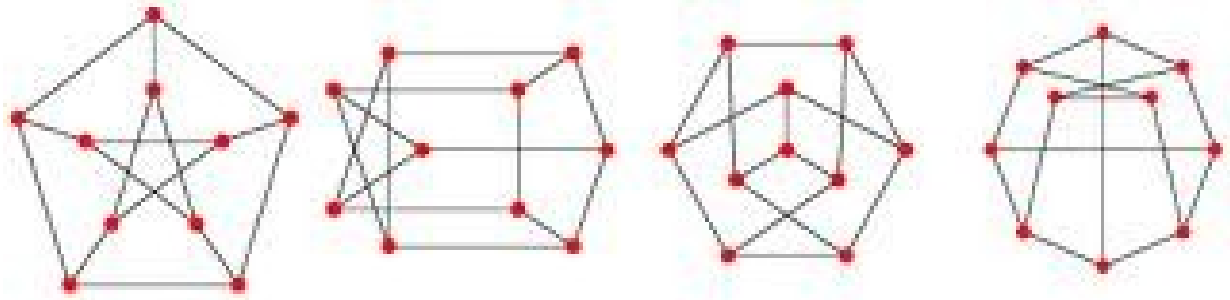
Các cặp đồ thị sau có đẳng cấu không?. Nếu có thì hãy xây dựng một song ánh f?



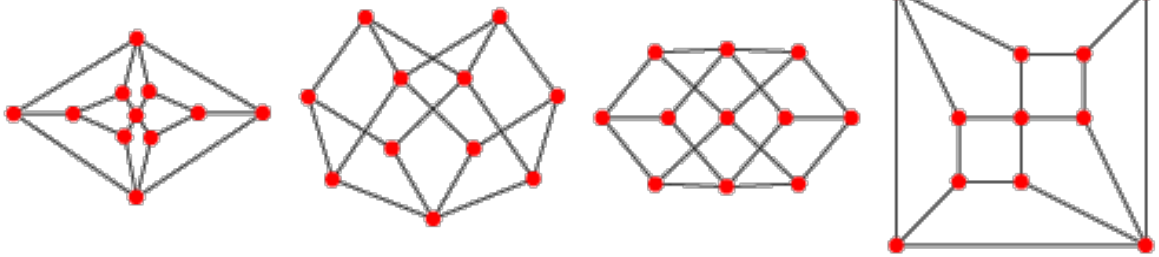
G_1



G_2



Đồ thị Petersen



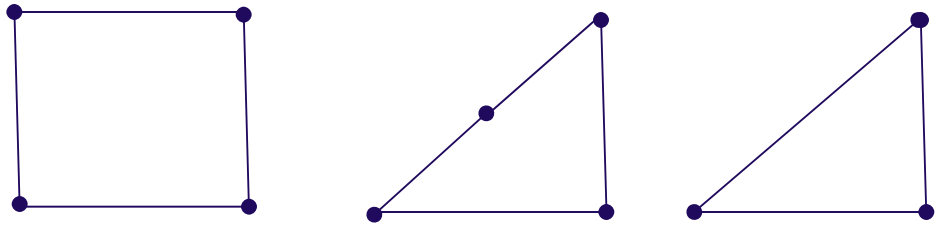
Đồ thị Herschel

Một số đồ thị đặc biệt

- **Đồ thị đồng cấu:**

  Phép chia cạnh (u,v) của đồ thị là việc loại bỏ cạnh này khỏi đồ thị và thêm vào đồ thị một đỉnh mới w cùng với hai cạnh (u,w), (w, u) .

  Hai đồ thị G=(V,E) và H=(W,F) được gọi là đồng cấu nếu chúng có thể thu được từ cùng một đồ thị nào đó nhờ phép chia cạnh.

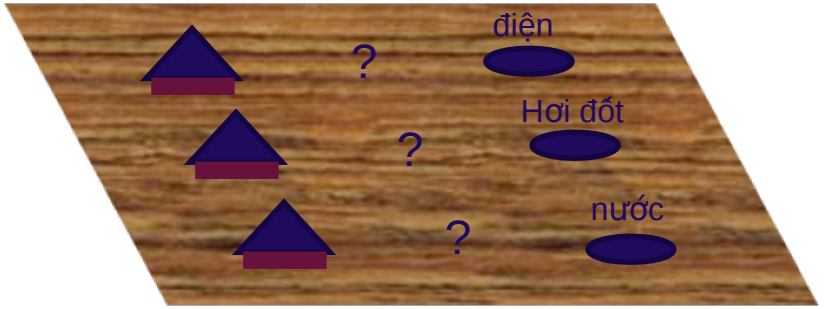


Một số đồ thị đặc biệt

- **Đồ thị phẳng:**

Bài toán 3 căn hộ:

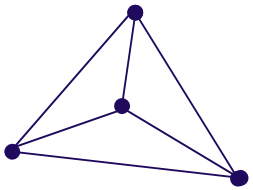
Cần xây dựng một hệ thống cung cấp điện, hơi đốt và nước cho ba căn hộ sao cho mỗi căn hộ đều được nối với các nguồn cung cấp trên và đường dẫn của chúng không cắt nhau



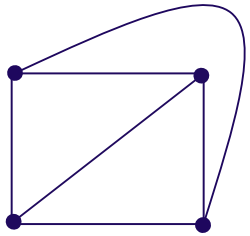
Để giải quyết bài toán trên, ta sẽ sử dụng khái niệm đồ thị phẳng.

Một số đồ thị đặc biệt

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



K_4

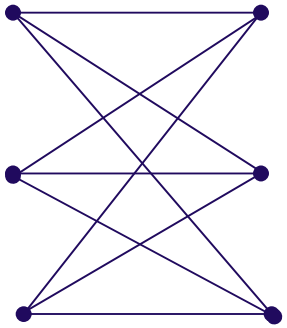


K_4

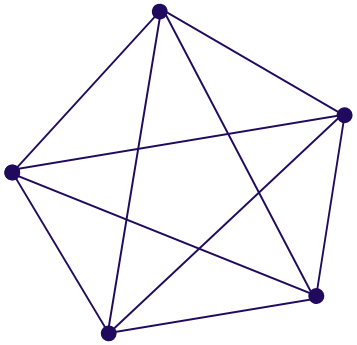
Định lý Kuratowski:
(dùng kiểm tra một đồ thị có là phẳng hay không)

Đồ thị G là phẳng $\Leftrightarrow$ G không chứa đồ thị con đồng cấu với $K_{3,3}$ hoặc K_5

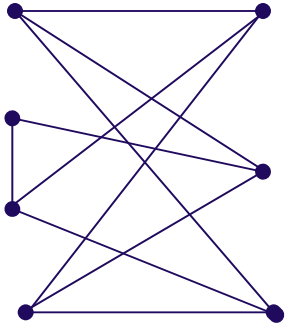
Một số đồ thị đặc biệt

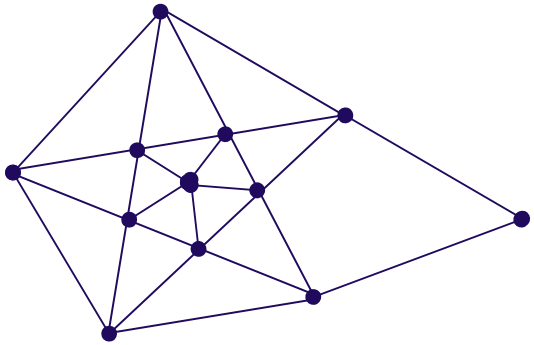


$K_{3,3}$



K_5

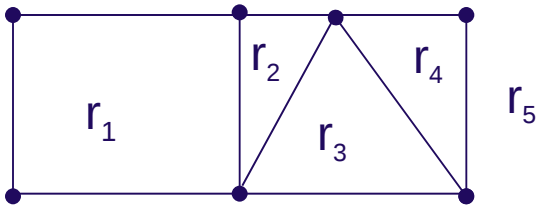




Một số đồ thị đặc biệt

<u>Định lý</u>: (Công thức Euler)

G là đồ thị phẳng liên thông, G có n đỉnh, m cạnh, r là số miền của mặt phẳng bị chia bởi biểu diễn phẳng của G. Ta có: $r = m - n + 2$

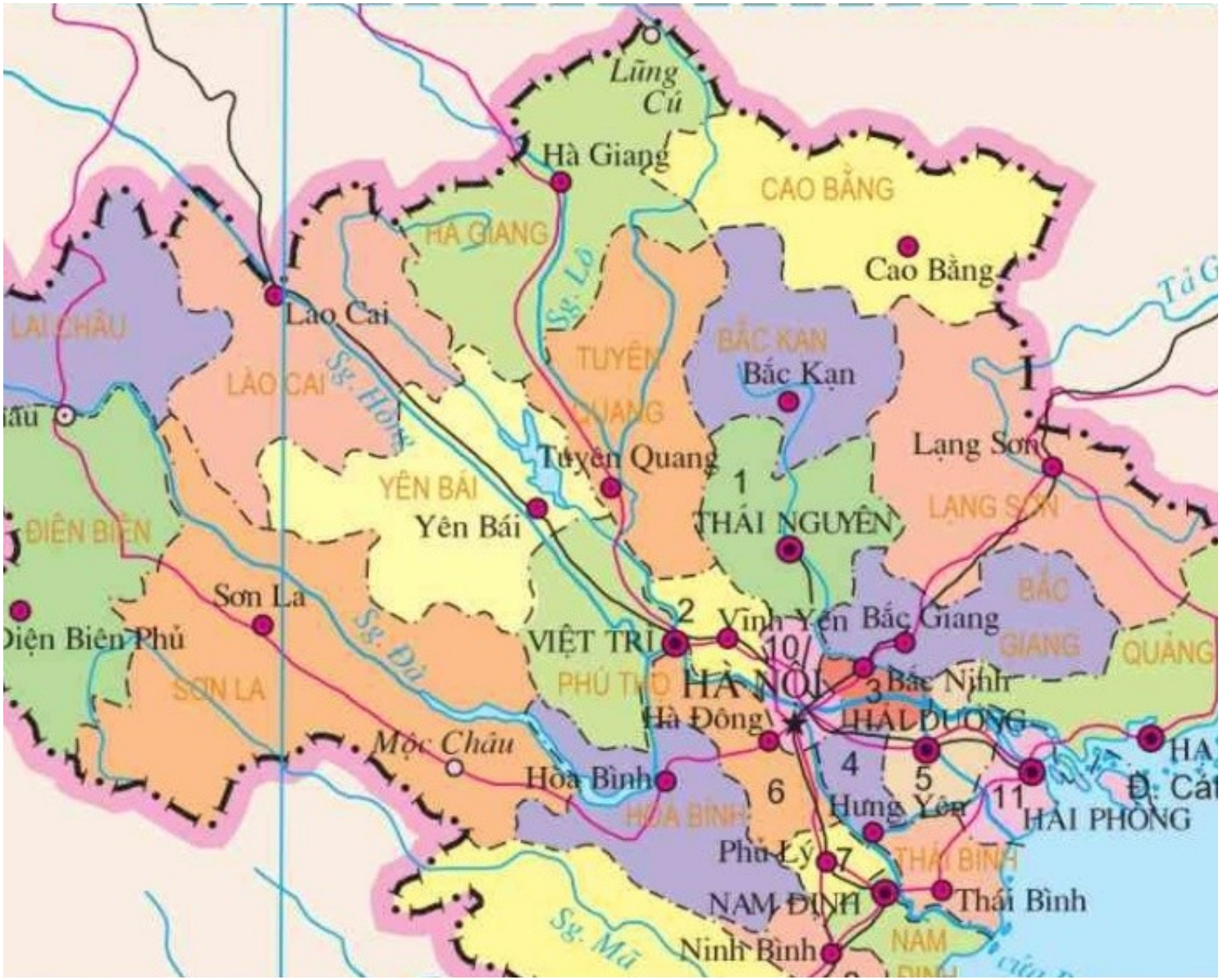


Số đỉnh: 7

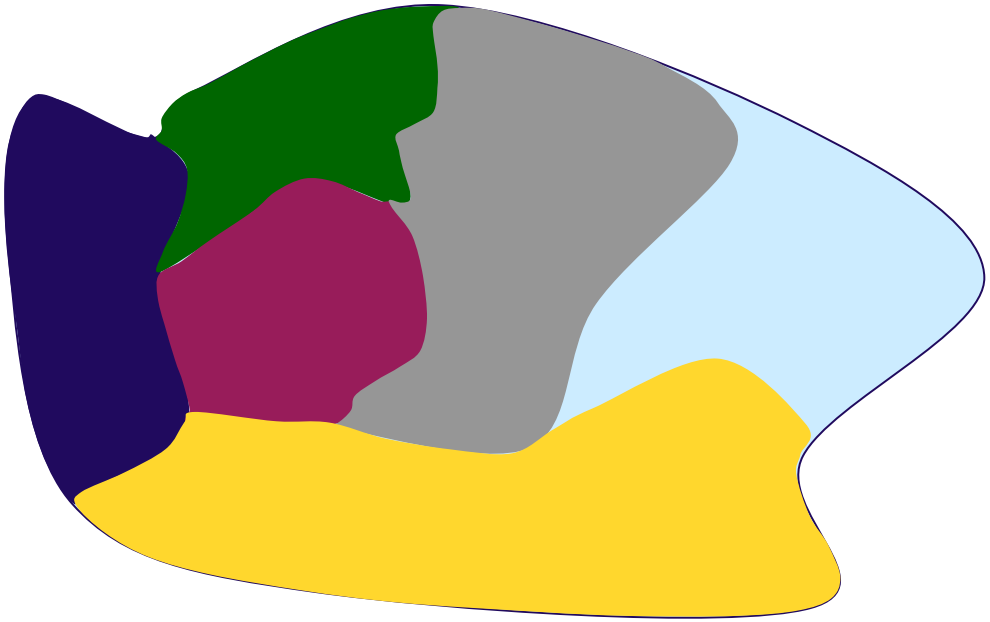
Số cạnh: 10

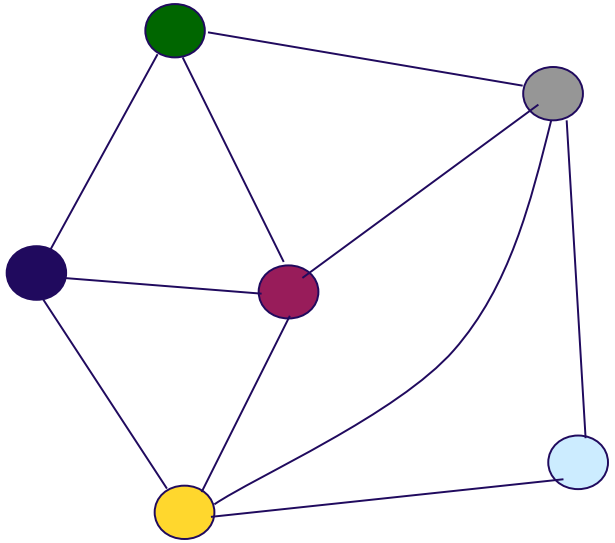
Số miền: 10 – 7 + 2 = 5

Săc số của đồ thị



Săc số của đồ thị





Săc số của đồ thị

Định nghĩa:

Tô màu một đồ thị vô hướng là một sự gán màu cho các đỉnh sao cho hai đỉnh kề nhau phải khác màu nhau.

Số màu (sắc số) của một đồ thị là số màu tối thiểu cần thiết để tô màu đồ thị này.

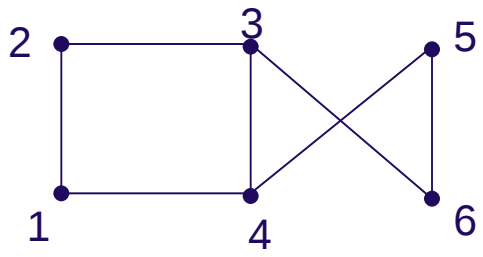
Thuật toán tô màu Welch-Powell

B1: Săp xếp danh sách các đỉnh theo thứ tự bậc giảm dần

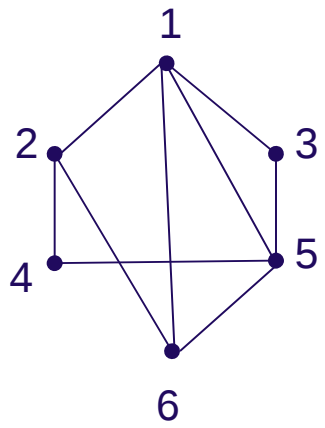
B2: Chọn đỉnh v chưa tô trên danh sách theo thứ tự từ trái sang phải, chọn một màu để tô đỉnh v và các đỉnh không kề với v

B3: Lặp lại B2 đến khi tất cả các đỉnh đều được tô.

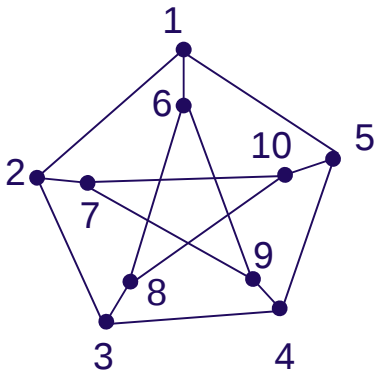
Săc số của đồ thị

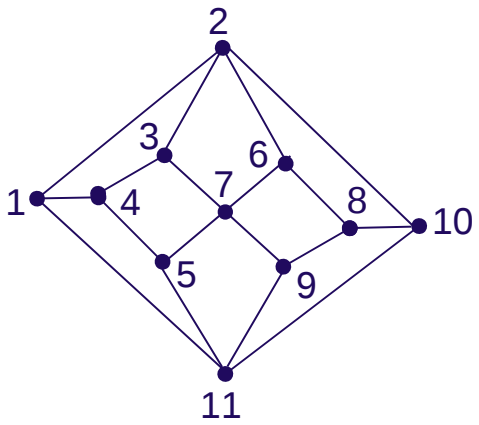


Đỉnh	4	3	5	6	1	2
Bậc	3	3	2	2	2	2
Màu	1	2	2	1	2	1



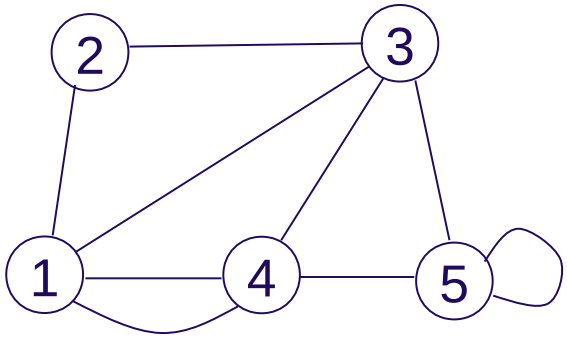
Đỉnh	1	5	2	6	3	4
Bậc	4	4	3	3	2	2
Màu	1	2	2	3	3	1





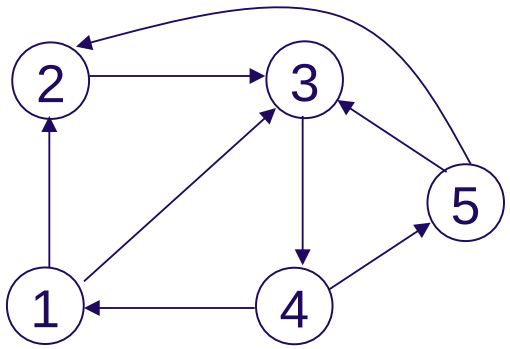
CHƯƠNG 2: BIỂU DIỄN ĐỒ THỊ

- **<u>Ma trận kề</u>:**
  Cho G=(V,E), V={1,2,3,…,n}, ma trận kề A=($A_{i,j}$) của G là ma trận vuông cấp n xác định bởi:
  $A_{i,j}$=số cạnh (cung) từ i đến j



$A_{i,j}$=

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



$A_{i,j}$=

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

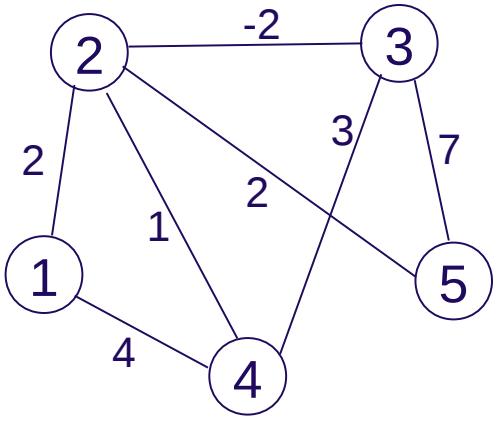
Ma trận trọng số

- **<u>Ma trận trọng số</u>**:

  Cho G=(V,E) là một đồ thị có trọng số, nghĩa là mỗi cạnh (i,j) $\in$ E đều có một giá trị c(i,j) gọi là trọng số của cạnh.

  Ma trận trọng số $A_{i,j} = \begin{cases} \text{trọng số cạnh|cung (i,j) nếu (i,j)} \in E \\ \theta \text{ nếu (i,j)} \notin E \end{cases}$

  Trong đó là một trong các giá trị: 0, ∞, +∞, -∞



	1	2	3	4	5
1	0	2	∞	4	∞
2	2	0	-2	1	2
3	∞	-2	0	3	7
4	4	1	3	0	∞
5	∞	2	7	∞	0

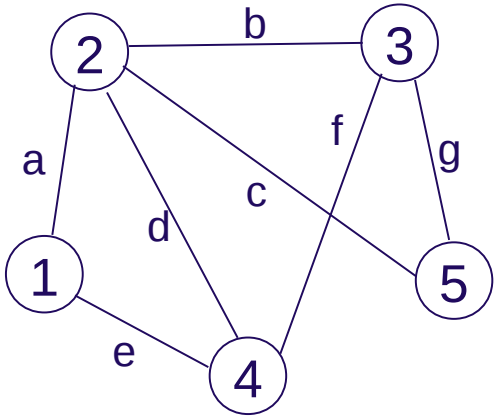
Ma trận liên kết

- **<u>Ma trận liên kết (ma trận liên thuộc đỉnh-cạnh):</u>** Cho G=(V,E) với V={1,2,3,…,n}; E=(e_1, e_2,..., e_m). Ma trận liên kết của G là ma trận A=($A_{i,j}$) có n dòng, m cột được định nghĩa như sau:

Nếu G vô hướng thì $A_{i,j}$= { 1 nếu đỉnh i kề với cạnh e_j ; 0 nếu ngược lại

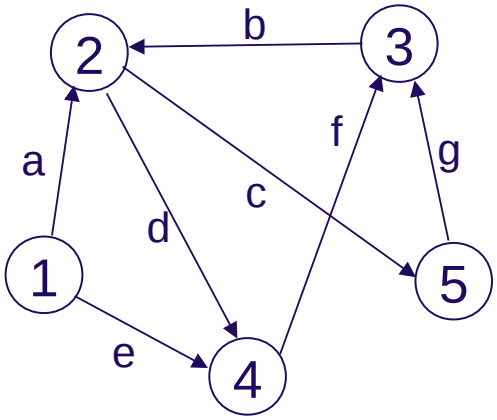
Nếu G vô hướng thì $A_{i,j}$= { 1 nếu e_j rời khỏi đỉnh i ; -1 nếu e_j đi đến đỉnh i ; 0 nếu e_j không kề với đỉnh i

Ma trận liên kết



$$A_{i,j} = \begin{cases} 1 \text{ nếu đỉnh i kề với cạnh } e_j \\ 0 \text{ nếu ngược lại} \end{cases}$$

	a	b	c	d	e	f	g
1	1	0	0	0	1	0	0
2	1	1	1	1	0	0	0
3	0	1	0	0	0	1	1
4	0	0	0	1	1	1	0
5	0	0	1	0	0	0	1



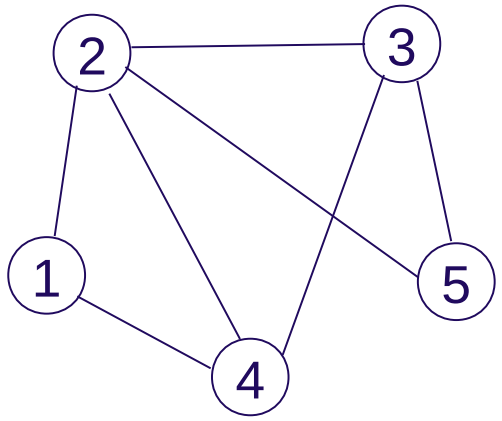
$$A_{i,j} = \begin{cases} 1 \text{ nếu } e_j \text{ rời khỏi đỉnh i} \\ -1 \text{ nếu } e_j \text{ đi đến đỉnh i} \\ 0 \text{ nếu } e_j \text{ không kề với đỉnh i} \end{cases}$$

	a	b	c	d	e	f	g
1	1	0	0	0	1	0	0
2	-1	-1	1	1	0	0	0
3	0	1	0	0	0	-1	-1
4	0	0	0	-1	-1	1	0
5	0	0	-1	0	0	0	1

Danh sách cạnh (cung)

- **<u>Danh sách cạnh(cung)</u>**:

  Trong trường hợp số cạnh ít hơn nhiều so với số cạnh thì người ta thường dùng danh sách cạnh(cung) để lưu trữ đồ thị. Mỗi cạnh được biểu diễn bởi đỉnh đầu và đỉnh cuối.

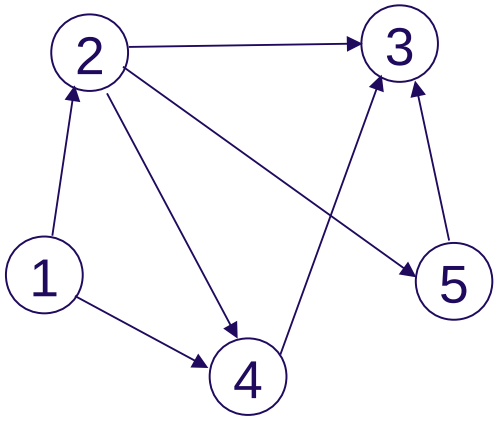


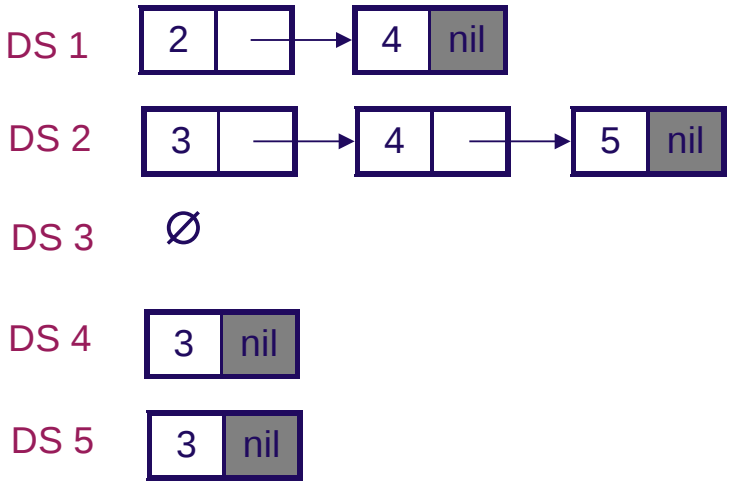
Đầu	Cuối
1	2
1	4
2	3
2	4
2	5
3	4
3	5

Danh sách kề

- **<u>Danh sách kề</u>**:

  Danh sách kề là danh sách lưu các đỉnh kề của một đỉnh nào đó. Nếu đồ thị có n đỉnh thì sẽ lưu trữ với n danh sách kề



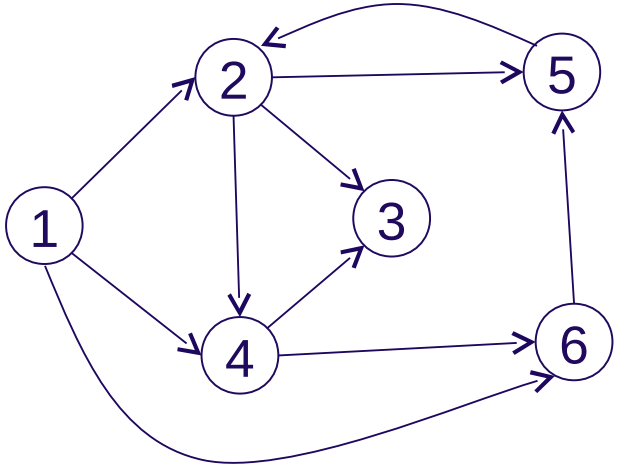


Nhận xét

Lưu trữ	Ưu điểm	Khuyết điểm
Ma trận kề	-Truy xuất nhanh các đỉnh kề	- Luôn sử dụng 2^n đơn vị bộ nhớ cho dù số cạnh rất ít
Ma trận liên kết	- Tiết kiệm bộ nhớ đối với đồ thị có ít cạnh	- Tìm đỉnh kề khó khăn
Danh sách cạnh	-Tiết kiệm bộ nhớ đối với đồ thị có ít cạnh	-Thực hiện nhiều phép so sánh khi tìm đỉnh kề -Trong trường hợp đồ thị có trọng số phải thêm đơn vị bộ nhớ
Danh sách kề	- Phân nhóm đỉnh kề rõ ràng thành các danh sách	-Thực hiện thao tác chậm do phải truy xuất tuần tự -Tìm một đỉnh là kề của những đỉnh nào phải duyệt hết các danh sách

Biểu diễn đồ thị trên máy tính

- Viết chương trình đọc một đồ thị vào máy tính bằng:
  - Ma trận kề
  - Ma trận liên kết
  - Danh sách cạnh
  - Danh sách kề



- Viết chương trình chuyển đổi qua lại giữa các hình thức lưu trữ trên
- Viết chương trình tìm các đỉnh có đỉnh kề là k với k nhập vào từ bàn phím và đồ thị được lưu trữ bằng danh sách kề

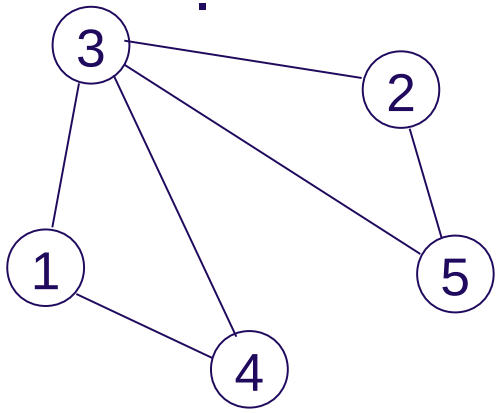
CHƯƠNG 3: CÁC THUẬT TOÁN DUYỆT VÀ TÌM KIẾM TRÊN ĐỒ THỊ

1) Tìm theo chiều sâu(Depth First Search - DFS)
2) Tìm theo chiều rộng(Breadth First Search - BFS)

Thuật toán DFS

Ý tưởng:

- Từ đỉnh v_1 nào đó chưa thăm, thăm v_1 rồi tìm đỉnh v_2 (chưa thăm) kề với v_1, thăm v_2 …
- Nếu tại một đỉnh v_i nào đó không còn đỉnh kề chưa thăm thì quay trở lại tìm đỉnh kề chưa thăm khác của v_{i-1} và thăm đỉnh này.
- Thuật toán lặp lại việc thăm cho đến khi tất cả các đỉnh đều được thăm.



Nếu bắt đầu từ đỉnh 1 thì thứ tự duyệt có thể là: 1, 3, 2, 5, 4

Thuật toán DFS

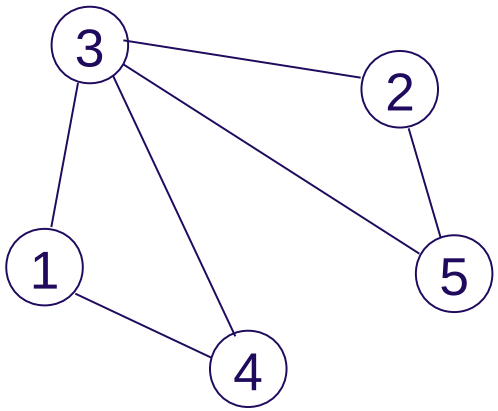
Thuật toán:

```
void DFS1(v)//duyệt một thành phần liên thông chứa v
{
    Tham_Dinh(v);
    tham[v]=1; //ghi nhận là đã thăm v để về sau không thăm nữa.
    For  (u ∈ Ke(v)) // xét tất cả các đỉnh u kề với v
            If (!tham[u])  DFS1(u); //neu u chua thăm, thăm u
}
void DFS() //duyệt tất cả các thành phần liên thông
{
    for (v∈ V)  tham[v]=0; //ban đầu tất cả các đỉnh đều chưa thăm.
    for (v∈ V) //xét tất cả các dinh
            if (!tham[v])  DFS1(v); // neu đỉnh v chua tham thi tham v
}
```

Thuật toán BFS

Ý tưởng:

- Từ đỉnh v nào đó chưa thăm, cất v vào hàng đợi.
- Lấy từ hàng đợi một đỉnh v, thăm v, rồi cất các đỉnh u chưa thăm kề với v vào hàng đợi…
- Lặp lại cho tới khi hàng đợi rỗng.



1	2	3	4	5
1				
	3	4		
		4	2	5
			2	5
				5

Thứ tự duyệt theo BFS là: 1, 3, 4, 2, 5

Thuật toán BFS

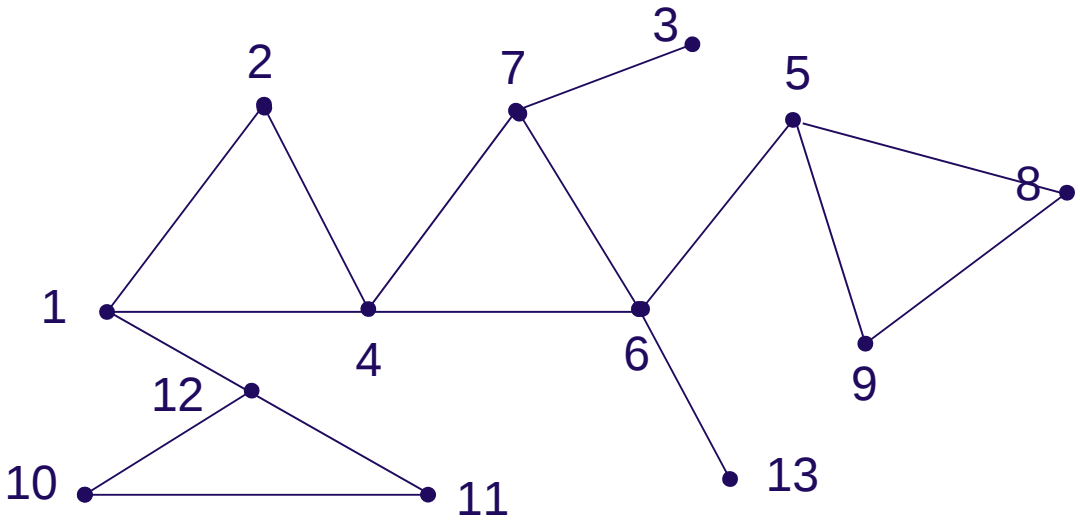
Thuật toán:

```
void BFS1(v) //duyệt một thành phần liên thông
{
    queue=∅; //khoi tạo hàng đợi rỗng
    push(queue,v); //cất v vào hàng đợi
    tham[v]=1; //ghi nhận là đã thăm v để về sau không thăm nữa.
    while (queue <>∅) // trong khi hàng đợi còn khác rỗng
      {
                v=pop(queue); //lay mot dinh v từ hàng đợi
                for  (u ∈ Ke(v)) // xét các đỉnh u kề với v
                     if(!tham[u]) //nếu u chưa thăm
                     {
                          push(queue,u); //cất u vào  hàng đợi
                          tham[u]=1; //ghi nhan u tham roi
                     }
       }
}
```

Thuật toán BFS

```
void BFS() //duyệt tất cả các thành phần liên thông
{
    for  (v ∈ V)  tham[v]=0; //ban đầu tất cả các đỉnh đều chưa thăm.
    for (v ∈ V) //xét tất cả các dinh
        if  (!tham[v])  BFS1(v); // neu đỉnh v chua tham thi tham v
}
```

Dùng thuật toán DFS và BFS duyệt đồ thị sau:



Tìm số thành phần liên thông

Hãy cho biết đồ thị có bao nhiêu thành phần liên thông và mỗi thành phần liên thông gồm những đỉnh nào?

Ý tưởng:

Do số thành phần liên thông bằng số lần DFS() gọi DFS1() hoặc BFS() gọi BFS1(), nên ta dùng biến stplt để đếm số thành phần liên thông, mỗi lần DFS() gọi DFS1() hoặc BFS() gọi BFS1() ta tăng biến stplt lên 1.

Khi thăm v thay vì gán thăm[v] = true (=1) ta gán thăm[v] = stplt (số hiệu thành phần liên thông chứa v).

Tìm số thành phần liên thông

Thuật toán:

```
void DFS1(v) //duyệt một thành phần liên thông
{   Tham_Dinh(v);
    tham[v]=stplt; //ghi nhận là đã thăm v để về sau không thăm nữa.
    For  (u ∈ Ke(v)) // xét tất cả các đỉnh u kề với v
    If(!tham[u])
            DFS1(u); //neu u chua thăm, thăm u
}
void DFS() //duyệt tất cả các thành phần liên thông
{   for  (v ∈ V)  tham[v]=0; //ban đầu tất cả các đỉnh đều chưa thăm.
    stplt=0;
    for (v ∈ V) //xét tất cả các dinh
            if  (!tham[v]) {
                    stplt++;
                    DFS1(v); // neu đỉnh v chua tham thi tham v
            }
}
```

Tìm số thành phần liên thông

```
void BFS1(v) //duyệt một thành phần liên thông
{
    queue=∅; //khoi tạo hàng đợi rỗng
    push(queue,v); //cất v vào hàng đợi
    tham[v]=stplt; //ghi nhận là đã thăm v để về sau không thăm nữa.
    while (queue) // trong khi hàng đợi còn khác rỗng
    {       v=pop(queue); //lay v từ hàng đợi
            Tham_Dinh(v);
            for  (u ∈ Ke(v)) // xét các đỉnh u kề với v
                        if (!tham[u]) //nếu u chưa thăm
                        {    push(queue,u); //cất u vào  hàng đợi
                             tham[u]=stplt; //ghi nhan u tham roi
                        }
    }
}
```

Bài tập

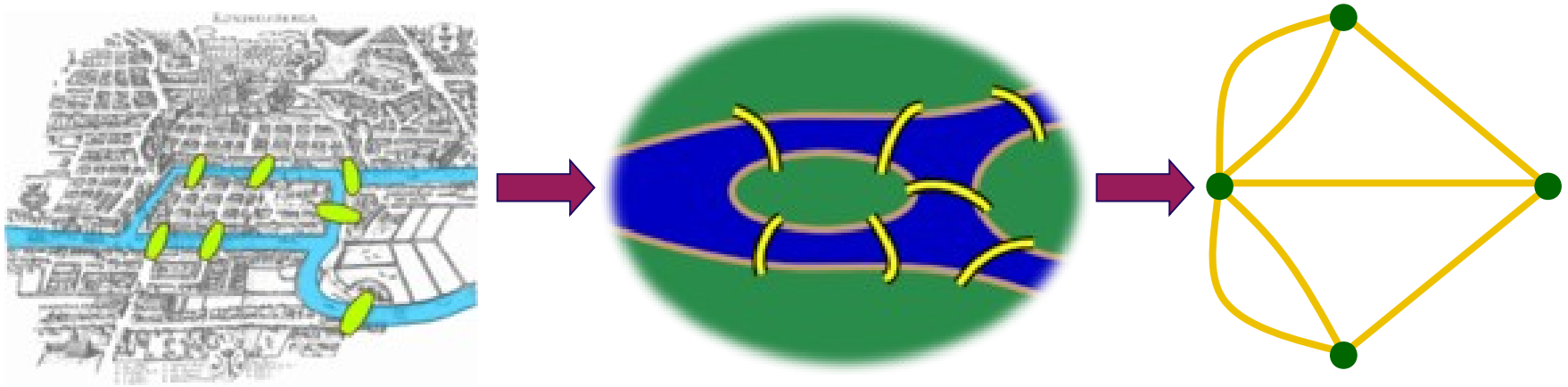
- Viết chương trình duyệt đồ thị với thuật toán DFS dùng cấu trúc stack
- Viết chương trình duyệt đồ thị bằng thuật toán DFS và BFS với đồ thi được lưu trữ bằng danh sách kề.

Leonhard Euler (15/04/1707 -18/09/1783)
Người Thuỵ Sĩ, là nhà toán học, vật lý học

William Rowan Hamilton (04/08/1805 – 02/09/1865), người Ireland, là một nhà toán học, vật lý và thiên văn học

Đồ thị Euler

<u>Bài toán bảy cây cầu</u>:
Có thể bắt đầu từ một điểm và đi qua mỗi cây cầu đúng một lần rồi quay lại điểm xuất phát hay không ?

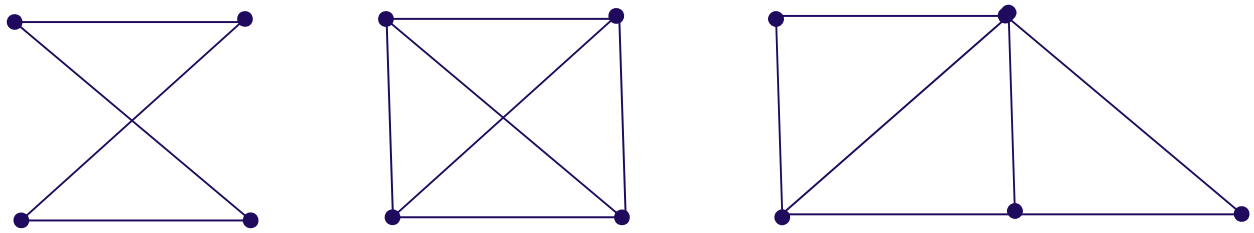


Bản đồ Königsberg thời Euler, mô tả vị trí thực của bay cây cầu và sông Pregel.

Năm 1736 **Leonhard Euler** đã chứng minh rằng điều đó là không thể được.

Đồ thị Euler

- **Đường đi Euler:**
  Đường đi qua tất cả các cạnh của đồ thị, mỗi cạnh đúng một lần gọi là đường đi Euler.
- **Chu trình Euler:**
  Chu trình đi qua tất cả các cạnh của đồ thị, mỗi cạnh đúng một lần gọi là chu trình Euler.
- **Đồ thị Euler, nửa Euler:**
  Đồ thị có chu trình Euler gọi là đồ thị Euler, đồ thị có đường đi Euler gọi là đồ thị nửa Euler.



G_1 G_2 G_3

Đồ thị Euler

Định lý Euler

a/ G là đồ thị vô hướng liên thông.

G là đồ thị Euler $\Leftrightarrow$ mọi đỉnh của G đều có bậc chẵn.

b/ G là đồ thị có hướng liên thông.

G là đồ thị Euler $\Leftrightarrow$ bậc vào và bậc ra của mỗi đỉnh là bằng nhau

Định lý nửa Euler

Cho đồ thị vô hướng liên thông G.

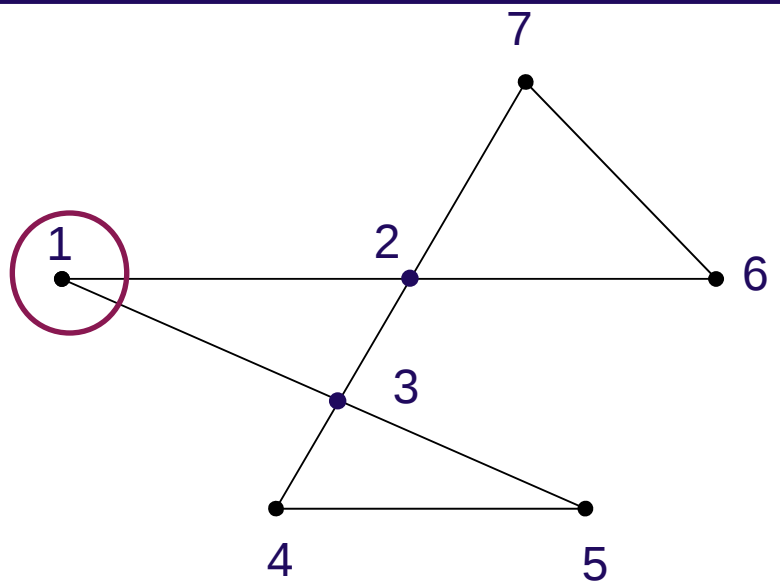
G là nửa Euler $\Leftrightarrow$ G có không quá 2 đỉnh bậc lẻ (có 0 đỉnh bậc lẻ hoặc có 2 đỉnh bậc lẻ).

Đồ thị Euler

Thuật toán tìm chu trình Euler

```
void Euler()
{   stack= ∅; CE= ∅; // CE là tập chứa các đỉnh của chu trình Euler
    Chọn một đỉnh x bất kỳ, cất x vào stack //x gọi là đỉnh xuất phát.
    While (stack ≠ ∅)
    {       x = phần tử ở đỉnh stack ;
            if (x còn đỉnh kề)
            {
                    chọn y kề x, cất y vào stack;
                    loại bỏ cạnh (x,y)
            }
            else //x khong con dinh ke
            {
                    lấy x  ra khỏi stack ; cất x vào tập CE }
            }
    }
    xuất tập CE;
}
```

Đồ thị Euler



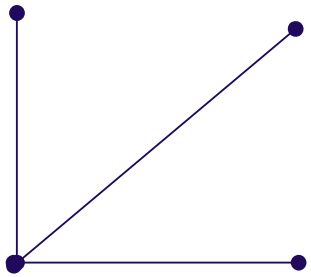
							3											
						5	5	5						2				
			1		4	4	4	4	4				7	7	7			
		3	3	3	3	3	3	3	3	3		6	6	6	6	6		
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

CE = {1, 3, 5, 4, 3, 2, 7, 6, 2, 1}

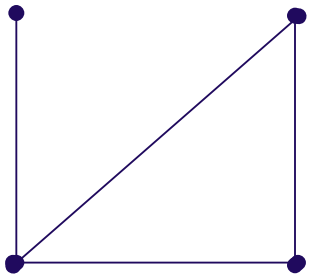
Chu trình Euler:1->3->5->4->3->2->7->6->2->1

Đồ thị Hamilton

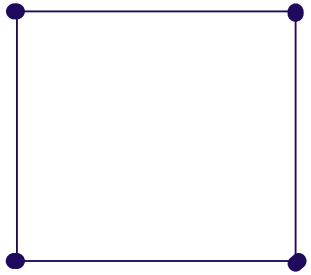
- **Đường đi Hamilton**:
  Là đường đi qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần gọi là đường đi Hamilton.
- **Chu trình Hamilton:**
  Là chu trình qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần gọi là chu trình Hamilton.



G_1



G_2



G_3

Đồ thị Hamilton

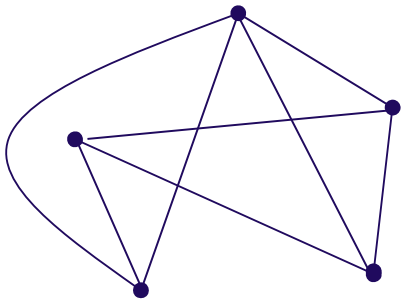
<u>Định lý Dirak</u>:

a/ G là đơn đồ thị vô hướng có n đỉnh (n>2).

$\forall$ đỉnh u, deg(u) $\geq$ n/2 $\Rightarrow$G là đồ thị Hamilton

b/ G là đơn đồ thị có hướng liên thông với n đỉnh.

$\forall$ đỉnh u, $deg^+(u) \geq n/2$, $deg^-(u) \geq n/2$, $\Rightarrow$G là đồ thị Hamilton.



Đồ thị Hamilton

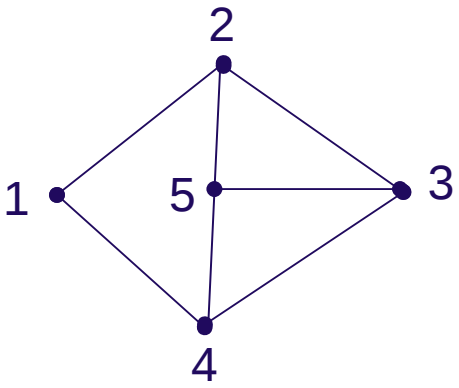
Thuật toán liệt kê các chu trình và đường đi Hamilton:

```
void  hamilton(i) //tìm đỉnh thứ i trên chu trình hamilton
{
    for (j ∈ kề(i – 1))
    {
        if (i=n+1) và (j=v0) {
            Xuất chu trình x[1], x[2],…,x[1]
        else
            if(! Tham[j]) {
                x[i]=j; //Chọn j làm đỉnh thứ i trong chu trình
                tham[j]=1;
                Timdinh(i+1);
                tham[j]=0;
             }
     }
}
```

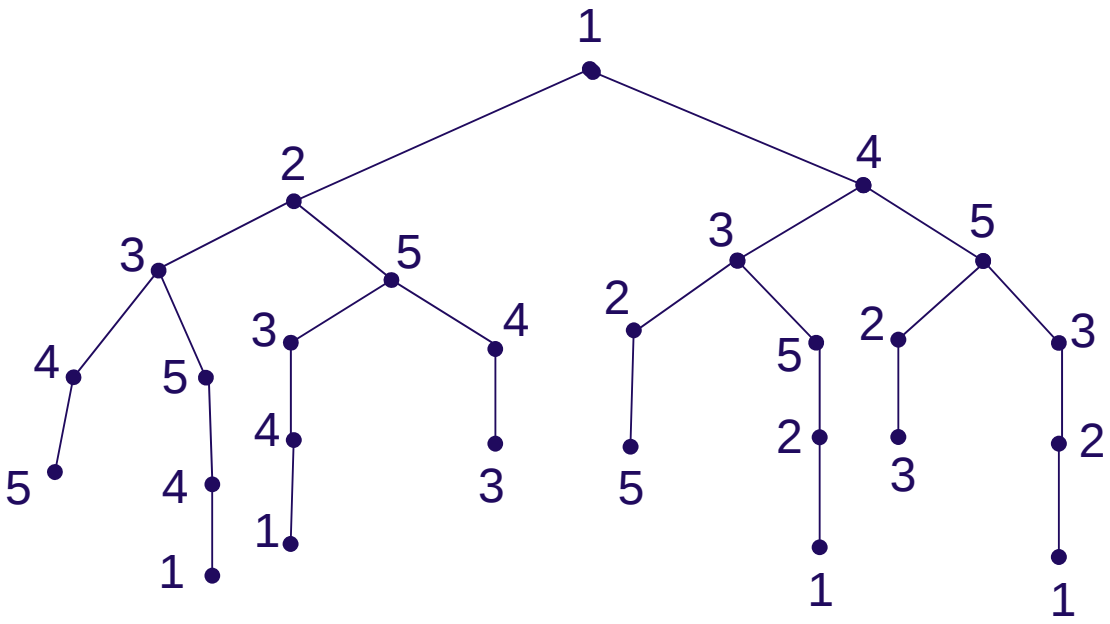
Đồ thị Hamilton

```
void Hamilton()
{
    for i=1 to n
        tham[i]=0; //gán tất cả các đỉnh là chưa thăm
    x[1]=v0;
    tham[v0]=1;
    timdinh(2); //gọi hàm tìm đỉnh thứ 2
}
```

Đồ thị Hamilton



Cây đường đi và chu trình Hamilton

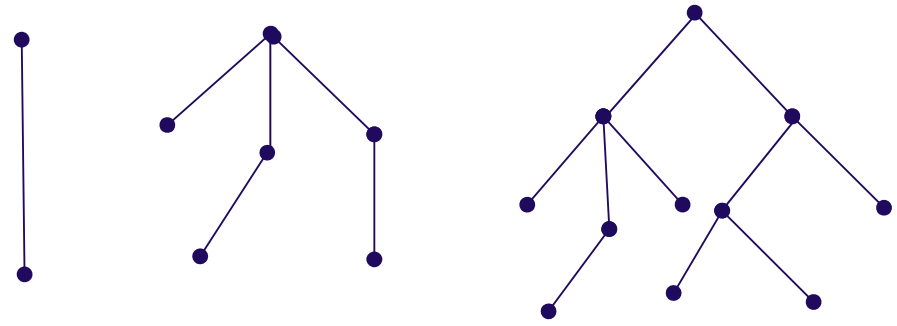


Bài tập

- Viết chương trình kiểm tra đồ thị Euler, đồ thị Hamilton
- Viết chương trình tìm chu trình Euler và đường đi Euler
- Viết chương trình tìm chu trình Hamilton và đường đi hamilton

Định nghĩa:

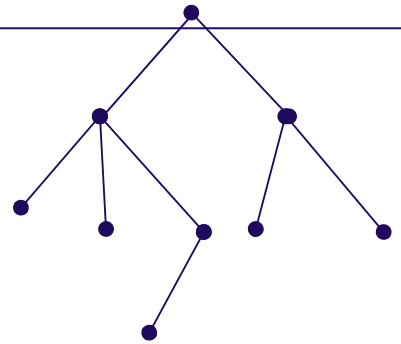
- Cây là một đồ thị liên thông không có chu trình
- Rừng là một đồ thị có nhiều thành phần liên thông, mỗi thành phần liên thông là một cây.



Rừng gồm 3 cây T_1, T_2, T_3

Định lý: Cho T là một đồ thị có n≥ 2 đỉnh. Những điều sau đây tương đương

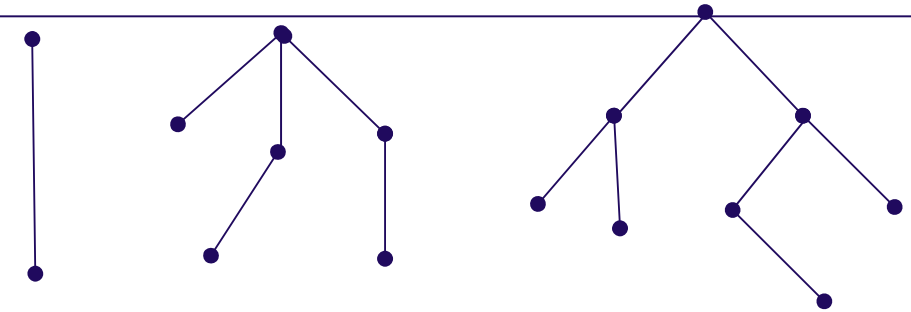
- *T là cây*
- *T không có chu trình và có n – 1 cạnh*
- *T liên thông và có n – 1 cạnh*
- *T liên thông và mỗi cạnh là một cầu*



T

Cây T có 9 đỉnh, 8 cạnh

Hệ luận: Nếu G là một rừng n đỉnh, p cây thì số cạnh của G là: m=n-p



T_1 T_2 T_3

Rừng có số đỉnh: 16

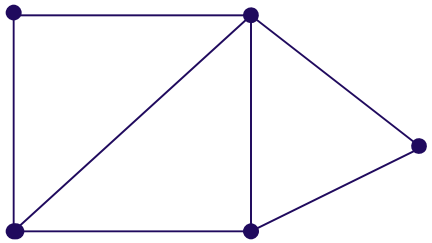
Số cây: 3

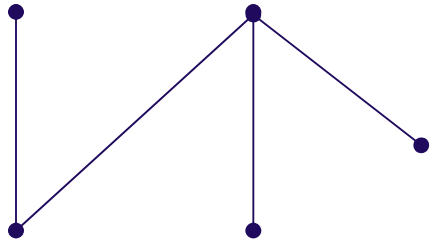
Số cạnh: 16-3=13

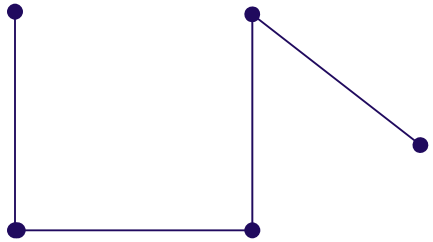
Cây khung của đồ thị

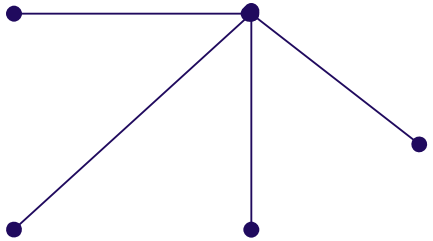
Định nghĩa cây khung:

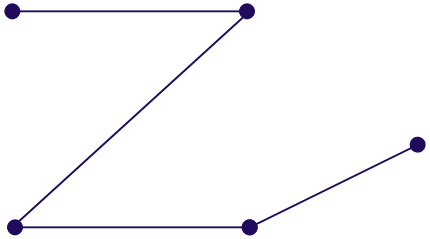
Cho G=(V,E) là đồ thị vô hướng liên thông. Cây khung của G là cây T=(V,F) với $F \subset E$

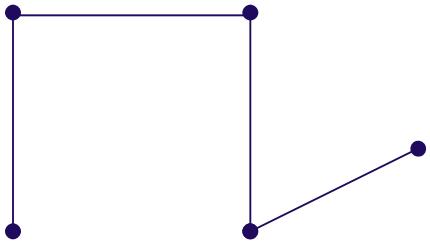












Thuật toán tìm cây khung

Thuật toán DFS:

```
void DFS1(v)
{
    tham[v]=1;    //ghi nhận là đã thăm v để về sau không thăm nữa.
    For (u ∈ Ke(v))    // xét tất cả các đỉnh u kề với v
    if (!tham[u])
    {                  //T la tap chua cac canh cua cay khung
        T=T∪(v,u);    //them canh (v,u) vao tap T
        DFS1(u);     //neu u chua thăm, thăm u
    }
}
void DFS()
{
    for  (v ∈ V)  tham[v]=0;
    T=∅;            // T là tập cạnh của cây khung, khởi trị ban đầu là
    rỗng
    DFS1(dxp);   // dxp la mot dinh xuat phat nao do cua do thị
    xuất tập T;
}
```

Thuật toán tìm cây khung

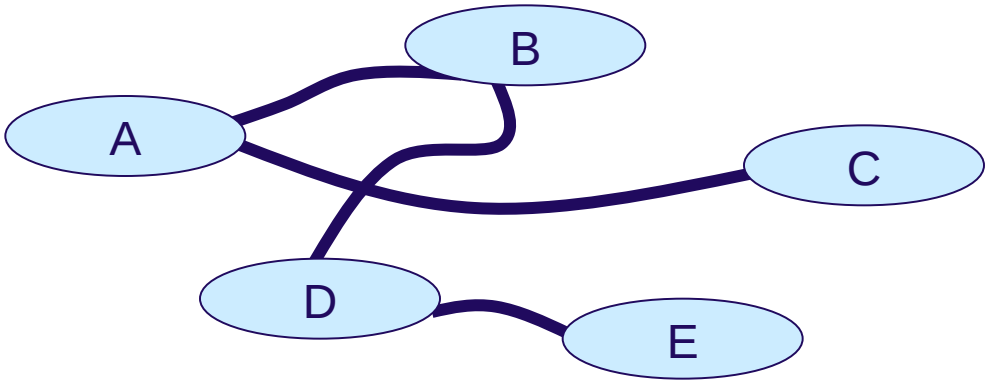
Thuật toán BFS:

```
void BFS1(v)
{   queue=∅;  push(queue,v); tham[v]=1;
    while  (queue ≠ ∅) {
          v=pop(queue);
          for (u ∈ Ke(v))
                If (!tham[u]) {
                      push(queue,u); tham[u]=1;
                      T=T∪(v,u); //them canh (v,u) vao tap T
                }
    }
}
void BFS()
{   for (v ∈ V)  tham[v]=0;
    T=∅; // T là tập cạnh của cây khung
    BFS1(dxp); //dxp la mot dinh xuất phát nào đó của đồ thị
    xuat tap T;
}
```

Cây khung ngắn nhất

Bài toán xây dựng hệ thống đường sắt:

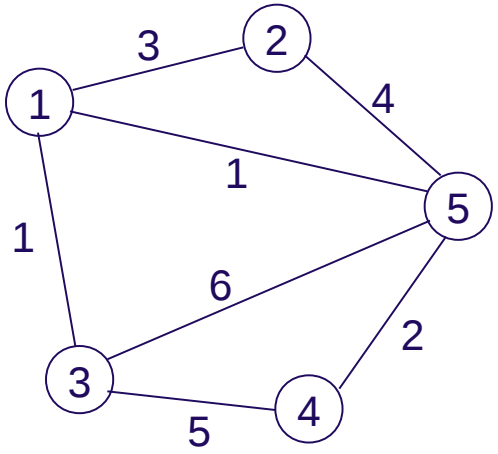
Cần xây dựng một hệ thống đường sắt nối n thành phố sao cho giữa 2 thành phố bất kỳ luôn có đường đi và tổng chi phí xây dựng là nhỏ nhất.

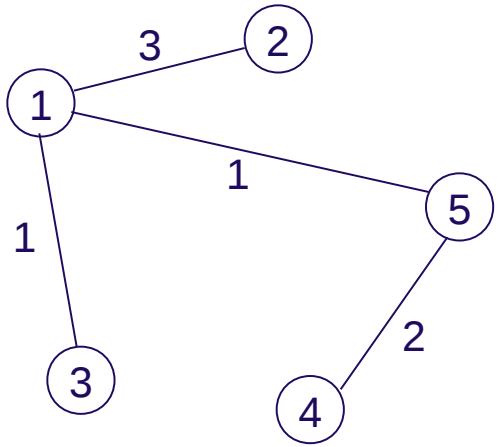


Cây khung ngắn nhất

Định nghĩa cây khung ngắn nhất:

Cho một đồ thị vô hướng G, trên mỗi cạnh của G được gán một trọng số c(i,j). Cây khung T của G được gọi là cây khung ngắn nhất nếu tổng trọng số các cạnh của T là nhỏ nhất





Thuật toán tìm cây khung ngắn nhất

Thuật toán Kruskal:

Ý tưởng:

Bước 1: Săp các cạnh theo thứ tự không giảm của trọng số, T=∅

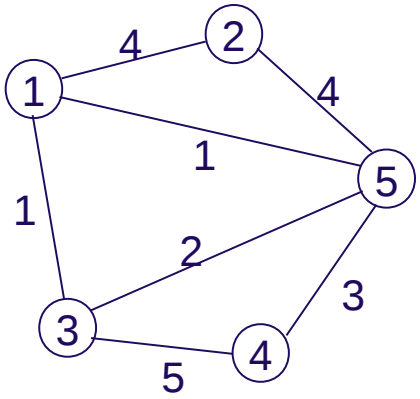
Bước 2: Lần lượt duyệt trong danh sách cạnh đã săp xếp theo thứ tự trọng số từ nhỏ đến lớn, chọn cạnh bổ sung vào tập T với điều kiện việc bổ sung này không tạo thành chu trình.

Bước 3: Tiếp tục thực hiện bước 2 cho đến khi T có n-1 cạnh thì dừng.

Thuật toán Kruskal

Săp các cạnh theo thứ tự không giảm của trọng số

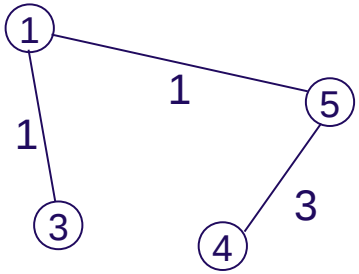
Cạnh	(1,3)	(1,5)	(3,5)	(4,5)	(1,2)	(2,5)	(3,4)
Trọng số	1	1	2	3	4	4	5



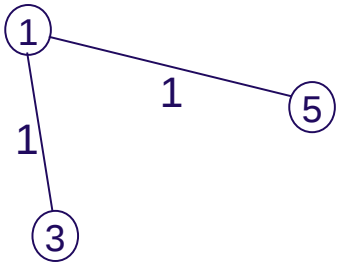
T=∅

Bổ sung vào T cạnh (1,3)

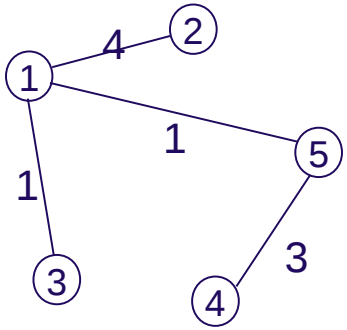
Bổ sung vào T cạnh (4,5)



Bổ sung vào T cạnh (1,5)



Bổ sung vào T cạnh (1,2)



Thuật toán Kruskal

```
void Kruskal()
{
    T=∅ ;
    while ((|T| < n-1) && (E ≠ ∅)) //T chưa đủ n-1 cạnh và còn cạnh để chọn
    {
        Chọn e là cạnh có độ dài nhỏ nhất trong E;
        E = E\{e}; //loại e khỏi tập cạnh
        if  (T ∪ {e} không chứa chu trình)
            T= T ∪ {e} ; //thêm e vào T
    }
    if  (|T| < n-1)  đồ thị không liên thông;
    else Xuất cây T
}
```

Thuật toán Prim

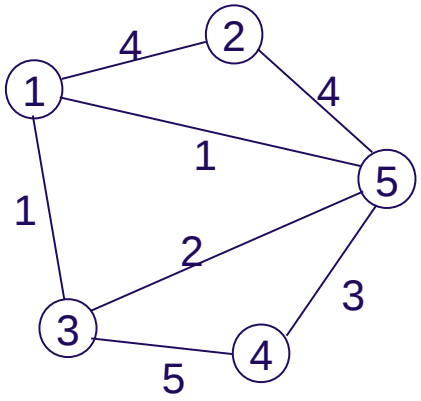
Ý tưởng:

Bước 1: Gọi T là tập chứa các cạnh của cây khung; T=∅.
S là tập chứa các đỉnh của cây khung; S={x} (với x là đỉnh bắt kỳ thuộc đồ thị)

Bước 2: Tìm cạnh (x,y) ∈ w(s) sao cho cạnh này có trọng số bé nhất trong w(s).
T=T∪(x,y); S=S∪{y}

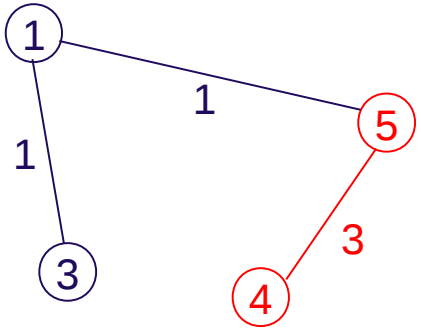
Bước 3: Lặp bước 2 đến khi T có n-1 cạnh thì dừng.

Thuật toán Prim

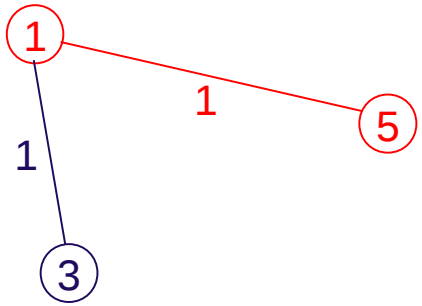


T=∅, S={1},
w(s)={(1,2), (1,3), (1,5)}
chọn (1,3)
T=T∪(1,3)={(1,3)}

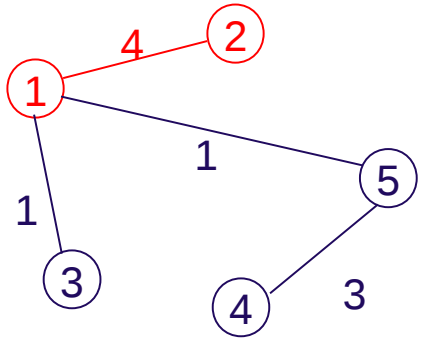
S={1,3,5},
w(s)={(1,2), (2,5), (3,4), (4,5)}
chọn (4,5)
T=T∪(4,5)={(1,3), (1,5), (4,5)}



S={1,3}
w(s)={(1,2), (1,5), (3,4),(3,5)}
chọn (1,5)
T=T∪(1,5)={(1,3), (1,5)}



S={1,3,5,4},
w(s)={(1,2), (2,5)}
chọn (1,2)
T=T∪(1,2)={(1,3), (1,5), (4,5),(1,2)}



Thuật toán Prim

```
void  Prim()
{
    //bước khởi tạo
    chon s la mot dinh nao do cua do thi;
    V_H={s} ; T= ∅ ; d[s]=0; near[s]=s;
    for (v ∈ V \V_H)
    {
        d[v]=c[v,s]; near[v]=s;
    }
// buoc lap
    stop=false;
    while (! stop)
    {
        Tim u ∈ V \V_H thoa man:d[u] =min{d[v]: v ∈ V \V_H } ;
        V_H= V_H { u} ; T = T ∪ { (u, near[u])} ;
```

Thuật toán Prim

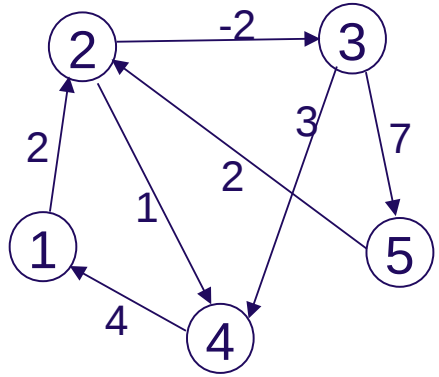
```
    if  (| V_H| ==n)
    {
        H=( V_H,T) la cay khung nho nhat cua do thi;
        stop:=true;
    }
    else
        for (v ∈ V\ V_H)
            if (d[v]>c[v,u]) //neu v gan dinh u moi ket nap vao cay
                                khung
            {
                d[v]=c[v,u]; //cap nhat lai nhan
                near[v]=u;
            }
    }
}
```

CHƯƠNG 6: ĐƯỜNG ĐI NGẮN NHẤT

Một số khái niệm:

Cho G=(V,E) là một đồ thị có hướng được biểu diễn bởi ma trận trọng số A

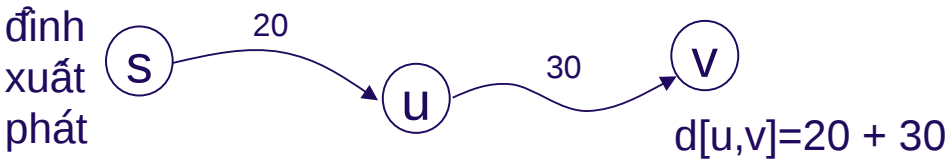
- Trọng số của một cung (u,v) ký hiệu là a[u,v] là giá trị của ma trận trọng số tại dòng u cột v.



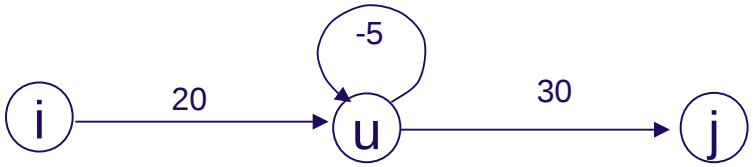
	1	2	3	4	5
1	0	2	∞	∞	∞
2	∞	0	-2	1	∞
3	∞	∞	0	3	7
4	4	∞	∞	0	∞
5	∞	2	∞	∞	0

Một số khái niệm:

- Đường di ngắn nhất từ đỉnh s đến đỉnh v là đường đi có độ dài nhỏ nhất từ s đến v, ký hiệu là d[s,v] hay d[v] (được hiểu ngầm từ s đến v)



- Nếu G có chu trình độ dài âm trên đường đi từ i đến j thì đường đi từ i đến j không tồn tại



Thuật toán Ford-Bellman

Điều kiện: Đồ thị không có chu trình âm

Thuật toán:

Dữ liệu vào:

Đồ thị có hướng G=(V,E) với n đỉnh,

s là đỉnh xuất phát, a[u,v] là ma trận trọng số thực;

Giả thiết: Đồ thị không có chu trình âm.

Dữ liệu ra:

d[v] là khoảng cách từ đỉnh s đến tất cả các đỉnh v còn lại

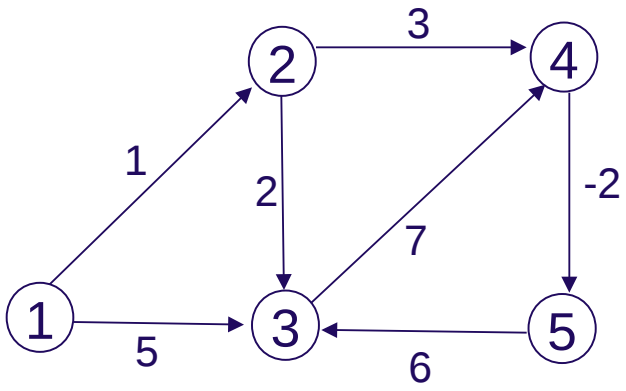
Trước[v] ghi nhận đỉnh đi trước v trong đường đi ngắn nhất từ s đến v.

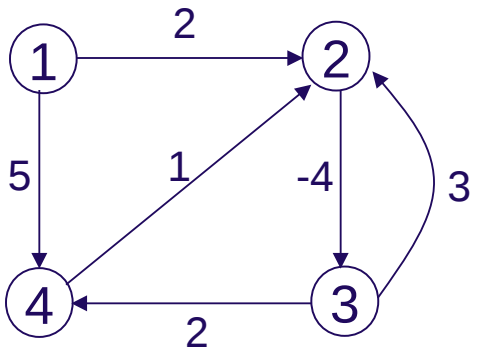
Thuật toán Ford-Bellman

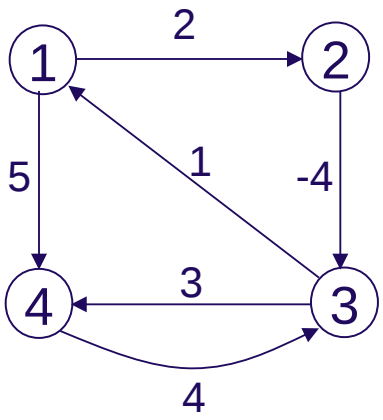
```
void Ford_Bellman()
{
    // Khởi tạo
    for (v ∈ V)
    {
        d[v]=a[s,v]; Truoc[v]=s; //s là đỉnh xuất phát
    }
    d[s]=0;
    for (k=1;k<= n;k++) //số bước lặp
        for (v ∈ V\{s})
            for (u ∈ V)
                if (d[v]>d[u] +a[u,v])
                {
                    d[v]=d[u]+a[u,v];
                    Truoc[v]=u;
                }
}
```

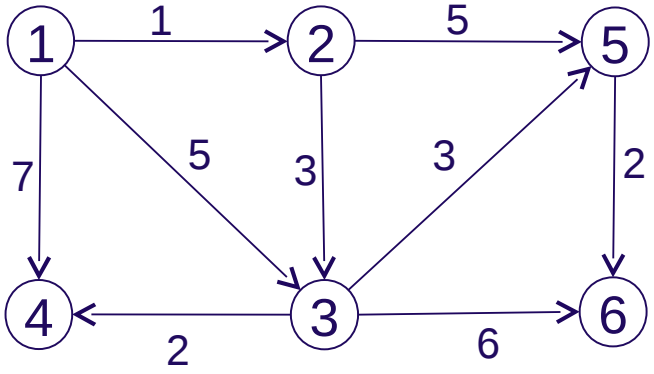
Thuật toán Ford-Bellman

Tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh khác:









Thuật toán Dijkstra

Điều kiện: Đồ thị không trọng số âm

Thuật toán:

Dữ liệu vào:

Đồ thị có hướng G=(V,E) với n đỉnh,

s là đỉnh xuất phát, a[u,v] >=0

Dữ liệu ra:

Khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại d[v]

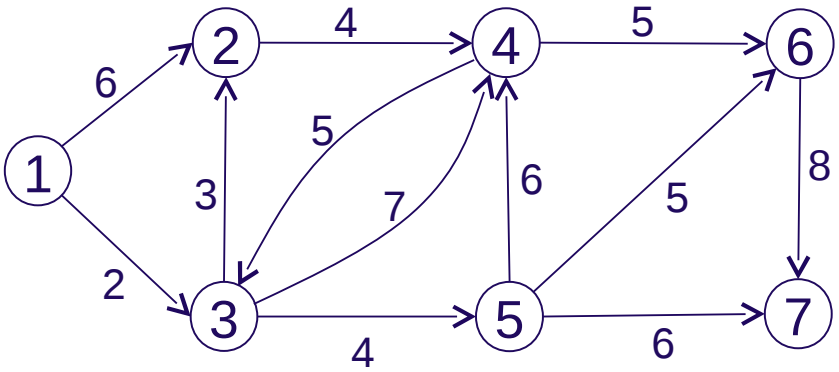
Truoc[v], đỉnh đi trước v trong đường đi ngắn nhất từ s đến v

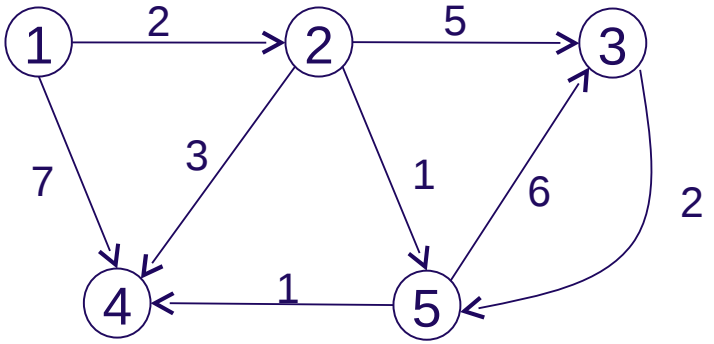
Thuật toán Dijkstra

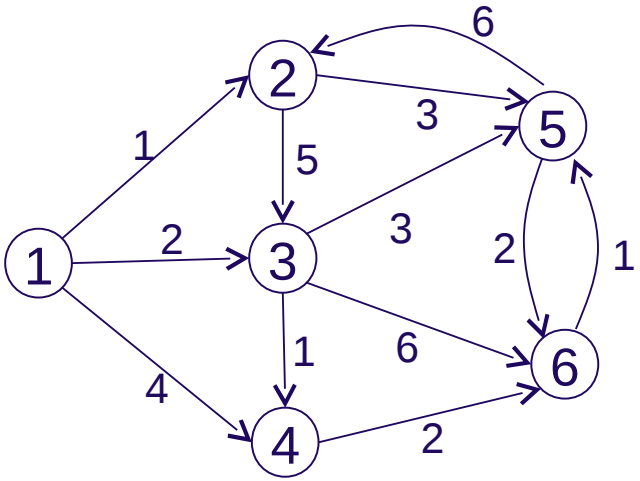
```
void Dijkstra()
{
    for (v ∈ V) // Khởi tạo
    {
        d[v]=a[s,v]; Truoc[v]=s;
    }
    d[s]=0; T:=V\{s}; // T là tập các đỉnh có nhãn tạm thời
    while (T !=∅)
    {
        Tìm đỉnh u ∈ T thoả mãn d[u]=min {d[z]: z ∈ T} ;
        T=T\{u} ; // Cố định nhãn của đỉnh u
        for (v ∈ T)
        if (d[v]>d[u]+a[u,v])
        {
            d[v]=d[u]+a[u,v]; Truoc[v]=u;
        }
    }
}
```

Thuật toán Dijkstra

Tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh khác:







Thuật toán Floyd

- Tìm đường đi giữa tất cả các cặp đỉnh
- Có thể giải bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị bằng cách sử dụng n lần thuật toán mô tả ở phần trước, trong đó ta sẽ chọn s lần lượt là các đỉnh của đồ thị. Khi đó ta thu được thuật toán với độ phức tạp O(n4) (Ford_Bellman) hoặc O(n3) (Dijkstra)

Ta có thuật toán Floyd tốt hơn với độ phức tạp O(n3)

Dữ liệu vào:

Đồ thị cho bởi ma trận trọng số a[i,j], i, j =1, 2,. . ,n.

Dữ liệu ra:

Ma trận đường đi ngắn nhất giữa các cặp đỉnh: d[i,j] cho độ dài đường đi ngắn nhất từ đỉnh i đến đỉnh j.

Ma trận ghi nhận đường đi: q[i,j] ghi nhận đường đi ngắn nhất từ i đến j.

Thuật toán Floyd

Ý tưởng:

- Tìm đường đi giữa tất cả các cặp đỉnh của G
- Sử dụng 2 ma trận $D_{i,j}$ (lưu trọng số)và $Q_{i,j}$ (lưu đường đi)
- Bước khởi tạo (bước 1) tính:

$D_1 = D_{i,j}$ là ma trận trọng số của G

$$Q_1 = Q_{i,j} = \begin{cases} j \text{ nếu } (i,j) \in E \\ 0 \text{ nếu } (i,j) \notin E \end{cases}$$

- Tại bước lặp thứ k ta tính D_k và Q_k như sau:

$$D_k(i,j) = \begin{cases} D_{k-1}(i,k) + D_{k-1}(k,j) & \text{nếu } D_{k-1}(i,j) > D_{k-1}(i,k) + D_{k-1}(k,j) \\ D_{k-1}(i,j) & \text{nếu ngược lại} \end{cases}$$

$$Q_k(i,j) = \begin{cases} Q_{k-1}(i,k) & \text{nếu } D_{k-1}(i,j) > D_{k-1}(i,k) + D_{k-1}(k,j) \\ Q_{k-1}(i,j) & \text{nếu ngược lại} \end{cases}$$

- Thuật toán dừng khi đã thực hiện n bước lặp

Thuật toán Floyd

Xét bước lặp thứ k = 6

D_5 =

K=5

	1	2	3	4	k	6	7
1	0	6	4	7	∞	9	∞
2	8	0			4		
3		∞		9	7		
4		2			9		
k	2	∞	8	1	0	∞	5
6	6	4	∞	1	5	0	
7		1			6		0

D_6 =

K=6

	1	2	3	4	5	6	7
1	0	6	4	7	∞	9	∞
2	6	0			4		
3		∞		8	7		
4		2			9		
5	2	∞	8	1	0	∞	5
6		4			5		
7		1			6		

Thuật toán Floyd

K=5

Q_5 =

	1	2	3	4	k	6	7
1							
2	3					4	
3				6		7	
4							
k							
6							
7							

K=6

Q_6 =

	1	2	3	4	5	6	7
1							
2	4						
3				7			
4							
5							
6							
7							

Thuật toán Floyd

```
void Floyd()
{
    for i=1 to n
    for j=1 to n //Bước khởi tạo
    {
        d[i][j]=a[i][j];
        if (d[i][j] = ∞ hay d[i][j] = 0)
            q[i][j]=0;
        else    q[i][j]=j;
    }
    for k=2 to n // bước lặp
        for i=1 to n
            for j=1 to n
            if (d[i][j]>d[i][k]+d[k][j])
            {
                d[i][j]=d[i][k]+d[k][j]; q[i][j]=q[i][k];
            }
}
```

Thuật toán Floyd

Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh: