MỤC LỤC

CHƯƠNG 1

ĐẠI CƯƠNG VỀ ĐỒ THỊ

Mục tiêu:

- Hiểu được các khái niệm, định nghĩa liên quan đến đồ thị có hướng và đồ thị vô hướng như bậc, các dạng đỉnh, cạnh, liên thông, chu trình, đường đi.
- Phân biệt được các dạng đồ thị đặc biệt.
- Thực hiện được các phép biến đổi và các phép toán trên đồ thị.
- Mô hình hóa một bài toán thực tế về dạng đồ thị.

1.1. CÁC KHÁI NIỆM CƠ BẢN

1.1.1. Đồ thị

Định nghĩa: Một đồ thị là một bộ **G = (V, E)**, trong đó:

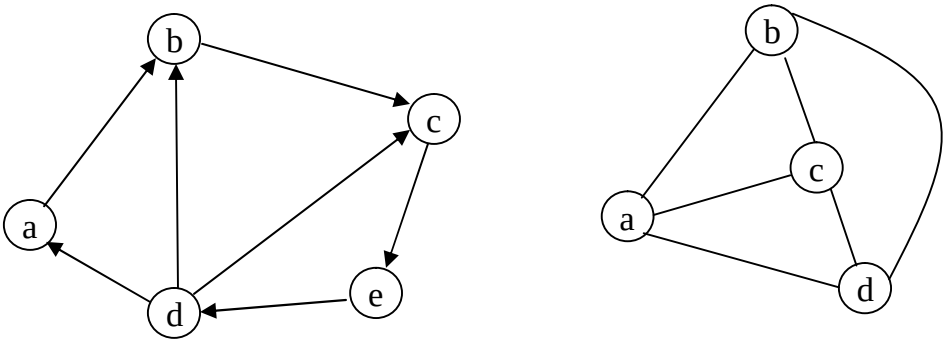
$V \neq \emptyset$ là tập hợp các đỉnh,

$E=\{(u,v) \mid u, v \in V\}$ là tập hợp các cạnh.

Cặp đỉnh $(x, y) \in E$ không có thứ tự được gọi là cạnh vô hướng, ngược lại gọi là cạnh có hướng. Cạnh có hướng còn được gọi là cung.

Đồ thị chỉ gồm các cạnh có hướng được gọi là *đồ thị có hướng*, đồ thị chỉ gồm các cạnh vô hướng được gọi là *đồ thị vô hướng*.

Ví dụ 1.1:

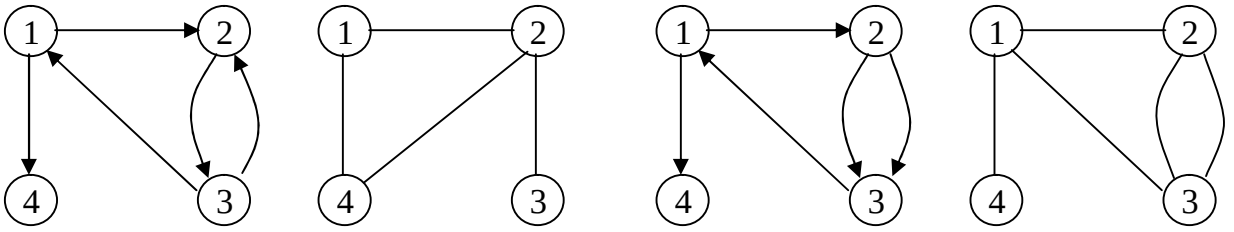


Hình 1.1: Đồ thị có hướng và đồ thị vô hướng

Đơn đồ thị: Một đồ thị G=(V,E) được gọi là *đồ thị đơn nếu mỗi cặp đỉnh chỉ có tối đa 1 cạnh/cung.*

Đa đồ thị: Một đồ thị G=(V, E) được gọi là đa đồ thị nếu có ít nhất một cặp đỉnh được nối với nhau bởi nhiều cạnh/cung **cùng chiều**.

Ví dụ 1.2:

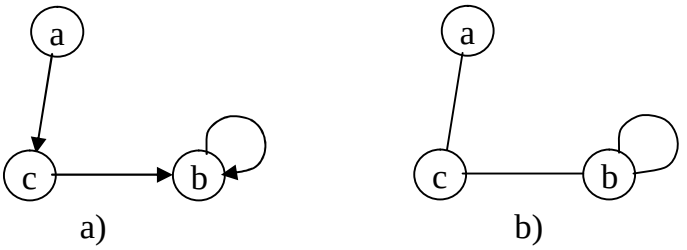


Hình 1.2: a,b: đơn đồ thị; c,d: đa đồ thị

Một số thuật ngữ:

- **Khuyên**: *Cạnh/cung được gọi là khuyên nếu có đỉnh đầu trùng với đỉnh cuối.*

  Ví dụ 1.3:



Hình 1.3: Khuyên tại đỉnh b của đồ thị có hướng và vô hướng

- **Cạnh/cung song song**: *Hai cạnh/cung được gọi là song song (hay lặp) nếu chúng có 2 đỉnh trùng nhau.*

  Ví dụ 1.4:

Hình 1.4: Cạnh có hướng song song và cạnh vô hướng song song

- **Đỉnh kề**: Nếu (u,v) là một cung của đồ thị có hướng thì đỉnh v được gọi là kề của đỉnh u. Nếu (u,v) là một cạnh của đồ thị vô hướng thì đỉnh v kề với đỉnh u và đỉnh u cũng kề với đỉnh v.
- **Cạnh kề**: *Hai cạnh/cung được gọi là kề nhau nếu chúng có chung một đỉnh*.
- **Cạnh liên thuộc**: *Cạnh (u,v) được gọi là liên thuộc (hay kề) với 2 đỉnh u, v.*
- **Bậc của đỉnh**: *Số cạnh liên thuộc với đỉnh v gọi là bậc của v, ký hiệu d(v).* Đối với đỉnh có khuyên thì bậc tính là 2 cho mỗi khuyên.

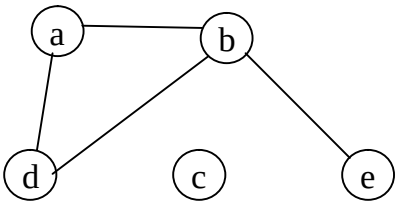
Ví dụ 1.5:

Trong hình 1.2b bậc của đỉnh 1 là 2, bậc của đỉnh 2 là 3, bậc của đỉnh 3 là 1, bậc của đỉnh 4 là 2.

Trong hình 1.3b bậc của đỉnh a là 1, bậc của đỉnh b là 3, bậc của đỉnh c là 2.

- **Đỉnh cô lập, đỉnh treo**: *Trong đồ thị vô hướng đỉnh có bậc 0 gọi là đỉnh cô lập, đỉnh có bậc 1 gọi là đỉnh treo.*

Ví dụ 1.6: Trong đồ thị dưới đây, đỉnh c là đỉnh cô lập (có bậc 0), đỉnh e là đỉnh treo (có bậc 1)



Hình 1.5: Đỉnh cô lập và đỉnh treo

- **Cung vào/ra**: Cung (u,v) gọi là cung ra khỏi đỉnh u và là cung vào đỉnh v
- **Bán bậc của đỉnh**: Số cung vào đỉnh v gọi là bán bậc vào (còn gọi là nữa bậc trong) của v, ký hiệu là $d^-(v)$. Số cung ra khỏi đỉnh v gọi là bán bậc ra (còn gọi là nữa bậc ngoài) của v, ký hiệu $d^+(v)$.

Ví dụ 1.7: Với đồ thị có hướng trong hình 1.2a thì bán bậc ra và bán bậc vào của các đỉnh được cho trong bảng dưới đây:

Đỉnh v	**$d^-(v)$**	**$d^+(v)$**
1	1	2
2	2	1
3	1	2
4	1	0

Định lý: Cho đồ thị G=(V,E), m là số cung hay số cạnh của G

i) Nếu G có hướng thì $\sum_{i\in V} d^-(i) = \sum_{i\in V} d^+(i) = m$

ii) $\sum_{i\in V} d(i) = 2m$

iii) Số đỉnh bậc lẻ là một số chẵn

Chứng minh:

i) Vì mỗi cung (u,v) được tính bậc một lần bậc ra của đỉnh u và một lần bậc vào của đỉnh v nên số bậc ra = số bậc vào = số cung.

ii) Mỗi cạnh (u,v) được tính bậc một lần trong d(u) và một lần trong d(v), như vậy mỗi cạnh được tính 2 bậc nên tổng số bậc = 2 lần số cạnh.

iii) Gọi V_1 ={các đỉnh có bậc lẻ}; V_2 ={các đỉnh có bậc chẵn}

theo ii) thì $\sum_{i\in V} d(i) = \sum_{i\in V1} d(i) + \sum_{i\in V2} d(i) = 2m$ (1)

Do $\forall$ v$\in V_2$ thì d(v) chẵn nên $\sum_{i\in V2} d(i)$ chẵn (vì tổng các số chẵn là số chẵn)

nên từ (1) suy ra $\sum_{i\in V1} d(i)$ là số chẵn, mà d(i) lẻ ($\forall$i$\in V_1$) nên $\sum_{i\in V1} d(i)$ là tổng các số lẻ có giá trị là số chẵn, suy ra số các đỉnh bậc lẻ là một số chẵn.

1.1.2. Đường đi, chu trình, đối chu trình

Đường đi: Đường đi trong đồ thị là một dãy các đỉnh $v_1, v_2, v_3,\ldots, v_n$ với $(v_i, v_{i+1}) \in E$, i=1, 2,.., n. Số cạnh của đường đi được gọi là độ dài của đường đi.

Chu trình: Là đường đi mà có đỉnh đầu trùng với đỉnh cuối.

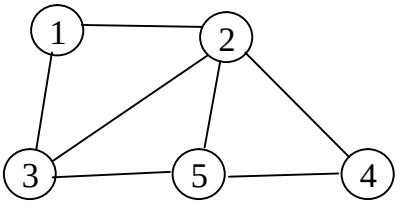
Một đường đi hay chu trình được gọi là đơn nếu không có cạnh/cung bị lặp lại, gọi là sơ cấp nếu không có đỉnh nào bị lặp lại.

Đối chu trình: Cho đồ thị G=(V,E) và A$\subset$V, đối chu trình xác định bởi A ký hiệu w(A) được định nghĩa như sau:

w(A)={e$\in$E/ e có một đỉnh trong A}

Một đối chu trình w được gọi là sơ cấp nếu không tồn tại w'$\subset$w sao cho G-w' không liên thông (tính liên thông xem phần 1.1.3).

Ví dụ 1.8: Xét đồ thị vô hướng G:



Hình 1.6: Đồ thị vô hướng

Một số đường đi, chu trình và đối chu trình trên đồ thị như sau:

1, 3, 5, 2, 3, 5, 4 (đường đi không đơn vì có cạnh (3,5) bị lặp lại)

1, 2, 3, 5, 2, 4 (đường đi đơn vì không có cạnh nào lặp lại nhưng không sơ cấp vì có đỉnh 2 bị lặp lại)

1, 2, 3, 5, 4 (đường đi sơ cấp)

4, 5, 2, 3, 5, 2, 4 (Chu trình không đơn vì có cạnh (5,2) bị lặp lại)

5, 2, 3, 1, 2, 4, 5 (chu trình đơn vì không có cạnh nào lặp lại nhưng không sơ cấp vì có đỉnh 2 bị lặp lại)

4, 5, 3, 2, 4 (chu trình sơ cấp)

A={3,5}, w(A)={(3,1),(3,2),(5,2),(5,4)} không sơ cấp vì có w'={(5,2)}và G-w' không liên thông.

A={1,2}, w(A)={(1,3),(2,3),(2,5),(2,4)} là sơ cấp vì không tồn tại w'⊂w để G-w' không liên thông.

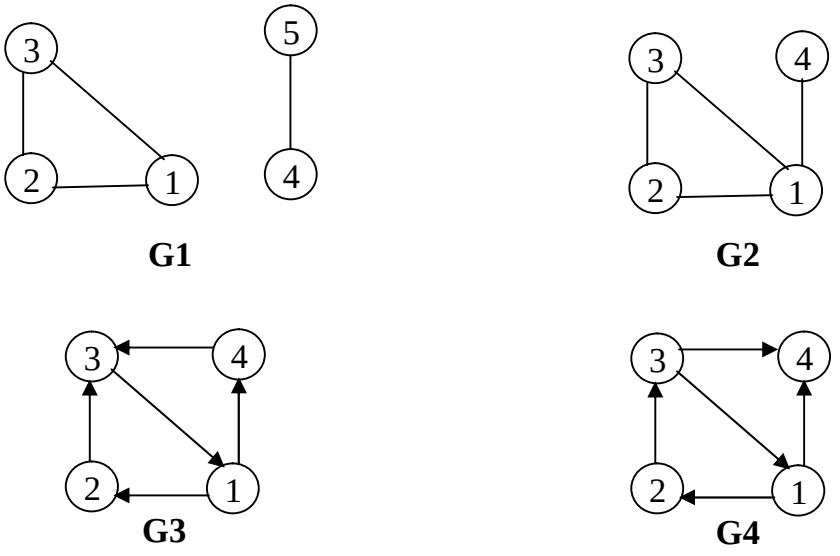
1.1.3. Tính liên thông

Đồ thị liên thông: Một đồ thị G=(V,E) được gọi là liên thông nếu hai đỉnh bất kỳ u,v ∈ V luôn có đường đi từ u đến v và ngược lại. Một đồ thị vô hướng không liên thông thì luôn chia được thành các đồ thị con liên thông, mỗi đồ thị con gọi là một thành phần liên thông.

Liên thông mạnh: Đồ thị G có hướng liên thông thì được gọi là liên thông mạnh.

Liên thông yếu: Nếu đồ thị G có hướng không liên thông nhưng đồ thị vô hướng tương ứng liên thông thì G được gọi là liên thông yếu.

Ví dụ 1.9:



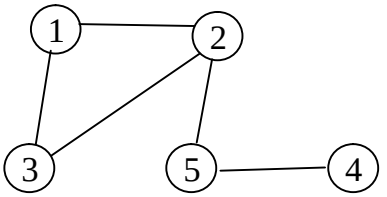
Hình 1.7: Đồ thị liên thông, không liên thông

Trong hình trên, đồ thị G1 có 2 thành phần liên thông, đồ thị G2 liên thông, đồ thị G3 liên thông mạnh, G4 liên thông yếu

Đỉnh rẽ nhánh: Đỉnh u được gọi là đỉnh rẽ nhánh nếu việc loại bỏ u cùng với các cạnh liên thuộc làm tăng số thành phần liên thông.

Cạnh cầu: Cạnh e được gọi là cạnh cầu nếu việc loại bỏ e làm tăng số thành phần liên thông

Ví dụ 1.10: Đồ thị trong hình dưới đây có 2 đỉnh rẽ nhánh là 2 và 5. Có 2 cạnh cầu là (2,5), (4,5)

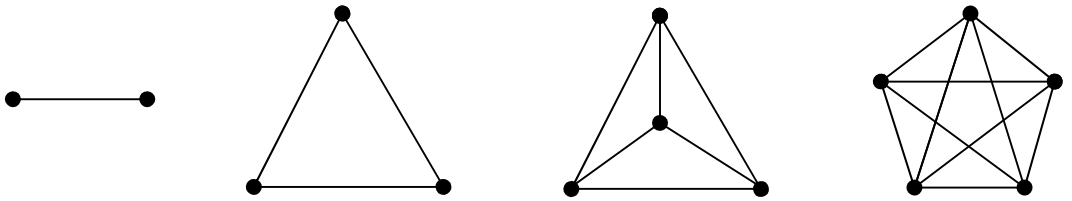


Hình 1.8: Đồ thị có đỉnh rẽ nhánh và cạnh cầu

1.2. CÁC DẠNG ĐỒ THỊ

Đồ thị đủ: Đồ thị đủ cấp n là đơn đồ thị mà giữa 2 đỉnh bất kỳ luôn có một cạnh. Ký hiệu: K_n

Ví dụ 1.11: Hình dưới đây liệt kê một số đồ thị đủ cấp 2, 3, 4, 5



Hình 1.9: Đồ thị đủ cấp 2,3,4,5

Định lý: Đồ thị đủ n đỉnh có n(n-1)/2 cạnh

Chứng minh:

Đỉnh thứ 1 sẽ nối với n-1 đỉnh còn lại nên có n-1 cạnh

Đỉnh thứ 2 sẽ nối với n-2 đỉnh còn lại nên có n-2 cạnh

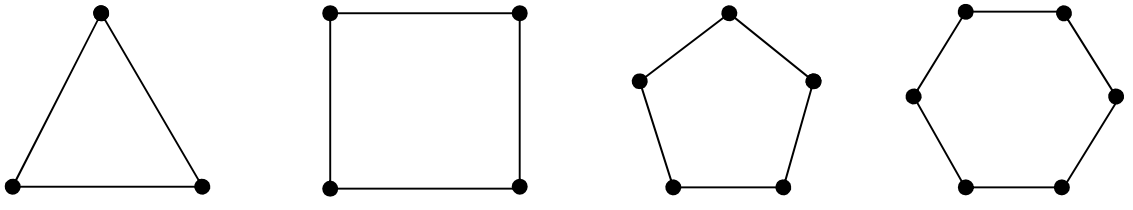
...

Đỉnh thứ n-1 sẽ nối với n-(n-1) đỉnh còn lại nên có 1 cạnh

Số cạnh của K_n là (n-1)+(n-2)+...+1=n(n-1)/2

Đồ thị vòng: Đồ thị vòng n đỉnh ($n \geq 3$) là đồ thị gồm các cạnh (v_1,v_2), (v_2,v_3),..., (v_{n-1},v_n), (v_n,v_1) với v_1, v_2,...,v_n là các đỉnh. Ký hiệu đồ thị là C_n

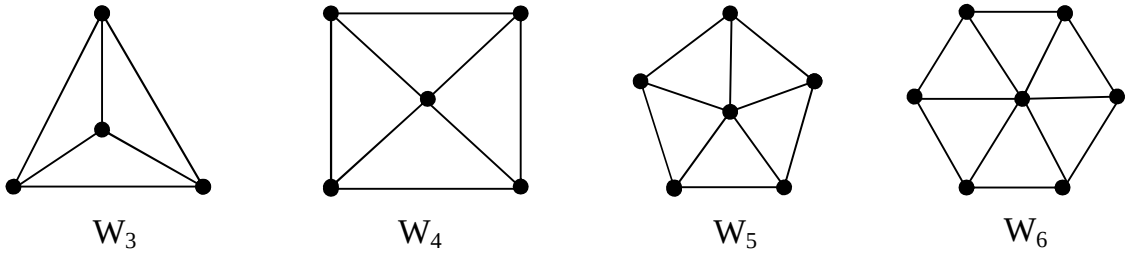
Ví dụ 1.12: Đồ thị vòng cấp 3,4,5,6



Hình 1.10: Đồ thị vòng

Đồ thị bánh xe: Đồ thị vòng C_n được bổ sung thêm một đỉnh mới nối với tất cả các đỉnh còn lại thì được gọi là đồ thị bánh xe, ký hiệu là W_n

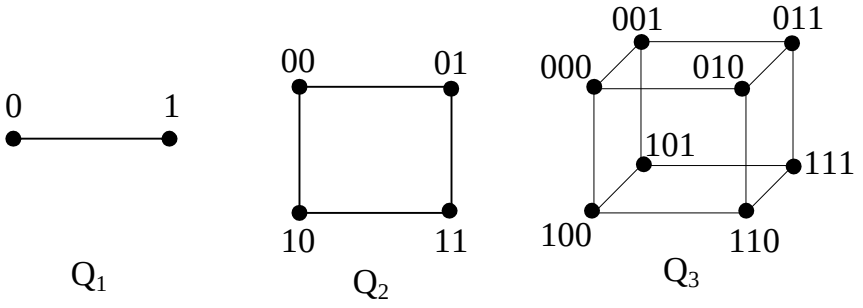
Ví dụ 1.13:



Hình 1.11: Đồ thị bánh xe

Đồ thị lập phương: Đồ thị lập phương ký hiệu Q_n là đồ thị gồm 2^n đỉnh, mỗi đỉnh biểu diễn một chuỗi nhị phân n bit. Trong đó 2 đỉnh kề nhau chỉ khác nhau duy nhất 1 bit.

Ví dụ: 1.14:

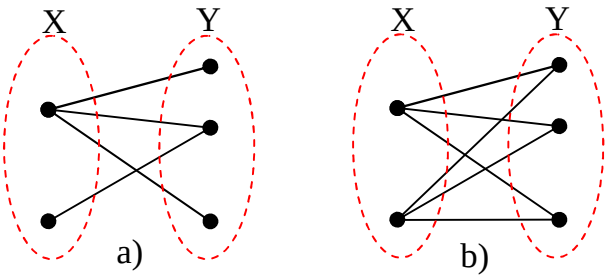


Hình 1.12: Đồ thị lập phương

Đồ thị hai phía: Một đồ thị G=(V, E) được gọi là đồ thị 2 phía nếu như tập đỉnh V của nó có thể phân hoạch thành 2 tập X và Y sao cho mỗi cạnh e ∈ E chỉ nối một đỉnh trong X với một đỉnh trong Y. (Đồ thị 2 phía còn có thể gọi là đồ thị lưỡng phân, đồ thị 2 phần, đồ thị phân đôi). Ký hiệu G=(X∪Y, E)

Đồ thị hai phía đủ: Đồ thị 2 phía G =(X∪Y, E) được gọi là 2 phía đủ nếu mỗi đỉnh trong X được nối với tất cả các đỉnh trong Y hay ngược lại. Ký hiệu $K_{m,n}$. Với m là số đỉnh trong tập X, n là số đỉnh trong tập Y.

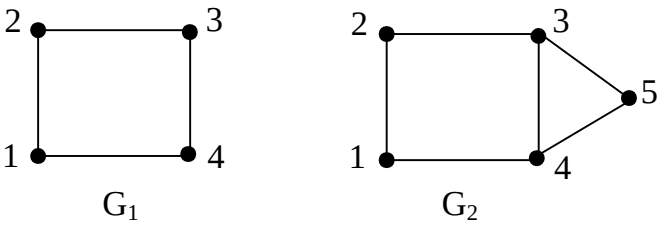
Ví dụ 1.15:



Hình 1.13: Đồ thị 2 phía a), đồ thị 2 phía đủ $K_{2,3}$ b)

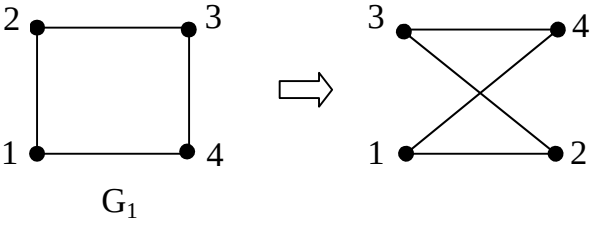
Định lý: G là đồ thị hai phía ⇔ G không có chu trình độ dài lẻ

Ví dụ 1.16:



Hình 1.14: Đồ thị hai phía và không hai phía

Trong hình trên, G_1 là đồ thị 2 phía vì không có chu trình độ dài lẻ, G_2 không là đồ thị hai phía vì có chu trình độ dài lẻ là 3, 4, 5, 3. Đồ thị G_1 có thể biểu diễn lại như sau:



Hình 1.15: Biểu diễn đồ thị hai phía

Thuật toán kiểm tra đồ thị hai phía:

Xét đơn đồ thị vô hướng G=(V,E)

Bước 1: Chọn một đỉnh bất kỳ v ∈ V, đặt X={v}

Bước 2: Tìm tập Y={các đỉnh kề của các đỉnh trong X}. Nếu X∩Y≠∅ thì G không phải là đồ thị hai phía, dừng. Ngược lại qua bước 3

Bước 3: Tìm tập T={các đỉnh kề của các đỉnh trong Y}. Nếu $T \cap Y \neq \varnothing$ thì G không phải đồ thị hai phía, dừng. Nếu T=X thì G là đồ thị hai phía, dừng. Ngược lại, gán X=T và lặp lại bước 2.

Ví dụ 1.17: Xét 2 đồ thị G_1 và G_2 như trong hình 1.14

Dùng thuật toán kiểm tra đồ thị G_1 và G_2 có là hai phía hay không ?

Xét đồ thị G_1:

Đặt X={1}, Y={2,4}, T={1,3}

X={1,3}, Y={2,4}, T={1,3} dừng vì T=X $\Rightarrow$ G là đồ thị hai phía

Xét đồ thị G_2:

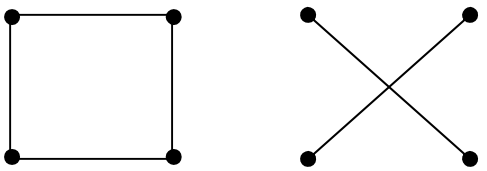
Đặt X={1}, Y={2,4}, T={1,3,5}

X={1,3,5}, Y={2,3,4,5}, dừng vì $X \cap Y=\{3,5\} \neq \varnothing \Rightarrow$ G không là đồ thị hai phía

Đồ thị chính qui bậc k: Đồ thị vô hướng G được gọi là chính qui bậc k (*hay còn gọi là k-đều*) nếu mọi đỉnh của G đều có bậc k. Các đồ thị vòng là chính qui bậc 2, đồ thị đủ K_n chính qui bậc n-1.

Đồ thị bù: Hai đơn đồ thị G và G' được gọi là bù với nhau nếu chúng có chung các đỉnh, cạnh nào thuộc G thì không thuộc G' và ngược lại. Ký hiệu: G' = $\overline{G}$.

Ví dụ 1.18: G và đồ thị bù của G

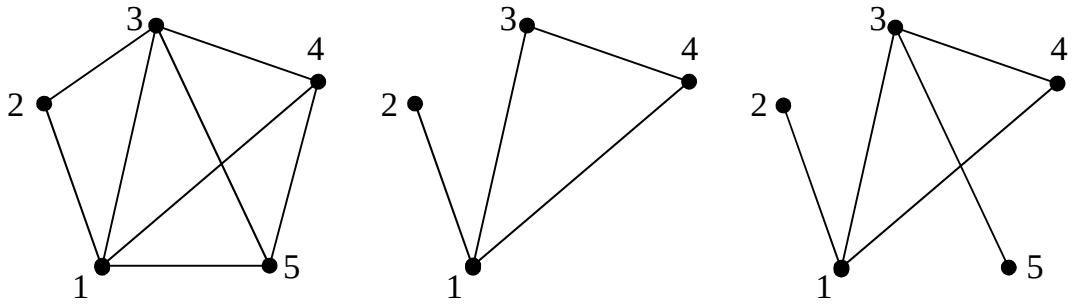


G G'

Hình 1.16: Đồ thị bù

Đồ thị con: Đồ thị G'=(V',E') được gọi là đồ thị con của G=(V,E) nếu $V' \subseteq V$, $E' \subseteq E$ và $\forall(u,v) \in E' \Rightarrow u,v \in V'$. Trường hợp V'=V thì G' được gọi là đồ thị bộ phận hay đồ thị khung của G.

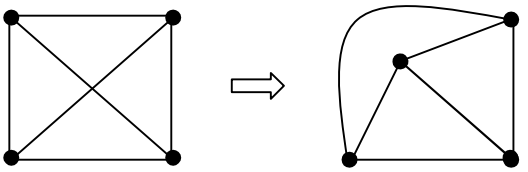
Ví dụ 1.19: Hình 1.16 dưới đây trình bày đồ thị G' là đồ thị con, G" là đồ thị bộ phận của G



Hình 1.17: Đồ thị con và đồ thị bộ phận

Đồ thị phẳng: Đồ thị G được gọi là đồ thị phẳng nếu có thể vẽ G trên một mặt phẳng sao cho các cạnh của nó không cắt nhau trừ ở đỉnh. Cách vẽ như vậy gọi là biểu diễn phẳng của đồ thị.

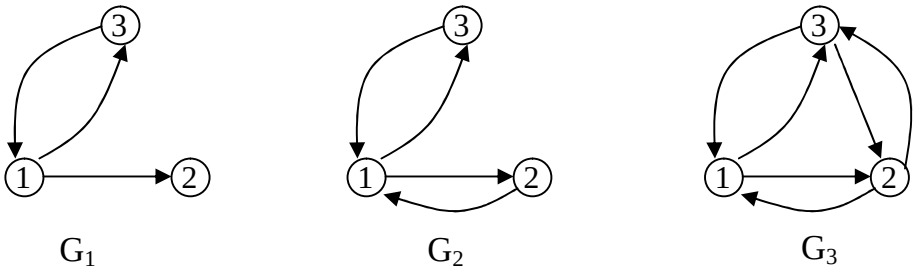
Ví dụ 1.20: Đồ thị phẳng được vẽ lại với các cạnh không cắt nhau.



Hình 1.18: Đồ thị phẳng

Đồ thị đối xứng: Cho G=(V, E) là đồ thị có hướng, G được gọi là đồ thị đối xứng nếu (u,v) $\in$ E $\Rightarrow$ (v,u) $\in$ E. G được gọi là *đối xứng đủ* nếu giữa 2 đỉnh bất kỳ đều có 2 cung ngược chiều nhau.

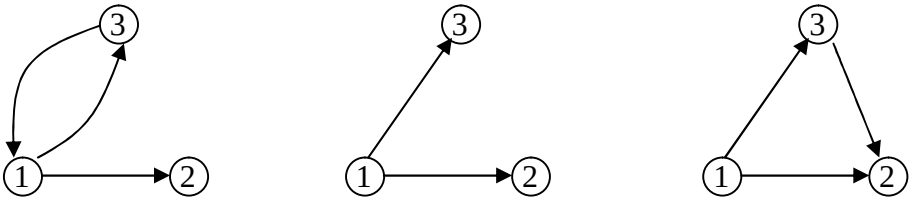
Ví dụ 1.2: Trong hình bên dưới, đồ thị G_1 không đối xứng, G_2 đối xứng, G_3 đối xứng đủ



Hình 1.19: Đồ thị đối xứng

Đồ thị phản xứng: Cho G=(V,E) là đồ thị có hướng, G được gọi là đồ thị phản xứng nếu (u,v) $\in$ E $\Rightarrow$ (v,u) $\notin$ E. G được gọi là phản xứng đủ nếu G đơn và giữa 2 đỉnh bất kỳ có đúng một cung, ký hiệu là T_n

Ví dụ 1.22: G_1 không phản xứng, G_2 phản xứng, G_3 phản xứng đủ

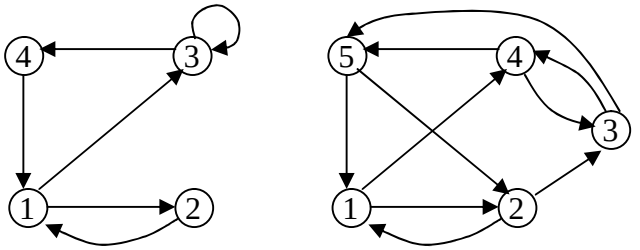


G_1 G_2 G_3

Hình 1.20: Đồ thị phản xứng

Đồ thị căn bằng: Đồ thị có hướng G được gọi là căn bằng (hay giả đối xứng) nếu $d^+(v)=d^-(v)$ $\forall v \in V$. Nếu G đơn và $d^+(v)=d^-(v)=k$ $\forall v \in V$ thì G được gọi là k-đều.

Ví dụ 1.23: Đồ thị G_1 căn bằng, G_2 là 2-đều

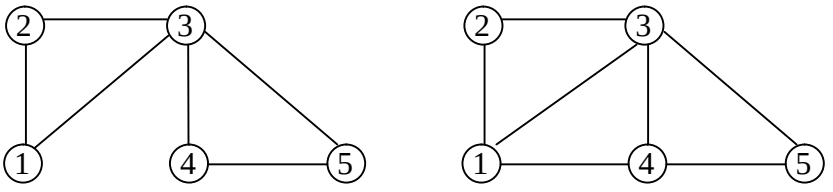


G_1 G_2

Hình 1.21: Đồ thị căn bằng

Đồ thị song liên thông: Đồ thị G được gọi là song liên thông nếu G liên thông và không chứa đỉnh rẽ nhánh.

Ví dụ 1.24: Đồ thị G_1 liên thông nhưng không song liên thông vì có đỉnh rẽ nhánh là 3, đồ thị G_2 là song liên thông.



G_1 G_2

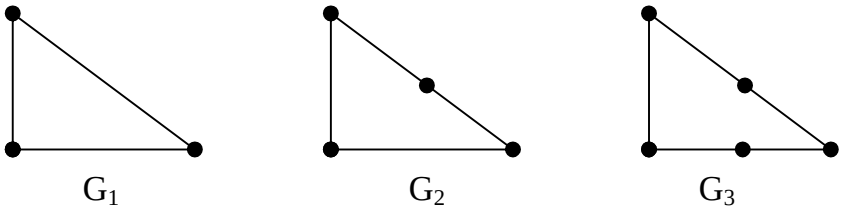
Hình 1.22: Đồ thị song liên thông

1.3. MỘT SỐ PHÉP BIẾN ĐỔI TRÊN ĐỒ THỊ

1.3.1. Phép phân chia sơ cấp

Phép phân chia sơ cấp là việc chia cạnh (u,v) của một đồ thị G bằng cách loại bỏ cạnh này khỏi G và thêm vào một đỉnh mới w cùng với hai cạnh (u,w) và (w,v). Hai đồ thị G_1 và G_2 được gọi là đồng cấu (còn gọi là đồng phôi) nếu chúng có được bằng cách thực hiện một dãy các phép phân chia sơ cấp trên một đồ thị nào đó.

Ví dụ 1.25:

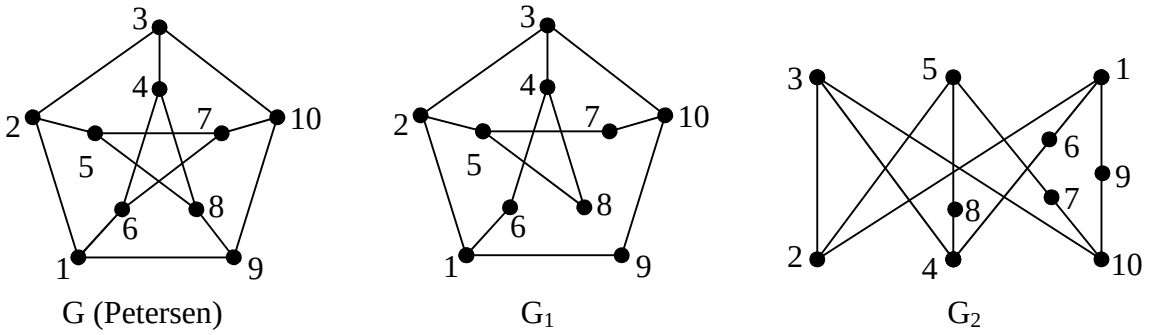


Hình 1.23: Dãy các phép phân chia sơ cấp trên đồ thì G_1

G_2 và G_3 là đồng cấu vì có được bằng cách chia cạch từ G_1

Định lý Kuratowski: Đồ thị G là phẳng ⇔ G không chứa đồ thị con đồng cấu với $K_{3,3}$ hoặc K_5.

Ví dụ 1.26: Theo định lý Kuratowski, đồ thì G dưới đây không phẳng vì có chứa đồ thị con G_1 đồng cấu với $K_{3,3}$. G_2 được vẽ lại từ G_1. Ta nhận thấy rằng khi thực hiện các phép biến đổi sơ cấp trên G_2 bằng cách bỏ đi các đỉnh 6,7,8,9 thì G_2 sẽ đồng cấu với $K_{3,3}$



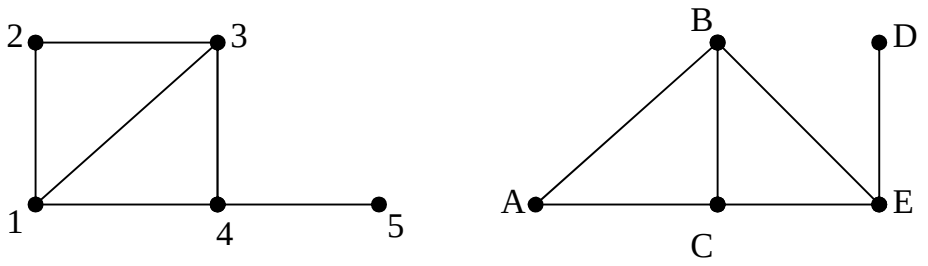
Hình 1.24: Đồ thi đồng cấu

1.3.2. Phép đẳng cấu đồ thị

Hai đồ thị $G_1=(V_1,E_1)$ và $G_2=(V_2,E_2)$ được gọi là đẳng cấu nếu tồn tại một song ánh:

f: $V_1 \rightarrow V_2$ sao cho $(u,v) \in E_1 \Leftrightarrow (f(u),f(v)) \in E_2$

Ví dụ 1.27: Kiểm tra hai đồ thị đẳng cấu:



Hình 1.25: Hai đồ thị đẳng cấu

Hai đồ thị trên đẳng cấu vì tồn tại một song ánh f với f(1)=C, f(2)=A, f(3)=B, f(4)=E, f(5)=D

1.3.3. Phép toán trên đồ thị

Cho hai đồ thị $G_1=(V_1,E_1)$, $G_2=(V_2,E_2)$, ta định nghĩa một số phép toán sau:

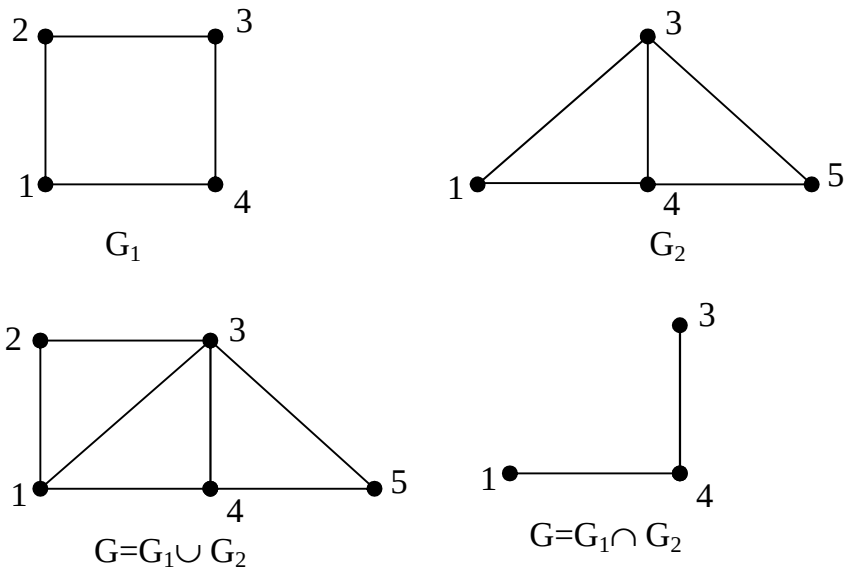
1.3.3.1. Phép hội

Phép hội giữa hai đồ thị $G_1(V_1,E_1)$ và $G_2(V_2,E_2)$ có kết quả là đồ thị G=(V,E) với $V=V_1\cup V_2$ và $E=E_1\cup E_2$, ký hiệu là $G=G_1\cup G_2$

1.3.3.2. Phép giao

Phép giao giữa hai đồ thị $G_1(V_1,E_1)$ và $G_2(V_2,E_2)$ có kết quả là đồ thị G=(V,E) với $V=V_1\cap V_2$ và $E=E_1\cap E_2$, ký hiệu là $G=G_1\cap G_2$

Ví dụ 1.28:



Hình 1.26: Phép hội và giao giữa 2 đồ thị

1.4. SẮC 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 có màu khác nhau. Sắc số của một đồ thị là số màu tối thiểu cần thiết để tô màu đồ thị này.

Một trong những ứng dụng dễ nhận thấy nhất của sắc số đồ thị là tô màu cho các vùng trên bản đồ.

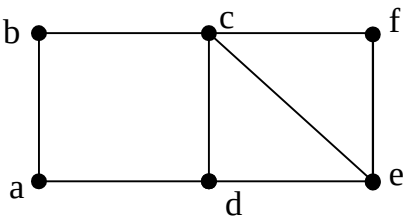
Thuật toán tô màu Welch-Powell:

Bước 1: Sắp xếp các đỉnh của đồ thị theo thứ tự bậc giảm dần.

Bước 2: Chọn đỉnh v chưa tô trên danh sách đỉnh đã sắp xếp theo thứ tự bậc từ lớn đến nhỏ. Chọn một màu để tô cho đỉnh v và các đỉnh không kề với v (lưu ý: khi tô màu cho các đỉnh không kề với v cần kiểm tra xem các đỉnh trong tập này có kề với nhau hay không, nếu xuất hiện hai đỉnh kề nhau thì chọn 1 đỉnh để tô màu).

Bước 3: Lặp lại bước 2 cho đến khi tất cả các đỉnh đều được tô và dừng thuật toán.

Ví dụ 1.29: Tìm sắc số của đồ thị sau:



Hình 1.27: Đồ thị vô hướng G

Sắp xếp các đỉnh theo thứ tự bậc giảm dần và tô màu

Đỉnh	c	d	e	a	b	f
Bậc	4	3	3	2	2	2
Màu	1	2	3	1	2	2

1.5. KẾT CHƯƠNG

Chương này trình bày tổng quan về đồ thị, các khái niệm cơ bản, thuật ngữ liên quan đến đồ thị như: đồ thị vô hướng, có hướng, liên thông, chu trình, đường đi, sắc số và một số dạng đồ thị đặc biệt. Hiểu được các khái niệm này sẽ làm nền tảng cho việc tìm hiểu, vận dụng các kiến thức, các thuật toán về lý thuyết đồ thị trong các chương tiếp theo.

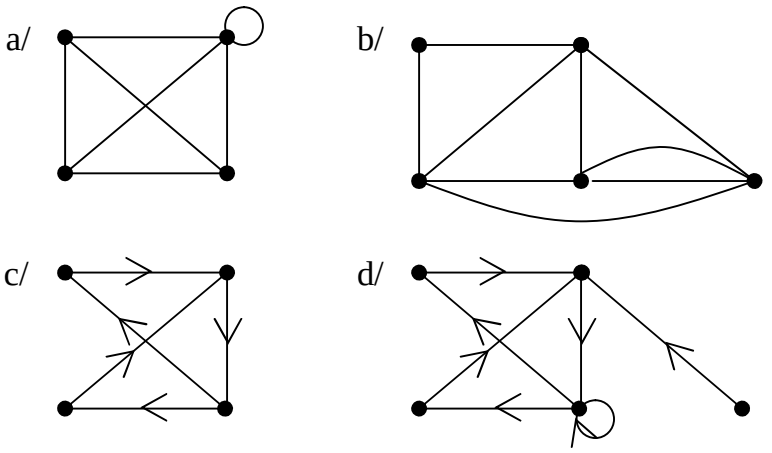
1.6. BÀI TẬP

1) Để thiết kế một hệ thống các tuyến xe buýt lưu thông giữa các thành phố A,B,C,D,E người ta mô tả như sau: Giữa hai thành phố A và B có 2 tuyến, A và D có một tuyến, A và C có một tuyến, C và D có một tuyến, B và C có 2 tuyến, C và E có một tuyến, B và E có một tuyến, riêng thành phố E có một tuyến nội bộ.

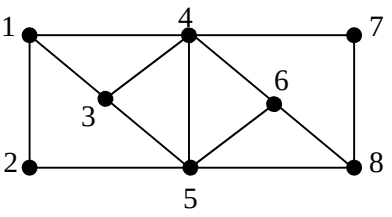
a/ Hãy mô hình hóa bài toán trên thành một đồ thị vô hướng, mỗi tuyến xe giữa 2 thành phố được xem như một cạnh vô hướng của đồ thị.

b/ Hãy cho biết số đỉnh, số cạnh, tổng số bậc của đồ thị trên và cho biết đồ thị trên là đồ thị đơn hay đa đồ thị.

2) Hãy cho biết các đồ thị sau thuộc loại đồ thị nào (đơn đồ thị, đa đồ thị, đồ thị liên thông, song liên thông, liên thông mạnh, liên thông yếu)



3) Cho đồ thị như sau:



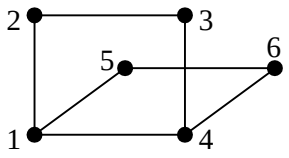
a/ Cấp của đồ thị là bao nhiêu ?. Xác định bậc của từng đỉnh trên đồ thị

a/ Hãy tìm chu trình đơn đi qua 4 đỉnh, 5 đỉnh, 6 đỉnh, 7 đỉnh

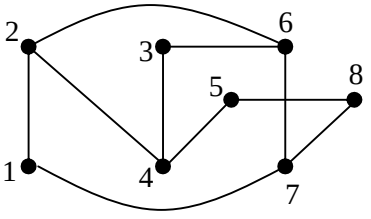
b/ Hãy tìm chu trình sơ cấp đi qua 4 đỉnh, 5 đỉnh, 6 đỉnh, 7 đỉnh

c/ Tìm các đối chu trình xác định bởi A={1,3}, B={4,5}, C={1,3,6}. Cho biết đối chu trình nào là sơ cấp, không sơ cấp, giải thích.

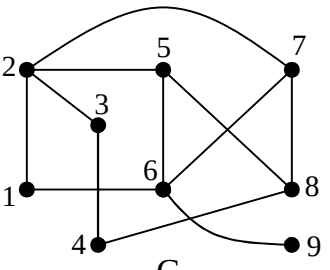
4) Trong các đồ thị sau, đồ thị nào là 2 phía. Nếu là đồ thị hai phía thì vẽ lại đồ thị ở dạng hai phía.



G_1



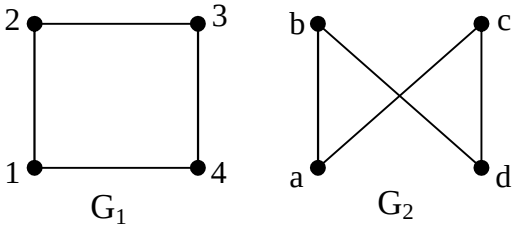
G_2



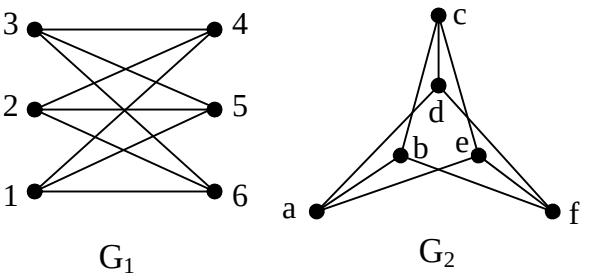
G_3

5) Kiểm tra các cặp đồ thị sau có đẳng cấu không. Nếu có hãy chỉ ra một song ánh f giữa G_1 và G_2.

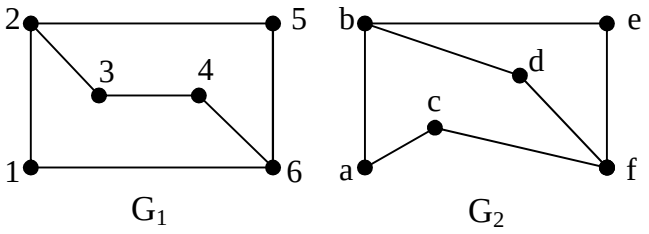
a/



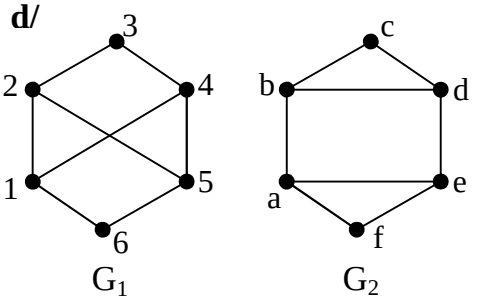
b/



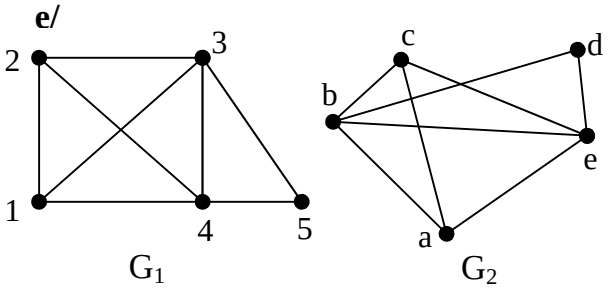
c/



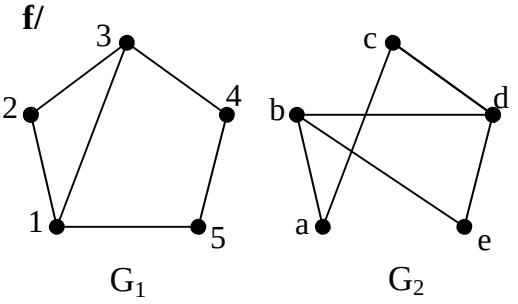
d/



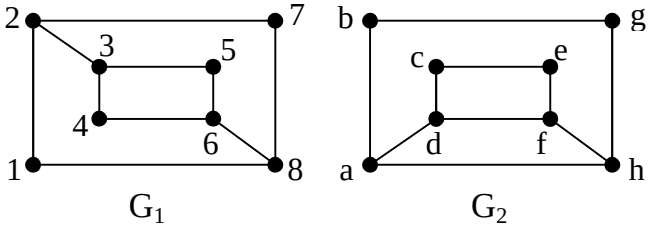
e/



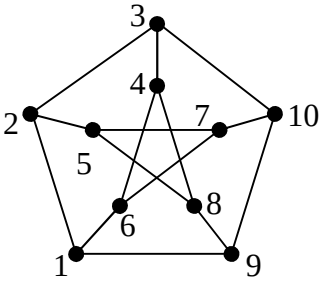
f/



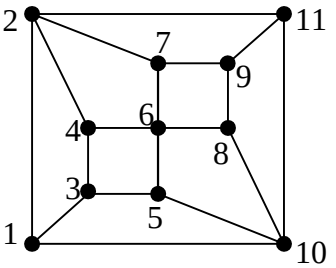
g/



6) Tìm sắc số của đồ thị sau:

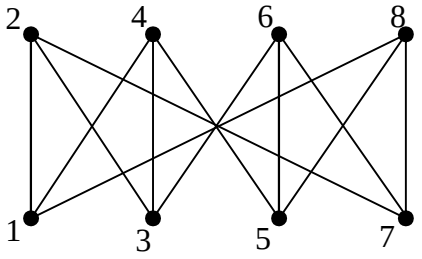


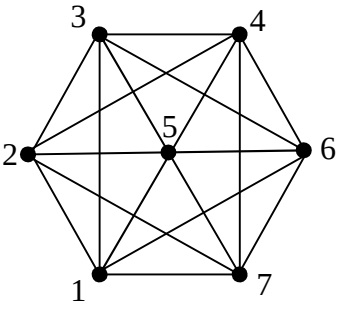
Petersen

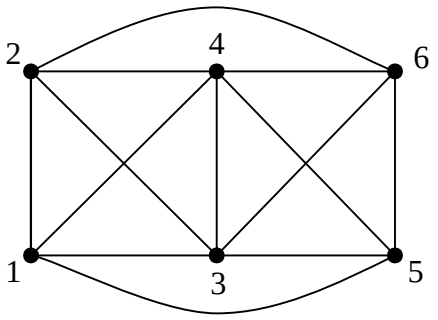


Herschel

7) Áp dụng định lý Kuratowski kiểm tra các đồ thị sau có phẳng không ?







CHƯƠNG 2

BIỂU DIỄN ĐỒ THỊ

Mục tiêu:

- Biểu diễn được đồ thị theo các cấu trúc như: ma trận kề, ma trận trọng số, ma trận liên kết, danh sách cạnh, danh sách kề.
- Phân tích được ưu nhược điểm của từng phương pháp biểu diễn, thực hiện chọn lựa cấu trúc biểu diễn thích hợp cho một đồ thị cụ thể.

2.1. MA TRẬN KỀ

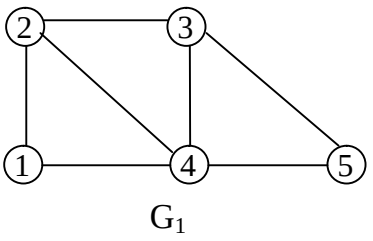
2.1.1. Định nghĩa

Cho đồ thi G=(V,E), với V={1,2,3,...n}. Ma trận kề A của G là ma trận vuông cấp n xác định bởi: A[i,j]=d. Với d là số cạnh/cung đi từ đỉnh i đến đỉnh j.

Nếu G là đồ thị vô hướng (hay có hướng đối xứng) thì A là ma trận đối xứng qua đường chéo chính.

Nếu G là đồ thị đơn thì A[i,j]=1 nếu có cạnh/cung nối từ i đến j, ngược lại A[i,j]=0.

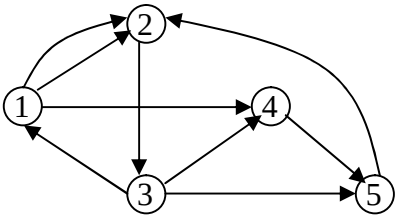
Ví dụ 2.1: Biểu diễn các đồ thị G_1 và G_2 bằng ma trận kề.



G_1

A=

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



G_2

A=

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

Hình 2.1: Biểu diễn đồ thị bằng ma trận kề

2.1.2. Bậc và đỉnh kề dựa trên ma trận kề

Xét đồ thì vô hướng:

Bậc của đỉnh k trong đồ thị vô hướng được xác định bởi tổng các phần tử của dòng k hoặc cột k trong ma trận kề. Sử dụng ma trận kề giúp xác định nhanh đỉnh kề của một đỉnh k nào đó trong đồ thị G hay xác định k là kề của những đỉnh nào. Trong máy tính ma trận kề được lưu trữ bằng cấu trúc mảng hai chiều A[i,j] với i và j là chỉ số dòng và cột tương ứng trong mảng. Nếu A[i,j]>=1 thì j là đỉnh kề của đỉnh i (i cũng là đỉnh kề của j). Như vậy để xác định các đỉnh kề của đỉnh k ta duyệt qua các đỉnh trong dòng k như sau:

```
for j=1 to n do
begin
    if(A[k,j]≥ 1) then
        j là đỉnh kề của đỉnh k
end;
```

Ngược lại, để xác định k là kề của những đỉnh nào ta duyệt qua các đỉnh trong cột k:

```
for i=1 to n do
begin
    if(A[i,k]≥ 1) then
        k là đỉnh kề của đỉnh i
end;
```

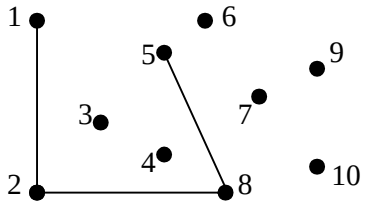
Xét đồ thị có hướng:

Bán bậc ra của đỉnh k là tổng các phần tử trên dòng k của ma trận kề, bán bậc vào của k là tổng các phần tử trên cột k của ma trận kề.

Việc xác định đỉnh kề của đỉnh k và k là kề của những đỉnh nào thực hiện tương tự như đồ thị vô hướng.

Trong trường hợp đồ thị số đỉnh lớn hơn nhiều so với số cạnh thì việc sử dụng ma trận kề trở nên kém hiệu quả về mặt bộ nhớ lưu trữ trên máy tính.

Ví dụ 2.2: Biểu diễn đồ thị sau bằng ma trận kề:



G

A=

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	1	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	1	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

Hình 2.2: Ma trận kề của đồ thị với số đỉnh lớn hơn nhiều so với số cạnh

Khi xác định đỉnh kề ta chỉ quan tâm đến những vị trí có giá trị >=1 trong ma trận kề. Trong ví dụ trên, ma trận có quá nhiều vị trí có giá trị 0 nhưng máy tính vẫn phải lưu trữ, điều này dẫn đến lãng phí bộ nhớ. Do đó, với đồ thị như trên thì việc biểu diễn bằng ma trận kề là không phù hợp.

2.2. MA TRẬN TRỌNG SỐ

2.2.1. Định nghĩa

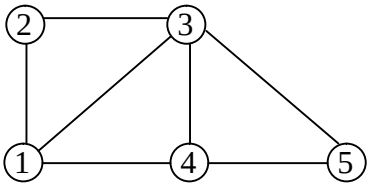
Cho đơn đồ thị G=(V,E) với V={1,2,3,...n}, mỗi cạnh của G được gán một giá trị c[i,j]∈ Z gọi là trọng số của cạnh (i,j). Ma trận trọng số A của G là ma trận vuông cấp n được xác định bởi:

$$A[i,j]=\begin{cases} c[i,j] & \text{nếu } (i,j) \in E \\ \theta \quad -2 & \text{nếu } (i,j) \notin E \end{cases}$$

Trong đó θ có thể là một trong các giá trị: 0, ∝, ∘, -∞

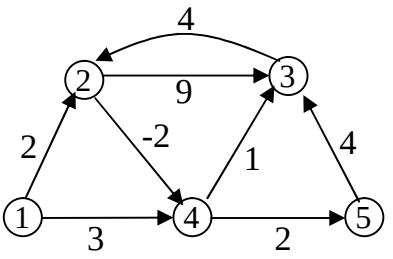
Nếu G là đồ thị vô hướng thì ma trận trọng số của G đối xứng qua đường chéo chính.

Ví dụ 2.3: Biểu diễn các đồ thị G_1 và G_2 bằng ma trận trọng số:



G_1

	1	2	3	4	5
1	0	3	5	1	∞
2	3	0	−2	∞	∞
3	5	−2	0	−3	7
4	1	∞	−3	0	9
5	∞	∞	7	9	0



G_2

A=

	1	2	3	4	5
1	0	2	∞	3	∞
2	∞	0	9	−2	∞
3	∞	4	0	∞	∞
4	∞	∞	1	0	2
5	∞	∞	4	∞	0

Hình 2.3: Ma trận trọng số của đồ thị

2.2.2. Bậc và đỉnh kề dựa trên ma trận trọng số

Xét đồ thị vô hướng:

Bậc của đỉnh k được xác định bằng cách đếm các phần tử có giá trị khác {0,∞} trên cột k hoặc dòng k của ma trận trọng số. Nếu A[i,j] ≠ {0,∞} thì j là đỉnh kề của đỉnh i (và i cũng kề j).

Xét đồ thị có hướng:

Bán bậc ra của đỉnh k được tính bằng cách đếm các phần tử có giá trị khác {0,∞} trên dòng k, bán bậc vào tính bằng cách đếm các phần tử có giá trị khác {0,∞} trên cột k.

Việc xác định đỉnh kề của đỉnh k được thực hiện tương tự như ma trận kề.

Ma trận trọng số thường được dùng để biểu diễn đồ thị cho các bài toán về đường đi ngắn nhất, cây khung ngắn nhất, do đó ma trận trọng số còn được gọi là ma trận khoảng cách.Vấn đề lưu trữ trên máy tính thì ma trận trọng số cũng không hiệu quả nếu như đồ thị có số đỉnh lớn hơn nhiều so với số cạnh.

2.3. MA TRẬN LIÊN KẾT

2.3.1. Định nghĩa

Cho đơn đồ thị G=(V,E) có n đỉnh và m cạnh. Ma trận liên kết A của G là ma trận có n dòng, m cột được định nghĩa như sau:

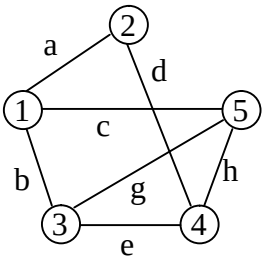
Nếu G vô hướng:

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

Nếu G có hướng:

$$A[i,j]=\begin{cases} 1 & \text{nếu cạnh } e_j \text{ rời khỏi đỉnh i} \\ -1 & \text{nếu cạnh } e_j \text{ đi đến đỉnh i} \\ 0 & \text{nếu cạnh } e_i \text{ không kề với đỉnh} \end{cases}$$

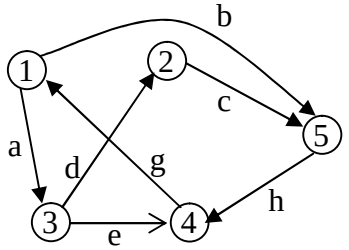
Ví dụ 2.4: Biểu diễn đồ thị G_1 và G_2 bằng ma trận liên kết.



G_1

A=

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



G_2

A=

	a	*b*	*c*	*d*	*e*	*g*	*h*
1	1	1	0	0	0	−1	0
2	0	0	1	−1	0	0	0
3	−1	0	0	1	1	0	0
4	0	0	0	0	−1	1	−1
5	0	−1	−1	0	0	0	1

Hình 2.4: Ma trận liên kết của đồ thị G_1 và G_2

2.3.2. Bậc và đỉnh kề dựa trên ma trận liên kết

Xét đồ thị vô hướng:

Ma trận liên kết được biểu diễn trên máy tính bởi mảng 2 chiều A[i,j]. với i đại diện cho đỉnh và j đại diện cho cạnh của đồ thị. Bậc của đỉnh k được tính bằng tổng giá trị các phần tử trên dòng k. Việc xác định đỉnh kề của một đỉnh phức tạp hơn ma trận kề và được thực hiện bằng cách duyệt qua các cột trên dòng k, nếu cột nào có giá trị 1 thì lần lượt duyệt trên cột đó, nếu tại vị trí i ≠ k và A[i,j]=1 thì i là đỉnh kề của đỉnh k.

```
for j=1 to m do
begin
      if(A[k,j]=1) then
      begin
          for i=1 to n do
            begin
                  IF(i!=k và A[i,j]=1)
                        i là đỉnh kề của đỉnh k
            end;
       end;
end;
```

Ví dụ 2.5: Xác định đỉnh kề của đỉnh k=3 trong đồ thị vô hướng G_1 ở hình 2.4

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

Hình 2.5: Xác định đỉnh kề của đỉnh 3

Đầu tiên ta lần lượt duyệt qua hết các cột tại vị trí dòng k=3, cột b, e, g có giá trị 1 nên duyệt trên các cột này. Khi duyệt trên cột b, tại vị trí i=1 có giá trị 1 và i≠k nên 1 là đỉnh kề của đỉnh 3, thực hiện tương tự cho các cột e, g ta tìm được 4 và 5 cũng là đỉnh kề của đỉnh 3.

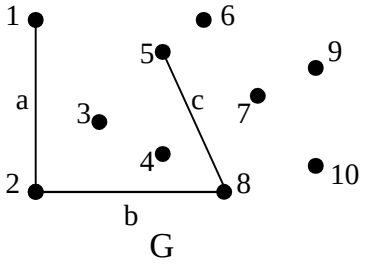
Xét đồ thị có hướng:

Xét đỉnh k bất kỳ, bán bậc ra của k là số các phần tử có giá trị bằng 1 trên dòng k, bán bậc vào là số các phần tử có giá trị bằng -1 trên dòng k.

Việc xác định đỉnh kề cũng tương tự như đồ thị vô hướng nhưng cần phân biệt đỉnh đầu và đỉnh cuối của cung (bài tập).

Ma trận liên kết sử dụng mảng 2 chiều có n dòng, m cột để lưu trữ đồ thị nên sẽ có n×m phần tử. Do đó, nếu đồ thị có số cạnh nhiều hơn số đỉnh thì ma trận liên kết tỏ ra kém hiệu quả hơn ma trận kề, ngược lại nếu đồ thị có số cạnh ít hơn số đỉnh thì sử dụng ma trận liên kết sẽ ít tốn bộ nhớ hơn. Thật vậy, chúng ta hãy biểu diễn lại đồ thị trong ví dụ 2.2 bằng ma trận liên kết như sau:

Ví dụ 2.6



G

A=

	a	*b*	*c*
1	1	0	0
2	1	1	0
3	0	0	0
4	0	0	0
5	0	0	1
6	0	0	0
7	0	0	0
8	0	1	1
9	0	0	0
10	0	0	0

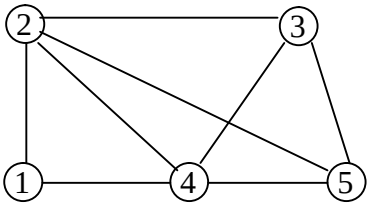
Hình 2.6: Ma trận liên kết

2.4. DANH SÁCH CẠNH

2.4.1. Định nghĩa

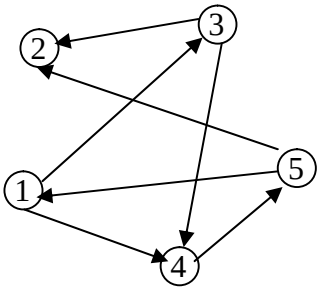
Cho đồ thị G=(V,E) có m cạnh, mỗi cạnh e∈E được lưu trữ trong hai danh sách: một danh sách lưu trữ đỉnh đầu và danh sách còn lại lưu trữ đỉnh cuối. Do đó để biểu diễn tất cả các cạnh trong G, ta dùng 2 mảng một chiều có kích thước là m, mảng thứ nhất sẽ lưu danh sách các đỉnh đầu và mảng còn lại lưu danh sách các đỉnh cuối.

Ví dụ 2.7: Biểu diễn đồ thị G sau bằng danh sách cạnh:



G_1

Đầu	Cuối
1	2
1	4
2	3
2	4
2	5
3	4
3	5
4	5



G_2

Đầu	Cuối
1	3
1	4
3	2
3	4
4	5
5	1
5	2

Hình 2.7: Danh sách cạnh của đồ thị

2.4.2. Bậc và đỉnh kề dựa trên danh sách cạnh

Đề xác định bậc và đỉnh kề của đỉnh k bất kỳ của G ta lần lượt duyệt qua hai danh sách Đầu và Cuối.

Đối với đồ thị vô hướng:

- Đếm số lần xuất hiện của đỉnh k trong hai danh sách là tổng bậc của đỉnh k.
- Nếu Đầu[i]=k thì Cuối[i] là đỉnh kề của k, ngược lại nếu Cuối[i]=k thì Đầu[i] là đỉnh kề của k.

Đối với đồ thị có hướng:

- Bán bậc ra của k là tổng số lần k xuất hiện trong danh sách Đầu, Bán bậc vào của k là tổng số lần k xuất hiện trong mảng Cuối.

- Nếu Đầu[i]=k thì Cuối[i] là đỉnh kề của k.

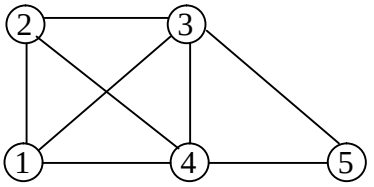
Để biểu diễn đồ thị G có m cạnh thì danh sách cạnh sử dụng 2 mảng một chiều để lưu trữ các đỉnh đầu và cuối của cạnh/cung nên có tất cả là 2*m phần tử cho cả 2 mảng. Hơn nữa, khi xác định đỉnh kề hay tổng bậc của đỉnh ta đều phải duyệt qua hết tất cả các phần tử của cả hai mảng nên phương pháp này chỉ thích hợp cho các đồ thị có số cạnh ít hơn nhiều so với đỉnh.

2.5. DANH SÁCH KỀ

2.5.1. Định nghĩa

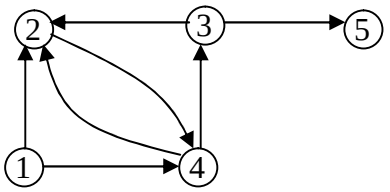
Cho đồ thị G=(V,E), để lưu trữ các đỉnh kề của một đỉnh k∈V ta sử dụng danh sách gọi là danh sách kề. Nếu G có n đỉnh thì cần n danh sách kề để lưu trữ tất cả các đỉnh kề.

Ví dụ 2.8: Biểu diễn các đồ thị sau bằng danh sách kề.



G_1

Danh sách	Đỉnh kề
1	2, 3, 4
2	1, 3, 4
3	1, 2, 4, 5
4	1, 2, 3, 5
5	3, 4

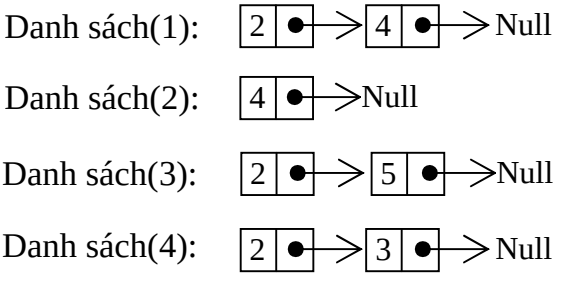


G_2

Danh sách	Đỉnh kề
1	2, 4
2	4
3	2, 5
4	2, 3
5	∅

Hình 2.8: Danh sách kề của đồ thị

Trong máy tính ta sử dụng danh sách liên kết để biểu diễn danh sách kề, các đỉnh kề trong một danh sách liên kết móc nối với nhau một cách tuần tự. Các danh sách kề của đồ thị G_2 được thể hiện bằng danh sách liên kết như sau:



2.5.2. Bậc và đỉnh kề dựa trên danh sách kề

Đối với đồ thị vô hướng: Duyệt danh sách kề của đỉnh k

- Bậc của đỉnh k chính là số phần tử của danh sách kề
- Các đỉnh kề của k là các phần tử trong danh sách kề

Đối với đồ thị có hướng:

- Duyệt danh sách kề tại đỉnh k, số phần tử trong danh sách là bán bậc ra của k. Tính bán bậc vào bằng cách duyệt qua hết tất cả các danh sách, số danh sách kề chứa k là bán bậc vào của k.
- Duyệt danh sách kề tại đỉnh k, các phần tử trong danh sách chính là các đỉnh kề của đỉnh k.

Ví dụ 2.9: Xác định bậc và các đỉnh kề của đỉnh 4 trong đồ thị có hướng G_2 ở hình 2.8:

Duyệt danh sách kề tại đỉnh 4, danh sách này có 2 phần tử là 1 và 2 nên đỉnh 4 có 2 đỉnh kề và bán bậc ra bằng 2. Duyệt qua hết tất cả các danh sách để tìm bán bậc vào, ta thấy rằng có 2 danh sách có chứa 4 là danh sách tại đỉnh 2 và 3 nên bán bậc vào là 2 (số danh sách chứa 4).

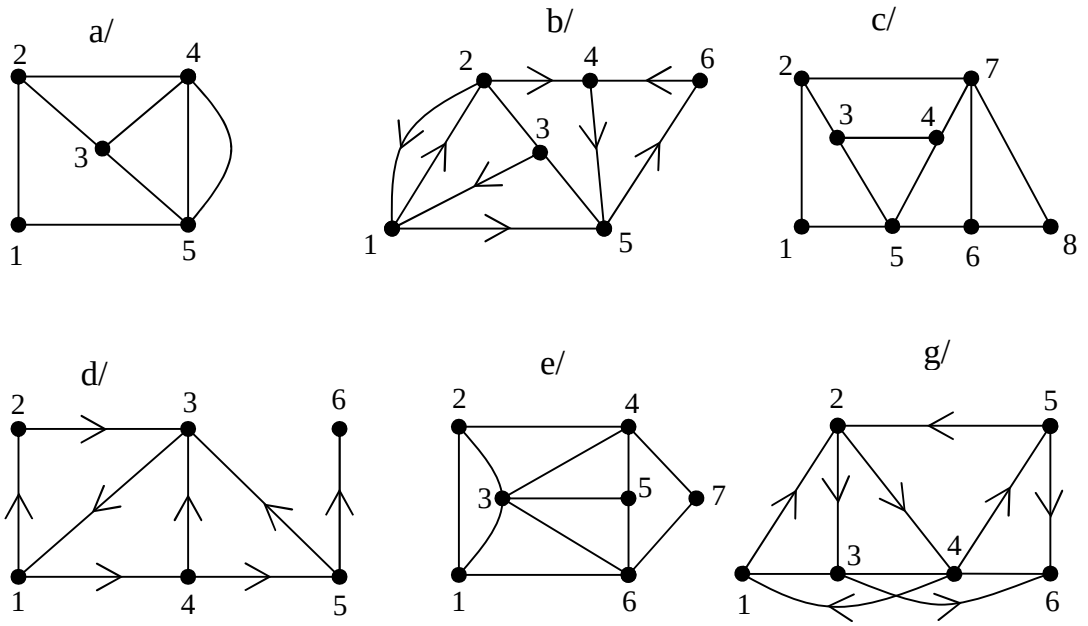
Sử dụng danh sách kề để biểu diễn đồ thị giúp xác định nhanh các đỉnh kề của một đỉnh và bán bậc ra. Nhưng để xác định bán bậc vào hay xác định một đỉnh v là kề của những đỉnh nào thì ta phải duyệt qua hết tất cả các danh sách, điều này sẽ dẫn đến số phép toán thực hiện sẽ tăng lên. Hơn nữa việc tìm kiếm trên danh sách liên kết theo cơ chế tuần tự có nghĩa là trong mỗi danh sách ta phải duyệt tuần tự từ phần tử đầu tiên nên thời gian thực hiện sẽ tăng lên đáng kể.

2.6. KẾT CHƯƠNG

Trong thực tế nhiều bài toán được giải quyết bằng cách mô hình hóa về đồ thị, với những đồ thị có số đỉnh và cạnh tương đối lớn thì không thể thiếu sự trợ giúp của máy tính. Do đó việc chọn lựa phương pháp và cấu trúc dữ liệu phù hợp để biểu diễn đồ thị trên máy tính là điều rất quan trọng. Trong chương này chúng ta đã tìm hiểu 5 phương pháp biểu diễn đồ thị là: Ma trận kề, ma trận trọng số, ma trận liên kết, danh sách cạnh, danh sách kề. Mỗi phương pháp đều có những ưu và nhược điểm riêng. Do đó việc chọn lựa phương pháp nào cho phù hợp còn tùy thuộc vào nhiều yếu tố như: tính chất của đồ thị, bộ nhớ máy tính, thời gian thực hiện...

2.7. BÀI TẬP

1) Biểu diễn các đồ thị sau bằng ma trận kề, ma trận liên kết, danh sách cạnh và danh sách kề:



2) Hãy vẽ các đồ thị được cho bởi các ma trận như sau:

a/

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

b/

	a	*b*	*c*	*d*	*e*	*g*	*h*	*i*
1	1	1	0	0	0	0	0	0
2	−1	0	1	1	−1	1	0	0
3	0	−1	0	0	0	0	0	1
4	0	0	−1	−1	1	0	−1	0
5	0	0	0	0	0	−1	0	−1
6	0	0	0	0	0	0	1	0

3) Dùng ngôn ngữ lập trình C hoặc Pascal, viết chương trình nhập và xuất đồ thị dưới dạng ma trận kề, ma trận liên kết, danh sách cạnh, danh sách kề.

4) Dùng ngôn ngữ lập trình C hoặc Pascal, viết chương trình đọc vào một đồ thị từ file văn bản (.txt) và xuất đồ thị dưới dạng ma trận kề, ma trận liên kết, danh sách cạnh, danh sách kề.

5) Viết chương trình đọc vào một đồ thị, sau đó nhập vào một đỉnh và xuất ra bậc của đỉnh đó. Nếu là đồ thị có hướng thì xuất ra bán bậc vào và bán bậc ra của đỉnh đó.

6) Viết chương trình đọc vào một đồ thị ở dạng ma trận liên kết, nhập vào một đỉnh và xuất ra các đỉnh kề của đỉnh đó.

7) Viết chương trình đọc vào một đồ thị và kiểm tra đồ thị có 2 phía hay không. Nếu là đồ thị 2 phía thì xuất ra các đỉnh trong từng phía của đồ thị.

8) Viết chương trình đọc vào một đồ thị và kiểm tra đồ thị có đối xứng không.

9) Viết chương trình đọc vào một đồ thị và cho biết sắc số của đồ thị đó.

10) Viết chương trình đọc vào một đồ thị ở dạng ma trận kề sau đó chuyển sang dạng X và xuất ra màn hình. Với X là:

a/ Ma trận liên kết

b/ Danh sách cạnh

c/ Danh sách kề

CHƯƠNG 3

CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ

Mục tiêu:

- Hiểu được các thuật toán DFS và BFS
- Thực hiện duyệt đồ thị bằng thuật toán DFS và BFS
- Vận dụng thuật toán DFS và BFS để tìm đường đi và kiểm tra tính liên thông trên đồ thị.

3.1. TÌM KIẾM THEO CHIỀU SÂU

Thuật toán DFS (Depth First Search):

DFS là thuật toán được sử dụng để tìm kiếm hay duyệt qua (thăm) tất cả các đỉnh thuộc các thành phần liên thông của đồ thị một cách có hệ thống với kỹ thuật gọi là ưu tiên độ sâu. Thuật toán có nhiều ứng dụng giải các bài toán về liên thông, cây khung, đường đi...Thuật toán có thể được cài đặt bằng phương pháp đệ qui hay phương pháp lặp sử dụng cấu trúc stack để khử đệ qui. Độ phức tạp của thuật toán là O(n+m) với n là số đỉnh và m là số cạnh của đồ thị.

Ý tưởng:

- Xuất phát từ một đỉnh v cho trước chưa thăm

- Thăm đỉnh v và tìm đỉnh u nào đó chưa thăm kề với v, thăm u và lặp lại quá trình này cho đến khi tất cả các đỉnh đều được thăm.

- Nếu tại một đỉnh u nào đó mà không còn đỉnh kề chưa thăm thì quay trở lại đỉnh trước của u (có đỉnh kề là u trong đường đi đến u) và tìm các đỉnh kề chưa thăm của đỉnh này.

Thuật toán đệ quy:

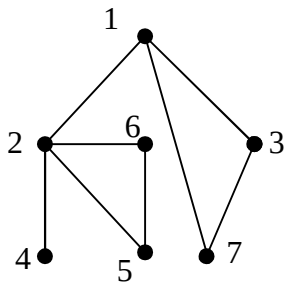
```
procedure DFS(v)
begin
    thăm[v]=1;
    for(u ∈ kề(v)) do
    begin
        if(thăm[u]=0) then
```

```
                DFS(u); //Gọi đệ quy
        end;
end;
```

Thuật toán trên chỉ duyệt qua một thành phần liên thông của đồ thị. Để duyệt tất cả các phần liên thông ta thực hiện như sau:

```
for(v ∈ V) do thăm[v]=0;
for(v ∈ V) do
        if(thăm[v]=0)  then DFS(v);
```

Ví dụ 3.1: Xuất phát từ đỉnh 1, dùng thuật toán DFS duyệt qua tất cả các đỉnh của đồ thị được cho như sau:



Hình 3.1: Đồ thị vô hướng

Xuất phát từ đỉnh 1 chưa thăm, *thăm đỉnh 1,* tìm các đỉnh kề đỉnh 1 là 2,3,7

Chọn đỉnh 2 chưa thăm kề với 1, *thăm đỉnh 2,* tìm các đỉnh kề với 2 là 1,4,5,6

Chọn đỉnh 4 chưa thăm kề với 2, *thăm đỉnh 4,* tìm các đỉnh kề với 4 là 2

Không có đỉnh kề nào của 4 mà chưa thăm nên quay trở lại đỉnh trước của đỉnh 4 là 2 và chọn đỉnh kề chưa thăm khác của đỉnh 2.

Chọn đỉnh 5 chưa thăm kề với 2, *thăm đỉnh 5,* tìm các đỉnh kề với 5 là 2,6

Chọn đỉnh 6 chưa thăm kề với 5, *thăm đỉnh 6,* các đỉnh kề với 6 là 2,5

Không có đỉnh kề nào của 6 mà chưa thăm nên quay trở lại đỉnh trước của đỉnh 6 là 5 và chọn đỉnh kề chưa thăm khác của đỉnh 5.

Không có đỉnh kề nào của 5 mà chưa thăm nên quay trở lại đỉnh trước của đỉnh 5 là 2 và chọn đỉnh kề chưa thăm khác của đỉnh 2.

Không có đỉnh kề nào của 2 mà chưa thăm nên quay trở lại đỉnh trước của đỉnh 2 là 1 và chọn đỉnh kề chưa thăm khác của đỉnh 1.

Chọn đỉnh 3 chưa thăm kề với 1, *thăm đỉnh 3,* tìm các đỉnh kề của 3 là 1,7

Chọn đỉnh 7 chưa thăm kề với 3, *thăm đỉnh 7,* tìm các đỉnh kề của 7 là 1,3

Không có đỉnh kề nào của 7 mà chưa thăm nên quay trở lại đỉnh trước của đỉnh 7 là 3 và chọn đỉnh kề chưa thăm khác của đỉnh 3.

Không có đỉnh kề nào của 3 mà chưa thăm nên quay trở lại đỉnh trước của đỉnh 3 là 1 và chọn đỉnh kề chưa thăm khác của đỉnh 1.

Không có đỉnh kề nào của 1 mà chưa thăm nên thuật toán dừng.

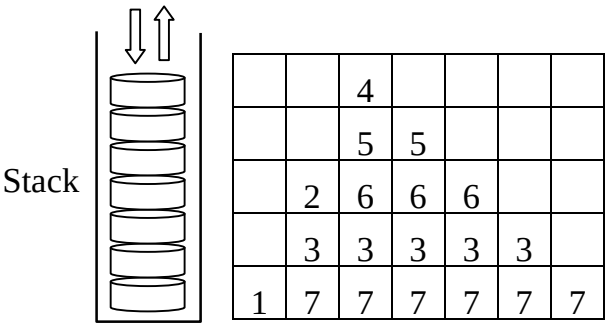
Theo thứ tự thăm của các đỉnh, kết quả duyệt bằng thuật toán DFS là: 1, 2, 4, 5, 6, 3, 7

✍ **Lưu ý:** Trong quá trình thực hiện thuật toán, nếu một đỉnh có nhiều đỉnh kề chưa thăm thì ưu tiên chọn đỉnh kề được đánh số nhỏ hơn.

Ngoài ra DFS còn có thể biểu diễn dưới dạng *thuật toán lặp*, dùng cấu trúc stack để khử đệ quy như sau:

```
procedure DFS (v)
begin
    stack=∅; // Khởi tạo stack
    push(stack,v) //Bỏ v vào stack
    while(stack != ∅)
    begin
        u=pop(stack); //Lấy một đỉnh ra khỏi stack
        begin
            thăm[u]=1; //đánh dấu u đã thăm rồi
            for (k ∈ kề(u)) do
                if(k∉ stack && thăm[k]=0) then
                    push(stack, k)
        end;
    end;
end;
```

Ví dụ 3.2: Dùng thuật toán DFS không đệ quy duyệt qua tất cả các đỉnh của đồ thị hình 3.1 như sau:



Hình 3.2: Duyệt đồ thị bằng thuật toán DFS không đệ quy

Khởi tạo: stack=∅, đỉnh xuất phát là 1, bỏ đỉnh 1 vào stack

Bước lặp: điều kiện lặp stack!= ∅

- Lấy đỉnh 1 ra khỏi stack: pop(stack), đánh dấu đỉnh 1 đã thăm: thăm[1]=1, tìm đỉnh kề chưa thăm của đỉnh 1 mà không có trong stack: 2, 3, 7 và bỏ tất cả các đỉnh kề này vào stack theo thứ tự từ lớn đến nhỏ.

- Lấy đỉnh 2 ra khỏi stack, đánh dấu đỉnh 2 đã thăm: thăm[2]=1, tìm các đỉnh kề chưa thăm của 2 mà không chứa trong stack: 4,5,6 và bỏ tất cả các đỉnh kề này vào stack theo thứ tự từ lớn đến nhỏ.

- Lấy đỉnh 4 ra khỏi stack, đánh dấu đỉnh 4 đã thăm: thăm[4]=1, đỉnh 4 không còn đỉnh kề nào chưa thăm nên không có đỉnh nào bỏ vào stack.

- Lấy đỉnh 5 ra khỏi stack, đánh dấu đỉnh 5 đã thăm: thăm[5]=1, đỉnh 5 có đỉnh kề 6 chưa thăm nhưng 6 đã có trong stack nên cũng không có đỉnh nào bỏ vào stack.

- Lấy đỉnh 6 ra khỏi stack, đánh dấu đỉnh 6 đã thăm: thăm[6]=1, đỉnh này cũng không có đỉnh kề nào chưa thăm.

- Lấy đỉnh 3 ra khỏi stack, đánh dấu đỉnh 3 đã thăm: thăm[3]=1, đỉnh 3 có đỉnh kề 7 chưa thăm nhưng đã có trong stack nên không có đỉnh nào thêm vào stack.

- Lấy đỉnh 7 ra khỏi stack, đánh dấu đỉnh 7 đã thăm: thăm[7]=1, đỉnh này không có đỉnh kề nào chưa thăm.

- stack=∅, thoát khỏi vòng lặp và kết thúc thuật toán. khi đó ta thu được dãy các đỉnh đã thăm qua như sau: 1, 2, 4, 5, 6, 3, 7

3.2. TÌM KIẾM THEO CHIỀU RỘNG

Thuật toán BFS (Breadth First Search):

Thuật toán BFS dùng để duyệt và tìm kiếm trên đồ thị cũng tương tự như thuật toán DFS nhưng thay vì dùng cấu trúc stack thì BFS dùng cấu trúc hàng đợi để thực hiện, kỹ thuật này gọi là tìm kiếm ưu tiên chiều rộng. Thuật toán có độ phức tạp là O(m+n).

Ý tưởng:

- Xuất phát từ đỉnh v cho trước chưa thăm, bỏ v vào hàng đợi.

- Lấy từ hàng đợi đỉnh u, thăm u và bỏ các đỉnh kề chưa thăm của u vào hàng đợi.

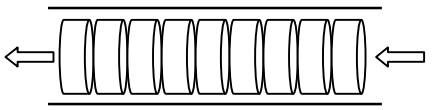
- Lặp lại cho đến khi hàng đợi rỗng.

Thuật toán:

procedure BFS (v)

```
begin
    queue=∅; // Khởi tạo hàng đợi
    push(queue,v) //Bỏ v vào stack
    while(queue != ∅)
    begin
        u=pop(queue);
        thăm[u]=1; //đánh dấu u đã thăm rồi
        for (k ∈ kề(u)) do
            if(k∉ queue && thăm[k]=0) then
                push(queue, k)
    end;
end;
```

Ví dụ: 3.3: Xét đồ thị như trong hình 3.1, dùng thuật toán BFS để duyệt đồ thị:



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

Hình 3.3: Duyệt đồ thị bằng thuật toán BFS

Khởi tạo: queue=∅, đỉnh xuất phát là 1, bỏ đỉnh 1 vào queue

Bước lặp: điều kiện lặp queue!= ∅

- Lấy đỉnh 1 ra khỏi queue: pop(queue), đánh dấu đỉnh 1 đã thăm: thăm[1]=1, tìm đỉnh kề chưa thăm của đỉnh 1 mà không có trong queue: 2, 3, 7 và bỏ tất cả các đỉnh kề này vào queue theo thứ tự từ nhỏ đến lớn.

- Lấy đỉnh 2 ra khỏi queue, đánh dấu đỉnh 2 đã thăm: thăm[2]=1, tìm các đỉnh kề chưa thăm của 2 mà không chứa trong queue: 4,5,6 và bỏ tất cả các đỉnh kề này vào queue theo thứ tự từ nhỏ đến lớn.

- Lấy đỉnh 3 ra khỏi queue, đánh dấu đỉnh 3 đã thăm: thăm[3]=1, đỉnh 3 không còn đỉnh kề nào chưa thăm mà không có trong queue nên không có đỉnh nào bỏ vào queue.

- Lấy đỉnh 7 ra khỏi queue, đánh dấu đỉnh 7 đã thăm: thăm[7]=1, đỉnh 7 không còn đỉnh kề nào chưa thăm.

- Lấy đỉnh 4 ra khỏi queue, đánh dấu đỉnh 4 đã thăm: thăm[4]=1, đỉnh này cũng không có đỉnh kề nào chưa thăm.

- Lấy đỉnh 5 ra khỏi queue, đánh dấu đỉnh 5 đã thăm: thăm[5]=1, đỉnh 5 có đỉnh kề 6 chưa thăm nhưng đã có trong queue nên không có đỉnh nào thêm vào queue.

- Lấy đỉnh 6 ra khỏi queue, đánh dấu đỉnh 6 đã thăm: thăm[6]=1, đỉnh này không có đỉnh kề nào chưa thăm.

- queue=∅, thoát khỏi vòng lặp và kết thúc thuật toán. khi đó ta thu được dãy các đỉnh đã thăm qua như sau: 1, 2, 3, 7, 4, 5, 6

3.3. MỘT SỐ ỨNG DỤNG

3.3.1. Tìm đường đi giữa hai đỉnh

Cho G là một đồ thị với s và t là hai đỉnh bất kỳ của G. Hãy tìm đường đi từ s đến t.

Để giải quyết bài toán này ta áp dụng thuật toán DFS hoặc BFS duyệt qua hết các đỉnh thuộc cùng một thành phần liên thông với đỉnh s. Sau khi kết thúc thuật toán, nếu thăm[t]=0 thì không tồn tại đường đi từ s đến t, ngược lại là có đường đi. Để ghi nhận đường đi từ s đến t ta sử dụng thêm mảng trước[v] để lưu lại đỉnh trước của đỉnh v trong đường đi từ s đến t.

Dùng thuật toán DFS đệ quy:

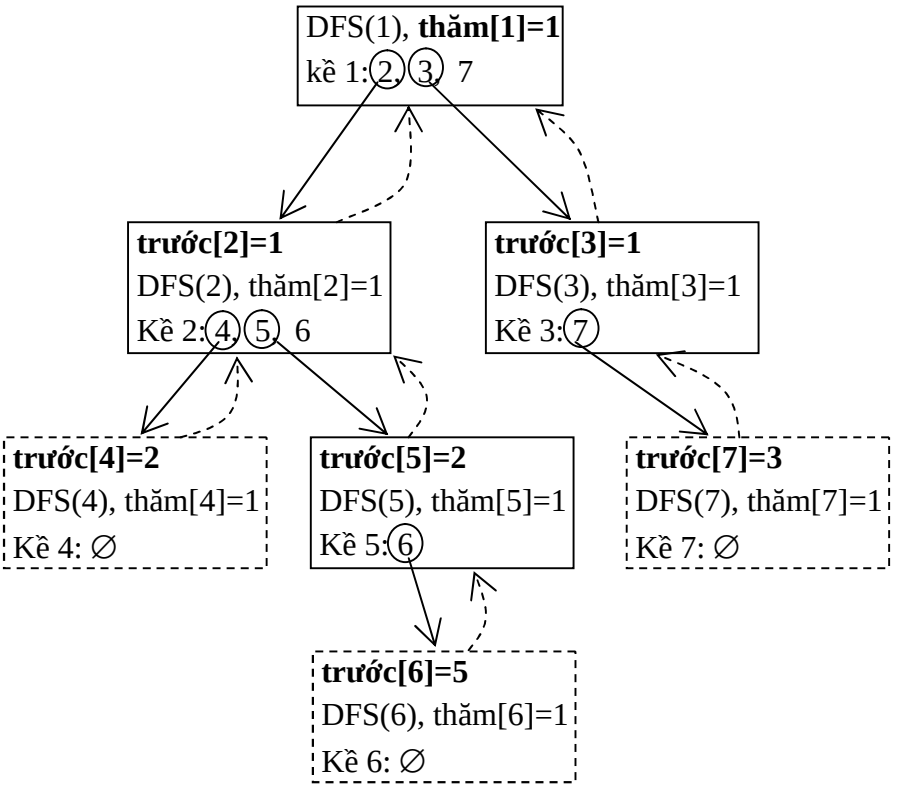
```
procedure DFS(v)
begin
    thăm[v]=1;
    for(u ∈ kề(v)) do
        if(thăm[u]=0) then
        begin
            trước[u]=v; //Lưu lại đỉnh trước của u là v
            DFS(u); //Goi de quy
        end;
end;
```

Dùng thuật toán BFS:

```
procedure BFS (v)
begin
    queue=∅; // Khởi tạo hàng đợi
    push(queue,v) //Bỏ v vào stack
    while(queue != ∅)
```

```
        begin
            u=pop(queue);
            thăm[u]=1; //đánh dấu u đã thăm rồi
            for (k ∈ kề(u)) do
                if(k∉ queue && thăm[k]=0) then
                begin
                    trước[k]=u; //Lưu lại đỉnh trước của k là u
                    push(queue, k);
                end;
        end;
    end;
```

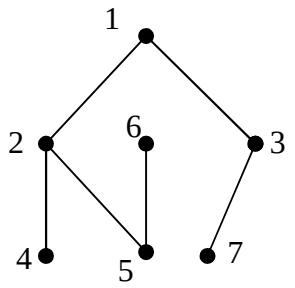
Ví dụ 3.4: Xét đồ thị G như trong hình 3.1, dùng thuật toán DFS duyệt qua đồ thị có lưu lại đường đi với đỉnh xuất phát là s=1. Từng bước của thuật toán được mô tả trong sơ đồ sau:



Hình 3.4: Xác định đường đi từ đỉnh xuất phát đến đỉnh khác

Khi kết thúc thuật toán, tất cả các đỉnh đều được thăm (thăm[v]=1, ∀v∈ V) nên từ đỉnh xuất phát s=1 có đường đi đến tất cả các đỉnh còn lại.

Đường đi từ đỉnh xuất phát s=1 đến các đỉnh khác được xác định dựa vào giá trị của mảng trước[v]: trước[2]=1, trước[3]=1, trước[4]=4, trước[5]=2, trước[6]=5, trước[7]=3. đường đi được thể hiện như sau:



Hình 3.5: Đường đi từ đỉnh 1 đến các đỉnh khác

3.3.2. Tìm các thành phần liên thông của đồ thị

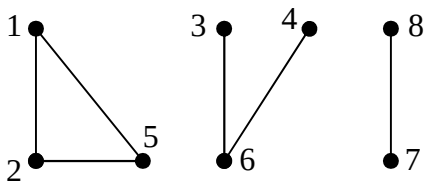
Cho đồ thị G vô hướng, kiểm tra G có liên thông hay không. Nếu G không liên thông thì cho biết đồ thị có bao nhiêu thành phần liên thông, mỗi thành phần liên thông gồm những đỉnh nào?

Để giải bài toán trên ta có thể dùng thuật toán DFS hay BFS, trong đó có dùng biến k để đếm số thành phần liên thông của đồ thị, mỗi lần chương trình gọi thuật toán DFS hay BFS thì giá trị của k được tăng thêm 1 đơn vị:

```
for(v∈ V) do thăm[v]=0;
k=0;
for(v∈ V) do
    if(tham[v]=0) do
    begin
        k=k+1;
        DFS(v)
    end;

//Thuật toán DFS
    procedure DFS(v)
    begin
        thăm[v]=k;
        for(u ∈ kề(v)) do
            if(thăm[u]=0) then
                DFS(u); //Goi de quy
    end;
```

Ví dụ: 3.5: Hãy kiểm tra đồ thị sau có liên thông hay không, nếu không liên thông thì tìm các thành phần liên thông tương ứng:



G

Hình 3.6: Kiểm tra đồ thị liên thông

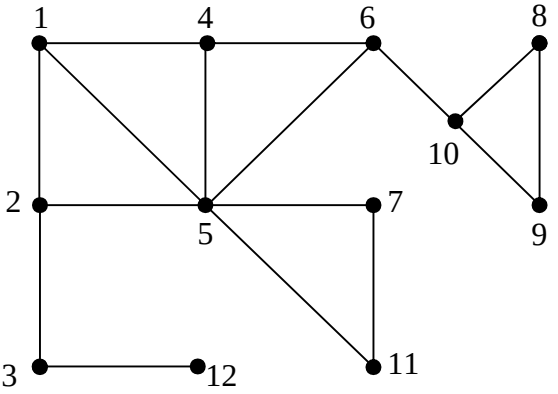
Theo thuật toán trên, ban đầu tất cả các đỉnh đều chưa thăm: thăm[v]=0$\forall v \in V$
Lần thứ nhất chương trình gọi DFS sẽ duyệt qua hết các đỉnh trong thành phần liên thông thứ nhất: 1,2,5
Lần thứ hai chương trình gọi DFS sẽ duyệt qua hết các đỉnh trong thành phần liên thông thứ hai: 3,4,6
Lần thứ ba chương trình gọi DFS sẽ duyệt qua hết các đỉnh trong thành phần liên thông thứ ba: 7,8

3.4. KẾT CHƯƠNG

DFS và BFS là hai thuật toán được biết đến rộng rãi khi thực hiện tìm kiếm trên đồ thị, mỗi thuật toán có một thứ tự duyệt khác nhau. Ngoài ra, các thuật toán này còn có nhiều ứng dụng trong việc giải quyết các bài toán về liên thông, tìm đường đi, tìm cây khung.

3.5. BÀI TẬP

1) Cho đồ thị vô hướng như sau:



Xuất phát từ đỉnh 1, hãy duyệt đồ thị trên bằng thuật toán:
a/ DFS đệ quy
b/ DFS không đệ quy
c/ BFS
d/ Thực hiện câu a,b,c xuất phát từ đỉnh 5.
2) Cài đặt thuật toán DFS và BFS duyệt đồ thị bằng ngôn ngữ C hoặc Pascal.

3) Cài đặt thuật toán kiểm tra có đường đi từ đỉnh s đến đỉnh t trên đồ thị hay không. Nếu có, xuất ra đường đi từ đỉnh s đến đỉnh t.
4) Cài đặt thuật toán kiểm tra đồ thị có liên thông hay không. Nếu không liên thông thì xuất ra số thành phần liên thông và danh sách các đỉnh trong mỗi thành phần liên thông.

CHƯƠNG 4
CÂY

Mục tiêu:

- Hiểu được cây là một khái niệm quan trọng trong lý thuyết đồ thị, cấu trúc dữ liệu và giải thuật.
- Ứng dụng cấu trúc cây trong các giải thuật tìm kiếm, giải thuật sắp xếp, biểu diễn bài toán quyết định (cây quyết định), biểu diễn quá trình tính toán các biểu thức đại số và nhiều bài toán khác.

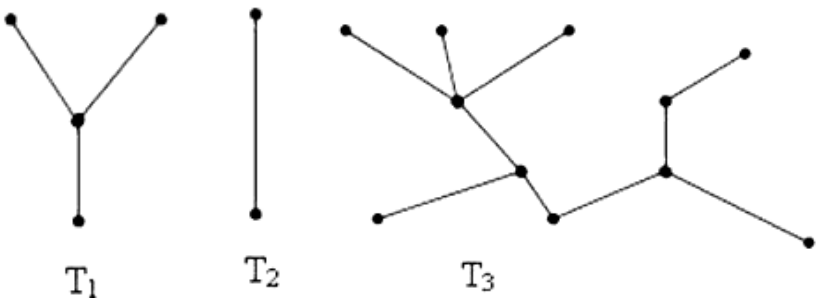
4.1. ĐỊNH NGHĨA

4.1.1. Cây

a) Cho G là đồ thị vô hướng. G được gọi là *một cây* nếu G liên thông và không có chu trình đơn.

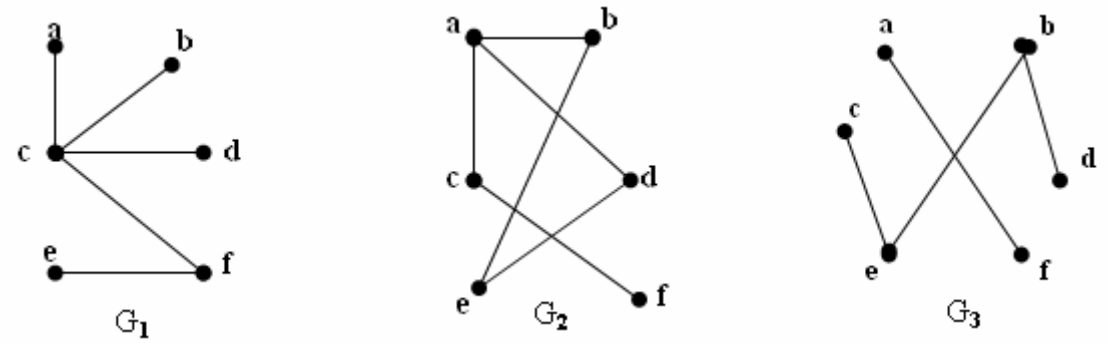
b) *Rừng* là đồ thị mà mỗi thành phần liên thông của nó là một cây.

Ví dụ 4.1:



Hình 4.1: Rừng gồm có 3 cây T_1, T_2, T_3

Ví dụ 4.2:



Hình 4.2: Đồ thị G_1, G_2, G_3

G_1 là cây

G_2 không là cây (do chứa chu trình)

G_3 không là cây (do không liên thông)

Chú ý:

Định nghĩa cây hàm ý nói rằng mọi cây đều không chứa khuyên, cũng không chứa cạnh song song.

Định lý 1: (Điều kiện cần và đủ (cây))

Cho T là đồ thị vô hướng có n đỉnh. Các phát biểu sau đây là tương đương:

(1) T là cây

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

(3) T liên thông và có n-1 cạnh

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

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

(6) T không có chu trình đơn và nếu thêm vào một cạnh giữa hai đỉnh không kề nhau thì có một chu trình đơn duy nhất.

Chứng minh

(1) ⇒ (2): T là cây ⇒ T không chứa chu trình và có n-1 cạnh

- Hiển nhiên T không chứa chu trình (vì T là cây)
- Ta cần chứng minh T có n – 1 cạnh. Xét T_n là cây có n đỉnh. Ta sẽ chứng minh quy nạp theo n
  - Với n = 2, cây có 2 đỉnh thì có 1 cạnh. Đúng
  - Giả sử mọi cây có k đỉnh thì sẽ có k – 1 đỉnh
  - Xét T_{k+1} là cây có k+1 đỉnh. Dễ thấy rằng trong cây T_{k+1} luôn tồn tại ít nhất 1 đỉnh treo

    Loại đỉnh treo này (cùng với cạnh nối) ra khỏi T_{k+1} ta được đồ thị T' có k đỉnh. Dễ thấy T' vẫn liên thông và không có chu trình (do T_{k+1} không có chu trình)

    Suy ra T' là cây. Theo giả thiết quy nạp, T' có k đỉnh thì sẽ có k – 1 cạnh. Vậy cây T_{k+1} có k cạnh. (đpcm)

(2) ⇒ (3): T không chứa chu trình và có n-1 cạnh ⇒ T liên thông và có n-1 cạnh

  - Theo giả thiết T có n – 1 cạnh

- Ta chỉ cần chứng minh T liên thông
- Giả sử T có k thành phần liên thông với số đỉnh lần lượt là n_1, ..., n_k
- Khi đó mỗi thành phần liên thông của T sẽ là một cây và sẽ có số cạnh lần lượt là $n_1 – 1$, $n_2 – 2$, ..., $n_k – 1$
- Suy ra, số cạnh của T sẽ là $n_1 – 1 + n_2 – 2 + ... + n_k – 1 = n – k$
- Theo giả thiết, số cạnh của cây là n – 1. Từ đó, suy ra k = 1 hay T chỉ có một thành phần liên thông. Suy ra T liên thông (đpcm)

(3) ⇒ (4): T liên thông và có n-1 cạnh ⇒ T liên thông và mỗi cạnh của nó đều là cầu

- Theo giả thiết T liên thông
- Ta chỉ cần chứng minh mỗi cạnh của T đều là cầu

  Xét (u, v) là cạnh bất kỳ của T. Nếu bỏ (u, v) ra khỏi T, ta sẽ được đồ thị T' có n đỉnh và n – 2 cạnh

  Ta đã chứng minh được đồ thị có n đỉnh và n – 2 cạnh thì không thể liên thông

  Vậy nếu bỏ cạnh (u, v) ra thì sẽ làm mất tính liên thông của đồ thị. Suy ra (u, v) là cầu. (đpcm)

(4) ⇒ (5): T liên thông và mỗi cạnh của nó đều là cầu ⇒ Hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi đơn

- Xét u, v là hai đỉnh bất kỳ trong T.
- Do T liên thông nên luôn tồn tại đường đi giữa u và v. Ta sẽ chứng minh đường đi này là duy nhất.

  Giả sử có hai đường đi đơn khác nhau giữa u và v. Khi đó hai đường đi này sẽ tạo thành một chu trình.

  Suy ra,các cạnh trên chu trình này sẽ không thể là cầu được – Mâu thuẫn.

  Vậy giữa u và v chỉ có thể tồn tại đúng 1 đường đi đơn. (đpcm)

(5) ⇒ (6): Hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi đơn ⇒ T không có chu trình đơn và nếu thêm vào một cạnh giữa hai đỉnh không kề nhau thì có một chu trình đơn duy nhất.

- T không thể có chu trình, vì nếu có chu trình thì giữa hai đỉnh trên chu trình này sẽ có 2 đường đi đơn khác nhau – mâu thuẫn với GT.
- Giả sử ta thêm vào T cạnh (u,v) bất kỳ (trước đó không có cạnh này trong T).

- Khi đó cạnh này sẽ tạo với đường đi duy nhất giữa u và v trong T tạo thành 1 chu trình duy nhất. (Vì nếu tạo thành 2 chu trình thì chứng tỏ trước đó có 2 đường đi khác nhau giữa u và v – mâu thuẫn với giả thiết)

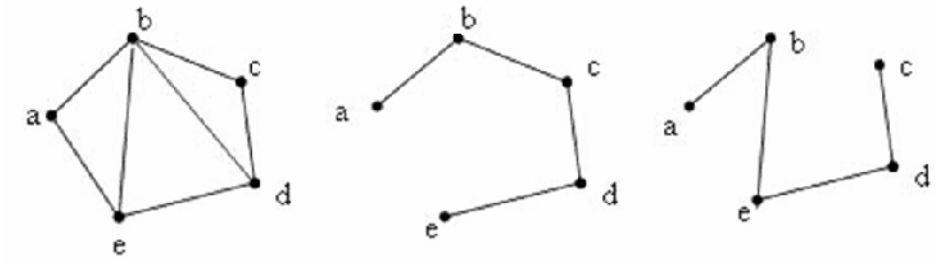
(6) ⇒ (1): T không có chu trình đơn và nếu thêm vào một cạnh giữa hai đỉnh không kề nhau thì có một chu trình đơn duy nhất ⇒ T là cây

- Hiển nhiên T không chứa chu trình (theo giả thiết).
- Giả sử T không liên thông. Khi đó T sẽ có nhiều hơn 1 thành phần liên thông
- Suy ra, nếu thêm vào một cạnh bất kỳ giữa hai đỉnh thuộc 2 thành phần liên thông khác nhau sẽ không tạo thêm chu trình nào – mâu thuẫn với giả thiết.
- Vậy, T phải liên thông. Suy ra T là cây. (đpcm)

4.1.2. Định nghĩa cây tối đại

Định nghĩa: Giả sử G = (V, E) là đồ thị vô hướng liên thông. Cây T=(V, F) với F ⊆ E được gọi là cây tối đại (cây phủ, cây bao trùm hay cây khung) của đồ thị G.

Ví dụ 4.3:



Hình 4.3: Đồ thị và các cây khung của nó

Định lý (sự tồn tại của cây tối đại):

Mọi đồ thị liên thông đều chứa ít nhất một cây tối đại.

Thuật toán tìm cây tối đại của đồ thị G:

Bước 1: Chọn tùy ý $v \in V$ và khởi tạo $X:=\{v\}$; $T:=\varnothing$

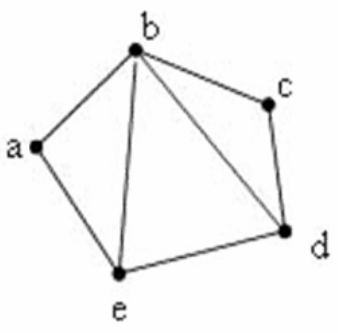
Bước 2: Chọn $w \in V \setminus X$ sao cho có một cạnh e nào đó của G nối w với một đỉnh trong X

Bước 3: Gán $X:= X \cup \{w\}$ và $T:=T \cup \{e\}$

Bước 4: Nếu T đủ $n-1$ phần tử thì dừng, ngược lại tiếp tục bước 2.

Ví dụ 4.4: Minh họa thuật toán tìm cây tối đại

Cho đồ thị sau:



Hình 4.4: Đồ thị vô hướng G

Tập V = {a, b, c, d, e},n=5, X:= ∅; T:=∅;

B1: Chọn a∈ V, khởi tạo X:={a}; T:=∅;

B2: Chọn b∈ V\X = {b, c, d, e} (vì (a, b) ∈ G);

B3: X:= {a, b}; T:= {(a,b)}

B2: Chọn e∈ V\X = { c, d, e} (vì (b, e) ∈ G);

B3: X:= {a, b, e}; T:= {(a,b), (b, e)}

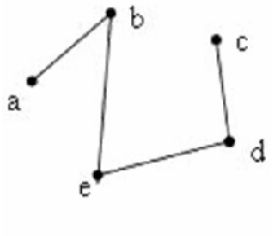
B2: Chọn d∈ V\X = { c, d} (vì (e, d) ∈ G);

B3: X:= {a, b, e, d}; T:= {(a,b), (b, e), (e, d)}

B2: Chọn c∈ V\X = { c} (vì (d, c) ∈ G);

B3: X:= {a, b, e, d, c}; T:= {(a,b), (b, e), (e, d), (d, c)}

T có 4 phần tử (n – 1 = 4) → thuật toán dừng.



Hình 4.5: Cây khung của G

4.2. CÂY KHUNG NGẮN NHẤT

Một số bài toán ứng dụng cây khung ngắn nhất:

- **Bài toán 1: Bài toán xây dựng hệ thống đường sắt**

Giả sử ta muốn xây dựng một hệ thống đường sắt nối n thành phố sao cho hành khách có thể đi từ bất kỳ một thành phố nào đến bất kỳ một trong các thành phố còn lại. Mặt khác trên quan điểm kinh tế đòi hỏi là chi phí xây dựng hệ thống đường phải nhỏ nhất. Rõ ràng đồ thị mà đỉnh là các thành phố còn các cạnh là các tuyến đường sắt nối các thành phố tương ứng với phương án xây dựng tối ưu phải là cây. Vì vây, 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 các cạnh chính là chi phí xây dựng đường ray nối hai thành phố tương ứng.

- **Bài toán 2: Bài toán nối mạng máy tính:**

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

4.2.1. Định nghĩa

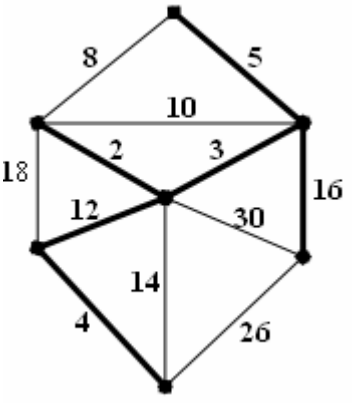
Cho G = (V, E) là đồ thị vô hướng liên thông. Mỗi cạnh *e* của đồ thị G được gán với một số thực *c(e)*, gọi là độ dài của nó. Giả sử, *H=(V, T)* là cây khung của đồ thị G. Ta gọi độ dài *c(H)* của cây khung *H* là tổng độ dài tất cả các cạnh của nó

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

Bài toán tìm cây khung ngắn nhất:

Trong số tất cả các cây khung của đồ thị G hãy tìm một cây khung có độ dài nhỏ nhất, và cây khung đó được gọi là cây khung ngắn nhất của đồ thị.

Ví dụ 4.5: Cây khung ngắn nhất của đồ thị G được vẽ bằng các cạnh tô đậm như hình dưới đây:



Hình 4.6: Cây khung ngắn nhất của đồ thị G

4.2.2. Thuật toán Kruskal

Giới thiệu thuật toán Kruskal:

Cho G là đồ thị liên thông, có trọng số, n đỉnh. Thuật toán sẽ xây dựng tập cạnh *T* của cây khung ngắn nhất *H=(V, T)*.

Ý tưởng:

Thuật toán đề xuất kết nạp lần lượt các cạnh vào tập cạnh *T* theo nguyên tắc như sau: Ưu tiên kết nạp các cạnh có trọng số nhỏ hơn, và khi nó không tạo chu trình với tập cạnh đã kết nạp trước đó.

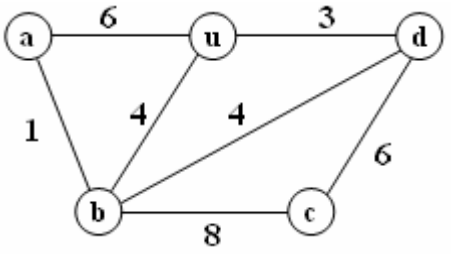
Thuật toán:

Bước 1: Chọn cạnh ngắn nhất e_1 trong các cạnh của G.

Bước 2: Khi đã chọn được k cạnh e_1, e_2, e_3, ..., e_k thì chọn tiếp cạnh e_{k+1} ngắn nhất trong các cạnh còn lại của G sao cho không tạo thành chu trình với các cạnh đã chọn trước.

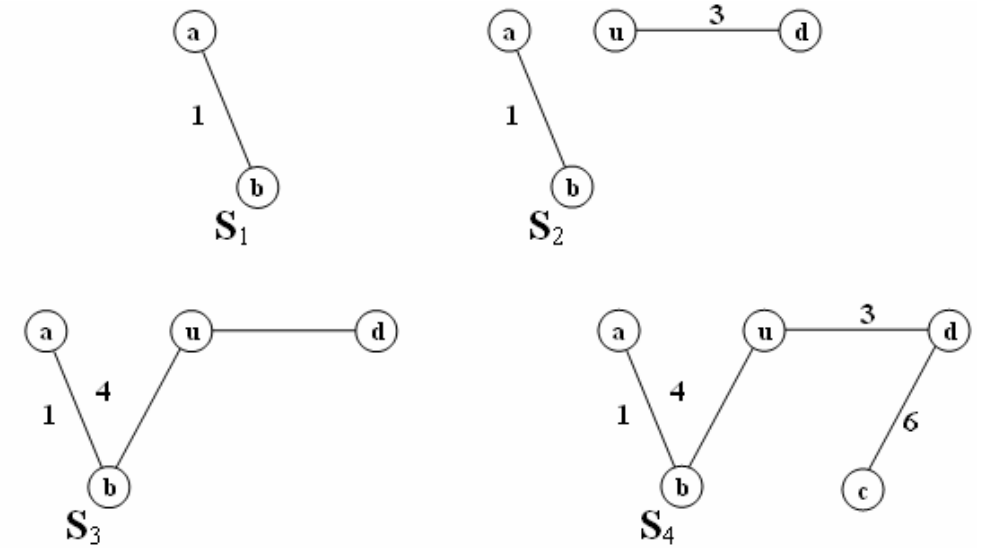
Bước 3: Chọn đủ n-1 cạnh thì dừng.

Ví dụ 4.6: Dùng thuật toán Kruskal tìm cây khung ngắn nhất của đồ thị sau:



Hình 4.7: Đồ thị vô hướng G

Minh họa thuật toán Kruskal:



Diễn giải:

Trọng số	Cạnh	
1	(a, b)	Chọn
3	(u, d)	Chọn
4	(b, u)	Chọn
4	(b, d)	Không chọn vì tạo thành chu trình: b u d b
6	(a, u)	Không chọn vì tạo thành chu trình: a b u a
6	(c, d)	Chọn. Dừng vì đã đủ cạnh
8	(b, c)	

Cài đặt thuật toán:

```
Procedure Kruskal;
Begin
    T:=∅;
    While |T| < (n-1) and (E≠∅)
    Begin
        Chọn e là cạnh có độ dài nhỏ nhất trong E;
        E:=E\ {e};
        If (T ∪ {e} không chứa chu trình) then      T:=T ∪ {e};
    End;
    If (|T| < n – 1) then đồ thị không liên thông
End;
```

✍ **Lưu ý:** Khi cài đặt thuật toán Kruskal phải dùng thuật toán sắp xếp nào đó để sắp xếp các cạnh theo thứ tự tăng dần. Đây cũng là một hạn chế của thuật toán.

4.2.3. Thuật toán Prim

Giới thiệu thuật toán Prim:

Thuật toán Krusal làm việc trên các cạnh nên sẽ kém hiệu quả nếu đồ thị có quá nhiều cạnh như các đồ thị dày (đồ thị với số cạnh $m \approx n(n-1)/2$). Đối nghịch với Kruskal, thuật toán Prim làm việc trên các đỉnh, nên sẽ hiệu quả hơn với các đồ thị dày. Và có thế thấy đa số các đồ thị trong thực tế có số đỉnh không lớn còn số cạnh rất lớn nên Prim tỏ ra hiệu quả hơn Kruskal, mặc dù cài đặt có phức tạp hơn. Thuật toán Prim còn được gọi là phương pháp lân cận gần nhất.

Ý tưởng:

Prim đề xuất cách xây dựng đồng thời tập đỉnh đã kết nạp *V* và tập cạnh đã kết nạp *T* cho cây khung ngắn nhất theo nguyên tắc như sau:

- Bắt đầu từ một đỉnh *u* bất kỳ của đồ thị, đầu tiên ta nối *u* với đỉnh lân cận gần nó nhất, chẳng hạn là đỉnh *v*. Nghĩa là trong số các cạnh kề với đỉnh *u*, cạnh *(u, v)* có độ dài nhỏ nhất.
- Trong số các cạnh kề với hai đỉnh *u* và *v* ta tìm cạnh có độ dài nhỏ nhất, cạnh này dẫn đến đỉnh thứ ba *w*, và ta thu được cây bộ phận gồm ba đỉnh và hai cạnh.
- Quá trình này sẽ được tiếp tục cho đến khi ta thu được cây gồm *n* đỉnh và *n – 1* cạnh, đây sẽ là cây khung ngắn nhất cần tìm.

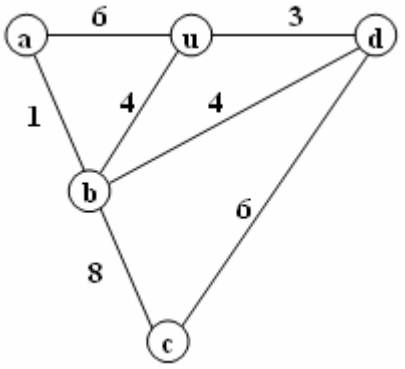
Thuật toán Prim:

Bước 1: Chọn một đỉnh bất kỳ v_1 để có cây T_1 chỉ gồm một đỉnh

Bước 2: Khi đã chọn cây T_k thì chọn tiếp cây $T_{k+1}=T_k \cup e_{k+1}$. Trong đó e_{k+1} là cạnh có độ dài nhỏ nhất trong các cạnh có một đầu mút thuộc T_k và đầu mút kia không thuộc T_k

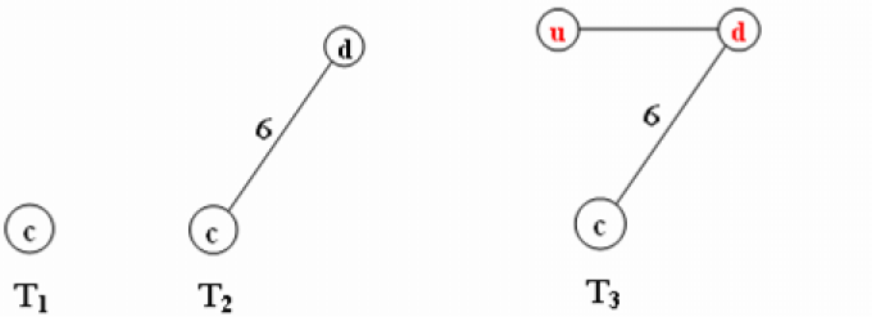
Bước 3: Chọn được cây T_n thì dừng.

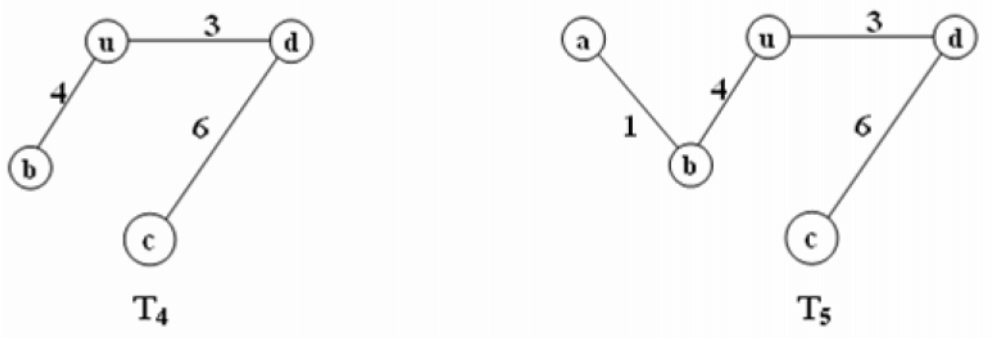
Ví dụ 4.7: Tìm cây khung ngắn nhất của đồ thị sau:



Hình 4.8: Đồ thị tìm cây khung ngắn nhất

Minh họa thuật toán Prim:





Cài đặt thuật toán:

```
Procedure Prim;
Begin
    VH:={rootPrim}; //Bất kỳ!
    T:= ∅ ;
    While |VH| < n do
    Begin
        FindMinEdge(u,v); //Tìm cạnh (u, v) có độ dài nhỏ nhất
        v → VH;  //Kết nạp đỉnh v vào VH
        (u,v) → T; //Kết nạp cạnh (u,v) vào T
    End;
End;
```

4.3. CÂY CÓ HƯỚNG

4.3.1. Định nghĩa cây có hướng

Cho G = (V, E) là một đồ thị có hướng. G được gọi là cây có hướng nếu:

(1) G không có chu trình,

(2) G có gốc

Hay:

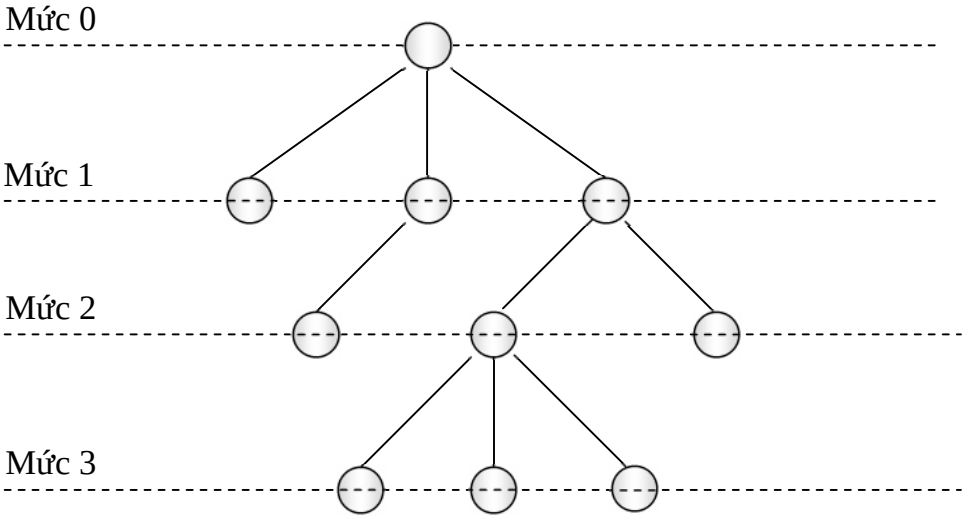
Cây có hướng là đồ thị có hướng mà *đồ thị vô hướng nền*[1] của nó là một cây.

Cây có gốc là một cây có hướng, trong đó có một đỉnh (nút) đặc biệt, gọi là gốc, từ gốc có đường đi đến mọi đỉnh khác của cây.

Một số khái niệm cơ bản:

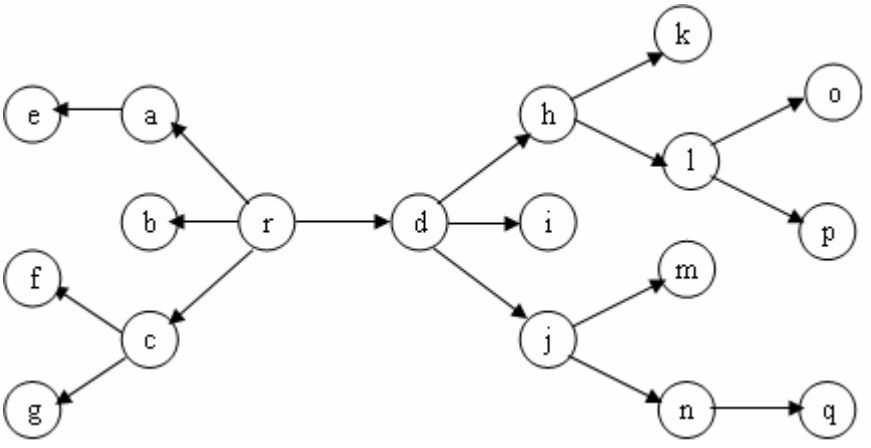
[1] Đồ thị vô hướng nền là đồ thị thu được từ đồ thị có hướng G bằng cách xóa bỏ các chiều mũi tên trên các cung

- Hai đỉnh (nút) được nối với nhau bằng một nhánh trực tiếp thì nút trên gọi là ***nút cha***, nút còn lại gọi là ***nút con***.
- ***Nút gốc*** là nút không có nút cha.
- ***Nút lá (đỉnh treo)*** là nút có bậc bằng 0.
- ***Nút nhánh (đỉnh trong)*** là nút có bậc khác 0 và không phải là gốc.
- Các nút có cùng một nút cha gọi là ***nút anh em (nút đồng cấp)***.
- ***Mức của một nút*** là độ dài đường đi từ gốc đến nút đó.



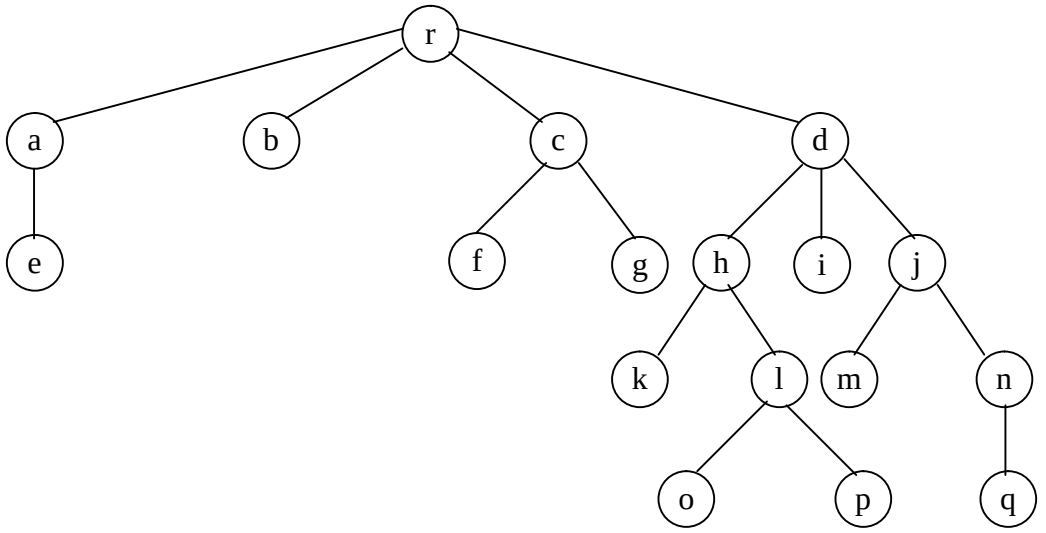
Hình 4.9: Các mức của cây

Ví dụ 4.8:



Hình 4.10: Đồ thị G

Cây có gốc ở hình 4.10 thường được vẽ như trong hình dưới đây để làm rõ mức của các đỉnh.



Hình 4.11: Cây có gốc của đồ thị G

Ví dụ 4.9: Cho cây có gốc *T* trong hình 4.10 ở trên

- Cây có gốc r
- *d* là con của *r*, *j* là con của *d*, ….
- *r* là cha của *d*, *d* là cha của *j*, …
- Đỉnh treo (hay lá): *e, b, c,*… đỉnh trong (nút nhánh): *r, d, j, n*

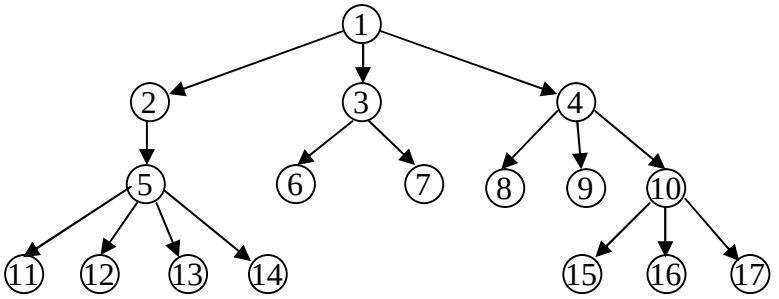
4.3.2. Định nghĩa:

Một cây có gốc T được gọi là *cây m-phân* nếu mỗi đỉnh của T có nhiều nhất là m con.

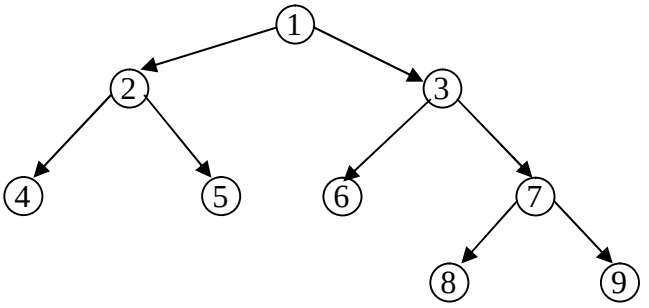
Cây *2–phân* được gọi là *cây nhị phân*.

Trong một cây nhị phân, mỗi con được chỉ rõ là con bên trái hay con bên phải; con bên trái (t.ư. phải) được vẽ phía dưới và bên trái (t.ư. phải) của cha.

Cây có gốc T được gọi là một *cây m-phân đủ* nếu mỗi đỉnh trong của T đều có đúng m con.



a) Cây 4 phân



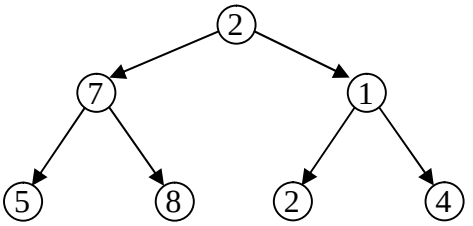
b) Cây nhị phân đủ

Hình 4.12: Cây 4 phân và cây nhị phân đủ

4.3.3. Phép duyệt cây (Cây nhị phân)

Có nhiều thuật toán duyệt cây nhị phân, các thuật toán đó khác nhau chủ yếu ở thứ tự thăm các đỉnh.

Ví dụ 4.10: Giả sử ta có cây nhị phân sau:

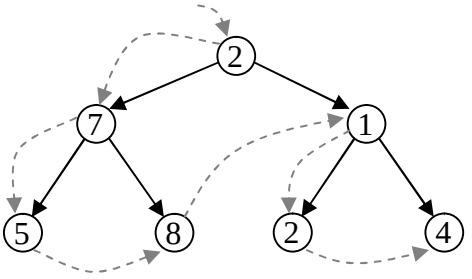


Hình 4.13: Cây có hướng T

- **Phép duyệt tiền thứ tự (NLR)**

  1. Thăm gốc r.
  2. Duyệt cây con bên trái của T(r) theo tiền thứ tự.
  3. Duyệt cây con bên phải của T(r) theo tiền thứ tự.

Duyệt cây nhị phân T trong hình 4.13 theo phép duyệt **NLR**:



Vậy thứ tự các đỉnh được in theo thứ tự là: 2, 7, 5, 8, 1, 2, 4

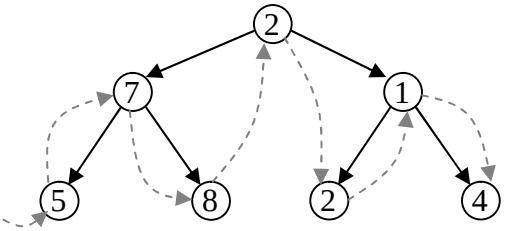
- **Phép duyệt trung thứ tự (LNR)**

  1. Duyệt cây con bên trái của T(r) theo trung thứ tự.

2. Thăm gốc r.

3. Duyệt cây con bên phải của T(r) theo trung thứ tự.

Duyệt cây nhị phân T trong hình 4.12 theo phép duyệt **LNR**:



Vậy thứ tự các đỉnh được in theo thứ tự: 5, 7, 8, 2, 2, 1, 4

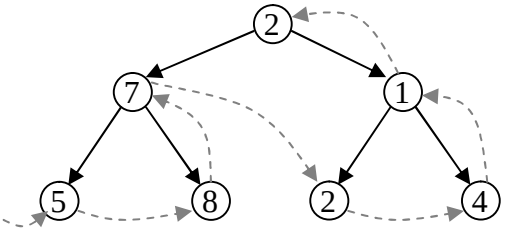
- **Phép duyệt hậu thứ tự (LRN)**

1. Duyệt cây con bên trái của T(r) theo hậu thứ tự.

2. Duyệt cây con bên phải của T(r) theo hậu thứ tự.

3. Thăm gốc r.

Duyệt cây nhị phân T trong hình 4.12 theo phép duyệt **LRN**:



Vậy thứ tự các đỉnh được in theo thứ tự là: 5, 8, 7, 2, 4, 1, 2

4.3.4. Ký pháp Balan

Thông thường, một biểu thức số học được biểu diễn theo ký pháp trung tố

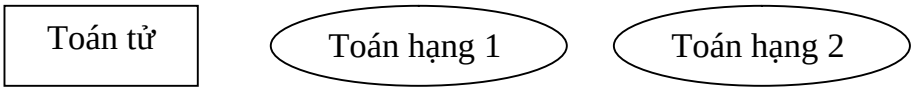
- Dấu phép toán (toán tử) nằm giữa hai toán hạng
- Thứ tự thực hiện các phép toán được xác định trong việc sử dụng các cặp dấu ngoặc hoặc quy định một thứ tự ưu tiên giữa các phép toán
- Tính toán giá trị biểu thức sẽ khá phức tạp (trên máy tính)

Ví dụ 4.11: Biểu thức dạng trung tố:

4 + 5; A + B* C

Có thể biểu diễn các biểu thức số học mà không dùng đến dấu ngoặc bằng cách sử dụng: ký pháp tiền tố hoặc ký pháp hậu tố

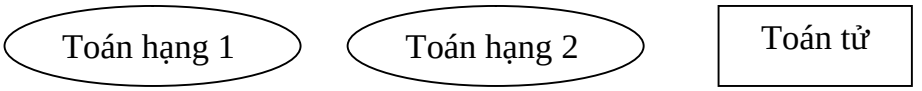
Trong ký pháp tiền tố: Toán tử luôn được đặt trước 2 toán hạng



Ví dụ 4.12: Biểu thức dạng tiền tố

+ 4 5; + A * B C

Trong ký pháp hậu tố: Toán tử luôn được đặt sau 2 toán hạng



Ví dụ 4.13: Biểu thức dạng hậu tố

4 5 +; A B C * +

Người ta gọi cách biểu diễn biểu thức theo dạng tiền tố là ký pháp Ba Lan, còn cách biểu diễn theo dạng hậu tố là ký pháp Ba Lan ngược, để ghi nhớ đóng góp của nhà toán học và lôgic học Ba Lan Lukasiewicz (1878-1956) trong vấn đề này.

Trong phần tiếp theo, chúng ta sẽ chỉ tìm hiểu cách chuyển từ biểu thức dạng trung tố sang biểu thức dạng hậu tố vì quá trình tính toán giá trị của biểu thức hậu tố khá tự nhiên đối với máy tính.

Ý tưởng là đọc biểu thức từ trái sang phải, nếu gặp một toán hạng (con số hoặc biến) thì push toán hạng này vào ngăn xếp; nếu gặp toán tử, lấy hai toán hạng ra khỏi ngăn xếp (stack), tính kết quả, đẩy kết quả trở lại ngăn xếp. Khi quá trình kết thúc thì con số cuối cùng còn lại trong ngăn xếp chính là giá trị của biểu thức đó.

Ví dụ 4.14: Biểu thức trung tố:

5 + ((1 + 2) * 4) + 3

Biểu thức này được biểu diễn lại dưới dạng hậu tố:

5 1 2 + 4 * + 3 +

Quá trình tính toán sẽ diễn ra theo như bảng dưới đây:

Ký tự	**Thao tác**	**Trạng thái Stack**
5	Push 5	5
1	Push 1	5, 1
2	Push 2	5, 1, 2
+	Tính 1 + 2 Push 3	5, 3
4	Push 4	5, 3, 4

*	Tính 4*3 Push 12	5, 12
+	Tính 12 + 5 Push 17	17
3	Push 3	17, 3
+	Tính 17 + 3 Push 20	20

Thuật toán:

(Thuật toán này dựa theo cơ chế ngăn xếp và ý tưởng chung của thuật toán là duyệt biểu thức từ trái sang phải)

- Nếu gặp một toán hạng (con số hoặc biến) thì ghi nó vào chuỗi kết quả (chuỗi kết quả là biểu thức trung tố).
- Nếu gặp dấu mở ngoặc, đưa nó vào stack.
- Nếu gặp một toán tử (gọi là o_1), thực hiện hai bước sau:
  - Nếu còn có một toán tử o_2 ở đỉnh stack và độ ưu tiên của o_1 nhỏ hơn hay bằng độ ưu tiên của o_2 thì lấy o_2 ra khỏi stack và ghi vào kết quả.
  - Push o_1 vào stack.
- Nếu gặp dấu ngoặc đóng thì lấy các toán tử trong stack ra và ghi vào kết quả cho đến khi lấy được dấu ngoặc mở ra khỏi stack.
- Khi đã duyệt hết biểu thức trung tố, lần lượt lấy tất cả toán hạng (nếu có) từ ngăn xếp ra và ghi vào chuỗi kết quả.

Để dễ hiểu, bạn hãy quan sát quá trình thực thi của thuật toán qua một ví dụ cụ thể sau:

Ví dụ 4.15: Biểu thức cần chuyển đổi: 3+4*2/(1-5)

Ký tự	Thao tác	Stack	Chuỗi hậu tố
3	Ghi 3 vào K.qua		3
+	Push +	+	
4	Ghi 4 vào K.qua		3 4
*	Push *	+ *	
2	Ghi 2 vào K.qua		3 4 2
/	(Lấy / so sánh với *)	+ /	3 4 2 *

	Lấy * ra khỏi Stack, ghi vào K.qua, Push /		
(	Push (	+ / (	3 4 2 *
1	Ghi 1 vào K.qua	+ / (	3 4 2 * 1
-	Push -	+ / (-	3 4 2 * 1
5	Ghi 5 vào K.qua	+ / (-	3 4 2 * 1 5
)	Pop cho đến khi lấy được (, ghi các toán tử pop được ra K.qua	+ /	3 4 2 * 1 5 -
	Pop tất cả các toán tử ra khỏi ngăn xếp và ghi vào K.qua		3 4 2 * 1 5 – / +

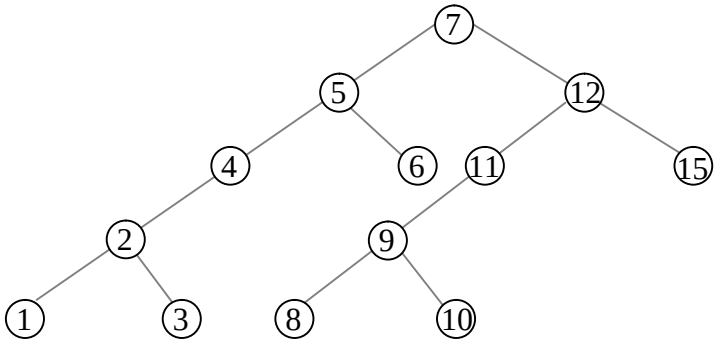
4.4. KẾT CHƯƠNG

Chương này trình bày định nghĩa một đồ thị như thế nào là một cây, những điều kiện cần và đủ để đồ thị là cây, thuật toán tìm cây tối đại của một đồ thị G. Hiểu được và vận dụng cây khung ngắn nhất, cũng như những thuật toán tìm cây khung ngắn nhất vào các bài toán thực tế.

4.5. BÀI TẬP

1) Cho biết số nút lá của Cây 3 – phân đầy đủ có 100 đỉnh.

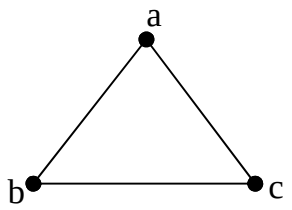
2) Xét cây nhị phân:



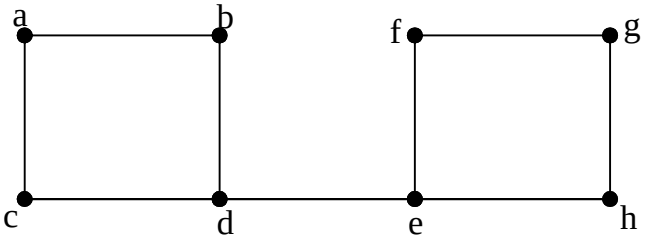
Hãy duyệt cây theo các thứ tự: tiền thứ tự, trung thứ tự, hậu thứ tự. Có nhận xét gì về giá trị của các khóa khi duyệt theo trung thứ tự.

3) Hãy vẽ tất cả các cây khung của các đồ thị sau:

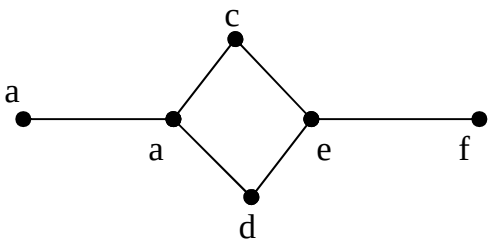
a)



b)



c)

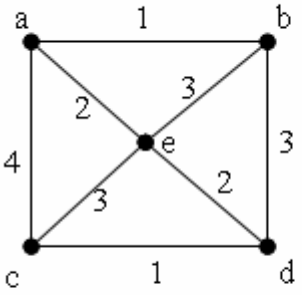


4) Đồ thị sau có tất cả bao nhiêu cây khung ?

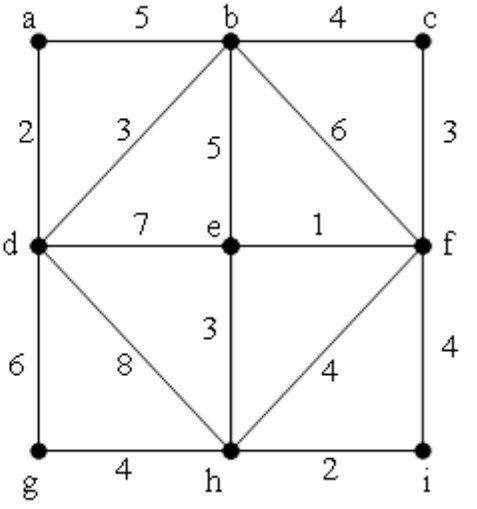
a/ K_3 b/ $K_{2,2}$ c/ C_n d/ W_n

5) Hãy tìm cây khung nhỏ nhất của đồ thị sau bằng thuật toán Prim

a)



b)



6) Hãy áp dụng thuật toán Krusksal để tìm cây khung nhỏ nhất của các đồ thị có trong câu 5.

CHƯƠNG 5

CÁC BÀI TOÁN VỀ ĐƯỜNG ĐI

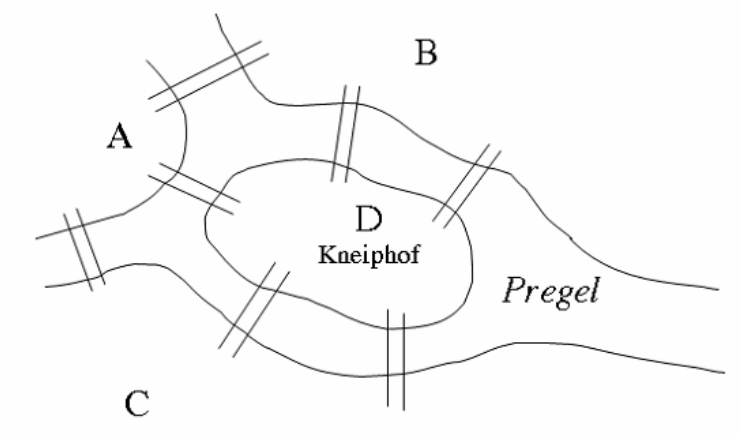
Mục tiêu**:**

- Hiểu khái niệm và các dấu hiệu nhận biết đồ thị Euler, Hamilton
- Hiểu được các thuật toán tìm đường đi ngắn nhất như Dijkstra, Ford-Bellman, Floyd và vận dụng các thuật toán này vào các bài toán tìm đường đi trong thực tế.

5.1. ĐỒ THỊ EULER

5.1.1. Bài toán về 7 cây cầu ở Konigsberg (Bài toán Euler)

Thành phố Konigsberg thuộc nước Cộng hòa Litva có con sông Pregel chảy qua, giữa sông có cù lao Kneiphof tạo nên bốn vùng đất. Người ta đã xây dựng 7 cây cầu để nối các vùng đất này lại với nhau như hình vẽ dưới đây:



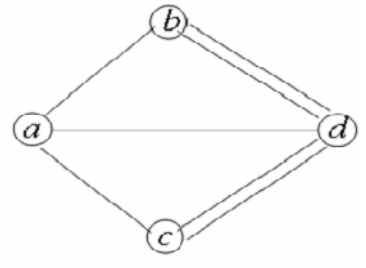
Hình 5.1: Bảy cây cầu trên sông Pregel

Có thể một lần đi qua tất cả 7 chiếc cầu này hay không?

Chuyển bài toán về dạng đồ thị:

- Mỗi vùng là một đỉnh
- Mỗi chiếc cầu là một cạnh

Đồ thị được xây dựng từ bài toán Euler



Hình 5.2: Đa đồ thị biểu diễn thành phố Konigsberg

Bài toán đồ thị đặt ra: Có thể đi qua tất cả các cạnh của đồ thị, sao cho mỗi cạnh chỉ đi qua đúng một lần được không?

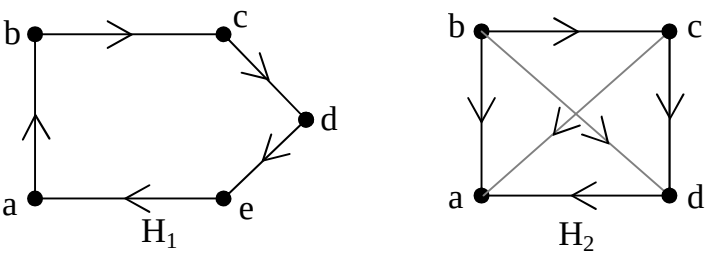
5.1.2. Định nghĩa:

Giả sử G là đơn (đa) đồ thị vô (có) hướng:

- Chu trình Euler trong G là chu trình đơn đi qua tấ cả các cạnh của đồ thị. Nếu G có chu trình Euler thì G được gọi là đồ thị Euler
- Đường đi Euler trong G là đường đi đơn qua tất cả các cạnh của đồ thị. Nếu G có đường đi Euler thì G được gọi là đồ thị nửa Euler.

Nhận xét: Rõ ràng đồ thị Euler là nửa Euler, nhưng điều ngược lại không đúng.

Ví dụ 5.1:

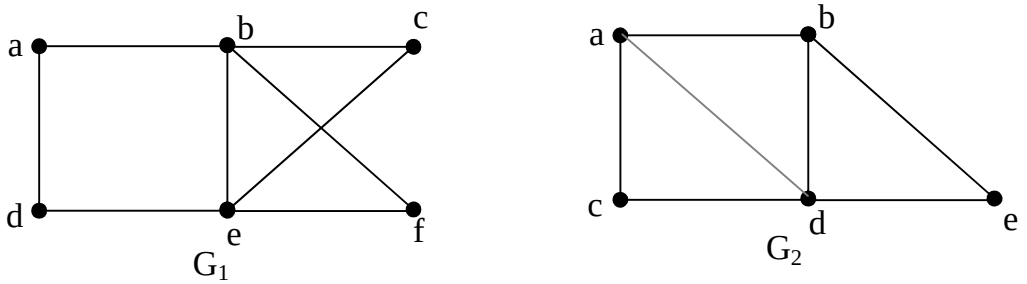


Hình 5.3: Đồ thị có hướng H_1, H_2

Đồ thị H_1 trong hình 5.3 là đồ thị Euler vì nó có chu trình Euler *a, b, c, d, e, a*

Đồ thị H_2 không có chu trình cũng như đường đi Euler.

Ví dụ 5.2:



Hình 5.4: Đồ thị vô hướng G_1, G_2

Đồ thị G_1 trong hình 5.4 là đồ thị Euler vì nó có chu trình Euler: *a, d, e, b, f, e, c, b, a.*

Đồ thị G_2 trong hình 5.4 là đồ thị nửa Euler vì nó có đường đi Euler: *a, c, d, e, b, d, a, b* (Đồ thị G_2 không có chu trình Euler).

Bổ đề

Cho đồ thị G = (V, E), nếu mọi đỉnh *u* của G có $deg(u) \geq 2$ thì G có chu trình.

Chứng minh:

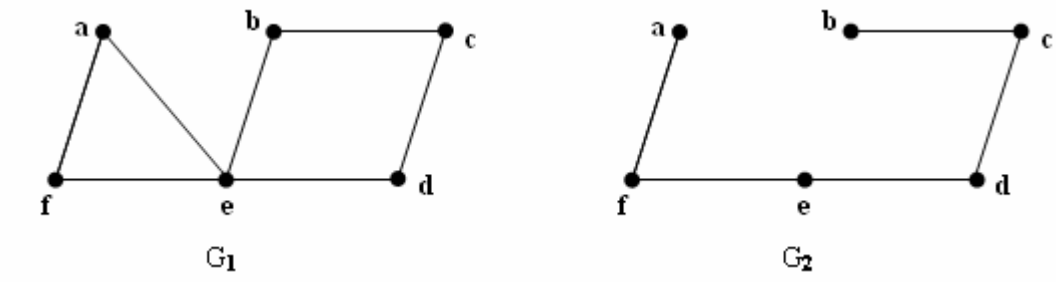
Nếu G có cạnh lặp thì khẳng định của bổ đề là hiển nhiên.

Nếu G là đơn đồ thị:

Gọi $v \in G$ (bất kỳ). Xây dựng theo quy nạp đường đi $\boldsymbol{v \rightarrow v_1 \rightarrow v_2 \rightarrow ...}$

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

Ví dụ 5.3:



Hình 5.5: Đồ thị G_1, G_2

Ta thấy các đỉnh trong đồ thị G_1 đều có bậc chẵn (deg(a)=2, deg(b)=2, deg(c)=2, deg(d)=2, deg(e)=4, deg(f)=2) và G_1 có một chu trình là: *a, f, e, d, c, b, e, a.*

Đỉnh a trong đồ thị G_2 có deg(a)=1, deg(b)=1 và rõ ràng G_2 không có chu trình

Định lý 1 (Euler)

Đồ thị vô hướng, liên thông G = (V, E) có chu trình Euler khi và chỉ khi mọi đỉnh đều có bậc chẵn.

Chứng minh:

- (Điều kiện cần) G có chu trình Euler $\Rightarrow$ Mọi đỉnh đều bậc chẵn

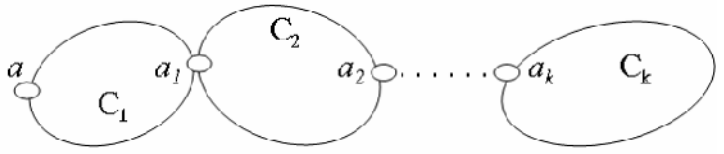
Giả sử G có chu trình Euler P. Do đó, cứ mỗi lần *P* đi qua một đỉnh *u* bất kỳ của G thì bậc của *u* tăng lên 2 (một cạnh vào và một cạnh ra tại *u*).

Mặt khác, mỗi cạnh của đồ thị chỉ xuất hiện đúng một lần, suy ra *mỗi đỉnh của đồ thị G đều có bậc chẵn.*

- (Điều kiện đủ) Mọi đỉnh đều bậc chẵn ⇒ G có chu trình Euler

  Xuất phát từ một đỉnh *a* nào đó, ta lập dãy cạnh kề liên tiếp cho đến khi hết khả năng đi tiếp. Theo giả thiết, mọi đỉnh đều có bậc chẵn nên dãy cạnh lập được phải kết thúc tại *a*. Từ đó ta thu được chu trình C_1. Nếu đã vét hết cạnh thì đó chính là chu trình cần tìm.

  Nếu vẫn còn cạnh thì do tính liên thông của đồ thị G thì phải tồn tại một cạnh nào đó chưa chọn nhưng kề với một đỉnh a_1 nào đó (a_1 tồn tại trong C_1). Từ a_1 và tiếp tục quá trình như trên cho đến khi hết khả năng đi tiếp, ta được chu trình C_2 ...



Hình 5.6: Các chu trình kề nhau

Khi đã vét hết cạnh ta lập được chu trình Euler cho đồ thị như sau:

Từ đỉnh a đi theo nửa trên của chu trình C_1 cho đến a_1, lại tiếp tục từ a_1 đi theo nửa trên của C_2 cho đến a_2 ... Khi đã đến chu trình con cuối cùng ta đi ngược lại theo các nửa dưới của các chu trình con ... và cuối cùng trở về a. Ta nhận được một chu trình Euler.

Ví dụ 5.4: Xét 2 đồ thị như trong hình 5.4

Ta thấy tất cả các đỉnh trong G_1 đều là đỉnh có bậc chẵn (deg(a)=2, deg(b)=4, deg(c)=2, deg(d)=2, deg(e)=4, deg(f)=2) và rõ ràng G_1 có chu trình Euler: *a, d, e, b, f, e, c, b, a.*

Đỉnh a trong G_2 có deg(a)=3 (bậc lẻ) và G_2 không có chu trình Euler.

Định lý 2

Đồ thị vô hướng, liên thông G = (V, E) là nửa Euler (tức là G có đường đi Euler mà không có chu trình Euler) khi và chỉ khi G có không quá hai đỉnh bậc lẻ.

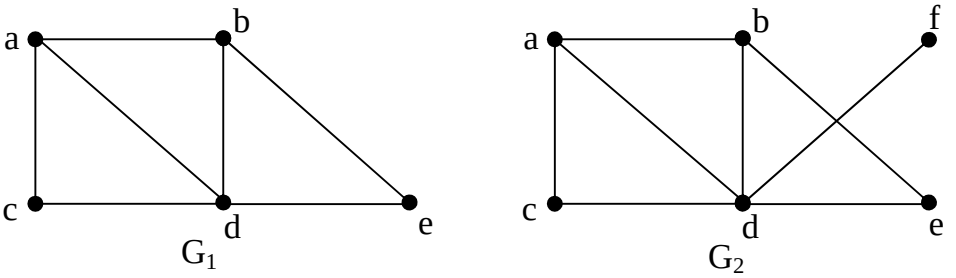
Chứng minh:

Nếu G có không quá hai đỉnh bậc lẻ thì số đỉnh bậc lẻ của G chỉ có thể là 0 hoặc là 2.

- Nếu số đỉnh bậc lẻ của G là 0 thì theo định lý 1: G là đồ thị Euler (tức G là nửa Euler, vì mọi đồ thị Euler luôn là nửa Euler)
- Nếu số đỉnh bậc lẻ của G là 2, giả sử là *u* và *v*.

  Gọi H là đồ thị thu được từ G bằng cách thêm vào G một đỉnh mới *w* và hai cạnh *(w, u)* và *(w, v)*. Khi đó, $\forall u \in H$: *deg(u) chẵn*, theo định lý 1 H có chu trình Euler. Xóa bỏ khỏi chu trình này đỉnh *w* và hai cạnh kề với nó, ta thu được đường đi Euler trong G, tức G là nửa Euler.

Ví dụ 5.5:



Hình 5.7: Đồ thị G_1, G_2

G_1 chỉ có 2 đỉnh bậc lẻ, đó là đỉnh a và b; nên G_1 có đường đi Euler là: *a, c, d, a, b, d, e, b.* Do đó, theo định lý 2 thì G_1 là đồ thị nửa Euler.

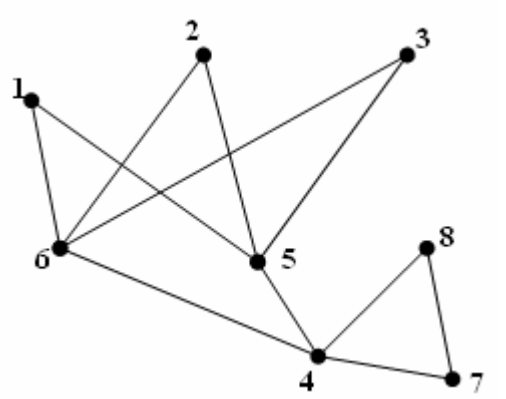
G_2 có 4 đỉnh bậc lẻ (đỉnh a, b, d, f), do đó theo định lý 2 thì G_2 không là đồ thị nửa Euler.

5.1.3. Giải thuật xây dựng chu trình Euler

```
Procedure Euler_Cycle;
Begin
    STACK := ∅;
    CE := ∅;  /* CE - Chu trình Euler */
    /* Chọn u là một đỉnh nào đó của đồ thị */
    push(STACK,u); //bỏ u vào stack
    While (STACK ≠ ∅) do
    Begin
        x:= pop(STACK); /* x là phần tử đầu của STACK */
        if (Kề(x) ≠ ∅) then
            begin
                y:= đỉnh đầu tiên trong danh sách kề Ke(x);
                push(STACK, y);
                /* Loại bỏ cạnh (x, y) khỏi đồ thị */
                Ke(x) := Ke(x) \ {y}; Ke(y) :=Ke(y) \ {x};
```

```
                end;
            else
                begin
                    x=pop(STACK);
                    bỏ x vào CE;
                end;
        end;
    End;
```

Ví dụ 5.6 (minh họa giải thuật):



Hình 5.8: Đồ thị G

Đỉnh v	Ke(v)
1	6, 5
2	5, 6
3	6, 5
4	6, 5, 7, 8
5	4, 3, 2, 1
6	4, 3, 2, 1
7	4, 8
8	4, 7

Minh họa giải thuật:

Chọn *v* là đỉnh 3

STACK	CE
3	∅
3, 6	∅
3, 6, 4	∅
3, 6, 4, 5	∅
3, 6, 4, 5, 3	∅

3, 6, 4, 5	3
3, 6, 4, 5, 2	3
3, 6, 4, 5, 2, 6	3
3, 6, 4, 5, 2, 6, 1	3
3, 6, 4, 5, 2, 6, 1, 5	3
3, 6, 4	3, 5, 1, 6, 2, 5
3, 6, 4, 7	3, 5, 1, 6, 2, 5
3, 6, 4, 7, 8	3, 5, 1, 6, 2, 5
3, 6, 4, 7, 8, 4	3, 5, 1, 6, 2, 5
∅	3, 5, 1, 6, 2, 5, 4, 8, 7, 4, 6, 3

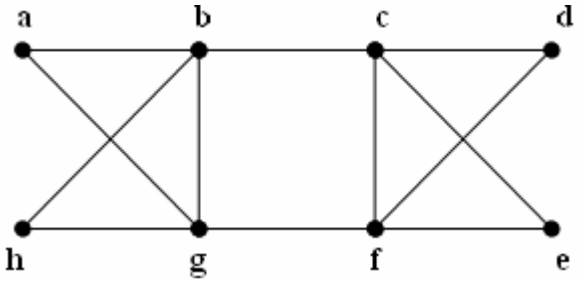
Chu trình Euler của đồ thị G là: 3, 5, 1, 6, 2, 5, 4, 8, 7, 4, 6, 3.

5.1.4. Thuật toán Fleury (Thuật toán Flor) để tìm chu trình Euler

Bắt đầu từ một đỉnh *u* bất kỳ của G và tuân theo qui tắc sau:

1. Mỗi khi đi qua một cạnh nào đó thì xóa nó đi, sau đó xóa đỉnh cô lập (nếu có).
2. Qui tắc 2: Không bao giờ đi qua cầu trừ phi không còn cách đi nào khác.

Ví dụ 5.7: Tìm chu trình Euler của đồ thị sau:



Hình 5.9: Đồ thị G

Chu trình Euler trong G là: a, b, c, f, d, c, e, f, g, h, b, g, a

5.2. ĐỒ THỊ HAMILTON

Trong mục này chúng ta xét bài toán tương tự trong mục trước chỉ khác là ta quan tâm đến đường đi qua tâm đến đường đi qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần.

5.2.1. Định nghĩa

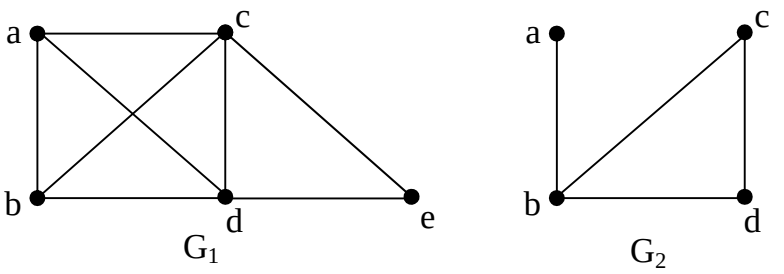
Giả sử G là đơn đồ thị vô (có) hướng:

- *Chu trình Hamilton* là chu trình xuất phát từ một đỉnh, đi thăm tất cả các đỉnh còn lại mỗi đỉnh đúng một lần, cuối cùng quay trở lại đỉnh xuất phát. Đồ thị có chu trình Hamilton gọi là đồ thị Hamilton.

- *Đường đi Hamilton* là đường đi qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần. Đồ thị có đường đi Hamilton gọi là đồ thị nửa Hamilton.

Nhận xét: Rõ ràng đồ thị Hamilton là nửa Hamilton, nhưng điều ngược lại không đúng.

Ví dụ 5.8:



Hình 5.10: Đồ thị G_1, G_2

G_1 là đồ thị Hamilton vì có chu trình Hamilton: *a, b, d, e, c, a*

G_2 là đồ thị nửa Hamilton vì có đường đi Hamilton mà không có chu trình Hamilton: *a, b, d, c*

Nhận biết đồ thị Hamilton:

- Chưa có dấu hiệu để nhận biết một đồ thị có là đồ thị Hamilton hay không.
- Chưa có thuật toán để kiểm tra
- Các kết quả thu được chỉ ở dạng điều kiện đủ
- Các phát biểu đều ở dạng: "Nếu G có số cạnh đủ lớn thì G là Hamilton".

Định lý 3 (Dirak 1952)

Đơn đồ thị vô hướng G = (V, E) có n đỉnh (với n ≥ 2). Nếu mọi đỉnh *v* của đồ thị đều có *deg(v) ≥ n/2* thì G là đồ thị Hamilton.

Định lý Dirak cho đồ thị có hướng:

Cho đồ thị có hướng, liên thông mạnh G = (V, E) và có n đỉnh. Nếu mọi đỉnh *v* của G đều có $\deg^+(v) \geq n/2$, $\deg^-(v) \geq n/2$ thì G là Hamilton.

5.2.2. Thuật toán xây dựng chu trình Hamilton

Ý tưởng:

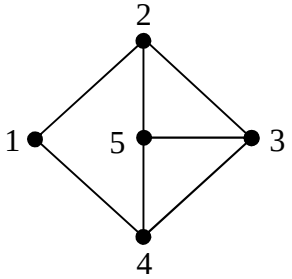
- Bắt đầu từ 1 đỉnh bất kỳ, đi theo con đường dài nhất có thể được (depth – first)

- Nếu đường đó chứa mọi đỉnh và có thể nối 2 đỉnh đầu và cuối bằng 1 cạnh thì đó là chu trình Hamilton
- Nếu trái lại ta lùi lại một đỉnh để mở con đường theo chiều sâu khác
- Cứ tiếp tục quá trình trên cho đến khi thu được chu trình Hamilton

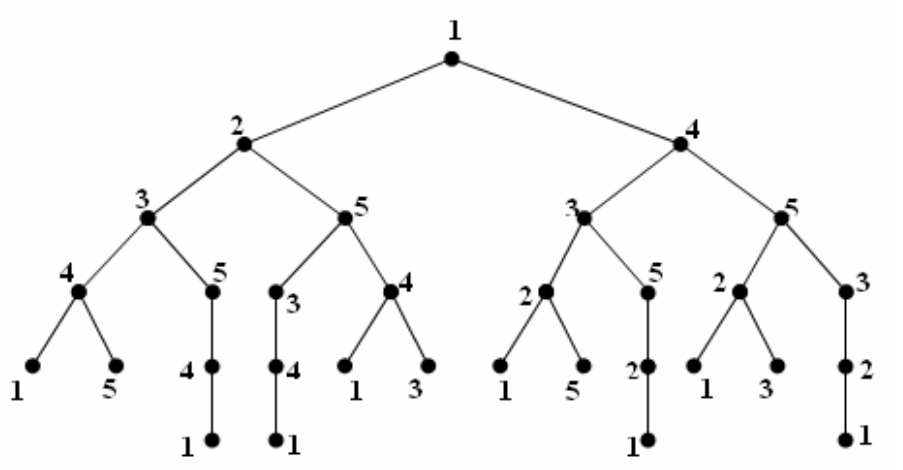
Thuật toán:

```
Procedure Hamilton(k);
    /* Liệt kê các chu trình Hamilton thu được bằng việc phát triển dãy các đỉnh
    (X[1], ..., X[k – 1]) của đồ thị G = (V, E) cho bởi danh sách kề: Ke(v), v ∈ V */
Begin
    for  y ∈ Ke(X[k – 1]) do
        if (k = n+1) and (y = v0) then Xuất(X[1], ..., X[n], v0)
        else
            if Chuaxet[y] then
            begin
                X[k] := y;
                Chuaxet[y]:=false;
                Hamilton(k+1);
                Chuaxet[y]:=true;
            end;
End;
/* Main program */
Begin
    for v ∈ V do Chuaxet[v]:=true;
    X[1]:=v0; /* v0 là một đỉnh nào đó của đồ thị */
    Chuaxet[v0]:= false;
    Hamilton(2);
End.
```

Ví dụ 5.9: Cho đồ thị G sau, tìm chu trình Hamilton của G (nếu có)



Áp dụng thuật toán tìm chu trình Hamilton, các đường đi và chu trình Hamilton được thể hiện dạng cây dưới đây:



Hình 5.11: Cây đường đi và chu trình Hamilton

5.3. BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT

Bài toán: Tìm ra đường đi ngắn nhất từ điểm A đến điểm B trong tất cả các đường đi có thể. Hiện nay có rất nhiều phương pháp để giải quyết bài toán này. Thế nhưng, thông thường, các thuật toán được xây dựng dựa trên cơ sở lý thuyết đồ thị tỏ ra là các thuật toán có hiệu quả nhất. Trong phần này, chúng ta sẽ xét một số thuật toán như vậy.

5.3.1. Các khái niệm mở đầu

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

Nếu dãy $v_0, v_1, ..., v_p$ là đường đi trên G, thì độ dài của nó được định nghĩa là tổng

$$\sum_{i=1}^{p} w(v_{i-1}, v)$$

tức là, độ dài của đường đi chính là tổng các trọng số trên các cung của nó.

5.3.2. Định nghĩa bài toán tìm đường đi ngắn nhất

Bài toán tìm đường đi ngắn nhất được phát biểu như sau: Tìm đường đi có độ dài nhỏ nhất từ một đỉnh xuất phát $s \in V$ đến đỉnh cuối (đích) $t \in V$. Đường đi như vậy ta sẽ gọi là đường đi ngắn nhất từ s đến t, còn độ dài của nó ta sẽ ký hiệu là $d(s,t)$ và còn gọi là khoảng cách từ s đến t (khoảng cách định nghĩa như vậy có thể là số âm). Nếu như không tồn tại đường đi từ s đến t thì ta sẽ đặt $d(s,t)=\infty$.

Rõ ràng, nếu như mỗi chu trình trong đồ thị đều có độ dài dương, trong đường đi ngắn nhất không có đỉnh nào bị lặp lại (đường đi không có đỉnh lặp lại sẽ gọi là đường đi sơ cấp). Mặt khác nếu trong đồ thị có chu trình với độ dài âm (chu trình như vậy để gọi ngắn gọn ta gọi là *chu trình âm* hay *mạch âm*) thì khoảng cách giữa một số cặp đỉnh nào đó của đồ thị có thể là không xác định.

5.3.3. Định nghĩa ma trận khoảng cách (trọng số)

Cho $G = (V, E)$, $V = \{v_1, v_2, ..., v_n\}$ là đơn đồ thị có trọng số. Ma trận khoảng cách của G là ma trận $D = (d_{ij})$ xác định như sau:

$$d_{ij} = \begin{cases} 0 & khi\ i = j \\ w(v_i, v_j) & khi\ (v_i, v_j) \in E \\ \infty & khi\ (v_i, v_j) \notin E \end{cases}$$

5.3.4. Thuật toán Dijkstra

Năm 1959 E. W. Dijkstra đưa ra một thuật toán rất hiệu quả để giải bài toán đường đi ngắn nhất.

Bài toán: Cho G = (V, E) đơn, liên thông, có trọng số dương (w(u, v) > 0 với mọi u khác v). Tìm đường đi ngắn nhất từ u_0 đến v và tính khoảng cách $d(u_0, v)$

Phương pháp:

Xác định tuần tự các đỉnh có khoảng cách đến u_0 từ nhỏ đến lớn

1. Trước tiên đỉnh có khoảng cách nhỏ nhất đến u_0 là u_0.
2. Trong $V\backslash\{u_0\}$ tìm đỉnh có khoảng cách đến u_0 nhỏ nhất (đỉnh này phải là một trong các đỉnh kề với u_0) giả sử đó là u_1
3. Trong $V\backslash\{ u_0, u_1\}$ tìm đỉnh có khoảng cách đến u_0 nhỏ nhất (đỉnh này phải là một trong các đỉnh kề với u_0 hoặc u_1) giả sử đó là u_2
4. Tiếp tục như trên cho đến bao giờ tìm được khoảng cách từ u_0 đến mọi đỉnh.

Nếu G có n đỉnh thì: $0 = d(u_0, u_0) < d(u_0, u_1) \le d(u_0, u_2) \le \ldots \le d(u_0, u_{n-1})$

Thuật toán Dijkstra:

Bước 1: i:=0, S:= $V\backslash\{u_0\}$, $L(u_0)$:=0, L(v):=∞ với mọi $v \in S$ và đánh dấu đỉnh v bởi (∞,-). Nếu n=1 thì xuất $d(u_0, u_0)$=0= $L(u_0)$

Bước 2: Với mọi $v \in S$ và kề với u_i (nếu đồ thị có hướng thì v là đỉnh sau của u_i), đặt $L(v):=\min\{L(v), L(u_i)+w(u_iv)\}$. Xác định $k=\min L(v)$, $v \in S$

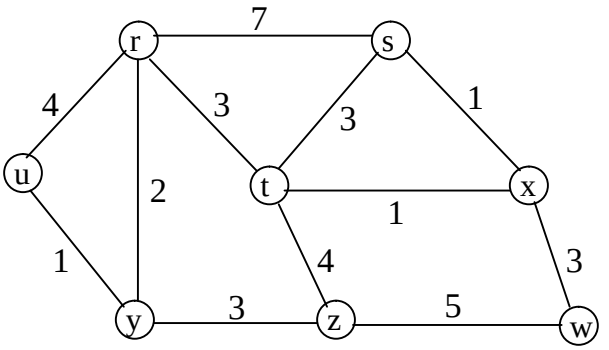
Nếu $k = L(v_j)$ thì xuất $d(u_0, v_j)=k$ và đánh dấu v_j bởi $(L(v_j);u_i)$

$u_{i+1}:=v_jS:=S\backslash\{ u_{i+1}\}$

Bước 3: i:=i+1

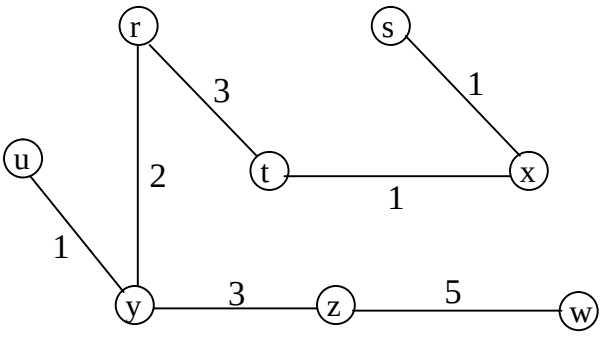
- Nếu i = n – 1 thì kết thúc
- Nếu không thì quay lại bước 2.

Ví dụ 5.10: Tìm đường đi ngắn nhất từ đỉnh u đến các đỉnh còn lại.



Hình 5.12: Đồ thị G-Tìm đường đi ngắn nhất

u	**r**	**s**	**t**	**x**	**y**	**z**	**w**
0*	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)
-	(4, u)	(∞, -)	(∞, -)	(∞, -)	(1, u)*	(∞, -)	(∞, -)
-	(3, y)*	(∞, -)	(∞, -)	(∞, -)	-	(4, y)	(∞, -)
-	-	(10, r)	(6, r)	(∞, -)	-	(4, y)*	(∞, -)
-	-	(10, r)	(6, r)*	(∞, -)	-	-	(9, z)
-	-	(9, t)	-	(7, t)*	-	-	(9, z)
-	-	(8, x)*	-	-	-	-	(9, z)
-	-	-	-	-	-	-	(9, z)*



Hình 5.13: Cây đường đi ngắn nhất từ đỉnh u đến các đỉnh còn lại

✍ **Lưu ý:** Thuật toán Dijkstra không áp dụng được cho đồ thị có trọng số âm.

5.3.5. Thuật toán Ford – Bellman

Tìm đường đi ngắn nhất từ u_0 đến các đỉnh hoặc chỉ ra đồ thị có mạch âm.

Bước 1: $L_0(u_0) = 0$ và $L_0(v) = \infty \ \forall v \neq u_0$. Đánh dấu đỉnh v bằng (∞, -); k = 1.

Bước 2: $L_k(u_0) = 0$ và

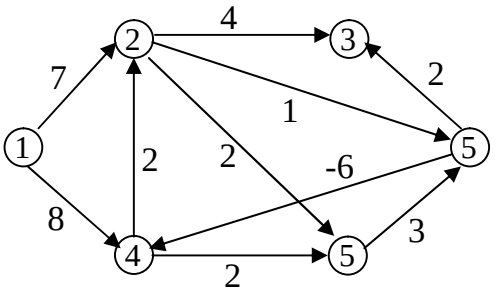
$L_k(v) = \min\{L_{k-1}(u)+w(uv)/u$ là đỉnh trước của v$\}$

Nếu $L_k(v) = L_{k-1}(y)+w(yv)$ thì đánh dấu đỉnh v bởi $(L_k(v), y)$

Bước 3: Nếu $L_k(v) = L_{k-1}(v)$ với mọi v, tức $L_k(v)$ ổn định thì dừng. Ngược lại đến bước 4.

Bước 4: Nếu k = n thì dừng. G có mạch âm. Nếu k ≤ n-1 thì trở về bước 2 với k:=k+1

Ví dụ 5.11: Tìm đường đi ngắn nhất từ 1 đến các đỉnh còn lại:

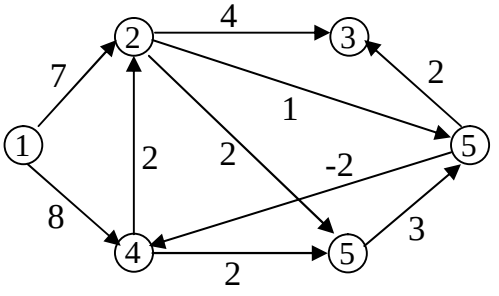


Hình 5.14: Đồ thị G-Tìm đường đi ngắn nhất với Ford Bellman

k	**1**	**2**	**3**	**4**	**5**	**6**
0	0	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)
1	0	(7, 1)	(∞, -)	(8, 1)	(∞, -)	(∞, -)
2	0	(7, 1)	(11, 2)	(8, 1)	(9, 2)	(8, 2)
3	0	(7, 1)	(10, 6)	(2, 6)	(9, 2)	(8, 2)
4	0	(4, 4)	(10, 6)	(2, 6)	(4, 4)	(8, 2)
5	0	(4, 4)	(8, 2)	(2, 6)	(4, 4)	(5, 2)
6	0	(4, 4)	(7, 6)	(-1, 6)	(4, 4)	(5, 2)

k = n = 6. $L_k(i)$ chưa ổn định nên đồ thị có mạch âm. Chẳng hạn: $4 \rightarrow 2 \rightarrow 6 \rightarrow 4$ có độ dài -3

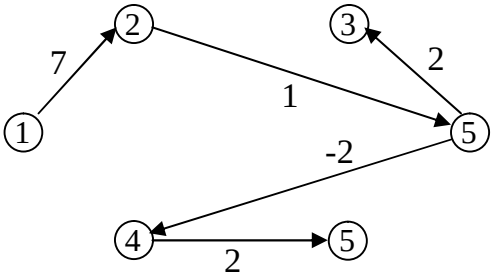
Ví dụ 5.12: Tìm đường đi ngắn nhất từ 1 đến các đỉnh còn lại



Hình 5.15: Đồ thị G

k	1	2	3	4	5	6
0	0	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)
1	0	(7, 1)	(∞, -)	(8, 1)	(∞, -)	(∞, -)
2	0	(7, 1)	(11, 2)	(8, 1)	(9, 2)	(8, 2)
3	0	(7, 1)	(10, 6)	(6, 6)	(9, 2)	(8, 2)
4	0	(7, 1)	(10, 6)	(6, 6)	(8, 4)	(8, 2)
5	0	(7, 1)	(10, 6)	(6, 6)	(8, 4)	(8, 2)

Cây đường đi ngắn nhất sau khi áp dụng thuật toán Ford-Bellman như sau:



Hình 5.16: Cây đường đi ngắn nhất từ 1 đến các đỉnh còn lại

5.3.6. Thuật toán Floyd

Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh hoặc chỉ ra đồ thị có mạch âm. Ngoài ma trận khoảng cách D ta còn dùng ma trận Q = (q_{ij}), trong đó:

$$Q_{ij} = \begin{cases} j \text{ khi } ij \in E \\ 0 \text{ khi } ij \notin E \end{cases}$$

Bước 1: D_0 = D, Q_0 = Q, k =1.

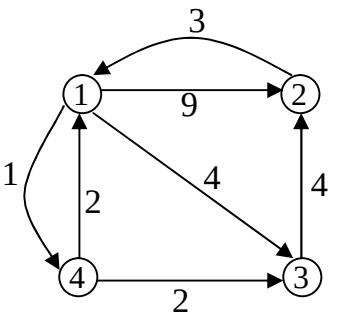
Bước 2: Với i = 1 đến n, với j = 1 đến n. Đặt

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

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

Bước 3: Nếu k = n thì dừng. Nếu k<n thì trở lại Bước 2 với k:=k+1

Ví dụ: 5.13: Dùng thuật toán Floyd, hãy tìm đường đi ngắn nhất giữa tất các cặp đỉnh của đồ thị sau:



Đồ thị trên có ma trận khoảng cách D và ma trận đường đi Q như sau:

D=

	1	2	3	4
1	0	9	4	1
2	3	0	∞	∞
3	∞	4	0	∞
4	2	∞	2	0

Q=

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

D_1=

	1	2	3	4
1	0	9	4	1
2	3	0	[7]	[4]
3	∞	4	0	∞
4	2	[11]	2	0

Q_1=

	1	2	3	4
1	0	2	3	4
2	1	0	[1]	[1]
3	0	2	0	0
4	1	[1]	3	0

D_2=

	1	2	3	4
1	0	9	4	1
2	3	0	7	4
3	[7]	4	0	[8]
4	2	11	2	0

Q_2=

	1	2	3	4
1	0	2	3	4
2	1	0	1	1
3	[2]	2	0	[2]
4	1	1	3	0

D_3=

	1	2	3	4
1	0	[8]	4	1
2	3	0	7	4
3	7	4	0	8
4	2	[6]	2	0

Q_3=

	1	2	3	4
1	0	[3]	3	4
2	1	0	1	1
3	2	2	0	2
4	1	[3]	3	0

D_4=

	1	2	3	4
1	0	[7]	[3]	1
2	3	0	[6]	4
3	7	4	0	8
4	2	6	2	0

Q_4=

	1	2	3	4
1	0	[4]	[4]	4
2	1	0	[1]	1
3	2	2	0	2
4	1	3	3	0

Đến đây thuật toán dừng và ta xác định được đường đi giữa 2 đỉnh bất kỳ dựa vào ma trận Q_4 và tổng trọng số trên đường đi tương ứng dựa vào ma trận D_4

Giả sử ta cần xác định đường đi ngắn nhất từ đỉnh 3 đến đỉnh 4, thực hiện như sau:

Tính Q[3,4]=2, nghĩa là đường đi ngắn nhất từ 3 đến 4 sẽ đi qua đỉnh 2. Tiếp tục tìm đường đi từ 2 đến 4

Tính Q[2,4]=1, nghĩa là đường đi ngắn nhất từ 2 đến 4 sẽ đi qua đỉnh 1, tiếp tục tìm đường đi từ 1 đến 4.

Tính Q[1,4]=4, dừng. vì đã đến đỉnh cuối (đỉnh 4)

Vậy đường đi ngắn nhất từ 3 đến 4 là: 3, 2, 1, 4

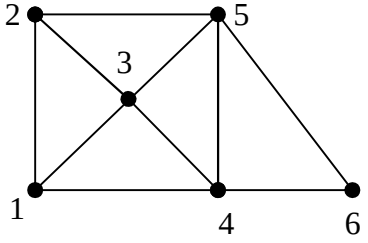
Trọng số ngắn nhất tương ứng là D[3,4]=8

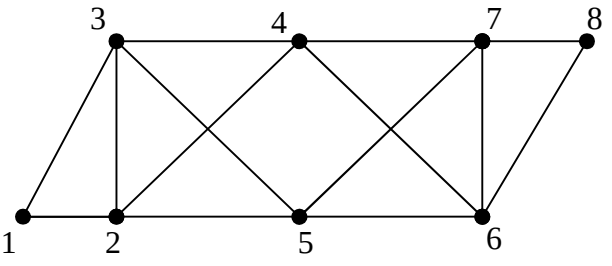
5.4. KẾT CHƯƠNG

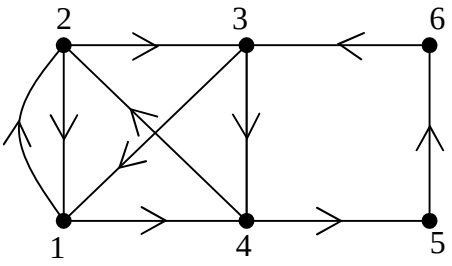
Chương này trình bày hai loại đồ thị đặc biệt là đồ thị Euler và đồ thị Hamilton, các thuật toán tìm chu trình Euler, chu trình Hamilton và các dấu hiệu nhận biết hai loại đồ thị này. Các khái niệm về đường đi ngắn nhất và các thuật toán thông dụng để tìm đường đi ngắn nhất giữa các đỉnh trên đồ thị như: Thuật toán Dijkstra, Ford-Bellman, Floyd. Hiểu và vận dụng được các thuật toán này sẽ giải quyết được nhiều bài toán về tìm đường đi trong thực tế.

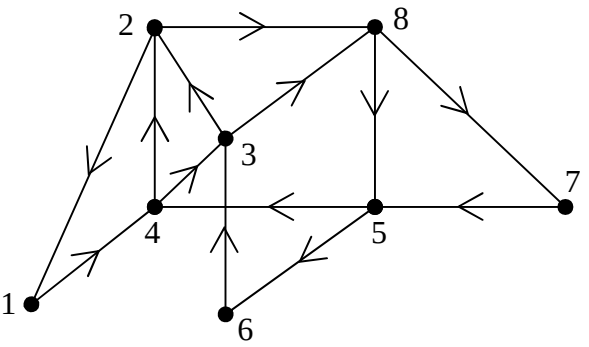
5.5. BÀI TẬP

1) Trong các đồ thị sau, đồ thị nào là Euler, nửa Euler? Nếu là đồ thị Euler, hãy tìm một chu trình Euler. Nếu là đồ thị nửa Euler, hãy tìm một đường đi Euler.

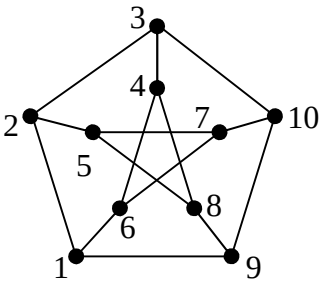




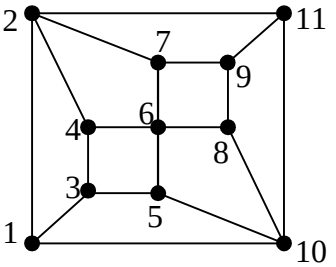




2) Hãy tìm một đường đi Hamilton có trong các đồ thị sau:

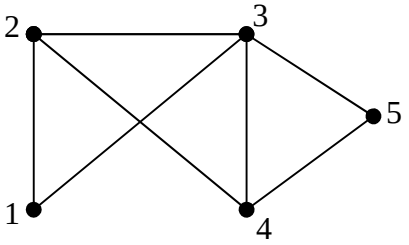


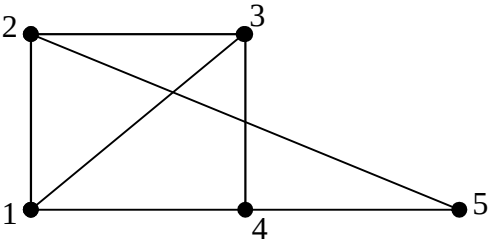
Petersen



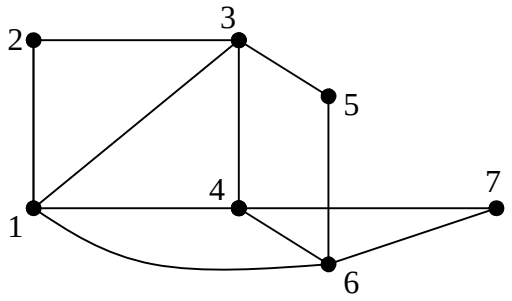
Herschel

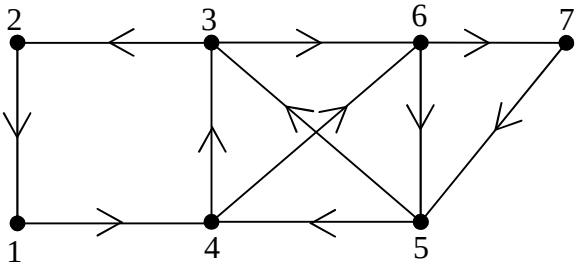
3) Dùng thuật toán xây dựng chu trình Hamilton, hãy tìm tất cả các chu trình Hamilton có trong đồ thị sau:



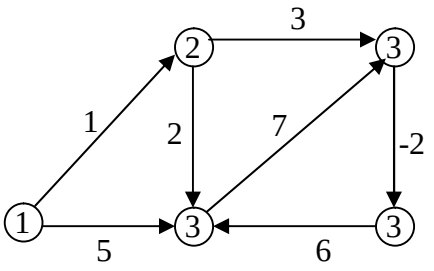


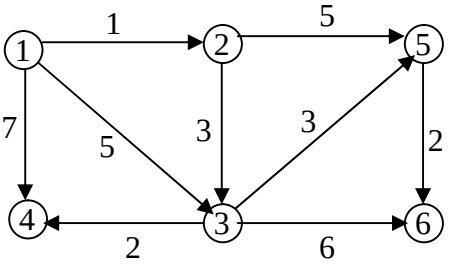
4) Dùng thuật toán xây dựng chu trình Euler, hãy tìm chu trình Euler của các đồ thị sau:



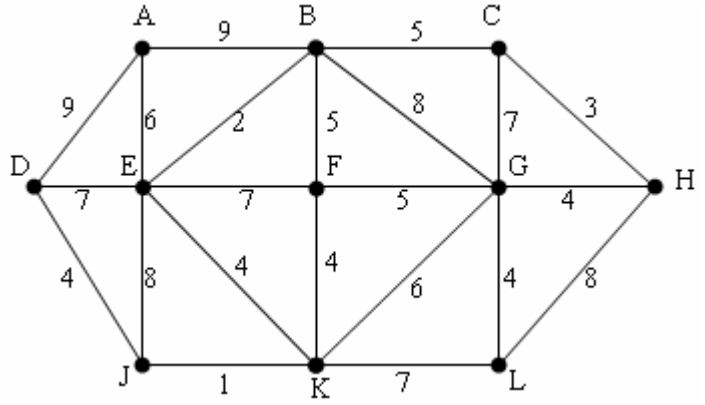


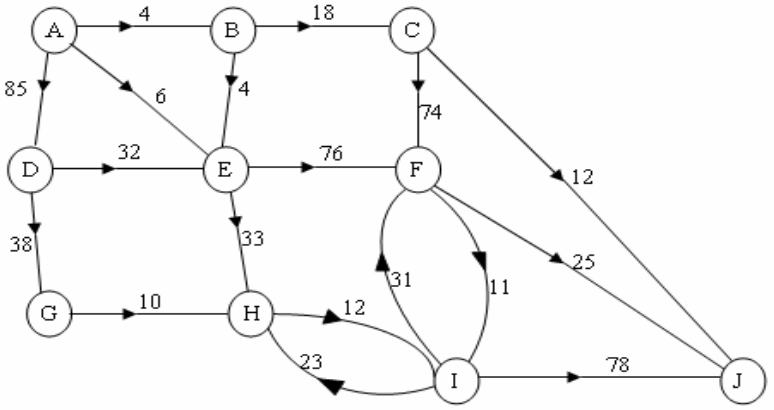
5) Dùng thuật toán Ford-Bellman tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh khác của các đồ thị sau:





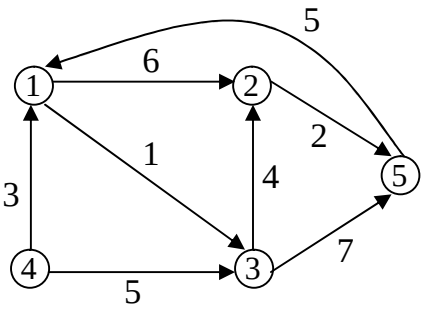
6) Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh A đến các đỉnh khác của các đồ thị sau:





7) Dùng thuật toán Floyd tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của các đồ thị sau:

	1	2	3	4
1	∞	7	5	∞
2	∞	∞	7	6
3	∞	∞	∞	∞
4	4	1	11	∞



TÀI LIỆU THAM KHẢO

[1] Trần Ngọc Danh, *Toán rời rạc nâng cao,* Nhà xuất bản Đại học quốc gia TP.HCM, 2004

[2] Hoàng Chí Thành, Đồ thị và thuật toán, NXB Giáo dục, 2007

[3] Nguyễn Đức Nghĩa, Nguyễn Tô Thành, Toán rời rạc, NXB Đại học Quốc Gia Hà Nội, 2003