HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI GIẢNG MÔN

TOÁN RỜI RẠC 2

Giảng viên: TH.S. Phan Thị Hà

Điện thoại/E-mail: hathiphan@yahoo.com

Bộ môn: Công nghệ phần mềm

Học kỳ/Năm biên soạn: I/2009-2010

NỘI DUNG

- Các thuật ngữ về đồ thị
- Biểu diễn đồ thị và sự đẳng cấu
- Tính liên thông của đồ thị
- Tính liên thông và bậc của đỉnh
- Đường đi E và Ha
- Bài toán đường đi ngắn nhất
- Đồ thị phẳng
- Tô màu đồ thị
- **Cây và cây khung của đồ thị**

GIỚI THIỆU: Lý thuyết đồ thị

- Là một trong những ngành khoa học được ra đời từ rất sớm.
- Được dùng để mô hình hóa bằng hình học và giải quyết nhiều bài toán trong thực tế.
- Các bài toán thường được mô tả thông qua các đỉnh và các đường nối giữa các đỉnh, thể hiện mối liên hệ giữa chúng.

CHƯƠNG 1:Các thuật ngữ cơ bản về đồ thị

- 1.1. Định nghĩa và khái niệm
- 1.2. Đồ thị vô hướng
- 1.3. Đồ thị có hướng
- 1.4. Đồ thị có trọng số
- 1.5 Đồ thị. phân đôi
- 1.6. Những đồ thị đơn đặc biệt
- 1.7. Một vài ứng dụng của các đồ thị đặc biệt
- 1.8. Đồ thị con

1.Định nghĩa và khái niệm

❖ ***Định nghĩa:*** đồ thị G là 1 cặp G = (V,E), trong đó:

- V: tập hợp các đỉnh
- E: tập hợp các cạnh, $E \subseteq [V]^2$

1.Định nghĩa và khái niệm
Định nghĩa khác về đồ thị

❖ Đồ thị là một cặp G = (*V*, *F*), trong đó:

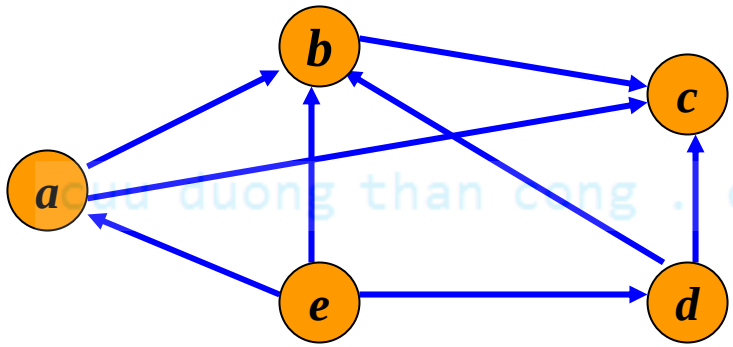
- *V* là tập hợp các đỉnh,

- F: $V \rightarrow 2^V$, được gọi là *ánh xạ kề*.

❖ Sự tương đương của hai định nghĩa:

$$\forall x, y \in V: \quad (x, y) \in E \quad \Leftrightarrow \quad y \in F(x).$$

Xét lại ví dụ 1



❖ Ánh xạ kề của các đỉnh trong hình trên:

$F(a) = \{b, c\}$, $F(b) = \{c\}$, $F(c) = \varnothing$,

$F(d) = \{b, c\}$ và $F(e) = \{a, b, d\}$.

1.Định nghĩa và khái niệm
Đồ thị vô hướng và có hướng

- Cạnh (x, y) ∈ E là *cạnh vô hướng* khi cặp đỉnh (x, y) không có thứ tự

  Đồ thị vô hướng là đồ thị chỉ chứa các cạnh vô hướng
- Cạnh (x, y) là *cạnh có hướng* khi cặp đỉnh x, y có thứ tự

  Đồ thị có hướng là đồ thị chỉ chứa các cạnh có hướng

1.Định nghĩa và khái niệm
Đồ thị đối xứng

❖ Đồ thị G = (*V*, *E*) được gọi là *đối xứng* nếu:

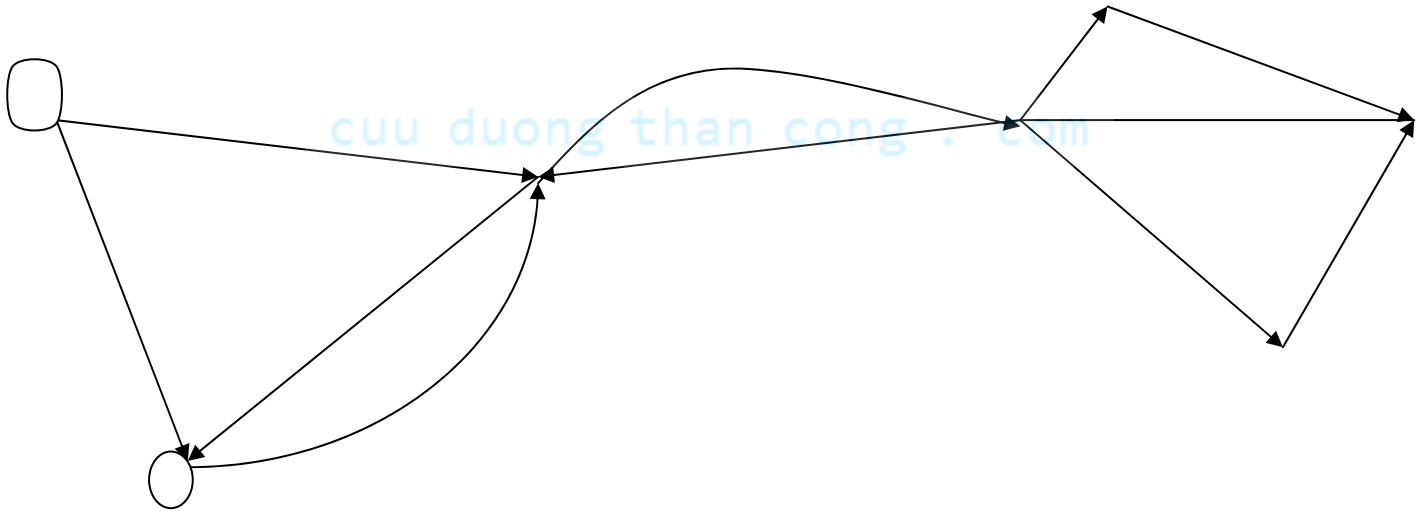
$$\forall x, y \in V: (x, y) \in E \Leftrightarrow (y, x) \in E.$$

➔ Các đồ thị vô hướng đều đối xứng.

1.Định nghĩa và khái niệm
Đơn đồ thị và đa đồ thị

- Đồ thị G = (*V, E*) trong đó mỗi cặp đỉnh chỉ được nối tối đa bằng 1 cạnh được gọi là *đơn đồ thị* hay còn gọi tắt là *đồ thị*.
- Đồ thị trong đó có ít nhất 1 cặp đỉnh được nối với nhau nhiều hơn một cạnh được gọi là *đa đồ thị*.
- Một giả đồ thị G = (V, E) trong đó các cặp đỉnh đuwcj nối với nhau là 1 hoặc nhiều cạnh và mỗi đỉnh có thể có khuyên.
- Tóm lại, giả đồ thị là loại đồ thị vô hướng tổng quát

Đơn ĐT có hướng(có thể có khuyên nhưng không có cạnh bội cùng chiều)

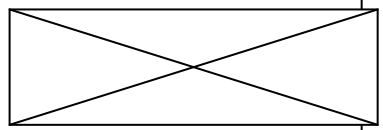


Bảng tổng hợp

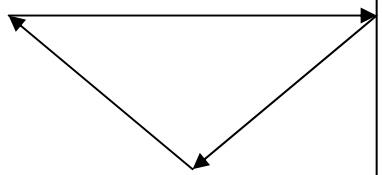
❖ Loại đồ thị	Cạnh	Có cạnh bội?	Có khuyên?
Đơn đồ thị	VH	Không	Không
Đa đồ thị	VH	Có	Không
Giả đồ thị	VH	Có	Có
Đồ thị có hướng	CH	Không	Có
Đa đồ thị có hướng	CH	Có	Có

1.Định nghĩa và khái niệm
Đồ thị đầy đủ

❖ Đồ thị vô hướng G = (V,E) được gọi là ***đồ thị đầy đủ***, nếu mỗi cặp đỉnh đều có cạnh nối giữa chúng.



❖ Đồ thị ***có hướng*** G = (V,E) được gọi ***là đồ thị đầy đủ***, nếu mỗi cặp đỉnh đều có cung nối giữa chúng (chiều của cung có thể tùy ý).



1.Định nghĩa và khái niệm

Bậc của đỉnh trong ĐT vô hướng

- Giả sử G = (V, E) là một đồ thị, ta gọi bậc của một đỉnh là số cạnh kề với đỉnh đó.
  - ***Đỉnh cô lập*** là đỉnh không nối với bất kỳ đỉnh nào.
  - ***Đỉnh treo*** là đỉnh có bậc bằng 1.

- Ký hiệu: d(v) là bậc của đỉnh v trong đồ thị G.

- Chú ý: khuyên của đồ thị được tính là bậc 2.

1.Định nghĩa và khái niệm

Bậc của đỉnh trong ĐTcó hướng

- Trong đồ thị có hướng:
  - ***bậc- vào*** của đỉnh v ký hiệu là d-(v) là số các cạnh có đỉnh cuối là v.
  - ***Bậc- ra*** của đỉnh v ký hiệu là d+(v) là số các cạnh có đỉnh đầu là v.
  - (Như vậy một khuyên tại một đỉnh sẽ góp thêm 1 đơn vị vào bậc vào và 1 đơn vị vào bậc ra của đỉnh này).

1.Định nghĩa và khái niệm

Bậc của đỉnh

- ĐL1(*Định lý bắt tay):*. Cho G = (V,E) là một đồ thị vô hướng có e cạnh. Khi đó $2e = \sum d(v)$
  - (Định lý này đúng cả khi đồ thị có cạnh bội hoặc các khuyên)
- **Định lý 2**. Trong một đồ thị vo hướng số các đỉnh bậc lẻ là một số chẵn.

1.Định nghĩa và khái niệm

Đồ thị con và đồ thị riêng

- Giả sử G = (V, E) là một đồ thị.

  Đồ thị G' = (V', E') được gọi là *đồ thị con* của đồ thị G nếu:

  V'⊆V và E'⊆E.

- Đồ thị G" = (V, E") với E" ⊆ E, được gọi là *đồ thị riêng* của đồ thị G.

1.Định nghĩa và khái niệm

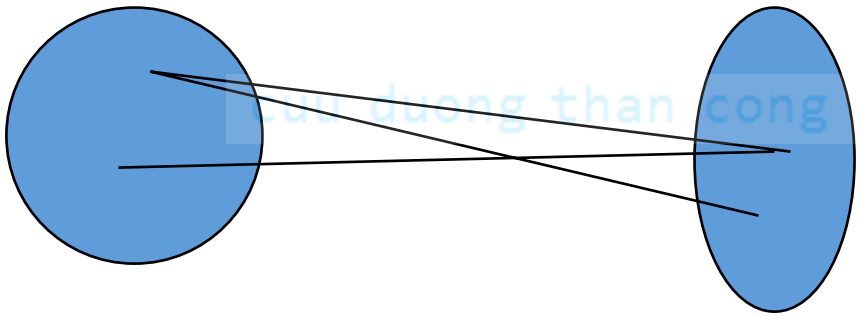
T/C Đồ thị con và riêng

- Mỗi tập con các đỉnh V' của đồ thị G tương ứng duy nhất với một đồ thị con.
- Để xác định một đồ thị con ta chỉ cần nêu tập đỉnh của nó.
- Đồ thị riêng là đồ thị giữ nguyên tập đỉnh và bỏ bớt một số cạnh.

1.Định nghĩa và khái niệm

Đồ thị phân đôi

- Một đồ thị đơn G được gọi là đồ thị ***phân đôi*** nếu tập các đỉnh V có thể phân làm hai tập con không rỗng, rời nhau V1 và V2 sao cho mỗi cạnh của đồ thị nối một đỉnh của V1 với một đỉnh của V2.



1.Định nghĩa và khái niệm

Sự đẳng hình

- Hai đồ thị $G_1 = (V_1, E_1)$ và $G_2 = (V_2, E_2)$ được gọi là *đẳng hình* với nhau nếu tồn tại một song ánh S trên các tập đỉnh bảo toàn các cạnh:

  $\forall x, y \in V_1: (x, y) \in E1 \Leftrightarrow (S(x), S(y)) \in E2.$

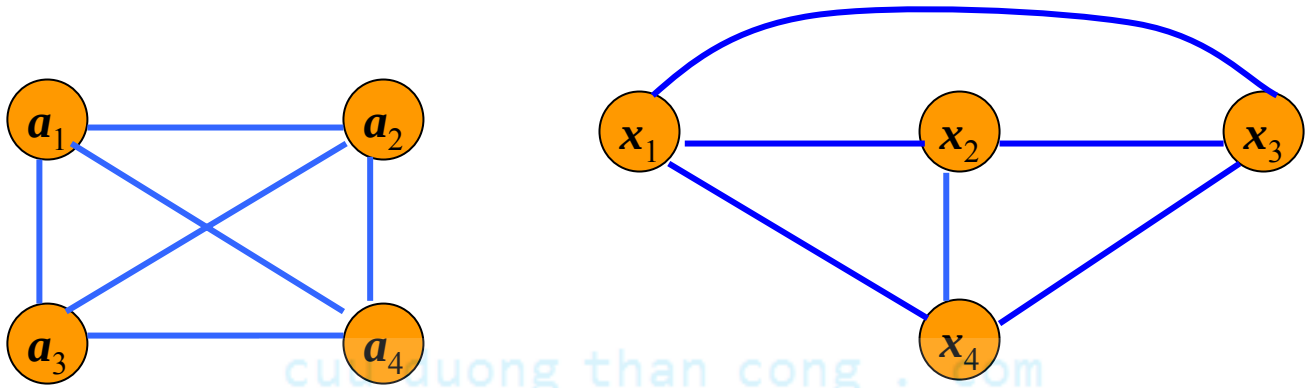
- Hai đồ thị đẳng hình chỉ khác nhau về tên gọi của các đỉnh và cách biểu diễn bằng hình vẽ.

  Do vậy, ta không phân biệt hai đồ thị đẳng hình với nhau

Ví dụ đẳng hình

Hai đồ thị sau là đẳng hình với song ánh:

$$S(a_i) = x_i \quad , i = 1, 2, 3, 4.$$



Chương2.Biểu diễn đồ thị

- Danh sách kề
- Ma trận liền kề
- Ma trận liên thuộc
- Danh sach canh

1.Biểu diễn bằng ma trận liền kề

- Cho G = (V, E) là một đồ thị có các đỉnh được đánh số:1, 2, ... , n.
- Ma trận vuông A cấp n được gọi là ma trận liền kề của đồ thị G nếu:

$$\forall i, j \in V, A[i,j] = \begin{cases} 1 \text{ nếu giữa I,j có cạnh kề} \\ 0 \text{ không có cạnh kề} \end{cases}$$

- Đồ thị G là đối xứng khi và chỉ khi ma trận kề A là đối xứng.
- Một khuyên được tính là 1 cạnh

- Cho G = (V, E)một gỉa đồ thị có các đỉnh được đánh số:1, 2, ... , n.
-
-
  - $\forall i, j \in V$, $A[i,j] = \begin{cases} m & \text{nếu giữa I,j có m cạnh} \\ 0 & \text{không có cạnh kề} \end{cases}$
- Đồ thị G là đối xứng khi và chỉ khi ma trận kề A là đối xứng.
- Một khuyên được tính là 1 cạnh

BÀI GIẢNG MÔN

TOÁN RỜI RẠC 2

GIẢNG VIÊN: TH.S.PHAN THỊ HÀ

BỘ MÔN: CÔNG NGHỆ PHẦN MỀM

BÀI GIẢNG MÔN

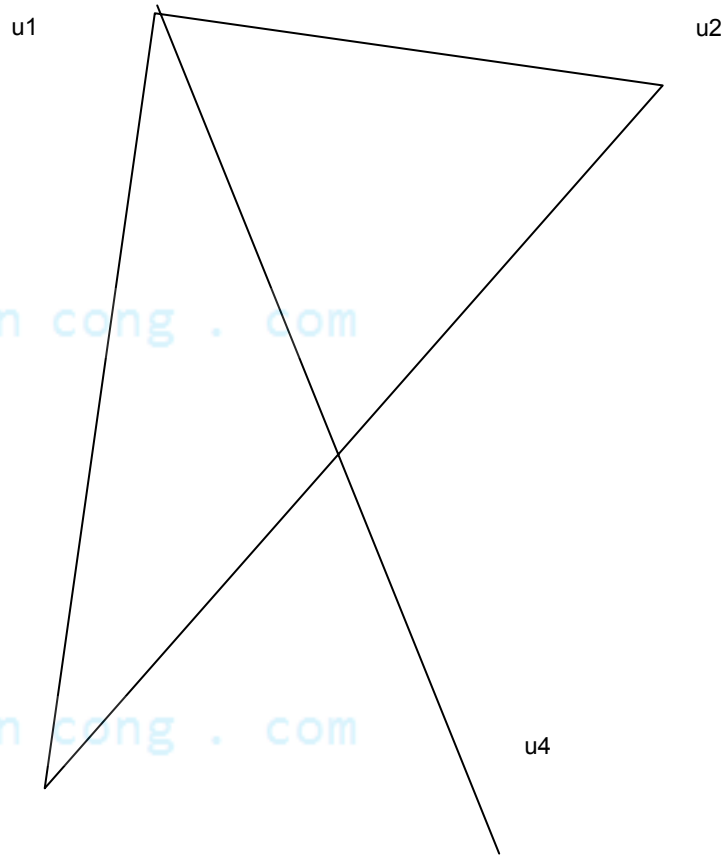
TOÁN RỜI RẠC 2

GIẢNG VIÊN: TH.S.PHAN THỊ HÀ

BỘ MÔN: CÔNG NGHỆ PHẦN MỀM

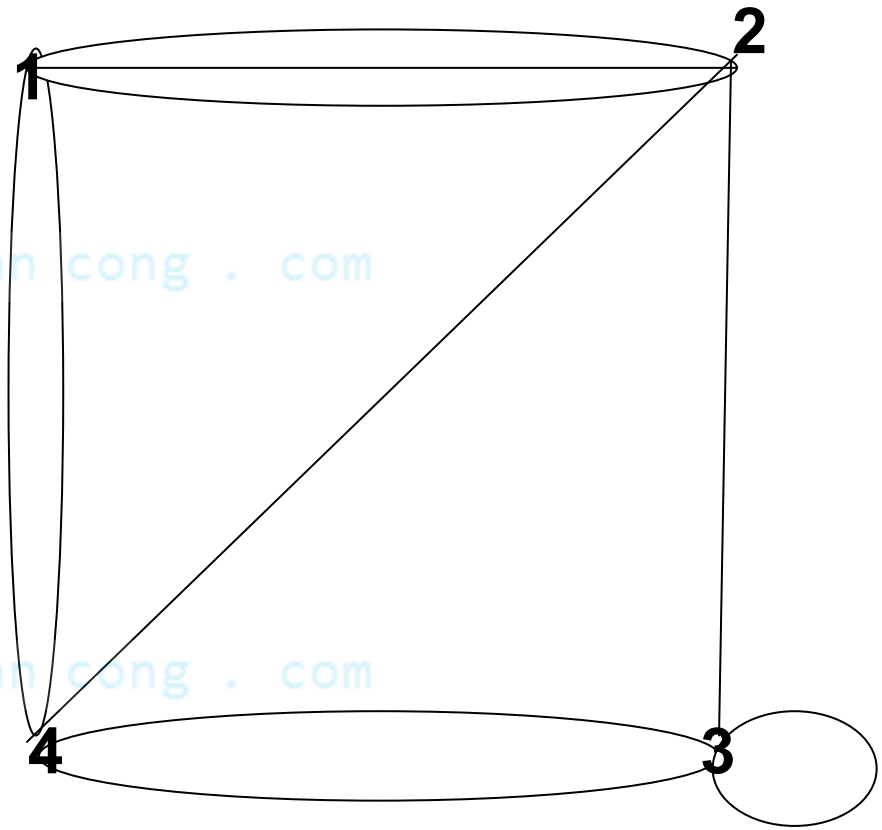
VD

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$



- Ma trận kề cũng có thể dùng để biểu diễn đồ thị vô hướng có khuyên và có cạnh bội. Khi có cạnh bội ma trận liền kề không còn là ma trận không-một nữa, vì phần tử ở vị trí (i,j) của ma trận này ***bằng số cạnh bội nối đỉnh ui và đỉnh uj***. Khuyên tại đỉnh ui được coi là cạnh nối đỉnh ui với chính nó và được tính là một cạnh. Tất cả các đồ thị vô hướng, kể cả đa đồ thị và giả đồ thị đều có ma trận liền kề đối xứng.

$$\begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}$$



2.Biểu diễn bằng danh sách kề

- Với mỗi đỉnh của đồ thị ta xây dựng một danh sách móc nối chứa các đỉnh kề với đỉnh này: Danh sách này được gọi là danh sách kề.
- Một đồ thị được biểu diễn bằng một mảng các danh sách kề.

```
void Dske_To_MTke(void){
        int dau, cuoi; char str[132], tu[12];
        //Doc va chuyen doi thanh ma tran ke
        fp = fopen("DSKE.IN","r");
        if(fp==NULL){
                printf("\n Do thi khong ton tai"); getch();
return;
        }
        // doc gia so dinh cua do thi
```

```
fgets(str, 132, fp);n = atoi(str);
        printf("\n So din do thi:%d",n);Init();
        for( dau=1; dau<=n; dau++){
                fgets(str,132,fp);
                for(int j=0, p=0; j<=strlen(str); j++){
                        //tach tu
                        if(str[j]!=' '&&str[j]!='\t'&&str[j]!='\0'&& str[j]!='\n'){
                                tu[p]=str[j]; p++;
                        }
                        else if( p>0 && (str[j]==' '||str[j]=='\t'||str[j]=='\0'||str[j]=='\n')){
                                        tu[p]='\0'; p=0;cuoi=atoi(tu);
                                        if(cuoi!=0) { m++; A[dau][cuoi]=1; }
                                        //thiet lap gia tri cho ma tran ke
                        }
                }
        }
        fclose(fp);}
```

```
fgets(str, 132, fp);n = atoi(str);
        printf("\n So din do thi:%d",n);Init();
        for( dau=1; dau<=n; dau++){
                fgets(str,132,fp);
                for(int j=0, p=0; j<=strlen(str); j++){
                        //tach tu
                        if(str[j]!=' '&&str[j]!='\t'&&str[j]!='\0'&& str[j]!='\n'){
                                tu[p]=str[j]; p++;
                        }
                        else if( p>0 && (str[j]==' '||str[j]=='\t'||str[j]=='\0'||str[j]=='\n')){
                                        tu[p]='\0'; p=0;cuoi=atoi(tu);
                                        if(cuoi!=0) { m++; A[dau][cuoi]=1; }
                                        //thiet lap gia tri cho ma tran ke
                        }
                }
        }
        fclose(fp);}
```

```
printf("\n Ghi vao Ma tran ke:");
        fp=fopen("MATRANKE.OUT","w");
        fprintf(fp,"%3d",n);
        for(int i=1; i<=n; i++){
                printf("\n");fprintf(fp,"%s","\n");
                for(int j=1; j<=n; j++){
                        printf("%3d",A[i][j]);
                        fprintf(fp,"%3d",A[i][j]);
                }
        }
        fclose(fp); getch();
}
```

3.Ma trận liên thuộc

- Các ma trận liên thuộc cũng có thể được dùng để biểu diễn các cạnh bội và khuyên. G = (V,E) V = {v1, v2, ..., vn} E = {e1, e2, ..., em}
- Ma trận liên thuộc của đồ thị G là M = [mij], trong đó mij =1 nếu cạnh ej liên thuộc với đỉnh vi và = 0 nếu cạnh ej không liên thuộc với đỉnh vi .

4.Sự đẳng cấu của đồ thị

ĐN: Các đồ thị đơn G1 =(V1,E1) và G2 =(V2,E2) là đẳng cấu nếu có hàm song ánh f từ V1 lên V2 sao cho các đỉnh a và b là liền kề trong G1 nếu và chỉ nếu f(a) và f(b) là liền kề trong G2 với mọi a và b trong V1. Hàm f như thế được gọi là ***một đẳng cấu***.

Hay: Hai đồ thị G = (V,E) và G' = (V',E') gọi là đẳng cấu với nhau nếu :

4.Sự đẳng cấu của đồ thị

- có một phép tương ứng 1 – 1(song ánh) giữa 2 tập V, V'
- và có một phép tương ứng 1 – 1 giữa 2 tập hợp E, E'

Sao cho:

nếu cạnh e = (v,w) $\in$ E tương ứng với cạnh e' = (v',w') $\in$ E' thì cặp đỉnh v, w $\in$ V cũng là tương ứng của cặp đỉnh v', w' $\in$ V

❖ G, G' đẳng cấu nếu tồn tại một song ánh φ: V→V' sao cho: (i, j) $\in$ E (φ(i), φ(j)) $\in$ E'.

4.Sự đẳng cấu của đồ thị

- Nếu G, G' là đẳng cấu qua ánh xạ φ thì hai đồ thị:
  - Có cùng số đỉnh, tức là |V| = |V'|
  - Có cùng số cạnh: |E| = |E'|
  - Có cùng số đỉnh với bậc cho sẵn
  - Số đỉnh kề với đỉnh $i \in V$ và $\varphi(i) \in V'$ là như nhau.

Chương3. Tính liên thông của đồ thị

- Đường đi
- Tính liên thông trong đồ thị vô hướng
- Tính liên thông trong đồ thị có hướng
- Đếm đường đi giữa các đỉnh
- Phương pháp duyệt đồ thị

1.Đường đi

- Cho đồ thị G = (V, E)
  - Đường đi trong đồ thị là một dãy các đỉnh:

    $<x_1, x_2, \ldots, x_i, \ldots, x_{k-1}, x_k>$

    sao cho mỗi đỉnh (trừ đỉnh đầu) đều có cạnh kề với đỉnh trước đó.

    $\forall i = 2, 3, \ldots, k-1, k : \quad (x_{i-1}, x_i) \in E.$
- Đường đi này đi từ đỉnh đầu x_1 đến đỉnh cuối x_k.

1.Đường đi

- Độ dài của đường đi là tổng các cạnh thuộc đường đi đó.
- Đường đi là đường đi đơn nếu các cạnh trên nó xuất hiện không quá 1 lần .
- Đường đi được gọi là sơ cấp nếu nó đi qua mỗi đỉnh đúng 1 lần

2.Chu trình

- *Chu trình* là một đường đi khép kín (đỉnh cuối trùng với đỉnh đầu của đường đi).

  $[x_1, x_2, \ldots, x_{k-1}, x_1]$

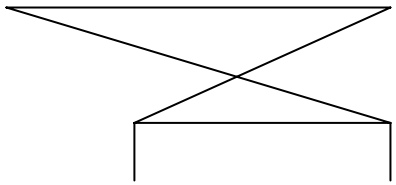
- Tuy nhiên để cho gọn, trong ký hiệu của chu trình, thường không viết đỉnh cuối:

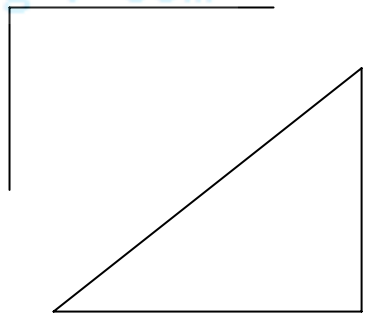
  $[x_1, x_2, \ldots, x_{k-1}]$

- *Chu trình đơn:* là chu trình mà các đỉnh trên nó khác nhau từng đôi.

3.Tính liên thông trong đồ thị vô hướng

- Một đồ thị vô hướng được gọi là ***liên thông*** nếu có đường đi giữa hai đỉnh bất kỳ của đồ thị.





3.Tính liên thông của đồ thị vô hướng

❖ **Định lý 1**. Giữa mọi cặp đỉnh phân biệt của một đồ thị vô hướng liên thông luôn luôn có đường đi đơn.

3.Tính liên thông của đồ thị có hướng

- **ĐN** . Đồ thị có hướng được gọi là ***liên thông mạnh*** nếu có đường đi từ a tới b và từ b tới a với mọi đỉnh a và b bất kỳ của đồ thị.
- **ĐN**. Đồ thị có hướng được gọi là ***liên thông yếu*** nếu có đường đi giữa hai đỉnh bất kỳ của đồ thị vô hướng nền. (Như vậy đồ thị liên thông mạnh thì cũng liên thông yếu)

3.Tính liên thông của đồ thị vô hướng

Định nghĩa:

- Đồ thị liên thông không có bất kỳ chu trình nào được gọi là một ***cây.***
- một nhóm cây trong đó không có 2 cây nào được nối với nhau được gọi là một ***rừng***.
- ***Rừng bao trùm*** của một đồ thị là rừng và là đồ thị con của đồ thị đó và chứa tất cả các nút của đồ thị; rừng gồm các cây T1, T2,... ,Tm trong đó cây Ti chứa tất cả các nút liên thông với nút gốc của nó trong đồ thị.
- Nếu ***rừng bao trùm*** chỉ gồm một cây thì cây đó được gọi là ***Cây bao trùm*** của đồ thị đó.

4.Đếm đường đi giữa các đỉnh

❖ **Định lý 2**. Cho G là một đồ thị với ma trận liền kề A theo thứ tự các đỉnh v1,v2, ...,vn (với các cạnh vô hướng hoặc có hướng hay là cạnh bội, hoặc có thể có khuyên). Số các đường đi khác nhau độ dài r từ vi đến vj trong đó r là một số nguyên dương, bằng giá trị của phần tử (i,j) của ma trận A^r.(cm-SGK)

```
void nhan(int kq[][10],int a[][10],int n)
{
      int tmp[10][10];

     for(int i = 1; i <= n; i++)
     {
            for( int j = 1; j <= n;j ++)
          {tmp[i][j]=0;
                  for( int k = 1 ; k <= n; k++)

                  tmp[i][j] += kq[i][k]*a[k][j];
           }
      }

      for(int i = 1; i <= n; i++){
            for( int j = 1; j <= n;j ++){
                  kq[i][j]=tmp[i][j];}}

}
```

5.*Phương pháp duyệt đồ thị*
Duyệt đồ thị theo độ sâu (Depth - first traverse)

- Tư tưởng cơ bản của thuật toán là bắt đầu tại một đỉnh *v0* nào đó, chọn một đỉnh *u* bất kỳ kề với *v0* và lấy nó làm đỉnh duyệt tiếp theo. Cách duyệt tiếp theo được thực hiện tương tự như đối với đỉnh v0 với đỉnh bắt đầu là *u*.

- Để kiểm tra việc duyệt mỗi đỉnh đúng một lần, chúng ta sử dụng một mảng *chuaxet[]* gồm *n* phần tử (tương ứng với *n* đỉnh), nếu đỉnh thứ *i* đã được duyệt, phần tử tương ứng trong mảng chuaxet[] có giá trị *FALSE*. Ngược lại, nếu đỉnh chưa được duyệt, phần tử tương ứng trong mảng có giá trị *TRUE*. Thuật toán có thể được mô tả bằng thủ tục đệ qui *DFS ()* trong đó: *chuaxet* - là mảng các giá trị logic được thiết lập giá trị *TRUE*.

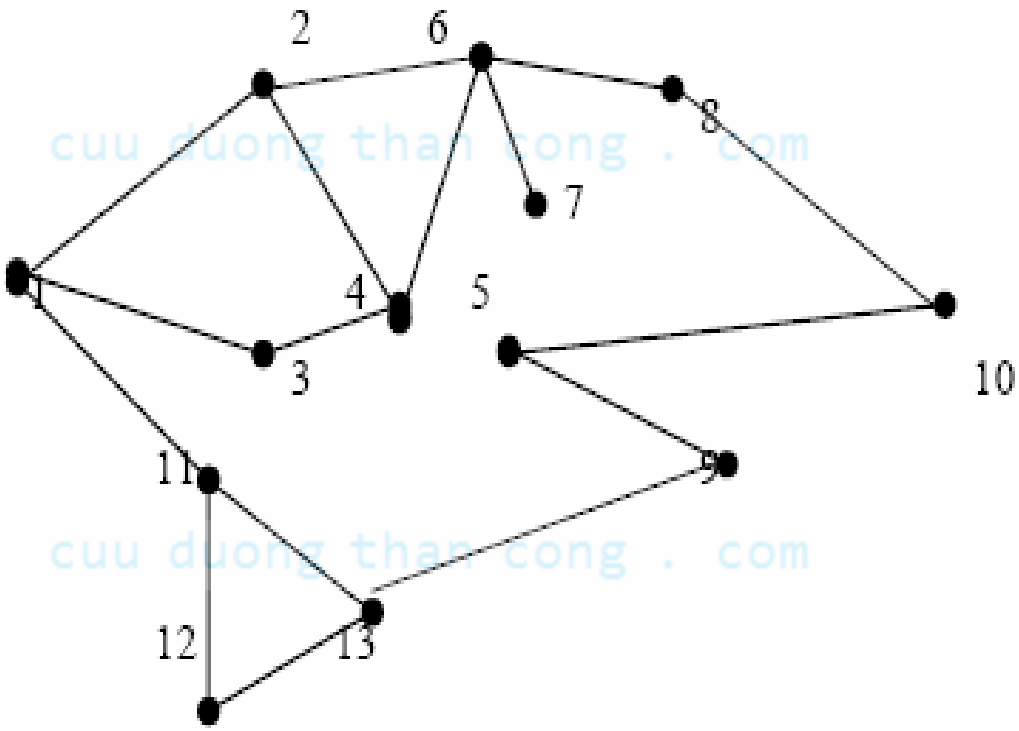
```
void DFS( int v){
  Thăm_Đỉnh(v); chuaxet[v] := FALSE;
  for  ( u ∈ke(v) ) {
if (chuaxet[u] )
                DFS(u);
  }
}
```

- Thủ tục *DFS()* sẽ thăm tất cả các đỉnh cùng thành phần liên thông với *v* mỗi đỉnh đúng một lần. Để đảm bảo duyệt tất cả các đỉnh của đồ thị (có thể có nhiều thành phần liên thông), chúng ta chỉ cần thực hiện duyệt như sau:

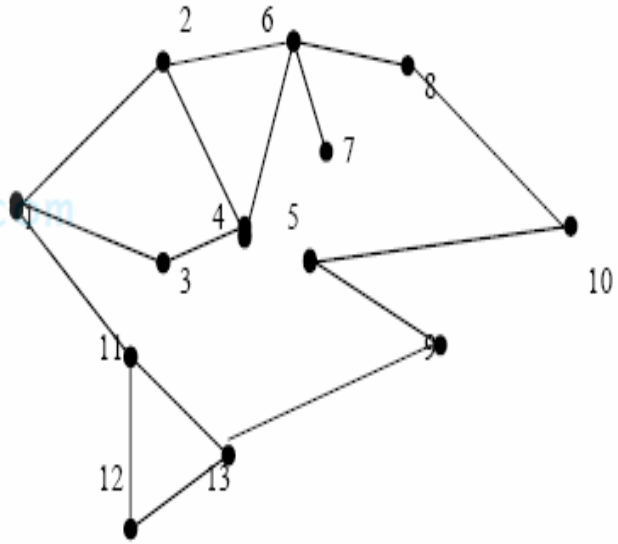
```
{
    for (i=1; i≤n ; i++)
        chuaxet[i] := TRUE; /* thiết lập giá trị ban đầu cho mảng chuaxet[]*/
    for (i=1; i≤n ; i++)
        if (chuaxet[i] )
            DFS( i);
}
```

VD:

duyệt đồ thị sau theo chiều sâu



Đỉnh bắt đầu duyệt	Các đỉnh đã duyệt	Các đỉnh chưa duyệt
DFS(1)	1	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(2)	1, 2	3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(4)	1, 2, 4	3, 5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(3)	1,2,4, 3	5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(6)	1,2,4,3, 6	
DFS(7)	1,2,4,3, 6,7	
DFS(8)	1,2,4,3, 6,7,8	
DFS(10)	1,2,4,3, 6,7,8,10	
DFS(5)	1,2,4,3, 6,7,8,10,5	
DFS(9)	1,2,4,3, 6,7,8,10,5,9	
DFS(13)	1,2,4,3, 6,7,8,10,5,9,13	11, 12
DFS(11)	1,2,4,3, 6,7,8,10,5,9,13,11	12
DFS(11)	1,2,4,3, 6,7,8,10,5,9,13,11,12	ϕ



Kết quả duyệt: 1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11, 12

```
void DFS(int G[][MAX], int n, int v, int chuaxet[]){
        int u;
        printf("%3d",v);chuaxet[v]=FALSE;
        for(u=1; u<=n; u++){
                if(G[v][u]==1 && chuaxet[u])
                        DFS(G,n, u, chuaxet);
        }
}
```

```
void main(void){
    int G[MAX][MAX], n, chuaxet[MAX];
    Init(G, &n);
    for(int i=1; i<=n; i++)
        chuaxet[i]=TRUE;
    printf("\n\n");
                    DFS( G,n, 2, chuaxet);
    getch();
}
```

- *Bước 1 (Khởi tạo):*
- Stack = ∅; Push(Stack, u); <Thăm đỉnh u>; Chuaxet[u] = False;
- *Bước 2 (Lặp):*
- whlie(Stack≠ ∅) {
- s = Pop(Stack);
- for each t∈ Ke(s) do {
- if (Chuaxet[t]){
- <Thăm đỉnh t>; Chuaxet[t] = False;
- Push(Stack, s) ; Push(Stack, t) ; break ;
- }
- }
- }
- *Bước 3 (Trả lại kết quả):*
- Return(Tập đỉnh đã duyệt);

```
// thao tac voi stack
#include<define.h>
typedef struct{
        int top;
        int node[100];
} stack;
int empty(stack *ps){
        if(ps->top==-1) return(1);
        return(0);
}
int viewpop(stack *ps){
Int x=ps->node[ps->top]; return(x)

}
```

```
int full(stack *ps){
        if(ps->top==100-1) return(1);
        return(0);
}
void push(stack *ps,int p){
        ps->top++;
        ps->node[ps->top]=p;
}
int pop(stack *ps){
        return(ps->node[ps->top--]);
}
```

```
void F09_1(void){
        stack s;
        int x,a;
        printf("Moi ban nhap so :");scanf("%d",&x);
        printf("       he co so :");scanf("%d",&a);
        printf("Ket qua :");
        s.top=-1;
        do {
                push(&s,x%a);
                x=x/a;
        } while (x>0);
        while (empty(&s)==0) {
                x=pop(&s);
                printf("%d",x);
        }
        getch();
}
```

Hoặc thuật toán có thể được cài đặt theo stack như sau:

```
❖ void Dtraverse(kmatran a, kvecto &chuaxet, int k, int n)
❖  {stack S[20];int i,h;
❖   push(k);chuaxet[k]=false;
❖ while(!empty())
❖    {h=pop();
❖     printf("%5d",h);
❖     for(i=n;i>=1;i--)
❖      if(chuaxet[i] && a[h][i]) {push(i); chuaxet[i]=0;break;}
❖    }
❖  return;
❖ }
```

Sử dụng để kiểm tra tính liên thông như sau

```
int LienThong(kmatran a,int n)
 {Stack<int> S(20);int i,h;kvecto chuaxet;
  for(i=1;i<=n;i++) chuaxet[i]=true;
  S.push(1);chuaxet[1]=false;
  while(!S.empty())
   {h=S.pop();
    for(i=n;i>=1;i--)
     if(chuaxet[i] && a[h][i]) {S.push(i);chuaxet[i]=0; break;}
   }
  for(i=1;i<=n;i++) if(chuaxet[i]) return(false);
  return(true);
 }
```

Duyệt đồ thị theo chiều rộng

- Để ý rằng, với thuật toán tìm kiếm theo chiều sâu, đỉnh thăm càng muộn sẽ trở thành đỉnh sớm được duyệt xong. Đó là kết quả tất yếu vì các đỉnh thăm được nạp vào stack trong thủ tục đệ qui. Khác với thuật toán tìm kiếm theo chiều sâu, thuật toán tìm kiếm theo chiều rộng thay thế việc sử dụng stack bằng hàng đợi queue. Trong thủ tục này, đỉnh được nạp vào hàng đợi đầu tiên là v, các đỉnh kề với v *(v1, v2, . . ., vk)* được nạp vào queue kế tiếp. Quá trình duyệt tiếp theo được bắt đầu từ các đỉnh còn có mặt trong hàng đợi.
- Để ghi nhận trạng thái duyệt các đỉnh của đồ thị, ta cũng vẫn sử dụng mảng *chuaxet[]* gồm *n* phần tử thiết lập giá trị ban đầu là *TRUE*. Nếu đỉnh *i* của đồ thị đã được duyệt, giá trị *chuaxet[i]* sẽ nhận giá trị *FALSE*. Thuật toán dừng khi hàng đợi rỗng. Thủ tục *BFS* dưới đây thể hiện quá trình thực hiện của thuật toán:

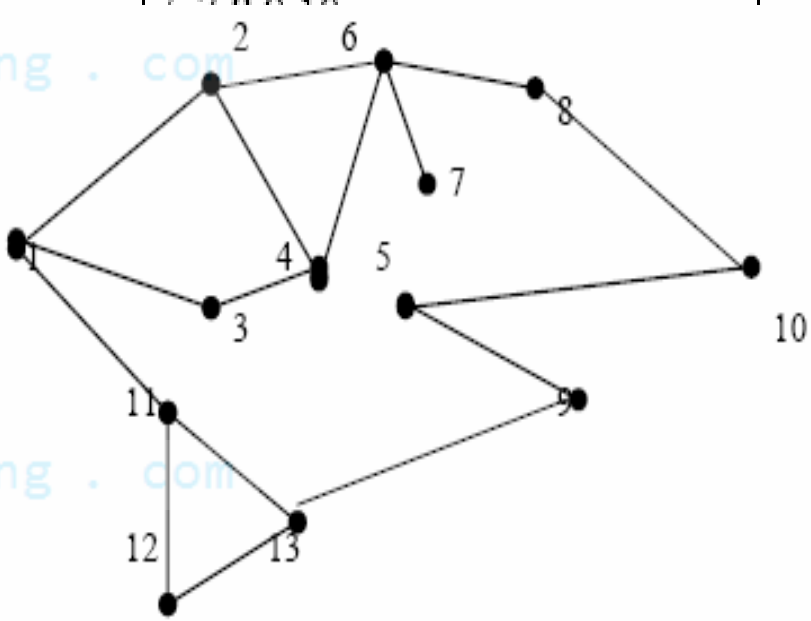
```
void BFS(int u){
        queue = ϕ;
        u <= queue; /*nạp u vào hàng đợi*/
        chuaxet[u] = false;/* đổi trạng thái của u*/
        while (queue ≠ ϕ ) { /* duyệt tới khi nào hàng đợi rỗng*/
                queue<=p; /*lấy p ra từ khỏi hàng đợi*/
                Thăm_Đỉnh(p); /* duyệt xong đỉnh p*/
                for  (v ∈ ke(p) ) { /* đưa các đỉnh v kề với p nhưng chưa
    được xét vào hàng đợi*/
                        if (chuaxet[v] ) {
                                v<= queue; /*đưa v vào hàng đợi*/
                                chuaxet[v] = false;/* đổi trạng thái của v*/
                        }
                }
        } /* end while*/
}/* end BFS*/
```

```
void BFS( int i)
{
        int u, dauQ, cuoiQ, j;
        dauQ=1; cuoiQ=1;QUEUE[cuoiQ]=i;chuaxet[i]=FALSE;
        /* thiết lập hàng đợi với đỉnh đầu là i*/
        while(dauQ<=cuoiQ){
                u=QUEUE[dauQ];
printf("%3d",u);dauQ=dauQ+1; /* duyệt đỉnh đầu hàng đợi*/
                for(j=1; j<=n;j++){
                        if(G[u][j]==1 && chuaxet[j] ){
                                cuoiQ=cuoiQ+1;
                                QUEUE[cuoiQ]=j;
                                chuaxet[j]=FALSE;
                        }
                }
        }
}
```

- Thủ tục *BFS* sẽ thăm tất cả các đỉnh cùng thành phần liên thông với *u*. Để thăm tất cả các đỉnh của đồ thị, chúng ta chỉ cần thực hiện đoạn chương trình dưới đây:

```
{
    for (u=1; u≤n; u++)
        chuaxet[u] = TRUE;
    for (u∈V )
        if (chuaxet[u] )
            BFS(u);
}
```

Các đỉnh đã duyệt	Các đỉnh trong hàng đợi	Các đỉnh còn lại
ϕ	ϕ	1,2,3,4,5,6,7,8,9,10,11,12,13
1	2, 3, 11	4,5,6,7,8,9,10,12,13
1, 2	3, 11, 4, 6	5,7,8,9,10,12,13
1, 2, 3	11, 4, 6	5,7,8,9,10,12,13
1, 2, 3, 11	4, 6, 12, 13	5,7,8,9,10
1, 2, 3, 11, 4	6,12,13	
1, 2, 3, 11, 4, 6	12,13, 7, 8	
1, 2, 3, 11, 4, 6,12	13, 7, 8	
1, 2, 3, 11, 4, 6,12, 13	7, 8, 9	
1, 2, 3, 11, 4, 6,12, 13,7	8, 9	
1, 2, 3, 11, 4, 6,12, 13, 7, 8	9, 10	
1, 2, 3, 11, 4, 6,12, 13, 7, 8, 9	10, 5	
1,2,3,11, 4, 6,12, 13, 7, 8, 9,10	5	ϕ
1,2,3,11,4,6,12,13,7, 8, 9,10, 5	ϕ	ϕ



Số thành phần liên thông của đồ thị

- Cạnh cầu- cạnh khớp-canh cat
- Đỉnh khớp-đỉnh cắt-điểm khớp- dinh tru
- Đếm số thành phần liên thông của đồ thị

CHƯƠNG 4. CHU TRÌNH EULER VÀ CHU TRÌNH HAMILTON

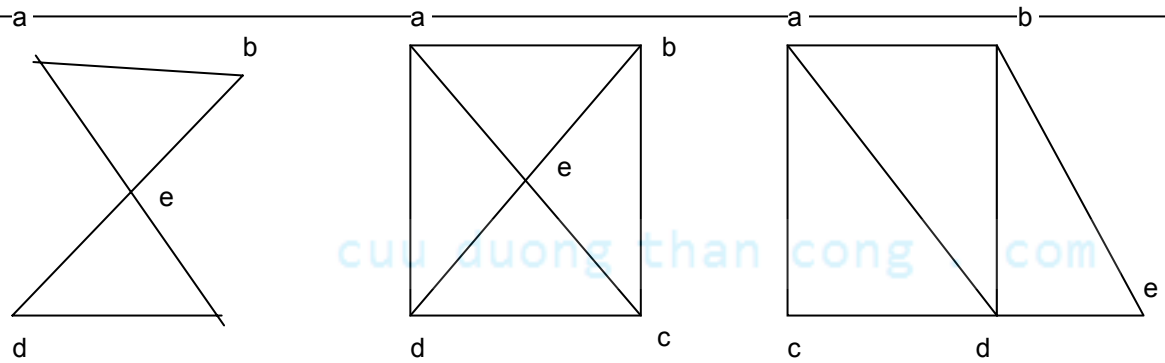
- 1.Khái niệm về đường đi và chu trình Euler
- 2.Các điều kiện cần và đủ cho chu trình và đường đi Euler
- 3. Thuật toán tìm chu trình Euler
- 4. Đường đi và chu trình Hamilton

1. Mở đầu

Định nghĩa

- Chu trình đơn chứa tất cả các cạnh của đồ thị được gọi là ***chu trình Euler***.
- ***Đường đi Euler*** trong G là đường đi đơn chứa mọi cạnh của G.

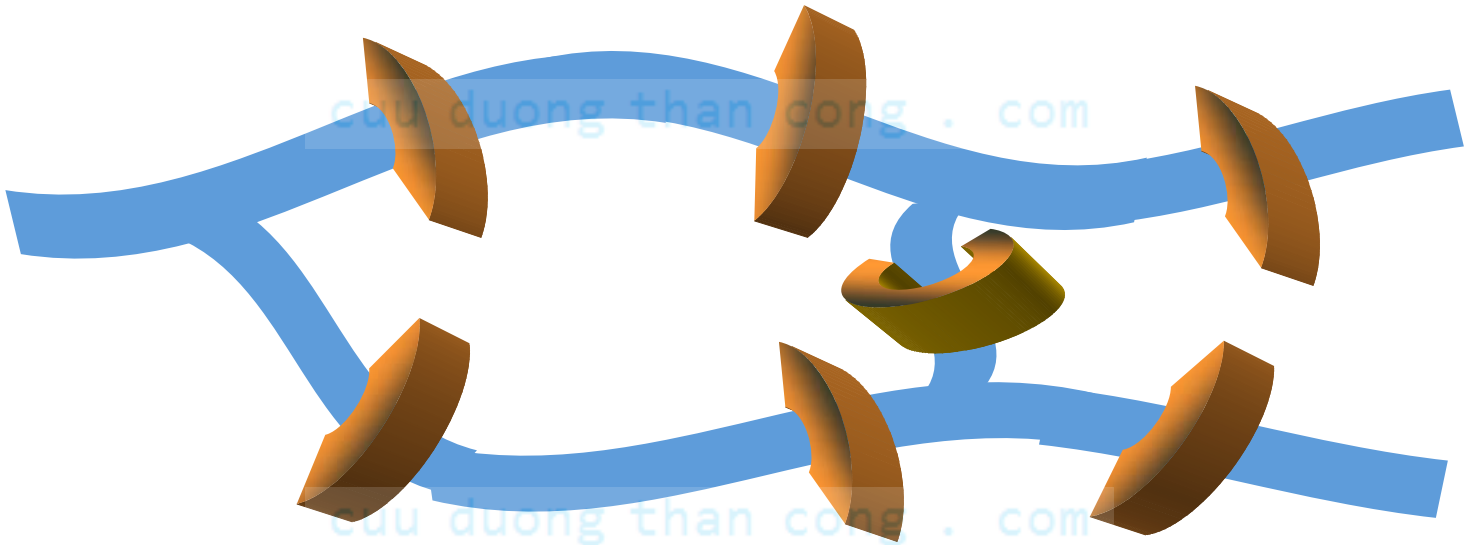
vd



- Đồ thị G1 có chu trình Euler, thí dụ a,e,c,d,e,b,a. Cả hai đồ thị G2 và G3 đều không có chu trình Euler. Tuy nhiên G3 có đường đi Euler, cụ thể là a,c,d,e,b,d,a,b

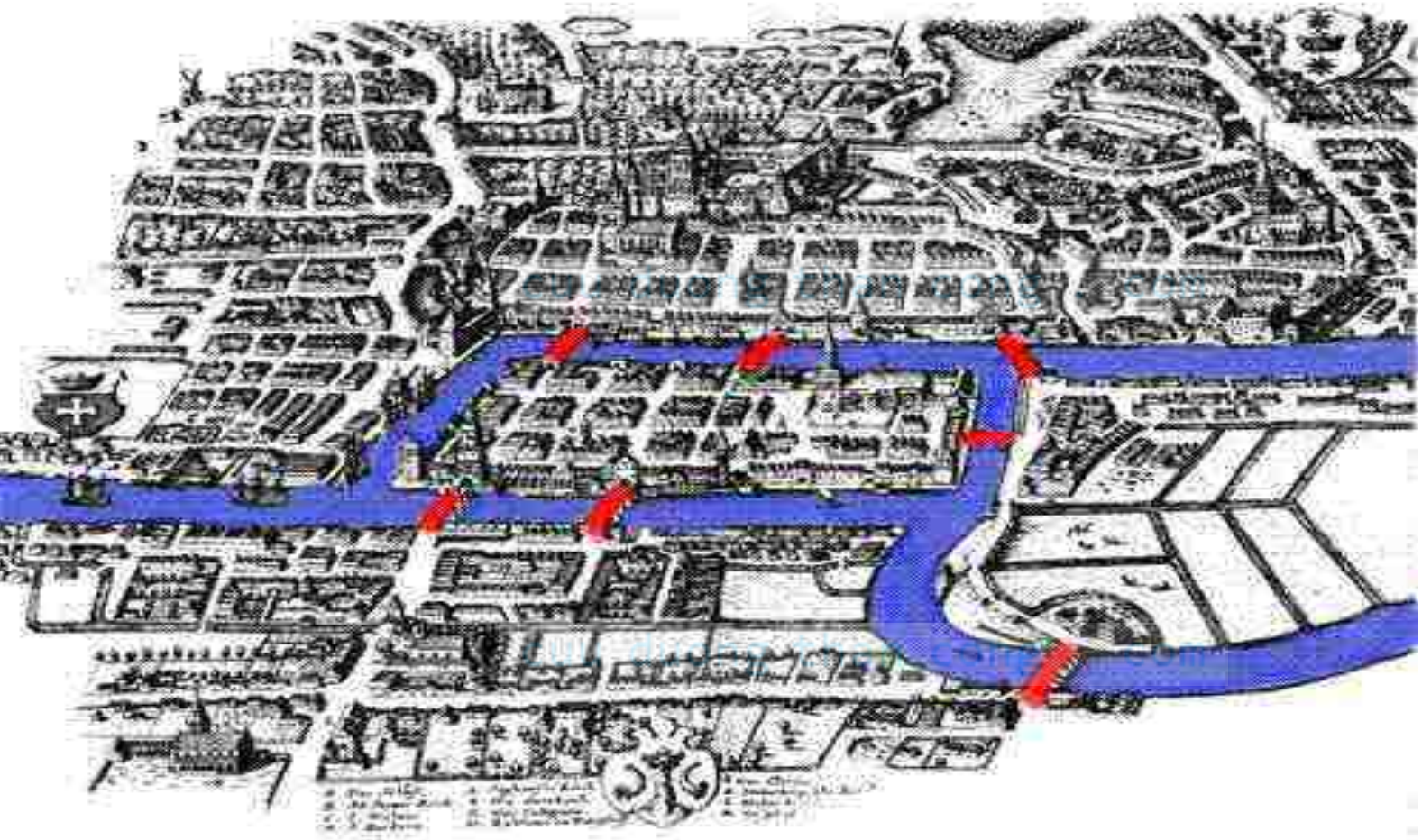
Đường đi Euler

Bài toán. Thị trấn Königsberg chia thành 4 phần bởi các nhánh của dòng sông Pregel

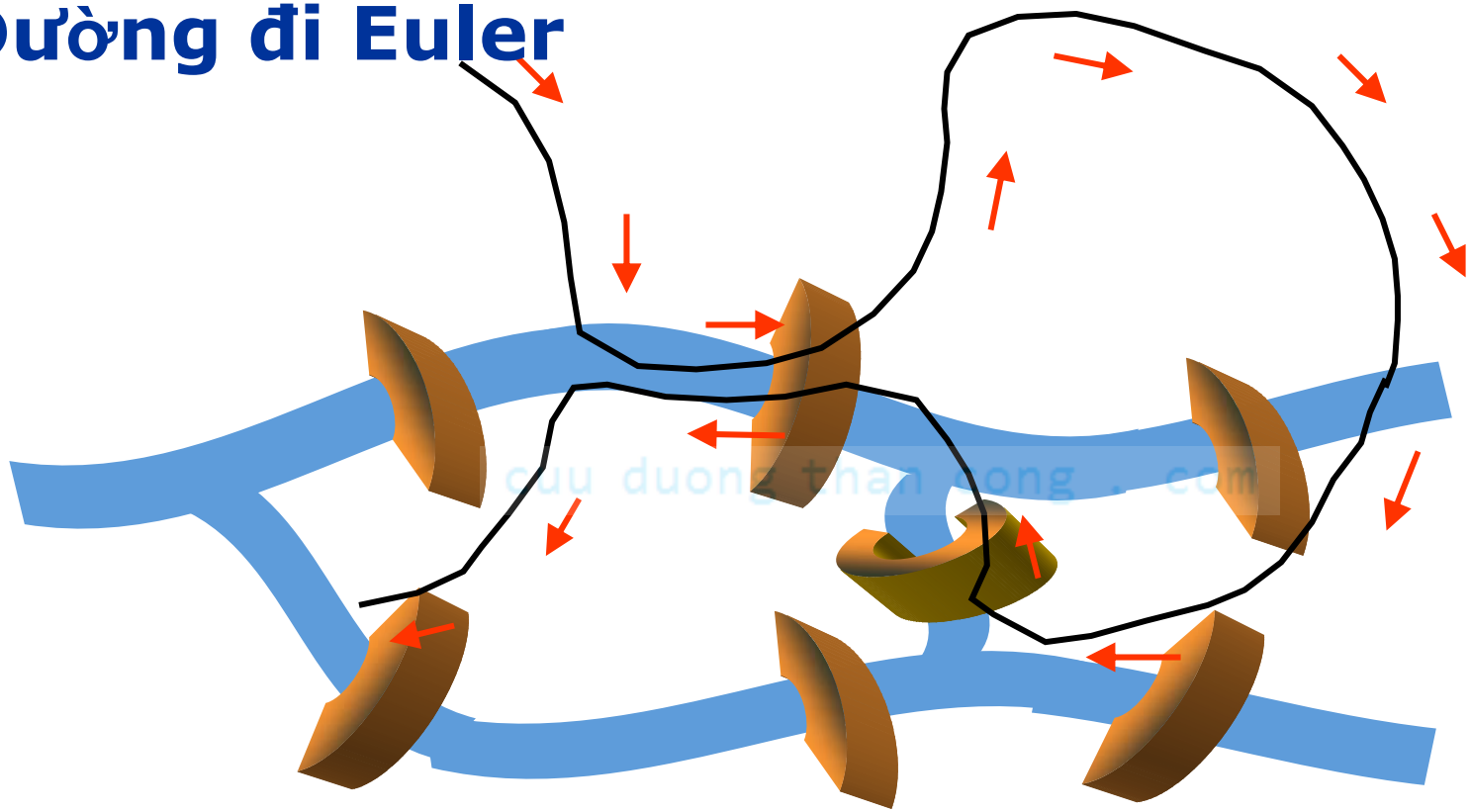


Bốn phần này được nối kết bởi 7 cây cầu

Đường đi Euler



Đường đi Euler



Câu hỏi. Có thể đi qua bảy cây cầu mà không có cây cầu nào đi quá 1 lần

1. Mở đầu

- ***Chu trình Euler là một đường đi Euler*** nhưng ngược lại thì chưa chắc đã đúng (ví dụ trên đồ thị G3 thì a,c,d,e,b,d,a,b là đường Euler, nhưng không phải là chu trình Euler).
- Chu trình Euler có thể đi qua một đỉnh hai lần (ví dụ chu trình a,e,c,d,e,b,a trên đồ thị G1) và có thể không đi qua một số đỉnh nào đó nếu các đỉnh này là các ***đỉnh cô lập***, tức là các đỉnh có bậc bằng 0

- Chu trình Euler chỉ liên quan đến các cạnh của đồ thị, vì vậy ta có thể thêm một số bất kỳ các đỉnh có bậc 0 (đỉnh cô lập) vào đồ thị G thì chu trình Euler của G vẫn không thay đổi.

2.Các điều kiện cần và đủ cho chu trình và đường đi Euler

Định lý sau đây cho ta điều kiện cần và đủ để một đa đồ thị có chu trình Euler:

- **Đồ thị vô hướng**

  Định lý 1 . Một đa đồ thị không có điểm cô lập có chu trình Euler nếu và chỉ nếu đồ thị là liên thông và mỗi đỉnh của nó đều có bậc chẵn.

- Đối với đồ thị có hướng ta có định lý sau:

  Định lý 2. Một đa đồ thị có hướng không có đỉnh cô lập tồn tại chu trình Euler nếu và chỉ nếu đồ thị là liên thông yếu đồng thời bậc-vào và bậc-ra của mỗi đỉnh là bằng nhau.

3.Thuật toán tìm chu trình Euler

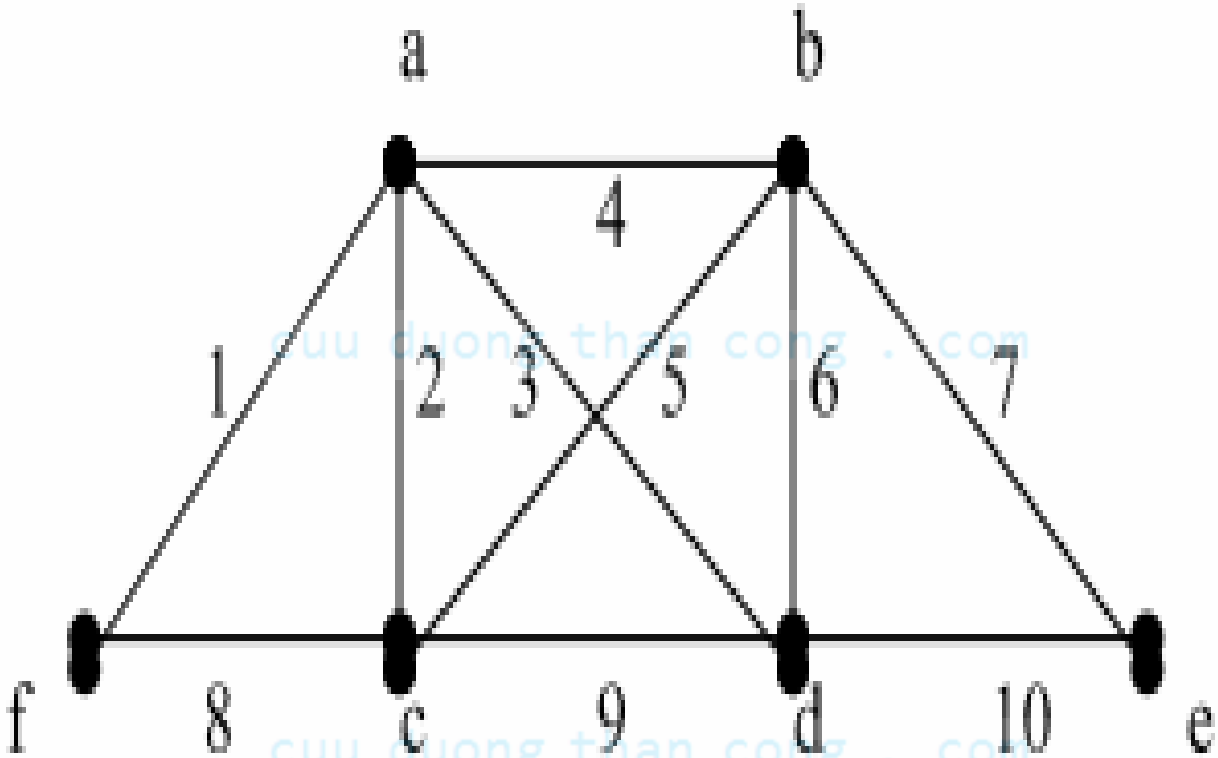
- Bài toán: Cho đồ thị G = (V,E). Hãy tìm chu trình Euler của đồ thị G nếu có.
- Thuật toán:
- Bước 1: Kiểm tra tính liên thông của ĐT

  Nếu G liên thông thì chuyển sang bước 2,

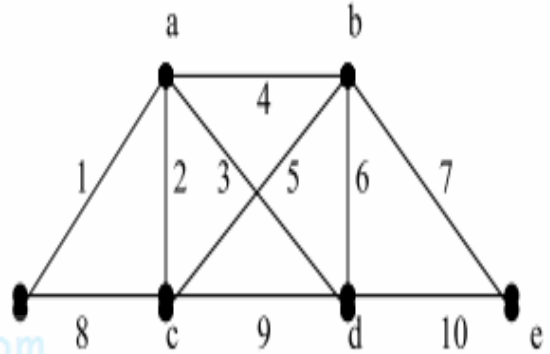
  ngược lại thì thông báo là chúng ta chỉ xét đồ thị liên thông và dừng thuật toán. (Vì có thể thấy rằng nếu đồ thị có một thành phần liên thông còn phần còn lại là các điểm cô lập thì vẫn có thể có chu trình Euler, vì chu trình Euler chỉ quan tâm đến việc đi qua các cạnh mà không quan tâm đến việc đi qua các đỉnh).
- Bước 2: Kiểm tra xem ĐK cần và đủ của chu trình, đường đi euler

Bước 3: Xây dựng thuật toán tìm chu trình Euler đơn trong G sao cho tất cả các cạnh của G đều có chu trình đơn đi qua và chỉ đi qua một lần bằng cách sau:

- Tạo mảng CE để ghi đường đi và một Stack để xếp các đỉnh sẽ xét. Đầu tiên xếp một đỉnh u nào đó của đồ thị vào Stack.
- Xét đỉnh v nằm trên cùng của Stack và thực hiện:
- Nếu v là đỉnh cô lập thì lấy v ra khỏi Stack và đưa vào CE.
- Nếu v là liên thông với đỉnh x thì đưa x vào ngăn xếp sau đó xóa cạnh (v,x);
- Quay lại bước 3. cho tới khi ngăn xếp rỗng thì dừng. Kết quả đường đi Euler được chứa trong CE theo thứ tự ngược lại.


a
b
4
1
2
3
5
6
7
f
8
c
9
d
10
e

Bước	Giá trị trong stack	Giá trị trong C	
1	F		
2	f, a		
3	f, a, c		
4	f,a,c,f		
5	f, a, c	f	3, 4, 5, 6, 7, 9, 10
6	f, a, c, b	f	3, 4, 6, 7, 9, 10
7	f, a, c, b, d	f	3, 4, 7, 9, 10
8	f, a, c, b, d,c	f	3, 4, 7, 10
9	f, a, c, b, d	f, c	3, 4, 7, 10
10	f, a, c, b, d, e	f, c	3, 4, 7
11	f, a, c, b, d, e, b	f, c	3, 4



12	f, a, c, b, d, e, b, a	f, c	3
13	f, a, c, b, d, e, b, a, d	f, c	
14	f, a, c, b, d, e, b, a	f, c, d	
15	f, a, c, b, d, e, b	f,c,d,a	
16	f, a, c, b, d, e	f,c,d,a,b	
17	f, a, c, b, d	f,c,d,a,b,e	
18	f, a, c, b	f,c,d,a,b,e,d	
19	f, a, c	f,c,d,a,b,e,d,b	
20	f, a	f,c,d,a,b,e,d,b,c	
21	F	f,c,d,a,b,e,d,b,c,a	
22		F,c,d,a,b,e,d,b,c,a,f	

```
void CTEuler(kmatran a, int n, int k kvecto CT, int &nCT)
 {Stack S;kmatran B;int i,j,h,t;
 SetEqual(B,a,n);//Dat b=a de khoi phuc lai gia tri sau nay
  push(&S,k);//Dua dinh bat ky vao Stack, o day ta chon dinh 1
  nCT=0;//Ban dau chu trinh chua co phan tu nao
  while(empty(&S))
   {h=viewtop(&S);i=1;
    while(i<=n && a[h][i]==0) i++;//Tim i dau tien de a[h][i]#0
    if(i==n+1) //h da la dinh co lap, dua h vao chu trinh CT
     {nCT++;CT[nCT]=h; pop(&S,h);}//Lay dinh co lap ra khoi Stack
     else
     {push(&S,i);a[h][i]--;a[i][h]--;}//Loai canh (i,h) khoi do thi
   }
  //Dao lai chu trinh cho hop ly hon
  i=1;j=nCT;
  while(i<j)
   {t=CT[i];CT[i]=CT[j];CT[j]=t;i++;j--;}
  SetEqual(a,B,n);
```

4.Đường đi và Chu trình Hamilton

- Đường đi và chu trình Euler chỉ liên quan đến các cạnh của đồ thị. Tuy nhiên câu hỏi tương tự có thể đặt ra đối với các đỉnh. Thí dụ trong một mạng lưới giao thông ta lại quan tâm đến việc xuất phát từ một thành phố, liệu ta có thể đến thăm tất cả các thành phố khác. mỗi thành phố thăm đúng một lần và cuối cùng trở lại thành phố xuất phát?

ĐỊNH NGHĨA

Cho đồ thị G = (V,E).

- Đường đi R được gọi là đường đi Hamilton nếu nó đi qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần.
- Chu trình C được gọi là chu trình Hamilton nếu nó xuất phát từ một đỉnh v nào đó, đi qua tất cả các đỉnh, mỗi đỉnh đúng một lần rồi trở lại điểm v. Đồ thị có chu trình Hamilton được gọi là đồ thị Hamilton.
- Đồ thị chứa đường đi Hamilton được gọi là đồ thị nửa Hamilton.
- Trong các ví dụ sau chúng ta sẽ thấy là đồ thị Hamilton là nửa Hamilton nhưng ngược lại không luôn luôn đúng.

- **Định lý 3**. Giả sử G là một đơn đồ thị liên thông với n đỉnh trong đó $n \geq 3$.
- **Định lý 4**. Giả sử G = (V,E) là đồ thị có hướng đầy đủ. Khi đó trong đồ thị G tồn tại đường Hamilton.

Thuật toán tìm chu trình Hamilton

```
void Hamilton( int k) {
/* Liệt kê các chu trình Hamilton của đồ thị bằng cách
phát triển dãy đỉnh
(X[1], X[2], . . ., X[k-1] ) của đồ thị G = (V, E) */
      for y ∈ Ke(X[k-1]) {
            if (k==n+1) and (y == v0) then
Ghinhan(X[1], X[2], . . ., X[n], v0);
            else {if  chuaxet[y]
                  {X[k]=y; chuaxet[y] = false;
                  Hamilton(k+1);
                  chuaxet[y] = true;}
            }
      }
}
```

- }

```
#define MAX 50
#define TRUE 1
#define FALSE 0
int A[MAX][MAX], C[MAX], B[MAX];
//int B[MAX];
int i, d,n;
void Init(){
int i, j;FILE *fp;
fp= fopen("DDHMTON.IN", "r");
        if(fp==NULL){
                printf("\n Khong co file input");
                getch(); return;
        }
        fscanf(fp,"%d",&n);
printf("\n So dinh do thi:%d", n);
printf("\n Ma tran ke:");
for(i=1; i<=n; i++){
                printf("\n");
                for(j=1; j<=n; j++){
                        fscanf(fp, "%d", &A[i][j]);
                        printf("%3d", A[i][j]);
                }
}
fclose(fp);
for (i=1; i<=n;i++)
                C[i]=0;
```

```
void Result(){
int i;
printf("\n ");
printf("%3d ", B[1]);

for(i=n; i>0; i--)
printf("%3d", B[i]);
        d++;
}
```

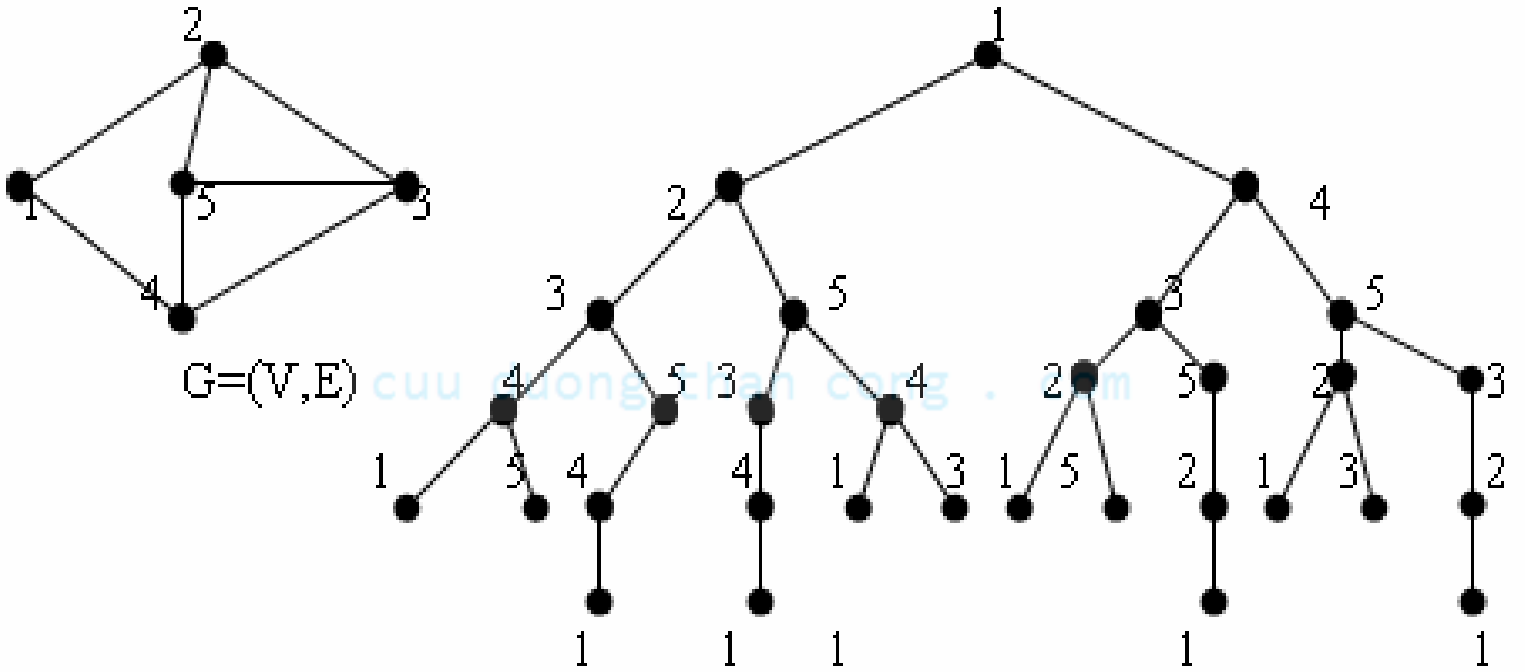
```
void Hamilton(int A[][MAX],int *B, int *C, int i){
        for(int j=1; j<=n; j++){
          if(A[B[i-1]][j]==1) {
                        if((i==n+1)&& (j==1))
                                Result() ;
                        else
                        if (C[j]==0 ){ B[i]=j; C[j]=1;
                                Hamilton(A,B,C,i+1);
                                C[j]=0; } } }
                }

 main(void){
        B[1]=1; int i=2; d=0;

   Init();
   Hamilton(A,B,C,i);
        if(d==0)
                printf("\n Khong co duong di Hamilton");

        getch();
}
```

- Chương trình chính được thể hiện như sau:
- *{*
- *for (v∈V) chuaxet[v] = true; /*thiết lập trạng thái các đỉnh*/*
- *X[1] = v0; (*v0 là một đỉnh nào đó của đồ thị*)*
- *chuaxet[v0] = false;*
- *Hamilton(2);*
- *}*



Hình 6.9. Cây tìm kiếm chu trình Hamilton.

```
void Result(){
    int i;
    printf("\n ");
    printf("%3d ", B[1]);

    for(i=n; i>0; i--)
        printf("%3d", B[i]);
    d++;
}
```

```
void Hamilton(int A[][MAX],int *B, int *C, int i){
        for(int j=1; j<=n; j++){
          if(A[B[i-1]][j]==1) {
                    if((i==n+1)&& (j==1))
                              Result() ;
                         else
                    if (C[j]==0 ){ B[i]=j; C[j]=1;
                                        Hamilton(A,B,C,i+1);
                                        C[j]=0; }  }  }
                 }
```

```
main(void){
        B[1]=1; int i=2; d=0;

    Init();
    Hamilton(A,B,C,i);
        if(d==0)
                printf("\n Khong co duong di Hamilton");

        getch();
}
}
```

```
void Result(void){
        int i;
        printf("\n ");
        for(i=n; i>0; i--)
                printf("%3d", B[i]);
        d++;
}
void Hamilton(int *B, int *C, int i){
        int j, k;
        for(j=1; j<=n; j++){
                if(A[B[i-1]][j]==1 && C[j]==0){
                        B[i]=j; C[j]=1;
                        if(i<n) Hamilton(B, C, i+1);
                        else  Result();
                        C[j]=1;
                }
        }
}
```

```
void main(void){
        B[0]=1; i=1;d=0;
    Init();
        Hamilton(B,C,i);
        if(d==0)
                printf("\n Khong co duong di Hamilton");
        getch();
}
```

CHƯƠNG 5. Tìm đường đi ngắn nhất

- 1.Mở đầu
- 2.Thuật toán Dijkstra xác định các đường đi ngắn nhất từ một đỉnh đến các đỉnh còn lại
- 3. Thuật toán Floyd xác định đường đi ngắn nhất giữa mọi cặp đỉnh
- 4. Đồ thị phẳng
- 5.Tô màu đồ thị

1. MỞ ĐẦU

- Trọng số của canh:là giá trị đươc ghi trên mỗi cạnh của đồ thị.
- Đồ thị có trọng số là đồ thị mà trên đó mỗi cạnh có 1 trọng số
- Bài toán tìm đường đi ngắn nhất giữa hai đỉnh u và v chỉ có nghĩa khi thực sự có đường đi từ đỉnh u tới đỉnh v. Vậy ở đây ta sẽ giả thiết đồ thị đang khảo sát là đơn đồ thị liên thông. Phải:
  - Tìm 1 đường đi (u->v) có tổng trọng số trên các cạnh mà đường đi đó phải đi qua đạt giá trị min

B..Thuật toán Dijkstra xác định các đường đi ngắn nhất từ một đỉnh đến các đỉnh còn lại

- Giả sử ta xét đồ thị vô hướng G = (V,E), có trọng số.Ký hiệu trọng số của cạnh giữa hai đỉnh i và j là wij, đặt wii = 0 và wij =∞(giữa I và j không có đường đi trực tiếp), còn lại Wi,j khác 0 và khác ∞

- Thay vì tìm chỉ một khoảng cách bé nhất từ a đến b, ta tìm một số khoảng cách bé nhất từ a đến các đỉnh khác trong đó có b.
- Kí hiệu: $w(x,y); D_i(y); p_i(y)$

Thuật toán Dijkstra

B0:Đầu tiên :Đưa x0=a vào tập S

- Tính khoảng cách trực tiếp từ các đỉnh $y \in V \backslash a$ tới đỉnh đầu a là $D_0(y)=w(a,y)$. Tìm đỉnh có khoảng cách nhỏ nhất đặt là x1, tức là: $D_0(x1)=\min\{D_0(y)\}$.

  Đặt (đánh dấu) D(x1)=D0(x1);

  p0(y)=a nếu w(a,y)!=vôcùng

  =-1nếu w(a,y)=vôcùng

B1: Đưa x1 vào tập S là tập các đỉnh đã được gán nhãn s={a,x1).

- Tính lại các khoảng cách từ đỉnh v1=a tới các đỉnh y còn lại $\in$ V\S có thể thông qua x1 hoặc đi trực tiếp: $D_1(y)=\min(D_0(y),D(x1)+w(x1,y)$. Với y $\in$ V\S

  Lấy đỉnh có khoảng cách nhỏ nhất đăt là x2. Tức là D1(x2)=min{D1(y) với y $\in$ V\S).

  Đặt D(x2)=D1(x2).

$$P_1(y) = \begin{cases} P_0(y) & khi \quad D_1(y) = D_0(y) \\ x_1 & khi \quad D_1(y) = D_0(x_1) + w(x_1, y) \end{cases}$$

.....

Bk:Đưa xk vào S=>S={a,x1,...,xk)

- Tính lại các khoảng cách từ đỉnh v1=a tới các đỉnh y còn lại$\in$ V\S có thể thông qua xk hoặc đi trực tiếp từ a :$D_k(y)=\min(D_{k-1}(y), D(x_k)+w(x_k,y)$. Với y $\in$ V\S
- Xác định x_{k+1} sao cho $Dk(x_{k+1}) = \min \{D_k(y) \mid y \in V\S\}$.Đặt(đánh dấu) $D(x_{k+1})=Dk(x_{k+1})$

.....

$$P_k(y) = \begin{cases} P_{k-1}(y) & khi \quad D_k(y) = D_{k-1}(y) \\ x_k & khi \quad D_k(y) = D_{k-1}(x_k) + w(x_k, y) \end{cases}$$

Cứ như thế cho tới khi đỉnh cuối b xuất hiện trong S thì dừng thuật toán lại

Lúc này ta có đường đi ngắn nhất là :

a, ..., P(P(P(b))), P(P(b)), P(b), b

và độ dài đường đi này là D(b)

3.Thuật toán Floyd xác định đường đi ngắn nhất giữa mọi cặp đỉnh

- Với những giả thiết như trong thuật toán Dijkstra, ta xét bài toán tìm đường đi ngắn nhất giữa mọi cặp đỉnh. Chúng ta dùng ma trận D[i,j] để lưu độ dài đường đi ngắn nhất giữa 2 cặp đỉnh i và j và ma trận P[i,j] để lưu đường đi ngắn nhất từ i đến j. Thí dụ P[i,j] = k có nghĩa là nút k đứng trước nút j trong đường đi ngắn nhất từ i đến j. Phương pháp này được xây dựng bởi Floyd trên cơ sở sử dụng những ý tưởng của phương pháp Warshall trong xác định bao đóng chuyển tiếp. Cụ thể ta thực hiện các bước sau:

Thuật toán

- Ta gọi Dk là ma trận chứa khoảng cách ngắn nhất từ đỉnh i đến đỉnh j trong các đường đi chỉ chứa các đỉnh trong tập {1,2,...,k}. Ta có thể thực hiện theo từng bước như sau:

- Bước 0: Khởi đầu ta đặt D0[i,j] = W[i,j] là đường đi trực tiếp, không qua nút nào cả.
- Bước 1: D1[i,j] = min (D0[i,j], D0[i,1] + D0[1,j])
- Bước k: Dk[i,j] = min (Dk-1[i,j], Dk-1[i,k] + Dk-1[k,j])
- ...
- Đến bước n ta sẽ được ma trận D cần tìm. Trong mỗi bước ta ghi lại vết của đường đi trong ma trận P[i,j]

Giả mã

```
void floyd(int d[][n],int p[][n])
 {int k,i,j;
  //Khoi tao ma tran D va P
  for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
   {d[i][j] = w[i][j];p[i][j]=-1;}/*p[i][j]=-1 nghia la truc tiep
```

```
//Bat dau vong lap
khong qua dinh nao*/
 for(k=1;k<=n;k++)
 for(i=1;i<=n;i++)
  if(d[i][k]>0 && d[i][k]<vocung)//Co duong di tu i den k
   for(j=1;j<=n;j++)
    if(d[k][j]>0 && d[k][j]<vocung)//Co duong di tu k den j
      if(d[i][j] !=0 && d[i][j]>d[i][k] + d[k][j])

   {d[i][j] = d[i][k] + d[k][j];p[i][j]=k;}
 };
```

Chương 6. Đồ thị phẳng

cuu duong than cong . com

GIẢNG VIÊN: TH.S.PHAN THỊ HÀ
BỘ MÔN: CÔNG NGHỆ PHẦN MỀM

ĐỒ THỊ PHẲNG

- Bài toán ba biệt thự và ba nhà máy
- Đồ thị phẳng
- Các điều kiện cho tính phẳng của đồ thị
- Sắc số của đồ thị phẳng

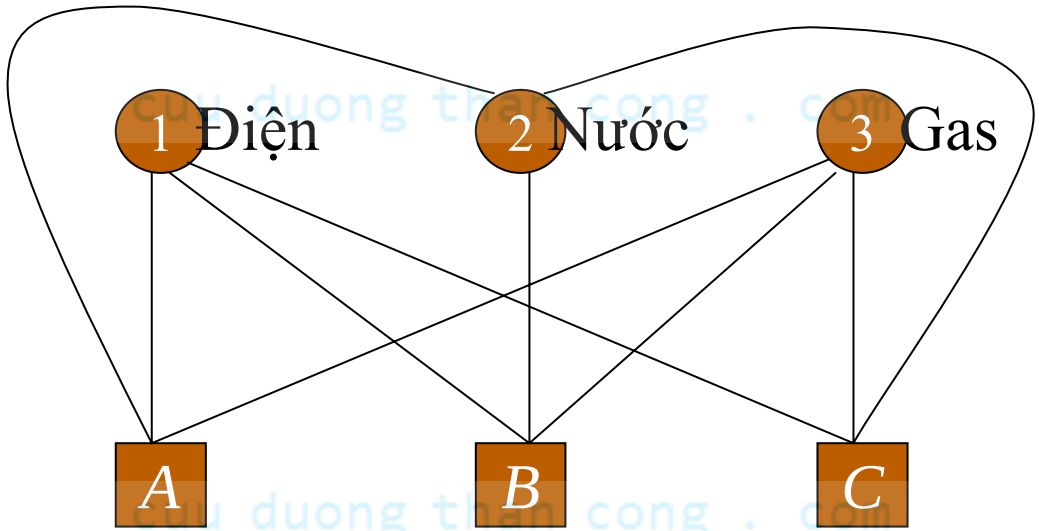
BÀI TOÁN BA BIỆT THỰ VÀ BA NHÀ MÁY

- Bài toán: Trong một thị trấn có ba biệt thự và ba nhà máy cung cấp: điện, nước và khí đốt.

  Mỗi biệt thự đều muốn mắc đường cáp điện ngầm, đường ống cấp nước, đường ống cấp khí đốt riêng từ nhà mình đến ba nhà máy mà không gặp đường ống của các biệt thự khác.

  Hỏi rằng có làm được những đường đi như thế hay không?

BÀI TOÁN BA BIỆT THỰ VÀ BA NHÀ MÁY (tiếp)



ĐN ĐỒ THỊ PHẲNG

❖ ***Định nghĩa 1***

Đa đồ thị vô hướng G được gọi là đồ thị phẳng nếu có thể biểu diễn nó trên mặt phẳng sao cho không có hai cạnh nào cắt nhau, trừ tại đỉnh.

- *Diện hữu hạn* của một đồ thị phẳng là một miền kín của mặt phẳng được giới hạn bằng các cạnh của đồ thị sao cho có thể nối hai điểm bất kỳ thuộc diện này bằng một nét liền mà không cắt một cạnh nào.

- Đồ thị còn có một *diện vô hạn*, đó là phần bù trên mặt phẳng của hợp các diện hữu hạn.

ĐỒ THỊ PHẲNG (tiếp)

❖ ***Định lý 1:*** Số diện hữu hạn của một đa đồ thị phẳng G bằng chu số của đồ thị này.

Chứng minh: Quy nạp theo số diện hữu hạn h của G.

- $h = 1$: chỉ có một chu trình đơn duy nhất, đó chính là biên của diện này. Suy ra chu số bằng 1.
- $(h\text{-}1) \Rightarrow (h)$: Giả sử đồ thị phẳng G với n đỉnh, m cạnh và p mảng liên thông có h diện.

ĐỒ THỊ PHẲNG (tiếp)

Chứng minh định lý:

Lập đồ thị G' từ G bằng cách bớt đi cạnh *e* nào đó trên biên của một diện để số diện hữu hạn bớt đi 1. Khi đó, G' có *h*-1 diện.

Theo giả thiết quy nạp, c(G') = *h*-1 = (*m* - 1) - *n* + *p* (*p* không đổi vì chỉ bớt đi một cạnh trên chu trình).

Suy ra, số diện hữu hạn của G là:

$$h = m - n + p = c(G).$$ □□ □

ĐỒ THỊ PHẲNG (tiếp)

❖ ***Hệ quả 1***

Nếu đa đồ thị phẳng G có n đỉnh, m cạnh, p mảng liên thông và h diện thì: $n - m + h = p + 1$

(công thức Euler tổng quát)

Chứng minh:

Số diện của đồ thị phẳng bằng số diện hữu hạn cộng thêm 1 (diện vô hạn), bằng chu số cộng 1. Vậy thì,

$h = m - n + p + 1$.

Do đó, $n - m + h = p + 1$. □

ĐỒ THỊ PHẲNG (tiếp)

❖ ***Hệ quả 2***

Trong một đơn đồ thị phẳng có ít nhất một đỉnh có bậc không quá 5.

***Chứng minh*:**

Không mất tính tổng quát có thể giả thiết rằng đơn đồ thị là liên thông.

Trong đơn đồ thị phẳng mỗi diện hữu hạn được giới hạn bởi ít nhất 3 cạnh, mà mỗi cạnh thuộc nhiều nhất là hai diện nên: $3h \leq 2m \Rightarrow h \leq 2m/3$.

ĐỒ THỊ PHẲNG (tiếp)

Phản chứng: Giả sử mọi đỉnh của đồ thị G đều có bậc ít nhất là 6.

Khi đó, tổng tất cả các bậc của các đỉnh của trong G = $2m \geq 6n$.

Theo công thức Euler thì: $n - m + h = 1 + p = 2$

Ta có: . Suy ra điều vô lý. □

CÁC ĐIỀU KIỆN CHO TÍNH PHẲNG CỦA ĐỒ THỊ

❖ ***Định lý 2***

Giả sử G là một đồ thị và G’ là đồ thị con của nó.

1. Đồ thị G phẳng thì G’ cũng phẳng.
2. Đồ thị G’ không phẳng thì G cũng không phẳng.

CÁC ĐIỀU KIỆN CHO TÍNH PHẲNG CỦA ĐỒ THỊ (tiếp)

Ký hiệu: δ là độ dài của chu trình ngắn nhất hoặc là số cạnh của đồ thị G nếu không có chu trình.

Số δ được gọi là *đai* của đồ thị.

❖ ***Định lý 3***

Nếu G là đồ thị phẳng *n* đỉnh và đai của nó là $\delta \geq 3$ thì: $m \leq \delta(n-2)/(\delta-2)$.

Chứng minh: Do $h.\delta \leq 2m$ nên theo công thức Euler: $\delta(m - n + 2) \leq 2m$.

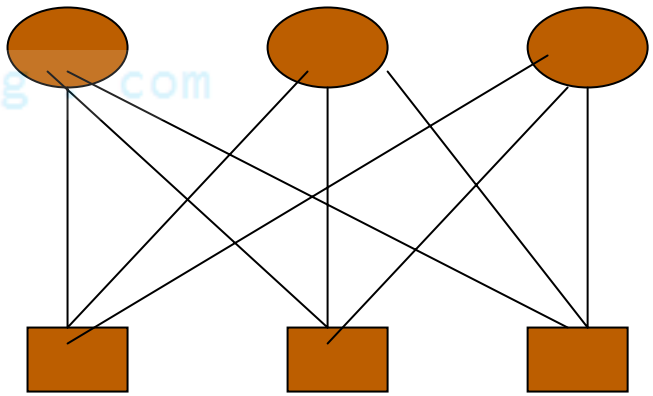
Suy ra điều phải chứng minh. □

VÍ DỤ.1

Bài toán ba biệt thự và ba nhà máy:

Đai của đồ thị trên là $\delta = 4$.

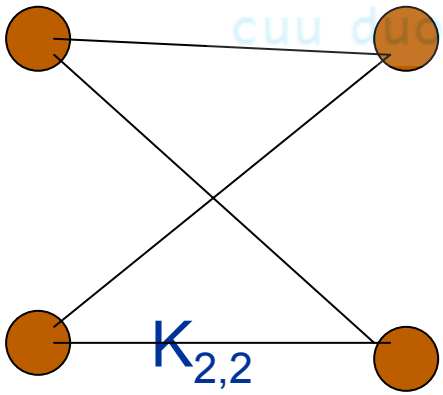
Vậy thì: $m = 9 > 4.(6-2)/(4-2) = 8$

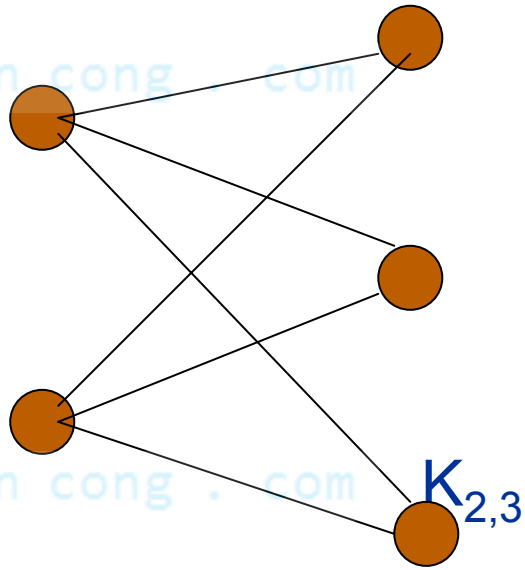


Theo Định lý 5, đồ thị trên không phẳng.

CÁC ĐIỀU KIỆN CHO TÍNH PHẲNG CỦA ĐỒ THỊ (tiếp)

Đồ thị hai phần đầy đủ $K_{m,n}$ là một đơn đồ thị có $m+n$ đỉnh gồm m đỉnh "bên trái" và n đỉnh "bên phải" sao cho mỗi đỉnh "bên trái" đều kề với mọi đỉnh "bên phải".





CÁC ĐIỀU KIỆN CHO TÍNH PHẲNG CỦA ĐỒ THỊ (tiếp)

❖ ***Hệ quả .3***

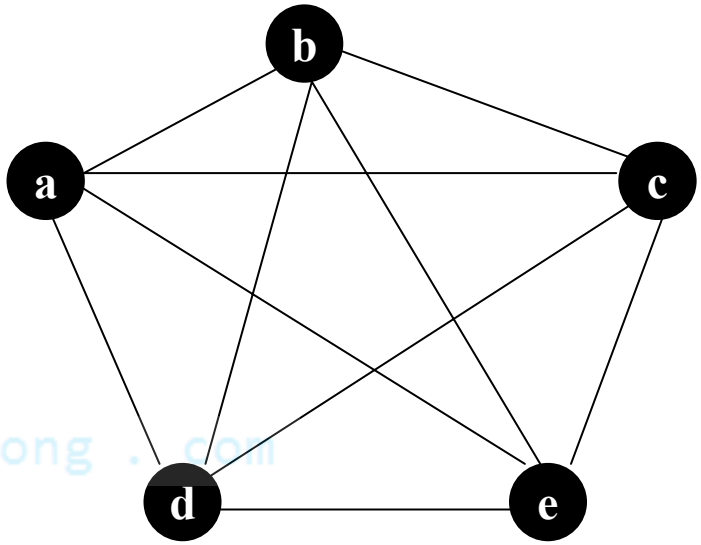
Đồ thị hai phần đầy đủ $K_{m,n}$ là đồ thị phẳng khi và chỉ khi $m \leq 2$ hoặc $n \leq 2$.

VÍ DỤ 2

Đồ thị đầy đủ 5 đỉnh:

Đồ thị có đai $\delta = 3$. Vậy

$m = 10 > 3.(5-2)/(3-2) = 9$



Đồ thị K_5 không phẳng. Từ đó suy ra, đồ thị đầy đủ K_n với $n \geq 5$ là không phẳng.

CÁC ĐIỀU KIỆN CHO TÍNH PHẲNG CỦA ĐỒ THỊ (tiếp)

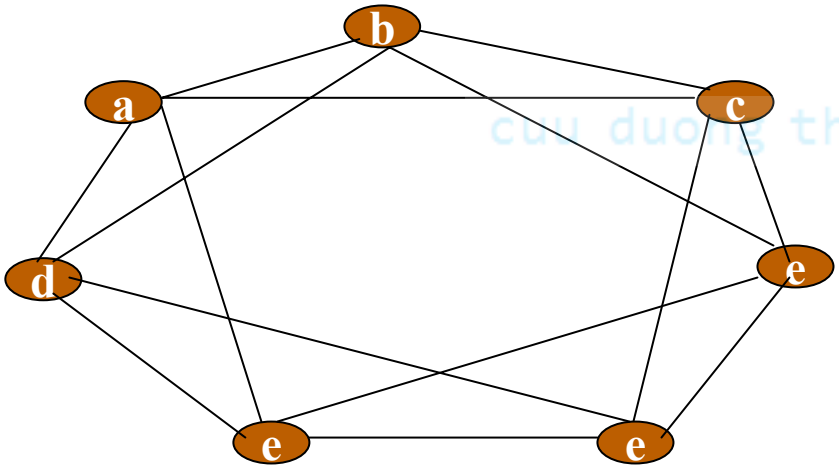
❖ ***Định lý .3*** (*Kuratowski*)

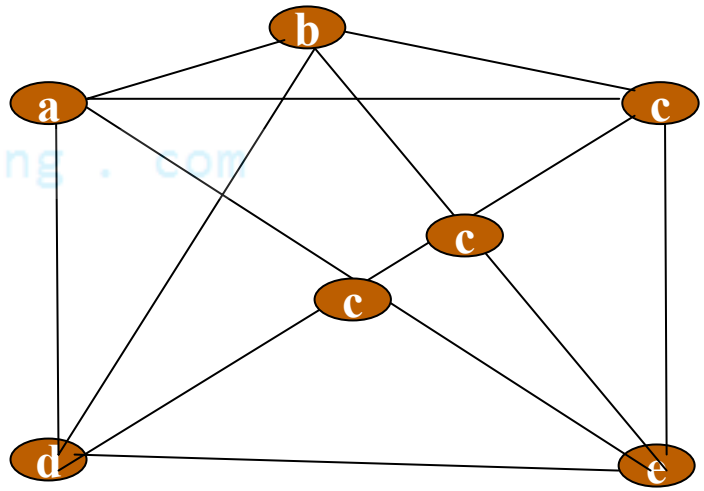
Đồ thị là phẳng khi và chỉ khi nó không chứa cấu hình $K_{3,3}$ hoặc K_5.

❖ Ta có thể áp dụng định lý Kuratowski để xét tính chất phẳng của đồ thị.

VD3

Xét đồ thị sau đây (bên phải) và hình vẽ lại của nó (bên trái):





Đồ thị này chứa cấu hình K_5. Do vậy, nó không phẳng.

Tô màu đồ thị

- **Định nghĩa 2**. ***Số màu*** của một đồ thị là số tối thiểu các màu cần thiết để tô màu đồ thị này.
- **Định lý 1**. *Định lý Bốn màu*. Số màu của đồ thị phẳng là không lớn hơn 4.

SẮC SỐ CỦA ĐỒ THỊ PHẲNG

❖ ***Định lý 10.4*** (*Kemple - Heawood*)

Mọi đồ thị phẳng không có đỉnh nút đều có sắc số không lớn hơn 5.

***Chứng minh*:**

Quy nạp theo số đỉnh *n* của đồ thị.

- *n* = 1, 2, 3, 4, 5 : Hiển nhiên đúng.
- (*n*-1) $\Rightarrow$ (*n*): theo Hệ quả 10.3, G có ít nhất 1 đỉnh *x* bậc không quá 5.

Bỏ đỉnh *x* ra khỏi G, ta được G'.

SẮC SỐ CỦA ĐỒ THỊ PHẲNG (tiếp)

Theo quy nạp, G’ có sắc số không vượt quá 5.

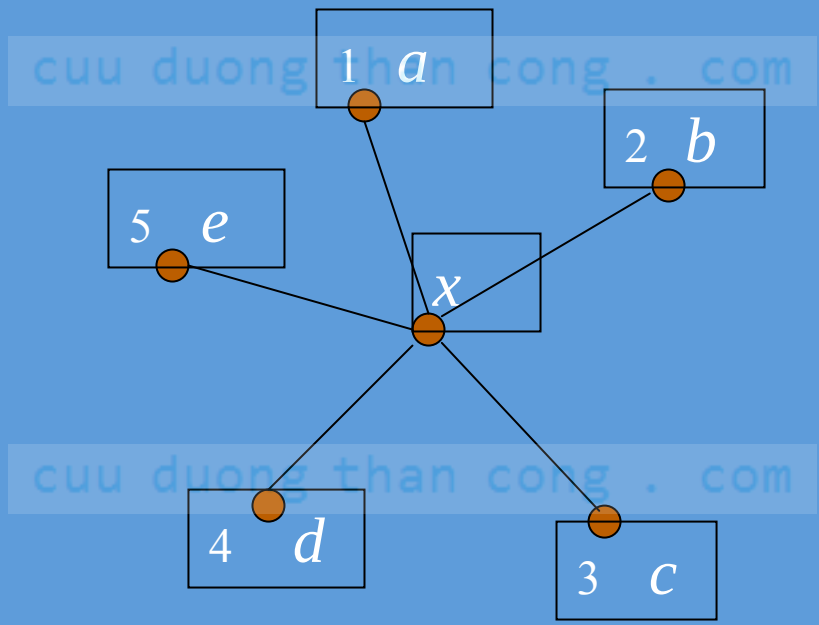
Lấy một cách tô màu nào đấy của G’.

- Nếu các đỉnh kề với đỉnh *x* được tô bằng ít hơn 5 màu thì vẫn còn thừa màu để tô cho *x*.

Sắc số của G bằng sắc số của G’.

SẮC SỐ CỦA ĐỒ THỊ PHẲNG (tiếp)

- Nếu đỉnh x kề với 5 đỉnh và các đỉnh kề với x được đánh số thứ tự theo chiều kim đồng hồ và tô bằng 5 màu thì ta đổi màu của các đỉnh để dành ra màu cho đỉnh x.



SẮC SỐ CỦA ĐỒ THỊ PHẲNG (tiếp)

Xét tất cả các đường đi trong G bắt đầu từ đỉnh *a* và chỉ gồm các đỉnh tô bằng màu 1 và màu 3.

Xét hai trường hợp:

1) Trong các đường này không có đường nào đi qua đỉnh *c* thì ta có thể tráo đổi màu 1 với màu 3 cho tất cả các đỉnh trên các đường đi ấy. Sau đó, ta tô màu 1 cho đỉnh *x*.

SẮC SỐ CỦA ĐỒ THỊ PHẲNG (tiếp)

2) Ngược lại, nếu có đường đi từ *a* đến *c* gồm toàn các đỉnh được tô bằng màu 1 hoặc 3 thì đường này cùng hai cạnh (*c*,*x*) và (*x*,*a*) tạo thành chu trình trong G.

Do tính phẳng nên hai đỉnh *b* và *d* không thể cùng nằm bên trong hoặc bên ngoài chu trình này được. Suy ra, không có đường đi nào nối *b* với *d* gồm các đỉnh chỉ tô màu 2 hoặc 4.

Vậy lại có thể tráo đổi màu 2 với màu 4 cho các đỉnh trên các đường đi qua đỉnh *b*. Khi đó hai đỉnh *b* và *d* cùng màu 4.

Tô màu 2 cho đỉnh *x*. □ □

Chương 7.TÔ MÀU(BÀI TOÁN BỐN MÀU)

❖ ***Bài toán:***

- Vào khoảng năm năm mươi của thế kỷ 19 Gazri, một thương gia người Anh, khi tô màu bản đồ hành chính nước mình, đã nhận ra rằng luôn có thể tô được bằng 4 màu.
- Năm 1852 ông ta thông báo giả thuyết này cho De Morgan.
- Năm 1878 Keli đã đăng bài toán trên trong Tuyển tập các công trình của Hội Toán học Anh, gây nên sự chú ý của nhiều người.

BÀI TOÁN BỐN MÀU (tiếp)

- Năm 1976, ba nhà khoa học người Mỹ là K.Appel, W. Haken và J. Koch chứng minh bằng máy tính điện tử được rằng giả thuyết của Gazri là đúng.

Định lý 10.5 (*Appel - Haken*):

Mọi đồ thị phẳng không có đỉnh nút đều có sắc số không quá 4.

Dễ thấy rằng, đồ thị vô hướng đầy đủ K_n ($n \geq 5$) có sắc số lớn hơn 4 nên không phẳng.

Nhóm 2
- Bùi văn Đạm- nhóm 2: itbuivandam@gmail.com: 0982782663
- Vũ Văn Thuận- nhóm2: vuthuan1989@gmail.com: 01649607770

Nhóm 3:
- Hoàng văn Nam: daigiathattien@yahoo.com.vn 0936916986
- Nguyễn Xuân Cường;xcuongnd@gmail.com 0123558813

Nhóm 5.
- nguyễn thị Tĩnh: nguyentinh.ptit.94@gmail.com 01644626065
- Nguyễn văn Nam nvnam.chjnsu@gmail.com 01685757972

Nhóm 6
- Phạm Quag Chiến chienptit.94@gmail.com 01677657894
- Hoàng Thị Hằng hoanghangnd2810@gmail.com 01666858700