Chương 7

CÁC THUẬT TOÁN ĐỒ THỊ

Nội dung

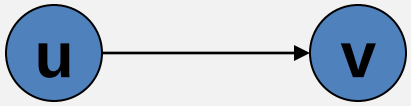
1. Một số khái niệm cơ bản
2. Biểu diễn đồ thị
3. Tìm kiếm trên đồ thị
4. Thành phần liên thông
5. Thành phần liên thông mạnh
6. Tìm đường đi ngắn nhất

7.1. Một số khái niệm cơ bản

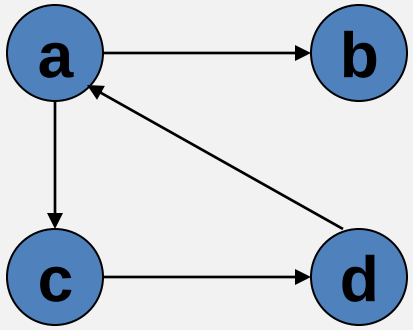
- Đồ thị được sử dụng để mô hình hóa các bài toán bao gồm một tập các đối tượng có quan hệ với nhau theo 1 cách nào đó.
- Ví dụ:
  - Một mạng truyền thông
  - Bản đồ đường đi giữa các thành phố
- Việc giải quyết các bài toán trở thành việc giải quyết một bài toán trên đồ thị.
- Ví dụ:
  - Tìm đường đi ngắn nhất
  - Tìm cây bao trùm ngắn nhất
  - Tìm các thành phần liên thông…

Một số khái niệm cơ bản (tt)

- **Một đồ thị định hướng G = <V, E>**
  - V là tập các đỉnh, E là tập các cung nối các đỉnh
  - Mỗi cung là một cặp đỉnh có thứ tự (u,v), ký hiệu u->v
  - Nếu có cung (u,v) ta nói đỉnh v kề với đỉnh u



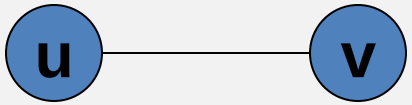
Một cung của đồ thị



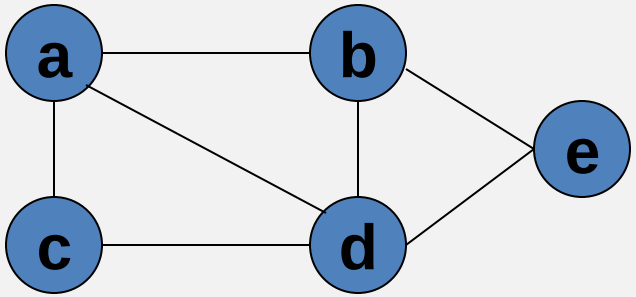
Đồ thị định hướng

Một số khái niệm cơ bản (tt)

- **Một đồ thị vô hướng G = <V, E>**
  - V là tập các đỉnh, E là tập các cạnh nối các đỉnh.
  - Mỗi cạnh là một cặp đỉnh không có thứ tự (u,v).
  - Nếu có cạnh (u,v) ta nói đỉnh u và v kề nhau.



Một cạnh của đồ thị



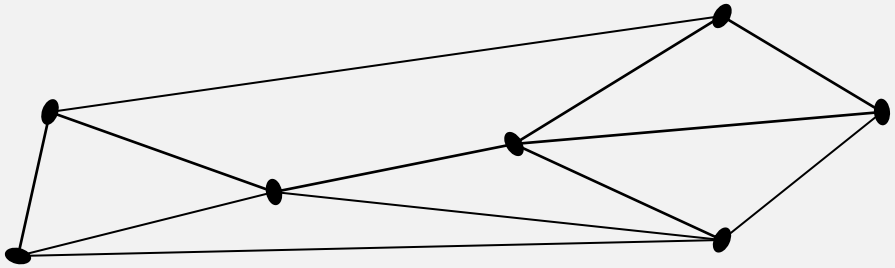
Đồ thị vô hướng

Một số khái niệm cơ bản (tt)

- **Đồ thị có trọng số**
  - Đồ thị mà mỗi cung/cạnh của đồ thị được gắn với một số c(u,v)
  - Số c(u,v) được gọi là trọng số (giá/độ dài) của cung/cạnh (u,v)
- **Đường đi đơn**
  - Là một dãy hữu hạn các đỉnh (v_0, v_1, …, v_k) khác nhau, ngoại trừ có thể $v_0 = v_k$, và v_{i+1} là đỉnh kề của v_i (i = 1, 2 …, k-1).
  - Với đồ thị có trọng số độ dài đường đi được tính là tổng trọng số trên các cạnh trên đường đi.
  - Với đồ thị không có trọng số độ dài đường đi là k.

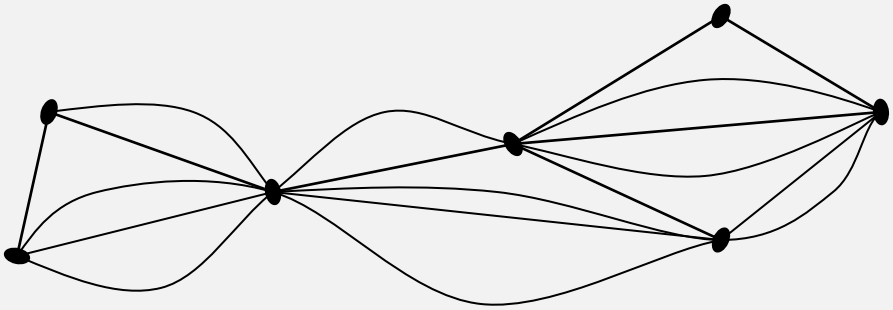
Một số khái niệm cơ bản (tt)

- **Đồ thị đơn**



Là đồ thị vô hướng có nhiều nhất một cạnh nối hai đỉnh

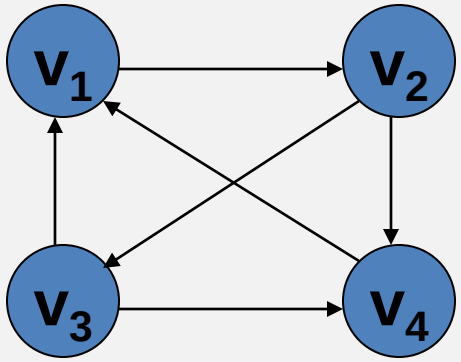
- **Đa đồ thị**



Là đồ thị vô hướng mà hai đỉnh có thể được nối bởi nhiều hơn một cạnh

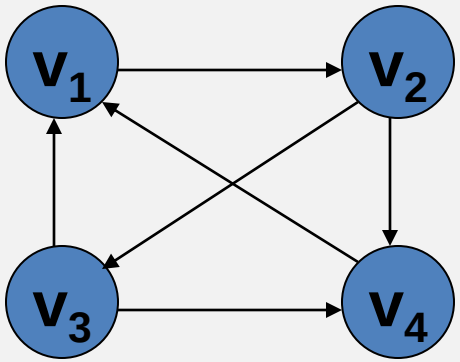
Một số khái niệm cơ bản (tt)

- **Chu trình**
  - Là đường đi khép kín, tức là đường đi (v_0, v_1, …, v_k) có $v_0 = v_k$.
- **Ví dụ**
  - Trong đồ thị định hướng dưới đây thì (v_4, v_1, v_2, v_3) là một đường đi từ v_4 đến v_3, và (v_4, v_1, v_2, v_4) là một chu trình.

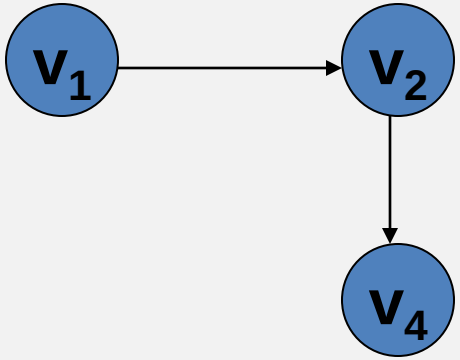


Một số khái niệm cơ bản (tt)

- **Đồ thị con**
  - Đồ thị G' = <V', E'> là đồ thị con của G = <V, E> nếu V' $\subseteq$ V và E' $\subseteq$ E.
- **Ví dụ:**



Đồ thị



Đồ thị con

7.2. Biểu diễn cấu trúc dữ liệu của đồ thị

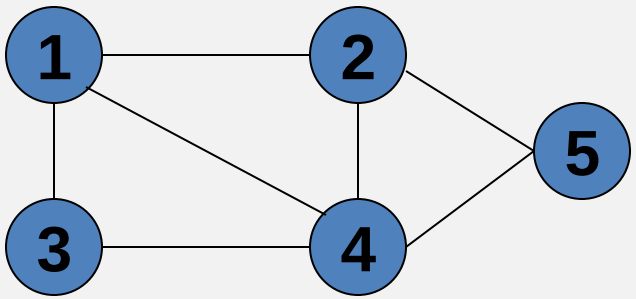
- **Để giải quyết các vấn đề của đồ thị, cần lưu trữ đồ thị trong bộ nhớ máy tính, vì thế cần phải biểu diễn CTDL của đồ thị.**
- **Có hai cách biểu diễn**
  - **Biểu diễn bằng ma trận kề (lưu trữ đồ thị bằng mảng hai chiều).**
  - **Biểu diễn bằng danh sách kề (lưu trữ đồ thị bằng danh sách móc nối).**

7.2.1. Biểu diễn đồ thị bằng ma trận kề

– **Với đồ thị không có trọng số G=<V,E>, với V={v_1, v_2, …, v_n}, được lưu trữ trong mảng hai chiều A[1..n][1..n] trong đó:**

$$A[i,j] = \begin{cases} 1 & \text{Nếu } (v_i, v_j) \text{ có cạnh nối} \\ 0 & \text{Nếu (vi, vj) không có cạnh nối} \end{cases}$$

– **Ví dụ: cho đồ thị vô hướng**



Đồ thị vô hướng

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

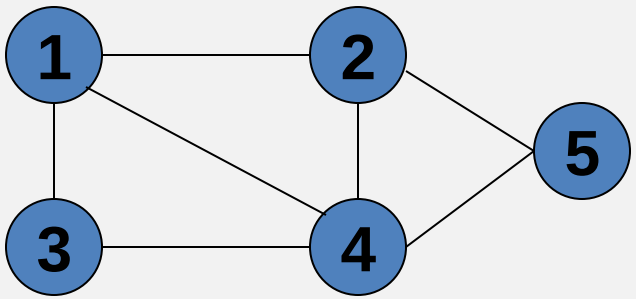
Mảng biểu diễn đồ thị

Biểu diễn đồ thị bằng ma trận kề (tt)

– **Với đồ thị không có trọng số G=<V,E>, với V={v_1, v_2, …, v_n}, được lưu trữ trong mảng hai chiều A[1..n][1..n] trong đó:**

$$A[i,j] = \begin{cases} 1 & \text{Nếu } (v_i, v_j) \text{ có cung/cạnh nối} \\ 0 & \text{Nếu (vi, vj) không có cung/cạnh nối} \end{cases}$$

– **Ví dụ 1: cho đồ thị vô hướng**



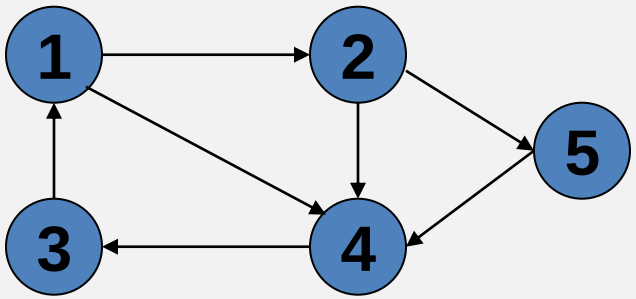
Đồ thị vô hướng

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

Mảng biểu diễn đồ thị

Biểu diễn đồ thị bằng ma trận kề (tt)

– **Ví dụ 2**: cho đồ thị định hướng



Đồ thị định hướng

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

Mảng biểu diễn đồ thị

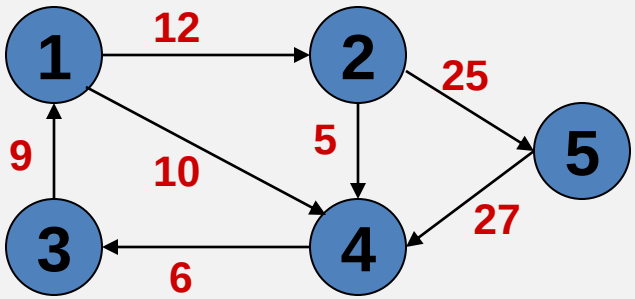
- **Nhận xét:**
  - **Ưu điểm:** Với 2 đỉnh bất kỳ ta biết ngay rằng có cung/cạnh nối hay không trọng số của cung/cạnh đó (nếu là đồ thị trọng số)
  - **Nhược điểm:** Lãng phí bộ nhớ (do chỉ có một số cung/cạnh)

Biểu diễn đồ thị bằng ma trận kề (tt)

– **Với đồ thị có trọng số G=<V,E>, với V={v_1, v_2, …, v_n}, với c(v_i, v_j) là trọng số trên cung/cạnh (v_i, v_j), khi đó:**

$$A[i,j] = \begin{cases} c(v_i, v_j) & \text{Nếu } (v_i, v_j) \text{ có cung/cạnh nối} \\ 0 & \text{Nếu } (v_i, v_j) \text{ không có cung/cạnh nối} \end{cases}$$

– **Ví dụ 1: cho đồ thị định hướng**



Đồ thị định hướng

0	12	0	10	0
0	0	0	5	25
9	0	0	0	0
0	0	6	0	0
0	0	0	27	0

Mảng biểu diễn đồ thị

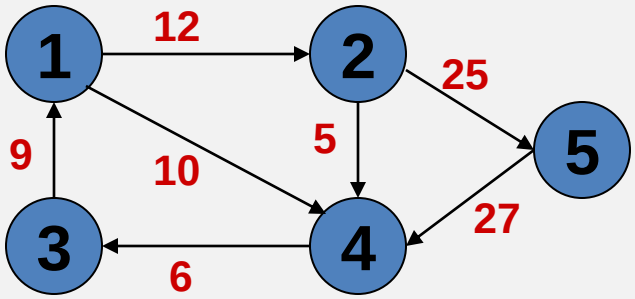
7.2.2. Biểu diễn đồ thị bằng danh sách kề

- **Cách biểu diễn**
  - Với mỗi đỉnh, lập một danh sách các đỉnh kề với nó
  - Danh sách các đỉnh kề là một danh sách móc nối
  - Mỗi thành phần trong danh sách gồm số hiệu đỉnh, trọng số của cung/cạnh nối
  - Sử dụng mảng A[1..n], trong đó A[i] là con trỏ trỏ tới đầu danh sách các đỉnh kề của đỉnh thứ I
- **Nhận xét:**
  - ***Ưu điểm:*** Tiết kiệm bộ nhớ
  - ***Nhược điểm:*** muốn biết có cung/cạnh (v_i, v_j) hay không (và trọng số của nó) ta phải duyệt danh sách các đỉnh kề của v_i.

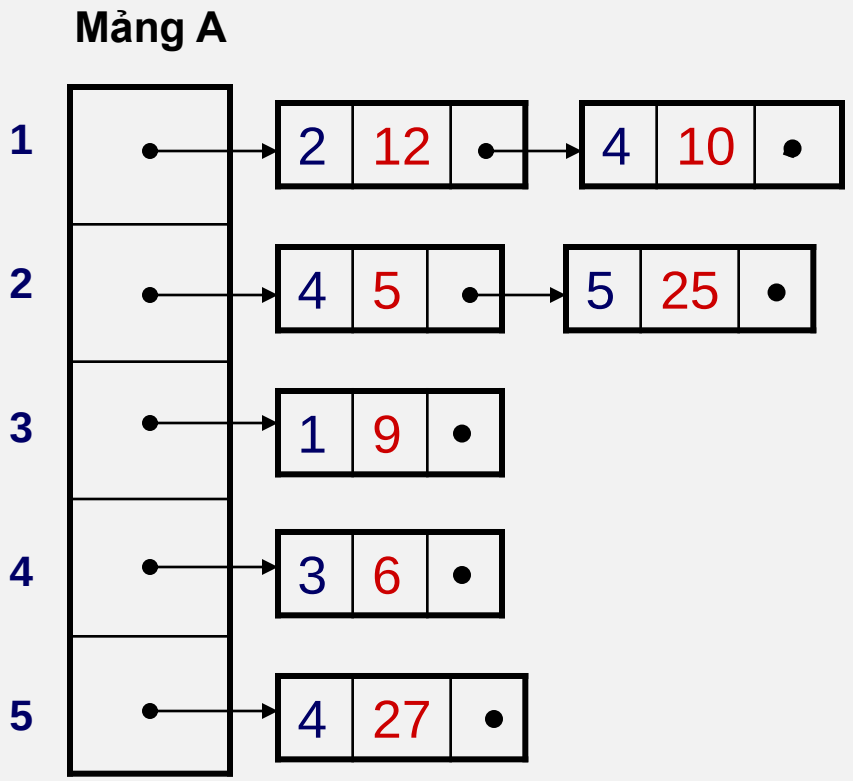
Biểu diễn đồ thị bằng danh sách kề (tt)

- **Ví dụ :**

cho đồ thị định hướng



Đồ thị định hướng



Mảng các danh sách kề

Biểu diễn đồ thị bằng danh sách kề (tt)

- **Cấu trúc dữ liệu**

```
#define N 100
struct node
{
    unsigned int index;          //số hiệu đỉnh
    float cost;                  //trọng số cung/cạnh
    struct node *next;
};
typedef struct member *node;
member A[N];
```

7.3. Tìm kiếm trên đồ thị

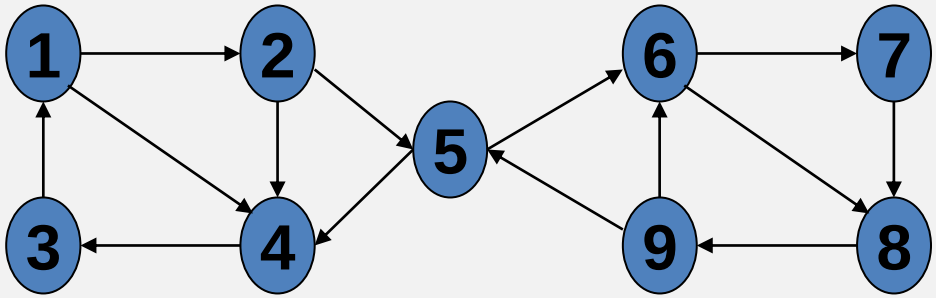
- **Có hai phương pháp**
  - **Tìm kiếm theo chiều rộng – Breath First Search**
  - **Tìm kiếm theo chiều sâu – Depth First Search**

7.3.1. Tìm kiếm theo chiều rộng - BFS

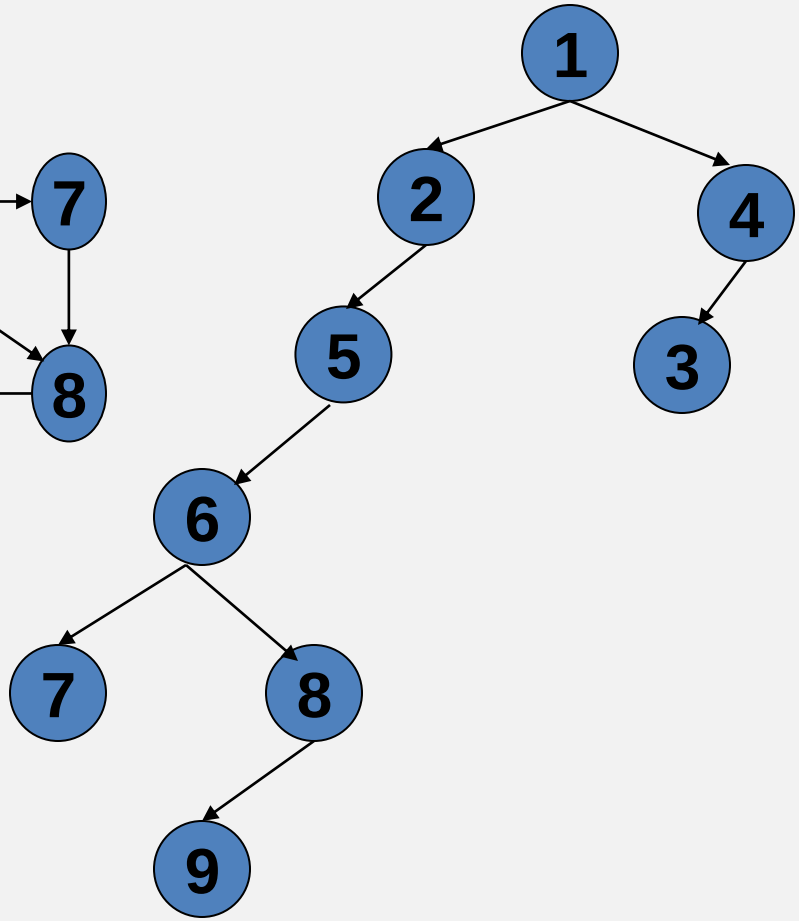
- Chọn một đỉnh u và thăm đỉnh đó
- Từ đỉnh u lần lượt thăm các đỉnh v kề u và chưa được thăm.
- Đỉnh v nào được thăm trước thì các đỉnh kề nó cũng được thăm trước
- Quá trình trên được tiếp tục cho tới khi không thể đến thăm đỉnh nào nữa.

Tìm kiếm theo chiều rộng – BFS (tt)

- **Ví dụ:** cho đồ thị



- Xuất phát từ đỉnh 1
- Quá trình đi thăm từ đỉnh u sẽ tạo ra cây gốc u



Tìm kiếm theo chiều rộng – BFS (tt)

- **Thuật toán**
  - **Cho G=<V, E> với V={1, 2, n}**
  - **Sử dụng hàng đợi Q để lưu các đỉnh đã được thăm nhưng chưa thăm các đỉnh kề của nó**
  - **Sử dụng mảng father[1..n] để lưu lại vết của đường đi xuất phát từ đỉnh u, father[w] = v nếu w được thăm từ v. Ban đầu mảng father được khởi tạo giá trị -1**
  - **Danh sách các đỉnh kề của v ký hiệu là Adj(v)**

Tìm kiếm theo chiều rộng – BFS (tt)

```
Procedure BFS(u)
{
        Father[u]=0;
        Creat(Q);
        Add(Q,u);
        While not empty(Q) Do
        { Del(Q,v);
          For mỗi w trong Adj(v) Do
                if father[w]= -1 then
                {       father[w]=v;
                        Add(Q,w);
                }
        }
}
```

Tìm kiếm theo chiều rộng – BFS (tt)

```
Procedure BreathTravelsal(G)
{
        for u=1 to n do  father[u]=-1;
        for u=1 to n do
          if father[u]=-1 then  BFS(u);
}
```

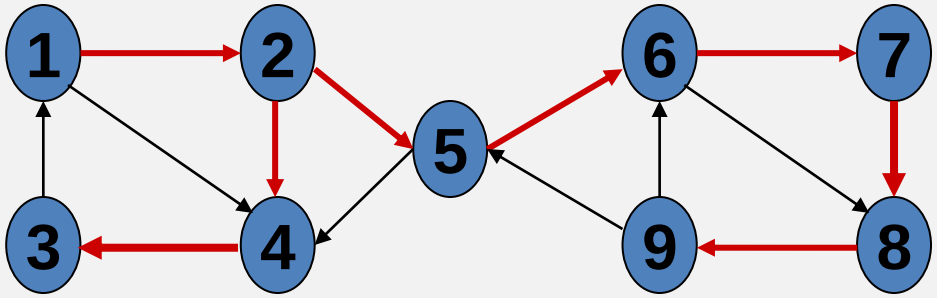
Chú ý: nếu G là đồ thị không có trọng số thì khi gọi thủ tục BFS(u) tìm ra đường đi ngắn nhất từ u đến các đỉnh có thể đạt tới từ u, sử dụng mảng father để xây dựng đường đi ngắn nhất tìm ra.

7.3.2. Tìm kiếm theo chiều sâu – DFS

- Xuất phát từ đỉnh u được thăm
- Thăm đỉnh w là đỉnh kề của u chưa được thăm
- Tìm kiếm theo chiều sâu xuất phát từ w
- Khi một đỉnh v đã được thăm mà mọi lân cận của nó đã được thăm, phép tìm kiếm quay lại đỉnh cuối cùng vừa được thăm mà đỉnh kề w của nó chưa được thăm, một phép tìm kiếm theo chiều sâu xuất phát từ w được thực hiện.

Tìm kiếm theo chiều sâu – DFS (tt)

- **Ví dụ:** cho đồ thị



- Xuất phát từ đỉnh 1
- Thứ tự các đỉnh được thăm:

  1, 2, 4, 3 (quay lại 2), 5, 6, 7, 8, 9

Tìm kiếm theo chiều sâu – DFS (tt)

```
Procedure DFS(u)
{
    i=i+1;
    pre_visite[u]=i;
    for mỗi w trong Adj(u) do
        if pre_visite[w]=0 then DFS(w);
    k=k+1;
    post_visited[u]=k;
}
```

Tìm kiếm theo chiều sâu – DFS (tt)

```
Procedure DepthTravelsal(G)
{
    for u=1 to n do  pre_visite[u]=0;
    i=0; k=0;
    for u=1 to n do
        if pre_visite[u]=0 then DFS(u);
}
```

7.4. Thành phần liên thông

- Đồ thị vô hướng G=<V, E> được gọi là liên thông nếu tồn tại đường đi nối mọi cặp đỉnh của nó.
- Nếu G không liên thông thì một đồ thị con liên thông cực đại của G được gọi là một thành phần liên thông
- Áp dụng thuật toán tìm kiếm theo chiều sâu, sẽ tìm được tất cả các thành phần liên thông.
- Thủ tục Comp_Search(G), tìm các thành phần liên thông
  - Biến c đếm số thành phần liên thông
  - Mảng num[1..n] lưu số hiệu chứa mỗi đỉnh
- Thủ tục Comp(u) đi theo độ sâu từ u, và gán cho các đỉnh đạt tới từ u một số hiệu thành phần liên thông c nào đó

Thành phần liên thông (tt)

```
Procedure Comp (u);
{
    num[u]=c;
    for each w in Adj(u) do
        if num[w]=0 then
            Comp(w);
}
```

```
Procedure Comp_Search(G);
{
    for u=1 to n do  num[u]=0;
    c=0;
    for u=1 to n do
        if num[u]=0 then
        {       c=c+1;
                Comp(u);
        }
}
```

7.5. Thành phần liên thông mạnh

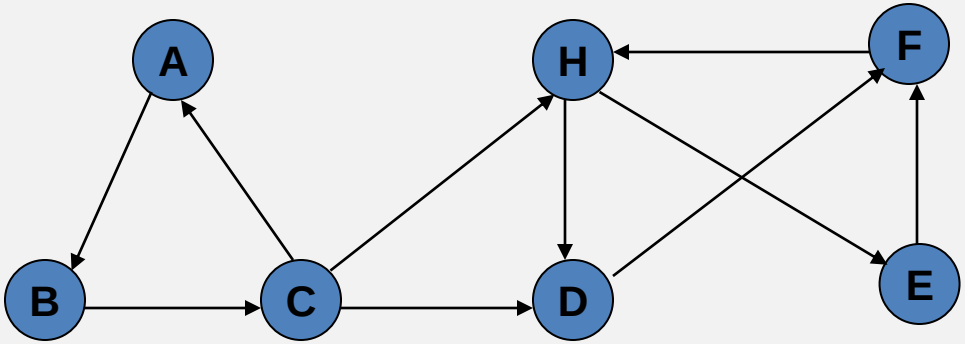
- Đồ thị vô hướng G=<V, E> được gọi là liên thông mạnh nếu mọi cặp đỉnh (u,v) của nó đều tồn tại đường đi từ u đến v và từ v đến u.
- Trong đồ thị định hướng không liên thông mạnh ta gọi một đồ thị con liên thông mạnh cực đại của nó là một thành phần liên thông mạnh.
- Sử dụng thuật toán tìm kiếm theo chiều sâu, ta có thể xây dựng được thuật toán xác định thành phần liên thông mạnh, gồm các bước:

Thành phần liên thông mạnh (tt)

- Duyệt đồ thị theo chiều sâu và đánh số các đỉnh của nó theo thứ tự sau (post_visited)
- Xây dựng đồ thị G’ từ đồ thị G bằng cách đổi hướng tất cả các cung
- Áp dụng DFS(u) duyệt G’ bắt đầu từ đỉnh u có pre_visited[u] lớn nhất. Khi đi từ u mà chưa thăm hết các đỉnh của G’ thì lại duyệt G’ bắt đầu từ đỉnh w có pre_visited[w] lớn nhất trong các đỉnh còn lại. Lặp lại quá trình cho đến khi tất cả các đỉnh của G’ được thăm
- Mỗi cây nhận được ở bước 3 là một thành phần liên thông mạnh của đồ thị G.

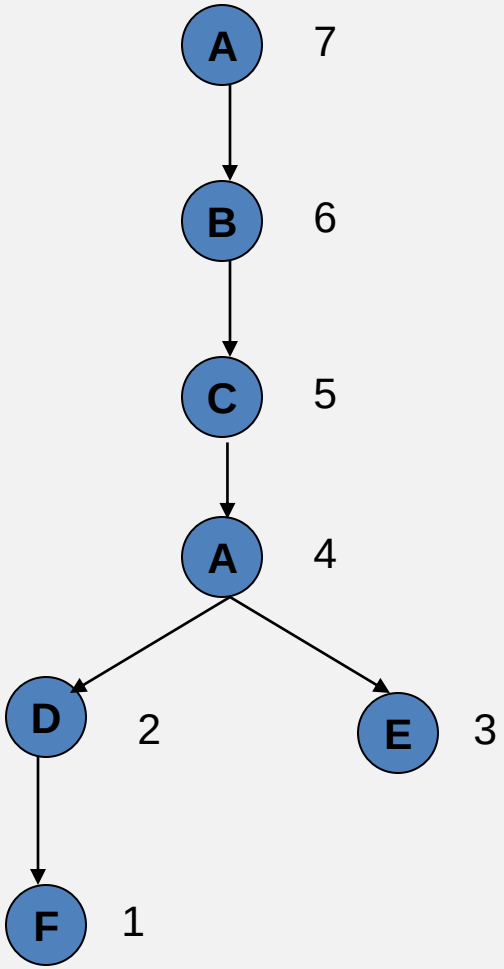
Thành phần liên thông mạnh (tt)

Ví dụ: cho đồ thị G



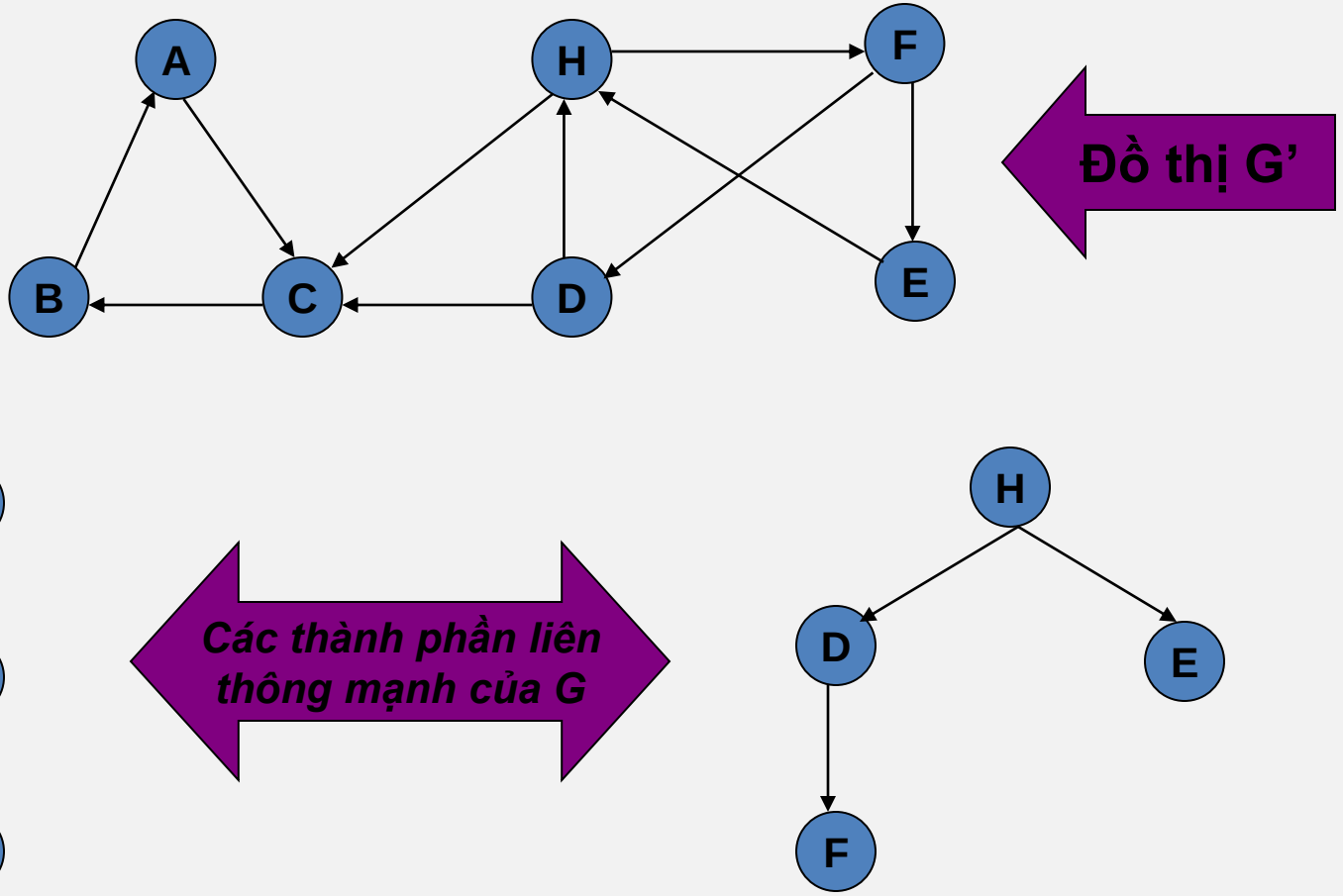
Đồ thị định hướng G

DFS



Thành phần liên thông mạnh (tt)

- Ví dụ



7.6. Tìm đường đi ngắn nhất

- Duyệt đồ thị theo chiều rộng để tìm đường đi ngắn nhất.
- Giả thiết G = <V, E> là đồ thị định hướng có trọng số.
- Trọng số trên cung (u,v) là c(u,v) >= 0.

- Độ dài đường đi từ v_0 đến v_k là $\delta(v_0,v_k) = \sum_{i=0}^{k-1} c(v_i,v_{i+1})$
- Xét hai thuật toán
  - Tìm đường đi ngắn nhất từ một đỉnh nguồn đến các đỉnh còn lại của đồ thị.
  - Tìm đường đi ngắn nhất giữa mọi cặp điểm của đồ thị.

7.6.1. Thuật toán Dijkstra

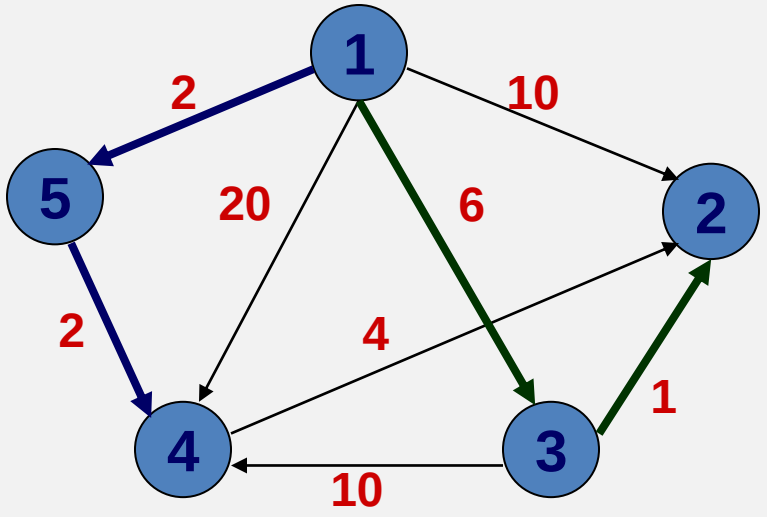
- Thuật toán Dijkstra giải quyết bài toán các lộ trình ngắn nhất nguồn đơn trên đồ thị định hướng có trọng số
- Thuật toán dựa trên kỹ thuật tham lam
- Ta giả thiết G = <V, E> có tập đỉnh được đánh số từ 1 đến n và đỉnh nguồn được chọn là 1.
- Đồ thị được biểu diễn bởi ma trận kề C[1..n][1..n].

Thuật toán Dijkstra

- **Mô tả thuật toán**
  - Tập S lưu các đỉnh mà đường đi ngắn nhất tới chúng đã tìm được
  - Mảng D[2..n] với mỗi D[u] lưu độ dài đường đi ngắn nhất từ 1 đến u
  - Ban đầu S chỉ chứa 1, khởi tạo D[u]=C[1,u] với u=2...n
  - Tại mỗi bước chọn v $\in$ V-S, mà D[v] nhỏ nhất và thêm v vào S
  - Xác định D[w] với w $\in$ S, nếu D[v]<D[w] thì D[w]=D[v]
  - Khi S=V thì D[u] lưu độ dài đường đi ngắn nhất từ 1 đến u, với u=2...n
  - Mảng P[2..n] lưu vết của đường đi, P[w]=v nếu w kề v trên đường đi

Thuật toán Dijkstra

- **Ví dụ**



Kết quả thực hiện thuật toán Dijkstra trên đồ thị G, với đỉnh nguồn 1

Bước	v	V-S	D	P
Khởi tạo	-	{2, 3, 4, 5}	[10, 6, 20, 2]	[1, 1, 1, 1]
1	5	{2, 3, 4}	[10, 6, 4, 2	[1, 1, 5, 1]
2	4	{2, 3}	[8, 6, 4, 2]	[4, 1, 5, 1]
3	3	{2}	[7, 6, 4, 2]	[3, 1, 5, 1]

Thuật toán Dijkstra

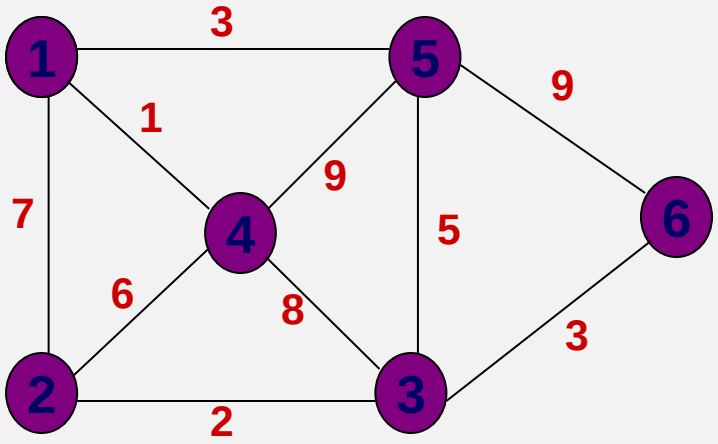
```
Procedure  Dijkstra();
Begin
    S:=S ∪ {1};
    For u:=2 To n Do
    begin
        D[u] := C[1,u];
        P[u] := 1;
    end;
    While (V-S <> ∅) Do
    begin
        Chọn v ∈ V-S mà D[v] nhỏ nhất;
        S := S ∪ {v};
        For (mỗi w ∈ V-S) Do
            IF (D[v]+c(v,w) < D[w]) THEN
            begin
                D[w] := D[v] + c(v,w);
                P[w] := v;
            end;
    end;
End;
```

7.6.2. Cây bao trùm ngắn nhất

- **Khái niệm**
  - G=<V,E> là đồ thị vô hướng với $c(u,v) \geq 0$, $(u,v) \in E$
  - Giả sử G là đồ thị liên thông
  - Gọi T là cây bao trùm của đồ thị G nếu T là đồ thị con liên thông, không có chu trình và chứa tất cả các đỉnh của G
  - Độ dài của cây T là tổng độ dài của các cạnh tạo thành cây
  - T là cây bao trùm ngắn nhất nếu T có độ dài ngắn nhất

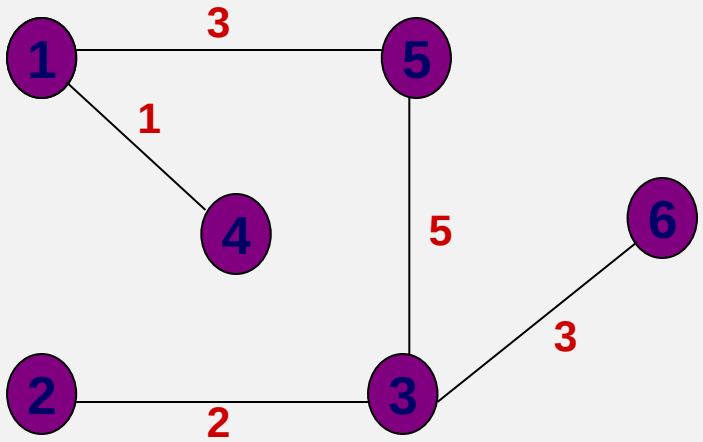
Cây bao trùm ngắn nhất (tt)

- **Ví dụ**



Đồ thị vô hướng, liên thông G

Cây bao trùm ngắn nhất của G



Thuật toán PRIM

- Thuật toán tìm cây bao trùm ngắn nhất
- Thuật toán sử dụng tập U chứa các đỉnh kề các cạnh trong T, ban đầu U chứa một đỉnh bất kỳ trong G
- T là tập các cạnh trong G, ban đầu T rỗng
- Ở mỗi bước ta chọn (u,v) ngắn nhất sao cho $u \in U$ và $v \in V-U$, rồi thêm v vào U và thêm (u,v) vào T
- Tiếp tục phát triển cây T cho đến khi U=V, khi đó T là cây bao trùm ngắn nhất trong G

Thuật toán PRIM

Procedure **Prim();**

Begin

U := U ∪ {u-một đỉnh tùy ý}

T := ∅;

Repeat

Chọn cạnh (u,v) ngắn nhất với u∈U và v∈V-U

U := U ∪ {v};

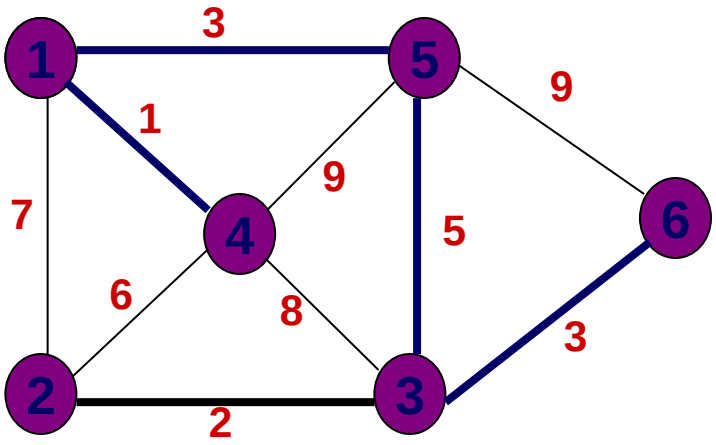
T := T ∪ {(u,v)}

Until (U=V);

End;

Thuật toán PRIM

- **Ví dụ**



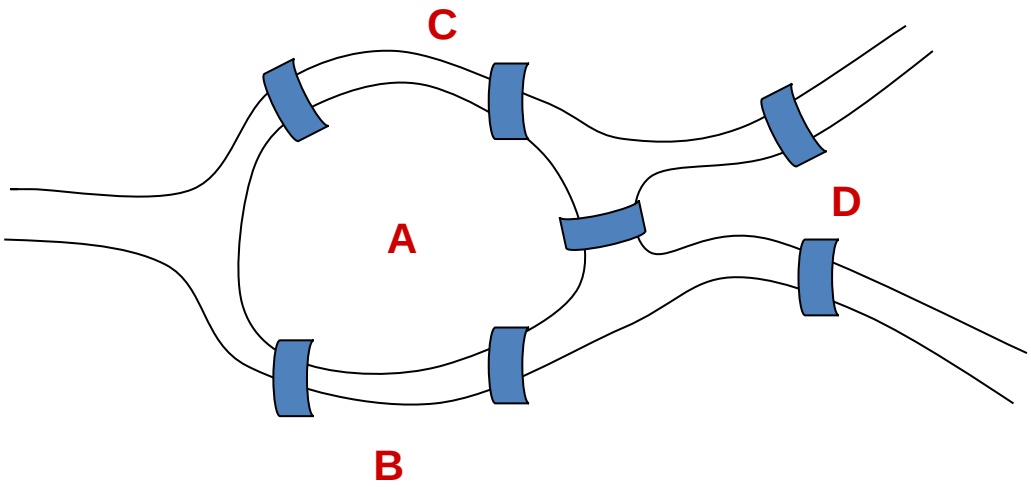
Đồ thị vô hướng, liên thông G

Bảng kết quả khi thực hiện thuật toán PRIM trên đồ thị G

Bước	(u,v)	U
khởi tạo		{1}
1	(1,4)	{1, 4}
2	(1,5)	{1, 4, 5}
3	(5,3)	{1, 4, 5, 3}
4	(3,2)	{1, 4, 5, 3, 2}
5	(3,6)	{1, 4, 5, 3, 2, 6}

7.6.3. Đường đi Euler – Chu trình Euler

- Thành phố Konigsberg- Phổ TK18 (Kaliningrad-Nga)

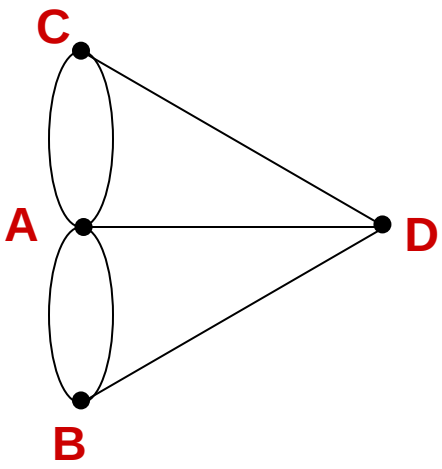


Leonhard Euler giải bài toán này năm 1736

Bài toán: Có cách nào xuất phát từ một điểm trong thành phố đi qua tất cả các cầu mỗi cầu đi qua đúng 1 lần và quay về điểm xuất phát?

Đường đi Euler – Chu trình Euler (tt)

- Bài toán được chuyển về



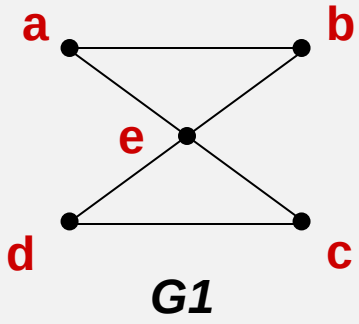
Đa đồ thị biểu diễn thành phố Konigsberg

Bài toán: Có tồn tại chu trình đơn trong đa đồ thị chứa tất cả các cạnh?

Đường đi Euler – Chu trình Euler (tt)

- **Định nghĩa:** Đường đi/Chu trình đơn chứa tất cả các cạnh của đồ thị G được gọi là Đường đi/chu trình Euler.

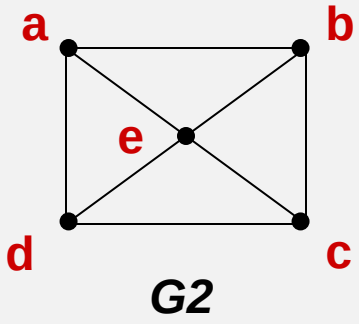
- **Ví dụ:**



G1

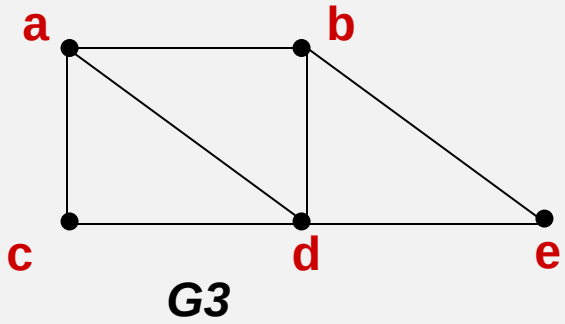
Chu trình Euler

a, e, c, d, e, b, a



G2

Không có chu trình hay đường đi Euler



G3

Đường đi Euler

a, c, d, e, b, d, a, b

Đường đi Euler – Chu trình Euler (tt)

- **Điều kiện cần và đủ cho chu trình/đường đi Euler**
  - ***Định lý 1***: Một đa đồ thị liên thông có chu trình Euler khi và chỉ khi mỗi đỉnh của nó đều có bậc chẵn
  - ***Định lý 2***: Một đa đồ thị liên thông có đường đi Euler nhưng không có chu trình Euler khi và chỉ khi nó có đúng hai đỉnh bậc lẻ

Đường đi Euler – Chu trình Euler (tt)

- **Bài toán**
  - **Input**: G = <V, E> là một đa đồ thị liên thông và có tất cả các đỉnh có bậc chẵn. V = {1, 2, ..., n}
  - **Output**: Chu trình Euler

Mô tả thuật toán

- Tập S lưu các đỉnh mà chu trình Euler đi qua, Ban đầu S chứa 1 đỉnh u nào đó, v $\in$ V.
- Tại mỗi bước thêm đỉnh w vào S, với w kề với v là đỉnh vừa được thêm vào S ở bước trước đó và loại bỏ cạnh (v,w) trong E
- Tiếp tục quá trình cho đến khi loại bỏ hết các cạnh của G

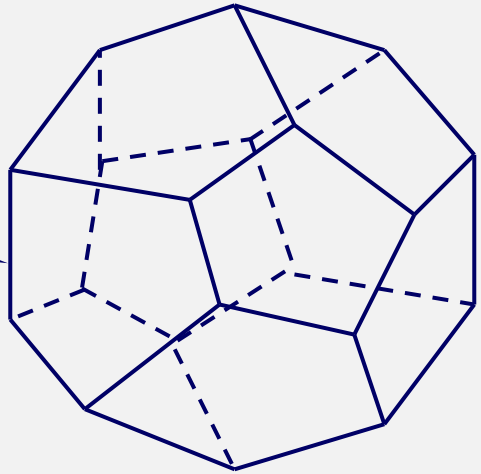
Đường đi Euler – Chu trình Euler (tt)

- **Thuật toán**

```
Procedure  Euler(G, S);
Begin
        S := {v}  // một đỉnh bất kỳ trong G
        While (E <> ∅) do
        begin
                - Tìm w kề với v;
                S := S + {w};
                E := E – {(v, w)};
        end;
End;
```

7.6.4. Đường đi Hamilton – Chu trình Hamilton

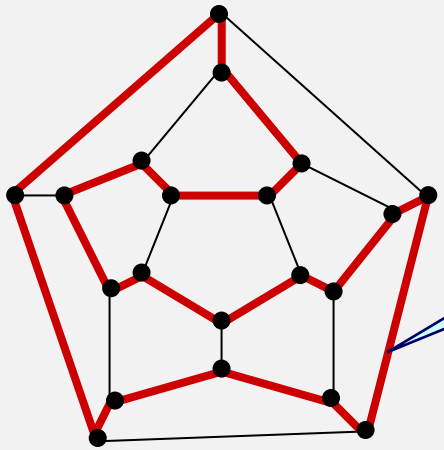
Chò chơi đi vòng quanh thế giới – William Rowan Hamilton – Ailen - 1857



- **Một hình khối có 12 mặt, mỗi mặt là một ngũ giác đều**
- **Mỗi đỉnh trong khối là tên một thành phố (TP).**
- **Bài toán: Từ một TP, đi thăm 19 TP còn lại, mỗi TP đúng một lần và quay về TP ban đầu.**

Đường đi Hamilton – Chu trình Hamilton (tt)

- **Bài toán được đưa về dạng**
  - **Input:** Đơn đồ thị G = <V,E>, V = {v_0, v_1, ..., v_n}
  - **Output:** Có hay không một chu trình đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần?



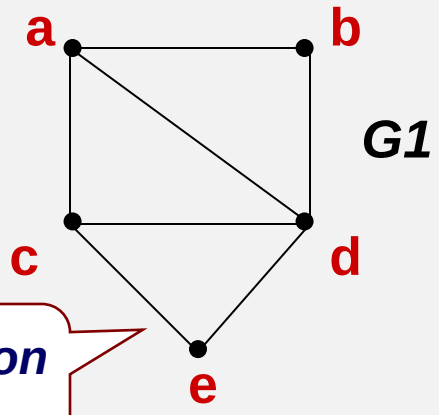
Lời giải của chò chơi đi vòng quanh thế giới

Đường đi Hamilton – Chu trình Hamilton (tt)

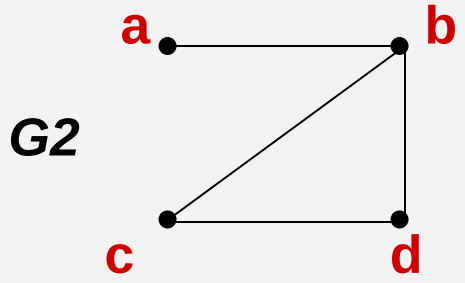
- **Định nghĩa**
  - **Đơn đồ thị G = <V, E>, V = {v_0, v_1, ..., v_n}**
  - **(v_0, v_1, ..., v_n) với $v_i \neq v_j$ ($0 \leq i \neq j \leq n$) được gọi là đường đi Hamilton.**
  - **(v_0, v_1, ..., v_n, v_0) là chu trình Hamilton nếu (**v0, v1, ..., vn**) là đường đi Hamilton.**

Đường đi Hamilton – Chu trình Hamilton (tt)

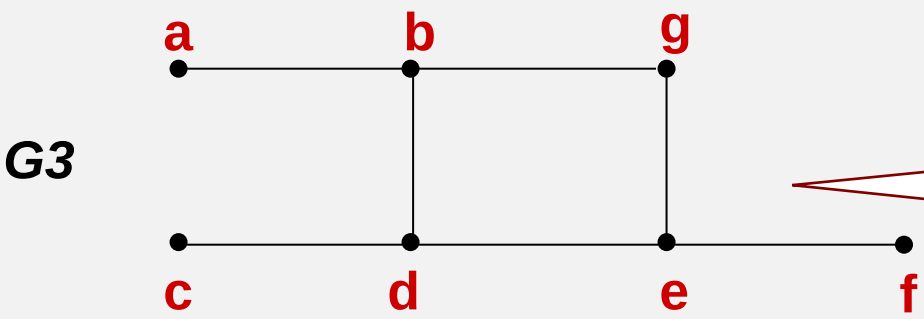
- **Ví dụ**



Có chu trình Hamilton (a, b, d, e, c, a)



Không có chu trình nhưng có đường đi Hamilton (a, b, c, d)



Không có chu trình và đường đi Hamilton

Đường đi Hamilton – Chu trình Hamilton (tt)

- **Định lý 3**
  - **Đơn đồ thị G = <V, E>, liên thông có n đỉnh, trong đó n ≥ 3.**
  - **G có chu trình Hamilton khi và chỉ khi mỗi đỉnh của nó đều có bậc ≥ [n/2].**
  - **(v_0, v_1, ..., v_n, v_0) là chu trình Hamilton nếu (v_0, v_1, ..., v_n) là đường đi Hamilton.**

Thuật toán tìm chu trình Hamilton

```
void TRY(i, u, ok) {
   //Khởi tạo việc chọn đỉnh được thăm tiếp theo
   do{Chọn đỉnh v trong danh sách đỉnh kề của u
      if (H[v] == 0) {
         if (v khác đỉnh xuất phát){
            H[v] = i; //Ghi nhớ đỉnh v được thăm
            PUSH (S, v);//Đẩy v vào ngăn xếp S
            TRY(i+1, v, ok); //Thăm đỉnh tiếp theo
            if (not ok){ //Không thành công
               H[v] = 0; //Xóa ghi nhớ
               POP(S,v); //Lấy v ra khỏi ngăn xếp S
            }
         }else {
            IF (i==n+1) {
               PUSH(S,v); ok=1; //Thành công
         }
      }
   }
   while (not ok) or (chưa hết danh sách kề với u);
}
```

Trân trọng cảm ơn…!

Design by Minh An