Chương 6: ĐỒ THỊ(GRAPHIC)

Data structures and Algorithms

Nội dung chính

- Các khái niệm cơ bản về đồ thị
- Cài đặt đồ thị
- Duyệt đồ thị

Nội dung chính

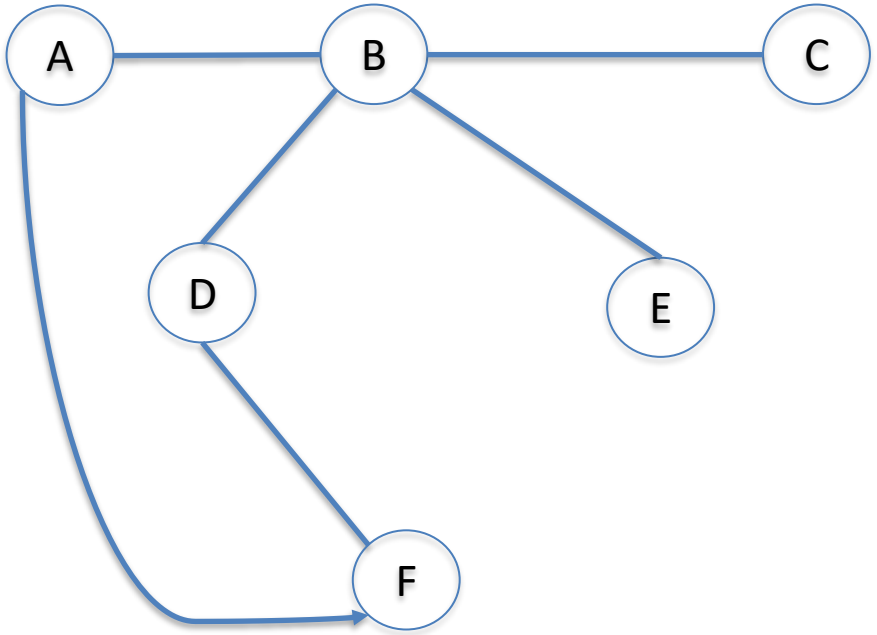
- Các khái niệm cơ bản về đồ thị
- Cài đặt đồ thị
- Duyệt đồ thị

Các khái niệm cơ bản về đồ thị

Đồ thị G là một cấu trúc gồm hai thành phần:

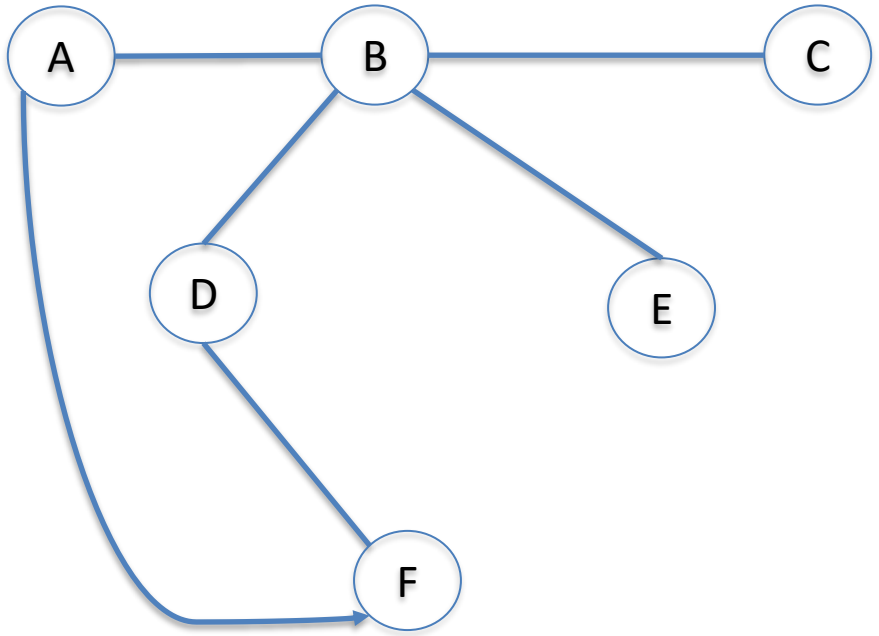
- $V = \{v_0, v_1, \dots, v_{m-1}\}$: tập hữu hạn gồm m đỉnh (nút, hay điểm).
- $E = \{e_0, e_1, \dots, e_{n-1}\}$, với $e_i = (v_j, v_k)$: tập hữu hạn gồm n cạnh (hay cung) nối các cặp đỉnh.

Kí hiệu đồ thị G là G(V, E).



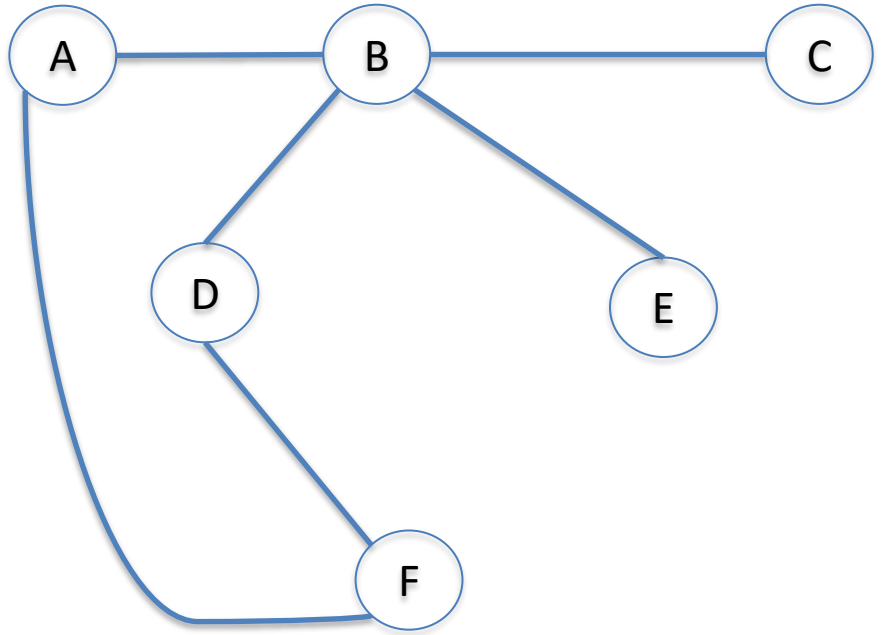
Các khái niệm cơ bản về đồ thị

Giả sử v_i là một đỉnh của đồ thị G(V,E).
Khi đó, ta gọi v_j là *đỉnh kề* của v_i
nếu $(v_i, v_j) \in E$.
Đồng thời, bản thân (v_i, v_j) cũng là *cạnh kề* của v_i và v_j.

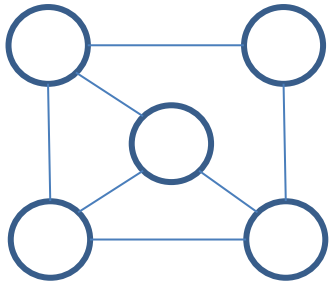


Các khái niệm cơ bản về đồ thị

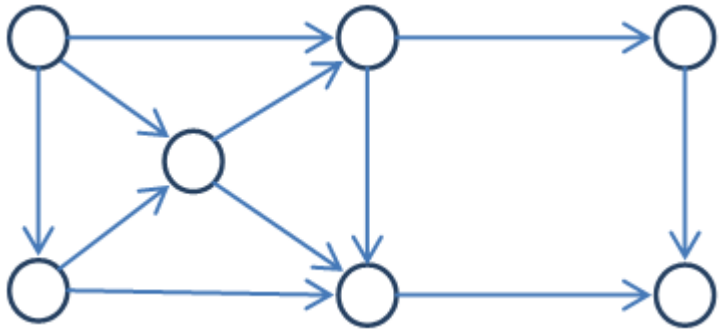
- Đường đi (path) từ u đến v là một dãy các đỉnh $x_0, x_1, \ldots, x_{n-1}, x_n$, $u = x_0$, $v = x_n$ và (x_i, x_{i+1}) là các cung thuộc E, u là đỉnh đầu, v là đỉnh cuối
  - Số lượng các cung trên một đường đi gọi là độ dài đường đi.
  - Đường đi đơn (single path) là đường đi mà mọi đỉnh trên đó trừ đỉnh đầu và cuối, đều khác nhau
  - Đỉnh đầu và cuối trùng nhau gọi là chu trình (cycle)



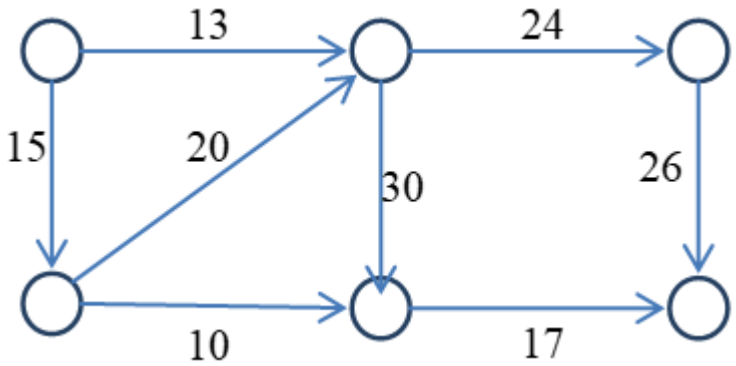
Phân loại đồ thị



Đồ thị vô hướng



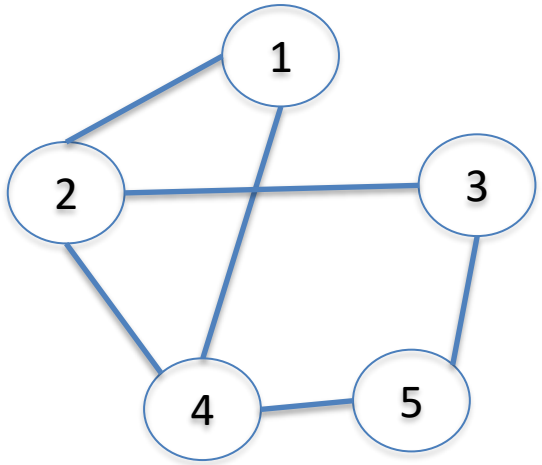
Đồ thị có hướng



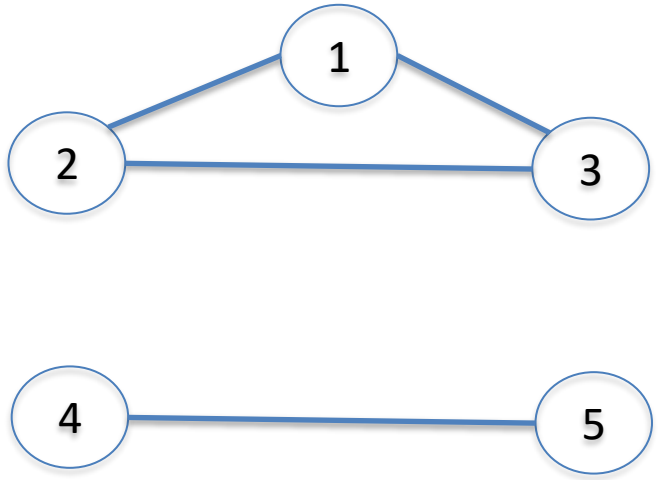
Đồ thị có trọng số

Đồ thị liên thông

- Đồ thị liên thông là đồ thị luôn có đường đi giữa 2 đỉnh bất kỳ



Đồ thị liên thông



Đồ thị không liên thông

Một số thao tác cơ bản trên đồ thị

- Duyệt đồ thị: truy nhập tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần. Có hai phương pháp duyệt đồ thị
  - Duyệt theo chiều sâu
  - Duyệt theo chiều rộng.
- Tìm kiếm một đỉnh hoặc một cạnh: là thao tác tìm một đỉnh hay một cạnh thỏa mãn một điều kiện nào đó, ví dụ như tìm đỉnh có giá trị cho trước, hay tìm cạnh có trọng số cho trước. Các giải thuật tìm kiếm này thường dựa vào các phương pháp duyệt đồ thị.
- Tìm đường đi ngắn nhất:
  - Tìm đường đi ngắn nhất giữa hai điểm.
  - Tìm đường đi ngắn nhất giữa một điểm và tất cả các điểm còn lại.

Cài đặt đồ thị

- Cài đặt bằng ma trận kề (lưu trữ tuần tự)
- Cài đặt bằng danh sách kề (lưu trữ móc nối)

Cài đặt đồ thị bằng ma trận kề

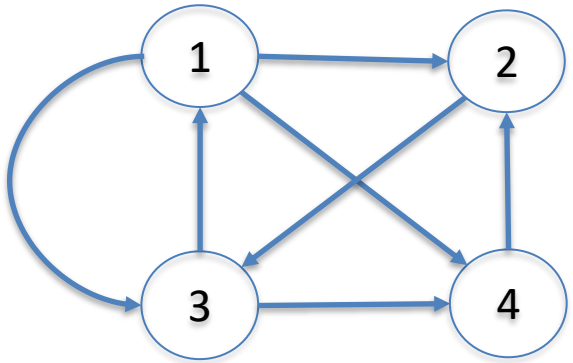
- Xét đồ thị có hướng G(V,E) với V có n đỉnh (n>=1). Giả sử các đỉnh được đánh số từ 1 đến n.
- Đồ thị G có thể biểu diễn dưới dạng ma trận vuông A kích thước nxn các phần tử A[i,j] sẽ nhận giá trị 1 hoặc 0
  - A[i,j] = 1, nếu G tồn tại một cung đi từ đỉnh i đến j
  - A[i,j] = 0, nếu không có cung từ đỉnh i đến j

Cài đặt đồ thị bằng ma trận kề

- Đồ thị vô hướng : nếu tồn tại cung đi từ i tới j thì cũng tồn tại một cung đi từ j tới i. Aij = 1 thì Aji = 1.
- Ma trận biểu diễn đồ thị vô hướng là ma trận đối xứng qua đường chéo chính

G1 =

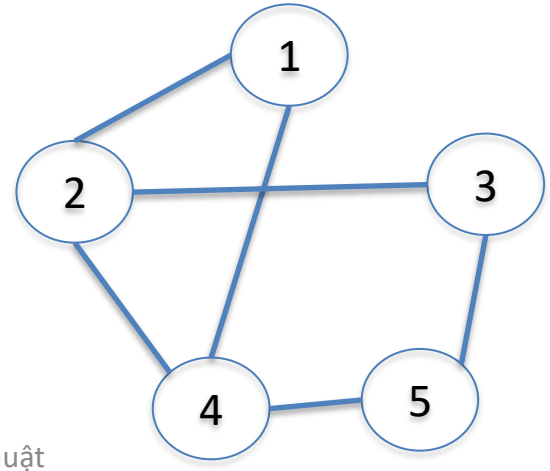
0	1	1	1
0	0	1	0
1	0	0	1
0	1	0	0



G1

G2 =

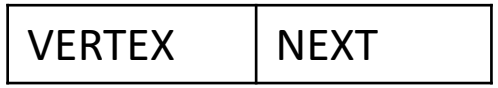
0	1	0	1	0
1	0	1	1	0
0	1	0	0	1
1	1	0	0	1
0	0	1	1	0



G2

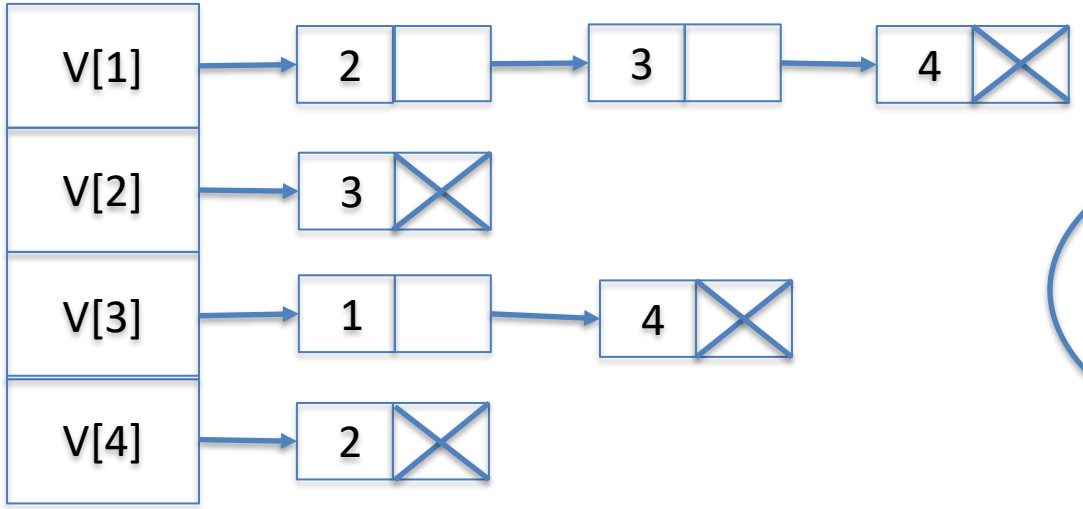
Cài đặt đồ thị bằng danh sách kề

- Mỗi đỉnh sẽ ứng với một danh sách móc nối
- Mỗi nút trong danh sách có quy cách
  - VERTEX: chứa số j ứng với đỉnh j là lân cận của đỉnh i (1<= i< = n)
  - NEXT: chứa con trỏ, trỏ tới nút tiếp theo trong danh sách

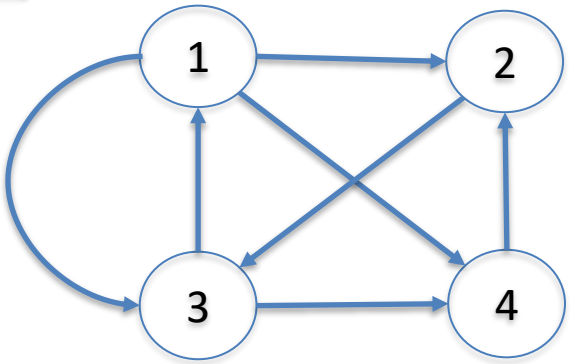


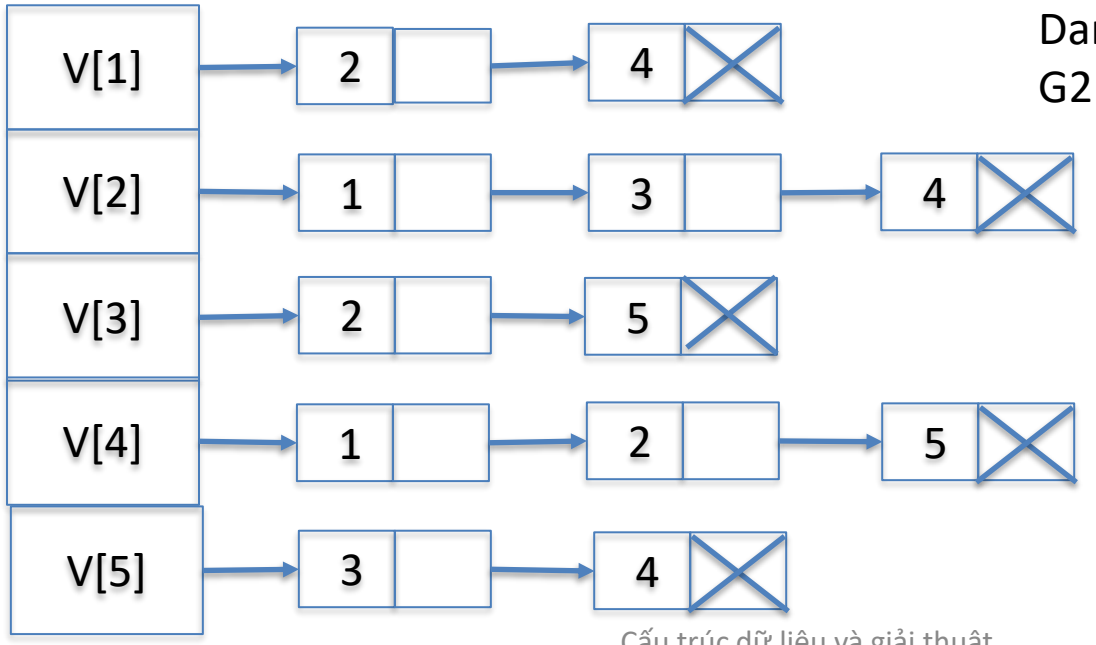
- Danh sách có m nút (nút danh sách), nếu đỉnh i có m đỉnh kề.
- Nút đầu danh sách: là các phần tử của một vector V có kích thước bằng n, phần tử V[i] chứa địa chỉ nút đầu tiên

Cài đặt đồ thị bằng danh sách kề

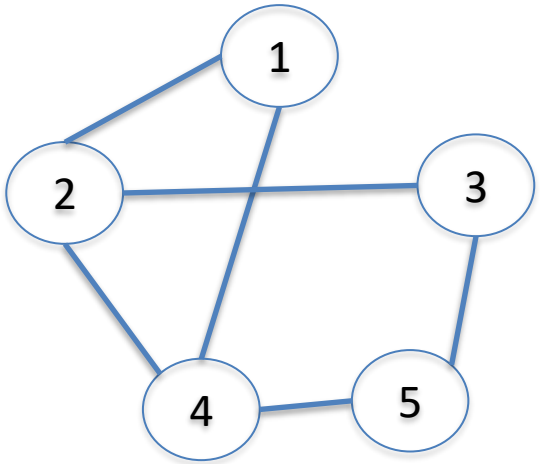


Danh sách kề (lân cận) biểu diễn G1





Danh sách kề (lân cận) biểu diễn G2



Duyệt đồ thị

Cho trước đồ thị $G(V, E)$ và nút xuất phát $v \in V$. Tìm cách duyệt tất cả các đỉnh của G, mỗi đỉnh đúng một lần, bắt đầu từ nút v.

Có hai phương pháp duyệt cơ bản:

- Duyệt theo chiều rộng (Breadth – First Search)
- Duyệt theo chiều sâu (Depth – First Search)

Duyệt đồ thị theo chiều sâu

- Ý tưởng:
  - Đỉnh xuất phát v được thăm
  - Đỉnh w chưa được thăm mà là lân cận của v được chọn và một phép tìm kiếm theo chiều sâu xuất phát từ w được thực hiện
  - Khi một đỉnh u được với tới mà mọi đỉnh lân cận với nó được thăm rồi, ta quay ngược lên đỉnh cuối cùng vừa được thăm (mà còn có đỉnh z chưa được thăm) và phép tìm kiếm theo chiều sâu xuất phát từ z được thực hiện.
  - Giải thuật kết thúc khi không còn nút nào chưa được thăm mà vẫn có thể với tới được từ một nút đã được thăm

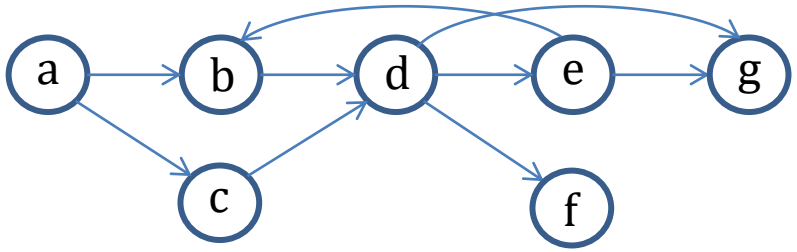
Duyệt đồ thị theo chiều sâu

- Giải thuật:

```
Procedure DFS(G,v)
{
    Visited(v)=1;// đánh dấu đỉnh v đã được thăm
    For each unvisited adjacent w to v do
        if Visited(w)=0 then call DFS(w);
    return;
}
```

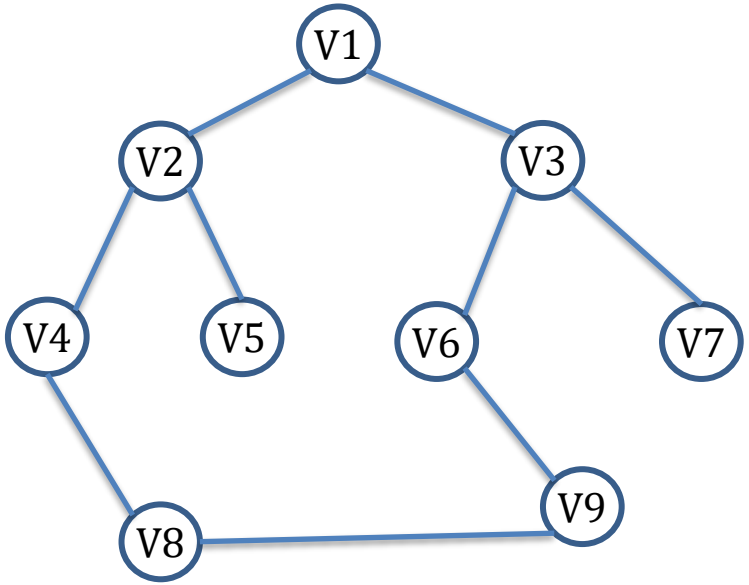
Duyệt đồ thị theo chiều sâu

- Ví dụ



Đồ thị G1

$DFS(\text{G1,a}) \Rightarrow$ a, b, d, e, g, f, c



Đồ thị G2

$DFS(\text{G2,V1}) \Rightarrow$

$DFS(\text{G2,V2}) \Rightarrow$

$DFS(\text{G2,V4}) \Rightarrow$

Duyệt đồ thị theo chiều rộng

Ý tưởng:

- Thăm nút *v*
- Tìm *VA* là tập các nút kề của *v* mà chưa được thăm
- Thăm lần lượt các nút trong *VA*
- Lặp lại giải thuật cho từng nút trong *VA*.

Duyệt đồ thị theo chiều rộng

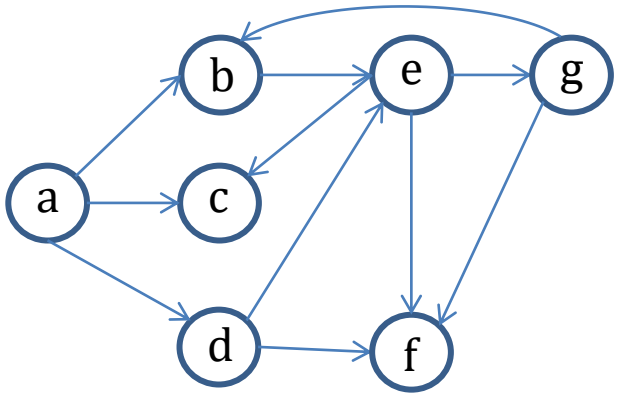
```
Procedure BFS(G, v)
{
Visited(v)=1;
Khởi tạo queue Q với v đã được nạp vào
while Q không rỗng do begin
      call deQueue(v,Q); //lấy đỉnh v ra khỏi Q
      for each unvisited adjacent w to v do
            if Visited(w) =0 then begin
                  call enQueue(w,Q);
                  Visited(w)=1;
            end;
end;
return;
}
```

Cần một cấu trúc dữ liệu trung gian để lưu các nút kề mà sau này sẽ lần lượt được thăm. Có thể thấy danh sách kiểu *hàng đợi* là cấu trúc phù hợp vì các nút được lưu trước sẽ được thăm trước

Duyệt đồ thị theo chiều rộng

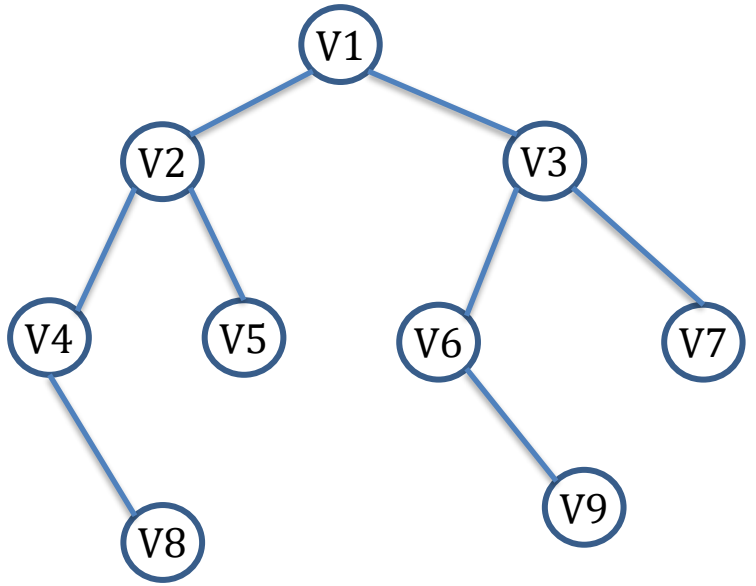
Ví dụ:

-



Đồ thị G1

$BFS(G1,a) \Rightarrow$ a, b, c, d, e, f, g



Đồ thị G2

$BFS(G2,V1) \Rightarrow$

$BFS(G2,V2) \Rightarrow$

$BFS(G2,V4) \Rightarrow$

Thuật toán áp dụng đồ thị

- Bài toán tìm cây khung có giá cực tiểu
- Bài toán tìm đường đi
- Bài toán sắp xếp topo

Bài toán tìm cây khung có giá cực tiểu

- Cây khung:

Đồ thị G liên thông thì một phép tìm kiếm theo chiều sâu hay chiều rộng, xuất phát từ một đỉnh bất kỳ cũng cho phép thăm được mọi đỉnh của G. Khi đó, các cung của G được phân làm 2 tập:

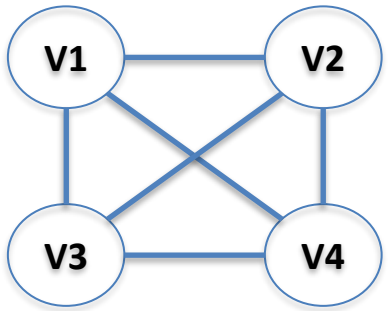
- Tập T gồm tất cả các cung được dùng tới hoặc được duyệt qua trong phép tìm kiếm
- Tập b gồm các khung còn lại

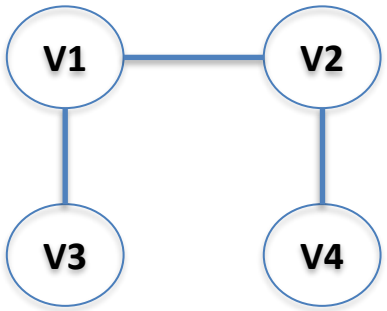
Các cung trong T cùng với các đỉnh tương ứng tạo thành cây khung của G, gồm 2 loại:

- Cây khung theo chiều rộng
- Cây khung theo chiều sâu

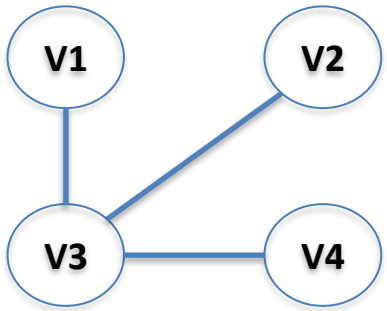
Bài toán tìm cây khung có giá cực tiểu

Ví dụ cây khung:

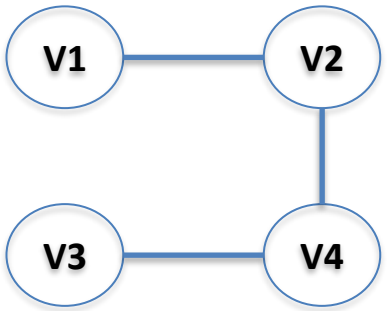




a)



b)



c)

Bài toán tìm cây khung có giá cực tiểu

- Cây khung có giá cực tiểu (áp dụng cho đồ thị liên thông có trọng số)
- Giá của một cây khung là tổng trọng số ứng với các cành (cung) của cây đó
- Cây khung có giá cực tiểu T được xây dựng dần dần từng cung một
- Các cung được đưa vào T dựa theo thứ tự không giảm của giá tương ứng với chúng
- Một cung được đưa vào T nếu nó không tạo nên một chu trình với các cung đã có trong T
- Đồ thị G liên thông có n đỉnh thì có n-1 cung được chọn đưa vào T

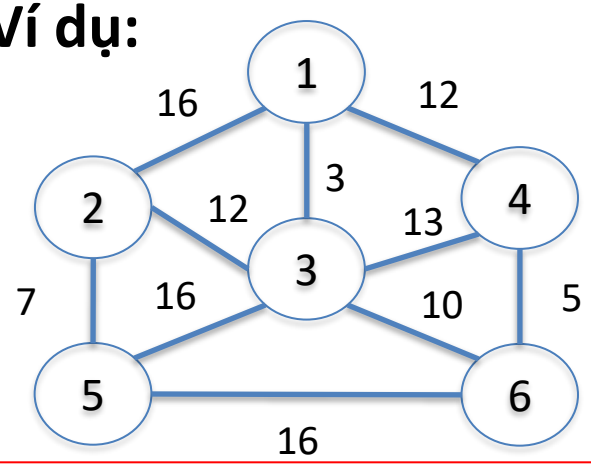
Bài toán tìm cây khung có giá cực tiểu

- Giải thuật Kruskal

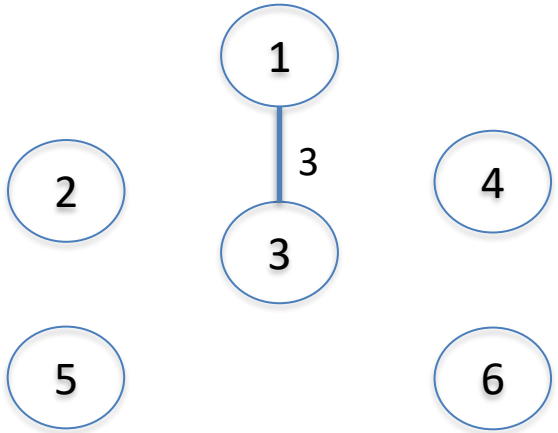
Procedure Kruskal (G){
T=Ø
While T chứa ít hơn n-1 cung **do begin**
    chọn một cung (v,w) từ E có giá trị thấp nhất
    loại (v,w) khỏi E
    if (v,w) không tạo nên chu trình trong T **then**
        đưa (v,w) vào T
    else loại bỏ (v,w)
End
Return

Bài toán tìm cây khung có giá cực tiểu

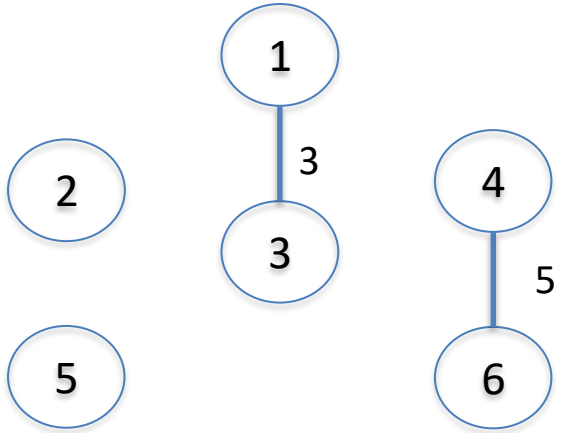
Ví dụ:



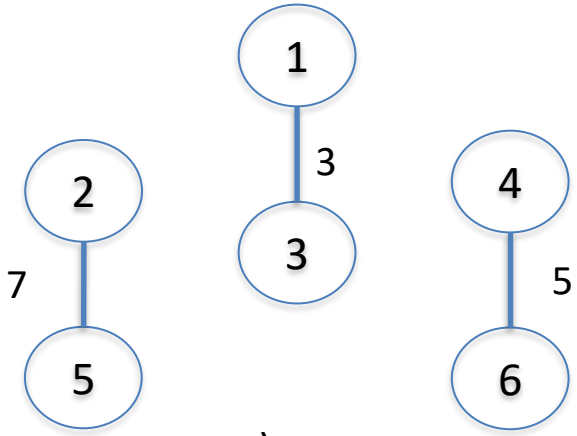
Dãy các cung được xét đưa vào cây khung với giá cực tiểu theo thứ tự không giảm về giá?



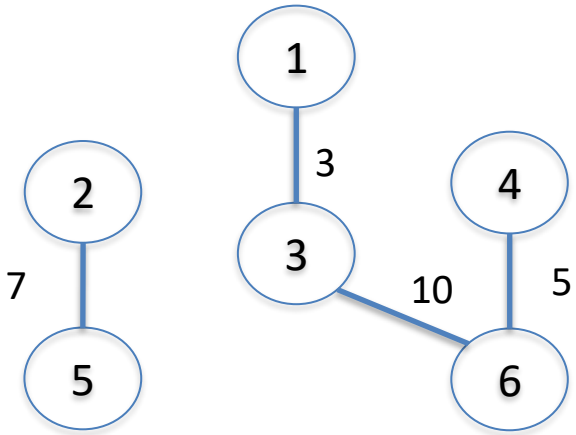
a)



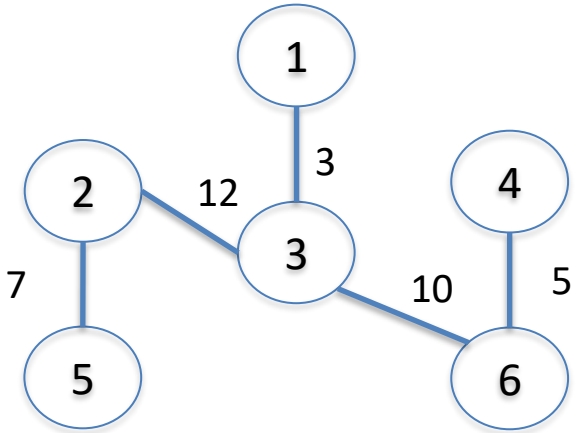
b)



c)



d)



e)

Bài toán tìm đường đi

- Xét một đồ thị G(V,E) với n đỉnh được đánh số từ 1 đến n
- Đồ thị G(V,E) được biểu diễn máy bởi ma trận lân cận A.
  - Nếu Aij = 1 -> có 1 cung đi từ i tới j bằng 1.
  - Nếu Aij = 0 -> vẫn có thể tồn tại đường từ i tới j qua một số đỉnh khác
- Có hay không đường đi từ đỉnh i đến j ?
  - Không
  - Có ít nhất một đường

Ma trận đường đi (path matrix)

- Ma trận đường đi của đồ thị G được định nghĩa như sau
- $P = A \vee A^{(2)} \vee A^{(3)} \vee \ldots \vee A^{(n)}$
- $A^{(2)} = A \wedge A$
- v: là toán tử hoặc (tuyển) OR
- ^ : là toán tử và (hội) AND
- $P_{ij} = 1$: có đường đi từ i đến j
- $P_{ij} = 0$: không có đường đi từ i đến j
- Thuật toán WARSHALL tính ma trận P

A	B	A v B	A^B
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

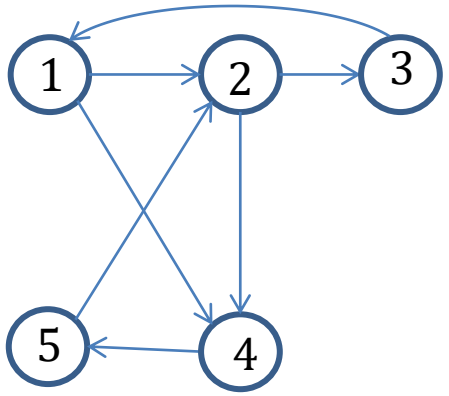
Thuật toán WARSHALL

Procedure WARSHALL (A,n,P)

```
1. For i = 1 to n do
        For j = 1 to n do
            P[i,j] = A[i,j];
2. For k = 1 to n do
        For i = 1 to n do
            For j = 1 to n do
                P[i,j] = P[i,j] v (P[i,k] ^ P[k,j]);
3. Return
```

Ví dụ : Tìm đường đi

- Cho đồ thị G có ma trận lân cận A có dạng như sau
- Tìm $A^{(2)}_{ij}$? Có tồn tại đường đi từ i đến j có độ dài bằng 2 không?



A =

i \ j	1	2	3	4	5
1	0	1	0	1	0
2	0	0	1	1	0
3	1	0	0	0	0
4	0	0	0	0	1
5	0	1	0	0	0

Ví dụ: Tìm đường đi

- Xét hai ma trận A, B kích thước nxn
- $C = A \vee B \Rightarrow C_{ij} = A_{ij} \vee B_{ij}$
- $D = A \wedge B \Rightarrow D_{ij} = \vee_{k=1}^{n} (A_{ik} \wedge B_{kj})$
- Ta có : $A^{(2)} = A \wedge A$; $P = \vee_{k=1}^{n} A^{(k)}$
- Nếu $A^{(2)}_{ij} = 1$: có đường đi từ i đến j có độ dài bằng 2
- Nếu $A^{(2)}_{ij} = 0$: Không có đường đi từ i đến j có độ dài bằng 2
- $P_{ij} = 1$: có đường đi từ ij không xác định độ dài

Ví dụ: Tìm đường đi

A

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

$A^{(2)}$ =

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

$A^{(3)}$ =

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

$A^{(4)}$ =

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

$A^{(5)}$ =

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

P =

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Tất cả các phần tử của P đều bằng 1 => Đồ thị G liên thông mạnh

Bài toán sắp xếp tô pô

- Có n công việc đánh số từ 1 đến n
- Quan hệ giữa các công việc : thực hiện trước “<“
- Nếu i < j : i phải được thực hiện trước j
- Giả sử có 8 công việc sắp xếp như sau

2 < 3	4 < 3	6 < 1
2 < 4	4 < 5	7 < 2
3 < 6	5 < 6	8 < 2

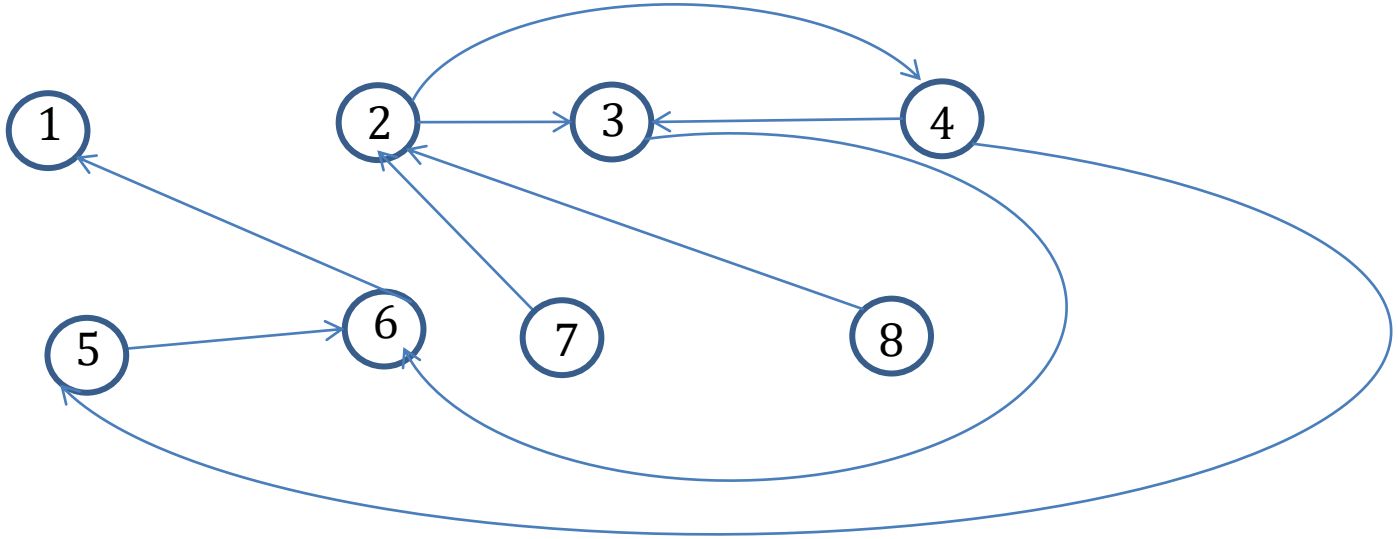
8	7	2	4	5	3	6	1	(3)
7	8	2	4	3	5	6	1	(4)

Bài toán sắp xếp tô pô

- Tính chất thứ tự bộ phận (partial order)
  - Nếu x < y , y < z thì x < z (bắc cầu)
  - Nếu x < y thì không tồn tại y < x (không đối xứng)
  - Không tồn tại x < x (không phản xạ)
- S chứa các phần tử có tính chất thứ tự bộ phận, S được gọi là tập có thứ tự bộ phận
- Bài toán sắp xếp tô pô là sắp xếp các phần tử của S theo một thứ tự tuyến tính sao cho thứ tự các bộ phận vẫn được đảm bảo

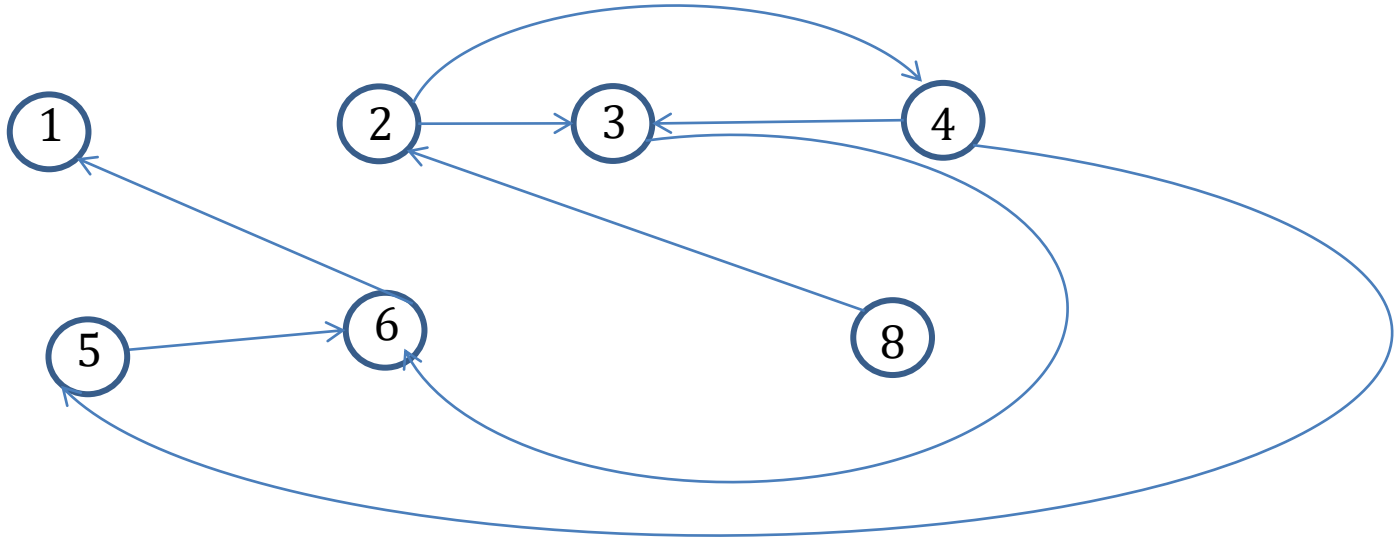
Bài toán sắp xếp tô pô

- Biểu diễn tập S bằng một đồ thị định hướng theo quy tắc sau
  - Mỗi phần tử của S là một đỉnh
  - Phần tử i trước phần tử j sẽ có một cung có hướng trên đồ thị (i, j) trên đồ thị

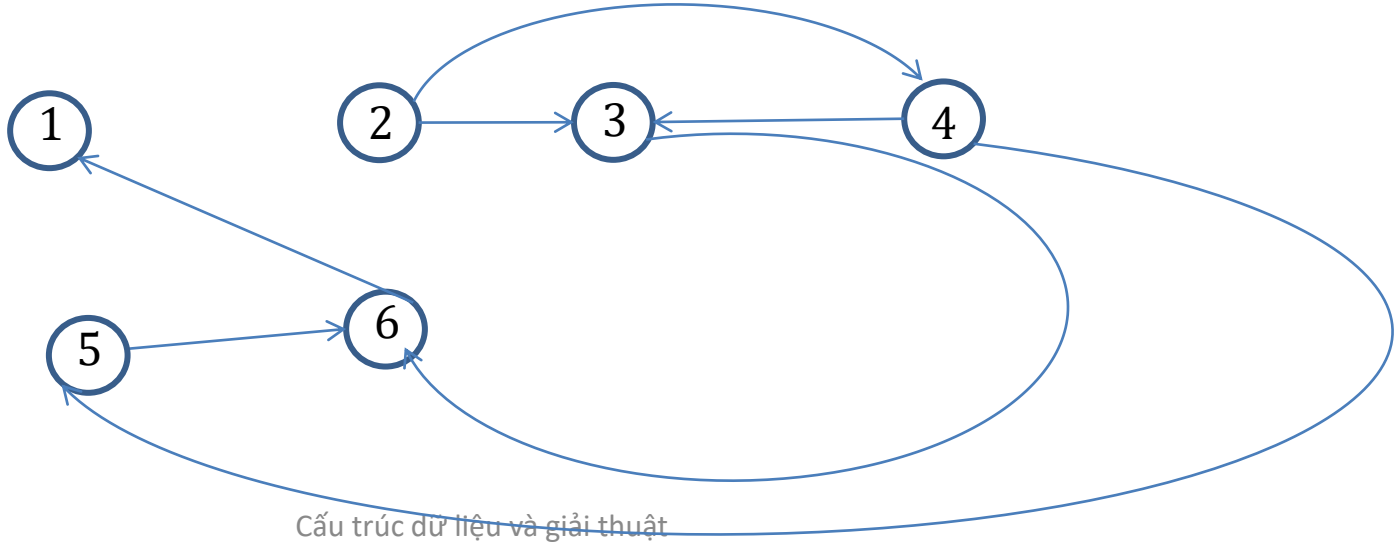


Sắp xếp tô pô

7

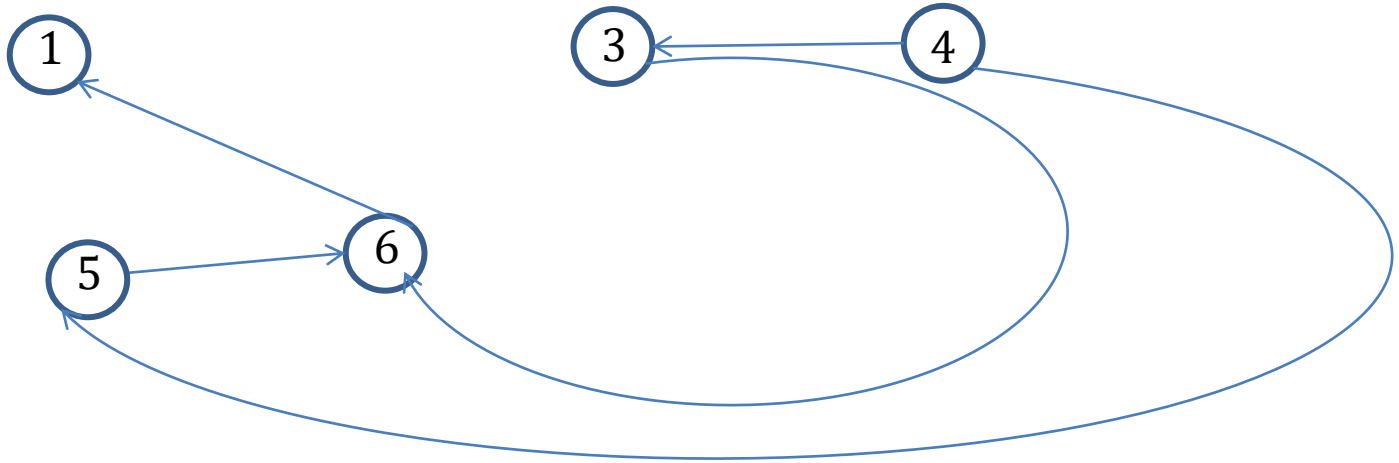


8

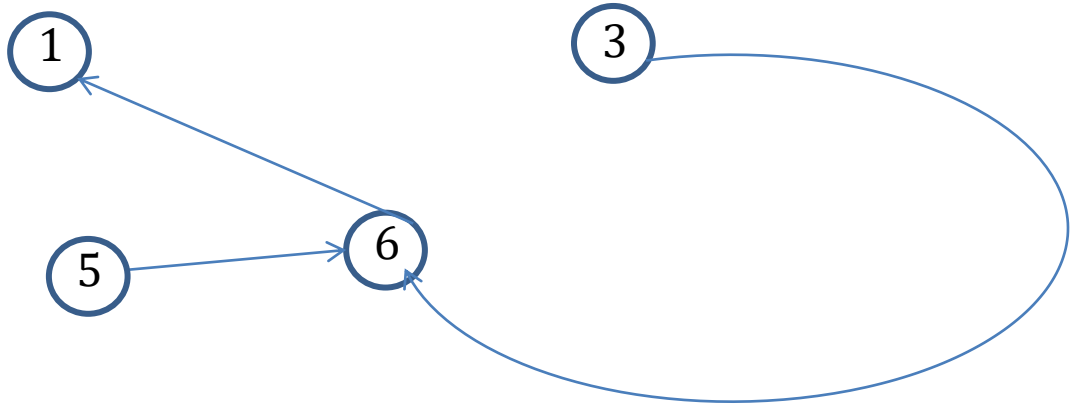


Sắp xếp tô pô

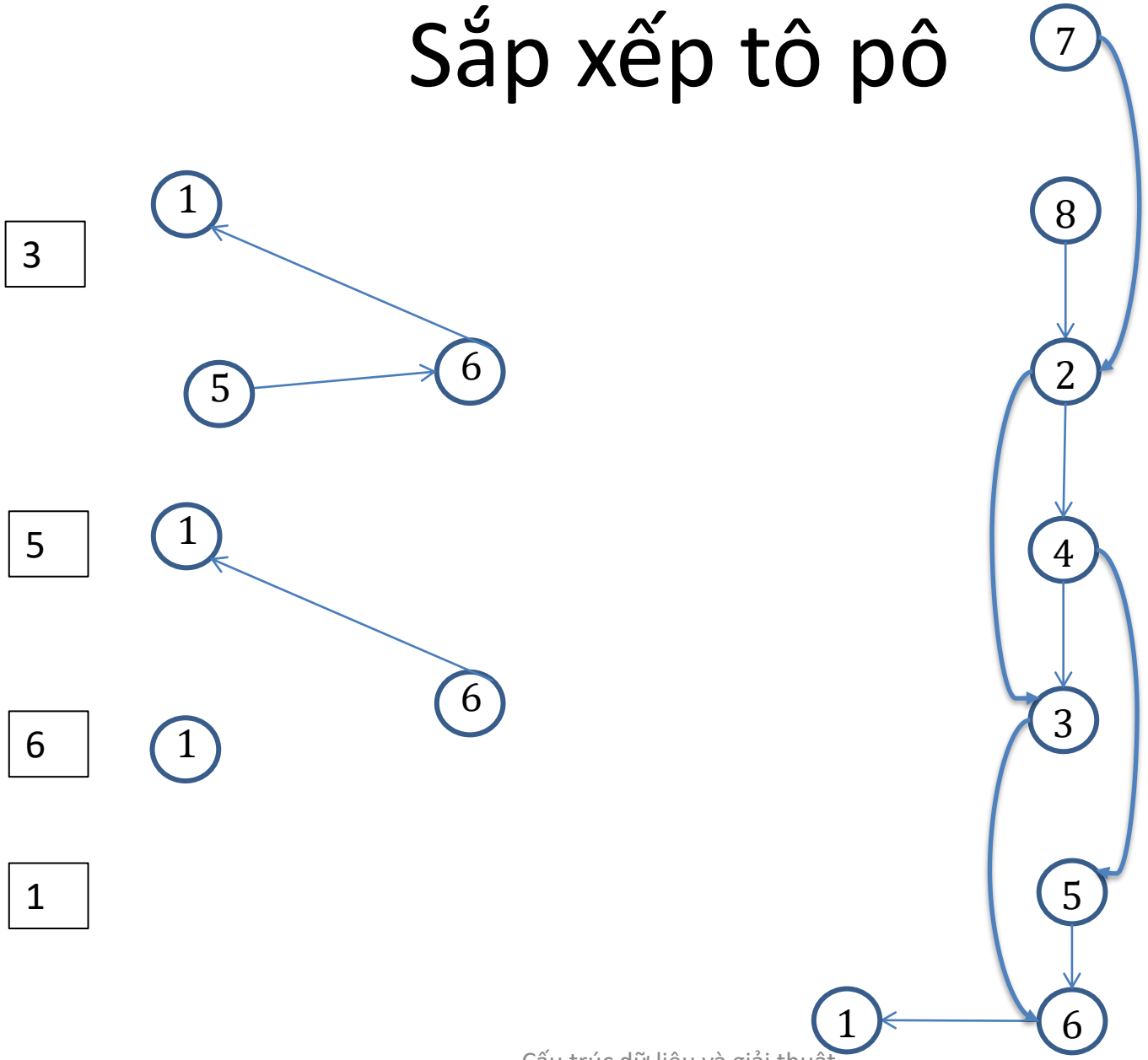
2



4



Sắp xếp tô pô



Bài tập

- Một tập S gồm 7 công việc được đánh số từ 1-7 với quan hệ thực hiện trước, thoả mãn : 7<5, 5<3,3<6,5<6,2<1,2<6, 4<2
  - Hãy vẽ đồ thị biểu diễn S
  - Minh hoạ giải thuật sắp xếp topo bằng hình vẽ
  - Nêu kết quả thực hiện giải thuật Topo – order

Bài toán tìm đường đi ngắn nhất

Bài toán: Tìm đường đi ngắn nhất từ một đỉnh nguồn s đến tất cả các đỉnh còn lại trên đồ thị với trọng số không âm trên các cạnh.

Trong thuật toán, mỗi đỉnh v được lưu trữ các thông tin sau:

- k[v]: Biến logic có giá trị True nếu đã tìm được đường đi ngắn nhất từ s đến v, ban đầu biến này được khởi tạo là False
- d[v]: khoảng cách ngắn nhất hiện biết từ s đến v. Ban đầu biến này được khởi tạo giá trị ∞ với mọi đỉnh, ngoại trừ d[s] được đặt bằng 0
- p[v]: là đỉnh đi trước v trong đường đi có độ dài d[v]. Ban đầu các biến này được khởi tạo rỗng (chưa biết)

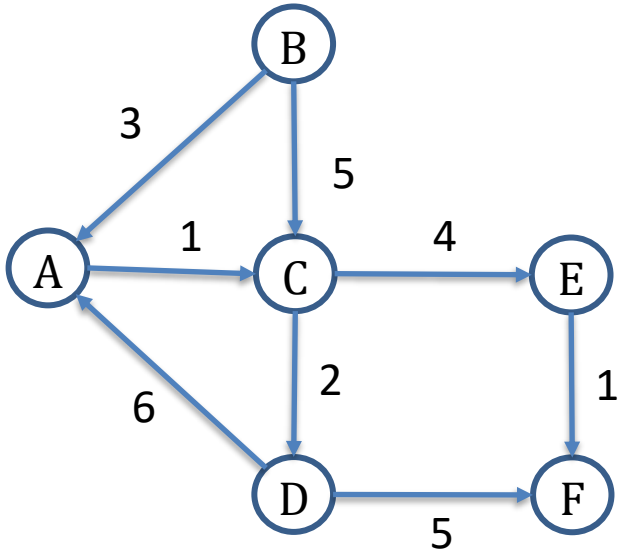
Thuật toán Dijkstra

Lặp lại các thao tác sau đây cho đến khi tất cả các đỉnh được khảo sát xong (nghĩa là k[v] bằng True với mọi v):

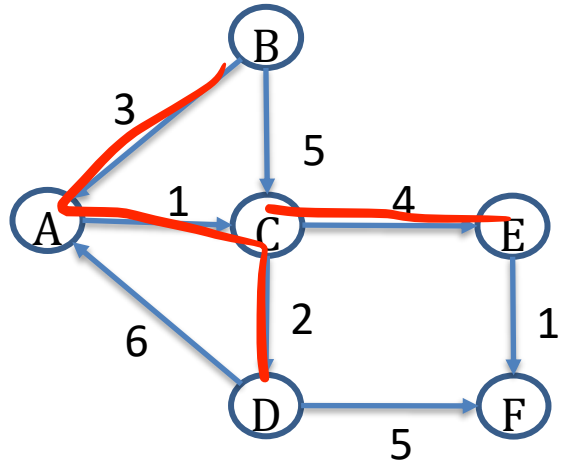
- Trong tập đỉnh với k[v] =False, chọn đỉnh v có d[v] là nhỏ nhất.
- Đặt k[v]=True.
- Với mỗi đỉnh w kề với v và có k[w]=False, ta kiểm tra d[w]>d[v] + c(v,w). Nếu đúng thì đặt lại d[w]=d[v]+c(v,w) và đặt p[w]=v. ở đây c(v,w) là khoảng cách từ v đến w.

Thuật toán Dijkstra

Ví dụ: Tìm đường đi ngắn nhất từ đỉnh B tới các đỉnh còn lại



Thuật toán Dijkstra

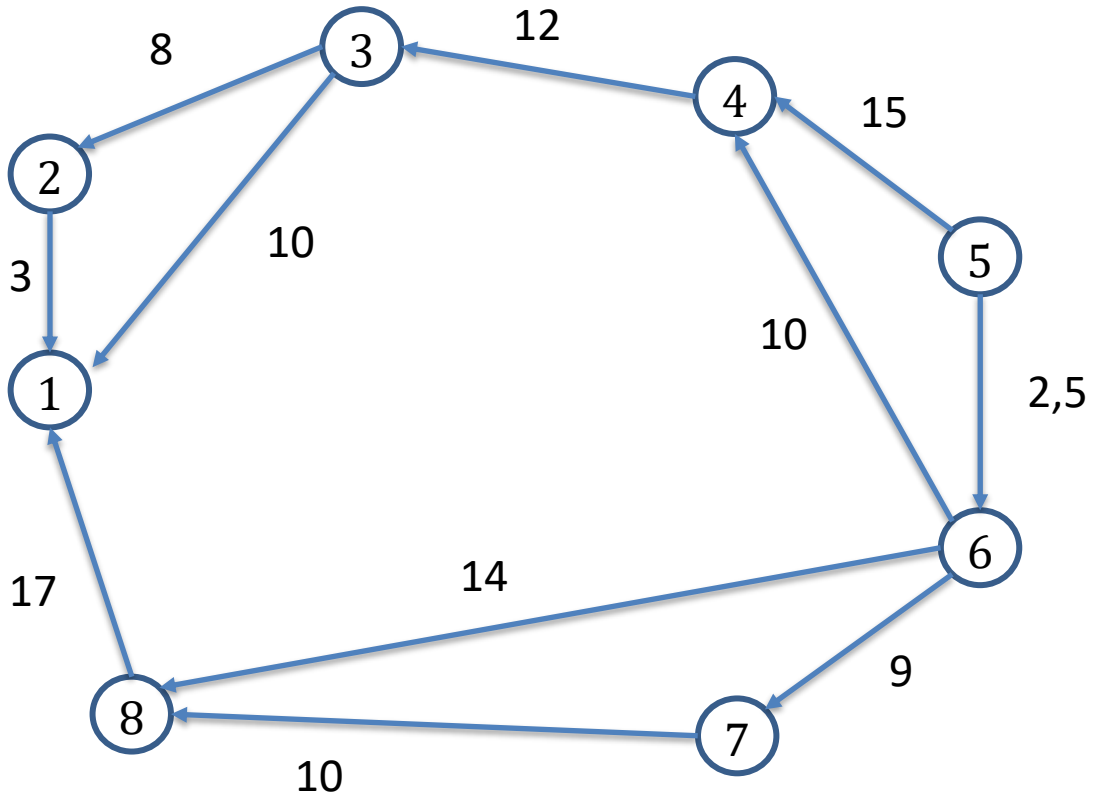


Bước lặp	A			B			C			D			E			F		
	d	p	k	d	p	k	d	p	k	d	p	k	d	p	k	d	p	k
Khởi tạo	∞	-	F	0	-	F	∞	-	F	∞	-	F	∞	-	F	∞	-	F
1	3	B	F	**0**	**-**	**T**	5	B	F	∞	-	F	∞	-	F	∞	-	F
2	**3**	**B**	**T**				4	A	F	∞	-	F	∞	-	F	∞	-	F
3							**4**	**A**	**T**	6	C	F	8	C	F	∞	-	F
4										**6**	**C**	**T**	8	C	F	11	D	F
5													**8**	**C**	**T**	9	E	F
6																**9**	**E**	**T**

```
Procedure Dijkstra (G,s)
For v ∈ V do{
    d[v]=∞;
    p[v]=null;
    k[v]=false;}
d[s]=0;
T=V;
While T≠ θ do {
    v = dinh co d[v] nho nhat trong
T;
    k[v]=true;
    T=T-{v};
    for (w∈Adj(v))&&!k[w] do{
        if d[w]>d[v]+c[w,v]{
            d[w]=d[v]+c[w,v];
            p[w]=v;
        }
    }
}
```

Thuật toán Dijkstra

Ví dụ: Tìm đường đi ngắn nhất từ đỉnh 5 tới các đỉnh còn lại



Bước lặp	1			2			3			4			5			6			7			8		
	d	p	k	d	p	k	d	p	k	d	p	k	d	p	k	d	p	k	d	p	k	d	p	k
Khởi tạo	∞	-	F	∞	-	F	∞	-	F	∞	-	F	0	-	F	∞	-	F	∞	-	F	∞	-	F
1	∞	-	F	∞	-	F	∞	-	F	15	5	F	**0**	**-**	**T**	2,5	5	F	∞	-	F	∞	-	F
2	∞	-	F	∞	-	F	∞	-	F	12,5	6	F				**2,5**	**5**	**T**	11,5	6	F	16,5	6	F
3	∞	-	F	∞	-	F	∞	-	F	12,5	6	F							**11,5**	**6**	**T**	16,5	6	F
4	∞	-	F	∞	-	F	24,5	4	F	**12,5**	**6**	**T**										16,5	6	F
5	33,5	8	F	∞	-	F	24,5	4	F													**16,5**	**6**	**T**
6	33,5	8	F	32,5	3	F	**24,5**	**4**	**T**															
7	33,5	8	F	**32,5**	**3**	**T**																		
8	33,5	8	T																					