

Tên bài giảng Đồ thị

Môn học: **Thuật toán và ứng dụng**
Chương: 4
Hệ: Đại học
Giảng viên: TS. Phạm Đình Phong
Email: phongpd@utc.edu.vn

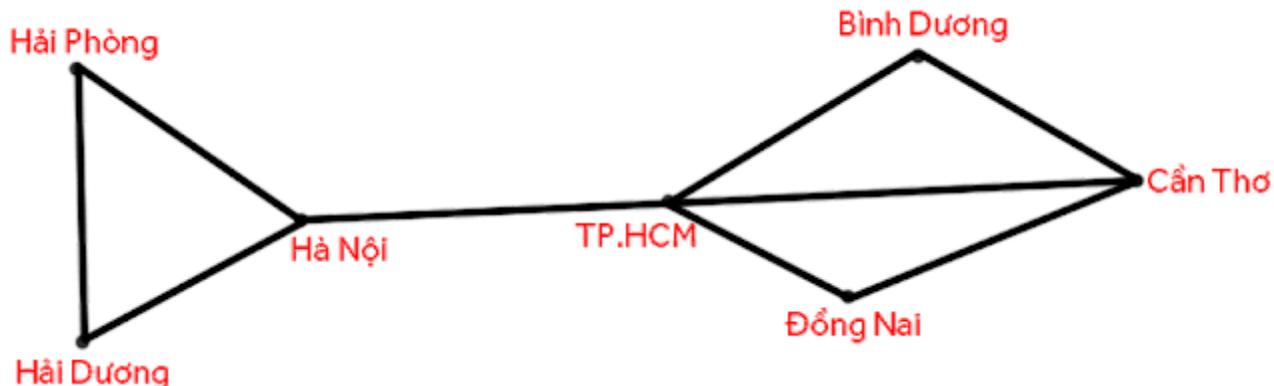
Nội dung bài học

- 1. Đồ thị vô hướng, có hướng**
- 2. Thành phần liên thông**
- 3. Đường đi trên đồ thị, đường đi ngắn nhất, tô màu đồ thị**
- 4. Cây bao trùm**
- 5. Luồng cực đại**

Đồ thị vô hướng, có hướng

- Khái niệm đồ thị

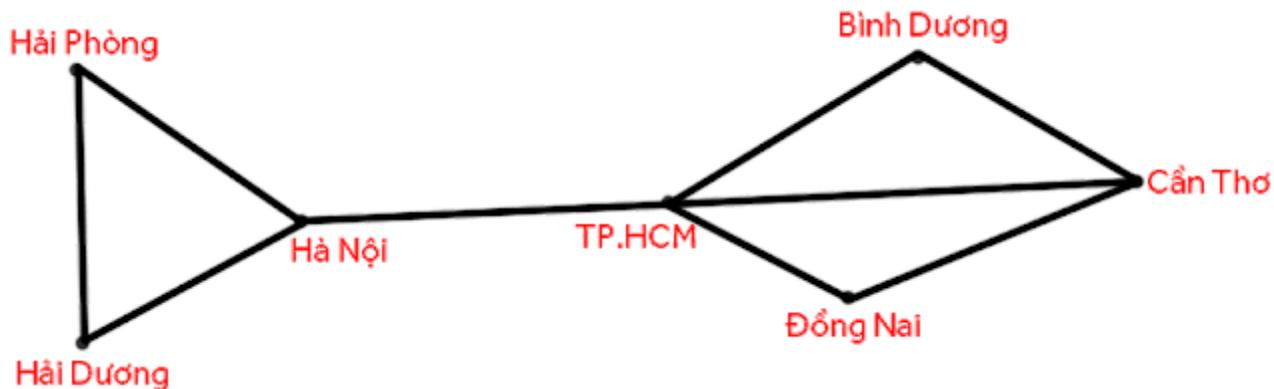
- Đồ thị là một cấu trúc dữ liệu rời rạc bao gồm các đỉnh và các cạnh nối các cặp đỉnh này
- Phân biệt các loại đồ thị khác nhau bởi **kiểu** và **số lượng cạnh** nối hai đỉnh nào đó của đồ thị



- Hình trên là một mạng máy tính được biểu diễn bởi một đồ thị

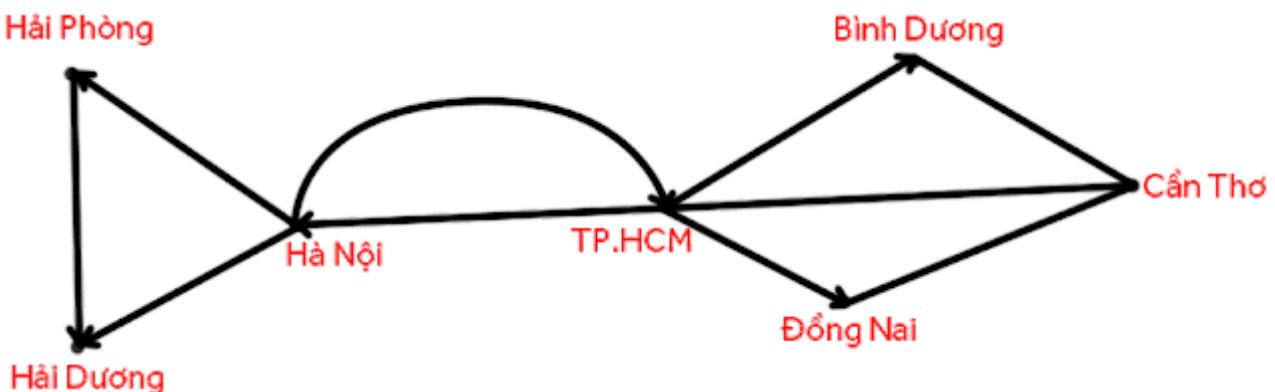
Đồ thị vô hướng, có hướng

- Đồ thị vô hướng
 - G là một cặp không có thứ tự $G = (V, E)$
 - V là tập các đỉnh phân biệt, E là tập các cặp không thứ tự chứa các đỉnh được gọi là cạnh



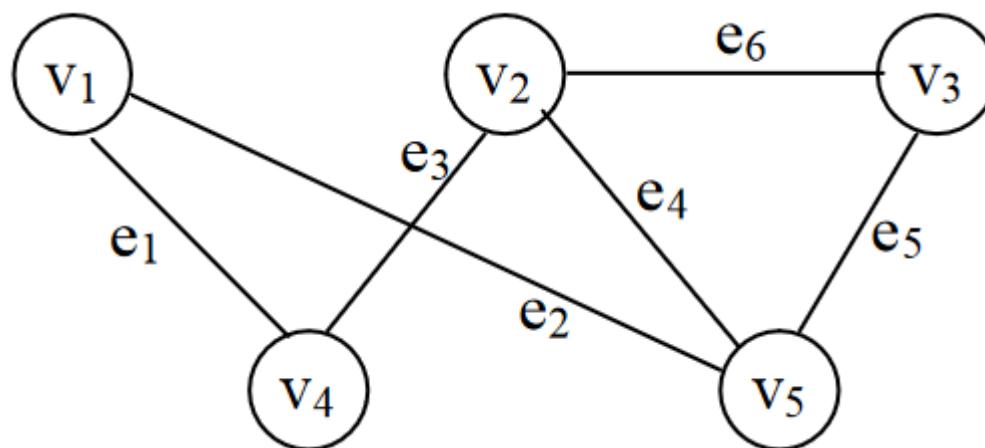
Đồ thị vô hướng, có hướng

- Đồ thị có hướng
 - G là một cặp có thứ tự $G = (V, E)$
 - V là tập các đỉnh, E là tập các cặp có thứ tự chứa các đỉnh được gọi là các cạnh có hướng hoặc cung
 - Một cạnh $e = (x, y)$ được coi là có hướng từ x tới y ; x được gọi là điểm đầu/gốc và y được gọi là điểm cuối/ngọn của cạnh



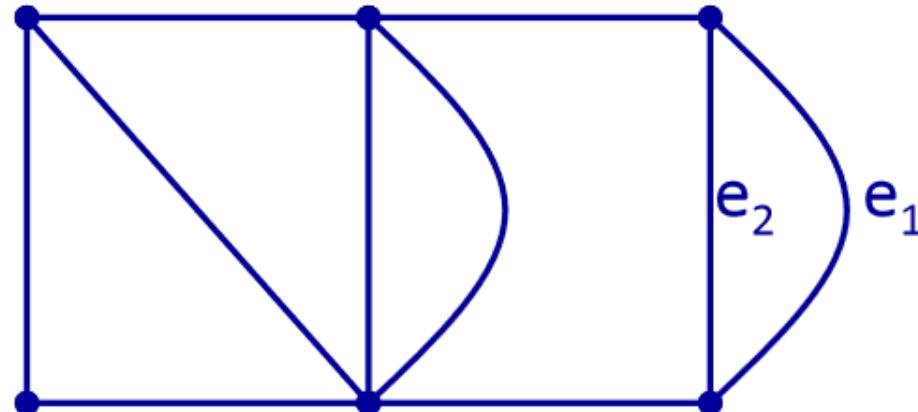
Đồ thị vô hướng, có hướng

- Đơn đồ thị vô hướng
 - Đơn đồ thị vô hướng $G = (V, E)$ bao gồm V là tập các đỉnh khác rỗng và E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh



Đồ thị vô hướng, có hướng

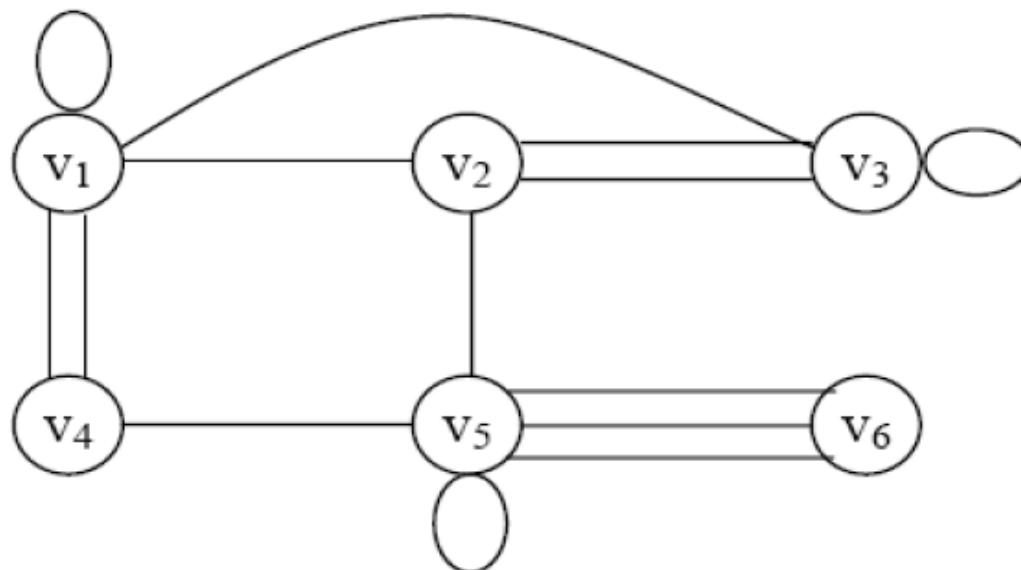
- Đa đồ thị vô hướng
 - Đa đồ thị vô hướng $G = (V, E)$ bao gồm V là tập các đỉnh khác rỗng, E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh. Hai cạnh e_1 và e_2 được gọi là cạnh lặp (bội hay song song) nếu chúng cùng tương ứng với một cặp đỉnh



Đồ thị vô hướng, có hướng

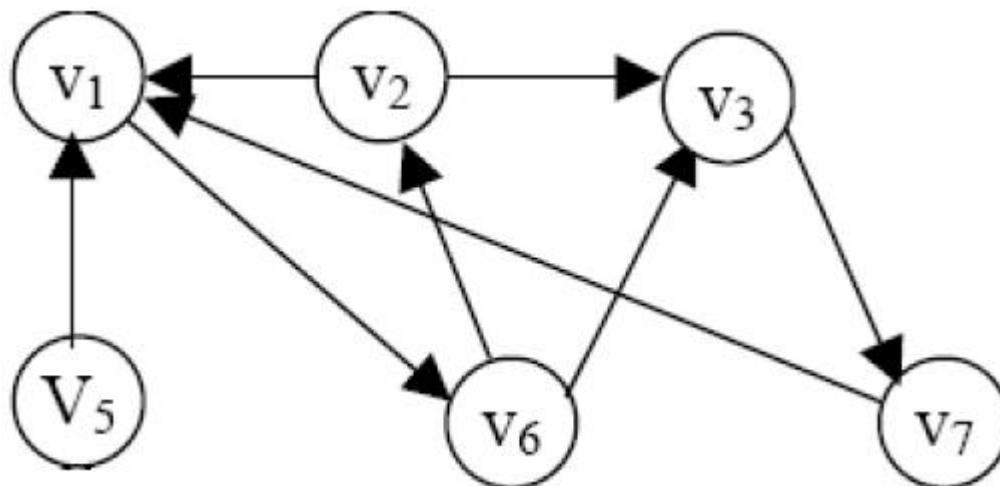
- Giả đồ thị vô hướng

- Giả đồ thị vô hướng $G = (V, E)$ bao gồm V là tập các đỉnh khác rỗng và E là tập các cặp không có thứ tự gồm hai phần tử (không nhất thiết phải khác nhau) của V gọi là cạnh
- Tập các cạnh bao gồm các cặp đỉnh trùng nhau được gọi là các khuyên



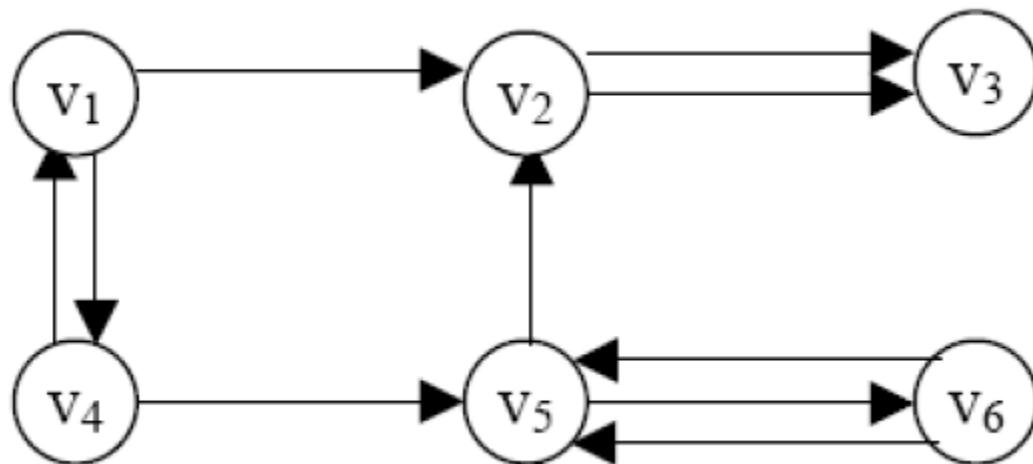
Đồ thị vô hướng, có hướng

- Đơn đồ thị có hướng
 - Đơn đồ thị có hướng $G = (V, E)$ bao gồm V là tập các đỉnh khác rỗng và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung



Đồ thị vô hướng, có hướng

- Đa đồ thị có hướng
 - Đa đồ thị có hướng $G = (V, E)$ bao gồm V là tập các đỉnh khác rỗng và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung. Hai cung e_1, e_2 tương ứng với cùng một cặp đỉnh được gọi là cung lặp



Đồ thị vô hướng, có hướng

- Các loại đồ thị

Loại đồ thị	Cạnh	Có cạnh bội	Có khuyên
Đơn đồ thị vô hướng	Vô hướng	Không	Không
Đa đồ thị vô hướng	Vô hướng	Có	Không
Giả đồ thị vô hướng	Vô hướng	Có	Có
Đồ thị có hướng	Có hướng	Không	Có
Đa đồ thị có hướng	Có hướng	Có	Có

Thành phần liên thông

- Đường đi
 - **Đường đi** độ dài n từ đỉnh u đến đỉnh v trên đồ thị vô hướng $G = \langle V, E \rangle$ là dãy: $x_0, x_1, \dots, x_{n-1}, x_n$ trong đó n là số nguyên dương, $x_0 = u$, $x_n = v$, $(x_i, x_{i+1}) \in E$, $i = 0, 1, 2, \dots, n - 1$
 - Có thể biểu diễn đường đi thành dãy các cạnh:
 $(x_0, x_1), (x_1, x_2), \dots, (x_{n-1}, x_n)$
 - Đỉnh u là đỉnh đầu, đỉnh v là đỉnh cuối của đường đi
 - Đường đi có đỉnh đầu trùng với đỉnh cuối ($u = v$) được gọi là **chu trình**
 - Đường đi hay chu trình được gọi là đơn nếu như không có cạnh nào lặp lại

Thành phần liên thông

• Liên thông

- Đồ thị vô hướng được gọi là **liên thông** nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó
- Ta gọi **đồ thị con** của đồ thị $G = (V, E)$ là đồ thị $H = (W, F)$, trong đó $W \subseteq V$ và $F \subseteq E$
- Trong trường hợp đồ thị là **không liên thông**, nó sẽ rã ra thành một số **đồ thị con liên thông** không có đỉnh chung. Những **đồ thị con liên thông** như vậy ta sẽ gọi là các **thành phần liên thông** của đồ thị

Thành phần liên thông

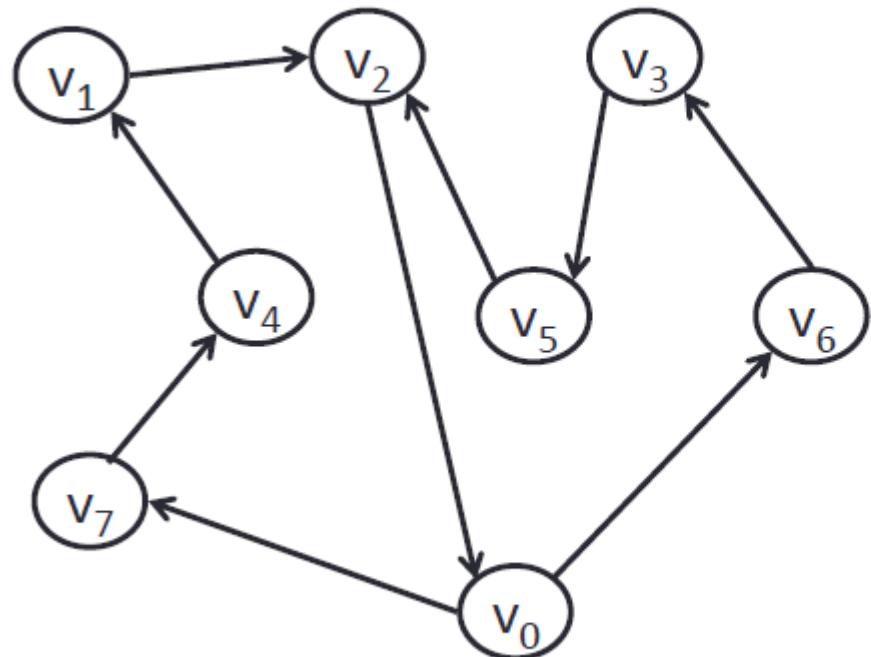
- Biểu diễn đồ thị
 - **Ma trận kè** (thích hợp cho các đồ thị dày)
 - Ma trận $A = \{a_{i,j} : i, j = 1, 2, \dots, n\}$ với $a_{i,j} = 0$, nếu $(i, j) \notin E$ và $a_{i,j} = 1$, nếu $(i, j) \in E$, $i, j = 1, 2, \dots, n$ gọi là ma trận kè của đồ thị G
 - Tính chất ma trận kè của đồ thị vô hướng
 - Tính đối xứng: $a_{i,j} = a_{j,i}$, $i, j = 1, 2, \dots, n$
 - Tổng các phần tử trên dòng i (cột j) bằng bậc của đỉnh i (đỉnh j)
 - Tính chất ma trận kè của đồ thị có hướng
 - Không có tính đối xứng
 - Tổng các phần tử trên dòng i bằng bán bậc ra của đỉnh i ($\deg^+(i)$) và tổng các phần tử trên cột j bằng bán bậc vào của đỉnh j ($\deg^-(j)$)

Thành phần liên thông

- Biểu diễn đồ thị
 - **Danh sách kề**
 - Biểu diễn ma trận kề lãng phí không gian khi đồ thị thừa (có rất ít cạnh) do cần n^2 ô nhớ
 - VD: Mỗi nút giao thông thường chỉ có 3-4 con đường giao nhau \rightarrow số cạnh = $O(n)$
 - Danh sách kề là cách tiếp cận hiệu quả để lưu trữ đồ thị thừa
 - Đỉnh v kề với đỉnh u nếu $(u, v) \in E$
 - Mỗi đỉnh giữ một danh sách các đỉnh kề với nó (các cạnh đi ra). Nếu các cạnh có trọng số thì cũng lưu trữ các trọng số vào danh sách kề này
 - Yêu cầu không gian $O(|V|+|E|)$

Thành phần liên thông

- Biểu diễn đồ thị
 - Danh sách kè



0	6, 7
1	2
2	0
3	5
4	1
5	2
6	3
7	4

Thành phần liên thông

- Biểu diễn đồ thị
 - **Ma trận liên thuộc**
 - Trong một số trường hợp, chúng ta có thể dùng ma trận liên thuộc để biểu diễn một đồ thị có hướng $G = (V, E)$

$$b_{ij} = \begin{cases} -1 & \text{nếu cạnh } j \text{ đi ra khỏi đỉnh } i, \\ 1 & \text{nếu cạnh } j \text{ đi vào đỉnh } i, \\ 0 & \text{ngược lại} \end{cases}$$

Thành phần liên thông

- Duyệt đồ thị
 - Là cách đi qua tất cả các đỉnh của đồ thị sao cho mỗi đỉnh chỉ được thăm một lần
 - Thăm bằng cách đánh số các đỉnh tăng dần
 - Sau khi duyệt, trả về tập các cạnh đã đi qua
 - Có hai cách duyệt
 - Theo chiều rộng
 - Theo chiều sâu

Thành phần liên thông

- Tìm kiếm theo chiều rộng
 - Thăm tất cả các đỉnh kề với đỉnh u trước khi tiếp tục với các đỉnh khác
 - sử dụng một cấu trúc dữ liệu hàng đợi (queue) để lưu trữ thông tin trung gian thu được trong quá trình tìm kiếm
 - 1. Thêm đỉnh gốc vào queue và đánh dấu đỉnh gốc.
 - 2. Nếu queue chưa rỗng, lấy ra đỉnh u đầu tiên khỏi queue. Xét các đỉnh v kề với đỉnh u
 - Nếu đỉnh v đã được đánh dấu thì bỏ qua.
 - Nếu v chưa được đánh dấu thì thêm đỉnh v vào queue và đánh dấu đỉnh v .
 - 3. Nếu queue rỗng, dừng quá trình tìm kiếm

Thành phần liên thông

- Tìm kiếm theo chiều rộng
 - Mã giả của thuật toán BFS

```
void BFS (int u) { //duyệt các đỉnh cùng thành phần liên thông với u
    queue = Ø;
    queue.enqueue(u); //nạp u vào hàng đợi
    chuaxet[u] = false; // đổi trạng thái của u
    while (queue ≠ Ø) { // duyệt tới khi nào hàng đợi rỗng
        p = queue.dequeue(); // lấy p ra từ khỏi hàng đợi
        Tham_dinh(p); // duyệt xong đỉnh p
        for (v ∈ ke(p)) { //duyệt mọi đỉnh v kề với p
            if (chuaxet[v] ) {
                queue.enqueue(v); // đưa v vào hàng đợi
                chuaxet[v] = false; // đổi trạng thái của v
            } //end if
        } //end for
    } //end while
}
```

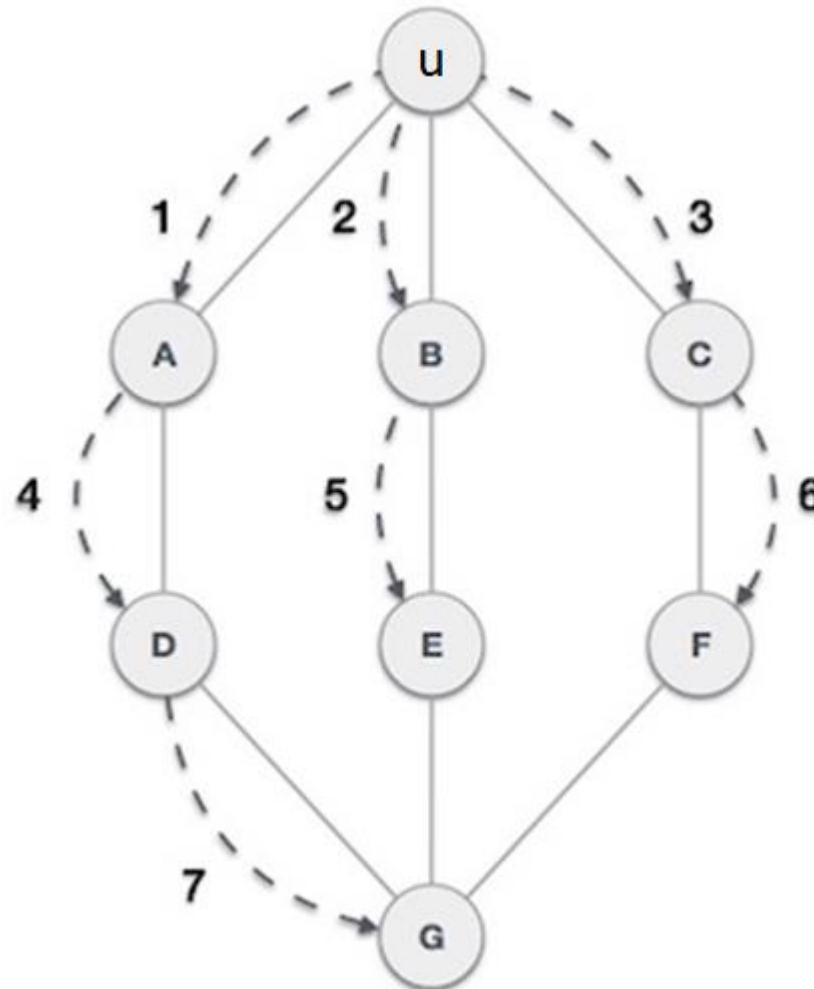
Thành phần liên thông

- Tìm kiếm theo chiều rộng
 - Để thăm tất cả các đỉnh của đồ thị thuộc các thành phần liên thông, thực hiện đoạn chương trình sau

```
for (u = 0; u < n; u++)  
    chuaxet[u] = true;  
for (u ∈ V)  
    if (chuaxet[u])  
        BFS(u);
```

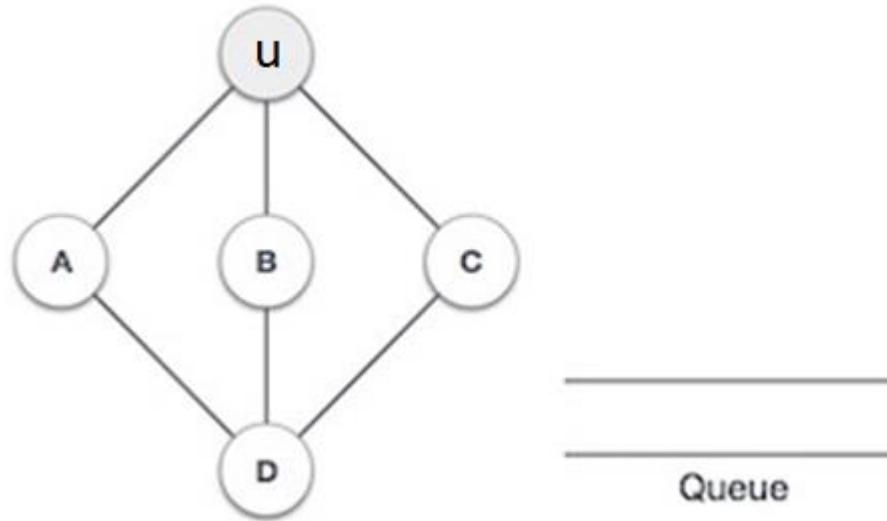
Thành phần liên thông

- Tìm kiếm theo chiều rộng
 - Ví dụ 1:



Thành phần liên thông

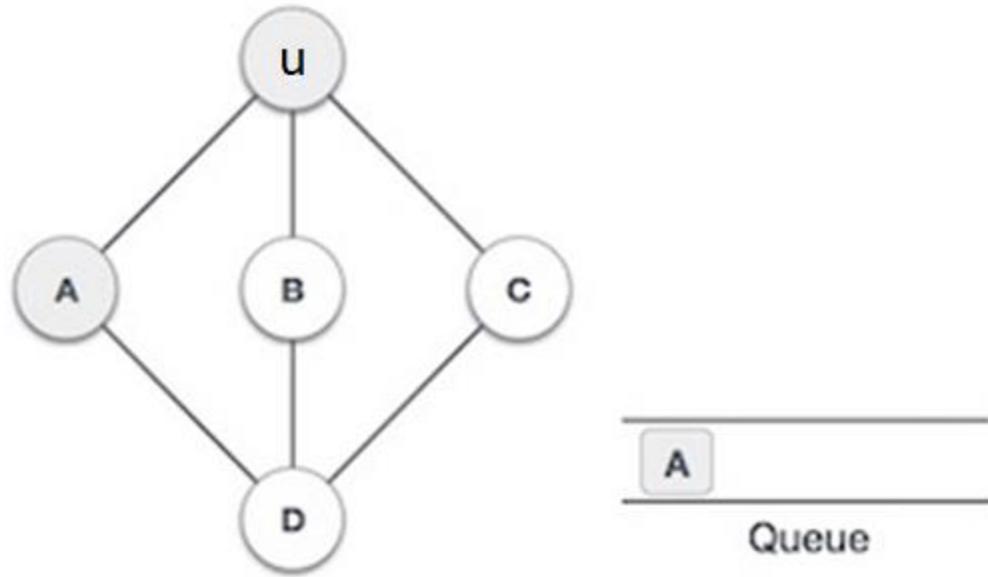
- Tìm kiếm theo chiều rộng
 - Ví dụ 2:



- Khởi tạo hàng đợi rỗng
- Bắt đầu duyệt đỉnh u (đỉnh bắt đầu) và đánh dấu đỉnh này là đã duyệt.

Thành phần liên thông

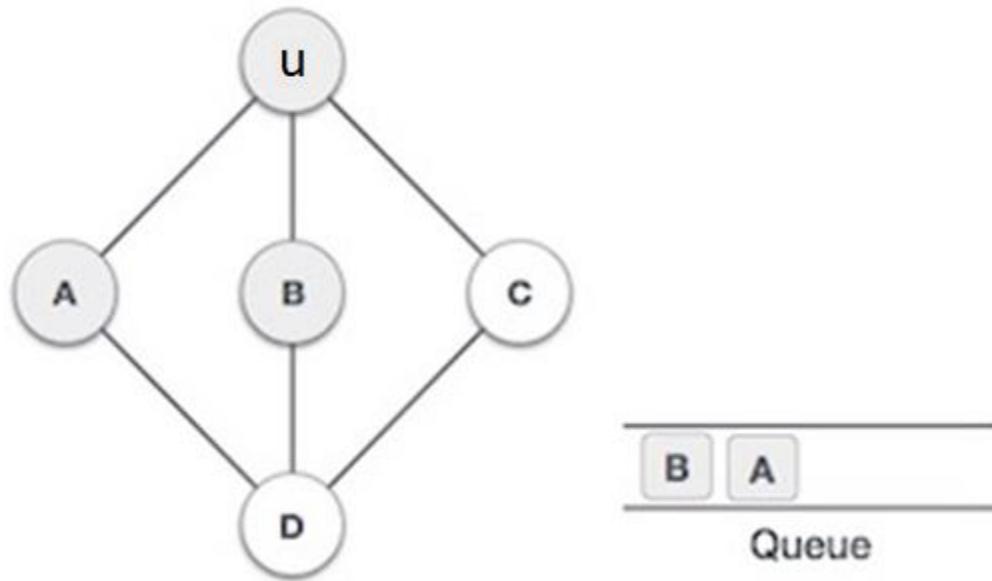
- Tìm kiếm theo chiều rộng
 - Ví dụ 2:



- Tìm đỉnh liền kề với u mà chưa được duyệt \rightarrow có 3 đỉnh và theo thứ tự từ trái qua phải chọn đỉnh A đánh dấu là đã duyệt và xếp A vào hàng đợi

Thành phần liên thông

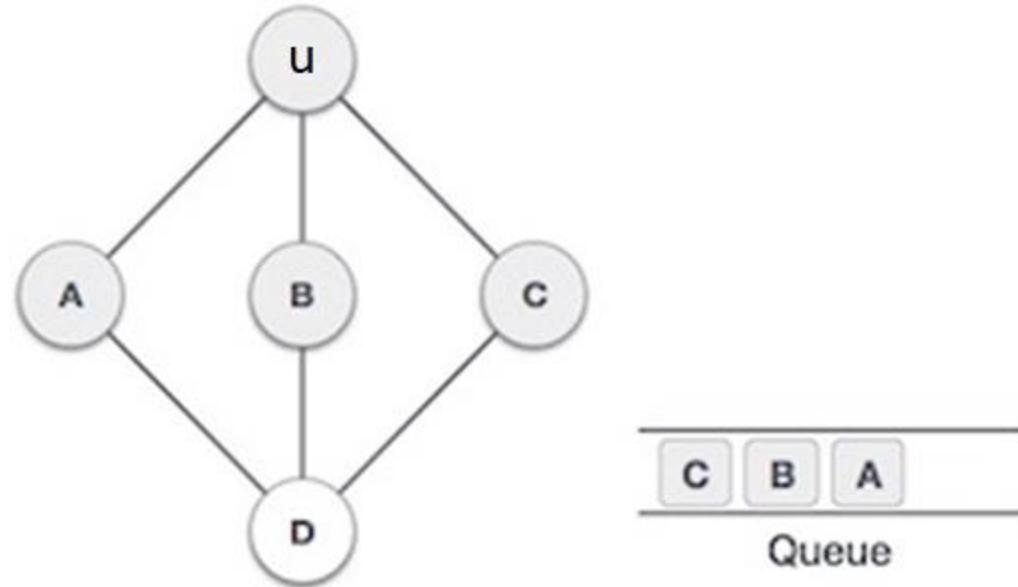
- Tìm kiếm theo chiều rộng
 - Ví dụ 2:



- Tiếp tục duyệt đỉnh liền kề với u là B. Đánh dấu là đã duyệt và xếp đỉnh B vào hàng đợi

Thành phần liên thông

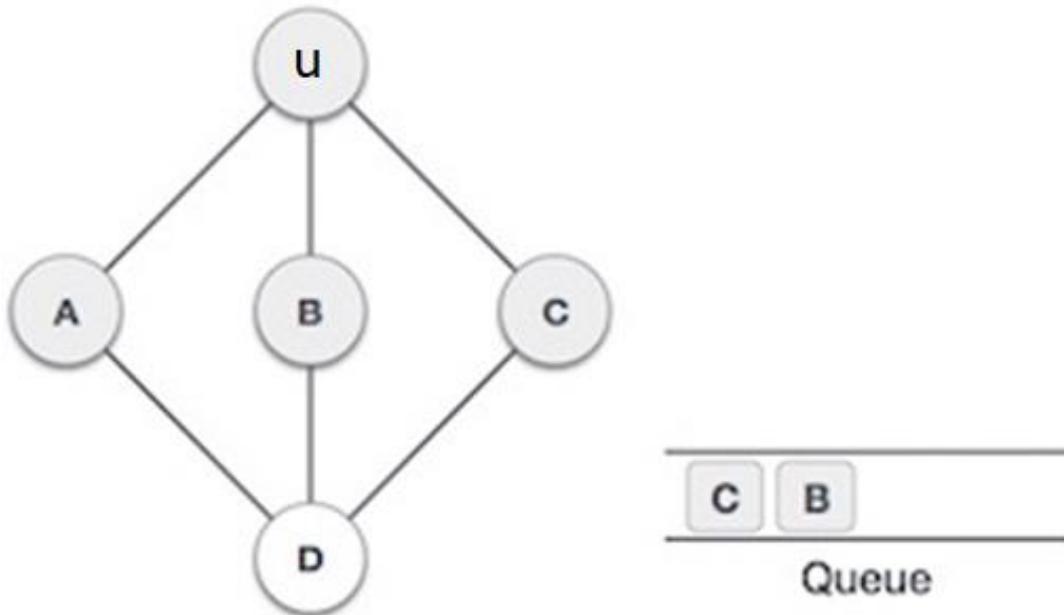
- Tìm kiếm theo chiều rộng
 - Ví dụ 2:



- Tiếp tục duyệt đỉnh liền kề với u là C. Đánh dấu là đã duyệt và xếp đỉnh này vào hàng đợi

Thành phần liên thông

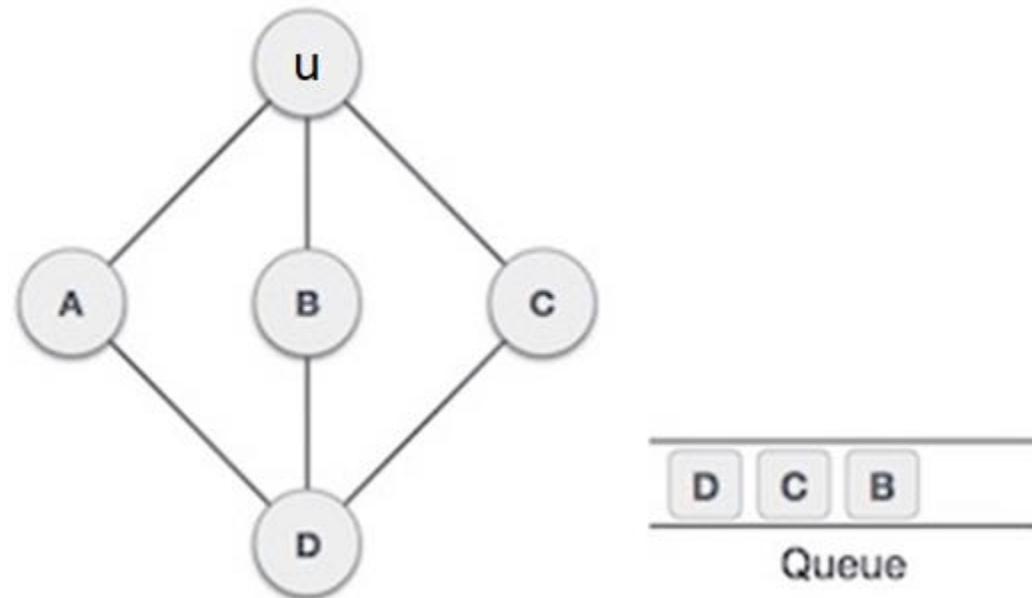
- Tìm kiếm theo chiều rộng
 - Ví dụ 2:



- Bây giờ đỉnh u không còn đỉnh nào liền kề mà chưa được duyệt → rút A từ hàng đợi

Thành phần liên thông

- Tìm kiếm theo chiều rộng
 - Ví dụ 2:



- Từ đỉnh A → có đỉnh liền kề là D và là đỉnh chưa được duyệt. Đánh dấu đỉnh D là đã duyệt và xếp vào hàng đợi
- Thuật toán tiếp tục cho đến khi hàng đợi rỗng

Thành phần liên thông

- Tìm kiếm theo chiều rộng
 - **Phân tích thuật toán:**
 - **Không gian:** $2|V|$, với V là tập các đỉnh của đồ thị và $|V|$ là số đỉnh của đồ thị
 - **Thời gian:** $O(|E| + |V|)$, với V và E là tập các đỉnh và cung của đồ thị vì trong trường hợp xấu nhất, mỗi đỉnh và cung của đồ thị được thăm đúng một lần. $O(|E| + |V|)$ nằm trong khoảng từ $O(|V|)$ đến $O(|V|^2)$, tùy theo số cung của đồ thị

Thành phần liên thông

- Tìm kiếm theo chiều rộng
 - **Ưu điểm**
 - Xét duyệt tất cả các đỉnh để trả về kết quả
 - Nếu số đỉnh là hữu hạn, thuật toán chắc chắn tìm ra kết quả
 - **Nhược điểm**
 - Mang tính chất vét cạn, không nên áp dụng nếu duyệt số đỉnh quá lớn
 - Mang tính chất mù quáng vì duyệt tất cả đỉnh mà không chú ý đến thông tin trong các đỉnh để duyệt hiệu quả, dẫn đến duyệt qua các đỉnh không cần thiết

Thành phần liên thông

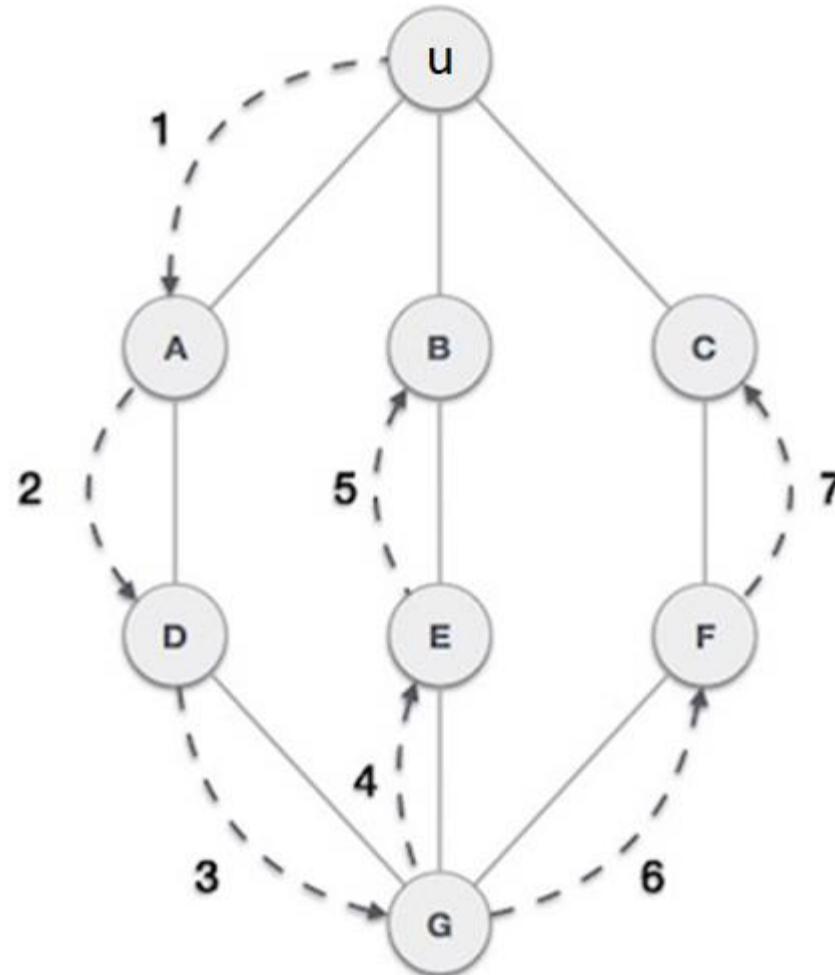
- Tìm kiếm theo chiều rộng
 - **Ứng dụng:**
 - Tìm các thành phần liên thông của đồ thị hoặc kiểm tra đồ thị hai phía
 - Tìm đường đi ngắn nhất và cây bao trùm tối thiểu trong đồ thị không trọng số
 - Tìm tất cả các nút lân cận của một mạng ngang hàng
 - Crawlers in Search Engines: bắt đầu từ một trang nguồn và lẩn theo các liên kết từ trang nguồn đó, và làm tương tự trong các trang được lẩn tới
 - Các trang website mạng xã hội: tìm các người dùng kết nối tới một người dùng cụ thể với khoảng cách k
 - Hệ thống định vị GPS: tìm các địa điểm lân cận
 - Quảng bá thông tin mạng (Broadcasting in Network)

Thành phần liên thông

- Tìm kiếm theo sâu
 - Thăm một đỉnh kề của u , sau đó thăm tiếp một đỉnh kề của đỉnh kề đó
→ Sử dụng ngăn xếp (stack) để ghi nhớ đỉnh liền kề để bắt đầu việc tìm kiếm
 - Thuật toán tiếp tục cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó thuật toán quay lui về đỉnh vừa mới tìm kiếm ở bước trước

Thành phần liên thông

- Tìm kiếm theo sâu
 - Ví dụ:



Thành phần liên thông

- Tìm kiếm theo sâu
 - **Thuật toán không đệ quy**

```
void DFS( int u ) { //Duyệt đỉnh cùng thành phần liên thông với u
    stack = Ø;          //Khởi tạo stack rỗng
    stack.push(u);      //Đưa u vào stack
    while (stack ≠ Ø) { //Lặp cho tới khi stack rỗng
        v = stack.pop(); //Lấy v ra khỏi stack
        Tham_dinh(v);   //Duyệt xong đỉnh v
        if (chuaxet[v]) { //Nếu v chưa được xét
            chuaxet[v] = false; //Đánh dấu v là đã được xét
            for (w ∈ ke(v)) { //Duyệt các đỉnh kề với v
                if (chuaxet[w]) stack.push(w); //Nếu w chưa được xét
            }
        }
    }
}
```

Thành phần liên thông

- Tìm kiếm theo sâu
 - Thuật toán đệ quy

```
void DFS(int u) {  
    Tham_dinh(u);  
    chuaxet[u] = false;  
    for (v ∈ ke(u)) {           //Xét các cạnh kề u  
        if (chuaxet[v]) DFS(v); //Gọi đệ quy để duyệt  
    }  
}
```

Thành phần liên thông

- Tìm kiếm theo sâu
 - Để duyệt tất cả các đỉnh của đồ thị vì có thể có nhiều thành phần liên thông → thực hiện duyệt như sau

```
for (i = 0; i < n; i++)  
    chuaxet[i] = true;  
  
for (i = 0; i < n; i++)  
    if (chuaxet[i] )  
        DFS(i);
```

Thành phần liên thông

- Tìm kiếm theo sâu
 - **Ứng dụng:**
 - Xác định các thành phần liên thông của đồ thị
 - Xác định các thành phần liên thông mạnh của đồ thị có hướng
 - Kiểm tra một đồ thị có phải là đồ thị phẳng hay không
 - Cây bao trùm tối thiểu trên đồ thị có trọng số

Thành phần liên thông

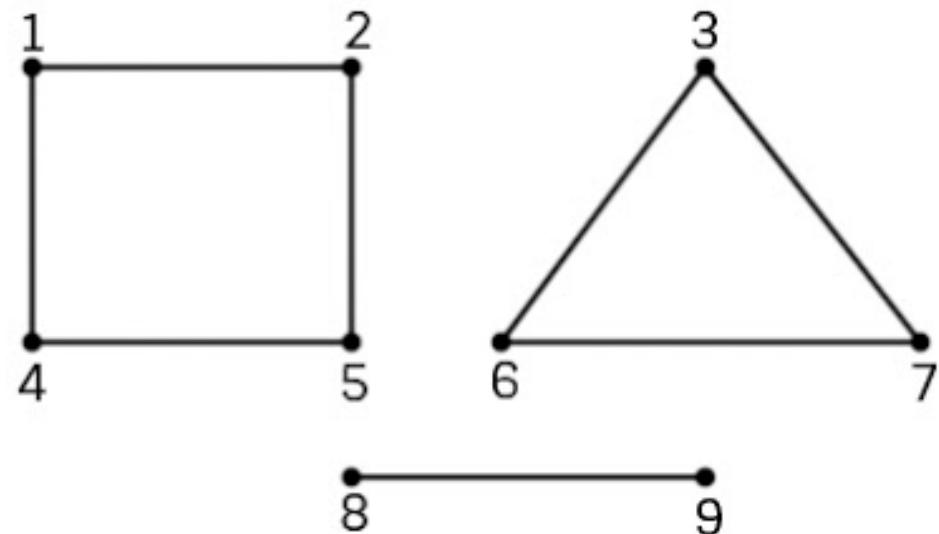
- Tìm các thành phần liên thông
 - Cho đồ thị vô hướng $G = (V, E)$, tìm tất cả những thành phần liên thông của G
 - Áp dụng thuật toán BFS hoặc DFS cho đỉnh nguồn cho trước u , tìm tất cả những đỉnh cùng thành phần liên thông của u

Thành phần liên thông

- Tìm các thành phần liên thông
 - Nhận xét:
 - Nếu đồ thị liên thông thì **số thành phần liên thông của nó là 1** → tương đương với phép duyệt theo thủ tục BFS hoặc DFS được gọi đến đúng một lần (**BFS & DFS không sử dụng đệ quy**)
 - Nếu đồ thị không liên thông thì số thành phần liên thông **lớn hơn 1**
 - Có thể tách chúng thành những **đồ thị con** liên thông
 - Trong phép duyệt đồ thị, số thành phần liên thông của nó bằng số lần gọi tới thủ tục BFS hoặc DFS

Thành phần liên thông

- Tìm các thành phần liên thông
 - Nhận xét
 - Nếu đỉnh i được duyệt thuộc thành phần liên thông thứ $j = \text{so_tplt}$, ta ghi nhận ***chuaxet***[i] = so_tplt
 - Các đỉnh cùng thành phần liên thông nếu chúng có cùng giá trị trong mảng ***chuaxet***[]



Thành phần liên thông

- Tìm các thành phần liên thông
 - Thuật toán

```
void BFS (int u) { //Duyệt các đỉnh cùng thành phần liên thông với u
    queue = Ø; //Khởi tạo queue rỗng
    queue.enqueue(u); //nạp u vào hàng đợi
    so_tplt++;
    chuaxet[u] = so_tplt; //so_tplt là biến toàn cục
    while (queue ≠ Ø) {
        p = queue.dequeue(); // lấy p ra khỏi queue
        for (v ∈ ke(p)) { // v thuộc tập đỉnh kề của p
            if (chuaxet[v] ) {
                queue.enqueue(v); //nạp v vào hàng đợi
                chuaxet[v] = so_tplt; //v có cùng thành phần liên thông với p
            }
        }
    }
}
```

Thành phần liên thông

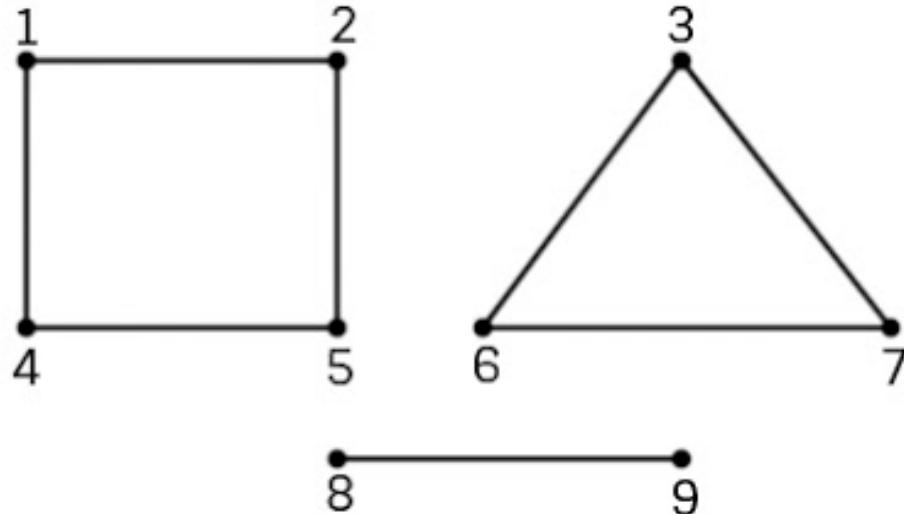
- Tìm các thành phần liên thông
 - Thuật toán

```
void Tim_tp_lienthong() {  
    for (i = 0; i < n; i++)  
        chuaxet[i] = 0;  
    for (i = 0; i < n; i++) //Duyệt các đỉnh cùng TPLT với i  
        if (chuaxet[i] == 0) {  
            so_tplt++;  
            BFS(i);  
        }  
}
```

Thành phần liên thông

- Tìm các thành phần liên thông

- Ví dụ



Thành phần
liên thông

Kết quả duyệt
BFS

- | | |
|---|-----------------|
| 0 | Chưa thực hiện |
| 1 | BFS(1): 1,2,4,5 |
| 2 | BFS(3): 3,6,7 |
| 3 | BFS(8): 8,9 |

Giá trị trong mảng chuaxet[]

- | |
|---------------------------------|
| chuaxet[] = {0,0,0,0,0,0,0,0,0} |
| chuaxet[] = {1,1,0,1,1,0,0,0,0} |
| chuaxet[] = {1,1,2,1,1,2,2,0,0} |
| chuaxet[] = {1,1,2,1,1,2,2,3,3} |

Đường đi ngắn nhất

- Bài toán
 - Bài toán tìm đường đi ngắn nhất **nguồn đơn** là bài toán tìm một đường đi giữa hai đỉnh sao cho tổng các trọng số của các cạnh tạo nên đường đi đó là nhỏ nhất
 - Bài toán tìm đường đi ngắn nhất giữa **mọi cặp đỉnh** là một bài toán tương tự trong đó ta phải tìm các đường đi ngắn nhất cho mọi cặp đỉnh u và v

Đường đi ngắn nhất

- Thuật toán Dijkstra

- Có nhiều biến thể và phổ biến nhất là dùng để giải bài toán tìm đường đi ngắn nhất từ một đỉnh nguồn tới tất cả các đỉnh khác trong đồ thị (**trọng số dương**)

- **Thuật toán**

B1. Khởi tạo tập sptSet rỗng (các đỉnh \in đường đi ngắn nhất)

B2. Gán tất cả các đỉnh của đồ thị với giá trị **khoảng cách là vô cùng**. Giá trị của đỉnh nguồn là 0.

B3. while (sptSet không chứa tất cả các đỉnh)

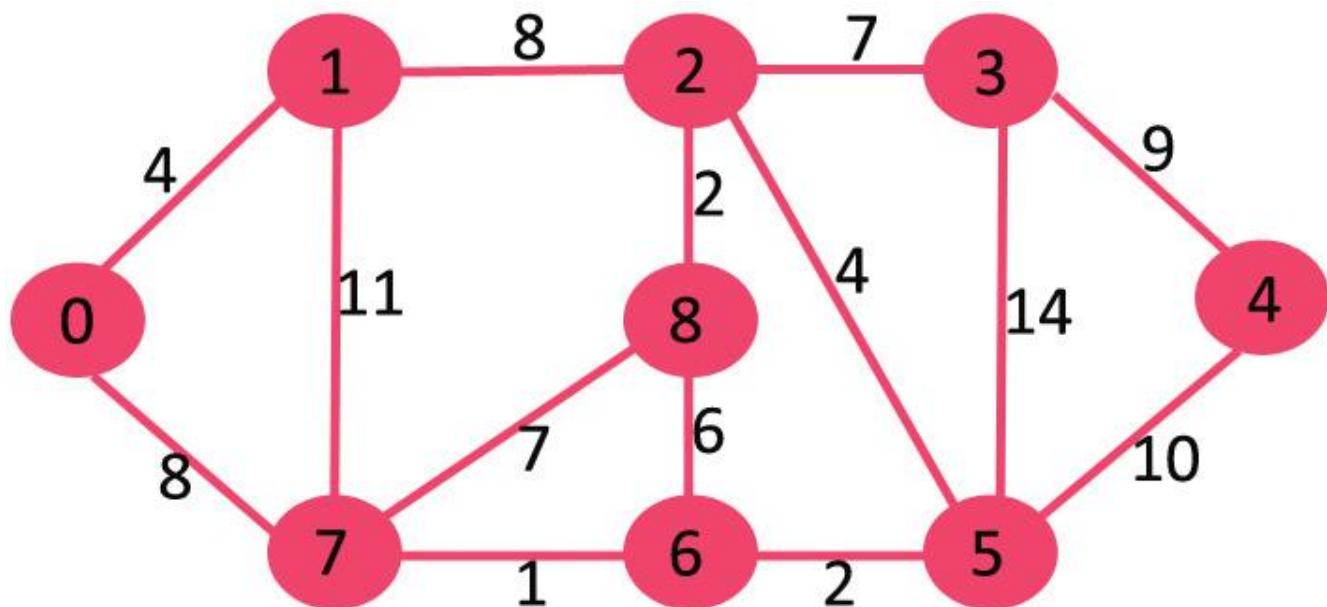
- a. Chọn một đỉnh **$u \notin sptSet$** và có khoảng cách nhỏ nhất

- b. **Thêm u vào tập sptSet**

- c. Cập nhật giá trị khoảng cách của tất cả các đỉnh v kề u không thuộc sptSet \rightarrow tổng khoảng cách **từ đỉnh nguồn đến đỉnh v đi qua u** nhỏ hơn giá trị hiện tại của v

Đường đi ngắn nhất

- Thuật toán Dijkstra
 - Ví dụ

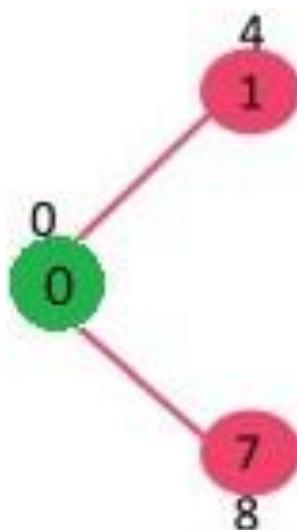
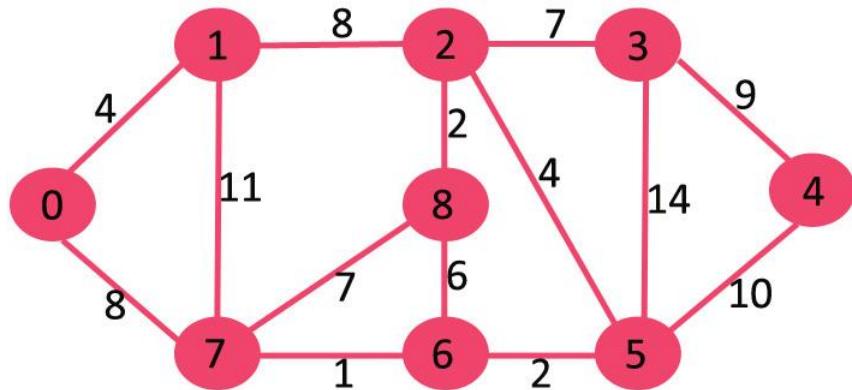


- $sptSet = \{0, \text{INF}, \text{INF}, \text{INF}, \text{INF}, \text{INF}, \text{INF}, \text{INF}\}$ với INF là giá trị dương vô cùng

Đường đi ngắn nhất

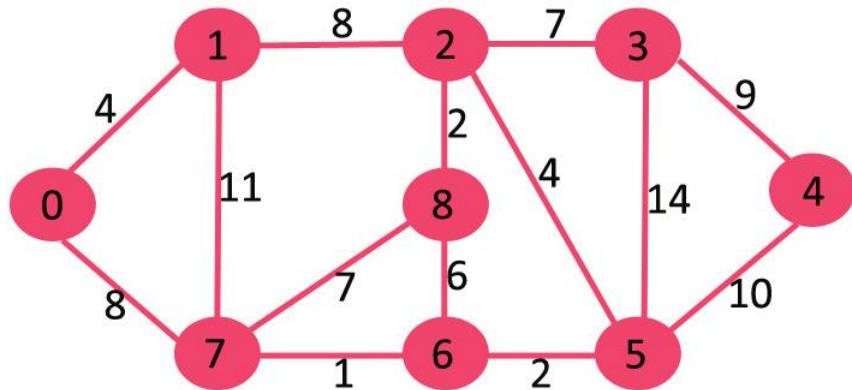
- Thuật toán Dijkstra
 - Ví dụ

- Chọn đỉnh có giá trị khoảng cách nhỏ nhất \rightarrow đỉnh 0 và sptSet = {0}.
- Cập nhật các đỉnh kề của đỉnh 0 là 1 và 7
- Khoảng cách của đỉnh 1 và 7 được cập nhật là 4 và 8

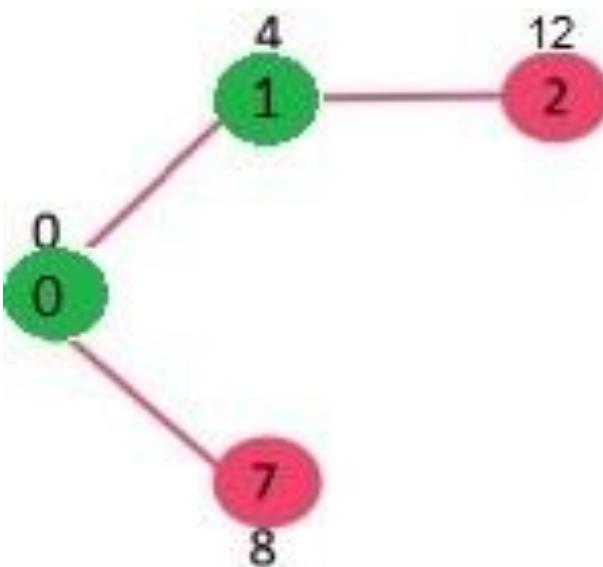


Đường đi ngắn nhất

- Thuật toán Dijkstra
 - Ví dụ

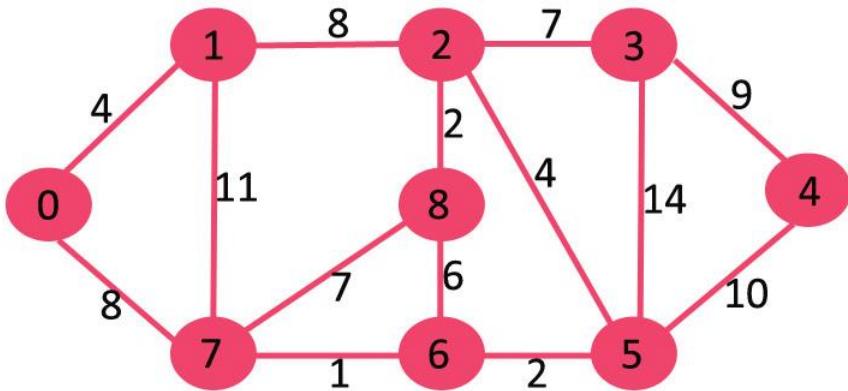


- Chọn đỉnh có khoảng cách nhỏ nhất không thuộc sptSet
→ đỉnh 1 được thêm vào sptSet → sptSet = {0, 1}
- Cập nhật khoảng cách của các đỉnh kề đỉnh 1 → giá trị của đỉnh 2 thành 12

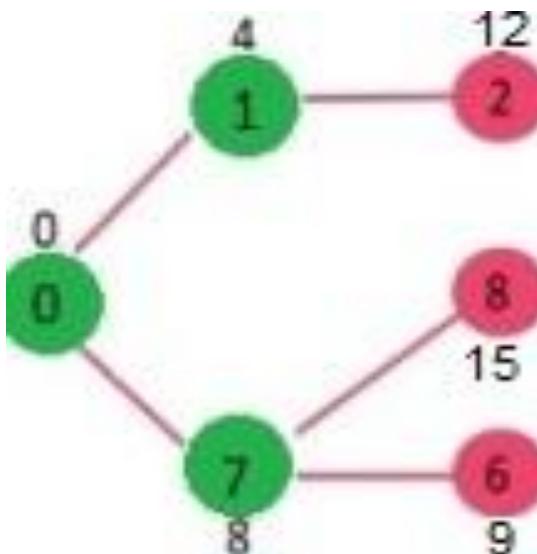


Đường đi ngắn nhất

- Thuật toán Dijkstra
 - Ví dụ

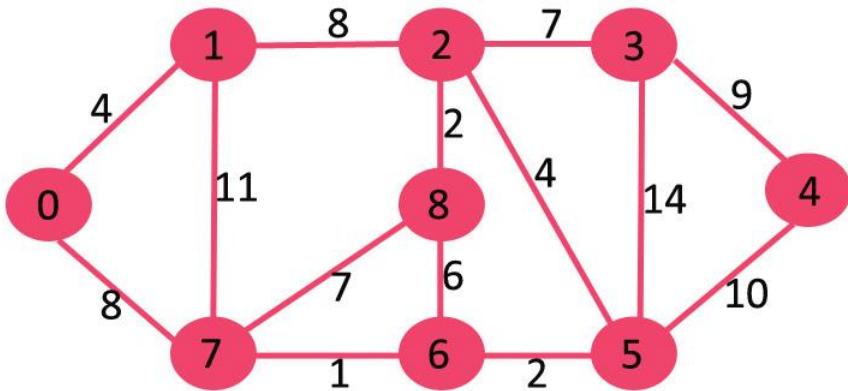


- Chọn đỉnh có khoảng cách nhỏ nhất không thuộc sptSet
→ đỉnh 7 được thêm vào sptSet → $sptSet = \{0, 1, 7\}$
- Cập nhật khoảng cách của các đỉnh kề đỉnh 7 → giá trị của đỉnh 6 và 8 thành 15 và 9

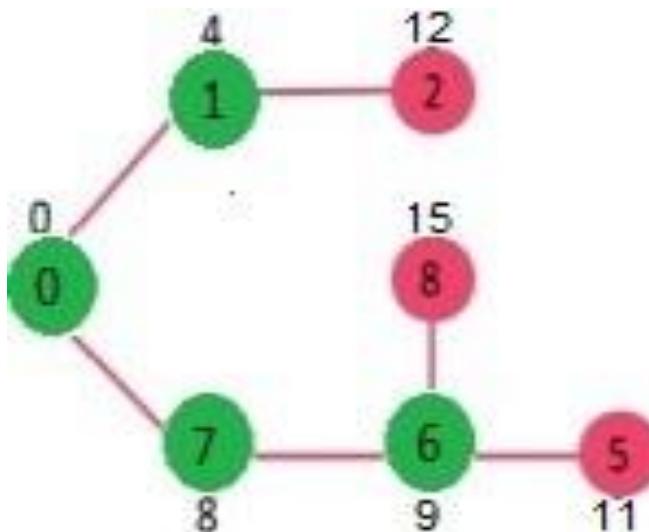


Đường đi ngắn nhất

- Thuật toán Dijkstra
 - Ví dụ



- Chọn đỉnh có khoảng cách nhỏ nhất không thuộc sptSet
→ đỉnh 6 được thêm vào sptSet → sptSet = {0, 1, 7, 6}
- Cập nhật khoảng cách của các đỉnh kề đỉnh 6 → giá trị của đỉnh 5 thành 11

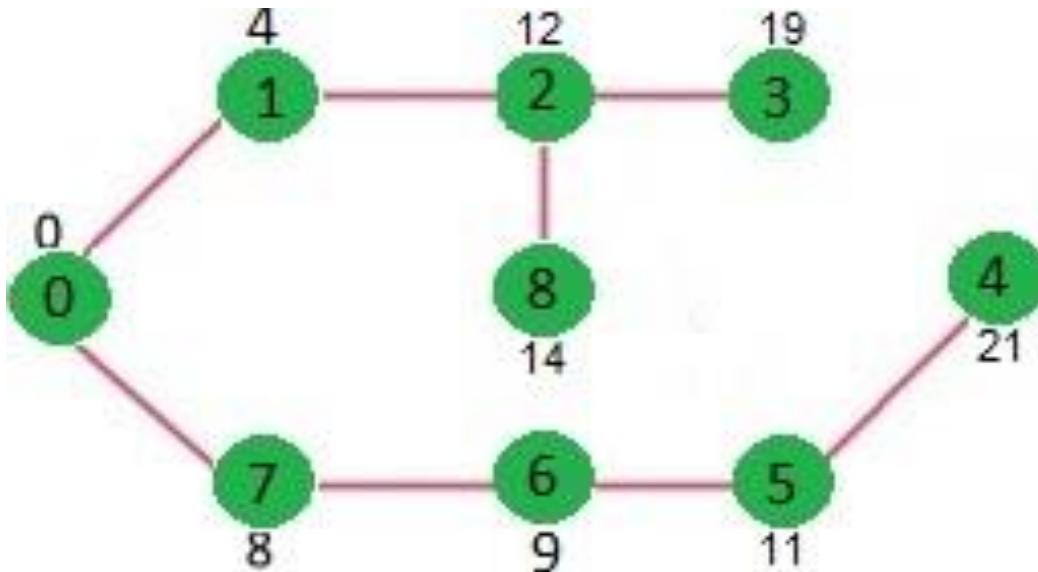


Đường đi ngắn nhất

- Thuật toán Dijkstra

- Ví dụ

- Lặp cho đến khi sptSet chứa tất cả các đỉnh của đồ thị
 - Kết quả như sau



- Độ phức tạp là $O(|V|^2)$

Đường đi ngắn nhất

- Thuật toán Bellman Ford
 - Tìm các đường đi ngắn nhất nguồn đơn trong một đồ thị có hướng có trọng số
 - Một số cung có thể có trọng số âm
 - Đơn giản hơn Dijkstra, phù hợp với các hệ phân tán
 - Độ phức tạp về thời gian là $O(VE)$

Đường đi ngắn nhất

- Thuật toán Bellman Ford
 - **Thuật toán**

Bước 1. Gán tất cả các đỉnh của đồ thị với giá trị **khoảng cách là vô cùng**. Giá trị của đỉnh nguồn là 0. Tạo mảng $\text{dist}[]$ với kích thước $|V|$ có giá trị là vô cùng (INF), đỉnh nguồn có giá trị 0

Bước 2. Thực hiện $|V| - 1$ lần các công việc sau

Với mỗi cạnh uv , thực hiện

If $\text{dist}[u] \neq \text{INF}$ && $\text{dist}[v] > \text{dist}[u] + \text{trọng số của } uv$ **then**
 $\text{dist}[v] = \text{dist}[u] + \text{trọng số của } uv$

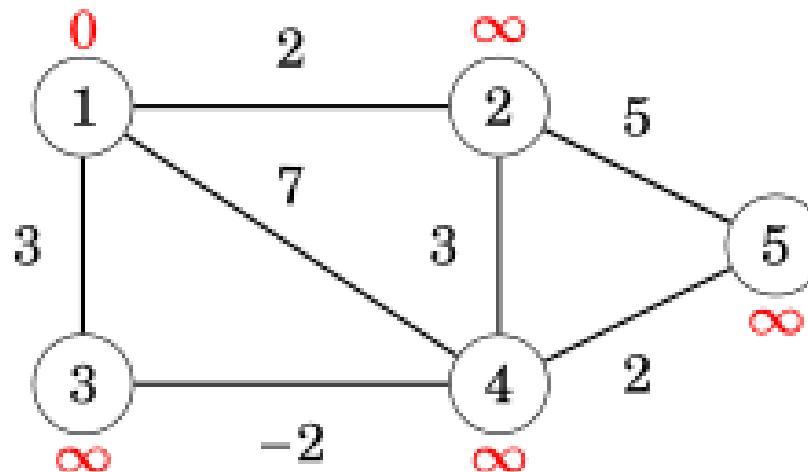
Bước 3. Báo cáo có chu trình âm. Với mỗi cạnh uv , thực hiện

If $\text{dist}[v] > \text{dist}[u] + \text{trọng số của } uv$ **then** “Đồ thị bao gồm chu trình âm”

Bước 2 đảm bảo khoảng cách ngắn nhất nếu đồ thị không có chu trình số âm. Nếu thực hiện lặp tất cả các cạnh thêm một lần nữa và thu được đường đi ngắn hơn đối với bất kỳ đỉnh nào thì có một chu trình âm.

Đường đi ngắn nhất

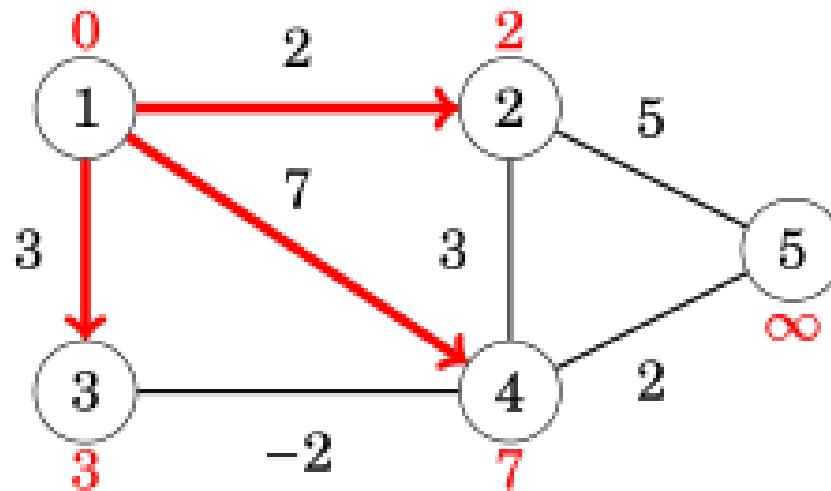
- Thuật toán Bellman Ford
 - Ví dụ



Xét đồ thị có hướng trên với giả định chỉ đi từ đỉnh có số thứ tự **thấp hơn** tới đỉnh có số thứ tự **cao hơn**. Số có màu đỏ cạnh mỗi đỉnh là độ dài đường đi ngắn nhất từ gốc tới đỉnh đó và đỉnh gốc là đỉnh 1

Đường đi ngắn nhất

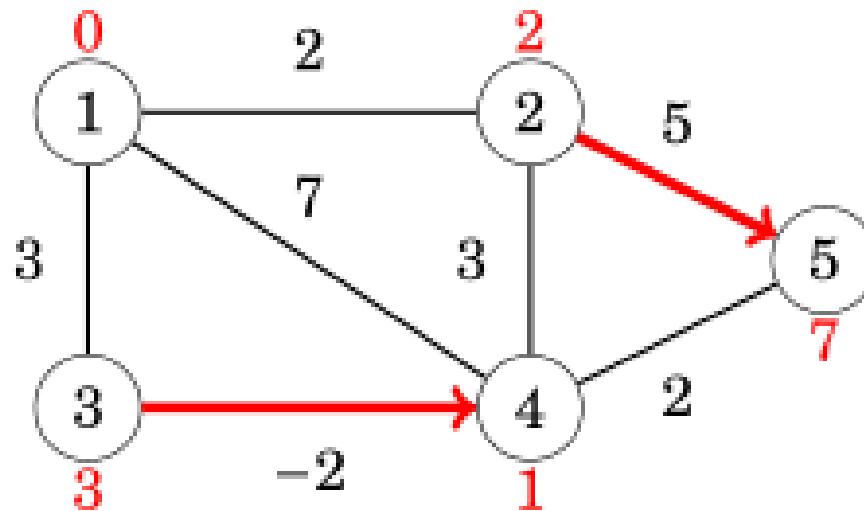
- Thuật toán Bellman Ford
 - Ví dụ



Thực hiện lần duyệt đầu tiên, cập nhật được đường đi ngắn nhất thông qua các cạnh (1, 2); (1, 3); (1, 4)

Đường đi ngắn nhất

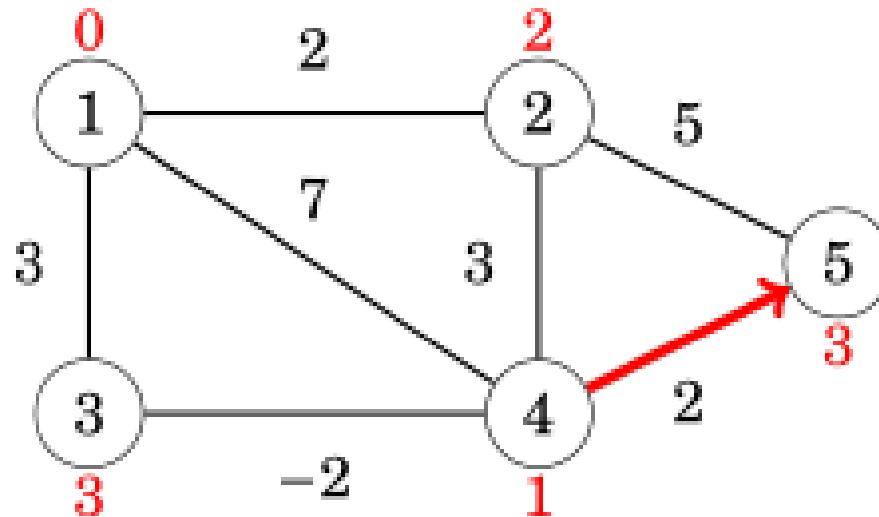
- Thuật toán Bellman Ford
 - Ví dụ



Thực hiện lần duyệt thứ 2, cạnh (2, 5) và (3, 4) là các cạnh tối ưu

Đường đi ngắn nhất

- Thuật toán Bellman Ford
 - Ví dụ



Thực hiện lần duyệt thứ 3, chỉ có cạnh (4, 5) cải tiến đường đi tối ưu

Tới lần duyệt thứ 4, không còn cạnh nào có thể tối ưu hóa bất kỳ đường đi ngắn nhất nào nữa và có thể dừng duyệt

Đường đi ngắn nhất

- Thuật toán Floyd-Warshall
 - Giải bài toán đường đi ngắn nhất cho mọi cặp đỉnh trong đồ thị có cạnh mang trọng số (dương hoặc âm) dựa trên khái niệm các **Đỉnh trung gian**
 - Lưu ý là trong đồ thị không được có chu trình nào có tổng trọng số các cạnh là âm
 - Độ phức tạp về thời gian: $O(V^3)$

Đường đi ngắn nhất

- Thuật toán Floyd-Warshall
 - Các bước của thuật toán với đồ thị N đỉnh

Bước 1. Khởi tạo mảng hai chiều $dist[][]$ chứa đường đi ngắn nhất giữa hai đỉnh bất kỳ, có giá trị là **vô cùng** nếu không có đường nối trực tiếp, ngược lại \rightarrow trọng số của đường nối đó

Bước 2. Tìm tất cả các đường đi ngắn nhất trực tiếp giữa hai cặp đỉnh, sau đó sử dụng $k = 1$ đỉnh đầu tiên làm đỉnh trung gian, sử dụng $k = 2$ đỉnh đầu tiên làm đỉnh trung gian, ..., cho đến khi sử dụng tất cả N đỉnh làm đỉnh trung gian

Bước 3. Tối thiểu hóa các đường đi ngắn nhất giữa hai cặp đỉnh bất kỳ trong bước trước. Với hai đỉnh (i, j) bất kỳ, đường đi ngắn nhất là:

$$\min(\text{dist}[i][j], \text{dist}[i][k] + \text{dist}[k][j])$$

trong đó, $\text{dist}[i][k]$ là đường đi ngắn nhất chỉ sử dụng k đỉnh trung gian đầu tiên, $\text{dist}[k][j]$ là đường đi ngắn nhất giữa cặp đỉnh k và j

Đường đi ngắn nhất

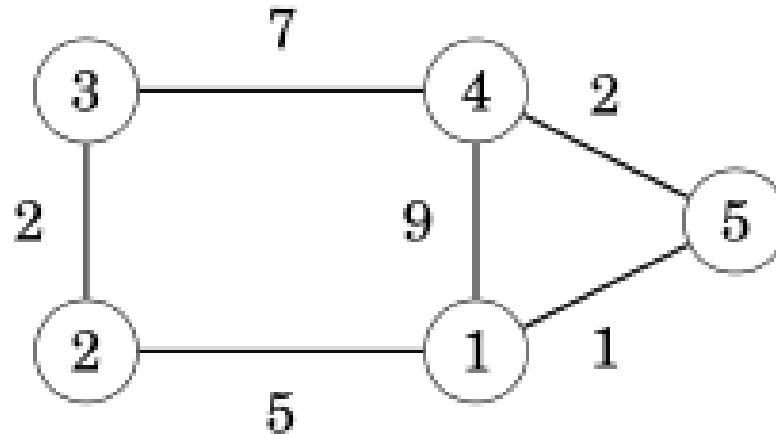
- Thuật toán Floyd-Warshall

- Vì đường đi ngắn nhất sẽ là một kết nối của đường đi ngắn nhất từ i đến k và sau đó là từ k đến j, nên ta có đoạn mã sau

```
for (int k = 1; k <= n; k++) {  
    for (int i = 1; i <= n; i++) {  
        for (int j = 1; j <= n; j++) {  
            dist[i][j] = min( dist[i][j], dist[i][k] + dist[k][j] );  
        }  
    }  
}
```

Đường đi ngắn nhất

- Thuật toán Floyd-Warshall
 - Ví dụ



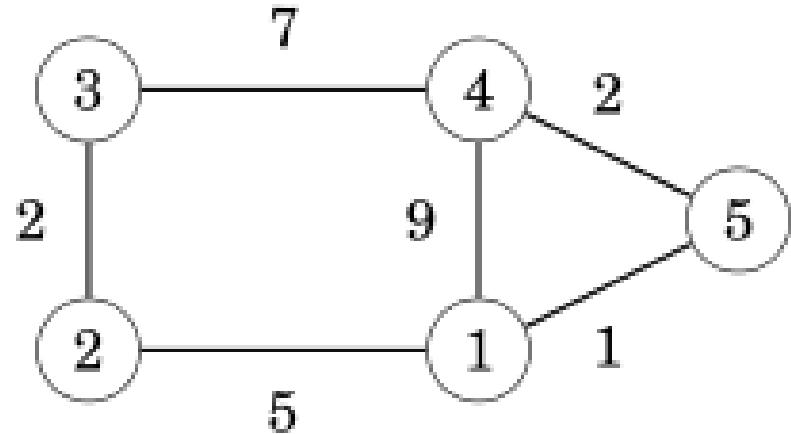
- Xét đồ thị trên, mảng $dist[][]$ có các giá trị

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

Đường đi ngắn nhất

- Thuật toán Floyd-Warshall
 - Ví dụ

Đầu tiên, $k = 1$, tức lấy đỉnh 1 làm trung gian \rightarrow xuất hiện đường đi từ đỉnh 2 tới đỉnh 4 (độ dài 14), và từ đỉnh 2 tới đỉnh 5 (độ dài 6)



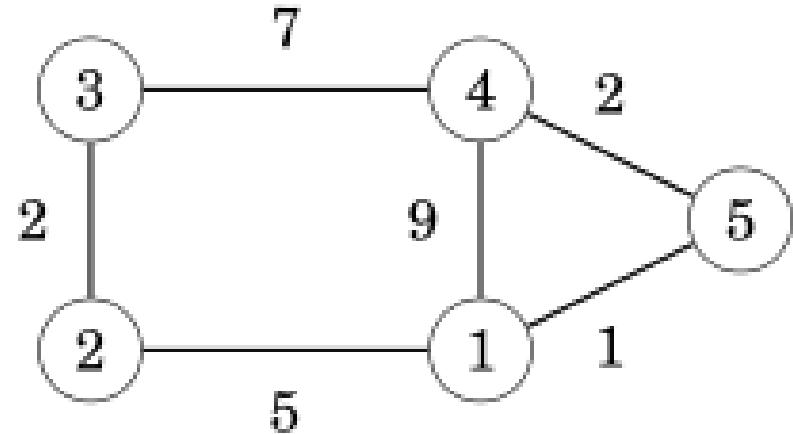
Đường đi trung gian qua đỉnh 1 để đi từ đỉnh 4 tới đỉnh 5 không tối ưu về chiều dài ($9 + 1 > 2$) nên không cập nhật lại đường đi ngắn nhất giữa 2 đỉnh 4 và 5

	1	2	3	4	5
1	0	5	∞	9	1
2	5	0	2	14	6
3	∞	2	0	7	∞
4	9	14	7	0	2
5	1	6	∞	2	0

Đường đi ngắn nhất

- Thuật toán Floyd-Warshall
 - Ví dụ

Tiếp theo, $k = 2$, tức lấy đỉnh 2 hoặc cả đỉnh 1 và đỉnh 2 làm trung gian \rightarrow Đường đi từ đỉnh 3 tới đỉnh 1 (độ dài 7), từ đỉnh 3 tới đỉnh 5 (độ dài 8) được hình thành



Đường đi từ đỉnh 3 tới đỉnh 4 không cập nhật độ dài do:
 $(7 < 2 + 5 + 9)$

	1	2	3	4	5
1	0	5	7	9	1
2	5	0	2	14	6
3	7	2	0	7	8
4	9	14	7	0	2
5	1	6	8	2	0

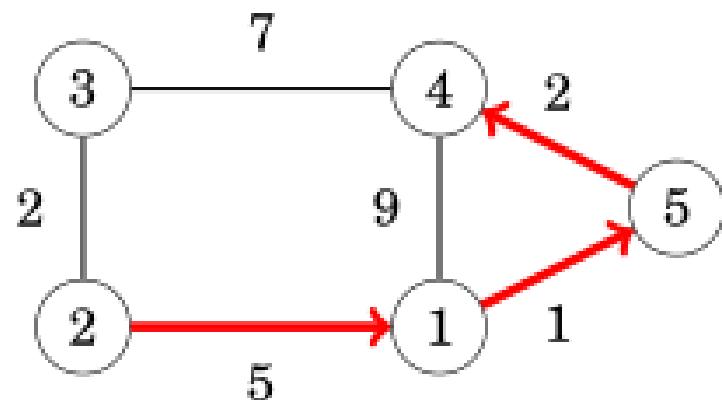
Đường đi ngắn nhất

- Thuật toán Floyd-Warshall
 - Ví dụ

Cứ tiếp tục lựa chọn k như vậy cho tới hết, ta sẽ thu được mảng hai chiều hoàn chỉnh

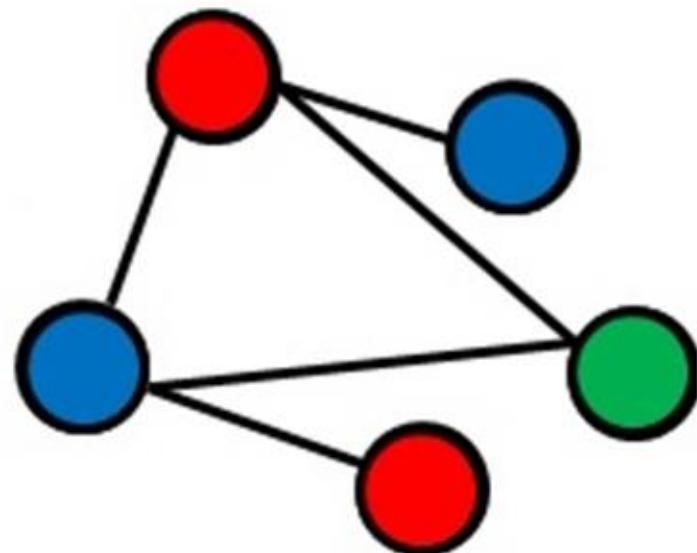
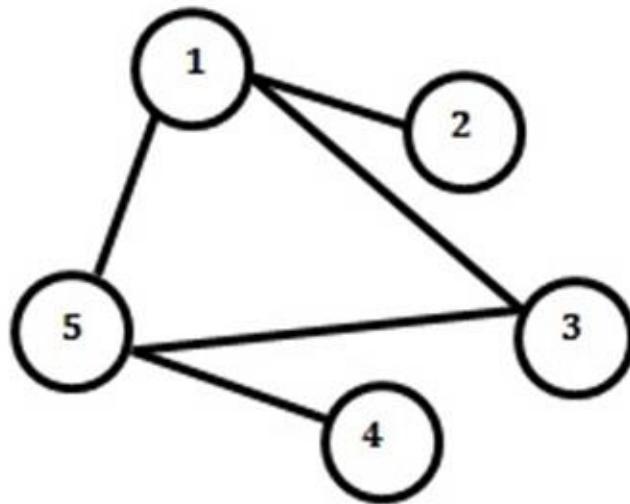
	1	2	3	4	5
1	0	5	7	3	1
2	5	0	2	8	6
3	7	2	0	7	8
4	3	8	7	0	2
5	1	6	8	2	0

Giả sử, qua mảng này, ta thấy đường đi ngắn nhất từ **đỉnh 2** tới **đỉnh 4** có **độ dài 8**. Dựa theo đồ thị thì nó là **đoạn đường màu đỏ** trong hình bên



Tô màu đồ thị

- Bài toán tô màu đồ thị
 - Tô màu (đỉnh) đồ thị là việc thực hiện gán màu cho mỗi đỉnh của đồ thị, sao cho hai đỉnh kề nhau không cùng một màu và số màu được sử dụng là ít nhất
 - **Số màu ít nhất** có thể sử dụng để tô màu đồ thị được gọi là **sắc số** (chromatic number) của đồ thị đó

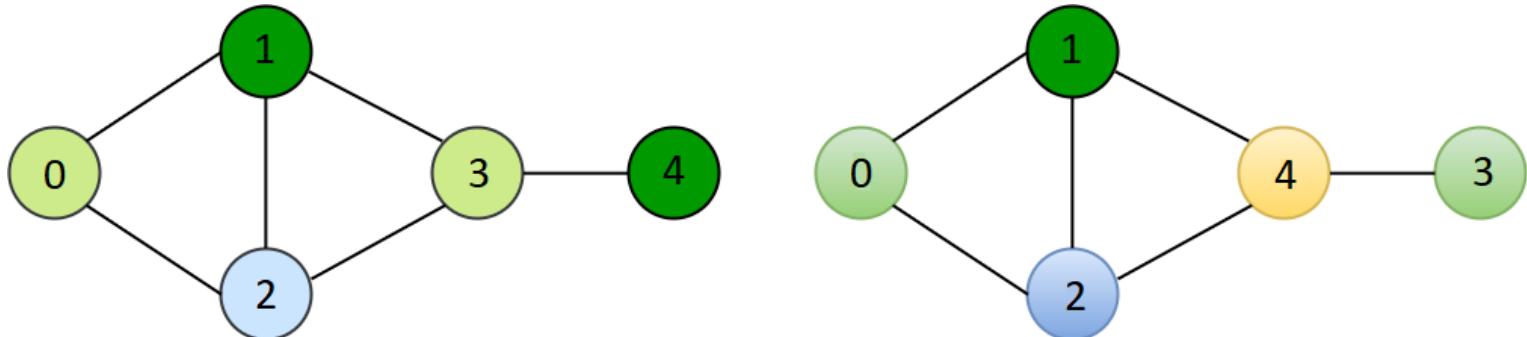


Tô màu đồ thị

- Thuật toán tham lam cơ bản tô màu đồ thị
 - Bài toán tô màu đồ thị với số màu tối thiểu thuộc lớp bài toán **NP-đầy đủ**
 - Thuật toán tham lam cơ bản không sử dụng nhiều hơn $d + 1$ màu với d là bậc lớn nhất của đồ thị
 - **Thuật toán**
 - B1. Tô màu đỉnh đầu tiên với màu thứ nhất.
 - B2. Thực hiện các thao tác sau với $V - 1$ đỉnh còn lại.
Xem xét đỉnh được chọn hiện thời v và tô màu đỉnh v với màu được đánh số thấp nhất mà **chưa được sử dụng** để tô **bất kỳ** đỉnh nào **liền kề** với v trước đó.
Nếu tất cả các màu đã được sử dụng trước đó xuất hiện trên đỉnh liền kề với v , gán một màu mới cho nó.

Tô màu đồ thị

- Thuật toán tham lam cơ bản tô màu đồ thị
 - Phân tích thuật toán**
 - Độ phức tạp về thời gian: $O(V^2 + E)$
 - Không phải lúc nào cũng sử dụng số màu ít nhất
 - Số màu được sử dụng đôi khi phụ thuộc vào **thứ tự** các đỉnh được tô → thứ tự các đỉnh được tô là quan trọng → cần tìm cách sắp xếp các đỉnh được tô để thuật toán làm việc hiệu quả hơn



Tô màu đồ thị

- Thuật toán Welsh Powell

- **Ý tưởng:** bắt đầu với các đỉnh có bậc lớn nhất → xử lý các đỉnh có khả năng **xung đột** sớm nhất có thể

Bước 1. Tính giá trị bậc của các đỉnh trong V. Lập danh sách $V' = [v_1, v_2, \dots, v_n]$ là các đỉnh của đồ thị được sắp xếp theo thứ tự bậc giảm dần: $d(v_1) > d(v_2) > \dots > d(v_n)$. Gán màu $c = 1$

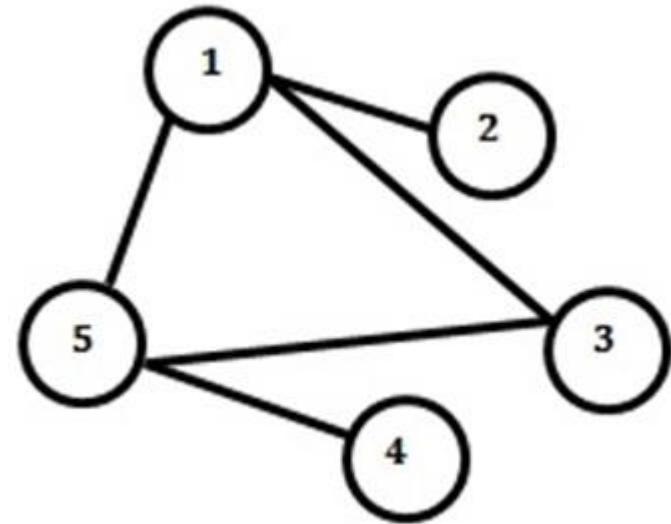
Bước 2. Tô màu c cho đỉnh đầu tiên trong danh sách V' . Duyệt lần lượt các đỉnh khác trong V' và chỉ tô màu c cho các đỉnh không kề đỉnh đã có màu c .

Bước 3. Kiểm tra nếu tất cả các đỉnh trong V đã được tô màu thì thuật toán kết thúc, đồ thị đã sử dụng c màu để tô. Ngược lại, nếu vẫn còn đỉnh chưa được tô thì chuyển sang bước 4.

Bước 4. Loại khỏi danh sách V' các đỉnh đã được tô màu. Các đỉnh trong V' vẫn theo thứ tự bậc giảm dần. Gán $c = c + 1$ và quay lại bước 2

Tô màu đồ thị

- Thuật toán Welsh Powell
 - Ví dụ



Bước 1. Sau sắp xếp V' có thứ tự là [1, 5, 3, 2, 4]. Gán $c = 1$

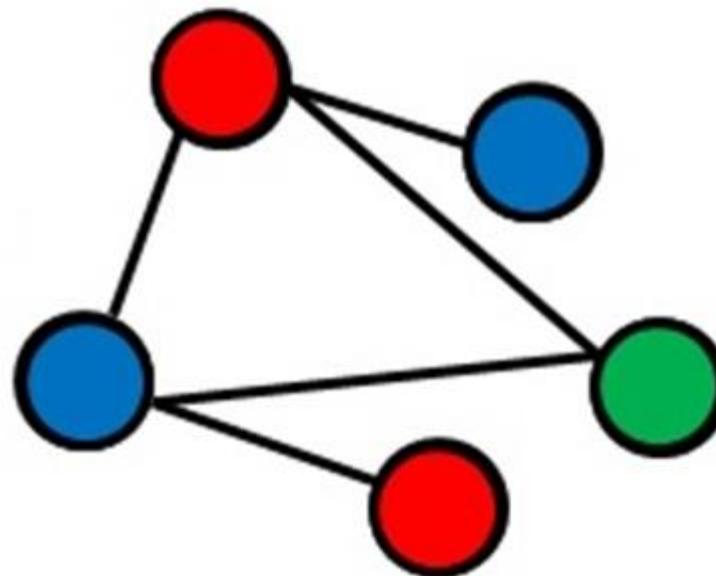
Bước 2. Tô màu 1 (đỏ) cho đỉnh 1. Tô các đỉnh còn lại trong V' .
Đỉnh 2, 3 và 5 kề đỉnh 1 nên chưa tô màu cho các đỉnh này. Đỉnh 4 không kề với đỉnh 1 \rightarrow thực hiện tô màu 1 (đỏ) cho đỉnh 4.

Bước 3. Kiểm tra thấy vẫn còn các đỉnh trong V chưa được tô màu nên chuyển sang bước 4

Bước 4. Loại bỏ các đỉnh 1, 4 đã được tô màu ra khỏi V' ta thu được $V' = [5, 3, 2]$. Đổi màu mới $\rightarrow c = 2$ và lặp lại từ bước 2

Tô màu đồ thị

- Thuật toán Welsh Powell
 - Ví dụ
 - Cuối cùng ta thu được đồ thị đã được tô bởi 3 màu



Tô màu đồ thị

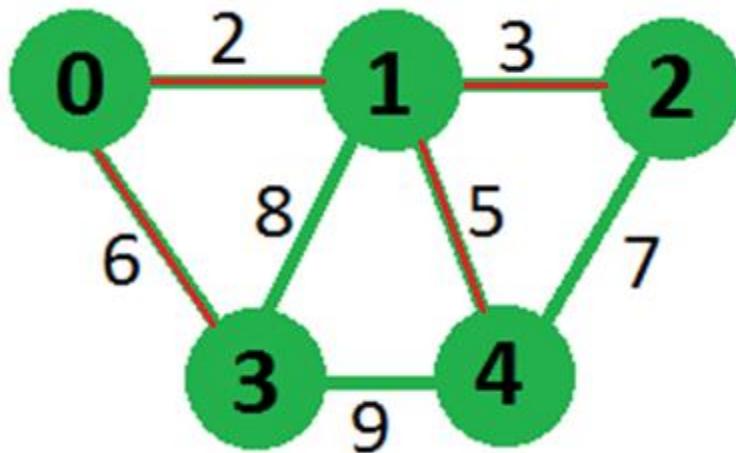
- Ứng dụng
 - Lập lịch và xếp thời khóa biểu
 - Gán tần số cho trạm mobile
 - Kiểm tra một đồ thị là đồ thị phân đôi hay không
 - Tô màu bản đồ

Cây bao trùm

- Cho một đồ thị vô hướng hoặc có hướng, một cây bao trùm (**cây khung**) của đồ thị đã cho là một đồ thị con và là một **cây** kết nối tất cả các đỉnh của đồ thị
- Một đơn đồ thị có thể có nhiều cây bao trùm
- Cây bao trùm tối thiểu (MST) của một đồ thị vô hướng có trọng số và liên thông là cây bao trùm có tổng trọng số nhỏ hơn hoặc bằng tổng trọng số của bất kỳ cây bao trùm nào khác
- Một cây bao trùm tối thiểu có $(V - 1)$ cạnh, trong đó V là số đỉnh của đồ thị

Cây bao trùm

- Cây bao trùm tối thiểu (MST)



- Cây có tổng trọng số nhỏ nhất là 16 với các cạnh là:
0-1, 1-2, 0-3 và 1-4

Cây bao trùm

- Thuật toán Kruskal

Bước 1. Sắp xếp tất cả các cạnh của đồ thị theo thứ tự tăng dần của các trọng số.

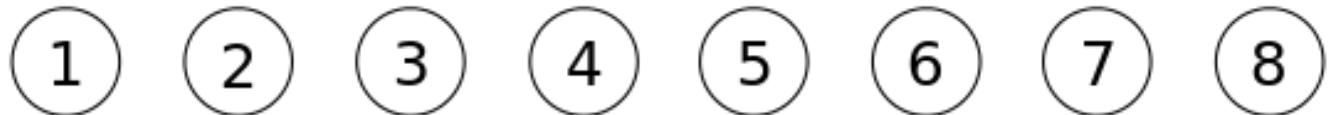
Bước 2. Chọn cạnh có trọng số nhỏ nhất. Kiểm tra xem nó có **tạo thành một chu trình** đối với cây bao trùm hiện đang xem xét hay không. Nếu không thì đưa cạnh đó vào cây bao trùm, ngược lại thì loại bỏ cạnh này.

Bước 3. Lặp lại Bước 2 cho tới khi có $V-1$ cạnh trong cây bao trùm.

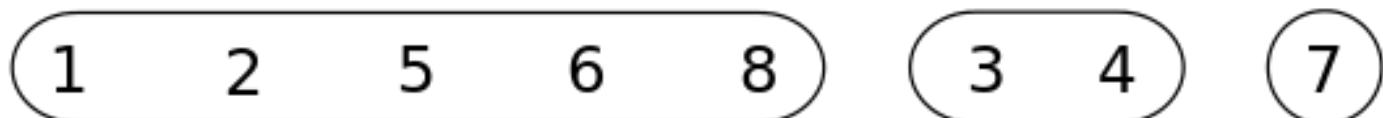
Độ phức tạp về thời gian là $O(E \log E)$ hoặc $O(E \log V)$
Trong bước 2 của thuật toán cần áp dụng thuật toán phát hiện **chu trình trong đồ thị**

Cây bao trùm

- Cấu trúc dữ liệu các tập rời nhau (DS)
 - Là cấu trúc dữ liệu lưu trữ một tập các phần tử được phân chia thành một số tập con rời nhau (không chồng lên nhau) → là một tập hợp của các tập hợp
 - Coi mỗi tập con là **một cây** → mỗi DS là một rừng gồm nhiều cây. Để đơn giản thì ban đầu mỗi tập con chỉ có một phần tử.
 - **Ví dụ:** Mỗi tập con có duy nhất một phần tử



- Mỗi tập con có nhiều phần tử



Cây bao trùm

- Thuật toán Union_find
 - Thực hiện hai thao tác:
 - **Find**: xác định một phần tử cụ thể x nằm trong tập con nào. Cũng dùng để xác định xem hai phần tử có cùng nằm trong cùng một tập con hay không
→ Xác định gốc là cha đại diện cho tập con
 - **Union**: nối hai tập con thành một tập con lớn hơn
- Ứng dụng để kiểm tra xem một đồ thị có tồn tại chu trình hay không
- Với mỗi cạnh, tạo các tập con sử dụng cả hai đỉnh của cạnh. Nếu cả hai đỉnh nằm trong cùng một tập con thì xuất hiện một chu trình

Cây bao trùm

- Thuật toán Union_find
 - Ví dụ về phát hiện chu trình:

0 1 2 → ba tập con ban đầu
-1 -1 -1 → mảng parent[]

Xử lý lần lượt các phần tử như sau:

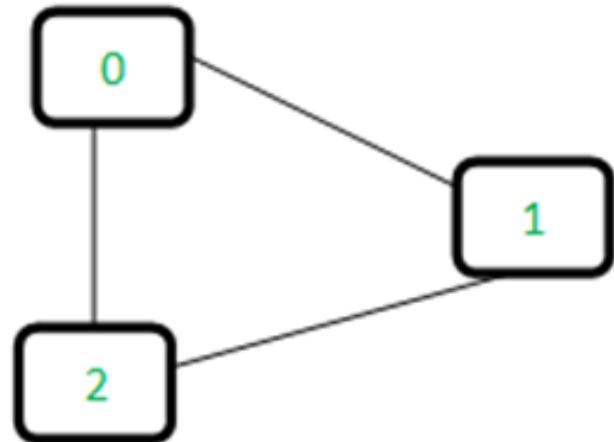
Cạnh 0-1: tìm các tập con chứa các đỉnh 0 và 1. Vì chúng thuộc hai tập con khác nhau nên hợp chúng lại.

0 1 2 <----- 1 là cha của 0 (1 là **đại diện** của tập con {0, 1})
1 -1 -1

Cạnh 1-2: 1 nằm trong tập con 1 và 2 nằm trong tập con 2. Do đó, hợp chúng lại.

0 1 2 <----- 2 là cha của 1 (2 là **đại diện** cho tập con {0, 1, 2})
1 2 -1

Cạnh 0-2: 0 nằm trong tập con 2 và 2 cũng nằm trong tập con 2. Do đó, nếu thêm cạnh này thì tạo thành một chu trình.



Cây bao trùm

- Thuật toán Union_find
 - Hai thao tác **Find** và **Union** được cài đặt như sau:

```
int find(int parent[], int i)
{
    if (parent[i] == -1)
        return i;
    return find(parent, parent[i]);
}
```

```
void Union(int parent[], int x, int y)
{
    int xset = find(parent, x);
    int yset = find(parent, y);
    parent[xset] = yset;
}
```

Cây bao trùm

- Thuật toán Union_find theo hạng

3

/

2

/

1

/

0

Thuật toán Union_find có độ phức tạp về thời gian là $O(n)$

Có thể tối ưu thuật toán Union_find để có $O(\text{Log}n)$. Ý tưởng là luôn gắn cây có độ sâu nhỏ hơn vào gốc của cây có độ sâu lớn hơn. Kỹ thuật này gọi là **hợp theo hạng**.

Ví dụ, tập gồm các tập con một phần tử: 0 1 2 3

Hợp (0, 1)

1 2 3

/

0

Hợp (1, 2)

1 3

/ \

0 2

Hợp (2, 3)

1

/ | \

0 2 3

Cây bao trùm

- Thuật toán Union_find theo hạng

```
struct subset
{
    int parent;
    int rank;
};
```

```
//Giả sử mỗi tập con một phần tử ban đầu có cha là chính nó
int find(subset subsets[], int i)
{
    // Tìm gốc và tạo gốc là cha của i
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}
```

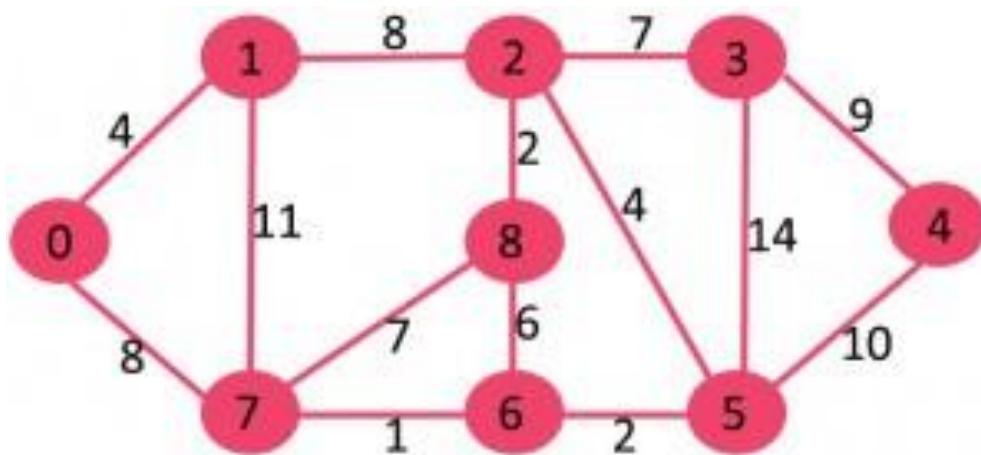
Cây bao trùm

- Thuật toán Union_find theo hạng

```
void Union(subset subsets[], int x, int y) {  
    int xroot = find(subsets, x);  
    int yroot = find(subsets, y);  
  
    // Gắn cây có hạng nhỏ hơn vào cây có hạng cao hơn  
    // (Hợp theo hạng)  
    if (subsets[xroot].rank < subsets[yroot].rank)  
        subsets[xroot].parent = yroot;  
    else if (subsets[xroot].rank > subsets[yroot].rank)  
        subsets[yroot].parent = xroot;  
  
    // Nếu hạng bằng nhau thì x là gốc và tăng hạng một đơn vị  
    else {  
        subsets[yroot].parent = xroot;  
        subsets[xroot].rank++;  
    }  
}
```

Cây bao trùm

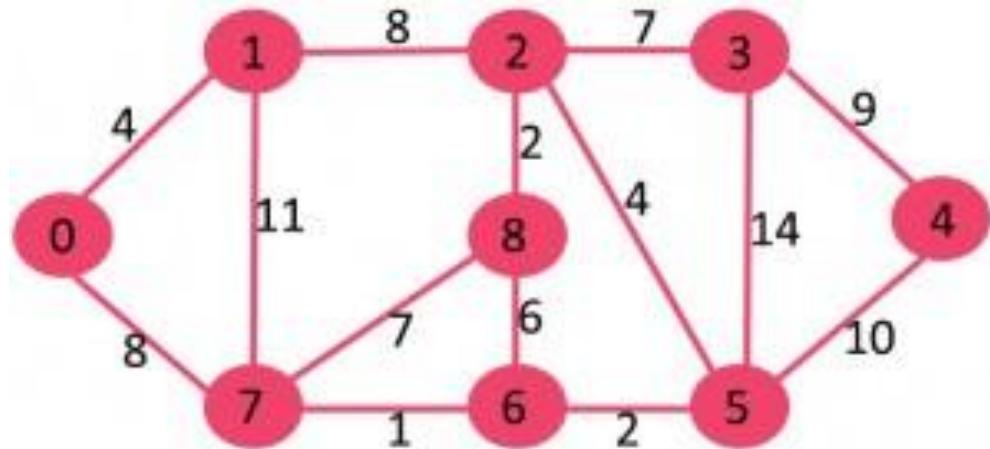
- Thuật toán Kruskal
 - Là thuật toán tham lam → chọn cạnh có trọng số nhỏ nhất không tạo thành chu trình để xây dựng cây
 - Ví dụ, ta có đồ thị như hình dưới



Cây bao trùm

- Thuật toán Kruskal
 - Sau khi sắp xếp lại các trọng số ta có:

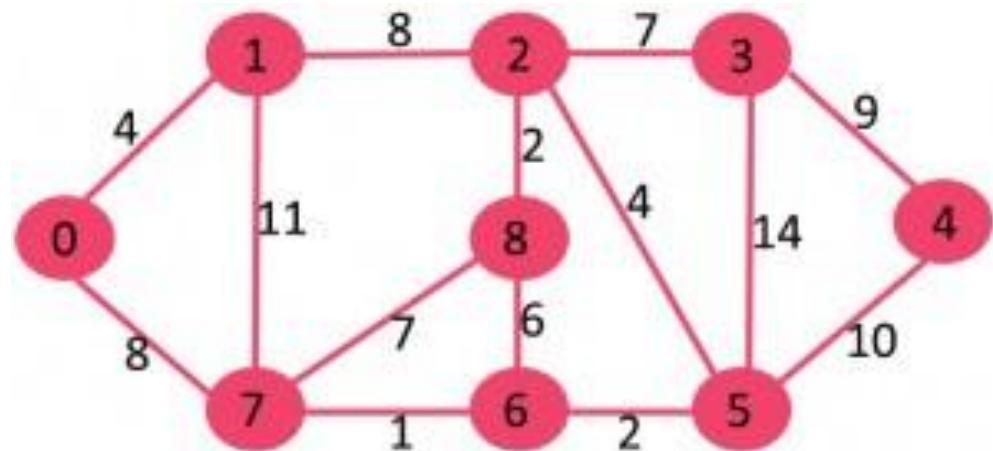
Weight	Src	Dest
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5



Cây bao trùm

- Thuật toán Kruskal

Thực hiện chọn lần lượt các cạnh từ danh sách đã được sắp xếp



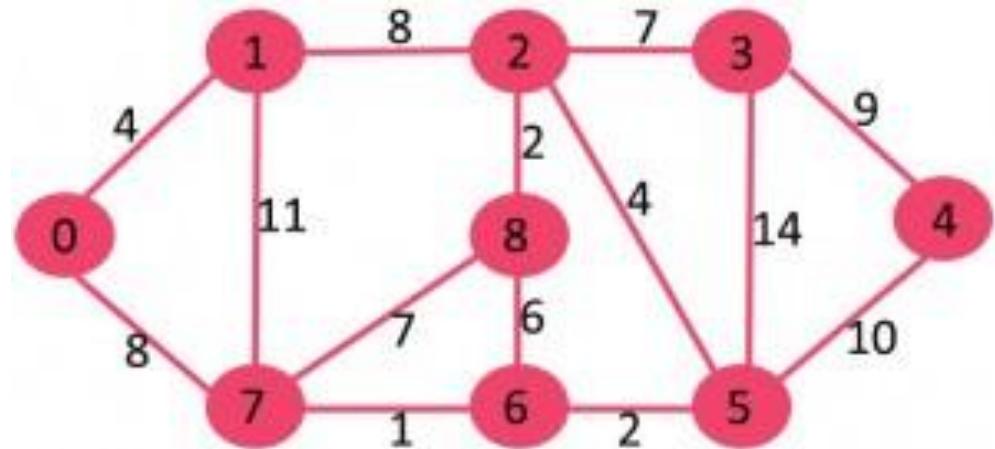
- Chọn cạnh 7-6: Không tạo thành chu trình, đưa vào MST.



Cây bao trùm

- Thuật toán Kruskal

Thực hiện chọn lần lượt các cạnh từ danh sách đã được sắp xếp



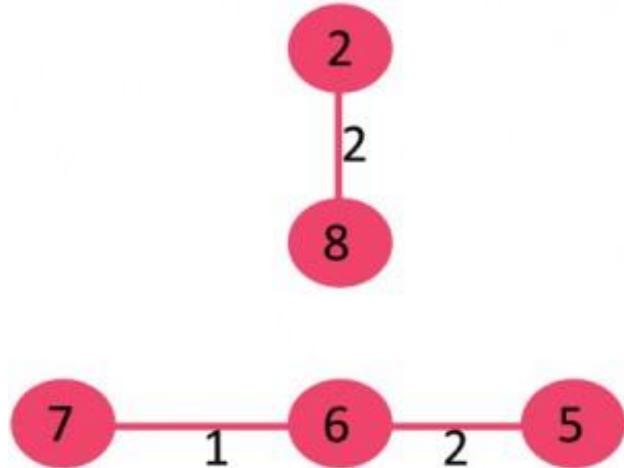
2. Chọn cạnh 8-2: Không tạo thành chu trình, đưa vào MST.



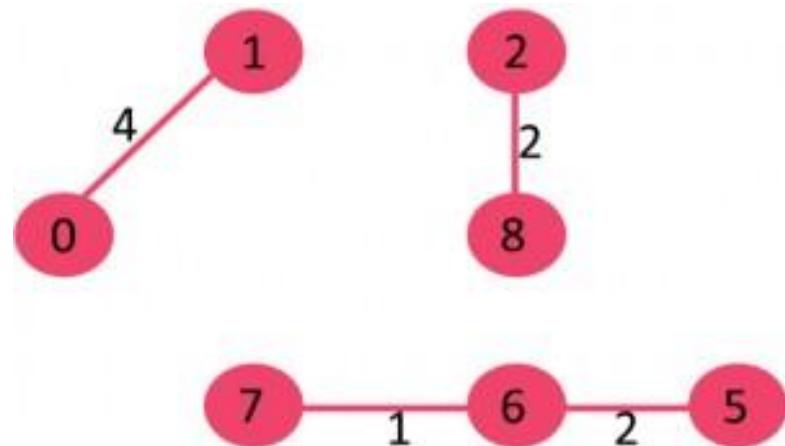
Cây bao trùm

- Thuật toán Kruskal

3. Chọn cạnh 6-5: Không tạo thành chu trình, đưa vào MST.

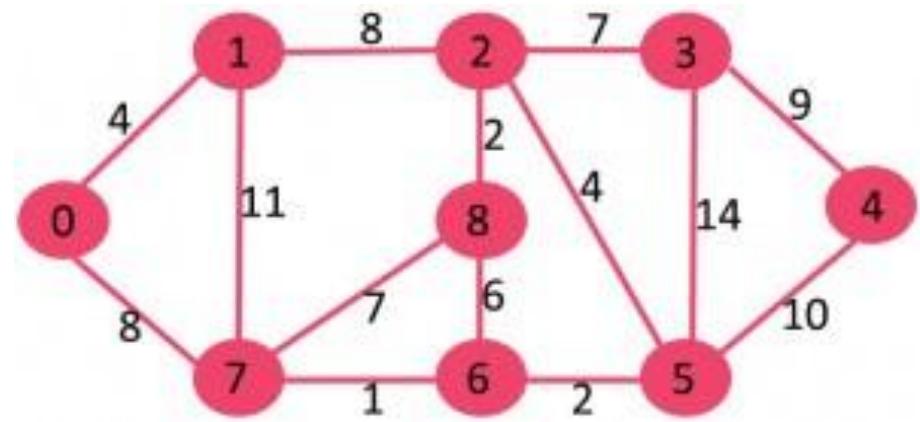


4. Chọn cạnh 0-1: Không tạo thành chu trình, đưa vào MST.

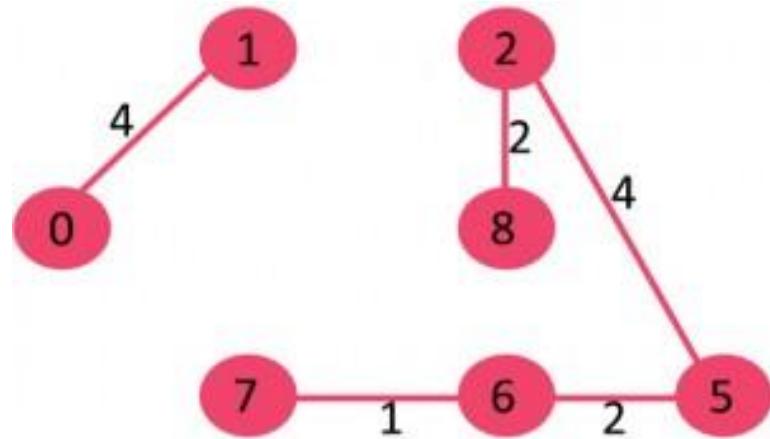


Cây bao trùm

- Thuật toán Kruskal



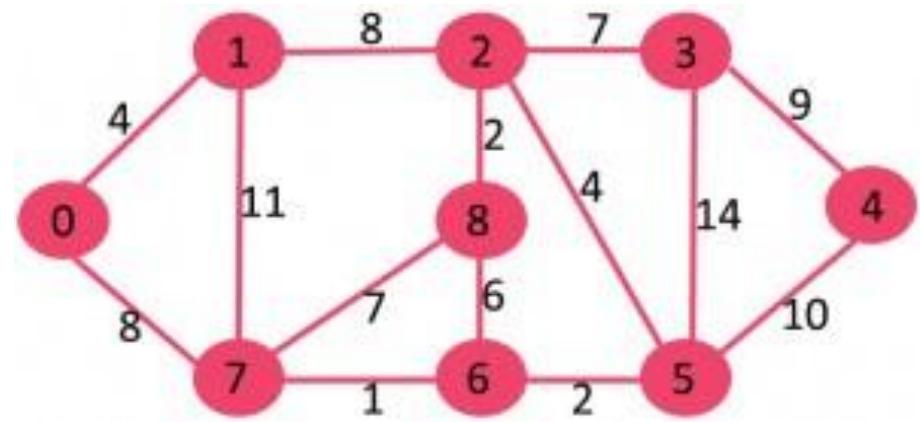
5. Chọn cạnh 2-5: Không tạo thành chu trình, đưa vào MST.



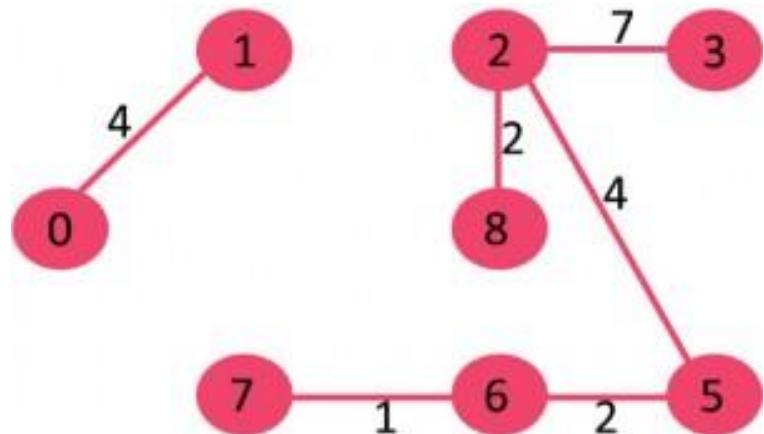
6. Chọn cạnh 8-6: Tạo thành chu trình → loại bỏ.

Cây bao trùm

- Thuật toán Kruskal



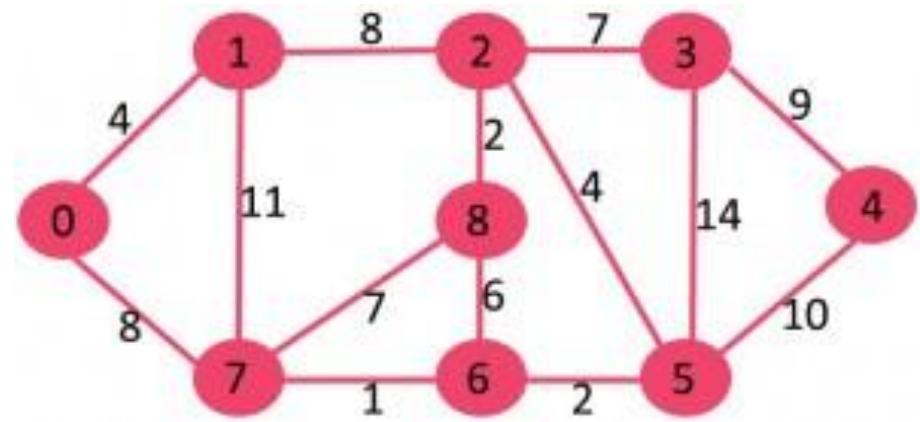
7. Chọn cạnh 2-3: Không tạo thành chu trình, đưa vào MST.



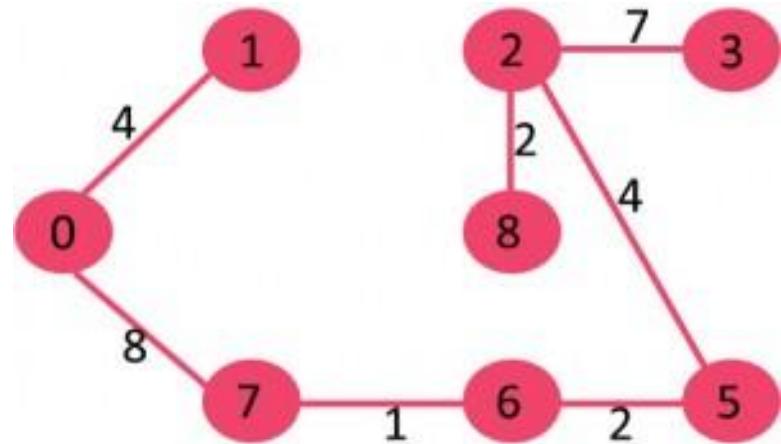
8. Chọn cạnh 7-8: Tạo thành chu trình → loại bỏ

Cây bao trùm

- Thuật toán Kruskal



9. Chọn cạnh 0-7: Không tạo thành chu trình, đưa vào MST.

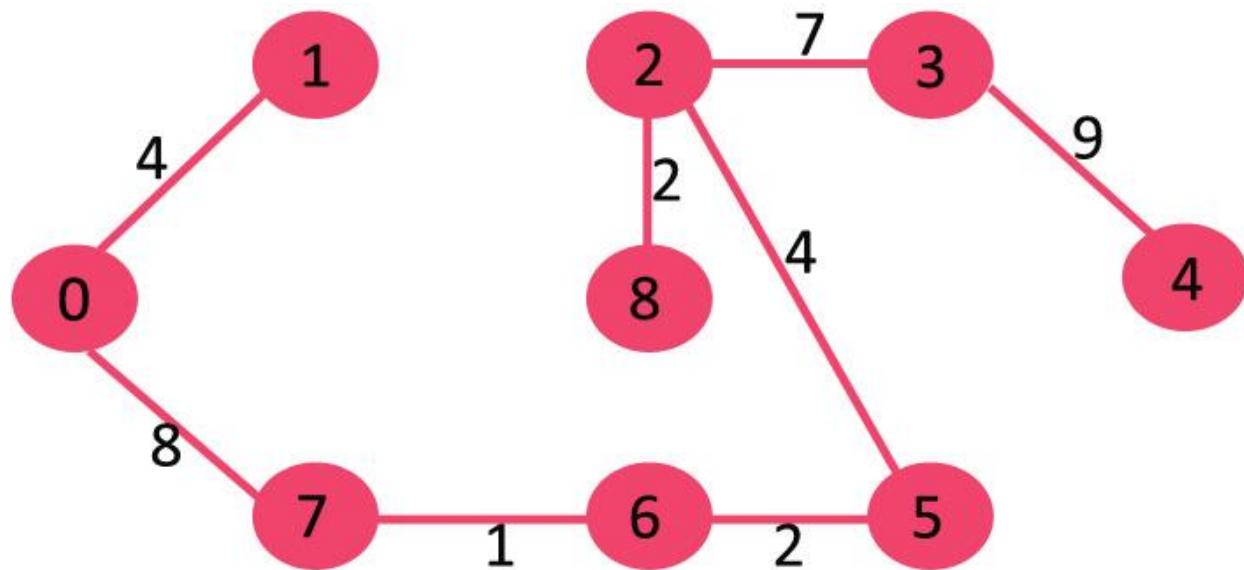


10. Chọn cạnh 1-2: Tạo thành chu trình → loại bỏ

Cây bao trùm

- Thuật toán Kruskal

11. Chọn cạnh 3-4: Không tạo thành chu trình, đưa vào MST.



Đã chọn được $|V|-1 = 7$ cạnh \rightarrow dừng thuật toán

Cây bao trùm

- Thuật toán Prim
 - Là thuật toán theo chiến lược tham lam
 - Bắt đầu với MST rỗng
 - Duy trì hai tập đỉnh, tập thứ nhất là các đỉnh **nằm trong MST** và tập thứ hai **chưa** được đưa vào MST
 - Mỗi bước xem xét **các cạnh nối hai tập đỉnh** và chọn đỉnh có trọng số **nhỏ nhất**
 - Đưa cạnh được chọn vào MST
 - Độ phức tạp về thời gian là $O(|V|^2)$

Cây bao trùm

- Thuật toán Prim

Bước 1. Khởi tạo cấu trúc mstSet lưu các đỉnh đã nằm trong MST

Bước 2. Gán khóa cho các đỉnh giá trị **dương vô cùng**, đỉnh được chọn đầu tiên có khóa là 0

Bước 3. Lặp cho đến khi mstSet chứa tất cả các đỉnh

Bước 3.1. Chọn đỉnh u chưa nằm trong mstSet và có giá trị khóa nhỏ nhất

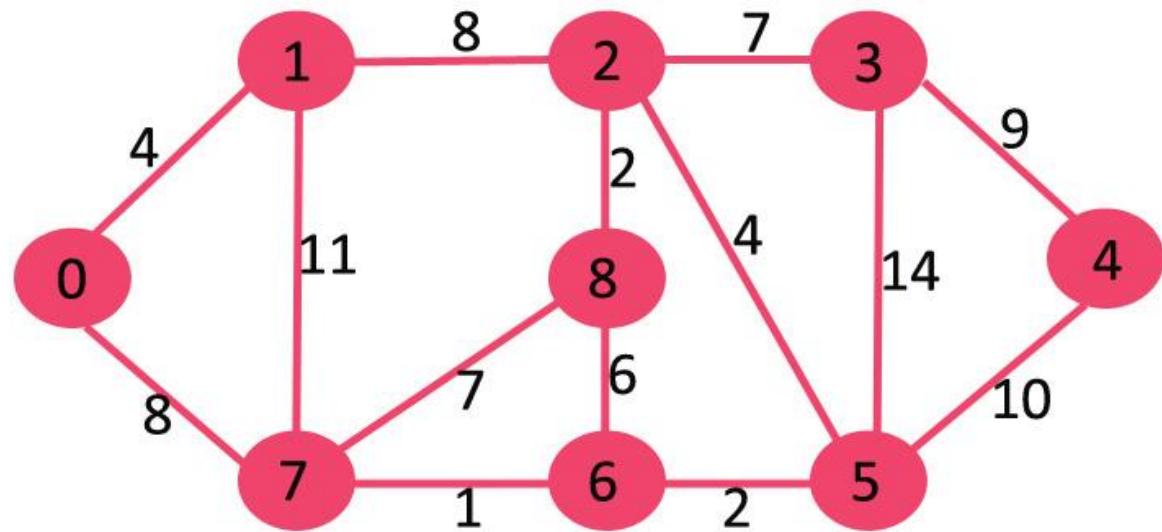
Bước 3.2. Thêm u vào mstSet

Bước 3.3. Cập nhật giá trị khóa của tất cả các đỉnh kề u. Với mỗi cạnh v kề u, nếu trọng số của cạnh u-v nhỏ hơn giá trị khóa trước đó của v thì cập nhật giá trị khóa là trọng số của u-v

Cây bao trùm

- Thuật toán Prim

- Ví dụ:



- Thiết lập $mstSet = \{0, INF, INF, INF, INF, INF, INF, INF\}$ với INF là giá trị dương vô cùng

Cây bao trùm

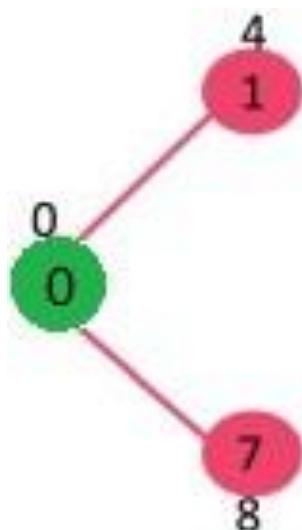
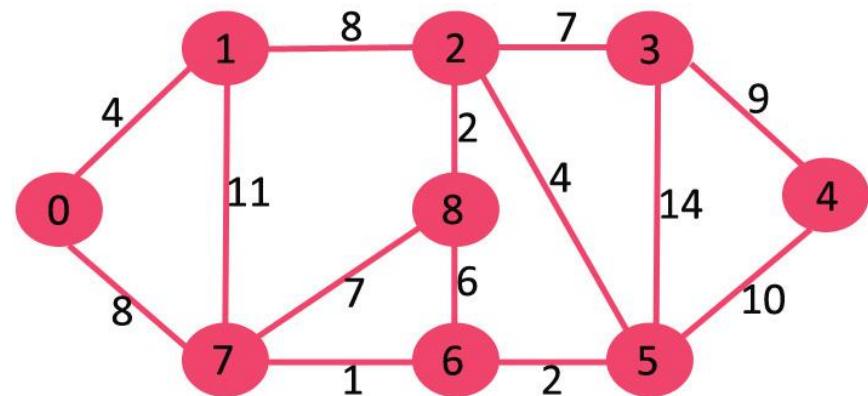
- Thuật toán Prim
 - Ví dụ:

Chọn đỉnh 0 (có khóa nhỏ nhất), thêm vào mstSet.

→ mstSet = {0}

Cập nhật khóa của tất cả các đỉnh kề 0 là 1 và 7

→ Khóa của các đỉnh 1 và 7 thành 4 và 8



Cây bao trùm

- Thuật toán Prim
 - Ví dụ:

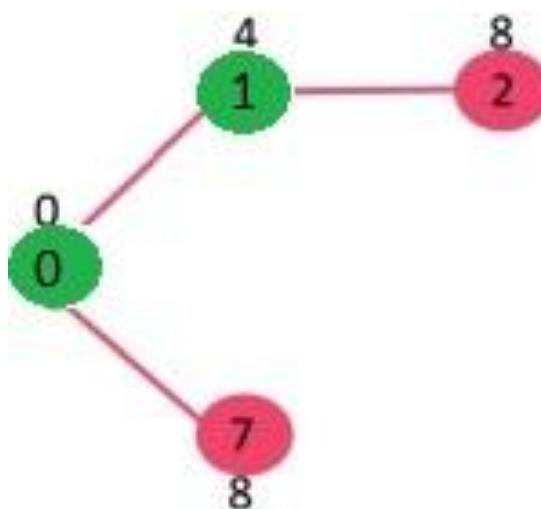
Chọn đỉnh có khóa nhỏ nhất không nằm trong MST.

→ Đỉnh 1 được chọn và thêm vào mstSet

→ mstSet = {0, 1}

Cập nhật khóa của tất cả các đỉnh kề 1 là 8

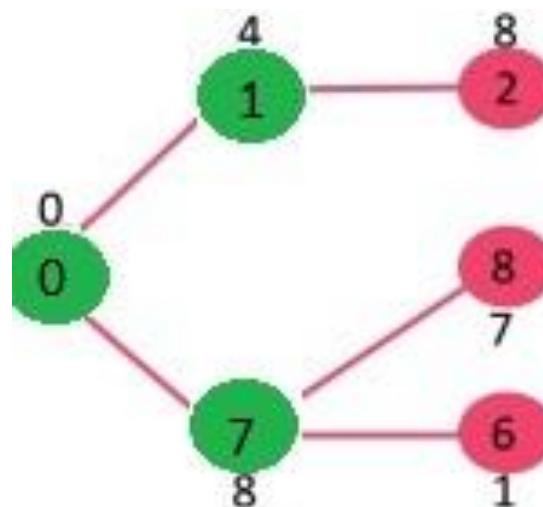
→ Khóa của đỉnh 2 thành 8



Cây bao trùm

- Thuật toán Prim
 - Ví dụ:

Chọn đỉnh có khóa nhỏ nhất không nằm trong MST.
→ Chọn đỉnh 2 hoặc 7, chọn 7 và thêm vào mstSet
→ $mstSet = \{0, 1, 7\}$
Cập nhật khóa của tất cả các đỉnh kề 7 là 6 và 8
→ Khóa của các đỉnh 6 và 8 thành 1 và 7



Cây bao trùm

- Thuật toán Prim
 - Ví dụ:

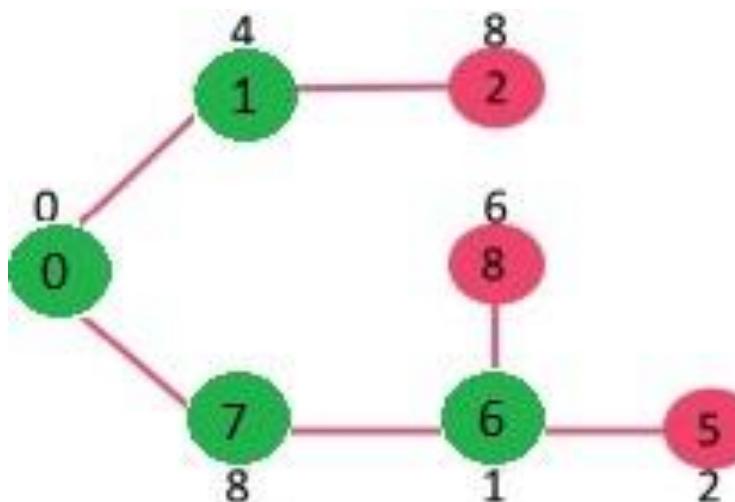
Chọn đỉnh có khóa nhỏ nhất không nằm trong MST.

→ Chọn đỉnh 6 và thêm vào mstSet

→ mstSet = {0, 1, 7, 6}

Cập nhật khóa của tất cả các đỉnh kề 6 là 5 và 8

→ Khóa của các đỉnh 5 và 8 thành 2 và 6

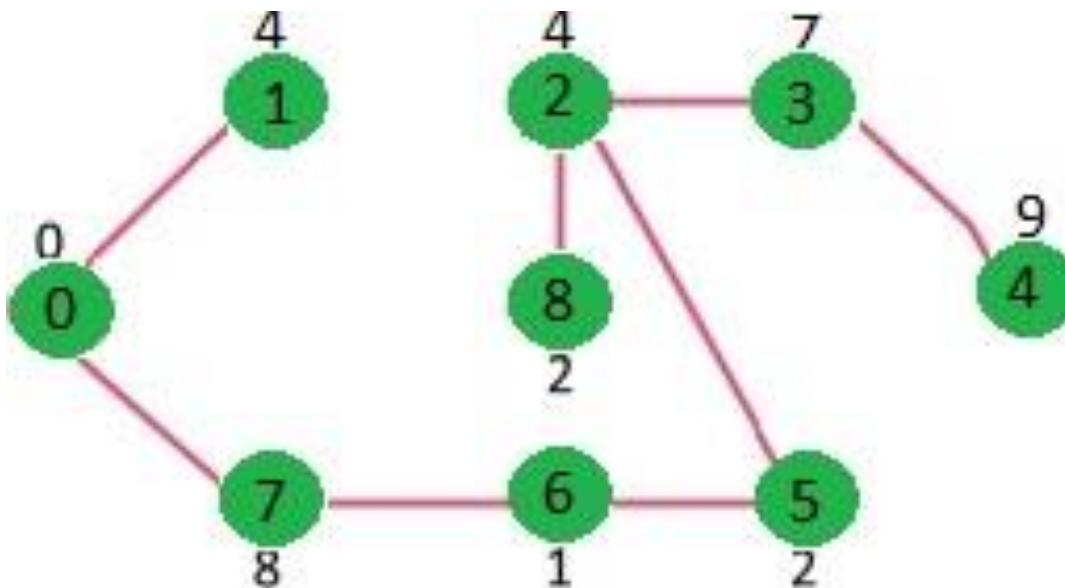
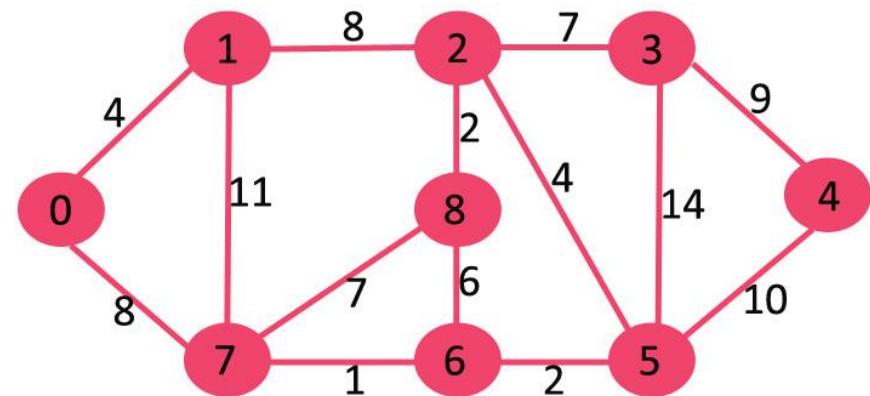


Cây bao trùm

- Thuật toán Prim
 - Ví dụ:

Tiếp tục các bước chọn cạnh như vậy cho đến khi tất cả các đỉnh của đồ thị nằm trong mstSet

Cuối cùng ta thu được cây bao trùm nhỏ nhất

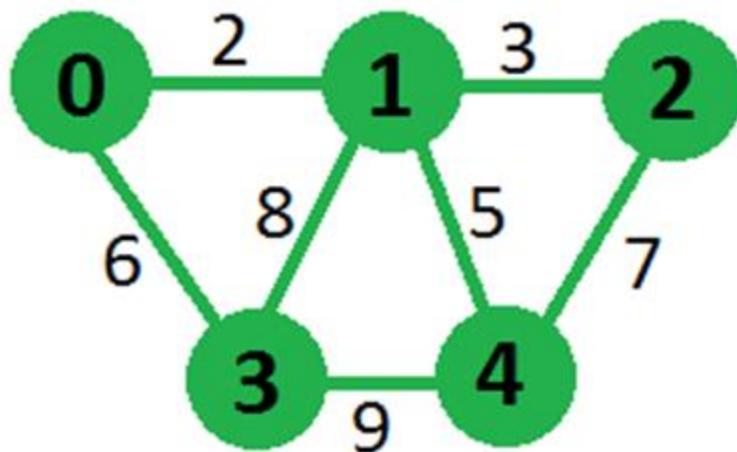


Cây bao trùm

- Ứng dụng
 - Thiết kế mạng: điện thoại, lưới điện, cáp TV, mạng máy tính, đường xá, ...
 - Sử dụng trong các thuật toán tìm nghiệm xấp xỉ giải các bài toán NP-khó như bài toán người du lịch, cây steiner nhỏ nhất, ...

Cây bao trùm

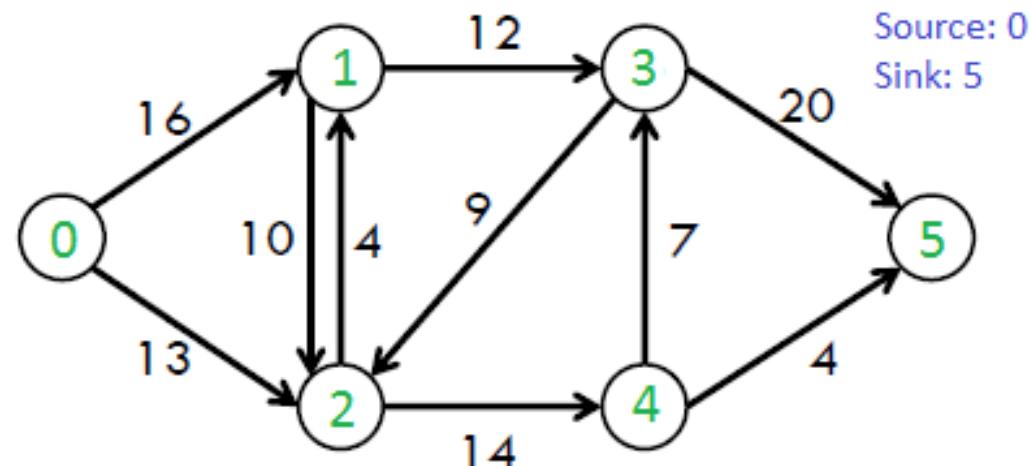
- Bài tập
 - Tìm cây có **tích** các trọng số là nhỏ nhất (Minimum Product Spanning Tree)



Cây trên có tích các trọng số nhỏ nhất là 180
→ Các cạnh là: 0-1, 1-2, 0-3 và 1-4

Luồng cực đại

- Mạng luồng (flow networks)
 - là một đồ thị có hướng thỏa các điều kiện:
 - Có 2 đỉnh đặc biệt là đỉnh phát (source), ký hiệu là s và đỉnh thu (sink), ký hiệu là t
 - Không có cung đi vào đỉnh phát hay ra từ đỉnh thu
 - Với mỗi cung (u, v) , được gán một số thực dương $c(u, v)$ là sức chứa



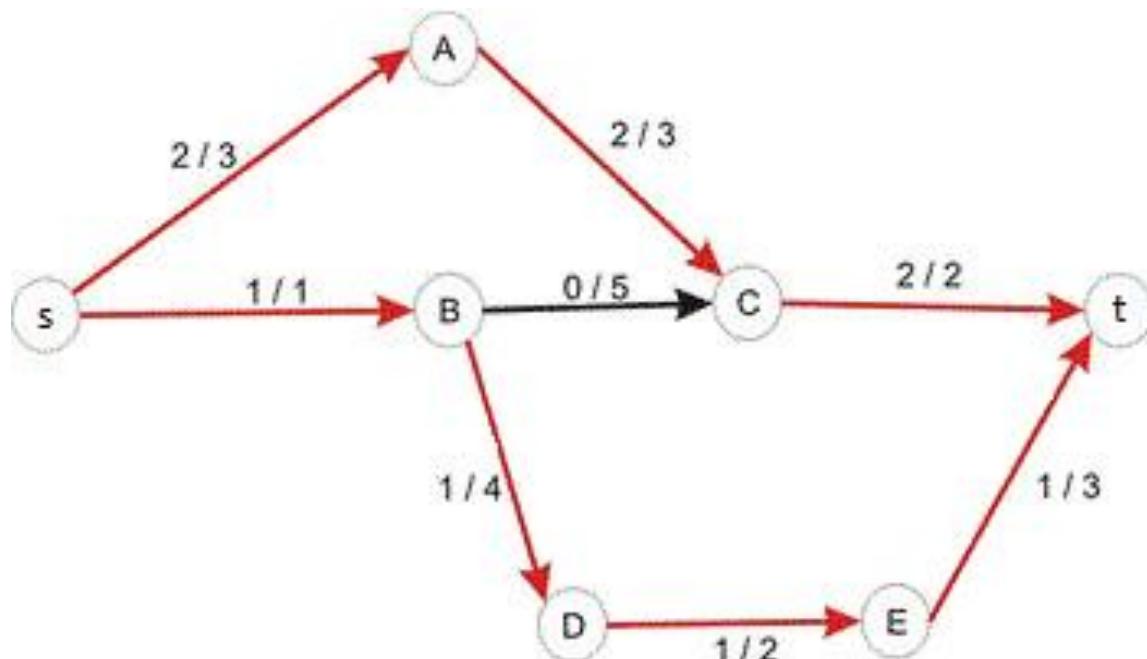
Biểu diễn mạng luồng bằng $G = (V, s, t, c)$ với $s, t \in V$

Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - Cho một mạng (network) có dạng một đồ thị có hướng $G = (V, E)$ (V là tập đỉnh, E là tập cạnh) có:
 - $e = (u, v) \in E$ như một kênh truyền tải từ u đến v có sức chứa $c(e) = c[u, v]$ (giá trị luồng tối đa có thể qua e)
 - Một đỉnh phát s (source) và một đỉnh thu t (sink)
 - Yêu cầu: với mỗi kênh truyền tải $e = [u, v] \in E$ cần xác định giá trị $f[u, v]$ ($f[u, v] \leq c[u, v]$) được gọi là luồng (flow) trên kênh e , sao cho:
$$\sum_{v \in V} f[v, u] = \sum_{w \in V} f[u, w] \quad (\forall u \in V / \{s, t\})$$
 (tổng luồng đi vào bằng tổng luồng đi ra)

Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - Ví dụ: hình dưới đây biểu diễn một luồng cực đại trên mạng và mỗi cạnh của nó được gán nhãn là $f[u, v]/c[u, v]$ (giá trị luồng và sức chứa của kênh)



Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Bài toán:** Cho một mạng luồng, chúng ta cần tìm một **luồng hợp lệ** sao cho tổng luồng ra từ đỉnh phát cũng như đi vào đỉnh thu là cực đại
 - Một **luồng hợp lệ** trên mạng G là một hàm $f: V \times V \rightarrow \mathbb{R}$ thỏa:
 - Luồng trên mỗi cung **không vượt quá** sức chứa của nó: $\forall u, v \in V, f(u, v) \leq c(u, v)$
 - Đổi xứng: $\forall u, v \in V, f(u, v) = -f(v, u)$
 - Bảo toàn luồng qua mỗi đỉnh: tổng luồng đi vào một đỉnh bằng tổng luồng đi ra khỏi đỉnh đó. Tức là:

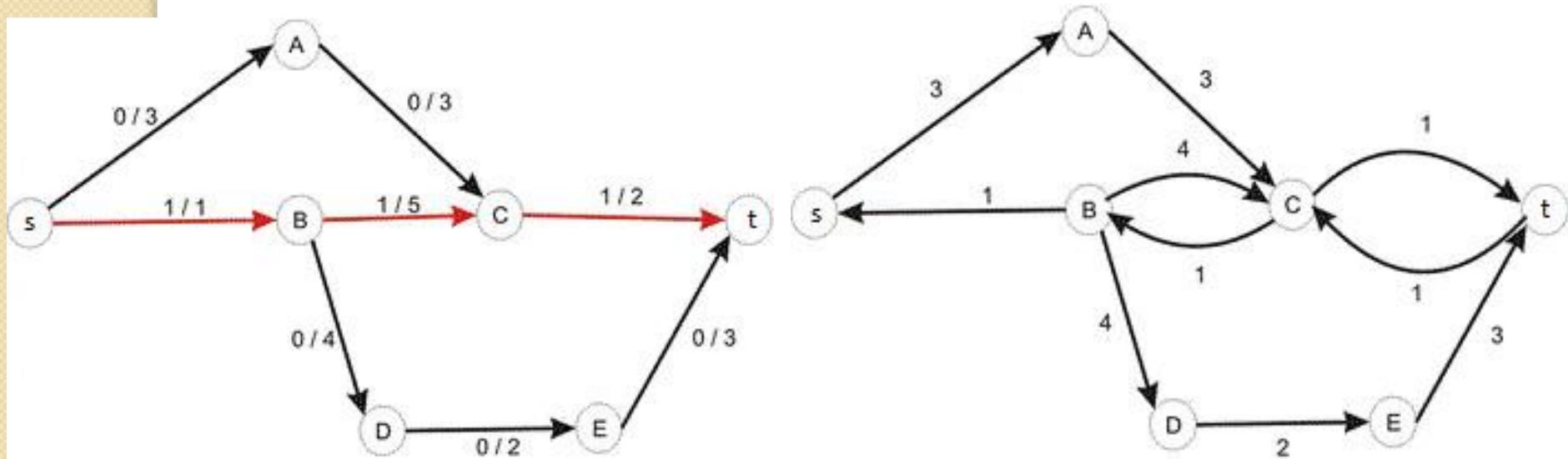
$$\forall u \notin \{s, t\}, \sum_{(k, u) \in E} f(k, u) = \sum_{(u, v) \in E} f(u, v)$$

Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Mạng thặng dư** (Residual network)
 - Mạng thặng dư cho chúng ta biết sức chứa còn lại trên mạng sau khi đã gửi một số luồng qua nó
 - Từ một mạng (V, s, t, c) và một luồng f ta xây dựng một mạng thặng dư (V, s, t, r) với $r(u,v) = c(u,v) - f(u,v)$
 - Mạng thặng dư có thể chứa một số cung mà mạng ban đầu không có. Chẳng hạn, khi gửi một luồng $f(u,v) > 0$ qua cung (u,v) thì ta cũng gửi một luồng $-f(u,v)$ qua cung (v,u) , khi đó $r(v,u) = c(v,u) + f(u,v) > 0$ có thể tạo ra cung (v,u) mà mạng ban đầu chưa có

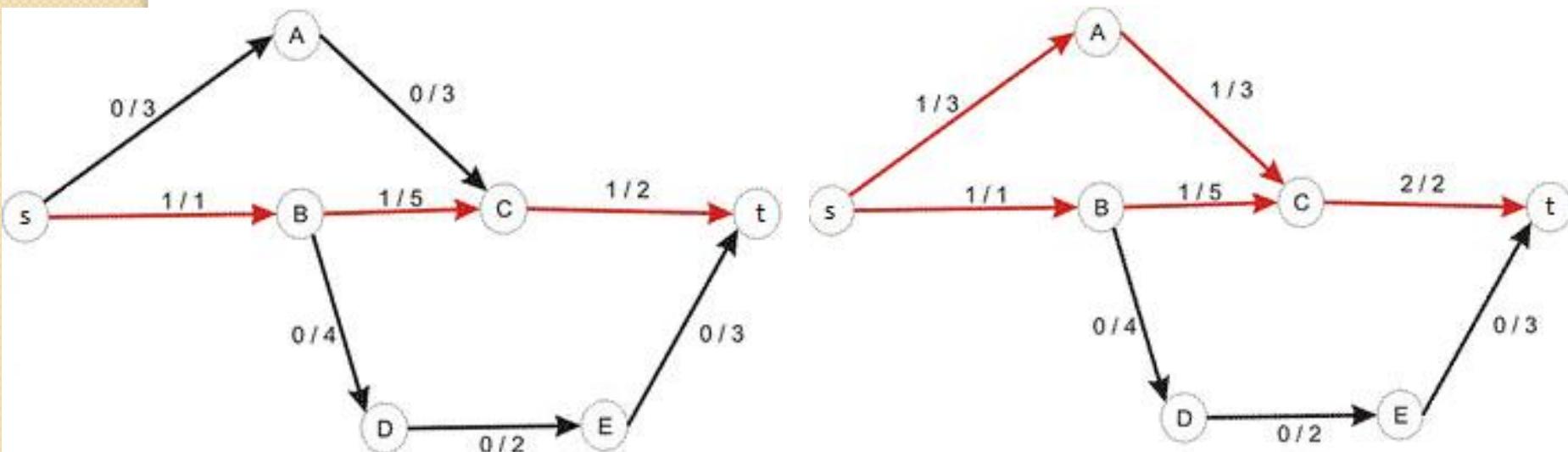
Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Mạng thặng dư** (Residual network)



Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Đường tăng luồng** (augmenting path)
 - Đường tăng luồng là một đường đi đơn từ đỉnh phát s (source) đến đỉnh thu t (sink) trong mạng thặng dư $G' = (V, s, t, r)$ mà kenh trên đường đi chưa bị bão hòa. Tức là $f'[u,v] < r[u,v]$, một kenh $e'(u,v)$ được gọi là bão hòa nếu $f'(u,v) = r(u,v)$

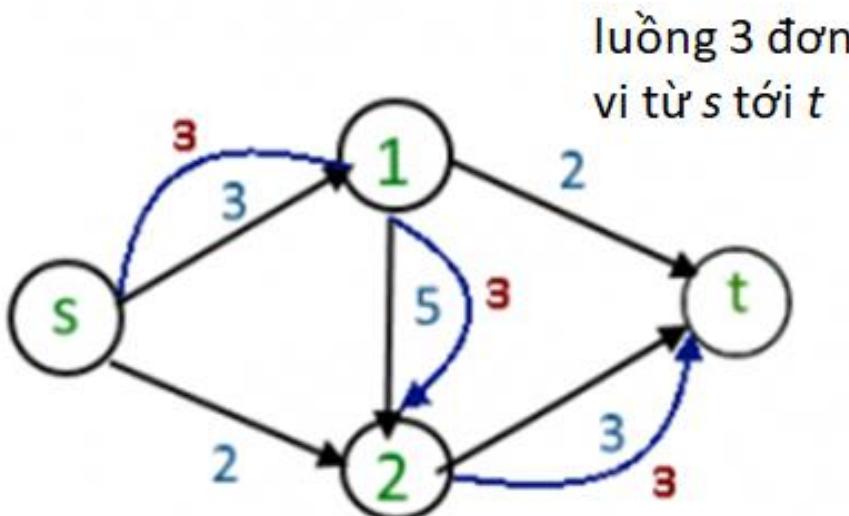


Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Ford–Fulkerson**
 - Ý tưởng chính của thuật toán Ford-Fulkerson là tăng dần giá trị của luồng trên mạng cho đến khi nó đạt cực đại
 - Với mỗi bước, trên mạng G hiện tại, chúng ta tìm một đường đi từ s tới t mà vẫn có thể gửi luồng qua được, đường đi này gọi là đường tăng luồng
 - Sau đó tiến hành gửi một luồng qua G sao cho luồng mới vẫn hợp lệ. Thuật toán kết thúc khi không còn tìm thấy đường tăng luồng nữa

Luồng cực đại

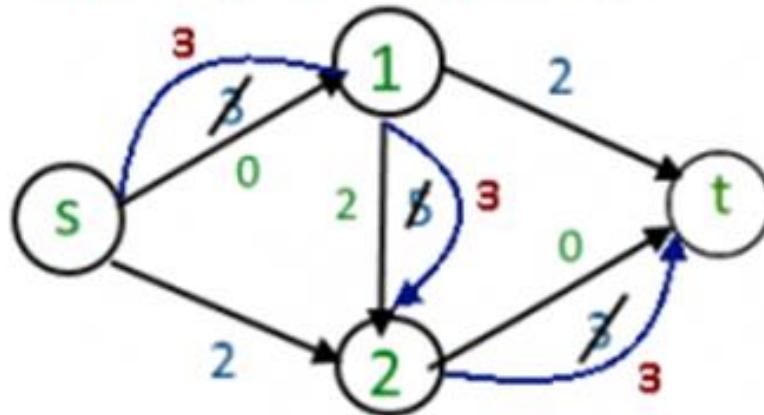
- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Ford–Fulkerson**
 - Phương pháp tham lam đơn giản
 - Cách tiếp cận tham lam đối với bài toán luồng cực đại là bắt đầu với luồng bằng không và luôn tạo ra các luồng với giá trị cao hơn. Tiếp tục bằng cách gửi thêm luồng trên một số đường đi từ s đến t



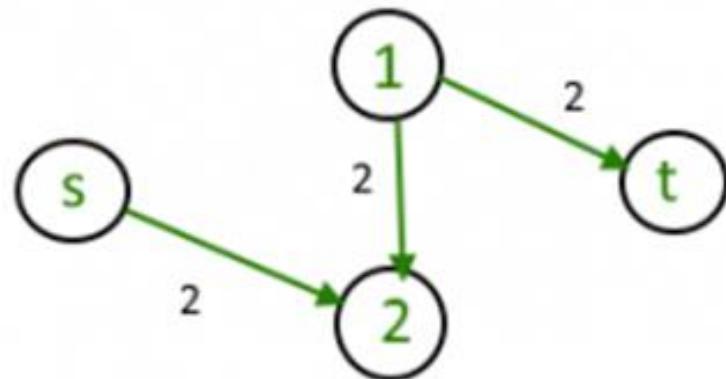
Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Ford–Fulkerson**
 - Phương pháp tham lam đơn giản

Sức chứa còn lại của các cạnh thể hiện bằng màu xanh lá cây



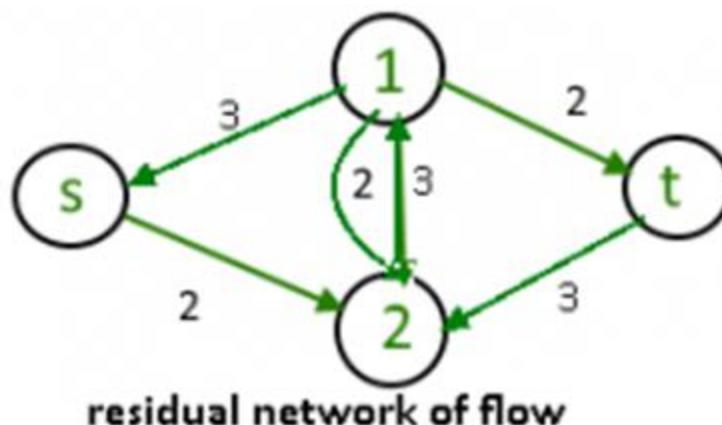
Sau khi loại bỏ các cạnh có sức chứa bằng 0



Với đồ thị trên, không còn luồng nào từ s tới t nữa \rightarrow luồng cực đại là **3** mà thực tế là **5**

Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Ford–Fulkerson**
 - Sử dụng mạng thặng dư
 - Ý tưởng là mở rộng thuật toán lam thông thường bằng cách cho phép các thao tác “hoàn tác”
 - Từ điểm mà thuật toán bị mắc kẹt như trong hình vừa rồi, chúng ta muốn định tuyến thêm luồng hai đơn vị theo cạnh $(s, 2)$, đi ngược cạnh $(1, 2)$, hoàn tác 2 trong số 3 đơn vị của lần lặp trước và dọc theo cạnh $(1, t)$



Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Ford–Fulkerson**
 - Sử dụng mạng thặng dư
 - Hai đỉnh không có cạnh nối trên mạng thặng dư có sức chứa bằng 0
 - Khởi tạo mạng thặng dư là mạng luồng ban đầu với các cạnh có sức chứa thặng dư bằng với sức chứa của mạng luồng ban đầu do chưa có luồng nào qua
 - Để tìm một đường tăng luồng \rightarrow sử dụng BFS hoặc DFS trên mạng thặng dư
 - Sử dụng BFS sẽ tìm được đường đi từ đỉnh phát đến đỉnh thu

Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Ford–Fulkerson**
 - Sử dụng mạng thặng dư
 - BFS cũng xây dựng mảng parent[] để duyệt đường đi đã tìm được và tìm kiếm luồng có thể có qua đường đi này bằng cách tìm **sức chứa thặng dư nhỏ nhất** dọc theo đường đi. Sau đó, luồng đường đi tìm thấy được thêm vào luồng tổng thể
 - Cần phải cập nhật các sức chứa thặng dư trên đồ thị thặng dư → Thực hiện trừ luồng đường đi từ tất cả các cạnh dọc theo đường đi và thêm luồng đường đi dọc theo các cạnh ngược lại (vì sau này có thể cần gửi luồng theo hướng ngược lại)
 - **Độ phức tạp về thời gian:** $O(|E||V|^3)$

Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Edmonds-Karp**
 - Ý tưởng của thuật toán Edmonds-Karp là sử dụng thuật toán BFS trong Ford Fulkerson và BFS luôn chọn đường đi với số cạnh tối thiểu

Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Dinic**
 - Trong thuật toán Edmonds-Karp, BFS được sử dụng để tìm đường tăng luồng và gửi luồng qua đường này
 - Trong thuật toán Dinic, BFS được sử dụng để kiểm tra xem có thể **thêm luồng được** không và dùng để xây dựng **đồ thị mức**
 - Trong đồ thị mức, mỗi đỉnh được gán một mức là khoảng cách ngắn nhất (tính bằng số cạnh) của nút đó tính từ đỉnh phát
 - Khi đồ thị mức được xây dựng, nhiều luồng được gửi cùng lúc sử dụng đồ thị mức này (**đây là lý do thuật toán này hiệu quả hơn thuật toán Edmond Karp**)

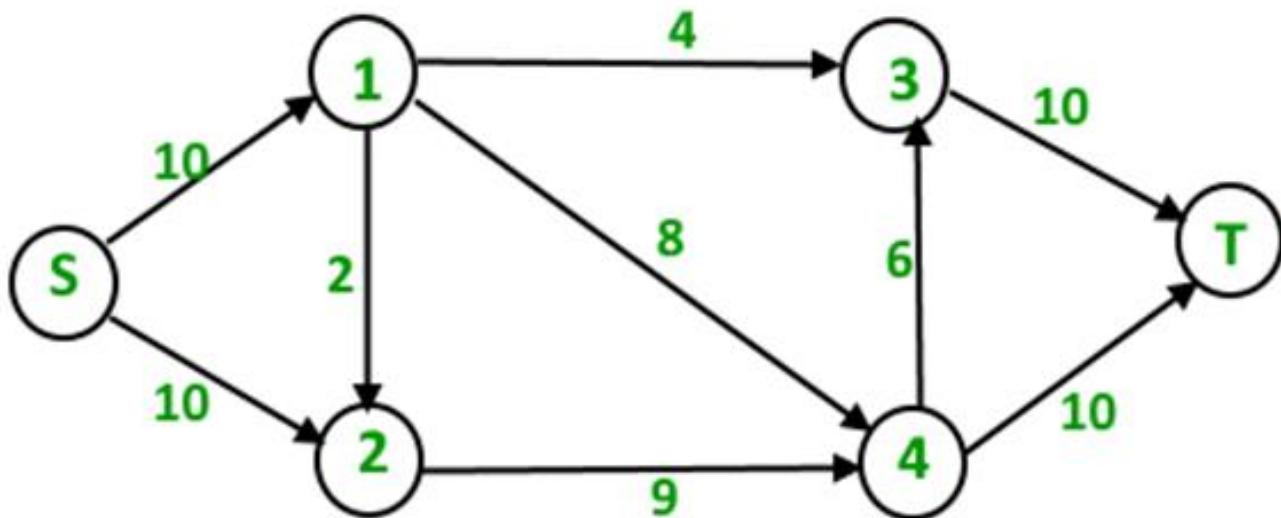
Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Dinic, các bước:**
 - 1) Khởi tạo đồ thị thặng dư G (giống đồ thị gốc)
 - 2) Thực hiện BFS trên G để xây dựng một đồ thị mức (hoặc gán mức cho các đỉnh) và kiểm tra xem có thể **thêm luồng được** không
 - a) Nếu không thể thêm luồng thì kết thúc
 - b) Gửi nhiều luồng trong G sử dụng đồ thị mức cho đến khi không thể gửi thêm luồng nào nữa. Sử dụng đồ thị mức ở đây có nghĩa là trong mọi luồng, mức của các nút thuộc đường đi phải là $0, 1, 2, \dots$ (theo thứ tự) từ s tới t

Độ phức tạp về thời gian: $O(|E|/V|^2)$

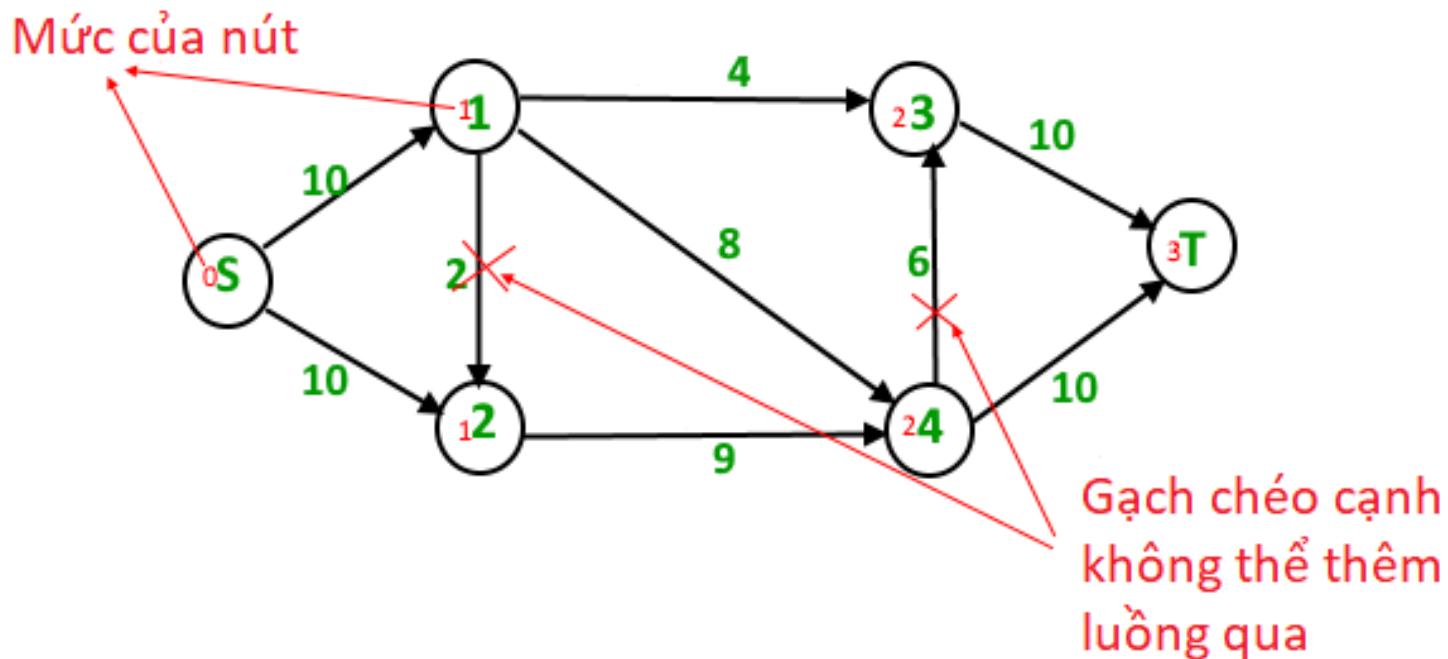
Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Dinic, mô phỏng:**
 - Đồ thị thặng dư khởi tạo như hình bên dưới (giống như đồ thị gốc)
 - Gán biến TotalFlow = 0



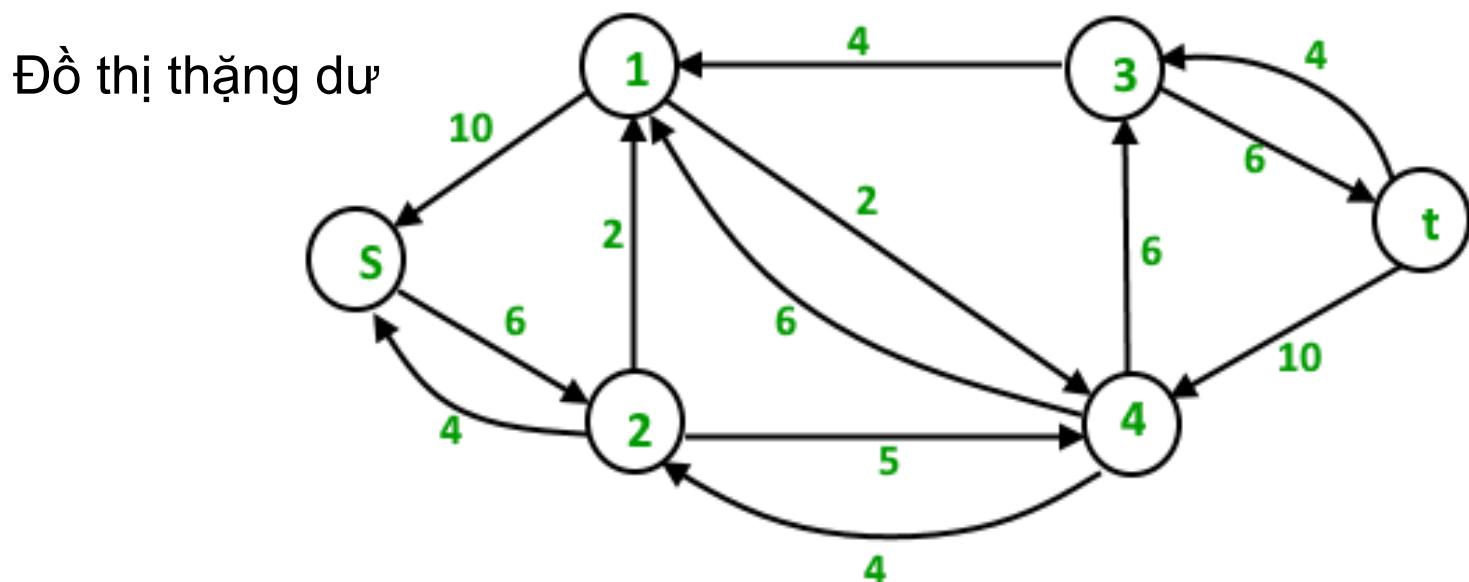
Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Dinic, mô phỏng:**
 - Vòng lặp đầu tiên: gán mức cho tất cả các đỉnh sử dụng thuật toán BFS và kiểm tra xem có thể thêm luồng được không (có đường đi $s-t$ trên đồ thị thặng dư)



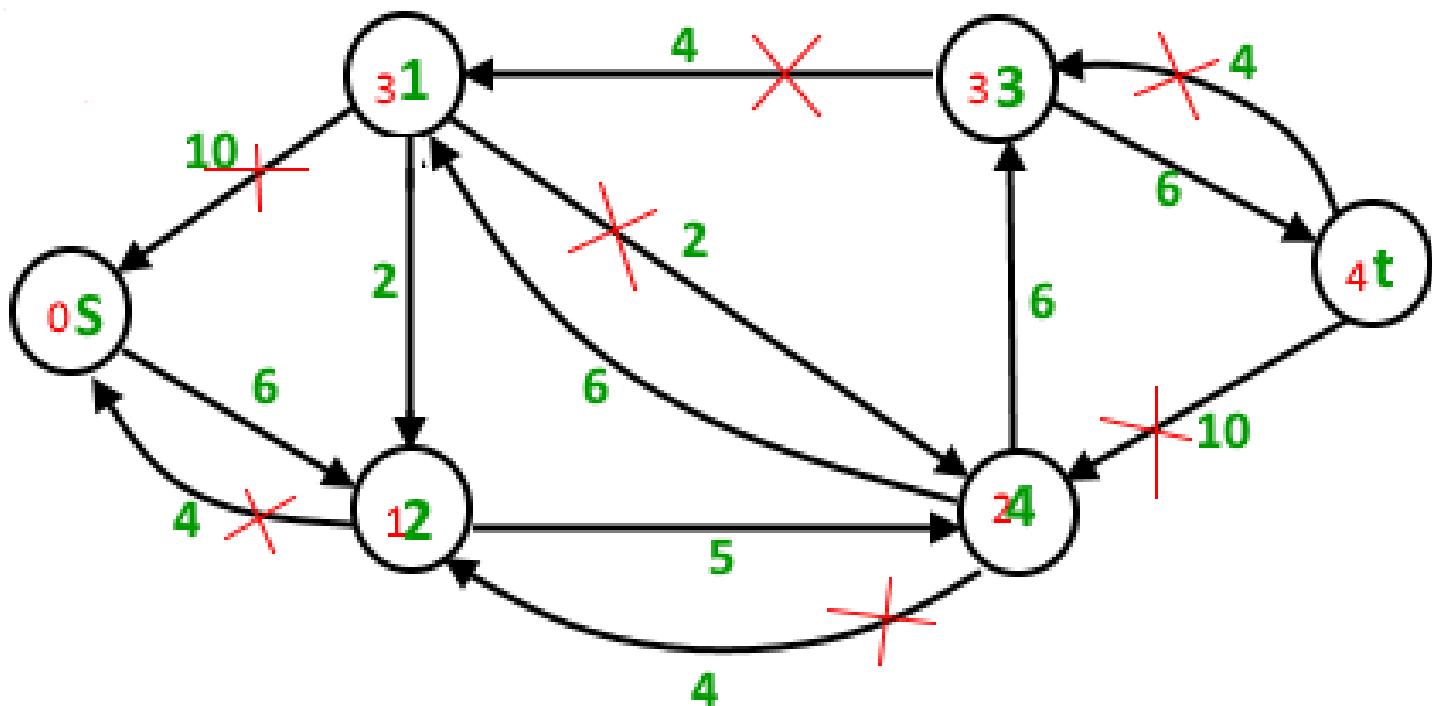
Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Dinic, mô phỏng:**
 - Gửi ba luồng cùng một lúc
 - 4 đơn vị của luồng trên đường đi $s - 1 - 3 - t$.
 - 6 đơn vị của luồng trên đường đi $s - 1 - 4 - t$.
 - 4 đơn vị của luồng trên đường đi $s - 2 - 4 - t$.
 - Totalflow = Totalflow + 4 + 6 + 4 = 14



Luồng cực đại

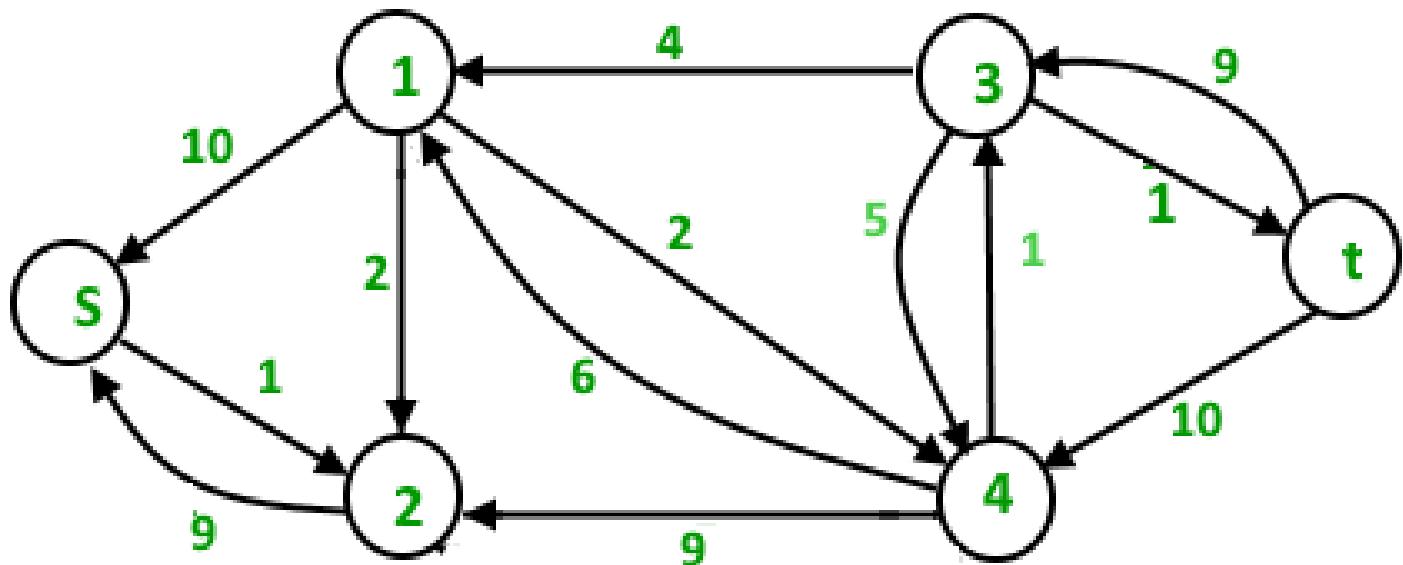
- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Dinic, mô phỏng:**
 - Vòng lặp thứ hai: gán mức cho tất cả các nút sử dụng BFS trên đồ thị thặng dư thu được ở bước trước và kiểm tra xem có thể thêm luồng được nữa không



Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Dinic, mô phỏng:**
 - Chỉ có thể gửi thêm một luồng tại thời điểm này
→ 5 đơn vị của luồng trên đường đi $s - 2 - 4 - 3 - t$
 $\text{Totalflow} = \text{Totalflow} + 5 = 19$

Đồ thị thặng dư mới



Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Thuật toán Dinic, mô phỏng:**
 - Vòng lặp thứ ba: Chạy thuật toán BFS, tạo đồ thị mức và kiểm tra xem có thêm được luồng nữa không. Tại thời điểm này, không có đường đi s-t trong đồ thị thặng dư nên thuật toán kết thúc

Luồng cực đại

- Luồng cực đại trên mạng - Maxflow network
 - **Ứng dụng**
 - Cực đại hóa giao thông với lưu lượng giao thông giới hạn
 - Cực đại hóa luồng gói tin trong các mạng máy tính

Luồng cực đại

- Bài tập
 - **Bài 1:** Xét đồ thị tương ứng hệ thống ống dẫn dầu. Trong đó các ống tương ứng với các cung, điểm phát là tàu chở dầu, điểm thu là bể chứa, các điểm nối của ống là các nút của đồ thị. Khả năng thông qua của các cung tương ứng là tiết diện các ống. Cần phải tìm luồng dầu lớn nhất có thể bơm từ tàu chở dầu vào bể chứa

Luồng cực đại

- Bài tập
 - **Bài 2:** Bài toán cặp ghép: có m chàng trai và n cô gái. Mỗi chàng trai ưa thích một số cô gái. Hãy tìm cách ghép cặp sao cho số cặp ghép được là nhiều nhất

TỔNG KẾT

1. Tổng quan