

Khai Thác Dữ Liệu Đồ Thị

NỀN TẢNG LÝ THUYẾT ĐỒ THỊ

Giảng viên: Lê Ngọc Thành

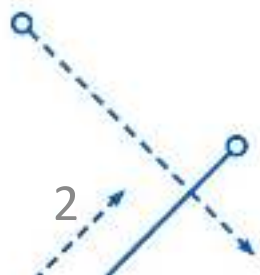
Email: lnthanh@fit.hcmus.edu.vn



fit@hcmus

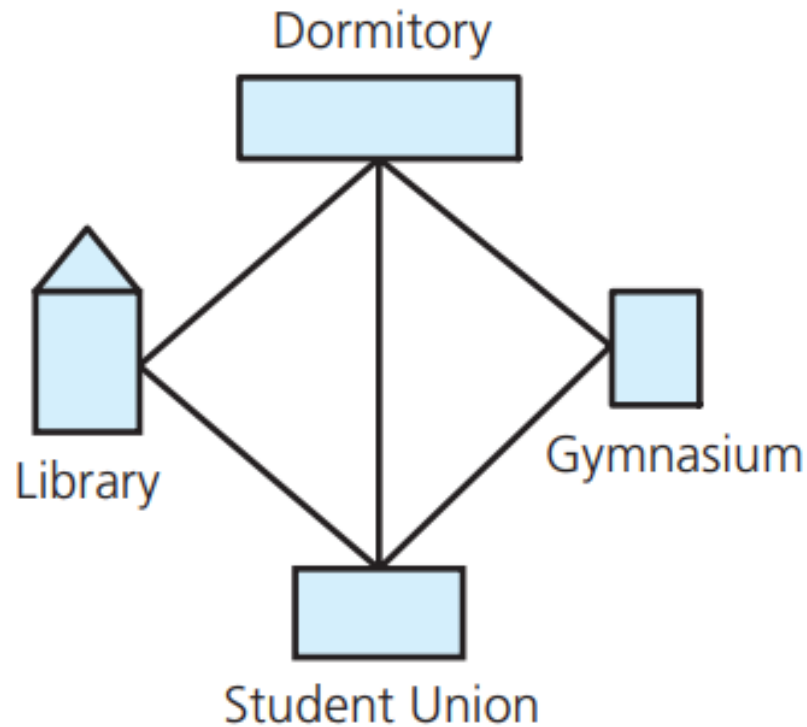
Nội dung

- **Khái niệm và thuộc tính đồ thị**
- Các loại đồ thị
- Lưu trữ đồ thị
- Một số thuật toán cơ bản trên đồ thị
 - Thuật toán duyệt đồ thị
 - Thuật toán sắp xếp topo
 - Thuật toán cây khung
 - Thuật toán đường đi ngắn nhất
- Lát cắt và luồng
- So khớp đồ thị



Đồ thị

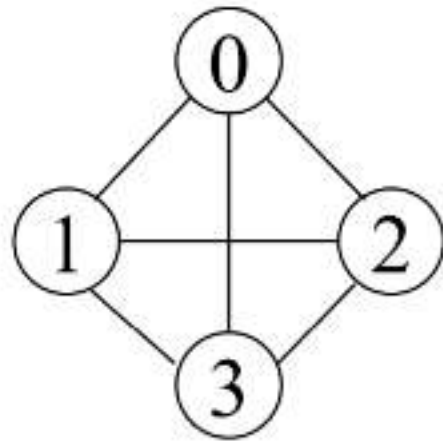
- Một **đồ thị** G bao gồm hai tập: một tập V chứa các đỉnh (vertice, node), và một tập E chứa các cạnh kết nối các đỉnh với nhau.



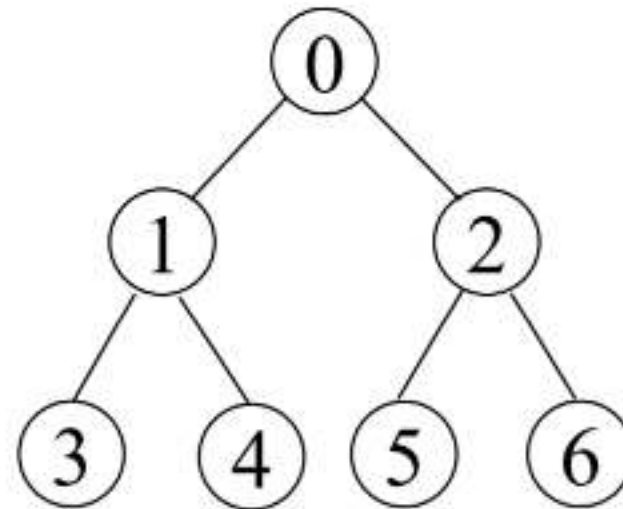
Ví dụ, bản đồ của một trường Đại học là một đồ thị với các đỉnh là các tòa nhà và các cạnh là đường đi giữa các tòa.



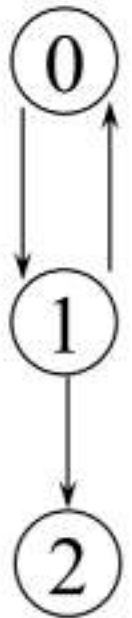
Ví dụ về đồ thị



G_1



G_2



G_3

$$V(G_1) = \{0, 1, 2, 3\}$$

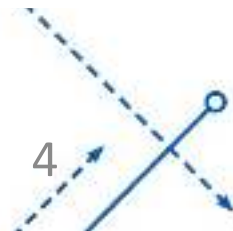
$$V(G_2) = \{0, 1, 2, 3, 4, 5, 6\}$$

$$V(G_3) = \{0, 1, 2\}$$

$$E(G_1) = \{(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)\}$$

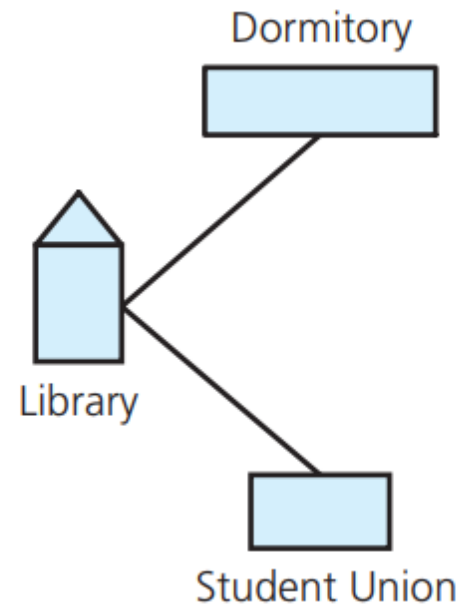
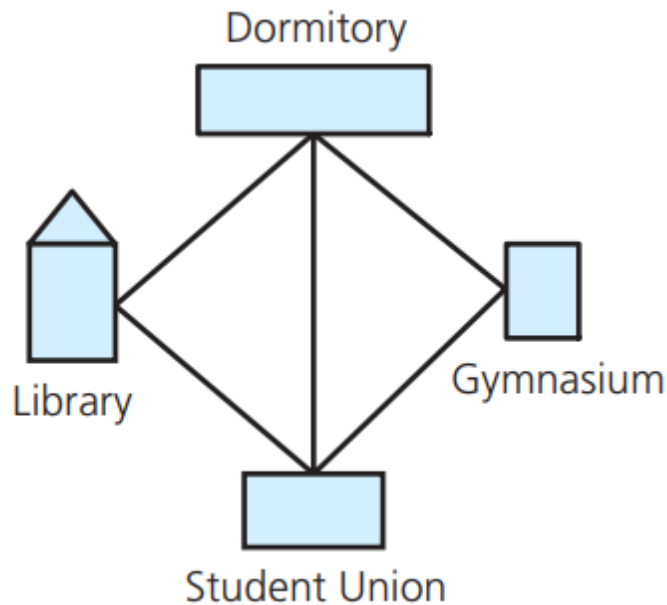
$$E(G_2) = \{(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6)\}$$

$$E(G_3) = \{<0, 1>, <1, 0>, <1, 2>\}$$



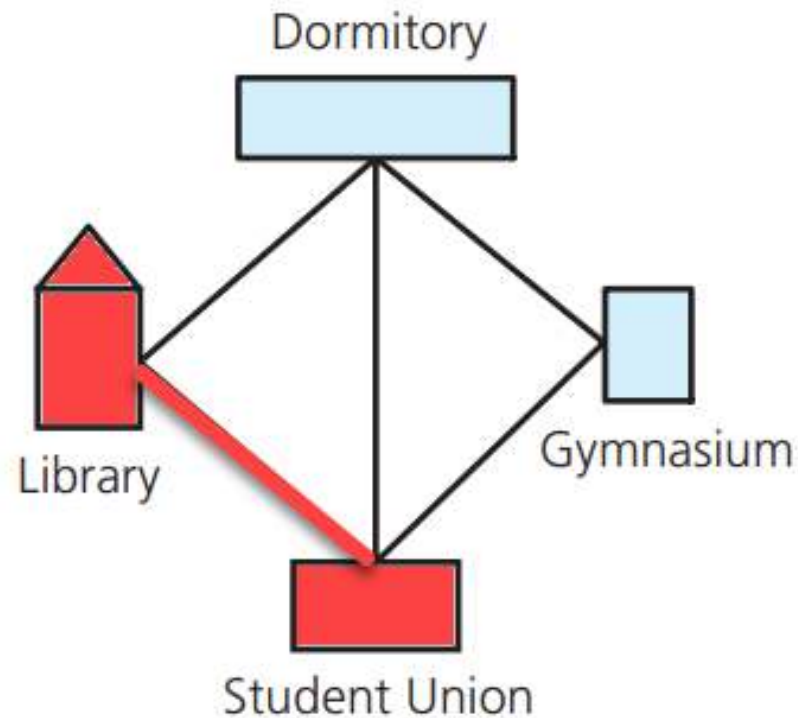
Đồ thị con

- Đồ thị con bao gồm tập con các đỉnh và tập con các cạnh của đồ thị.

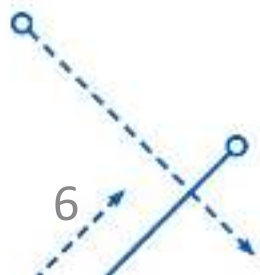


Liên kề

- Hai đỉnh của một đồ thị là liên kề nếu được liên kết bởi một cạnh.

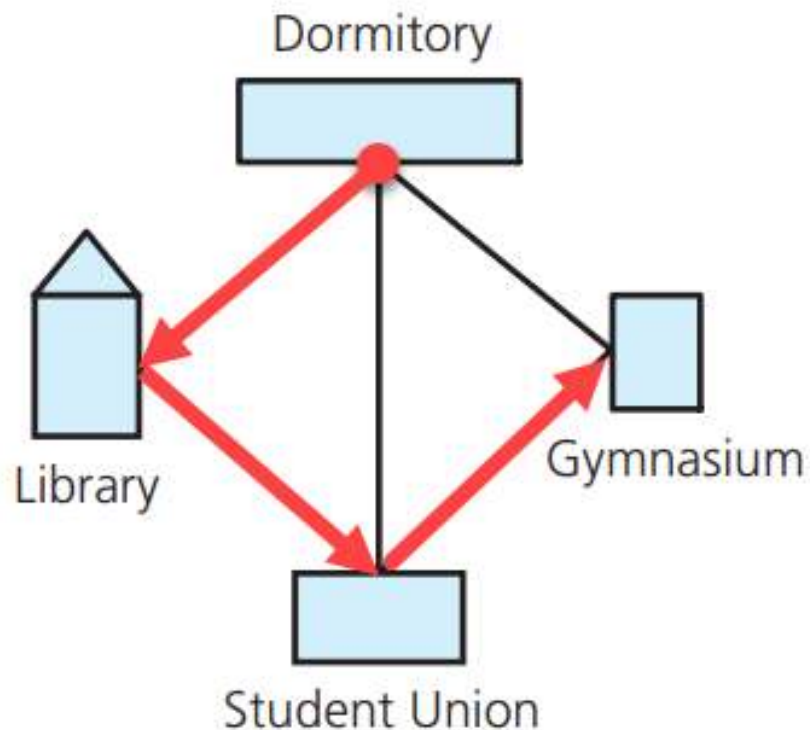


The Library và the Student Union là liên kề.

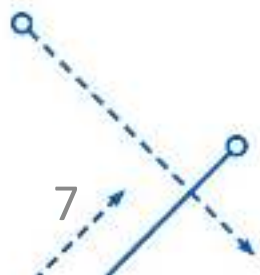


Đường đi (Path)

- **Đường đi** giữa hai đỉnh là một chuỗi các cạnh bắt đầu tại một đỉnh và kết thúc ở một đỉnh khác.

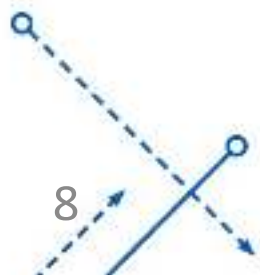
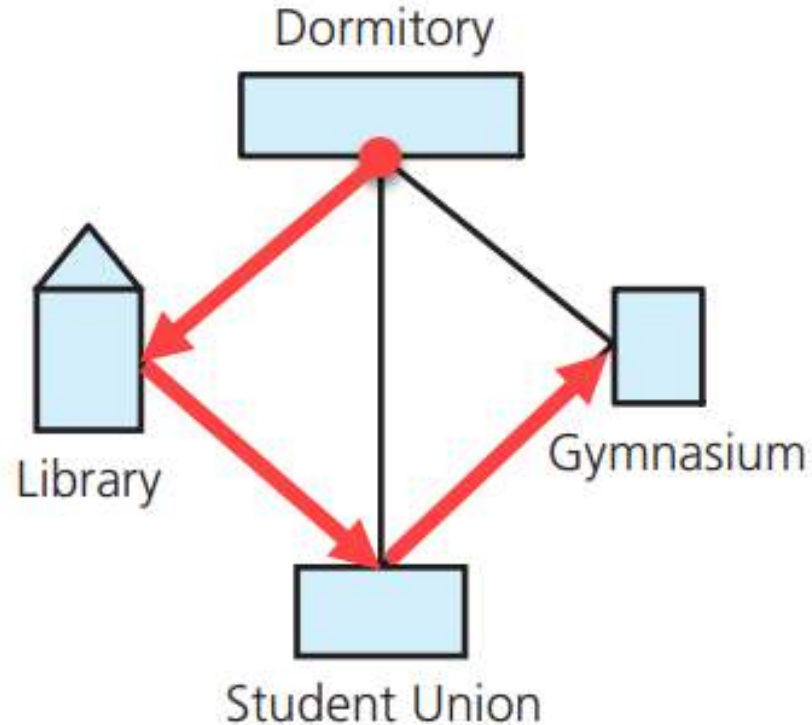


Có một đường đi bắt đầu từ Dormitory, qua Library, đến Student Union, và cuối cùng đến Gymnasium,



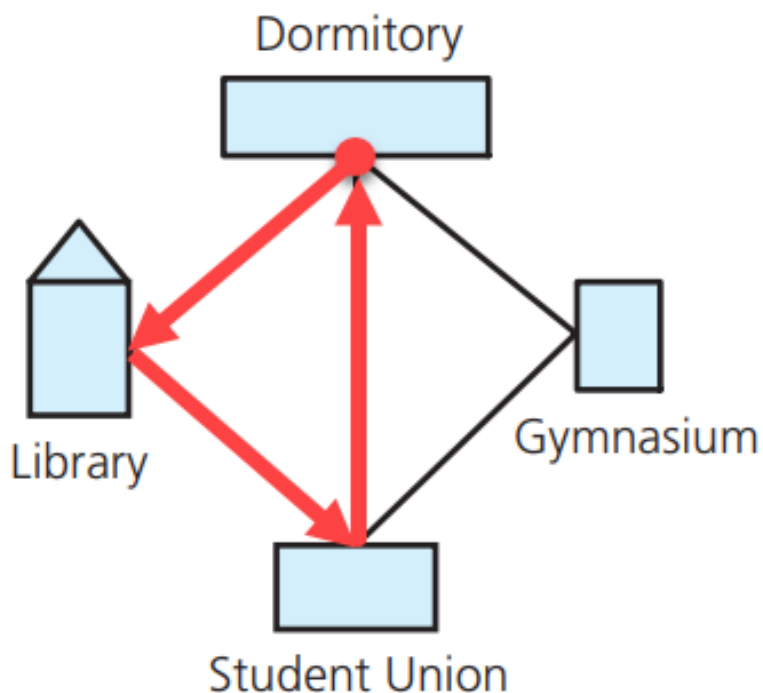
Đường đi đơn (Simple Path)

- **Đường đi đơn** là đường không đi qua cùng một đỉnh nhiều hơn một lần.



Chu trình (Cycle)

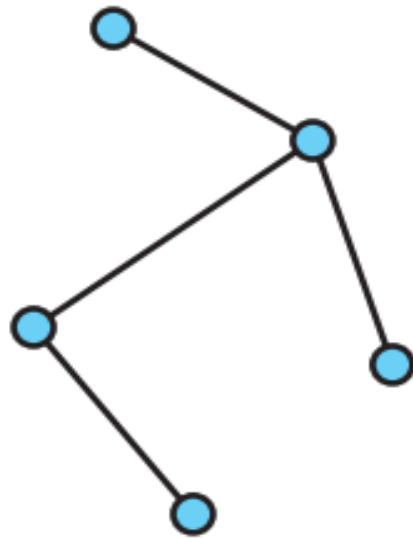
- **Chu trình** là một đường đi bắt đầu và kết thúc tại cùng một đỉnh; một chu trình đơn là một chu trình không đi qua các đỉnh khác nhiều hơn một lần.



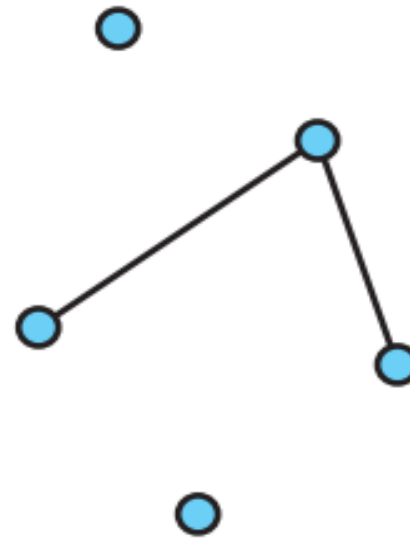
Đường đi Library–Student Union–Gymnasium–Dormitory–Library là một chu trình đơn

Đồ thị liên thông và không liên thông

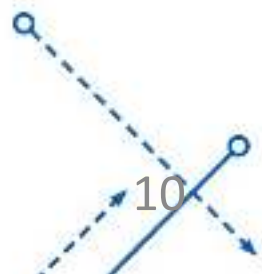
- Một đồ thị là **liên thông** nếu mỗi cặp đỉnh phân biệt có một đường đi giữa chúng.
- Có nghĩa là, trong một đồ thị liên thông, bạn có thể đi từ bất kỳ đỉnh nào đến bất kỳ đỉnh nào khác bằng cách đi theo một đường dẫn.



Connected Graph

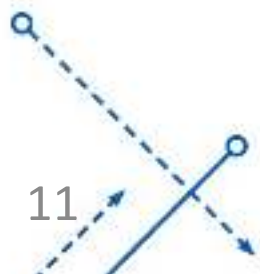
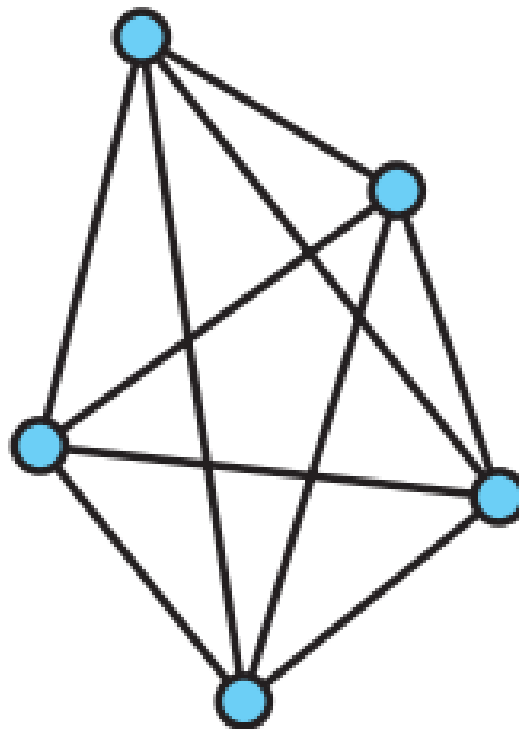


Disconnected Graph



Đồ thị đủ (Complete graph)

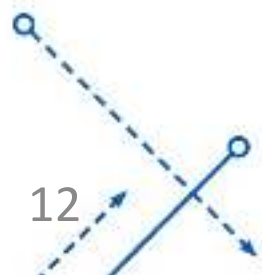
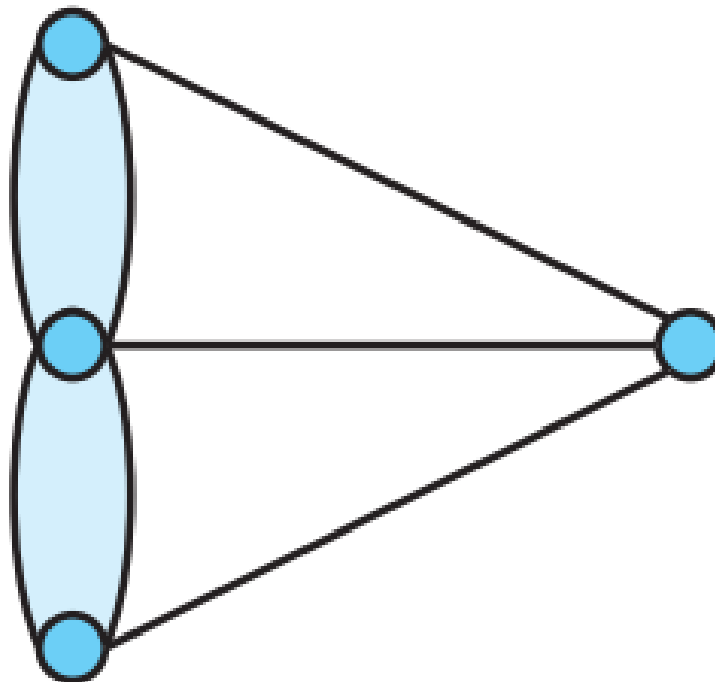
- Trong một **đồ thị đủ**, mỗi cặp đỉnh phân biệt có một cạnh ở giữa chúng.
- Một đồ thị đủ cũng là liên thông, nhưng đồ thị liên thông thì chưa chắc đủ.



Đa đồ thị (Multigraph)

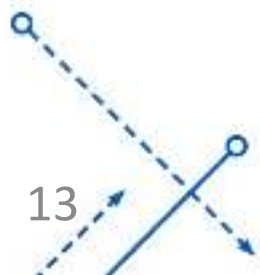
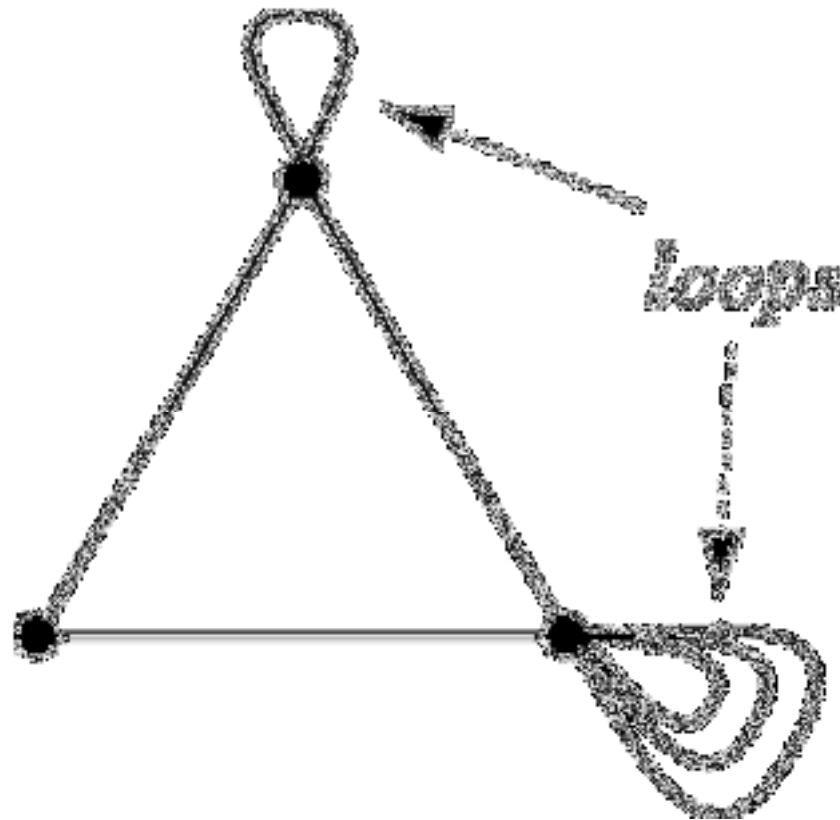
=> Graph ko cho phép tồn tại 2 cạnh giữa 2 đỉnh (giữa 2 đỉnh chỉ có 1 cạnh)

- Bởi vì một đồ thị có một tập hợp các cạnh, một đồ thị không thể có các cạnh trùng lặp giữa các đỉnh.
- Tuy nhiên, một **đa đồ thị** cho phép có nhiều cạnh.
 - Do đó, một đa đồ thị không phải là một đồ thị. Các cạnh của đồ thị không thể bắt đầu và kết thúc ở cùng một đỉnh.



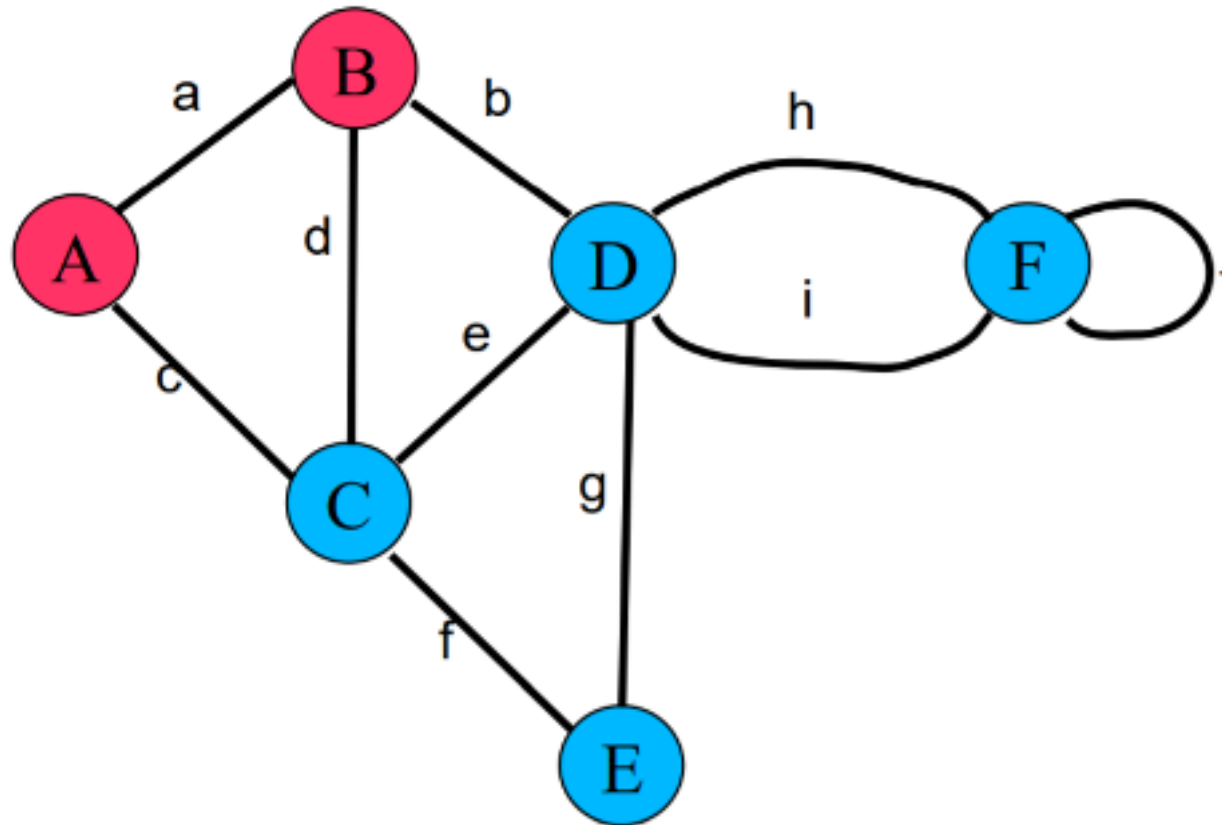
Self edge

- Các cạnh của đồ thị bắt đầu và kết thúc tại cùng một đỉnh được gọi là **self edge** hoặc vòng lặp.



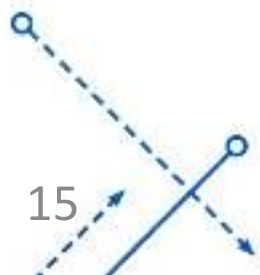
Gán nhãn (Label)

- Bạn có thể *gán nhãn* các cạnh của đồ thị.



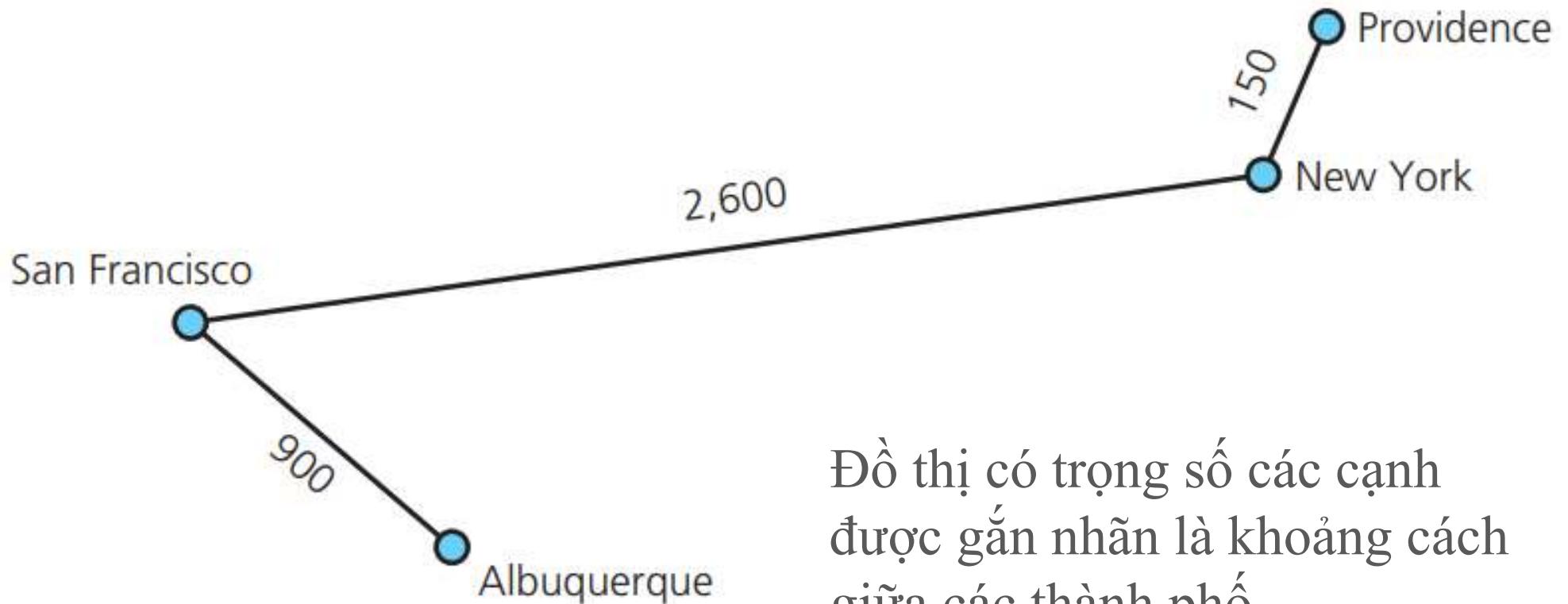
Nội dung

- Khái niệm và thuộc tính đồ thị
- **Các loại đồ thị**
- Lưu trữ đồ thị
- Một số thuật toán cơ bản trên đồ thị
 - Thuật toán duyệt đồ thị
 - Thuật toán sắp xếp topo
 - Thuật toán cây khung
 - Thuật toán đường đi ngắn nhất
- Lát cắt và luồng
- So khớp đồ thị



Đồ thị trọng số (Weighted graph)

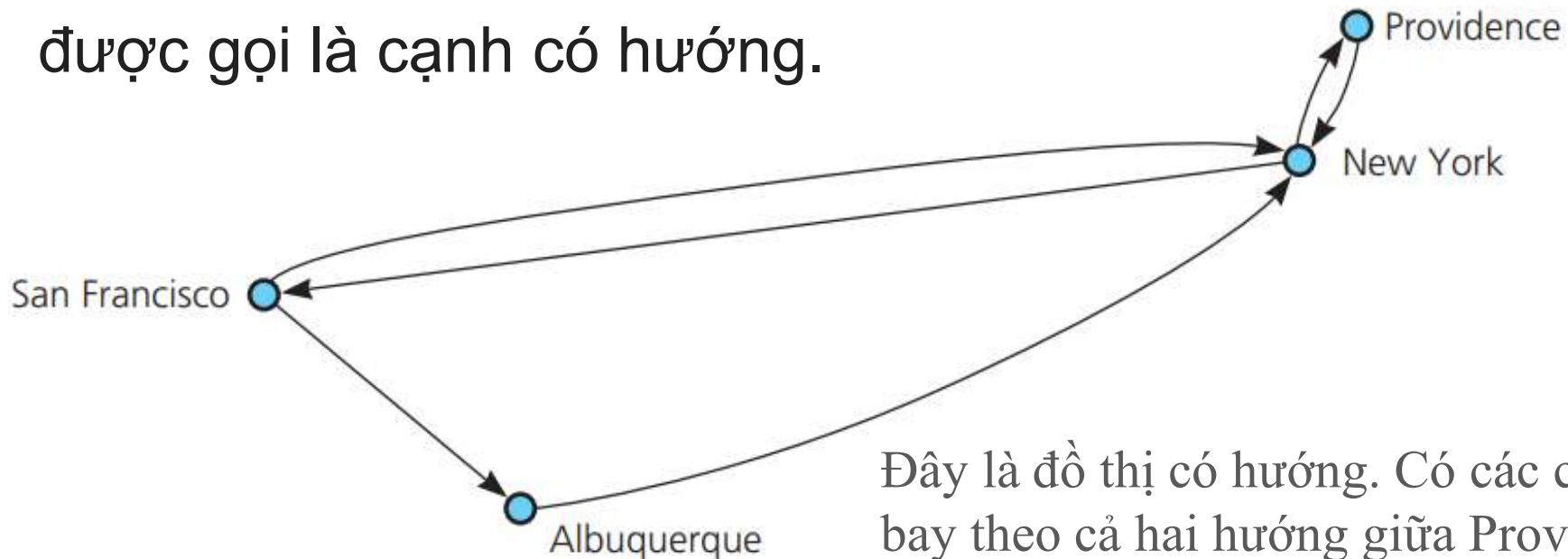
- Khi những nhãn biểu diễn là giá trị số, đồ thị được gọi là **đồ thị trọng số**.



Đồ thị có trọng số các cạnh được gắn nhãn là khoảng cách giữa các thành phố

Đồ thị vô hướng và có hướng

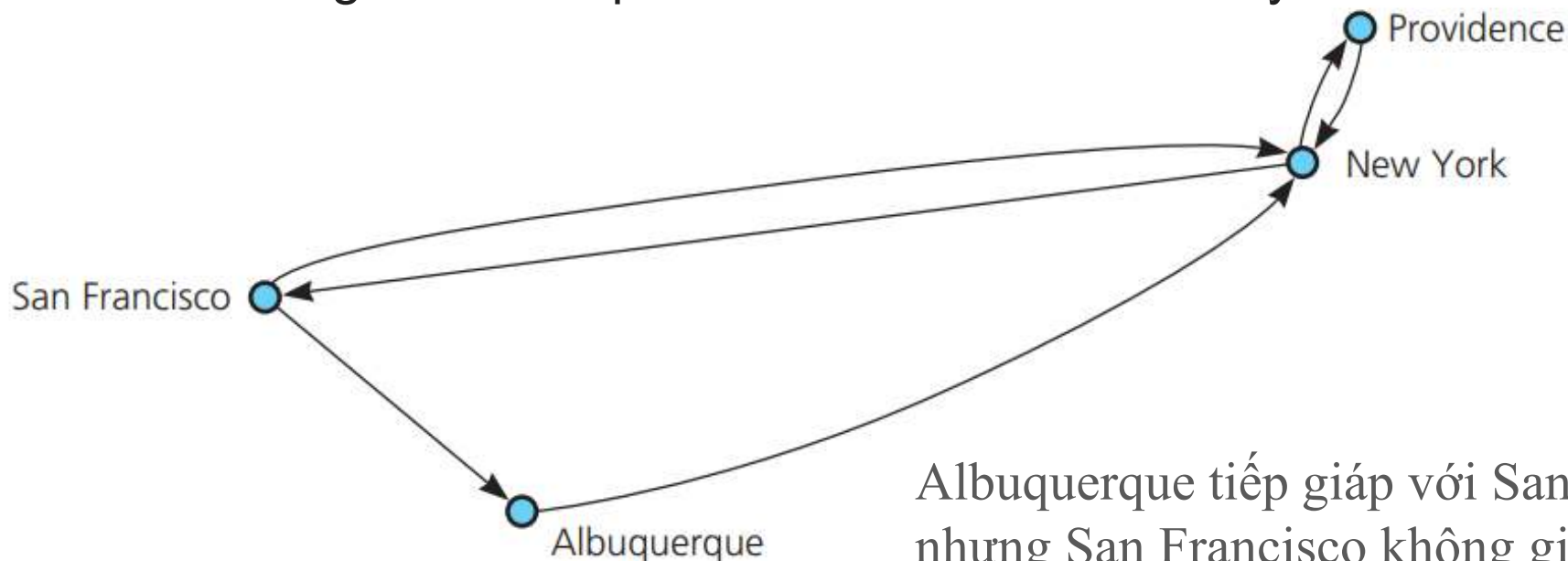
- Trong **đồ thị vô hướng**, các cạnh không có hướng. Nghĩa là, ta có thể di chuyển theo một trong hai hướng dọc theo các cạnh giữa các đỉnh trong đồ thị vô hướng.
- Ngược lại, mỗi cạnh trong **đồ thị có hướng**, sẽ có hướng và được gọi là cạnh có hướng.



Đây là đồ thị có hướng. Có các chuyến bay theo cả hai hướng giữa Providence và New York, nhưng mặc dù có chuyến bay từ San Francisco đến Albuquerque, không có chuyến bay nào từ Albuquerque đến San Francisco.

Đồ thị vô hướng và có hướng

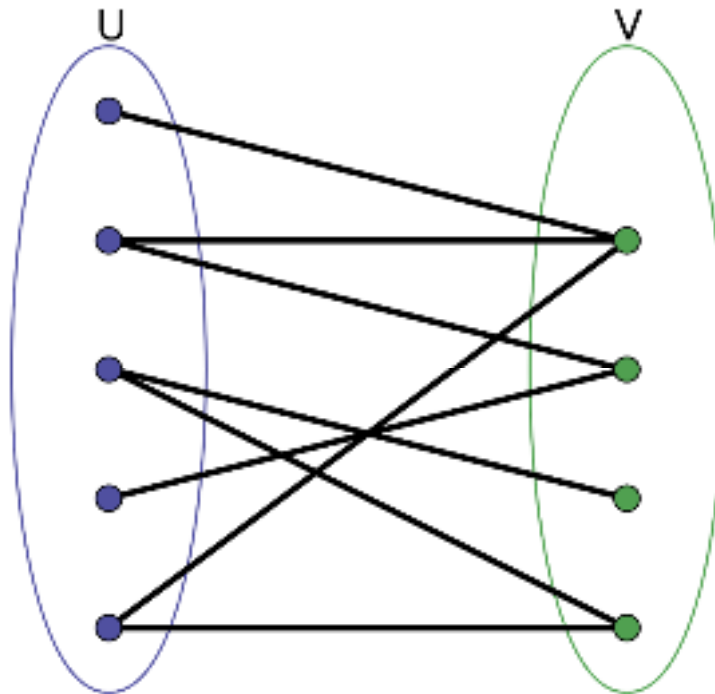
- Định nghĩa của các đỉnh liền kề không hoàn toàn giống nhau ở cả hai loại đồ thị.
- Trong digraph:
 - Nếu có một cạnh hướng từ đỉnh x đến đỉnh y , thì y là liền kề với x . (Nói cách khác, y là một con cháu của x và x là một tổ tiên của y .)
Và không nhất thiết phải tuân theo x liền kề với y .



Albuquerque tiếp giáp với San Francisco, nhưng San Francisco không giáp với Albuquerque.

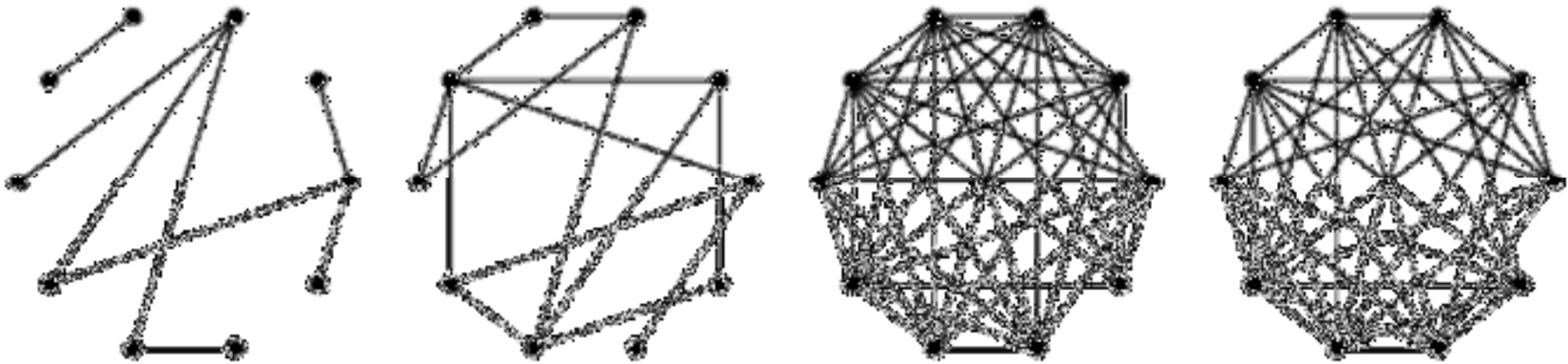
Đồ thị lưỡng cực

- **Đồ thị lưỡng cực** (bipartite graph) là đồ thị mà các đỉnh có thể được chia thành hai tập không giao nhau và tất cả các cạnh trong đồ thị kết nối chỉ các đỉnh trên các tập khác nhau.



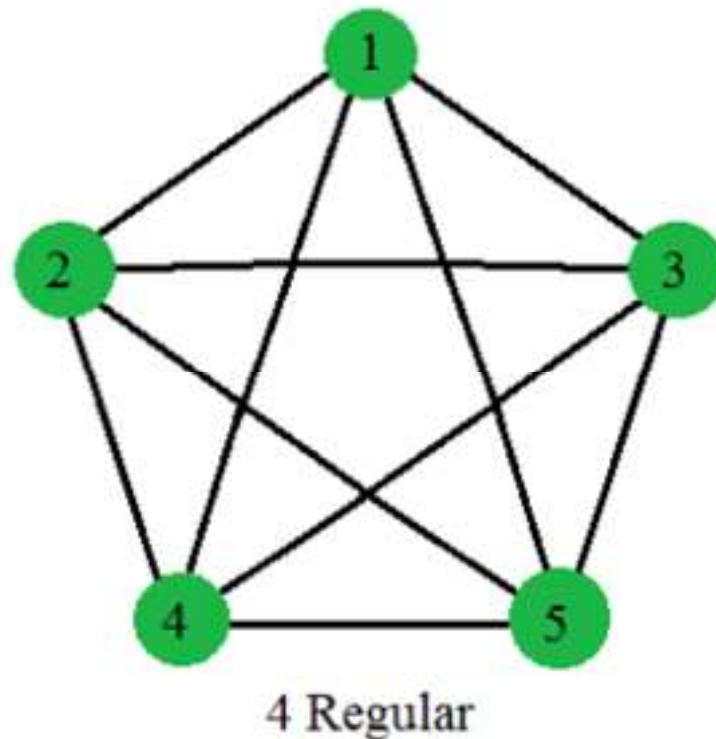
Đồ thị ngẫu nhiên

- **Đồ thị ngẫu nhiên** (random graph) là đồ thị mà các cạnh được phát sinh ngẫu nhiên.



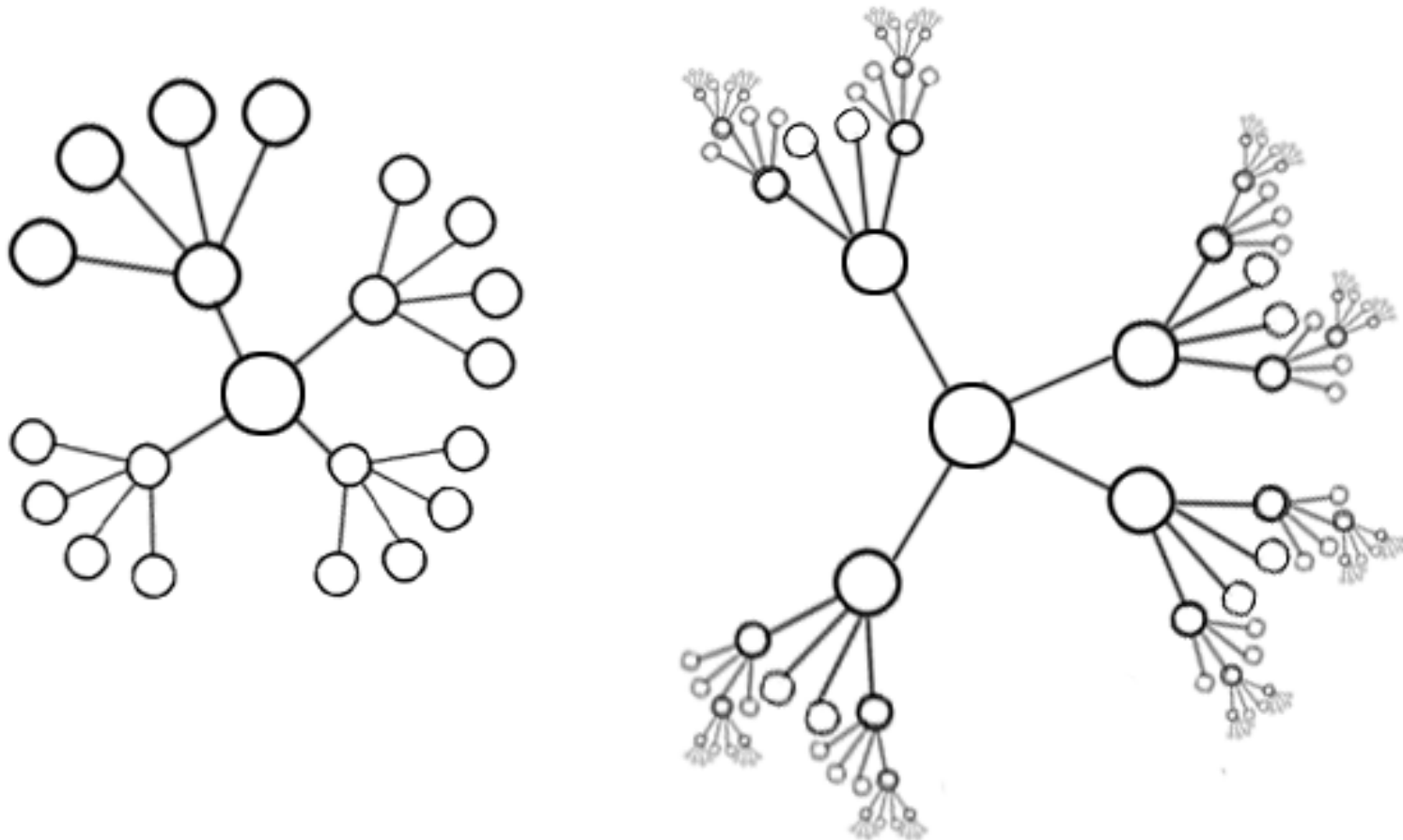
Đồ thị chuẩn tắc

- **Đồ thị chuẩn tắc** (regular graph) là đồ thị mà tất cả các đỉnh có cùng bậc.
- Từ đó có thể thấy tất cả đồ thị đủ đều là đồ thị chuẩn tắc



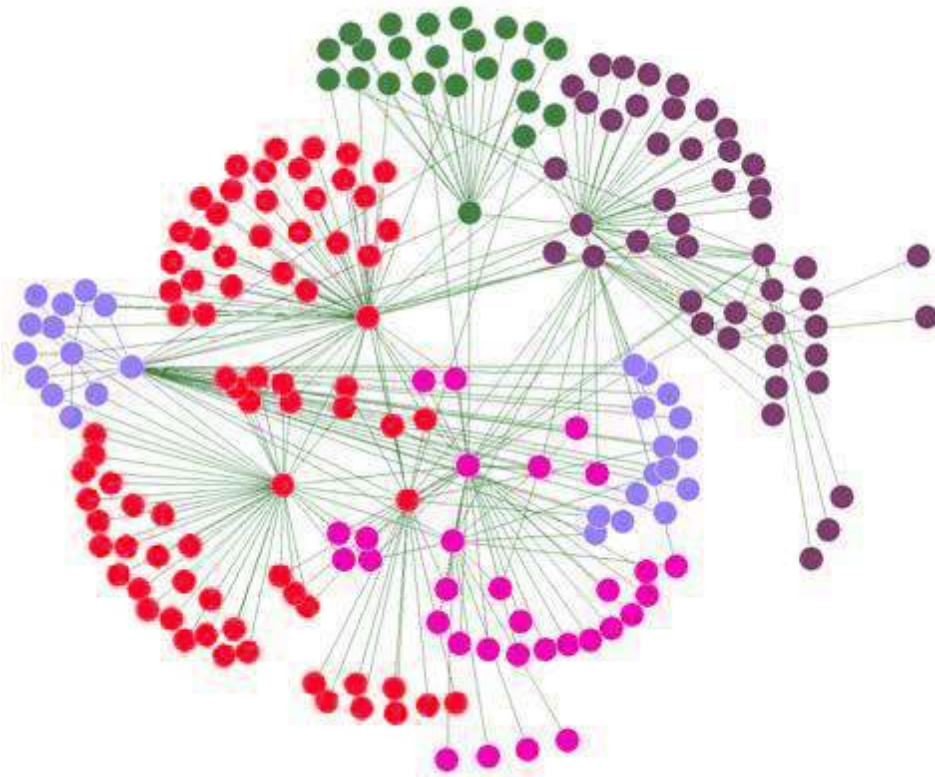
Đồ thị scale-free

- Một đồ thị được gọi là **scale-free** nếu đặc trưng của nó độc lập với kích thước của đồ thị (số đỉnh). Nghĩa là khi đồ thị phình ra, cấu trúc bên dưới vẫn tương tự.



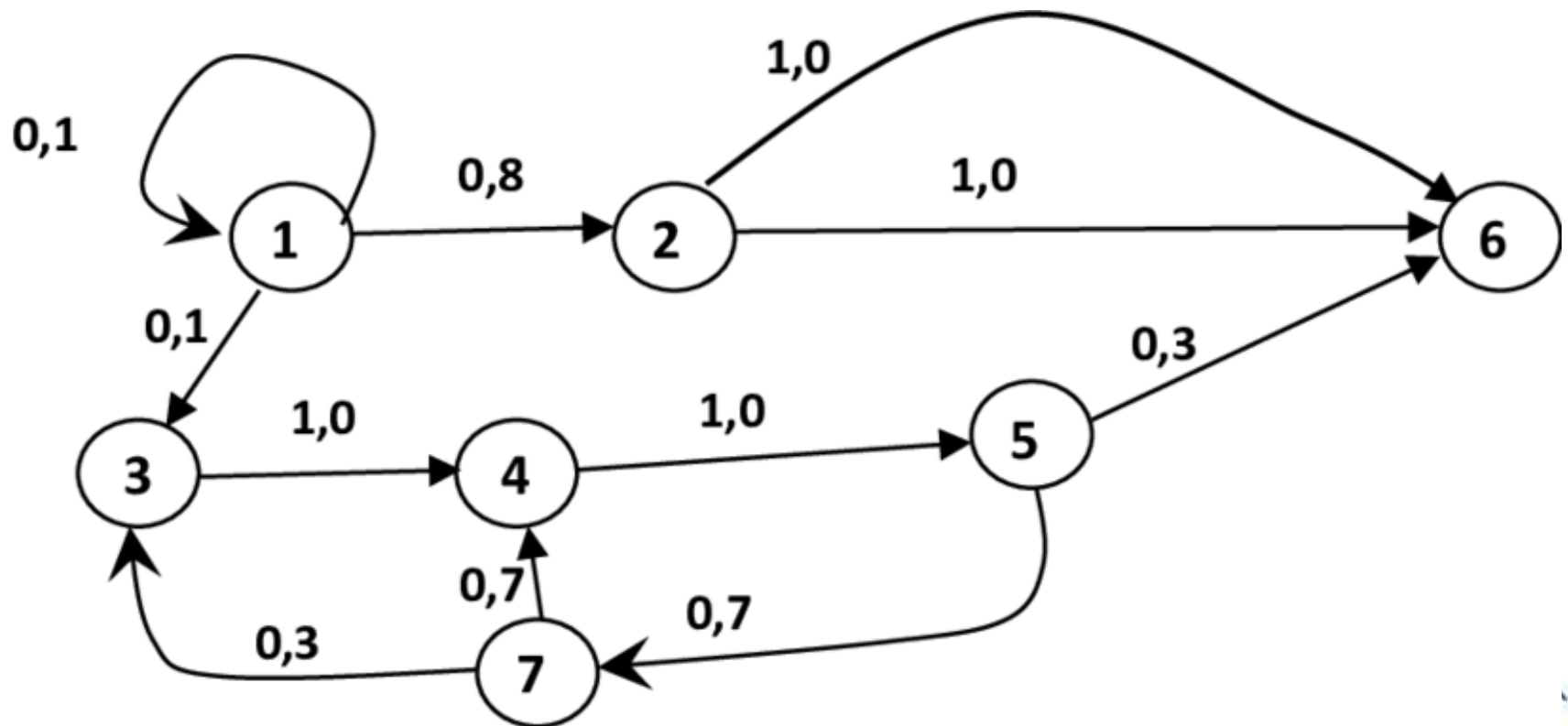
Đồ thị thế giới nhỏ

- **Đồ thị thế giới nhỏ** (small-world) là đồ thị mà đường đi trung bình giữa mọi đỉnh thì nhỏ. Nghĩa là một đỉnh có thể đến bất kỳ đỉnh nào qua chỉ vài cạnh.



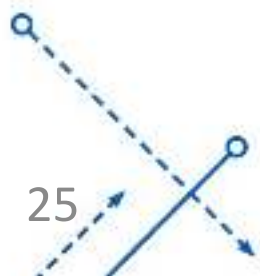
Đồ thị ngẫu nhiên

- **Đồ thị ngẫu nhiên** (stochastic graph) là đồ thị mà trọng số mỗi cạnh được biểu diễn với giá trị xác suất thể hiện khả năng di chuyển giữa hai đỉnh. Tổng trọng số các cạnh ở mỗi đỉnh bằng 1.



Nội dung

- Khái niệm và thuộc tính đồ thị
- Các loại đồ thị
- **Lưu trữ đồ thị**
- Một số thuật toán cơ bản trên đồ thị
 - Thuật toán duyệt đồ thị
 - Thuật toán sắp xếp topo
 - Thuật toán cây khung
 - Thuật toán đường đi ngắn nhất
- Lát cắt và luồng
- So khớp đồ thị



Lưu trữ đồ thị

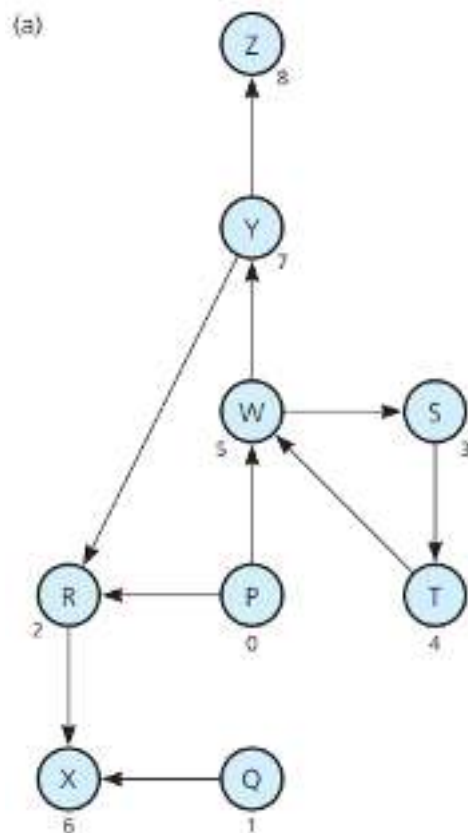
- Có nhiều cách khác nhau để lưu trữ đồ thị trong hệ thống máy tính.
- Cấu trúc dữ liệu được sử dụng phụ thuộc vào cả cấu trúc đồ thị và thuật toán được sử dụng để thao tác trên đồ thị.
- Hai cách triển khai phổ biến nhất của đồ thị là ma trận kề và danh sách kề.



Ma trận kề

- **Ma trận kề** cho một đồ thị có n đỉnh được đánh số $0, 1, \dots, n - 1$ là:

- Một ma trận mảng $n \times n$ sao cho ma trận $[i][j]$ là 1 nếu có một cạnh từ đỉnh i đến đỉnh j và 0 nếu không có.

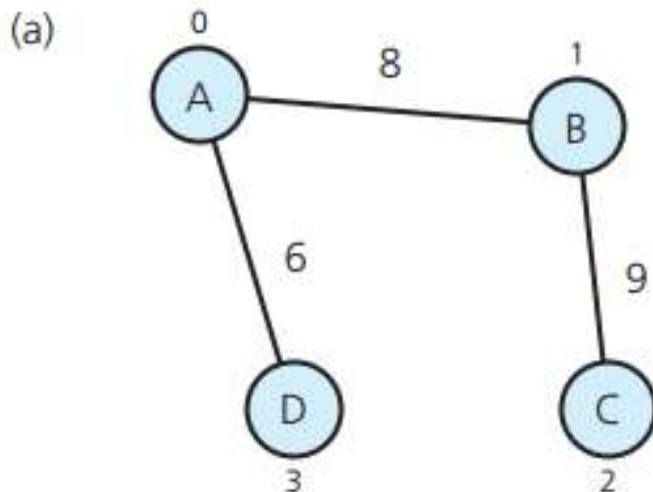


(b)

		0	1	2	3	4	5	6	7	8
		P	Q	R	S	T	W	X	Y	Z
0	P	0	0	1	0	0	1	0	0	0
1	Q	0	0	0	0	0	0	1	0	0
2	R	0	0	0	0	0	0	1	0	0
3	S	0	0	0	0	1	0	0	0	0
4	T	0	0	0	0	0	1	0	0	0
5	W	0	0	0	1	0	0	0	1	0
6	X	0	0	0	0	0	0	0	0	0
7	Y	0	0	1	0	0	0	0	0	1
8	Z	0	0	0	0	0	0	0	0	0

Ma trận kề

- Khi đồ thị có trọng số:
 - Đặt ma trận $[i][j]$ là trọng số gắn nhãn cạnh từ đỉnh i đến đỉnh j , thay vì đơn giản là 1
 - Cho ma trận $[i][j]$ bằng ∞ thay vì 0 khi không có cạnh nào từ đỉnh i đến đỉnh j .



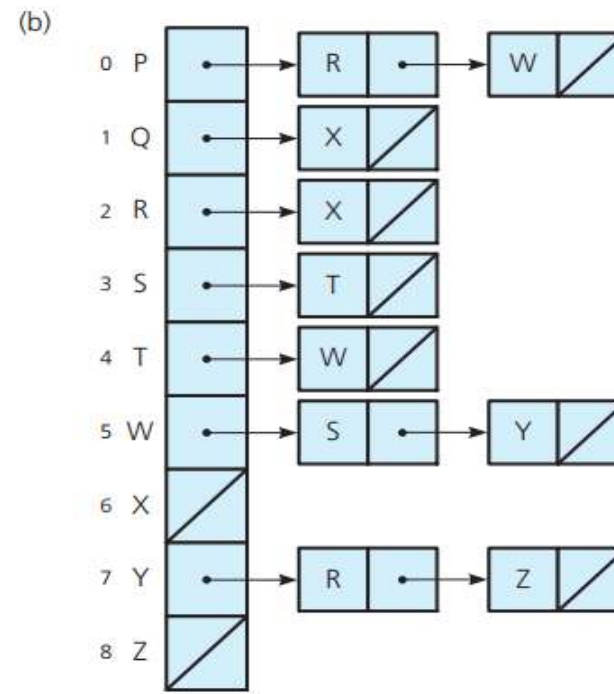
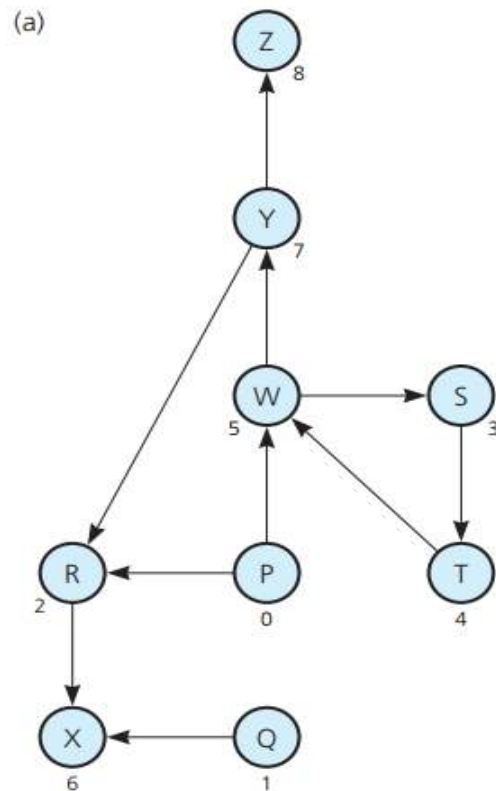
(b)

		0	1	2	3
		A	B	C	D
0	A	∞	8	∞	6
1	B	8	∞	9	∞
2	C	∞	9	∞	∞
3	D	6	∞	∞	∞

Lưu ý rằng ma trận kề cho một đồ thị vô hướng là đối xứng; nghĩa là ma trận $[i][j]$ bằng ma trận $[j][i]$.

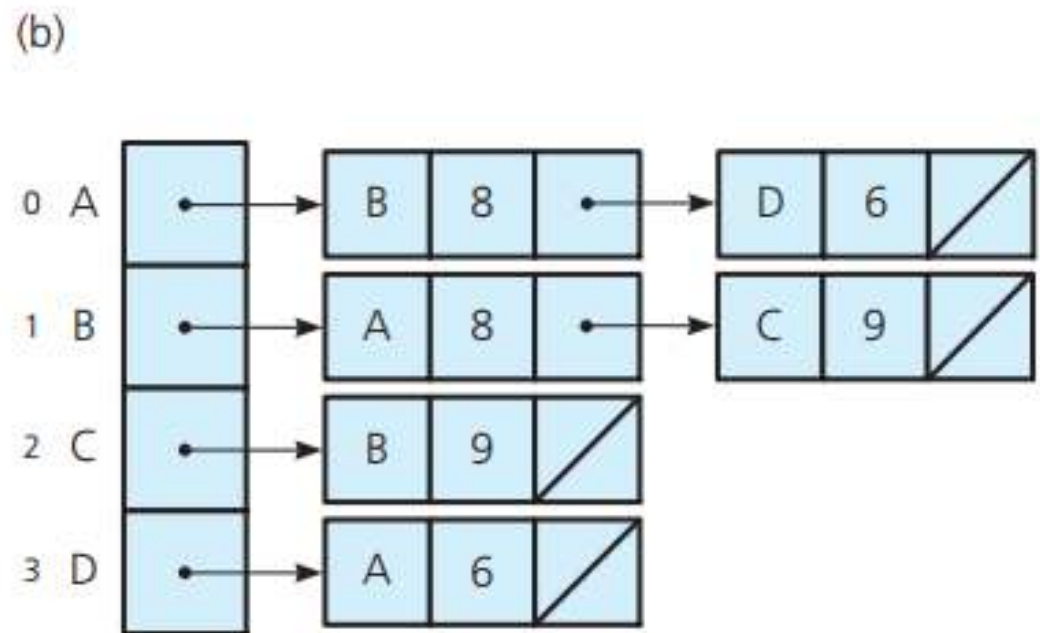
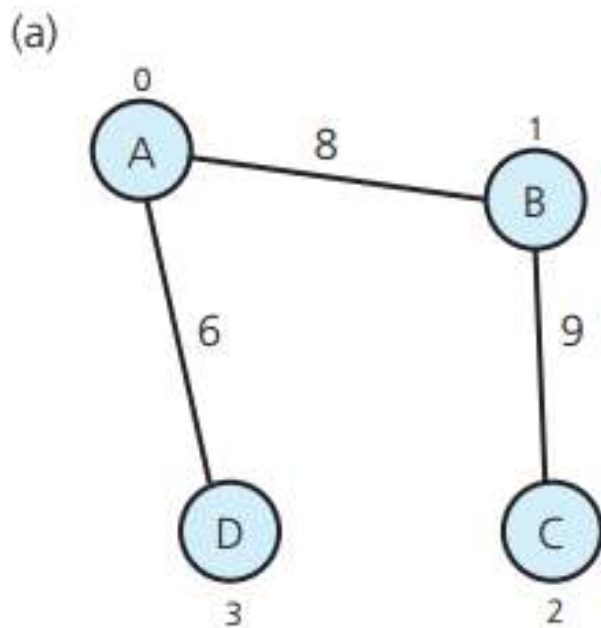
Danh sách kề

- **Danh sách kề** cho một đồ thị có n đỉnh được đánh số $0, 1, \dots, n - 1$ bao gồm n chuỗi liên kết.
 - Chuỗi liên kết thứ i có một nút cho đỉnh j nếu và chỉ khi đồ thị chứa một cạnh từ đỉnh i đến đỉnh j . Nút này có thể chứa giá trị của đỉnh j , nếu có.



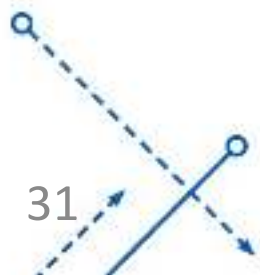
Danh sách kề

Danh sách kề cho một đồ thị vô hướng xử lý mỗi cạnh giống như nó là hai cạnh có hướng ngược nhau.



Cách nào tốt hơn?

- Cách triển khai đồ thị nào trong hai cách này — ma trận kề hoặc danh sách kề — tốt hơn?
 - Câu trả lời phụ thuộc vào cách ứng dụng cụ thể của bạn sử dụng đồ thị.
- Ví dụ:
 - Xác định xem có một cạnh từ đỉnh i đến đỉnh j -> ma trận kề
 - Tìm tất cả các đỉnh kề với một đỉnh i đã cho -> danh sách kề



Cách nào tốt hơn?

- Xem xét các yêu cầu về không gian của hai triển khai.
 - Nhìn bề ngoài, ma trận kề yêu cầu ít bộ nhớ hơn danh sách kề, bởi vì mỗi entry trong ma trận đơn giản là một số nguyên, trong khi mỗi nút danh sách chứa cả giá trị để xác định đỉnh và con trỏ.
 - Tuy nhiên, ma trận kề luôn có n^2 entry, trong khi số nút trong danh sách kề bằng số cạnh trong đồ thị có hướng hoặc gấp đôi số đó đối với đồ thị vô hướng. Mặc dù danh sách kề cũng có n con trỏ đầu, nó thường yêu cầu lưu trữ ít hơn ma trận kề



Nội dung

- Khái niệm và thuộc tính đồ thị
- Các loại đồ thị
- Lưu trữ đồ thị
- **Một số thuật toán cơ bản trên đồ thị**
 - Thuật toán duyệt đồ thị
 - Thuật toán sắp xếp topo
 - Thuật toán cây khung
 - Thuật toán đường đi ngắn nhất
- Lát cắt và luồng
- So khớp đồ thị



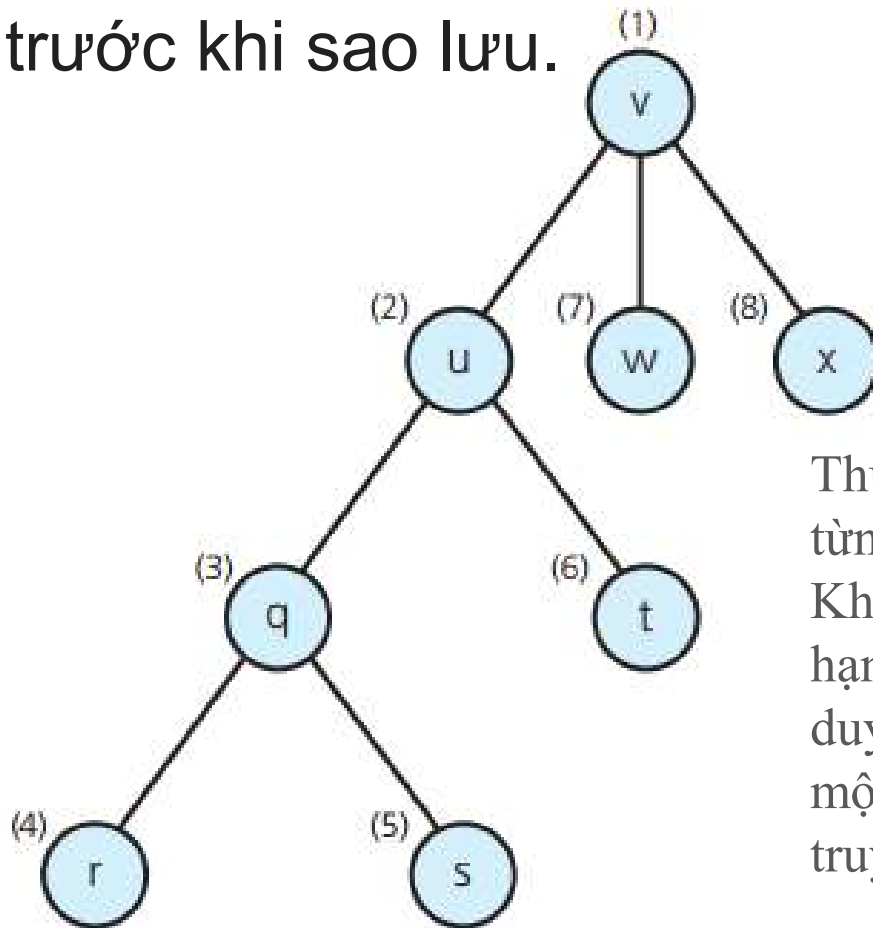
Duyệt đồ thị

- Một phép duyệt đồ thị bắt đầu từ đỉnh v sẽ thăm tất cả các đỉnh w mà có đường đi giữa v và w .
 - Nếu một đồ thị không được kết nối, một đường đi ngang qua đồ thị bắt đầu từ đỉnh v sẽ chỉ truy cập vào một tập hợp con của các đỉnh của đồ thị. Tập hợp con này được gọi là thành phần kết nối chứa v .
 - Nếu một đồ thị có chứa một chu trình, thì một thuật toán duyệt đồ thị có thể lặp vô hạn. Để tránh trường hợp không may đó xảy ra, thuật toán phải đánh dấu từng đỉnh trong một lần truy cập và không bao giờ được truy cập một đỉnh nhiều hơn một lần.



Depth-First Search

- Từ một đỉnh v nhất định, chiến lược tìm kiếm theo độ sâu (DFS) của phương pháp duyệt đồ thị sẽ tiến hành dọc theo đường đi từ v càng sâu vào đồ thị càng tốt trước khi sao lưu.

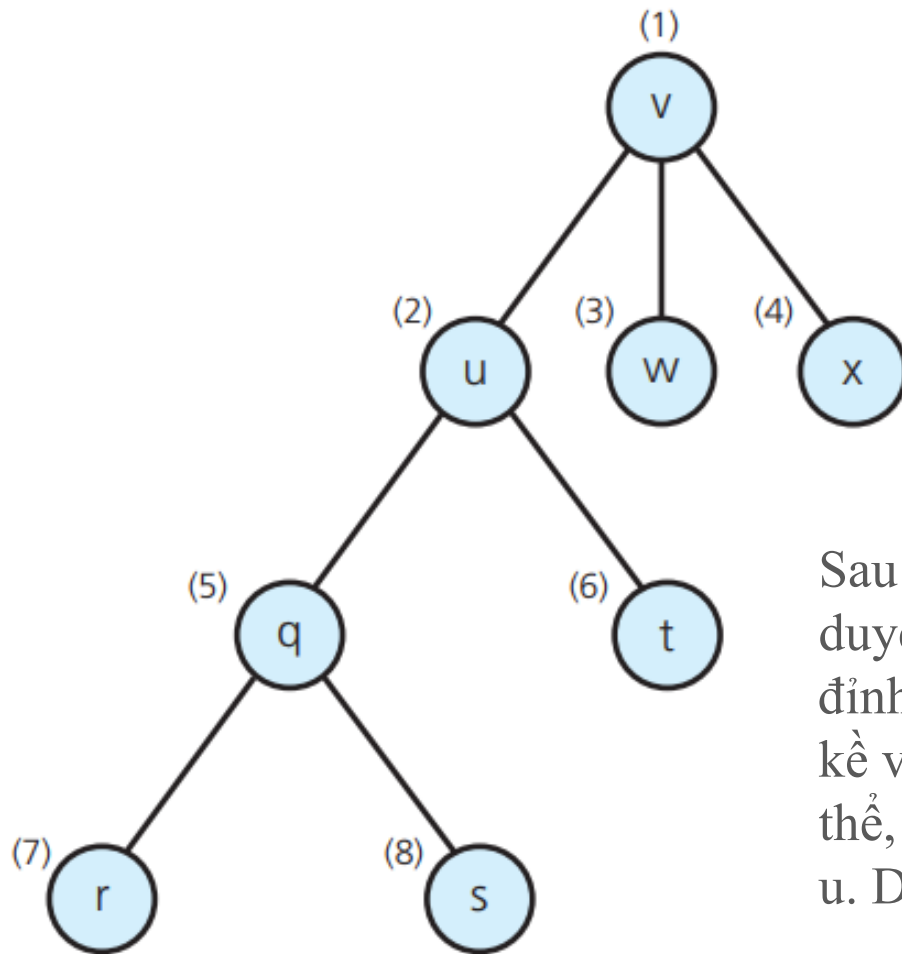


Thuật toán duyệt DFS đánh dấu và sau đó thăm từng đỉnh v , u , q và r .

Khi đường truyền đạt đến một đỉnh — chẳng hạn như r — không có đỉnh liền kề chưa được duyệt, nó sẽ sao lưu và truy cập, nếu có thể, một đỉnh liền kề không được duyệt. Do đó, việc truyền tải trở lại q và sau đó truy cập s .

Breadth-first search

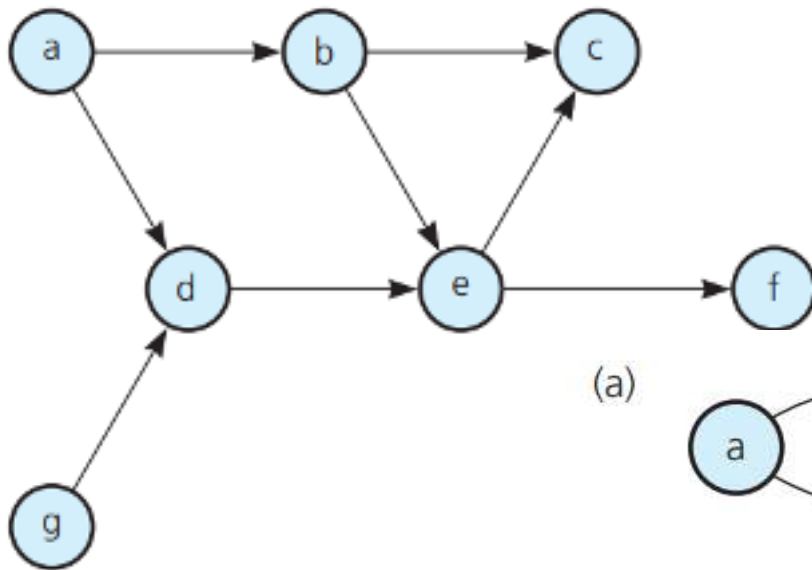
- **Breadth-first search** (BFS): chiến lược duyệt đồ thị truy cập mọi đỉnh kề với v mà nó có thể trước khi truy cập bất kỳ đỉnh nào khác.



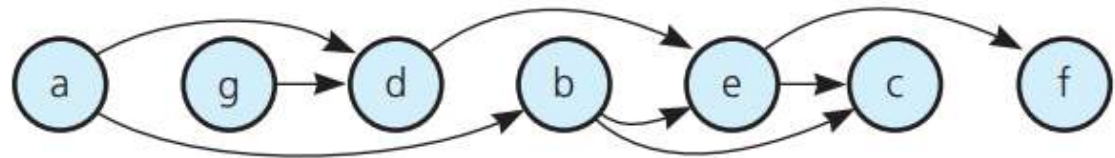
Sau khi đánh dấu và truy cập v, thuật toán duyệt BFS đánh dấu và sau đó truy cập từng đỉnh u, w và x. Vì không có đỉnh nào khác liên kết với v, nên thuật toán BFS sẽ truy cập, nếu có thể, tất cả các đỉnh không được sắp xếp kề với u. Do đó, các duyệt qua q và t.

Sắp xếp topo

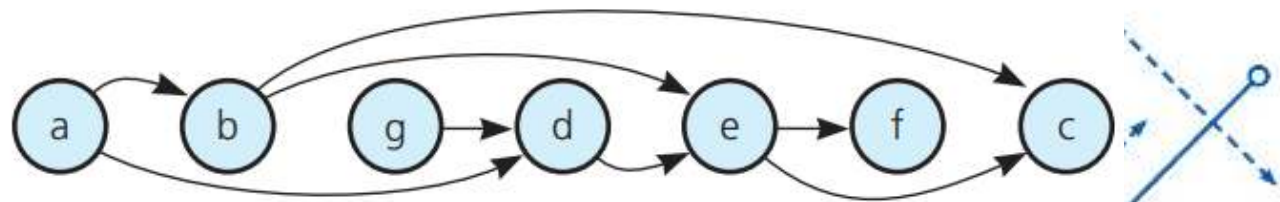
- Một đồ thị có hướng không có chu trình có thứ tự tự nhiên. Nó là một trật tự tuyến tính, được gọi là một trật tự tôpô.
- Việc sắp xếp các đỉnh thành một thứ tự tôpô được gọi là sắp xếp tôpô.



(a)



(b)

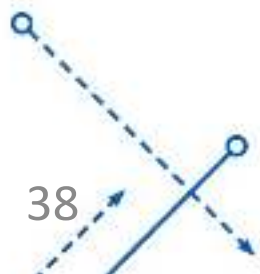


Nếu bạn sắp xếp các đỉnh của một đồ thị có hướng một cách tuyến tính và theo một thứ tự tôpô, tất cả các cạnh sẽ hướng về một hướng.

Thuật toán sắp xếp topo

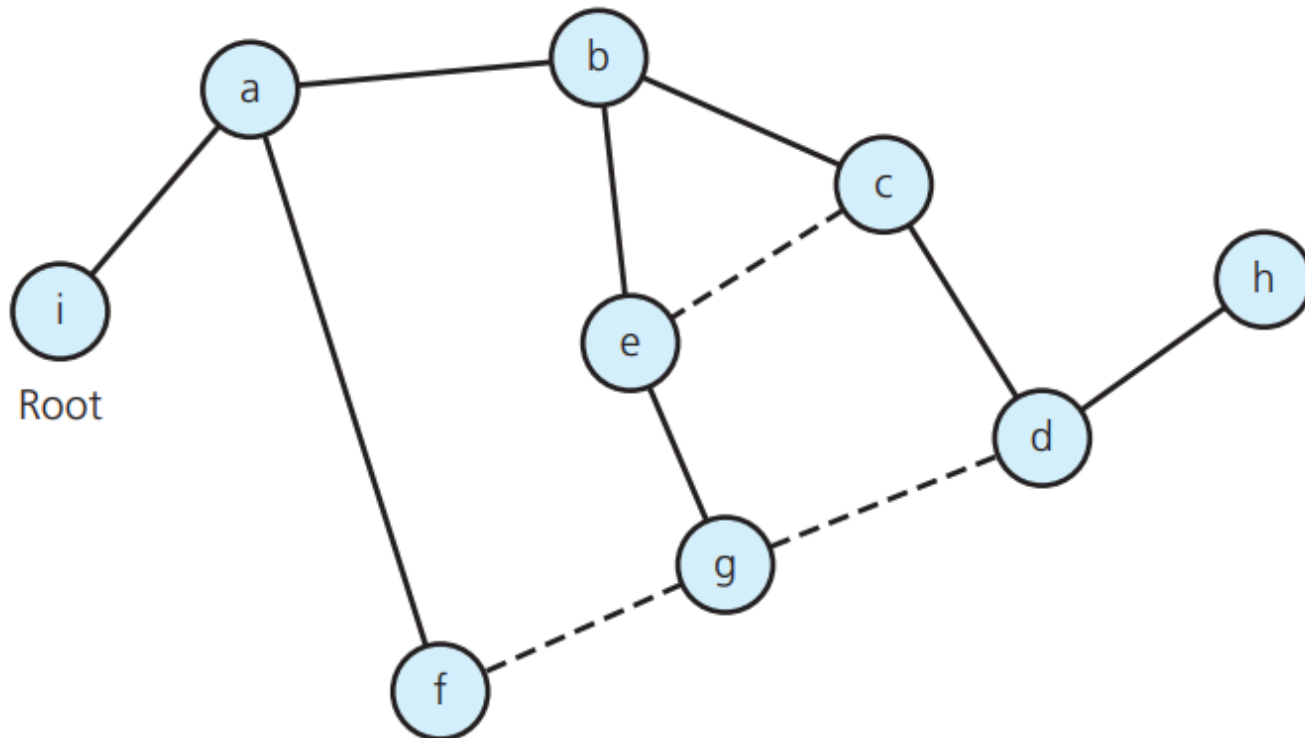
- Bước 1: Tìm một đỉnh không có đỉnh kế tiếp.
- Bước 2: loại bỏ đỉnh này và tất cả các cạnh dẫn đến nó trên đồ thị và thêm nó vào đầu danh sách các đỉnh.
- Bước 3: lặp lại bước 2 cho đến khi đồ thị trống. Danh sách các đỉnh sẽ theo thứ tự tôpô.

```
// Arranges the vertices in graph theGraph into a
// topological order and places them in list aList.
topSort1(theGraph: Graph, aList: List)
  n = number of vertices in theGraph
  for (step = 1 through n) {
    Select a vertex v that has no successors
    aList.insert(1, v)
    Remove from theGraph vertex v and its edges
  }
```



Cây khung

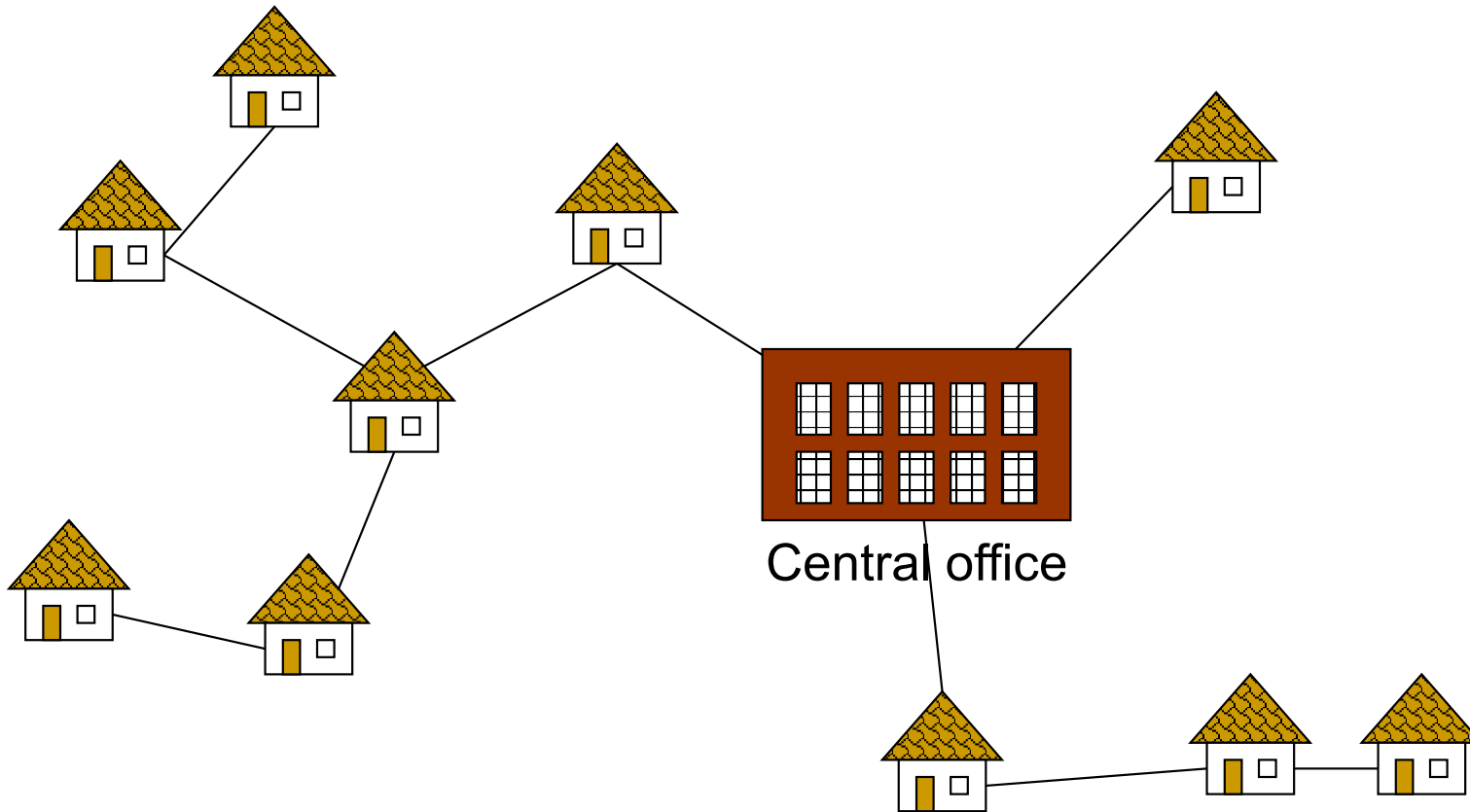
- **Cây khung** (spanning tree) của đồ thị vô hướng được kết nối G là một đồ thị con của G chứa tất cả các đỉnh của G và đủ các cạnh của nó để tạo thành một cây.



- 



Cây khung tối thiểu

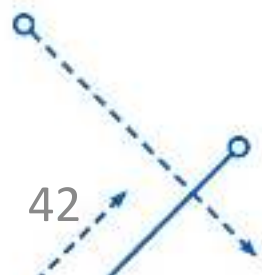
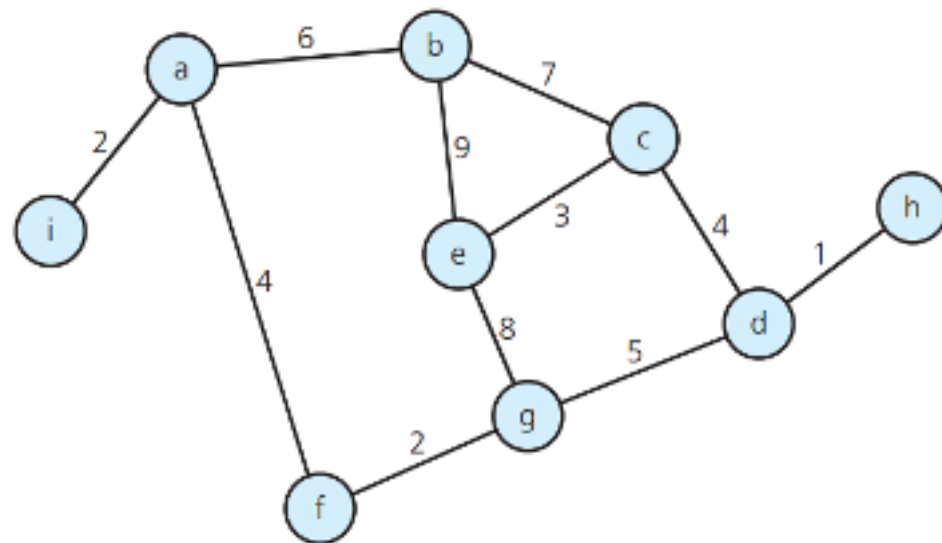


Giảm thiểu tổng chiều dài dây kết nối **TẤT CẢ** khách hàng

Cây khung tối thiểu

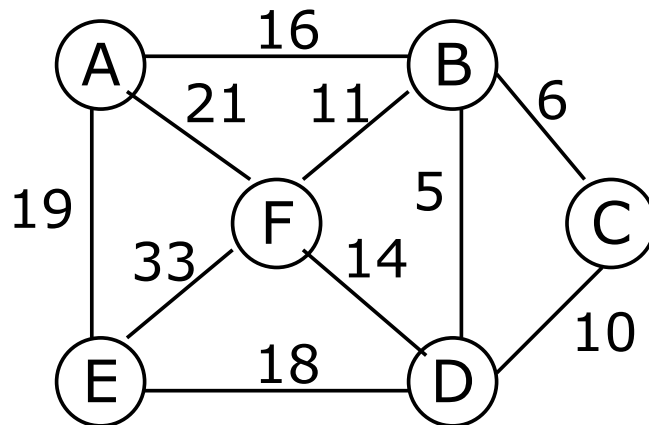
Trong hình sau:

- Các đỉnh trong đồ thị đại diện cho n thành phố.
- Một cạnh giữa hai đỉnh chỉ ra rằng nó được đặt một đường dây điện thoại giữa các thành phố mà các đỉnh đại diện
- Trọng lượng của mỗi cạnh thể hiện chi phí lắp đặt của đường dây điện thoại.

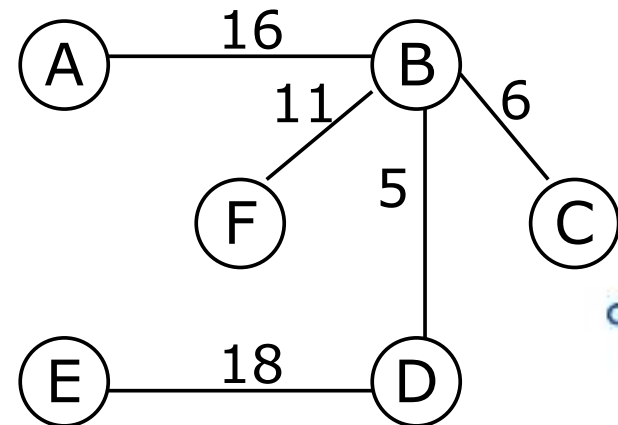


Cây khung tối thiểu

- Chúng ta biết rằng:
 - Có thể có nhiều hơn một cây khung
 - Giá thành của các cây khác nhau có thể khác nhau
- Giải quyết vấn đề bằng cách chọn cây khung với chi phí nhỏ nhất (tổng trọng số các cạnh (chi phí) là nhỏ nhất) -> cây khung tối thiểu (MST).
- Mặc dù có thể có một số cây khung tối thiểu cho một đồ thị cụ thể, nhưng chi phí của chúng là bằng nhau.



Đồ thị liên thông không liên thông



Cây khung nhỏ nhất

Đường đi ngắn nhất

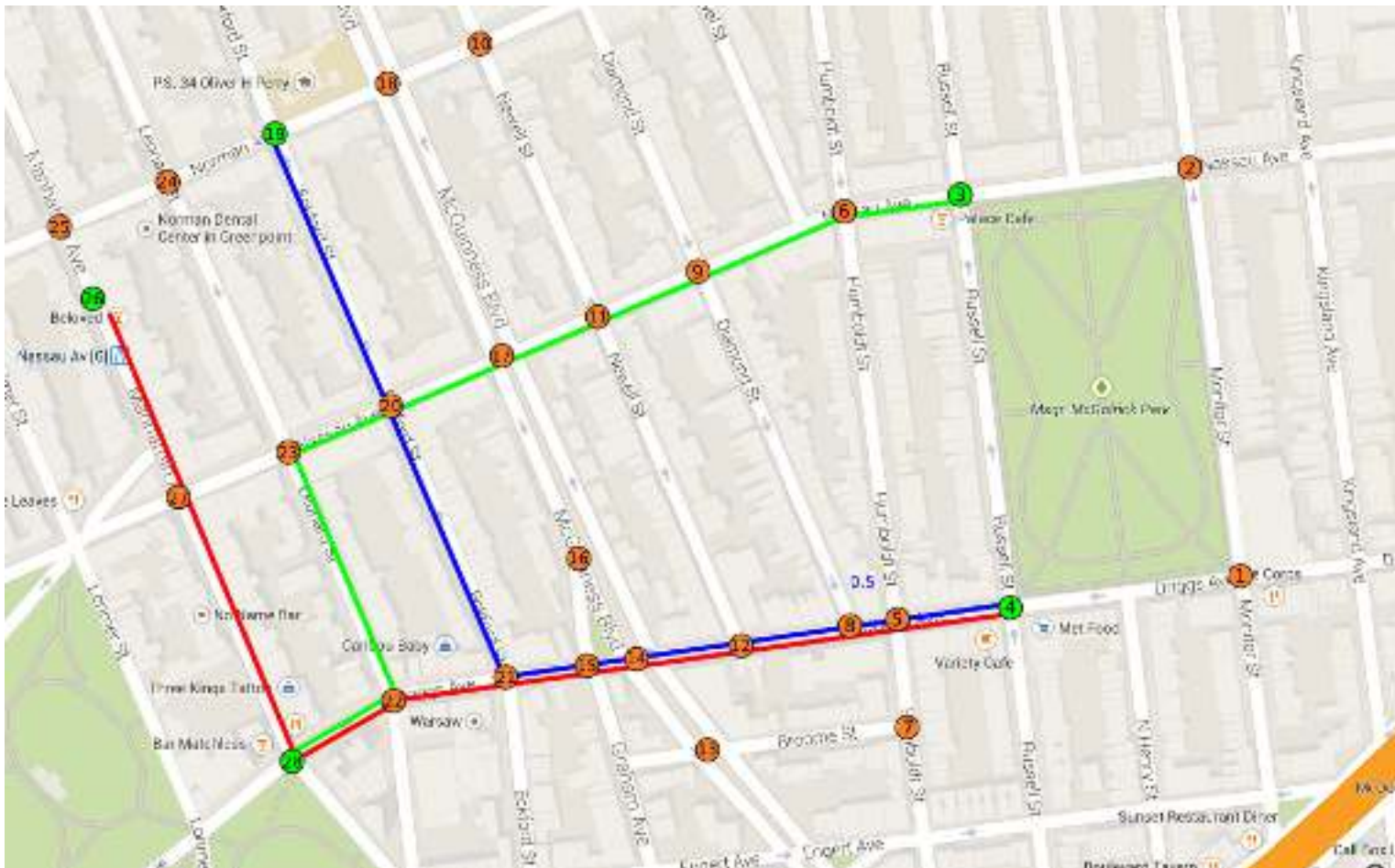
- Tìm con đường ngắn nhất giữa hai đỉnh cụ thể trong bản đồ thành phố.



Small section of Williamsburg in Brooklyn, NY

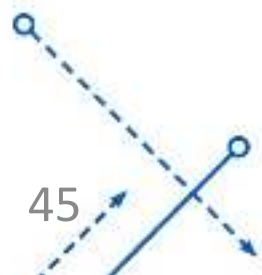
Đường đi ngắn nhất

- Đường đi ngắn nhất giữa hai đỉnh trong đồ thị là đường đi có tổng trọng số các cạnh nhỏ nhất của nó.

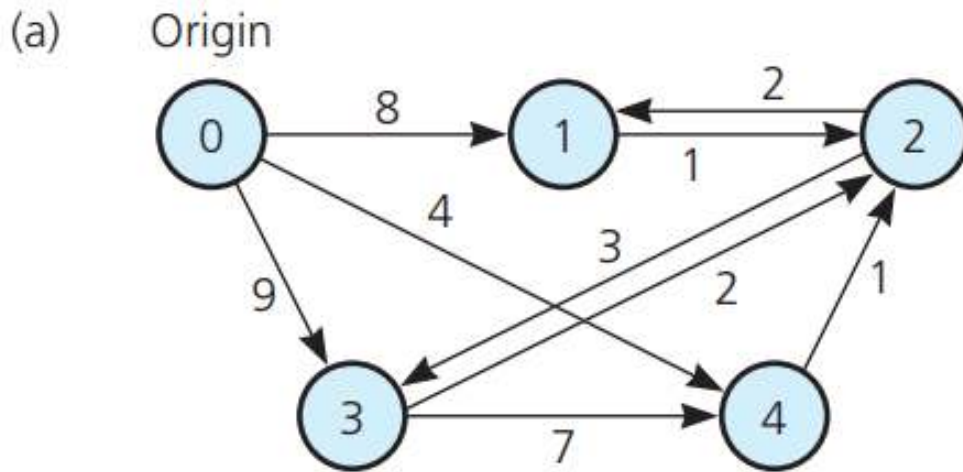


Test 1/Red: 4 -> 26
Test 2/Blue: 4 -> 19
Test 3/Green: 3 -> 28

Result of Dijkstra's algorithm directions



Đường đi ngắn nhất



(b)

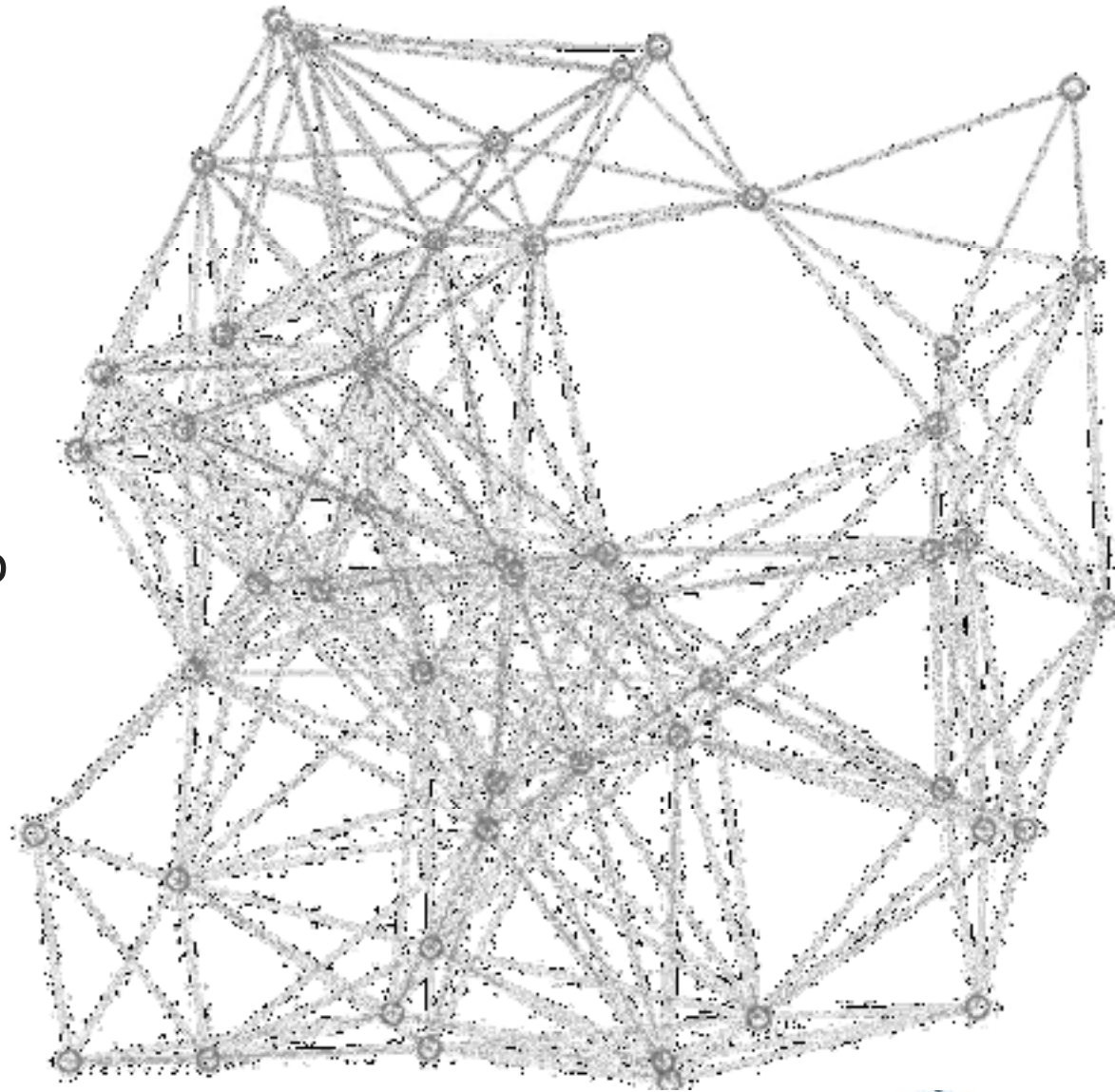
	0	1	2	3	4
0	∞	8	∞	9	4
1	∞	∞	1	∞	∞
2	∞	2	∞	3	∞
3	∞	∞	2	∞	7
4	∞	∞	1	∞	∞

Đường đi ngắn nhất từ đỉnh 0 đến đỉnh 1 trong đồ thị ở hình trên không phải là cạnh giữa 0 và 1 — chi phí của nó là 8 — mà là đường đi từ 0 đến 4 đến 2 đến 1, với chi phí là 7.



Thuật toán Dijkstra

- Thuật toán Dijkstra xác định các đường đi ngắn nhất giữa một điểm gốc đã cho và tất cả các đỉnh khác.
- Thuật toán sử dụng một tập hợp đỉnh Tập các đỉnh đã chọn và một trọng số mảng, trong đó trọng số $[v]$ là trọng số của đường đi ngắn nhất từ đỉnh 0 đến đỉnh v .



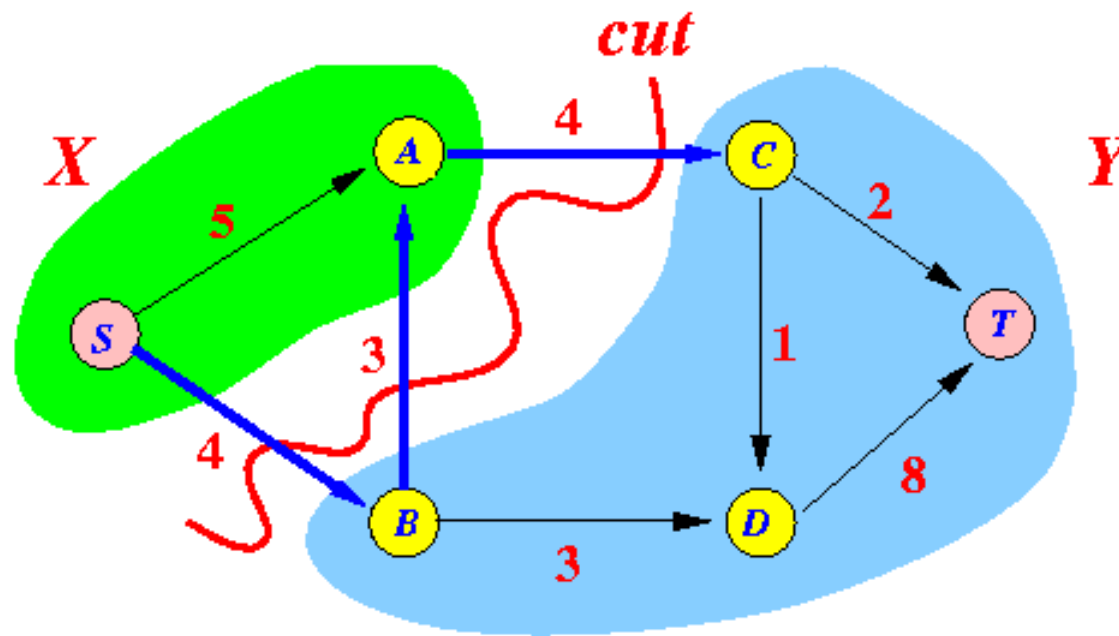
Nội dung

- Khái niệm và thuộc tính đồ thị
- Các loại đồ thị
- Lưu trữ đồ thị
- Một số thuật toán cơ bản trên đồ thị
 - Thuật toán duyệt đồ thị
 - Thuật toán sắp xếp topo
 - Thuật toán cây khung
 - Thuật toán đường đi ngắn nhất
- **Lát cắt và luồng**
- So khớp đồ thị



Lát cắt

- **Lát cắt** (cut) là một cách phân chia tập hợp các đỉnh của đồ thị thành 2 tập con không giao nhau.
- **Tập cắt** (cut-set) là tập hợp các cạnh có mỗi đầu nằm trong tập con sau khi cắt.

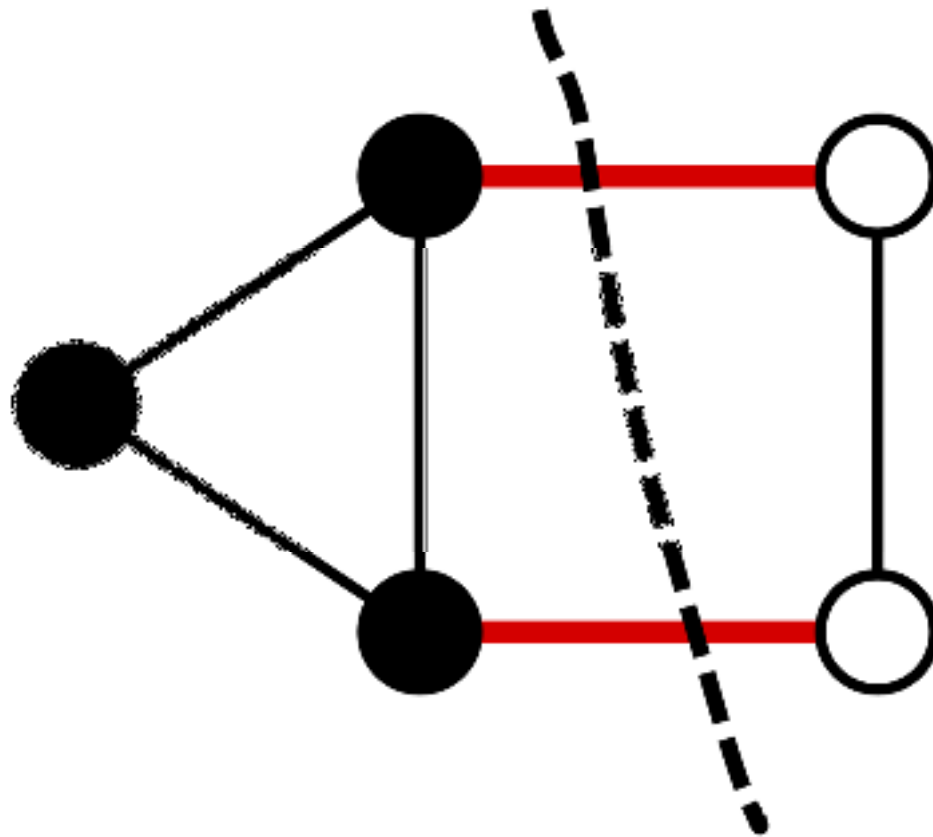


Cut-set = $\{ (S,B), (B,A), (A,C) \}$



Lát cắt nhỏ nhất

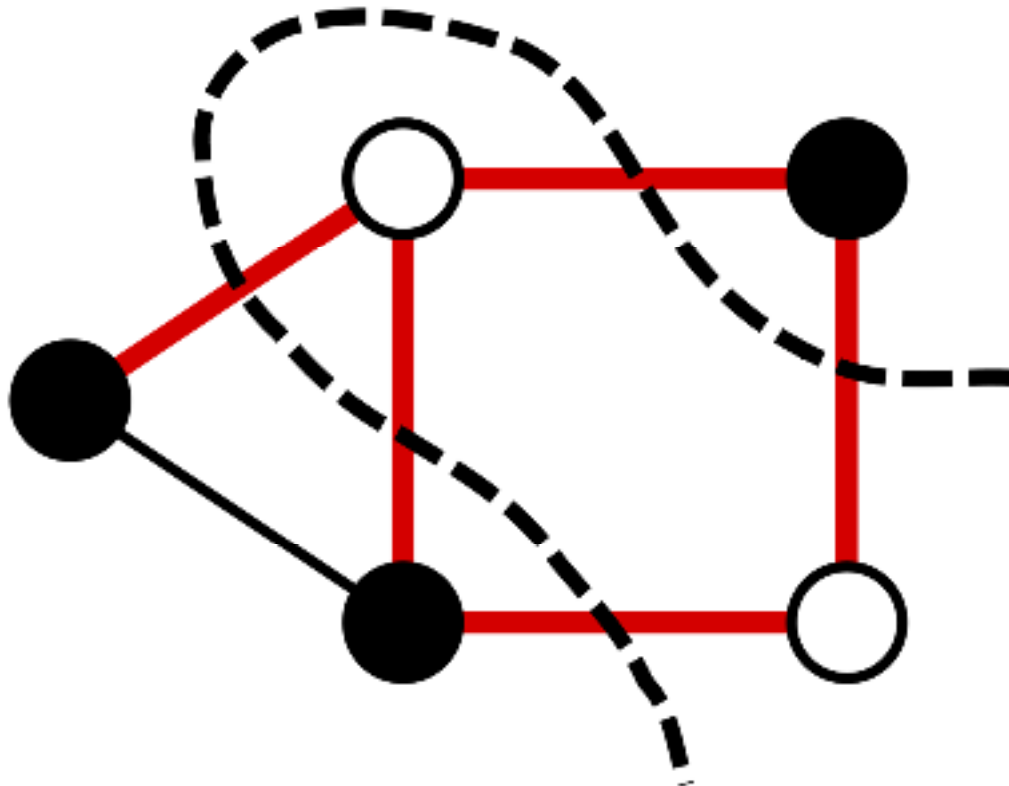
- Một lát cắt là nhỏ nhất khi kích thước/trọng lượng của nó không lớn hơn bất kỳ lát cắt nào khác.



Kích thước lát cắt này là 2

Lát cắt lớn nhất

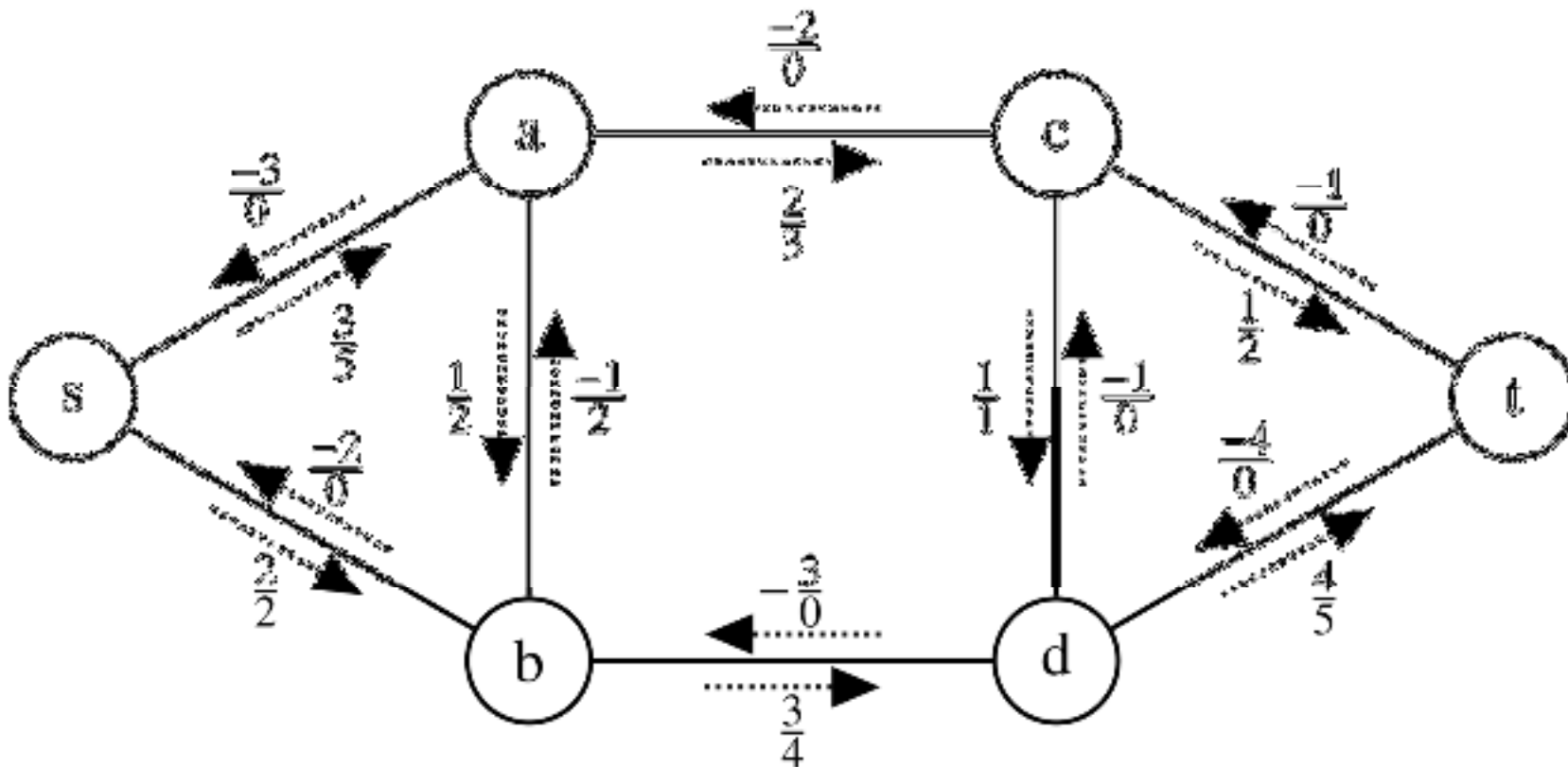
- Một lát cắt là lớn nhất khi kích thước/trọng lượng của nó không nhỏ hơn bất kỳ lát cắt nào khác.



Kích thước lát cắt này là 4

Luồng

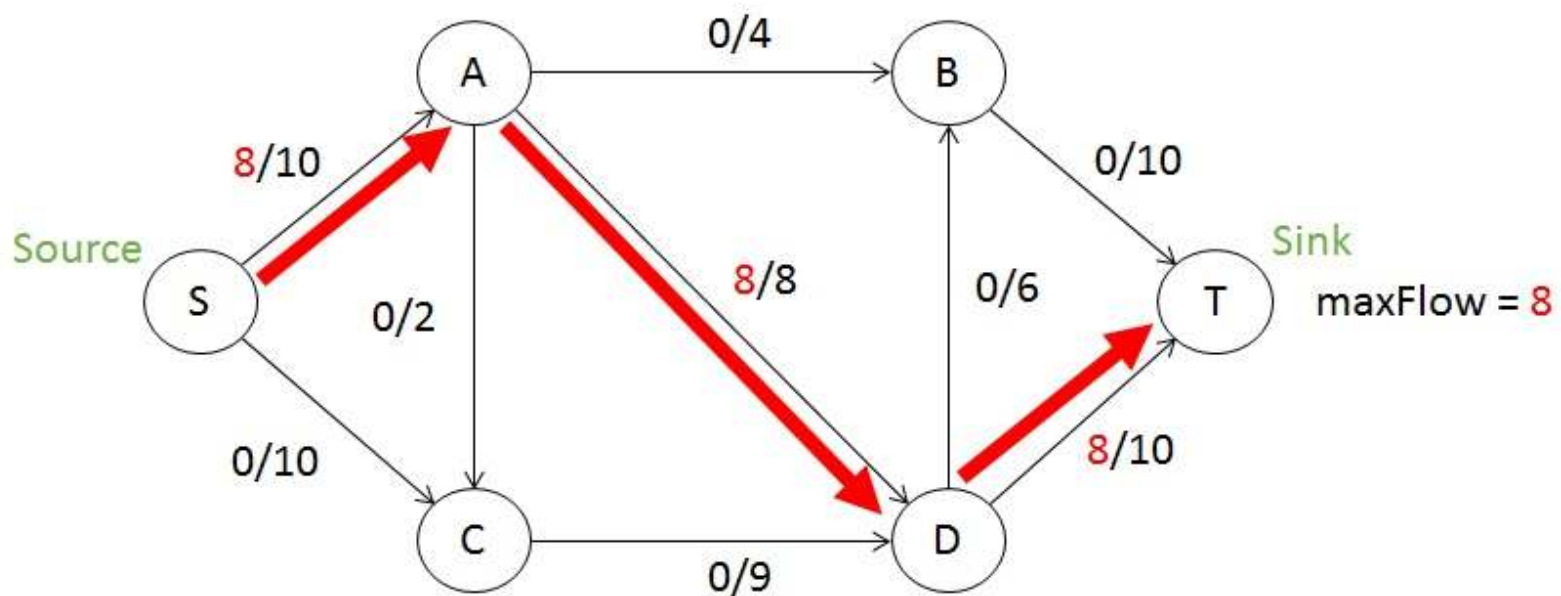
- **Luồng** (flow) thể hiện khả năng tải của một cạnh trong đồ thị.



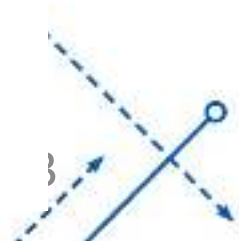
Luồng từ s tới t với tổng khối lượng là 5

Luồng cực đại

- **Luồng cực đại** là luồng mà có thể đạt được một tỉ lệ chuyển tải cực đại.
- Giá trị cực đại của một luồng từ s tới t bằng với lát cắt tối thiểu giữa s và t.

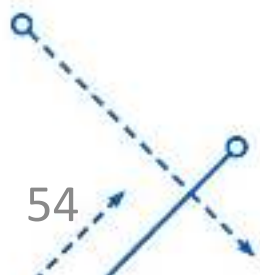


$$S \rightarrow A \rightarrow D \rightarrow T = 8$$



Nội dung

- Khái niệm và thuộc tính đồ thị
- Các loại đồ thị
- Lưu trữ đồ thị
- Một số thuật toán cơ bản trên đồ thị
 - Thuật toán duyệt đồ thị
 - Thuật toán sắp xếp topo
 - Thuật toán cây khung
 - Thuật toán đường đi ngắn nhất
- Lát cắt và luồng
- **So khớp đồ thị**

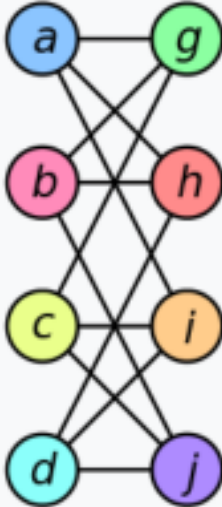
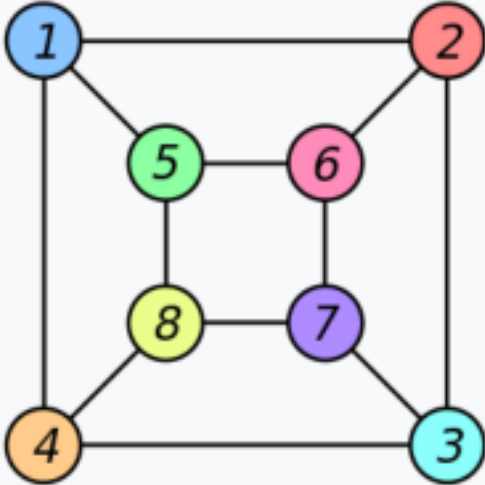


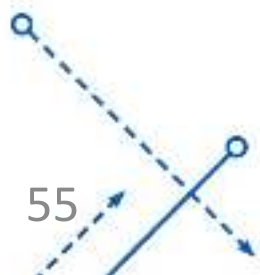
So khớp đồ thị - đẳng cấu

- Hai đồ thị G và H được gọi là **đẳng cấu** (isomorphic) nếu tồn tại một phép song ánh:

$$f: V(G) \rightarrow V(H)$$

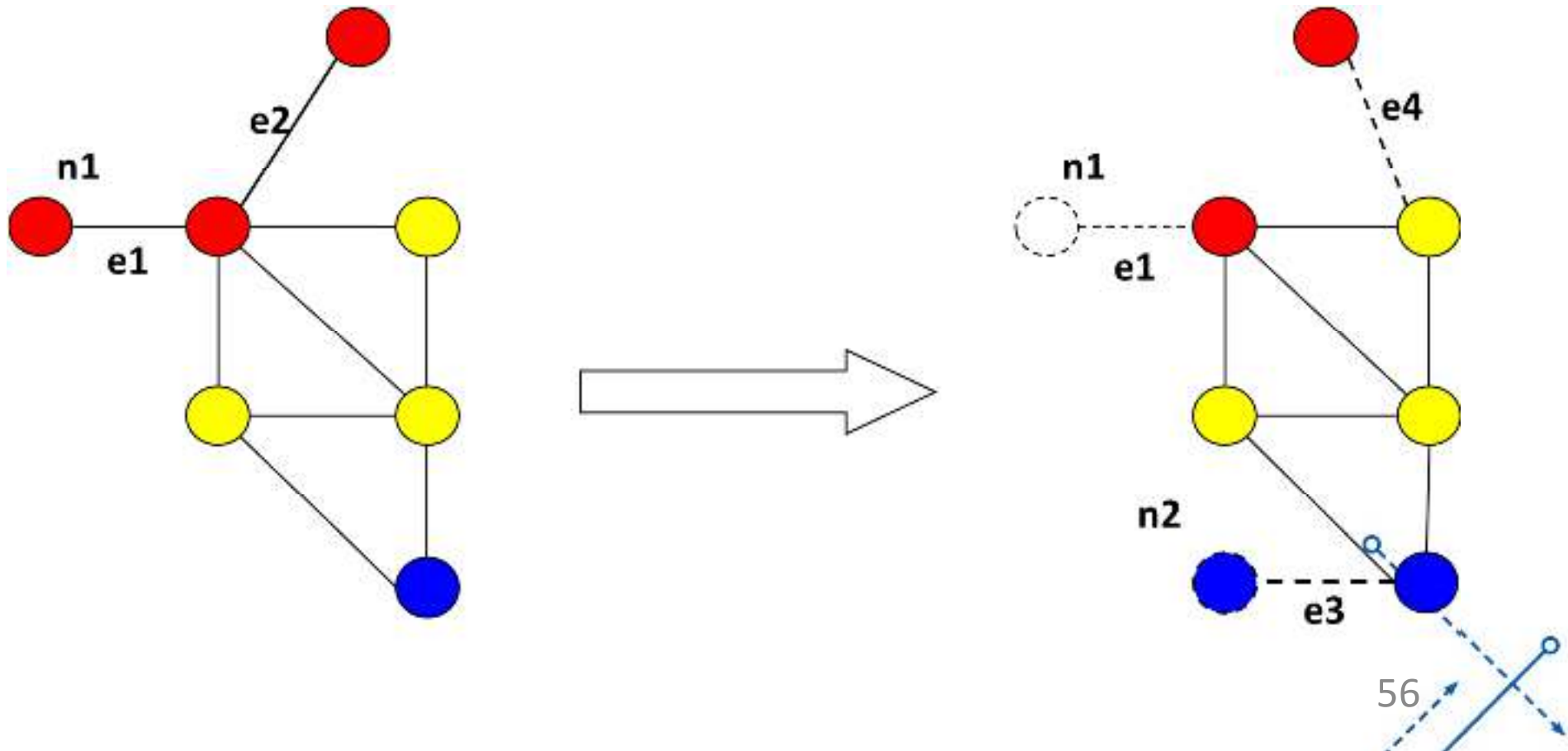
và nếu có cạnh nối giữa hai đỉnh u, v trong G thì cũng sẽ có cạnh nối giữa hai đỉnh $f(u)$ và $f(v)$ trong H.

Graph G	Graph H	An isomorphism between G and H
		$\begin{aligned} f(a) &= 1 \\ f(b) &= 6 \\ f(c) &= 8 \\ f(d) &= 3 \\ f(g) &= 5 \\ f(h) &= 2 \\ f(i) &= 4 \\ f(j) &= 7 \end{aligned}$



So khớp đồ thị - khoảng cách điều chỉnh

- Khoảng cách điều chỉnh đồ thị** (graph edit distance) là số thay đổi cần thiết để chuyển một đồ thị thành đồ thị còn lại.



Tài liệu tham khảo

- Mihalcea, Rada, and Dragomir Radev. *Graph-based natural language processing and information retrieval*. Cambridge university press, 2011.
- Frank M. Carrano, Data Abstraction and Problem Solving with C++: Walls and Mirrors, Frank M. Carrano, 6th Edition
- Minimum Spanning Tree in Graph - Week 13. 2 Problem: Laying Telephone Wire Central office.

