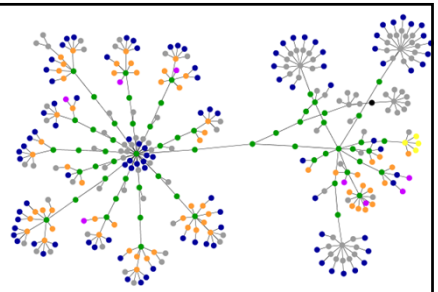
Chương 7
Đồ thị

Hiepnd@soict.hut.edu.vn

Nội dung

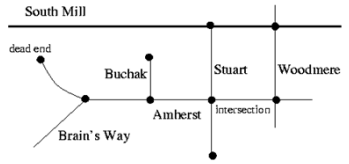
- Các khái niệm cơ bản
- Biểu diễn đồ thị
- Thuật toán duyệt đồ thị và ứng dụng
- Một số bài toán trên đồ thị

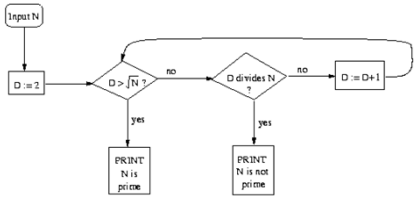


Các khái niệm cơ bản

Đồ thị

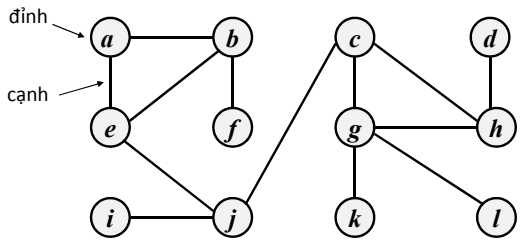
- Đồ thị – graph: mô tả *các mối quan hệ*
  - Mạng Internet
  - Mạng lưới đường giao thông
  - Sơ đồ cấu trúc điều khiển
  - Mạng xã hội
  - Mạch điện
  - …





Các khái niệm cơ bản

Một ***đồ thị G*** bao gồm một tập *V*(*G*) ***các đỉnh*** (***nút***) và một tập *E*(*G*) ***các cạnh*** (***cung***) là các cặp đỉnh.

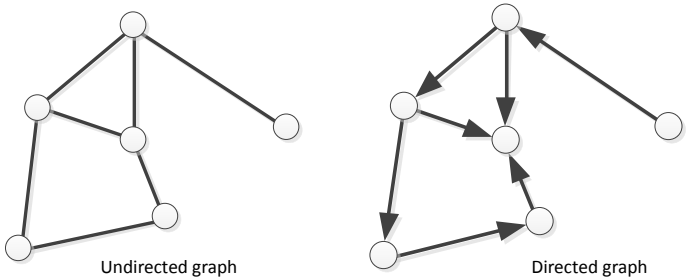


V = { *a, b, c, d, e, f, g, h, i, j, k, l* } 12 đỉnh

E = { (*a, b*), (*a, e*), (*b, e*), (*b, f*), (*c, j*), (*c, g*), (*c, h*), (*d, h*), (*e, j*), (*g, k*), (*g, l*), (*g, h*), (*i, j*) } 13 cạnh

Các khái niệm cơ bản

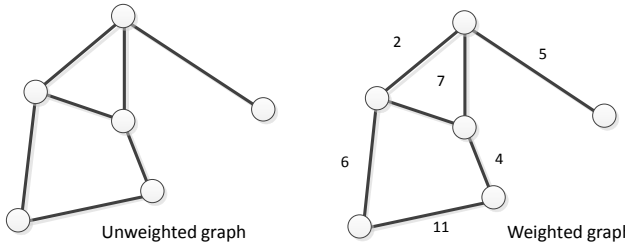
- **Đồ thị vô hướng – undirected graph**: Đồ thị G(V,E) là vô hướng nếu các cạnh (u,v) là một bộ **không** có thứ tự
- **Đồ thị có hướng – directed graph**: Đồ thị G(V,E) là có hướng nếu các cạnh (u,v) là một bộ **có** thứ tự



Undirected graph Directed graph

Các khái niệm cơ bản

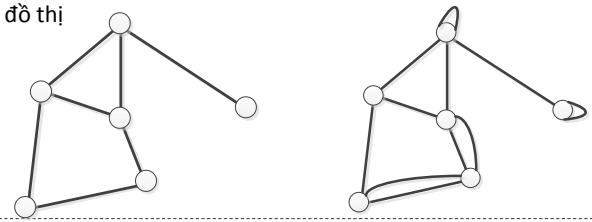
- **Đồ thị có trọng số - weighted graph**: Đồ thị G(V,E) là có trọng số nếu các cạnh hoặc đỉnh là được gán một giá trị số (trọng số)
- **Đồ thị không trọng số - unweighted graph**: Các cạnh, đỉnh không được gán trọng số hoặc có trọng số giống nhau



Unweighted graph Weighted graph

Các khái niệm cơ bản

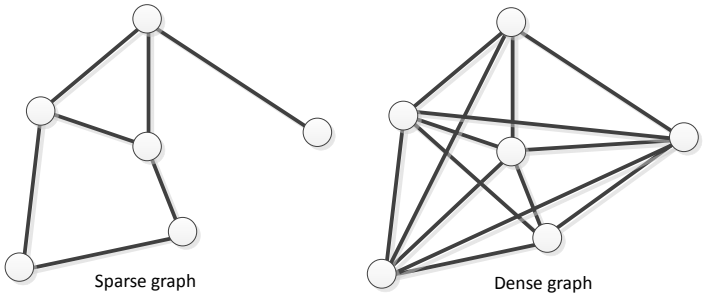
- **Đơn đồ thị - Simple graph**: giữa hai đỉnh chỉ tồn tại một cạnh nếu có
- **Không phải đơn đồ thị - non- simple graph**: trên đồ thị tồn tại một số kiểu cạnh đặc biệt:
  - **Cạnh vòng – self-loop**: xuất phát và kết thúc tại cùng một đỉnh
  - **Đa cạnh – multiedge**: một cạnh được lặp lại nhiều hơn một lần trên đồ thị



Simple graph Non-simple graph

Các khái niệm cơ bản

- **Đồ thị thưa – Sparse**: số cạnh trên đồ thị ít. Thường là tỉ lệ tuyến tính với số đỉnh
- **Đồ thị dày – Dense**: số cạnh trên đồ thị lớn

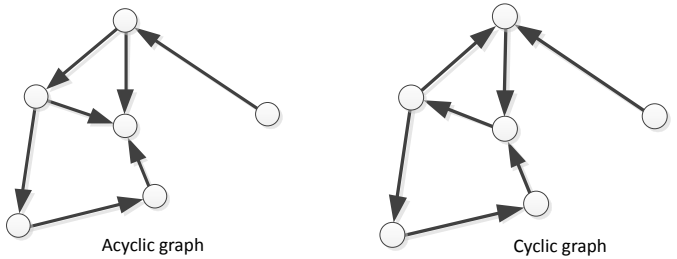


Sparse graph

Dense graph

Các khái niệm cơ bản

- **Đồ thị có chu trình – cyclic graph**: trong đồ thị có tồn tại chu trình
- **Đồ thị không có chu trình – acyclic graph**: trong đồ thị có tồn tại chu trình

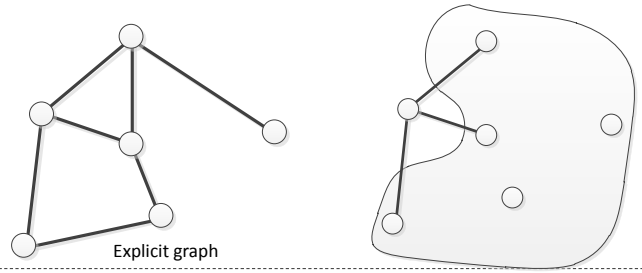


Acyclic graph

Cyclic graph

Các khái niệm cơ bản

- **Đồ thị tường minh – explicit graph**: các phần của đồ thị được xây dựng tường minh
- **Đồ thị không tường minh – implicit graph**: các phần của đồ thị không được xây dựng một cách tường minh, nhưng sẽ được xây dựng khi chúng ta sử dụng.

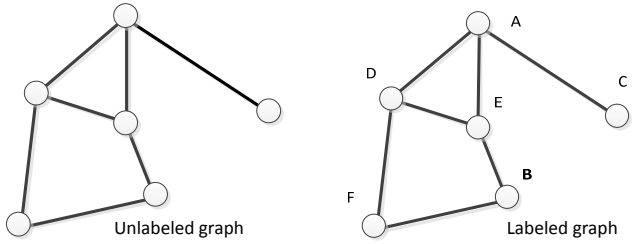


Explicit graph

Implicit graph

Các khái niệm cơ bản

- **Đồ thị có nhãn – labeled graph**: các đỉnh được gán nhãn để phân biệt với nhau
- **Đồ thị không có nhãn – unlabeled graph**: các đỉnh là như nhau (không được phân biệt). VD bài toán kiểm tra hai đồ thị là đồng hình – **isomorphism testing**.

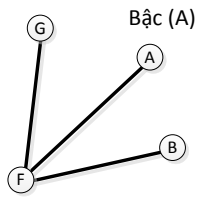


Unlabeled graph

Labeled graph

Các khái niệm cơ bản

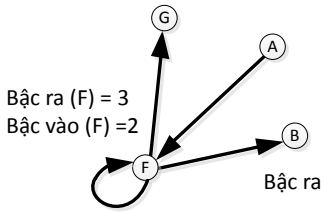
- **Bậc- degree** của đỉnh là số cạnh xuất phát (kết thúc) từ đỉnh đó
- Đỉnh u, v là **kề nhau** – **adjacent**: nếu tồn tại cạnh nối giữa u,v



Bậc (A) = 1

Bậc (F) = 3

A và F là kề nhau



Bậc ra (F) = 3
Bậc vào (F) =2

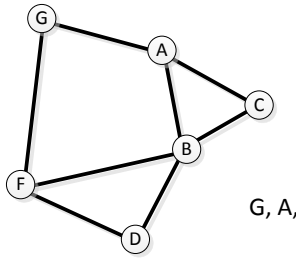
Bậc ra (B) = 0

A kề F nhưng F không kề với A

Cạnh (F, B): F là đỉnh nguồn, B là đỉnh đích

Các khái niệm cơ bản

- **Đường đi – path** giữa hai đỉnh v_1, v_k là một chuỗi các đỉnh $v_1, v_2, .., v_k$, trong đó (v_i, v_{i+1}) là một cạnh trên đồ thị ($i = 0, .., k-1$)
- **Độ dài đường đi**: số cạnh trên đường đi (số đỉnh -1)



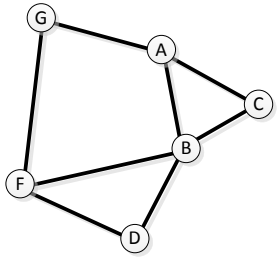
Một đường đi giữa G, D là G, A, B, D

Đường đi đơn: các đỉnh không lặp lại
G, A, C, B, D

G, A, B, C, A, B, D không phải đường đi đơn

Các khái niệm cơ bản

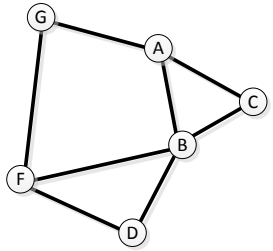
- **Chu trình – cycle**: đường đi trong đó có điểm đầu và điểm cuối trùng nhau
- **Chu trình đơn**: không có đỉnh nào trùng nhau trừ đỉnh đầu và đỉnh cuối



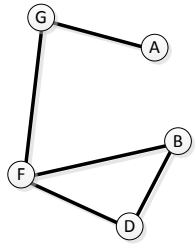
Chu trình: G, A, B, C, A, B, D, F, G
Chu trình đơn: G, A, B, D, F, G

Các khái niệm cơ bản

- Đồ thị $G'(V', E')$ là **đồ thị con** của $G(V, E)$ nếu
  - $V' \subseteq V$
  - $E' \subseteq E$



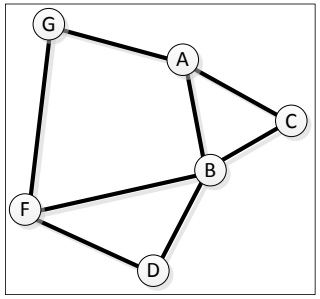
$G(V, E)$



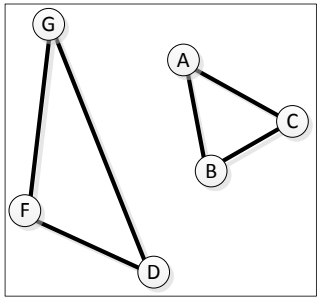
$G'(V', E')$

Các khái niệm cơ bản

- **Đồ thị liên thông – connected graph**: luôn tồn tại đường đi giữa hai cặp đỉnh bất kỳ trên đồ thị



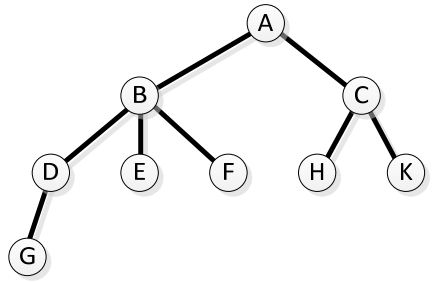
Connected graph



Not connected graph

Các khái niệm cơ bản

- **Quiz**: Cây có phải đồ thị liên thông?

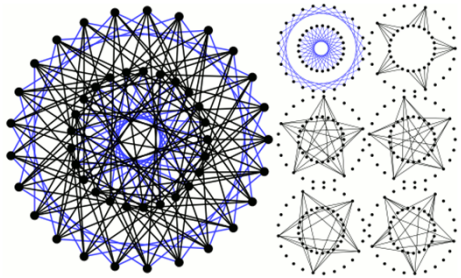


- Số cạnh, đỉnh trên cây?

Các khái niệm cơ bản

- **Biểu diễn mạng xã hội**: Facebook, LinkedIn, Twister,…

- Đồ thị có hướng hay vô hướng?
- Có trọng số hay không trọng số?
- Có cạnh vòng?
- Số bạn của một thành viên?
- Bạn có biết cô ấy/ anh ấy?
- ….



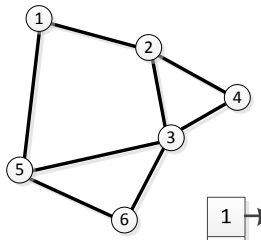
Biểu diễn đồ thị

- Ma trận kề
- Danh sách kề

Biểu diễn đồ thị

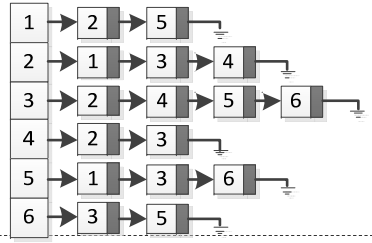
- Hai phương pháp biểu diễn cơ bản:
  - **Ma trận kề (Adjacency Matrix)**: Biểu diễn $G(V, E)$ Dùng ma trận M kích thước $n \times n$ (trong đó $|V| = n$). Nếu tồn tại cạnh giữa đỉnh i và j thì $M[i, j] = 1$, ngược lại bằng 0.
  - **Danh sách kề (Adjacency List)**: Biểu diễn $G(V, E)$ dùng mảng các danh sách móc nối. Mỗi danh sách móc nối tương ứng với 1 đỉnh chứa danh sách các đỉnh kề với đỉnh đó.

Biểu diễn đồ thị



	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	1	0	0
3	0	1	0	1	1	1
4	0	1	1	0	0	0
5	1	0	1	0	0	1
6	0	0	1	0	1	0

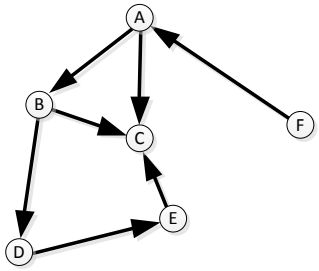
Ma trận kề



Danh sách kề

Biểu diễn đồ thị

Quiz. Biểu diễn đồ thị có hướng $G(V, E)$ sau



Ma trận kề

Danh sách kề

Ma trận kề và danh sách kề

	Ma trận kề	Danh sách kề
Kiểm tra cạnh (u, v)	$O(1)$	$O(\max(degree(u), degree(v)))$
Tìm bậc của u	$O(n)$	$O(degree(u))$
Kích thước bộ nhớ	$O(\|V\|^2)$	$O(\|V\| + \|E\|)$
Thêm xóa cạnh (u, v)	$O(1)$	$O(\max(degree(u), degree(v)))$
Duyệt trên đồ thị	$O(\|V\|^2)$	$O(\|V\| + \|E\|)$

NOTE: Nên dùng danh sách kề để biểu diễn các đồ thị trong trường hợp tổng quát

Biểu diễn đồ thị

```
#define MAXV 1000 /* maximum number of vertices */
typedef struct {
   int y; /* adjacency info */
   int weight; /* edge weight, if any */
   struct NODE *next; /* next edge in list */
} NODE;

typedef struct {
   NODE *edges[MAXV+1]; /* adjacency info */
   int degree[MAXV+1]; /* outdegree of each vertex */
   int nvertices; /* number of vertices in graph */
   int nedges; /* number of edges in graph */
   bool directed; /* is the graph directed? */
} GRAPH;
```

Biểu diễn đồ thị

- Thư viện các hàm về đồ thị thông dụng:
  - LEDA: http://www.algorithmic-solutions.com/leda/
  - Boost: http://www.boost.org/

Duyệt đồ thị

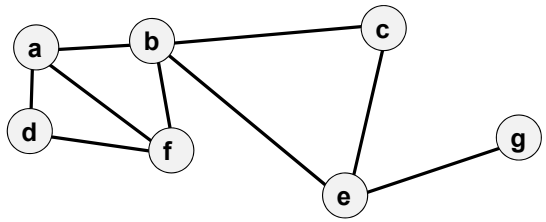
- Tìm kiếm theo chiều rộng
- Tìm kiếm theo chiều sâu

Tìm kiếm theo chiều rộng - BFS

- Cho đồ thị G(V, E), một đỉnh nguồn s
- Thuật toán tìm kiếm theo chiều rộng (***Breadth-first search*** - BFS) tìm kiếm tất cả các đỉnh mà có thể tới được từ đỉnh s. Tính toán đường đi ngắn nhất từ đỉnh s đến các đỉnh có thể tới được từ s.
- Để theo dõi quá trình duyệt, ta sử dụng các màu
  - Màu trắng: đỉnh chưa được thăm
  - Màu xám: đỉnh đang được thăm
  - Màu đen: đỉnh đã được thăm
- Đỉnh màu trắng có thể chuyển sang xám và màu xám có thể chuyển sang đen.

Tìm kiếm theo chiều rộng - BFS

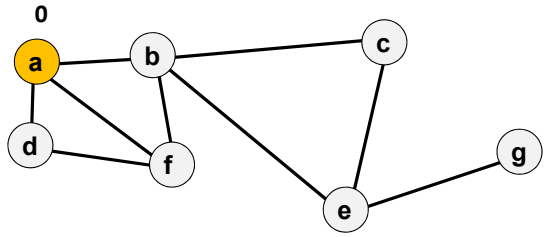
- Để quản lý các đỉnh đang được thăm (màu xám), một hàng đợi được sử dụng



Đỉnh bắt đầu là a

Queue

Tìm kiếm theo chiều rộng - BFS

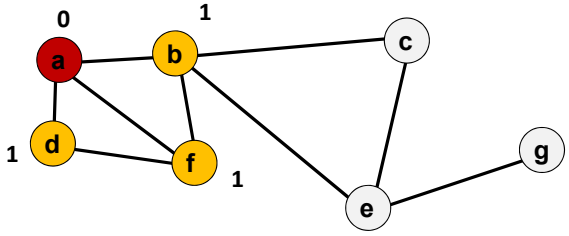


A được đưa vào queue

Queue | a

0

Tìm kiếm theo chiều rộng - BFS

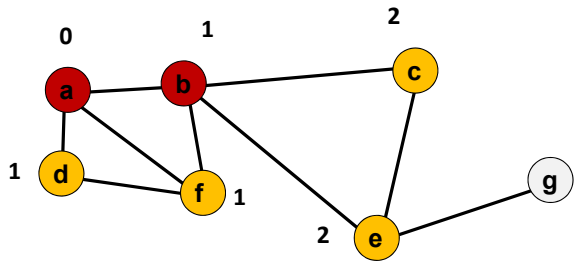


- a được lấy khỏi queue
- Các đỉnh kề a chưa được thăm là b, d, f được đẩy vào queue
- a đã được thăm xong, chuyển sang màu đen

Queue | b | d | f

1 1 1

Tìm kiếm theo chiều rộng - BFS

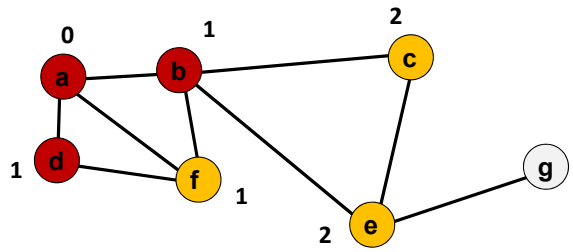


- b được lấy khỏi queue
- Các đỉnh kề b chưa được thăm là c, e được đẩy vào queue
- b đã được thăm xong, chuyển sang màu đen

Queue | d | f | c | e

1 1 2 2

Tìm kiếm theo chiều rộng - BFS

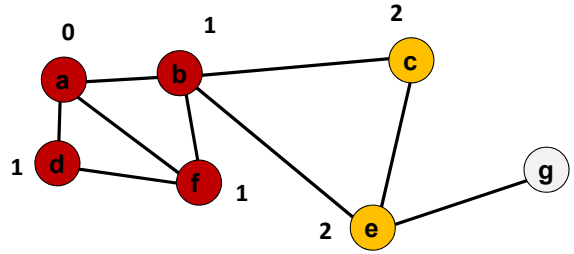


•d được lấy khỏi queue
•Không có đỉnh kề d mà chưa được thăm
•d đã được thăm xong, chuyển sang màu đen

Queue

f	c	e
1	2	2

Tìm kiếm theo chiều rộng - BFS

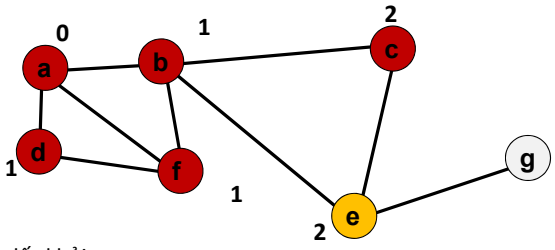


•f được lấy khỏi queue
•Không có đỉnh kề f mà chưa được thăm
•f đã được thăm xong, chuyển sang màu đen

Queue

c	e
2	2

Tìm kiếm theo chiều rộng - BFS

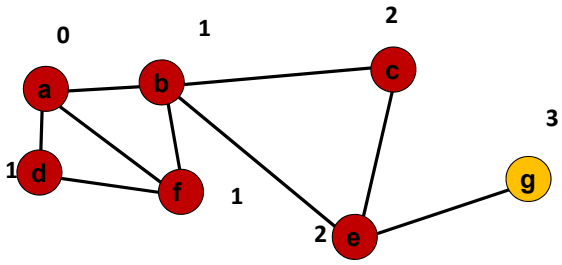


•c được lấy khỏi queue
•Không có đỉnh kề c mà chưa được thăm
•c đã được thăm xong, chuyển sang màu đen

Queue

e
2

Tìm kiếm theo chiều rộng - BFS

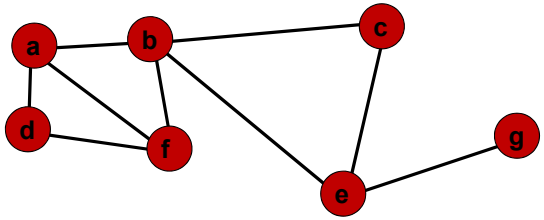


•e được lấy khỏi queue
•Đỉnh kề c mà chưa được thăm là g được đẩy vào queue
•e đã được thăm xong, chuyển sang màu đen

Queue

g
3

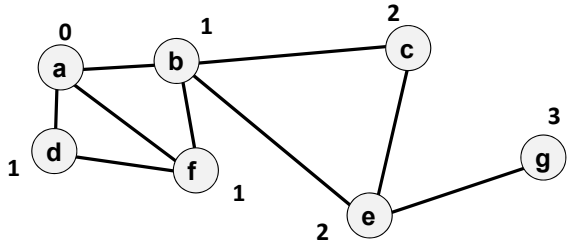
Tìm kiếm theo chiều rộng - BFS



- g được lấy khỏi queue
- Không có đỉnh kề g mà chưa được thăm
- g đã được thăm xong, chuyển sang màu đen

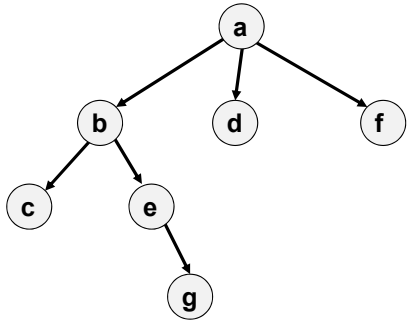
Queue

Tìm kiếm theo chiều rộng - BFS



Đồ thị sau khi đã thăm xong các đỉnh bắt đầu từ đỉnh a

Tìm kiếm theo chiều rộng - BFS



Cây khung thu được khi duyệt đồ thi bắt đầu từ đỉnh a

Tìm kiếm theo chiều rộng - BFS

- Thuật toán BFS, đồ thị G(V, E) và đỉnh bắt đầu là s
  - Gán tất cả các đỉnh trong G màu trắng
  - Thêm s và Queue, chuyển màu đỉnh s sang màu xám
  - Lặp : trong khi queue còn khác rỗng
    - Lấy 1 đỉnh u ra khỏi queue
    - Với các đỉnh kề v của đỉnh u mà chưa được thăm (màu trắng) theo thứ tự
      - thêm v vào queue
      - Đổi màu v sang màu xám
    - Đổi màu u sang màu đen

Tìm kiếm theo chiều rộng - BFS

- Độ phức tạp của BFS
  - $O(n^2)$ nếu dùng ma trận kề
  - O(|V|+|E|) nếu dùng danh sách kề

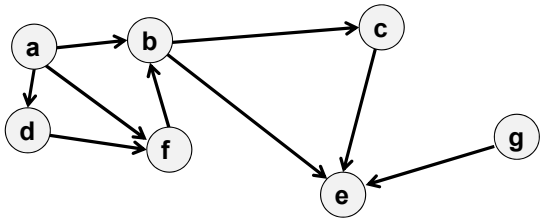
Tìm kiếm theo chiều sâu - DFS

- Thuật toán tìm kiếm theo chiều sâu **Depth-first search (DFS)**
  - Giống BFS nhưng ưu tiên tìm kiếm các đỉnh sâu hơn trước.
  - Khi một đỉnh được thăm xong, quay lui trở lại để xét tiếp các đỉnh đang thăm trước đó.
  - Sử dụng thêm tem thời gian để lưu thời gian thăm các đỉnh
- Tem thời gian là 1 số nguyên trong khoảng 1 đến 2|V|
- Với mỗi đỉnh v

$$d[v] < f[v]$$

Tìm kiếm theo chiều sâu - DFS

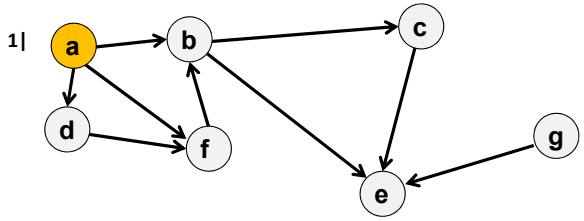
Ví dụ: cho đồ thị có hướng G(V, E)



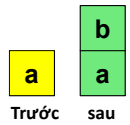
Đỉnh bắt đầu là a

Đẩy a vào stack

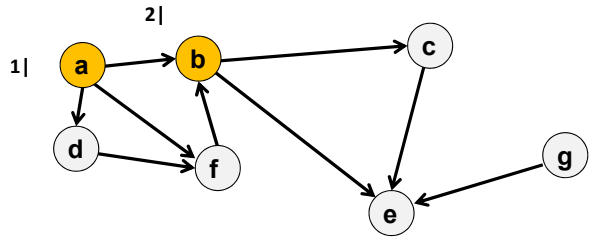
Tìm kiếm theo chiều sâu - DFS



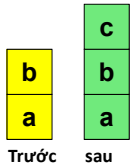
•Thăm đỉnh ở đầu stack là a
•Trong các đỉnh kề với a chưa được thăm(màu trắng) ta đẩy đỉnh tiếp theo vào stack (đỉnh b)



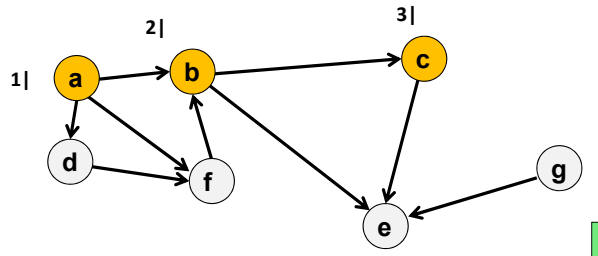
Tìm kiếm theo chiều sâu - DFS



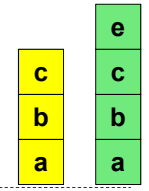
•Đỉnh thăm tiếp theo là đỉnh ở đầy stack (đỉnh b)
•Đỉnh kề với b mà chưa thăm là c và e, ta đẩy c vào stack



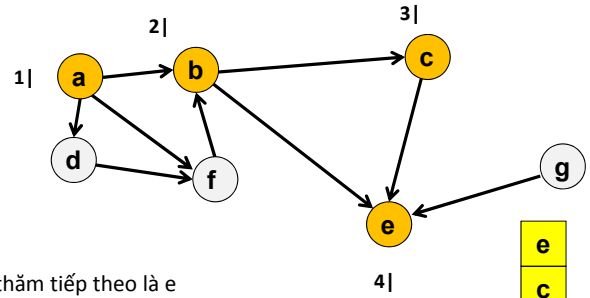
Tìm kiếm theo chiều sâu - DFS



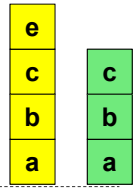
•Đỉnh thăm tiếp theo là c
•Đỉnh kề của c được đẩy vào stack là e



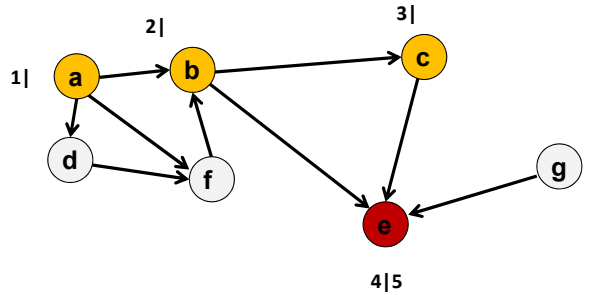
Tìm kiếm theo chiều sâu - DFS



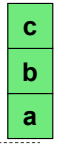
•Đỉnh thăm tiếp theo là e
•e không có đỉnh nào kề mà chưa được thăm nữa nên ta kết thúc thăm e (chuyển màu đen)
•Lấy e ra khỏi stack



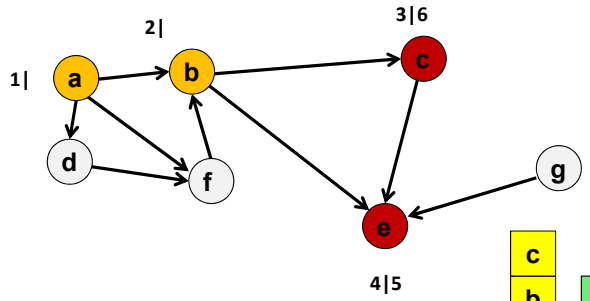
Tìm kiếm theo chiều sâu - DFS



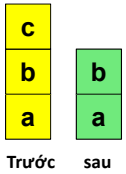
Kết thúc thăm e



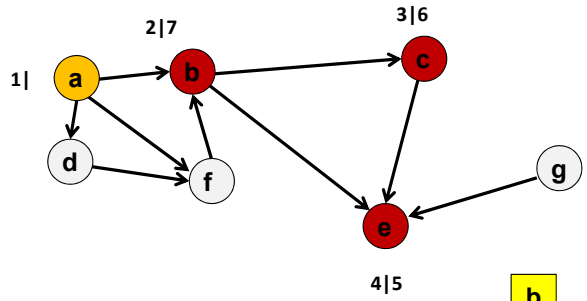
Tìm kiếm theo chiều sâu - DFS



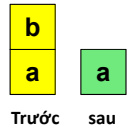
- Đỉnh thăm tiếp là c, không có đỉnh nào kề c mà chưa thăm nên ta kết thúc thăm c (chuyển màu đen).
- Đẩy c ra khỏi stack



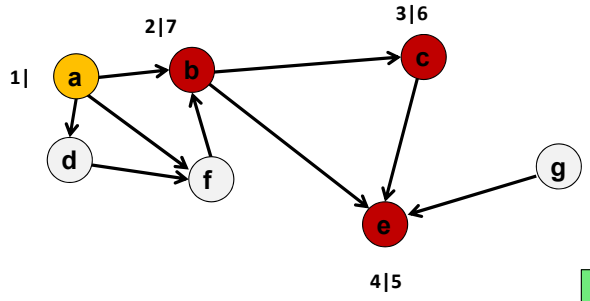
Tìm kiếm theo chiều sâu - DFS



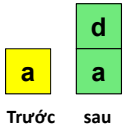
•Đỉnh thăm tiếp là b, không có đỉnh nào kề b mà chưa thăm nên ta kết thúc thăm b (chuyển màu đen).
•Đẩy b ra khỏi stack



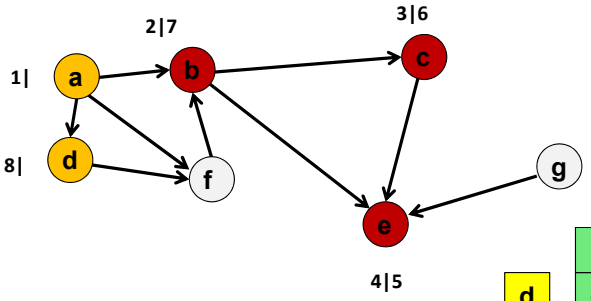
Tìm kiếm theo chiều sâu - DFS



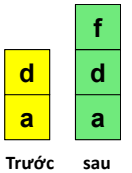
Đỉnh thăm tiếp theo là a, đỉnh kề của a chưa thăm là d,f. Ta đẩy d vào stack



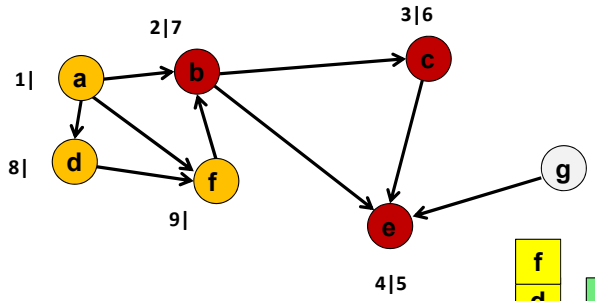
Tìm kiếm theo chiều sâu - DFS



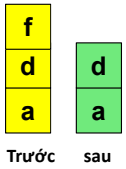
Đỉnh thăm tiếp theo là d, đỉnh kề của d chưa thăm là f. Ta đẩy f vào stack



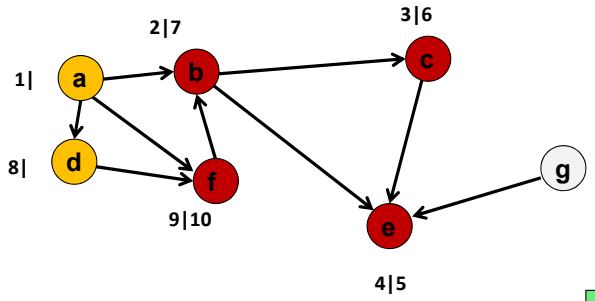
Tìm kiếm theo chiều sâu - DFS



Đỉnh thăm tiếp theo là f, không còn đỉnh nào kề f mà chưa được thăm. Ta kết thúc thăm f (chuyển f sang màu đen) và đẩy f ra khỏi stack

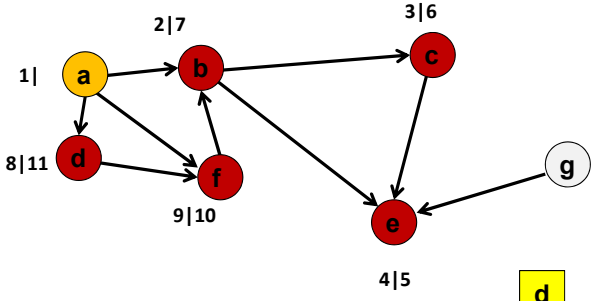


Tìm kiếm theo chiều sâu - DFS

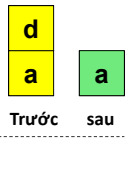


Kết thúc thăm f

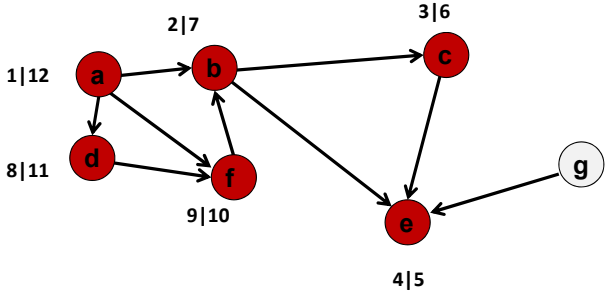
Tìm kiếm theo chiều sâu - DFS



Đỉnh thăm tiếp là d, không có đỉnh nào kề b mà chưa thăm nên ta kết thúc thăm d (chuyển màu đen).
Đẩy d ra khỏi stack

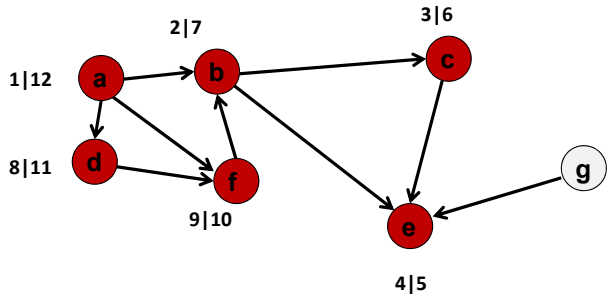


Tìm kiếm theo chiều sâu - DFS



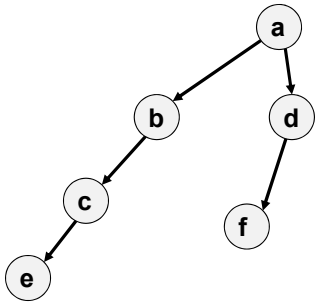
- Đỉnh thăm tiếp là a, không có đỉnh nào kề a mà chưa thăm nên ta kết thúc thăm a (chuyển màu đen).
- Đẩy a ra khỏi stack

Tìm kiếm theo chiều sâu - DFS



Stack đã rỗng, ta dừng thuật toán DFS tại đây!
Các đỉnh được thăm và thời gian thăm trên hình

Tìm kiếm theo chiều sâu - DFS

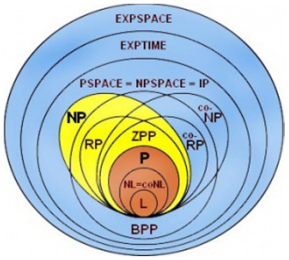


Cây khung thu được khi duyệt đồ thị bắt đầu từ đỉnh a

Tìm kiếm theo chiều sâu - DFS

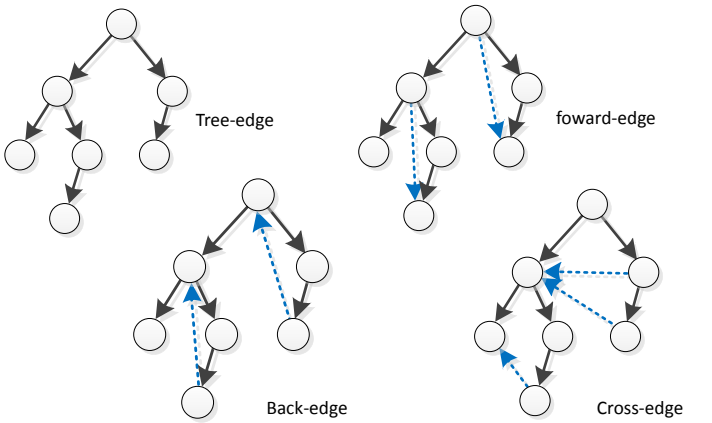
- Thuật toán DFS, đồ thị G(V, E) và đỉnh bắt đầu là s
  - Gán tất cả các đỉnh trong G màu trắng
  - Thêm s và stack, chuyển màu đỉnh s sang màu xám
    - Lặp: trong khi stack còn khác rỗng
      - Xét đỉnh u ở đầu stack
      - Nếu còn tồn tại đỉnh kề với u chưa được thăm (đỉnh có màu trắng)
        - Thêm đỉnh kề v đầu tiên của u vào stack
        - Đổi màu v sang màu xám
      - Ngược lại :
        - Đổi màu u sang màu đen
        - Đẩy u ra khỏi stack

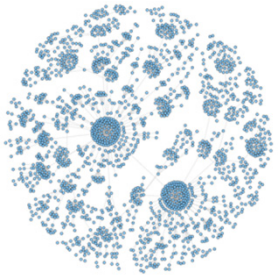
Tìm kiếm theo chiều sâu - DFS



- Độ phức tạp của DFS
  - Cỡ $O(|V|^2)$ nếu dùng ma trận kề
  - Cỡ $O(|V|+|E|)$ nếu dùng danh sách kề

Các loại cạnh trên cây theo DFS/BFS

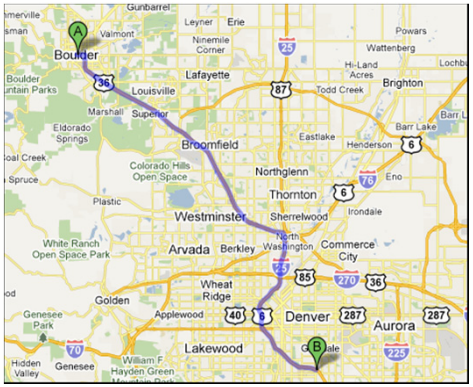




Ứng dụng của các thuật toán duyệt đồ thị

Ứng dụng của các thuật toán duyệt đồ thị

- **Bài toán 1. Tìm đường đi ngắn nhất giữa hai đỉnh trên đồ thị**
  - Dùng BFS?
  - Dùng DFS?

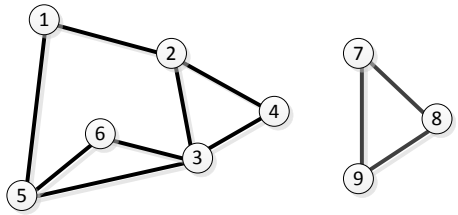


Bài toán 1. Tìm đường đi ngắn nhất

- Trường hợp đồ thị không có trọng số âm:
  - Thuật toán **Dijkstra**: Cải tiến từ BFS
  - Dùng Priority queue lưu trữ các đỉnh theo quãng đường
- Đồ thị có trọng số âm
  - Dijkstra không thực hiện được?
- Trường hợp không có chu trình âm
  - Thuật toán **Ford Bellman**
- Đồ thị có chu trình âm
  - Thuật toán **Floyd Warshall**

Ứng dụng của các thuật toán duyệt đồ thị

- **Bài toán 2. Thành phần liên thông – connected components**
  Thành phần liên thông của 1 đồ thị vô hướng là tập đỉnh lớn nhất mà luôn tồn tại đường đi giữa 2 đỉnh bất kỳ trong đó

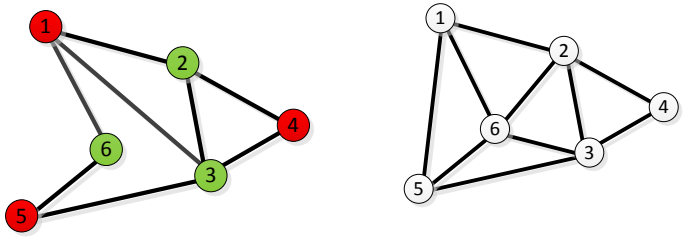


Bài toán 2. Thành phần liên thông

- Kiểm tra đồ thị liên thông
  - Thực hiện DFS hoặc BFS
  - Tất cả các đỉnh đều được thăm → liên thông
- Tìm kiếm thành phần liên thông
  - Thực hiện DFS hoặc BFS
  - Đưa ra tập đỉnh liên thông lớn nhất

Ứng dụng của các thuật toán duyệt đồ thị

- **Bài toán 3. Bi-coloring graphs** – tô màu các đỉnh trên đồ thị bằng 2 màu. Kiểm tra xem có thể tô màu các đỉnh trên đồ thị chỉ bằng 2 màu sao cho 2 đỉnh của mỗi cạnh đều có màu khác nhau

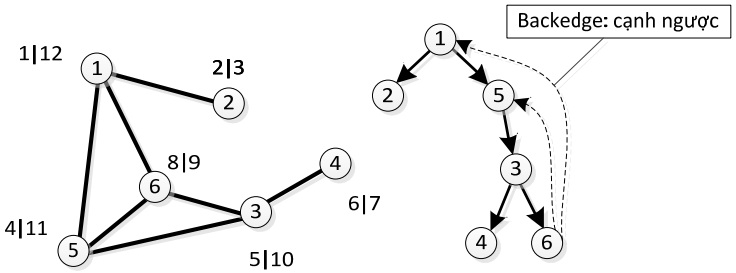


Bài toán 3. Bi-coloring

- Kiểm tra bằng cách duyệt đồ thị
  - Thực hiện BFS để đánh màu các đỉnh
  - Hai mút của 1 cạnh phải có 2 màu khác nhau
  - Nếu tồn tại 1 cạnh mà 2 mút có cùng màu → không tô được bằng 2 màu
- Bài toán tô màu đồ thị tổng quát
  - Dùng số màu ít nhất để tô cho các đỉnh sao cho không cạnh nào có 2 mút trùng màu
  - Là bài toán thuộc lớp NP-Khó
  - Không có thuật toán hiệu quả để giải trong trường hợp tổng quát

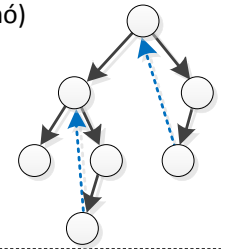
Ứng dụng của các thuật toán duyệt đồ thị

- **Bài toán 4. Tìm chu trình trên đồ thị**



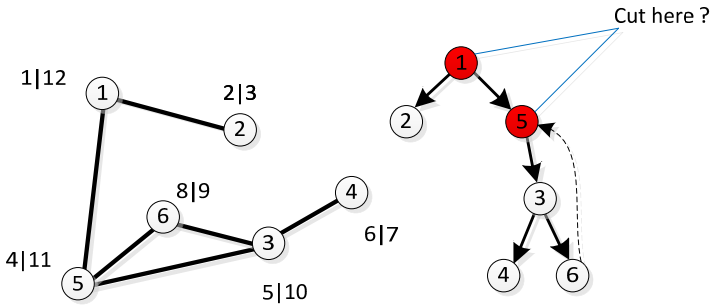
Bài toán 4. Tìm chu trình

- Phát hiện chu trình bằng Back Edge
  - Duyệt đồ thị bằng DFS
  - Tìm cạnh ngược – back edge
- Cạnh ngược – back edge: cạnh từ nút con cháu đến nút tổ tiên (không phải nút cha trực tiếp của nó)



Ứng dụng của các thuật toán duyệt đồ thị

- **Bài toán 5. Tìm đỉnh khớp (đỉnh cắt)– *articulation vertex (cut node)***. Đỉnh mà nếu bị gỡ bỏ sẽ làm đồ thị liên thông bị tách thành 2 phần

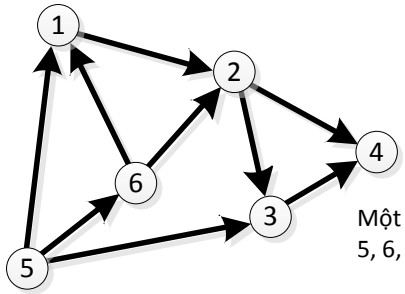


Bài toán 5. Articulation vertex

- Thực hiện **brute force**: duyệt tất cả các đỉnh trên cây để tìm đỉnh cắt
  - Thời gian thực hiện lớn $O(|V|(|V| + |E|))$
- Thực hiện tìm trên cây khung thu được sau khi DFS
  - DFS trên đồ thị vô hướng chỉ thu được tree edge và back edge
  - Nút lá trên cây khung thu được không phải là đỉnh cắt
  - Thời gian thực hiện $O(|V| + |E|)$
- Các loại đỉnh cắt – ***articulation vertex*** có thể:
  - Nút gốc có nhiều hơn 1 con
  - Nút mà các nút con cháu của nó không có cạnh ngược đến nút tổ tiên

Ứng dụng của các thuật toán duyệt đồ thị

- **Bài toán 6. Sắp xếp topo – topological sorting**: cho một đồ thị vô hướng không có chu trình – DAG, hãy đưa ra một thứ tự các đỉnh trong đó nếu có cạnh (u,v) thì u phải đứng trước v



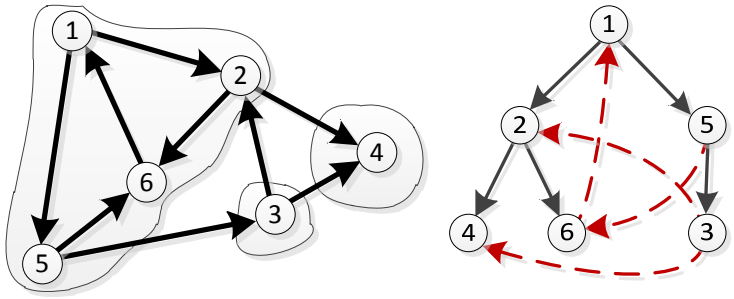
Một thứ tự topo của đồ thị là 5, 6, 1, 2, 3, 4

Bài toán 6. Topological sorting

- Dùng DFS:
  - Nếu có cạnh (u,v) thì thời điểm kết thúc thăm u bao giờ cũng lớn hơn thời điểm kết thúc thăm v
  - Thực hiện DFS trên toàn bộ đồ thị và đưa ra các đỉnh theo thứ tự thời điểm kết thúc thăm giảm dần
- Dùng BFS
  - Đỉnh mà không có cạnh tới phải đứng đầu trong danh sách topo
  - Tìm đỉnh không có cạnh tới, đưa đỉnh này vào danh sách topo sau đó xoá các cạnh xuất phát từ đỉnh đó

Ứng dụng của các thuật toán duyệt đồ thị

- **Bài toán 7. Thành phần liên thông mạnh – strongly connected component**. Đó là một phần của một đồ thị có hướng trong đó luôn tồn tại đường có hướng giữa hai cặp đỉnh bất kỳ.



Bài toán 7. strongly connected component

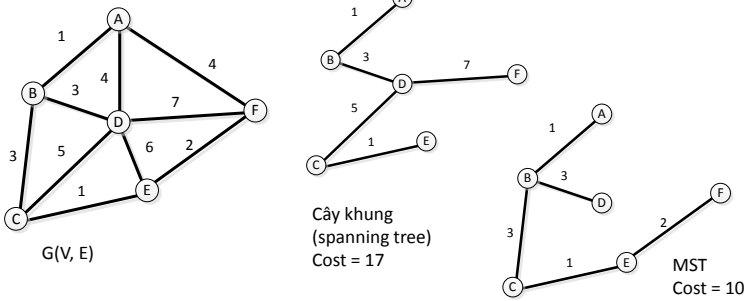
- Kiểm tra đồ thị là ***strongly connected***
  - Thực hiện DFS từ 1 đỉnh v tuỳ ý
  - Đổi hướng các đỉnh trên đồ thị cũ và thực hiện DFS lại với đỉnh v
- Bài toán chia đồ thị thành các tập con trong đó mỗi tập con là một thành phần liên thông mạnh
  - Không có một đỉnh nào là cùng thuộc 2 tập con liên thông mạnh trên đồ thị
  - Thuật toán
    - **Kosaraju**: cần 2 lần DFS trên đồ thị
    - **Tarjan**
    - **Gabow**

    } 1 Lần DFS
  - Thời gian $O(|V| + |E|)$

Ứng dụng của các thuật toán duyệt đồ thị

- **Bài toán 8. Cây khung có trọng số nhỏ nhất trên đồ thị - minimum spanning tree (MST)**

Cây khung: là cây chứa mà kết nối tất cả các đỉnh của đồ thị



Bài 8. Minimum spanning tree

- **Thuật toán PRIM**
  - Đồ thị $G(V, E)$, cây khung $T(V', E')$ ($V' = V$ khi kết thúc)
  - Khởi tạo
    - $E' = \emptyset, V' = \emptyset$
    - $Cost = 0$
    - $V' = V' \cup \{v\}$ (v là đỉnh bất kỳ trên $G(V, E)$)
  - Lặp cho tới khi $V' = V$
    - Tìm cạnh $(u, v), u \in V', v \in V \setminus V'$ có trọng số nhỏ nhất
    - $E' = E' \cup \{(u, v)\}$
    - $V' = V' \cup \{v\}$
    - $Cost = Cost + weight(u, v)$

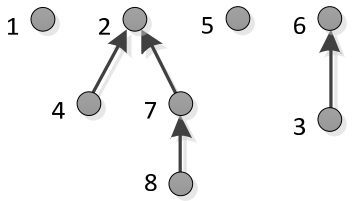
Bài 8. Minimum spanning tree

- **Thuật toán Kruskal**
  - Đồ thị $G(V, E)$, cây khung $T(V', E')$ ($V' = V$ khi kết thúc)
  - Khởi tạo
    - Sắp xếp các cạnh theo thứ tự tăng dần trọng số (priority queue Q)
    - $E' = \emptyset, V' = \emptyset$
    - $Cost = 0, Count = 0$
  - Lặp cho tới khi $Count = |E| - 1$
    - Lấy cạnh (u, v) tiếp theo trong Q
    - Nếu thêm (u, v) vào T mà không tạo thành chu trình
      - □ $V' = V' \cup \{u, v\}; E' = E' \cup \{(u, v)\}$
      - □ $Cost = Cost + weight(u, v); Count = count + 1$
    - Ngược lại, loại bỏ (u, v) khỏi Q

Bài 8. Minimum spanning tree

- **PRIM**
  - Lặp $|V|$ lần, mỗi lần phải duyệt $|E|$ cạnh ? Thời gian $O(|V| * |E|)$?
  - Mỗi lần lặp chỉ xét thêm $\leq |V| - 1$ cạnh nhỏ nhất → $O(|V|^2)$
  - Cài đặt dùng hàng đợi ưu tiên phức tạp giúp giảm thời gian thực hiện xuống $O(|E| + |V| \log |V|)$
- **Kruskal**
  - Sắp xếp $|E|$ cạnh mất thời gian $O(|E| \log |E|)$
  - Lặp $|V| - 1$ lần, mỗi lần thêm cạnh và kiểm tra chu trình mất thời gian thực hiện cỡ $|V| * |E|$ nên thời gian Kruskal cỡ
    $$O(|V| * |E| + |E| \log |E|)$$
  - Cải thiện thao tác kiểm tra chu trình trong Kruskal, dùng cấu trúc các tập độc lập – Union data structure

Kiểu cấu trúc Union



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

- Hai thao tác của Union
  - $Find(i)$: tìm gốc (nhãn) của phần tử i
  - $Union(i, j)$: thực hiện hợp hai tập chứa i và tập chứa j thành một tập

Kiểu cấu trúc Union

- Áp dụng vào Kruskal để kiểm tra chu trình
  - Thêm cạnh (u, v) mà tạo thành chu trình: u, v phải thuộc cùng 1 tập
  - Nếu (u, v) không tạo thành chu trình, ta thêm (u, v) vào cây và thực hiện hợp hai tập chứa u và v thành một
- Thời gian thực hiện của Kruskal giảm xuống còn $O(|E| \log|E|)$
- Kruskal nhanh hơn PRIM trên đồ thị thưa trong trường hợp tổng quát (PRIM có thời gian thực hiện cỡ $O(|V|^2)$)

Bài 8. Minimum spanning tree

- **<u>Một số biến thể của bài toán</u>**
  - **Cây khung có trọng số lớn nhất – Maximum Spanning Tree**
    - Đảo dấu các trọng số của đồ thị cũ
  - **Cây khung có tích trọng số nhỏ nhất – Minimum Product Spanning Tree**
    - Chuyển trọng số về logarithm $\log(a * b) = \log(a) + \log(b)$
  - **Cây khung giảm thiểu nghẽn – Minimum Bottleneck Spanning Tree**: tìm cây khung mà tối thiểu trọng số cạnh lớn nhất
    - Tất cả các MST đều có tính chất này
- **Steiner Tree** – Cây Steiner.
- **Low-degree Spanning Tree**: tìm cây khung nhỏ nhất mà có bận của nút lớn nhất là nhỏ