Chương 7 : Đồ thị và các thuật toán liên quan

Trịnh Anh Phúc [1]

[1]Bộ môn Khoa Học Máy Tính, Viện CNTT & TT,
Trường Đại Học Bách Khoa Hà Nội.

Ngày 9 tháng 7 năm 2014

Giới thiệu

1. Đồ thị
   - Định nghĩa
   - Các loại đồ thị
   - Đường đi, chu trình và tính liên thông của đồ thị
2. Biểu diễn đồ thị
   - Biểu diễn đồ thị bởi ma trận
   - Biểu diễn đồ thị bằng danh sách kề
   - Biểu diễn đồ thị bằng danh sách cạnh
3. Các thuật toán duyệt đồ thị
   - Thuật toán tìm kiếm theo chiều rộng - BFS
   - Thuật toán tìm kiếm theo chiều sâu - DFS
4. Bài toán cây khung nhỏ nhất
   - Thuật toán Kruskal
   - Cấu trúc dữ liệu biểu diễn phân hoạch
5. Bài toán đường đi ngắn nhất
   - Thuật toán Dijkstra
   - Cài đặt thuật toán với các cấu trúc dữ liệu

Đồ thị

Định nghĩa

Đồ thị G là cấu trúc rời rạc gồm hai tập

- tập đỉnh V là tập hữu hạn
- tập cạnh E là tập hữu hạn có thể rỗng gồm các cặp (u,v) với u,v $\in$ V

Ta ký hiệu như sau G= (V,E)

Các loại đồ thị

Tuy thuộc vào kiểu cạnh, ta có thể chia đồ thị thành hai loại chính

Đơn (đa) đồ thị vô hướng G=(E,V) là cặp gồm :

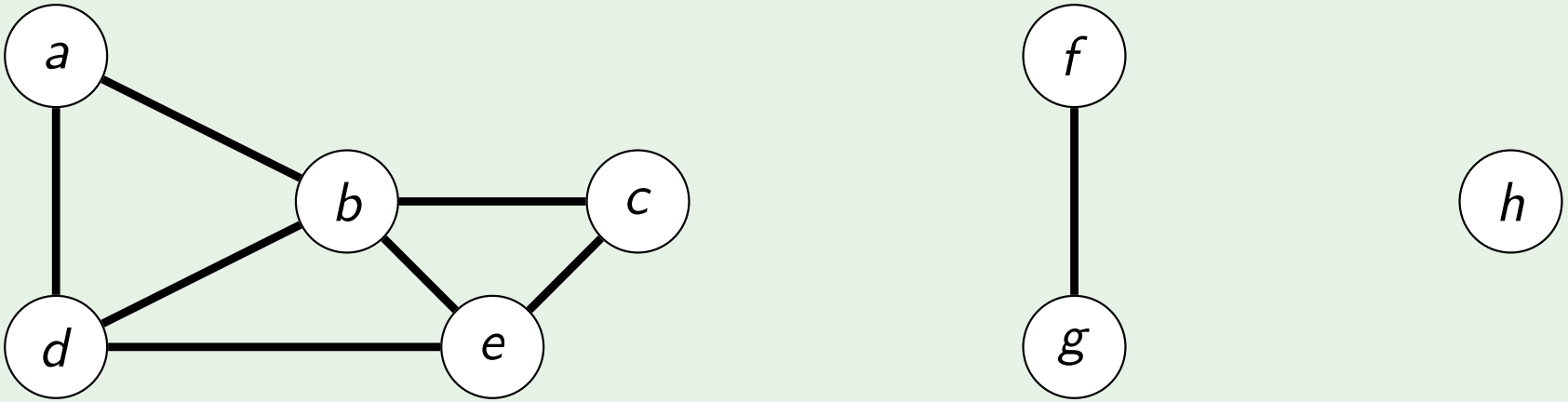
- Tập đỉnh V là tập hữu hạn các phần tử, gọi là các **đỉnh**
- Tập cạnh E là tập (họ) các cặp *không có thứ tự* dạng (u,v), u, v $\in$ V, u $\neq$ v. Các phần tử này của E gọi là các **cạnh**.

Đơn (đa) đồ thị có hướng G=(E,V) là cặp gồm :

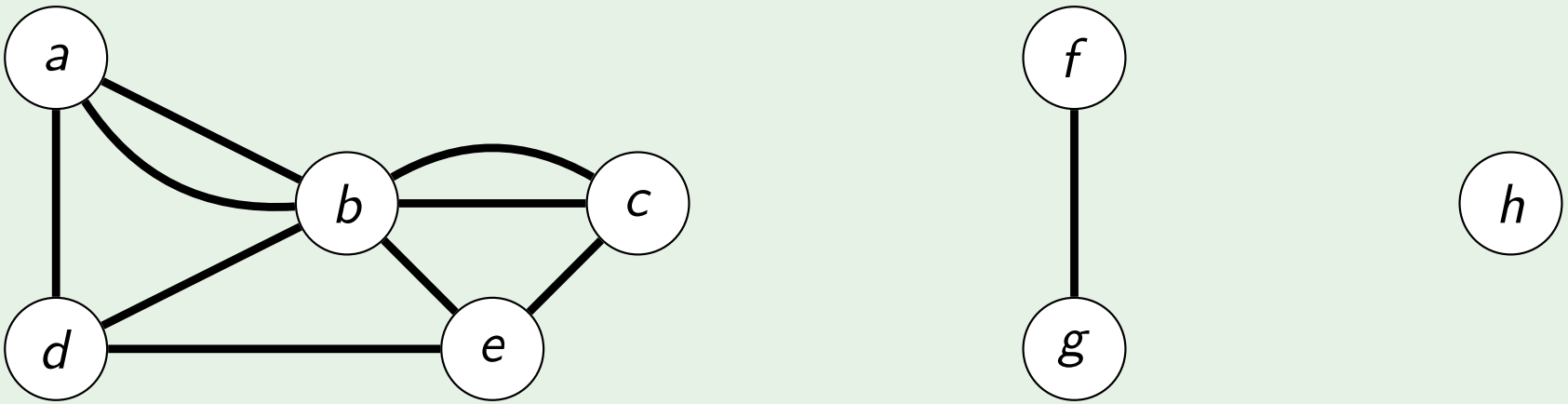
- Tập đỉnh V là tập hữu hạn các phần tử, gọi là các **đỉnh**
- Tập cạnh E là tập (họ) các cặp *có thứ tự* dạng (u,v), u, v $\in$ V, u $\neq$ v. Các phần tử này của E gọi là các cạnh có hướng hay các **cung**.

Các loại đồ thị (tiếp)

Đơn đồ thị vô hướng $G_1 = (V_1, E_1)$

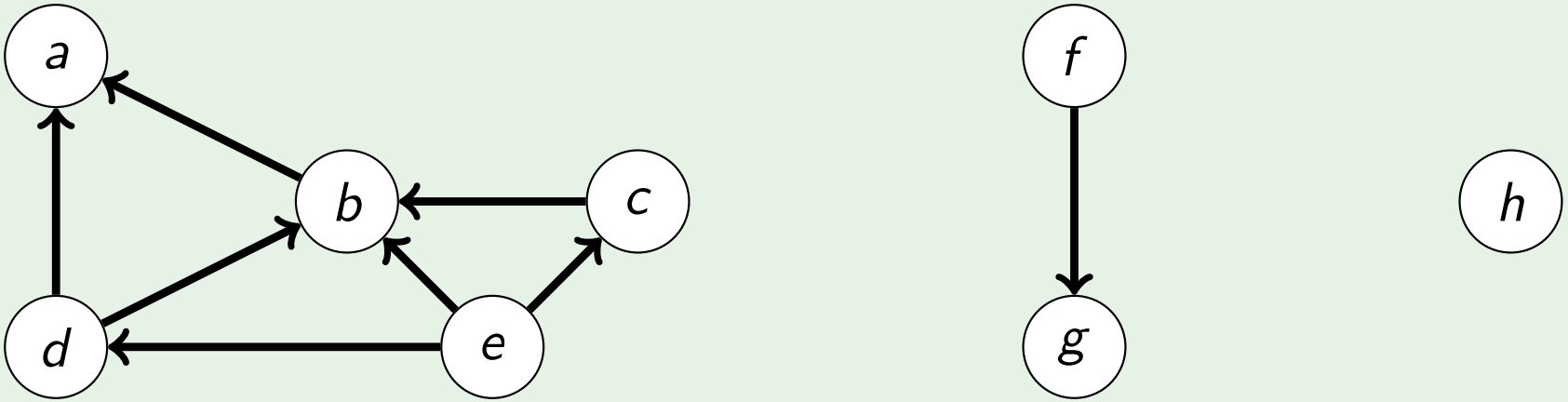


Đa đồ thị vô hướng $G_2 = (V_2, E_2)$

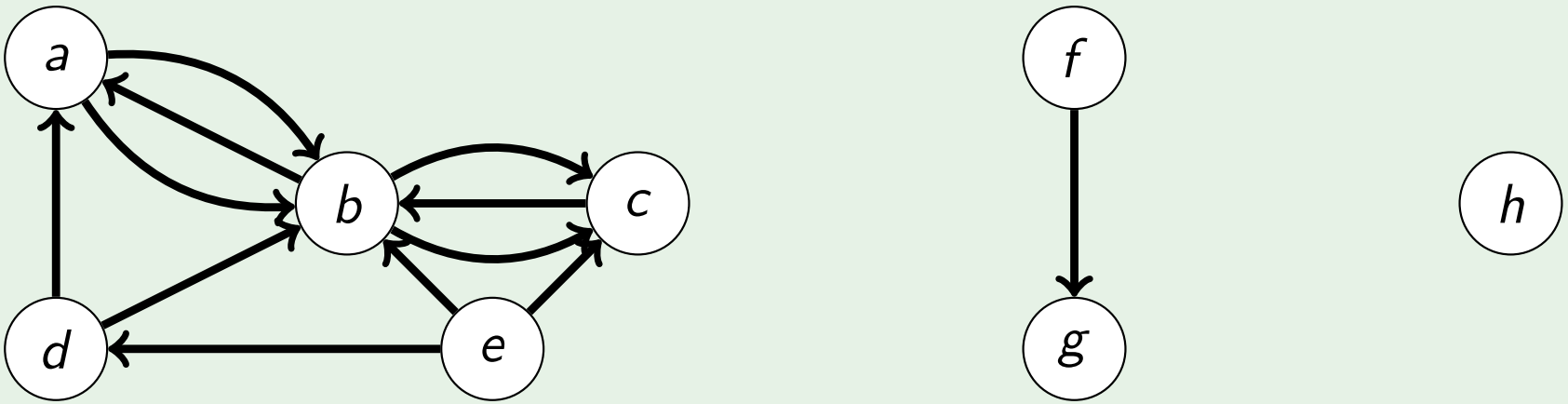


Các loại đồ thị (tiếp)

Đơn đồ thị có hướng $G_3 = (V_3, E_3)$



Đa đồ thị có hướng $G_4 = (V_4, E_4)$



Các loại đồ thị (tiếp)

Bảng phân loại đồ thị

Loại đồ thị	Kiểu cạnh	Có lặp cạnh ?
Đơn đồ thị vô hướng	Vô hướng	Không
Đa đồ thị vô hướng	Vô hướng	Có
Đơn đồ thị có hướng	Có hướng	Không
Đa đồ thị có hướng	Có hướng	Có

Các thuật ngữ mối quan hệ giữa các đỉnh và cạnh đồ thị

Kề nhau, liên thuộc, nối, đầu mút trong đồ thị vô hướng G = (V,E)

Giả sử e = (u,v) $\in$ E với u,v $\in$ V, ta nói

- u,v là *kề nhau/lân cận/nối với nhau*
- Cạnh e là *liên thuộc* với hai đỉnh u và v
- Cạnh e *nối* u và v
- Hai đỉnh u và v là *đầu mút* của cạnh e

Các thuật ngữ mối quan hệ giữa các đỉnh và cạnh đồ thị (tiếp)

Kề nhau, đi ra khỏi, đi vào, đi từ.. tới.., đỉnh đầu, đỉnh cuối trong đồ thị có hướng G = (V,E)

Giả sử e = (u,v) $\in$ E với u,v $\in$ V, ta nói

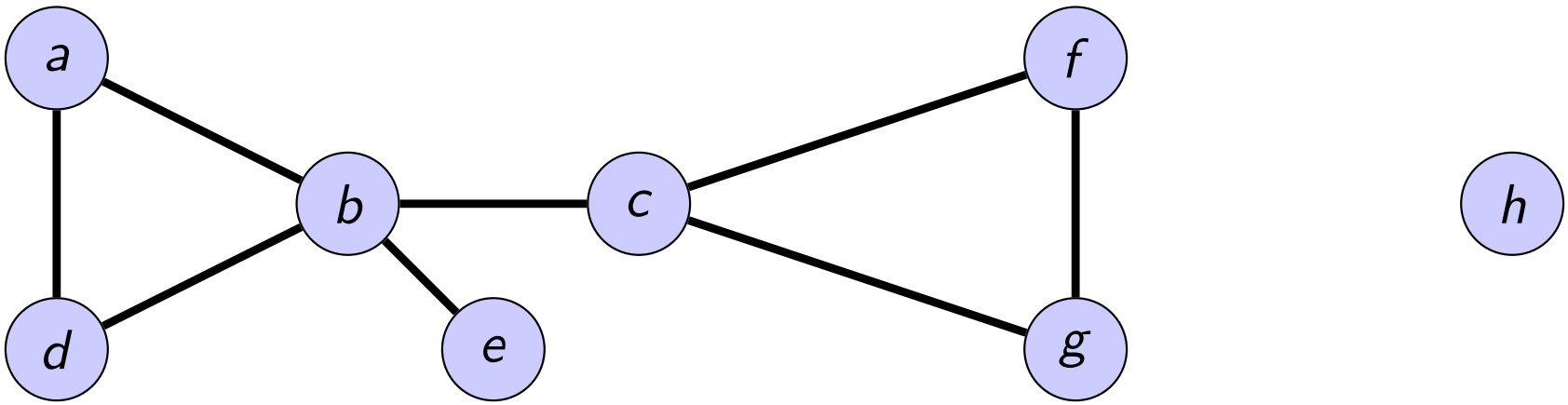
- u,v là *kề nhau*
- Cạnh e *đi ra khỏi* u và *đi vào* v
- Cạnh e *nối* u với v, e *đi từ u tới v*
- *Đỉnh đầu* của e là u
- *Đỉnh cuối* của e là v

Các thuật ngữ mối quan hệ giữa các đỉnh và cạnh đồ thị (tiếp)

Bậc của đỉnh trong đồ thị vô hướng G

Bậc của đỉnh v $\in$ V, ký hiệu deg(v), thuộc đồ thị vô hướng G=(V,E) là số cạnh liên thuộc với nó.

- *Đỉnh cô lập* có bậc bằng không
- *Đỉnh treo* có bậc bằng một



Đỉnh treo e có deg(e) = 1, đỉnh cô lập h có deg(h) = 0

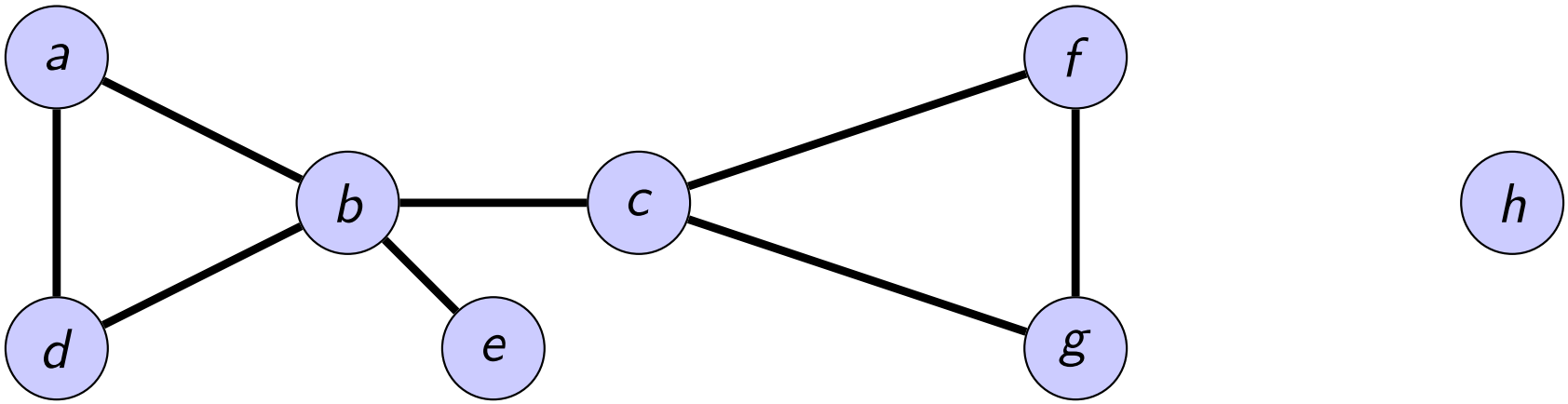
Các thuật ngữ mối quan hệ giữa các đỉnh và cạnh đồ thị (tiếp)

Định lý bắt tay

Giả sử đồ thị vô hướng G=(V,E) khi đó $\sum_{v \in V} deg(v) = 2|E|$

Hệ quả của định lý bắt tay

Trong một đồ thị vô hướng bất kỳ, số lượng các đỉnh bậc lẻ luôn là số chẵn



Có hai đỉnh bậc lẻ là deg(e)=1, deg(c)=3

Các thuật ngữ mối quan hệ giữa các đỉnh và cạnh đồ thị (tiếp)

Bậc của đỉnh trong đồ thị có hướng G

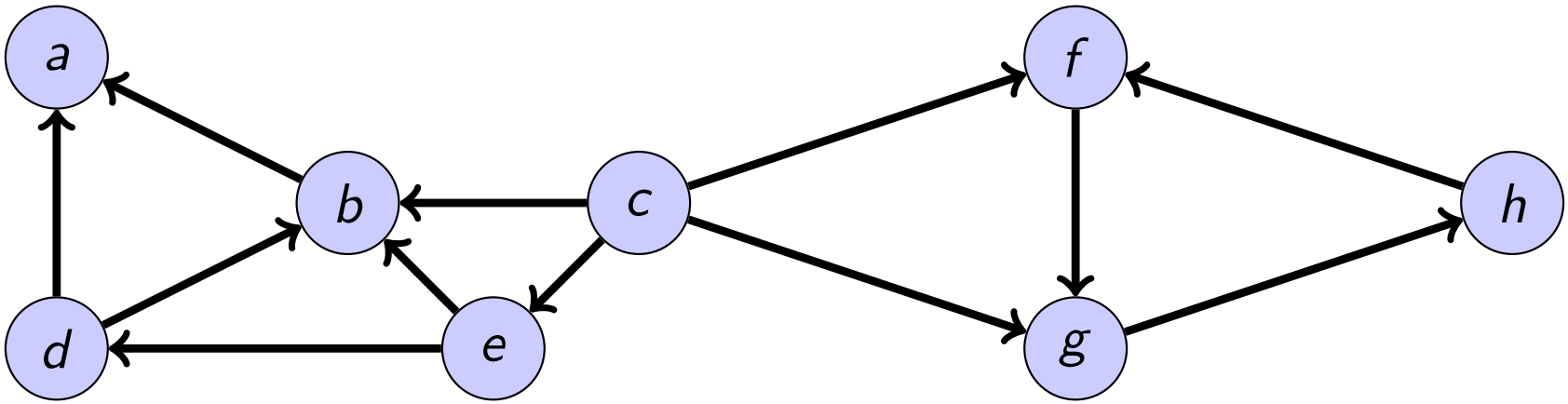
Cho G=(V,E) là đồ thị có hướng, v $\in$ V, ta nói

- *Bán bậc vào* của v, $deg^-(v)$ là số cạnh đi vào v
- *Bán bậc ra* của v, $deg^+(v)$ là số cạnh đi từ v
- *Bậc* của v, $deg(v) = deg^-(v) + deg^+(v)$ là tổng số bán bậc vào và ra của v

Kéo theo các định nghĩa sau

- *Đỉnh nguồn* là đỉnh v có $deg^+(v) > 0$ và $deg^-(v) = 0$
- *Đỉnh đích* là đỉnh v có $deg^+(v) = 0$ và $deg^-(v) > 0$

Các thuật ngữ mối quan hệ giữa các đỉnh và cạnh đồ thị (tiếp)



Đỉnh nguồn c : $deg^+(c) = 4$ và $deg^-(c) = 0$

Đỉnh đích a : $deg^+(a) = 0$ và $deg^-(a) = 2$

Các thuật ngữ mối quan hệ giữa các đỉnh và cạnh đồ thị (tiếp)

Định lý về bậc trong đồ thị có hướng G

Giả sử G=(V,E) là đồ thị có hướng thì

$$\sum_{v \in V} deg^{-}(v) = \sum_{v \in V} deg^{+}(v) = \frac{1}{2} \sum_{v \in V} deg(v) = |E|$$

Đường đi, chu trình và tính liên thông của đồ thị

Đường đi

Đường đi P độ dài n, từ đỉnh u đến đỉnh v, trên đồ thị G=(V,E) là dãy

$$P : x_0, x_1, \cdots, x_n$$

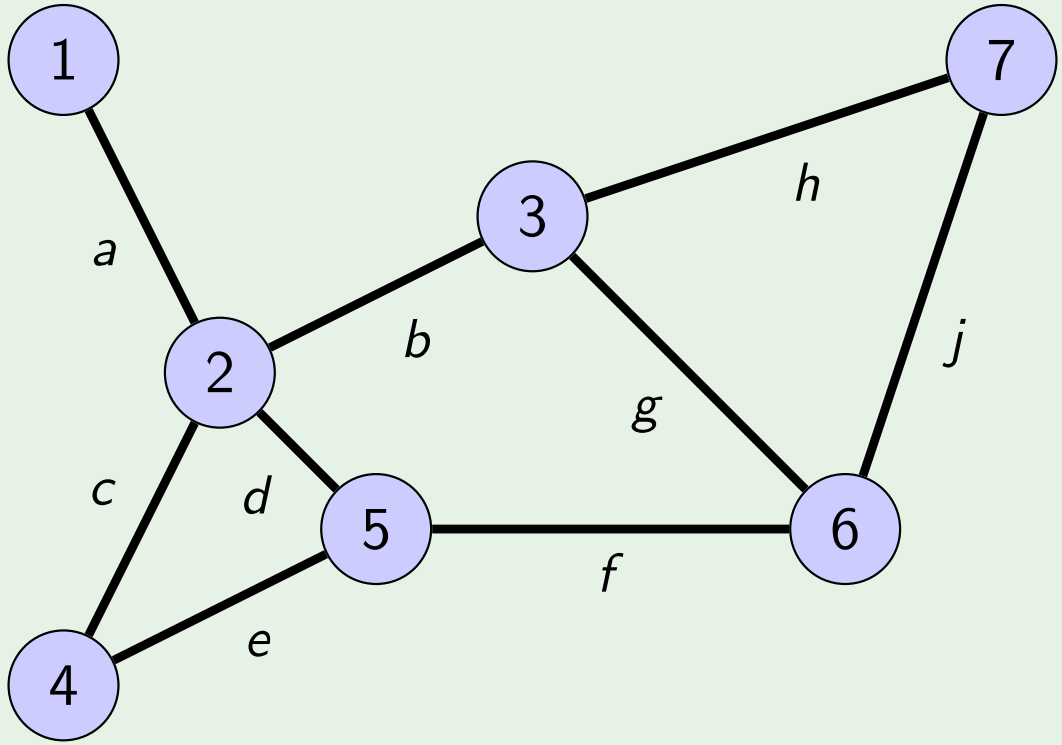
trong đó $x_0 \equiv u, x_n \equiv v, (x_i, x_{i+1}) \in E, i = 0, 1, \cdots, n-1$. Đỉnh u đc gọi là *đỉnh đầu*, còn đỉnh v đc gọi là *đỉnh cuối* của đường đi P.

Ta có thêm các định nghĩa về hai loại đường đi như sau

- Đường đi P đc gọi là *đường đi đơn* nếu không có đỉnh nào bị lặp trên nó.
- Đường đi P đc gọi là *đường đi cơ bản* nếu không có cạnh nào bị lặp trên nó.

Đường đi, chu trình và tính liên thông của đồ thị (tiếp)

Ví dụ về đường đi



Dãy các đỉnh 1,2,3,6,7 là *đường đi đơn* từ đỉnh 1 đến đỉnh 7 qua các cạnh a,b,g,j.

Dãy các đỉnh 1,2,4,5,2,3,7 là *đường đi cơ bản* từ đỉnh 1 đến đỉnh 7 qua các cạnh a,c,e,d,b,h.

Đường đi, chu trình và tính liên thông của đồ thị (tiếp)

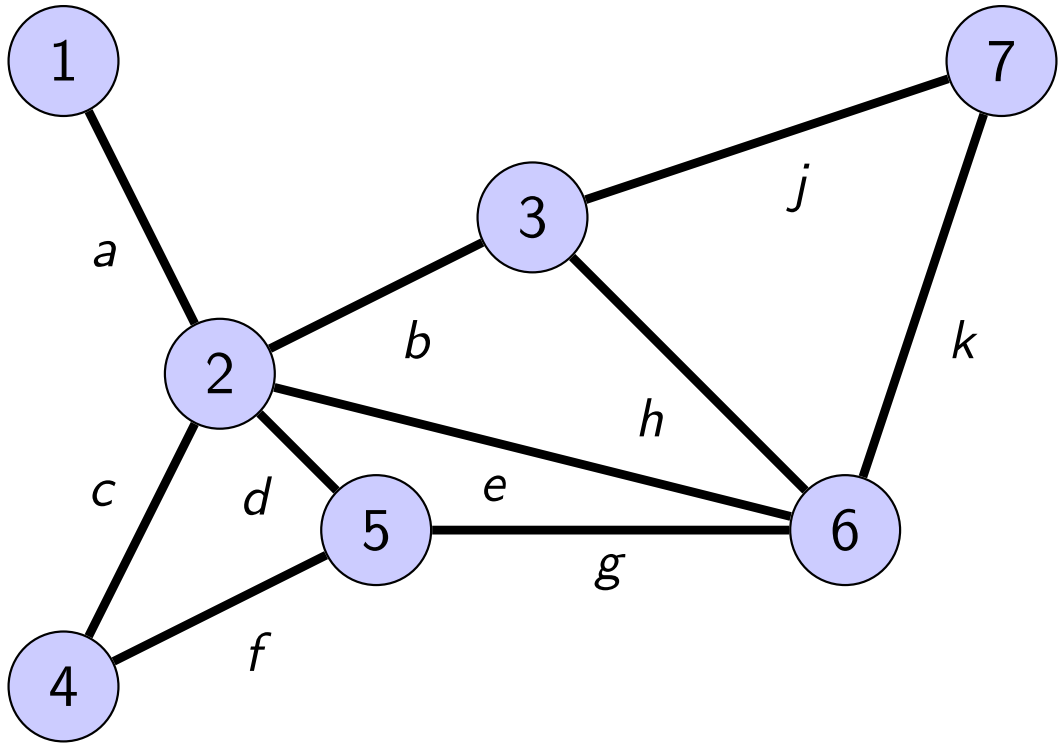
Chu trình

Đường đi cơ bản có đỉnh đầu trùng với đỉnh cuối (tức là $u \equiv v$) được gọi là *chu trình*.

Chu trình đơn

Chu trình được gọi là *chu trình đơn* nếu ngoại trừ đỉnh đầu và đỉnh cuối trùng nhau, không có đỉnh nào lặp lại.

Đường đi, chu trình và tính liên thông của đồ thị (tiếp)



Dãy 4,2,3,6,2,5,4 là chu trình qua các cạnh c,b,h,e,d,f
Dãy 4,2,3,6,5,4 là chu trình đơn qua các cạnh c,b,h,g,f

Đường đi, chu trình và tính liên thông của đồ thị (tiếp)

Liên thông (connected)

Đồ thị vô hướng G=(V,E) được gọi là *liên thông* nếu luôn tìm được đường đi nối hai đỉnh u, v $\in$ V bất kỳ của nó.

Thành phân liên thông (connected component)

Đồ thị con liên thông cực đại của đồ thị vô hướng G được gọi là *thành phân liên thông* của nó.

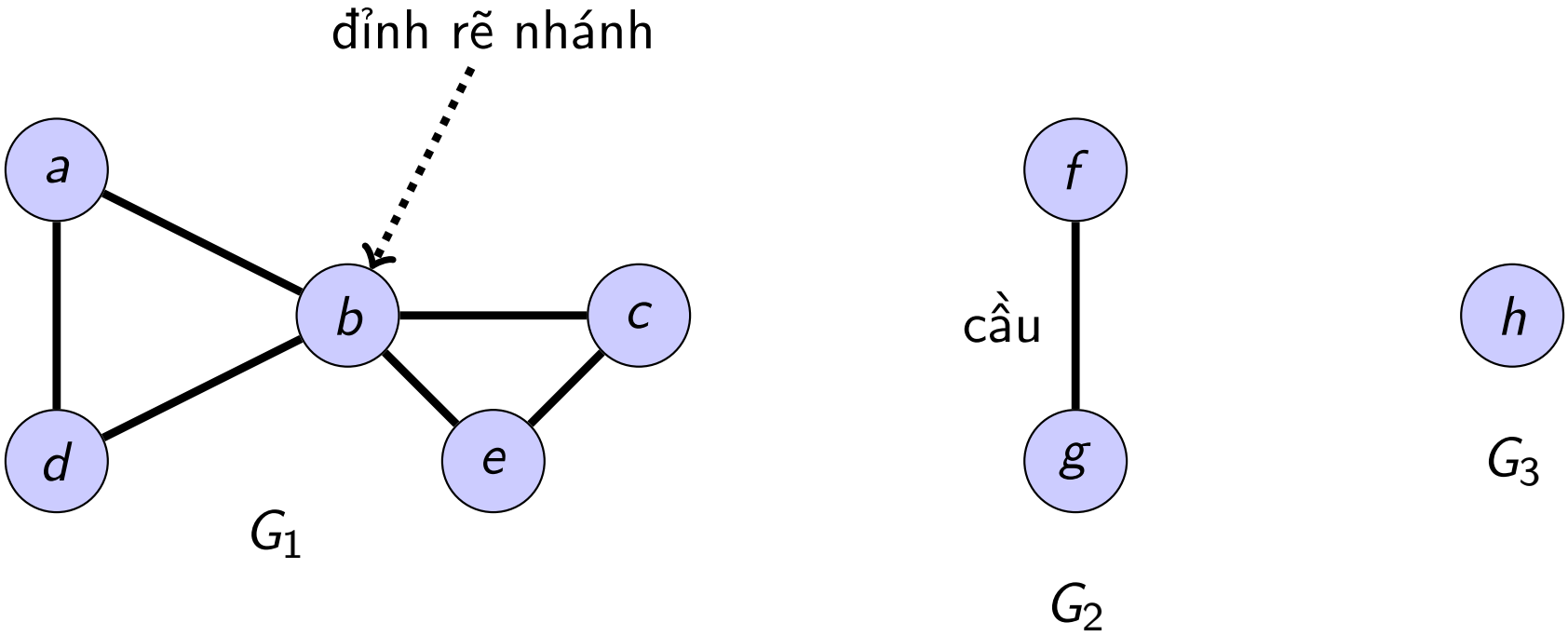
Đỉnh rẽ nhánh (cut vertex)

Đỉnh rẽ nhánh là đỉnh mà việc loại bỏ nó làm tăng thành phân liên thông của đồ thị.

Cầu (bridge)

Cầu là cạnh mà việc loại bỏ nó làm tăng số thành phần liên thông của đồ thị.

Đường đi, chu trình và tính liên thông của đồ thị (tiếp)



Đồ thị G có 3 thành phần liên thông G_1, G_2 và G_3

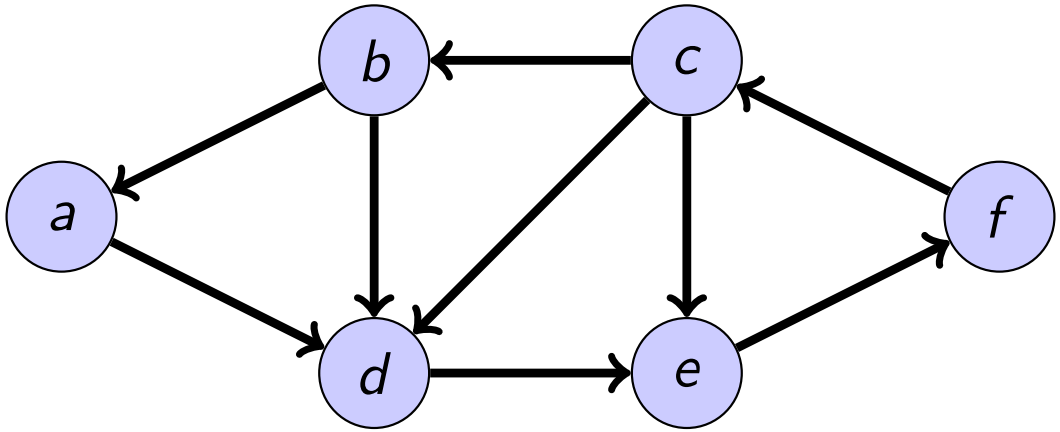
Đường đi, chu trình và tính liên thông của đồ thị (tiếp)

Liên thông trong đồ thị có hướng

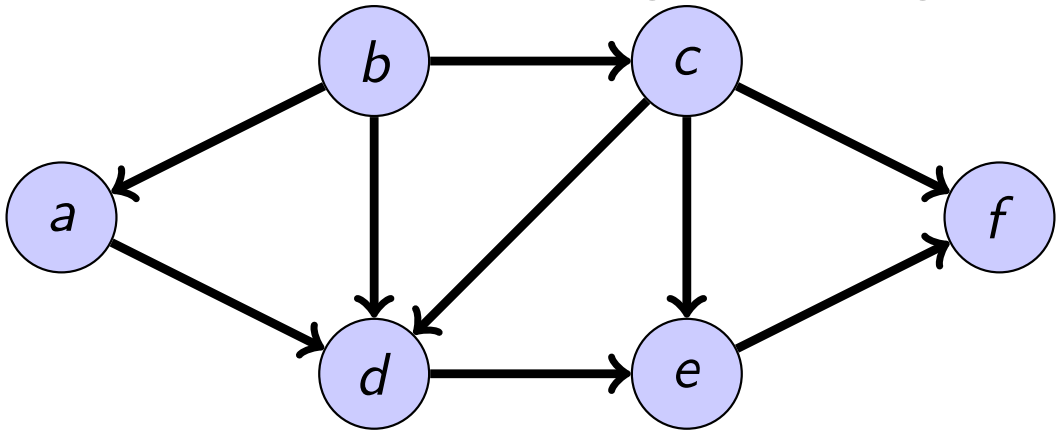
Đồ thị có hướng G=(V,E) được gọi là

- *liên thông mạnh* (strongly connected) nếu như luôn tìm được đường đi nối hai đỉnh bất kỳ của nó.
- *liên thông yếu* (weakly connected) nếu như đồ thị vô hướng thu được từ nó bởi việc bỏ đi hướng của tất cả các cạnh là đồ thị vô hướng liên thông.

Đường đi, chu trình và tính liên thông của đồ thị (tiếp)



Đồ thị G là đồ thị có hướng liên thông mạnh



Đồ thị G là đồ thị có hướng liên thông yếu

Các phép toán cơ bản của ADT đồ thị

Giả sử cho đồ thị G=(V,E)

- khởi tạo đồ thị (create)
- hủy (destroy)
- lấy số cạnh của đồ thị |E|
- lấy số đỉnh đỉnh của đồ thị |V|
- đồ thị là vô hướng hay có hướng
- bổ sung cạnh (insert an edge)
- loại bỏ cạnh (remove an edge)
- có cạnh nối giữa hai đỉnh
- duyệt đỉnh kề của một đỉnh cho trước

Các thao tác phức tạp hơn với ADT đồ thị

Giả sử cho đồ thị G=(V,E)

- Tìm giá trị của một số đặc trưng số của đồ thị (số thành phần liên thông, ...)
- Tìm một số tập con cạnh đặc biệt (cặp cạnh ghép, bè, chu trình, cây khung ...)
- Tìm một số tập con đỉnh đặc biệt (phủ đỉnh, phủ cạnh, tập độc lập, ...)
- Trả loài truy vấn về một số tính chất của đồ thị (song liên thông, phẳng ...)
- Các bài toán tối ưu trên đồ thị : Bài toán cây khung nhỏ nhất, bài toán đường đi ngắn nhất, bài toán luồng cựu đại trong mạng, ...

Biểu diễn đồ thị

Có nhiều cách biểu diễn cấu trúc dữ liệu đồ thị trong máy tính, ta chọn ba cách sẽ đc trình bầy lần lượt như sau

- Sử dụng ma trận
- Danh sách kề
- Danh sách cạnh

Có hai vấn đề cần quan tâm khi chọn cách biểu diễn đồ thị

- Dung lượng bộ nhớ mà cách biểu diễn đòi hỏi
- Thời gian - độ phức tạp tính toán - để đáp ứng các yêu cầu thường xuyên khi xử lý đồ thị như 'liệt kê đỉnh kề đỉnh v', 'có cạnh nối giữa u và v' ...

Biểu diễn đồ thị bởi ma trận

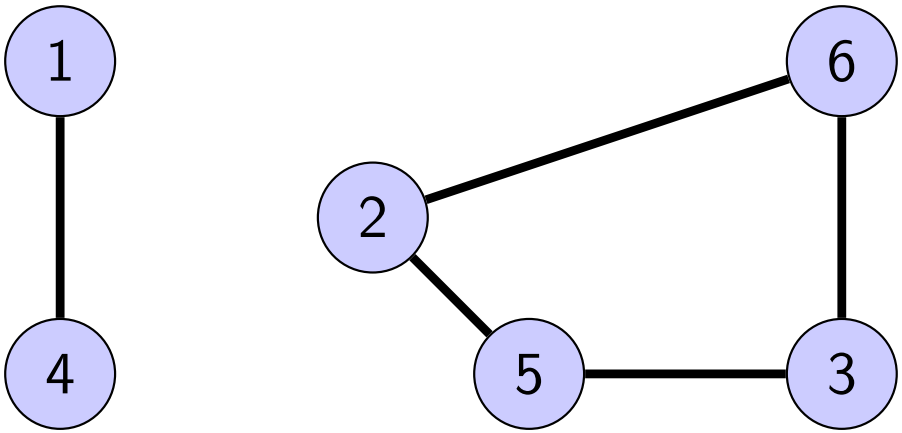
Ma trận kề (adjacency matrix)

Cho dồ thị G=(V,E) thì ma trận kề biểu diễn đồ thị G là ma trận A khích thước $|V| \times |V|$ trong đó mỗi cạnh e=(u,v) tương ứng với một phần tử khác không ở vị trí (u,v) của ma trận, nghĩa là

$$A(u,v) = \begin{cases} 1, & (u,v) \in E \\ 0, & (u,v) \notin E \end{cases}$$

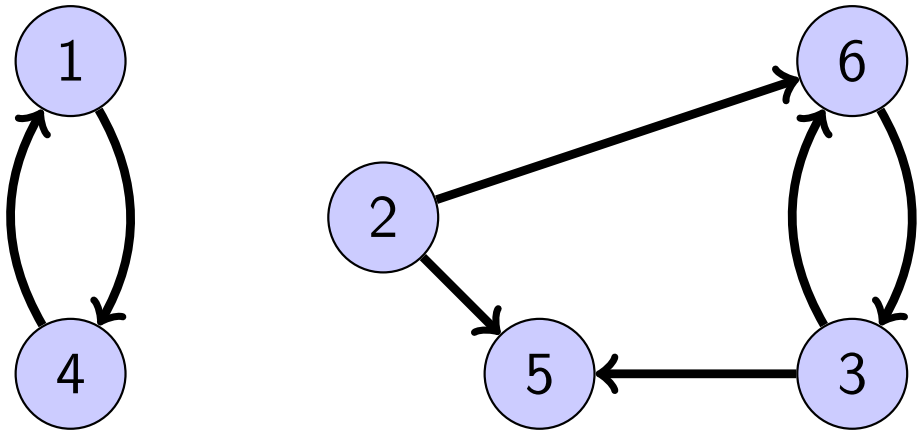
Bộ nhớ đòi hỏi kích thước là : $\Theta(|V|^2)$

Biểu diễn đồ thị bởi ma trận (tiếp)



$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Đồ thị vô hướng G và ma trận kề A tương ứng



$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Đồ thị có hướng G và ma trận kề A tương ứng

Biểu diễn đồ thị bởi ma trận (tiếp)

Tính chất của ma trận kề

Giả sử A là ma trận kề của đồ thị vô hướng G

- A là ma trận đối xứng $A = A^T$
- deg(v) là tổng các phần tử khác không trên hàng thứ v của A

Giả sử A là ma trận kề của đồ thị có hướng G

- Bán bậc vào của đỉnh v : $deg^-(v)$ là tổng các phần tử khác không trên cột thứ v của A
- Bán bậc ra của đỉnh v : $deg^+(v)$ là tổng các phần tử khác không trên hàng thứ v của A

Ưu điểm nổi bật của ma trân kề là để biết (u,v) có là cạnh của đồ thị hay không, ta chỉ mất thời gian O(1).

Biểu diễn đồ thị bởi ma trận (tiếp)

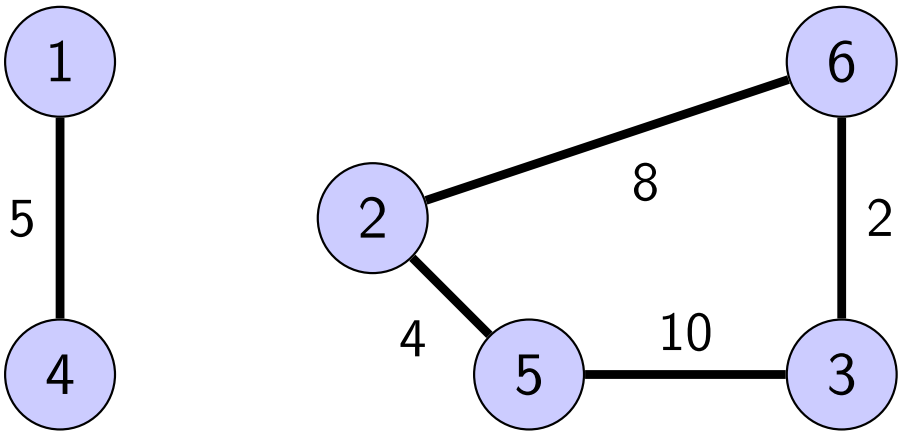
Ma trận trọng số

Trong trường hợp đồ thị có trọng số trên cạnh, ta có thể dùng ma trận trọng số C kích thước |V|x|V| với trọng số c mỗi cạnh e=(u,v) tương ứng với giá trị của phần tử ở vị trí (u,v)

$$C(u,v) = \begin{cases} c(u,v), & (u,v) \in E \\ \theta, & (u,v) \notin E \end{cases}$$

trong đó θ là giá trị đặt biêt chỉ cặp (u,v) không là cạnh. Thường đc gán các giá trị sau : $-\infty, +\infty, 0$

Biểu diễn đồ thị bởi ma trận (tiếp)



$$C = \begin{bmatrix} 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 8 \\ 0 & 0 & 0 & 0 & 10 & 2 \\ 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 10 & 0 & 0 & 0 \\ 0 & 8 & 2 & 0 & 0 & 0 \end{bmatrix}$$

Đồ thị có trọng số G và ma trận trọng số C tương ứng với $\theta = 0$

Biểu diễn đồ thị bởi ma trận (tiếp)

Ma trận liên thuộc đỉnh/cạnh

Xét đồ thị G=(V,E) trong đó

- $V = \{1, 2, ..., n\}$ là tập đỉnh
- $E = \{e_1, e_2, ..., e_m\}$ là tập cung

là *đơn đồ thị có hướng*.
Ma trận liên thuộc đỉnh cạnh A biểu diễn G có kích thước $|V| \times |E|$ được xây dựng theo quy tắc

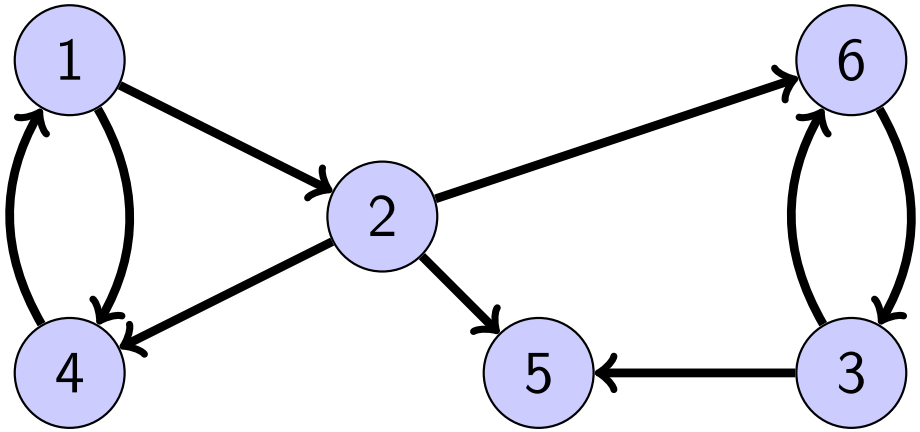
$$a_{ij} = \begin{cases} 1, & \text{nếu đỉnh i là đỉnh đầu của cung j} \\ -1, & \text{nếu đỉnh i là đỉnh cuối của cung j} \\ 0, & \text{nếu đỉnh i không là đầu mút của cung j} \end{cases}$$

trong đó $i = 1, 2, ..., n$ và $j = 1, 2, ..., m$

[1]

[1] Dễ dàng tính $deg^-(v)$ và $deg^+(v)$

Biểu diễn đồ thị bởi ma trận (tiếp)



Đơn đồ thị có hướng G

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccccc} (1,2) & (1,4) & (2,5) & (2,6) & (3,5) & (3,6) & (4,1) & (6,3) \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 & 0 & 1 \end{pmatrix} \end{array}$$

Ma trận liên thuộc đỉnh/cạnh biểu diễn đồ thị G

Biểu diễn đồ thị bằng danh sách kề

Biểu diễn đồ thị bằng danh sách kề

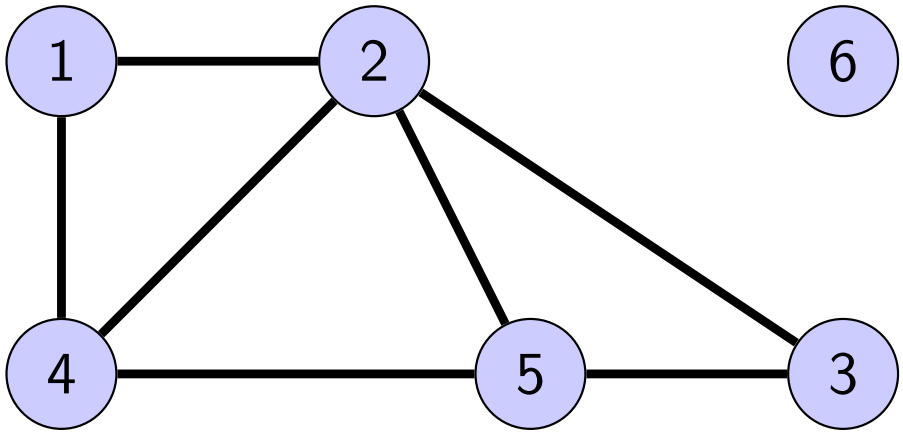
Khi cần biểu diễn đồ thị thưa (sparse graphs) nghĩa là $|E| = O(|V|)$, ta thường sử dụng danh sách kề. Danh sách kề của đỉnh v là tập

$$Adj(v) = \{u : (v, u) \in E\}$$

2

[2] Theo Bruno R. Preiss trong cuốn Data Structures and Algorithms with Object-Oriented Design Patterns in C++. Với đồ thị đặc (dense graph) thì $|E| = \Theta(|V|^2)$ còn với đồ thị thưa $|E| = O(|V|)$

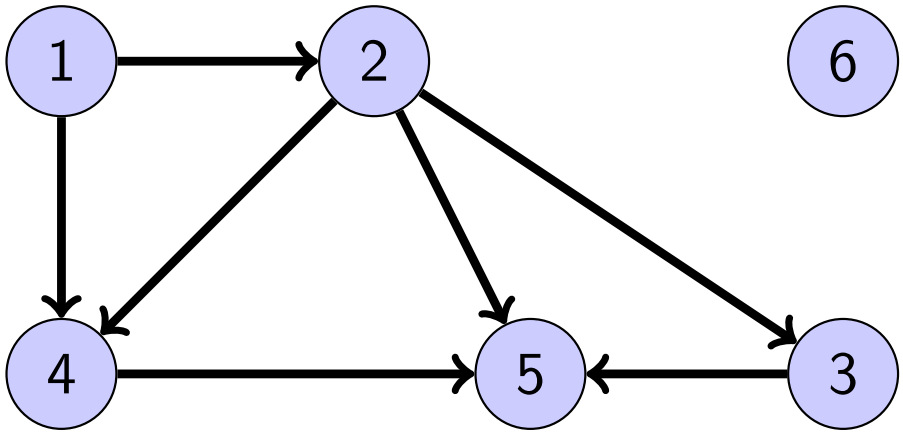
Biểu diễn đồ thị bằng danh sách kề (tiếp)



Đồ thị vô hướng G_1

v	Adj(v)
1	2,5
2	1,3,4,5
3	2,5
4	1,2,5
5	2,3,4
6	

Biểu diễn đồ thị bằng danh sách kề (tiếp)

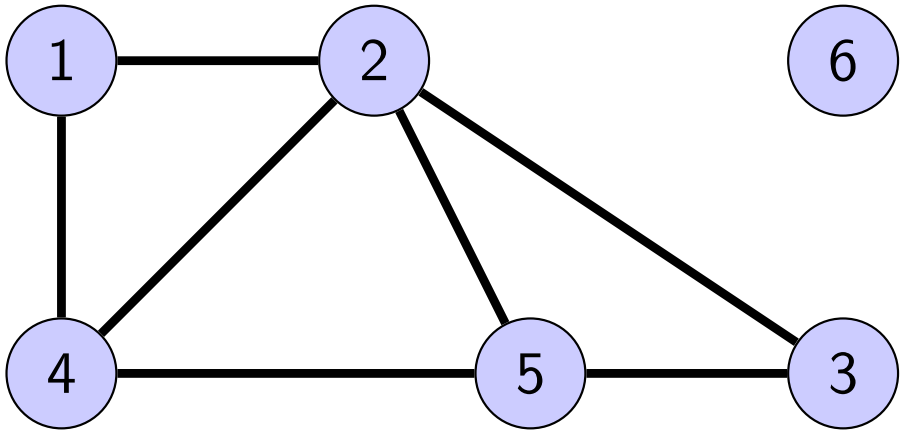


Đồ thị có hướng G_2

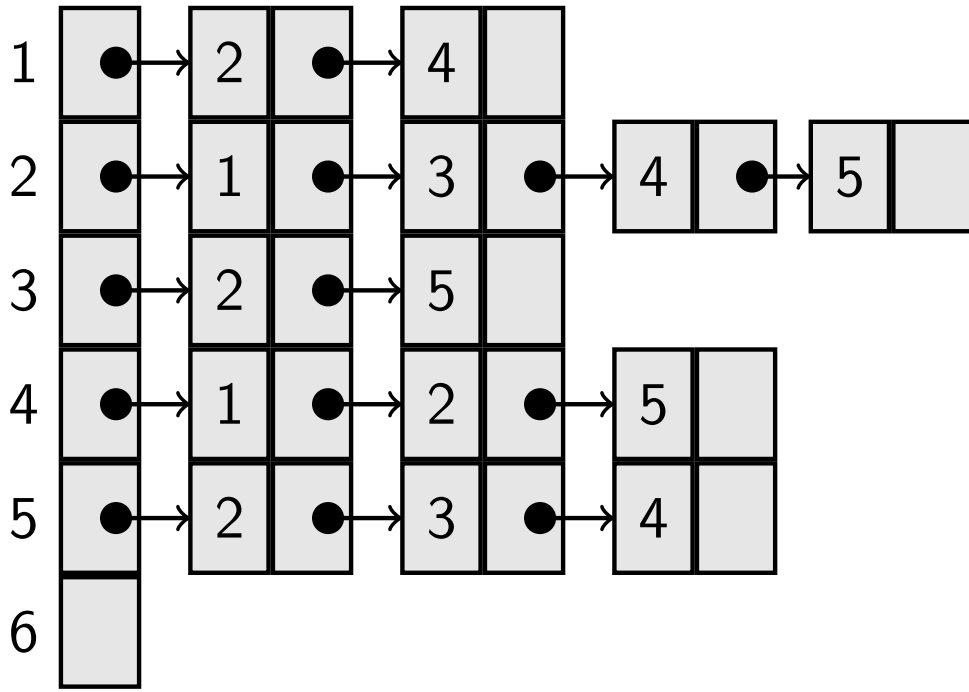
v	Adj(v)
1	2,4
2	3,4,5
3	5
4	5
5	
6	

Biểu diễn đồ thị bằng danh sách kề (tiếp)

Để cài đặt danh sách kề, ta có thể sử dụng n danh sách liên kết, mỗi danh sách chứa đỉnh kề của đỉnh v $\in$ V trong đồ thị G=(V,E)



Đồ thị vô hướng G



Biểu diễn đồ thị bằng danh sách kề (tiếp)

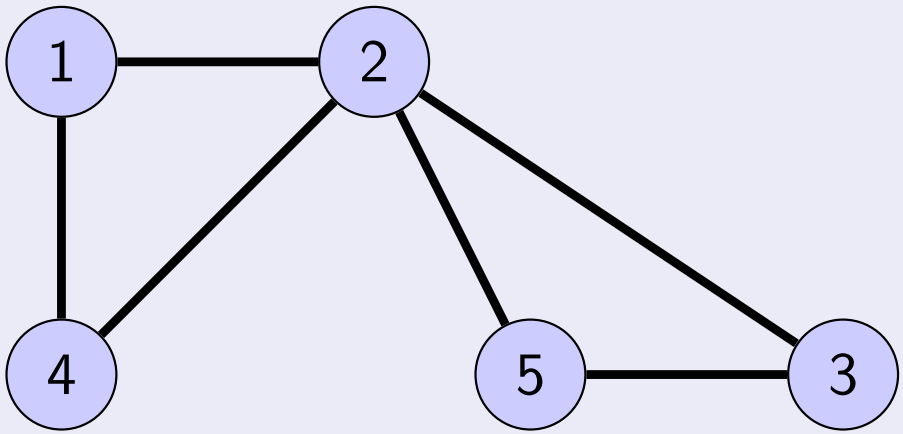
Mã nguồn C

```
#define MAXVERTICES 500
typedef struct node *nodeptr;
typedef struct node {
    int vertex;
    nodeptr link;
} node;
nodeptr graph[MAXVERTICES];
```

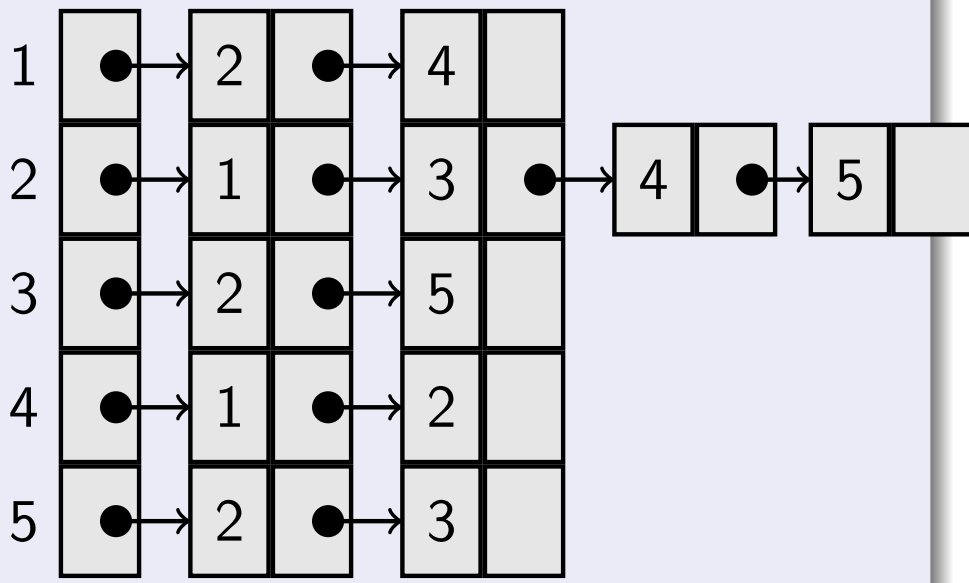
Biểu diễn đồ thị bằng danh sách kề (tiếp)

Phân tích yêu cầu bộ nhớ

- Đối với đồ thị vô hướng : giả sử mỗi con trỏ đòi hỏi a đơn vị bộ nhớ và mỗi nút của danh sách đòi hỏi b đơn vị bộ nhớ



Đồ thị vô hướng G

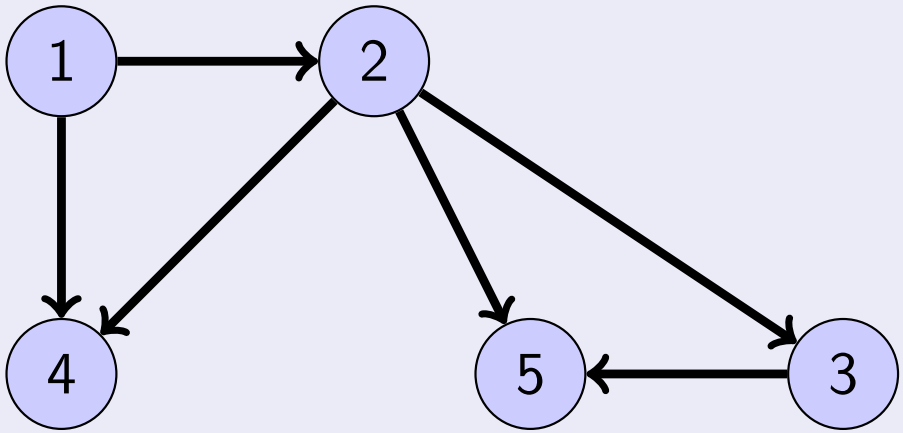


Khi đó biểu diễn đồ thị vô hướng bởi danh sách kề đòi hỏi $a|V| + 2b|E|$ đơn vị bộ nhớ

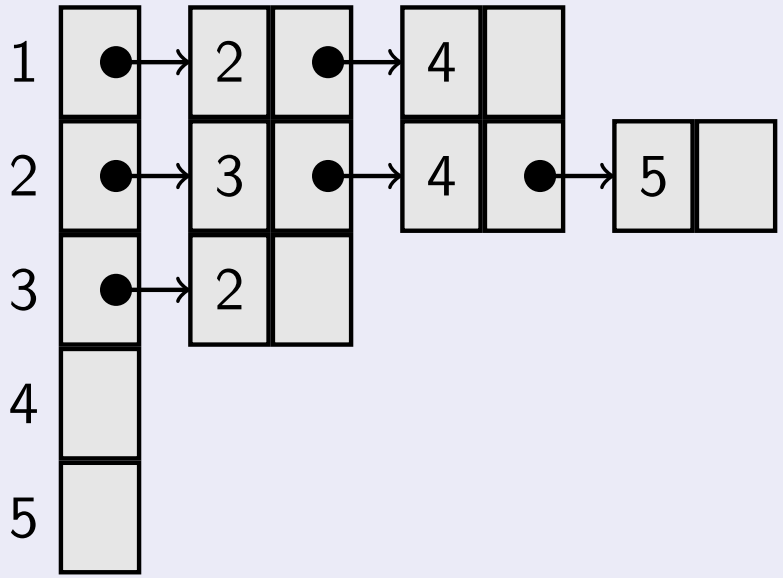
Biểu diễn đồ thị bằng danh sách kề (tiếp)

Phân tích yêu cầu bộ nhớ (tiếp)

- Đối với đồ thị có hướng : giả sử mỗi con trỏ đòi hỏi a đơn vị bộ nhớ và mỗi nút của danh sách đòi hỏi b đơn vị bộ nhớ



Đồ thị có hướng G



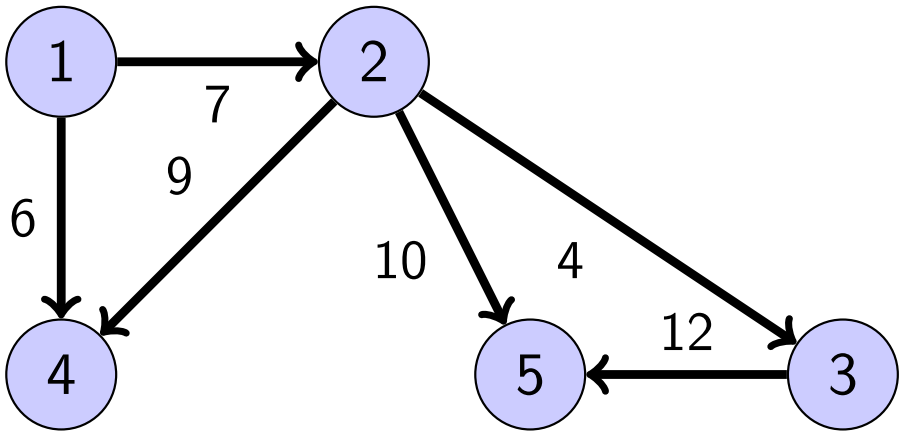
Khi đó biểu diễn đồ thị vô hướng bởi danh sách kề đòi hỏi a|V| + b|E| đơn vị bộ nhớ

Tổng cộng bộ nhớ cần để biểu diễn đồ thị bởi danh sách kề là $\Theta(|V| + |E|)$ nhỏ hơn nhiều so với cách biểu diễn ma trận kề $|V|^2$

Biểu diễn đồ thị bằng danh sách cạnh

Danh sách cạnh

Với mỗi cạnh e=(u,v) ta cất giữ dau[e] = u, cuoi[e] = v đồng thời c[e] = trọng số cạnh e (nếu là đồ thị có trọng số).



Đồ thị có hướng có trọng số G

e	dau[e]	cuoi[e]	c[e]
1	1	2	7
2	1	4	6
3	2	3	4
4	2	4	9
5	2	5	10
6	3	5	12

Danh sách cạnh tương ứng của G

Biểu diễn đồ thị (tiếp)

Các thao tác cơ bản thường gặp khi xử lý đồ thị

- incidentEdges(v) - duyệt các đỉnh kề của đỉnh v
- areAdjacent(v,w) - trả lại giá trị đúng khi v,w kề nhau
- insertVertex(z) - bổ sung đỉnh z
- insertEdge(v,w,e) - bổ sung cạnh e=(v,w)
- removeVertex(v) - loại bỏ đỉnh v
- removeEdge(e) - loại bỏ cạnh e

Biểu diễn đồ thị (tiếp)

Bảng đánh giá bộ nhớ và thời gian thực hiện các thao tác tương ứng đồ thị G=(V,E) với $|V| = n$, $|E| = m$

Thao tác	Danh sách cạnh	Danh sách kề	Ma trận kề
Bộ nhớ	n+m	n+m	n^2
incidentEdges(v)	m	deg(v)	m
areAdjacent(v,w)	m	min(deg(v),deg(w))	1
insertVertex(z)	1	1	n^2
insertEdge(v,w,e)	1	1	1
removeVertex(v)	m	deg(v)	n^2
removeEdge(e)	1	1	1

Các thuật toán duyệt đồ thị

Phép duyệt đồ thị (Graph Traversal hoặc Graph Searching)

Là việc duyệt qua mỗi đỉnh và mỗi cạnh của đồ thị. Ta sẽ xét hai thuật toán cơ bản duyệt đồ thị

- Tìm kiếm theo chiều rộng (Breath First Search - BFS)
- Tìm kiếm theo chiều sâu (Depth First Search - DFS)

Ý tưởng chung

trong quá trình thực hiện thuật toán, mỗi đỉnh ở một trong 3 trạng thái :

- chưa thăm
- đã thăm nhưng chưa duyệt xong
- đã duyệt xong

quá trình duyệt sẽ đc bắt đầu từ một đỉnh v nào đó.

Thuật toán tìm kiếm theo chiều rộng

Đầu vào

Đồ thị G=(V,E) có hướng hoặc vô hướng và đỉnh xuất phát s $\in$ V.

Đầu ra

Với mọi v $\in$ V

- $d[v]$ = khoảng cách từ s đến v
- $\pi[v]$ - đỉnh đi trước v trong đường đi ngắn nhất từ s đến v
- Xây dựng cây BFS gốc tại s chứa các đỉnh đạt đến được từ v

Ta sẽ sử dụng mầu để ghi nhận các trạng thái của đỉnh

- Trắng (white) - chưa thăm
- Vàng (yellow) - đã thăm nhưng chưa duyệt
- Xanh lá cây (green) - đã duyệt xong

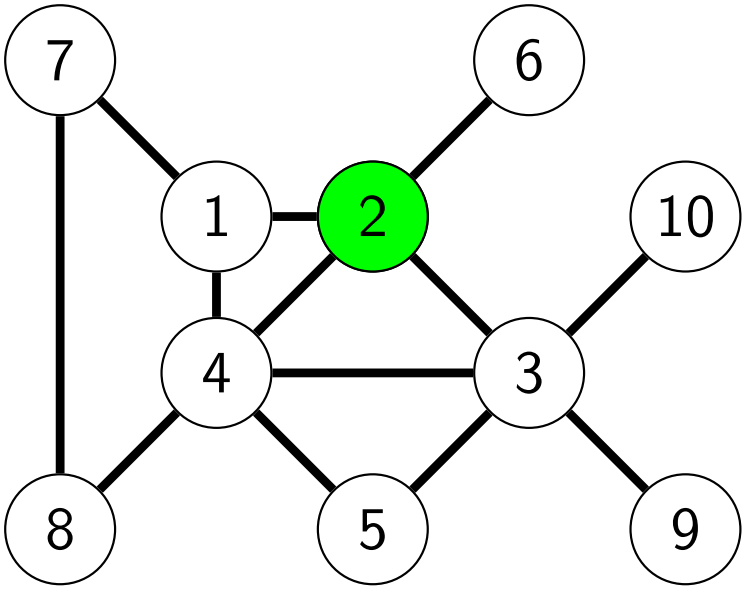
Thuật toán tìm kiếm theo chiều rộng (tiếp)

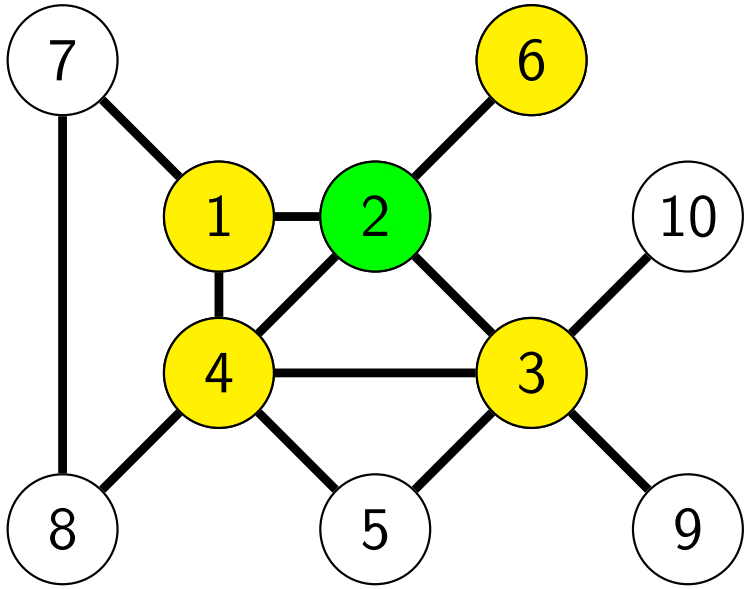
BFS(G,s)

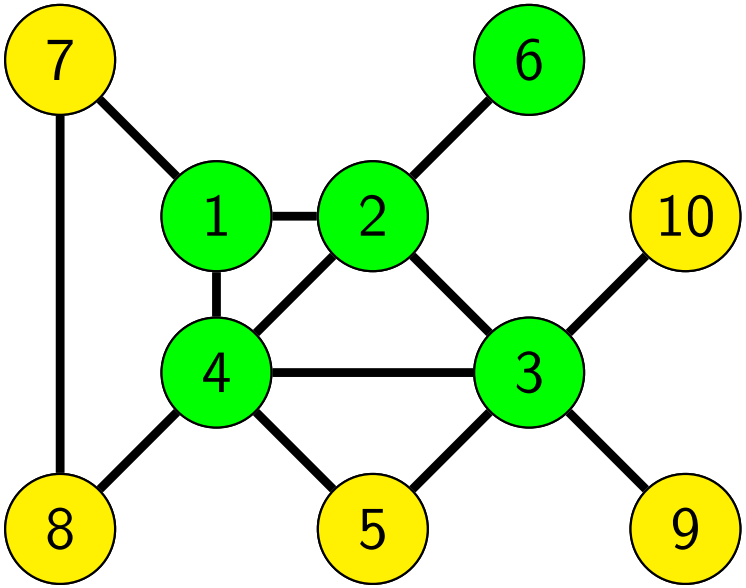
1. **for** với mỗi $u \in V - \{s\}$ **do**
   color[u] $\leftarrow$ white ; d[u] $\leftarrow \infty$; π[u] $\leftarrow$ NULL
   endfor
2. color[s] $\leftarrow$ green; d[u] $\leftarrow$ 0; π[u] $\leftarrow$ NULL
3. Q $\leftarrow \emptyset$; enqueue(Q,s)
4. **while** Q $\neq \emptyset$ **do**
   u $\leftarrow$ dequeue(Q)
   for v $\in$ Adj[u] **do**
   if color[v] = white **then**
   color[v] $\leftarrow$ yellow; d[v] $\leftarrow$ d[u] +1; π[v] $\leftarrow$ u; enqueue(Q,v)
   endif
   endfor
   color[u] $\leftarrow$ green
   endwhile

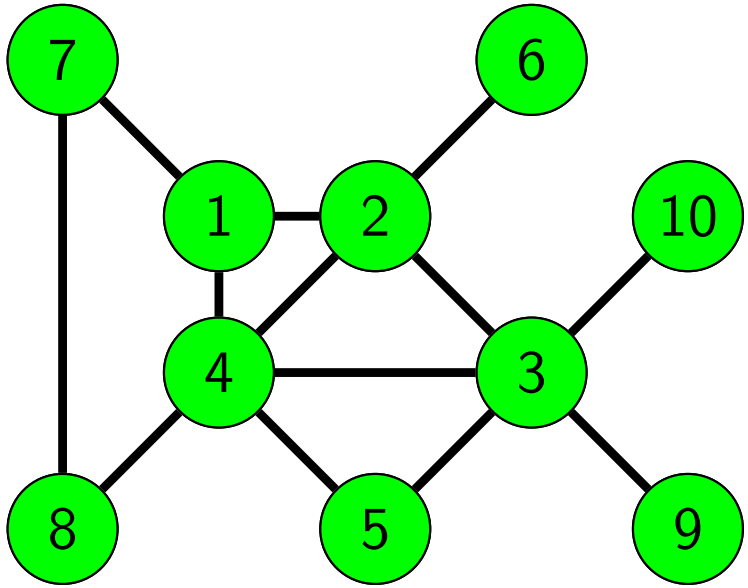
EndProcedure

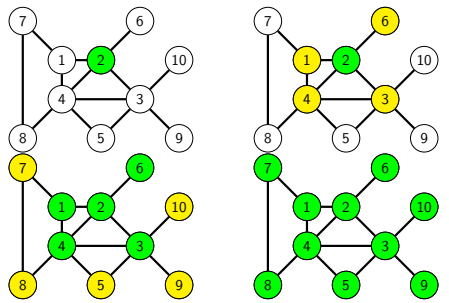
Thuật toán tìm kiếm theo chiều rộng (tiếp)











Điểm xuất phát là đỉnh 2, đỉnh mầu trắng là chưa thăm, đỉnh mẫu xanh lá cây là đã duyệt, đỉnh mẫu vàng đã thăm chưa duyệt. Bốn bước BFS là hình minh họa đc sắp xếp từ trái sang phải, trên xuống dưới. Chú ý, Adj[u] sẽ lấy các đỉnh có chỉ số lần lượt nhỏ đến lớn để xếp vào hàng đợi Q

Thuật toán tìm kiếm theo chiều rộng (tiếp)

Ta có các định lý sau khẳng định tính đúng đắn của BFS

- BFS cho phép thăm tất cả các đỉnh v $\in$ V đến được từ s
- Khi thuật toán kết thúc d[v] cho ta độ dài đường đi ngắn nhất (theo số cạnh) từ đỉnh s đến v
- Với mỗi đỉnh v đạt đến được từ s, π[v] cho ta đường đi trước đỉnh v trong đường đi ngắn nhất từ s đến v.

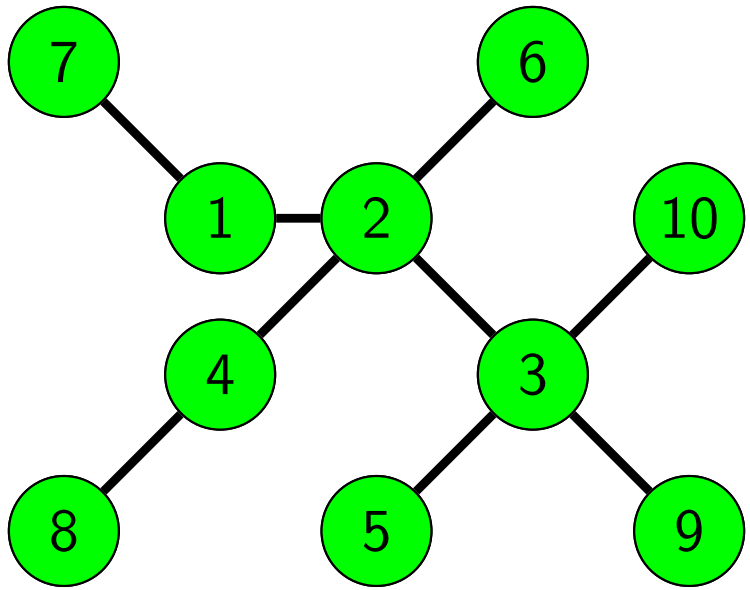
Thuật toán tìm kiếm theo chiều rộng (tiếp)

Cây tìm kiếm chiều rộng - Breath-First-Tree

- Đối với đồ thị G=(V,E) với đỉnh xuất phát s, ký hiệu $G_\pi = (V_\pi, E_\pi)$ là đồ thị với
  - $V_\pi = \{v \in V : \pi[v] \neq NULL\} \cup \{s\}$
  - $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$
- Đồ thị G_π được gọi là cây BFS(s) :
  - V_π chứa tất cả các đỉnh đạt đến được từ s và
  - với mọi $v \in V_\pi$, đường đi từ s đến v trên G_π là đường đi ngăn nhất từ s đến v trên G.
- Các cạnh trong E_π được gọi là các cạnh của cây $|E_\pi| = |V_\pi| - 1$

Thuật toán tìm kiếm theo chiều rộng (tiếp)

Ví dụ cây tìm kiếm chiều rộng có gốc là 2 theo ví dụ minh họa trên



Thuật toán tìm kiếm theo chiều rộng (tiếp)

Độ phức tạp của BFS

- Thuật toán loại bỏ mỗi đỉnh khỏi hàng đợi Q đúng một lần, do đó thao tác dequeue thực hiện |V| lần
- Với mỗi đỉnh, thuật toán duyệt tất cả các đỉnh kề của nó và thời gian xử lý đỉnh kề là hằng số. Như vậy thời gian thực hiện câu lệnh **if** trong vòng lặp **while** là bằng hằng số nhân với số cạnh kề với đỉnh đang xét.
- Do đó tổng số thời gian thực hiện việc duyệt qua tất cả các đỉnh là bằng một hằng số nhân với số cạnh |E|
- Thời gian tông cộng : $O(|V|) + O(|E|) = O(|V| + |E|)$ hay $O(|V|^2)$

Thuật toán tìm kiếm theo chiều sâu

Đầu vào

Cho đồ thị G=(V,E) - đồ thị vô hướng hoặc có hướng, điểm bắt đầu duyệt là đỉnh u

Đầu ra

Với mỗi v $\in$ V, ta có

- d[v] : thời điểm bắt đầu thăm
- f[v] = thời điểm kết thúc thăm
- π[v] : đỉnh từ đó ta đến thăm đỉnh v

Rừng tìm kiếm theo chiều sâu (Forest of Depth-First-Trees)

- $G_\pi = (V, E_\pi)$
- $E_\pi = \{(\pi[v], v) : v \in V \text{ và } \pi[v] \neq NULL\}$

Thuật toán tìm kiếm theo chiều sâu (tiếp)

Thuật toán sử dụng chương trong con DFS-Visit(u) sau đây

DFS-Visit(u)

1. time $\leftarrow$ time + 1
2. d[u] $\leftarrow$ time
3. color[u] $\leftarrow$ yellow
4. **for** mỗi v $\in$ Adj[u] **do**
5. **if** color[v] = white **then** $\pi[v] \leftarrow$ u; DFS-Visit(v); **endif**
6. **endfor**
7. color[u] $\leftarrow$ green
8. time $\leftarrow$ time + 1
9. f[u] $\leftarrow$ time

EndProcedure

Thuật toán tìm kiếm theo chiều sâu (tiếp)

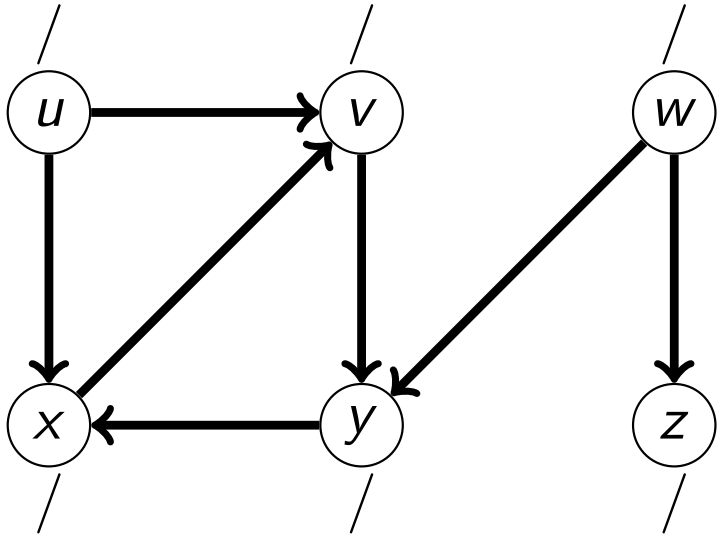
Thuật toán tìm kiếm chiều sâu

DFS(G)

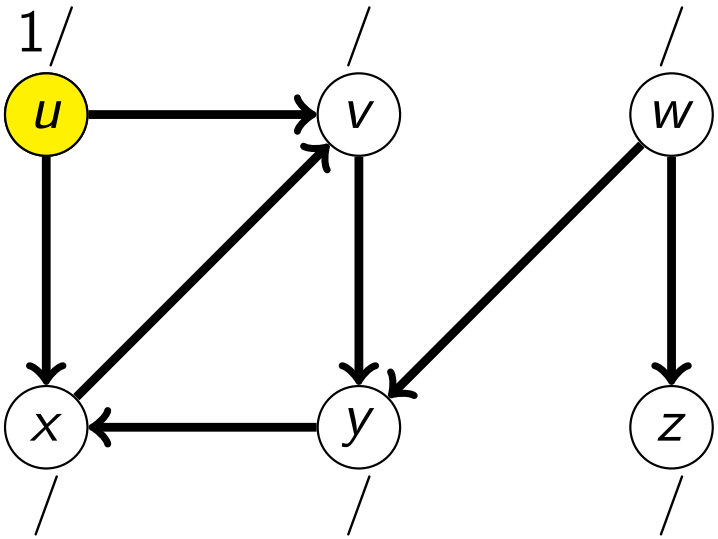
1. **for** u $\in$ V **do**
2. color[u] $\leftarrow$ white; π[u] $\leftarrow$ NULL
3. **endfor**
4. time $\leftarrow$ 0
5. **for** u $\in$ V **do**
6. **if** color[u] = white **then** DFS-Visit(u) **endif**
7. **endfor**

EndProcedure

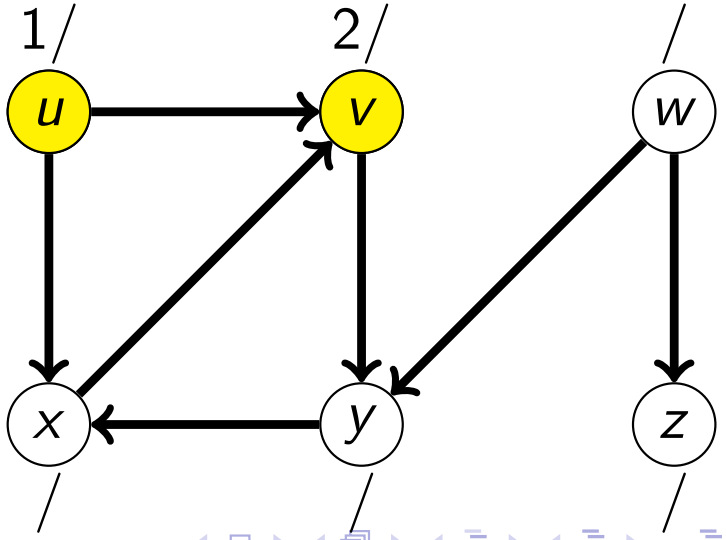
Thuật toán tìm kiếm theo chiều sâu (tiếp)



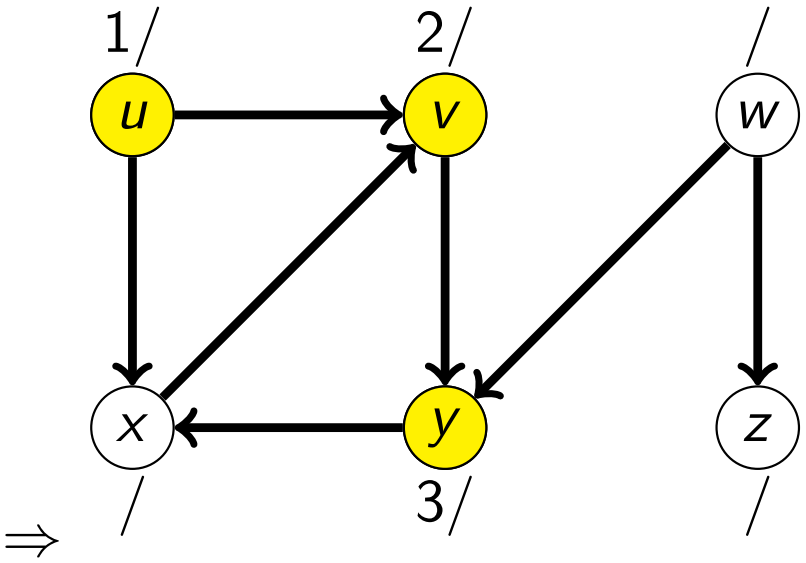
Ban đầu tất cả các nút đc tô trắng, cạnh mỗi nút là d[u]/f[u]

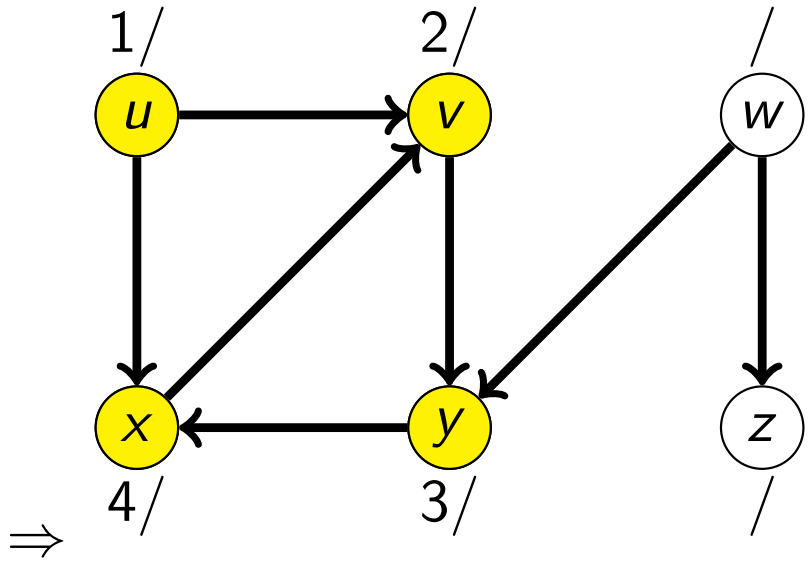


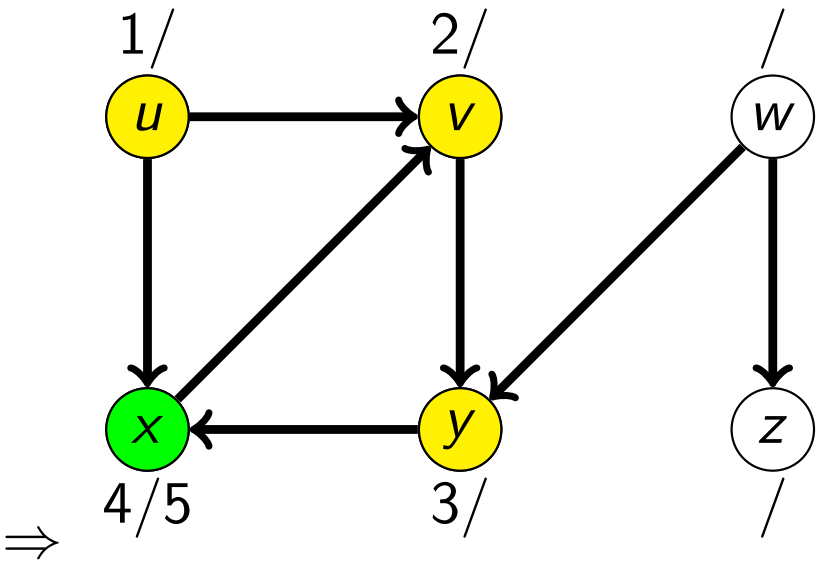
$\Rightarrow$

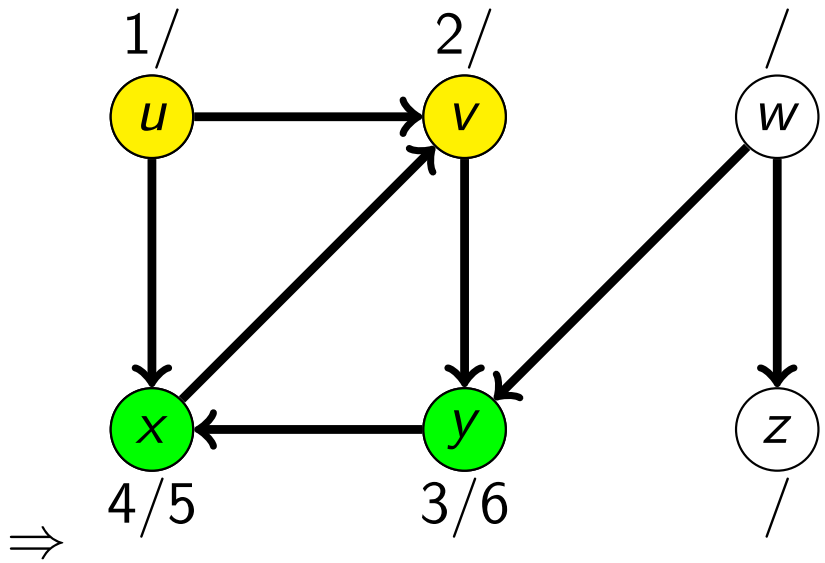


Thuật toán tìm kiếm theo chiều sâu (tiếp)

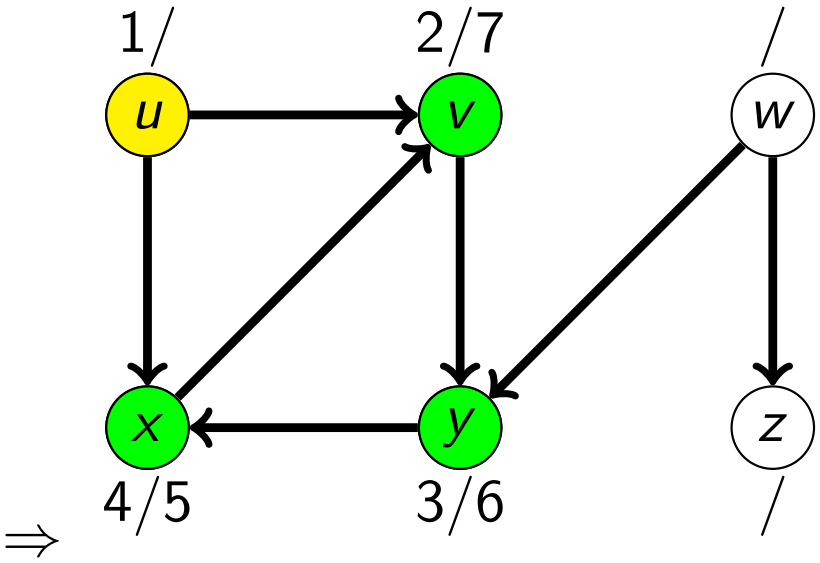


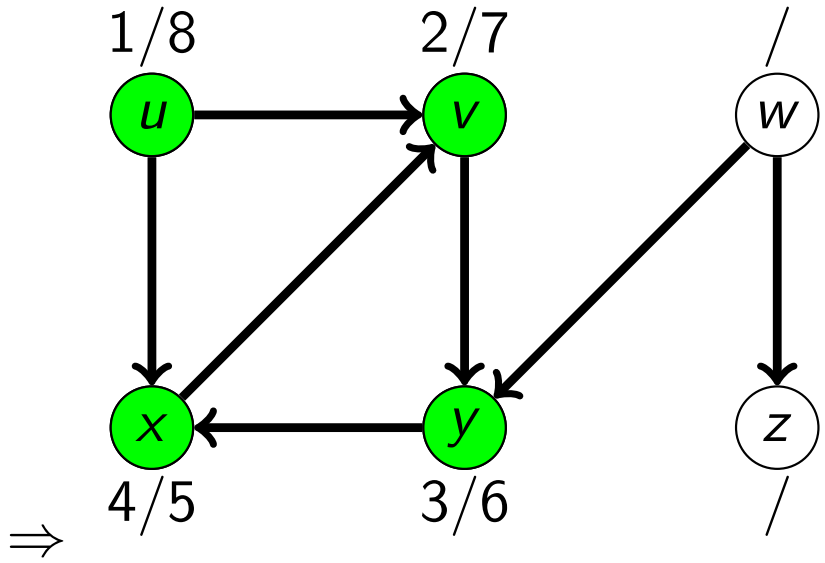


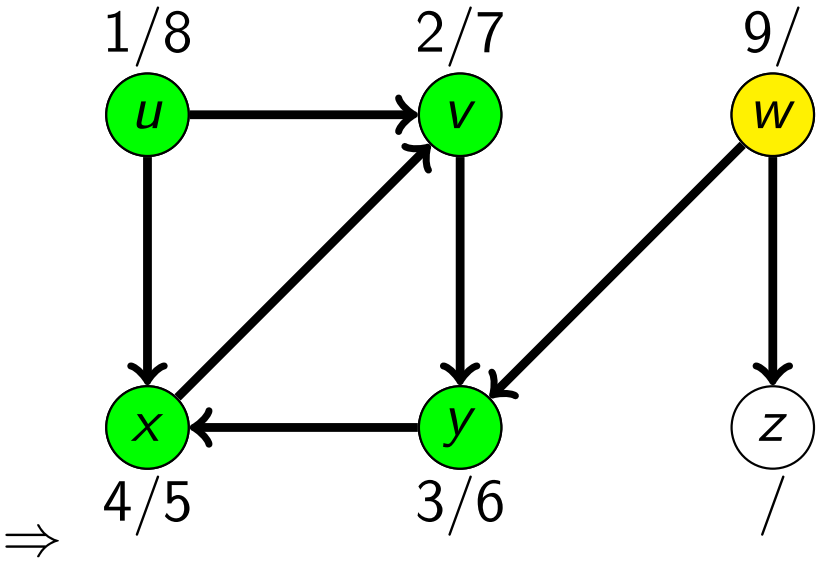


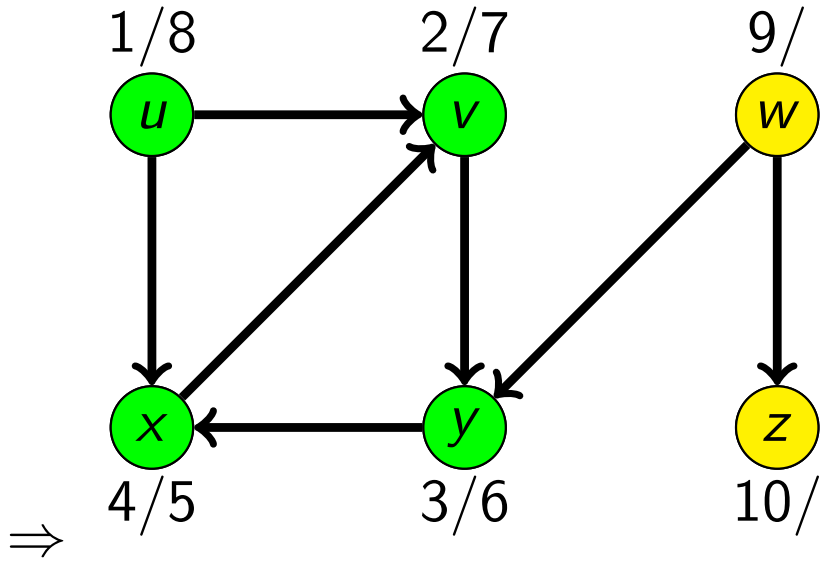


Thuật toán tìm kiếm theo chiều sâu (tiếp)

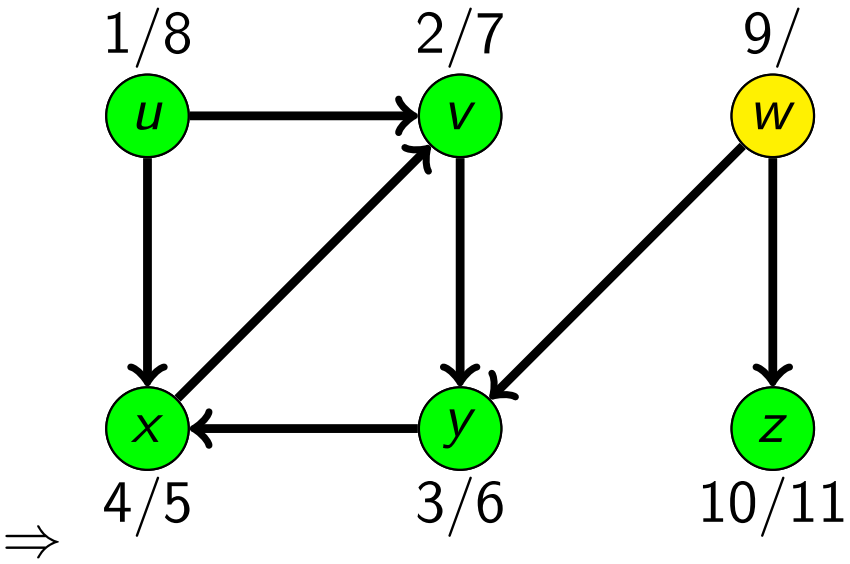


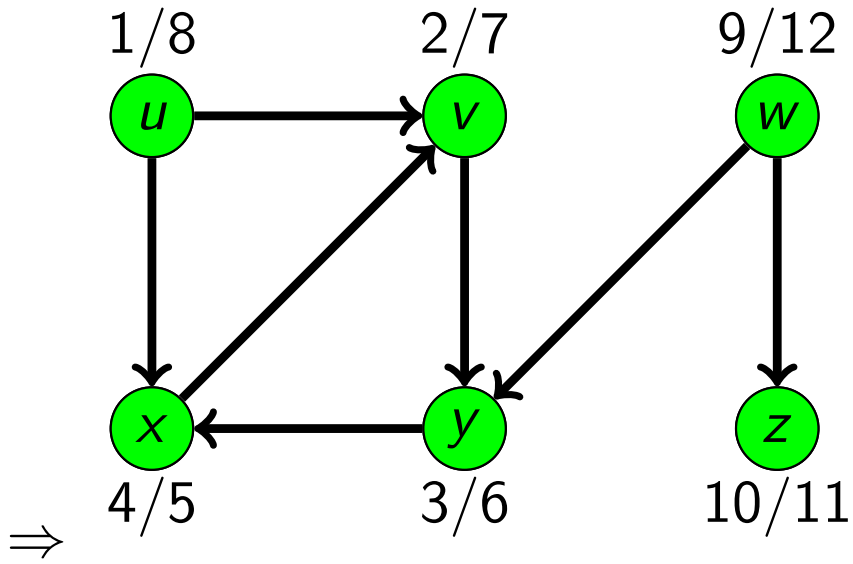






Thuật toán tìm kiếm theo chiều sâu (tiếp)





Thuật toán tìm kiếm theo chiều sâu (tiếp)

Các tính chất của DFS

- Rừng DFS là phụ thuộc vào thứ tự của vòng lặp **for** duyệt đỉnh trong DFS(G) và DFS-Visit(u)
- Để gỡ đệ quy hàm DFS-Visit(u) ta có thể dùng ngăn xếp. Vậy điểm khác biệt cơ bản là các điểm đang được thăm trong DFS được cất giữ vào ngăn xếp thay vì hàng đợi như BFS.
- Các khoảng thời gian thăm (d[v],f[v]) của các đỉnh có cấu trúc lồng nhau

Thuật toán tìm kiếm theo chiều sâu (tiếp)

Độ phức tập của DFS

- Thuật toán thăm mỗi đỉnh v $\in$ V đúng một lần, do đó tổng thời gian thăm đỉnh là $\Theta(|V|)$
- Với mỗi đỉnh v duyệt qua tất cả các đỉnh kề, với mỗi đỉnh kề thực hiện thao tác với thời gian hằng số. Do đó việc duyệt qua tất cả các đỉnh mất thời gian là

$$\sum_{v \in V} |Adj[v]| = \Theta(|E|)$$

- Tổng cộng : $\Theta(|V|) + \Theta(|E|) = \Theta(|V| + |E|)$ hay $\Theta(|V|^2)$

Như vậy, DFS và BFS có cùng độ phức tạp tính toán

Thuật toán tìm kiếm theo chiều sâu (tiếp)

Phân loại cạnh

DFS tạo ra một cách phân loại các cạnh của đồ thị đã cho :

- Cạnh của cây (tree edge) : là cạnh mà theo đó từ một đỉnh ta đến thăm một đỉnh mới
- Cạnh ngược (back edge) : đi từ con cháu (descendent) đến tổ tiên (ancestor)
- Cạnh tới (forward edge) : đi từ tổ tiên đến hậu duệ
- Cạnh vòng (cross edge) : cạnh nối hai đỉnh không có quan hệ họ hàng

Thuật toán tìm kiếm theo chiều sâu (tiếp)

Phân loại cạnh (tiếp)

Để nhận biết cạnh (u,v) thuộc loại cạnh nào, ta dựa vào mầu của đỉnh v khi lần đầu cạnh (u,v) được khảo sát :

- trắng : (u,v) là cạnh của cây
- vàng : (u,v) là cạnh ngược
- xanh : (u,v) là cạnh tới hoặc vòng. Để phân loại rõ hơn (u,v) là cạnh tới hay cạnh vòng, ta dùng định lý cấu trúc lồng nhau để xem quan hệ họ hàng giữa chúng

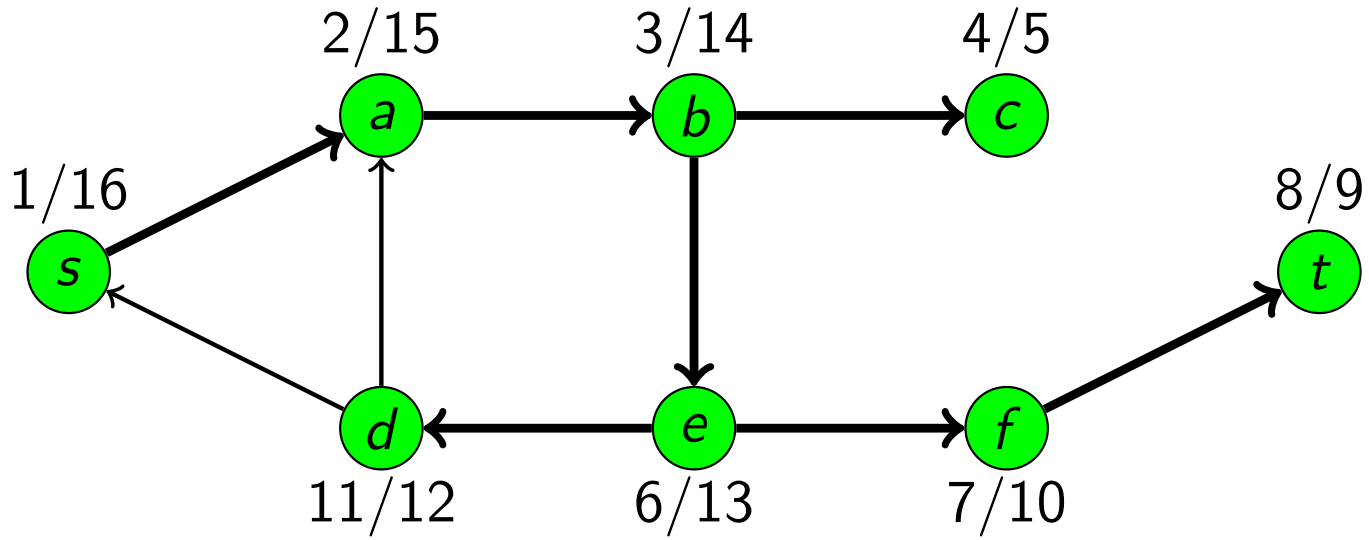
Thuật toán tìm kiếm theo chiều sâu (tiếp)

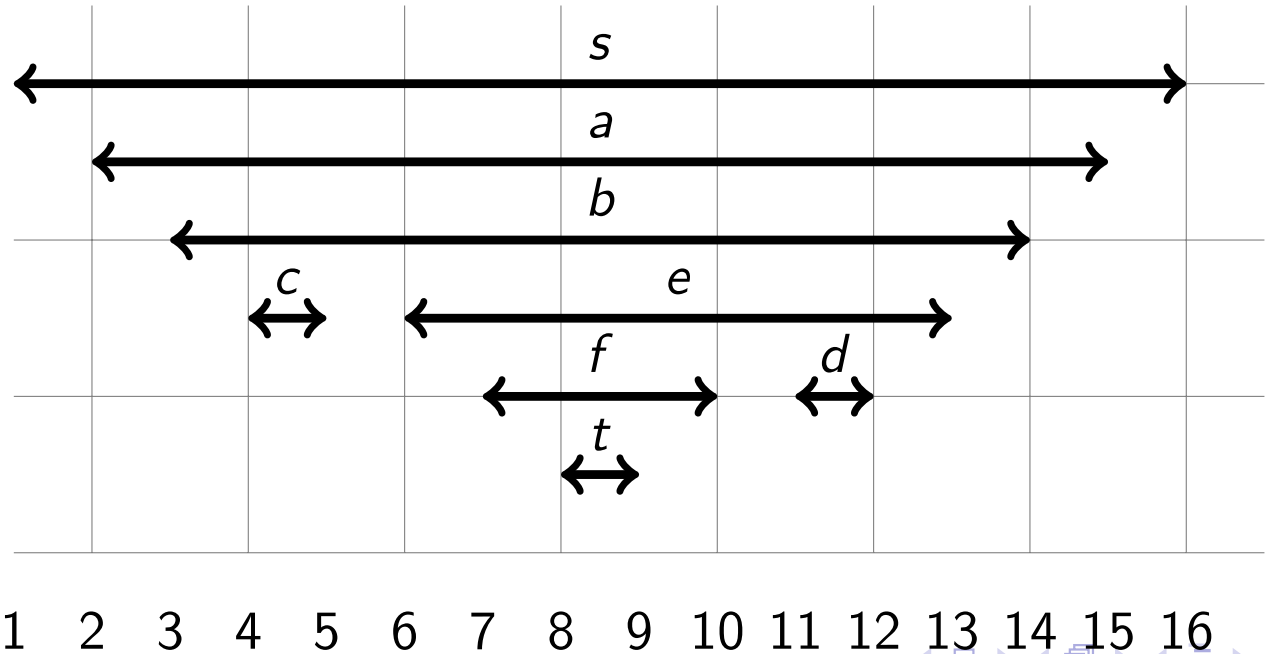
Định lý cấu trúc lồng nhau

Với mọi u,v $\in$ V thuộc G=(V,E) chỉ có thể xảy ra các tình huống sau :

- d[u] < f[u] < d[v] < f[v] hoặc d[v] < f[v] < d[u] < f[u] (nghĩa là hai khoảng thời gian thăm của u và v rời nhau) thì khi đó u và v không có quan hệ tổ tiên - hậu duệ $\Rightarrow$ cạnh vòng
- d[u] < d[v] < f[v] < f[u] (nghĩa là khoảng thời gian thăm của v lồng trong khoảng thời gian thăm của u) thì khi đó v là hậu duệ của u $\Rightarrow$ cạnh tới
- d[v] < d[u] < f[u] < f[v] (nghĩa là khoảng thời gian thăm của u là lồng trong khoảng thời gian thăm của v) thì khi đó u là hậu duệ của v $\Rightarrow$ cạnh ngược

Thuật toán tìm kiếm theo chiều sâu (tiếp)





Thuật toán tìm kiếm theo chiều sâu (tiếp)

Định lý

Đối với đồ thị vô hướng G=(V,E) thuật toán tìm kiếm chiều sâu DFS chỉ sản sinh ra hai loại cạnh

- Cạnh của cây
- Cạnh ngược

Lưu ý là nhiều ứng dụng của DFS chỉ đòi hỏi nhận biết hai loại cạnh này.

Bài toán cây khung nhỏ nhất

Định nghĩa bài toán

Cho đồ thị G=(V,E) là đồ thị vô hướng liên thông có trọng số trên các cạnh c[e], e $\in$ E thì cây $T = (V, E_T)$ với $E_T \subset E$ đc gọi là cây khung của G. Độ dài của cây khung T là tổng trọng số trên các cạnh của nó

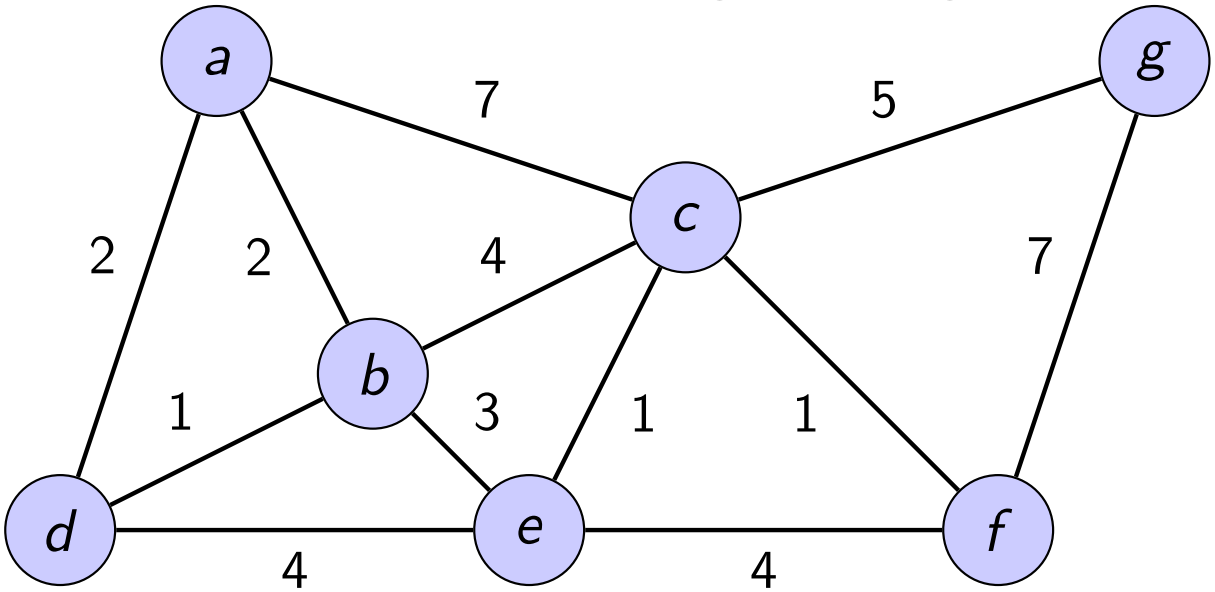
$$c(T) = \sum_{e \in E_T} c[e]$$

Bài toán đặt ra là tìm cây khung T^* sao cho độ dài nhỏ nhất

$$T^* = \arg\min c(T)$$

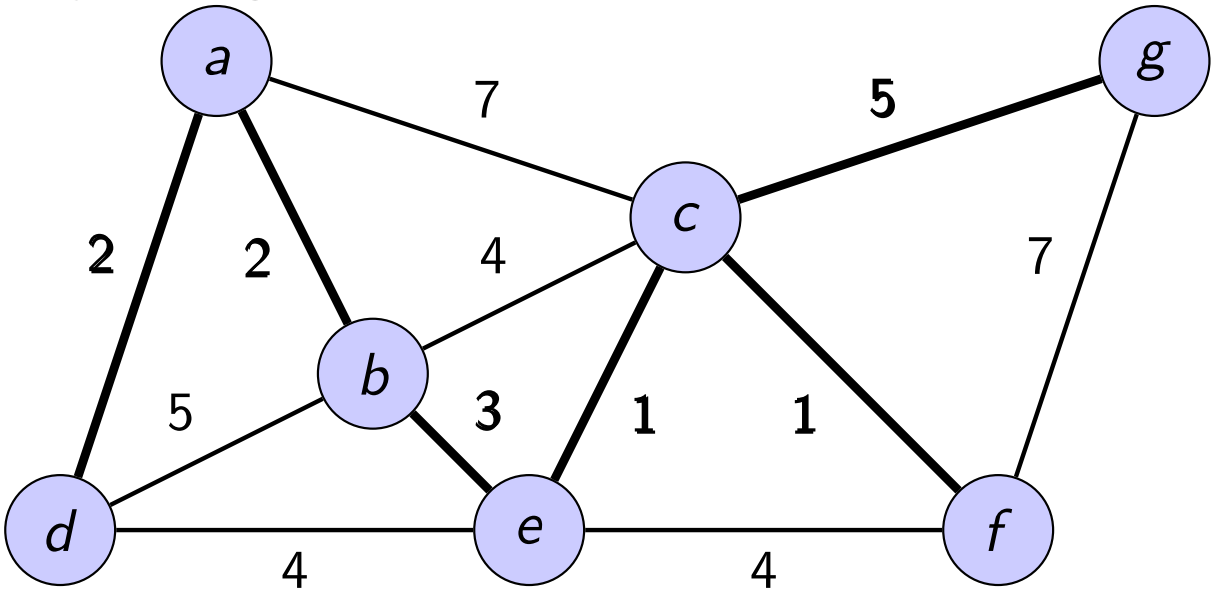
Bài toán cây khung nhỏ nhất (tiếp)

Ví dụ, xét đồ thị G vô hướng có trọng số



Bài toán cây khung nhỏ nhất (tiếp)

Cây khung nhỏ nhất T^* được thể hiện bởi các đường cạnh nối in đậm



Bài toán cây khung nhỏ nhất (tiếp)

Thuật toán Kruskal

Thuật toán sẽ xây dựng tập cạnh E_T của cây khung nhỏ nhất $T = (V, E_T)$ theo từng bước

1. Sắp xếp các cạnh của đồ thị G=(V,E) theo chiều thứ tự không giảm độ dài vào một danh sách
2. Gán tập cạnh ban đầu là rỗng $E_T = \emptyset$
3. Duyệt trong danh sách, lấy cạnh có độ dài nhỏ nhất mà việc bổ sung vào E_T không tạo thành chu trình
4. Tiếp tục bước 3 đến khi tập E_T có đúng |V| - 1 cạnh

Bài toán cây khung nhỏ nhất (tiếp)

Thuật toán Kruskal (tiếp)

Kruskal-Algorithm

$E_T \leftarrow \emptyset$

while ($|E_T| < |V| - 1$ and $E_T \neq \emptyset$) **do**

< Chọn e là cạnh có *độ dài nhỏ nhất* trong E >

E $\leftarrow$ E -{e}

if ($E_T \cup e$ không chứa chu trình) **then** $E_T \leftarrow E_T \cup e$ **endif**

endwhile

if ($|E_T| < |V| - 1$) **then** <Đồ thị không liên thông> **endif**

EndProcedure

Cấu trúc dữ liệu biểu diễn phân hoạch

Đặt vấn đề

- Trong thuật toán Kruskal, bước đòi hỏi khối lượng tính toán lớn là sắp xếp các cạnh của đồ thị theo thứ tự không giảm của độ dài. Với đồ thị m cạnh thì thời gian đòi hỏi $O(m \log m)$. Tuy nhiên, để xây dựng cây khung n-1 cạnh thì nói chung ta không cần đến việc sắp xếp thứ tự toàn bộ tập cạnh mà chỉ cần xét phần trên của dãy chữa p«m cạnh. Do đó thay vì sắp xếp toàn bộ dãy cạnh ta sẽ dùng min-heap *để vun p cạnh nhỏ nhất lên gốc*.

Cấu trúc dữ liệu biểu diễn phân hoạch (tiếp)

Đặt vấn đề (tiếp)

- Vấn đề thứ hai, là việc lựa chọn cạnh để bổ sung đòi hỏi có thủ tục kiểm tra $E_T \cup \{e\}$ có chứa chu trình hay không. Để ý rằng, các cạnh trong E_T ở các bước lặp trung gian sẽ tạo thành một rừng. Vậy cạnh e cần khảo sát sẽ tạo thành chu trình với các cạnh trong E_T khi và chỉ khi *cả hai đỉnh đầu của nó thuộc vào cùng một cây con trong rừng trên*. Suy ra, cạnh e phải nối hai cây khác nhau trong E_T. Vậy giải pháp là
  - Phân hoạch tập các đỉnh của đồ thị thành các tập con không giao nhau, mỗi tập được xác định bởi một cây con trong T^* (đc hình thành ở các bước do việc bổ sung cạnh vào T^*)

  Có hai thủ tục cần xây dựng
  - Kiểm tra hai đỉnh đầu e=(u,v) có thuộc vào hai tập con khác nhau hay không
  - Nếu khẳng định, hợp hai tập con tương ứng thành một tập

Cấu trúc dữ liệu biểu diễn phân hoạch (tiếp)

Cấu trúc dữ liệu mô tả các tập không giao nhau

Cho tập $V = \{1, 2, 3, \cdots, n\}$, ta cần xây dựng cấu trúc dữ liệu mô tả các tập con trong phân hoạch của tập V với các phép toán cơ bản sau

- Makeset(x) - tạo một tập con chứa duy nhất x.
- Union(x,y) - thay hai tập con x và y bởi tập là hợp của chúng.
- Find(x) - trả lại tên tập con chứa x.

Để biểu diễn một tập con $X \subset V$, ta sẽ sử dụng cấu trúc cây có gốc là phần tử thuộc X, đây chính là tên của tập con. Mỗi phần tử $x \in X$ sẽ có con trỏ parent[x] chỉ đến cha của nó, nếu x là gốc thì parent[x] = x

Cấu trúc dữ liệu biểu diễn phân hoạch (tiếp)

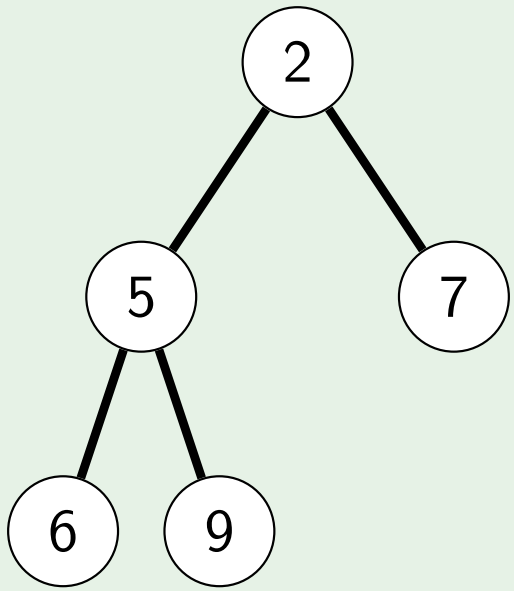
Ví dụ

Giả sử $V = \{1, 2, 3, \cdots, 9\}$ có ba tập

- $V_1 = \{1, 3, 4\}$
- $V_2 = \{2, 5, 6, 7, 9\}$
- $V_3 = \{8\}$

Rừng biểu diễn ba tập không giao nhau trên có thể là

Cây tương ứng với V_1

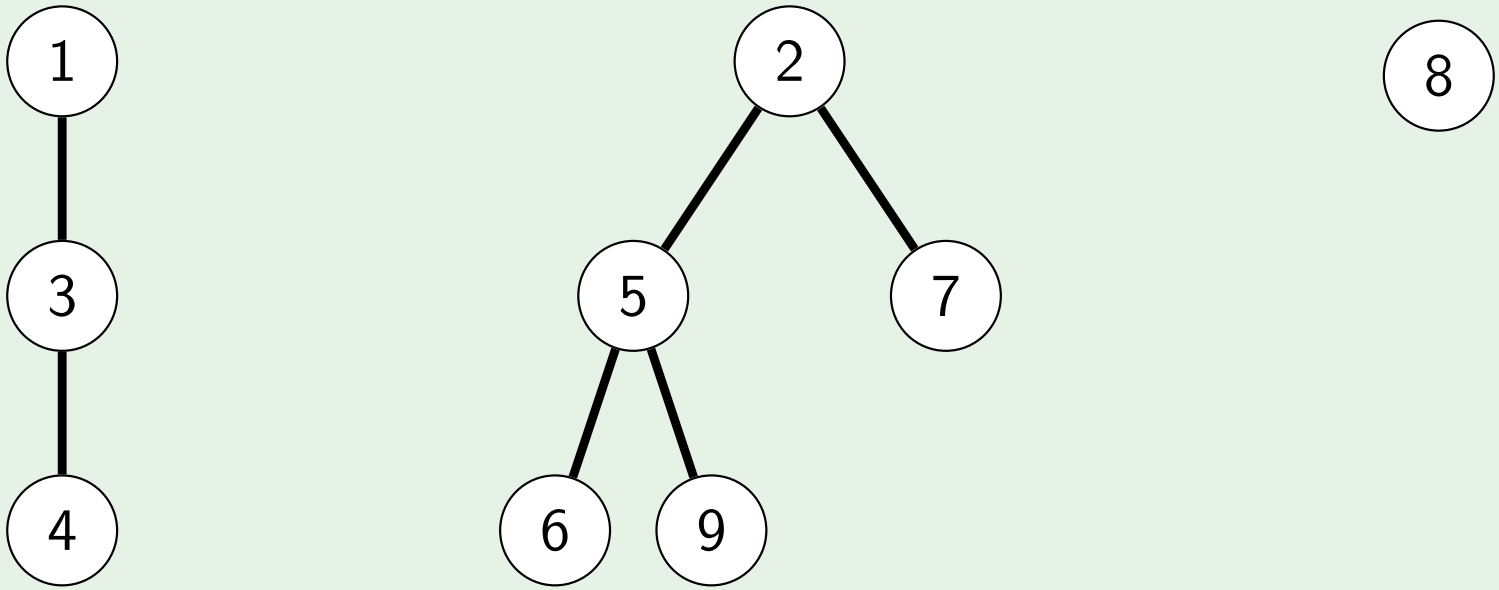


Cây tương ứng với V_2

Cây tương ứng với V_3

Cấu trúc dữ liệu biểu diễn phân hoạch (tiếp)

Ví dụ (tiếp)



Vậy mảng parent để biểu diễn rừng gồm ba cây tương ứng V_1, V_2 và V_3 là

i	1	2	3	4	5	6	7	8	9
parent	1	2	1	3	2	5	2	8	5

Cấu trúc dữ liệu biểu diễn phân hoạch (tiếp)

Các phép toán cơ bản

- tạo tập con chứa x

  Procedure MakeSet(x)

  parent[x] $\leftarrow$ x;

  EndProcedure

- tìm tên tập con chứa x

  Function Find(x)

  while x $\neq$ parent[x] **do** x $\leftarrow$ parent[x] **endwhile**

  return x

  EndFunction

- để nối hai tập con chứa x và y, ta chữa lại biến trỏ của gốc của cây chứa x cho nó trỏ đến gốc cây con chứa y

  Procedure Union(x,y)

  u $\leftarrow$ Find(x); v $\leftarrow$ Find(y)

  parent[u] $\leftarrow$ v

  EndProcedure

Cấu trúc dữ liệu biểu diễn phân hoạch (tiếp)

Phân tích thời gian của các phép toán

Trong ba phép toán, Find(x) đc sử dụng nhiều nhất, phụ thuộc vào chiều cao của cây. Trong trường hợp xấu nhất, cây là đường thẳng như V_1 thì độ cao của cây gồm k đỉnh sẽ là k-1. Suy ra, hàm Find(x) có thời gian tính O(n). Vậy làm sao để giảm chiều cao của cây ?

- Khi hợp hai cây, ta sẽ điều chỉn con trỏ của gốc của cây có ít đỉnh hơn. Để ghi nhận số đỉnh của một cây, ta sẽ sử dụng thêm mảng biến num[v] chứa số phần tử của cây con với gốc tại v.

Cấu trúc dữ liệu biểu diễn phân hoạch (tiếp)

Các phép toán cơ bản được cải tiến

- **Procedure** MakeSet(x)
  parent[x] $\leftarrow$ x; num[x] $\leftarrow$ 1
  EndProcedure
- **Procedure** Union(x,y)
  u $\leftarrow$ Find(x); v $\leftarrow$ Find(y)
  if num[u] $\leq$ num[v] **then**
  parent[u] $\leftarrow$ v; num[u] $\leftarrow$ num[u] + num[v]
  else
  parent[v] $\leftarrow$ u; num[v] $\leftarrow$ num[u] + num[v]
  endif
  EndProcedure

Cấu trúc dữ liệu biểu diễn phân hoạch (tiếp)

Bổ đề

Giả sử quá trình thực hiện nối cây bắt đầu từ các cây chỉ có một đỉnh. Khi đó độ cao của các cây xuất hiện khi thực hiện thủ tục nối không vượt quá $\log n$ (sử dụng quy nạp để chứng minh). Suy ra, các thao tác Find(x) và Union(x,y) được thực hiện với thời gian $O(\log n)$.

Cấu trúc dữ liệu biểu diễn phân hoạch (tiếp)

Procedure Kruskal-UF

1. $E_T \leftarrow \emptyset$
2. **for** $v \in V$ **do** MakeSet(v) **endfor**
3. *Sắp xếp các cạnh trong E theo thứ tự không giảm của trọng số*
4. **for** (u,v) $\in$ E **do**
5. **if** Find(u) $\neq$ Find(v) **then**
6. $E_T \leftarrow E_T \cup \{(u, v)\}$; Union(u,v)
7. **endif**
8. **endfor**
9. **if** $|E_T| <$ |V|-1 **then**
   đồ thị không liên thông
   else
   E_T là tập cạnh của cây khung nhỏ nhất
   endif

EndProcedure

Cấu trúc dữ liệu biểu diễn phân hoạch (tiếp)

Phân tích thời gian của thuật toán Kruskal-UF

- Dòng 1-2 : O(|V|)
- Dòng 3 : sắp xếp danh sách cạnh O(|E| log |E|)
- Dòng 4-8 : các thao tác với phân hoạch phải lặp lại không quá |E| lần, suy ra O(|E| log |E|)
- Dòng 9 : O(1)

Tổng cộng : O(|E| log |E|)

3

[3] chú ý, với đồ thị thưa thì $|E| = O(|V|)$

Bài toán đường đi ngắn nhất

Đặt vấn đề

Cho đồ thị có *trọng số* trên các cạnh, ví dụ trọng số này có thể là :

- khoảng cách
- chi phí
- thời gian

trong cách biểu diễn sơ đồ giao thông. Đối với đồ thị dạng này câu hỏi thương đặt ra :

- Đi từ đỉnh A đến đỉnh B theo đường nào ngắn nhất ?
- Đi từ đỉnh A đến đỉnh B theo đường nào rẻ nhất ?
- Đi từ đỉnh A đến đỉnh B theo đường nào nhanh nhất ?
- Đi từ đỉnh A đến đỉnh B theo đường *tối ưu* theo tiêu chí tổng hợp nào đó ?

$\Rightarrow$ Quy về bài toán tìm đường đi ngắn nhất - shortest path problem

Bài toán đường đi ngắn nhất (tiếp)

Định nghĩa về độ dài đường đi

Cho đồ thị có hướng G=(V,E) với trọng số trên cạnh c(e) e $\in$ E, giả sử s,t $\in$ V và P(s,t) là đường đi từ s đến t trên đồ thị

$$P(s,t) : s \equiv v_0, v_1, \cdots, v_{k-1}, v_k \equiv t$$

Ta gọi độ dài đường đi P(s,t) là tổng trọng số trên các cung của nó, ký hiệu là ρ(P(s,t)), thì theo định nghĩa

$$\rho(P(s,t)) = \sum_{e \in P(s,t)} c(e) = \sum_{i=0}^{k-1} c(v_i, v_{i+1})$$

Đường đi ngắn nhất từ s đến t là đường đi có tổng trọng số trên là *nhỏ nhất* trong tất cả các đường đi từ s đến t. Người ta thường ký hiệu độ dài nhỏ nhất này là $\delta(s,t)$

Bài toán đường đi ngắn nhất (tiếp)

Ba dạng đường đi ngắn nhất cơ bản

1. Tìm đường đi ngắn nhất giữa hai đỉnh cho trước
2. Tìm đường đi ngắn nhất từ đỉnh nguồn s tới tất cả các đỉnh còn lại
3. Tìm đường đi ngắn nhất giữa hai đỉnh bất kỳ

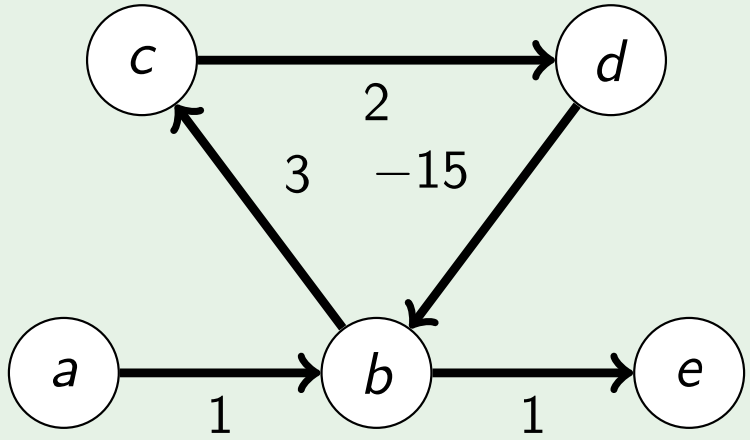
Bài toán đường đi ngắn nhất (tiếp)

Nhận xét về các dạng bài toán tìm đường đi ngắn nhất

- Dễ thấy là các bài toán được dẫn theo thứ tự từ đơn giản đến phức tạp, thêm nữa nếu ta có thuật toán giải một trong ba bài toán thì thuật toán đó cũng có thể áp dụng để giải hai bài còn lại.
- Đường đi ngắn nhất giữa hai đỉnh nào đó có thể không tồn tại. Chẳng hạn nếu không có đường đi từ s đến t thì rõ ràng không có đường đi ngắn nhất từ s đến t.
- Nếu đồ thị chứa cạnh có trọng số âm thì có thể xảy ra tình huống : độ dài đường đi giữa hai đỉnh có thể làm bé tùy ý (hay đồ thị có chứa chu trình âm)

Bài toán đường đi ngắn nhất (tiếp)

Đồ thị chứa chu trình âm



Ta có đường đi từ a đến e đồng thời đi qua chu trình C : b,c,d,b là k lần. Vậy độ dài đường đi này sẽ là

$$\rho P(a,e) = c(a,b) + k\rho(C) + c(b,e) = 2 - 10 \times k \to -\infty \text{ khi } k \to \infty$$

Vậy độ dài đường đi có thể làm nhỏ tùy ý (hay đường đi ngắn nhất không tồn tại $\nexists\, \delta(a,e)$)

Thuật toán Dijkstra

Thuật toán Dijkstra được đề xuất để giải bài toán

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

Khi thực hiện thuật toán, với mỗi đỉnh v

- k[v] : biến boolean có giá trị true nếu ta đã tìm được đường đi ngắn nhất từ s đến v, ban đầu nó đc khởi tạo là false.
- d[v] : khoảng cách ngắn nhất hiện biết từ s đến v. Ban đầu biến đc khởi tạo giá trị $+\infty$ đối với mọi đỉnh, ngoại trừ d[s] được đặt bằng 0.
- p[v] : là đỉnh đi trước đỉnh v trong đường đi có độ dài d[v]. Ban đầu các biến được khởi tạo rỗng.

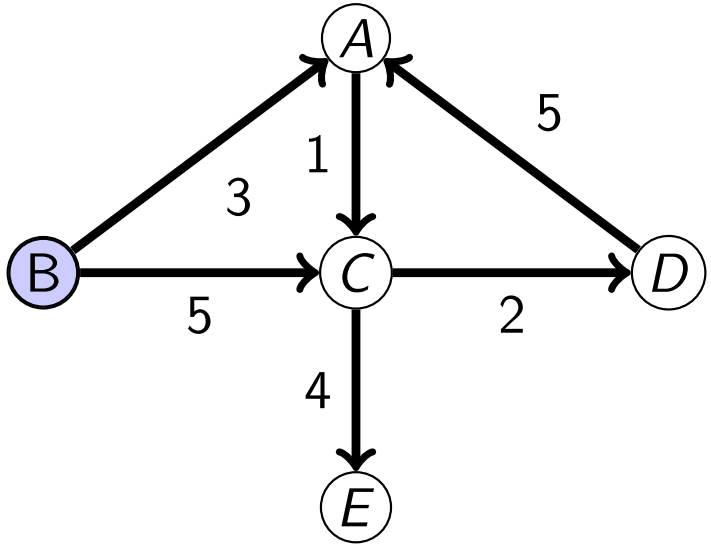
Thuật toán Dijkstra (tiếp)

Các thao tác lặp của thuật toán

Thuật toán lặp lại các thao tác sau đây cho đến khi tất cả các đỉnh đc khảo sát xong (nghĩa là k[v] là true với mọi biến)

- trong tập đỉnh với k[v] = False, chọn đỉnh v có d[v] là độ dài nhỏ nhất.
- đặt k[v] = True.
- với mỗi đỉnh w kề với v và có k[v] = false, ta kiểm tra d[w] > d[v] + c(v,w). Nếu đúng thì đặt lại d[w] = d[v] + c(v,w) và đặt p[w] = v.

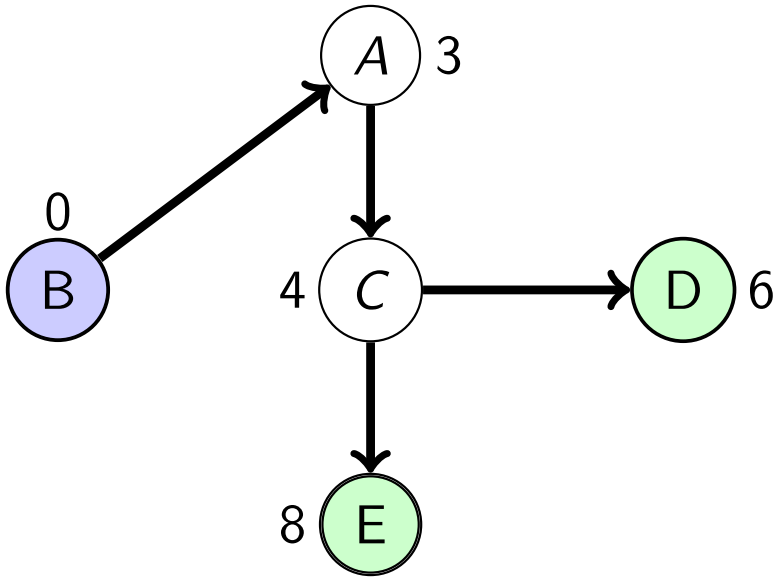
Thuật toán Dijkstra (tiếp)



Bc	A			**B**			C			D			E		
lặp	d	p	k	d	p	k	d	p	k	d	p	k	d	p	k
kt	∞	-	F	0	-	F	∞	-	F	∞	-	F	∞	-	F
1	3	B	F	**0**	-	**T**	5	B	F	∞	-	F	∞	-	F
2	**3**	**B**	**T**				4	A	F	∞	-	F	∞	-	F
3							**4**	**A**	**T**	6	C	F	8	C	F
4										**6**	**C**	**T**	8	C	F
5													**8**	**C**	**T**

Thuật toán Dijkstra (tiếp)

Tập các cạnh $\{(p[v], v) : v \in V - \{B\}\}$ tạo được một cây được gọi là **cây đường đi ngắn nhất**



Số viết bên cạnh mỗi đỉnh là độ dài đường đi ngắn nhất từ đỉnh B đến nó hay giá trị d[v]

Cài đặt thuật toán với các cấu trúc dữ liệu

Procedure Dijkstra-Table(G,s)

1. **for** u $\in$ V **do** d[u] $\leftarrow \infty$; p[u] $\leftarrow$ NULL; k[u] $\leftarrow$ false; **endfor**
2. d[s] $\leftarrow$ 0
3. T $\leftarrow$ V
4. **while** T $\neq \emptyset$ **do**
5. u $\leftarrow$ đỉnh có d[u] là *nhỏ nhất* trong T
6. k[u] $\leftarrow$ True
7. T $\leftarrow$ T - {u}
8. **for** (v $\in$ Adj[u] **and** k[v] = false) **do**
9. **if** (d[v] > d[u] + c[u,v]) **then**
10. d[v] $\leftarrow$ d[u] + c[u,v]; p[v] $\leftarrow$ u;
11. **endif**
12. **endfor**
13. **endwhile**

EndProcedure

Cài đặt thuật toán với các cấu trúc dữ liệu

```
Procedure Dijkstra-Table(G,s)
1  for u ∈ V do d[u] ← ∞; p[u] ← NULL; k[u] ← false; endfor
2  d[s] ← 0
3  T ← V
4  while T ≠ ∅ do
5      u ← đỉnh có d[u] là nhỏ nhất trong T
6      k[u] ← True
7      T ← T - {u}
8      for (v ∈ Adj[u] and k[v] = false) do
9          if (d[v] > d[u] + c[u,v]) then
10             d[v] ← d[u] + c[u,v]; p[v] ← u;
11         endif
12     endfor
13 endwhile
EndProcedure
```

Chú ý, câu lệnh 5 có yêu cầu tìm giá trị nhỏ nhất trong số các phần tử còn lại của bảng. Còn câu lệnh 8-12 thì duyệt qua các cạnh của đồ thị. Vậy tổng cộng Dijkstra-Table(G,s) đòi hỏi thời gian là $O(|V|^2) + O(|E|) = O(|V|^2+|E|)$. Do tại mỗi bước, ta cần tìm ra đỉnh có nhãn d[v] nhỏ nhất, nên để thực hiện thao tác này ta có thể dùng hàng đợi có ưu tiên dùng đống-min (min-heap) để thực hiện việc vun, cũng như lấy ra giá trị nhỏ nhất của đồng.

Cài đặt thuật toán với các cấu trúc dữ liệu (tiếp)

Procedure Dijkstra-Heap(G,s)

1. **for** u ∈ V **do** d[u] ← ∞; p[u] ← NULL; k[u] ← false; **endfor**
2. d[s] ← 0 // đỉnh nguồn là s
3. H ← Build-Min-Heap(d[v]) // Xây dựng hàng đợi có ưu tiên
4. **while** H ≠ ∅ **do**
5. u ← Extract-Min(H); // loại bỏ gốc của đống
6. k[u] ← True
7. T ← T - {u}
8. **for** (v ∈ Adj[u] **and** k[v] = false) **do**
9. **if** (d[v] > d[u] + c[u,v]) **then**
10. d[v] ← d[u] + c[u,v]; p[v] ← u; Decrease-Key(H,v,d[v]);
11. **endif**
12. **endfor**
13. **endwhile**

EndProcedure

Cài đặt thuật toán với các cấu trúc dữ liệu (tiếp)

Phân tích thời gian tính của thuật toán

- Vòng lặp **for** ở dòng 1 đòi hỏi thời gian O(|V|)
- Việc khởi tạo đống min ở dòng 3 đòi hỏi thời gian O(|V|)
- Vòng lặp **while** bắt đầu từ dòng 4 lặp |V| lần do đó thao tác Extract-Min thực hiện |V| lần và đòi hỏi thời gian tổng cộng O(|V| log |V|)
- Thao tác vun lại đống Decrease-Key() ở dòng 10 phải thực hiện không quá O(|E|) lần. Do đó thời gian thực hiện của thao tác này trong thuật toán là O(|E|log|V|)

Vậy tổng cộng thời gian tính của thuật toán là O((|E|+|V|)log |V|)

4

[4] Xem thêm về đống Fibonnacci