CHƯƠNG 2
ĐỒ THỊ DẠNG CÂY & DAG

Bùi Tiến Lên

Đại học Khoa học Tự nhiên TPHCM

1/1/2018

Nội dung

1. ĐỒ THỊ DẠNG CÂY

2. CÂY TRONG TIN HỌC

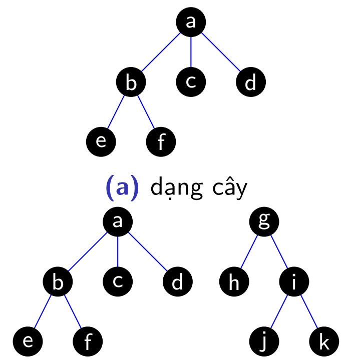
3. ĐỒ THỊ DẠNG DAG

ĐỒ THỊ DẠNG CÂY

Các khái niệm cơ bản

Định nghĩa 2.1

- **Cây** (**Tree**) là đồ thị liên thông và không có chu trình
- **Rừng** (**Forest**) là đồ thị không có chu trình

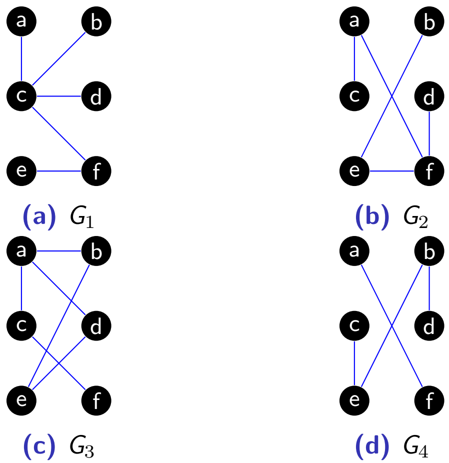


(a) dạng cây

(b) dạng rừng

Hình 2.1: Các đồ thị dạng cây & rừng

Các khái niệm cơ bản (cont.)



(a) G_1 (b) G_2 (c) G_3 (d) G_4

Hình 2.2: Đồ thị dạng cây và không phải dạng cây

Các khái niệm cơ bản (cont.)

Từ định nghĩa ta có những nhận xét sau

Nhận xét

- Một đồ thị dạng cây sẽ không có cạnh khuyên và không có cạnh song song
- Một đồ thị dạng rừng sẽ có p thành phần liên thông và mỗi thành phần liên thông là một đồ thị dạng cây

Định lý về sự tồn tại đỉnh treo

Định lý 2.1

Nếu một cây có n đỉnh với $n \geq 2$ thì cây chứa ít nhất hai đỉnh treo

Chứng minh

- Cho cây $T = (V, E)$, xét cạnh $(a, b) \in E$
- Gọi $P = u...a...b...v$ là đường đi sơ cấp dài nhất trên cây T chứa cạnh (a, b)
- Ta nhận thấy u và v phải là 2 đỉnh treo

■

Các định nghĩa về cây

Cho một đồ thị G có n đỉnh, các phát biểu sau là tương đương

Định lý 2.2 (daisy chain)

1. *Đồ thị G là cây.*
2. *Giữa hai đỉnh bất kỳ của đồ thị G, tồn tại duy nhất một dây chuyền nối chúng.*
3. *Đồ thị G liên thông tối thiểu (nghĩa là G liên thông và nếu xóa đi bất kỳ cạnh nào của nó thì nó không còn liên thông nữa).*
4. *Thêm một cạnh nối 2 đỉnh bất kỳ của G thì đồ thị sẽ chứa một chu trình duy nhất.*
5. *Đồ thị G liên thông và có $n - 1$ cạnh.*
6. *Đồ thị G không có chu trình và có $n - 1$ cạnh.*

Các định nghĩa về cây (cont.)

Từ các định nghĩa tương đương về cây ta có những nhận xét sau về đồ thị n đỉnh

Nhận xét

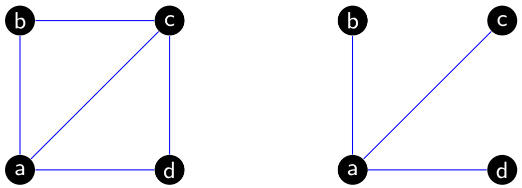
- Cây là một *cấu trúc tiết kiệm cạnh nhất* nhưng vẫn đảm bảo sự liên thông
- Cây là một *cấu trúc không bền vững* vì tính liên thông sẽ bị mất khi bỏ đi bất cứ một cạnh nào

Cây khung

Định nghĩa 2.2

Cho một đồ thị $G = (V, E)$ liên thông và $T = (V, F)$ là đồ thị bộ phận của G. Nếu T là cây thì nó được gọi là **cây khung** (**spanning tree**) của đồ thị G.

Cây khung (cont.)



(a) đồ thị **(b)** cây khung

Hình 2.3: Đồ thị và cây khung

Định lý về sự liên thông & cây khung

Định lý 2.3

Một đồ thị liên thông nếu và chỉ nếu nó có cây khung

Chứng minh

- ($\Leftarrow$) Đồ thị G có cây khung T thì đồ thị G liên thông. Điều này suy ra tự định nghĩa về cây và cây khung
- ($\Rightarrow$) Đồ thị G liên thông thì đồ thị G có cây khung T
  - Nếu đồ thị G có một chu trình C thì loại bỏ một cạnh của chu trình để tạo ra đồ thị con G'. Đồ thị G' là một đồ thị liên thông
  - Tiếp tục quá trình loại bỏ cạnh cho đến khi không thể làm được thì sẽ được đồ thị con T không có chu trình. Đồ thị T là cây khung

■

Thuật toán Prim tìm cây khung

Cho đồ thị $G = (V_G, E_G)$ là một đồ thị liên thông có n đỉnh. Xác định cây khung $T = (V_T, E_T)$ của đồ thị G.

Algorithm 1 Thuật toán Prim

- **Bước 1**: Chọn tùy ý đỉnh $v \in V_G$ và khởi tạo $V_T = \{v\}$ và $E_T = \emptyset$
- **Bước 2**: Tìm cạnh $e = (x, y)$ sao cho $x \in V_T$ và $y \in V_G \setminus V_T$. Nếu không tìm thấy thì **DỪNG (1)**
- **Bước 3**: Thực hiện thêm đỉnh và thêm cạnh $V_T = V_T + \{y\}$, $E_T = E_T + \{e = (x, y)\}$
- **Bước 4**: Nếu T có đủ n đỉnh thì **DỪNG (2)** ngược lại tiếp tục quay lại bước 2

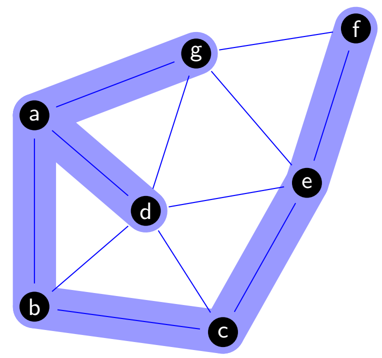
Minh họa thuật toán Prim tìm cây khung

Cho đồ thị G dưới và đỉnh bắt đầu là a. Cây khung $T = (V_T, E_T)$ được tìm như sau

Bảng 2.1: Bảng tính toán

Thêm vào V_T	Thêm vào E_T
a	
b	(a, b)
d	(a, d)
g	(a, g)
c	(b, c)
e	(c, e)
f	(e, f)

Hình 2.4: Đồ thị G và cây khung



Thuật toán DFS tìm cây khung

Cho một đồ thị liên thông $G = (V, E)$ có n đỉnh $V = \{v_1, ..., v_n\}$. Hãy tìm cây khung T

Algorithm 2 Tìm cây khung

Khởi tạo cây T với đỉnh v bất kỳ
DFS_SPANNING_TREE(v)
procedure DFS_SPANNING_TREE(v)
  for mỗi đỉnh u kề với v **do**
    if $u \notin T$ **then**
      Thêm đỉnh u và cạnh (v, u) vào cây T
      DFS_SPANNING_TREE(u)

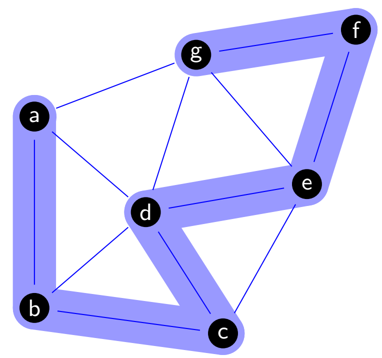
Minh họa DFS tìm cây khung

Cho đồ thị G dưới và đỉnh bắt đầu là a. Cây khung $T = (V_T, E_T)$ được tìm như sau

Bảng 2.2: Bảng tính toán

Thêm vào V_T	Thêm vào E_T
a	
b	(a, b)
c	(b, c)
d	(c, d)
e	(d, e)
f	(e, f)
g	(f, g)

Hình 2.5: Đồ thị G và cây khung



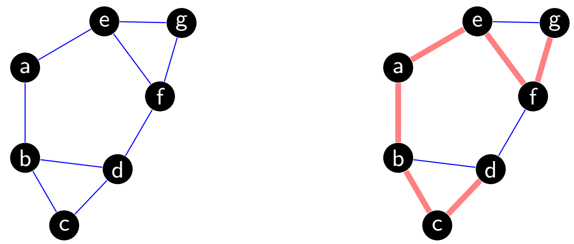
Tập các chu trình cơ sở

Định nghĩa 2.3

Cho một đồ thị vô hướng liên thông $G = (V, E)$ và $T = (V, F)$ là cây khung của đồ thị này.

- Đồ thị $T + e$ với $e \in E - F$ có duy nhất một chu trình đơn C_e được gọi là chu trình cơ sở
- Tập hợp $\Omega = \{C_e | e \in E - F\}$ được gọi là tập hợp các chu trình cơ sở

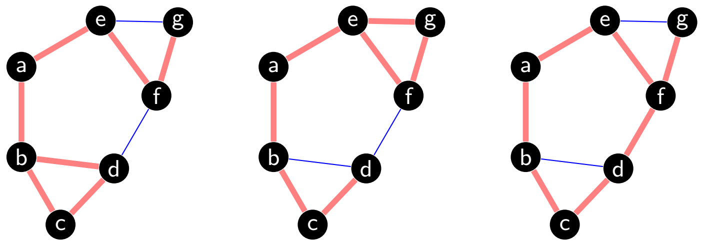
Tập các chu trình cơ sở (cont.)



(a) đồ thị **(b)** cây khung

Hình 2.6: Đồ thị và cây khung

Tập các chu trình cơ sở (cont.)



(a) $C_{(b,d)}$ **(b)** $C_{(e,g)}$ **(c)** $C_{(d,f)}$

Hình 2.7: Tập các chu trình cơ sở

Tập các chu trình cơ sở (cont.)

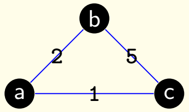
Tính chất 2.1

1. *Tập các chu trình cơ sở phụ thuộc vào cây khung; nghĩa là, các cây khung khác nhau thì tập các chu trình cơ sở tương ứng cũng khác nhau*
2. *Đồ thị G có n đỉnh và m cạnh thì sẽ có $m - n + 1$ chu trình cơ sở*
3. *Mọi chu trình đơn C của đồ thị G sẽ là **tổng** của một số chu trình cơ sở*

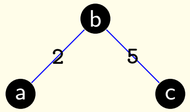
$$C = \sum_{C_i \in \Omega} C_i \qquad (2.1)$$

Cây khung có trọng số nhỏ nhất & lớn nhất

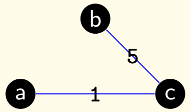
Ví dụ 2.1



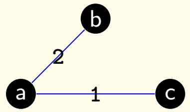
(a) đồ thị



(b) cây khung



(c) cây khung



(d) cây khung

Hình 2.8: Các cây khung

Cây khung có trọng số nhỏ nhất & lớn nhất (cont.)

Định nghĩa 2.4

Cho một đồ thị có trọng số $G = (V, E, L)$ và $\mathbb{T}$ là tập các cây khung của G

- Đồ thị $T \in \mathbb{T}$ được gọi là **cây khung có trọng số nhỏ nhất** (**minimum spanning tree**) nếu trọng số của nó nhỏ nhất

$$T = \arg\min_{T \in \mathbb{T}}(L(T)) \tag{2.2}$$

- Đồ thị $T \in \mathbb{T}$ được gọi là **cây khung có trọng số lớn nhất** (**maximum spanning tree**) nếu trọng số của nó lớn nhất

$$T = \arg\max_{T \in \mathbb{T}}(L(T)) \tag{2.3}$$

Ứng dụng cây khung có trọng số nhỏ nhất

- Bài toán xây dựng đường cao tốc
- Bài toán xây dựng hệ thống mạng máy tính

Thuật toán tìm cây khung nhỏ nhất

Có rất nhiều các kỹ thuật khác nhau để tìm cây khung. Ở đây, chỉ xem xét hai thuật toán thông dụng để tìm cây khung nhỏ nhất

- Thuật toán Prim
- Thuật toán Kruskal

Thuật toán Prim

Cho đồ thị có trọng số $G = (V_G, E_G, L_G)$ là một đồ thị liên thông có n đỉnh. Xác định cây khung $T = (V_T, E_T)$ của đồ thị G.

Algorithm 3 Thuật toán Prim

- **Bước 1**: Chọn tùy ý đỉnh $v \in V_G$ và khởi tạo $V_T = \{v\}$ và $E_T = \emptyset$
- **Bước 2**: Tìm tập hợp các cạnh $e = (x, y)$ sao cho $x \in V_T$ và $y \in V_G \setminus V_T$. Nếu tập hợp rỗng thì **DỪNG (1)**; ngược lại chọn cạnh $e = (x, y)$ có trọng số nhỏ nhất
- **Bước 3**: Thực hiện thêm đỉnh và thêm cạnh $V_T = V_T + \{y\}$ và $E_T = E_T + \{e\}$
- **Bước 4**: Nếu T có đủ n đỉnh thì **DỪNG (2)** ngược lại tiếp tục quay lại bước 2

Thuật toán Prim (cont.)

Định lý 2.4

Thuật toán Prim là đúng đắn

Chứng minh

Sinh viên tự chứng minh ■

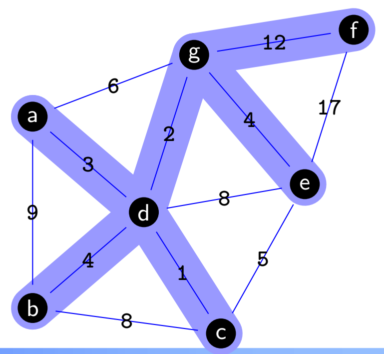
Minh họa thuật toán Prim

Cho đồ thị có trọng số G dưới và đỉnh bắt đầu là a. Cây khung nhỏ nhất $T = (V_T, E_T)$ được tìm như sau

Bảng 2.3: Bảng tính toán

Thêm vào V_T	Thêm vào E_T
a	
d	(a, d)
c	(d, c)
g	(d, g)
b	(d, b)
e	(g, e)
f	(g, f)

Hình 2.9: Đồ thị G và cây khung nhỏ nhất



Thuật toán Kruskal

Cho đồ thị $G = (V_G, E_G, L_G)$ là một đồ thị liên thông có n đỉnh. Xác định cây khung $T = (V_T, E_T)$ của đồ thị G.

Algorithm 4 Thuật toán Kruskal

- **Bước 1**: Sắp xếp các cạnh trong E_G theo trọng số tăng dần thành một danh sách L.
- **Bước 2**: Khởi tạo $V_T = \emptyset$ và $E_T = \emptyset$
- **Bước 3**: Lần lượt duyệt từng cạnh $e = (x, y)$ trong danh sách L. Nếu đồ thị T cùng với cạnh e không chứa chu trình thì loại cạnh e ra khỏi danh sách L và thực hiện $V_T = V_T + \{x, y\}$ và $E_T = E_T + \{e\}$
- **Bước 4**: Nếu T có đủ $n - 1$ cạnh thì dừng lại ngược lại quay lại bước 2

Thuật toán Kruskal (cont.)

Định lý 2.5

Thuật toán Kruskal là đúng đắn

Chứng minh

Sinh viên tự chứng minh ■

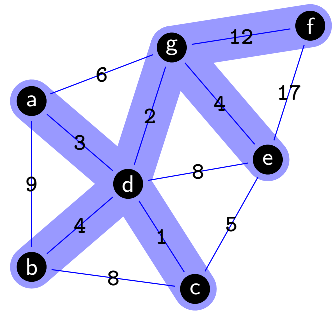
Minh họa thuật toán Kruskal

Cho đồ thị có trọng số G dưới. Danh sách cạnh L
(c,d),(d,g),(a,d),(b,d),(e,g),(c,e),(a,g),(b,c),(d,e),(a,b),(f,g),(e,f)

Bảng 2.4: Bảng tính toán

Thêm vào V_T	Thêm vào E_T
$\{c, d\}$	(c, d)
$\{d, g\}$	(d, g)
$\{a, d\}$	(a, d)
$\{b, d\}$	(b, d)
$\{e, g\}$	(e, g)
$\{f, g\}$	$(f.g)$

Hình 2.10: Đồ thị G và cây khung nhỏ nhất



Cây có hướng

Định nghĩa 2.5

Cho đồ thị có hướng $G = (V, E)$, đồ thị G được gọi là cây có hướng nếu nó thỏa mãn hai điều kiện sau:

1. G không có chu trình
2. G có gốc

Các định nghĩa tương đương về cây có hướng

Định nghĩa 2.6

Cho một đồ thị có hướng $G = (V, E)$ gồm n đỉnh. Các phát biểu sau là tương đương

1. Đồ thị G là một cây có hướng
2. Đồ thị G có một đỉnh r và từ r tồn tại một đường đi duy nhất đến các đỉnh còn lại
3. Đồ thị G tựa liên thông mạnh tối tiểu (tức là nếu xóa bớt bất kỳ một cạnh nào thì G sẽ không còn tựa liên thông mạnh)
4. Đồ thị G liên thông và có đỉnh r sao cho $d^{-}(r) = 0$ và $d^{-}(v) = 1$ với mọi $v \neq r$
5. Đồ thị G không có chu trình và có đỉnh r sao cho $d^{-}(r) = 0$ và $d^{-}(v) = 1$ với mọi $v \neq r$

Sự tồn tại cây có hướng

Định lý 2.6

Cho G là một đồ thị có hướng

- *Nếu G chứa một đồ thị bộ phận là cây T có hướng thì G tựa liên thông mạnh.*
- *Nếu G tựa liên thông mạnh thì G có chứa một đồ thị bộ phận T là cây có hướng*

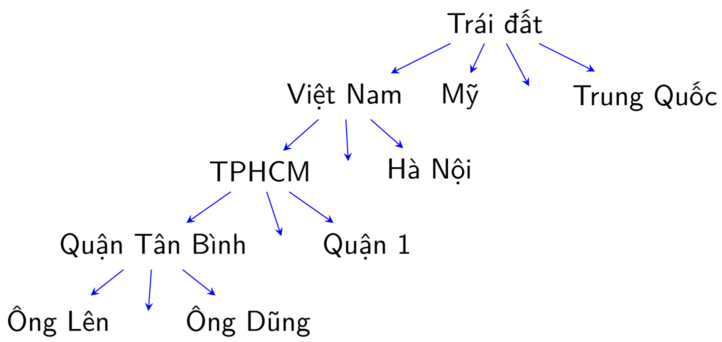
Ứng dụng của cây

Cấu trúc cây thể hiện tính "phân cấp", "kế thừa" do đó có thể biểu diễn được những cấu trúc như

- Cây gia phả (trong các dòng họ)
- Cây phân cấp các loài (trong sinh học)
- Cây thư mục (trong máy tính)

Ứng dụng của cây (cont.)

- Quản lý hành chính phân cấp toàn thế giới

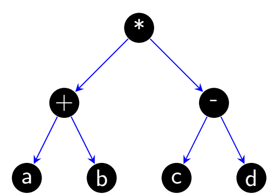


Hình 2.11: Quản lý hành chính toàn cầu

Ứng dụng của cây (cont.)

- Biểu thức toán học có thể được biểu diễn bằng cây. Ví dụ cây dưới đây dùng để biểu diễn biểu thức

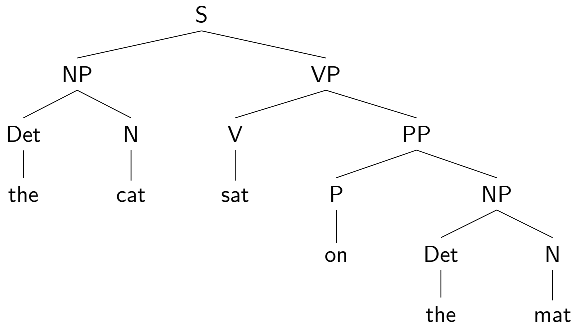
$$(a + b) * (c - d)$$



Hình 2.12: Cây biểu thức

Ứng dụng của cây (cont.)

- Các nhà ngôn ngữ học thường dùng cây ngữ pháp để biểu diễn cấu trúc ngữ pháp của một câu. Ví dụ sau đây dùng để biểu diễn câu *"the cat sat on the mat"*



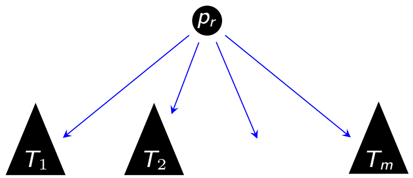
Hình 2.13: Cây ngữ pháp

CÂY TRONG TIN HỌC

Khái niệm

- **Cây** (**Tree**) T gồm một tập hợp n **đỉnh** - **nút** (**node**) $\{p_1, p_2, ..., p_n\}$
  - Trong đó một nút duy nhất là nút gốc p_r; nghĩa là có đường đi đến tất cả các đỉnh còn lại
  - Các nút còn lại được chia thành m tập hợp không giao nhau $\{T_1, T_2, ..., T_m\}$ và mỗi tập này lại là **cây con** (**child tree**)
  - Nếu $n = 0$ thì cây T là **cây rỗng** (**null tree**)

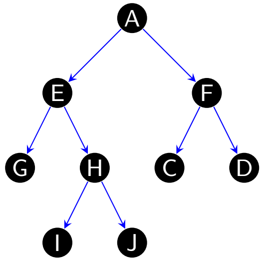
Khái niệm (cont.)



Hình 2.14: Cây trong tin học

Các thuật ngữ liên quan đến cây

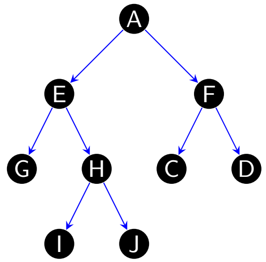
- Nút (**node**): là những phần tử trong cây



Hình 2.15: Cây có các nút {A, B, C, D, E, F, G, H, J}

Các thuật ngữ liên quan đến cây (cont.)

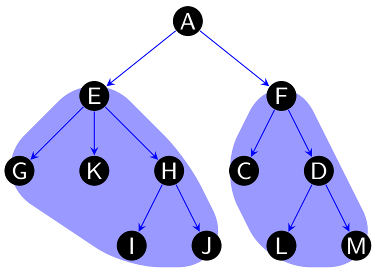
- Nhánh hay cạnh (**branch**): là cạnh mũi tên nối giữa hai nút trong cây
- Nút cha (**parent node**) và nút con (**child node**) là hai quan hệ được định nghĩa trên một cạnh, nút cha là nút đầu cạnh và nút con là nút cuối cạnh



Hình 2.16: Nút E là nút cha của H, nút H là nút con của E

Các thuật ngữ liên quan đến cây (cont.)

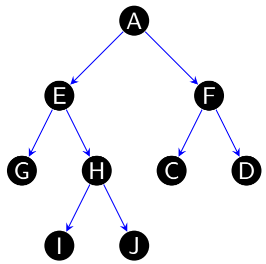
- **Cây con** (**subtree**)



Hình 2.17: Nút A có hai cây con

Các thuật ngữ liên quan đến cây (cont.)

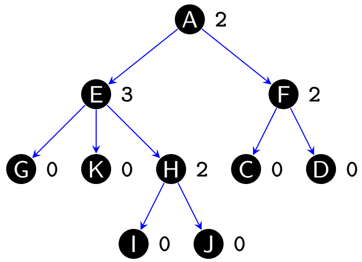
- Nút gốc (**root node**): là nút không có cha
- Nút lá (**leaf node**): là nút không có con
- Nút nội (**internal node**): là nút có cha và có con
- Nút anh em (**sibling node**): là những nút có cùng cha



Hình 2.18: Nút A là nút gốc; nút G, I, J, C, D là nút lá; nút E, H, F là nút nội; nút G và H là anh em

Các thuật ngữ liên quan đến cây (cont.)

- Bậc của nút (**node degree**): là tổng số nút con của nút này

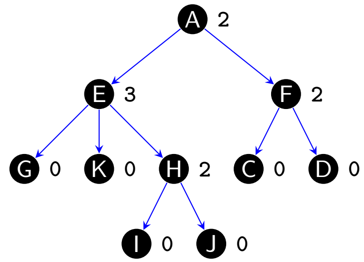


Hình 2.19: Cây và bậc của các nút

Các thuật ngữ liên quan đến cây (cont.)

- Bậc của cây (**tree degree**): là bậc lớn nhất của các nút của cây

$$\deg(T) = \max(\deg(p_i), p_i \in T) \tag{2.4}$$

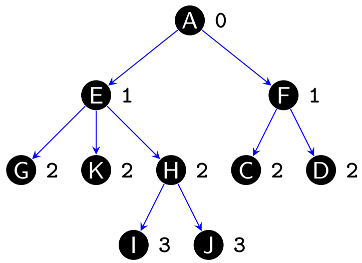


Hình 2.20: Bậc của cây là 3

Các thuật ngữ liên quan đến cây (cont.)

- Mức của nút (**node level**):

$$level(p) = \begin{cases} 0 & p = root \\ level(parent(p)) + 1 & p \neq root \end{cases} \tag{2.5}$$

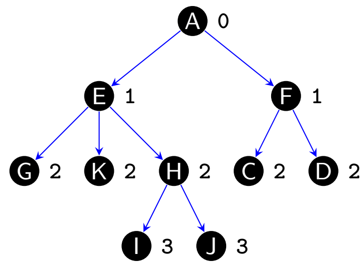


Hình 2.21: Cây và mức của các nút

Các thuật ngữ liên quan đến cây (cont.)

- Chiều cao của cây (**tree height**):

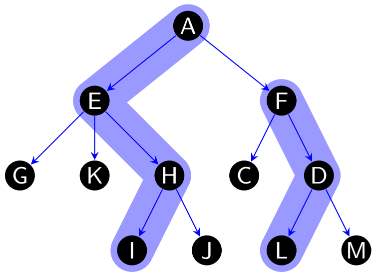
$$height(T) = \max(level(p_i) + 1, p_i \in T) \tag{2.6}$$



Hình 2.22: Chiều cao của cây là 4

Các thuật ngữ liên quan đến cây (cont.)

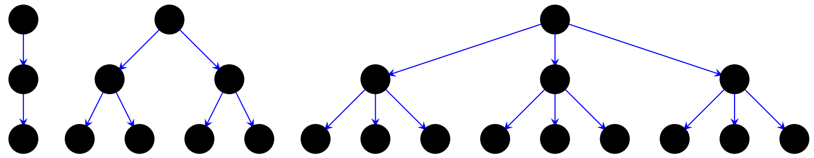
- Đường đi (**path**): là một dãy các nút khác nhau $\{p_1, p_2, ..., p_k\}$ sao cho (p_i, p_{i+1}) là cạnh. Nút p_1 gọi nút đầu hay là nút tổ tiên (**ancestor**) và p_k là nút cuối hay là nút con cháu (**descendant**).



Hình 2.23: {A, E, H, I} và {F, D, L} là các đường đi, {A, E, C} không phải là đường đi. Nút A là tổ tiên của I và nút I là con cháu của A.

Phân loại cây

- Cây tuyến tính (**linear tree**): là cây có bậc bằng 1
- Cây nhị phân (**binary tree**): là cây có bậc bằng 2
- Cây tam phân (**ternary tree**): là cây có bậc bằng 3
- Cây m-nhánh (**m-way tree**): là cây có bậc bằng m



Hình 2.24: Các loại cây

Một số loại cây nhị phân

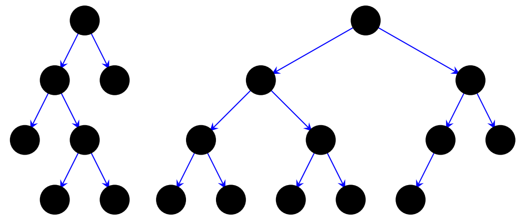
Định nghĩa 2.7

Một số cây nhị phân đặc biệt

- Cây nhị phân đầy đủ (**full binary tree**): là cây mà mỗi nút có 0 hoặc 2 nút con
- Cây nhị phân hoàn chỉnh (**complete binary tree**): là cây mà có
  1. Đầy đủ các nút từ mức 0 đến $h-1$ (h là chiều cao của cây)
  2. Riêng mức h thì các nút liên tiếp từ trái sang phải

Một số loại cây nhị phân (cont.)

Hình dưới minh họa cây đầy đủ và cây hoàn chỉnh.



Hình 2.25: Các loại cây đầy đủ và hoàn chỉnh

Các định lý về cây nhị phân

Định lý 2.7

1. *Nếu T là cây nhị phân thì sẽ không có quá 2^k nút có mức $k \geq 0$*
2. *Nếu T là cây nhị phân có chiều cao là h thì số nút lá tối đa của cây là 2^{h-1}*
3. *Nếu T là cây nhị phân có chiều cao là h thì số nút tối đa của cây là $2^h - 1$*
4. *Nếu T là một cây nhị phân có n nút thì chiều cao nhỏ nhất có thể của cây là là $\log_2(n+1)$*

Các định lý về cây nhị phân (cont.)

Định lý 2.8

Cho T là một cây nhị phân đầy đủ. l là số nút lá và i là số nút nội

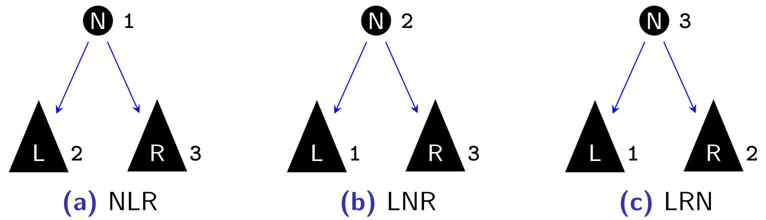
$$l = i + 2 \tag{2.7}$$

Duyệt cây nhị phân

Có thể sử dụng thuật toán duyệt tổng quát như DFS hay BFS. Ngoài ra có thể sử dụng những kỹ thuật duyệt đặc thù cho cây nhị phân như

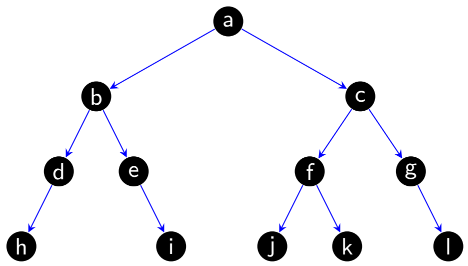
- Duyệt các nút của cây theo thứ tự NLR; nghĩa là duyệt nút trước (N), sau đó duyệt cây con trái (L), cuối cùng duyệt cây con phải (R)
- Duyệt các nút của cây theo thứ tự LNR
- Duyệt các nút của cây theo thứ tự LRN

Duyệt cây nhị phân (cont.)



Hình 2.26: Ba kiểu duyệt đặc thù của cây

Duyệt cây nhị phân (cont.)



Hình 2.27: Duyệt cây bằng 5 cách DFS, BFS, NLR, LNR, LRN

Một số cây trong tin học

- Cây nhị phân tìm kiếm (**Binary Search Tree**)
- Cây Huffman (**Huffman Tree**)
- Cây DFS (**DFS Tree**)
- Cây trò chơi (**Game Tree**)

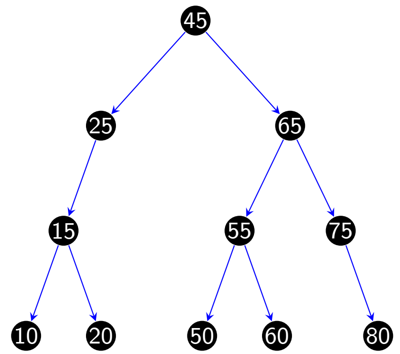
Cây nhị phân tìm kiếm

Định nghĩa 2.8

Cây nhị phân tìm kiếm là cây có:

- Mỗi nút của cây có một giá trị khóa (**key**) và đây là giá trị duy nhất
- Tại một nút p bấy kỳ
  1. Tất cả các nút của cây con trái đều có khóa nhỏ hơn khóa của p
  2. Tất cả các nút của cây con phải đều có khóa lớn hơn khóa của p

Cây nhị phân tìm kiếm (cont.)



Hình 2.28: Cây nhị phân tìm kiếm

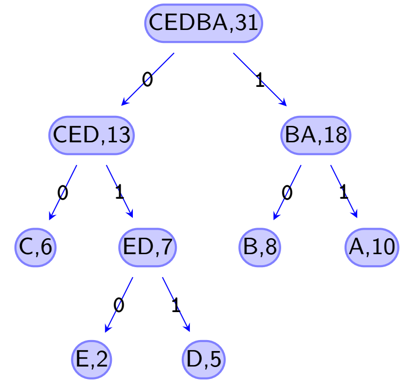
Cây Huffman

Định nghĩa 2.9

Cây Huffman là cây nhị phân sao cho

- Nhãn của nút
  - Nhãn của nút lá là một ký tự
  - Nhãn của nút cha sẽ là chuỗi tổng của nhãn hai nút con
  - Nhãn của nút con trái sẽ có thứ tự từ điển trước nút con phải
- Trọng số của nút
  - Trọng số của nút lá bằng số lần xuất hiện của nhãn ký tự
  - Trọng số của nút cha bằng tổng trọng số của các nút con
  - Nút con trái có giá trị trọng số nhỏ hơn hoặc bằng trọng số của nút con trái phải
- Mỗi cạnh sẽ được gán một giá trị: cạnh trái là 0 và cạnh phải là 1

Cây Huffman (cont.)



Hình 2.29: Cây Huffman

Cây DFS

Định nghĩa 2.10

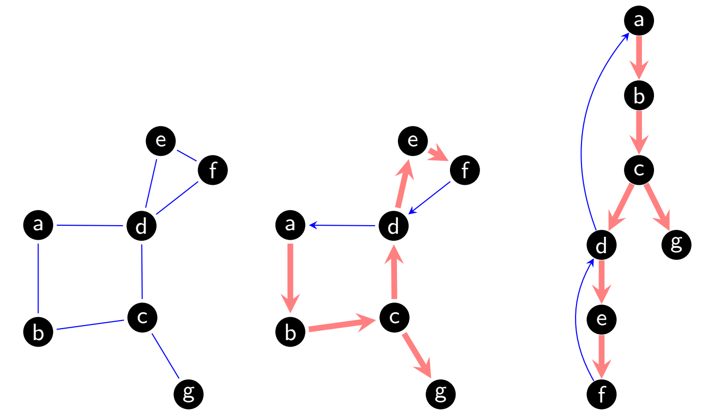
Cây DFS của một đồ thị G là cây có hướng được sinh ra bằng thuật toán duyệt DFS trên đồ thị G

Cho một đồ thị liên thông $G = (V, E)$ có n đỉnh $V = \{v_1, ..., v_n\}$.

Algorithm 5 Thuật toán xây dựng cây DFS

```
Khởi tạo cây T với đỉnh v bất kỳ
DFS_TREE(u)
procedure DFS_TREE(v)
    for mỗi đỉnh u kề với v do
        Thêm đỉnh u và cạnh (v, u) vào cây T
        DFS_TREE(u)
```


Cây DFS (cont.)



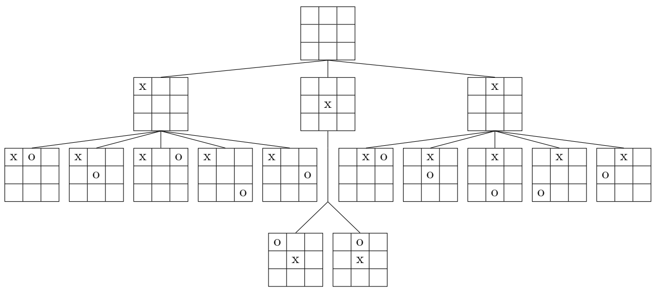
(a) đồ thị vô hướng **(b)** cây DFS **(c)** cây DFS

Hình 2.30: Đồ thị vô hướng và cây DFS

Cây trò chơi

Định nghĩa 2.11

Cây trò chơi là đồ thị trong đó mỗi nút là một trạng thái của trò chơi và mỗi cung tương đương với một bước đi



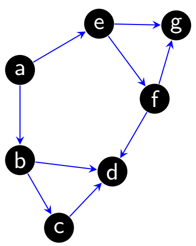
Hình 2.31: Cây trò chơi Tic-Tac-Toe

ĐỒ THỊ DẠNG DAG

Đồ thị DAG

Định nghĩa 2.12

Đồ thị có hướng và không có chu trình được gọi là đồ thị DAG (**Directed Acyclic Graph**)



Hình 2.32: Đồ thị dạng DAG

Sắp xếp thứ tự

Định nghĩa 2.13

Sắp xếp thứ tự hay **sắp xếp tô pô** cho một đồ thị có hướng G là cách đánh thứ tự các đỉnh sao cho mọi cạnh có hướng (u, v) thì u luôn có thứ tự trước v

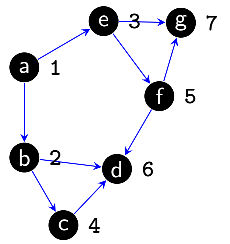
Định lý 2.9

Điều kiện cần và đủ để một đồ thị có hướng G có thể sắp xếp thứ tự là G là đồ thị DAG

Chứng minh

Sinh viên tự chứng minh ■

Sắp xếp thứ tự (cont.)



Hình 2.33: Một cách sắp thứ tự các đỉnh

Sắp xếp thứ tự (cont.)

Algorithm 6 Thuật toán sắp xếp thứ tự

```
L ← ∅
S ← Các đỉnh không có cạnh vào
while S ≠ ∅ do
    lấy một đỉnh v từ S
    thêm đỉnh v vào L
    for đỉnh u kề với v do loại bỏ cạnh (u, v) khỏi đồ thị
        if u không có cạnh vào then
            thêm u vào S
if đồ thị vẫn còn cạnh then
    xuất thông báo không thể sắp xếp thứ tự
else
    xuất danh sách L chứa các đỉnh theo thứ tự
```


Tài liệu tham khảo

Diestel, R. (2005).
Graph theory. 2005.
Springer-Verlag.

Moore, E. F. (1959).
The shortest path through a maze.
Bell Telephone System.

Rosen, K. H. and Krithivasan, K. (2012).
Discrete mathematics and its applications.
McGraw-Hill New York.

Tarjan, R. (1972).
Depth-first search and linear graph algorithms.
SIAM journal on computing, 1(2):146–160.

Tài liệu tham khảo (cont.)

Trần, T. and Dương, D. (2013).
Giáo trình lý thuyết đồ thị. 2013.
NXB Đại Học Quốc Gia TPHCM.

West, D. B. et al. (2001).
Introduction to graph theory.
Prentice hall Englewood Cliffs.