Chương 4 : Cây

Trịnh Anh Phúc, Nguyễn Đức Nghĩa [1]

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

Ngày 5 tháng 11 năm 2014

Giới thiệu

Định nghĩa và các khái niệm

Định nghĩa cây

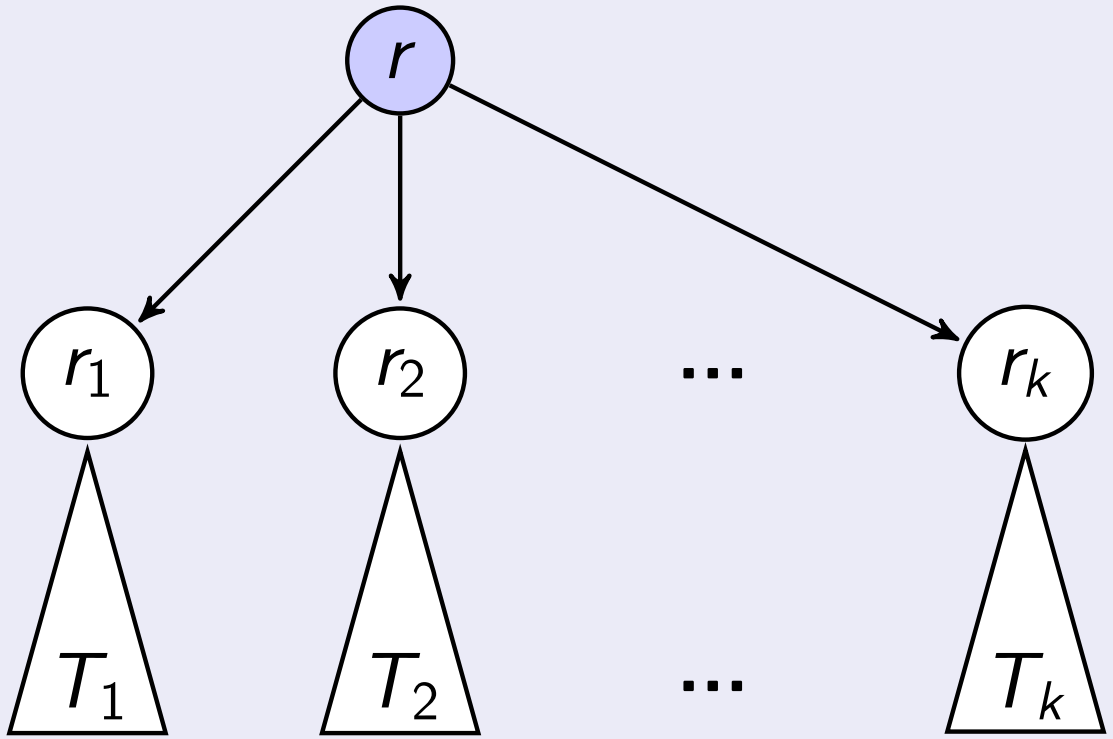
Cây bao gồm các nút, có một nút đặt biệt được gọi là nút gốc (root) và các cạnh nối các nút. Cây được định nghĩa đệ qui như sau

- **Bước cơ sở** : một nút r được coi là cây và r được gọi là gốc cây.
- **Bước đệ qui** : Giả sử $T_1, T_2, \cdots, T_k$ là các cây với gốc là $r_1, r_2, \cdots, r_k$, ta có thể xây dựng cây mới bằng cách đặt r làm nút cha (parent) của các nút $r_1, r_2, \cdots, r_k$. Trong cây mới tạo ra r là gốc và $T_1, T_2, \cdots, T_k$ là các cây con của gốc r. Các nút $r_1, r_2, \cdots, r_k$ được gọi là con của nút r.

Định nghĩa và các khái niệm

Định nghĩa cây (tiếp)

Hình minh họa định nghĩa đệ qui của cây



Định nghĩa và các khái niệm

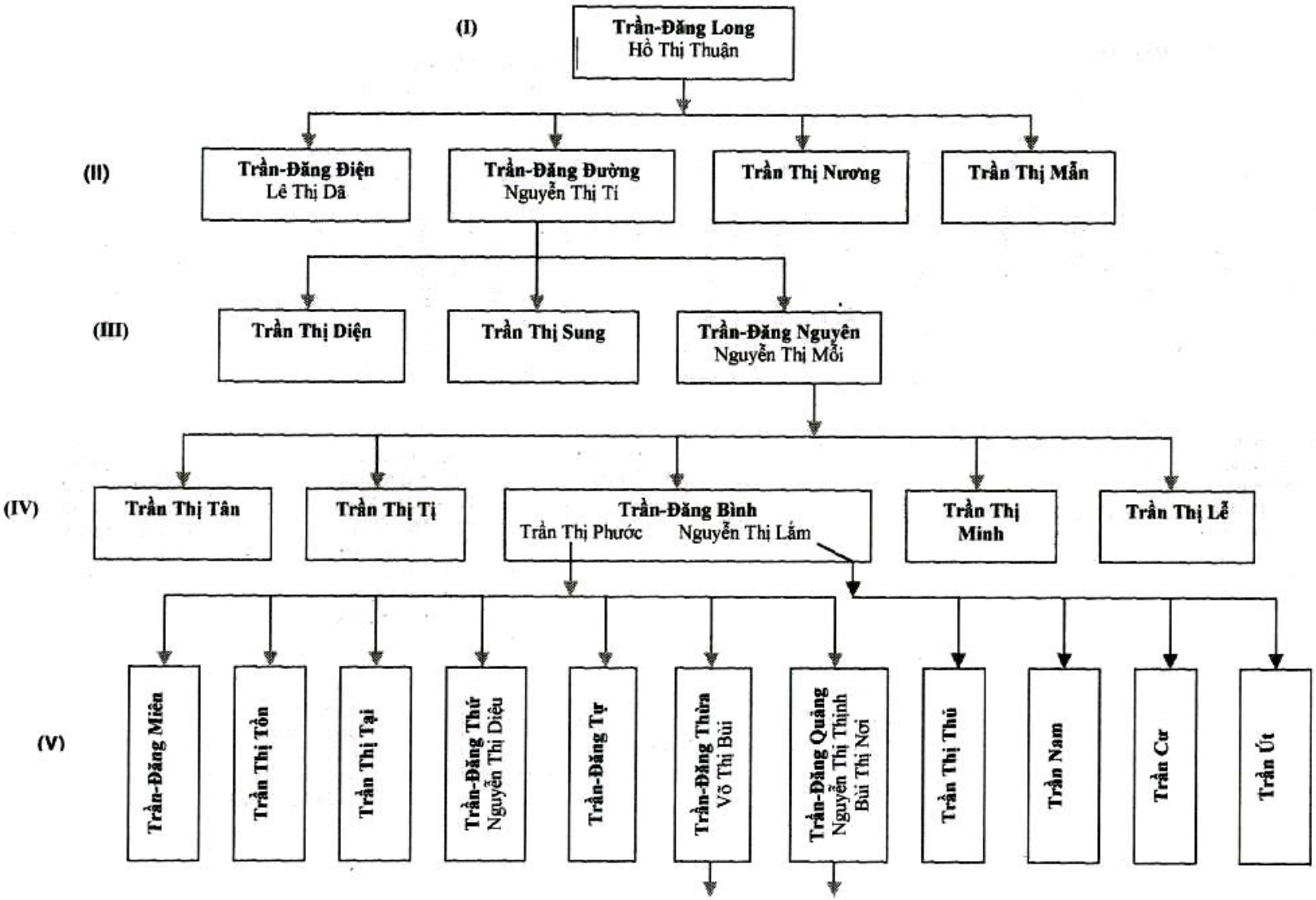
Các ứng dụng của dữ liệu trừu tượng cây

Cây trong ứng dụng thực tế

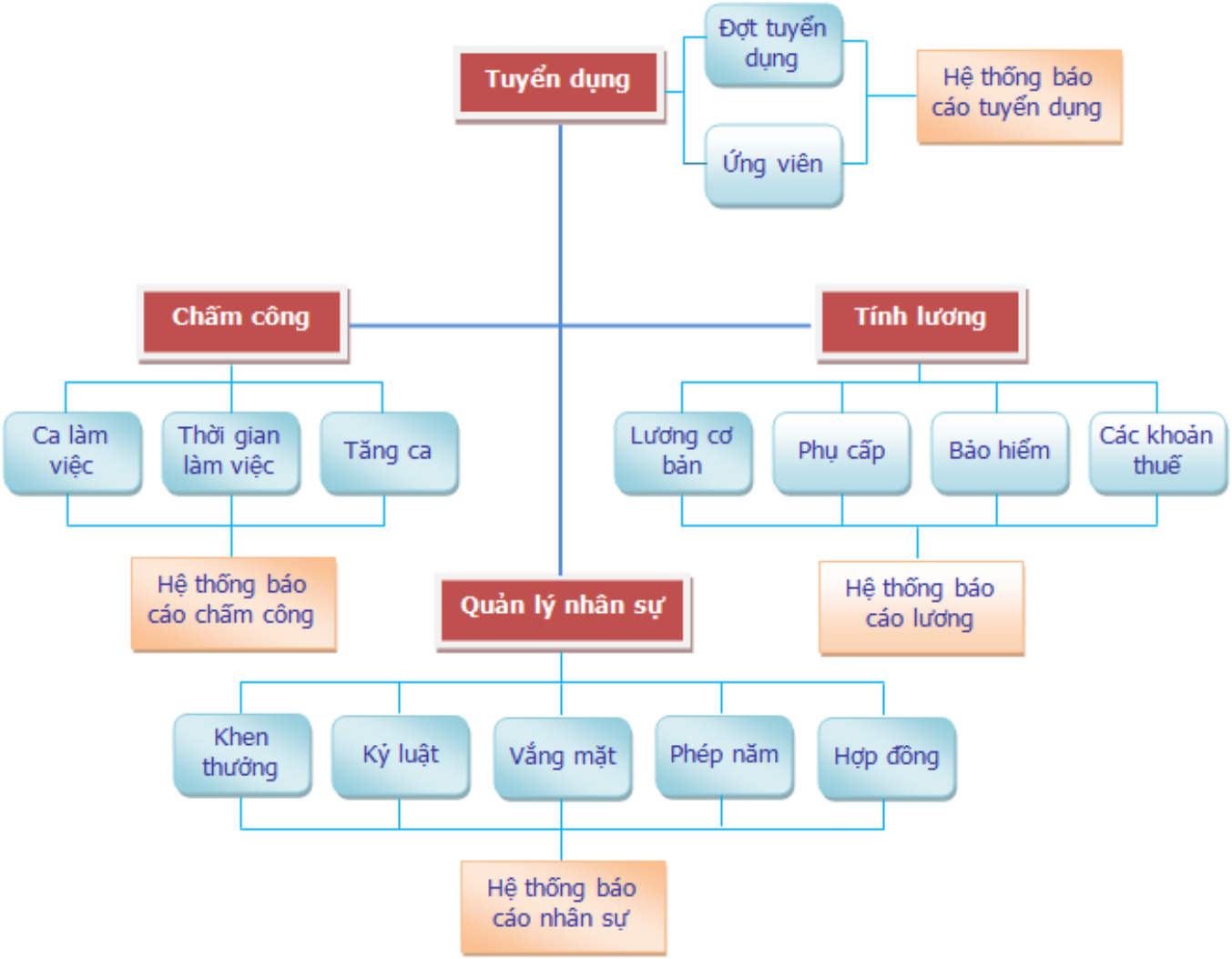
- Biểu đồ lịch thi đấu
- Cây gia phả
- Biểu đồ phân cấp quản lý
- Cây thư mục quản lý file
- Cây biểu thức
- ....

Sau đây là một vài hình ảnh minh họa các ứng dụng này

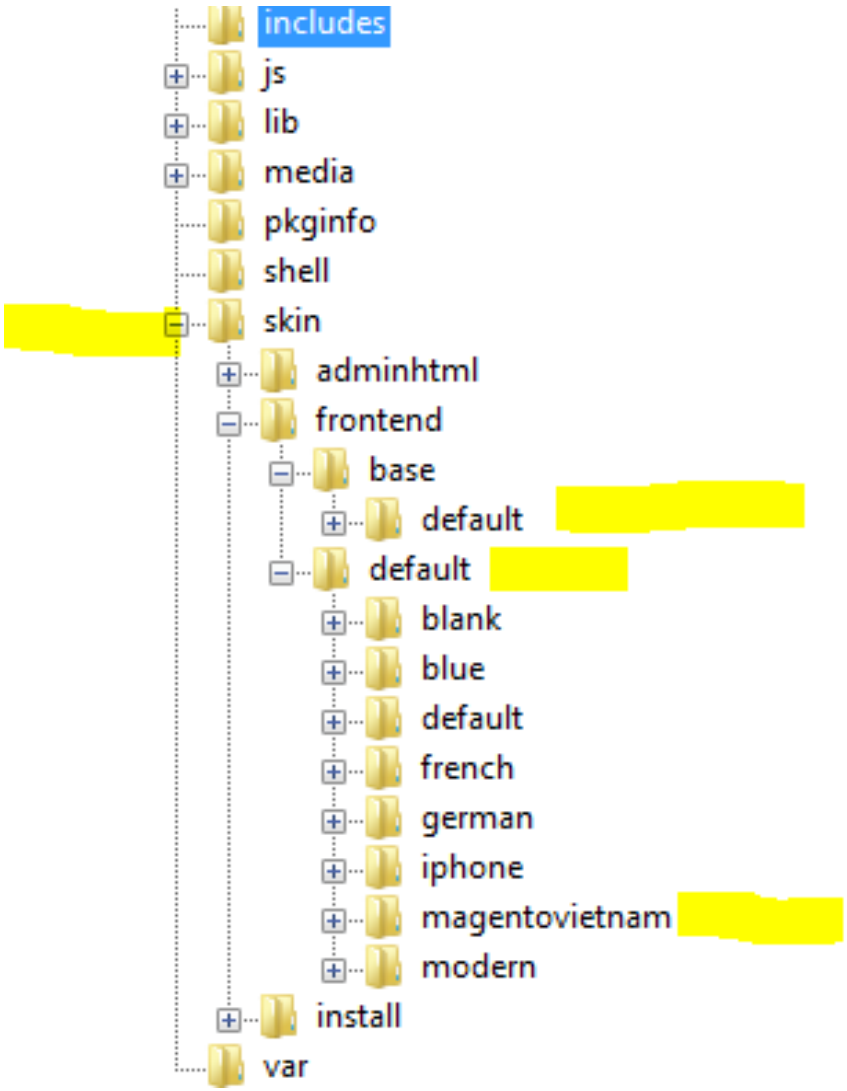
Ứng dụng cây gia phả



Ứng dụng biểu đồ phân cấp quản lý



Ứng dụng cây thư mục



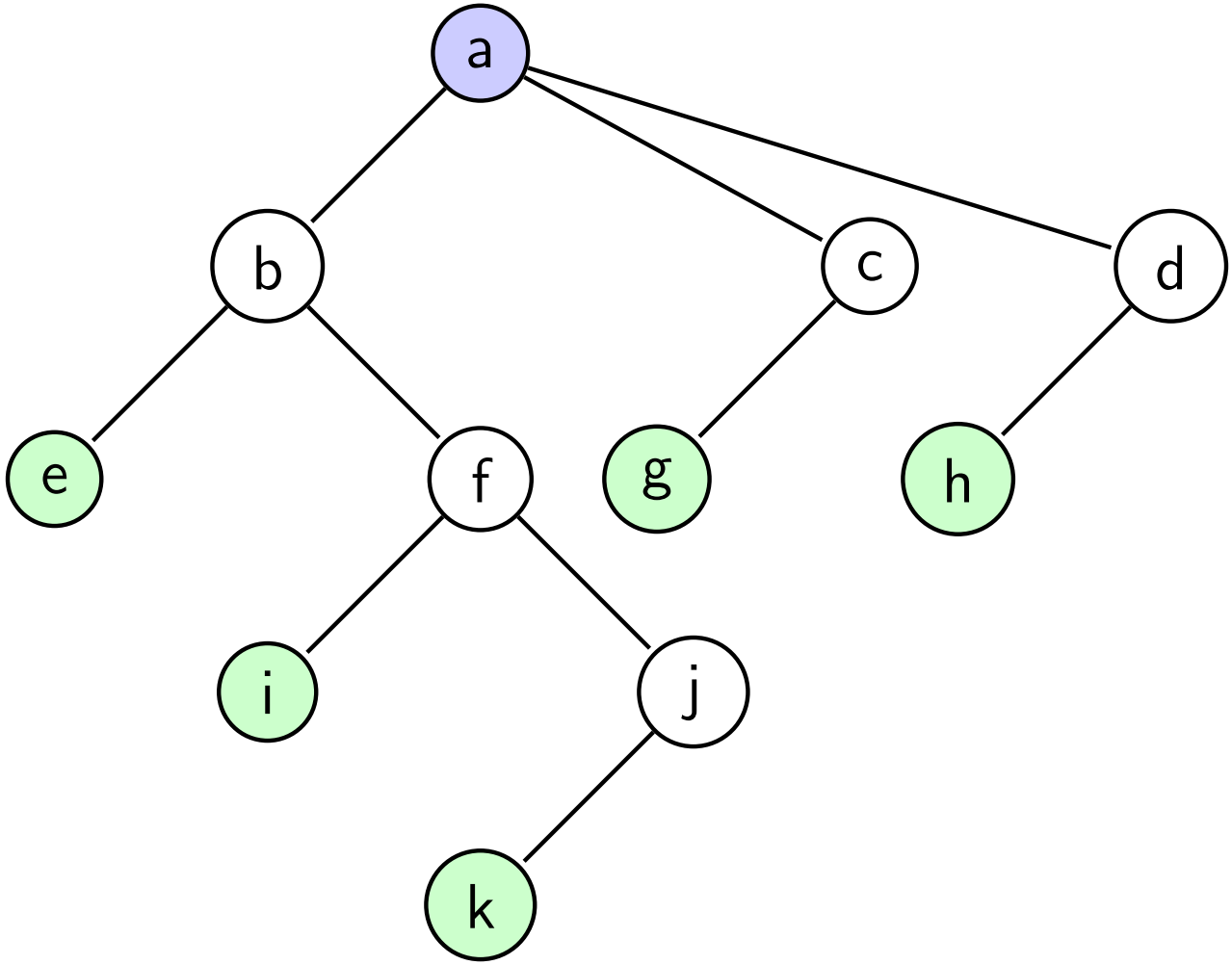
Cây

Các thuật ngữ chính

- Nút - node
- Gốc - root
- Lá - leaf
- Con - child
- Cha - parent
- Tổ tiên - ascentors
- Hậu duệ - descendants
- Anh em - sibling
- Chiều cao - hight
- Nút trong - internal node
- Đường đi - path

Cây

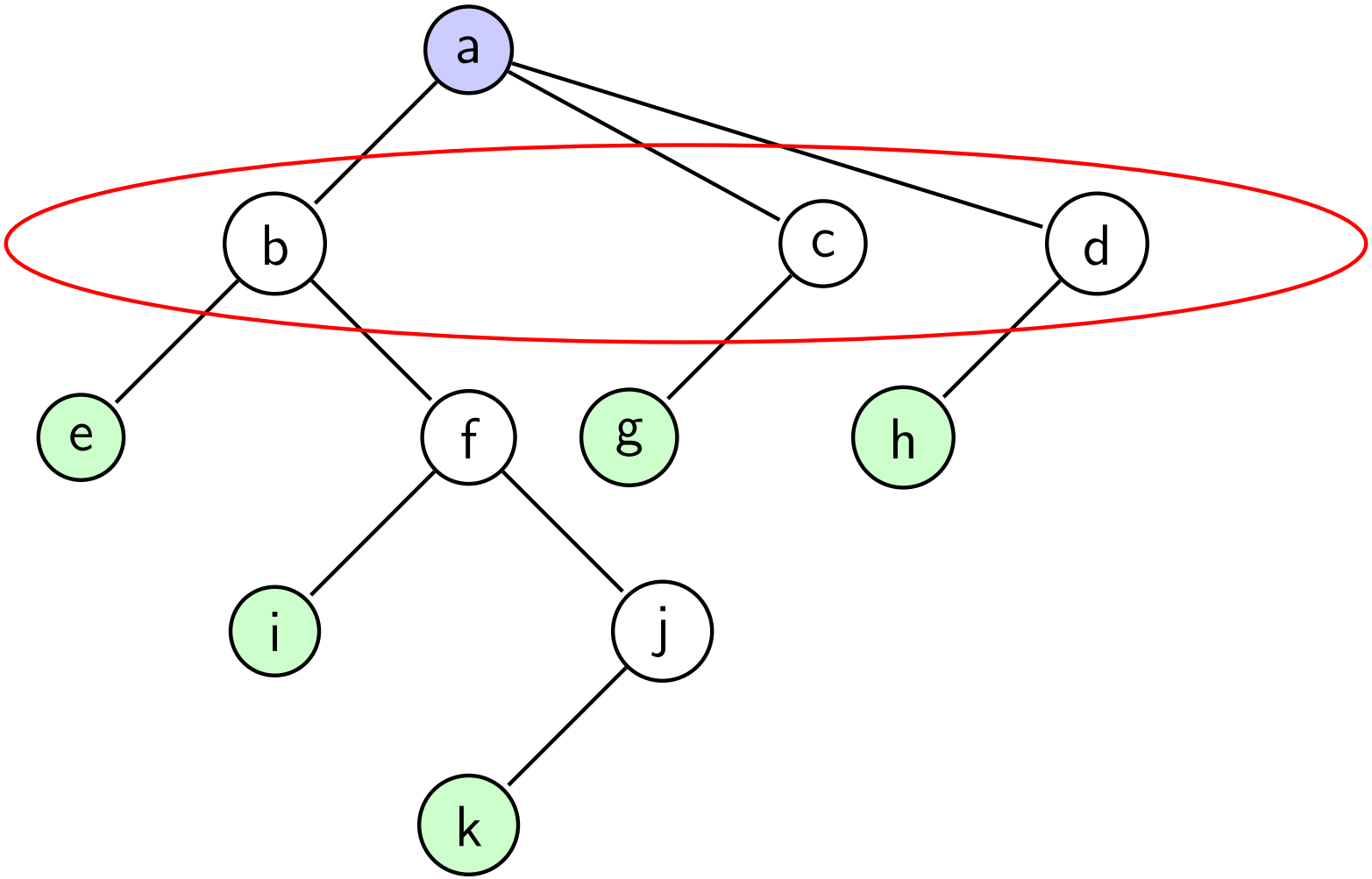
Phân loại các nút trong cây



Chú thích : Nút gốc mầu xanh thẫm, nút lá mầu xanh lá cây còn nút trong mầu trắng.

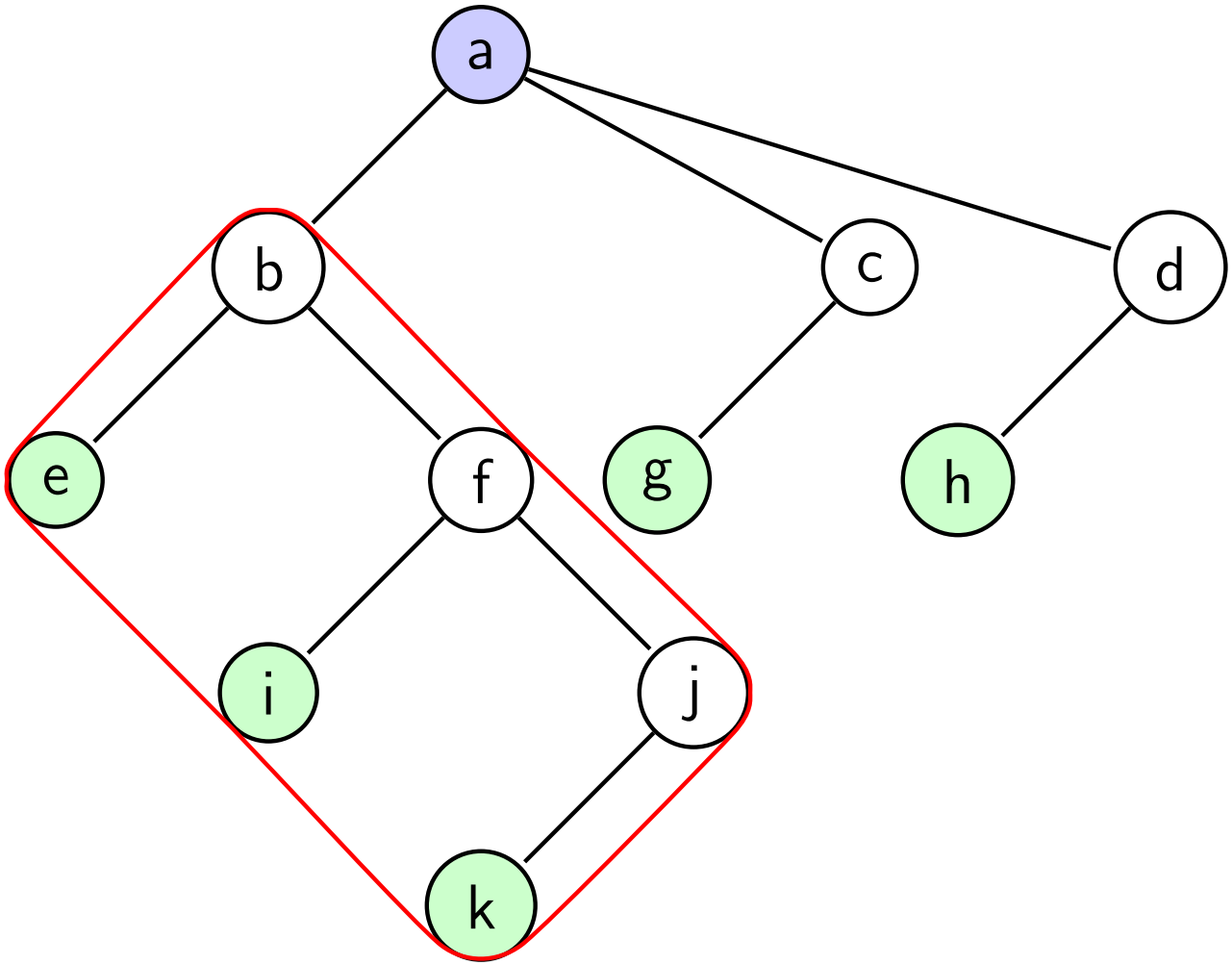
Cây

Các nút cùng cha gọi là các nút anh&em. Trong hình là ba nút b,c,d có cùng nút cha là a, được đánh dấu bởi hình elíp đỏ.



Cây

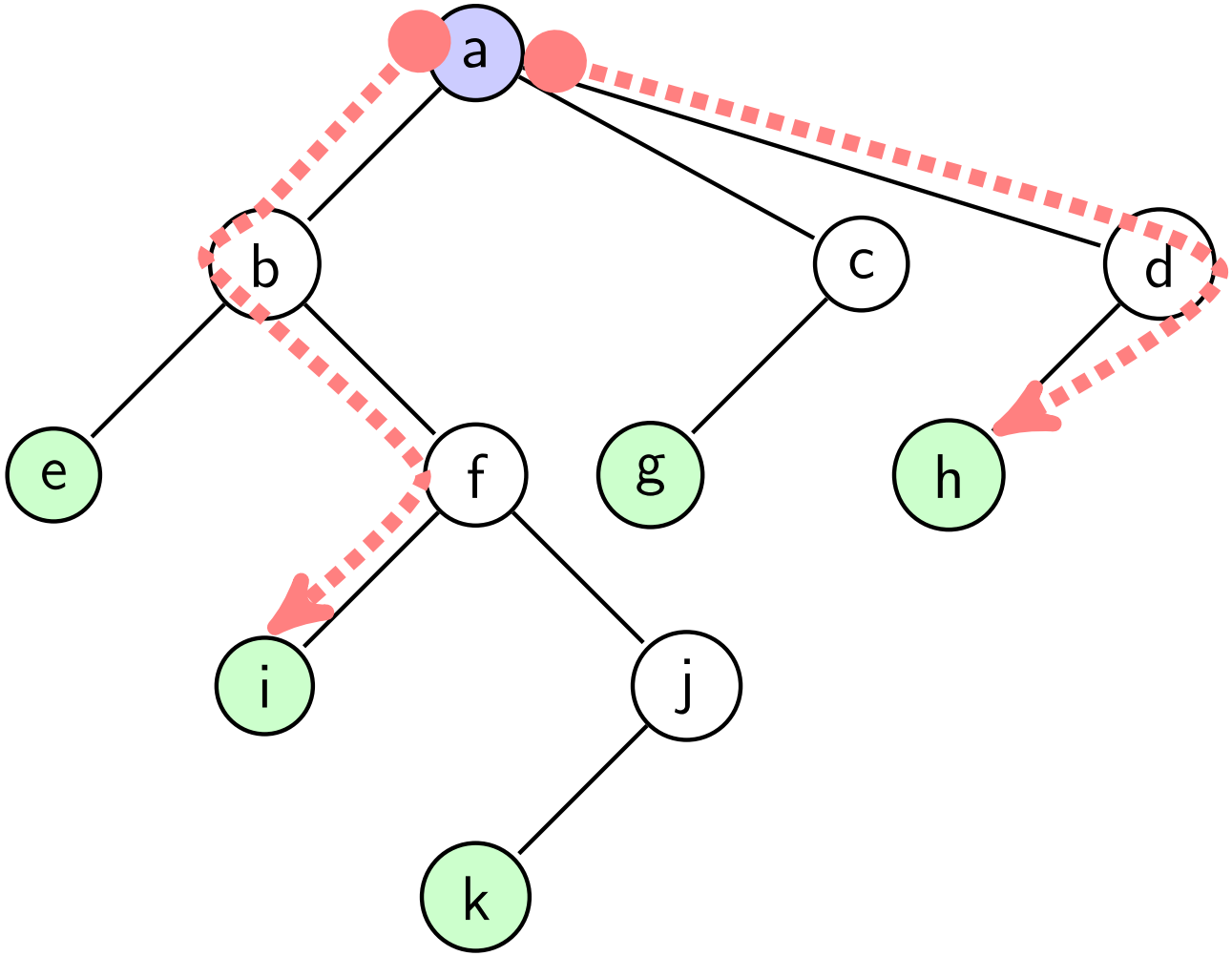
Cây con của nút gốc a,



Chú thích : Vòng tròn bao mầu đỏ chỉ ra một cây con của nút gốc a.

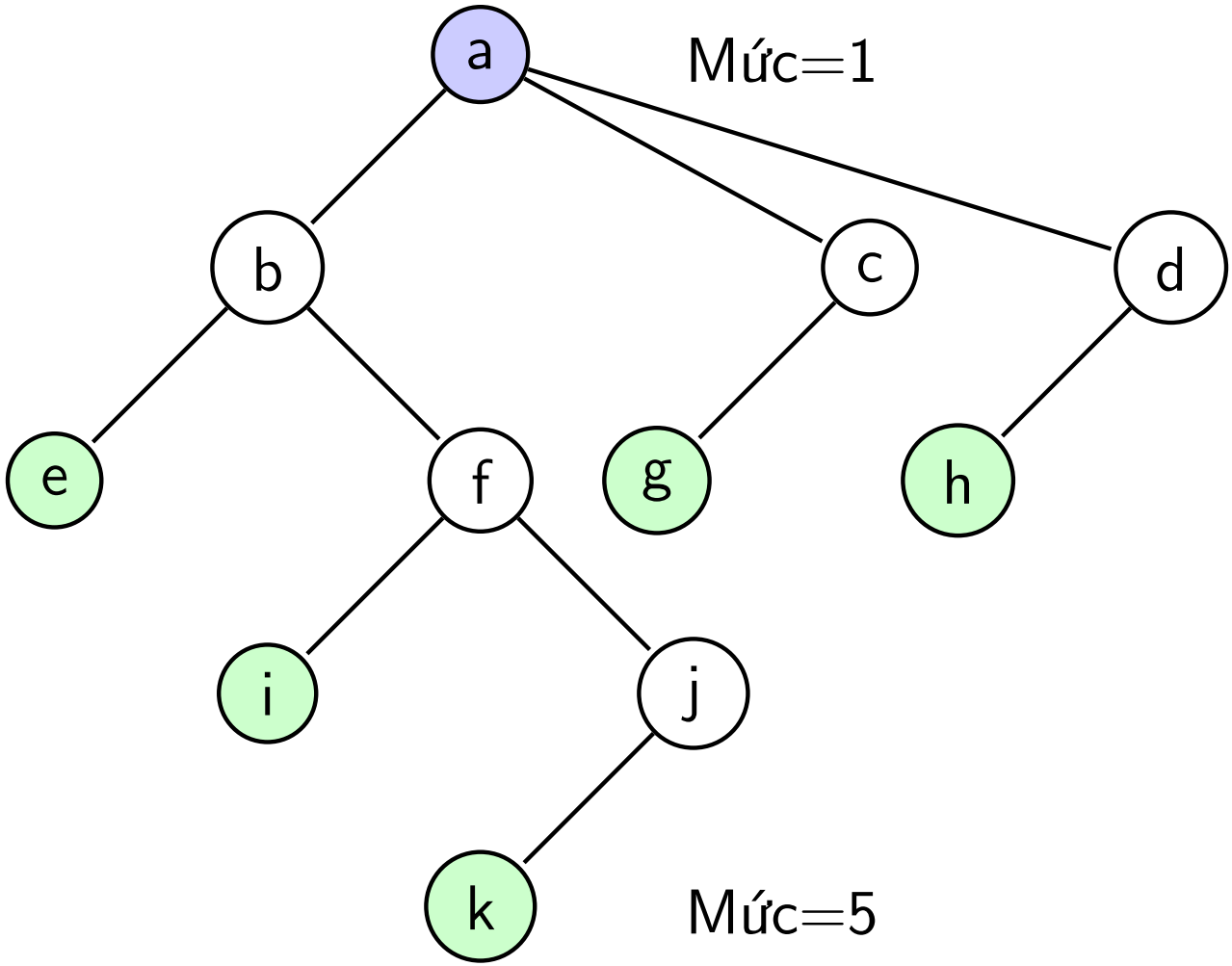
Cây

Đường đi trên cây từ nút gốc a đến các nút lá i và h (gạch nét dứt mầu đỏ). Đường thứ nhất {a,b,f,i} và đường thứ hai là {a,d,h}.



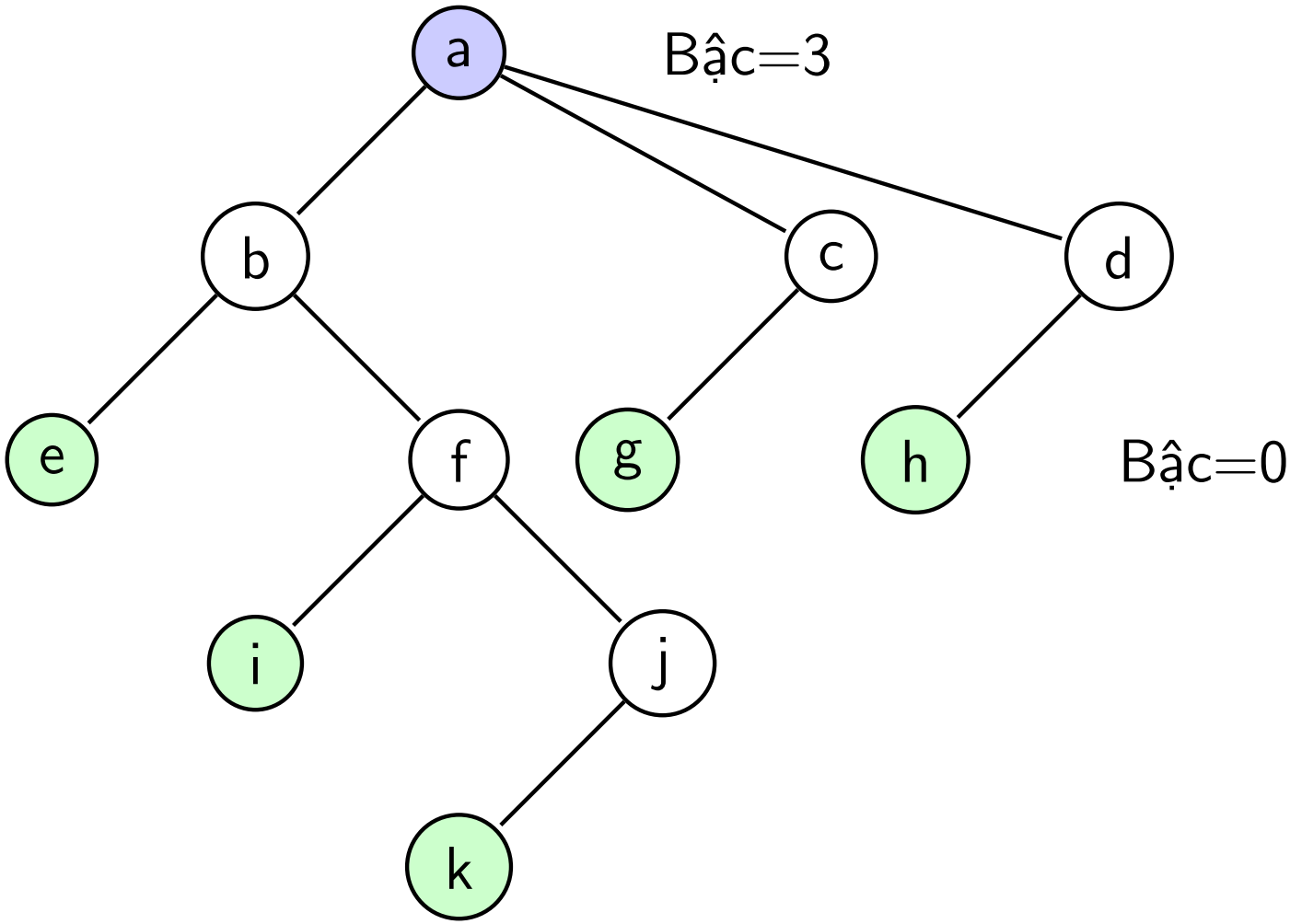
Cây

Độ cao của cây và độ sâu của cây. Do nút gốc có mức 1 nên nút lá xa nhất k có mức chính là độ cao và độ sâu của cây, vậy cây trên có độ cao là 5.



Cây

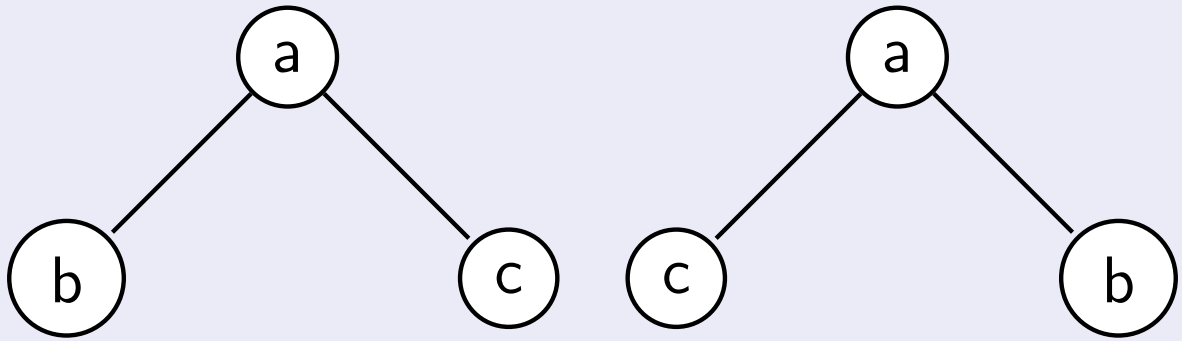
Bậc (degree) của một nút là số nút con của nó. Vậy nút gốc a có bậc 3 trong khi nút lá như h luôn có bậc 0.



Cây

Cây có thứ tự

Thứ tự của các nút trên cây : Các nút con của một nút thường được sắp xếp theo thứ tự từ **trái sang phải**

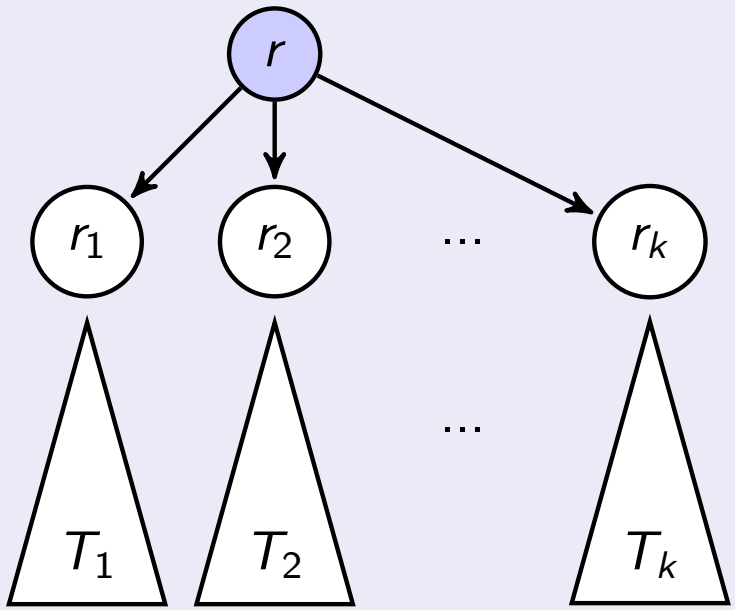


Như vậy rõ ràng hai cây trên **khác nhau** do thứ tự nút con của nút a là khác nhau. Hay nút b được xếp trước nút c trong cây bên trái, trong khi nó được xếp sau nút c trong cây bên phải.

Xếp thứ tự các nút

Có ba cách thông thường để xác định thứ tự các nút

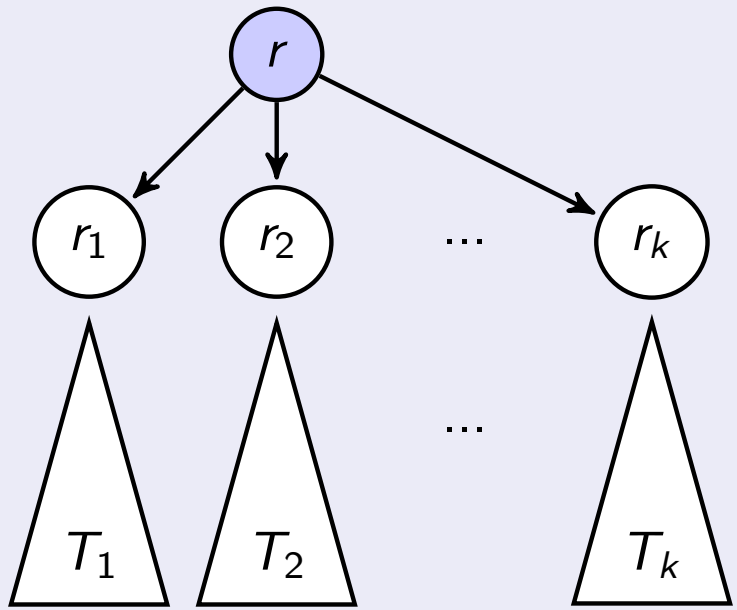
1. Thứ tự trước (Preorder)
2. Thứ tự giữa (Inorder)
3. Thứ tự sau (Postorder)



Cây

Thứ tự trước - Preorder traversal

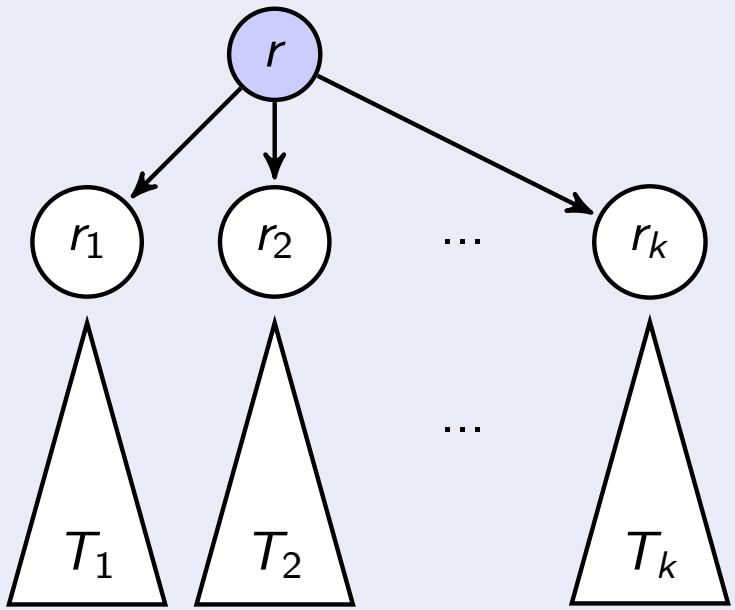
- Gốc r của cây
- tiếp đến là các nút của T_1 được duyệt theo thứ tự trước
- tiếp đến là các nút của T_2 được duyệt theo thứ tự trước...
- và cuối cùng là các nút của T_k được duyệt theo thứ tự trước



Cây

Thứ tự sau - Postorder traversal

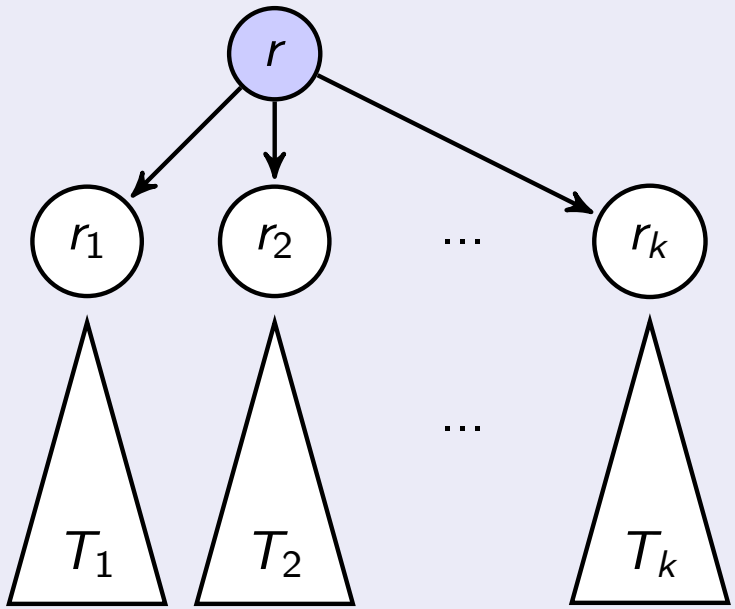
- các nút của cây T_1 theo thứ tự sau
- tiếp đến là các nút của T_2 được duyệt theo thứ tự sau...
- tiếp đến là các nút của T_k được duyệt theo thứ tự sau
- sau cùng là nút gốc r



Cây

Thứ tự giữa - Inorder traversal

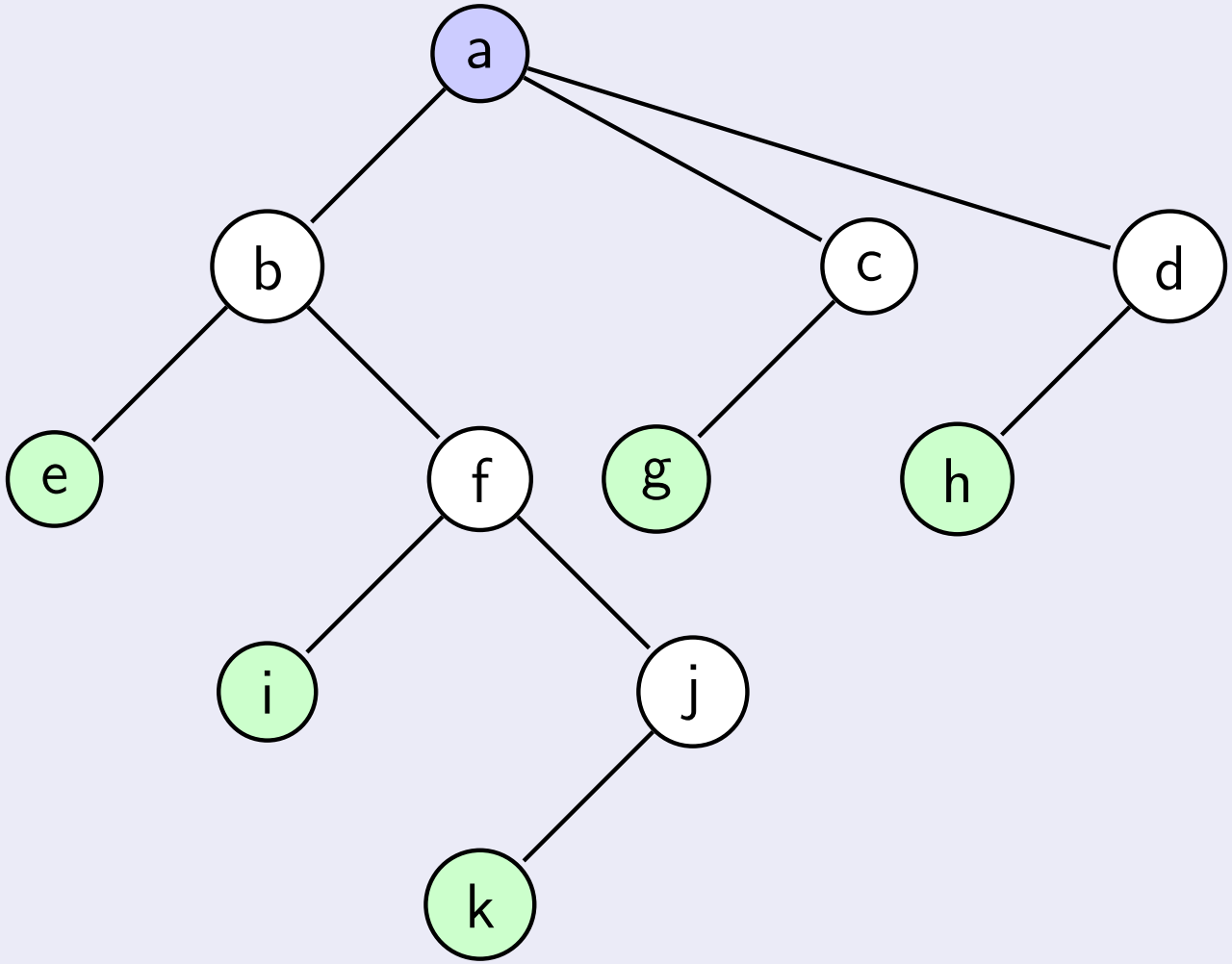
- các nút của cây T_1 theo thứ tự giữa
- tiếp đến nút gốc r
- tiếp đến các nút của cây $T_2, \cdots, T_k$ mỗi nhóm nút được xét theo thứ tự giữa.



Cây

Dãy thứ tự trước

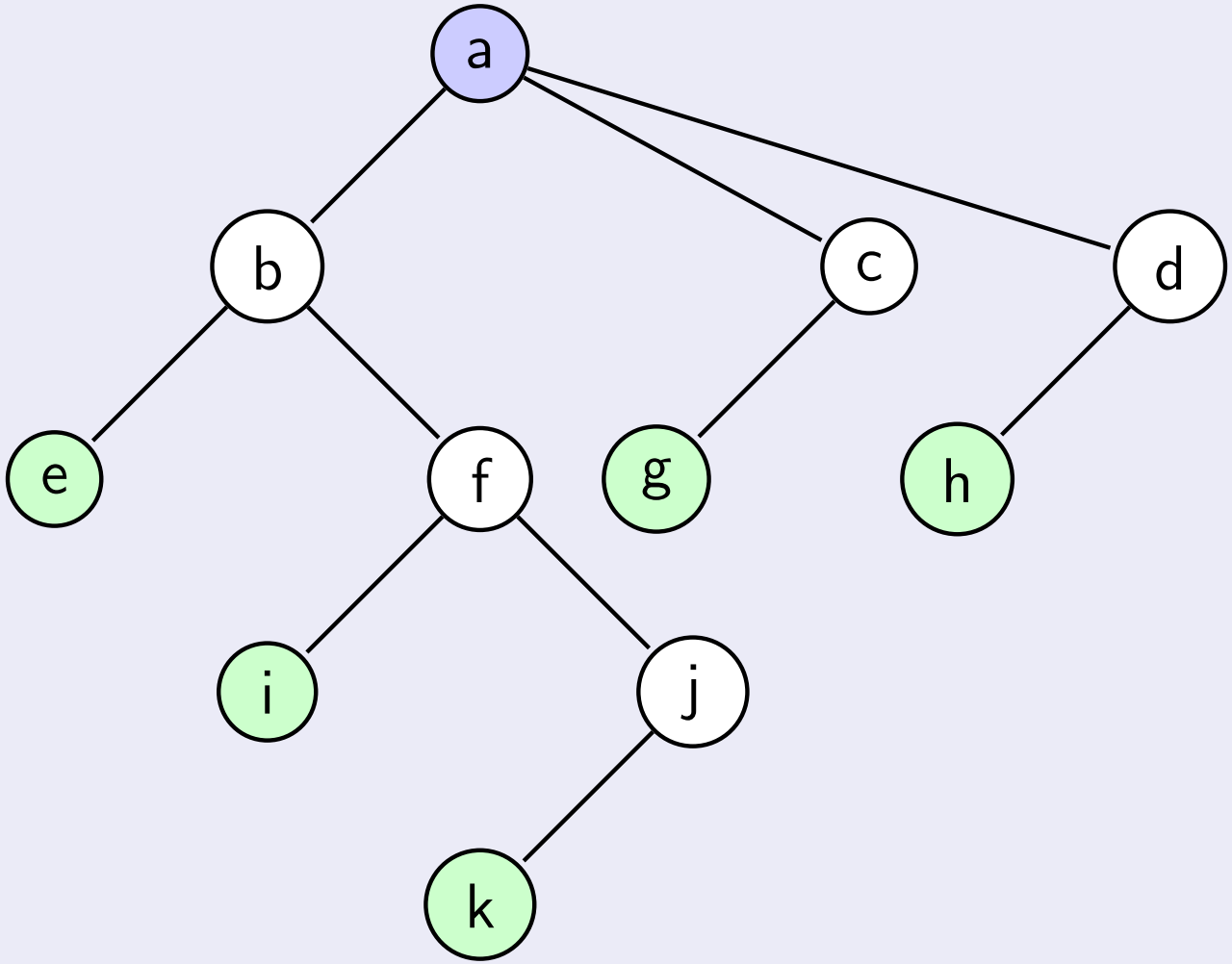
Cây minh họa có dãy thứ tự trước là : a,b,e,f,i,j,k,c,g,d,h



Cây

Dãy thứ tự sau

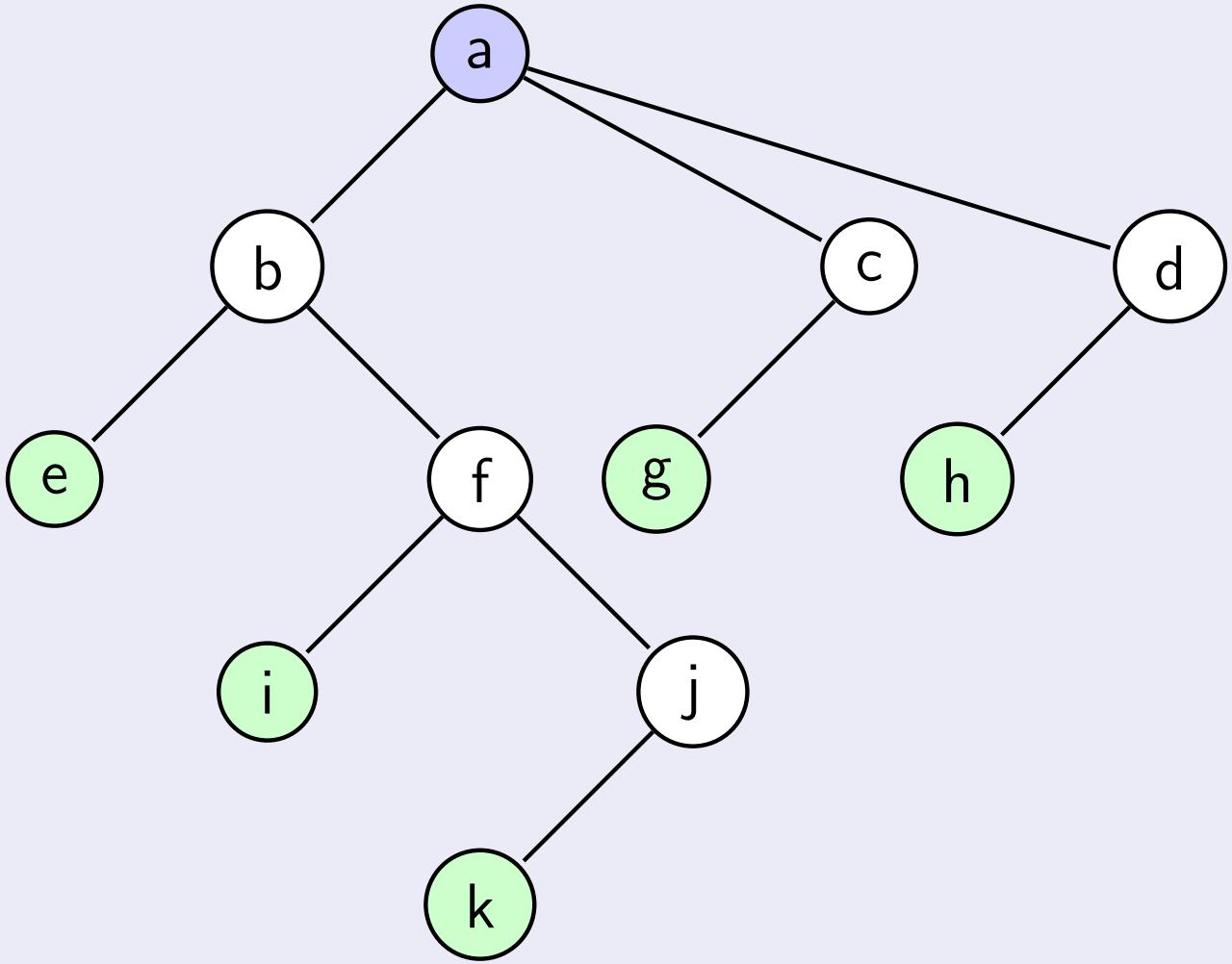
Cây minh họa có dãy thứ tự sau là : k,j,i,f,e,g,h,d,c,b,a



Cây

Dãy thứ tự giữa

Cây minh họa có dãy thứ tự giữa là : e,i,k,j,f,b,g,c,h,d,a



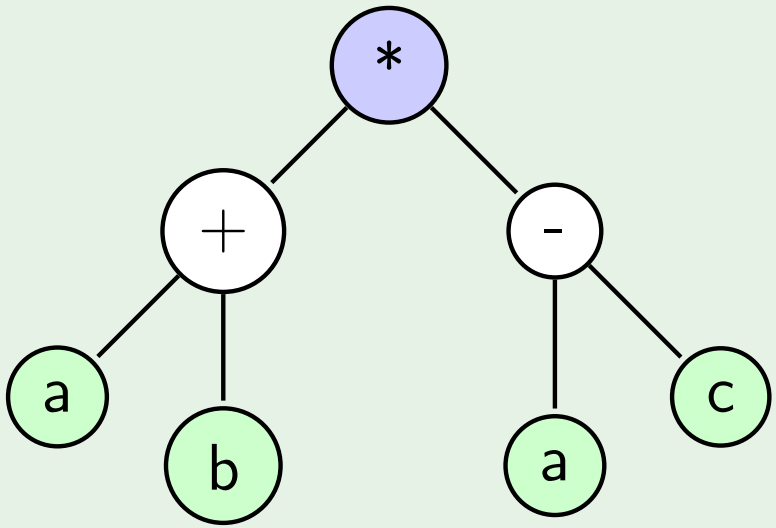
Cây

Cây có nhãn - labaled tree

Thông thường người ta gán cho mỗi nút nột nhãn hoặc một giá trị, cũng giống như ta gán mỗi nút trong danh sách tuyến tính một phần tử (element). Nghĩa là nhãn của nút không chỉ là tên gọi mà mang ý nghĩa giá trị của nút đó. Ví dụ rõ nhất là cây biểu thức ...

Cây

Cây biểu thức - expression tree



Biểu thức : (a+b)*(a-c)

Qui tắc biểu diễn cây biểu thức là :

- Mỗi nút lá là một số hạng và chỉ gồm số hạng đó
- Mỗi nút trong được gán một phép toán. Với phép toán hai ngôi E_1 q E_2, ví dụ của q = {+,-,*,/}, thì cây con trái biểu diễn biểu thức E_1 còn cây con phải biểu diễn E_2.

Cây

Cấu trúc dữ liệu trừu tượng cây

Cũng như với danh sách, ta cũng có các phép toán làm việc với nó

- **parent(T,n)** hàm này trả lại nút cha của của nút n trong cây T.
- **leftmostchild(n,T)** hàm trả lại nút con trái nhất của nút n trong cây T.
- **rightsibling(n,T)** hàm trả lại em phải của nút n trong cây T.
- **label(n,T)** trả lại nhãn của nút n trong cây T.
- **root(T)** trả lại nút gốc của cây T.
- **makeNull(T)** biến cây T thành rỗng.

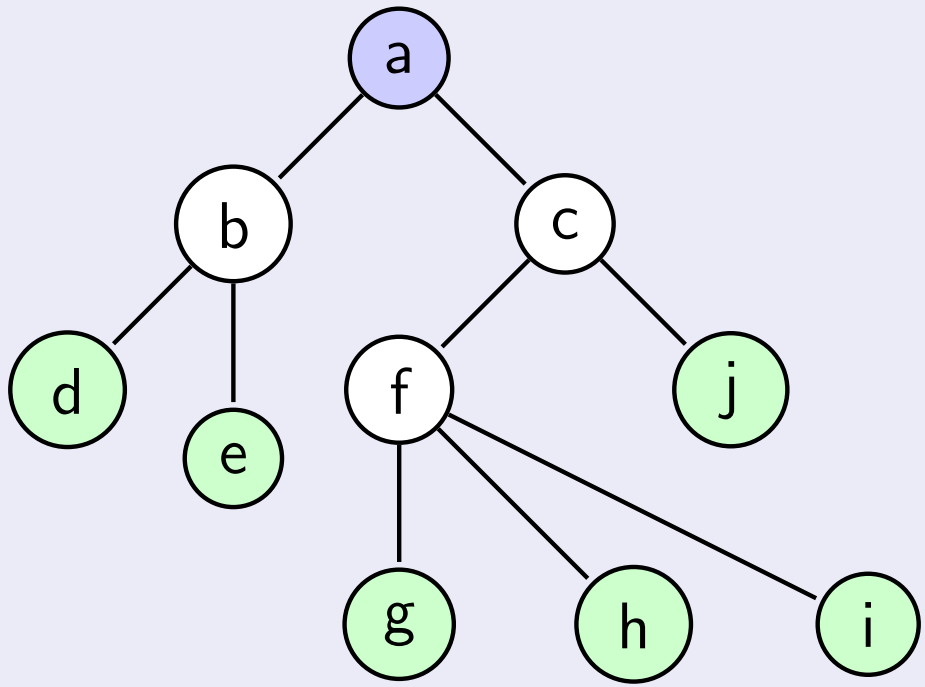
Cũng có ba cách cài đặt cây

- dùng mảng (Array representation)
- danh sách con (Lists of children representation)
- dùng con trái và em phải (The most-child, right-sibling representation)

Cây

Cấu trúc dữ liệu trừu tượng cây dùng mảng

Giả sử T là cây với các nút đặt tên là 1,2,...,n. Khi dùng mảng, ta đặt $A[i] = j$ nếu nút j là cha của nút i và $A[i] = 0$ nếu nút i là nút góc.



	a	a	b	b	c	c	f	f	f

Cấu trúc dữ liệu trừu tượng cây dùng mảng (tiếp)

Hạn chế của biểu diễn cây dùng mảng

- Cách dùng mảng không thích hợp với thao tác con. Khi cho nút n, ta sẽ mất thời gian để xác định các con của nút n, hoặc mức hay chiều cao của cây.
- Cách dùng mảng không cho biết thứ tự các nút con có cùng nút cha. Vì thế các phép toán như **leftmostchild()** hay **rightsibling()** là không thực hiên được.

Bởi vậy, cách biểu diễn này chỉ dùng trong một số trường hợp.

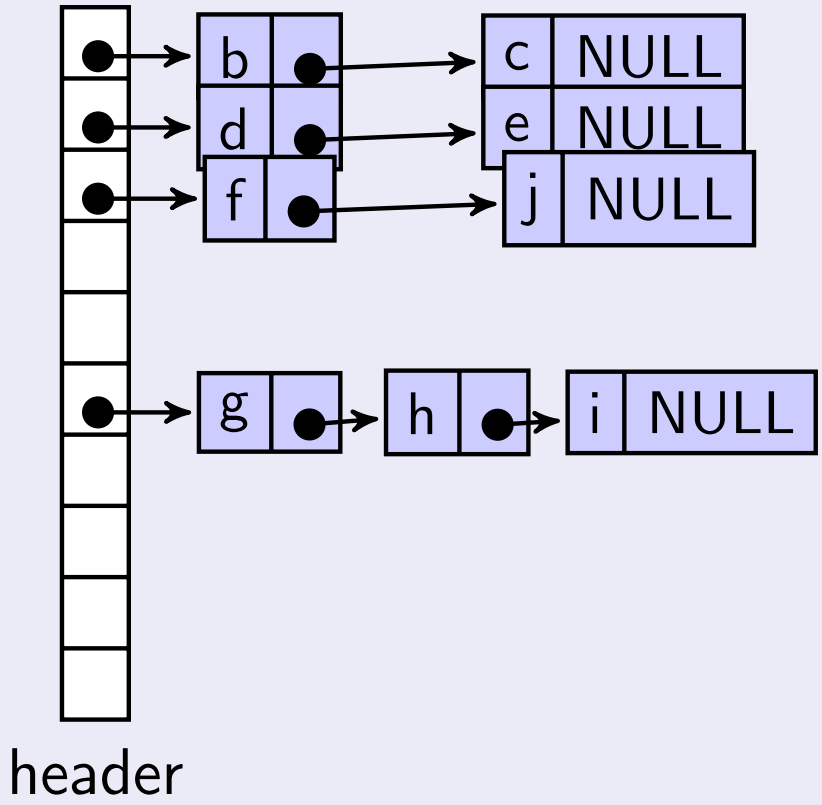
Cấu trúc dữ liệu trừu tượng cây dùng danh sách các con

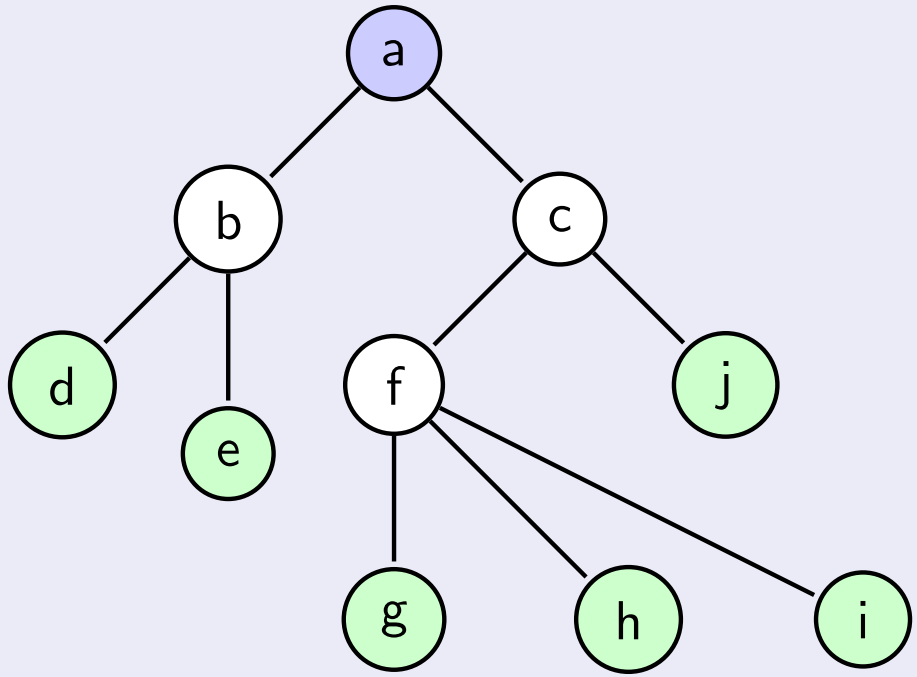
Trong trường hợp này, với mỗi nút của cây ta cất giữ một danh sách kết nối các con của nó. Danh sách con được dùng bởi danh sách móc nối đã giới thiệu trong chương trước. Cần có một mảng các con trỏ trỏ đến đầu các danh sách con tương ứng với từng nút 1,2,...,n

$$header[i]$$

trong đó i là số tương ứng nhãn các nút trong cây.

Cấu trúc dữ liệu trừu tượng cây dùng danh sách các con (tiếp)





Cây

Cấu trúc dữ liệu trừu tượng cây dùng con trái và em kế cận phải

Theo nhận xét, mỗi một nút của cây chỉ có thể có :

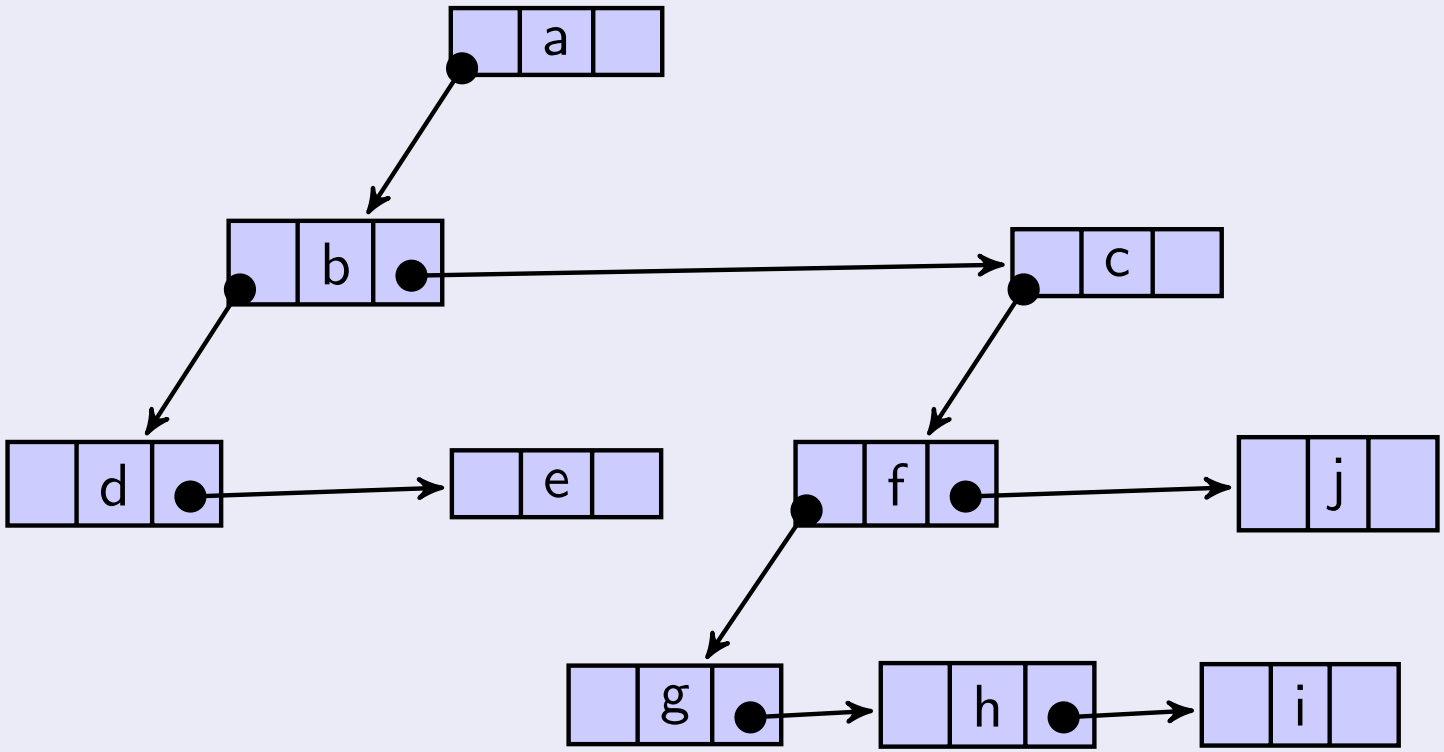
- hoặc là không có con, hoặc có con đúng một con nút cực trái.
- hoặc không có em kế cận phải, hoặc có đúng một nút em kế cận phải.

Vì vậy để biểu diễn cây ta có thể lưu trữ thông tin về con cực trái và em kế cận phải của mỗi nút. Ta có thể sử dụng mô tả trong C như sau

```
struct Tnode{
    char label;
    struct Tnode *leftmostchild;
    struct Tnode *rightsibling;
}
typedef struct Tnode treeNode;
treeNode Root;
```

Cấu trúc dữ liệu trừu tượng cây dùng con trái và em kế cận phải (tiếp)

Tiếp tục với ví dụ minh họa trước đó, mỗi nút được biểu diễn bởi 3 phần. Phần trái là con trỏ trỏ đến con trái nhất, phần giữa là nhãn nút và phần bên phải là con trỏ trỏ đến nút em kế cận phải.



Cây nhị phân

Cây nhị phân - binary tree

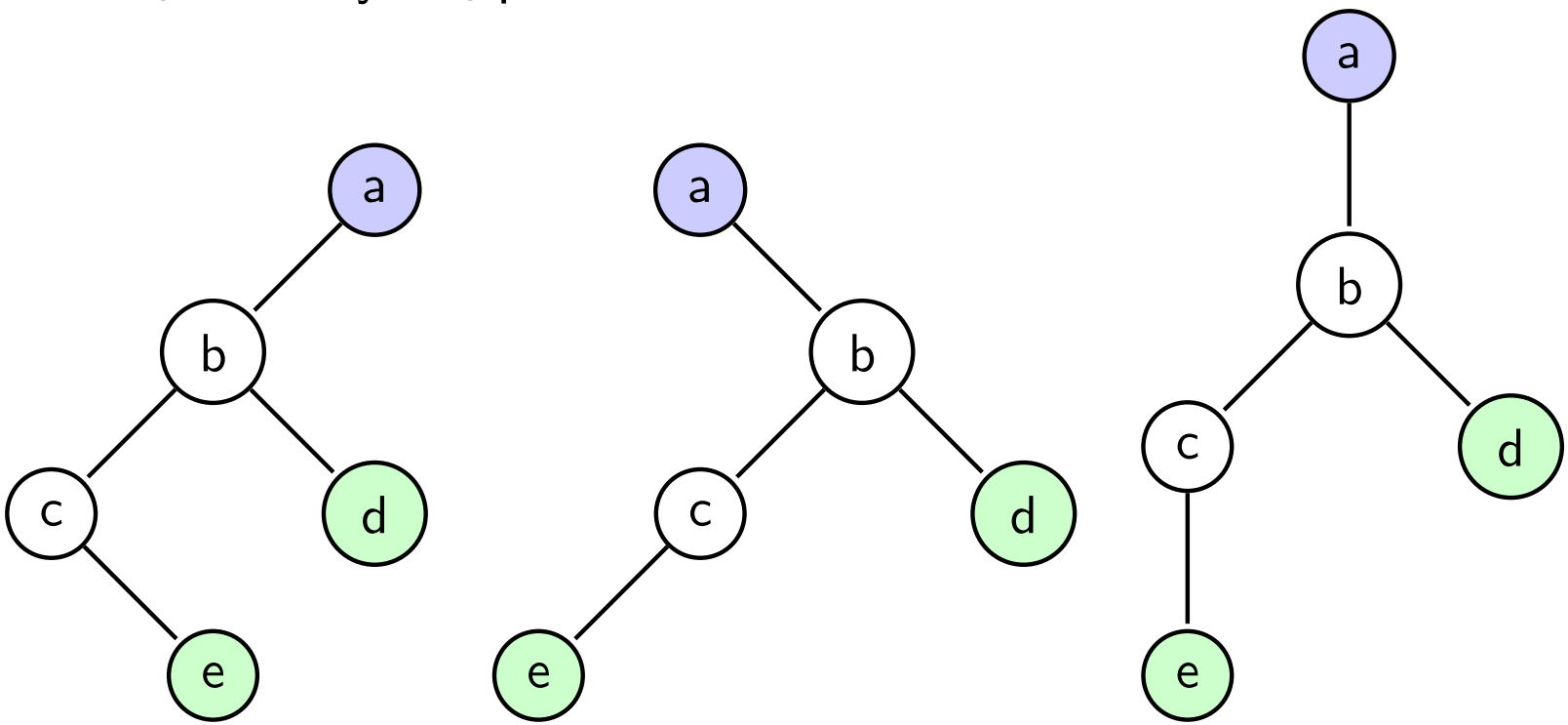
Định nghĩa : Cây nhị phân là cây mà mỗi nút có nhiều nhất là hai nút con. Vì mỗi nút chỉ có hai con nên ta sẽ gọi chúng là con trái và con phải. Chú ý là cây nhị phân giản lược so với cây tổng quát nên ta không cần xác định thứ tự các nút con.

Tính chất của cây nhị phân

- Số đỉnh lớn nhất ở trên mức i của cây nhị phân là 2^{i-1}, với $i \geq 1$
- Một cây nhị phân với chiều cao h có không quá $2^h - 1$ nút, với $h \geq 1$
- Một cây nhị phân có n nút có chiều cao tối thiểu là $\lceil \log_2(n+1) \rceil$

Cây nhị phân

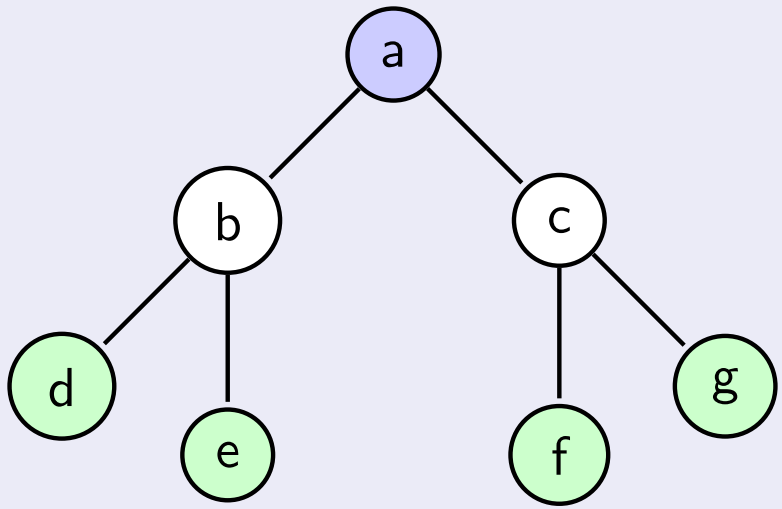
Minh họa ba cây nhị phân



Cây nhị phân

Cây nhị phân đầy đủ - full binary tree

Định nghĩa : Cây nhị phân đầy đủ là cây nhị phân mà mỗi nút có đúng hai nút con đồng thời các nút lá cùng độ sâu.



Tính chất của cây nhị phân đầy đủ

- Cây nhị phân đầy đủ với độ sâu d có $2^d - 1$ nút.

Cây nhị phân

Cây nhị phân hoàn chỉnh - complete binary tree

Định nghĩa Cây nhị phân độ sâu d thỏa mãn :

- là cây nhị phân đầy đủ nếu không tính đến các nút ở độ sâu d, hay mức cao nhất.
- tất cả các nút tại độ sâu d lệch sang trái nhất có thể.

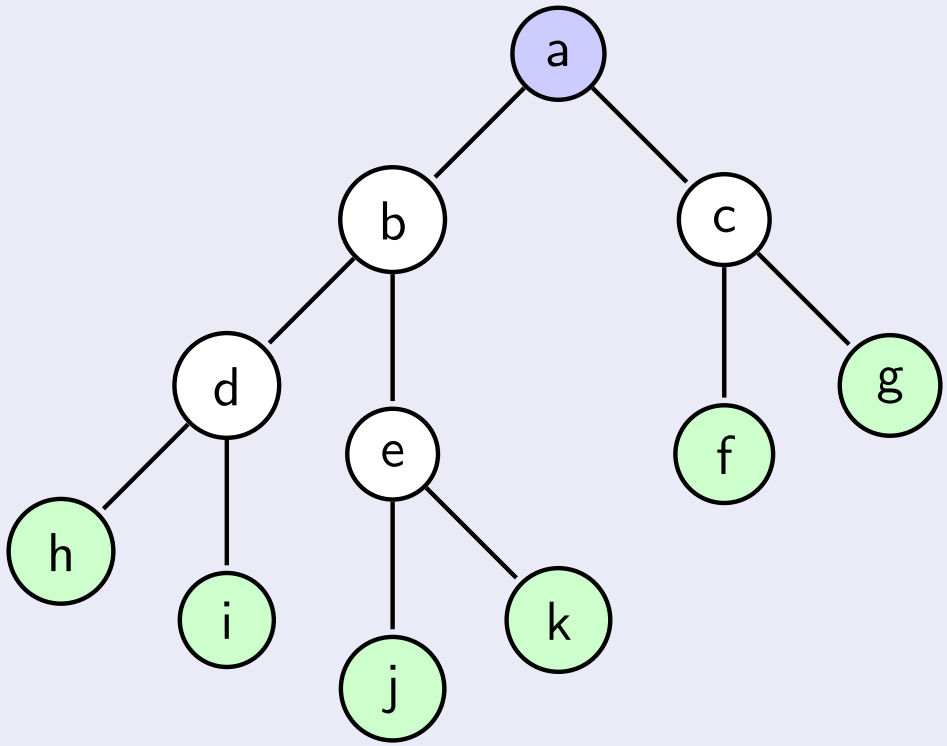
Tính chất

- Cây nhị phân hoàn chỉnh có độ sâu d thì số nút của nó nằm trong khoảng từ 2^{d-1} đến $2^d - 1$

Cây nhị phân

Cây nhị phân hoàn chỉnh (tiếp)

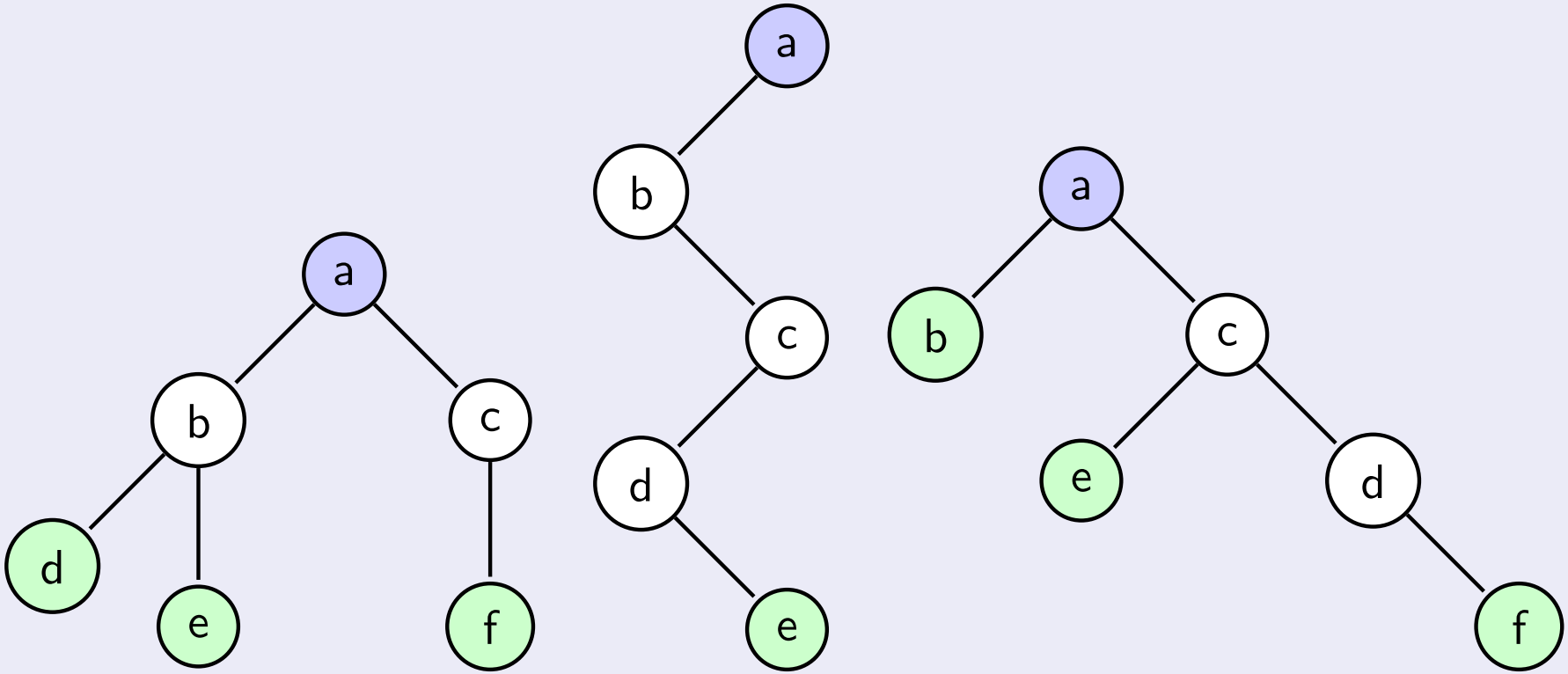
Ví dụ về cây nhị phân hoàn chỉnh



Cây nhị phân

Cây nhị phân cân đối - balanced binary tree

Định nghĩa Cây nhị phân được gọi là cân đối nếu chiều cao cây con trái và cây con phải của các nút là không lệch nhau quá một đơn vị.



Cây nhị phân

Biểu diễn cây nhị phân

Để biểu diễn cây nhị phân trong máy tính, ta cũng có hai cách

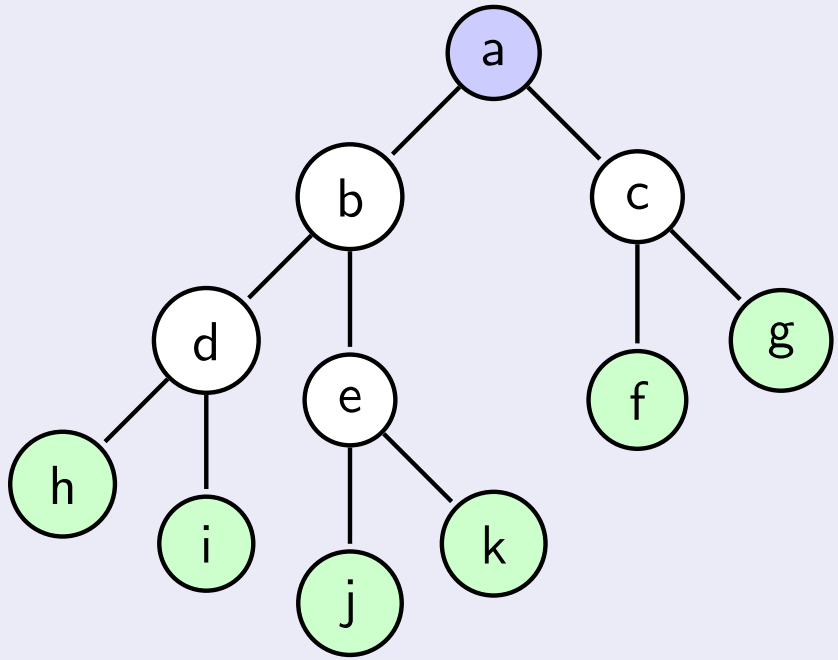
- sử dụng mảng
- sử dụng con trỏ

Khi biểu diễn cây nhị phân sử dụng mảng, ta làm tương tự như khi biểu diễn cây tổng quát. Tuy nhiên, trong trường hợp cây nhị phân hoàn chỉnh, sử dụng cách biểu diễn này ta có thể cài đặt hiệu quả nhiều phép toán trên cây, cả phép toán đối với nút con.

Cây nhị phân

Biểu diễn cây nhị phân với mảng

Chú ý, các nút được đánh chỉ số trong mảng từ trên xuống dưới và từ trái qua phải. Với chỉ sô $i = 1, 2, \cdots, n$ với n là số nút trên cây

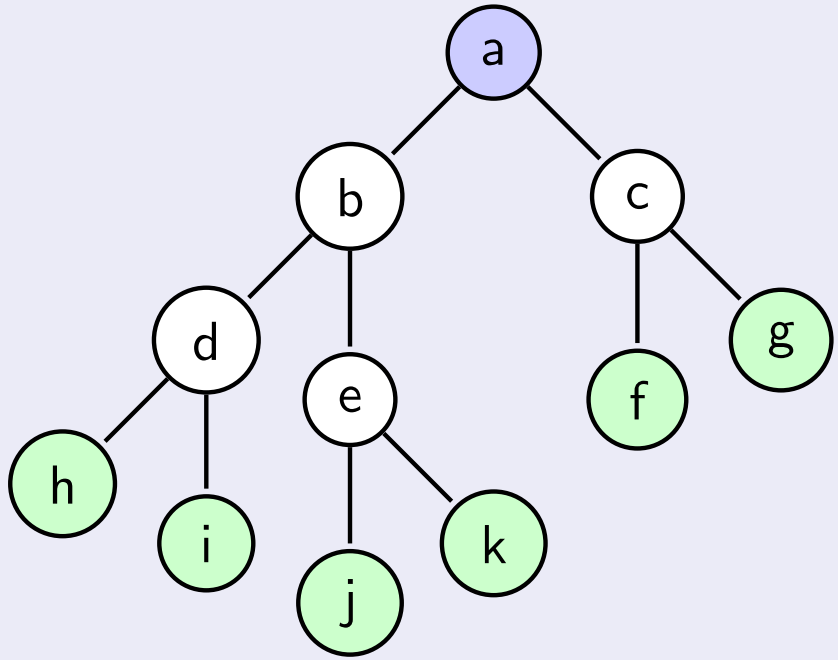


	a	a	b	b	c	c	d	d	e	e

Cây nhị phân

Biểu diễn cây nhị phân hoàn chỉnh với mảng

Chú ý, các nút được đánh chỉ số trong mảng từ trên xuống dưới và từ trái qua phải. Với chỉ sô $i = 1, 2, \cdots, n$ với n là số nút trên cây



a	b	c	d	e	f	g	h	i	j	k

Biểu diễn cây nhị phân hoàn chỉnh với mảng (tiếp)

Các phép toán trên cây nhị phân hoàn chỉnh biểu diễn bằng mảng $A[i]$ với chỉ số $i = 1, 2, \cdots, n$ với n là số nút trên cây.

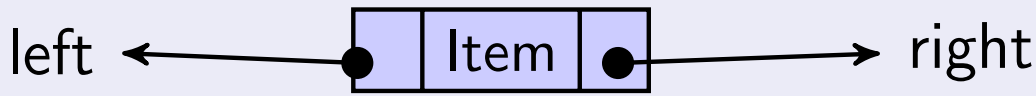
Để tìm	**Sử dụng**	**Hạn chế**
Con trái của $A[i]$	$A[2 * i]$	$2 * i \leq n$
Con phải của $A[i]$	$A[2 * i + 1]$	$2 * i + 1 \leq n$
Cha của $A[i]$	$A[i/2]$	$i > 1$
Gốc	$A[1]$	A khác rỗng
Nút $A[i]$ là lá	Đúng	$2 * i > n$

Cây nhị phân

Biểu diễn cây nhị phân bằng con trỏ

Mỗi nút trong cây nhị phân sẽ gồm ba thành phần

- con trỏ đến con trái (left)
- phần tử chứa kiểu dữ liệu (item)
- con trỏ đến con phải (right)



Cài đặt trong C

```
struct Tnode{
    DataType Item;
    struct Tnode *left;
    struct Tnode *right;
}
typedef struct Tnode treeNode;
```

Cây nhị phân

Biểu diễn cây nhị phân bằng con trỏ

Các phép toán cơ bản cây nhị phân có kiểu dữ liệu số nguyên

```
struct Tnode{
    int Item;
    struct Tnode *left;
    struct Tnode *right;
}
typedef struct Tnode treeNode;
treeNode *makeTreeNode(int x);
void freeTree(treeNode *tree);
void printPreorder(treeNode *tree);
void printPostorder(treeNode *tree);
void printInorder(treeNode *tree);
int countNode(treeNode *tree);
int depth(treeNode *tree);
```

Cây nhị phân

Biểu diễn cây nhị phân bằng con trỏ với phép toán tạo nút mới

```
treeNode *makeTreeNode(int x){
    treeNode *newNode = NULL;
    newNode = (treeNode*)malloc(sizeof(treeNode));
    if(newNode==NULL){
        printf("Het bo nho \n");
        exit(1);
    }else{
        newNode->left = NULL;
        newNode->right = NULL;
        newNode->Item = x;
    }
    return newNode;
}
```

Cây nhị phân

Biểu diễn cây nhị phân bằng con trỏ với phép toán đếm số nút

```
int countNodes(treeNode *tree){
    if(tree==NULL) return 0;
    else{
        int ld = countNodes(tree->left);
        int rd = countNodes(tree->right);
        return 1+ld+rd;
    }
}
```

Cây nhị phân

Biểu diễn cây nhị phân bằng con trỏ với phép toán tính độ sâu

```
int depth(treeNode *tree){
    if(tree==NULL) return 0;
    int ld = depth(tree->left);
    int rd = depth(tree->right);
    return 1+(ld>rd ? ld : rd);
}
```

Cây nhị phân

Biểu diễn cây nhị phân bằng con trỏ với phép toán loại bỏ cây

```
void freeTree(treeNode *tree){
    if(tree=NULL) return;
    freeTree(tree->left);
    freeTree(tree->right);
    free(tree);
    return;
}
```

Cây nhị phân

Biểu diễn cây nhị phân bằng con trỏ với phép toán duyệt cây theo thứ tự trước

Duyệt đệ qui theo thứ tự trước

- Thăm nút
- Thăm cây con trái theo thứ tự trước
- Thăm cây con phải theo thứ tự trước

Mã nguồn ngôn ngữ C

```
void printPreorder(treeNode *tree){
    if(tree!=NULL)
    {
        printf("%5d",tree->Item);
        printPreorder(tree->left);
        printPreorder(tree->right);
    }
}
```

Cây nhị phân

Biểu diễn cây nhị phân bằng con trỏ với phép toán duyệt cây theo thứ tự giữa

Duyệt đệ qui theo thứ tự giữa

- Thăm cây con trái theo thứ tự giữa
- Thăm nút
- Thăm cây con phải theo thứ tự giữa

Mã nguồn ngôn ngữ C

```
void printInorder(treeNode *tree){
    if(tree!=NULL)
    {
        printInorder(tree->left);
        printf("%5d",tree->Item);
        printInorder(tree->right);
    }
}
```

Cây nhị phân

Biểu diễn cây nhị phân bằng con trỏ với phép toán duyệt cây theo thứ tự sau

Duyệt đệ qui theo thứ tự sau

- Thăm cây con trái theo thứ tự sau
- Thăm cây con phải theo thứ tự sau
- Thăm nút

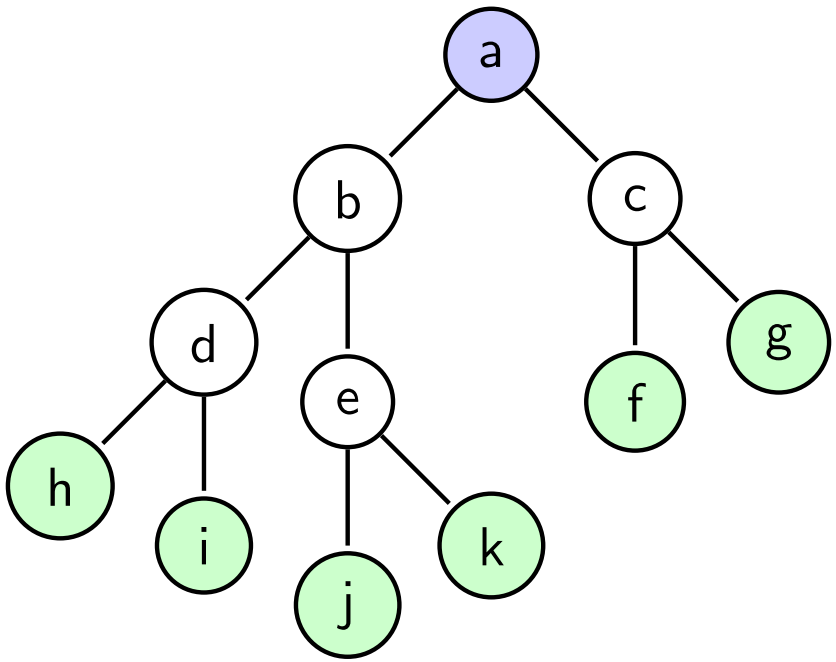
Mã nguồn ngôn ngữ C

```
void printPostrder(treeNode *tree){
    if(tree!=NULL)
    {
        printPostorder(tree->left);
        printPostorder(tree->right);
        printf("%5d",tree->Item);
    }
}
```

Cây nhị phân

Minh họa ba phép duyệt (trước, giữa, sau) của cây nhị phân dưới đây

- Duyệt trước : a,b,d,h,i,e,j,k,c,f,g
- Duyệt giữa : h,d,i,b,j,e,k,a,f,c,g
- Duyệt sau : h,i,d,j,k,e,b,f,g,c,a



Cây nhị phân

Ứng dụng 1 : cây nhị phân biểu thức - expression binary tree

Cây biểu thức nhị phân trong đó :

- mỗi nút lá chứa một toán hạng
- mỗi nút trong chứa phép toán một ngôi
- Các cây con trái và các cây con phải chứa hai vế của biểu thức phép toán hai ngôi.

Các mức thể hiện mức độ ưu tiên của phép toán :

- Mức của các nút trên cây cho biết trình tự thực hiện các phép toán (lưu ý, không sử dụng dấu ngoặc trong cây nhị phân biểu thức)
- Các phép toán mức cao được thực hiện trước
- Phép toán ở gốc được thực hiện sau cùng

Cây nhị phân

Ứng dụng 1 : cây nhị phân biểu thức - expression binary tree (tiếp)

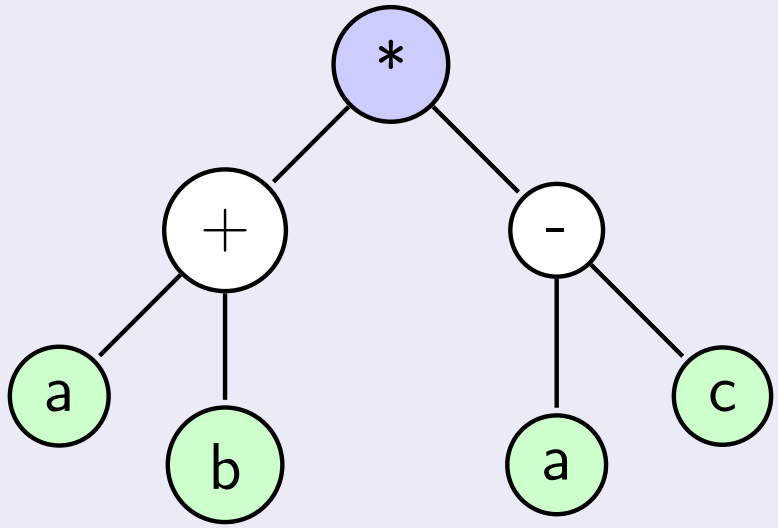
Duyệt cây nhị phân biểu thức có thể cho ta các biểu thức dưới dạng trung tố, tiền tố và hậu tố

- Duyệt cây biểu thức theo thứ tự trước (preorder) cho ta biểu thức dưới dạng tiền tố.
- Duyệt cây biểu thức theo thứ tự giữa (inorder) cho ta biểu thức dưới dạng trung tố.
- Duyệt cây biểu thức theo thứ tự sau (postorder) cho ta biểu thức dưới dạng hậu tố.

Cây nhị phân

Ứng dụng 1 : cây nhị phân biểu thức (tiếp)

Ví dụ minh họa



- Biểu thức tiền tố : * + a b - a c
- Biểu thức trung tố : (a+b)*(a-c)
- Biểu thức hậu tố : a b + a c - *

Cây quyết định - decision tree

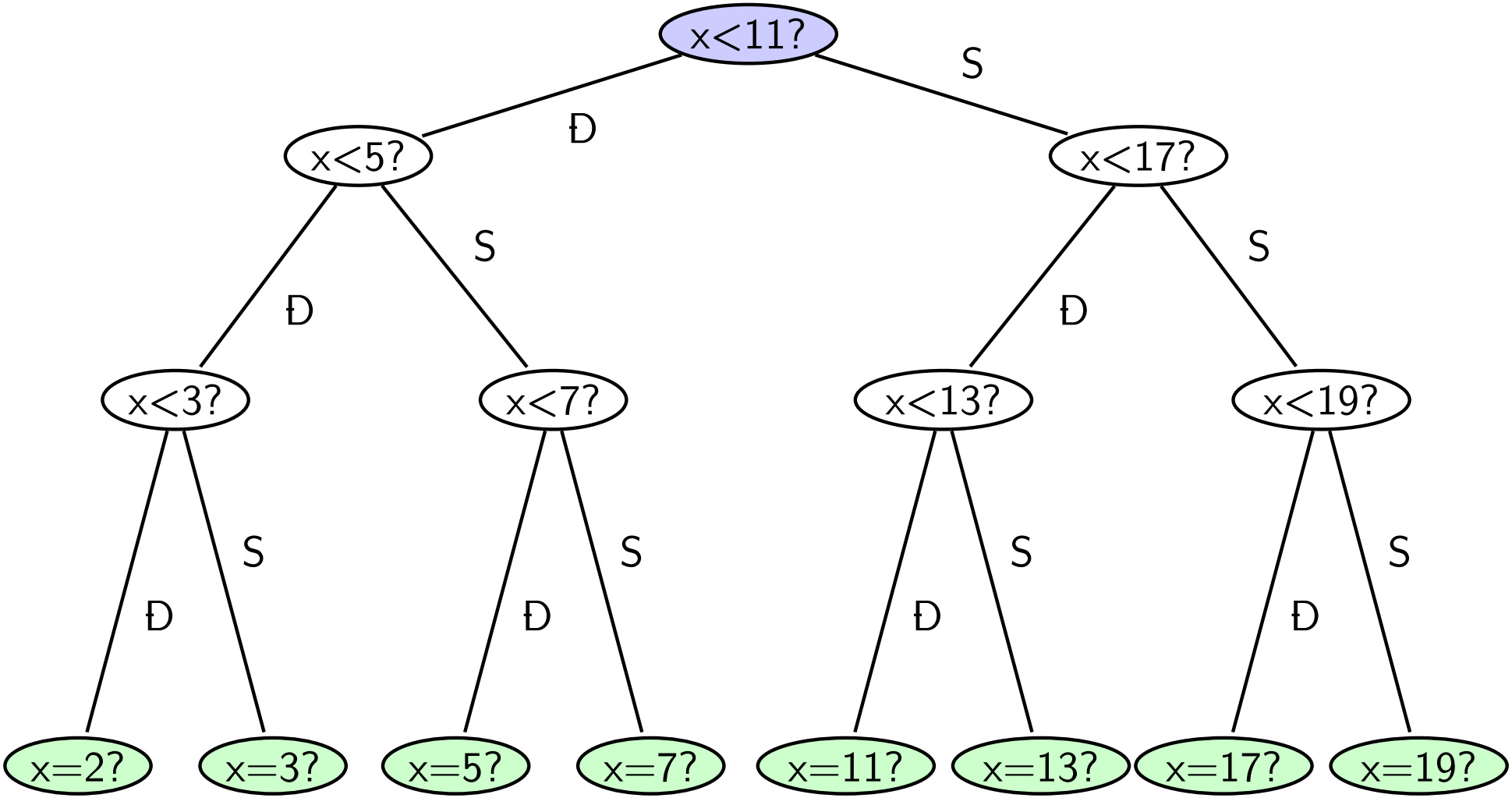
Cây quyết định là một cây mà mỗi nút trong nó là một truy vấn đối với dữ liệu. Các cạnh của cây tương ứng với khả năng trả lời của câu hỏi. Mỗi lá của nó tương ứng với một đầu ra.
Để xác định quyết định, dữ liệu được đi vào từ gốc cây - truy vấn ở gốc. Sau đó đi xuống đến lá thì biết được đầu ra - thực chất là một quyết định cuối cùng.
Các ứng dụng

- Quyết định phân loại
- Quyết định hỏi/đáp trong marketing
- Cây quyết định mờ (fuzzy decision tree) là một mô hình hiệu quả trong lĩnh vực học máy.

Cây quyết định - decision tree (tiếp)

Bài toán tra từ điển Cho một mảng gồm n số được sắp xếp theo thứ tự tăng dần và một số x. Cần xác định xem x có mặt trong mảng đã cho hay không. Nếu câu trả lời khẳng định thì cần chỉ ra vị trí của x trong mảng. Thực ra người ta gọi đây là bài toán tra từ điển, bởi cuốn từ điển được phân chia theo thứ tự alphabet từ A đến Z. Khi tra từ ta lật giở theo mục từ để tìm đến trang từ điển cần tra cứu.

Xét mảng A = (2,3,5,7,11,13,17,19) thì cây quyết định có dạng như sau



Đ : Đúng, S: Sai

Mã Huffman

Mã Huffman - Huffman code

Giả sử trong văn bản có bảng chữ cái C, với mỗi chữ cái $c \in C$ ta biết tần suất xuất hiện của nó trong văn bản là $f(c)$. Cần tìm cách mã hóa văn bản sử dụng ít bộ nhớ nhất. Có hai loại mã hóa hay dùng

- Mã hóa với độ dài cố định có đặc điểm dễ mã hóa cũng như dễ giải mã nhưng đòi hỏi bộ nhớ phải lớn
- Mã phi tiền tố (prefix free code) là cách mã hóa mỗi ký tự c bởi một xâu nhị phân code(c) sao cho mã của một ký tự bất kỳ không là đoạn đầu của bất cứ mã của ký tự nào khác trong số các ký tự còn lại. Tuy đòi hỏi việc mã hóa (code) và giải mã (decode) phức tạp nhưng lại đòi hỏi ít bộ nhớ hơn.

Mã hóa độ dài cố định có ví dụ như mã ASCII (độ dài cố định 8 bit). Ngoài ra ta cũng có mã hóa Morse, căn cứ vào thống kê tần số của các chữ cái trong từ điển tiếng Anh. Chú ý là mã Morse không phải là mã phi tiền tố. Trong khi mã Huffman được trình bầy dưới đây lại là mã phi tiền tố.

Mã Huffman (tiếp)

Mã Huffman (tiếp)

Mỗi mã phi tiền tố có thể biểu diễn bởi một cây nhị phân T mà mỗi lá của nó tương ứng với một chữ cái và cành của nó được gán cho một trong hai số 1 và 0. Mã của một chữ cái c là một dãy nhị phân gồm các số gán cho các cạnh trên đường đi từ gốc đến là tương ứng với c.

Bài toán

Tìm cách mã hóa tối ưu, tức cây nhị phân T làm tối thiểu hóa tổng độ dài có trọng số

$$B(T) = \sum_{c \in C} f(c) \times depth(c)$$

trong đó $depth(c)$ là độ dài đường đi từ gốc đến lá tương ứng với ký tự c còn $f(c)$ là tần số xuất hiện của ký tự c.

Mã Huffman (tiếp)

Mỗi mã phi tiền tố có thể biểu diễn bởi một cây nhị phân T mà mỗi lá của nó tương ứng với một chữ cái và cành của nó được gán cho một trong hai số 1 và 0. Mã của một chữ cái c là một dãy nhị phân gồm các số gán cho các cạnh trên đường đi từ gốc đến là tương ứng với c.

Bài toán

Tìm cách mã hóa tối ưu, tức cây nhị phân T làm tối thiểu hóa tổng độ dài có trọng số

$$B(T) = \sum_{c \in C} f(c) \times depth(c)$$

trong đó $depth(c)$ là độ dài đường đi từ gốc đến lá tương ứng với ký tự c còn $f(c)$ là tần số xuất hiện của ký tự c.

Lý do để tối thiểu hóa hàm trên đơn giản là làm giảm số bit cần mã hóa khi truyền thông tin bảng chữ C lên đường truyền.

Mã Huffman (tiếp)

Thuật toán

Ý tưởng thuật toán : chữ cái có tầm suất nhỏ hơn cần được gán cho lá có khoảng cách đến gốc là lớn hơn. Ngược lại, chữ cái có tần suất xuất hiện lớn hơn cần gần được gán gần nút gốc hơn.

Thuật toán : Mã hóa (code)

Procedure Huffman(C, f)
n $\leftarrow$ $|C|$; /* Gán cho n số các ký tự trong C*/
Q $\leftarrow$ C;
for i $\leftarrow$ 1 **to** n **do**
    x,y $\leftarrow$ 2 chữ cái có tần xuất nhỏ nhất trong Q;
    /*Tạo nút p là nút cha của x,y với */
    f(p) $\leftarrow$ f(x) + f(y);
    Q $\leftarrow$ Q loại bỏ {x,y};Q $\leftarrow$ Q thêm p;
endfor
End

Mã Huffman (tiếp)

Thuật toán

Ý tưởng thuật toán : chữ cái có tần suất nhỏ hơn cần được gán cho lá có khoảng cách đến gốc là lớn hơn. Ngược lại, chữ cái có tần suất xuất hiện lớn hơn cần gần được gán gần nút gốc hơn.

Thuật toán : Mã hóa (code)

```
Procedure Huffman(C, f)
n ← |C|; /* Gán cho n số các ký tự trong C*/
Q ← C;
for i ← 1 to n do
    x,y ← 2 chữ cái có tần xuất nhỏ nhất trong Q;
    /*Tạo nút p là nút cha của x,y với */
    f(p) ← f(x) + f(y);
    Q ← Q loại bỏ {x,y};Q ← Q thêm p;
endfor
End
```

Mã xây dựng theo thuật toán trên gọi là mã Huffman. Giải thuật có thởi gian cài là O(n log n) với n là số chữ cái trong C.

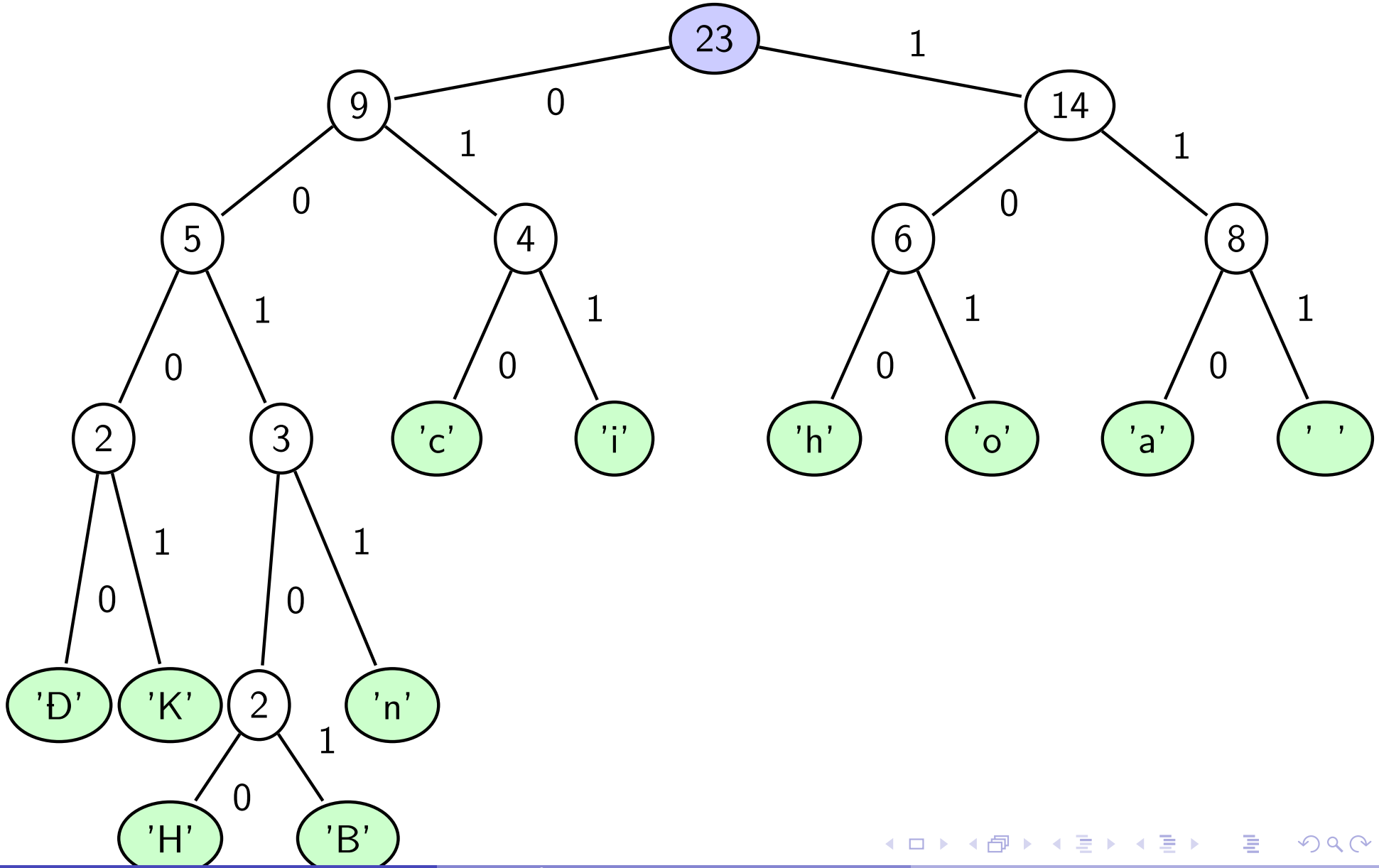
Mã Huffman (tiếp)

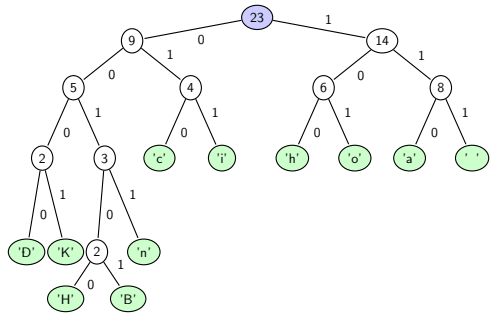
Ví dụ về mã Huffman

Cho xâu chữ không dấu : "Đai hoc Bach khoa Hanoi" có được bảng chữ có cả tần số xuất hiện giảm dần của ký tự như sau

Ký tự	'a'	' '	'h'	'o'	'c'	'i'	'Đ'	'K'	'H'	'B'	'n'
f(c)	4	4	3	3	2	2	1	1	1	1	1

Mã Huffman (tiếp)





Mã hóa Huffman cho mỗi ký tự (nút lá) chính là mã nhị phân đường đi từ gốc đến lá. Các nút trong có số chỉ chính tần số của nút p được thêm vào tập Q trong giải thuật mã hóa Huffman. Mã hóa của mỗi ký tự là dãy bit {0,1} nằm trên đường đi từ gốc đến lá, tương ứng từng ký tự.

Mã Huffman (tiếp)

Giải mã Huffman

Procedure Huffman-Decode(B)
/* Trong đó B là xâu nhị phân mã hóa văn bản theo mã hóa Huffman */
<Khởi động p là gốc của cây Huffman>
    while <chưa đạt đến kết thúc của xâu B> **do**
    x ← bit tiếp theo trong xâu B
    if (x=0) **then** p ← con trái của p **else** p ← con phải của p **endif**
    if (p là nút lá) **then**
        <hiển thị ký tự tương ứng nút lá>
        <đặt lại p làm gốc của cây Huffman>
    endif
    endwhile
End

Cây gọi đệ qui

Cây gọi đệ qui

Đây là một công cụ *quan trọng* khi phân tích giải thuật đệ qui, cây gọi đệ qui được định nghĩa như sau

- Nút gốc r của cây là lần gọi đầu của giải thuật
- Nút lá của cây tương ứng bước cơ sở
- Nhánh cây từ nút cha $f(n+1)$ đến các nút con $f(k)$ với $k \leq n$ tương ứng bước gọi đệ qui của hàm $f(n+1)$

Cây gọi đệ qui (tiếp)

Ví dụ tính dãy số Fibonnacci

Dãy số Fibonacci đc định nghĩa đệ qui như sau :

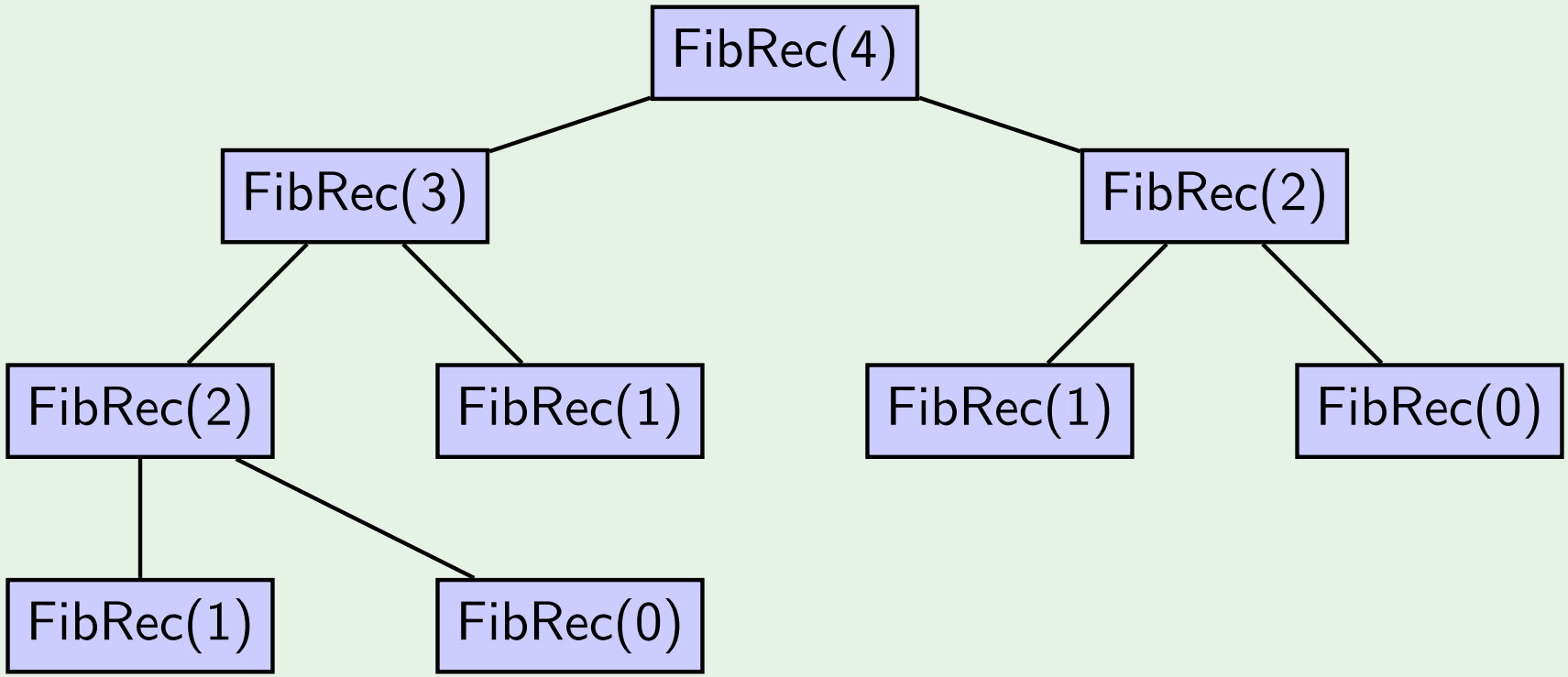
- Bước cơ sở : $F(0) = 1, F(1) = 1$;
- Bước đệ qui : $F(n) = F(n\text{-}1) + F(n\text{-}2)$ với $n \geq 2$

Hàm đệ qui viết bằng ngôn ngữ C

```
int FibRec(int n){
    if(n<=1) return 1;
    else return FibRec(n-1) + FibRec(n-2);
}
```

Cây gọi đệ qui (tiếp)

Ví dụ tính dãy số Fibonnacci (tiếp)



Cây gọi đệ qui tính số Fibonnacci với lần gọi đầu FibRec(4)

Tổng kết

- Định nghĩa đệ qui và các thuật ngữ cấu trúc dữ liệu cây
- Các phép toán trên cấu trúc dữ liệu cây
- Cách biểu diễn CTDL cây trong máy tính
- Cây nhị phân định nghĩa và tính chất
- Các ứng dụng của cây nhị phân