

# Chương 5 - Cây

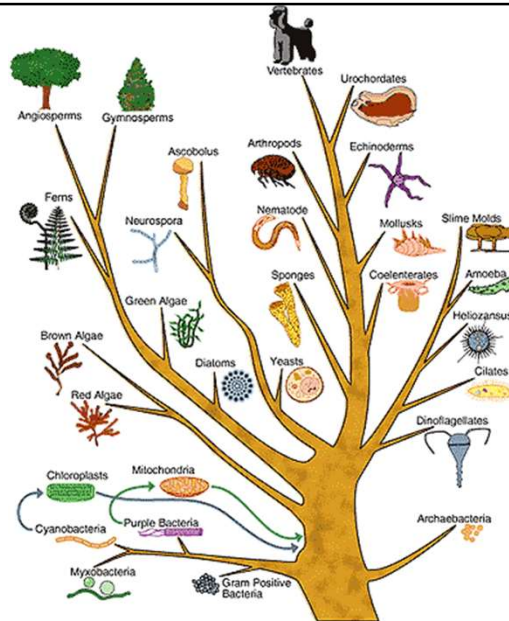
Cây tổng quát, biểu diễn cây, duyệt cây, prefix tree,  
cây nhị phân, suffix tree

## Nội dung

- Khái niệm cơ bản
- Biểu diễn cây tổng quát
- Một số bài toán cơ bản trên cây tổng quát
- Cây nhị phân
- Ứng dụng của cây nhị phân
- Một số bài toán trên cây nhị phân
- Một số mở rộng của cây tổng quát

## Khái niệm

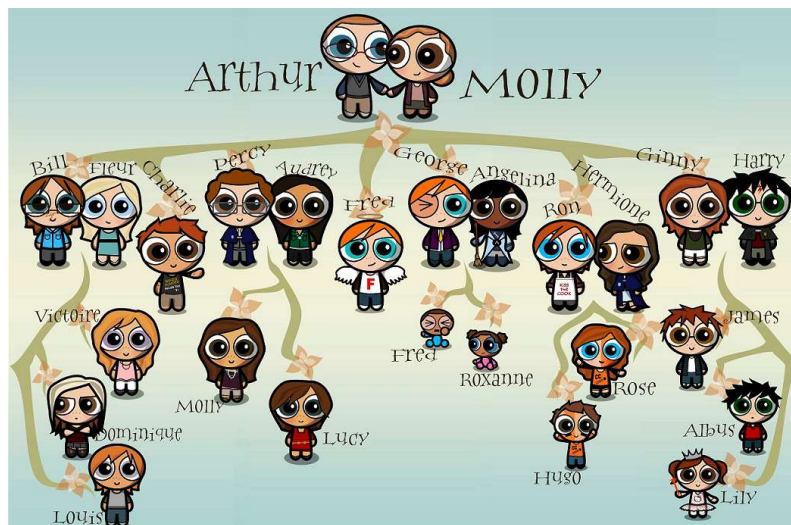
- Cây - tree
- Là kiểu cấu trúc dữ liệu phi tuyến
- Biểu diễn mối *quan hệ có thứ bậc* giữa các đối tượng
- Quan hệ giữa các nút là quan hệ cha-con (tổ tiên- con cháu) hoặc ngang hàng



11/18/2021

hiepd@soict.hust.edu.vn

3

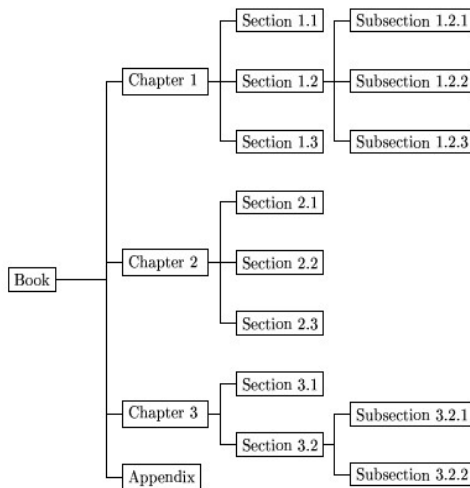


- Cây phả hệ - family tree

11/18/2021

hiepd@soict.hust.edu.vn

4



Mục lục sách



Cây thư mục

11/18/2021

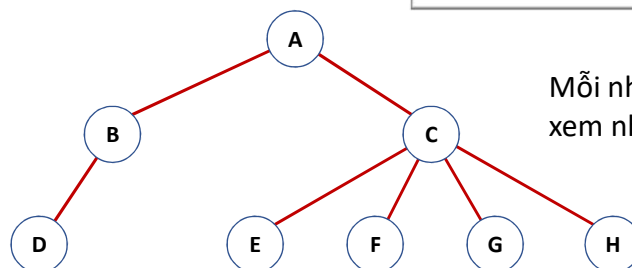
hiepn@soict.hust.edu.vn

5

## Khái niệm cơ bản

- Các nút ở mức ngoài cùng được gọi là *nút lá* – *leaf node*
- Các nút không phải nút lá được gọi là *nút trong* – *internal node*
- *Nút gốc* – là nút mà không có nút nào đứng trên nó

### Example of a Simple Directory Structure



Mỗi nhánh có thể được xem như là một cây con

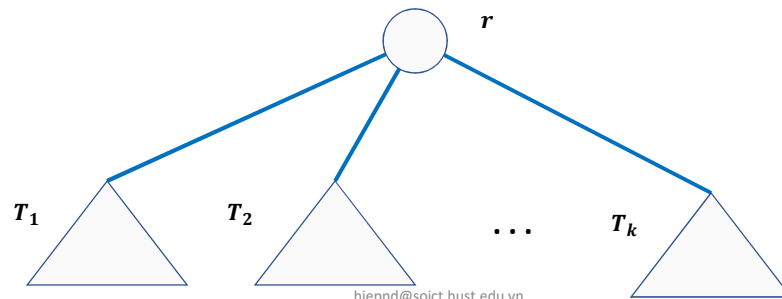
11/18/2021

hiepn@soict.hust.edu.vn

6

## Khái niệm cơ bản

- Định nghĩa: Cây là một tập hợp gồm các nút.
  - Tập này có thể rỗng,
  - nếu không thì nó bao gồm một nút  $r$  được gọi là gốc và một tập rỗng hoặc khác rỗng các cây con  $T_1, T_2, \dots, T_k$  mà có nút gốc được nối trực tiếp bằng một cạnh từ nút gốc  $r$



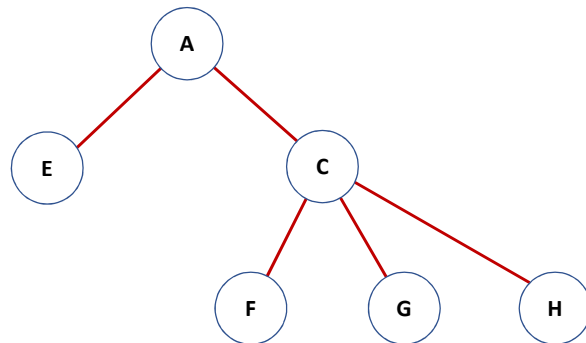
11/18/2021

hiepdnd@soict.hust.edu.vn

7

## Khái niệm cơ bản

- Các nút cùng cha là anh em (sibling)
  - E và C, hoặc F, G và H
- Nút gốc: là nút không có nút nào đứng trên
  - A là gốc
- Nút lá: nút không có nút con
  - E, F, G, H
- Nút trong: Nút có ít nhất 1 con
  - A, C



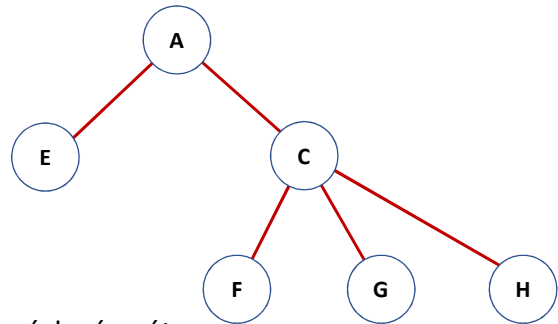
Cây tổng quát: 1 nút có thể không có con, hoặc có thể có vô số con

11/18/2021

hiepdnd@soict.hust.edu.vn

8

## Khái niệm cơ bản



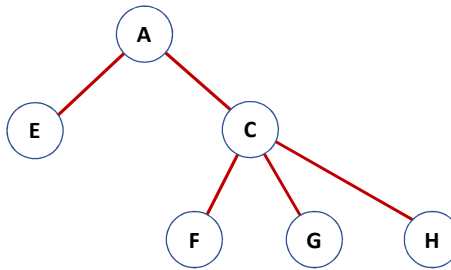
- Đường đi từ nút  $n_i$  tới nút  $n_j$  trên cây là một danh sách các nút  $n_i, n_{i+1}, \dots, n_j$  trong đó nút  $n_i$  là cha của nút  $n_{i+1}$ .
- Độ dài đường đi: là số cạnh trên đường đi.
  - Đường đi qua  $n$  đỉnh thì có độ dài  $n - 1$
  - Đường đi từ nút đến chính nó có độ dài là 0
- Đường đi từ A đến H là A, C, H có độ dài 2
- Đường đi từ C đến C có độ dài 0

11/18/2021

hiepn@soict.hust.edu.vn

9

## Khái niệm cơ bản



- Độ sâu/mức (depth/level) của nút: độ dài đường đi từ nút gốc đến chính nó
  - Độ sâu của nút gốc A là 0, của nút G, H là 2
- Độ cao (height) của nút: là độ dài đường đi lớn nhất từ nó đến một nút lá
  - Độ cao của nút lá E, F, G, H là 0, của nút A là 2
- Độ cao của cây: là độ cao của nút gốc
- Độ sâu của cây: là độ sâu lớn nhất của nút lá trên cây

11/18/2021

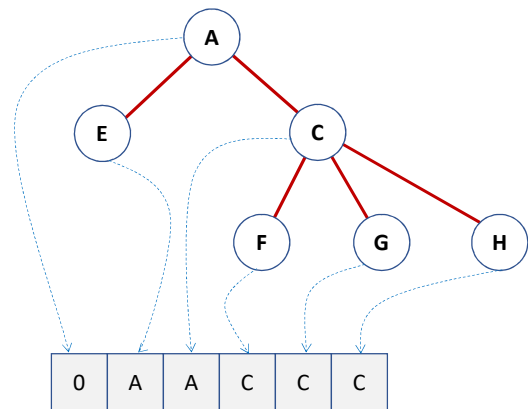
hiepn@soict.hust.edu.vn

10

## Biểu diễn cây

- Biểu diễn cây: thông qua nhãn nút cha của nó (cấu trúc liên tiếp)

- Lưu dần các nút theo mức, mức 0 sẽ lưu đầu tiên
- Tìm nút cha của nút hiện tại nhanh
- Thêm/xóa nút phức tạp, thường phải dịch phần tử



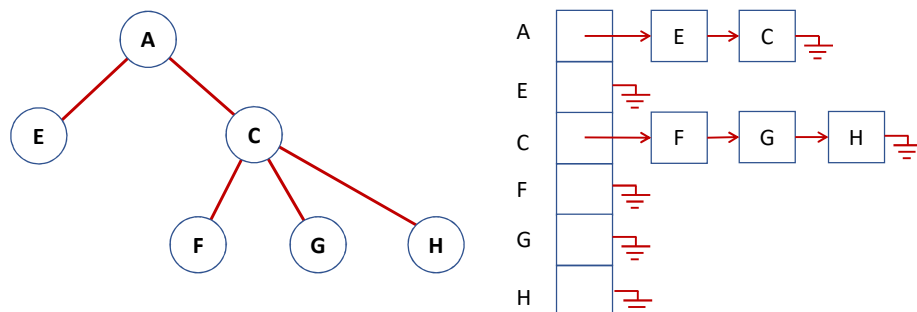
11/18/2021

hiepnd@soict.hust.edu.vn

11

## Biểu diễn cây

- Biểu diễn cây: danh sách các nút con (mảng của các danh sách liên kết)



- Danh sách các nút con của nút hiện tại biểu diễn bằng danh sách liên kết đơn
- Thông thường, con trái nhất sẽ đứng đầu danh sách
- Nút không có con thì danh sách tương ứng là rỗng

11/18/2021

hiepnd@soict.hust.edu.vn

12

## Biểu diễn cây

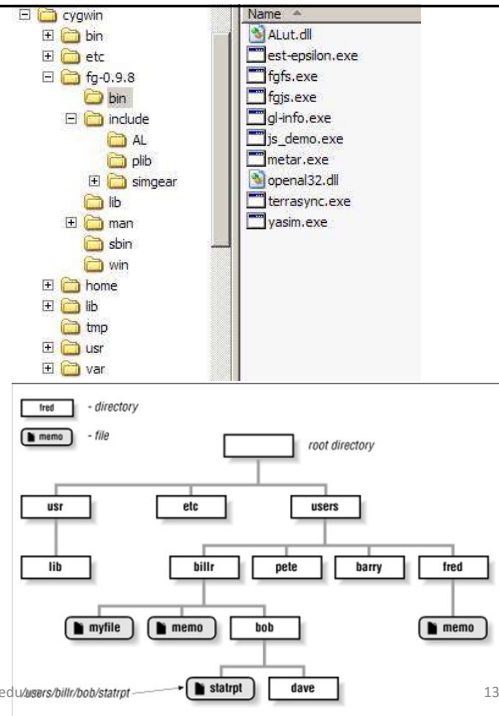
- Biểu diễn cây: Con đầu tiên và các nút anh chị em kế tiếp (cấu trúc liên kết)

```
struct TreeNode
{
    DATA_TYPE info;
    struct TreeNode *firstChild;
    struct TreeNode *nextSibling;
}
```

Trong thực tế bổ sung thêm con trỏ tới nút cha để thêm và xóa nút nhanh hơn

11/18/2021

hiepdnd@soict.hust.edu.vn



13

## Duyệt cây

### • Thuật toán duyệt cây:

- Phân biệt bởi việc thăm nút con trước hay là xử lý dữ liệu tại nút hiện tại trước
- Các nút con thông thường sẽ được thăm theo thứ tự con trái nhất trước sau đó đến các nút con kế tiếp
- Cây tổng quát có các phương án duyệt cây thông dụng
  - Duyệt theo thứ tự trước – gọi đệ quy hoặc dùng stack
  - Duyệt theo thứ tự sau – gọi đệ quy hoặc dùng stack
  - Duyệt theo mức – dùng hàng đợi

11/18/2021

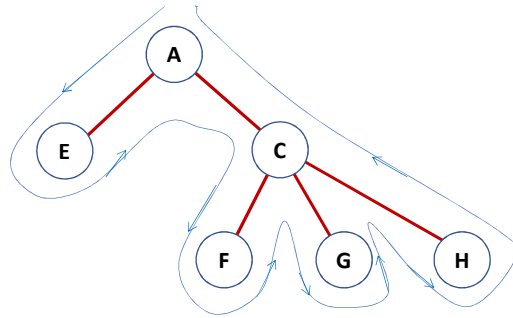
hiepdnd@soict.hust.edu.vn

14

## Duyệt cây

- **Duyệt cây theo thứ tự trước** (pre-order traversal): xử lý dữ liệu trên nút hiện tại trước, sau đó xử lý tiếp đến các nút con của nó

```
preOrder(root){
  if(root == NULL) return;
  processs(root);
  for each p = r1, r2, ..., rk {
    preOrder(p);
  }
}
```



In ra các đỉnh khi duyệt theo thứ tự trước: A, E, C, F, G, H

11/18/2021

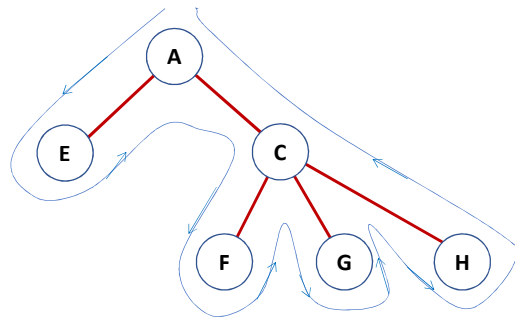
hiepdn@soict.hust.edu.vn

15

## Khái niệm

- **Duyệt cây theo thứ tự sau** (post-order traversal): xử lý dữ liệu trên các nút con trước sau đó mới xử lý dữ liệu trên nút hiện tại

```
postOrder(root){
  if(root == NULL) return;
  for each p = r1, r2, ..., rk {
    postOrder(p);
  }
  processs(root);
}
```



In ra các đỉnh khi duyệt theo thứ tự sau: E, F, G, H, C, A

11/18/2021

hiepdn@soict.hust.edu.vn

16



## Thao tác với cây tổng quát

- `find(r, label)`: tìm và trả về nút có nhãn label trên cây có gốc là r
- `insert(r, label)`: tạo một nút có nhãn label, chèn vào cuối danh sách nút con của nút r
- `height(r)`: trả về độ cao của nút r
- `depth(r)`: trả về độ sâu của nút r
- `parent(r)`: trả về nút cha của nút r
- `count(r)`: trả về số nút trên cây có gốc là r
- `countLeaves(r)`: trả về số nút lá trên cây có gốc là r

```
#include <stdio.h>
#include <stdlib.h>
#include <queue>

using namespace std;
typedef struct TREENODE
{
    char label;
    struct TREENODE* firstChild;
    struct TREENODE* nextSibling;
} TreeNode;
```

11/18/2021

hiepnd@soict.hust.edu.vn

17

```
TreeNode* makeNode(char label)
{
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->label = label;
    newNode->firstChild = NULL;
    newNode->nextSibling = NULL;
    return newNode;
}
```

```
// chen nut da duoc tao newNode vao nut con moi nhat (phai nhat) cua nut root
TreeNode* insertNode(TreeNode* root, TreeNode* newNode)
{
    if (root == NULL) root = newNode;
    if (root->firstChild == NULL) root->firstChild = newNode;
    else
    {
        TreeNode* rightmost = root->firstChild;
        while (rightmost->nextSibling != NULL) rightmost = rightmost->nextSibling;
        rightmost->nextSibling = newNode;
    }
    return root;
}
```

11/18/2021

hiepnd@soict.hust.edu.vn

18

```
// chen nut da duoc tao newNode vao nut con moi nhat (phai nhat)
// cua nut root, ham khong can tra ve gia tri
void insertNode2(TreeNode** root, TreeNode* newNode)
{
    if (*root == NULL) *root = newNode;
    if ((*root)->firstChild == NULL) (*root)->firstChild = newNode;
    else {
        TreeNode* rightmost = (*root)->firstChild;
        while (rightmost->nextSibling != NULL) rightmost = rightmost->nextSibling;
        rightmost->nextSibling = newNode;
    }
}
```

```
// tao va chen nut moi thanh cay con phai nhat cua nut root
// Chu ý: root phai khac NULL
void insertNode_v2(TreeNode* root, char label)
{
    if (root == NULL) return;
    TreeNode* newNode = makeNode(label);
    if (root->firstChild == NULL) root->firstChild = newNode;
    else {
        TreeNode* rightmost = root->firstChild;
        while (rightmost->nextSibling != NULL) rightmost = rightmost->nextSibling;
        rightmost->nextSibling = newNode;
    }
}
```

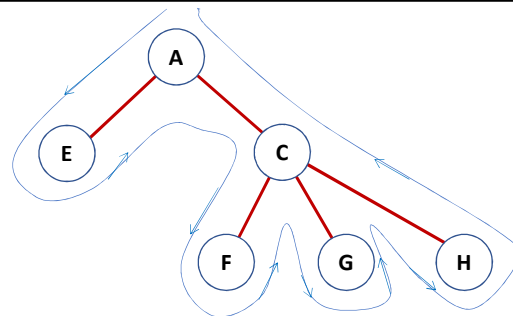
11/18/2021

hiepdnd@soict.hust.edu.vn

19

```
// duyet cay theo muc
void levelTraversal(TreeNode* root)
{
    if (NULL == root) return;
    queue<TreeNode*> Q;
    Q.push(root);

    while (Q.size() > 0)
    {
        TreeNode* p = Q.front();
        Q.pop();
        printf("%c, ", p->label);
        TreeNode* q = p->firstChild;
        while (NULL != q)
        {
            Q.push(q);
            q = q->nextSibling;
        }
        printf("\n");
    }
}
```



```
int main()
{
    TreeNode* root = makeNode('A');
    insertNode_v2(root, 'E');
    insertNode_v2(root, 'C');
    insertNode_v2(root->firstChild->nextSibling, 'F');
    insertNode_v2(root->firstChild->nextSibling, 'G');
    insertNode_v2(root->firstChild->nextSibling, 'H');
    levelTraversal(root);
    return 0;
}
```

11/18/2021

hiepdnd@soict.hust.edu.vn

20

```
// duyệt cây theo thứ tự trước
void preOrderTravesal(TreeNode* root)
{
    if (NULL == root) return;
    printf("%c, ", root->label);
    TreeNode* p = root->firstChild;
    while (NULL != p) {
        preOrderTravesal(p);
        p = p->nextSibling;
    }
}
```

```
// duyệt cây theo thứ tự sau
void postOrderTravesal(TreeNode* root)
{
    if (NULL == root) return;
    TreeNode* p = root->firstChild;
    while (NULL != p) {
        postOrderTravesal(p);
        p = p->nextSibling;
    }
    printf("%c, ", root->label);
}
```

```
// đếm số lượng nút trên cây
int countNodes(TreeNode* root)
{
    if (NULL == root) return 0;
    TreeNode* p = root->firstChild;
    int sum = 0;
    while (NULL != p) {
        sum += countNodes(p);
        p = p->nextSibling;
    }
    return sum + 1;
}
```

```
int countLeaves(TreeNode* root) {
    if (root == NULL) return 0;
    int sum = 0;
    TreeNode* p = root->firstChild;
    if (p == NULL) sum = 1;
    while (p != NULL) {
        sum += countLeaves(p);
        p = p->nextSibling;
    }
    return sum;
}
```

11/18/2021

hiepdnd@soict.hust.edu.vn

21

```
// tìm nút cha của nút p trên cây có gốc là root
TreeNode* parent(TreeNode* root, TreeNode* p) {
    if (root == NULL) return NULL;
    TreeNode* q = root->firstChild;
    while (q != NULL) {
        if (p == q) return root;
        TreeNode* pp = parent(q, p);
        if (pp != NULL) return pp;
        q = q->nextSibling;
    }
    return NULL;
}
```

```
// tính chiều cao của nút (chiều cao là 1)
int height(TreeNode* root) {
    if (root == NULL) return 0;
    int maxh = 0;
    TreeNode* q = root->firstChild;
    while (q != NULL) {
        int h = height(q);
        if (h > maxh) maxh = h;
        q = q->nextSibling;
    }
    return maxh + 1;
}
```

11/18/2021

hiepdnd@soict.hust.edu.vn

22

```
// tinh chieu cao cua nut
// coi nut la chieu cao la 0
int height(TreeNode* root) {
    if (root == NULL) return -1; // cay rong
    TreeNode* q = root->firstChild;
    if (NULL == q) return 0; // nut hien tai la la
    int maxh = 0;
    while (q != NULL) {
        int h = height(q);
        if (h > maxh) maxh = h;
        q = q->nextSibling;
    }
    return maxh + 1;
}
```

11/18/2021

hiepdnd@soict.hust.edu.vn

23

```
// Tim do sau nut co nhan label tren cay goc la root
// dung de quy
int depth(TreeNode* root, char label, int d) {
    // d la do sau cua nut root
    if (root == NULL) return -1;
    if (root->label == label) return d;
    TreeNode* p = root->firstChild;
    while (p != NULL) {
        if (p->label == label) return d + 1;
        int dv = depth(p, label, d + 1);
        if (dv > 0) return dv;
        p = p->nextSibling;
    }
    return -1;
}

int calDepth(TreeNode* root, char v) {
    return depth(root, v, 0);
}
```

```
// Tim do sau nut p tren cay goc la root dung de quy
// d la do sau cua nut root

int depth(TreeNode* root, TreeNode* p, int d) {
    if (root == NULL) return -1;
    if (root == p) return d;
    TreeNode* q = root->firstChild;
    while (q != NULL) {
        if (q == p) return d + 1;
        int dv = depth(q, p, d + 1);
        if (dv > 0) return dv;
        q = q->nextSibling;
    }
    return -1;
}

int calDepth(TreeNode* root, TreeNode* v) {
    return depth(root, v, 0);
}
```

11/18/2021

hiepdnd@soict.hust.edu.vn

24

```
// tính độ sâu của nút có nhãn label, không đệ quy
int depth(TreeNode* root, char label) {
    if (root == NULL) return -1;
    queue<pair<TreeNode*,int>> Q;
    Q.push(make_pair(root,0));

    int currentDepth;
    TreeNode* p;
    while (Q.size()>0) {
        p = Q.front().first;
        currentDepth = Q.front().second;
        Q.pop();
        if (p->label == label) return currentDepth;
        p = p->firstChild;
        while (p != NULL) {
            Q.push(make_pair(p, currentDepth+1));
            p = p->nextSibling;
        }
    }
    return -1;
}
```

11/18/2021

hiepnd@soict.hust.edu.vn

25



## Cây nhị phân

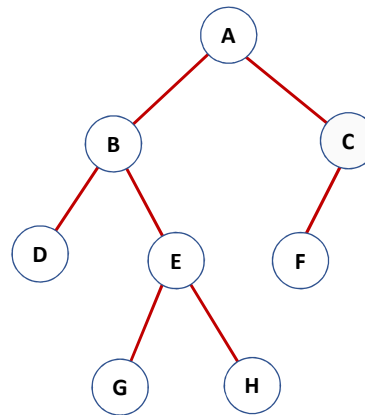
11/18/2021

hiepnd@soict.hust.edu.vn

26

## Cây nhị phân

- Cây nhị phân – binary tree:
  - Là một tập rỗng, hoặc
  - Có một nút gốc với hai cây nhị phân con của gốc gọi là cây con trái (left subtree) và cây con phải (right subtree)



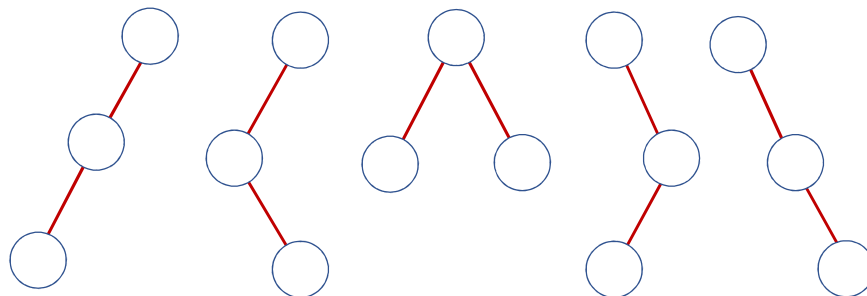
11/18/2021

hiepnd@soict.hust.edu.vn

27

## Cây nhị phân

- Một số cây nhị phân gồm 3 nút khác nhau



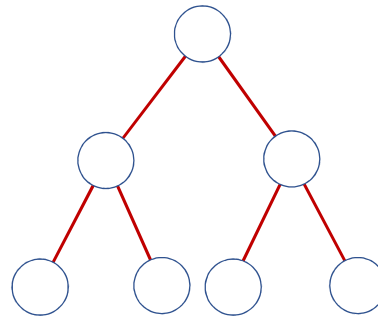
11/18/2021

hiepnd@soict.hust.edu.vn

28

## Cây nhị phân

- Cây nhị phân đầy đủ (full binary tree):
  - Các nút không phải là lá đều có 2 nút con
  - Các nút lá có độ sâu bằng nhau
- Cây nhị phân đầy đủ chiều cao  $h$  có bao nhiêu nút lá?
- Cây nhị phân đầy đủ có bao nhiêu nút?



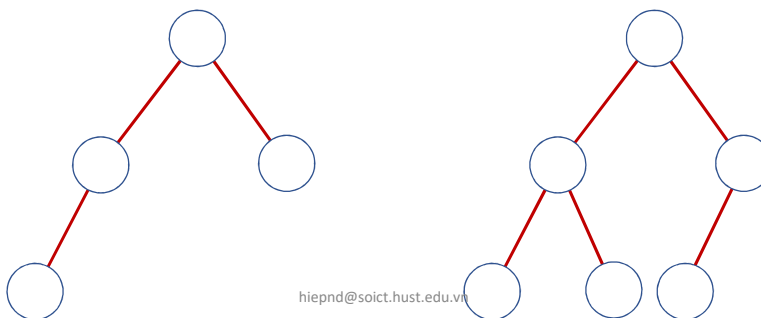
11/18/2021

hiepnd@soict.hust.edu.vn

29

## Cây nhị phân

- Cây nhị phân hoàn chỉnh (complete binary tree) chiều cao  $h$ 
  - Các nút từ mức 0 tới  $h - 1$  là cây nhị phân đầy đủ
  - Một số nút ở mức  $h - 1$  có thể có 0 hoặc 1 con
  - Nếu  $i, j$  là các nút ở mức  $h - 1$ , thì  $i$  có nhiều con hơn  $j$  nếu  $i$  nằm ở bên trái  $j$



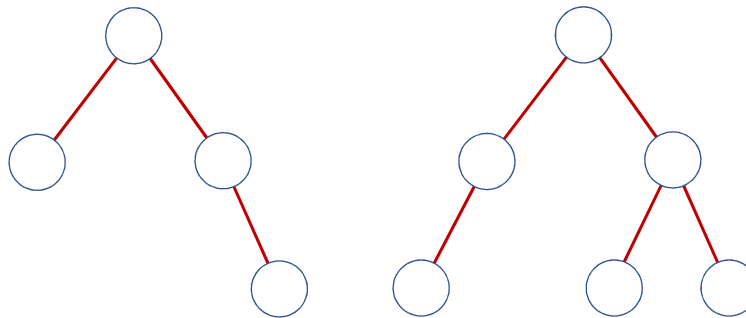
11/18/2021

hiepnd@soict.hust.edu.vn

30

## Cây nhị phân

- Cây nhị phân cân bằng (balanced binary tree): là cây nhị phân thỏa mãn
  - Với mọi nút thì chênh lệch chiều cao của cây con trái và cây con phải không quá 1
  - Mỗi cây nhị phân hoàn chỉnh là cây cân bằng



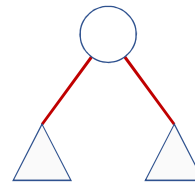
11/18/2021

hiepnd@soict.hust.edu.vn

31

## Duyệt cây – tree traversal

- Mỗi nút trên cây nhị phân gồm :
  - Giá trị chứa tại nút
  - Cây con trái
  - Cây con phải
- Duyệt cây theo thứ tự
  - Thứ tự trước – preorder: Giá trị → con trái → con phải
  - Thứ tự giữa – inorder : con trái → giá trị → con phải
  - Thứ tự sau – postorder : con trái → con phải → giá trị



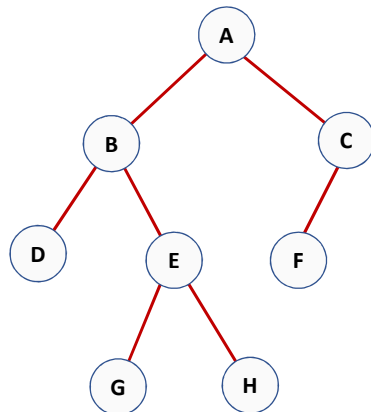
11/18/2021

hiepnd@soict.hust.edu.vn

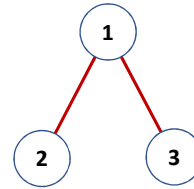
32



## Duyệt cây – tree traversal



- Thứ tự trước : 1, 2, 3
- Thứ tự giữa : 2, 1, 3
- Thứ tự sau : 2, 3, 1



Thứ tự trước : A, B, D, E, G, H, C, F

Thứ tự giữa : D, B, G, E, H, A, F, C

Thứ tự sau : D, G, H, E, B, F, C, A

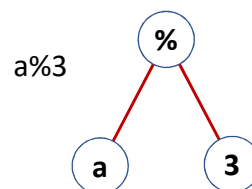
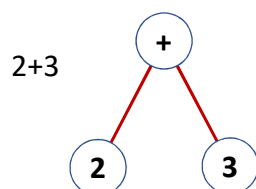
11/18/2021

hiepnd@soict.hust.edu.vn

33

## Cây biểu thức – expression tree

- *Cây biểu thức – expression tree*: xây dựng từ các toán tử và toán hạng
  - Toán tử 2 ngôi: gốc là toán tử, 2 nút con lần lượt là toán hạng trái và phải



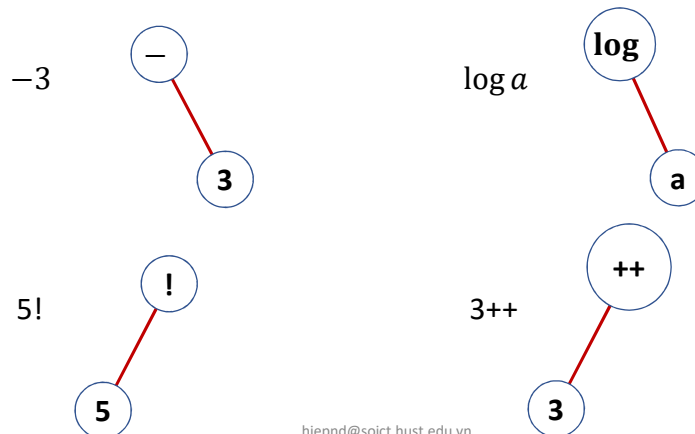
11/18/2021

hiepnd@soict.hust.edu.vn

34

## Cây biểu thức – expression tree

- Toán tử 1 ngôi: nút gốc biểu diễn toán tử, chỉ có 1 nút con biểu diễn toán hạng

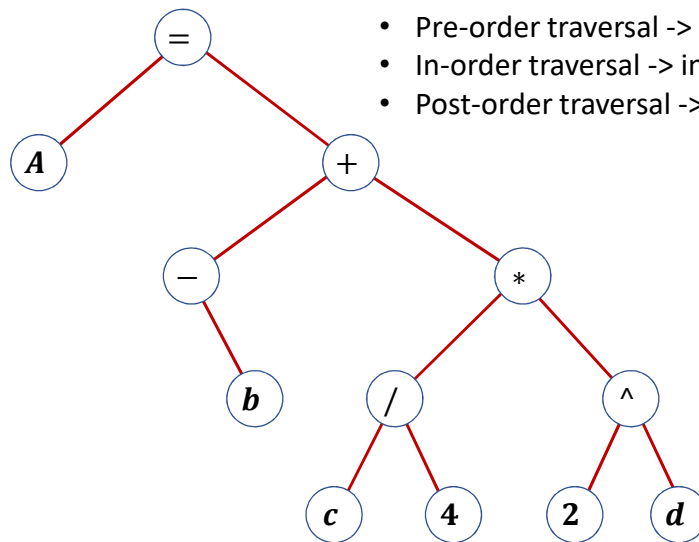


11/18/2021

hiepnd@soict.hust.edu.vn

35

$$A = -b + c/4 * 2^d$$



- Pre-order traversal  $\rightarrow$  prefix notation
- In-order traversal  $\rightarrow$  infix notation
- Post-order traversal  $\rightarrow$  postfix notation

11/18/2021

hiepnd@soict.hust.edu.vn

36

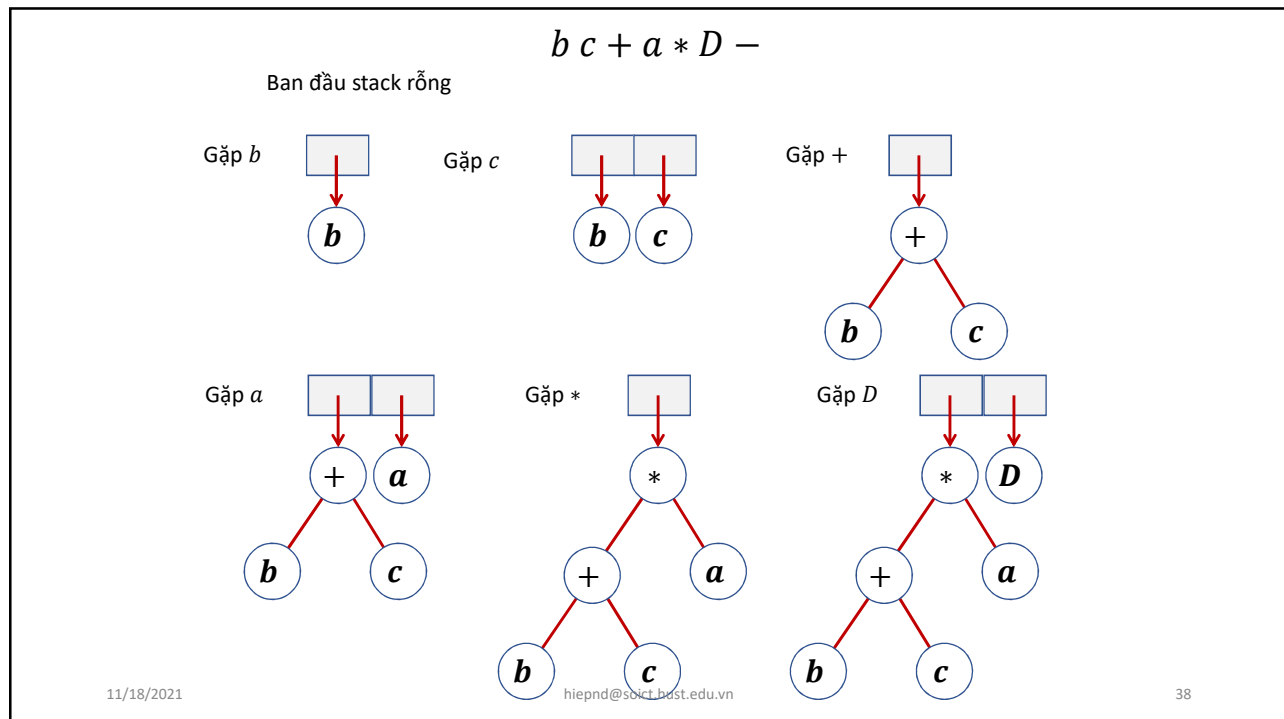
## Cây biểu thức – expression tree

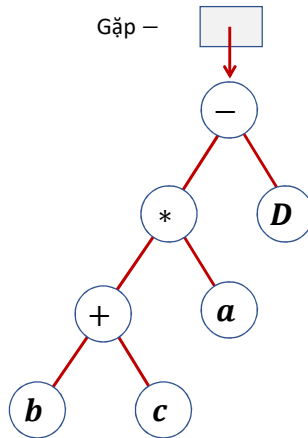
- Thuật toán xây dựng cây biểu thức từ biểu thức dạng hậu tố
- Đọc lần lượt các phần tử của biểu thức hậu tố
- Gặp toán hạng: tạo ra cây có 1 nút là toán hạng và đẩy vào stack
- Gặp toán tử:
  - Nếu là toán tử 1 ngôi: lấy ra 1 cây  $T$  trong stack, tạo ra 1 cây trong đó toán tử là nút gốc và cây  $T$  là nút con, sau đó đẩy cây này vào stack
  - Nếu là toán tử 2 ngôi: lấy ra 2 cây trong stack  $T_1, T_2$  ( $T_2$  được lấy ra trước), tạo cây trong đó toán tử này là gốc và  $T_1, T_2$  lần lượt là con trái và con phải của nó, sau đó đẩy cây này vào stack
- Sau khi duyệt xong biểu thức hậu tố, cây biểu thức là cây có gốc nằm ở đỉnh của stack

11/18/2021

hiepnd@soict.hust.edu.vn

37



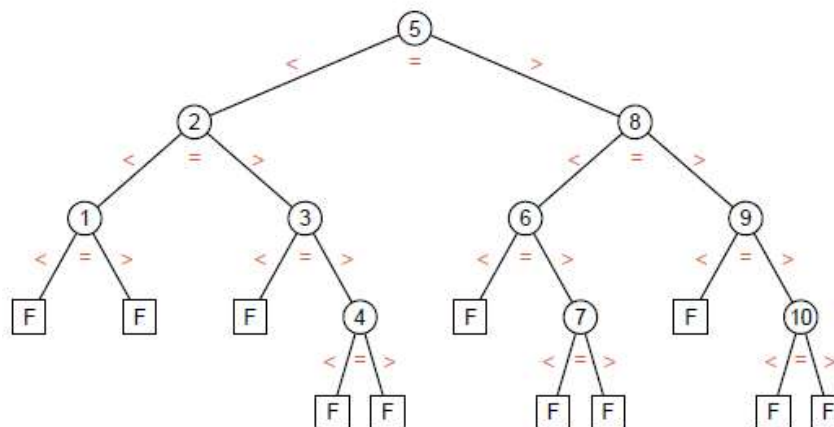
$$b\ c + a * D -$$


11/18/2021

hiepdnd@soict.hust.edu.vn

39

Cây quyết định, cây tìm kiếm nhị phân



**Cây quyết định cho thuật toán tìm kiếm nhị phân trên dãy số  
1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

11/18/2021

hiepdnd@soict.hust.edu.vn

40

## Biểu diễn cây nhị phân

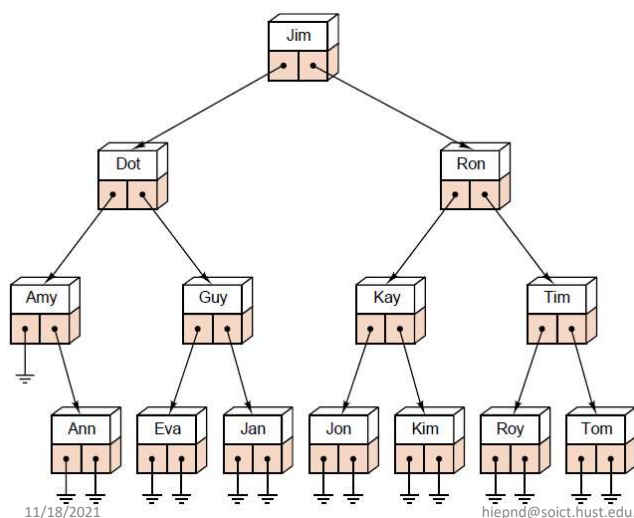
- Biểu diễn cây nhị phân
- Dùng cấu trúc liên tiếp (mảng): số lượng nút trên cây chiều cao  $h$  là giới hạn ( $\sum 2^i$ )
  - Truy nhập nhanh  $O(1)$
  - Lãng phí bộ nhớ lớn nếu cây chưa phải là dạng đầy đủ hoặc hoàn chỉnh
- Dùng cấu trúc liên kết: thường dùng hơn
  - Truy cập 1 nút chậm hơn
  - Tiết kiệm bộ nhớ hơn (trong trường hợp tổng quát)

11/18/2021

hiepdnd@soict.hust.edu.vn

41

## Biểu diễn cây nhị phân



11/18/2021

hiepdnd@soict.hust.edu.vn

42

```

struct TreeNode
{
    DATA_TYPE info;
    struct TreeNode *leftChild;
    struct TreeNode *rightChild;
}
  
```

- Biểu diễn bằng cấu trúc liên kết

## Biểu diễn cây nhị phân

- Duyệt cây theo thứ tự trước/giữa/sau

```
void preorderTraversal(TreeNode* root)
{
    if(root==NULL) return;
    printf("%d ",root->data);
    preorderTraversal(root->leftChild);
    preorderTraversal(root->rightChild);
}
```

```
void postorderTraversal(TreeNode* root)
{
    if(root==NULL) return;
    postorderTraversal(root->leftChild);
    postorderTraversal(root->rightChild);
    printf("%d ",root->data);
}
```

11/18/2021

hiepdnd@soict.hust.edu.vn

43

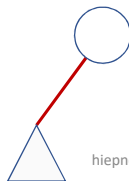
```
struct TreeNode
{
    int data;
    struct TreeNode *leftChild;
    struct TreeNode *rightChild;
}
```

```
void inorderTraversal(TreeNode* root)
{
    if(root==NULL) return;
    inorderTraversal(root->leftChild);
    printf("%d ",root->data);
    inorderTraversal(root->rightChild);
}
```

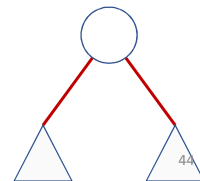
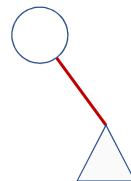
## Cài đặt thuật toán trên cây nhị phân

- Cây nhị phân có 5 trường hợp khi xử lý đệ quy
  - Cây rỗng
  - Nút gốc là nút lá
  - Nút gốc chỉ có con trái
  - Nút gốc chỉ có con phải
  - Nút gốc có đủ 2 con
- Tùy các bài toán mà có thể gộp các trường hợp lại với nhau

NULL



hiepdnd@soict.hust.edu.vn



11/18/2021

## Cây nhị phân

- Bài 1. Tìm nút có giá trị lớn nhất/ nhỏ nhất trên cây nhị phân
- Bài 2. Thêm/xóa nút trên cây nhị phân
- Bài 3. Kiểm tra 2 cây nhị phân đồng dạng
- Bài 4. Kiểm tra 2 cây nhị phân đối xứng
- Bài 5. Thêm nút mới vào cây theo mức lần lượt từ trái sang phải (tương tự trên cây tổng quát)
- Bài 6. In ra đường đi dài nhất trên cây từ gốc tới lá
- Bài 7. In ra nút lá nông nhất/sâu nhất trên cây
- Bài 8. Cho cây nhị phân mà giá trị tại nút là 0 hoặc 1 (gốc luôn là 1), in ra đường đi chứa toàn số 1 dài nhất từ gốc
- Bài 9. Định giá cây biểu thức

11/18/2021

hiepnd@soict.hust.edu.vn

45

## Cây nhị phân

- Bài 10. Cho cây nhị phân trong đó mỗi nút trong có 2 con và nút lá, cho thứ tự duyệt cây trước là B C D E A F G và thứ tự giữa C B A E F D G, hãy đưa ra thứ tự duyệt sau
- Bài 11. Tìm nút tổ tiên gần nhất của 2 nút trên cây
- Bài 12. kiểm tra xem các nút lá trên cây có phải cùng mức
- Bài 13. Cho 2 cây T1 và T2, kiểm tra xem T2 có phải là cây con của T1
- Bài 14. Cho cây nhị phân với nút là số nguyên, in ra các đường đi từ gốc có tổng giá trị  $\leq k$
- Bài 15. Loại các nút bị trùng lặp trên cây nhị phân

11/18/2021

hiepnd@soict.hust.edu.vn

46

```
#include <stdio.h>
#include <stdlib.h>
typedef struct TREENode
{
    char label;
    struct TREENode* left;
    struct TREENode* right;
} TREENode;

TREENode* makeNode(char label)
{
    TREENode* newNode =
        (TREENode*)malloc(sizeof(TREENode));
    newNode->label = label;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
```

```
// tao va chen nut moi thanh cay con phai nhat cua nut root
// Chu y: root phai khac NULL
// leftorRight = 0==> la con trai, 1 la con phai
void insertNode_v2(TREENode* root, char label, int leftorRight)
{
    if (root == NULL) return;
    TREENode* newNode = makeNode(label);
    if (leftorRight == 0) root->left = newNode;
    else root->right = newNode;
}

void preorderTraversal(TREENode* root)
{
    if (root == NULL) return;
    printf("%c ", root->label);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}
```

11/18/2021

hiepnd@soict.hust.edu.vn

47

```
void inorderTraversal(TREENode* root)
{
    if (root == NULL) return;
    inorderTraversal(root->left);
    printf("%c ", root->label);
    inorderTraversal(root->right);
}

void postorderTraversal(TREENode* root)
{
    if (root == NULL) return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%c ", root->label);
}
```

```
int countNode(TREENode* root)
{
    if (NULL == root) return 0;
    return 1 + countNode(root->left) + countNode(root->right);
}

int countLeaves(TREENode* root)
{
    if (NULL == root) return 0;
    if (NULL == root->left && NULL == root->right) return 1;
    return countLeaves(root->left) + countLeaves(root->right);
}

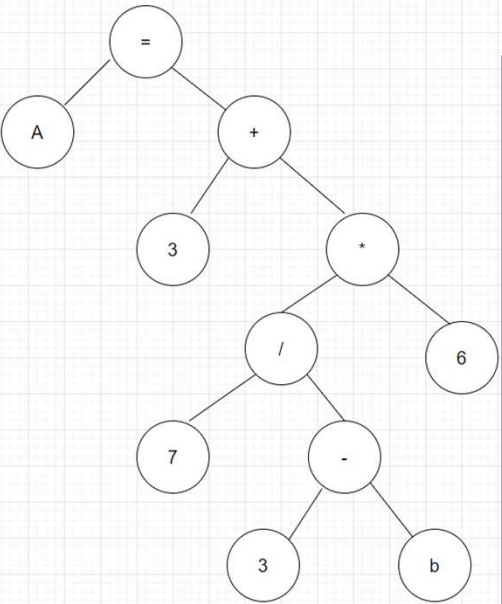
int countInternalNodes(TREENode* root)
{
    if (NULL == root) return 0;
    if (NULL == root->left && NULL == root->right) return 0;
    return 1 + countInternalNodes(root->left) +
        countInternalNodes(root->right);
}
```

11/18/2021

hiepnd@soict.hust.edu.vn

48





```

int main()
{
    TreeNode* root = makeNode('=');
    insertNode_v2(root, 'A', 0);
    insertNode_v2(root, '+', 1);
    insertNode_v2(root->right, '3', 0);
    insertNode_v2(root->right, '*', 1);
    insertNode_v2(root->right->right, '/', 0);
    insertNode_v2(root->right->right, '6', 1);
    insertNode_v2(root->right->right->left, '7', 0);
    insertNode_v2(root->right->right->left, '-', 1);
    insertNode_v2(root->right->right->left->right, '3', 0);
    insertNode_v2(root->right->right->left->right, 'b', 1);
    inorderTraversal(root);
    printf("\nSo luong nut: %d\n", countNode(root));
    printf("\nSo luong nut la: %d\n", countLeaves(root));
    printf("\nSo luong nut trong: %d\n", countInternalNodes(root));
    return 0;
}

```

11/18/2021 hiepnd@soict.hust.edu.vn 49

## Mở rộng của cây tổng quát