



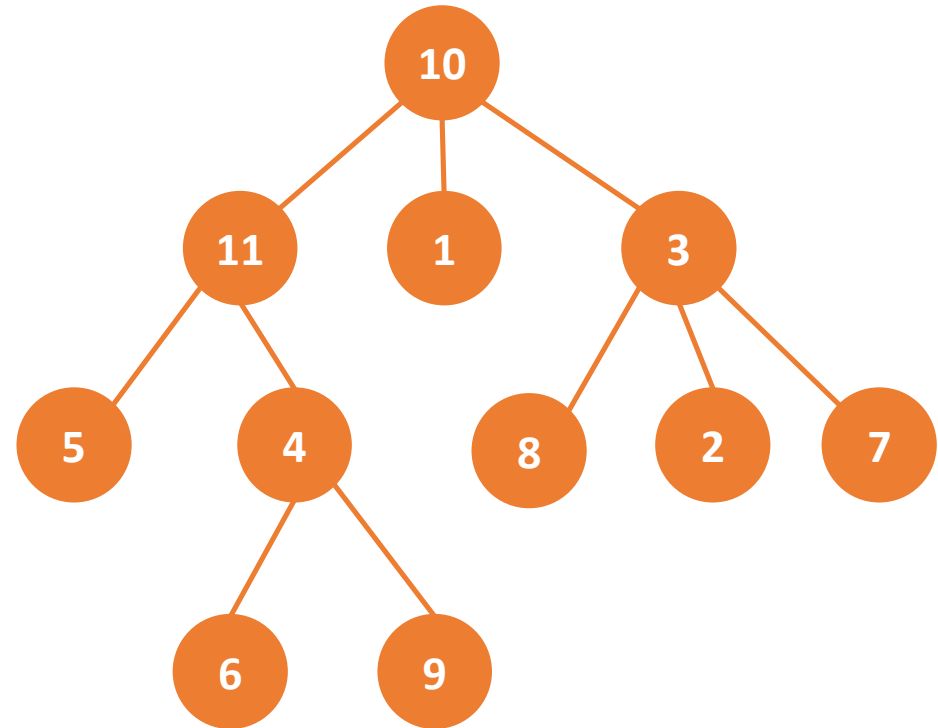
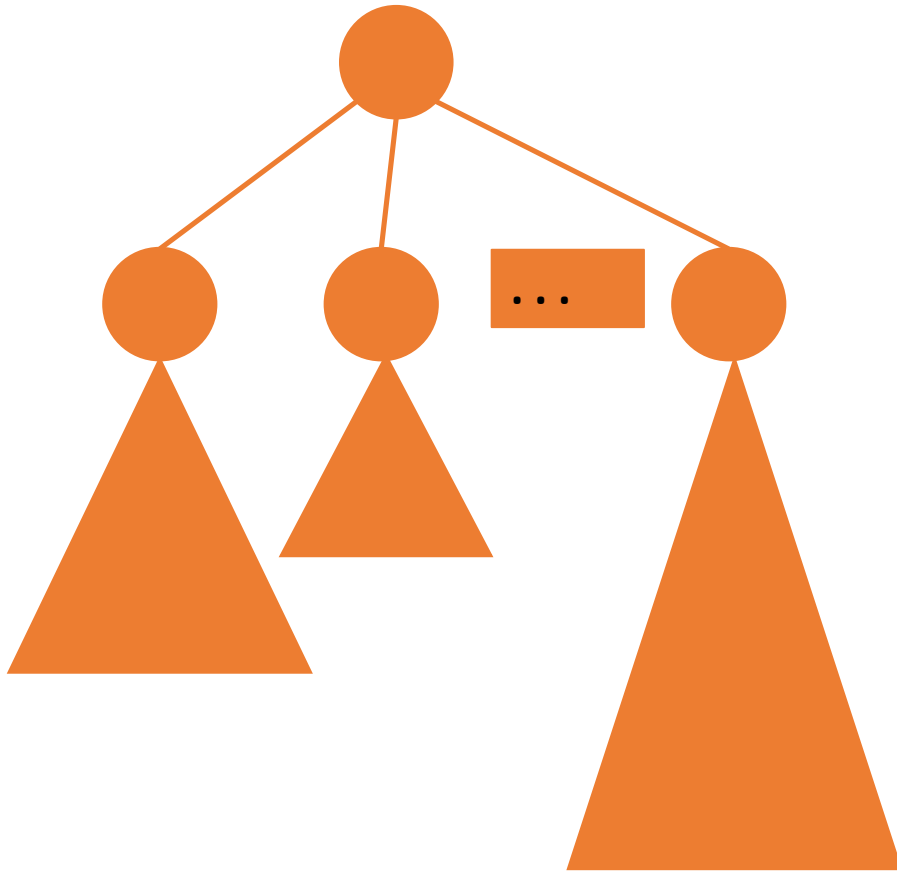
ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN Cây

Nội dung

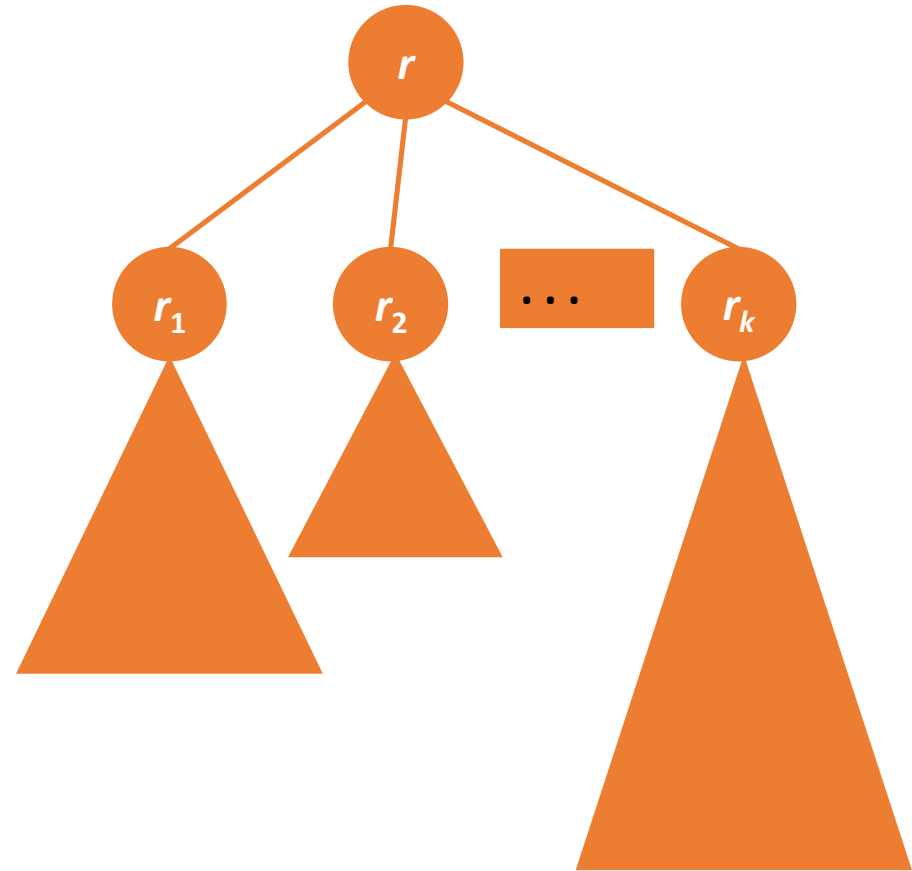
- Định nghĩa cây
- Các khái niệm trên cây
- Các phép duyệt cây
- Cấu trúc lưu trữ
- Các thao tác trên cây

Cây



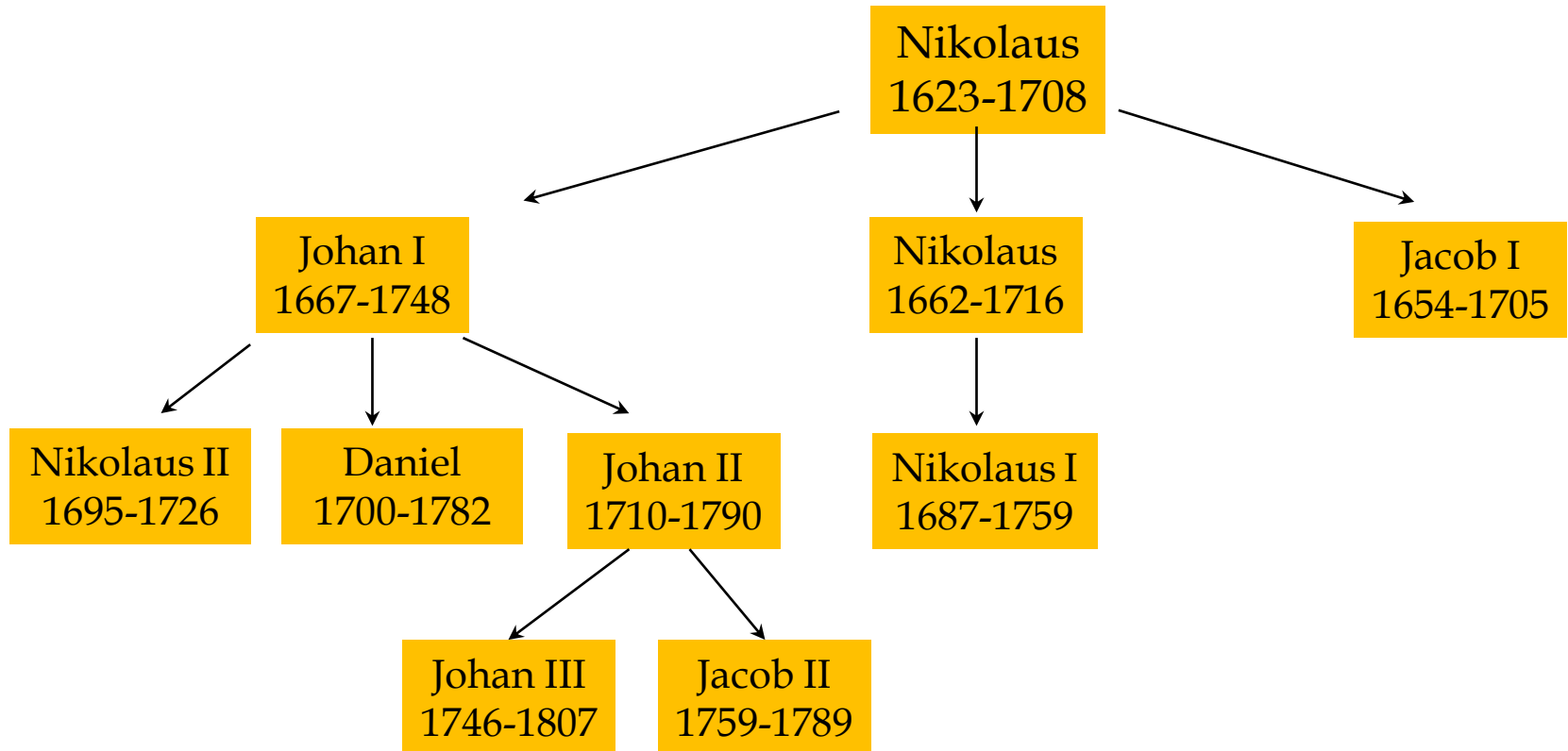
Định nghĩa cây

- Cấu trúc lưu trữ các đối tượng có quan hệ phân cấp
- Định nghĩa đệ quy
 - Bước cơ sở: r là 1 nút, cây T có nút gốc là r
 - Bước đệ quy:
 - Giả sử T_1, T_2, \dots, T_k là các cây có gốc là r_1, r_2, \dots, r_k
 - Nút r
 - Liên kết các nút r_1, r_2, \dots, r_k như là nút con của r sẽ tạo ra cây T

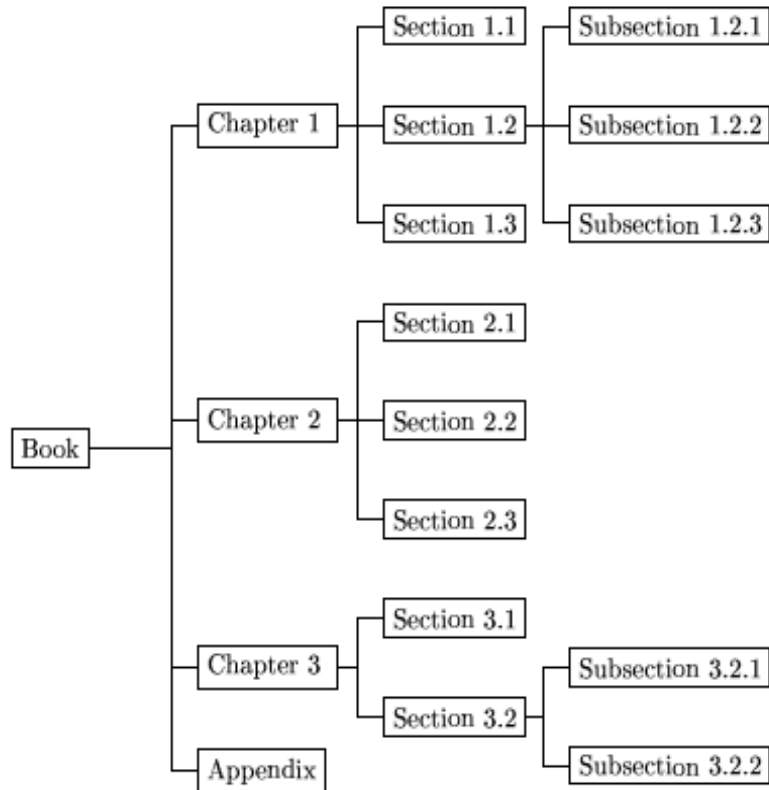


Các ứng dụng của cây: Cây gia phả

- Cây gia phả của các nhà toán học dòng họ Bernoulli



Các ứng dụng của cây

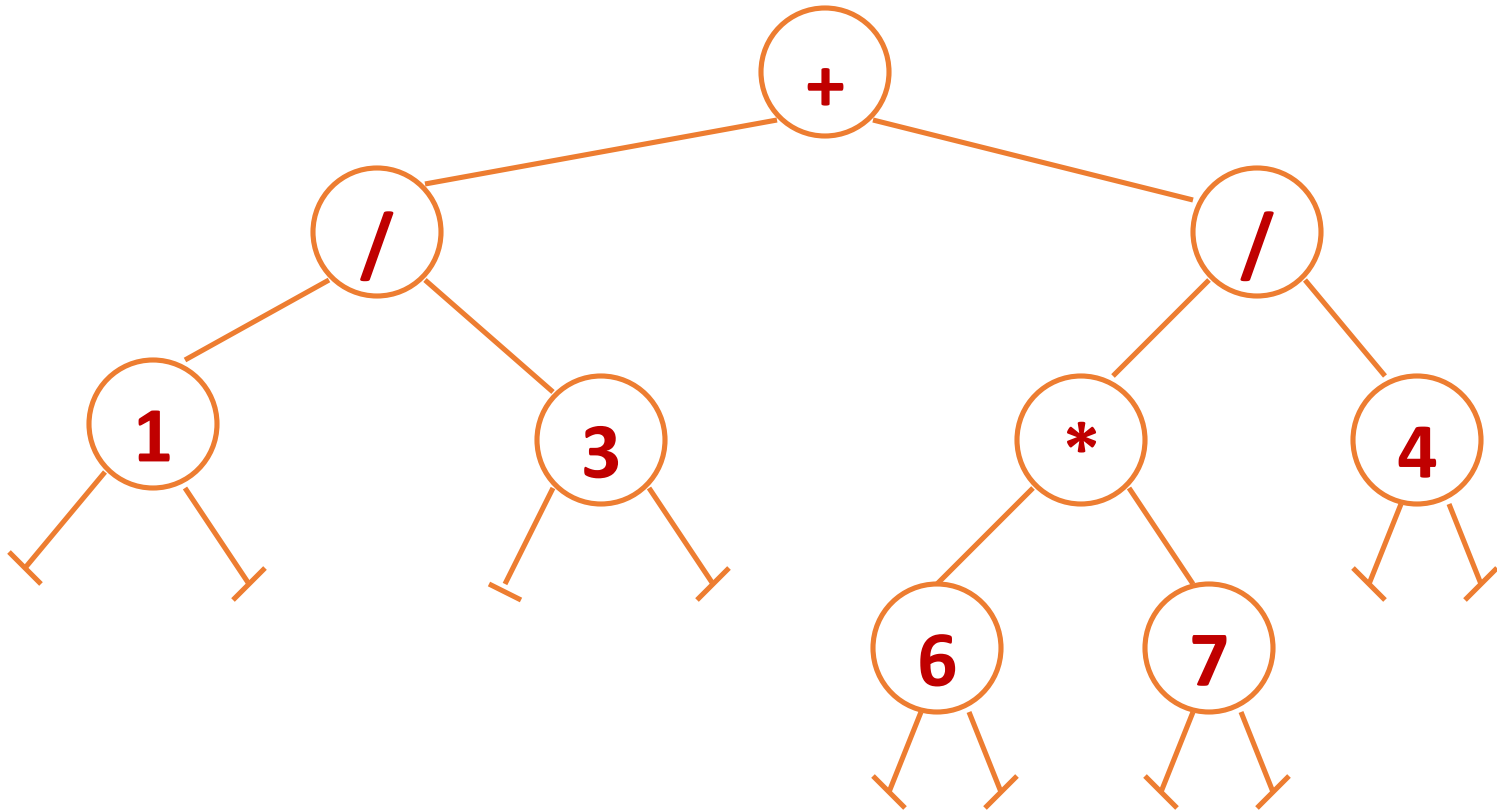


Mục lục sách



Cây thư mục

Các ứng dụng của cây: cây biểu thức



$$1/3 + 6*7/4$$

Các khái niệm trên cây

- Đường đi: dãy các nút x_1, x_2, \dots, x_q trong đó x_i là cha của x_{i+1} , $i = 1, 2, \dots, q-1$. Độ dài đường đi là $q-1$
- Nút lá: không có nút con
- Nút trong: có nút con
- Nút anh em: 2 nút u và v là 2 nút anh em nếu có cùng nút cha
- Tổ tiên: nút u là tổ tiên của v nếu có đường đi từ u đến v
- Con cháu: nút u là con cháu của v nếu v là tổ tiên của u
- Độ cao: độ cao của 1 nút là độ dài đường đi dài nhất từ nút đó đến nút lá + 1
- Độ sâu: độ sâu của 1 nút v là độ dài đường đi duy nhất từ nút gốc tới nút đó + 1

Các khái niệm trên cây

- **Gốc:** nút không có cha (ví dụ: nút A)
- **Anh em:** nút có cùng cha (ví dụ: B, C, D là anh em; I, J, K là anh em; E và F là anh em; G và H là anh em)
- **Nút trong:** nút có ít nhất 1 con (ví dụ: các nút màu xanh dương: A, B, C, F)
- **Nút ngoài (lá):** nút không có con (ví dụ: các nút xanh lá: E, I, J, K, G, H, D)
- **Tổ tiên** của 1 nút: là các nút cha / ông bà / cụ.. của nút đó.
- **Hậu duệ** của 1 nút: là các nút con/cháu/chắt... của nút đó
- **Cây con** của 1 nút: là cây có gốc là con của nút đó

Con của B:

- E, F

Cha của E:

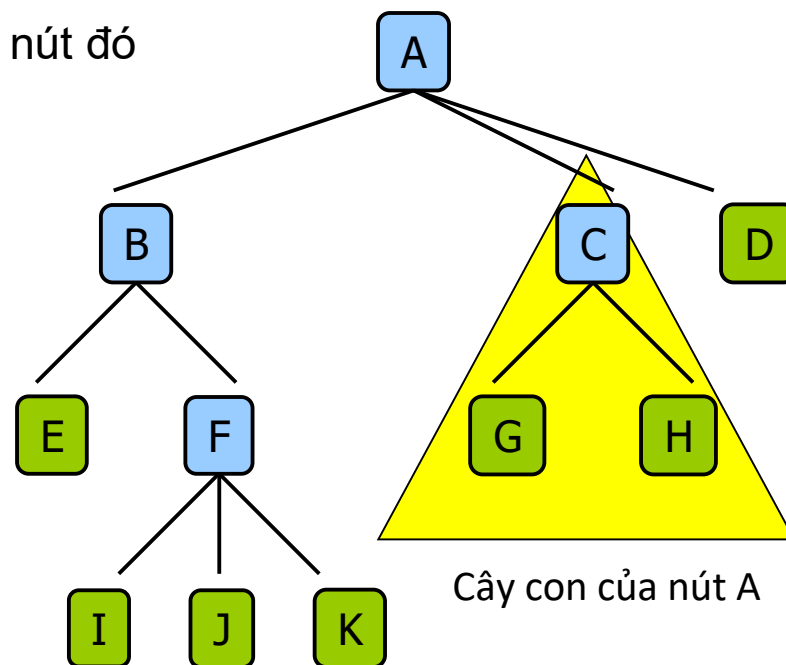
- B

Tổ tiên của F:

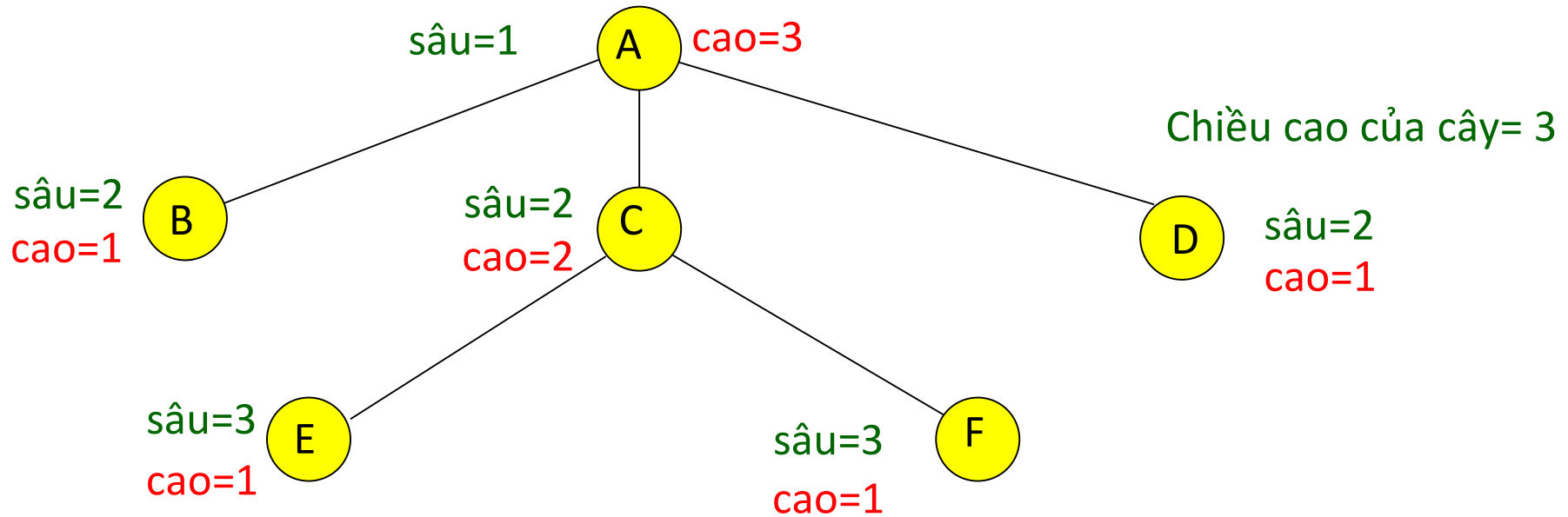
- B, A

Hậu duệ của B:

- E, F, I, J, K



Các khái niệm trên cây



Phép duyệt cây

- Thăm các nút của 1 cây theo 1 thứ tự nào đó
- 3 phép duyệt cây cơ bản
 - Duyệt theo thứ tự trước
 - Duyệt theo thứ tự giữa
 - Duyệt theo thứ tự sau
- Xét cây T có cấu trúc
 - Nút gốc r
 - Cây con T_1 (gốc r_1), T_2 (gốc r_2), ..., T_k (gốc r_k) theo thứ tự từ trái qua phải

Phép duyệt cây

- Duyệt theo thứ tự trước
 - Thăm nút gốc
 - Duyệt cây T_1 theo thứ tự trước
 - Duyệt cây T_2 theo thứ tự trước
 - ...
 - Duyệt cây T_k theo thứ tự trước

```
preOrder(r){  
    if(r = NULL) return;  
    visit(r);  
    for each p =  $r_1, r_2, \dots, r_k$  {  
        preOrder(p);  
    }  
}
```

Phép duyệt cây

- Duyệt theo thứ tự giữa
 - Duyệt cây T_1 theo thứ tự giữa
 - Thăm nút gốc r
 - Duyệt cây T_2 theo thứ tự giữa
 - ...
 - Duyệt cây T_k theo thứ tự giữa

```
inOrder(r){  
    if(r = NULL) return;  
    inOrder(r1);  
    visit(r);  
    for each p = r2, ..., rk {  
        inOrder(p);  
    }  
}
```

Phép duyệt cây

- Duyệt theo thứ tự sau
 - Duyệt cây T_1 theo thứ tự sau
 - Duyệt cây T_2 theo thứ tự sau
 - ...
 - Duyệt cây T_k theo thứ tự sau
 - Thăm nút gốc

```
postOrder(r){  
    if(r = NULL) return;  
    for each p =  $r_1, r_2, \dots, r_k$  {  
        postOrder(p);  
    }  
    visit(r);  
}
```

Cấu trúc dữ liệu biểu diễn cây

- Mảng:
 - Giả sử các nút trên cây được định danh $1, 2, \dots, n$
 - $a[1..n]$ trong đó $a[i]$ là nút cha của nút i
 - Cấu trúc lưu trữ đơn giản, tuy nhiên khó cài đặt rất nhiều thao tác trên cây
- Con trỏ liên kết: với mỗi nút, lưu 2 con trỏ
 - Con trỏ trỏ đến nút con trái nhất
 - Con trỏ trỏ đến nút anh em bên phải

Cấu trúc dữ liệu biểu diễn cây

```
struct Node{  
    int id;  
    Node* leftMostChild;// con tro tro den nut con trai nhat  
    Node* rightSibling;// con tro tro den nut anh em ben phai  
};  
Node* root;// con tro tro den nut goc
```


Thao tác trên cây

- $\text{find}(r, \text{id})$: tìm nút có định danh là id trên cây có gốc là r
- $\text{insert}(r, p, \text{id})$: tạo một nút có định danh là id , chèn vào cuối danh sách nút con của nút p trên cây có gốc là r
- $\text{height}(r, p)$: trả về độ cao của nút p trên cây có gốc là r
- $\text{depth}(r, p)$: trả về độ sâu của nút p trên cây có gốc là r
- $\text{parent}(r, p)$: trả về nút cha của nút p trên cây có gốc r
- $\text{count}(r)$: trả về số nút trên cây có gốc là r
- $\text{countLeaves}(r)$: trả về số nút lá trên cây có gốc là r

Thao tác trên cây

- Tìm một nút có nhãn cho trước trên cây

```
Node* find(Node* r, int v){  
    if(r == NULL) return NULL;  
    if(r->id == v) return r;  
    Node* p = r->leftMostChild;  
    while(p != NULL){  
        Node* h = find(p,v);  
        if(h != NULL) return h;  
        p = p->rightSibling;  
    }  
    return NULL;  
}
```

Thao tác trên cây

- Duyệt theo thứ tự trước

```
void preOrder(Node* r){  
    if(r == NULL) return;  
    printf("%d ",r->id);  
    Node* p = r->leftMostChild;  
    while(p != NULL){  
        preOrder(p);  
        p = p->rightSibling;  
    }  
}
```

Thao tác trên cây

- Duyệt theo thứ tự giữa

```
void inOrder(Node* r){  
    if(r == NULL) return;  
    Node* p = r->leftMostChild;  
    inOrder(p);  
    printf("%d ",r->id);  
    if(p != NULL)  
        p = p->rightSibling;  
    while(p != NULL){  
        inOrder(p);  
        p = p->rightSibling;  
    }  
}
```

Thao tác trên cây

- Duyệt theo thứ tự sau

```
void postOrder(Node* r){  
    if(r == NULL) return;  
    Node* p = r->leftMostChild;  
    while(p != NULL){  
        postOrder(p);  
        p = p->rightSibling;  
    }  
    printf("%d ",r->id);  
}
```

Thao tác trên cây

- Đếm số nút trên cây

```
int count(Node* r){  
    if(r == NULL) return 0;  
    int s = 1;  
    Node* p = r->leftMostChild;  
    while(p != NULL){  
        s += count(p);  
        p = p->rightSibling;  
    }  
    return s;  
}
```

Thao tác trên cây

- Đếm số nút lá trên cây

```
int countLeaves(Node* r){  
    if(r == NULL) return 0;  
    int s = 0;  
    Node* p = r->leftMostChild;  
    if(p == NULL) s = 1;  
    while(p != NULL){  
        s += countLeaves(p);  
        p = p->rightSibling;  
    }  
    return s;  
}
```

Thao tác trên cây

- Độ cao của một nút

```
int height(Node* p){  
    if(p == NULL) return 0;  
    int maxh = 0;  
    Node* q = p->leftMostChild;  
    while(q != NULL){  
        int h = height(q);  
        if(h > maxh) maxh = h;  
        q = q->rightSibling;  
    }  
    return maxh + 1;  
}
```


Thao tác trên cây

- Độ sâu của một nút

```
int depth(Node* r, int v, int d){
    // d là do sau của nút r
    if(r == NULL) return -1;
    if(r->id == v) return d;
    Node* p = r->leftMostChild;
    while(p != NULL){
        if(p->id == v) return d+1;
        int dv = depth(p,v,d+1);
        if(dv > 0) return dv;
        p = p->rightSibling;
    }
    return -1;
}

int depth(Node* r, int v){
    return depth(r,v,1);
}
```

Thao tác trên cây

- Tìm nút cha của một nút

```
Node* parent(Node* p, Node* r){  
    if(r == NULL) return NULL;  
    Node* q = r->leftMostChild;  
    while(q != NULL){  
        if(p == q) return r;  
        Node* pp = parent(p, q);  
        if(pp != NULL) return pp;  
        q = q->rightSibling;  
    }  
    return NULL;  
}
```

Cây nhị phân

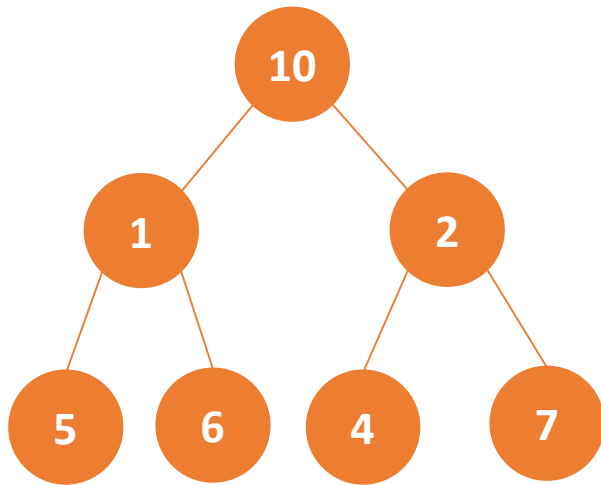
- Mỗi nút có nhiều nhất 2 nút con
- Hai con trỏ xác định nút con trái và nút con bên phải

```
struct BNode{  
    int id;  
    BNode* leftChild; // con trỏ đến nút con trái  
    BNode* rightChild; // con trỏ đến nút con phải  
};
```

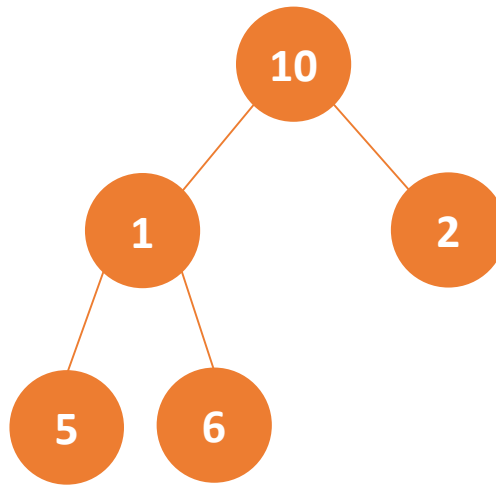
- **leftChild = NULL**: có nghĩa nút hiện tại không có con trái
- **rightChild = NULL**: có nghĩa nút hiện tại không có con phải
- Có thể áp dụng sơ đồ thuật toán trên cây tổng quát cho trường hợp cây nhị phân

Cây nhị phân

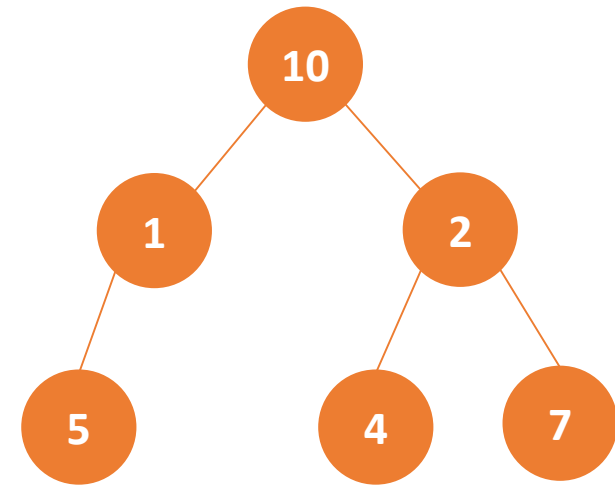
- Phân loại



Cây nhị phân hoàn chỉnh



Cây nhị phân đầy đủ



Cây nhị phân cân bằng

Thao tác trên cây nhị phân

- Duyệt theo thứ tự giữa
 - Duyệt theo thứ tự giữa cây con bên trái
 - Thăm nút gốc
 - Duyệt theo thứ tự giữa cây con bên phải

```
void inOrder(BNode* r) {  
    if(r == NULL) return;  
    inOrder(r->leftChild);  
    printf("%d ", r->id);  
    inOrder(r->rightChild);  
}
```

Thao tác trên cây nhị phân

- Duyệt theo thứ tự trước
 - Thăm nút gốc
 - Duyệt theo thứ tự giữa cây con bên trái
 - Duyệt theo thứ tự giữa cây con bên phải

```
void preOrder(BNode* r) {  
    if(r == NULL) return;  
    printf("%d ", r->id);  
    preOrder(r->leftChild);  
    preOrder(r->rightChild);  
}
```

Thao tác trên cây nhị phân

- Duyệt theo thứ tự sau
 - Duyệt theo thứ tự giữa cây con bên trái
 - Duyệt theo thứ tự giữa cây con bên phải
 - Thăm nút gốc

```
void postOrder(BNode* r) {  
    if(r == NULL) return;  
    postOrder(r->leftChild);  
    postOrder(r->rightChild);  
    printf("%d ", r->id);  
}
```

Thao tác trên cây nhị phân

- Đếm số nút trên cây nhị phân

```
int count(BNode* r) {  
    if(r == NULL) return 0;  
    return 1 + count(r->leftChild) +  
            count(r->rightChild);  
}
```

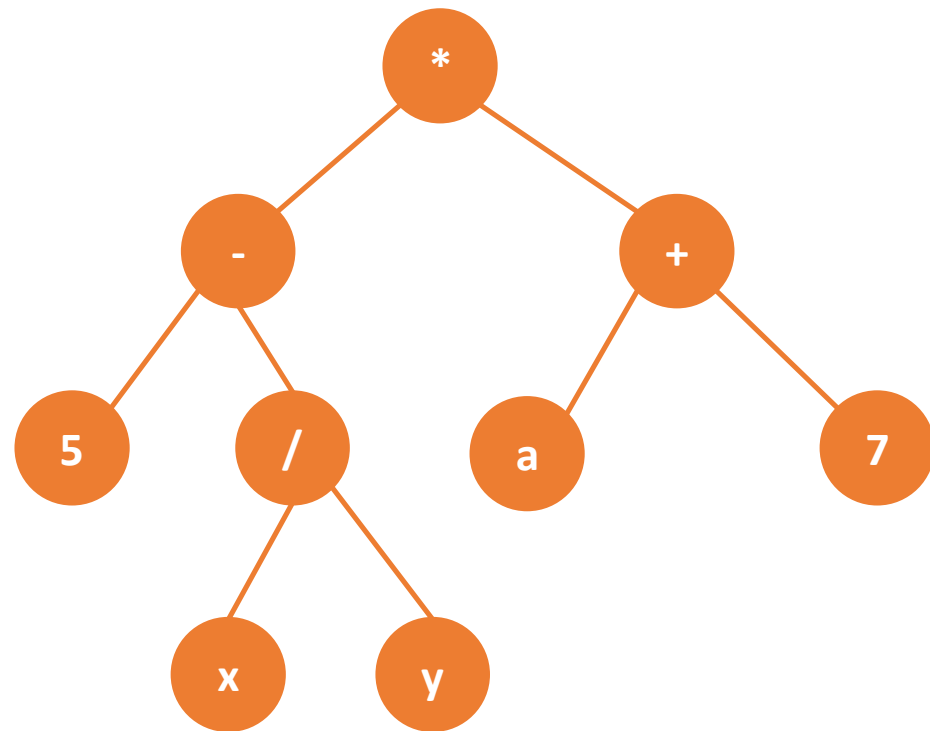

Cây biểu thức

- Là cây nhị phân
 - Nút giữa biểu diễn toán tử
 - Nút lá biểu diễn các toán hạng (biến, hằng)
- Biểu thức trung tố: dãy các nút được thăm trong phép duyệt cây theo thứ tự giữa:

$$(5 - x/y) * (a + 7)$$

- Biểu thức hậu tố: dãy các nút được thăm trong phép duyệt cây theo thứ tự sau:

$$5 \ x \ y \ / \ - \ a \ 7 \ + \ *$$



Tính giá trị của biểu thức hậu tố

- Khởi tạo stack S ban đầu rỗng

Tính giá trị của biểu thức hậu tố

- Khởi tạo stack S ban đầu rỗng
- Duyệt các phần tử của biểu thức hậu tố từ trái qua phải

Tính giá trị của biểu thức hậu tố

- Khởi tạo stack S ban đầu rỗng
- Duyệt các phần tử của biểu thức hậu tố từ trái qua phải
- Nếu gặp toán hạng thì đẩy toán hạng đó vào S

Tính giá trị của biểu thức hậu tố

- Khởi tạo stack S ban đầu rỗng
- Duyệt các phần tử của biểu thức hậu tố từ trái qua phải
- Nếu gặp toán hạng thì đẩy toán hạng đó vào S
- Nếu gặp toán tử **op** thì lần lượt lấy 2 toán hạng A và B ra khỏi S, thực hiện $C = B \text{ op } A$, và đẩy C vào S

Tính giá trị của biểu thức hậu tố

- Khởi tạo stack S ban đầu rỗng
- Duyệt các phần tử của biểu thức hậu tố từ trái qua phải
- Nếu gặp toán hạng thì đẩy toán hạng đó vào S
- Nếu gặp toán tử **op** thì lần lượt lấy 2 toán hạng A và B ra khỏi S, thực hiện $C = B \text{ op } A$, và đẩy C vào S
- Khi biểu thức hậu tố được duyệt xong thì giá trị còn lại trong S chính là giá trị của biểu thức đã cho