Bài 4:CÂY, CÂY NHỊ PHÂN, CÂY NHỊ PHÂN TÌM KIẾM

1. Cấu trúc cây

1.1. Định nghĩa 1:

Cây là một tập hợp T các phần tử (nút trên cây) trong đó có 1 nút đặc biệt T0 được gọi là gốc, các nút còn khác được chia thành những tập rời nhau T1, T2 , ... , Tn theo quan hệ phân cấp trong đó Ti cũng là một cây.

Nút ở cấp i sẽ quản lý một số nút ở cấp i+1. Quan hệ này người ta còn gọi là quan hệ cha-con.

1.2. Một số khái niệm cơ bản

- Bậc của một nút: là số cây con của nút đó .
- Bậc của một cây: là bậc lớn nhất của các nút trong cây. Cây có bậc n thì gọi là cây n-phân.
- Nút gốc: nút không có nút cha.
- Nút lá: nút có bậc bằng 0 .
- Nút nhánh: nút có bậc khác 0 và không phải là gốc .
- Mức của một nút:

  Mức (T0) = 1.

  Gọi T1, T2, T3, ... , Tn là các cây con của T0

  Mức (T1) = Mức (T2) = ... = Mức (Tn) = Mức (T0) + 1.
- Độ dài đường đi từ gốc đến nút x: là số nhánh cần đi qua kể từ gốc đến x.
- Chiều cao h của cây: mức lớn nhất của các nút lá.

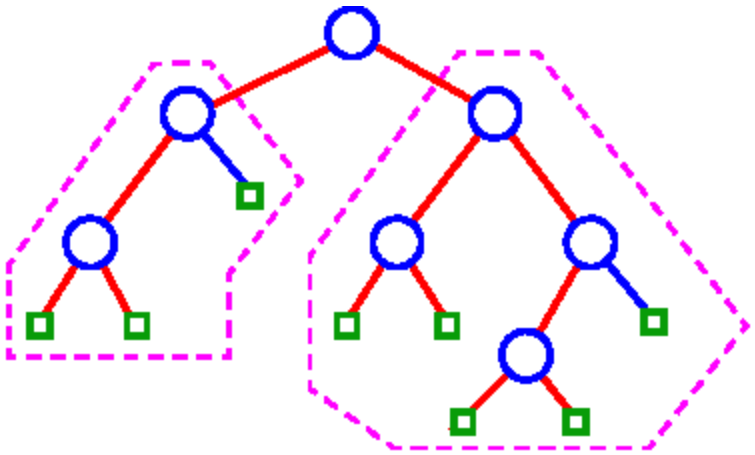
1.3. Một số ví dụ về đối tượng các cấu trúc dạng cây

- Sơ đồ tổ chức của một doanh nghiệp
- Sơ đồ tổ chức cây thư mục

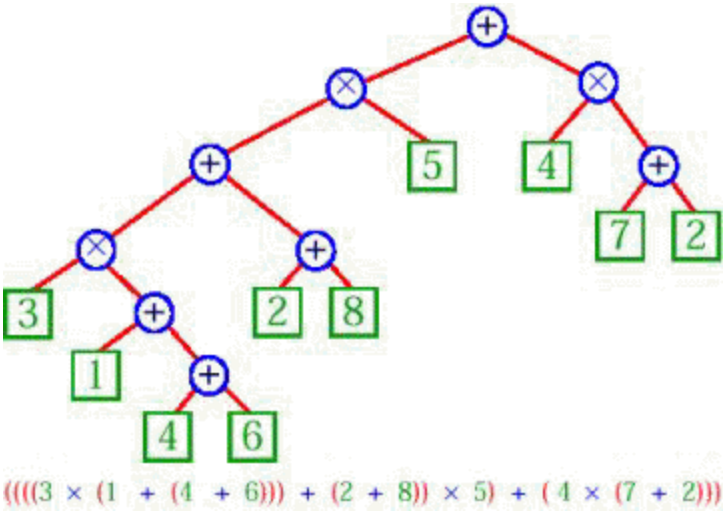
2. CÂY NHỊ PHÂN

2.1 Định nghĩa

Cây nhị phân là cây mà mỗi nút có tối đa 2 cây con

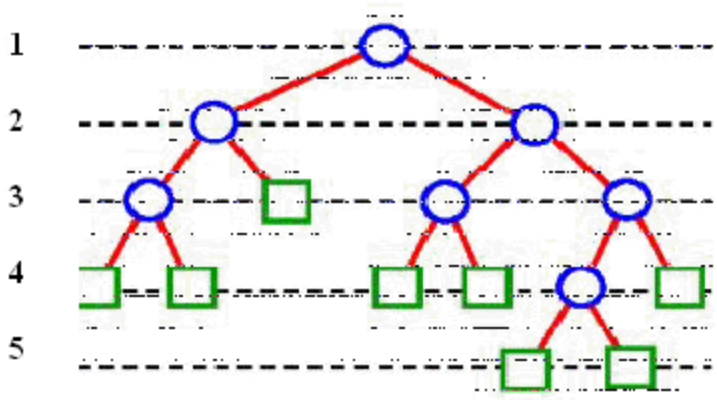


Cây nhị phân có thể ứng dụng trong nhiều bài toán thông dụng. Ví dụ dưới đây cho ta hình ảnh của một biểu thức toán học:



2.2. Một số tính chất của cây nhị phân:

- Số nút ở mức $I \leq 2^{I-1}$.
- Số nút ở mức lá $\leq 2^{h-1}$, với h là chiều cao của cây.
- Chiều cao của cây $h \geq \log_2 N$ (N - số nút trên trong cây).



2.3. Biểu diễn cây nhị phân T

Cây nhị phân là một cấu trúc bao gồm các phần tử (nút) được kết nối với nhau theo quan hệ "cha-con" với mỗi cha có tối đa 2 con. Để biểu diễn cây nhị phân ta chọn phương pháp cấp phát liên kết. Ứng với một nút, ta dùng một biến động lưu trữ các thông tin:

+ Thông tin lưu trữ tại nút.
+ Địa chỉ nút gốc của cây con trái trong bộ nhớ.
+ Địa chỉ nút gốc của cây con phải trong bộ nhớ.

Khai báo như sau:

```
typedef struct        tagTNODE
{
    Data   Key;//Data là kiểu dữ liệu ứng với thông tin lưu tại nút
    struct tagNODE *pLeft, *pRight;
```

```
}TNODE;
typedef TNODE    *TREE;
```

2.4. Các thao tác trên cây nhị phân

Thăm các nút trên cây theo thứ tự trước (Node-Left-Right)

```
void   NLR(TREE Root)
{
        if (Root != NULL)
       {
        <Xử lý Root>;       //Xử lý tương ứng theo nhu cầu
        NLR(Root->pLeft);
        NLR(Root->pRight);
        }
}
```

Thăm các nút trên cây theo thứ tự giữa (Left- Node-Right)

```
 void  LNR(TREE Root)
{
        if (Root != NULL)
       {
        LNR(Root->Left);
        <Xử lý Root>;                    //Xử lý tương ứng theo nhu cầu
        LNR(Root->Right);
        }
}
```

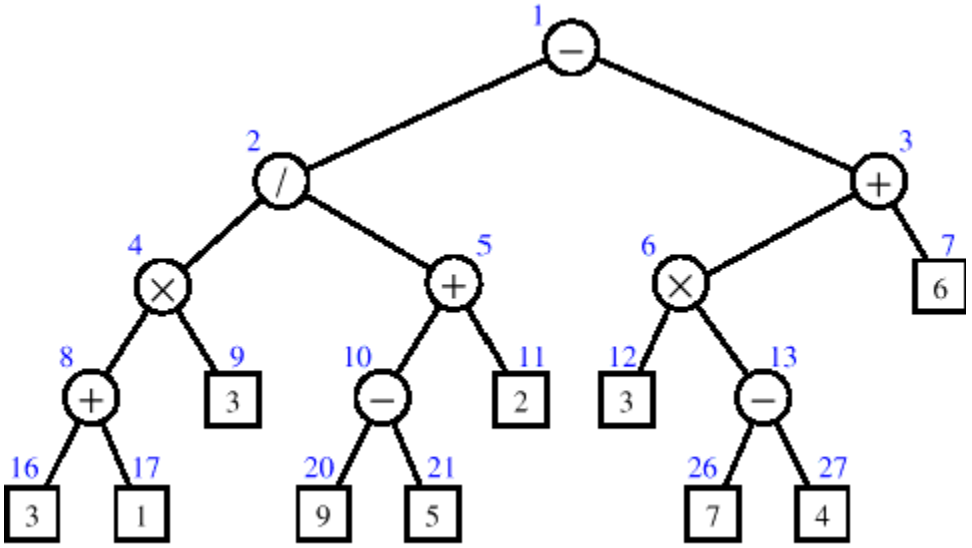
Thăm các nút trên cây theo thứ tự sau (Left-Right-Node)

```
void   LRN(TREE Root)
{
```

```
        if (Root != NULL)
        {
        LRN(Root->Left);
        LRN(Root->Right);
        <Xử lý Root>;       //Xử lý tương ứng theo nhu cầu
        }
}
```

Ứng dụng phương pháp này trong việc tính tổng kích thước của thư mục.

Ứng dụng tính toán giá trị của biểu thức.



$(3 + 1)\times3/(9 – 5 + 2) – (3\times(7 – 4) + 6) = –13$

2.5. Biểu diễn cây tổng quát bằng cây nhị phân

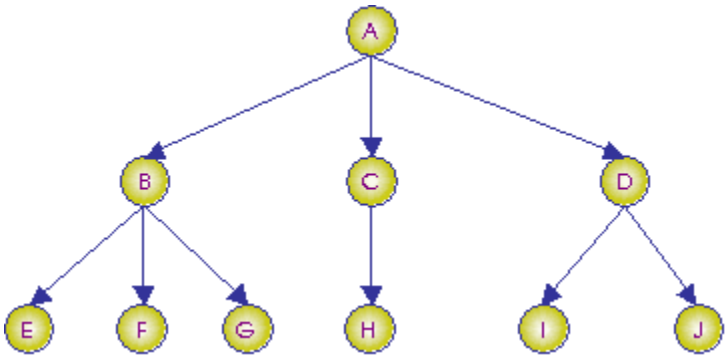
Nhược điểm của các cấu trúc cây tổng quát là bậc của các nút trên cây có thể rất khác nhau ⇒ việc biểu diễn gặp nhiều khó khăn và lãng phí. Hơn nữa, việc xây dựng các thao tác trên cây tổng quát phức tạp hơn trên cây nhị phân nhiều.

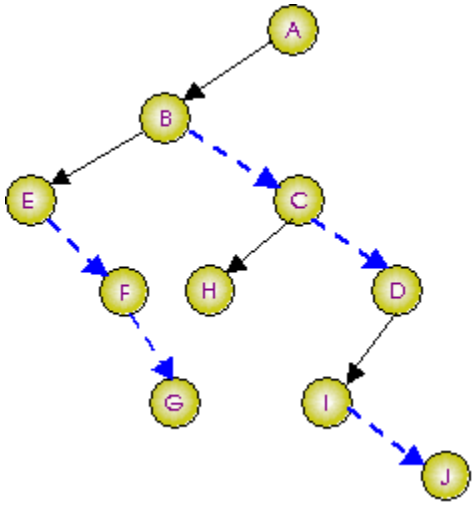
Vì vậy, nếu không quá cần thiết phải sử dụng cây tổng quát, người ta sẽ biến đổi cây tổng quát thành cây nhị phân.

Ta có thể biến đổi một cây bất kỳ thành một cây nhị phân theo qui tắc sau:

- Giữ nút con trái nhất làm con trái.
- Các nút con còn lại biến đổi thành nút con phải.

VD: Giả sử có cây tổng quát như hình sau:





Cây nhị phân tương ứng sẽ như sau:

2.6. Một cách biểu diễn cây nhị phân khác

Đôi khi, trên cây nhị phân, người ta quan tâm đến cả quan hệ chiều cha con. Khi đó, cấu trúc cây nhị phân có thể định nghĩa lại như sau:

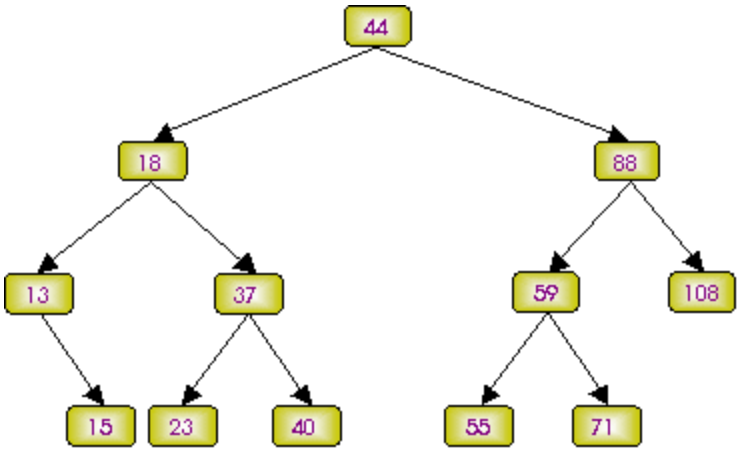
```
typedef struct tagTNode
{
DataType     Key;
struct tagTNode*   pParent;
struct tagTNode*   pLeft;
struct tagTNode*   pRight;
}TNODE;
typedef TNODE     *TREE;
```

3. CÂY NHỊ PHÂN TÌM KIẾM

3.1. Định nghĩa:

Cây nhị phân tìm kiếm (CNPTK) là cây nhị phân trong đó tại mỗi nút, khóa của nút đang xét lớn hơn khóa của tất cả các nút thuộc cây con trái và nhỏ hơn khóa của tất cả các nút thuộc cây con phải.

Dưới đây là một ví dụ về cây nhị phân tìm kiếm:



Nhờ ràng buộc về khóa trên CNPTK, việc tìm kiếm trở nên có định hướng. Hơn nữa, do cấu trúc cây việc tìm kiếm trở nên nhanh đáng kể. Chi phí tìm kiếm trung bình chỉ khoảng $\log_2 N$.

Trong thực tế, khi xét đến cây nhị phân chủ yếu người ta xét CNPTK.

3.2. Các thao tác trên cây

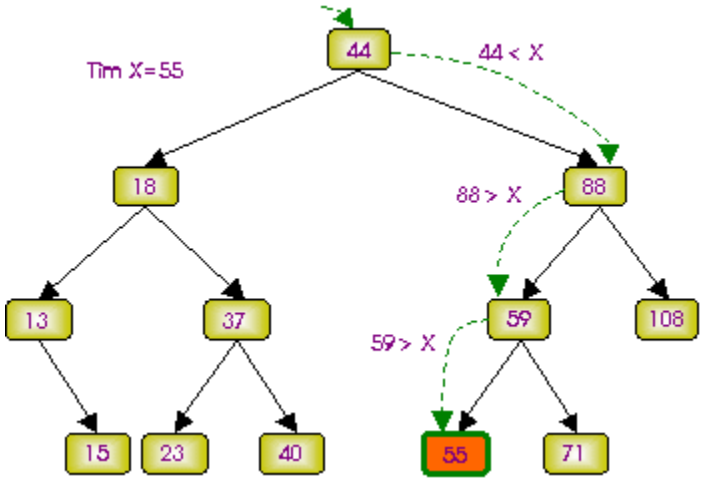
3.2.1. Thăm các nút trên cây

3.2.2. Tìm một phần tử x trong cây

TToán:

Dễ dàng thấy rằng số lần so sánh tối đa phải thực hiện để tìm phần tử X là bằng h, với h là chiều cao của cây.

Ví dụ: Tìm phần tử 55



3.3.3. Thêm một phần tử x vào cây

Việc thêm một phần tử X vào cây phải bảo đảm điều kiện ràng buộc của CNPTK. Ta có thể thêm vào nhiều vị trí khác nhau trên cây, nhưng nếu thêm vào một nút lá thì sẽ dễ nhất do ta có thể thực hiện quá trình tương tự thao tác tìm kiếm. Khi chấm dứt quá trình tìm kiếm ta sẽ tìm được vị trí cần thêm.

Hàm insert trả về giá trị –1, 0, 1 khi không đủ bộ nhớ, gặp nút cũ hay thành công:

```
int insertNode(TREE &T, Data X)
{
if(T)
{
        if(T->Key == X)    return 0; //đã có
        if(T->Key > X)
        return insertNode(T->pLeft, X);
        else
        return insertNode(T->pRight, X);
}
        T       = new TNode;
        if(T == NULL)      return -1; //thiếu bộ nhớ
        T->Key       = X;
        T->pLeft =T->pRight = NULL;
        return 1;  //thêm vào thành công
}
```

2.4. Hủy một phần tử có khóa x

Việc hủy một phần tử X ra khỏi cây phải bảo đảm điều kiện ràng buộc của CNPTK.

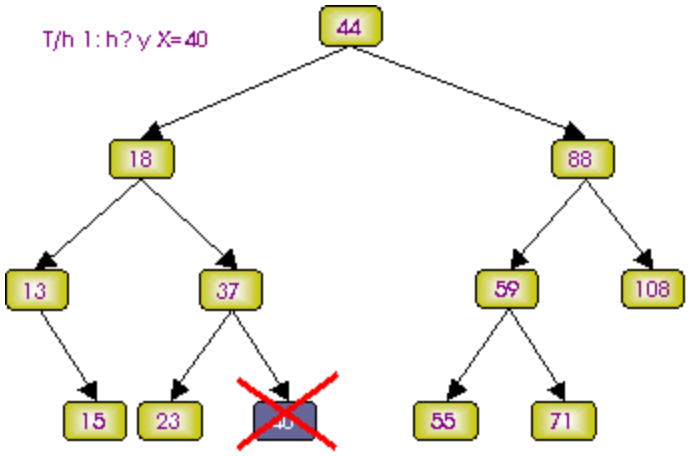
Có 3 trường hợp khi hủy nút X có thể xảy ra:

X - nút lá.

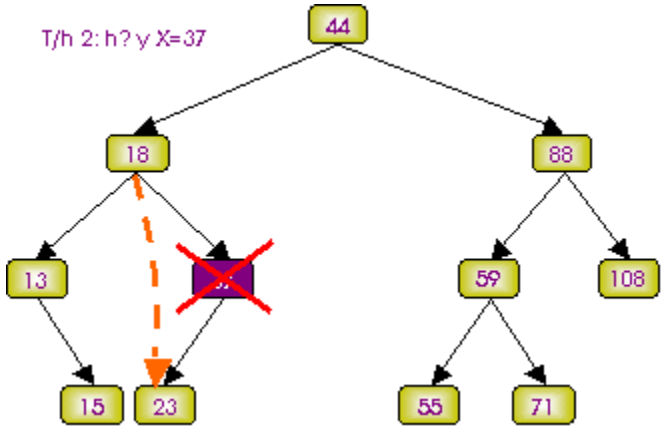
X - chỉ có 1 cây con (trái hoặc phải).

X có đủ cả 2 cây con

Trường hợp thứ nhất: chỉ đơn giản hủy X vì nó không móc nối đến phần tử nào khác.



Trường hợp thứ hai: trước khi hủy X ta móc nối cha của X với con duy nhất của nó.



Trường hợp cuối cùng: ta không thể hủy trực tiếp do X có đủ 2 con ⇒ Ta sẽ hủy gián tiếp. Thay vì hủy X, ta sẽ tìm một phần tử thế mạng Y. Phần tử này có tối đa một con. Thông tin lưu tại Y sẽ được chuyển lên lưu tại X. Sau đó, nút bị hủy thật sự sẽ là Y giống như 2 trường hợp đầu.

Vấn đề là phải chọn Y sao cho khi lưu Y vào vị trí của X, cây vẫn là CNPTK.

Có 2 phần tử thỏa mãn yêu cầu:

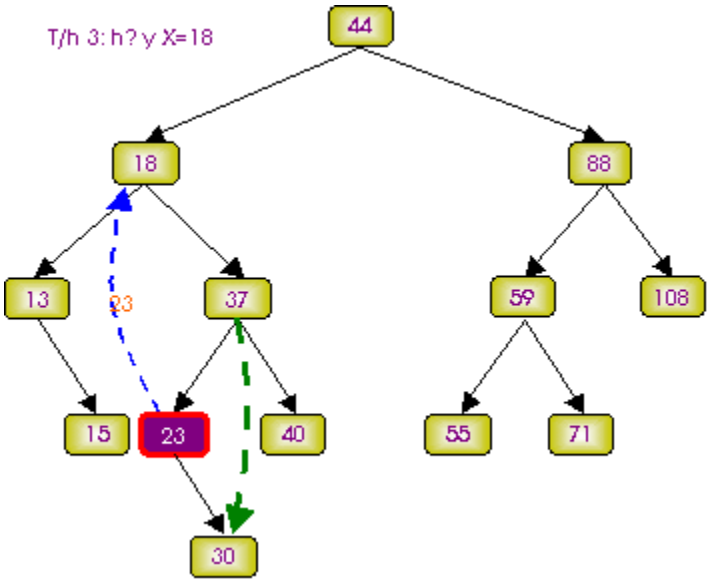
Phần tử nhỏ nhất (trái nhất) trên cây con phải.

Phần tử lớn nhất (phải nhất) trên cây con trái.

Việc chọn lựa phần tử nào là phần tử thế mạng hoàn toàn phụ thuộc vào ý thích của người lập trình. Ở đây, cháng tôi sẽ chọn phần tử (phải nhất trên cây con trái làm phân tử thế mạng.

VD:

Cần hủy phần tử 18.



2.5. ĐÁNH GIÁ

Tất cả các thao tác Tìm kiếm, Thêm mới, Xóa trên CNPTK đều có độ phức tạp trung bình O(h), với h là chiều cao của cây

Trong trong trường hợp tốt nhất, CNPTK có n nút sẽ có độ cao h = log2(n). Chi phí tìm kiếm khi đó sẽ tương đương tìm kiếm nhị phân trên mảng có thứ tự.

Tuy nhiên, trong trường hợp xấu nhất, cây có thể bị suy biến thành 1 DSLK. Lúc đó các thao tác trên sẽ có độ phức tạp O(n). Vì vậy cần có cải tiến cấu trúc của CNPTK để đạt được chi phí cho các thao tác là log2(n).