CHƯƠNG 4

CÂY

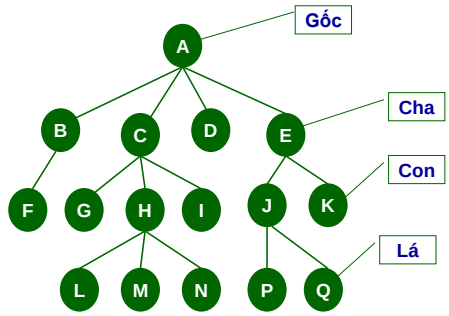
4.1. Cây và các khái niệm liên quan

4.1.1. Định nghĩa cây

– Định nghĩa 1: Cây là một đồ thị liên thông không có chu trình.

– Định nghĩa 2: Một cây là tập hợp hữu hạn các nút trong đó có một nút đặc biệt gọi là gốc (root). Giữa các nút có mối quan hệ phân cấp gọi là quan hệ cha-con.

Định nghĩa cây (tt)



4.1.2. Các khái niệm liên quan

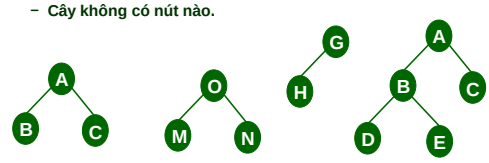
- **Bậc của một nút:**
  - Là số nút con của nút đó.
- **Bậc của một cây:**
  - Là bậc của nút có bậc lớn nhất trên cây đó. Cây có bậc n thì gọi là cây n – phân.
- **Nút gốc:**
  - Là nút đặc biệt, không có nút cha.
- **Nút lá:**
  - Là nút có bậc bằng 0 (không có nút con).
- **Nút nhánh:**
  - Là nút có bậc khác 0 và không phải là nút gốc.

Các khái niệm (tt)

- **Mức của một nút**
  - Gốc có mức 1.
  - Nếu nút cha có mức i thì các nút con có mức i+1.
- **Chiều cao của cây:**
  - Là mức của nút có mức lớn nhất có trên cây.
- **Đường đi:**
  - Dãy các nút $N_1, N_2, ..., N_k$ được gọi là đường đi nếu N_i là cha của N_{i+1} $(1 \le i \le k-1)$.
- **Độ dài của đường đi:**
  - Là số nút trên đường đi trừ đi 1.
- **Cây con:**
  - Là cây có gốc là một nút nhánh, lá.

Các khái niệm (tt)

- **Cây được sắp thứ tự:**
  - Các nút được sắp theo một thứ tự nhất định.
- **Rừng:**
  - Là tập hợp hữu hạn các cây phân biệt.
- **Cây rỗng:**
  - Cây không có nút nào.

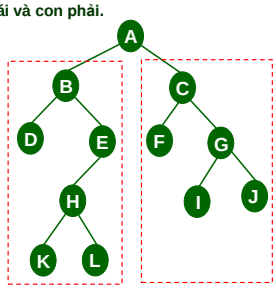


Cây sắp thứ tự

Rừng gồm 3 cây

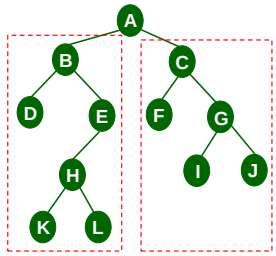
4.2. Cây nhị phân

- **Định nghĩa:**
  - Là cây mà mỗi nút không có quá 2 nút con, hai nút con (nếu có) được gọi là con trái và con phải.
- **Cây con trái:**
  - Là cây có gốc là nút con trái.
- **Cây con phải:**
  - Là cây có gốc là nút con phải.



Cây nhị phân (tt)

- **Tính chất:**
  - Số nút tối đa ở mức i trên cây nhị phân là $2^{(i-1)}$ ($i \geq 1$).
  - Số nút tối đa trên cây nhị phân chiều cao h là $2^h - 1$ ($h \geq 1$).



4.2.1. Lưu trữ cây nhị phân

4.2.1.1. Lưu trữ kế tiếp

- Phương pháp tự nhiên nhất để biểu diễn cây nhị phân là chỉ ra nút con trái và nút con phải của mỗi nút.
- Sử dụng một mảng để lưu trữ các nút của cây nhị phân.
- Mỗi nút của cây được biểu diễn bởi CẤU TRÚC gồm ba thành phần:

  infor: mô tả thông tin gắn với mỗi nút

  left: chỉ nút con trái

  right: chỉ nút con phải.

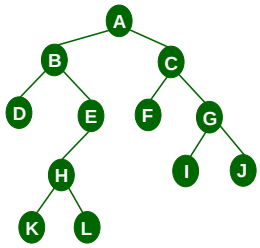
Lưu trữ kế tiếp (tt)

- Giả sử các nút của cây được được đánh số từ 0 đến MAX - 1, dữ liệu của các nút trên cây có kiểu là Item. Khi đó cấu trúc dữ liệu biểu diễn cây nhị phân được khai báo như sau:

```
#define MAX N
//Khai báo kiểu dữ liệu Item (nếu cần)
struct Node
{
      Item infor;
      int letf;
      int right;
};
Node T[MAX];
```


Lưu trữ kế tiếp (tt)

TT	infor	left	right
0	A	1	2
1	B	3	4
2	C	5	6
3	D	0	0
4	E	9	0
5	F	0	0
6	G	13	14
7	Ø	-	-
8	Ø	-	-
9	H	19	20
10	I	0	0
11	J	0	0
12	Ø	-	-
13	Ø	-	-
…	…	…	…



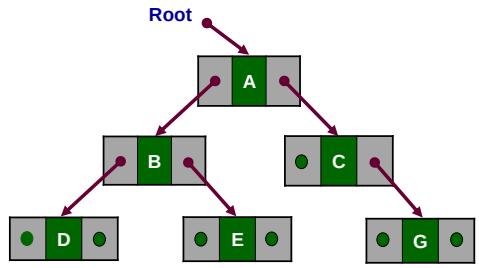
4.2.1.2. Lưu trữ móc nối

- Cách lưu trữ này khắc phục được nhược điểm của cách lưu trữ kế tiếp, đồng thời phản ánh được dạng tự nhiên của cây.
- Trong cách lưu trữ móc nối, mỗi nút tương ứng với một phần tử nhớ có qui cách như sau:
  - **infor:** ứng với thông tin (dữ liệu) của nút
  - **left:** là con trỏ, trỏ tới cây con trái của nút đó
  - **right:** là con trỏ, trỏ tới cây con phải của nút đó

Lưu trữ móc nối (tt)

```
//Khai báo kiểu dữ liệu Item
struct  Node
{
  Item  infor;
  Node  *left, *right;
};
typedef Node *TRO;
TRO Root;
//Root = NULL -> cây rỗng
```


Lưu trữ móc nối (tt)



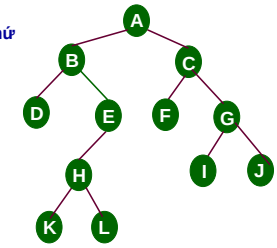
4.2.2. Duyệt cây nhị phân

- Duyệt cây nhị phân là "thăm" lần lượt các nút trên cây theo một thứ tự nhất định, mỗi nút một lần.
- Có 4 phương pháp duyệt cây:
  - TOP – DOWN – LEFT – RIGHT
  - Duyệt theo thứ tự trước – PreOrder
  - Duyệt theo thứ tự giữa – InOrder
  - Duyệt theo thứ tự sau – PostOrder

TOP – DOWN – LEFT – RIGHT

- Thăm lần lượt các nút theo thứ tự từ mức trên xuống mức dưới.
- Ở mỗi mức thăm theo thứ tự từ trái sang phải.
- Ví dụ:

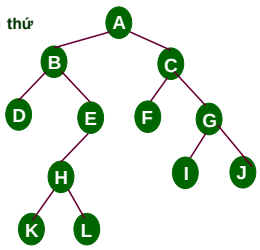
A B C D E F G H I J K L



Duyệt theo thứ tự trước

- Nếu cây không rỗng
  - Thăm gốc
  - Duyệt cây con trái theo thứ tự trước.
  - Duyệt cây con phải theo thứ tự trước.
- Ví dụ:

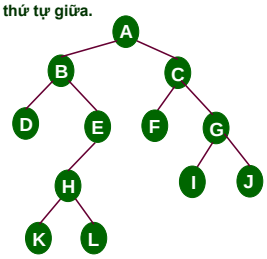
A B D E H K L C F G I J



Duyệt theo thứ tự giữa

- Nếu cây không rỗng
  - Duyệt cây con trái theo thứ tự giữa.
  - Thăm gốc.
  - Duyệt cây con phải theo thứ tự giữa.
- Ví dụ:

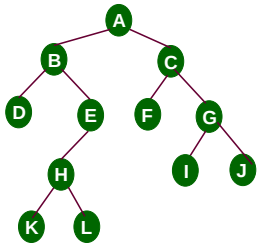
D B K H L E A F C I G J



Duyệt theo thứ tự sau

- Nếu cây không rỗng
  - Duyệt cây con trái theo thứ tự sau.
  - Duyệt cây con phải theo thứ tự sau.
  - Thăm gốc.
- Ví dụ:

D K L H E B F I J G C A



Cài đặt phép duyệt theo thứ tự sau

```
void preOrder (TRO Root)
{
   if (Root != NULL)
   {
        visit(Root);
        preOrder(Root->left);
        preOrder(Root->right);
   }
}
```


4.2.3. Ứng dụng cây nhị phân

- Một dạng cấu trúc lưu trữ được sử dụng trong thuật toán định giá biểu thức số học là cây nhị phân.
- Việc định giá biểu thức số học được thực hiện qua 3 bước:
  - Dựng cây nhị phân biểu diễn biểu thức.
  - Duyệt cây để được biểu thức dạng hậu tố.
  - Định giá biểu thức với ngăn xếp.

Định giá biểu thức số học

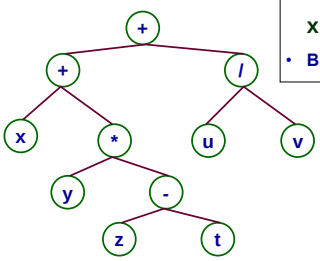
- Dựng cây nhị phân biểu diễn biểu thức
  - Gốc và các nút nhánh chứa các toán tử.
  - Lá cây chứa các toán hạng.
- Cách dựng:
  - Chọn toán tử có độ ưu tiên thấp nhất làm gốc.
  - Cây con trái là biểu thức bên trái toán tử được chọn, cây con phải là biểu thức bên phải.

Định giá biểu thức số học (tt)

x + y * (z - t) + u / v

- Duyệt cây theo thứ tự sau ta được:

  x y z t - * + u v / +
- Biểu thức hậu tố.



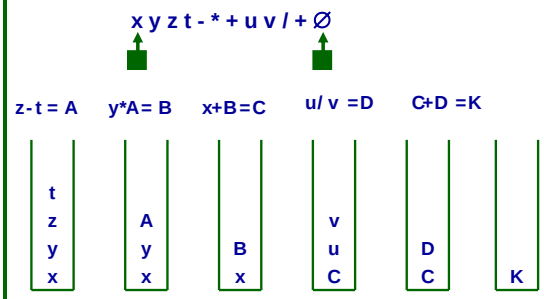
Định giá biểu thức số học (tt)

- Bài toán:
  - Đầu vào: Biểu thức dạng hậu tố, gọi là xâu vào.
  - Đầu ra: Giá trị biểu thức.
- Phương pháp lưu trữ:
  - Sử dụng ngăn xếp để lưu dữ liệu trong quá trình định giá biểu thức, kết quả trả về trong ngăn xếp.

Định giá biểu thức số học (tt) – Thuật toán

- **Bước 1:** Khởi tạo ngăn xếp rỗng, đầu đọc chỉ vào vị trí đầu tiên trên xâu vào
- **Bước 2:** Nếu đầu đọc chỉ vị trí kết thúc xâu vào sang bước 6, ngược lại sang bước 3
- **Bước 3:** Đọc 1 ký tự trên xâu vào, dịch chuyển đầu đọc sang phải, nếu là toán hạng sang bước 4, ngược lại sang bước 5
- **Bước 4:** Đẩy toán hạng vào ngăn xếp, rồi quay lại bước 2
- **Bước 5:** Lấy 2 toán hạng trong ngăn xếp, giả sử được lần lượt là A và B, thực hiện B T A (với T là toán tử vừa đọc được), được kết quả đẩy vào ngăn xếp và quay lại bước 2
- **Bước 6:** Nhận kết quả trong ngăn xếp, và kết thúc.

Định giá biểu thức số học (tt)

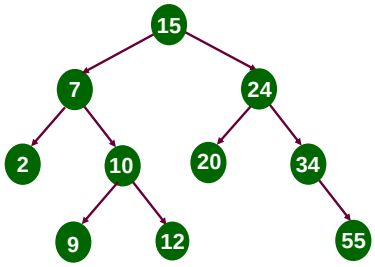


4.3. Cây nhị phân tìm kiếm

4.3.1. Định nghĩa

- Cây nhị phân tìm kiếm (CNPTK) là cây nhị phân hoặc rỗng hoặc không rỗng thì phải thoả mãn đồng thời các điều kiện sau:
  - ✓ Khoá của các nút thuộc cây con trái nhỏ hơn khoá của nút gốc
  - ✓ Khoá của nút gốc nhỏ hơn khoá của các nút thuộc cây con phải của nút gốc
  - ✓ Cây con trái và cây con phải của gốc cũng là cây nhị phân tìm kiếm

Định nghĩa (tt)



4.3.2. Cài đặt cấu trúc dữ liệu

- Mỗi nút trên cây nhị phân tìm kiếm có dạng.


```
//Khai báo kiểu dữ liệu Item
struct  Node {
   Item  infor;
   Node  *left, *right;
};
typedef Node *TRO; //Định nghĩa kiểu con trỏ
TRO  Root; //Khai báo con trỏ trỏ vào gốc cây
Root = NULL -> cây rỗng
```


4.3.3. Cài đặt các phép toán

4.3.3.1. Tìm kiếm một nút.

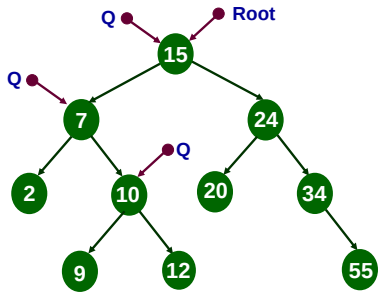
- Tìm kiếm một nút trên cây là một trong các phép toán quan trọng nhất đối với cây nhị phân tìm kiếm.
- ***Bài toán***:
  - ✓ **Đầu vào:** Cây nhị phân tìm kiếm có gốc được trỏ bởi con trỏ Root, khóa của nút cần tìm là K.
  - ✓ **Đầu ra:** Con trỏ trỏ vào nút tìm thấy, hoặc con trỏ NULL nếu không thấy.

Tìm kiếm một nút (tt)

- Nếu cây rỗng -> `return NULL`.
- Ngược lại so sánh khóa của nút gốc với khóa K.
  - Nếu bằng -> `return Root`.
  - Nếu lớn hơn -> tìm kiếm nút K ở cây con trái.
  - Nếu nhỏ hơn -> tìm kiếm nút K ở cây con phải.

Tìm kiếm một nút (tt)

- Tìm nút có khóa K = 10



Tìm kiếm một nút (tt)

- Hàm đệ quy

```
TRO search(TRO Root, KeyType K)
{
    if (Root ==NULL)
     return NULL;
    else if (Root->infor==K)
     return Root;
    else if (Root->infor>K)
     return search(Root->left,K);
    else return search(Root->right,K);
}
```


Tìm kiếm một nút (tt)

- Hàm lặp

```
TRO search(TRO Root, KeyType K)
{
    TRO Q;
    Q = Root;
    while (Q != NULL && Q->infor != K)
    {
     if (Q->infor > K)
          Q = Q->left;
     else Q = Q->right;
    }
    return Q;
}
```


4.3.3.2. Duyệt cây nhị phân tìm kiếm

- Duyệt cây theo thứ tự trước.
- Duyệt cây theo thứ tự giữa.
- Duyệt cây theo thứ tự sau.
- ***Lưu ý***:
  - Duyệt cây theo thứ tự giữa sẽ cho thứ tự các khóa trên cây theo thứ tự tăng dần.

4.3.3.3. Chèn một nút mới vào cây

- Chèn nút có khóa K vào cây như sau:
  - Nếu cây rỗng nút mới là gốc cây, return 1;
  - Ngược lại:
    - ✓ Nếu khóa gốc == K, nút đã có trên cây, return 0;
    - ✓ Nếu khóa gốc > K, chèn nút K vào cây con trái.
    - ✓ Nếu khóa gốc < K, chèn nút k vào cây con phải.

Chèn một nút mới vào cây (tt)

- **Hàm đệ quy**

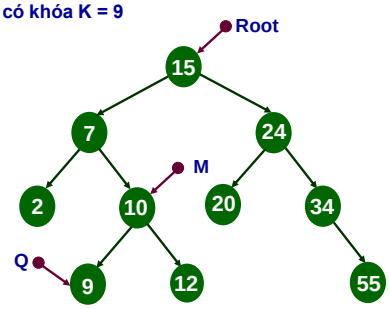
```
int insert(TRO &Root, KeyType K) {
   if (Root == NULL) {
      Root = new Node;
      Root->infor = K;
      Root->left = Root->right = NULL;
      return 1;
   }
   else if (Root->infor == K)
      return 0;
   else if (Root->infor > K)
      return insert(Root->left, K);
   else return insert(Root->right, K);
}
```


4.3.3.4. Loại bỏ một nút trên cây

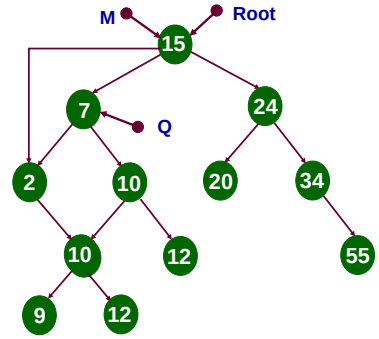
- **Trường hợp 1: Nút loại bỏ không phải là gốc.**
  - Tìm cha của nút này, và cho biết nó là con trái hay con phải của cha.
  - Nếu là nút lá.
    - Hủy nó đi.
    - Gán con trái/phải của cha nó bằng NULL.
  - Nếu là nút nhánh
    - Ghép cây con phải của nó vào bên phải nhất của cây con trái của nó (hoặc ngược lại).
    - Gán con trái/phải của cha nó bằng con trái của nó.
    - Hủy nó đi.

Loại bỏ một nút trên cây (tt)

Loại bỏ nút có khóa K = 9



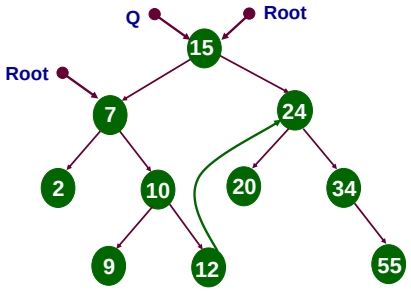
Loại bỏ một nút trên cây (tt)



Loại bỏ một nút trên cây (tt)

- **Trường hợp 2: Nút loại bỏ là gốc**
  - Ghép cây con phải của nó vào bên phải nhất của cây. con trái của nó (hoặc ngược lại).
  - Gán nó bằng con trái của nó (hoặc con phải).
  - Hủy nó đi.

Loại bỏ một nút trên cây (tt)



Ghép hai cây thành một cây

```
void Ghep(TRO &R1, TRO R2)
{
    if (R1 == NULL)
          R1 = R2;
    else
          Ghep(R1->right, R2);
}
```


Tìm cha của một nút

```
TRO Cha(TRO Root, int &u, TRO Q) {
    TRO M = Root;
    while (1) {
     if (Q->infor < M->infor)
          if (M->left == Q){
               u = -1;
               return M;
          }
          else   M = M->left;
     else if (M->right == Q){
          u = 1;
          return M;
     }
     else M = M->right;
    }
}
```


Loại bỏ một nút

```
void Xoa(TRO &Root, TRO Q) {
    int u;  TRO M;
    Ghep(Q->left, Q->right);
    if (Q == Root)
          Root=Root->left;
    else{
     M = Cha(Root, u, Q);
     if (u == -1)
          M->left = Q->left;
     else M->right = Q->left;
    }
    delete Q;
}
```


Bài tập

- **Cho dãy số: 15, 7, 24, 2, 10, 20, 34, 9, 12, 55**
- **Viết chương trình:**
  - ✓ Dựng cây nhị phân tìm kiếm với khóa của các nút lần lượt là các số trong dãy trên.
  - ✓ Hiển thị dãy khóa trên cây theo thứ tự trước.
  - ✓ Thêm nút có khóa là 28 vào cây, hiển thị lại cây theo thứ tự trước.
  - ✓ Nhập vào giá trị khóa K, cho biết nút có khóa K có trên cây không, nếu có xóa nó đi khỏi cây.