Cấu trúc dữ liệu và thuật giải

Tìm kiếm

Tìm kiếm

- **Tìm kiếm tuần tự**
  - **Thời gian tồi nhất là n**
  - **Chúng tôi có Độ phức tạp $O(n)$**
  - **Ứng dụng cho mảng (không sắp xếp) và danh sách liên kết**
- **Tìm kiếm nhị phân**
  - **Ứng dụng cho dãy đã sắp xếp**
  - **Thời gian thực hiện thuật toán $\log_2 n$**
  - **Độ phức tạp tính toán $O(\log n)$**

Tìm kiếm – Tìm kiếm nhị phân

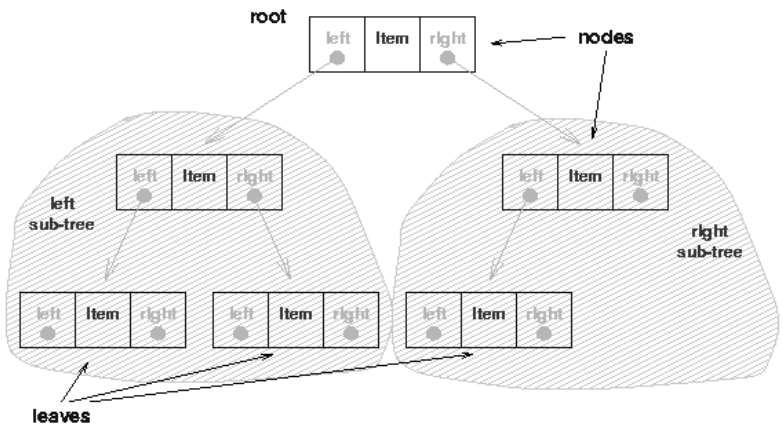
- Tạo ra một dãy đã sắp xếp
  - `Bổ xung vào danh sách liên kết (AddToCollection)`
    - Bổ xung giá trị vào các vị trí đúng
    - Tìm vị trí $c_1 \log_2 n$
    - Duyệt xuống $c_2 n$
    - Tổng quát $c_1 \log_2 n + c_2 n$ — Dominant term

      Hoặc $c_2 n$
  - Mỗi hoạt động gắn với dãy đã sắp xếp $O(n)$

? Chúng tôi có thể duy trì một dãy đã sắp xếp với các hoạt động chèn dẻ hơn?

Cây

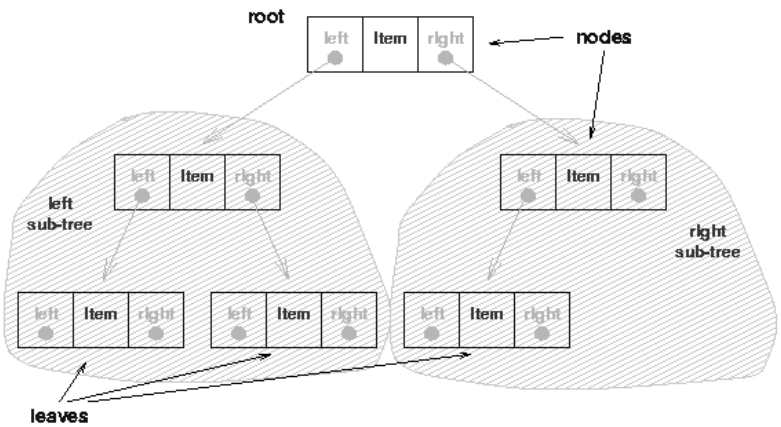
- **Cây nhị phân**
  - **Bao gồm**
    - **Node**
    - **Các cây con trái và cây con phải**
    - **Cả hai cây con đều là cây nhị phân**



Cây

- **Cây nhị phân**
  - **Bao gồm**
    - **Node**
    - **Các cây con trái và con phải**
    - **Cả hai đều là cây nhị phân**

Chú ý
Định nghĩa
đệ qui!

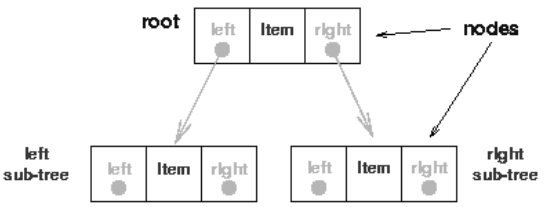


Mỗi cây con
Là một cây
Nhị phân

Cây – Thủ tục

- **Cấu trúc dữ liệu**

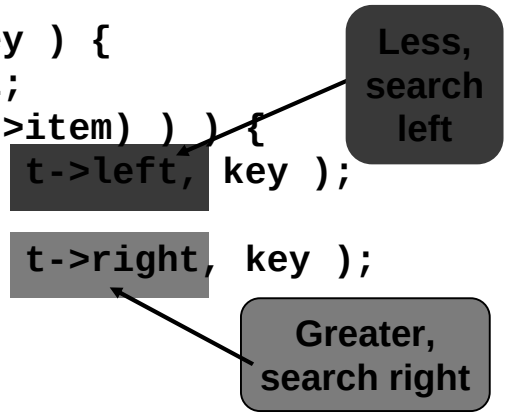
```
struct t_node {
     void *item;
     struct t_node *left;
     struct t_node *right;
     };

typedef struct t_node *Node;

struct t_collection {
     Node root;
     ……
     };
```



Cây – Thủ tục

- **Tìm (Find)**

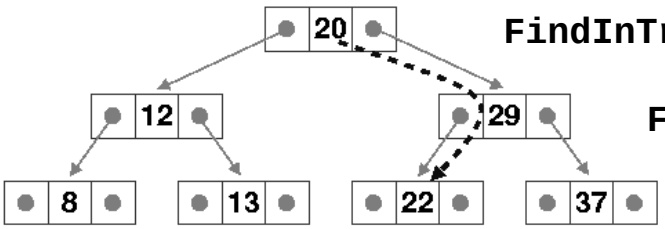
```
extern int KeyCmp( void *a, void *b );
/* Returns -1, 0, 1 for a < b, a == b, a > b */

void *FindInTree( Node t, void *key ) {
   if ( t == (Node)0 ) return NULL;
   switch( KeyCmp( key, ItemKey(t->item) ) ) {
      case -1 : return FindInTree( t->left, key );
      case 0:   return t->item;
      case +1 : return FindInTree( t->right, key );
      }
   }

void *FindInCollection( collection c, void *key ) {
   return FindInTree( c->root, key );
   }
```



Cây – Thủ tục

- **Find**
  - `key = 22;`
    `if ( FindInCollection( c , &key ) ) ….`

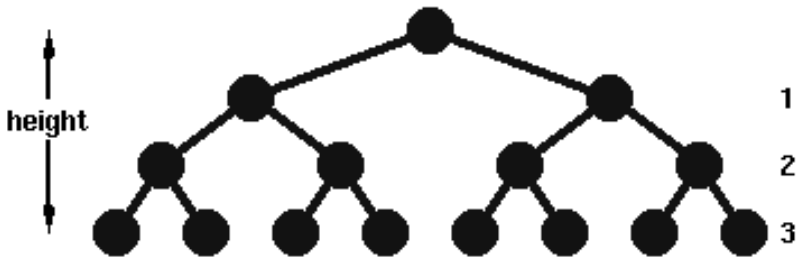
```
n = c->root;
FindInTree( n, &key );
```




```
FindInTree(n->right,&key );

FindInTree(n->left,&key );

return n->item;
```

Cây – hoạt động

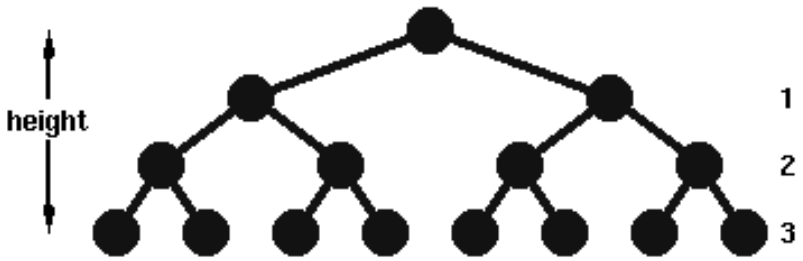
- **Find**
  - **Cây hoàn toàn**



  - **Chiều cao, h**
    - **Các node di truyển trên một con đường từ gốc đến một lá**
  - **Số các node, h**
    - $n = 1 + 2^1 + 2^2 + \ldots + 2^h = 2^{h+1} - 1$
    - $h = \text{floor}(\log_2 n)$

Cây – Hoạt động

- **Find**
  - **Cây hoàn toàn**



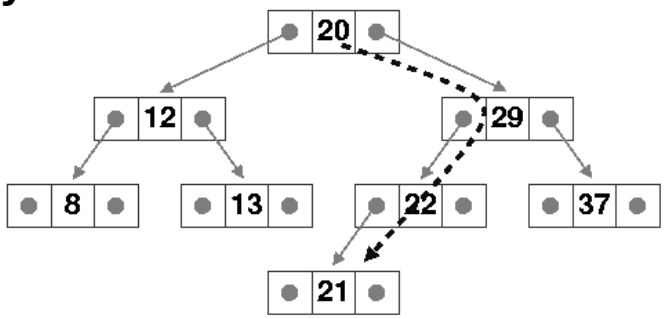
  - **Vì chúng tôi cần nhiều nhất $h+1$ phép so sánh, độ phức tạp $O(h+1)$ hoặc $O(\log n)$**
  - **Giống như tìm kiếm nhị phân**

Tổng kết

	Arrays Đơn giản, nhanh Không mềm dẻo	Linked List Đơn giản Mềm dẻo	Trees Vẫn đơn giản Mềm dẻo
Add	O(1) O(n) *inc sort*	O(1) *sort -> no adv*	
Delete	O(n)	O(1) - *any* O(n) - *specific*	
Find	O(n) O(logn) *Tìm kiếm nhị phân*	O(n) *(không tìm kiếm nhị phân)*	O(log n)

Cây – Bổ xung (Addition)

- **Cộng 21 vào cây**



- **Chúng tôi cần tối đa $h+1$ phép so sánh**
- **Tạo một node mới (thời gian là hằng số)**

∴ add lấy $c_1(h+1)+c_2$ hoặc $c \log n$

Ignoring low order terms

- **Vì thế việc bổ xung vào một cây m thời gian là $\log n$**

Cây - Addition – thủ tục

```
static void AddToTree( Node *t, Node new ) {
   Node base = *t;
   /* If it's a null tree, just add it here */
   if ( base == NULL ) {
      *t = new; return; }
   else
      if( KeyLess(ItemKey(new->item),ItemKey(base->item)) )
         AddToTree( &(base->left), new );
      else
         AddToTree( &(base->right), new );
   }

void AddToCollection( collection c, void *item ) {
        Node new, node_p;
        new = (Node)malloc(sizeof(struct t_node));
        /* Attach the item to the node */
        new->item = item;
        new->left = new->right = (Node)0;
        AddToTree( &(c->node), new );
        }
```

Cây – Bổ xung (Addition)

- **Find** $c \log n$
- **Add** $c \log n$
- **Delete** $c \log n$

- **Hiệu quả rất đáng kể!**

- ***Tuy nhiên vẫn bắt gặp một số trường hợp .....***

Cây – Bổ xung (Addition)

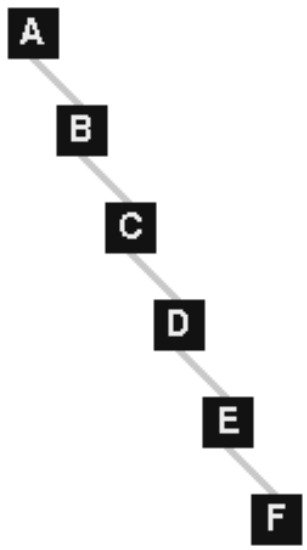
- **Lấy danh sách ký tự này và hình thành một cây**

A B C D E F

- ***??***

Trees - Addition

- **Nhận danh sách ký tự này và hình thành một cây A B C D E F**



- **Trong trường hợp này:**
  - ? **Find**
  - ? **Add**
  - ? **Delete**