

Cấu trúc dữ liệu và thuật toán

PGS. TS. Phạm Tuấn Minh

Trường Công nghệ Thông tin, Đại học Phenikaa

minh.phamtuan@phenikaa-uni.edu.vn

<https://sites.google.com/site/phamtuanminh/>

Chương 3: Cây và bảng băm

- ❑ Các khái niệm cây
- ❑ Cây tìm kiếm nhị phân
- ❑ Cây AVL
- ❑ Bảng băm

Cây tìm kiếm nhị phân

- ❑ **Tìm kiếm một phần tử**
- ❑ Cây tìm kiếm nhị phân
- ❑ Thao tác trên cây tìm kiếm nhị phân
 - Duyệt
 - Chèn nút
 - Xóa nút

1-3

Tìm kiếm phần tử trên danh sách liên kết

- ❑ Cho một danh sách liên kết các tên, kiểm tra xem một tên có trong danh sách không?



cur = cur->next;

```
while (cur!=NULL) {  
    if cur->item == "Irit"  
        found and stop searching;  
    else  
        cur = cur->next;  
}
```

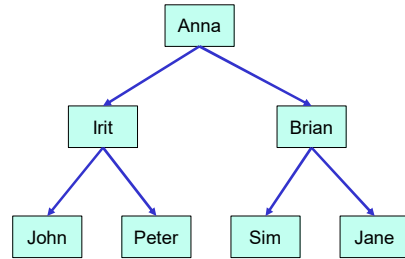
- ❑ Số nút phải duyệt qua khi thực hiện tìm kiếm:
 - Tốt nhất: 1 nút (John)
 - Tồi nhất: 7 nút (Sim)
 - Trung bình: $(1+2+\dots+7)/7 = 4$ nút
- ❑ **Không hiệu quả!**

Tìm kiếm phần tử trên cây nhị phân tìm kiếm

- ❑ Cho một cây nhị phân các tên, kiểm tra xem một tên có trong cây không?

Sử dụng **TreeTraversal (Pre-order)** để kiểm tra mọi nút

```
TreeTraversal(Node N)
  If N==NULL return;
  if N.item==given_name return;
  TreeTraversal(LeftChild);
  TreeTraversal(RightChild);
  Return;
```



- ❑ Số nút phải duyệt qua khi thực hiện tìm kiếm:
 - Tốt nhất: 1 nút (Anna)
 - Tồi nhất: 7 nút (Jane)
 - Trung bình: $(1+2+...+7)/7 = 4$ nút
- ❑ **Không hiệu quả!**

Có cách tổ chức dữ liệu để tìm kiếm nhanh hơn?

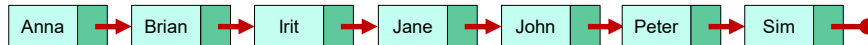
- ❑ Có cách nào để sắp xếp dữ liệu trên cây để có thể lưu trữ và xác định phần tử hiệu quả?
- ❑ Làm thế nào để tìm nhanh một phần tử trên cây nhị phân?



Có cách tổ chức dữ liệu để tìm kiếm nhanh hơn?

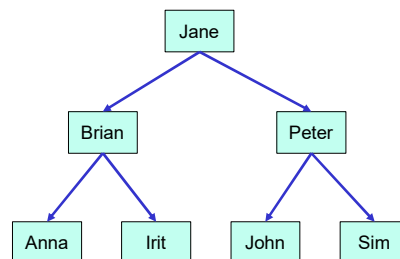


- ❑ Đối với danh sách: Sắp xếp theo thứ tự alphabet
- ❑ Chia thành nhóm
- ❑ Chọn một phần tử ở giữa, "Jane", nếu $X \neq \text{"Jane"}$
 - Với tên trước "Jane", tìm bên nhóm trái, bỏ qua nhóm phải
 - Chọn một phần tử ở giữa trong nhóm con, "Brian", nếu $X \neq \text{"Brian"}$:
 - Với tên trước "Brian", tìm bên nhóm trái
 - Với tên sau "Brian", tìm bên nhóm phải
 - Với tên sau "Jane", tìm bên nhóm phải, bỏ qua nhóm trái
 - Chọn một phần tử ở giữa trong nhóm con, "Peter", nếu $X \neq \text{"Peter"}$:
 - Với tên trước "Peter", tìm bên nhóm trái
 - Với tên sau "Peter", tìm bên nhóm phải



Có cách tổ chức dữ liệu để tìm kiếm nhanh hơn?

- ❑ Đối với danh sách: Sắp xếp theo thứ tự alphabet
- ❑ Chia thành nhóm
- ❑ Chọn một phần tử ở giữa, "Jane", nếu $X \neq \text{"Jane"}$
 - Với tên trước "Jane", tìm bên nhóm trái, bỏ qua nhóm phải
 - Chọn một phần tử ở giữa trong nhóm con, "Brian", nếu $X \neq \text{"Brian"}$:
 - Với tên trước "Brian", tìm bên nhóm trái
 - Với tên sau "Brian", tìm bên nhóm phải
 - Với tên sau "Jane", tìm bên nhóm phải, bỏ qua nhóm trái
 - Chọn một phần tử ở giữa trong nhóm con, "Peter", nếu $X \neq \text{"Peter"}$:
 - Với tên trước "Peter", tìm bên nhóm trái
 - Với tên sau "Peter", tìm bên nhóm phải
- ❑ **Tạo thành một cây tìm kiếm nhị phân (BST - Binary Search Tree)**



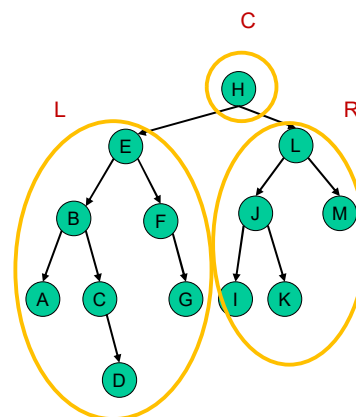
Cây tìm kiếm nhị phân

- Tìm kiếm một phần tử
- **Cây tìm kiếm nhị phân**
- Thao tác trên cây tìm kiếm nhị phân
 - Duyệt
 - Chèn nút
 - Xóa nút

1-9

Cây tìm kiếm nhị phân

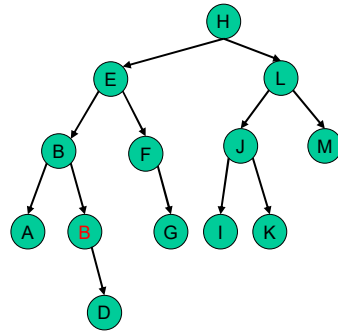
- Cây tìm kiếm nhị phân (BST - Binary Search Tree) là một dạng đặc biệt của cây nhị phân (BT)
- **Luật trên BST:** Tại mỗi nút **C**, $L < C < R$, trong đó
 - **C** là dữ liệu tại nút hiện tại
 - **L** là dữ liệu của **bất kì** nút nào từ cây con trái của C
 - **R** là dữ liệu của **bất kì** nút nào từ cây con phải của C



1-10

Cây tìm kiếm nhị phân

- ❑ Cây tìm kiếm nhị phân (BST - Binary Search Tree) là một dạng đặc biệt của cây nhị phân (BT)
- ❑ **Luật trên BST:** Tại mỗi nút C , $L < C < R$, trong đó
 - C là dữ liệu tại nút hiện tại
 - L là dữ liệu của bất kì nút nào từ cây con trái của C
 - R là dữ liệu của bất kì nút nào từ cây con phải của C
- ❑ Không được phép có dấu bằng "=" trên BST! Không được phép có nút lặp trên BST!

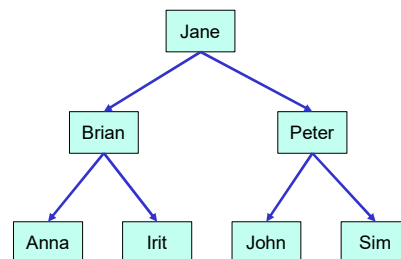


Đây không phải là BST

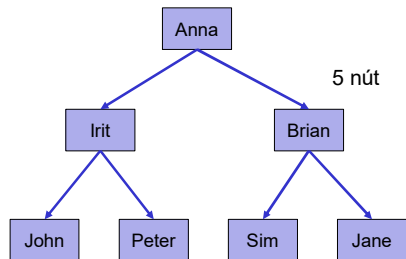
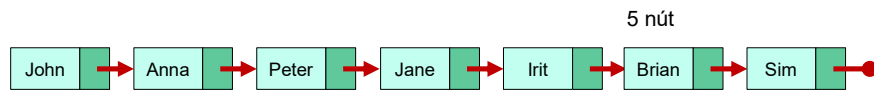
1-11

Việc tìm kiếm phần tử sẽ hiệu quả với BST

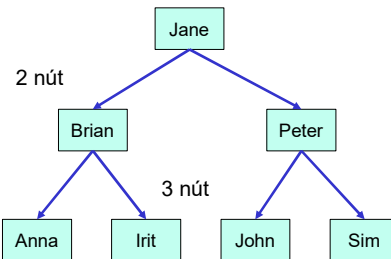
- ❑ Đây là BST, thỏa mãn $L < C < R$



Việc tìm kiếm phần tử sẽ hiệu quả với BST

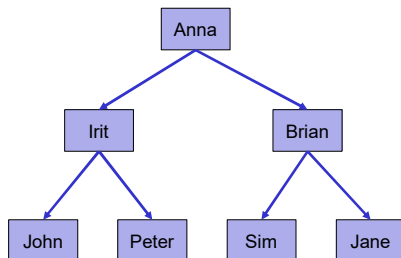


- ❑ Cần thăm bao nhiêu nút khi tìm kiếm?
 - Tốt nhất: 1 node (Anna)
 - Tồi nhất: 7 nút (Jane)
 - Trung bình: $(1+2+\dots+7)/7 = 4$ nút



- ❑ Cần thăm bao nhiêu nút khi tìm kiếm?
 - Tốt nhất: 1 node (Jane)
 - Tồi nhất: 3 nút (Anna)
 - Trung bình: $(1+2*2+3*4)/7 = 2,43$ nút

Việc tìm kiếm phần tử sẽ hiệu quả với BST

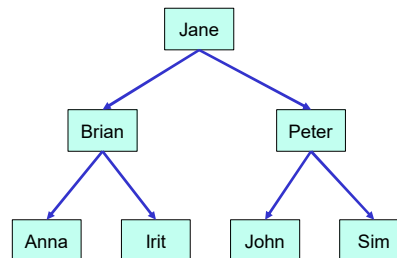


Không hiệu quả

- ❑ Cần thăm bao nhiêu nút khi tìm kiếm?

Tổng quát, với cây nhị phân **n nút**

 - Tốt nhất: Nút đầu tiên khi duyệt cây
 - Tồi nhất: **Nút cuối cùng** khi duyệt cây, **n**



Hiệu quả

- ❑ Cần thăm bao nhiêu nút khi tìm kiếm?

Tổng quát với BST n nút

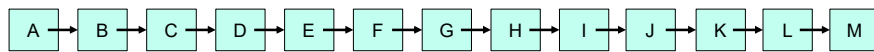
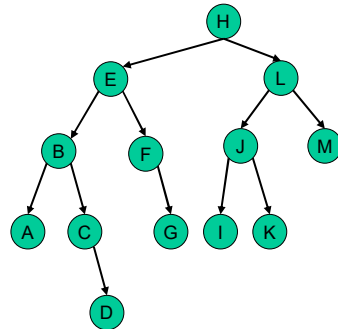
 - Tốt nhất: Nút đầu tiên khi duyệt cây
 - Tồi nhất: **nút lá, $h = \text{chiều cao của nút gốc}$. Giá trị nhỏ nhất $h = \log_2(n + 1) - 1$**

Khi n lớn, sự khác biệt về hiệu quả càng lớn

Ví dụ tìm kiếm trên BST

- Để tìm D, cần thăm H-E-B-C-D: 5 nút
- Để tìm bất kì nút nào trên cây, cần thăm nhiều nhất 5 nút

Hiệu quả

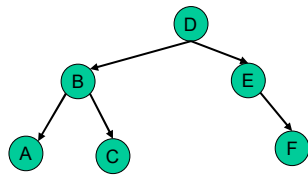


- Với danh sách liên kết, trường hợp tồi nhất là cần thăm tất cả các nút 13 nút

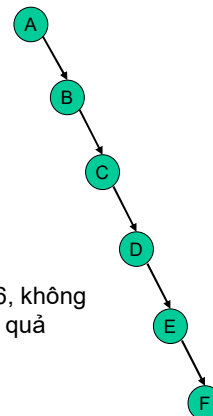
Không hiệu quả

1-15

Chú ý: Việc tìm kiếm là hiệu quả không phải với tất cả BST



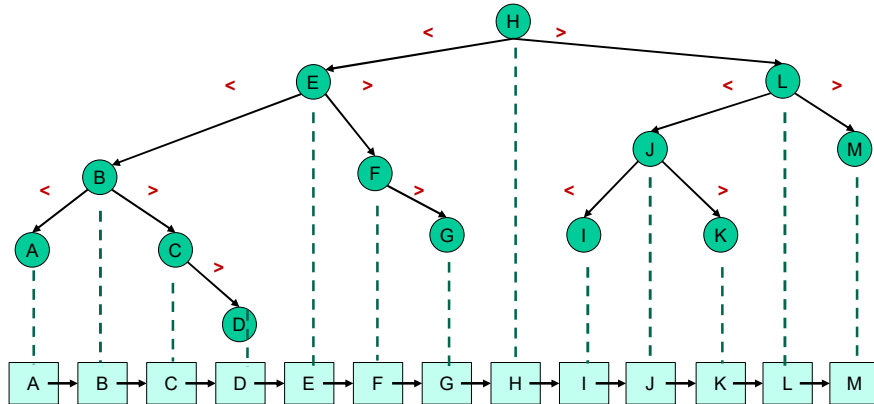
- Số nút tối đa cần kiểm tra: $h = 3$



- $h = 6$, không hiệu quả

1-16

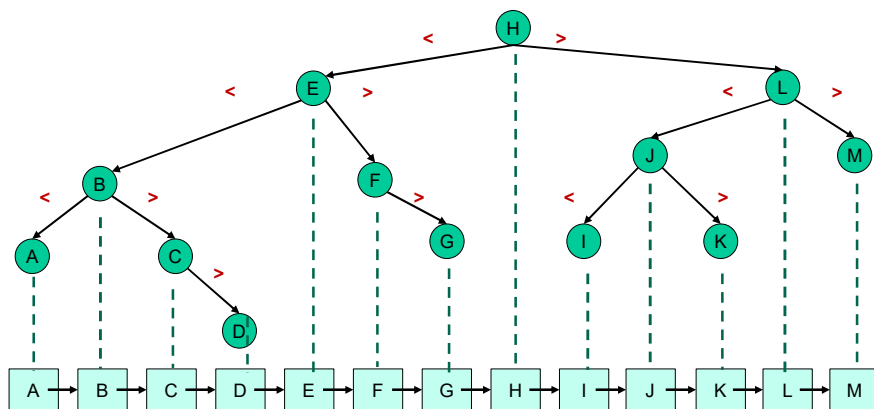
Ánh xạ: BST (thứ tự giữa) thành danh sách



- Nếu vẽ BST:
 - Cây con trái ở bên trái của nút hiện tại
 - Cây con phải ở bên phải của nút hiện tại
- Ánh xạ xuống trục x sẽ thu được danh sách sắp thứ tự

1-17

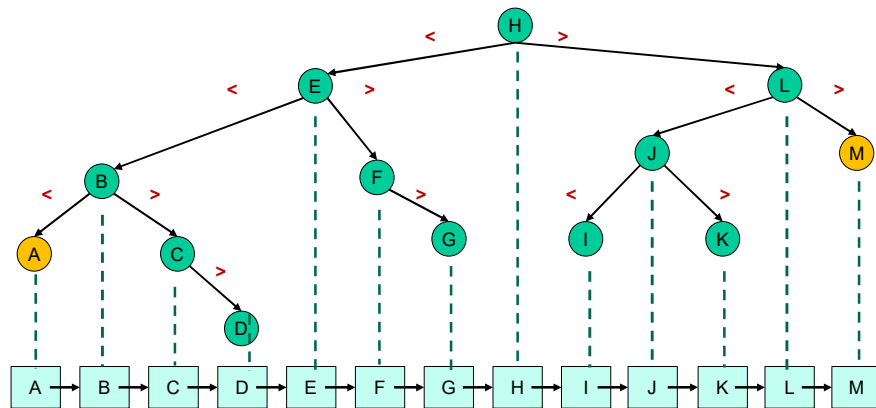
BST và duyệt thứ tự giữa



- Luật **L < C < R** đảm bảo thứ tự sắp xếp
- Duyệt thứ tự giữa trên BST cho kết quả một danh sách được sắp thứ tự

1-18

Đặc điểm của BST



- Cây tìm kiếm nhị phân đảm bảo rằng
 - Giá trị nhỏ nhất nằm ở nút trái nhất
 - Giá trị lớn nhất nằm ở nút phải nhất

1-19

Cây tìm kiếm nhị phân

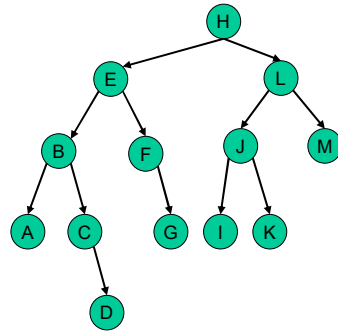
- Tìm kiếm một phần tử
- Cây tìm kiếm nhị phân
- **Thao tác trên cây tìm kiếm nhị phân**
 - **Duyệt**
 - Chèn nút
 - Xóa nút

1-20

Duyệt BST

- Duyệt BST (BSTT - BST Traversal) thăm BST để tìm kiếm một nút có giá trị nào đó
- Bắt đầu với mẫu TreeTraversal

```
void BSTT(btnode *cur) {  
    if (cur == NULL)  
        return;  
    // printf(cur->item);  
    BSTT(cur->left);  
    BSTT(cur->right);  
}
```

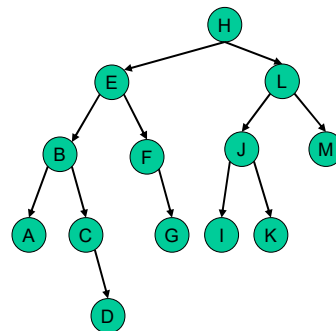


1-21

Duyệt BST

- Duyệt BST (BSTT - BST Traversal) thăm BST để tìm kiếm một nút có giá trị nào đó
- Bắt đầu với mẫu TreeTraversal

```
void BSTT(btnode *cur, char c) {  
    if (cur == NULL)  
        return;  
    if (c==cur->item) { printf("found!\n"); return;}  
    if (c < cur->item)  
        BSTT(cur->left,c);  
    else  
        BSTT(cur->right,c);  
}
```

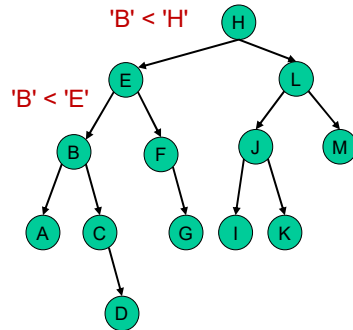


1-22

Duyệt BST

□ BSTT(root, 'B')

```
void BSTT(btnode *cur, char c) {  
    if (cur == NULL)  
        return;  
    if (c==cur->item) { printf("found!\n"); return;}  
    if (c < cur->item)  
        BSTT(cur->left,c);  
    else  
        BSTT(cur->right,c);  
}
```

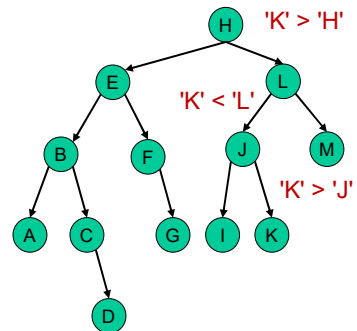


1-23

Duyệt BST

□ BSTT(root, 'K')

```
void BSTT(btnode *cur, char c) {  
    if (cur == NULL)  
        return;  
    if (c==cur->item) { printf("found!\n"); return;}  
    if (c < cur->item)  
        BSTT(cur->left,c);  
    else  
        BSTT(cur->right,c);  
}
```

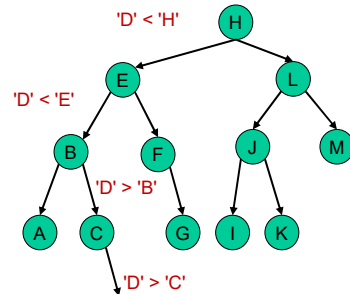


1-24

Duyệt BST

- ❑ Nếu phần tử cần tìm không có trên cây thì sao?
- ❑ Ví dụ bỏ nút 'D', và gọi BSTT(root, 'D')

```
void BSTT(btnode *cur, char c) {  
    if (cur == NULL)  
        { printf("can't find!"); return; }  
    if (c==cur->item) { printf("found!\n"); return;}  
    if (c < cur->item)  
        BSTT(cur->left,c);  
    else  
        BSTT(cur->right,c);  
}
```



1-25

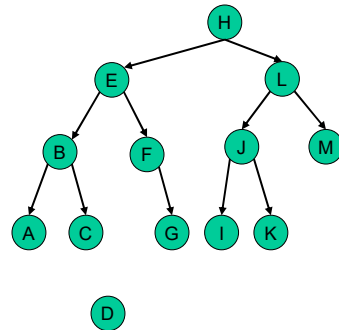
Cây nhị phân tìm kiếm

- ❑ Tìm kiếm một phần tử
- ❑ Cây nhị phân tìm kiếm
- ❑ **Thao tác trên cây nhị phân tìm kiếm**
 - Duyệt
 - **Chèn nút**
 - Xóa nút

1-26

Chèn một nút vào BST

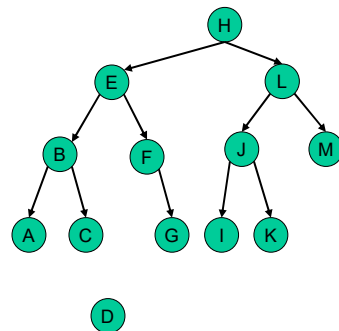
- Cho BST, thao tác chèn phải có kết quả là BST
- Làm sao biết vị trí đặt nút 'D'?



1-27

Chèn một nút vào BST

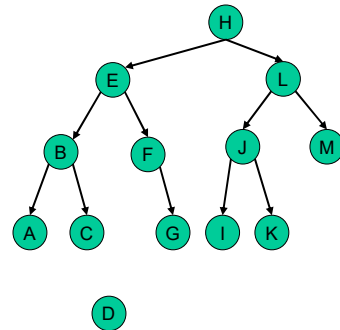
- Điểm then chốt: Cho BST và một giá trị cần chèn, có một vị trí duy nhất cho giá trị mới này trên BST



1-28

Chèn một nút vào BST

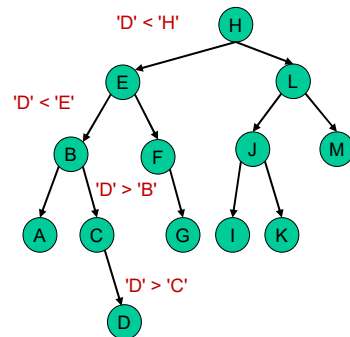
- ❑ Sử dụng BSTT() để tìm vị trí trống
- ❑ Chèn nút mới vào vị trí trống này



1-29

Chèn một nút vào BST

- ❑ Sử dụng BSTT(root, 'D') để **xác định vị trí trống sẽ chèn 'D'**
- ❑ **Chèn nút mới 'D'**

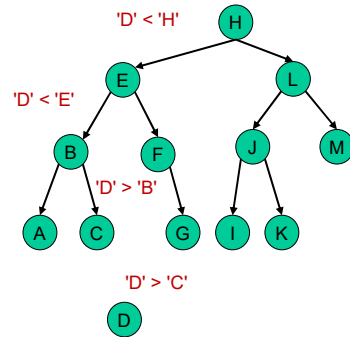


1-30

Xác định vị trí sẽ chèn

- Sử dụng BSTT(root, 'D') để xác định vị trí trống sẽ chèn 'D'

```
BTNode* BSTT2(btNode *cur, char c) {
    if (cur == NULL)
        { printf("can't find!"); return; }
    if (c==cur->item) { printf("found!\n"); return NULL;}
    if (c < cur->item) {
        if (cur->left == NULL)
            return cur;
        return BSTT2(cur->left,c);
    } else {
        if (cur->right == NULL)
            return cur;
        return BSTT2(cur->right,c);
    }
}
```



1-31

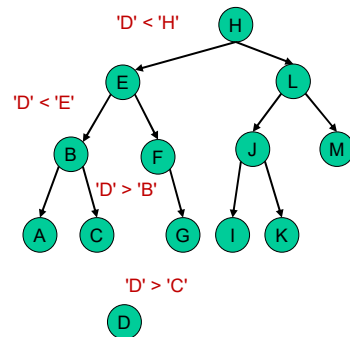
Chèn nút mới

- Sử dụng BSTT(root, 'D') để xác định vị trí trống sẽ chèn 'D'
- Chèn nút mới 'D'**

```
BTNode* posNode = BSTT2(&btNodeH,c);

BTNode *btNewNode = malloc(sizeof(BTNode));
btNewNode->item = c;
btNewNode->left = NULL;
btNewNode->right = NULL;

if (posNode == NULL) {
    printf("Phan tu da ton tai");
    return 0;
}
if (c < posNode->item)
    posNode->left = btNewNode;
else posNode->right = btNewNode;
```



1-32

Cây nhị phân tìm kiếm

- ❑ Tìm kiếm một phần tử
- ❑ Cây nhị phân tìm kiếm
- ❑ **Thao tác trên cây nhị phân tìm kiếm**
 - Duyệt
 - Chèn nút
 - **Xóa nút**

1-33

Xóa nút

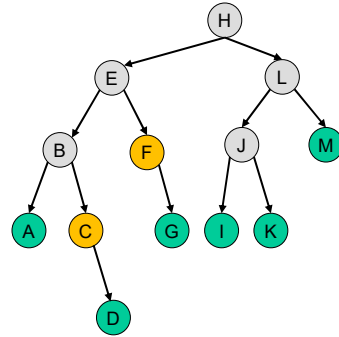
- ❑ Xóa nút phức tạp hơn chèn nút
- ❑ Kết quả BST sau khi xóa nút vẫn phải là BST
- ❑ Tuân theo luật: $L < C < R$

1-34

Xóa nút

□ Xóa một nút x, có ba trường hợp

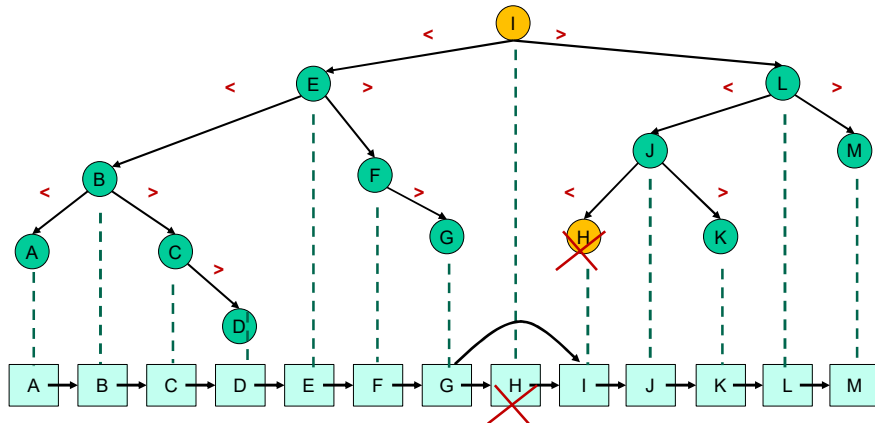
- 1) x không có nút con:
 - Xóa x
- 2) x chỉ có một con y:
 - Thay x bởi y
- 3) x có hai con:
 - Hoán đổi x với **nút sau** nó, rồi thực hiện 1) hoặc 2)



1-35

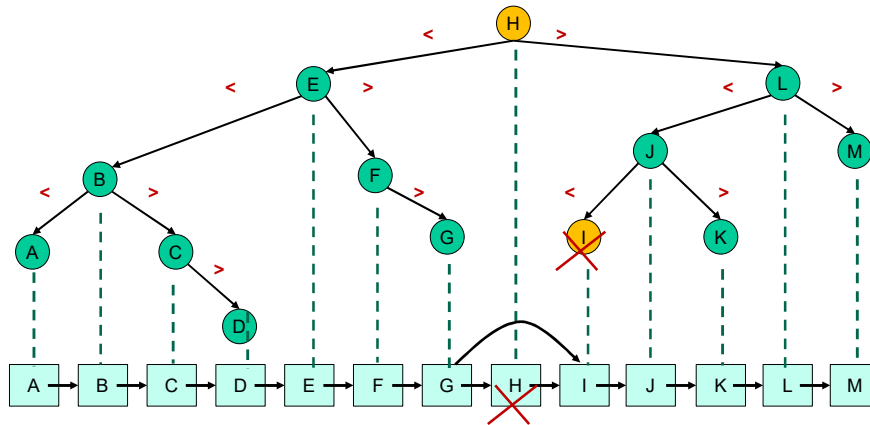
Xóa nút

- Thay thế một nút bởi nút sau nó (duyệt theo thứ tự giữa) đảm bảo duy trì luật BST ($L < C < R$)
- Duyệt theo thứ tự giữa cho kết quả là một danh sách sắp xếp theo thứ tự tăng dần.
- **Nút sau** là nút ngay sau trong danh sách đã sắp xếp, hay là nút tiếp theo sẽ được thăm khi duyệt theo thứ tự giữa
- X có hai con, vì vậy nút sau của X là nút có giá trị nhỏ nhất trên cây con phải của X
- Ví dụ: Nút sau của H là I, nút sau của E là F, nút sau của J là K



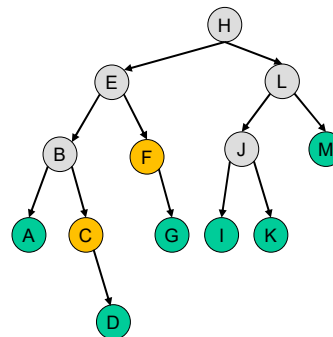
Xóa nút

- ❑ Xóa nút H



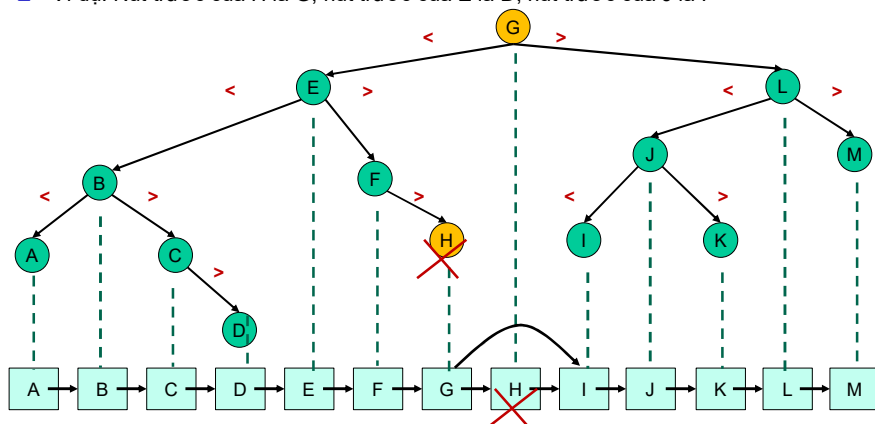
Câu hỏi

- ❑ Tại sao trường hợp 3 luôn dẫn tới trường hợp 1 hoặc trường hợp 2
 - Khi x có 2 con, nút sau của nó là nút có giá trị nhỏ nhất trên cây con phải. Vì vậy, nút sau không có nút con trái => Nút sau có thể không có nút con (trường hợp 1) hoặc có một nút con phải (trường hợp 2)
- ❑ Có thể hoán đổi x với nút trước thay vì nút sau
 - Được

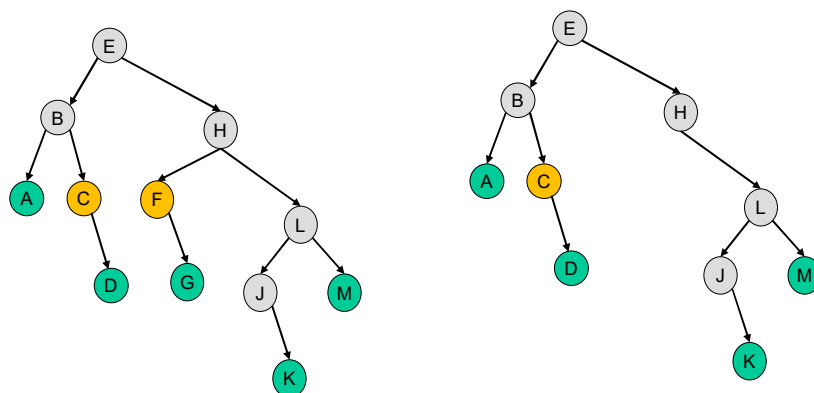


Xóa nút

- Thay thế một nút bởi **nút trước** nó (duyệt theo thứ tự giữa) đảm bảo duy trì luật BST ($L < C < R$)
- Duyệt theo thứ tự giữa cho kết quả là một danh sách sắp xếp theo thứ tự tăng dần.
- Nút sau/nút trước** là nút ngay **sau/trước** trong danh sách đã sắp xếp, hay là **nút tiếp theo sẽ được thăm/ nút ngay trước đã thăm** khi duyệt theo thứ tự giữa
- X có hai con, vì vậy nút trước của X là nút có giá trị lớn nhất trên cây con trái của X
- Ví dụ: Nút trước của H là G, nút trước của E là D, nút trước của J là I



Xóa nút



Tóm tắt

- ❑ Tìm kiếm một phần tử
- ❑ Cây nhị phân tìm kiếm
- ❑ **Thao tác trên cây nhị phân tìm kiếm**
 - Duyệt
 - Chèn nút
 - **Xóa nút**

1-41

Cấu trúc dữ liệu và giải thuật

- ❑ Nội dung bài giảng được biên soạn bởi PGS. TS. Phạm Tuấn Minh.

1-42