

# Bài 13. Tìm kiếm- Search

## Bài toán

**Input:** Cho một tập  $S$  các phần tử, mỗi phần tử là một bộ gồm khóa-giá trị (key-value) và một khóa  $k$  bất kỳ.

**Output:** Trong  $S$  có phần tử có khóa  $k$  hay không?

## Các phương pháp tìm kiếm

- ◆ Tìm kiếm tuần tự (Sequence search)
- ◆ Tìm kiếm nhị phân (Binary search)
- ◆ Bảng băm (Hash table)

# 1. Tìm kiếm tuần tự

◆ Tập **S** các phần tử được lưu bằng mảng hoặc danh sách liên kết.

◆ Thuật toán tìm kiếm:

- Xuất phát từ phần tử đầu của dãy, thực hiện so sánh khóa của nó với **k**. Nếu trùng nhau thì dừng lại, nếu không trùng thì lặp lại với phần tử tiếp theo.

- Quá trình dừng lại khi tìm thấy hoặc không còn phần tử nào nữa => Khi đó thông báo không tìm thấy.

Search

3

# 1. Tìm kiếm tuần tự

## Ví dụ 1:

◆ Cho dãy S: (0) (45) (3) (34) (13) (7) (43) (9) (11)

◆ Tìm xem trong dãy có phần tử  $k=33$

◆ Quá trình tìm kiếm



- Không tìm thấy

Search

4

# 1. Tìm kiếm tuần tự

## Ví dụ 2:

- ◆ Cho dãy S: 0 45 3 34 13 7 43 9 11
- ◆ Tìm xem trong dãy có phần tử  $k=13$
- ◆ Quá trình tìm kiếm



- Tìm thấy, tại vị trí 5

# 1. Tìm kiếm tuần tự

## Thuật toán

**Input:** Cho một dãy  $S$  các phần tử, mỗi phần tử là một bộ *key* và *value*. Một khóa  $k$  bất kỳ.

**Output:** Trong  $S$  có phần tử có khóa  $k$  hay không?

found = 0;    i = 1;

**while** ((chưa duyệt hết S) && (found == 0)) {

**if** (S[i].key == k)

        found = 1;

    i = i + 1; // Chuyển sang phần tử kế tiếp

}

**return** found;

# 1. Tìm kiếm tuần tự

## Thời gian chạy:

- ❖ Trong trường hợp xấu nhất thuật toán phải duyệt qua tất cả các phần tử của S.
- ❖ Vậy thời gian chạy là  $O(n)$

# 2. Tìm kiếm nhị phân

- ❖ Tập S được tổ chức lưu trữ dựa trên **mảng**
- ❖ Tập S được tổ chức lưu trữ dạng **cây nhị phân**

## 2.1 Tìm kiếm nhị phân trên mảng

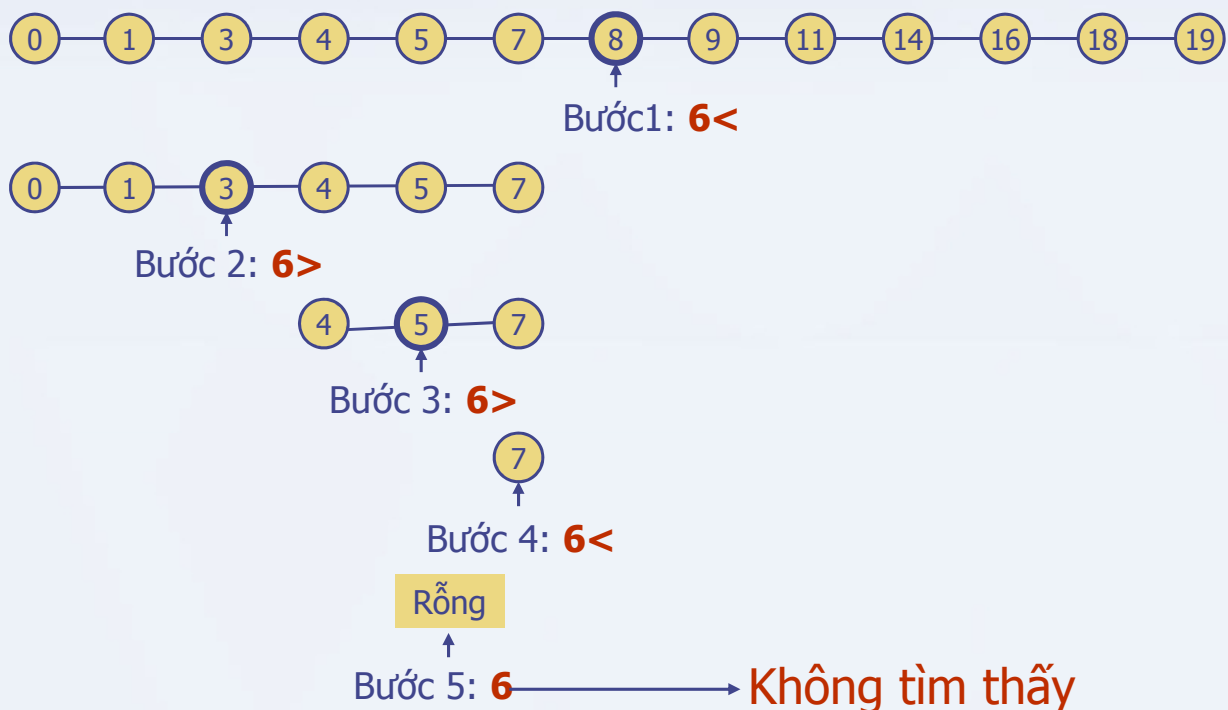
- ◆ Các phần tử của **S** được lưu trữ trong mảng và được sắp xếp theo thứ tự tăng dần (giảm dần) của giá trị khóa (key).
- ◆ Thuật toán tìm kiếm nhị phân được thiết kế dựa trên chiến lược chia để trị
- ◆ Thuật toán: So sánh khóa **k** với khóa của phần tử ở giữa dãy.
  - Nếu trùng thì thông báo tìm thấy và dừng
  - Nếu  $k >$  thì gọi đệ qui tìm trên nửa cuối dãy
  - Nếu  $k <$  thì gọi đệ qui tìm trên nửa đầu dãy
  - Quá trình tìm nếu phải tìm trong dãy rỗng thì dừng lại và thông báo không tìm thấy

Search

9

## 2.1 Tìm kiếm nhị phân trên mảng

• **Ví dụ 1:** Cho dãy dưới đây. Tìm phần tử  $k=6$

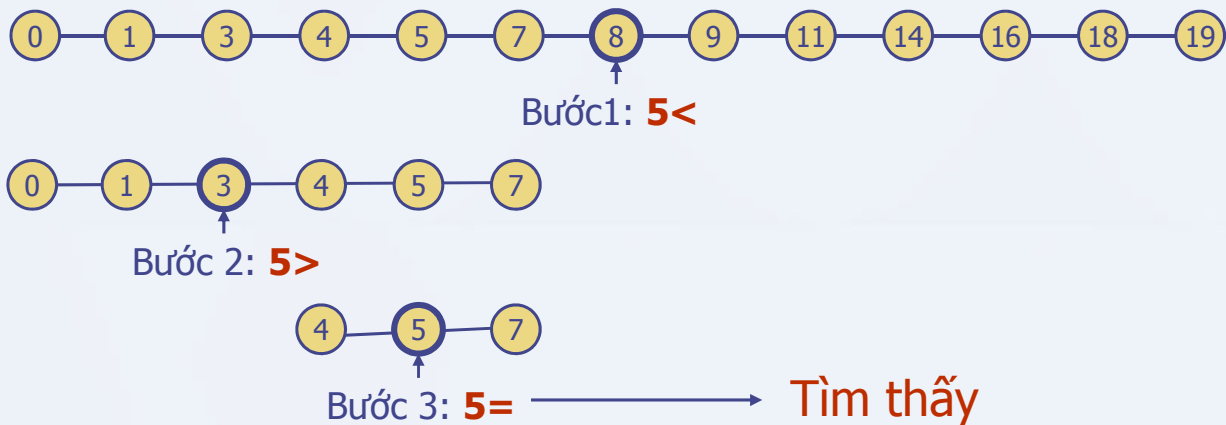


Search

10

## 2.1 Tìm kiếm nhị phân trên mảng

- **Ví dụ 2:** Cho dãy dưới đây. Tìm phần tử  $k=5$



## 2.1 Tìm kiếm nhị phân trên mảng

- **Thuật toán:**

```
Algorithm BinarySearch(S, k, n);  
    Found = 0;  
    i = 1; j = n;  
    while(i ≤ j && Found != 1){  
        mid = (i+j) / 2;  
        if (k == S[mid].Key) Found = 1;  
        else  
            if (k < S[mid].Key)  
                j = mid - 1;  
            else  
                i = mid + 1;  
    }  
    return Found;
```



## 2.1 Tìm kiếm nhị phân trên mảng

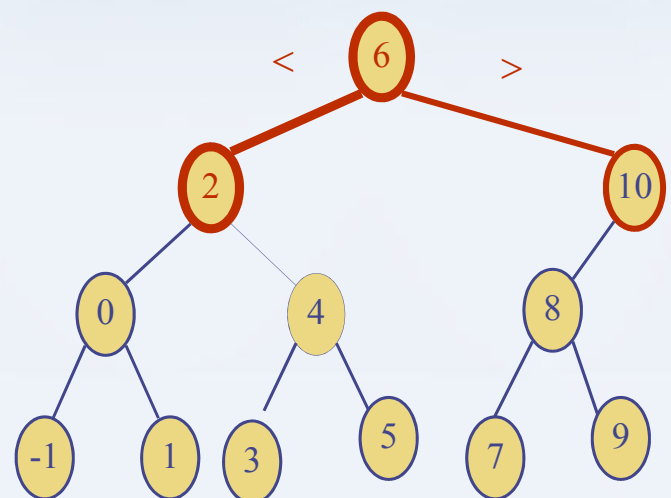
### • Thời gian chạy:

- ◆ Sau mỗi lần tìm kiếm, thì dãy cần tìm lại được chia đôi và ta chỉ phải tìm trên một nửa
- ◆ Trong trường hợp xấu nhất là không tìm thấy phần tử có khóa k. Và như vậy ta phải thực hiện chia đôi liên tiếp đến khi được dãy rỗng. Số lần thực hiện chia đôi là:  $\log_2 n$
- ◆ Vậy thời gian chạy là  $O(\log_2 n)$

## 2.2 Tìm kiếm trên cây TKNP

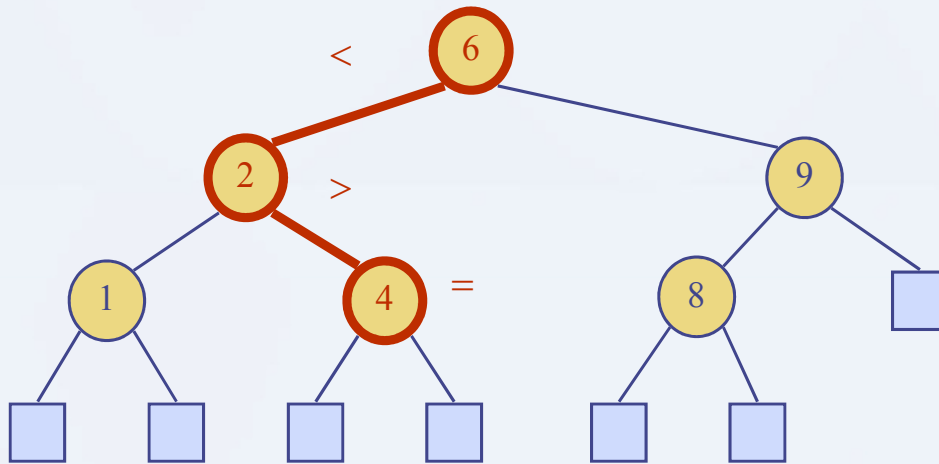
- ◆ Định nghĩa: cây tìm kiếm nhị phân là cây nhị phân thỏa mãn:

- Nút cha có khóa lớn hơn khóa của tất cả các nút của cây con bên trái, và nhỏ hơn khóa của tất cả các nút của cây con bên phải.
- Các nút con trái và phải cũng là cây tìm kiếm nhị phân



Ví dụ: Cây tìm kiếm nhị phân

# Cây tìm kiếm nhị phân (Binary search tree)

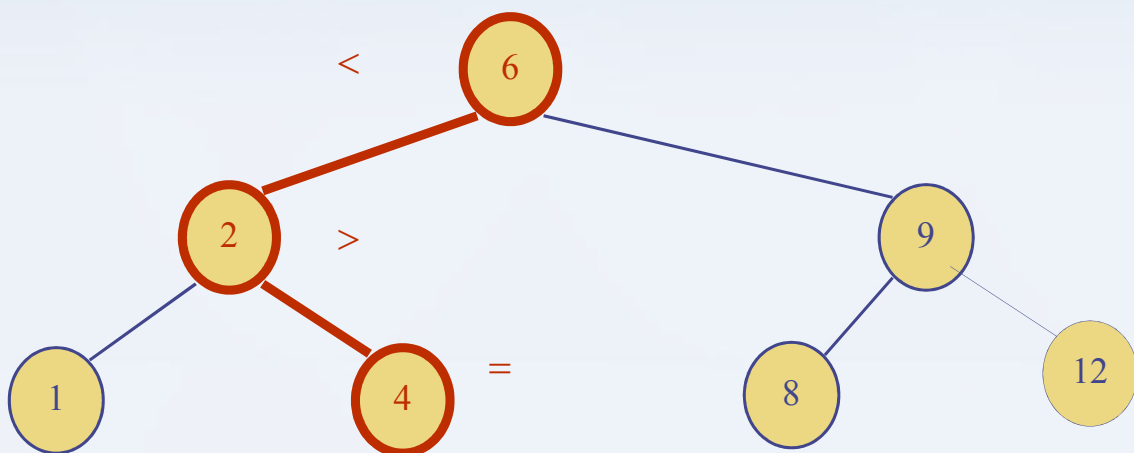


Search

15

## 2.2 Tìm kiếm trên cây TKNP

• Ví dụ: Find(4)



Search

16



# Cấu trúc Node biểu diễn cây nhị phân

## ■ Phương thức

### ■ Thuộc tính

- ◆ Keys *key*
- ◆ T elem
- ◆ Node \*Parent
- ◆ Node \*Left
- ◆ Node \*Right

Chú ý: Keys là tập các giá trị được sắp có thứ tự

- ◆ Node \*Parent()
- ◆ Node \*Left()
- ◆ Node \*Right()
- ◆ void setLeft(Node\*)
- ◆ void setRight(Node\*)
- ◆ void setParent(Node \*)
- ◆ int hasLeft()
- ◆ int hasRight()
- ◆ Object getElem()
- ◆ void setElem(T o)
- ◆ void setKey(Keys k)
- ◆ Keys getKey()

Search

17

# Cấu trúc của cây TKNP

## ◆ Thuộc tính

- Node<Keys,T> \* root

## ◆ Phương thức

- int size()
- int isEmpty()
- int isInternal(Node<Keys,T>\*)
- int isExternal(Node<Keys,T>\*)
- int isRoot(Node<Keys,T>\*)
- void preOrder(Node<Keys,T>\*)
- void inOrder(Node<Keys,T>\*)
- void postOrder(Node<Keys,T>\*)
- Node<Keys,T>\*TreeSearch(Keys, Node<Keys,T>\*)
- Node<Keys,T>\* InsertTree(Keys, T)
- void Remove(Keys)

## ❖ Các phương thức truy cập:

- Node<Keys,T> \*root()

Search

18

# Thuật toán tìm kiếm

◆ Tìm trong cây có nút có khóa bằng ***k*** không?

◆ Thuật toán

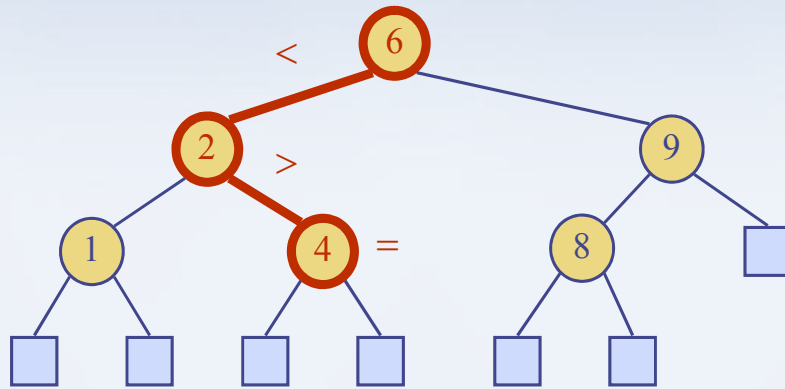
- Xuất phát từ nút gốc
- So sánh giá trị khóa của nút gốc với *k*
  - ◆ Nếu giá trị = ***k*** thì trả lại đ/c của nút đó và dừng lại
  - ◆ Nếu giá trị < ***k*** thì tiếp tục tìm trên cây con phải
  - ◆ Nếu giá trị > ***k*** thì tiếp tục tìm trên cây con trái.

Quá trình tìm dừng lại khi tìm **thấy** hoặc phải tìm trên cây **rỗng**, trả lại địa chỉ của nút mà thuật toán dừng lại

# Thuật toán giả mã

```
Node* TreeSearch(Keys k, Node<T>* v)
    if(v==NULL)
        return v;
    else
        if (k < v->getKey())
            return TreeSearch(k, v->Left());
        else if (k == v->getKey())
            return v;
        else // k > v->getKey()
            return TreeSearch(k, v->Right());
```

# Ví dụ



Tìm giá trị 4 trên cây:

- Gọi `T.TreeSearch(4, T.root())`
- Gọi `T.TreeSearch(4, T.root()->Left())`
- Gọi `T.TreeSearch(4, T.root()->Left()->Right())`

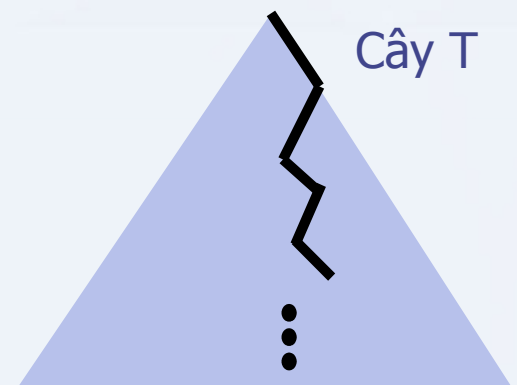
Search

21

## Phân tích thuật toán

- Là thuật toán đệ qui,
- Mỗi lần gọi đệ qui nó thực hiện một số phép toán cơ bản không đổi, vậy một lần gọi đệ qui cần thời gian là  $O(1)$
- Thực hiện gọi đệ qui dọc theo các nút, bắt đầu từ nút gốc, mỗi lần gọi đệ qui nó đi xuống một mức.
- Do đó số nút tối đa mà nó phải đi tới không vượt quá chiều cao  $h$  của cây.
- Thời gian chạy là  $O(h)$

◆ Trong trường hợp xấu nhất cây có chiều cao bằng bao nhiêu?



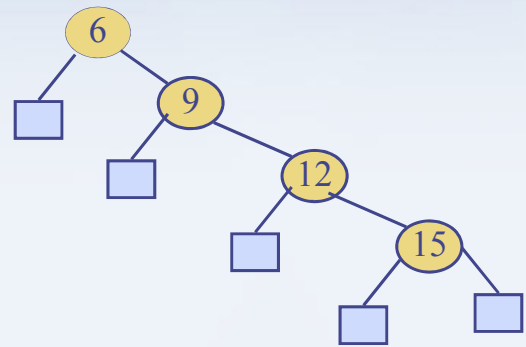
Search

22

# Một số trường hợp

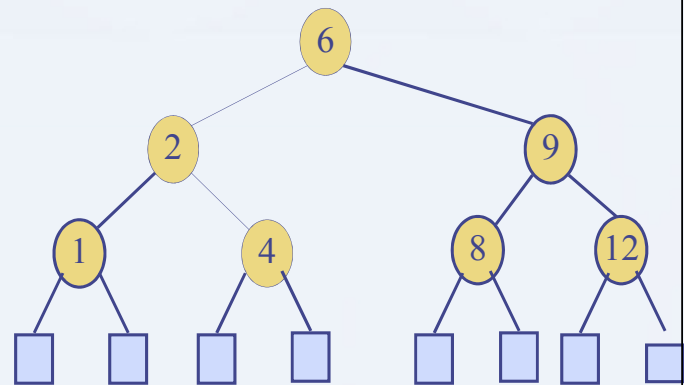
- ◆ Cây chỉ có cây con trái hoặc cây con phải

- Chiều cao cây bằng số nút của cây



- ◆ Cây hoàn chỉnh

- Chiều cao của cây là  $\log_2 n$



Search

23

## Bổ sung nút vào cây- Insertion

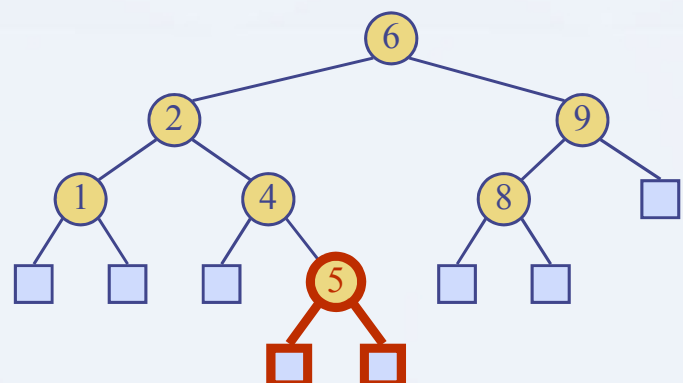
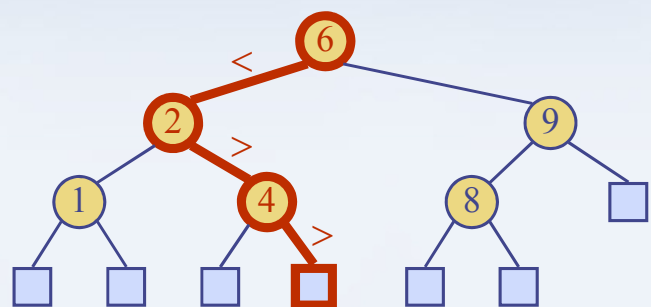
Sau khi bổ sung cây vẫn thỏa mãn tính chất cây TKNP

- ◆ Bổ sung một nút với khóa  $k$  và value  $x$

- Thực hiện tìm kiếm  $k$  trên cây

- ◆ Giả sử  $k$  không có trên cây

- Nếu cây rỗng thì gán nút cho gốc cây
- Ngược lại, kiểm tra
  - ◆ Nếu khóa  $k <$  khóa nút gốc thì chèn nút mới vào cây con bên trái
  - ◆ Nếu khóa  $k >$  khóa nút gốc thì chèn nút mới vào cây con bên phải



Search

24

# Hàm bổ sung nút vào cây- Insertion

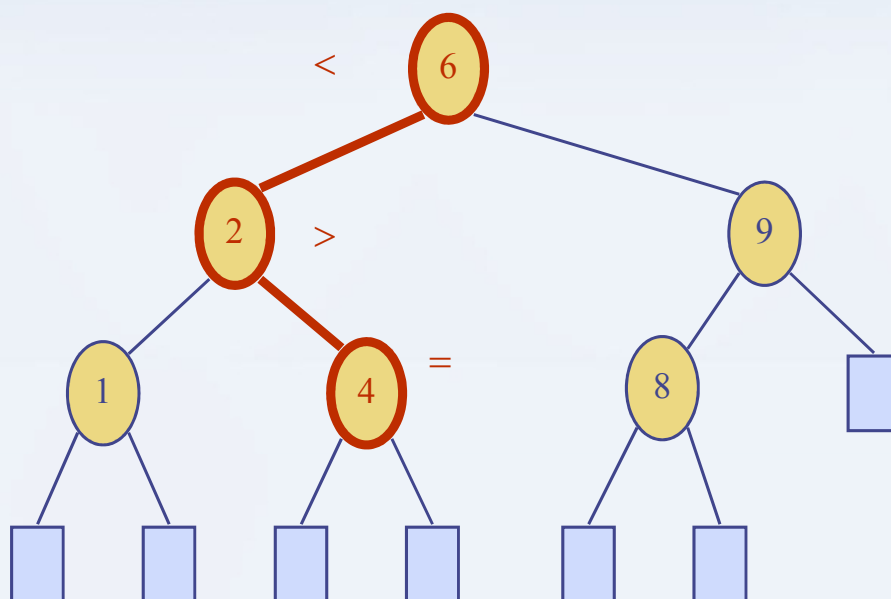
Sau khi bổ sung cây vẫn thỏa mãn tính chất cây TKNP

```
void InsertNode(Node *&root, Node *newNode){  
    if (root == NULL)  
        root = newNode;  
    else if (newNode->getKey() < root->getKey())  
        InsertNode(root->Left(), newNode);  
    else  
        InsertNode(root->Right(), newNode);  
}
```

Search

25

## Ví dụ



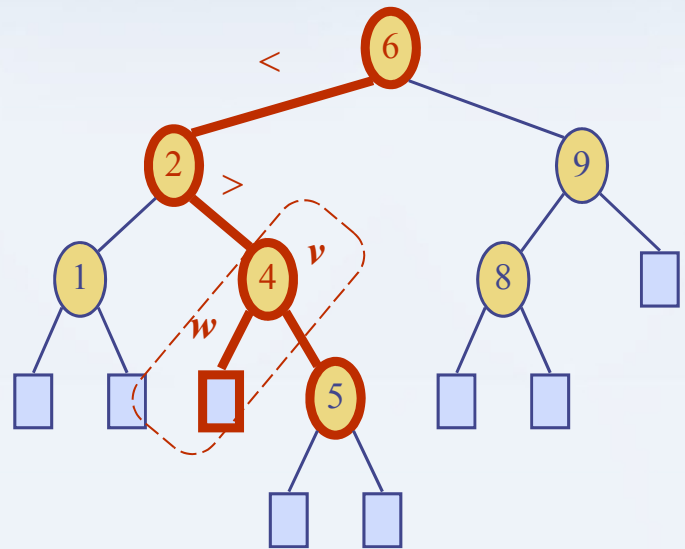
Search

26



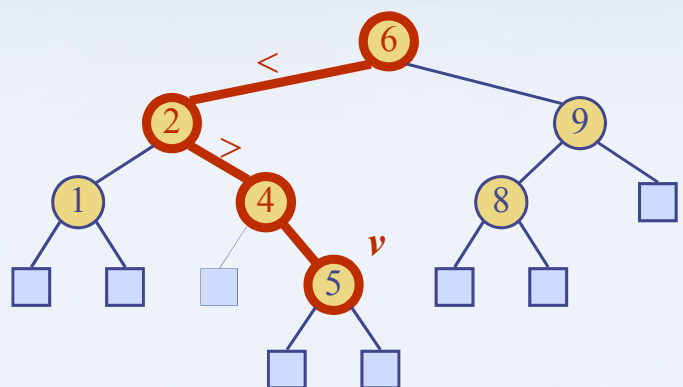
# Xóa – Sau khi xóa nút, cây vẫn thỏa mãn tính chất cây TKNP

- ◆ Xảy ra hai khả năng:
  - Nút cần xóa là nút trong
  - Nút cần xóa là nút ngoài
- ◆ Xóa một nút trong yêu cầu giải quyết lỗ hổng bên trong cây
- ◆ Để thực hiện thao tác **remove( $k$ )**, Chúng ta tìm kiếm đến nút có khóa  $k$
- ◆ Giả sử rằng khóa  $k$  có ở trên cây, và ta đặt  $v$  là nút lưu trữ  $k$



## Xóa nút ngoài

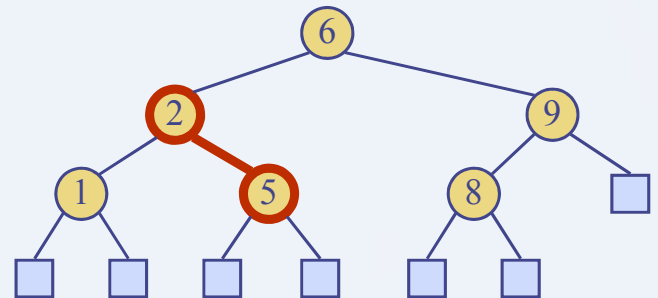
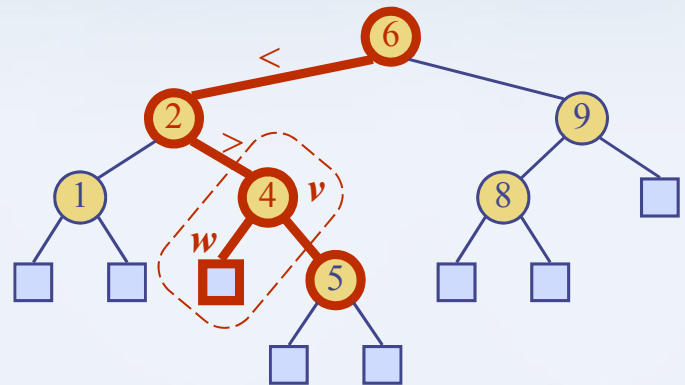
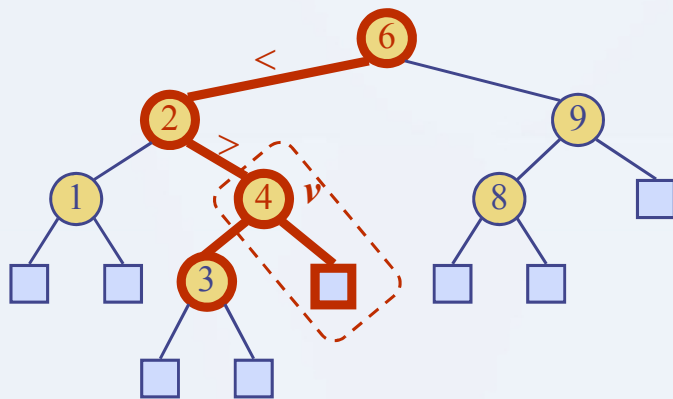
- ◆ Xóa bỏ nút  $v$





## Xóa nút trong $\nu$ chỉ có con trái hoặc phải

- Loại bỏ  $\nu$  và thay thế cây con của  $\nu$  vào vai trò của  $\nu$
- Ví dụ: xóa bỏ 4

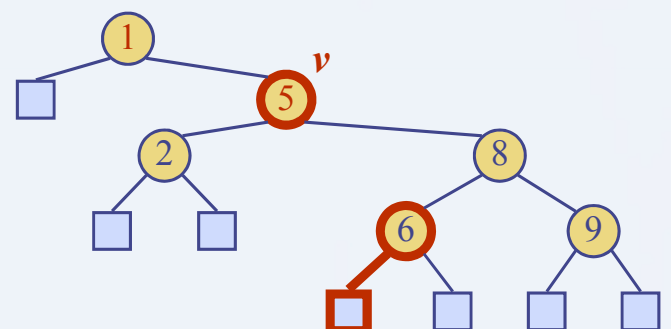
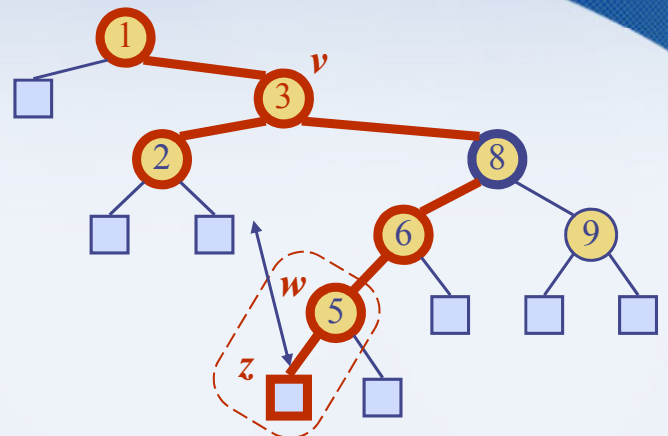


Search

29

## Xóa nút trong $\nu$ có cả hai nút con trái và phải

- Xóa nút đó và thay thế nó bằng nút có khóa lớn nhất trong các khóa nhỏ hơn khóa của nó (được gọi là "nút tiền nhiệm" - nút cực phải của cây con trái) hoặc nút con nhỏ nhất trong các khóa lớn hơn nó (được gọi là "nút kế vị" - nút cực trái của cây con phải). Cũng có thể tìm nút tiền nhiệm hoặc nút kế vị đối chỗ nó với nút cần xóa và sau đó xóa nó. Vì các nút kiểu này có ít hơn hai con nên việc xóa nó được quy về hai trường hợp trước.
- Ví dụ: xóa bỏ 3

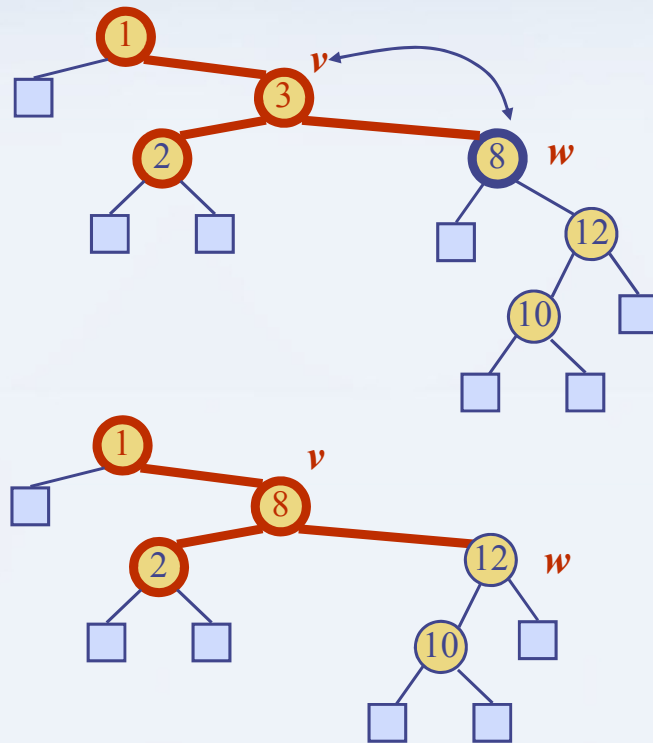


Search

30

# Ví dụ

- Xóa phần tử 3



Search

31

# Thuật toán giả mã

- ◆ Xây dựng phương thức duyệt trên một cây con của cây theo thứ tự giữa, hàm trả lại địa chỉ của nút đầu tiên được thăm
- ◆ Xây dựng hàm xóa bỏ một nút trên cây

Search

32

```

void Remove(Node*& node) {
    if (node->Left() == NULL) {
        Node* temp = node; node = node->Right(); delete temp;
    } else if (node->Right() == NULL) {
        Node* temp = node; node = node->Left(); delete temp;
    } else {
        Node** temp = &(node->Left());
        while ((*temp)->Right() != NULL) {
            temp = &((*temp)->right);
        }
        node->setKey ((*temp)-> getKey());
        node->setElem ((*temp)-> getElem());
        Remove(*temp);
    }
}

```

Search

33

Hàm duyệt cây con của một cây theo thứ tự  
giữa và trả lại đ/c của nút được thăm đầu tiên

```

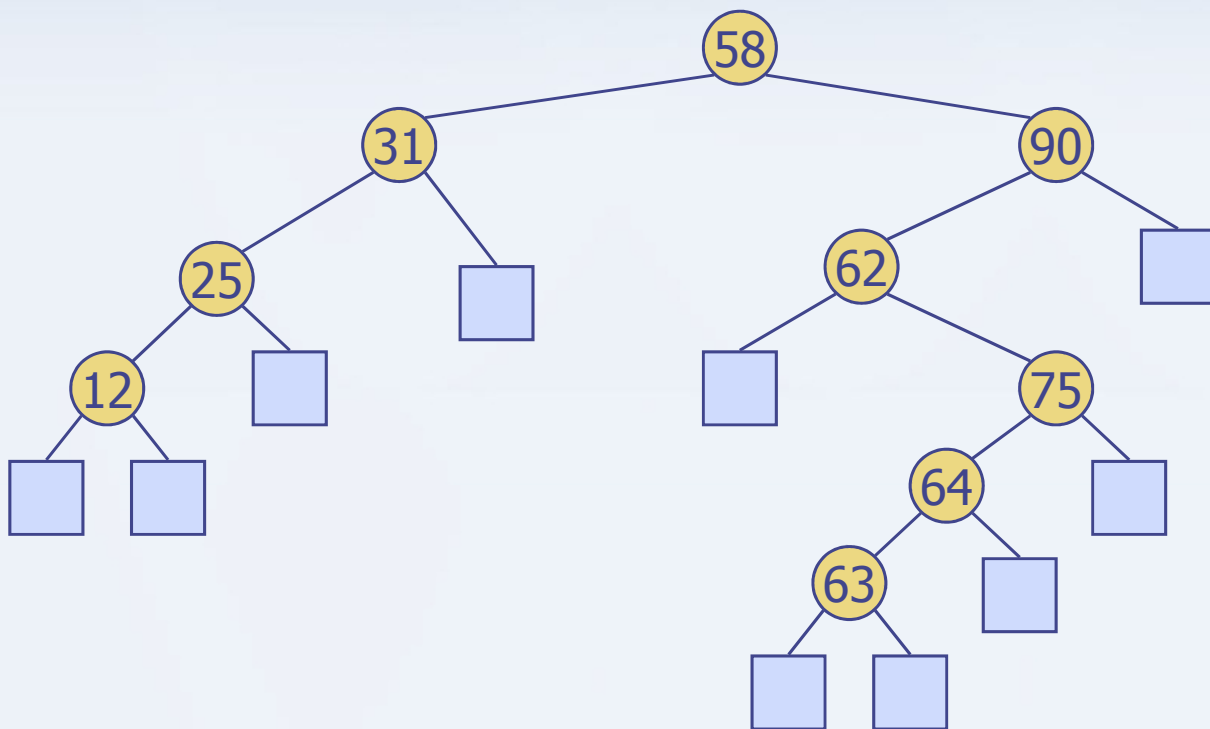
void InOrder(Node<T>*v, Node<T> *first, int &kt){
    if(v!=null && kt!=1){
        InOrder(v.Left(),first, kt);
        if(kt==0){
            first = v;
            kt=1;
        }
        InOrder(v.Right(),first, kt);
    }
}

```

Search

34

## Ví dụ: Mô tả từng bước xóa bỏ nút có key=58?



Search

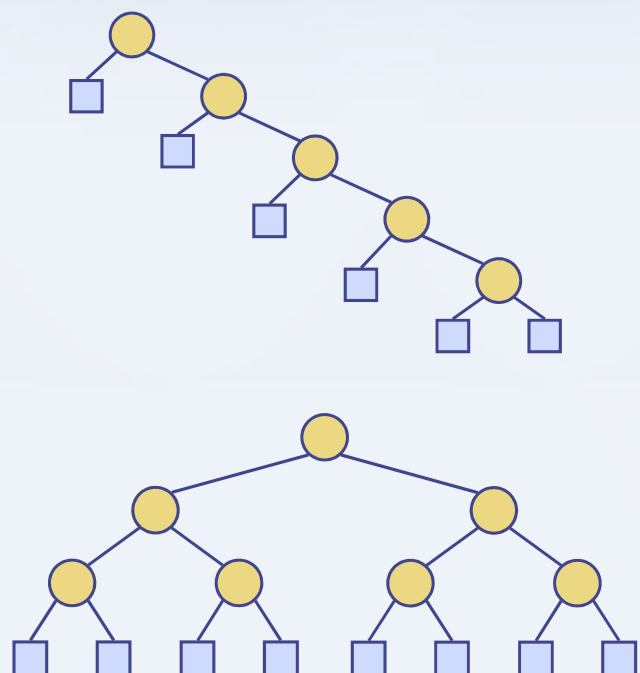
35

## Đánh giá thời gian

◆ Xem xét việc cài đặt một từ điển có  $n$  mục được cài đặt bằng một cây nhị phân tìm kiếm với chiều cao  $h$

- Bộ nhớ sử dụng  $O(n)$
- Phương thức **find**, **insert** và **remove** mất thời gian  $O(h)$

◆ Trong trường hợp xấu nhất chiều cao của cây là  $O(n)$  và trường hợp tốt nhất là  $O(\log n)$



Search

36

# Bài tập

- ◆ Xây dựng lớp cây tìm kiếm nhị phân
- ◆ Sử dụng lớp cây tìm kiếm nhị phân xây dựng một chương trình tra cứu từ điển có các chức năng sau:
  - Đọc dữ liệu từ điển nạp vào cây từ tệp
  - Bổ sung từ mới vào cây
  - Xóa bỏ một từ khỏi cây
  - Tìm kiếm từ
  - Lưu cây vào tệp

Hết