7.26 判断是否二叉搜索树

```
bool isBST(Node* root){
     static Node *prev = NULL;
     if (root){
          if (!isBST(root->left))
               return false;
          if (prev != NULL && root->key <= prev->key)
               return false;
          prev = root;
          return isBST(root->right);
     }
     return true;
}
```

7.27 插入一个元素到二叉搜索树中

```
template<class E, class K>
void Insert(BTreeNode *&BST, ElemType item){
    BTreeNode *temp = BST, *s = NULL;
    while (temp != NULL) {
        s = temp;
        if (item.key == temp->data.key)     //直到找到关键码相同的结点跳出循环
            break;
        else if (item.key < temp->data.key)//欲插入元素关键码比当前结点小，在左子树中继续查找
            temp = temp->left;
        else        //欲插入元素关键码比当前结点大，在右子树中继续查找
            temp = temp->right;
    }
    if (temp != NULL)     //元素已存在，该元素 count 值+1
        temp->data.count++;
    else {      //不存在相同元素，插入一个新的结点并将其 count 值置 1
        BTreeNode* p = new BTreeNode;
        p->data = item;
        p->data.count = 1;
        p->left = NULL;
        p->right = NULL;
        if (s == NULL)
            BST = p;
        else {
            if (item.key < s->data.key)
                s->left = p;
            else
                s->right = p;
        }
    }
}
```

补充题：

```
template<class E, class K>
bool BST<E, K>::RemoveMax(BSTNode<E, K> *&ptr) {
    BSTNode<E, K> *temp;
    if (ptr != NULL) {
        if (ptr->right == NULL && ptr->left == NULL)//左右子树都空，当前结点元素就是最大元素，直接删除
            delete ptr;
        else if (ptr->right != NULL) {    //右子树非空，最大元素肯定在右子树中
            BSTNode<E, K> *pre;    //记录右子树中序最后一个节点的父节点
            pre = ptr;
            temp = ptr->right;
            while (temp->right != NULL) {    //找右子树中序下最后一个结点
                pre = temp;
                temp = temp->right;
            }
            if (temp->left == NULL) { //若右子树中序下最后一个结点的左子树子树也为空则直接删除这个结点
                delete temp;
                return true;
            }
            else {    //否则将 temp 的左子树作为 temp 的父节点 pre 的右子树
                pre->right = temp->left;
                return true;
            }
        }
        else {    //右子树空而左子树非空，则根结点元素即最大元素
            ptr = ptr->left;    //将左子树根结点作为树根
            return true;
        }
    }
    return false;
}
```