Binary Search Tree.

Traversal

Inorder

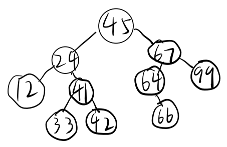
left → root → right

Preorder

root → left → right.

Postorder

left → right → root.



Preorder :

root → left → right.

(45, 24, 12, 41, 33, 42., 67, 64, 66, 99,)

Inorder :

Left → root → right

(12, 24, 33, 41, 42. 45, 64, 66, 67, 99.)

Postorder

left → right → root

(12, 33, 42, 41, 24, 66, 64, 99, 67, 45.)

Binary Search Tree Traversal.

1. Inorder Traversal (x)

```
inorder Traversal (X).
If  x ≠ NuLL.
    inorder Traversal (x.left)
    print (x.key)
    inorder Traversal (x.right)
```

Left → Root → Right.

2. Preorder Traversal (x)

```
preorder Traversal (X).
If  x ≠ NuLL.
    print (x.key)
    preorderTraversal (x.left)
    preorderTraversal (x.right)
```

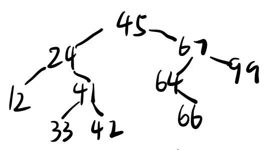
Root → left → Right

3. Post order Traversal (x)

```
post Traversal (X).
If  x ≠ NuLL.
    post Traversal (x.left)
    post Traversal (x.right)
  print (x.key)
```

Left → Right → Root

Binary Search Tree.



succ(45): 64
succ(42): 45

Successor(x)

x的后继是 x右子树中的最小节点、/ 比x大的值中的最小的值.

Predessor(x)

比x小的值中的最大的值.

Binary Search Tree Searching.

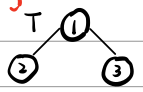
Given a pointer to the root of the tree and a key.
Return: a pointer to a node with key k if one exist;
otherwise, return Null

```
Search(x, k)
  if x == Null or k == x.key
      return x.
  if k < x.key
      return Search(x.left, k)
  else
      retur Search(x.right, k)
```

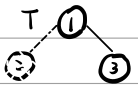
Deletion

Deleting a node z from a binary Search Tree T has 3 basic cases.

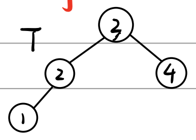
1. If z has no children



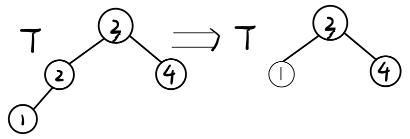
delet (T, 2) 将z替换成Null即可.



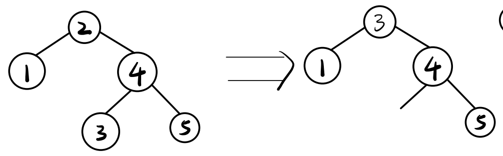
2. If z has one child



delet(T, 2) 将z节点的child移动到原本z的位置,
并替换其parent



3. If z has two children



find z's successor and replace z