- 二叉树
- 左孩子右兄弟
- 递归遍历Travesal
  - preorder
  - postorder
  - inorder
- 迭代遍历
  - preorder
  - inorder
  - level-order
- 二叉搜索树BST
  - search
  - min&max
  - successor
  - insert
  - remove
- 树堆Treap
  - Insert
  - remove

> 叶高度为0，根深度为0

二叉树

- full：0或2叶
- complete(完全)：除了最后一层都是满的，最后一层都靠左

- perfect：全满（教材里叫complete）

```
struct Node {
   Data data;
   Node *parent;
   Node *left;
   Node *right;
}
```

左孩子右兄弟

```
struct Node {
   Data data;
   Node *parent;
   Node *firstChild;
   Node *nextSibling;
}
```

递归遍历Travesal

preorder

> 深搜
>
> 目录展开的样子

PreorderTrav(r):

```
if (r != NULL)
   Visit(r)
   for (each child u of r)
      PreorderTrav(u)
```

postorder

> 从右向左深搜，再反过来
>
> 递归mergesort

PostorderTrav(r):

```
if (r != NULL)
   for (each child u of r)
      PostorderTrav(u)
   Visit(r)
```

inorder

> 从左向右，升序输出

InorderTrav(r):

```
if (r != NULL)
  InorderTrav(r.left)
  Visit(r)
  InorderTrav(r.right)
```

迭代遍历

> 需要栈

```
struct Frame {
  Node *node;
  bool visit;
  Frame(Node* n,bool v) {
    node = n;
    visit = v;
  }
}
```

Visit node or the subtree rooted at node.

preorder

PreorderTravIter(root):

```
Stack s
s.push(Frame(root,false))
while (!s.empty())
  f = s.pop()
  if (f.node != NULL)
    if (f.visit)
      Visit(f.node)
    else
      for (each child u of f.node)
        s.push(Frame(u,false))
      s.push(Frame(f.node,true))
```

inorder

InorderTravIter(root):

```
Stack s
s.push(Frame(root,false))
while (!s.empty())
  f = s.pop()
  if (f.node != NULL)
    if (f.visit)
      Visit(f.node)
    else
      s.push(Frame(f.node->right,false))
      s.push(Frame(f.node,true))
      s.push(Frame(f.node->left,false))
```

level-order

> 宽搜

> 迭代mergesort
> 先添加，再一个一个访问

LevelorderTrav(r):

```
Queue q
q.add(r)
while (!q.empty())
  node = q.remove()
  if (node != NULL)
    Visit(node)
    q.add(node->left)
    q.add(node->right)
```

二叉搜索树BST

> 宽搜是key的升序
> 时间复杂度就是h

Efficient implementation of OSet

	Search(S,k)	Insert(S,x)	Remove(S,x)
SimpleArray	$O(n)$	$O(1)$	$O(n)$
SimpleLinkedList	$O(n)$	$O(1)$	$O(1)$
SortedArray	$O(\log n)$	$O(n)$	$O(n)$
SortedLinkedList	$O(n)$	$O(n)$	$O(1)$
BinaryHeap	$O(n)$	$O(\log n)$	$O(\log n)$
BinarySearchTree	$O(h)$	$O(h)$	$O(h)$

BST also supports other operations of **OSet**, in $O(h)$ time.

But height of a n-node BST varies between $\Theta(\log n)$ and $\Theta(n)$.

search

> 递归，迭代

BSTSearch(x,k):

```
if (x==NULL or x.key==k)
  return x
else if (x.key>k)
  return BSTSearch(x.left,k)
else
  return BSTSearch(x.right,k)
```

BSTSearchIter(x,k):

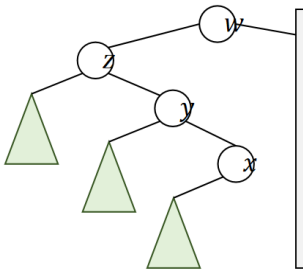
```
while (x!=NULL and x.key!=k)
  if (x.key>k)
    x = x.left
  else
    x = x.right
return x
```

min&max

> keep going left&right

successor

> 找到大于x的最小元素
>
> 讨论x是否有右孩子
>
> 向上查找直到有一个ancestor是左孩子



BSTSuccessor(x):

```
if (x.right!=NULL)
  return BSTMin(x.right)
y = x.parent
while (y!=NULL and y.right==x)
  x = y
  y = y.parent
return y
```

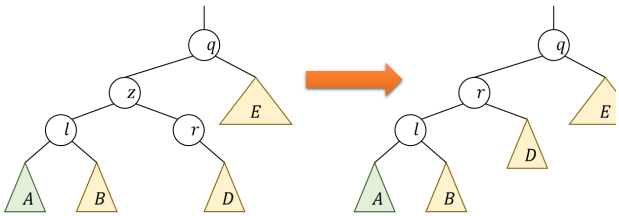
insert

> 从root开始，小了向左，大了向右
>
> 最后添加在叶子下
>
> 正确性说明：如果树中本来就有这个数，也该在这个位置上

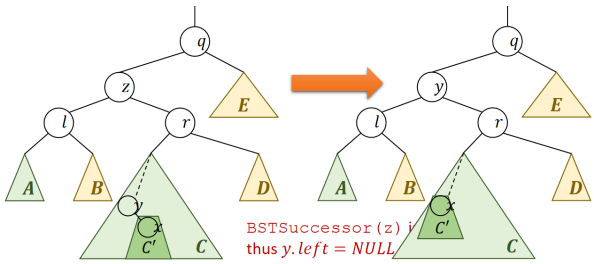
remove

> 讨论z的子节点数

- 无子：直接删
- 1子：把子提上来
- 2子，右子无左子：把右子提上来



- 2子，右子有左子：需要查找（successor）右子的左子树中最小的，替换上来

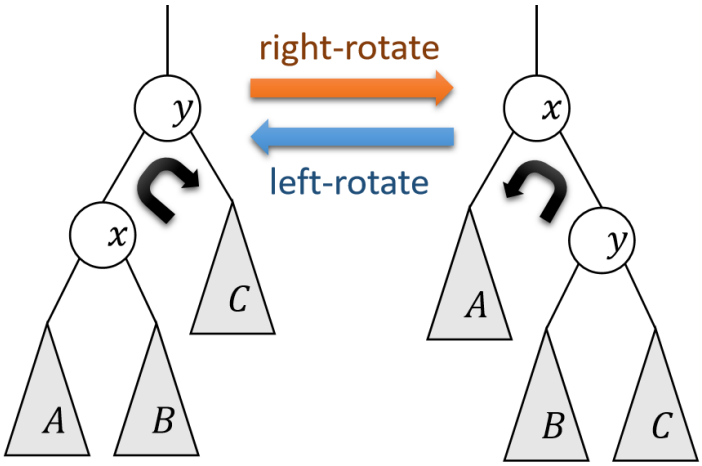


树堆Treap

> 不一定是二叉堆，最后一层靠左原则可能不满足
> key value : BST-property
> priority value: MinHeap-property

Insert

> 先赋值随机priority
> 先插入BST
> 堆调整，左右2rotate



Rotation changes level of x and y,
but preserves BST property.

remove

> rotate直到把x移到叶