【DS】Day17

Tags	
Date	@June 12, 2022
Summary	Binary Search Tree

【Week 4】Symbol Table

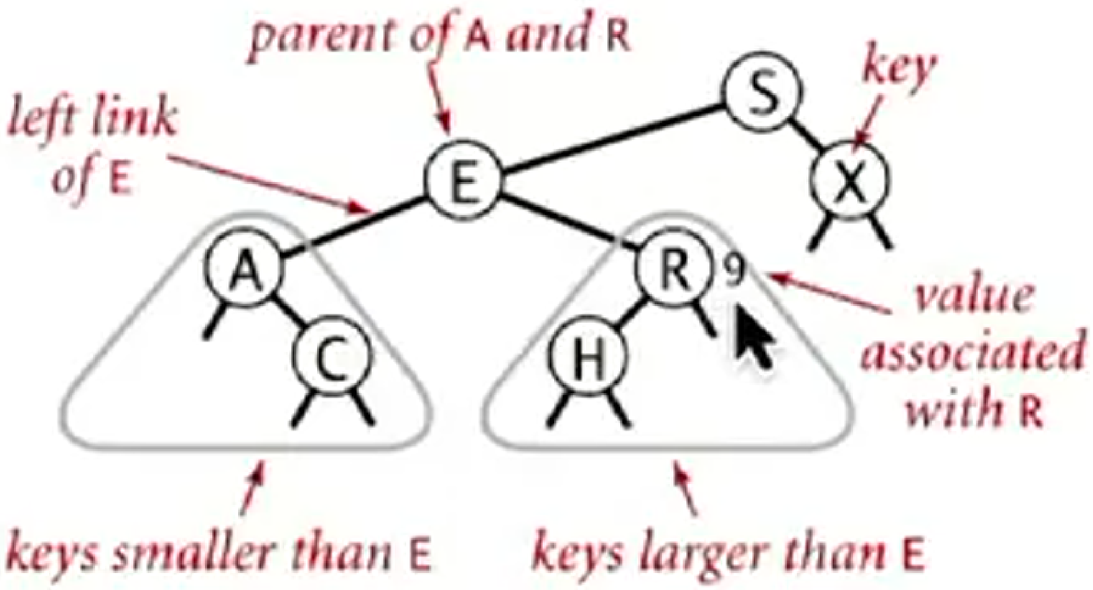
4.7 Binary Search Tree

A BST is a binary tree in symmetric order.

Symmetric order:

Each node has a key, and every node’s key is :

- Larger than all keys in its left subtree
- Smaller than all keys in its right subtree



```
private class Node {
  private Key key;
  private Value val;
  private Node left, right;
  private int count;

  public Node(Key key, Value value) {
    this.key = key;
    this.val = value;
  }
}
```

Binary Symbol Table Skeleton

```
public class BST<Key extends Comparable<Key>, Value> {
  private Node root;

  private class Node {}

  public void put(Key key, Value val) {
    root = put(root, key, val);
  }
  private Node put(Node x, Key key, Value val) {
    if (x == null) return new Node(key, val);
    int cmp = key.compareTo(x.key);

    if (cmp < 0) x.left = put(x.left, key, val);
    else if (cmp > 0) x.right = put(x.right, key, val);
    else x.val = val;

    x.count = 1 + size(x.left) + size(x.right);
    return x;
  }

  public Value get(Key key) {
    Node x = root;
    while (x != null) {
      int cmp = key.compareTo(x.key);
      if (cmp < 0) x = x.left;
      else if (cmp > 0) x = x.right;
      else return x.val;
    }
    return null;
  }

  public void delete(Key key) {
    root = delete(root, key);
  }

  private Node delete(Node x, Key key) {
    if (x == null) return null;
    int cmp = key.compareTo(x.val);
    if (cmp < 0) x.left = delete(x.left, key);
    else if (cmp > 0) x.right = delete(x.right, key);
    else {
      if (x.right == null) return x.left;
      if (x.left == null) return x.right;

      Node t = x;
      x = min(t.right);
      x.right = deleteMin(t.right);
      x.left = t.left;
    }
    x.count = 1 + size(x.left) + size(x.right);
    return x;
  }

  public Iterable<Key> iterator() {
    Queue<Key> q = new Queue<Key>();
    inorder(root, q);
    return q;
  }

  private void inorder(Node x, Queue<Key> q) {
    if (x == null) return;
    inorder(x.left, q);
    q.enqueue(x.val);
    inorder(x.right, q);
  }

  public int size() {
    return size(root);
  }

  private int size(Node x) {
    if (x == null) return 0;
    return x.count;
  }
}
```

4.8 Ordered Operations in BST

Computing the Floor

Floor: The greatest value that is smaller than or equal to the given key.

```
public Key floor(Key key) {
  Node x = floor(root, key);
  if (x == null) return null;
  return x.key;
}

private Node floor(Node x, Key key) {
  if (x == null) return null;
  int cmp = key.compareTo(x.key);

  if (cmp == 0) return x;
  else if (cmp < 0) return floor(x.left, key);

  Node t = floor(x.right, key);
  if (t != null) return t;

  return x;
}
```

Rank

Rank: How many keys < k?

```
public int rank(Key key) {
  return rank(key, root);
}

private int rank(Key key, Node x) {
  if (x == null) return 0;
  int cmp = key.compareTo(x.val);

  if (cmp < 0) return rank(key, x.left);
  else if (cmp > 0) return 1 + size(x.left) + rank(x.right);
  else return size(x.left);
}
```

4.9 Deletion in BST

To delete the minimum key:

- Go left until finding a node with a null left linnk
- Replace the node by its right link
- Update subtree counts

```
public void deleteMin() {
  root = deleteMin(root);
}

private Node deleteMin(Node x) {
  if (x == null) return null;
  if (x.left == null) return x.right;

  x.left = deleteMin(x.left);
  x.count = 1 + size(x.left) + size(x.right);
  return x;
}
```

Hibbard Deletion

To delete a node with key k: search for node t containing key k.

Case 0: 0 children. Delete t by setting parent link to null.

Case 1: 1 child. Delete t by replacing parent link.

Case 2:

- Find the smallest value x in the right subtree.
- Delete x in the right xubtree
- Put x in t’s spot