2020/11/04 - Binary Search Tree (Part 2)

04 Tháng Mười Một 2020 3:19 SA

SYNOPSIS

- Go over Lab 6 (Parts 2 + 3)

LAB 6

- As always, on Canvas, go to the "Lab 6.2" assignment and read the "lab6.html" file attached. All lab details will be there.

- There are three parts split into two submissions. This will work in your favour in the long run.

SUBMISSION COMMAND

LAB 6.1

```
tar -cvf lab6_1.tar BST.h
```

LAB 6.2

```
tar -cvf lab6_2.tar BST.h
```

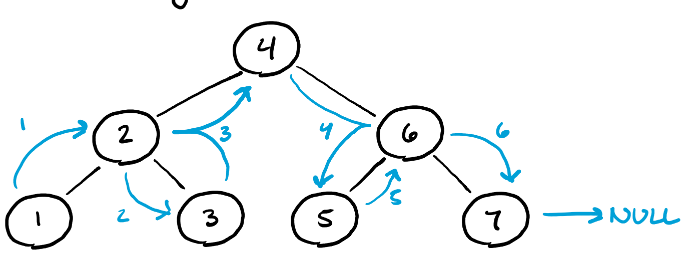
SKETCH "KEY"

- Your mileage may vary. Different TAs may expect different information. A sketch + (correct) text should do.

- I write this in a way that will help you in next parts. Maybe you don't match. Don't panic or stress over it.

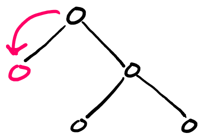
★ OPERATOR ++

- Single step in in-order traversal. May be done via considering where node came from.



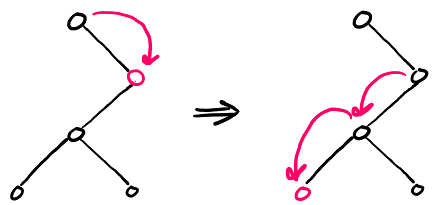
Initial

- Go as far left as possible. (bst::begin, not operator ++)

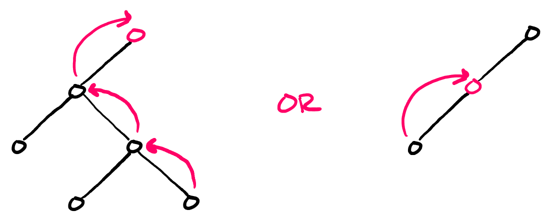


Step

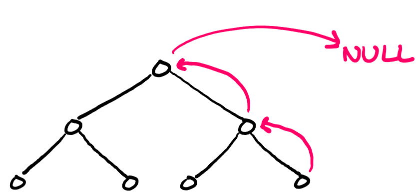
- If we can go right:
  - Go right once
  - Go left as much as possible



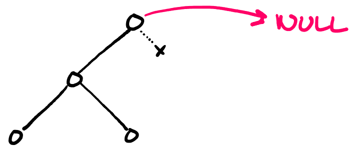
- Otherwise,
  - Go up
  - If the node we were just at is the right child of the node we are now at, go up again. REPEAT UNTIL NO LONGER TRUE.



  - If came from right and hit the root, set to NULL.

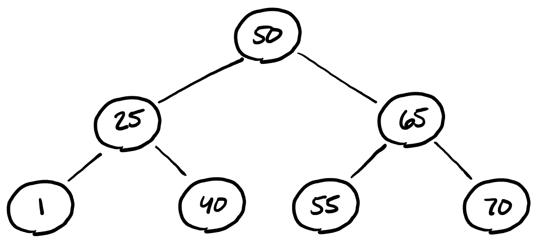


- If at root and right is NULL, set to NULL



★ LOWER-BOUND

- lower-bound ≤ key



LOWER-BOUND (1) = 1
LOWER-BOUND (25) = 25
LOWER-BOUND (26) = 40 ← It goes to the next one.
LOWER-BOUND (52) = 55
LOWER-BOUND (0) = 1
LOWER-BOUND (99) = NULL

INITIAL

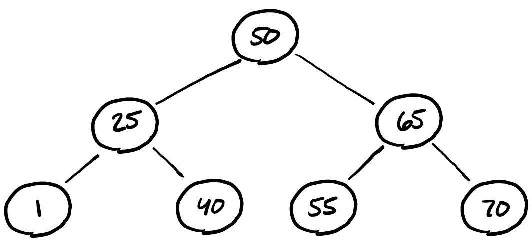
- Have two pointers:
  - t - Does tree traversing
  - n - Keeps track of last time t went left.
- n is NULL at the start. t is root.

Step

- As long as t isn't NULL...
  - If key (search key) ≤ t→key, go left. But before you do, log t in n.
  - If not ≤, have t go to the right.
    - No. n will not change here.
- Return iterator pointing to n.

☆ Upper-Bound

- key < upper-bound



Upper-Bound (1) = 25 ← It goes to the next one.
Upper-Bound (25) = 40
Upper-Bound (26) = 40
Upper-Bound (52) = 55
Upper-Bound (0) = 1
Upper-Bound (99) = NULL

Procedure

- Same as lower_bound, but < instead of ≤.

BST.h – PART 2

- Implement the iterator public subclass in bst.
- Implement the begin and end functions in bst.
- All but 2 functions are one-liners.
- Iterator prototype:

```
class iterator {
  public:
      //Constructor (Public)
      iterator();

      //Operator Overloads
      iterator & operator ++();
      Tkey     & operator * ();
      bool       operator ==(const iterator &) const;
      bool       operator !=(const iterator &) const;
  private:
      friend class bst<Tkey>;

      //Constructor (Private)
      iterator (node *);

      //Interior Pointer
      node *p;
}
```

- Other prototypes (public functions of bst introduced after iterator subclass):

```
    iterator begin();
    iterator end();
```

- So... all except bst<Tkey>::iterator::operator++ and bst<Tkey>::begin are one-liners.

- You might figure out that there are two possible ways to operator++ via Google or something. This is because of things like ++i (pre-increment) and i++ (post-increment). This lab wants the former.

- Function Definition Outlines:

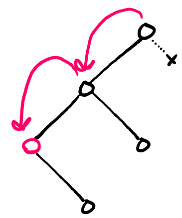
```
//Constructors (Public + Private)
template <class Tkey>
bst<Tkey>::iterator::iterator() {
  ⋮
}

template <class Tkey>
bst<Tkey>::iterator::iterator(bst<Tkey>::node *ptr) {
  ⋮
}

//Operator Overloads
template <class Tkey>
typename bst<Tkey>::iterator & bst<Tkey>::iterator::operator++() {
  ⋮
}

template <class Tkey>
Tkey & bst<Tkey>::iterator::operator*() {
  ⋮
}

template <class Tkey>
bool bst<Tkey>::iterator::operator==(const bst<Tkey>::iterator &rhs) const {
  ⋮
}
```

```
template <class TKey>
bool bst<TKey>::iterator::operator!=(const bst<TKey>::iterator &rhs) const {
  |
  |
}

//BST Functions
template <class TKey>
typename bst<TKey>::iterator bst<TKey>::begin() {
  |
  |
}

template <class TKey>
typename bst<TKey>::iterator bst<TKey>::end () {
  |
}
```

- Function Breakdown (One-Liners)
  - Both iterator constructors just set p. Either to NULL or to a specified value. Should be obvious what to do there.
  - operator * returns p's key.
  - operator== returns TRUE if p is the same as rhs.p.
  - operator!= is the opposite of operator==...
  - end() returns an iterator pointing to NULL. See sketch key to see why. (It's essentially one element past the last node. Hence NULL).
- Function Breakdown (Others)
  - operator++. Refer to sketch key. Algorithm is discussed there with diagrams.

- begin(). If Troot is NULL, return an iterator pointing NULL. Otherwise, start at Troot and go as far left as possible. Return an iterator pointing to left-most node.
  - Since it was asked in lab: No. Do not take a right.

(Ex



BST.h – PART 3

- Implement the lower_bound and upper_bound functions in bst.

- If done correctly, this is a two-for-one. A.k.a. write one and you have both (with a minor adjustment).

- Function Definition Outlines:

```
template <class TKey>
typename bst<TKey>::iterator bst<TKey>::lower_bound(const TKey &key){
  ¦
}


template <class TKey>
typename bst<TKey>::iterator bst<TKey>::upper_bound(const TKey &key){
  ¦
}
```

- Refer to sketch key. Procedure and examples are shown. #TeachBySledgehammer.