ECE 590.01
C++ Programming, Data structures, and Algorithms
Binary Search Trees

Admin
- Hwk 2
- Project Proposals
  - Due today
  - Due this week (apparently I said Fri somewhere?)
- Reading
  - Chapter 4

Remind us where we left off last time?
- Who can remind us what we talked about?
- C++ vs C
- OOP: Inheritance, design
- Sets
- Performance
- Big-O
- Iterators
- Leftover:
  - Maps

Map: the most useful ADT ever
- Sets are incredibly useful
- Maps are even more useful
  - Track mappings from keys to values
  - Probably the most ubiquitous thing we do in programs

- Interface
  - add(key, value)
  - find(key)
  - delete(key) [maybe?]
  - iterators [maybe?]
O(lg N): cut the problem in half

- Recall that
  - \( \lg N = x \) means \( 2^x = N \)
  - \( \lg 32768 = 15 \)
  - \( \lg 65536 = 16 \)
  - \( \lg 131072 = 17 \)
  - \( \lg 4 \text{ billion} \approx 32 \)
- To have a \( \lg N \) algorithm, we have to cut the problem size in half with each step

Binary Search on arrays

- Can perform binary search on a sorted array
  - Find a particular element in \( \lg N \) time
  - Look at middle element
  - Rule out half the array

Algorithm: setup
- Maintain low (inclusive) and high (exclusive) range
- Invariant: low \( \leq \) goal < hi
- Goal = index of value we want
  - If it's in the array
  - If not, we will know this when impossible equation
    - \( X \leq \) goal < \( X \)

Each step
- \( \text{mid} = (\text{hi} + \text{low}) / 2 \)
- \( \text{array[mid]} = \text{target? Done} \)
- \( \text{array[mid]} < \text{target? Rule out bottom half of range} \)
  - \( \text{low} = \text{mid} \)
- \( \text{array[mid]} > \text{target? Rule out top half of range} \)
  - \( \text{hi} = \text{mid} \)
Binary Search on arrays

- Example: looking for 319
  - Array[mid] is 723
  - Rule out top half and repeat

- Example: looking for 319
  - Array[mid] is 723
  - Rule out top half and repeat

- Example: looking for 319
  - Array[mid] is 723
  - Rule out top half and repeat
  - 123 is too small, rule out bottom half and repeat

- Example: looking for 319
  - Array[mid] is 723
  - Rule out top half and repeat
  - 123 is too small, rule out bottom half and repeat
  - 244 is too small, rule out bottom half and repeat

- Example: looking for 319
  - If the array has 319, it has to be here
  - If the array does not have 319, then what happens?
Binary Search on arrays

- Example: looking for 319
  - If the array has 319, it has to be here
    - Convince yourself of this
  - If the array does not have 319, then what happens?
    - We end up with hi = low
    - Know that we have ruled out entire array, and stop

Array binary search

- Suppose we implement a Set of ints with a sorted array
  - Want to check contains in \(O(\log N)\) time...

```java
class IntSet {
    int * array;
    int arraySize;
    ...

    bool contains(int x) {
        //let's write this
    }
}
```

Array binary search

```java
bool contains(int x) {
    int low = 0;
    int hi = arraySize;

    while (low < hi) {
        int mid = (low + hi) / 2;
    }
    return false;
}
```

If low >= hi, we have ruled out the entire Array, and can stop (and return false)
```java
bool contains(int x) {
    int low = 0;
    int hi = arraySize;
    while (low < hi) {
        int mid = (low + hi) / 2;
        if (array[mid] == x) { return true; }
        if (array[mid] < x) { low = mid + 1; }
        else { hi = mid; }
    }
    return false;
}
```

**Array binary search**

```java
bool contains(int x) {
    int low = 0;
    int hi = arraySize;
    while (low < hi) {
        int mid = (low + hi) / 2;
        if (array[mid] == x) { return true; }
        if (array[mid] < x) { low = mid + 1; }
        else { hi = mid; }
    }
    return false;
}
```

**Array binary search**

```java
bool contains(T x) {
    int low = 0;
    int hi = arraySize;
    while (low < hi) {
        int mid = (low + hi) / 2;
        if (array[mid] == x) { return true; }
        if (array[mid] < x) { low = mid + 1; }
        else { hi = mid; }
    }
    return false;
}
```

**Array binary search**

```java
bool contains(int x) {
    int low = 0;
    int hi = arraySize;
    while (low < hi) {
        int mid = (low + hi) / 2;
        if (array[mid] == x) { return true; }
        if (array[mid] < x) { low = mid + 1; }
        else { hi = mid; }
    }
    return false;
}
```

**Array binary search**

```java
bool contains(int x) {
    int low = 0;
    int hi = arraySize;
    while (low < hi) {
        int mid = (low + hi) / 2;
        if (array[mid] == x) { return true; }
        if (array[mid] < x) { low = mid + 1; }
        else { hi = mid; }
    }
    return false;
}
```
Template Set with binary search

template <class T> class Set {
    T * array;
    int arraySize;
    ...
    bool contains(T x) {
        //from previous slide
    }
};

Sorted array Set implementation

- Sorted array
  - \(O(\log N)\) contains \(\ll O(N)\)
  - \(O(N)\) insert \(\ll O(1)\)
  - Have to iterate through array to find spot
  - Then copy everything else down one space
  - Maybe a good tradeoff if we check contains much more than we insert?

- Can do better!
  - Binary Search Trees

ECE 590.01 (Hilton): BSTs

Binary Search Trees (BSTs)

- Pointer based data structure
  - Left + right (rather than “next”)
- Invariant:
  - Everything to the left is smaller
  - Everything to the right is greater (or equal)

BST-based Map

- Let’s switch from a Set to a Map
  - If you can do a Map you can do a Set

template<class K, class V> class BstMap {
    ...
};

Now templated over two things:
- \(K\) is the type of the Key
- \(V\) is the type of the Value
Remember: map keys to values
Maybe different types

Could have some AbstractMap which we extend for polymorphism, but that is a secondary concern

BST-based Map

template<class K, class V> class BstMap {
private:
    class BSTNode {
        public:
            K key;
            V val;
            BSTNode * left, * right;
            BSTNode(K k, V v) : key(k), val(v), left(NULL), right(NULL) {}
        };
        BSTNode * root;
    ...
};

Start with a BSTNode inner class a BstNode for the root
Add to BST

- Want to start with a way to add
  ```
  void add(K key, V val) {  ...}
  ```
- How do we do this?
  - Step 1?

Binary Search Trees (BSTs)

- Suppose we want to add 57
  - Where could it go?
  - With minimal “disruption”

Incorrect idea

- Suppose we want to add 57
  - Incorrect approach: always add as root
    - 2 and 43 are on the wrong side of 57 now!

Incorrect idea

- Suppose we want to add 57
  - Where could it go?
  - Can’t be the root… Must go left of 245… Right of 43…

Incorrect idea

- This spot is possible in this tree, but...
  - Need to know what comes below to be sure

Incorrect idea

- This spot is possible in this tree, but...
  - Need to know what comes below to be sure
  - 45 as left child of 119? 57 can’t go here anymore
Binary Search Trees (BSTs)

- Add at "leaves"
  - Bottom of tree, no more children
  - Find place by checking direction at each node

ECE 590.01 (Hilton): BSTs

BST add 3 ways

- Can approach in the same 3 ways as LL add

```cpp
void add (K key, V val) {
    BSTNode * b = new BSTNode(key, val);
    Make a new node
    if (root == NULL) {
        root = b;
        Check for empty tree
    }
    BSTNode * curr = root;
    while (1) {
        if (key < curr->key) {
            Write infinite looking loop
            BSTNode * curr = root;
            Then return out fo the middle
            while (1) {
                Write infinite looking loop
                BSTNode * curr = root;
                While this loops
                Ugly? Sometimes
        Go left?
    }
```
**BST add 3 ways**

- Can approach in the same 3 was as LL add

```cpp
void add (K key, V val) {
    BSTNode * b = new BSTNode(k,v);
    if (root == NULL) {
        root = b;
    }
    BSTNode * curr = root;
    while (1) {
        if (key < curr->key) {
            if (curr->left == NULL) {
                curr->left = b;
                return;
            }
            curr = curr->left;
        } else {
            // similar for right
        }
    }
}
```

**BST add 3 ways**

- Can approach in the same 3 was as LL add

```cpp
void add (K key, V val) {
    BSTNode * b = new BSTNode(k,v);
    if (root == NULL) {
        root = b;
    }
    BSTNode * curr = root;
    while (1) {
        if (key < curr->key) {
            if (curr->left == NULL) {
                curr->left = b;
                return;
            }
            curr = curr->left;
        } else { // similar for right
        }
    }
}
```

**Works, but ugly**

- Copy/paste = bad
- Previous approach works, but is ugly

```
Fortunately, we all have black belts in pointer kung fu
```

**Copy, paste, Replace left with right**

```
Copy, paste, Replace left with right
```

**Start out the same**

```
Start out the same
```

**But now, point at where we might Want to change**

```
But now, point at where we might Want to change
```
BST add 3 ways

- Can approach in the same way as LL add
  
  ```
  void add(K key, V val) {
      BSTNode * b = new BSTNode(k, v);
      BSTNode ** ptr = & root;
      while (*ptr != NULL) {
          if (key < (*ptr)->key) {
              ptr = & (*ptr)->left;
          } else {
              ptr = & (*ptr)->right;
          }
      }
      *ptr = b;
  }
  ```

Hey now we can easily write a loop condition that makes sense. We could do this before, but it would have been complex and cumbersome.

**Trees: recursive data structures**

- BSTs are naturally recursive data structures
  - If you have a BST and you go left, you have a BST
  - Same if you go right

- Can do add recursively too
  - Base case?
    - What is the simplest tree to add to?
**BST add 3 ways**

```cpp
void add (K key, V val, BSTNode * curr) {
    if (curr == NULL) {
        return new BSTNode (key,val);
    }
    if (key < curr->key) {
        curr->left = add (key, val, curr->left);
    } else {
        curr->right = add (key, val, curr->right);
    }
    return curr;
}
```

**Write helper (private)**

**Base case:**
Add to empty tree?
Tree with one node

**Recursive case?**
Set left/right to what you get recursing
Left/right

**Original (public) method calls helper**
Sets root to return value

```cpp
public:
void add (K key, V val) {
    root = add (key, val, root);
}
```

**BST-based Map**

- Now can add(key,val) in log N time (*)
- What about finding something?
  - V find(K key) ?
- (*) As we'll see later, we need to balance the tree to ensure log N time.

**BST find**

```cpp
V find (K key) {
    BSTNode * curr = root;
    // Start at the root
}```
BST find

V find (K key) {
BSTNode * curr = root;
while (curr != NULL) {
    if (curr->key == key) {
        return curr->val;
    }
    if (key < curr->key) {
        curr = curr->left;
    } else {
        curr = curr->right;
    }
}
}

ECE 590.01 (Hilton): BSTs 67

Keep looking
As long as we haven't
gone off the end of the tree

BST find

V find (K key) {
BSTNode * curr = root;
while (curr != NULL) {
    if (curr->key == key) {
        return curr->val;
    }
    if (key < curr->key) {
        curr = curr->left;
    } else {
        curr = curr->right;
    }
}
return ???;
}

ECE 590.01 (Hilton): BSTs 68

Maybe we found it?

BST find

V find (K key) {
BSTNode * curr = root;
while (curr != NULL) {
    if (curr->key == key) {
        return curr->val;
    }
    if (key < curr->key) {
        curr = curr->left;
    } else {
        curr = curr->right;
    }
}
throw exception();
}

ECE 590.01 (Hilton): BSTs 69

What do we return if it's not
found?

Note: can't be NULL,
May not be returning a pointer

One answer: make up a new one
and return it

My thought: worst possible design choice
But, this is what STL does in a lot of cases

Another answer: throw an exception
Exceptions

- Exceptions: similar to Java...
  - Except can throw anything (throw 22;)

- Unwinds call stack (returns from functions)
  - Until try(...) catch(...) (...) is found that can handle the exn

```cpp
try {
    someFun();
    anotherFun();
} catch(std::exception & e) {
    std::cerr << " Exception " << e.what() << endl;
}
```