Today

• Quiz 8
• BST erase
• Balanced BSTs

Assignments

• HW 7 out
Implementing erase for a BST

Some examples:

What should/could the tree look like after removing:

- Node a?
- Node e?
- Node b?
- Node d?

Removing nodes from a BST can be tricky! ... cases we need to consider:

- The tree is empty
- Item to be removed is not in the tree
- Node containing item to remove is a leaf
- Node containing item to remove has one child
- Node containing item to remove has two children
Removing a leaf node

- Only have to delete the node, set parent child pointer to `nullptr`

![Image of a tree with nodes a, b, and c, with node a removed and b as the new root.]

Removing a node with one child

- the parent of the node to be deleted “adopts” the child

![Image of a tree with nodes a, b, c, and d, with node a removed and b as the new root.]

Removing a node with two children

- This case is considerably more complicated
- Both children cannot be “adopted”

![Image of a tree with nodes a, b, c, and d, with node b removed and a and c as the new root.]

One Solution: Replace and delete replacement

“inorder” successor = next node in sort order (e.g., c is after b)

“inorder” predecessor = previous node in sort order (e.g., a is before b)

• we’ll replace contents of node to delete with inorder successor’s contents
• then delete the inorder successor

How do we find the inorder successor?

• go to right child
• then go left as far as possible

Note: The inorder successor will not have a left child

• which means it will have at most one (right) child
• which is an easier case for remove
Implementing erase

- Since we are just following a single path (even down to inorder successor)
- We can just use iteration (looping)

Instead, we’ll use a mix of recursion and iteration:

```cpp
// erase helper
Node* erase(const K key&, Node* st_root);
```

The basic idea ...

```cpp
if (key < st_root->key)
    st_root->left = erase(key, st_root->left);
else if (key > st_root->key)
    st_root->right = erase(key, st_root->right);
else if (key == st_root->key) {
    // case 1: left subtree is empty
    ...
    // case 2: right subtree is empty
    ...
    // case 3: inorder successor
    // use iteration to find, replace, delete inorder successor
    ...
}
return st_root;
```
More BST Concepts

In a "full" binary tree

- each root-to-leaf path has the same height $h$; and
- all internal nodes have two children

In a "complete" binary tree of height $h$

- the tree is full at height $h - 1$, and
- the leaf nodes are filled in from left to right
A balanced binary tree has for every node

- left and right subtrees that differ in height by at most 1

This is not a balanced binary tree

This is a balanced binary tree