Balanced BSTs and Heaps

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Balanced Trees

Your Objectives:

- Again, you already know about these! But...
  - Usually you just need to know the STL interface.
  - Harder problems will require augmented versions.
- Balanced Binary Trees
- Heaps

A rotation

```
0 template <class K, class V>
1 void AVLTree<K, V>::rotateLeft(Node*& t) {
2     Node* newSubRoot = t->right;
3     Node* temporary = newSubRoot->left;
4     newSubRoot->left = t;
5     t->right = temporary;
6     t = newSubRoot;
7     t->left->height
8         = std::max(heightOrNeg1(t->left->left),
9                    heightOrNeg1(t->left->right)) + 1;
10    t->height = std::max(heightOrNeg1(t->left),
11                         heightOrNeg1(t->right)) + 1;
12 }
```

BSTs

The classes:

- For existence, use `set`
- For Key-Value, use `map`

Common methods:

- `insert` vs `replace`
- `erase`, `clear`
- `operator[]` or `at`

Direct Access Table

- Trees can be faster (?) than hash tables.
- A Direct Access Table (DAT) is a hash table in which the unhashed object is the key.

```
0 char code[255];
1
2 for(char a='a'; a<'n'; a++) {
3    code[a] = a+13;
4    code[a+13] = a;
5 }
```

Priority Queue

- The `priority_queue` class is a max heap.
- Use also for partial sorting.
- Methods: `empty`, `size`, `push`, `top`, `pop`