CS106B
Spring 2016 Cynthia Lee

Section 6 (Week 7) – SOLUTION

Problem and solution authors include Marty Stepp, Jerry Cain and Cynthia Lee.

1. **removeLeaves.**

```
void BinaryTree::removeLeaves() {
    removeLeaves(root);
}

void BinaryTree::removeLeaves(TreeNode*& node) {
    if (node != NULL) {
        if (node->left == NULL && node->right == NULL) {
            delete node;
            node = NULL;
        } else {
            removeLeaves(node->left);
            removeLeaves(node->right);
        }
    }
}
```

2. **tighten.**

```
void BinaryTree::tighten() {
    tighten(root);
}

void BinaryTree::tighten(TreeNode*& node) {
    if (node != NULL) {
        tighten(node->left);
        tighten(node->right);
        if (node->left == NULL && node->right != NULL) {
            TreeNode* trash = node;
            node = node->right;
            delete trash;
        } else if (node->left != NULL && node->right == NULL) {
            TreeNode* trash = node;
            node = node->left;
            delete trash;
        }
    }
}
```

3. **Quadtrees**

```
static quadtree *gridToQuadtree(Grid<bool>& image,
                                int lowx, int highx, int lowy, int highy) {
   quadtree *qt = new quadtree;
   qt->lowx = lowx; qt->highx = highx - 1;
   qt->lowy = lowy; qt->highy = highy - 1;
   if (allPixelsAreTheSameColor(image, lowx, highx, lowy, highy)) {
      qt->isBlack = image[lowx][lowy];
      for (int i = 0; i < 4; i++) {
         qt->children[i] = NULL;
      }
   } else {
      int midx = lowx + ((highx - lowx) / 2);
      int midy = lowy + ((highy - lowy) / 2);
      qt->children[NW] = gridToQuadtree(image, lowx, midx, midy, highy);
      qt->children[NE] = gridToQuadtree(image, midx, highx, midy, highy);
      qt->children[SE] = gridToQuadtree(image, midx, highx, lowy, midy);
      qt->children[SW] = gridToQuadtree(image, lowx, midx, lowy, midy);
      // assume NW, NE, etc are constants/#defines
   }

   return qt;
}

static quadtree *gridToQuadtree(Grid<bool>& image) {
   return gridToQuadtree(image, 0, image.numCols(), 0, image.numRows());
}
```

(where `allPixelsAreTheSameColor` does exactly what it sounds like it does)

4. **Hashing (part 1).**

hash1 is valid, but not good because everything will get hashed to the same bucket.
hash2 is not valid, because "A" and "a" are equal, but will have different hash values.
hash3 is valid and good.
hash4 is not valid, because equal strings might not give the same hash value.

5. **Hashing (part 2).**

```
Bucket:                           After rehash:
0: baggage (30)                   0: null          6: baggage (30)
1: badcab (13)                    1: badcab (13)   7: null
2: feed (20) -> deadbeef (32)     2: null          8: feed (20) -> deadbeef (32)
3: cafe (15) -> cabbage (21)      3: cafe (15)     9: cabbage (21)
4: null                           4: null          10: null
5: null                           5: null          11: null
```