CSM 61B Fall 2020

Final Review

Mentoring 15: December 1, 2020

1 LLRB

1.1 Those darned Stanford Students are at it again! In an attempt to reverse-engineer your code, they stole your `get` and `put` methods! Fortunately, you understand red-black trees pretty well. Consider your now incomplete `RBTree` implementation.

```
public class RBTree<Key extends Comparable<Key>, Value> {
    private static final boolean RED   = true;
    private static final boolean BLACK = false;
    private Node root;     // root of the BST

    private class Node {
        private Key key;           // key
        private Value val;         // associated data
        private Node left, right;  // links to left and right subtrees
        private boolean color;     // color of parent link
        private int size;          // subtree count

        public Node(Key key, Value val, boolean color, int size) {
            this.key = key;
            this.val = val;
            this.color = color;
        }
    }

    public RBTree() {...}
    private Node rotateRight(Node h) {...}
    private Node rotateLeft(Node h){...}
    private void flipColors(Node h) {...}
    private boolean isRed(Node h) {...}  // returns false if node is black or null
    private int size(Node h) {...}

    public void put(Key key, Value val) {
        root = put(root, key, val);
        root.color = BLACK;
    }
```

```
private Value get(Key key) {
    Node runner = ________;
    while (____________) {
        int cmp = _______________;
        if (_______) {
            runner = __________;
        } else if (________) {
            runner = __________;
        } else {
            _________________;
        }
    }
    return null;
}
```

```
private Value get(Key key) {
   Node runner = root;
   while (runner != null) {
       int cmp = key.compareTo(runner.key);
       if (cmp < 0) {
           runner = runner.left;
       }
       else if (cmp > 0) {
           runner = runner.right;
       }
       else {
           return runner.val;
       }
   }
   return null;
```

Note how this is identical to how one would retrieve items in a regular binary search tree.

```
private Node put(Node h, Key key, Value val) {
    if (h == _____){
        return _____________;
    }
    int cmp = _____________;
    if  (_________) {
        h.left  = _____________;
    } else if (__________) {
        h.right = _____________;
    } else {
```

```
        h.val = _____;
    }
    if (_____________ && _____________)) {
        h = rotateLeft(h);
    }
    if (_____________  &&  _____________) {
        h = rotateRight(h);
    }
    if (_____________ && _____________) {
        _____________;
    }
    h.size = _____________;

    return h;
}

private Node put(Node h, Key key, Value val) {
    if (h == null){
        return new Node(key, val, RED, 1);
    }
    int cmp = key.compareTo(h.key);
    if  (cmp < 0) {
        h.left  = put(h.left,  key, val);
    } else if (cmp > 0) {
        h.right = put(h.right, key, val);
    } else {
        h.val = val;
    }
    if (isRed(h.right) && !isRed(h.left)) {
        h = rotateLeft(h);
    }
    if (isRed(h.left)  &&  isRed(h.left.left)) {
        h = rotateRight(h);
    }
    if (isRed(h.left)  &&  isRed(h.right)) {
        flipColors(h);
    }
    h.size = size(h.left) + size(h.right) + 1;

    return h;
}
```

Note how this simply sinks the current node down to its correct location

in the now-potentially unbalanced tree before proceeding to correct issues with the current link colorings. Go through the different cases with your section if they are unfamiliar.

2 Sorting

2.1 We have a list of points on the plane. Find the K closest points to the origin (0, 0). (Here, the distance between two points on a plane is the Euclidean distance.)

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in.)

Example
Input: points = [[3,3],[5,-1],[-2,4]], K = 2
Output: [[3,3],[-2,4]]
(The answer [[-2,4],[3,3]] would also be accepted.)

This problem is taken from Leetcode: https://leetcode.com/problems/k-closest-points-to-origin/

Intuition

Sort the points by distance, then take the closest K points.

Algorithm

In Java, we find the K-th distance by creating an array of distances and then sorting them. After, we select all the points with distance less than or equal to this K-th distance.

```
class Solution {
    public int[][] kClosest(int[][] points, int K) {
        int N = points.length;
        int[] dists = new int[N];
        for (int i = 0; i < N; ++i)
            dists[i] = dist(points[i]);

        Arrays.sort(dists);
        int distK = dists[K-1];

        int[][] ans = new int[K][2];
        int t = 0;
        for (int i = 0; i < N; ++i)
            if (dist(points[i]) <= distK)
                ans[t++] = points[i];
        return ans;
    }
```

```
    public int dist(int[] point) {
        return point[0] * point[0] + point[1] * point[1];
    }
}
```

3 Tries

3.1 Given a dictionary of n prefix strings and a sentence, describe a procedure to replace each word in the sentence with the shortest prefix string that can be found in the dictionary. If no such prefix string exists in the dictionary, leave the word as is.

For example, say we have the following prefix strings in our dictionary: `"bab"`, `"bat"`, `"rac"`, and `"racco"` and the sentence, `"the baboons battled the raccoon"`. Our output should be `"the bab bat the rac"`

We can put all the prefix strings in the dictionary into a trie. Then for every word in the given sentence, start searching for the word in the trie. Find the first matching prefix and replace it with the given word from our sentence. For example, if we are searching for the word `"raccoon"` in a trie made up of the prefix strings, `"rac"` and `"racco"`, we traverse down the trie on the letters `"r"`, `"a"`, `"c"`. Our trie indicates that `"rac"` is a word in the trie, so we stop searching and replace `"rac"` with `"raccoon"`. We do not want to continue traversing the trie because the first matching prefix we find will be our shortest prefix string.

4 Graphs

4.1 Briefly describe an efficient algorithm and the runtime for finding a minimum spanning tree in an undirected, connected graph $G = (V, E)$ when the edge weights satisfy:

(a) For all $e \in E$, $w_e = 1$. (All edge weights are 1.)

The key idea here is that any tree which connects all nodes is an MST. We can run DFS and take the DFS tree. You could also take a BFS tree, or run Prim's algorithm with a queue or stack instead of a priority queue (this would be equivalent to BFS/DFS). The runtime of this algorithm is in $\Theta(|V| + |E|) \in \Theta(|E|)$ for simple graphs.

(b) For all $e \in E$, $w_e \in \{1, 2\}$. (All edge weights are either 1 or 2.)

Run Prim's algorithm with a specialized priority queue, comprised of 2 regular queues. When we add items to the priority queue, we add it to the first queue if the weight is 1 and otherwise add it to the second queue (the weight must be 2). When we pop from the priority queue, we take from the first queue, unless it is empty, in which case we take from the second. The runtime of this algorithm is in $\Theta(|E|)$ as priority queue operations are now constant time.

5 Weighted QuickUnion

5.1 Describe how to construct a `WeightedQuickUnionUF` tree of maximum height.

To construct a max-height tree of N nodes, union two max-height trees with $\frac{N}{2}$ nodes each. The height of the final tree is 1 greater than each of the max-height $\frac{N}{2}$-node trees.