Red-Black Tree

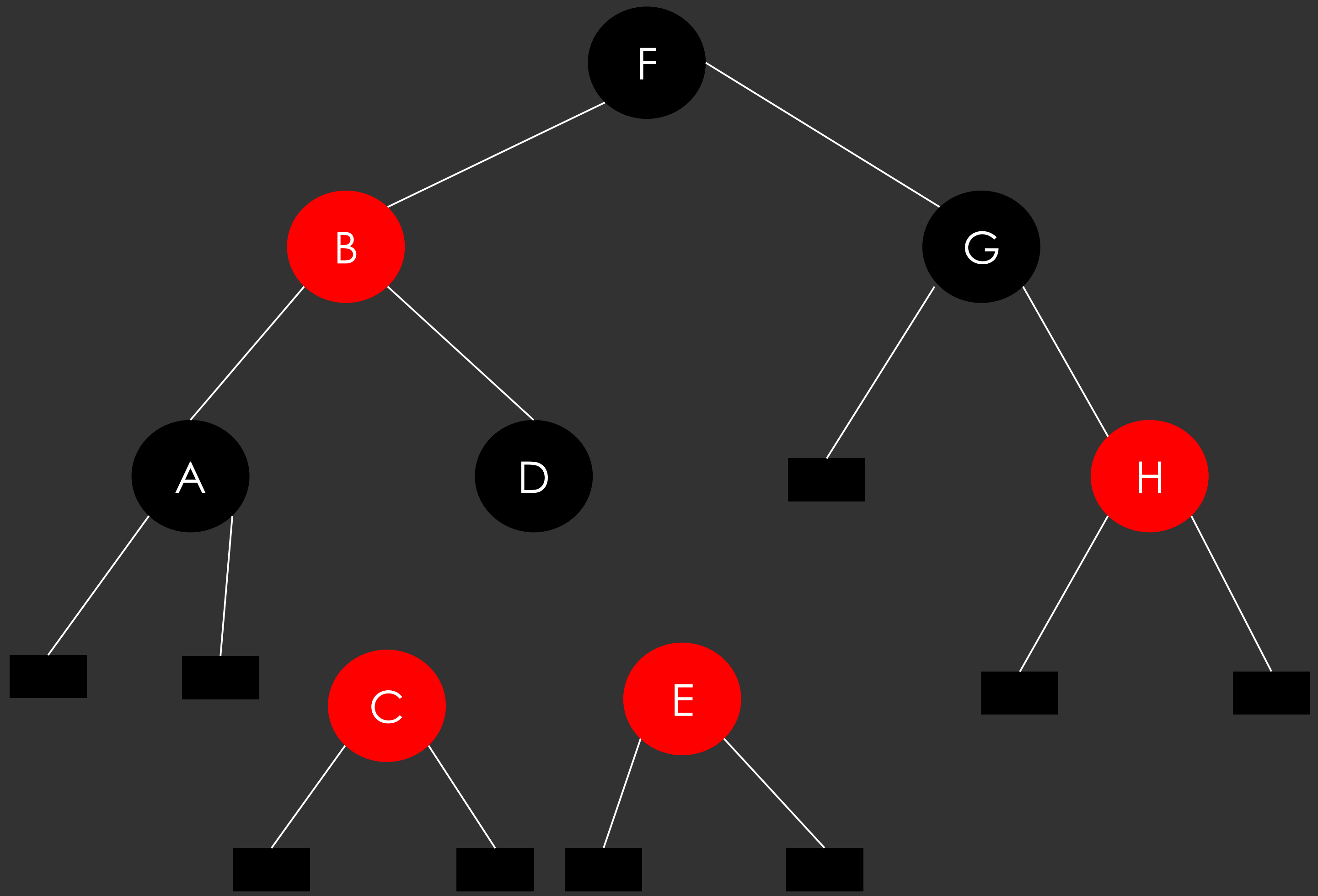
Alejandro Hahn
Hugo Valencia
Yessica Hernandez

¿WHAT IT IS?

- A Red-Black tree is a biniary search tree with an extra node attribute, the node color, which can be either RED or BLACK.
- The colors, following Red-Black principles, ensure that the longesth path from the root to a last element is not larger tan the double of the shortest one. This means that this tree is strongly balanced.

TREE PROPIERTIES

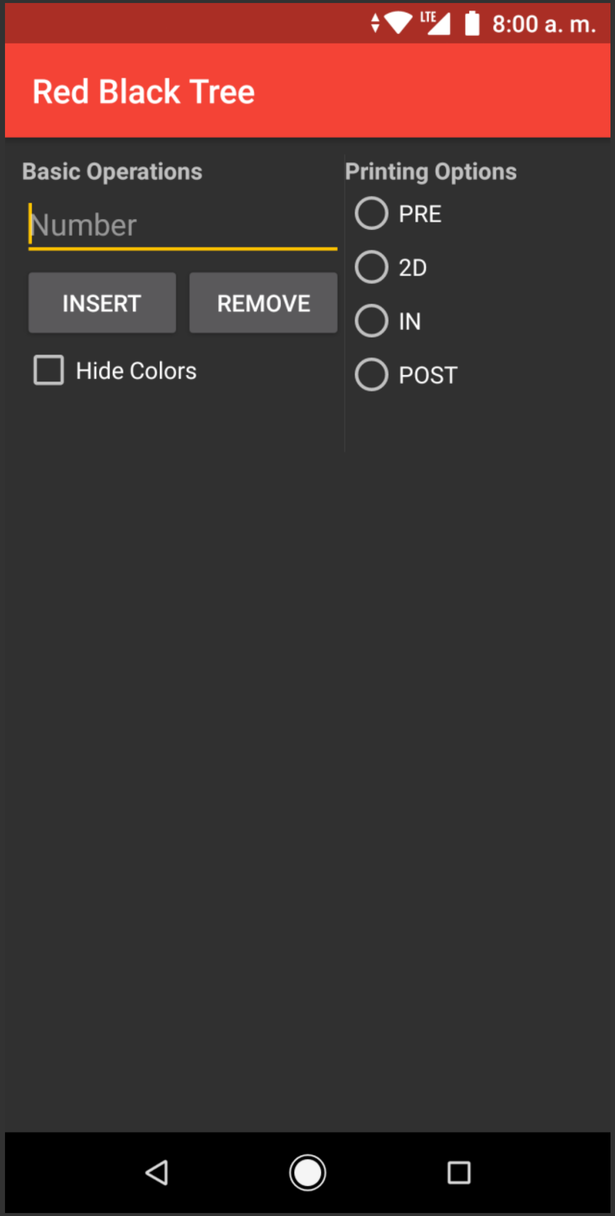
1. Every node is either red or black
2. The root is black
3. Every leaf (null) is black
4. If a node is red, then both childrens are black
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes



STRUCTURE COMPLEXITY

	Complexity
Space	O(n)
Search	O(log n)
Insert	O(log n)
Delete	O(log n)
Height	2 log(n+1)

THE APP



8:00 a. m.

Red Black Tree

Basic Operations

Number

INSERT REMOVE

□ Hide Colors

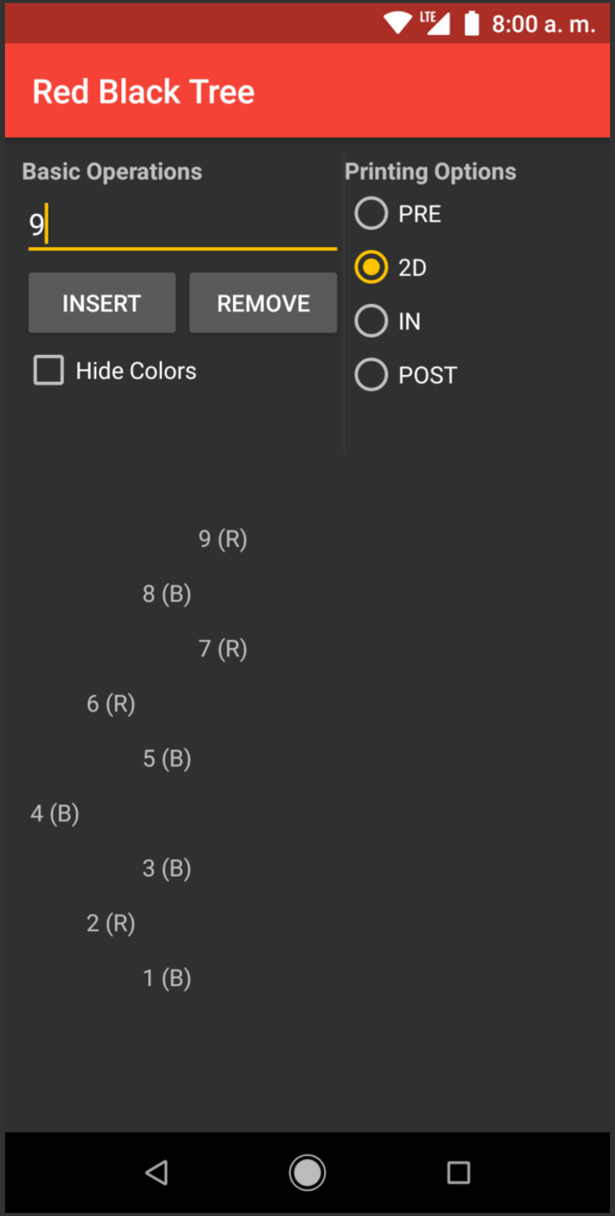
Printing Options

○ PRE

○ 2D

○ IN

○ POST



8:00 a. m.

Red Black Tree

Basic Operations

9

INSERT REMOVE

□ Hide Colors

Printing Options

○ PRE

● 2D

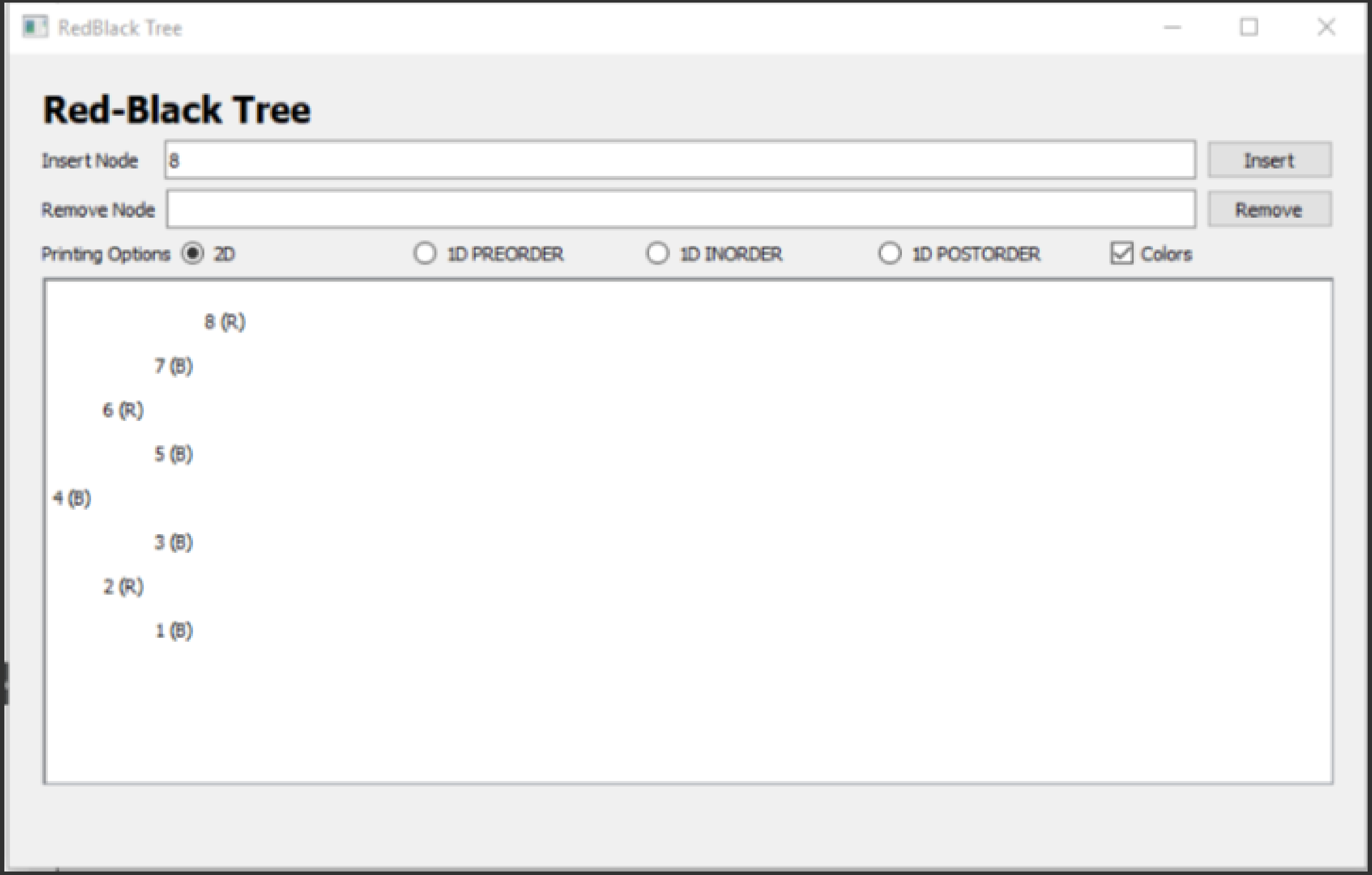
○ IN

○ POST

```
                9 (R)
            8 (B)
                7 (R)
        6 (R)
            5 (B)
    4 (B)
            3 (B)
        2 (R)
            1 (B)
```

GET IT ON
Google Play

THE APP



Red-Black Tree

Insert Node: 8

Remove Node:

Printing Options ● 2D ○ 1D PREORDER ○ 1D INORDER ○ 1D POSTORDER ☑ Colors

```
               8 (R)
          7 (B)
     6 (R)
          5 (B)
4 (B)
          3 (B)
     2 (R)
          1 (B)
```

THE CODE

```
class RedBlackNode {

    T value = {};
    Color color = BLACK;
    RedBlackNode<T>* left = nullptr;
    RedBlackNode<T>* right = nullptr;
    RedBlackNode<T>* parent = nullptr;

    static void RotateLeft(RedBlackTree<T>* t, RedBlackNode<T>* x) {}

    static void RotateRight(RedBlackTree<T>* t, RedBlackNode<T>* x) {}

};
```

THE CODE

```
class RedBlackTree {

public:
    RedBlackNode<T>* root = nullptr;

    void Insert(T value) {}

    void Delete(RedBlackNode<T>* x) {}

    void Delete(T value) {}

    RedBlackNode<T>* Search(T key) {}

    int Size() {}

    int Height() {}

    virtual std::vector<RedBlackNode<T>*> ToList(WalkOrder x) {}

    virtual void Print(WalkOrder x, bool c) {}

    virtual void Print(WalkOrder x, RedBlackNode<T>* n, bool c) {}

    virtual void PrintDefault(bool c) {}

    virtual void PrintDefault(RedBlackNode<T>* n, bool c) {}

    virtual void Print2D(bool c){}

    virtual void Print2D(RedBlackNode<T>* n, bool c) {}

    void InsertNode(RedBlackNode<T>* z) {}

    void DeleteNode(RedBlackNode<T>* z) {}

    void DeleteNodeV2(RedBlackNode<T>* z) {}

    void InsertRepair(RedBlackNode<T>* z) {}

    void DeleteRepair(RedBlackNode<T>* x) {}

    void Swap(RedBlackNode<T>* x, RedBlackNode<T>* y) {}

    RedBlackNode<T>* Minimum(RedBlackNode<T>* x) {}

    int HeightRecursive(RedBlackNode<T>* x) {}

    int SizeRecursive(RedBlackNode<T>* n) {}

    void ToListPreOrder(RedBlackNode<T>* n, std::vector<RedBlackNode<T>*>* list) {}

    void ToListInOrder(RedBlackNode<T>* n, std::vector<RedBlackNode<T>*>* list) {}

    void ToListPostOrder(RedBlackNode<T>* n, std::vector<RedBlackNode<T>*>* list) {}

    virtual void PrintPreOrder(RedBlackNode<T>* n, bool c) {}

    virtual void PrintInOrder(RedBlackNode<T>* n, bool c) {}

    virtual void PrintPostOrder(RedBlackNode<T>* n, bool c) {}

    virtual void Print2DUtil(RedBlackNode<T>* root, int space, bool c) {}
};
```

THE PROBLEMS

- Logic failure at books at which it try to access propierties of a null pointer
- We has to learn how to use Android NDK and QT

THAT’ S ALL

Questions? Comments?