# Assignment 3

## File structure

> Files in ***Bold and Italic*** are the test class of assignment questions.

- src
    - assignment_3
        - question_1
            - RedBlackTree.java
            - MinGapTree.java
            - ***Question1_Test.java***
        - question_2_3
            - GraphADT.java
            - GraphUtil.java
            - ***Question2_Test.java***
            - ***Question3_Test.java***

## Question 1

> You can test via ***Question1_Test.java*** .

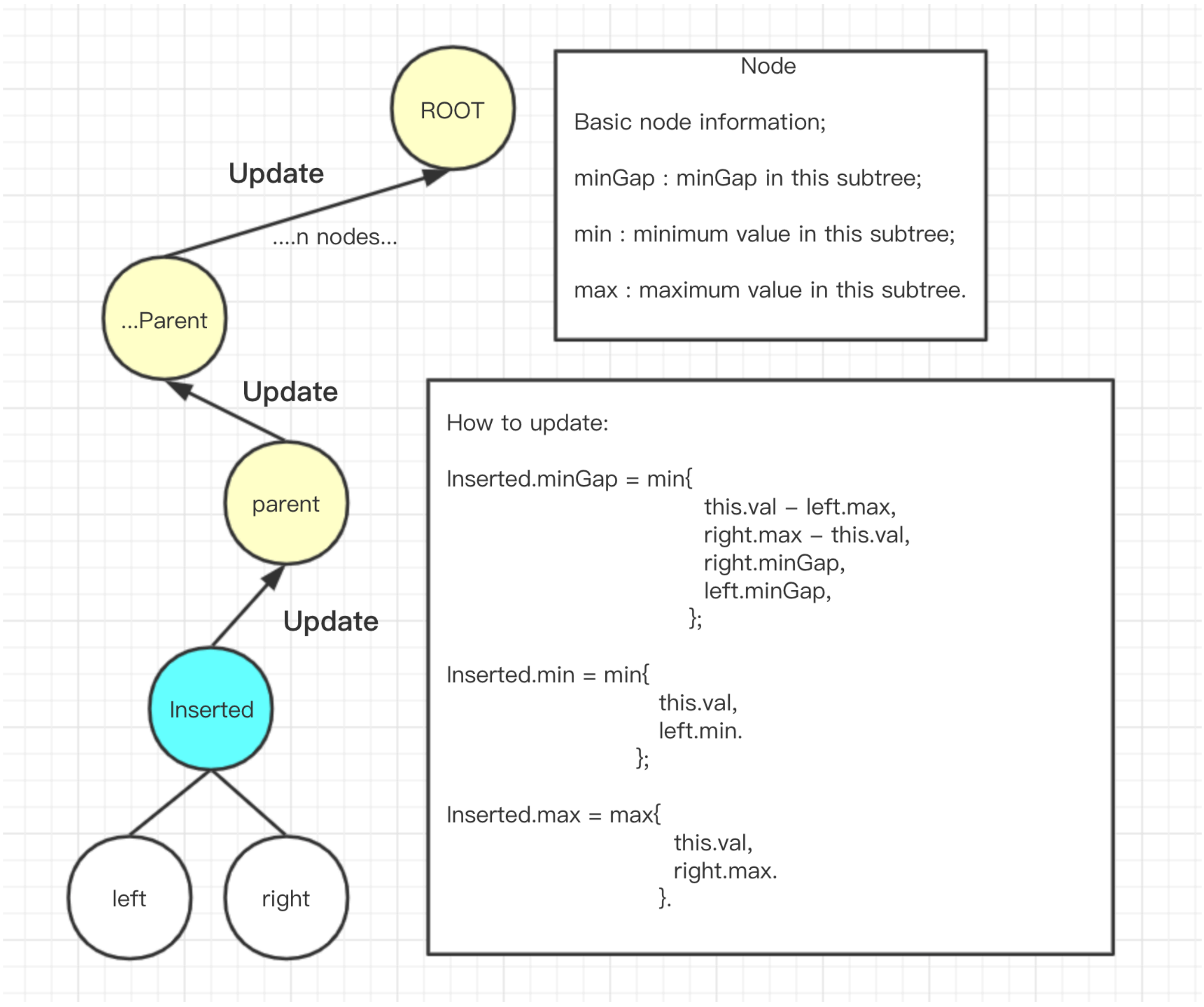
- RedBlackTree.java
    - class RedBlackTree
    - class Node
- MinGapTree.java
    - class MinGapTree **extends** RedBlackTree
    - class MinGapNode **extends** Node
- ***Question1_Test.java***

### Augmented DS

- MinGapNode
    - int minGap
    - int min
    - int max

### How it works

- Step.1 Update information after the insertion



Node

Basic node information;

minGap : minGap in this subtree;

min : minimum value in this subtree;

max : maximum value in this subtree.

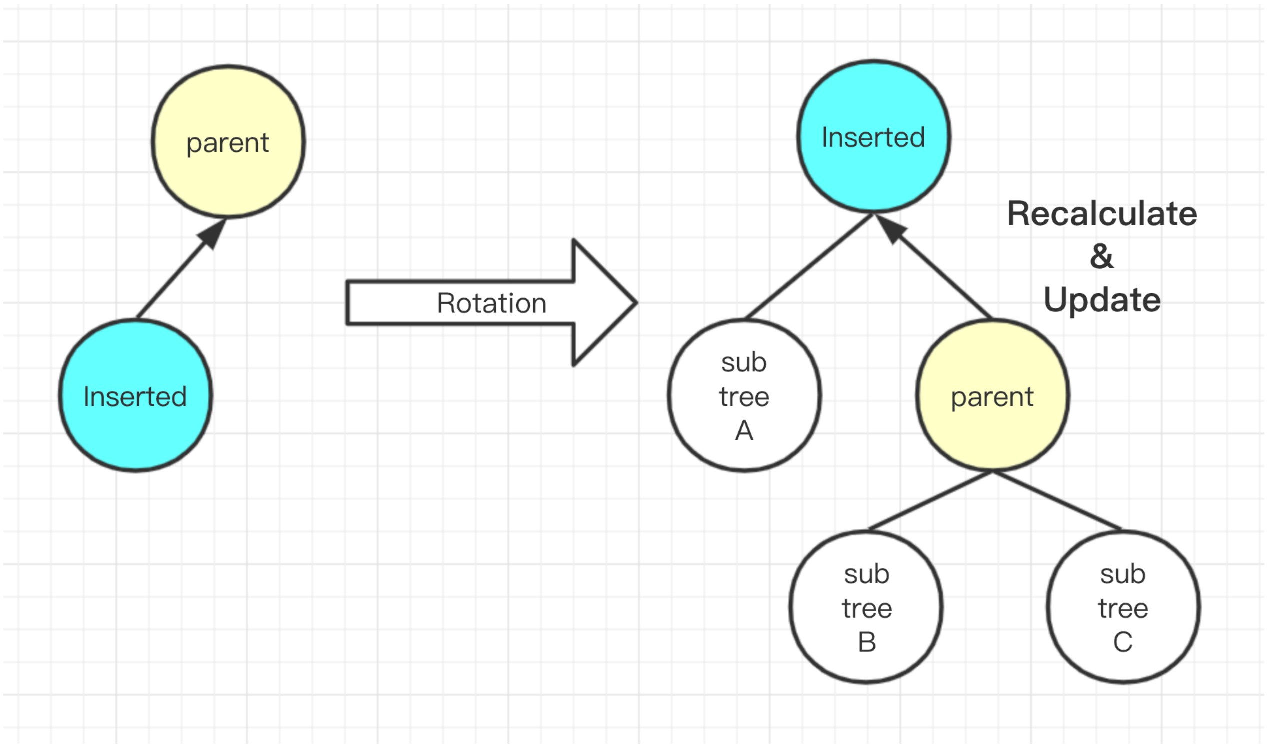
```
How to update:

Inserted.minGap = min{
                        this.val – left.max,
                        right.max – this.val,
                        right.minGap,
                        left.minGap,
                      };

Inserted.min = min{
                    this.val,
                    left.min.
                  };

Inserted.max = max{
                     this.val,
                     right.max.
                   }.
```

- Step.2 Update information between the Inserted Node and its parent after rotation.



- Step.3 (Finally) You can get MinGap from the root node.

## Question 2

> You can test via *Question2_Test.java* .

- GraphADT.java
    - class GraphADT
    - class Vertex
- GraphUtil.java
    - class GraphUtil -- **calculateDepths()**
    - class Info
        - Vertex vertex
        - int d
        - int m
- *Question2_Test.java*

### How it works (briefly)

- Step.1 Prepare a Info graph<Info<E>> that the Info<E> holds reference to Vertex<E> from the original graph.
    - This is an instruction from question description.
- Step.2
    - 2.1 DFS the original graph, and save the d(v) to each Info.d and m(v) = d(v)++ to info.m;
    - 2.2 Update the each info.m after a DFS() recursion returned.
- Done. (Details in code.)
- Extra:
    - The Info graph are constructed directed, so that it can easily become a tree for Question.3 usage.
    - **Original Graph is undirected.**

### Comments to Question 2

- Frankly, it is quite weird to let **Info** *hold the reference of* **Vertices of a Graph** and put **Info** into another **Graph**,

  which result in:

    - **Graph<Info<E>>** has a set of **Vertices** contains **Info<E>**
    - **Info<E>** actually contains a **Vertex<E>** from another **Graph<E>**
    - and the **Graph<E>** originally contains **Vertex<E>**
    - Extremelly complex and confusing... 😂 If I am asked to do this again, I would augment **Vertex**, let **Info** *extends* **Vertex**, instead.

## Question 3

> You can test this quesiton answer via ***Question3_Test.java*** .

- GraphADT.java
    - class GraphADT
    - class Vertex
- GraphUtil.java
    - class GraphUtil
    - class Info
    - class AnalyzedResult
- ***Question3_Test.java***

### How it works

- Step.1
    - Make sure original graph is undirected.
    - Make sure original graph is not empty.
    - Make sure original graph is connected.
- Step.2
    - Call the calculateDepths() to get the Info Graph which contains **d**, **m** and **deep first tree information**.
- Step.3 Find bridge and articulation point and save them to AnalyzedResult.
    - Algorithm based on Question 3 description.
    - And if the root has more than 1 subtree, it is an articulation point.
- Done. Return AnalytzedResult.