CS 700-34 **QuadTree**

Student: Shuhua Song

Due Date:
Soft Copy: 03/10/2020
Hard Copy: 03/12/2020

Submission Data:
Soft Copy: 03/10/2020
Hard Copy: 03/12/2020

Algorithm Steps:

I. Algorithm Steps for computeSquare()

```
Step 0: Given numRows, numCols
Step1: get the maximum value max from numRows and numCols,
        Set k = 1
Step2: if k < max, k ← k*2
Step3: repeat step 2 until k>=max
Step4: return k
```

II. Algorithm Steps for buildQTree

```
Step0: given upperR, upperC, size
Step1: create newQTnode←get a new Qtree node
        newNode.upperR= upperR
        newNode.upperC = upperC
        newNode.size = size
Step2: (recursion)
        If(size==1){
          newNode.color ←imgAry[upperR][upperC]
        }
        else{
          halfSize ← size/2
          newQtNode.NWkid ← buildQuadTree(imgAry, upR, upC, halfSize)
          newQtNode.NEkid ←  buildQuadTree(imgAry, upR, upC+halfSize, halfSize)
          newQtNode.SWkid ←buildQuadTree(imgAry, upR+halfSize, upC, halfSize)
          newQtNode.SWkid ←buildQuadTree(imgAry, upR+halfSize, upC+halfSize, halfSize)
          sumColor←sum of all the four kids' colors
          if sumColor == 0
               newQtNode's color ← 0
               set all newQtNode's four kids to null
               //let newQtNode as a leaf node and its color is 0
          else if sumColor == 4
               newQtNode's color ← 1
               set all newQtNode's four kids to null
               //let newQtNode as a leaf node and its color is 1
          else
               newQtNode's ← 5
                //let newQtNode as a non-leaf node and keeps 4 kids
          Step 3: return newQtNode
```

Code:

Image Class

```java
import java.io.*;
import java.util.Scanner;

public class Image {
    private static int numRows;
    private static int numCols;
    private static int minVal;
    private static int maxVal;

    int squareSize;
    int[][] imgAry;
    //Scanner inFile;
    Image(){
     numRows = 0;
     numCols = 0;
     minVal = 0;
     maxVal = 0;
     squareSize=0;
     imgAry = new int[numRows][numCols];
    }

    Image(int numRows, int numCols, int minval, int maxVal, int squareSize, int[][]
imgAry){
        this.numRows = numRows;
        this.numCols = numCols;
        this.minVal = minVal;
        this.maxVal = maxVal;
        this.squareSize = squareSize;
        this.imgAry = imgAry;
    }

    public static int computeSquare(int numRows, int numCols) {
        int square = Math.max(numRows, numCols);
        int power2 = 2;
        while(power2 < square){
            power2 = power2 * 2;
        }
        return power2;
    }

    public static void loadImage(int[][] imgAry, Scanner inFile){

        for(int i=0; i<numRows; i++){
            for(int j=0; j<numCols; j++){
                imgAry[i][j] = inFile.nextInt();
               // System.out.println(imgAry[i][j]);
            }
           // System.out.println();
        }
    }

    public void zero2DAry(int[][] imgAry){
      // squareSize = computeSquare(numRows, numCols);
       //imgAry = new int[squareSize][squareSize];
       for(int i=0; i<squareSize; i++){
           for(int j=0; j<squareSize; j++){
               imgAry[i][j] = 0;
           }
       }
    }

    public static void main(String[] args) throws IOException {
        Scanner inFile = new Scanner(new FileReader(args[0]));
        BufferedWriter outFile1 = new BufferedWriter(new FileWriter(new 
File(args[1])));
        BufferedWriter outFile2 = new BufferedWriter(new FileWriter(new 
File(args[2])));

       // int numRows, numCols, minVal, maxVal;
        numRows = inFile.nextInt();
        numCols = inFile.nextInt();
        minVal = inFile.nextInt();
        maxVal = inFile.nextInt();
        //System.out.println("numRows: " + numRows + ", numCols: " + numCols + ", 
minVal: " + minVal + ", maxVal: " + maxVal)

       // Image image = new Image(numRows, numCols, minVal, maxVal, 0, imgAry);
        int squareSize = computeSquare(numRows, numCols);
       // System.out.println("squareSize: " + squareSize);
        int[][] imgAry = new int[squareSize][squareSize];
        //image.zero2DAry(imgAry);
       /* for(int i=0; i<squareSize; i++){
            for(int j=0; j<squareSize; j++){
                System.out.println(imgAry[i][j]);
            }
        }

        */

        //load image
        loadImage(imgAry, inFile);


            QuadTree Qtree = new QuadTree();
            QtTreeNode qtRoot = Qtree.buildQuadTree(imgAry, 0, 0, squareSize);

            outFile1.write("PreOrder: \n");
            Qtree.preOrderTraversal(qtRoot, outFile1);
            outFile1.write("PostOrder: \n");
            Qtree.postOrderTraversal(qtRoot, outFile1);

        inFile.close();
        outFile1.close();
        outFile2.close();
    }

}
```

QtTreeNode Class

```
import java.io.BufferedWriter;
import java.io.IOException;

public class QtTreeNode {

   int color; //  0/1/2
   int upperR;
   int upperC;
   int squareSize;

   QtTreeNode NWkid;
   QtTreeNode NEkid;
   QtTreeNode SWkid;
   QtTreeNode SEkid;

   QtTreeNode(){
//      color = 1;
//      upperR = 0;
//      upperC = 0;
//
//      squareSize = 1; //?????
//      NWkid = null;
//      NEkid = null;
//      SWkid = null;
//      SEkid = null;
   }

   QtTreeNode(int color, int upperR, int upperC,  int squareSize, QtTreeNode NWkid,
QtTreeNode NEkid, QtTreeNode SWkid, QtTreeNode SEkid){
       this.color = color;
       this.upperR = upperR;
       this.upperC = upperC;
       this.squareSize = squareSize;
       this.NWkid = NWkid;
       this.NEkid = NEkid;
       this.SWkid = SWkid;
       this.SEkid = SEkid;
   }

   public void printQtNode(QtTreeNode node, BufferedWriter outFile) throws IOException
{
       //outFile.write("Nodes Output: \n");
       if(node.NWkid!=null && node.NEkid != null && node.SWkid != null &&
node.SEkid != null){
           outFile.write(node.color + " " + node.upperR +" " + node.upperC + " " +
node.NWkid.color + " "
                   + node.NEkid.color + " " + node.SWkid.color + " " +
node.SEkid.color + "\n");
       }else{
           outFile.write(node.color + " " + node.upperR +" " + node.upperC + " " +
null + " " + null + " "+ null + " " + null +"\n");
       }
   }

}
```

QuadTree class

```
import java.io.*;
import java.util.Scanner;

public class QuadTree {
    QtTreeNode Qtroot;
    //Image image;

    QuadTree(){
        Qtroot = new QtTreeNode();
    }


    public QtTreeNode buildQuadTree(int[][] imgAry, int upR, int upC, int size) {
        QtTreeNode newQtNode = new QtTreeNode(-1, upR, upC, size, null, null, null,
null);
        if (size == 1) {
            newQtNode.color = imgAry[upR][upC];//1 or 0
            //System.out.println("( " + upR + ", " + upC + " )\n");
        } else {
            int halfSize = size/2;
            newQtNode.NWkid = buildQuadTree(imgAry, upR, upC, halfSize);
            newQtNode.NEkid = buildQuadTree(imgAry, upR, upC+halfSize, halfSize);
            newQtNode.SWkid = buildQuadTree(imgAry, upR+halfSize, upC, halfSize);
            newQtNode.SEkid = buildQuadTree(imgAry, upR+halfSize, upC+halfSize,
halfSize);
            int sumColor = newQtNode.NWkid.color + newQtNode.NEkid.color +
newQtNode.SWkid.color + newQtNode.SEkid.color;
            if (sumColor == 0) {
                newQtNode.color = 0;
                newQtNode.NWkid = null;
                newQtNode.NEkid = null;
                newQtNode.SWkid = null;
                newQtNode.SEkid = null;
            } else if (sumColor == 4) {
                newQtNode.color = 1;
                newQtNode.NWkid = null;
                newQtNode.NEkid = null;
                newQtNode.SWkid = null;
                newQtNode.SEkid = null;
            } else {
                newQtNode.color = 5;
            }
        }
        return newQtNode;
    }

    public void preOrderTraversal(QtTreeNode Qt, BufferedWriter outFile) throws
IOException {
        if (Qt.SWkid == null & Qt.NEkid == null && Qt.SWkid == null && Qt.SEkid ==
null) {
            Qt.printQtNode(Qtroot, outFile);
        } else {
            Qt.printQtNode(Qt, outFile);
            preOrderTraversal(Qt.NWkid, outFile);
            preOrderTraversal(Qt.NEkid, outFile);
            preOrderTraversal(Qt.SWkid, outFile);
            preOrderTraversal(Qt.SEkid, outFile);
        }
    }

    public void postOrderTraversal(QtTreeNode Qt, BufferedWriter outFile) throws
IOException {
        if (Qt.SWkid == null & Qt.NEkid == null && Qt.SWkid == null && Qt.SEkid ==
null) {
            Qt.printQtNode(Qt, outFile);
            //return;
        } else {

            postOrderTraversal(Qt.NWkid, outFile);
            postOrderTraversal(Qt.NEkid, outFile);
            postOrderTraversal(Qt.SWkid, outFile);
            postOrderTraversal(Qt.SEkid, outFile);
            Qt.printQtNode(Qt, outFile);
        }
    }

}
```

SquareOutput

```
PreOrder:
5 0 0 5 5 5 5
5 0 0 0 5 5 5
0 0 0 null null null null
5 0 16 0 0 5 1
0 0 0 null null null null
0 0 0 null null null null
5 8 16 0 0 0 5
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 12 20 0 1 0 1
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 16 0 5 1 0 0
5 16 0 0 1 0 1
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
```

```
0 0 0 null null null null
5 16 16 5 0 0 5
5 16 16 1 0 1 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 24 24 5 5 1 1
5 24 24 0 0 1 1
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 24 28 0 0 1 1
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 0 32 0 5 5 0
0 0 0 null null null null
5 0 48 0 1 1 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 16 32 0 0 5 0
0 0 0 null null null null
0 0 0 null null null null
5 24 32 5 5 1 1
5 24 32 0 0 1 1
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 24 36 0 0 1 1
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
```

```
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 32 0 0 5 0 0
0 0 0 null null null null
5 32 16 0 1 0 5
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 40 24 5 5 0 0
5 40 24 1 1 0 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 40 28 1 1 0 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 32 32 5 0 0 0
5 32 32 1 0 5 0
0 0 0 null null null null
0 0 0 null null null null
5 40 32 5 5 0 0
5 40 32 1 1 0 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 40 36 1 1 0 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
```

```
0 0 0 null null null null
PostOrder:
0 0 0 null null null null
0 0 16 null null null null
0 0 24 null null null null
0 8 16 null null null null
0 8 20 null null null null
0 12 16 null null null null
0 12 20 null null null null
1 12 22 null null null null
0 14 20 null null null null
1 14 22 null null null null
5 12 20 0 1 0 1
5 8 16 0 0 0 5
1 8 24 null null null null
5 0 16 0 0 5 1
0 16 0 null null null null
1 16 4 null null null null
0 20 0 null null null null
1 20 4 null null null null
5 16 0 0 1 0 1
1 16 8 null null null null
0 24 0 null null null null
0 24 8 null null null null
5 16 0 5 1 0 0
1 16 16 null null null null
0 16 20 null null null null
1 20 16 null null null null
0 20 20 null null null null
5 16 16 1 0 1 0
0 16 24 null null null null
0 24 16 null null null null
0 24 24 null null null null
0 24 26 null null null null
1 26 24 null null null null
1 26 26 null null null null
5 24 24 0 0 1 1
0 24 28 null null null null
0 24 30 null null null null
1 26 28 null null null null
1 26 30 null null null null
5 24 28 0 0 1 1
1 28 24 null null null null
1 28 28 null null null null
```

```
5 24 24 5 5 1 1
5 16 16 5 0 0 5
5 0 0 0 5 5 5
0 0 32 null null null null
0 0 48 null null null null
1 0 56 null null null null
1 8 48 null null null null
0 8 56 null null null null
5 0 48 0 1 1 0
0 16 32 null null null null
0 16 40 null null null null
0 24 32 null null null null
0 24 34 null null null null
1 26 32 null null null null
1 26 34 null null null null
5 24 32 0 0 1 1
0 24 36 null null null null
0 24 38 null null null null
1 26 36 null null null null
1 26 38 null null null null
5 24 36 0 0 1 1
1 28 32 null null null null
1 28 36 null null null null
5 24 32 5 5 1 1
0 24 40 null null null null
5 16 32 0 0 5 0
0 16 48 null null null null
5 0 32 0 5 5 0
0 32 0 null null null null
0 32 16 null null null null
1 32 24 null null null null
0 40 16 null null null null
1 40 24 null null null null
1 40 26 null null null null
0 42 24 null null null null
0 42 26 null null null null
5 40 24 1 1 0 0
1 40 28 null null null null
1 40 30 null null null null
0 42 28 null null null null
0 42 30 null null null null
5 40 28 1 1 0 0
0 44 24 null null null null
0 44 28 null null null null
```

```
5 40 24 5 5 0 0
5 32 16 0 1 0 5
0 48 0 null null null null
0 48 16 null null null null
5 32 0 0 5 0 0
1 32 32 null null null null
0 32 40 null null null null
1 40 32 null null null null
1 40 34 null null null null
0 42 32 null null null null
0 42 34 null null null null
5 40 32 1 1 0 0
1 40 36 null null null null
1 40 38 null null null null
0 42 36 null null null null
0 42 38 null null null null
5 40 36 1 1 0 0
0 44 32 null null null null
0 44 36 null null null null
5 40 32 5 5 0 0
0 40 40 null null null null
5 32 32 1 0 5 0
0 32 48 null null null null
0 48 32 null null null null
0 48 48 null null null null
5 32 32 5 0 0 0
5 0 0 5 5 5 5
```

Not-Square Output

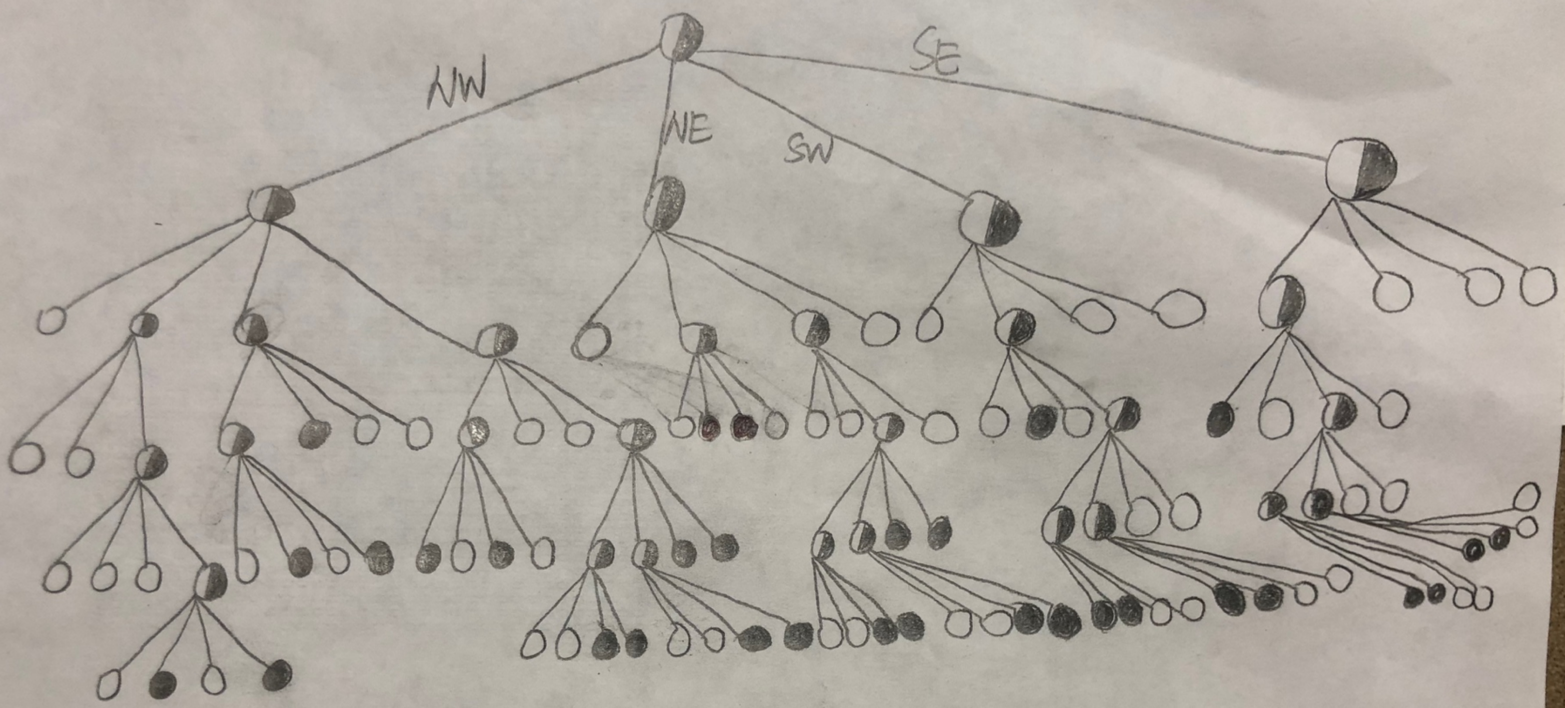
```
PreOrder:
5 0 0 5 5 0 0
5 0 0 0 0 0 5
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 16 16 5 5 5 5
5 16 16 0 0 0 1
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 16 24 0 0 5 1
```

```
0 0 0 null null null null
0 0 0 null null null null
5 20 24 1 1 5 5
0 0 0 null null null null
0 0 0 null null null null
5 22 24 1 1 0 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 22 26 1 1 0 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 24 16 0 1 0 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 24 24 0 1 0 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 0 32 0 0 5 0
0 0 0 null null null null
0 0 0 null null null null
5 16 32 5 0 5 0
5 16 32 1 0 1 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
5 24 32 1 0 0 0
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
0 0 0 null null null null
```

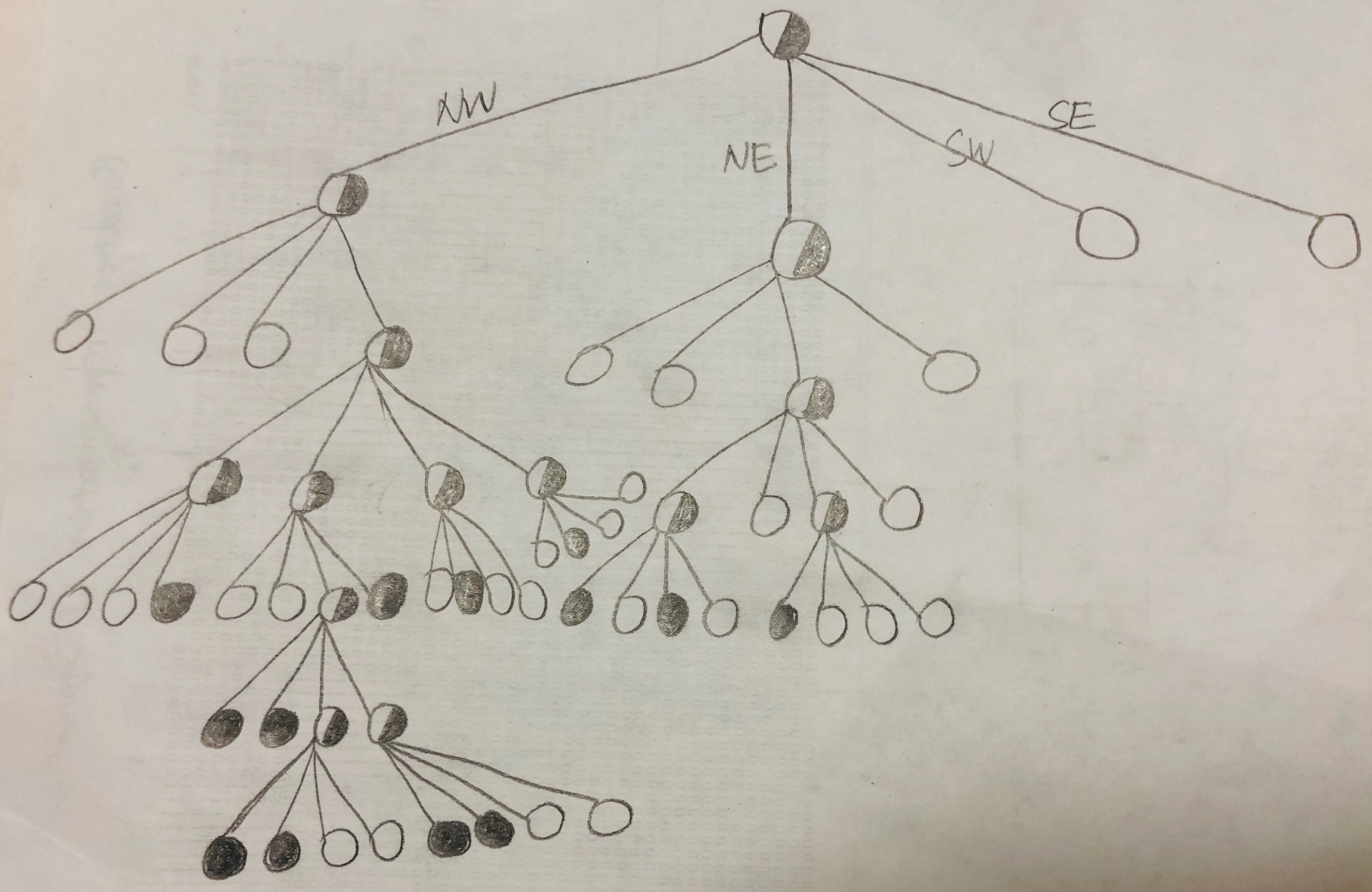
```
0 0 0 null null null null
PostOrder:
0 0 0 null null null null
0 0 16 null null null null
0 16 0 null null null null
0 16 16 null null null null
0 16 20 null null null null
0 20 16 null null null null
1 20 20 null null null null
5 16 16 0 0 0 1
0 16 24 null null null null
0 16 28 null null null null
1 20 24 null null null null
1 20 26 null null null null
1 22 24 null null null null
1 22 25 null null null null
0 23 24 null null null null
0 23 25 null null null null
5 22 24 1 1 0 0
1 22 26 null null null null
1 22 27 null null null null
0 23 26 null null null null
0 23 27 null null null null
5 22 26 1 1 0 0
5 20 24 1 1 5 5
1 20 28 null null null null
5 16 24 0 0 5 1
0 24 16 null null null null
1 24 20 null null null null
0 28 16 null null null null
0 28 20 null null null null
5 24 16 0 1 0 0
0 24 24 null null null null
1 24 28 null null null null
0 28 24 null null null null
0 28 28 null null null null
5 24 24 0 1 0 0
5 16 16 5 5 5 5
5 0 0 0 0 0 5
0 0 32 null null null null
0 0 48 null null null null
1 16 32 null null null null
0 16 36 null null null null
1 20 32 null null null null
```

```
0 20 36 null null null null
5 16 32 1 0 1 0
0 16 40 null null null null
1 24 32 null null null null
0 24 36 null null null null
0 28 32 null null null null
0 28 36 null null null null
5 24 32 1 0 0 0
0 24 40 null null null null
5 16 32 5 0 5 0
0 16 48 null null null null
5 0 32 0 0 5 0
0 32 0 null null null null
0 32 32 null null null null
5 0 0 5 5 0 0
```

QuadTree:



QuadTree representation of SquareData



QuadTree representation of non-Square Data