http://algs4.cs.princeton.edu/lectures/35SearchingApplications-2x2.pdf

```
public class Board {
    private static final int BLANK = 0;   // the blank square

    private final int N;            // the board size
    private int manhattan = -1;     // cache the Manhattan distance
    private final int[] tiles;

// string representation
    public String toString() {
        StringBuilder s = new StringBuilder();
        s.append(N + "\n");
        for (int k = 0; k < N*N; k++)
            s.append(String.format("%2d ", tiles[k]));
        s.append("\n");
        return s.toString();
    }

}
```

```
private ST<String, SET<Integer>> map = new ST<String, SET<Integer>>();
private ST<Integer, String> synsets = new ST<Integer, String>();
    private void buildMap(String synsetFile) {
        In in = new In(synsetFile);
        while (!in.isEmpty()) {
            String line = in.readLine();
            String[] tokens = line.split(",");
            int    id     = Integer.parseInt(tokens[0]);
            String synset = tokens[1];
            String defn   = tokens[2];


// find maximum id number
            if (id + 1 > V) V = id + 1;

            // add definition
            // definitions.put(id, defn);
            synsets.put(id, synset);

            // ignore synset for now
            String[] nouns = synset.split(" ");
            for (String noun : nouns) {
                if (!map.contains(noun))
                    map.put(noun, new SET<Integer>());
                map.get(noun).add(id);


}
```

For each object that is not a `String`, its `toString()` method is called to convert it to a `String`.

For example, if you need to concatenate a large number of strings, appending to a `StringBuilder` object is more efficient.

Although you add the `int` values as primitive types, rather than `Integer` objects, to `li`, the code compiles. Because `li` is a list of `Integer` objects, not a list of `int` values,

Linked List

```
public class Solution {
	public ArrayList<Integer> rotateArray(ArrayList<Integer> A, int B) {
		ArrayList<Integer> ret = new ArrayList<Integer>();
		for (int i = 0; i < A.size(); i++) {
			ret.add(A.get((i + B) % A.size()));
		}
		return ret;
	}
}
```

```
public class Solution {

	public ArrayList<Integer> rotateArray(ArrayList<Integer> A, int B) {

		ArrayList<Integer> ret =  new ArrayList<Integer>();
		for (int i = 0; i < A.size(); i++) {
			ret.add( A.get( (i+B)%A.size()));
		}
		return ret;
	}
```

num1bits

GCD

LCM

```
public class Solution {
	public ArrayList<Integer> postorderTraversal(TreeNode a) {
	    Stack<TreeNode> s1 = new Stack<>();
	    Stack<TreeNode> s2 = new Stack<>();
	    ArrayList<Integer> post = new ArrayList<Integer>();
	    TreeNode x;
```

```
        if (a == null) return null;

        s1.push(a);

        while(!s1.isEmpty()) {
           x = s1.pop();
           s2.push(x);

           if (x.left != null)
              s1.push(x.left);

           if (x.right != null)
              s1.push(x.right);

        }

        while(!s2.isEmpty()) {
           x = s2.pop();
           post.add(x.val);
        }

        return post;
    }
}
```

```
public class Solution {
    public ArrayList<Integer> postorderTraversal(TreeNode A)
{
        Stack<TreeNode> stack1, stack2;
        ArrayList<Integer> postorder;
        TreeNode node;

        stack1 = new Stack<>();
        stack2 = new Stack<>();
        postorder = new ArrayList<>();

        if (A == null)
```

```
            return null;

        stack1.push(A);

        while (!stack1.isEmpty()) {

            node = stack1.pop();
            stack2.push(node);

            if (node.left != null)
                stack1.push(node.left);

            if (node.right != null)
                stack1.push(node.right);

        }

        while (!stack2.isEmpty()) {
            node = stack2.pop();
            postorder.add(node.val);
        }

        return postorder;

    }
}



public class Solution {
    public int isSymmetric(TreeNode A) {
```

```
        if (A == null)
            return 0;

        return rec(A.left, A.right) ? 1 : 0;

    }


    public boolean rec(TreeNode node1, TreeNode node2) {

        if (node1 == null && node2 == null)
            return true;

        if (node1 == null || node2 == null)
            return false;

        if (node1.val != node2.val)
            return false;


        return rec(node1.left, node2.right) && rec(node1.right,
node2.left);

    }

}


public class Solution {
    public int isSymmetric(TreeNode a) {
```

```
if (a == null) return 1;

Queue<TreeNode> ql = new LinkedList<TreeNode>();
Queue<TreeNode> qr = new LinkedList<TreeNode>();

ql.add(a.left);
qr.add(a.right);

while (!ql.isEmpty() && !qr.isEmpty()){

   TreeNode l = ql.remove();
   TreeNode r = qr.remove();

   if (l == null && r == null) continue;

   if (l == null ll r == null) return 0;

   if (l.val != r.val) return 0;

else {
   ql.add(l.left);
   qr.add(r.right);

   ql.add(l.right);
   qr.add(r.left);

}
```

```
        }
        return 1;
    }
}
```

Close

for this input "7 1 2 -1 -1 3 -1 -1" why is the minDepth not 2? The solution says it should be three, but if you just travel down the left side of the tree, you'll reach the closest leaf only using 2 nodes, not 3. I'm a bit confused about this.

Close