QUEENS COLLEGE Department of Computer Science
CSCI 313 Practice problems on Trees and MultiWay Trees
Instructor: Alex Ryba

These are some of the solutions to practice problems. Not all problems have solutions here.

Solutions to some older problems might not make use of generics. Generics are now required in this course.

Problem 1 A new method *miniRemove* in a *GeneralTree* is applied to a node of the tree as follows. If the node is a leaf, it is removed. Otherwise its data value is replaced by the data value of its first child and its firstChild is recursively removed by the method miniRemove.

For example if the GeneralTree is as follows:

```
  1
    2     3  4
      5 6      7
                 8
```

then if miniRemove is applied to the root, the tree changes to:

```
  2
    5   3  4
      6     7
               8
```

Assume that GeneralTrees are implemented as in class, based on the following skeleton Java code for the classes GNode and GeneralTree.

```
public class GNode<T> extends Node<T> {
  //  The inherited instance variables from class Node<T> are:  parent and data
   ArrayList<GNode<T>> children;      //   can be empty, but is never null
  //   standard methods and constructors omitted.
}

class GeneralTree<T> extends Tree<T> {
  //  The inherited instance variable from class Tree<T> is:    Node<T> root
  //  standard methods and constructors omitted
  //  The new method has title:
  public void miniRemove(GNode<T> n) {
     // code omitted
  }
}
```

Write a Java implementation for the method called miniRemove.

Hint: You might find it useful to call standard *Tree* methods such as *isRoot*, *isLeaf* and the standard *GeneralTree* method *remove*. The *GNode* method *get(i)* that returns the i^{th} child of a node is also useful here.

You may assume that the parameter is a node of the tree, in particular it is not null.

Answer:

```
public void miniRemove(GNode<T> n) {
   if (isLeaf(n)) remove(n);
   else {
     GNode<T> c = n.children.get(0);
     n.setData(c.getData());
     miniRemove(c);
   }
}
```

Problem 2 The class Tree uses the implementtion that we studied in class. It has instance variables `TNode root` and `int size`. The `interface TNode` is as studied in class. It specifies three methods:

`Iterator<TNode> children(); TNode getParent(); String printData();`.

Write a tree method with title line: `ArrayList<TNode>`lowerLeaves(TNode n): that returns a list of all leaf nodes among the descendants of `TNode n`.

For example, the lowerLeaves for the node 6 in the following tree are 9, e, f and g.

```
              1
            /   \
           /     \
          2       3
        /  \     / \
       4    5   6   7
           /   / \   \
          8   9   a   b
                 / \
                c   d
              /    / \
             e    f   g
```

Answer:

```
  public ArrayList<TNode> lowerLeaves(TNode n) {
    ArrayList<TNode> answer = new ArrayList<>();
    Iterator<TNode> kids = n.children();
    if (!kids.hasNext()) answer.add(n);
    else while (kids.hasNext()) {
      TNode k = kids.next();
      ans.addAll(lowerLeaves(k));
    }
    return answer;
  }
```

Problem 3 The class Tree uses the implementtion that we studied in class. It has instance variables `TNode root` and `int size`. The `interface TNode` is as studied in class. It specifies three methods:

`Iterator<TNode> children(); TNode getParent(); String printData();`.

Write a tree method with title line: `int smallFamilies()` that returns the number of nodes with fewer than 2 children.

For the following tree, the method would return a count of 11. This is because the 11 nodes storing 4, 5, 7, 8, 9, b, c, e, f, g, h are the ones with fewer than 2 children.

```
              1
            /   \
           /     \
          2       3
         / \     / \
        4   5   6   7
           /   / \   \
          8   9   a   b
                 / \   \
                c   d   e
              /    / \
              f   g   h
```

Answer:

```
int smallFamilies() {
   if (root == null) return 0;
   return smallFamilies(root);
}

int smallFamilies(TNode n) {
   int ans = 0;
   Iterator<TNode> kids = n.children();
   int numberKids = 0;
   while (kids.hasNext()) {
       ans += smallFamilies(kids.next());
       numberKids += 1;
   }
   if (numberKids <= 1) ans += 1;
   return ans;
}
```

Problem 4 The class Tree uses the implementtion that we studied in class. It has instance variables `TNode root` and `int size`. The `interface TNode` is as studied in class. It specifies three methods:

`Iterator<TNode> children(); TNode getParent(); String printData();`.

Write a tree method with title line: `Iterator<TNode> singleChildren()` that returns an iterator giving all nodes in the tree that have no siblings.

For example, the iterator would give the nodes that store 8, b, e and f in the following tree.

```
             1
           /   \
          /     \
         2       3
        / \     / \
       4   5   6   7
          /   / \   \
         8   9   a   b
                / \   \
               c   d   e
              /   / \
             f   g   h
```

Answer:

```
Iterator<TNode> singleChildren() {
    ArrayList<TNode> singles = new ArrayList<>();
    singleChildren(root, singles);
    return singles.iterator();
}

void singleChildren(TNode n, ArrayList<TNode> singles) {
    Iterator<TNode> kids = n.children();
    TNode child = null;
    int numberKids = 0;
    while (kids.hasNext()) {
       child = kids.next();
       singleChildren(child, singles);
       numberKids += 1;
    }
    if (numberKids == 1) singles.add(child);
}
```