COL106: Data Structures, I Semester 2015-16

Ungraded practice problem 1

Making general trees and binary trees

September 10, 2015

The aim of this practice problem is two-fold, firstly to understand how to make a general tree given a set of nodes and their parent information and the second is to implement general trees using binary trees. We will discuss the second of these first.

General trees using binary trees. Remember in the minor we had the following:

Suppose we are given a java implementation which has a `Tree` class and a `Node` class. The `Tree` class has the following methods available:

- `public Node root()`: This returns the root of the tree.
- `public Boolean isEmpty()`: This returns 1 if the tree is empty, 0 otherwise.

The `Node` class has the following methods available:

- `public int data()`: This returns the integer data value stored in the node.
- `public int no_children()`: This returns the number of children of the node.
- `public Node child(int i)`: This returns the `Node` of the ith child, returning `null` if there are less than i children.

- `public Tree subtree(int i)`: This returns the `Tree` rooted at the ith child, returning an empty tree if there are less than i children.

Build the java implementation of this general tree structure. In order to do so first build an implementation of a java class `BinaryTree` and a class `BinaryNode` which implements a Binary Tree data structure. Then use the method of implementing a general tree using a Binary tree discussed in class (and described in Goodrich and Tamassia's book) to implement the general `Tree` class described above.

To test that your code works you can try out the two functions given in the minor. (Watch out, they may have some syntax errors!)

```
public int f1 (Tree T) {
    if (T.isEmpty()) {
       return 0;
    }
    else {
       Node curr = T.root();
       int temp = curr.data();
       if (temp < 0) { temp = 0;}
       int i = 0;

       while (i < curr.no_children()) {
          temp = temp + f1(curr.subtree(i));
          i++;
       }
       return temp;
    }
}

public int f2 (Tree T) {
     if (T.isEmpty()) {
        return 1;
     }
     else {
        Node curr = T.root();
        if (curr.no_children() > 2) {
           return 0;
```

```
}
else {
   int temp = 1;
   int i = 0;

   while (i < curr.no_children()) {
      if (temp > f2(curr.subtree(i)) {
         temp = 0;
      }
      else
         i++;
   }
   return temp;
}
```

But to test your code you first have to build a tree. To build a tree you have to take some description of a tree as input. This brings us to the second part of the assignment.

Making a tree. You are given an input file. One each line of the file is a entry of the form (x, y), both x and y are numbers and are actually identifiers for nodes in the tree. x is the name of the node and y is the name of the parent. If $y = -1$ then the node x has no parent. So an example file may be

```
(3,2)
(4,2)
(2,-1)
(5,4)
```

This corresponds to the tree with root `2`. This root has two children `3` and `4` and one of these children, namely `4`, has a further child `5`.

Given a file with a list of such entries check that the entries can be used to form a tree. For example, if I give you

```
(2,-1)
(3,-1)
(4,3)
```

This is not a tree since there are two roots. What are the other cases? Figure them out. If the file doesn't correspond to a tree throw an exception. If it does then build the tree and use it to run functions `f1` and `f2` below.