Seoul National University

M1522.000900 Data Structure

Fall 2016, Kang

Homework 5: Non-binary Trees (Chapter 6)

Due: November 7, 05:00 PM

Reminders

- The points of this homework add up to 100.
- Like all homeworks, this has to be done individually.
- Lead T.A.: Chiwan Park (chiwanpark@snu.ac.kr)
- Please type your answers in English. Illegible handwriting may get no points, at the discretion of the graders.
- If you have a question about assignments, please upload your question in eTL.
- If you want to use slipdays or consider late submission with penalties, please note that you are allowed one week to submit your assignment after the due date.

Remember that:

1. Whenever you are making an assumption, please state it clearly

Question 1

A potential alternative to the weighted union rule for combining two trees is the height union rule. The height union rule requires that the root of the tree with greater height become the root of the union. Explain why the weighted union rule is more efficient than the height union rule in average cases. You may assume the following trees t_1 and t_2 [15 points].

- t_1 has n_1 nodes with the height h_1
- t_2 has n_2 nodes with the height h_2
- $n_1 < n_2$ and $h_1 > h_2$

Question 2

Write a Java function that takes as input a general tree and returns the number of nodes in that tree. Write your function to use the **GenTree** and **GTNode** ADTs of Figure 1. [20 points]

```
/** General tree node ADT */
interface GTNode<E> {
  public E value();
  public boolean isLeaf();
  public GTNode<E> parent();
  public GTNode<E> leftmostChild();
  public GTNode<E> rightSibling();
  public void setValue(E value);
  public void setParent(GTNode<E> par);
  public void insertFirst(GTNode<E> n);
  public void insertNext(GTNode<E> n);
  public void removeFirst();
  public void removeNext();
}

/** General tree ADT */
interface GenTree<E> {
  public void clear();      // Clear the tree
  public GTNode<E> root();  // Return the root
  // Make the tree have a new root, give first child and sib
  public void newroot(E value, GTNode<E> first,
                                  GTNode<E> sib);
  public void newleftchild(E value); // Add left child
}
```

Figure 1. Interfaces for the general tree and general tree node.

Question 3

In union-find algorithm, one alternative to path compression that gives similar performance gains is called path halving. Let a sequence of nodes $v_1, v_2, \cdots, v_n$ be a path of the node i to the root. Then, v_1 and v_n are the node i and the root, respectively. In the path halving, we set the parent of odd-indexed nodes $v_1, v_3, \cdots, v_{2t+1}$ to its grandparent where $0 \leq t \leq (n-1)/2$. Figure 2 shows an example of the path halving from the node 1 to the root.

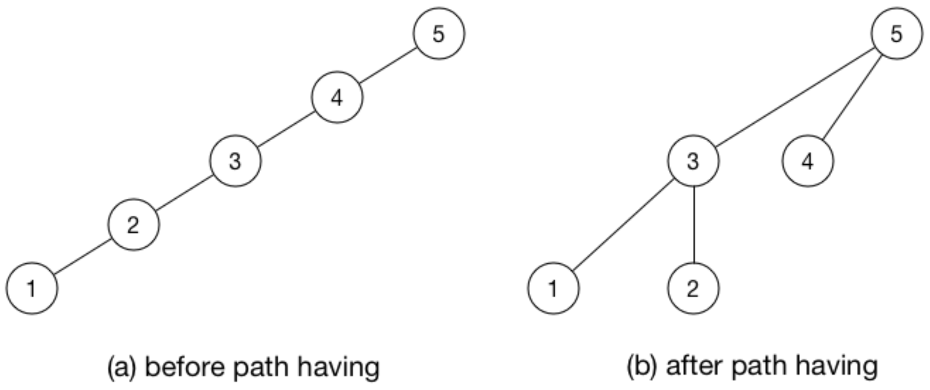

Figure 2. An example of the path halving.

Fill the `find` function below to implement find operation with path halving. [20 points]

```
public int find(int p) {
    // `find` returns the root of p's tree.
    // `parent` is an array which contains parent pointers.



}
```

Question 4

In the general union-find algorithm, there is no delete operation. A student proposes a function in Figure 3 to delete the node p from its containing set. Assume that a node which points to -1 is considered as deleted. However, the function doesn't work correctly. Explain why it doesn't work with a counter example. [15 points]

```
public void delete(Integer p) {
    // `delete` deletes p from its containing set.
    // `N` is the number of nodes.
    // `parent` is an array which contains parent pointers.
    for (int i = 0; i < N; i++) {
        if (parent[i] == p) {
            parent[i] = parent[p];
        }
    }
    parent[p] = -1;
}
```

Figure 3. The proposed function to delete node p from its containing set.

Question 5

Using the weighted union rule and path compression, show the array for the parent pointer implementation that results from the following series of equivalences on a set of objects indexed by the values 0 through 15. Initially, each element in the set should be in a separate equivalence class. When two trees to be merged are of the same size, make the root with greater index value be the child of the root with lesser index value [15 points].

(2, 3) (4, 5) (6, 5) (3, 5) (1, 0) (7, 8) (1, 8) (3, 8) (9, 10) (11, 14) (11, 10) (12, 13) (11, 13) (14, 1)

Question 6

Draw the binary tree representing the following sequential representation [15 points].

ABD//E//C/F//