Software 1 - Exercise 5

19/11/2006 – 26/11/2006

Part 1

The main purpose of this assignment is to practice implement a class given its **contract** and its **abstraction function**.

You will implement a class named *DisjointSets* that represents a set S of disjoint sets: S = {S1,S2,...,Sn}, where each Si is a set of non-negative integers. The class will support the following public methods:

```
// @pre x is not in any of the sets Si
// @abst S = old(S) U {{x}}
public void makeSet(int x);

// @pre x in Si , y in Sj
// @return true iff i == j
public boolean equiv(int x, int y);

// @pre x in Si , y in Sj , i !=j
// @abst S = old(S) - Si - Sj U {Sij} where Sij = Si U Sj
public void joinSets(int x, int y);

// @pre nothing
// @return true iff x is in some Si
public boolean inASet(int x);
```

One way to implement the class is by representing each set Si as a tree, where each node (associated with a nonnegative integer x) points to his parent. In this way, the root of a tree uniquely represents a set Si. An array named parent can be used to hold all these pointers. Specifically, for each number x, the value of parent[x] is the number of the parent node in the tree or −1 if the number x does not belong to any set Si. The root of a tree points to itself. The length of the array should be bigger than any number x that currently in one of the sets Si. Thus, there is a need to replace the current array with a bigger one when a new large number is added.
Given an object of the DisjointSets class, its abstract state is a set of disjoint sets of nonnegative integers, {S1,S2, ...,Sn}. To define the abstraction function, we first define a helper function **r(x)** as follows:

```
If parent[x] = -1 then r(x) = -1
Else if parent[x] == x then r(x) = x
Else r(x) = r(parent[x])
```

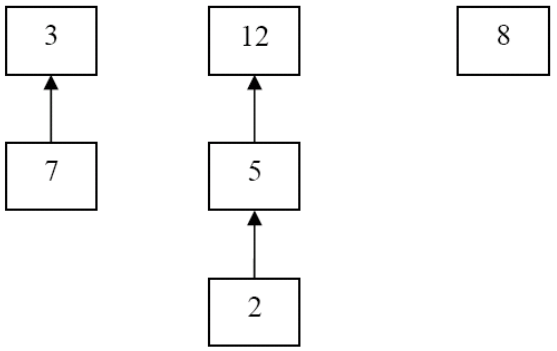
Now we can define the abstraction function that maps from parent, the field of the object, to a set of sets:
S(this) = {S_1, S_2, … S_n} such that

For all x, [for all i, $1 \le i \le n$, $x \notin S_i$] iff [$x \ge$ parent.length or parent[x] = −1]
For all $0 \le x,y \le$ parent.length, $x,y \in$ Si (x and y are in the same set) iff r(x) = r(y) ≠ −1

For example, for an object of the DisjointSets class with parent =

-1	-1	5	3	-1	12	-1	3	8	-1	-1	-1	12
0	1	2	3	4	5	6	7	8	9	10	11	12

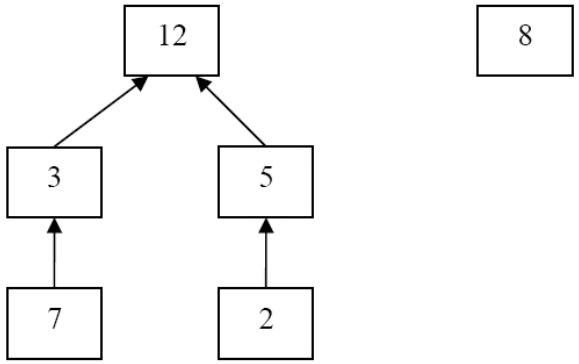
The associated trees are:



and the abstract state is S = { {3, 7} {12, 5, 2} {8} }. Applying *equiv(3,5)* will return false. If we apply *joinSets(7,5)* the parent array will be

-1	-1	5	12	-1	12	-1	3	8	-1	-1	-1	12
0	1	2	3	4	5	6	7	8	9	10	11	12

Where the associated trees are



and the abstract state is S = { {3, 7, 12, 5, 2} {8} }. If we apply *makeSet(4)* the abstract state will be S = { {3, 7, 12, 5, 2} {8} {4}}.

Your assignment:

Implement the *DisjointSets* class efficiently using the parent array (a template can be found on the course web site). Note that for your convenience you may need to define helper methods. For example, it may help to define a function that ensures that *parents[]* is long enough.

Part 2

The interface IIPAddress represents an **I**nternet **P**rotocol address, such as the addresses used in all computer networks and the internet. Some example IP addresses are:

127.0.0.1
192.168.1.10

As you can see, an address is composed of four 8-bit unsigned numbers, which can also be seen as a 32-bit number.

```
package il.ac.tau.cs.software1.ex5;

public interface IIPAddress {
	/**
	 * Returns a string representation of the IP address,
	 * for example "192.168.0.1"
	 */
	public String toString();
	/**
	 * @param ip IP address to which we compare this address
	 * @return true if both objects represent the same address
	 */
	public boolean equals(IIPAddress ip);
	/**
	 * Returns one of the four parts of the IP address
	 *
	 * @param index legal vales are [0..3], for example
	 *    192.168.0.1 would return 192 for index 0
	 * @return the desired part of the IP address (one of four parts)
	 *         [0..255]
	 */
	public short getOctet(int index);
	/**
	 * There are four classes of private networks
	 *   (http://en.wikipedia.org/wiki/IPv4#Private_networks)
	 *           10.0.0.0 - 10.255.255.255
	 *           172.16.0.0 - 172.31.255.255
	 *           192.168.0.0 - 192.168.255.255
	 *           169.254.0.0 - 169.254.255.255
	 *
	 * This query returns true if this object is a private network address
	 */
	public boolean isPrivateNetwork();
	/**
	 * This command receives as a parameter an IP address
	 * and uses it to mask the current address
	 * i.e. perform a bitwise 'and' operator on all four parts
	 *
	 * @param mask the IP address with which to mask
	 */
	public void mask(IIPAddress mask);
}
```

1. Write three implementations for the interface IIPAddress.
   - IIPAddressString which holds an IP address as a String internally
   - IIPAddresShortArray which holds the address as an array of shorts (four cells, 0..255 in each cell)
   - IIPAddresInt which holds the IP address as an integer

   Each implementation should have an appropriate constructor, and should implement all the methods from the interface

2. Write a factory class IIPAddressFactory which implements the following methods

```
public class IIPAddressFactory {
      public static IIPAddress createAddress(String ip) {…}

      public static IIPAddress createAddress(short[] ip) {…}

      public static IIPAddress createAddress(int ip) {…}
}
```

Each static method in the factory should create an instance of an implementation of IIPAddress, the decision which implementation to create is based on the type of input.

Good Luck :o)