

Software 1 - Exercise 5

29/03/2007 – 12/04/2007

Part 1

The main purpose of this assignment is to practice implementation of a class given its method description.

You will implement a class named *DisjointSets* that represents a set S of disjoint sets: $S = \{S_1, S_2, \dots, S_n\}$, where each S_i is a set of non-negative integers. The class will support the following public methods:

```
/**
 * Create a set containing x (i.e. {x}) and add it to
 * this object. The given integer x should not be a
 * member of any other set in self.
 */
public void makeSet(int x);

/**
 * Return true if and only if x and y belong to the same
 * set in this object.
 */
public boolean equiv(int x, int y);

/**
 * Find the different sets that x and y belong to. Remove
 * them from this object and add their union
 */
public void joinSets(int x, int y);

/**
 * Return true if and only if x is in some set of this
 * object
 */
public boolean inASet(int x);
```

One way to implement the class is by representing each set S_i of S 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 S_i . 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 S_i . 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 S_i . Thus, there is a need to replace the current array with a bigger one when a new large number is added.

Any object of the *DisjointSets* class represents a set of disjoint sets of nonnegative integers, $\{S_1, S_2, \dots, S_n\}$. This representation is formally defined by using the following helper function $r(x)$:

```

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 mapping from parent, the field of the object, to a set of sets:
 $S(\text{this}) = \{S_1, S_2, \dots, S_n\}$ such that

```

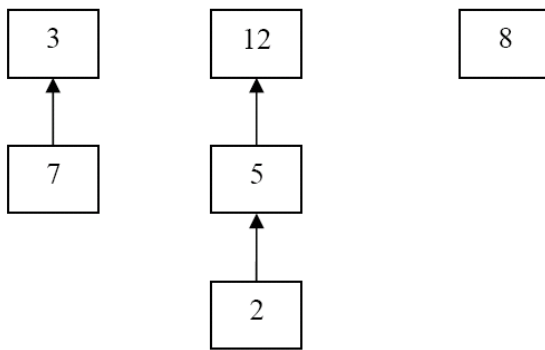
For all x, [for all i, 1 ≤ i ≤ n, x ∉ Si] iff [x ≥ parent.length or parent[x] = -1]
For all 0 ≤ x,y ≤ parent.length, x,y ∈ Si (x and y are in the same set) iff r(x) = r(y) ≠ -1

```

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

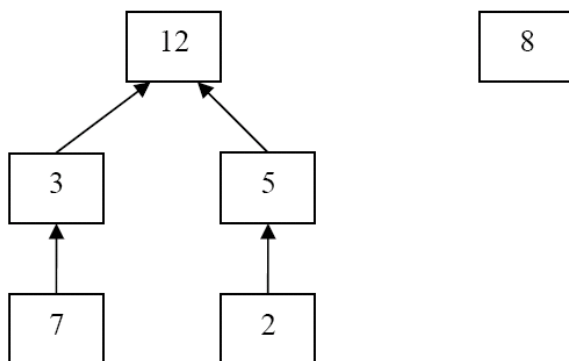
The associated trees are:



and the represented set of disjoint sets 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 represented set of disjoint sets is $S = \{ \{3, 7, 12, 5, 2\} \{8\} \}$. If we apply *makeSet(4)*, we get that $S = \{ \{3, 7, 12, 5, 2\} \{8\} \{4\} \}$.

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.

For each method, define the pre and post conditions. Then, use defensive programming to ensure the validity of the pre-conditions.

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
 - `IIpAddressShortArray` which holds the address as an array of shorts (four cells, 0..255 in each cell)
 - `IIpAddressInt` 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 class named `IIpAddressFactory` that 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 this class should create an instance of an implementation of `IIpAddress`, where the decision which implementation to create is based on the type of input.

Note: A class whose sole purpose is to create instances of other classes is called a *factory class*. Such classes are used to abstract the details of creating instances from clients, usually in order to hide the concrete class being used to implement an interface. This makes it easier to change implementations.

הוראות הגשה:

1. קראו בעיון את קובץ נוהלי הגשת התרגילים אשר נמצא באתר הקורס.
2. הגשת התרגיל (כקובץ zip המכיל את קבצי ה-java של התכניות שהתבקשת לכתוב) תעשה ע"י המערכת VirtualTAU (<http://virtual.tau.ac.il>) עד ליום חמישי 12.04.2007 בחצות. הוראות שימוש במערכת ישנם ב-
<http://virtual2002.tau.ac.il/upload/misc/main1.html>
3. כמו כן, יש לשים תדפיס של קבצי ה-java. בתא הדואר של בודק התרגילים עד יום ראשון 15.04.2007 בחצות. יש לציין שם מלא ות"ז.