CS211 ALGORITHMS & DATA STRUCTURES II

LAB 6

Dr. Phil Maguire

RED BLACK TREES

Pen and Paper Exercise

Add the following numbers into red-black tree. To make sure you get it right you should show clearly the colour changes and the rotations involved.

26 15 54 66 58 83 90 70 96 75 72

Programming Exercise

Alice's public key is (24852977, 2744, 8414508). You eavesdrop on the line and observe Bob send her the cipher (15268076, 743675). Extract the message by any means.

The programming challenge requires implementation of the same algorithm as above, with the added problem of giving to crack the private key (try brute force).

Warning: for the programming part, make sure to use *longs* rather than *ints* (you may need to put an 'l' at the end of the number). Also, make sure to keep modulo-ing every time the number in the calculation gets a little too big – never do large power multiplications straight off because Java cannot process large numbers like this.

As a result, it might be worthwhile to use these recursive methods for modulo multiplication and modulo raising to a power. These work efficiently and ensure that the numbers never get too big.

```
    public static long modPow(long number, long power, long modulus){
//raises a number to a power with the given modulus
//when raising a number to a power, the number quickly becomes too large to handle
//you need to multiply numbers in such a way that the result is consistently moduloed to keep it in the range
//however you want the algorithm to work quickly - having a multiplication loop would result in an O(n) algorithm!
//the trick is to use recursion - keep breaking the problem down into smaller pieces and use the modMult method to join them back together
        if(power==0)
            return 1;
        else if (power%2==0) {
            long halfpower=modPow(number, power/2, modulus);
            return modMult(halfpower,halfpower,modulus);
        }else{
            long halfpower=modPow(number, power/2, modulus);
            long firstbit = modMult(halfpower,halfpower,modulus);
            return modMult(firstbit,number,modulus);
        }
    }


     public static long modMult(long first, long second, long modulus){
//multiplies the first number by the second number with the given modulus
//a long can have a maximum of 19 digits. Therefore, if you're multiplying two ten digits numbers the usual way, things will go wrong
//you need to multiply numbers in such a way that the result is consistently moduloed to keep it in the range
//however you want the algorithm to work quickly - having an addition loop would result in an O(n) algorithm!
//the trick is to use recursion - keep breaking down the multiplication into smaller pieces and mod each of the pieces individually
        if(second==0)
            return 0;
        else if (second%2==0) {
            long half=modMult(first, second/2, modulus);
            return (half+half)%modulus;
        }else{
            long half=modMult(first, second/2, modulus);
            return (half+half+first)%modulus;
        }
     }
```

Bonus Question (2% extra)

The first person to email me the message will get an extra 2% CA. It's exactly the same idea, but the numbers are getting bigger. You might try the Baby Steps Giant Steps algorithm to reduce the complexity to O(sqrt(n)).

Alice’s public key is (83187685431865799, 231541186, 58121575766172118). You eavesdrop on the line and observe Bob send her the cipher (15714167638989179, 1416052582726447). Can you find the private key and extract the message Bob is sending? If you can, email me the code that did it with an explanation.