LAB 7

MyPowerMod and RSA

The goal of this lab is to write our own PowerMod function and then use it to build RSA encryption and decryption functions. We have already written functions for taking quotients and remainders, so feel free to use Mathematica's `Quotient` and `Mod` commands.

PROBLEM 7.1. What does the Mathematica function `PowerMod[a,b,m]` do? Start with $m = 26$ and $b = 1$, and explore from there.

PROBLEM 7.2. Write a `PositivePowerMod[a_,b_,m_]` command that assumes that b is positive and computes REM (a^b, m). Remember to use recursion and the fast-exponentiation approach (b might be a 10-digit number, or even longer).

PROBLEM 7.3. We are about to implement the Extended Euclidean Algorithm in Mathematica. In order to remember how it works, use it to find the Bézout coefficients for $a = 137$ and $b = 53$, using the table approach.

PROBLEM 7.4. Write a `ExtendedEuclideanTable[a_,b_]` that creates the table we generate with the Extended Euclidean Algorithm.

It should look roughly like this:

```
ExtendedEuclideanTable[a_,b_]:=Module[{EETable={},i=1},
     AppendTo[EETable, (*add the starting row*)];
     While[(*remainder of a and b is not zero*),
         (*add a new row to EETable*);
         i++;
     ];
     (*fill in the starting i-th row of s and t values*)
     While[(*we have not returned to the top*),
         i--;
         (*fill in the i-th row of s and t values*);
     ];
     EETable;
]
```

PROBLEM 7.5. Does `ExtendedEuclideanTable[137,53]` produce the same table as you made in Problem 7.3?

PROBLEM 7.6. By accessing the correct cells inside the table produced by `ExtendedEuclideanTable`, write functions `MyGCD[a_,b_]`, `MyBezout[a_,b_]` and `MyMultiplicativeInverse[a_,m_]`.

Hint: Tables are just lists of lists. To get the third element of a list `T`, use the command `T[[3]]`. To get the last one, use `T[[-1]]`.

PROBLEM 7.7. Write a `MyPowerMod[a_,b_,m_]` function by combining `MyMultiplicativeInverse` and `PositivePowerMod`.

PROBLEM 7.8. Make a `MakeRSAKeys[]` function that randomly chooses $\{e, d, n\}$. It should look something like:

```
MakeRSAKeys[a_,b_]:=Module[{p,q,e,n,m,d},,
     p=RandomPrime[10^9,10^10];
     (*Generate q and e, and compute n, m, and d*)
     {n,e,d}
]
```

Make sure you use `MyPowerMod` to compute d. Use `MakeRSAKeys[]` to generate RSA public and private keys, and post your public key on the whiteboard.

PROBLEM 7.9. Use `MyPowerMod` to create the functions

```
RSAEncode[text_,e_,n_]:=MyPowerMod[
    FromDigits[
        ToCharacterCodes[text]-97,
        26
    ],e,n];


RSADecode[y_,d_,n_]:=
    FromCharacterCodes[
        FromIntegerDigits[
            MyPowerMod[y,d,n],
            26
        ]+97;
    ]
```

Use these to send messages to other teams, or sign your announcements. This time, we can't (easily) crack your encryption, so make sure you're writing the correct numbers on the board.