FIT3155 S1/2023: Assignment 3
(Due midnight 11:55pm on Sunday 28 May 2023)

[Weight: $10 = 5 + 5$ marks.]

Your assignment will be marked on the performance/efficiency of your programs. You must write all the code yourself, and should not use any external library routines that interferes with the assessable items, except those that are considered standard. The usual input/output and other unavoidable routines are exempted. Importing `random` (for q1) and `numpy` (for using Matrix and numpy arrays to handle q2) is allowed.

Follow these procedures while submitting this assignment:

The assignment should be submitted online via moodle strictly as follows:

- All your scripts MUST contain your name and student ID.
- Bundle and upload your work as a ⟨studentid⟩.tar.gz or ⟨studentid⟩.zip archive.
  - Your archive should extract to a single directory which is your student ID.
  - This directory should contain a subdirectory for each of the two questions, named as: `q1/` and `q2/`.
  - Your corresponding scripts and work should be tucked within those subdirectories.
- Submit your zipped-archive electronically via Moodle.

Academic integrity, plagiarism and collusion

Monash University is committed to upholding high standards of honesty and academic integrity. As a Monash student your responsibilities include developing the knowledge and skills to avoid plagiarism and collusion. Read carefully the material available at https://www.monash.edu/students/academic/policies/academic-integrity (click) to understand your responsibilities. As per FIT policy, all submissions will be scanned via MOSS (click) and/or JPlag (click).

Generative AI not allowed!

This unit fully restricts you from availing/using generative AI to answer these assessed questions.

Question 1: Generating a public key [5 marks]

Background

RSA (Rivest–Shamir–Adleman) is a public-key encryption system that is popular in modern internet communications. A public key as the word suggests is publicly advertised. A system using RSA generates a public key in the form of a modulus n and exponent e using two large prime numbers. While n and e are public, the two prime numbers that were used to generate them are held secretly away from the public eye. Anyone wanting to communicate with the system uses the public key information to encrypt any message. A secure message can only be unlocked using the knowledge of the prime numbers behind the public key. The reliability of such a cryptosystem is purely due to the infeasibility of factorizing any number (especially large ones) into its prime factors on classical computers.[1]

Task on hand

In this exercise, inspired by RSA (but not exactly the same in details), we will be generating two integers, modulus n and exponent e, as a public key.[2]

The steps you need to follow to generate n and e are described in the corresponding sections below. To pull this off, you will have to implement on your own the following:

1. Modular exponentiation with repeated squaring.
2. Miller-Rabin randomized primality test with appropriate level of confidence that the test is probably correct.
3. Euclid algorithm to compute the greatest common divisor of any two numbers.

You are free to use the built-in basic arithmetic operations (addition, subtraction, multiplication, division and modulo division). However, note, as indicted above (see enumerated item 1), you will have to implement your own modular exponentiation using repeated squaring method.

Generating the modulus n

In this exercise, the modulus of the public key $n = p \times q$ is a product of two special kind of prime numbers p and q we will use in this exercise.

How are these prime numbers selected in this exercise? Suppose $d > 2 \in \mathbb{Z}^+$ is an input parameter to your program. Then p and q respectively are the smallest two prime integers of the form $2^x - 1$ where $x \geq d$. For all practical purposes, you are free to assume $d \leq 2000$ for this exercise.

[1] Unless, in the foreseeable future, a quantum computer with sufficient number of 'qubits' becomes a reality, to be able run a well-established integer factorization algorithm (Shor's algorithm) on it. It is claimed that what takes quadrillion years to factorize on a classical computer would only take seconds (!) using Shor's algorithm on a quantum computer.

[2] To keep things simple, we will not worry about the actual encryption and decryption of messages in this exercise, but for the more keen ones, you are encouraged to explore the mathematics behind these processes that will inform a fuller implementation of what you started here, for your own further learning.

Generating the exponent e

The exponent e is computed using p and q as follows. First a value λ is calculated such that $\lambda = \frac{(p-1)(q-1)}{\texttt{gcd}(p-1,q-1)}$. (You will have to implement Euclid's algorithm to compute `gcd`.)

The exponent e is randomly chosen in the range $[3, \lambda - 1]$ such that $\texttt{gcd}(e, \lambda) = 1$, that is e and λ are relatively prime.

Strictly follow the specification below to address this question:

Program name: `mykeygen.py`

Arguments to your program: input integer $d \geq 1$

Command line usage of your script:

python mykeygen.py <*d*>

Output files: You will have to write out two files:

1. `publickeyinfo.txt` giving information of the modulus n and exponent e.
2. `secretprimes.txt` giving the information of the two primes p and q

Output formats: See example below.

Example: For $d = 10$,

`publickeyinfo.txt` will contain:

```
# modulus (n)
1073602561
# exponent (e)
3187811
```

`secretprimes.txt` will contain:

```
# p
8191
# q
131071
```

(Note, in the `publickeyinfo.txt` file, the exponent is randomly chosen, so its value will vary each time you run your program with the same argument. Also note that, in both the output files, the markup lines starting with # should also be printed as shown in the example above.

Question 2: Implementing Tableau Simplex [5 Marks]

In this exercise you are required to implement Dantzig's Tableau simplex algorithm to solve a linear program in its standard form, that was covered in the Week 10 lecture.

Your program will read an input text file (see input format below) specifying a linear program. The linear program by default will be in a standard form. This means that the goal

is always to maximize a given linear objective function, involving decision variables that are always non-negative, subject to a set of linear constraints on the decision variables that are expressed in the form: LHS $\leq$ RHS.

Your goal is to find an optimal set of values for the decision variables and the resultant evaluation of the objective function given those values. To report these, you will have to implement the exact same Tableau method that we discussed during the week 10 lecture.

Strictly follow the specification below to address this question:

Program name: `mysimplex.py`

Argument to your program: Filename of an input file giving the details of the linear program in a specific format (see example below).

Command line usage of your script:

```
python mysimplex.py <filename of a file specifying LP>
```

Input format: As an example, if the linear program being considered is the following:

$$\text{maximize } z = x + 2y$$

subject to the constraints

$$\begin{aligned} 4x + y &\leq 44 \\ 3x + 2y &\leq 39 \\ 2x + 3y &\leq 37 \\ y &\leq 9 \\ -x + y &\leq 6 \end{aligned}$$

(with the implicit constraints that both x and y decision variables are non-negative (≥ 0)) then the input specification format for this linear programming problem will be as follows (including the # lines):

```
# numDecisionVariables
2
# numConstraints
5
#objective
1, 2
# constraintsLHSMatrix
4, 1
3, 2
2, 3
0, 1
-1, 1
# constraintsRHSVector
44
39
37
9
6
```

Output file name: `lpsolution.txt` (see format below)

Output format For the above example LP problem specified as input, the output would be in the following form (including the # lines):

```
# optimalDecisions
5, 9
# optimalObjective
23
```

```
-=o0o=-
  END
-=o0o=-
```