College of Engineering & Applied Sciences

CSPB 2824

Discrete Structures

Number Theory And Primes Mastery Workbook

University Of Colorado

2024

Discrete Structures - Number Theory And Primes Mastery Workbook

Mastery Workbook 6

Number Theory And Primes Mastery Workbook

I have neither given nor received unauthorized assistance.

Taylor James Larrechea

Problem 1

Problem Statement

Look up the RSA algorithm and write 3 specific questions or observations you have about how it works. Khan Academy is a good place to start. Khanacademy RSA Algorithm

Summary

I have heard of encryption in my experience regarding computer science for some time now. RSA, is obviously some form of encryption that is used to transmit data between parties in a safe manner. The example that was shown in the video from Khanacademy with the colors is a very good explanation to how encryption works with the use of colors.

From what I observed in the video, my **first** observation of the RSA algorithm is an algorithm that utilizes modulo arithmetic to encrypt a message. After this, I can see that RSA is utilizing an encryption method that aims to be really difficult in undoing (decrypting) encryption. This is primarily done with the use of modular exponentiation. From an outsiders point of view, I can see that this should be a very effective way of encrypting a message (or some other data) between two parties.

My **second** observation is that RSA incorporates something called prime factorization. The video uses the example of the number 30 to show that it could be factored like $5 \times 3 \times 2$ where the factors 5 and 3 are prime numbers. The videos on Khanacademy proceed to go on about the time complexity of factorizing numbers and the time that it could potentially take to factorize a very large number. This then dives into the topic of Euler's Totient function and how that relates to numbers.

My **third** and final observation that I made in regards to RSA is that RSA uses a combination of the previous techniques for encrypting and decrypting information. With the use of Euler's Totient function and prime factorization, we can see that RSA provides a safe and effective way in encrypting data between two parties. I found it interesting that this simple algorithm is used so widely amongst our lives and I am very excited to put this to use!

Problem 1 Summary

Procedure

- This problem involves writing a summary on the concept of RSA.

Key Concepts

- RSA is a data encryption technique that uses several mathematical principles and methods to safely encrypt and decrypt data.
- RSA utilizes FME (fast modular exponentiation) to calculate the modulo of large numbers. Precisely, FME is defined as
$$a^b \textbf{ mod } m.$$
- RSA utilizes Euler's Totient function, a function that calculates the number of integers up to a given integer n that are relatively prime to n. Euler's Totient function is precisely
$$\phi(p^k) = p^k - p^{k-1}.$$
- RSA utilizes public and private keys to encode and decode data. These keys are calculated with algorithms that utilize a myriad of mathematical principles.

Variations

- We could be asked to describe specific aspects about RSA.
  - We would then discuss the necessary components about the specific aspect to answer the question.

Problem 2

Problem Statement

Do as many of #1- 4 p. 255, and #3, #17, #25, #27 from section 4.3 as you need to understand the fundamental concepts on your own (not graded). On this page and the next, summarize essential definitions and concepts from Sec. 4.1 - 4.3. Pictures, concept maps, flash cards, or just a list of ideas are all good. We will grade this on the appearance of a serious attempt and clear and accurate references to the material. Really, this is for you.

Solution

I first want to walk through examples found from Section 4.2 on binary and decimal expansion.

Problem 1: Convert the decimal expansion of each of these integers to a binary expansion.

(a) 231

$$\begin{aligned}
231 \div 2 &= 115 \textbf{ remainder } \underline{1} \\
115 \div 2 &= 57 \textbf{ remainder } \underline{1} \\
57 \div 2 &= 28 \textbf{ remainder } \underline{1} \\
28 \div 2 &= 14 \textbf{ remainder } \underline{0} \\
14 \div 2 &= 7 \textbf{ remainder } \underline{0} \\
7 \div 2 &= 3 \textbf{ remainder } \underline{1} \\
3 \div 2 &= 1 \textbf{ remainder } \underline{1} \\
1 \div 2 &= 0 \textbf{ remainder } \underline{1}
\end{aligned}$$

Therefore the decimal expansion of 231 into a binary expansion is $\underline{11100111}$. What we are doing here, is we start with an integer value, divide by 2, and the remainder of that division is either a 1 or 0 that represents a digit of the binary number that we are seeking to find. We repeat this operation with the quotient of the division until the quotient is less than or equal to 2. The resulting string of 1s and 0s is then our binary number.

Problem 3: Convert the binary expansion of each of these integers to a decimal expansion.

(c) $(101010101)_2$

$$\begin{aligned}
(101010101)_2 &= \underline{1} \cdot 2^8 + \underline{0} \cdot 2^7 + \underline{1} \cdot 2^6 + \underline{0} \cdot 2^5 + \underline{1} \cdot 2^4 + \underline{0} \cdot 2^3 + \underline{1} \cdot 2^2 + \underline{0} \cdot 2^1 + \underline{1} \cdot 2^0 \\
&= 256 + 0 + 64 + 0 + 16 + 0 + 4 + 0 + 1 \\
&= 341
\end{aligned}$$

Therefore the binary expansion of $(101010101)_2$ into decimal expansion is 341. We first do this by counting the number of digits in our binary number (indexing from zero). In this example, we have a binary number that has 9 digits. We then proceed to take the binary digit (in the example above the binary digit is the number with the bar under it), multiply it with 2 raised to the power of the index of the binary number from left to right. We then sum all of these numbers together to get our final decimal expansion of the binary number.

Next, I want to walk through examples found from Section 4.3 on prime factorization.

Problem 3: Find the prime factorization of each of these integers.

(a) $88 = 11 \cdot 8 = 11 \cdot 2 \cdot 4 = 11 \cdot 2 \cdot 2 \cdot 2 = 11 \cdot 2^3$

(b) $126 = 7 \cdot 18 = 7 \cdot 2 \cdot 9 = 7 \cdot 2 \cdot 3 \cdot 3 = 7 \cdot 2 \cdot 3^2$

(c) $729 = 3 \cdot 243 = 3 \cdot 3 \cdot 81 = 3 \cdot 3 \cdot 3 \cdot 27 = 3 \cdot 3 \cdot 3 \cdot 3 \cdot 9 = 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 = 3^6$

(d) $1001 = 7 \cdot 143 = 7 \cdot 11 \cdot 13$

The example above illustrates how to factor a number into multiples of individual prime numbers. This is done by finding the first prime number that will have a mod of zero. We then take the following factors and break them down into factors of prime numbers, in some cases the factors are written as powers of primes with other primes.

Problem 17: Determine whether the integers in each of these sets are pairwise relatively prime.

(a) $11, 15, 19 \rightarrow$ gcd(11,15) = 1, gcd(11,19) = 1, gcd(15,19) = 1 $\therefore$ Yes

(b) $14, 17, 85 \rightarrow$ gcd(14,17) = 1, gcd(14,85) = 1, gcd(17,85) = 17 $\therefore$ No

(c) $25, 41, 49, 64 \rightarrow$ gcd(25,41) = 1, gcd(25,49) = 1, gcd(25,64) = 1, gcd(41,49) = 1, gcd(41,64) = 1, gcd(49,64) = 1 $\therefore$ Yes

(d) $17, 18, 19, 23 \rightarrow$ gcd(17,18) = 1, gcd(17,19) = 1, gcd(17,23) = 1, gcd(18,19) = 1, gcd(18,23) = 1, gcd(19,23) = 1 $\therefore$ Yes

This example illustrates what it means for a pair of integers to be pairwise relatively prime. In this case, for a set of integers to be considered pairwise relatively prime the greatest common devisor (gcd) in the set must all be one. In the above example, there is only one set of numbers where this is not true (14,17,85).

Problem 25: What are the greatest common divisors of these pairs of integers?

(a) $3^7 \cdot 5^3 \cdot 7^3, 2^{11} \cdot 3^5 \cdot 3^9 = 3^5 \cdot 5^3$

(b) $11 \cdot 13 \cdot 17, 2^9 \cdot 3^7 \cdot 5^5 \cdot 7^3 = 1$

(c) $23^{31}, 23^{17} = 23^{17}$

(d) $41 \cdot 43 \cdot 53, 41 \cdot 43 \cdot 53 = 41 \cdot 43 \cdot 53$

(e) $3^{13} \cdot 5^{17}, 2^{12} \cdot 7^{21} = 1$

(f) $1111, 0 = 1111$

This example is another example that illustrates what the gcd of two sets of numbers are. In this case, we need to examine each set of numbers and determine what the gcd is for each set first. Once we determine this, we move on to the second set of numbers to determine if the previous gcd is a multiple of the other sets gcd. If it is, then that is the gcd of both sets. If it is not, then we have to backtrack and find what the gcd between the two sets would be. The gcd in this case is expressed as a prime factorization.

These problems incorporate the following ideas from the sections in Rosen:

- **Binary Expansion:** The representation of an integer in another base in binary.
- **Decimal Expansion:** The representation of an integer in another base in decimal.
- **Prime Factorization:** The representation of an integer written in a factorized form of individual prime numbers.
- **Pairwise Prime:** The determination of whether a set of integers have a greatest common divisor of 1 or not.
- **Greatest Common Divisor:** The largest integer between two numbers where the modulo of the number in question is 0.
- **Least Common Multiple:** The smallest integer between two numbers that is a factor of both numbers.

Problem 2 Summary

Problem Statement

- For Problem 1, calculate the modulo of the quotient with 2, first starting with 231. Repeat the process until the dividend is 1.
- For Problem 3, calculate $b_n \cdot 2^n$, where b_n is the digit of the binary number and n is the index of the binary number digit in the binary number.
- For Problem 3 (of section 4.3), calculate the prime factorization by factoring a number into prime multiples that are raised to a power.
- For Problem 17, calculate the greatest common divisor (gcd) of two numbers to determine if they are relatively prime.
- For Problem 25, calculate the greatest common divisor of the set of numbers.
- Summarize the concepts in this problem.

Problem Statement

- **Binary Expansion:** The representation of an integer in another base in binary. To calculate the binary expansion, calculate the modulo of 2 with the quotient of the dividend. Repeat the process until the dividend is 1.
- **Decimal Expansion:** The representation of an integer in another base in decimal. To calculate the decimal expansion of a binary number we use

$$D = \sum_{i=0}^{n} b_i \cdot 2^i.$$

- **Prime Factorization:** The representation of an integer written in a factorized form of individual prime numbers. To calculate the prime factorization of a number, factor the number into prime multiples.
- **Pairwise Prime:** The determination of whether a set of integers have a greatest common divisor of 1 or not. Numbers are defined as pairwise prime if the greatest common divisor of the two numbers is 1.
- **Greatest Common Divisor:** The largest integer between two numbers where the modulo of the number in question is 0.
- **Least Common Multiple:** The smallest integer between two numbers that is a factor of both numbers.

Variations

- We could be given different numbers to perform these operations on.
  - In this case we would just use the same formulae to perform the operations that are requested of us.

Problem 3

Problem Statement

Read "The Sieve of Eratosthenes" on page 259. Notice it uses THM 2, when itsays "note that the composite integers not exceeding 100 must have a prime factor not exceeding 10."

(a) Consider Theorem 2 page 258 - show inductively that it is true with some examples.

(b) Re-Prove Theorem 2 page 258 in your own words, using our proof structure for this proof by contradiction.

Solution - Part (a)

Theorem: If n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$.

Examples:

$$n = 27, \sqrt{27} = 3\sqrt{3} \approx 5.19, 27 \div 3 = 9, 3 \leq 5.19$$
$$n = 42, \sqrt{42} = \sqrt{7 \cdot 6} \approx 6.48, 42 \div 3 = 14, 3 \leq 6.48$$
$$n = 56, \sqrt{56} = 2\sqrt{14} \approx 7.48, 56 \div 7 = 8, 7 \leq 7.48$$

Solution - Part (b)

Theorem: If n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$.

Negation: There exists a composite integer n such that all its prime divisors are greater than $\sqrt{n}$.

Proof By Contradiction:

$$n = ab \quad \text{(Definition of } n\text{)} \quad (1)$$
$$\sqrt{n} = \sqrt{ab} \quad \text{(Square root of } n\text{)} \quad (2)$$
$$\sqrt{n} = \sqrt{a}\sqrt{b} \quad \text{(Definition of square root)} \quad (3)$$
$$\frac{\sqrt{n}}{\sqrt{b}} = \sqrt{a} \quad \text{(Simplification by division)} \quad (4)$$
$$\frac{\sqrt{n}}{\sqrt{a}} = \sqrt{b} \quad \text{(Simplification by division)} \quad (5)$$
$$\frac{n}{b} = a \quad \text{(Squaring both sides)} \quad (6)$$
$$\frac{n}{a} = b \quad \text{(Squaring both sides)} \quad (7)$$
$$\frac{n}{b} < a \quad \text{(Premise)} \quad (8)$$
$$\frac{n}{a} < b \quad \text{(Premise)} \quad (9)$$
$$\frac{ab}{b} < a \quad \text{(Substitution)} \quad (10)$$
$$\frac{ab}{b} < b \quad \text{(Substitution)} \quad (11)$$
$$a < a \quad \text{(Simplification)} \quad (12)$$
$$b < b \quad \text{(Simplification)} \quad (13)$$

From lines (12) and (13), we have a contradiction of $a > a$ and $b > b$, therefore f n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$. $\square$

Problem 3 Summary

Procedure

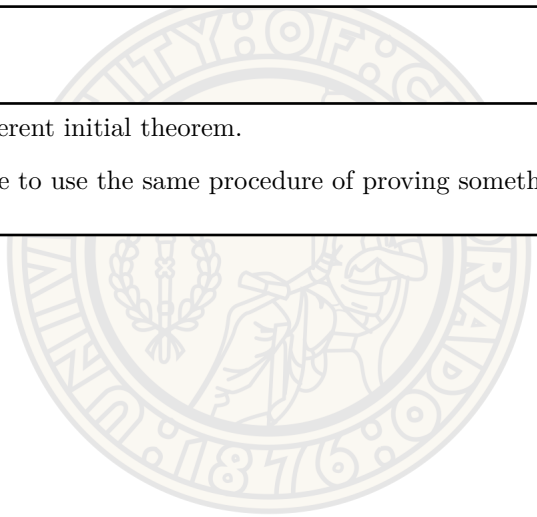
- For part (a), show examples of the theorem.
- For part (b):
  - Begin by stating the negation of the theorem.
  - Assume the negation and proceed to attempt to prove the negation.
  - Arrive at a contradiction.

Key Concepts

- This problem incorporates a proof by contradiction.
- To prove something by contradiction, we first must state the negation of the theorem.
- We then proceed to assume the negation and attempt to prove the negation with a direct proof.
- We then arrive at a contradiction and state that we have arrived at a contradiction and thus the original theorem must be true.

Variations

- We could be given a different initial theorem.
  - We would then have to use the same procedure of proving something by negation with this new theorem.



Problem 4

Problem Statement

Recall that we can think of modulo in terms of scoops and cups. The Euclidean Algorithm can also apply to scoops and cups and solves the question:

- "Given 2 measuring scoops, of a cups and b cups, what is the smallest measure in cups that can be made?"
- For example, given a 16 cup measure and a 25 cup measure.
- Fill 25 and remove 16, you are left with 9 cups. Pour the 9 cups into a new measuring scoop and mark it.
- Fill the 16 cup and pour into the new 9 cup measure. You are left with 7 cups. Pour the 7 cups into a new measuring scoop and mark it.
- Fill the 9 cup measure and pour into the new 7 cup measure . You are left with 2 cups. Pour the 2 cups into a new measuring scoop and mark it.
- Fill the 7 cups measure and pour it into the 2 cups measure 3 times, You are left with 1 cup.
- You now have a one cup measure.

You must bake a cake and need to measure exactly 1 cup. Can you make the cake given:

(a) 2 scoops measuring 17 and 88?

(b) 2 scoops measuring 35 and 88?

(c) 2 scoops measuring 37 and 53?

Solution

(a) 2 scoops measuring 17 and 88?

$$\begin{aligned} 88 &= 17 \cdot 5 + 3 \\ 17 &= 3 \cdot 5 + 2 \\ 3 &= 2 \cdot 1 + 1 \\ 2 &= 2 \cdot 1 + 0 \end{aligned}$$

From the above we see that the gcd is 1 and therefore we can make the cake.

(b) 2 scoops measuring 35 and 88?

$$\begin{aligned} 88 &= 35 \cdot 2 + 18 \\ 35 &= 18 \cdot 1 + 17 \\ 18 &= 17 \cdot 1 + 1 \\ 17 &= 17 \cdot 1 + 0 \end{aligned}$$

From the above we see that the gcd is 1 and therefore we can make the cake.

(c) 2 scoops measuring 37 and 53?

$$\begin{aligned} 53 &= 37 \cdot 1 + 16 \\ 37 &= 16 \cdot 2 + 5 \\ 16 &= 5 \cdot 3 + 1 \\ 5 &= 5 \cdot 1 + 0 \end{aligned}$$

From the above we see that the gcd is 1 and therefore we can make the cake.

Problem 4 Summary

Procedure

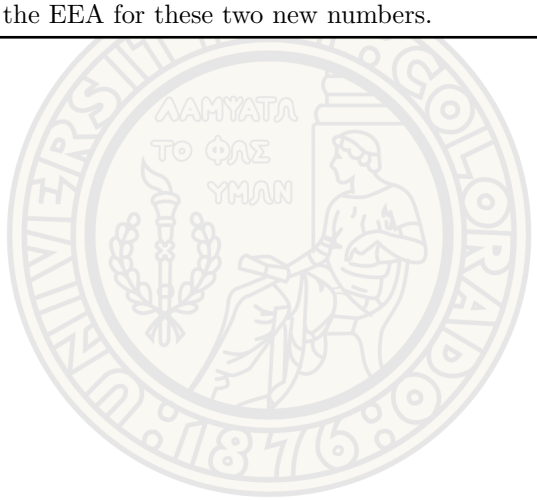
- Use the EEA (Extended Euclidean Algorithm) to find the GCD (Greatest Common Divisor) of the two scoops.

Key Variations

- The EEA calculates the GCD of two numbers.
- To use the EEA, we need to calculate the remainder of a dividend and a divisor.
  - After this, the divisor then becomes the dividend and the remainder becomes the quotient.
  - Repeat the above process until the remainder is zero.
- The last non zero remainder is the GCD of the two numbers.

Variations

- We could be given a different set of numbers of which we are asked to calculate the GCD.
  - We would then use the EEA for these two new numbers.



Problem 5

Problem Statement

Following the Square and Mod video, create a "Square and Mod Chart" for powers of 5 mod 17 , then solve the mod problems on the next page with this method.

Show some evidence of the process you are using to get the values below.

(a) 5^0 **mod** 17

(b) 5^1 **mod** 17

(c) 5^2 **mod** 17

(d) 5^4 **mod** 17

(e) 5^8 **mod** 17

(f) 5^{16} **mod** 17

(g) 5^{32} **mod** 17

(h) 5^{64} **mod** 17

(i) 5^{128} **mod** 17

(j) 5^{256} **mod** 17

Solution

(a) 5^0 **mod** $17 = 1$ **mod** $17 = \underline{1}$

(b) 5^1 **mod** $17 = 5$ **mod** $17 = \underline{5}$

(c) 5^2 **mod** $17 = 25$ **mod** $17 = \underline{8}$

(d) 5^4 **mod** $17 = 5^2 5^2$ **mod** $17 = (5^2$ **mod** $17)(5^2$ **mod** $17)$ **mod** $17 = (8)(8)$ **mod** $17 = 64$ **mod** $17 = \underline{13}$

(e) 5^8 **mod** $17 = 5^4 5^4$ **mod** $17 = (5^4$ **mod** $17)(5^4$ **mod** $17)$ **mod** $17 = (13)(13)$ **mod** $17 = 169$ **mod** $17 = \underline{16}$

(f) 5^{16} **mod** $17 = 5^8 5^8$ **mod** $17 = (5^8$ **mod** $17)(5^8$ **mod** $17)$ **mod** $17 = (16)(16)$ **mod** $17 = 256$ **mod** $17 = \underline{1}$

(g) 5^{32} **mod** $17 = 5^{16}5^{16}$ **mod** $17 = (5^{16}$ **mod** $17)(5^{16}$ **mod** $17)$ **mod** $17 = (1)(1)$ **mod** $17 = \underline{1}$

(h) 5^{64} **mod** $17 = 5^{32}5^{32}$ **mod** $17 = (5^{32}$ **mod** $17)(5^{32}$ **mod** $17)$ **mod** $17 = (1)(1)$ **mod** $17 = \underline{1}$

(i) 5^{128} **mod** $17 = 5^{64}5^{64}$ **mod** $17 = (5^{64}$ **mod** $17)(5^{64}$ **mod** $17)$ **mod** $17 = (1)(1)$ **mod** $17 = \underline{1}$

(j) 5^{256} **mod** $17 = 5^{128}5^{128}$ **mod** $17 = (5^{128}$ **mod** $17)(5^{128}$ **mod** $17)$ **mod** $17 = (1)(1)$ **mod** $17 = \underline{1}$

Synopsis

In the above examples, I used the formula that was from the lecture and recycled previous results. I was able to reuse results from previous statements with the law of exponents.

5^0 **mod** 17	5^1 **mod** 17	5^2 **mod** 17	5^4 **mod** 17	5^8 **mod** 17	5^{16} **mod** 17
1	5	8	13	16	1

All further modulos with this format can be expanded and written in a form where the results from this table can be used to determine the outcome.

Problem 5 Summary

Procedure

- Use the mathematical formula for FME on the 'simpler' numbers.
- Write the more complicated calculations in a manner such that they mimic the easier calculations.
- Use the results of these simpler calculations to calculate the more complex calculations.
- Create a chart of the results from the easier calculations.

Key Concepts

- This problem utilizes the FME (Fast Modular Exponentiation) technique. This is precisely

$$a^b \textbf{ mod } m.$$

- We use the results from the simpler calculations to simplify the more complex calculations.
- We can create a table of simpler calculations that can be used to calculate more complicated calculations.

Variations

- We could be asked to find the FME of different numbers where we build upon simpler calculations.
  - In this case, we would create a table of the simpler calculations.
  - We would then re-write the more complicated examples in terms of the simpler results.
  - Then finally we would use the table of simpler results to calculate the more complicated results.

Problem 6

Problem Statement

Use the chart and the Square 'n Mod method from the video to find:

(a) 5^{270} **mod** 17

(b) 5^{186} **mod** 17

(c) 5^{411} **mod** 17

Show all steps for each of the 3 problems.

5 mod 17 Chart

5^0 **mod** 17	5^1 **mod** 17	5^2 **mod** 17	5^4 **mod** 17	5^8 **mod** 17	5^{16} **mod** 17
1	5	8	13	16	1

All further modulos with this format can be expanded and written in a form where the results from this table can be used to determine the outcome.

Solution

(a) 5^{270} **mod** 17 $= 5^{256}5^{14}$ **mod** 17 $= (5^{16})^2 5^{14}$ **mod** 17 $= (5^{16})^2 5^8 5^4 5^2$ **mod** 17 $= (1)^2(16)(13)(8)$ **mod** 17 $= 1664$ **mod** 17 $= \underline{15}$

(b) 5^{186} **mod** 17 $= 5^{176}5^{10}$ **mod** 17 $= (5^{16})^{11}5^8 5^2$ **mod** 17 $= (1)^{16}(16)(8)$ **mod** 17 $= 128$ **mod** 17 $= \underline{9}$

(c) 5^{411} **mod** 17 $= 5^{400}5^{11}$ **mod** 17 $= (5^{16})^{25}5^8 5^3$ **mod** 17 $= (5^{16})^{25}5^8 5^2 5^1$ **mod** 17 $= (1)^{25}(16)(8)(5)$ **mod** 17 $= 640$ **mod** 17 $= \underline{11}$

In this problem, I tried to first find an even multiple of 16 in the exponent. I then rewrote the exponent in terms of this multiple and continued to use the law of exponents to further break down the problem so that I could use my lookup table. For multiples of 16 that were greater than 2, I wrote the exponentiation in a simplified form of a number raised to an exponent and then that number raised to another exponent. Since 5^{16} **mod** 17 $= 1$, we just got 1 raised to the power of another and that is always 1.

Problem 6 Summary

Procedure

- Write the FME calculations in a simpler form where we can use the results from the table to perform the calculations.

Key Concepts

- This problem utilizes writing complicated FME problems in a simpler form so that we can use results from simpler calculations to perform the complicated calculations.

Variations

- We could be given a different table of simple FME calculations to perform complicated FME calculations.
  - In this case we would write the complicated calculations in a simpler form and then use the results from the simpler calculations.

Problem 7

Problem Statement

Annotate Algorithm 5 and example 12 on page 254. Use the notes below to guide the annotation (either answer on the page or include in your annotation), then answer the questions that follow.

In example 12 on page 254:

(a) What is j?

(b) What is i?

(c) How is the algorithm here similar to Square and Mod?

(d) How is it different?

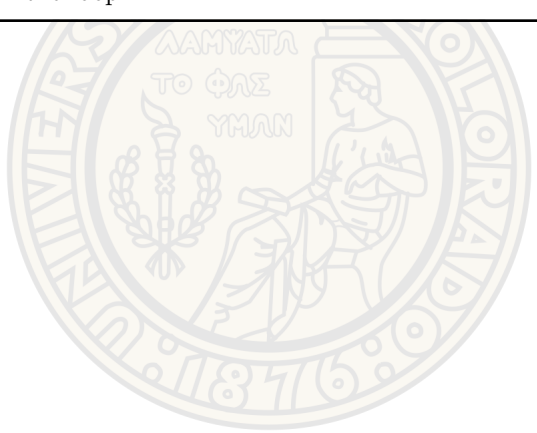
Look at Algorithm 5.

(a) What are the inputs?

(b) What is x?

(c) If $a_i = 1$ what happens in the loop?

(d) If $a_i = 0$ what happens in the loop?



Problem 8

Problem Statement

Review Sriram's video where he provides pseudocode for a FME algorithm.

- How is this algorithm is different from the book's algorithm?
- Will the loop in the algorithm terminate? What needs to be added to ensure it will terminate?

Problem 9

Problem Statement

Choose an FME algorithm to code:

- Book Version, Sriram's Version, Your own based on Square 'n Mod

(You can use section 4.2 exercises p. 255, #25-#28 to test your code.)
Briefly, explain why you choose which algorithm you did. Include a screenshot of your COMMENTED code (no black backgrounds)

Solution

I chose to use the books version on page 254 from **Section 4.2 - Integer Representation And Algorithms**. For me, this algorithm made more sense than that of Siriam's and I did not want to create a look up table or something similar for my own version. For my FME algorithm, I first made an algorithm that would calculate a binary number from a decimal number since the book's version of FME required that n be in base 2. I have included both algorithms in the code snippet below.

```
import math as math
# DecToBin - Converts a decimal number to binary
# Input:
#   n - Integer value that is to represent a number in decimal that is to be converted to binary
# Algorithm:
#   * Create an empty array for the digits of the binary number to be stored in
#   * While n is greater than 0, append the value of n mod 2 to the array
#       * Update the value of n by taking the floor of n divided by 2
#   * Reverse the array so that the digits are in correct order of the binary number
#   * Return the array
# Output:
#   This function returns an array of digits that represents a binary number
def DecToBin(n):
    array = []
    while(n > 0):
        array.append(n % 2)
        n = math.floor(n // 2)
    array.reverse()
    return array

# FME - A function to calculate Fast Modular Exponentiation (b^n mod m)
# Input:
#   b - Integer value that represents the base of of the FME (b) in (b^n mod m)
#   n - Integer value that represents the exponent value in FME (n) in (b^n mod m)
#   m - Integer value that represents the value that is being modulod against in FME (m) in (b^n mod
 m)
# Algorithm:
#   * Calculate the binary representation of n with DecToBin
#   * Reverse the binary number so that the last digit appears first in the array
#   * Set the return value (x) to 1
#   * Calculate power by taking the modulo of b and m
#   * Traverse the array of binary digits up to the length of the binary number
#       * If the current digit is a one, calculate x with (x * power) % m
#       * Update the power variable with (power * power) % m
#   * Return x after the for loop finishes completion
# Output:
#   x - Integer value that is returned from the expression of (b^n mod m)
def FME(b,n,m):
    DecInBin = DecToBin(n)
    DecInBin.reverse()
    x = 1
    power = b % m
    for i in range(0, len(DecInBin)):
        if (DecInBin[i] == 1):
            x = (x * power) % m
        power = (power * power) % m
    return x

```

Problem 9 Summary

Procedure

- Write code in Python for a function that converts a decimal number to binary.
- Write code in Python for a function that performs FME.
- Write a brief summary on the functions and how they were written.

Key Concepts

- This problem encapsulates how to write code in Python for an RSA project.

Variations

- We could be asked to write code for a different set of functions.
  - We would then write functions for these new operations.

Problem 10

Problem Statement

To motivate our use of Fast Modular Exponentiation FME, let's do a comparison of two ways we can calculate a^n **mod** m.

Write a new function called **NOT_FME** that returns the result of a^n **mod** m, using just the standard exponent calculation and the built in Python Mod function **mod** .

Use the variables, a, n, m a^n **mod** m.

Then use the Python function **pow()**, which calculates the same result using FME.

Do your own informal investigation to see if FME has a greater impact on the runtime when n is getting large or when a is getting large. (Hold one variable fixed and then explore with the other, then reverse).

Summarize your results on this page and use as many pages as you need to explore.

Solution

For starters, here is my function **NOT_FME1** that utilizes the standard exponentiation calculation that is built in with Python.

```
def NOT_FME1(a,n,m):
    return a**n % m

```

Next, we have my function **NOT_FME2** that utilizes the **pow()** function in Python for exponentiation calculations.

```
def NOT_FME2(a, n, m):
    return math.pow(a, n) % m

```

I tested these functions with the following loops and printed the time for their efficiencies.

```
end_of_loop_val = 100
start_time1 = time.time()
for i in range(0, end_of_loop_val):
    result = NOT_FME1(5, i, 17)
end_time1 = time.time()
start_time2 = time.time()
for i in range(0, end_of_loop_val):
    result = NOT_FME2(5, i, 17)
end_time2 = time.time()
elapsed_time1 = end_time1 - start_time1
elapsed_time2 = end_time2 - start_time2
print(f"Elapsed time for NOT_FME1: {elapsed_time1} seconds")
print(f"Elapsed time for NOT_FME2: {elapsed_time2} seconds")

```

I constantly tweaked the value 'end_of_loop_val' to see how their efficiencies would change. And one thing that I noticed with my functions was that for a large enough value of 'end_of_loop_val', **NOT_FME2** would break because the **pow()** function in Python is not able to handle large values for exponentiation. I found this interesting because I figured that it would be the other way around in that the built exponentiation calculation would fail at large values but it didn't. In fact, the largest value for 'end_of_loop_val' that I could set it to for both functions to run was 442. This means the exponent could only go up to 442 before it broke the **pow()** function.

Since I determined that **NOT_FME1** could handle larger numbers in the exponent (although, this could be running into the case where we are having an overflow for the integer values) I decided to test it against the **FME** algorithm with 'end_of_loop_val = 100000'. After running this test, I got the following output in my terminal:

```
Elapsed time for NOT_FME1: 54.408992767333984 seconds
Elapsed time for FME: 0.1533365249633789 seconds

```

We can clearly see that **FME** is lightyears faster than **NOT_FME1** in terms of raising a number to an exponent. We then shift to changing the values of a instead of n. In one final comparison of these algorithms, the following tests were ran.

```
end_of_loop_val = 100000
start_time1 = time.time()
for i in range(0, end_of_loop_val):
    result = NOT_FME1(i, 8, 17)
end_time1 = time.time()
elapsed_time1 = end_time1 - start_time1
start_time2 = time.time()
for i in range(0, end_of_loop_val):
    result = NOT_FME2(i, 8, 17)
end_time2 = time.time()
elapsed_time2 = end_time2 - start_time2
start_time3 = time.time()
for i in range(0, end_of_loop_val):
    result = FME(i, 8, 17)
end_time3 = time.time()
elapsed_time3 = end_time3 - start_time3
print(f"Elapsed time for NOT_FME1: {elapsed_time1} seconds")
print(f"Elapsed time for NOT_FME2: {elapsed_time2} seconds")
print(f"Elapsed time for FME: {elapsed_time3} seconds")

```

This produced one final output between all three algorithms.

```
Elapsed time for NOT_FME1: 0.01721668243408203 seconds
Elapsed time for NOT_FME2: 0.024130344390869 14 seconds
Elapsed time for FME: 0.04331684112548828 seconds

```

This shows that when a gets very large, it is actually **NOT_FME1** and **NOT_FME2** are more efficient in terms of time compared to that of **FME**. However, this is not taking into account for integer overflow in the **NOT_FME1** and **NOT_FME2** algorithms that may occur since **FME** is using binary expansion for its calculations. Overall, it is still probably more wise to use **FME** than it is to use the other methods.

Problem 10 Summary

Procedure

- Write code for different emulations of FME.
- Create an analysis of these methods.
- Comment on these methods

Key Concepts

- This problem showcases the different methods for calculating FME.
- FME is shown to be the most efficient method for calculating FME.

Variations

- We could be asked to compare other functions and analyze their differences.
  - We would then use the same process of comparing and contrasting the different functions.