Exercises, chapter 5, solutions

---

1. First, apply the fast method for prime factorization to n to obtain its two prime factors p and q. Next, compute $\varphi(n) = (p-1)(q-1)$. Finally, run the extended Euclid's algorithm to calculate d from the two numbers e and $\varphi(n)$ just like in step 4 of the description of RSA in Chapter 5.2.

2. (a)

a	b	$a \bmod b$
1305	1030	275
1030	275	205
275	205	70
205	70	65
70	65	5
65	5	0

The gcd is: 5

(b) Let $a \geq 0$ and $b \geq 1$ be integers.

First, consider any integer c that divides both a and b. By definition, $a = k_1 \cdot c$ and $b = k_2 \cdot c$ for some integers k_1 and k_2. Also, $(a \bmod b) = a - k_3 \cdot b$ for some integer k_3, so we can write $(a \bmod b) = k_1 \cdot c - k_3 \cdot k_2 \cdot c = (k_1 - k_3 \cdot k_2) \cdot c$. Since $k_1 - k_3 \cdot k_2$ is an integer, c divides $(a \bmod b)$.

Next, consider any integer d that divides both b and $(a \bmod b)$. As above, $b = \ell_1 \cdot d$ and $(a \bmod b) = \ell_2 \cdot d$ for some integers ℓ_1 and ℓ_2. From before, $(a \bmod b) = a - k_3 \cdot b$, which gives $a = k_3 \cdot b + (a \bmod b) = k_3 \cdot \ell_1 \cdot d + \ell_2 \cdot d = (k_3 \cdot \ell_1 + \ell_2) \cdot d$. Here, $k_3 \cdot \ell_1 + \ell_2$ is an integer, so d divides a.

We have just shown that the common divisors of a and b are the same as the common divisors of b and $(a \bmod b)$. In particular, the largest number in these two sets is the same, which means that $gcd(a, b) = gcd(b, a \bmod b)$.

3. The answer will vary from student to student. An example of a valid string T for a person (?) with the name "Introduction_to_Algorithms" is:

- `2024-Introduction_to_Algorithms-6` (The first five bits of $SHA_{256}(T)$ are 0.)

The problem can be solved by brute-force search. If we only need the first four bits of $SHA_{256}(T)$ to be equal to 0, we can do it by hand since the probability that a randomly selected number X makes the first four bits 0 is $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = 1/16$. It follows from probability theory that the number of times one has to choose an X until a suitable one is found is a geometric random variable whose expected value is $\frac{1}{1/16} = 16$. Thus, it's feasible to check different values of X to include in T by hand with an online tool such as `https://emn178.github.io/online-tools/sha256.html` until a good X is found. (We were slightly lucky above and only needed 6 tries, rather than the expected 16.) Note, however, that to find an X that makes more bits of $SHA_{256}(T)$ equal to 0, it is better to write a computer program that can try various values of X for us. This gives, e.g.,:

- `2024-Introduction_to_Algorithms-241` (The first eight bits of $SHA_{256}(T)$ are 0.)
- `2024-Introduction_to_Algorithms-9435049` (The first twenty-four bits of $SHA_{256}(T)$ are 0.)

Remark:
A number such as X in our problem that modifies the input to a cryptographic hash function is called a *nonce*. In general, finding a nonce that makes $SHA_{256}(T)$ satisfy some given requirements may require a lot of work, while verifying if a given nonce is good enough is much easier. This idea forms the basis of bitcoin mining, for example, where so-called "miners" compete against each other to be the first to find a good nonce for some set of data that encodes recently conducted financial transactions.