Algorithm

written by VickysLearning on Functor Network
original link: https://functor.network/user/2930/entry/1002

This post summarizes what I've learned from The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd ed.) by Donald E. Knuth (1997), Chapter 1 Section 1 [p. 1-9]. The exercises, algorithms and some answers discussed are drawn from this book.

Algorithms are similar to recipes or routines. However, rather than being executed by a person, algorithms are executed by a computer. Recipes and algorithms are quite similar in a sense that they both consist of finite sequence of operations designed to solve a problem. The key distinctions between recipes and algorithms are outlined by the following five features:

Finiteness: The sequence of operations must be finite and must eventually solve the problem.

Definiteness: Each step of an algorithm must be precisely defined

Input: There can be zero or many inputs.

Output: An algorithm must produce at least one output, which is the result of the problem the algorithm is trying to solve.

Effectiveness: The operations should be simple enough to be performed with just pen and paper and should have a finite length.

When an algorithm lacks finiteness, it is referred to as a computational method. One can define computation methods in terms of set theory.

Definition 1. *A* computational method *is a quadruple (Q, I, Ω, f), where Q is a a set represents the states of the computation, $I, \Omega \subseteq Q$ are the input and the outputs set respectively, and $f : Q \to Q$ is the computational rule. The element $x \in I$ represents a computational sequence $x_0, x_1, x_2, \ldots$, where*

$$x_0 = x \qquad \text{and} \qquad x_{k+1} = f(x_k) \quad \text{for } k \geq 0.$$

The computational sequence is terminated *in k steps if k is the smallest integer such that $x_k \in \Omega$. In* algorithms*, the termination step should be finite for all $x \in I$.*

Note that Definition 1 does not address the feature of effectiveness. To restrict the method to elementary operations, we impose additional constraints on (Q, I, Ω, f). For example: let A be a set of finite letters and A^* be the set of all strings on A. Let $N \in \mathbb{N}$ and Q be the set of all (σ, j), where $\sigma \in A^*$ and $j = 0, 1, 2, \ldots N$. Let $I \subseteq Q$ with $j = 0$, and let $\Omega \subseteq Q$ with $j = N$. If $\theta, \sigma \in A^*$, we say that θ occurs in σ if $\sigma = \alpha\theta\omega$ for some strings $\alpha, \omega \in A^*$.

Lastly, let $\theta_j, \phi_j \in A^*$, integers $a_j, b_j \in \mathbb{Z}$, for any $0 \leq j < N$, and $f : Q \to Q$ defined by

$$\begin{aligned}
&f\left((\sigma, j)\right) = (\sigma, a_j) && \text{if } \theta_j \text{ does not occur in } \sigma \\
&f\left((\sigma, j)\right) = (\alpha\phi_j\omega, b_j) && \text{if } \alpha \text{ the shortest possible string for which } \sigma = \alpha\theta_j\omega \\
&f\left((\sigma, j)\right) = (\sigma, N) &&
\end{aligned}$$

After doing the exercises, it seems clear what some of the notations represent. The positive integer a_j indicates which rule should be executed when a string θ_j from the j-th rule does not occur in σ. Furthermore, it seems that b_j refers to the next step to be executed when a string θ_j is replaced by a string ϕ_j in σ. At N, the algorithm terminates and the output is σ.

One example of an algorithm is the Euclid's algorithm for finding the greatest common divisor.

Algorithm 1 (Euclid's algorithm)**.** *Let $m, n \in \mathbb{Z}$, find their greatest common divisor, that is , the largest positive integer such that it divides both m and n.*
E1. [Ensure $m \geq n$.] If $m < n$, exchange $m \leftrightarrow n$.
E2. [Find remainder.] Divide m by n and let $0 \leq r < n$ be its remainder.
E3. [Is it zero?] If $r = 0$, the algorithm terminates; n is the output.
E4. [Reduce.] Set $m \leftarrow n, n \leftarrow r$, and go back to E2.

Let us try this algorithm by hand. Let $m = 10$ and $n = 122$, then we exchange $m \leftrightarrow n$ because otherwise the reminder is zero, which is not their greatest common divisor. Then the reminder of 122/10 is $r = 2$, since $122 = 12 \cdot 10 + 2$. Then we set $m \leftarrow 10$ and $n \leftarrow 2$, which has zero remainder because $10/2 = 5$. Therefore, the greatest common divisor of 122 and 10 is 2. Note that in Algorithm 1 step E4, we don't go back to E1, since $m > n$ always after the first step. This is proven in the Exercise 2.

As a remark about Algorithm 1, the notations that is used here will be used for other posts as well. The notation $m \leftarrow n$ means that we replace the current value of m with the current value n. We are not using $=$ since $m = n$ will mean something that can be checked. The exchange notation $m \leftrightarrow n$ can also be done by introducing an auxiliary variable, say t. Indeed, $m \leftrightarrow n$ is equivalent to $t \leftarrow m, m \leftarrow n, n \leftarrow t$. Lastly, the order of assignments is important. For instance $m \leftarrow n, n \leftarrow r$ is different than $n \leftarrow r, m \leftarrow n$. Indeed, the latter means $n \leftarrow r, m \leftarrow r$, which is equivalent to $n \leftarrow m \leftarrow r$.

0.0.1 Exercises

1. [10] To rearrange (a, b, c, d) to (b, c, d, a), we can do $t \leftarrow a, a \leftarrow b, b \leftarrow c, d \leftarrow t$.

2. [15] The positive integer m is always greater than n at the beginning of step E1, except possibly the first time this step occurs. Indeed, after it occurs for the first time, we have $m = qn + r$, where $0 \leq r < n$. After that, as long as $r \neq 0$, we replace m by n and n by r. Since $n > r$, we have that $m > n$ after the first step.

3. [20] We can change Algorithm E so that the replacement operations, such as $latexm \leftarrow n$, are avoided. Consider the example $m = 500$ and $n = 111$. According to Algorithm E, we have 500/111, which has a remainder $r = 56$. Instead of replacing $m \leftarrow n$, we can set $m \leftarrow 56$ and compute n/m. Then, 111/56 has a remainder $r = 55$. Because $r \neq 0$, in Algorithm E, we compute 56/55. Because $m \leftarrow 56$, we can this time set $n \leftarrow 55$ and therefore we are computing the remainder of m/n, which is 1. Lastly, if we set $m \leftarrow 1$, the remainder of n/m is zero. Set $n \leftarrow 0$, then m is the output. In the above procedure, we do not use $m \leftarrow n$. To summarize, we have the following updated algorithm:

   Algorithm 2. Let $m, n \in \mathbb{Z}$, find their greatest common divisor, that is , the largest positive integer such that it divides both m and n.
   F1. [Ensure $m \geq n$.] If $m < n$, exchange $m \leftrightarrow n$.
   F2. [Find remainder.] Divide m by n. Assign the remainder r to m, i.e. $m \leftarrow r$.
   F3. [Is it zero?] If $m = 0$, the output is n.
   F4. [Find remainder.] Divide n by m. Assign the remainder r to n, i.e. $n \leftarrow r$.
   F5. [Is it zero?] If $n = 0$, the output is m. Repeat to F2. There is no need to go back to F1, since $m > n$ always after the first exchange as proven in Exercise 2.

4. [16] The remainder of 6099/2166 is 1767. Then, the remainder of 2166/1767 is 399. Then, the remainder of 1767/399 is 171. Then, the remainder of 399/171 is 57. Lastly, the remainder of 171/57 is zero. Therefore, the greatest common divisor of 2166 and 6099 is 57.

5. [12] The "Procedure for Reading This Set of Books" is not an algorithm because it is not definite, effective and does not have an output. An example of why definiteness is lacking is in step 15, where it is not clearly defined where or for how long to sleep. Regarding the format difference with Algorithm 1, there is no letter in front of each step, and there is no short summary after each number.

6. [20] Let T_n be the average number of times step E2 (without executing E1) is performed. Let $n = 5$, then we need to compute the number of times steps E2 is performed for $m = 1, 2, 3, 4, 5$ since in step E2, only the remainder is relevant. Writing out how many times E2 is performed for $m = 1, 2, 3, 4, 5$ yields $T_5 = \frac{2+3+4+3+1}{5} = 2.6$.

7. [HM21] Suppose m is known and n is allowed to range over all positive integers. Let U_m be the average number of times step E2 is executed in Algorithm 1. The average U_m is well-defined because the positive integer keeps being reduced until the remainder is zero, at which point the algorithm terminates. Therefore U_m will not be ∞. Regarding the relation of U_m with T_m, consider the case $m = 5$ is fixed. Writing out the execution

of step E2, we will notice that for $n > m$, the number of times step E2 is executed increases by 1 compared to when $n = 5$ is fixed. Therefore $U_m = T_m + 1$. To confirm this, we compute the number of steps manually for $m = 5$. First, for $n = 1, 2, 3, 4, 5$, the number of times step E2 is executed are $1, 2, 3, 2, 1$ respectively. Then, for every $q > 1$ such that $n = 5q + \{1, 2, 3, 4, 5\}$, the number of times step E2 is executed are $3, 4, 5, 4, 2$. Suppose we do this q many times, then

$$\begin{aligned} U_5 &= \lim_{q\to\infty} \frac{q(3+4+5+4+2)+1+2+3+2+1}{5q+5} \\ &= \lim_{q\to\infty} \frac{q([2+1]+[3+1]+[4+1]+[3+1]+[1+1])+9}{5q+5} \\ &= \lim_{q\to\infty} \left(\frac{q(2+3+4+3+1)}{5q+5} + \frac{5q+9}{5q+5} \right) \\ &= \frac{2+3+4+3+1}{5} + 1 \\ &= T_5 + 1. \end{aligned}$$

8. [M25] The goal of this part of the exercise, is to rewrite the Algorithm 1 without step E1 into an "effective" formal algorithm. The given input is $a^m b^n$, and the question is, how should we define ϕ_j, θ_j, a_j and b_j? There is a hint that states to start with $r \leftarrow |m - n|, n \leftarrow \min(m, n)$. However, we do not see how the hint is related to the input. After looking at the answers and doing some research on the internet, we still don't see how the hint is suppose to help. Though, it clarifies how the algorithm proposed in the answer is suppose to mimic Euclid's algorithm.

   The idea is to adjust the input, such that the output is $a^{gcd(m,n)}$. To do this, there are sets of rules that we need to specify. These rules contain simple operations such that removing, adding and replacing strings. The rule $r \leftarrow |m - n|$ can be represented in terms of strings, by removing a's and b's until either of them disappears, depending on whether $m > n$ or $n < m$. For instance $|5 - 3|$ can be programmed by removing both a and b until a^2 remains, and therefore the answer is 2.

   Now, how about $n \leftarrow \min(m, n)$? We connect this substitution with the answer as follows: there needs to be a record of how much we subtract n from m. This is because we will use this number again to compute $|m-n|$. To do this, for instance, if $m = 5$ and $n = 3$, then we remove a's three times, and b's will disappear. We then add a new letter c^3, since we do removal operations three times and we are left with c^3a^2. If $m = 3$ and $n = 5$, then we are left with c^3b^2. To ensure that there is a loop and the string is of the form $a^m b^n$, we can replace all c's by a's and all a's by b's.

   In summary, we perform the following algorithm with the input $a^n b^n$. Let

$A = \{a, b, c\}$ and $1 \leq j < N = 5$. Define

	θ_j	ϕ_j	a_j	b_j
$j = 0$	ab	$\varnothing$	2	1
$j = 1$	$\varnothing$	c	0	0
$j = 2$	a	b	3	2
$j = 3$	c	a	4	3
$j = 4$	b	b	5	0

9. [M30] Suppose $C_1 = (Q_1, I_1, \Omega_1, f_1)$ and $C_2 = (Q_2, I_2, \Omega_2, f_2)$ are computational methods. The goal is to formulate a set-theoretic definition for the concept "C_2 is a representation of C_1". Somehow, there should be functions that translate objects from C_1 to C_2. In this exercise, we tried to understand how the answer sheet explain one approach to solve this problem.

First let us discuss the inputs and the outputs. The inputs in I_1 should be the inputs of I_2 as well. Though they must be embedded in their computational states. To formalize this, let $g : I_1 \to I_2$ and $h : Q_2 \to Q_1$. These functions must satisfy the constraint $h(g(x)) = x$ for $x \in I_1$. As for the outputs, we must ensure that the outputs of C_1 corresponds to the outputs of C_2, and vice versa. That is, $q \in \Omega_2$ if and only if $h(q) \in \Omega_1$. In other words, a computational state in C_2 is the outputs of the algorithm if and only if its translation via h are outputs in C_1.

Now how about the computational rules? Well, when translating a textbook algorithm into a computer program, we sometimes need several steps to convert the abstract rules into a form that a computer can execute. These rules in C_2, should translate the rules in C_1. Let $q \in Q_2$, we require that $f_1(h(q)) = h(f_2^j(q))$ for some $q \in \mathbb{Z}_{\geq 1}$.

Therefore, C_2 is a representation of C_1 if and only if there exists $h : Q_2 \to Q_1$, $g : I_1 \to I_2$, and $j \in \mathbb{Z}_{\geq 1}$, such that
a) $h(g(x)) = x$, where $\in I_1$;
b) $h(q) \in \Omega_1 \iff q \in \Omega_2$, where $q \in Q_2$;
c) $f_1(h(q)) = h(f_2^j(q))$ where $q \in Q_2$.

When reading this definition, one might wonder why the functions are not specified to be surjective, injective or bijective. For both of the functions, it might be because some states in C_1 may not have an exact counterpart in C_2, or the computational states in C_2 approximate the computational states in C_1. For instance, it is not possible to represent irrational numbers such as π and e in a computer. This is because computers can only store numbers using finite-length strings.