Randomized Algorithms 2023
Tutorial Sheet 1 (week 3)

A randomized algorithm for deciding whether two positive integers, represented succinctly by arithmetic circuits, are equal or not.

If we are given two positive integers written in binary, then it is obviously very easy for us to decide whether the two integers are equal or not.

However, if instead we are given two positive integers represented *succinctly*, using arithmetic circuits, deciding whether they are equal or not is a *much* more challenging task. In this tutorial you will be asks to devise a randomize polynomial time algorithm for this task and, importantly, to prove its correctness.

An *arithmetic circuit*, or equivalently a *straight-line program*, C, with m arithmetic gates, restricted to addition or multiplication gates only, and using only the input value 1, is defined by a sequence of arithmetic instructions of the following form:

$$\begin{aligned} g_0 &:= 1; \\ g_1 &:= g_{j_1} \odot_1 g_{k_1}; \\ &\dots \\ g_i &:= g_{j_i} \odot_i g_{k_i}; \\ &\dots \\ g_m &:= g_{j_m} \odot_m g_{k_m}; \end{aligned}$$

where, for all $i \in \{1, \dots, m\}$, $\odot_i \in \{+, *\}$, and both $j_i \in \{0, \dots, i-1\}$ and $k_i \in \{0, \dots, i-1\}$.

We define the "size", $\texttt{size}(C)$, of such a circuit C to simply be its number of gates m. Note that we can encode a circuit that has m gates using $O(m \log m)$ bits. Hence, as far as polynomial time algorithms are concerned, it is inconsequential whether we measure the size of an input arithmetic circuit by its number of gates or by its bit encoding length.

Each gate g_i of such an arithmetic circuit "computes" a unique postive integer, $\texttt{val}(g_i)$, in the natural way. We can define $\texttt{val}(g_i)$ by induction on i: for $i = 0$, $\texttt{val}(g_0) := 1$, and inductively, for $i > 0$, $\texttt{val}(g_i) := \texttt{val}(g_{j_i}) \odot_i \texttt{val}(g_{k_i})$.

We define the value, $\texttt{val}(C)$, of such a circuit C with m gates to simply be the value of its last gate, in other words, $\texttt{val}(C) := \texttt{val}(g_m)$.

Notice that with an arithmetic circuit C of $\texttt{size}(C) = n$ we can define double-exponentially large numbers using repeated squaring. Specifically, consider the following circuit C' given by:

$$
\begin{aligned}
g_0 &:= 1; \\
g_1 &:= g_0 + g_0; \\
g_2 &:= g_1 * g_1; \\
g_3 &:= g_2 * g_2; \\
&\ldots \\
g_n &:= g_{n-1} * g_{n-1};
\end{aligned}
$$

By an easy induction, for all $i \geq 1$, we have $\texttt{val}(g_i) = 2^{2^{i-1}}$. Hence $\texttt{val}(C') = 2^{2^{n-1}}$, even though $\texttt{size}(C') = n$. Notice that even just to write down the value $2^{2^{n-1}}$ in binary would require exponential space (2^{n-1} bits) as a function of the input circuit's size, n. Hence, it is not at all obvious how to "evaluate" an arbitrary arithmetic circuit in polynomial time.

Suppose we are given two arithmetic circuits, C_1 and C_2, where $\texttt{size}(C_1) = n_1$ and $\texttt{size}(C_2) = n_2$, and suppose we want to decide whether $\texttt{val}(C_1) = \texttt{val}(C_2)$.

Can we solve this decision problem efficiently (in polynomial time)? We can if we allow ourselves to use randomness.

This tutorial asks you to devise a randomized algorithm that, given as input two arithmetic circuits C_1 and C_2, of sizes n_1 and n_2, has the following properties:

(a.) The algorithm runs in time polynomial in the input size $n = n_1 + n_2$.

(b.) If $\texttt{val}(C_1) = \texttt{val}(C_2)$, then the algorithm always returns "YES".

(c.) If $\texttt{val}(C_1) \neq \texttt{val}(C_2)$, then the algorithm returns "NO" with probability at least 1/2.

(You are then asked to show that, using a standard repetition argument, if $\texttt{val}(C_1) \neq \texttt{val}(C_2)$ we can amplify the probability that the algorithm returns "NO" to $(1 - \frac{1}{2^n})$, while also maintaining both properties (a.) and (b.).)

In order to help you devise such an algorithm, we will provide the following "hints" as mathematical facts that you are allowed to use without proof. Proofs of the first three facts can be found in any good number theory textbook. For example, the following very nice book:
Victor Shoup, *A Computational Introduction to Number Theory and Algebra*, 2nd Edition, Cambridge University Press, 2008.
The fourth fact is about arithmetic circuits, and it is not hard to prove by induction on the size of the circuit. The last fact is a simple and well-known inequality relating to e, the base of the natural logarithm.

1. Let us briefly recall congruences and modular arithmetic. For integers $a, b \in \mathbb{Z}$, and a positive integer $N \geq 1$, we say that a is *congruent* to b modulo N, and we write

   $a \equiv b \pmod{N}$

   if N divides $a - b$, denoted $N \mid (a - b)$, or in other words if there exists an integer c such that $a - b = c \cdot N$.

   If $N \nmid (a - b)$, then we write $a \not\equiv b \pmod{N}$.

   Here are some useful facts about modular arithmetic:

   For any fixed modulus N, the congruence relation $\equiv \pmod{N}$ defines an equivalence relation on integers (i.e., a reflexive, symmetric, and transitive binary relation on integers). Furthermore, the congruence is "compatible" with integer addition and multiplication, meaning that, if $a \equiv a' \pmod{N}$ and $b \equiv b' \pmod{N}$, then $(a+b) \equiv (a'+b') \pmod{N}$, and $(a \cdot b) \equiv (a' \cdot b') \pmod{N}$.

2. For a positive integer $x \geq 1$, let $\pi(x)$ denote the total number of distinct prime numbers up to (and including) x.

   For example, $\pi(1) = 0$, $\pi(3) = 2$, and $\pi(9) = 4$.

   Let $\ln(x)$ denote the natural logarithm of the number $x > 0$. A fundamental theorem in number theory, the *prime number theorem*, states that $\pi(x) \sim \frac{x}{\ln x}$. In other words, $\lim_{x \to \infty} \frac{\pi(x)}{x/\ln x} = 1$. Rather than dealing with limits, we will be content to quote the following much easier to prove quantitative fact (one part of Chebyshev's theorem on the density of primes):

   Theorem: *For any positive integer* $x \geq 2$, $\pi(x) \geq \frac{1}{3} \cdot \frac{x}{\ln x}$.

3. Let us make a trivial number-theoretic observation:

   Let $\prod_{i=1}^{k} p_i$ denote the product of the first k distinct prime numbers, $p_1 = 2, p_2 = 3, p_3 = 5, \ldots, p_k$. Then clearly $\prod_{i=1}^{k} p_i \geq 2^k$. For the same reason, the product of *any* k distinct prime numbers must be $\geq 2^k$.

4. Let us state the following simple fact about arithmetic circuits. For any arithmetic circuit C of size n, $\texttt{val}(C) \leq 2^{2^{n-1}}$. In other words, the largest integer that can be computed by an arithmetic circuit of size n is $2^{2^{n-1}}$. (This is fairly easy to prove.)

5. Finally, let us state a basic inequality. Let e be the base of the natural logarithm. Then $\frac{1}{e} \geq (1 - \frac{1}{m})^m$, for all $m \geq 1$.

Using the above facts (none of which you need to prove), devise a randomized algorithm, i.e., an algorithm that is able to flip fair (independent) coins, for deciding whether two

integers given succinctly by two arithmetic circuits are equal or not, and prove that your algorithm has properties (a.), (b.), and (c.).

Hints:

- Consider choosing, uniformly at random, a "large enough" (but not "too large") random integer $N \geq 1$, and computing the value of both arithmetic circuits C_1 and C_2 modulo N, and then comparing those modulo N values with each other to see whether or not they are equal. Note that using Fact (1.) we can compute the value of each arithmetic circuit modulo N, by inductively computing the value modulo N at each gate in each circuit in a "bottom up" fashion, starting from gate 0, then gate 1, etc.
- How large (i.e., how many bits long) should the random number N be, so that we can, on the one hand, compute the values of both circuits efficiently (in time polynomial in n) and, on the other hand, still be sure that if the two arithmetic circuits do not describe the same number, then there is a "reasonably good probability" that their value modulo N will be different? (Use Facts (2.), (3.), and (4.) for this.)
- Finally, use Fact (5.) to "amplify" the success probability of the algorithm to $(1 - \frac{1}{2^n})$, by using repetitions.

Food for thought: can you devise a *deterministic* polynomial time algorithm for the same decision problem? (*Hint*: you would immediately become famous if you could do so.)

Kousha Etessami