Lecture 3: Lower bounds for Circuits and Turing Machines

Anup Rao

April 2, 2024

At the end of lecture last time, we were discussing how changes to the model of Turing machines do not lead to big changes in the measures of complexity associated with Turing machines. There are a few results of this type:

Claim 1. *A program written for an L-tape machine that runs in time $T(n)$ can be simulated by a program with 1 input tape, 1 work tape and 1 output tape in time $O(L \cdot T(n)^2)$.*

Sketch of Proof The idea is to encode the contents of all the new work arrays into a single work tape. To do this, we can use the first L locations on the work tape to store the first bit from each of the L arrays, then the next L locations to store the second bit from each of the L arrays, and so on. To encode the location of the pointers, we increase the size of the alphabet so that exactly one symbol from each tape is colored red. This encodes the fact that the pointer points to this symbol of the tape. The actual pointer in the new Turing machine will then do a big left to right sweep of the array to simulate a single operation of the old machine. ■

Claim 2. *A program written using symbols from a larger alphabet Γ that runs in time $T(n)$ can be simulated by a machine using the binary alphabet in time $O(\log|\Gamma| \cdot T(n))$.*

Sketch of Proof We encode every element of the old alphabet in binary. This requires $O(\log|\Gamma|)$ bits to encode each alphabet symbol. Each step of the original machine can then be simulated using $O(\log|\Gamma|)$ steps of the new machine. ■

The following theorem should not come as a surprise to most of you. It says that there is a machine that can compile and run the code of any other machine efficiently:

Theorem 3. *There is a turing machine M such that given the code of any Turing machine α and an input x as input to M, if α takes T steps to compute an output for x, then M computes the same output in $O(CT\log T)$ steps, where here C is a number that depends only on α and not on x.*

We shall say that a machine runs in time $t(n)$ if for every input x, the machine halts after $t(|x|)$ steps (here $|x|$ is the length of the string x). Similarly, we can measure the space complexity of the machine. The crucial point is that small changes to the model of Turing machines does not affect the time/space complexity of computing a particular function in a big way. Thus it makes sense to talk about the running time for computing a function f, and this measure is not really model dependent.

Lower bounds—Counting arguments

WE HAVE SHOWN THAT every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size at most $O(2^n/n)$, and on the other hand we show that for n large enough there is a function that cannot be computed by a circuit of size less than $2^n/(3n)$. The lower bound we prove here was first shown by Shanon. He introduced a really simple but powerful technique to prove it, called a *counting argument*.

Theorem 4. *For every large enough n, there is a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $2^n/3n$.*

Proof We shall count the total number of circuits of size s, where $s > n$. To define a circuit of size s, we need to pick the logical operator for each (non-input) gate, and specify where each of its two inputs come from. There are at most 3 choices for the logical operation, and at most s choices for where each input comes from. So the number of choices for each non-input gate is at most $3s^2$. The number of choices for an input gate is at most $n < 3s^2$. So, the total number of choices for each gate is at most $3s^2 + n$, and the number of possible circuits of size s is at most

$$(3s^2 + n)^s \leq (4s^2)^s = 2^{s \log(4s^2)} < 2^{3s \log s},$$

when $n > 4$.

This means that the total number of circuits of size $2^n/3n$ is less than $2^{3 \cdot \frac{2^n}{3n} \cdot n} = 2^{2^n}$. On the other hand, the number of functions $f : \{0,1\}^n \to \{0,1\}$ is exactly 2^{2^n}. Thus, not all these functions can be computed by a circuit of size $2^n/(3n)$. ■

Indeed, the above argument shows that the fraction of functions $f : \{0,1\}^n \to \{0,1\}$ that can be computed by a circuit of size $2^n/4n$ is at most $\frac{2^{\frac{3}{4} \cdot 2^n}}{2^{2^n}} = \frac{1}{2^{2^{n-2}}}$, which is extremely small.

Similar arguments can be used to show that not every function has an efficient branching program (as you will do on your homework).