Lecture 3: Counting arguments, Turing machines

Anup Rao

January 10, 2023

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*.

The basic idea is to count the number of circuits of size s, and count the number of functions that depend on n input bits. If the number of functions is larger, then there must be a function that cannot be computed by a circuit of size s. In order to get results, we will not actually have to be too precise about counting the number of circuits of size s, it will be enough to get a reasonable upper-bound.

Theorem 1. *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).

Not every function has an efficient communication protocol

LET US SEE another example of how counting arguments can be used to prove lower bounds.

Recall that a communication protocol for computing a function $f(x, y)$ specifies a way for Alice and Bob to communicate with each other in order to compute f. If x, y are n-bit strings, the number of such functions f is $2^{2^{2n}}$: indeed there are 2^{2n} inputs x, y, and for each choice of input, there are 2 choices for the output of the function.

The trivial protocol for computing an arbitrary function is to have Alice send her entire input to Bob. This takes n bits of communication. Here we show:

Theorem 2. *There is a function f that requires at least $n - 2$ bits of communication.*

Proof As with circuits, we do the proof by counting. To count the number of communication protocols, observe that for every prefix of messages communicated so far, there are 2 choices for who should send the next bit, there are 2^{2^n} choices for the function to use to send that next bit.

If the communication complexity is at most t bits, the number of strings of length at most t is at most 2^{t+1}, so the number of protocols is thus $\left(2 \cdot 2^{2^n}\right)^{2^{t+1}} = 2^{(2^n+1)2^{t+1}} \leq 2^{2^{n+t+2}}$. So if $t < n - 2$, the number of protocols is strictly less than the number of functions f. ■

Turing Machines

A TURING MACHINE IS ESSENTIALLY A PROGRAM written in a particular programming language. The program has access to three arrays and three pointers:

- x which is accessed using the pointer i. x is an array that can be read but not written into.
- y which is accessed using the pointer j. y can be read and written into.
- z which is accessed using the pointer k. z can only be written into.

The machine is described by its code. Each line of code reads the bits x_i, y_j, and based on those values, (possibly) writes new bits into y_j, z_k, and then possibly after incrementing or decrementing i, j, k, jumps to a different line of code or stops computing. Initially, the input is written in x and the goal is for the output to be written in z at the end. i, j, k are all set to 1 to begin with. The arrays all have a special symbol to denote the beginning of the tape and a special symbol to denote the blank parts of the tape.

For example here is a program that copies the input to the output using a single line:

1. If x_i is empty, then HALT. Else set $z_k = x_i$ and increment each of i, k. Jump to step 1.

Here is another that outputs the input bits which are in odd locations:

1. If x_i is empty, then HALT. Else set $z_k = x_i$, increment each of i, k and jump to step 2.
2. If x_i is empty, then HALT. Else increment each of i, k and jump to step 1.

The exact details of this model are not important. The main reason we introduce it is to have a fixed model of computation in mind. For example, it is easy to show that adding more tapes or increasing the alphabet size does not change the model significantly, as we shall discuss further next time.

Resources of Turing Machines

Once we have fixed the model, we can start talking about the *complexity* of computing a particular function $f : \{0,1\}^* \to \{0,1\}$. Fix a turing machine M that computes a function f. There are two main things that we can measure:

- Time. We can measure how many steps the turing machine takes in order to halt. Formally, the machine has running time $T(n)$ if on every input of length n, it halts within $T(n)$ steps.
- Space. We can measure the maximum value of j during the run of the turing machine. We say the space is $S(n)$ if on every input of length n, j never exceeds $S(n)$.

The following fact holds because in each step j can be incremented by at most 1:

Fact 3. *The space used by a machine is at most the time it takes for the machine to run.*