Lecture Notes: Computation Model

Yufei Tao
Department of Computer Science and Engineering
Chinese University of Hong Kong
taoyf@cse.cuhk.edu.hk

Computer science is a subject under mathematics. From your undergraduate study, you should have learned that, before you can even start to analyze the "running time" of an algorithm, you need to first define a computation model properly.

The RAM Model. This is perhaps the model you are most familiar with. In the Random Access Machine (RAM) model, the *memory* is an infinite sequence of *cells*, where each cell is a sequence of w bits for some integer w, and is indexed by an integer *address*. Each cell is also called a *word*; and accordingly, the parameter w is often referred to as the *word length*. The *CPU*, on the other hand, has a (constant) number of cells, each of which is called a *register*. The CPU can perform only the following *atomic* operations:

- Set a register to some constant, or to the content of another register.
- Compare two numbers in registers.
- Perform $+, -, \cdot, /$ on two numbers in registers.
- When an address x has been stored in a register, read the content of the memory cell at address x into a register, or conversely, write the content of a register into the memory cell.

The *time* (or *cost*) of an algorithm is measured by the number of atomic operations it performs. Note that the time is an integer.

A remark is in order about the word length w: it needs to be long enough to encode all the memory addresses! For example, if your algorithm uses n^2 memory cells for some integer n, then the word length will need to have at least $2\log_2 n$ bits. Unless otherwise stated, we consider that $w = \Theta(\log n)$, where n is the "input size", whose meaning will be clearly defined in every problem to be discussed in this course.

The Real-RAM Model. In the above model, the (memory/register) cells can store only integers. Next, we will slightly modify the model to deal with real values.

Note that simply "allowing" each cell to store a real value does not give a satisfactory model because it destroys the underlying rigor. For example, how many bits would you use for a real value? In fact, even if the number of bits *were* infinite, still we would not be able to represent all the real values even in a short interval like $[0, 1]$ — the set of real values in the interval is *uncountably* infinite!

We can alleviate this issue by introducing the concept of *black box*. We still allow a (memory/register) cell c to store a real value x, but in this case, the algorithm is forbidden to look *inside* c, that is, the algorithm has no control over the representation of x. In other words, c is now a black box, holding the value x *precisely* (by magic).

A black box remains as a black box after computation. For example, suppose that two registers are both storing $\sqrt{2}$. We can calculate their product 2, but the product must still be understood as a real value (even though it is an integer). This is similar to the requirement in C++ that the product of two float numbers remains as a float number.

Now we can formally extend the RAM model by introducing some additional rules:

- Each cell can store either an integer or a real value.
- For operations $+$, $-$, $\cdot$, and $/$, if one of the operand numbers is a real value, the result is a real value.
- We allow another atomic operation called the *floor*. If a cell contains a real value x, the operation $\lfloor x \rfloor$ converts the cell into an integer storing $\lfloor x \rfloor$, provided that $x \in [0, 2^w)$ (this condition ensures that $\lfloor x \rfloor$ can be stored in a word).
- Every cell in the input to the algorithm must be in the range $[0, 2^w)$.

We will call the new model the *real RAM* model.

We must be careful not to abuse the power of real-value computation because research [1] has shown that an unconstrained real RAM model can solve NP-hard problems in polynomial time. To avoid such oddity, we will adhere to the *constant multiplication-depth* requirement. To explain the condition, let us consider any real value x that is computed during an algorithm's execution. The computation of x can be modeled as a tree $T(x)$. Specifically, each leaf of $T(x)$ is either a constant or a real value in the input to the algorithm. Every internal node u of $T(x)$ is labeled with an operation $op(u) \in \{+, -, \cdot, /, \lfloor\rfloor\}$. Each operand required by $op(u)$ is stored at a child of u in $T(x)$, and the operation's output is stored at u. The value of x is stored at the root of $T(x)$. The constant multiplication-depth requirement states:

Any root-to-leaf path of $T(x)$ can contain only a constant number of multiplications.

Randomness. All the atomic operations discussed so far are *deterministic*. As a result, our models currently do not permit *randomization*, which is essential to apply to certain algorithmic techniques (such as hashing).

To fix the issue, we introduce one more atomic operation for the RAM and real-RAM models. This operation, named *RAND*, takes two non-negative integer parameters x and y, and returns an integer chosen uniformly at random from $[x, y]$. In other words, every integer in $[x, y]$ can be returned with probability $1/(y - x + 1)$. The values of x, y should be in $[0, 2^w - 1]$ because they each need to be encoded in a word.

Math Conventions. We will assume that you are familiar with the notations of $O(.), \Omega(.)$, $\Theta(.)$, $o(.)$, and $\omega(.)$. The notation $\tilde{O}(f(n_1, n_2, ..., n_x))$ represents the class of functions that are $O(f(n_1, n_2, ..., n_x) \cdot \text{polylog}(n_1 + n_2 + ... + n_x))$, namely, $\tilde{O}(.)$ hides a polylogarithmic factor. The symbol $\mathbb{R}$ denotes the set of real values.

References

[1] A. Schonhage. On the power of random access machines. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, volume 71, pages 520–529, 1979.