RAM with Randomization

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

So far, all our algorithms are **deterministic**. This lecture will introduce you to **randomized algorithms**. Such algorithms play an important role in computer science. They are often simpler, and sometimes can be provably faster.

Our current RAM model does not have any random mechanism yet! To fix it, we will include one more atomic operation and, accordingly, define an expected version of "algorithm cost".

RAM with Randomization

Recall that the core of the RAM model is a set of atomic operations. We now extend this set with one more atomic operation:

- **RANDOM(x, y)**: Given integers x and y (satisfying $x \leq y$), this operation returns an integer chosen **uniformly at random** in $[x, y]$.
  - Any of x, $x + 1$, $x + 2$, ... y has the same probability of being returned.

Deterministic Algorithms vs. Random Algorithms

An algorithm is **deterministic** if it never invokes the atomic operation RANDOM. Otherwise, the algorithm is **randomized**.

Recall that the **cost** of an algorithm is the length of the algorithm's execution (recall that an execution is a sequence of atomic operations).

On the same input, the cost of a deterministic algorithm is a fixed integer.

The cost of a randomized algorithm, however, is a **random variable**. Even on the same input, the cost may change every time the algorithm is executed.

Example

1. **do**
2. $\quad r = \text{RANDOM}(0, 1)$
3. **until** $r = 1$

How many times would Line 2 be executed? The answer is: "we don't know"! Every time the above is executed, it may produce a new sequence of atomic operations.

Expected Cost of a Randomized Algorithm

Let X be a random variable that equals the cost of an algorithm on an input. The **expected cost** of the algorithm on the input is the expectation of X.

Example 1

1. **do**
2. r = RANDOM(0, 1)
3. **until** $r = 1$

Let X be the cost of the above (randomized) algorithm. X equals 2 with probability $1/2$, 4 with probability $1/4$, 6 with probability $1/8$, ... In general, for $i \geq 1$:

$$\boldsymbol{Pr}[X = 2i] = 1/2^i.$$

Hence:

$$\boldsymbol{E}[X] = \sum_{i=1}^{\infty} \frac{2i}{2^i} = O(1)$$

where we used the fact that $\sum_{i=1}^{\infty}(i/2^i) = 2$.

Example 2

Problem "Find-a-Zero": Let A be an array of n integers, among which there is at least one 0. Design an algorithm to report an arbitrary position of A that contains a 0.

For example, suppose $A = (9, 18, 0, 0, 15, 0, 33, 17)$. An algorithm can report 3 (because $A[3] = 0$), 4, or 6.

Example 2

Consider the following randomized algorithm:

1. **do**
2. $\quad r = \text{RANDOM}(1, n)$
3. **until** $A[r] = 0$
4. **return** r

What is the expected cost of the algorithm? It depends on how many 0's there are in A:

- **Best case:** If all numbers in A are 0, the algorithm finishes in $O(1)$ time.
- **Worst case:** If A has only one 0, the algorithm finishes in $O(n)$ expected time (**think:** why?).

Worst-Case Expected Cost of a Randomized Algorithm

Under a problem size n, the **worst-case expected cost** (or just **expected cost** in short) of a randomized algorithm is the maximum expected cost of the algorithm on all possible inputs of size n.

Formally, let S_n be the (possibly infinite) set of all possible inputs of size n. Fix an algorithm A (which can be random or deterministic). For each input $I \in S_n$, define random variable $cost_A(I)$ as the cost of A on the input I. Then, the worst-case expected cost of A is a function of n:

$$f_A(n) = \max_{I \in S_n} \boldsymbol{E}[cost_A(I)].$$

Example 2 (cont.)

Remember array A has at least one 0.

1. **do**
2. $\quad r = \text{RANDOM}(1, n)$
3. **until** $A[r] = 0$
4. **return** r

Worst-case expected cost of the algorithm $= O(n)$

We now have a new RAM model.

Before finishing the lecture, we will tap into the power of randomization by witnessing a problem where randomized algorithms are **provably faster** than deterministic ones in terms of expected cost.

Think: what is the expected cost of a deterministic algorithm?

Power of Randomization

Problem "Find-a-Zero": Let A be an array of n integers, among which **half** of them are 0. Design an algorithm to report an arbitrary position of A that contains a 0.

For example, suppose $A = (9, 18, 0, 0, 15, 0, 33, 0)$. An algorithm can report 3, 4, 6, or 8.

Power of Randomization

1. **do**
2. $\quad r = \text{RANDOM}(1, n)$
3. **until** $A[r] = 0$
4. **return** r

The algorithm finishes in $O(1)$ expected time on **every input** A!

In contrast, **any** deterministic algorithm must probe at least $n/2$ integers of A in the worst case! In other words, any deterministic algorithm must have a worst case time of $\Theta(n)$, slower than the above randomized algorithm (in terms of expected cost).