RAM with Randomization

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
So far all our algorithms are deterministic, namely, they do not involve any randomization. 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 as well.

Our current RAM model, however, is not powerful enough for studying randomized algorithms—it does not have any random mechanism yet! We will fix it formally by incorporating one more atomic operation. Accordingly, we will also need extend the notions of “algorithm” and “cost of an algorithm”.
Recall that the core of the RAM model is a set of atomic operations. We now formally 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.
Recall that we defined an algorithm to be a sequence of atomic operations, and its cost to be the length of the sequence.

These definitions are no longer appropriate with the introduction of the RANDOM operation. This is because an “algorithm” that uses this new operation may produce a different sequence (even on the same input) every time it is executed! We will see an example in the next slide.
Algorithm

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

How many times would Line 2 be executed? The answer is—“we don’t know” (in fact, the line may be executed an infinite number of times)! Every time the above “algorithm” is executed, it may produce a new sequence of atomic operations generated.

To embrace randomization, next we will define “algorithm” and “cost” more carefully, by decoupling the “description” of an algorithm from its “execution”.

Yufei Tao
RAM with Randomization
An **algorithm** is a piece of description such that, when given an input, we can follow the description to generate a sequence of atomic operations in an unambiguous manner. Its **cost on the input** is the length of the sequence generated.

If an algorithm does not use the RANDOM operator, it is **deterministic**; otherwise, it is **randomized**.

**Note:** The cost of a randomized algorithm on an input may be a random variable $X$, in which case the **expected cost** of the algorithm on the input is defined to be the expectation of $X$. 
Example 1

1. do
2. $r = \text{RANDOM}(0, 1)$
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 $i \geq 1$:

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

Hence:

$$E[X] = \sum_{i=1}^{\infty} \frac{2i}{2^i} = 4 = O(1)$$ 

where we used the fact that $\sum_{i=1}^{\infty} (i/2^i) = 2$. 
Now let us see another example where the input size is not a constant.

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

What is the expected cost of the algorithm? The answer is “it depends”:

- If all numbers in $A$ are 0, the algorithm finishes in $O(1)$ time.
- If $A$ has only one 0, the algorithm finishes in $O(n)$ expected time because
  - $A[r]$ has $1/n$ probability of being 0.
  - In expectation, we need to repeat $n$ times to find the 0.

As before, we care about the worst expected time—in this case, $O(n)$. 
Under a problem size $n$, the **worst expected cost** of a randomized algorithm is the largest expected cost of the algorithm over all the possible inputs of size $n$.

We want to design algorithms whose worst expected costs grow slowly with time.

**Remark:** Henceforth, by “expected cost” of an algorithm, we always refer to its worst expected cost.
We now have a new RAM model. This is the computation model we will stick to in the rest of the course.

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

The algorithm finishes in \( O(1) \) expected time!

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 \( \Theta(n) \)—provably slower than the above randomized algorithm (in expectation).