Lecture 25

1 Cracking Eggs (or Asymmetric Cost Binary Search)

You have a 100 story building and an egg. This is an especially strong egg. There is some floor below which the egg will not break if dropped. What is the worst case upper bound on the number of drops you must make to determine this floor?

Answer: 99

Ideally we would like a binary search-like algorithm to determine the floor. However, the complication is that if we try a floor too high, then our egg breaks and we are stuck. So we must make sure our guess is always below the actual answer. So we are forced to start from the first floor and work our way up. This means the worst case would be when the floor is 100 or 99, which requires 99 drops.

Variant 2: You are now given 2 identical eggs (i.e. eggs with the same threshold floor). What is the worst case upper bound on the number of drops in this instance?

Answer: 14

The advantage of having 2 eggs is that we are now allowed to guess higher than the actual floor one time and still be ok. Suppose our upper bound is k. For our first drop we could try a floor as high as floor k. If the egg doesn't break, we keep moving up. However, if the egg breaks we can try all the floors below floor k and still determine the floor within k drops. After dropping an egg on floor k, we have k-1 remaining drops, and so the same logic applies. We can now move up to floor k + (k-1). So eventually we move up as high as $\sum_{1\leq i\leq k} i = \frac{k(k+1)}{2}$ The smallest value of k such that this value will be $\geq$ 99 is 14.

2 A Real World Example, Network Loading

Given a path in a network, we would like to find the greatest number of packets we can send along that path. As with dropping eggs off of buildings, there is an asymmetric cost. We will consider two cost models, gentle and severe. Let x be the flow through the network and u be the (unknown) threshold.

$$Cost_{gentle} = \begin{cases} \text{u-x} & \text{for } x < u \\ \alpha(\text{x-u}) & \text{for } x > u \end{cases} \tag{1}$$

$$Cost_{severe} = \begin{cases} \text{u-x} & \text{for } x < u \\ \text{u} & \text{for } x > u \end{cases} \tag{2}$$

In the real world, we consider the case where an adversary is changing u with time. This model has been analyzed in by Karp, Papadimitrou and Shekner [1].

For the remainder of this lecture, we will consider only the static u case with a severe cost function.

Let's analyze the worst case performance of different algorithms under these conditions, starting with the most obvious approach.

Binary Search: Naively, since there are O(log n) steps to determine u and each step has a worst case cost of n, we can bound the cost by O(nlogn). Is this tight however? It is easy to construct an example to prove this fact. Consider the case where u = n/2 + 1. This has a cost of O(nlogn) and so the bound is tight.

TCP:

The TCP algorithm is as follows:

repeat
  if $x > u$, $x \leftarrow \frac{x}{2}$
  if $x < u$, $x \leftarrow x + 1$
until you find u.

Starting naively once again, we can bound the cost by O(n^2). Once again, we can find an example to show that this bound is tight. Let u = $\frac{n}{2} - 1$.

SHRINK: Assume the threshold u is in the interval $[1..n]$. The Shrink algorithm will find u with a cost of O(nloglogn). The algorithm runs in two phases.

Phase 1: (Shrinking the upper bound)

$x \leftarrow n$
while ball breaks {
  $x \leftarrow \frac{x}{2}$
}

Since x decreases geometrically, the cost of phase one is bounded by O(n).

Phase 2: (Growing the lower bound)
In phase one, we have found an interval of size $x = 2^t$. We then progressively increase the lower bound of this interval by 2^{t-2} until we exceed the threshold. Since we make at most 4 tests, and the cost of each test can be upper bounded by 2^t, the total cost of this round is at most $4(2^t)$. In the next round, we are left with an interval of size 2^{t-2} which we will split into 16 intervals. The total cost in this round will be $16(2^{t-2})$. In general, the cost in the i^th round will be $4^i(2^{t-2i+2}) = 2^{t+2} = \mathrm{O}(2^t) = \mathrm{O(n)}$. We have at most $loglogn$ rounds, and so our total cost is O($nloglogn$). Of course, we will eventually reach a point where the interval size is less than one, in which case SHRINK performs exactly like TCP, incrementing the lower bound by 1.

Under these conditions, SHRINK actually outperforms TCP. However, we have only analyzed this algorithm under the static threshold condition. When we free this condition, TCP remains the optimal solution.

References

[1] Richard Karp, Elias Koutsoupias, Christos Papadimitriou, and Scott Shenker. (2000). *Optimization Problems in Congestion Control.* http://www.cs.ucla.edu/ elias/publications/paper-kkps00.ps