Algorithm Engineering

Probabilistic analysis and Randomized Algorithms

MD ATIQUR RAHMAN

The Hiring Problem

Suppose that you need to hire a new office assistant. Your previous attempts at hiring have been unsuccessful, and you decide to use an employment agency. The employment agency sends you one candidate each day. You interview that person and then decide either to hire that person or not. You must pay the employment agency a small fee to interview an applicant. To actually hire an applicant is more costly, however, since you must fire your current office assistant and pay a substantial hiring fee to the employment agency. You are committed to having, at all times, the best possible person for the job. Therefore, you decide that, **after interviewing each applicant, if that applicant is better qualified than the current office assistant, you will fire the current office assistant and hire the new applicant.** You are willing to pay the resulting price of this strategy, but you wish to estimate what that price will be.

Pseudo Code

```
HIRE-ASSISTANT(n)
1  best = 0          // candidate 0 is a least-qualified dummy candidate
2  for i = 1 to n
3      interview candidate i
4      if candidate i is better than candidate best
5          best = i
6          hire candidate i
```

The Cost Model

Interviewing has a low cost, say ci , whereas hiring is expensive, costing ch . Letting m be the number of people hired, the total cost associated with this algorithm is O(ci*n + ch*m). No matter how many people we hire, we always interview n candidates and thus always incur the cost ci*n associated with interviewing. We therefore concentrate on analyzing ch*m, the hiring cost. This quantity varies with each run of the algorithm.

Worst Case Analysis

In the worst case, we actually hire every candidate that we interview. This situation occurs if the candidates come in strictly increasing order of quality, in which case we hire n times, for a total hiring cost of O(ch*n). Of course, the candidates do not always come in increasing order of quality. In fact, we have no idea about the order in which they arrive, nor do we have any control over this order. Therefore, it is natural to ask what we expect to happen in a typical or average case.

Probabilistic Analysis

Probabilistic analysis is the use of probability in the analysis of problems. In order to perform a probabilistic analysis, we make assumptions about, the distribution of the inputs. **If we cannot describe a reasonable input distribution, and in these cases we cannot use probabilistic analysis.**

For the hiring problem, we can assume that the applicants come in a random order. What does that mean for this problem?

It means, we can rank each candidate with a unique number from 1 through n, using rank(i) to denote the rank of applicant i, and adopt the convention that a higher rank corresponds to a better qualified applicant. The ordered list <rank(1) ,rank(2),.....rank(n)> is a permutation of the list <1,2,....n>. **Saying that the applicants come in a random order is equivalent to saying that this list of ranks is equally likely to be any one of the nŠ permutations of the numbers 1 through n.**

Randomized Algorithms

In the hiring problem, it may seem as if the candidates are being presented to us in a random order, but we have no way of knowing whether or not they really are. Thus, in order to develop a randomized algorithm for the hiring problem, we must have greater control over the order in which we interview the candidates. We will, therefore, change the model slightly. We say that the employment agency has n candidates, and they send us a list of the candidates in advance. On each day, we choose, randomly, which candidate to interview. Instead of relying on a guess that the candidates come to us in a random order, we have instead gained control of the process and enforced a random order.

When we analyze a randomized algorithm, we fix the input and focus on how the algorithm's random choices affect its performance. The expected running time is the average time the algorithm takes, over many possible runs with different random decisions.

Task 1

Below is the implementation of the procedure RANDOM(a,b) that only makes calls to RANDOM(0,1). A call to RANDOM(a,b) returns an integer between a and b, inclusive, with each such integer being equally likely. A call to RANDOM(3,7) returns either 3, 4, 5, 6, or 7, each with probability 1/5. What is the expected running time of the below procedure, as a function of a and b?

Algorithm 1 RANDOM(a,b)

```
1: n = ⌈lg(b − a + 1)⌉
2: Initialize an array A of length n
3: while true do
4:     for i = 1 to n do
5:         A[i] = RANDOM(0, 1)
6:     end for
7:     if A holds the binary representation of one of the numbers in a through
   b then
8:         return number represented by A
9:     end if
10: end while
```

Solution

This becomes a geometric distribution problem. The expected number of while loop iterations until success (first success in geometric distribution) is, E[number of trials] = 1/p.
Here, p is the probability that the while loop stops on a given iteration is (b − a + 1)/2^{n}.

Now each while iteration cost n iteration of the inner for loop, so Expected running time is, E*n = [2^{n}*n] / [b-a+1].
Now, n = log_{2}(b-a+1) => 2^{n} = 2^{log_{2}(b-a+1)} => 2^{n} = (b-a+1)
Putting the value of 2^{n} and n, we get E = log_{2}(b-a+1) = **O(log_{2}(b-a))**

Task 2

Suppose that you want to output 0 with probability 1/2 and 1 with probability 1/2. At your disposal is a procedure BIASED-RANDOM , that outputs either 0 or 1. It outputs 1 with some probability p and 0 with probability 1-p, where 0 < p < 1, but you do not know what p is. Give an algorithm that uses BIASED-RANDOM as a subroutine, and returns an unbiased answer, returning 0 with equal probability of 1. Justify why your algorithm is unbiased.

1 = p(1-p) + (1-p)p => p(1-p) = 1/2

```
1:  for all eternity do
2:      a = BiasedRandom
3:      b = BiasedRandom
4:      if a > b then
5:          return 1
6:      end if
7:      if a < b then
8:          return 0
9:      end if
10: end for
```