CS590R: Class Notes 4

Ahsan Habib

1 Examples: Linearity of Expectation

Example 1. The expectation of the sum of two dice is 7, even if they are not independent.

Let X_1 and X_2 are two random variables that define the outcome of first and second dice respectively.

Let X be the random variable that defines the sum of the outcome of both dice.

From Linearity of expectation,

$$E[X] = E[X_1] + E[X_2]$$

$E[X] = 3.5 + 3.5 = 7$, (Example, Section 2.1, Notes 3).

Example 2. Assume that we flip N coins, what is the expected number of heads?

Using linearity of expectation we get $N \cdot \frac{1}{2}$.

Again, each outcome of the coin is a Bernoulli trail.

The outcome is either head or tail. Let X be a

Bernoulli random variable and p be the probability

of getting head in one trial.

From expectation of indicator random variable, $E[X] = p$.

If we repeat the process for N trials, and if each

outcome is defined by a random variable X_i, we get a Binomial random variable X $B(N, p)$ such that,

$$X = \sum_{i=1}^{N} X_i$$

$$Pr(X = k) = \binom{N}{k} p^k (1-p)^{N-k}$$

$$E[X] = \sum_{k=0}^{N} k \cdot Pr(X = k)$$

By direct summation we get $\sum_{i=0}^{N} i\binom{N}{i} 2^{-N}$.
Thus we prove

$$\sum_{i=0}^{N} i\binom{N}{i} 2^{-N} = \frac{N}{2}.$$

Example 3. Assume that N people checked
coats in a restaurants. The coats are mixed and
each person gets a random coat.
How many people got their own coats?
It's hard to compute $E[X] = \sum_{k=0}^{N} kPr(X = k)$. Instead we define N 0-1 random variables X_i, where $X_i = 1$ iff i got his
coat.

$$E[X_i] \;=\; 1 \cdot Pr(X_i = 1) \;+\; 0 \cdot Pr(X_i = 0) =$$

$$Pr(X_i = 1) = \frac{1}{N}.$$

$$E[X] \;=\; \sum_{i=1}^{N} E[X_i] \;=\; 1.$$

2 Analysis of QuickSort

We discuss analysis of randomized QuickSort and probabilistic
analysis of QuickSort in this section.
Randomized Algorithms:
The analysis is true for **any** input.
The sample space is the space of random choices
made by the algorithm.
The repeated runs are independent for a randomized algorithm.
Probabilistic Analysis:

The sample space is the space of all possible inputs.
If the algorithm is **deterministic** repeated
runs give the same output.

2.1 Randomized Quicksort

This is a randomized algorithm. It does
not assume anything about the input, and the algorithm takes
a random decision to select a pivot in each iteration. The algorithm
for the randomized quicksort is given below:
Procedure Q_S(S);
Input: A set S.
Output: The set S in sorted order.

1. If $|S| \leq 1$ then return S, else

2. (a) Choose a random element y uniformly from S.

   (b) Compare all elements of S to y. Let

$$S_1 = \{x \in S - \{y\} \mid x \leq y\}$$

$$S_2 = \{x \in S - \{y\} \mid x > y\}.$$

   (Elements in S_1 and S_2 are in the same
   order as in S.)

   (c) Return the list:

$$Q\_S(S_1), y, Q\_S(S_2).$$

In worst case the randomized quicksort requires $O(n^2)$ comparisons.
It happens when the pivot partitions the input into two sets of size
1 and $n-1$. However, on average, the algorithm
requires $O(n \log n)$ comparisons.
Let T = number of comparisons in a run of
QuickSort.

Theorem 2.1

$$E[T] \;=\; O(n \log n).$$

Proof:

Let $s_1, ..., s_n$ be the elements of S in sorted order.
For $i = 1, ..., n$, and $j > i$, define 0-1 random variable $X_{i,j}$, s.t.
$X_{i,j} = 1$ iff s_i is compared to s_j in the run of the algorithm.
s_i is compared to s_j iff either s_i or s_j is chosen as a "split item" before any of the $j - i - 1$ elements between s_i and s_j are chosen.
Elements are chosen uniformly at random $\rightarrow$ elements in the set $[s_i, s_{i+1}, ..., s_j]$ are chosen uniformly at random.

$$Pr(X_{i,j} = 1) = \frac{2}{j - i + 1}.$$

$$E[X_{i,j}] = \frac{2}{j - i + 1}.$$

The number of comparisons in running the algorithm is

$$T = \sum_{i=1}^{n} \sum_{j>i} X_{i,j}.$$

$$E[T] = E[\sum_{i=1}^{n} \sum_{j>i} X_{i,j}] =$$

$$\sum_{i=1}^{n} \sum_{j>i} E[X_{i,j}] = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j - i + 1} \leq$$

$$\sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{2}{k} \leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} = 2nH_n = 2n \log n + O(n)$$

□

Note: $\sum_{k=1}^{n} \frac{1}{k} = H_n = \theta(\log n)$.
The upper bound of H_n can be calculated as follows,

$$H_n \leq \int_1^n \frac{1}{x} dx = O(\log n)$$

.

2.2 Probabilistic Analysis of QuickSort.

The deterministic QuickSort does not decide about the pivot randomly, instead, it select the pivots deterministically. For example, the first element can always be the pivot. For a particular set of input, the deterministic QuickSort may need $O(n^2)$ comparisons such as a sorted list.
The probabilistic QuickSort assumes the input comes from a specified probability distribution. The algorithm can be randomized or deterministic.

Theorem 2.2 *The expected run time of (deterministic) Quicksort on a random input, uniformly chosen from all possible permutation of S is $O(n \log n)$.*

Proof:
Set $X_{i,j}$ as before.
If all permutations have equal probability, all permutations of $S_i, ..., S_j$ have equal probability, thus

$$Pr(X_{i,j}) = \frac{2(j-i)!}{(j-i+1)!} = \frac{2}{j-i+1}.$$

$$E[\sum_{i=1}^{n} \sum_{j>i} X_{i,j}] = O(n \log n).$$

□

3 Randomized Algorithm classification

A **Monte Carlo Algorithm** is a randomized algorithm that may produce an incorrect solution.
Example: Min-cut algorithm
For decision problems: A **one-side error** Monte Carlo algorithm errs only on one possible output, otherwise it is a **two-side error** algorithm.
A **Las Vegas** algorithm is a randomized algorithm that **always** produces the correct output.
In both types of algorithms the run-time is a random variable.
Q. Can we convert a Monte Carlo algorithm to a Las Vegas algorithm?
A. If we know the correct answer, we run the Monte Carlo repeatedly and verify the answer. The verification algorithms terminates as soon as a correct answer is found.
If e is the probability that a Monte Carlo algorithm returns incorrect solution. The probability that the algorithm will succeed is $1 - e$. How many iteration it requires for a verification algorithm that converts a Monte Carlo algorithm to a Las Vegas one.
Let X be the random variable that defines the number of iteration required for a verification algorithm. X is a geometric random variable with parameter p, where p is define as the probability of success for an event to occur.

$$Pr(X = k) = p(1-p)^{k-1}$$

$$E[X] = \sum_{k=0}^{\infty} kp(1-p)^{k-1} = 1/p$$

Thus, the number of iterations needed $= 1/(1-e)$