Homework 13

Samuel Pegg

November 2020

1 Page 122 5.2-3

Use indicator random variables to compute the expected value of the sum of n dice.

Let X be the sum of the n dice and let X_i be the number on the i^{th} die. Then

$$X = \sum_{i=1}^{n} X_i \quad \implies \quad E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

The probability that the i^{th} die shows k is simply $\frac{1}{6}$ (assuming fair dice), hence

$$E[X_i] = \sum_{k=1}^{6} kp(X_i = k) = \frac{1}{6}\sum_{k=1}^{6} k = \frac{1}{6} \times \frac{6 \times (6+1)}{2} = \frac{7}{2}$$

Substituting this back into the equation for $E[X]$, we get

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{7}{2} = \frac{7n}{2}$$

2 Page 142 5.4-2

Suppose that we toss balls into b bins until some bin contains two balls. Each toss is independent, and each ball is equally likely to end up in any bin. What is the expected number of ball tosses?

Define the indicator variable

$$I_k = \begin{cases} 1 & \text{there is a } k^{\text{th}} \text{ toss} \\ 0 & \text{otherwise} \end{cases}$$

By the Pigeonhole Principal, the maximum number of throws is $b + 1$. Hence the number of throws, X, is

$$X = \sum_{k=1}^{b+1} I_k \quad \implies \quad E[X] = \sum_{k=1}^{b+1} E[I_k]$$

But

$$E[I_k] = p(\text{previous } k-1 \text{ tosses had no collisions})$$

The first toss having no collisions has probability $1 = \frac{b}{b}$. The second toss having no collisions has probability $\frac{b-1}{b}$. The third toss is then $\frac{b-2}{b}$ since there are two places it cant go, and the i^{th} toss has probability $\frac{b-i+1}{b}$. So,

$$E[I_k] = \frac{b}{b} \times \frac{b-1}{b} \times \frac{b-2}{b} \times \cdots \times \frac{b-k+2}{b} = \frac{b!}{(b-k+1)!b^{k-1}}$$

Note that $\frac{b!}{(b-k+1)!} = P(b, k-1)$. Thus

$$E[X] = \sum_{k=1}^{b+1} E[I_k] = \sum_{k=1}^{b+1} \frac{P(b,k-1)}{b^{k-1}} = \sum_{k=0}^{b} \frac{P(b,k)}{b^k} = 1 + \sum_{k=1}^{b} \frac{P(b,k)}{b^k}$$

3 Page 144 5-2

Consider a randomised algorithm SCRAMBLE-SEARCH that works by first randomly permuting the input array and then running the deterministic linear search given above on the resulting permuted array.

3.1 Page 144 5-2 (h)

Letting k be the number of indices i such that $A[i] = x$, give the worst-case and expected running times of SCRAMBLE-SEARCH for the cases in which $k = 0$ and $k = 1$.

SCRAMBLE-SEARCH works exactly the same as the deterministic linear search, but with the additional running time taken to permute the array. Hence for $k = 1$ the worst case running time is n and the expected case is $\frac{n+1}{2}$. If $k = 0$ then every element must be searched, so both the average and worst case is n.

3.2 Page 144 5-2 (i)

Which of the three searching algorithms would you use? Explain your answer.

DETERMINISTIC-SEARCH since the expected running time is the same as SCRAMBLE-SEARCH but without the operations to permute the array. RANDOM-SEARCH's expected time when x is not in the list is $b(\ln b + O(1))$, whereas DETERMINISTIC-SEARCH always converges in n steps.

4 Page 204 8.4-4

We are given n points in the unit circle, $p_i = (x_i, y_i)$, such that $0 < x_i^2 + y_i^2 \leq 1$ for $i = 1, 2, \ldots, n$. Suppose that the points are uniformly distributed; that is, the probability of finding a point in any region of the circle is proportional to the area of that region. Design an algorithm with an average-case running time of $\Theta(n)$ to sort the n points by their distances $d_i = \sqrt{x_i^2 + y_i^2}$ from the origin.

We want n buckets $b_1, \ldots, b_n$ of equal size in the range $[0, 1)$. Since we are working with distances from the origin in the 2d plane, we should use concentric rings as buckets around the origin. The inner radius of the i^{th} ring, r_i, must be equal to the i^{th} sub-ring of the unit circle, i.e. multiply the unit circle by $\frac{i}{n}$. Hence

$$\pi r_i^2 = \frac{i}{n}\pi 1^2 \quad \Longrightarrow \quad r_i = \sqrt{\frac{i}{n}}$$

So the i^{th} bucket is simply

$$\{(x, y) \mid r_i \leq \sqrt{x^2 + y^2} < r_{i+1}\}$$