Section I.2: Expected Values and Applications to Algorithms

Definition I.2.1: A **finite random variable** X is a function from a finite sample space S to the real numbers.

Example I.2.1: Let S be the set of students at a college, X the function assigning each student his or her height, Y a similar function assigning weights. Then X and Y are random variables on S.

Definition I.2.2: Let X be a finite random variable, $x \in \Re$(reals). The **probability density function** f on random variable X is defined as $f(x) = P(X = x)$.

Example I.2.2: Let S be the set of outcomes of the throw of 2 dice.
$|S| = 36$. Let $s \in S$. Then $P(s) = \frac{1}{36}$. Let X be the sum of numbers on the dice. Then $P(X=7) = \frac{1}{6}$ and $P(X = 8) = \frac{5}{36}$. (Epp, p. 277)

Definition I.2.3: The **expected value** (weighted average) of a random variable X is defined as $E[X] = \sum_{x \in X} x \cdot P(X = x)$.

Example I.2.3: Consider a game where 2 coins are tossed. Find the expected value of one's winnings when
(a) One wins \$4 for 2 heads and loses \$2 for any other outcome.
(b) One wins \$3 for each head and loses \$2 for each tail.
(c) Which game would you prefer to play and why?

Solution: Compute the expected values in each case**.** We assume X and Y are the random variables in **(a)** and **(b)** respectively.

(a) $E[X] = 4 \cdot P(\text{H-H}) + (-2) \cdot P(\text{other throws}) = 4 \cdot (\frac{1}{4}) + (-2) \cdot (\frac{3}{4}) = \frac{4-6}{4} = -\frac{1}{2}$

(b) $E[Y] = 6 \cdot P(\text{H-H}) + (3-2) \cdot P(1 \text{ head, } 1 \text{ tail}) + (-4) \cdot P(\text{T-T})$
$= 6 \cdot (\frac{1}{4}) + 1 \cdot (\frac{1}{2}) + (-4) \cdot (\frac{1}{4}) = \frac{3}{2} + \frac{1}{2} - 1 = 1$

(c) The expected value in this case measures the average winnings of a person. A game is fair if and only $E[X] = 0$. The "house" wins if $E[X] < 0$. Since the average loss in (a) is \$0.50 and the average win in (b) is \$1, we would prefer to play the game in (b).

Example I.2.4: Let the sample space *S* be the set of outcomes when 2 dice are thrown (Epp, Ex. 6.1.2, pp. 276-77).
(a) Get the expected value E[*X*] when random variable *X* is defined as: $5 is won if a double other than 1-1 is thrown, $1 is lost if 1-1 is thrown, and $1 is lost for any other throw.
(b) Get the expected value E[*Y*] when random variable *Y* is defined as: $7 is won if a double other than 1-1 is thrown, $4 is lost if 1-1 is thrown, and $1 is lost for any other throw.
(c) Which game would you rather play? Explain.

Solution:
(a) $E[X] = 5 \cdot P(\text{double not 1-1}) + (-1) \cdot P(\text{1-1}) + (-1) \cdot P(\text{other})$

$$= 5 \cdot (\frac{5}{36}) + (-1) \cdot (\frac{1}{36}) + (-1) \cdot (\frac{30}{36}) = \frac{(25-1-30)}{36} = \frac{-6}{36} = -\frac{1}{6}$$

(b) $E(Y) = 7 * P(\text{double not 1-1}) + (-4) * P(\text{1-1}) + (-1) * P(\text{other})$

$$= 7 * (\frac{5}{36}) + (-4 \cdot \frac{1}{36}) + (-1) \cdot (\frac{30}{36}) = \frac{(35-4-30)}{36} = \frac{1}{36}$$

(c) Since the expected value is larger in (b) than (a), we would rather play game (b).

Theorem I.2.1: Linearity of Expectation: If *X* and *Y* are random variables on sample space *S*, then $E[X + Y] = E[X] + E[Y]$. Also, if *c* is a real constant, then $E[cX] = cE[X\}]$ (This theorem works for finitely many terms $X_1, \ldots, X_n$.)

Applications to Algorithms

Definition I.2.4: A **sentinel** is an extra item added to one end of a list to insure that a loop will terminate without having to include a separate check.

In **Algorithm Sequential Search** below, the item *X* put into the *n*+1-st element of the array is a sentinel. The advantage is avoiding a second comparison in the condition of the **while** loop. Without the sentinel, one would have to either compare index *i* with *n* or set up a Boolean flag *found*, initialize it to *false*, and assign it *true* if the item is found. A disadvantage is that if all elements in an array are within a certain range (e.g., all positive), one commonly assigns a value outside the range as a sentinel (e.g. *X* negative). The program will not execute properly if the range of the input variables is unknown and a sentinel *X* corresponds to a value of an input variable.

Sequential Search
Sequential Search is an algorithm that looks for an item X in an array A of n items, n a positive integer. The pseudo code is written below (in Epp's style). We wish to find the average number of comparisons of X to A[i], the i-th element of the array, by computing an expected value.

Algorithm Sequential Search
Input: Value X, n [size of array], array A[1], .. A[n] of values. We use X as a sentinel in location A[n+1].
Output: *location***,** a variable giving the position where X is found if X is found, 0 if X is not found.
Algorithm body:
$i := 1$**;** $A[n+1] := X$**;**
while $X \neq A[i]$
$i := i + 1$;
end while
if $i \leq n$ **then** *location* := i
else *location* := 0
end Algorithm Sequential Search

Example I.2.5: Assume that X is equally likely to be found in any position of array A**.**
(a) Get the number of comparisons of X to $A[i]$ if X is found in the i-th position.
(b) Get the number of comparisons if X is not found.
(c) Get the expected value E (weighted average) of the number of comparisons if X is equally likely to be found in any position i, $1 \leq i \leq n$.
(d) Get the expected value (weighted average) E of comparisons if half of the time X is not found and rest of the time X is equally likely to be in the any position.

Solution:
(a) The answer is the number of times we test the **while** condition until the condition is false ($X = A[i]$). The answer is i.

(b) If the item is not found, the **while** condition becomes false only when $X = A[n + 1]$. There are $n + 1$ comparisons.

(c)$E = \sum_{i=1}^{n} P(i) \cdot C(i)$ where $P(i)$ is probability that X is in the i-th position and $C(i)$ is the number of comparisons if X is in the i-th position.
Therefore, $E = (\frac{1}{n}) \cdot 1 + ... + (\frac{1}{n}) \cdot n = (\frac{1}{n}) \cdot (1 + ... + n) = (\frac{1}{n}) \frac{n \cdot (n+1)}{2} = \frac{(n+1)}{2}$,
using $1 + \ldots + n = \frac{n \cdot (n+1)}{2}$.

(d) Here $P(i) = \frac{1}{2n}$, $1 \leq i \leq n$, and $P(n+1) = \frac{1}{2}$. Therefore

$$E = \frac{1}{2n} \cdot (1 + ... + n) + (\frac{1}{2}) \cdot (n+1)$$
$$= (\frac{1}{2n}) \cdot \frac{n \cdot (n+1)}{2} + \frac{(n+1)}{2} = \frac{(n+1)}{4} + \frac{(n+1)}{2} = \frac{3(n+1)}{4}.$$

Palindromes

Definition I.2.5: A **palindrome** is a string that reads the same backward and forward. For example, "a", "aba", " civic", and "werrew" are palindromes, and "is", "abc", and "apple" are not.

The algorithm below examines a string of bits (consisting of 0's and 1's) stored in array $A[1] .. A[n]$ and returns Boolean values *true* or *false* indicating whether the string is or is not a palindrome. In the **while** loop of the algorithm, we exit if a value in the left half of the string does not match the corresponding value in the right half of the string, and we go to the index $\lfloor n/2 \rfloor$ just prior to the halfway point if matches have occurred up to that point . If we increment the index beyond $\lfloor n/2 \rfloor$, then we have a palindrome; otherwise we do not.

Algorithm Binary Palindrome
Input: *n* [positive integer] , *n* [size of array], array $A[1], .. A[n]$ of bits.
Output: Value *palindrome,* which is *true* if the string of bits in *A* is a palindrome, *false* otherwise.
Algorithm Body

```
i := 1
while A[i] = A[n - i + 1] and i ≤ ⌊n / 2⌋
        i := i + 1;
end while
if  i > ⌊n / 2⌋ then palindrome := true
                else palindrome := false
```

end Algorithm Binary Palindrome

Example I.2.6: Let $P(i)$ be the probability that string *A* is a palindrome up to the *i*-th position, (i.e., the first *i* characters match the last *i* characters), and $C(i)$ be the number of comparisons of $A[i]$ to $A[n - 1 + i]$. Get the expected value (weighted average) of the number of comparisons to determine whether the string of length *n* is a binary palindrome. Get this value for $n = 6$ and $n = 7$.

Solution: $P(1) = \frac{1}{2}$ since, given the bit in $A[1]$, the probability that *n*-th bit in $A[n]$ matches is $\frac{1}{2}$. Likewise, $P(2) = \frac{1}{4}$. We extend this argument to get $P(i) = \frac{1}{2^i}$. Also, $C(i) = i$. Therefore,

$$E = P(1) \cdot C(1) + ... + P\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \cdot C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) = \frac{1}{2} + \frac{1}{2} + \frac{3}{8} + ... + (\lfloor n/\mathbf{2} \rfloor/(2^{\left\lfloor \frac{n}{2} \right\rfloor})).$$

The answer for n= 6 is $\frac{1}{2} + \frac{1}{2} + \frac{3}{8} = \frac{11}{8}$.

The answer for n = 7 is the same since $\left\lfloor \frac{6}{2} \right\rfloor = \left\lfloor \frac{7}{2} \right\rfloor = 3$.

Exercises:

(1) Let *S* be the set of all outcomes when 3 coins are tossed (*S* = {HHH, TTH, HTH, THH, HTT, THT, HTT, TTT}). Assume that each value in *S* is equally likely.
(a) Find the expected value if $10 is won when all three tosses are the same and $2 is lost for any other outcome.
(b) Find the expected value if $12 is won for HHH or TTT and $3 is lost for any other outcome.
(c) Which of the above games is to your advantage? Explain.

(2) Let *S* be the sample space of outcomes when two dice are thrown.
(a) Assume $5 is lost if a 1-1 is thrown, $1 is won when the sum of numbers on the dice is 7, and no win or loss occurs otherwise. Find the expected value. Does the game favor you or the "house"? Explain.
(b) Assume $7 is lost if a 1-1 is thrown, $1 is won when the sum of the numbers on the dice is 7, and no win or loss occurs otherwise. Find the expected value. Does the game favor you or the "house"? Explain.

(3) Using the Sequential Search algorithm, assume that one-third of the time the item *X* is **not** found in the array and two-thirds of the time *X* is equally likely to be found in any position. Get the weighted average number of comparisons by finding the expected value. Compare your answer to those in Example I.2.5.

(4) The algorithm below finds the maximum value in array *A*[1] ... *A*[*n*] of values, *n* a positive integer.

Algorithm Maximum
Input: *n* [size of array], array *A*[1], …, *A*[*n*] of values.
Output: *max*, the largest value in the array
Algorithm Body:
    max := *A*[1]
    for i := 2 **to** *n*
        if *A*[i] > *max* **then** *max* := *A*[i]
    next i
end Algorithm Maximum

(a) How many comparisons of *max* to *A*[*i*] are there if the maximum element is the first element of array *A*, if the maximum element is second element, if the maximum element is the *i-th* element?

(b) Assume that maximum of *A*[1] .. *A*[*i*] is equally likely to be in the *j*-th position, $1 \le j \le i$. For $i = 1, 2, 3$, and general *i*, find the probability that *A*[*i*] is maximum element in the list *A*[1] ... *A*[*i*].

(c) Get the weighted average (expected value) of the number of comparisons of *A*[*i*] to *max* in terms of *n*. Then get this quantity for $n=1$, $n=2$, and $n=3$.

In (d) and (e) assume that we count the number of assignments **inside the loop.**

(d) The number of assignments *max* := A[*i*] can vary from 0 (best case) to *n*-1 (worst case) depending on the ordering of elements in A. Describe these orderings.

(e) If the maximum appears in the first or second position of *A*, there are zero or one assignments, within the loop respectively. If it appears in the third position of *A*, there are one or two assignments depending on whether the second largest element appears in position 1 or 2. Assuming the second largest element is equally likely in any position, get the weighted average of number of assignments when maximum is in the third position. Extend this thought process to the case where maximum is in the fourth position. Then get the expected value of the number of assignments when the maximum is in the fourth position and when it is in the fifth position. Assume that, once the second largest element is located, the third largest element is equally likely to be in any of the remaining positions. Also assume that, once the third largest element is located, the fourth largest element is equally likely to be in any of the remaining positions.