# Lecture #10: Discrete Probability for Computer Science I

## What To Do Before the Lecture

1. Watch the videos for Lecture #10 — noting that they will probably be understandable if you play them at double speed. If you do not have time for this then look at the "Key Concepts" document that is found, immediately after the videos for this lecture, on the course web site, instead.
2. ***Print*** and read through the rest of this document and — if you have time — try to solve the problems! These should help you to check that you understand how Turing machines process strings, and that you understand how to prove things about them.

## Problems To Be Solved

1. Consider the problem of searching for a copy of `true` in a Boolean array $A$ of length $n$, for a positive integer $n$. An algorithm that solves this problem — returning an integer $i$ such that $0 \leq i \leq n-1$ and $A[i] = \texttt{true}$ if the array contains a copy of `true` and throwing a `NoSuchElementException` otherwise — is shown in Figure 1 on page 2.

   The ***experiment***, to be considered, will concern the use of this algorithm to search for a copy of `true` in an input array. The ***sample space*** will include all possible input arrays with length $n$, so that

   $$\Omega = \{(\alpha_1, \alpha_2, \ldots, \alpha_n) \mid \alpha_i \in \{\texttt{true}, \texttt{false}\} \text{ for every integer } i \text{ such that } 1 \leq i \leq n\} \quad (1)$$

   — where an element

   $$\vec{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \quad (2)$$

   represents an input array $A$ such that $A[i] = \alpha_i$ for $1 \leq i \leq n$ — and so that $|\Omega| = 2^n$.

```
integer search ( boolean[] A ) {
1.  integer n := A.length
2.  integer i := 0
3.  while (i < n) {
4.    if (A[i] == true) {
5.      return i
      }
6.    i := i + 1
    }
7.  throw a NoSuchElementException
}
```

Figure 1: Algorithm to Search in a Boolean Array

Suppose we wish to analyze the number of elements of the array that are checked when the algorithm is executed — which is the same as the number of times that the test at line $4$ is checked before the execution of the algorithm ends. This can be modelled using a random variable

$$C : \Omega \to \mathbb{N} \tag{3}$$

where, for an element $\vec{\alpha}$ of the same space, $C(\vec{\alpha})$ is the number of times that this step is executed when the input is the array represented by $\vec{\alpha}$.

(a) Suppose that $\vec{\alpha}$ is as shown at line (2), above. How is $C(\vec{\alpha})$ related to $\alpha_1, \alpha_2, \ldots, \alpha_n$? How small can this value be? How large can it be?

(b) Suppose, now, that ***every input array with length $n$ is equally likely*** — so that the ***uniform probability distribution*** P : $\Omega \to \mathbb{R}$ will be used for an analysis.

Let $k$ be an integer such that $1 \leq k \leq n$. What is the probability that $C = k$, that is, the step at line $4$ is executed exactly $k$ times (during an execution on a "randomly chosen" input array, using this probability distribution)?

(c) What is the ***expected value*** of the random variable, $C$, with respect to this probability distribution?

2. Consider the experiment, involving the algorithm in Figure 1, once again — so that the same space $\Omega$ and the random variable $C$ are as above — but consider a different probability distribution.

   In particular, let $p$ be a real number such that $p < 1$, and let $\mathsf{P} : \Omega \to \mathbb{R}$ such that the following properties are satisfied:

   - For every integer $i$ such that $0 \leq i \leq n - 1$, $\alpha_i$ is `true` with probability $p$.
   - The events

     $$\alpha_0 = \texttt{true}, \alpha_1 = \texttt{true}, \alpha_2 = \texttt{true}, \ldots, \alpha_{n-1} = \texttt{true}$$

     are ***mutually independent***.

   (a) Once again, let $k$ be an integer such that $1 \leq k \leq n$. What is the probability that $C = k$, that is, the step at line $4$ is executed exactly $k$ times (during an execution on a "randomly chosen" input array, using this probability distribution)?

   (b) What is the ***expected value*** of the random variable, $C$, with respect to this probability distribution?

```
integer rSearch ( boolean[] A ) {
1. integer n := A.length
2. integer i := 0
3. while (i < n) {
4.    Choose an integer j uniformly and randomly from the set
                        {0, 1, 2, ..., n − 1}.
5.    if (A[j] == true) {
6.      return j
      }
7.    i := i + 1
    }
8. throw a NoSuchElementException
}
```

Figure 2: Randomized Algorithm to Search in a Boolean Array

3. Consider the same problem, but consider a *different* algorithm, namely the ***randomized algorithm*** shown in Figure 2.

   (a) Explain, as precisely as you can, why this algorithm ***does not*** solve the search problem that is being considered — but "comes close" in some significant sense. Describe, as precisely as you can, how this algorithm can produce *incorrect* output, or throw an exception when it should not.

Consider the execution of this algorithm on some ***fixed*** input array $A$. Once again, let us consider the number of times that array entries are checked, that is, the number of times that the step at line $5$ of this *new* algorithm is executed.

(b) Suppose, first, that $A[i] = \texttt{true}$ for every integer $i$ such that $0 \leq i \leq n-1$. Describe the ***sample space***, ***random variable*** and ***probability distribution*** that would be used in this first (simple) case.

(c) Suppose, next, that $A[i] = \texttt{false}$ for every integer $i$ such that $0 \leq i \leq n-1$, instead. Describe the ***sample space***, ***random variable***, and ***probability distribution*** that would be used in *this* (simple) case.

The more interesting situation is one where some of the entries of the input array $A$ are `true` and others are false. Let

$$S_T = \{i \in \mathbb{N} \mid 0 \leq i \leq n-1 \text{ and } A[i] = \texttt{true}\} \tag{4}$$

— so that $1 \leq |S_T| \leq n-1$ in this case. Let

$$p = \frac{|S_T|}{n} \tag{5}$$

— so that

$$\tfrac{1}{n} \leq p \leq 1 - \tfrac{1}{n}.$$

(d) Describe the ***sample space***, ***random variable***, and ***probability distribution*** that would be used in this (more interesting) case.

(e) Use this to compute the expected number of times that array entries are access in this case — expressed as a function of $p$ and (possibly) $n$.

## More Problems for Later

The following problems might also be of interest. You are welcome to discuss these with the instructor during office hours or on Piazza.

4. Consider the deterministic algorithm in Figure 1 once again. Let $k$ be an integer such that $1 \leq k \leq n$ and consider the case that the input array includes exactly $k$ copies of "`true`" and exactly $n - k$ copies of "`false`" — with each of the input arrays, that satisfy this condition, occurring with the same probability.

   Describe the probability distribution that models *this* situation. Then try to compute the expected number of times that the array is accessed (that is, the expected value of the random variable $C$) with respect to *this* probability distribution.

5. Unfortunately the randomized algorithm, considered in Question #3, was not guaranteed to return correct output in all cases.

   (a) Describe a very small change to this algorithm, that results in an algorithm that does almost the same thing but whose output is always correct.

   *Hint:* You will only need to change one line in the pseudocode. The deterministic algorithm, considered in Questions #1 and #2, will be useful.

   (b) Modify the analysis from this lecture presentation, to compute the expected number of times that array entries are accessed, when this modified algorithm is used.

6. Suppose you want to count the total number of executions of numbered steps in these algorithms, instead of just the number of times that array entries are checked.

   (a) How would you define a ***random variable*** that could be used for this?

   (b) How is this related to the random variable(s) that have been considered, above?

   (c) Describe a result that is described in the preparatory material for Lecture #10, that can used to relate the expected values of these random variables — so that the expected value of this new random variable is now easy to compute?