Dice, used with a plural verb, means small cubes marked with one to six dots, used in gambling games. Dice, used with a singular verb, means a gambling game in which these cubes are used. Dice (plural) can also refer to any small cubes, especially cube-shaped pieces of food (Cut the cheese into dice).

– MSN Encarta

Iacta alea est. (The die is cast.)

–Julius Caesar

I will never believe that God plays dice with the universe.

– Albert Einstein

Thus one comes to perceive, in the concept of independence, at least the first germ of the true nature of problems in probability theory.

– Kolmogorov

Lecture VIII QUICK PROBABILITY

We review the basic concepts of probability theory, using the axiomatic approach first expounded by A. Kolmogorov. His classic [13] is still an excellent introduction. The axiomatic approach is usually contrasted to the empirical or "Bayesian" approach that seeks to predict real world phenomenon with probabilistic models. Other source books for the axiomatic approach include Feller [7] or the approachable treatment of Chung [6]. Students familiar with probability may use the expository part of this Chapter as reference.

Probability in algorithmics arises in two main ways. In one situation, we have a deterministic algorithm whose input space has some probability distribution. We seek to analyze, say, the expected running time of the algorithm. The other situation is when we have an algorithm that makes random choices, and we analyze its behavior on any input. The first situation is considered less important in algorithmics because we typically do not know the probability distribution on an input space (even if such a distribution exists). By the same token, the second situation derives its usefulness from avoiding any probabilistic assumptions about the input space. Algorithms that make random decisions are said to be **randomized** and comes in two varieties. In one form, the algorithm may make a small error but its running time is worst-case bounded; in another, the algorithm has no error but only its expected running time is bounded.

There is an understandable psychological barrier to the acceptance of unbounded worst-case running time or errors in randomized algorithms. However, it must be realized that the errors in randomized algorithms are controllable by the user – we can make them as small as we like at the expense of more computing time. Should we accept an algorithm with error probability of 2^{-99}? In daily life, we accept and act on information with a much greater uncertainty than this.

More importantly, randomization is often the only effective computational tool available to attack intransigent problems. Until recently, the standard example of a problem not known to be in the class P (of deterministic polynomial time solvable problems), but which admits a randomized polynomial-time algorithm is the **Primality Testing** (deciding if a integer is prime). This problem is now known to be polynomial-time, thanks to a breakthrough by Agrawal, Kayal and Saxeena (2002). But the algorithm has complexity $O(n^{7.5})$ which it is not very practical. Thus randomized primality remains useful in practice. But randomization is not the universal acid for hard problems. For instance, the related problem of factorization of integers does not have a randomized polynomial-time algorithm.

There is a large literature on randomized algorithms. Motwani and Raghavan [15] gives a good overview of the field. Alon, Spencer and Erdos [3] treats the probabilistic method, not only in algorithmic but also in combinatorial analysis. Karp [10] discusses techniques for the analysis of recurrences that arise in randomized algorithms.

The first part of this chapter is a brief review of the elements of probability theory. The advanced reader may skip right to the algorithmic applications.

§1. Axiomatic Probability

All probabilistic phenomena occur in some probabilistic space, which we now formalize (axiomatize).

¶1. Sample space. Let Ω be any non-empty set, possibly infinite. We call Ω the **sample space** and elements in Ω are called **sample points**.

We use the following running examples of sample spaces:

(E1) $\Omega = \{H, T\}$ (coin toss). This represents a probabilistic space where there are two outcomes. Typically we identify these outcomes with the results of tossing a coin – head (H) or tail (T).

(E2) $\Omega = \{1, \ldots, 6\}$ (dice roll). This is a slight variation of (E1) representing the outcomes of the roll of dice, with six possible outcomes.

(E3) $\Omega = \mathbb{N}$ (the natural numbers). This is a significant extension of (E1) since there is a countably infinite number of outcomes.

(E4) $\Omega = \mathbb{R}$ (the real numbers). This is a profound extension because we have gone from discrete space to continuous space.

¶2. Event space. Sample spaces becomes more interesting when we give it some structure to form event spaces.

Let $\Sigma \subseteq 2^\Omega$ be a subset of the power set of Ω. The pair (Ω, Σ) is called an **event space** provided three axioms hold:

(A0) $\Omega \in \Sigma$.

(A1) $A \in \Sigma$ implies that its complement is in Σ, $\Omega - A \in \Sigma$.

(A2) If $A_1, A_2, \ldots$ is a countable sequence of sets in Σ then $\cup_{i \ge 1} A_i$ is in Σ.

We call $A \in \Sigma$ an **event** and singleton sets in Σ are called **elementary events**. The axioms (A0) and (A1) imply that $\emptyset$ and Ω are events (the "impossible event" and "inevitable event"). Moreover, the complement of an event is an event. The set of events are closed under countable unions by axiom (A2). But they are also closed under countable intersections, using de Morgan's law:

hey, a non-event is an event!

$$\bigcap_i \overline{A_i} = \overline{\bigcup_i A_i}. \tag{1}$$

In some contexts, an event space may be[1] called a **Borel field** or **sigma field**

Instead of "(Ω, Σ)", we could also simply call Σ the event space, since Ω is simply the unique set in Σ with the property that $\Sigma \subseteq 2^\Omega$. We use two standard notations for events: for events A and B, we write "A^c" or "$\overline{A}$" for the **complementary event** $\Omega \setminus A$. Also write "AB" for the event $A \cap B$ (why is this an event?). This is called the **joint event** of A and B. Two events A, B are **mutually exclusive** if $AB = \emptyset$.

We now show five basic ways to construct event spaces:

(i) Power set construction: Simply choose Σ to be

$$\Sigma = 2^\Omega, \tag{2}$$

known as the **discrete event space** for Ω. The singleton sets $\{\omega\}$ are called **elementary events**. This is the typical choice for finite sample spaces, such as the running examples (E1) and (E2). Below, we see why the discrete event space (2) may not work when Ω is infinite.

(ii) Generator set construction: let Ω be any set and $G \subseteq 2^\Omega$. Then there is a smallest event space $\overline{G}$ containing G; we call this the event space **generated by** G. For example, if $\Omega = \{1, \ldots, 6\}$ (dice example, E2) and $G = \{\{1,2,3\}, \{3,4,5,6\}\}$, then

by the Axiom of Choice!

$$\overline{G} = \{\emptyset, \{3\}, \{1,2\}, \{1,2,3\}, \{4,5,6\}, \{3,4,5,6\}, \{1,2,4,5,6\}, \{1,2,3,4,5,6\}\}. \tag{3}$$

The reader should verify that $\overline{G}$ here is an event space (wlog, focus on the red subsets, as the blue subsets are their complements).

(iii) Subspace construction: if Σ is an event space and $A \in \Sigma$, then we obtain a new event space $(A, \Sigma \cap 2^A)$, called the **subspace** of Σ **induced** by A. For example, let $A = \{1,2,3\}$ in the previous example (3). Then A induces the subspace $\{\emptyset, \{3\}, \{1,2\}, \{1,2,3\}\}$.

[1]Sometimes, "ring" is used instead of "field".

(iv) Disjoint Union construction: Suppose (Ω_i, Σ_i) $(i = 1, 2)$ are event spaces with disjoint sample spaces: $\Omega_1 \cap \Omega_2 = \emptyset$. We define their **disjoint union** $\Sigma_1 \uplus \Sigma_2$ over the sample space $\Omega := \Omega_1 \uplus \Omega_2$, with the set of events given by $\{A_1 \cup A_2 : A_i \in \Sigma_i\}$. Clearly, this generalizes to countable unions of disjoint event spaces.

(v) Product construction: Given event spaces (Ω_i, Σ_i) $(i = 1, 2)$, let $\Omega := \Omega_1 \times \Omega_2$ (Cartesian product) and

$$\Sigma_1 \times \Sigma_2 := \{A_1 \times A_2 : A_i \in \Sigma_i, i = 1, 2\}.$$

Note that $A_1 \times A_2$ is empty if A_1 or A_2 is empty. However, $\Sigma_1 \times \Sigma_2$ itself is not an event space. So we only use it as a generator set (Method (ii) above) to generate the **product event space** denoted

$$\Sigma_1 \otimes \Sigma_2 := (\Omega, \overline{\Sigma_1 \times \Sigma_2}).$$

The product of an event space Σ with itself may be denoted by Σ^2. For $n \geq 3$, this extends to the n-fold product, Σ^n. Finally, we can combine these product spaces using the countable disjoint union construction (Method (iii) above), giving the event space

$$\Sigma^* := \uplus_{n \geq 0} \Sigma^n.$$

¶3. Event Spaces in the Running Examples. Consider the product construction applied to the coin tossing example, $\Omega = \{H, T\}$. The event space Σ^n is the event space for a sequence of n coin tosses. Letting $n = \infty$ the space Σ^∞ represents coin tosses of arbitrary length. This is useful in studying the problem of tossing a coin until we see a head (we have no a priori bound on how many coin tosses we need).

We illustrate the standard way to create an event space for $\Omega = \mathbb{R}$, via a generating set $G \subseteq 2^\Omega$. Let G comprise the half-lines

$$H_r := \{x \in \mathbb{R} : x \leq r\} \tag{4}$$

for each $r \in \mathbb{R}$. This generates an event space $\overline{G}$ that is extremely important. It is called the **Euclidean Borel field** and denoted B^1 or $B^1(\mathbb{R})$. An element of B^1 is called an **Euclidean Borel set**. These sets are not easy to describe explicitly, but let us see that some natural sets belongs to B^1. We first note that singletons $\{r\}$, $r \in \mathbb{R}$, belong to B^1 because of (1):

Definition of $B^1 = B^1(\mathbb{R})$

$$\{r\} = H_r \cap \bigcap_{n \geq 1} H^c_{r-(1/n)}.$$

Then (A2) implies that any countable set belongs to B^1. A half-open interval $(a, b] = H_b \setminus H_a = (H_a \cup H_b^c)^c$ belongs to B^1. Since singletons are in B^1, we now conclude any open or closed interval belongs to B^1.

¶4. Probability space. So far, we have described concepts that probability theory shares in common with measure theory. Probability properly begins with the next definition: a **probability space** is a triple

measure theory is the foundation integral calculus

$$(\Omega, \Sigma, \Pr)$$

where (Ω, Σ) is an event space and $\Pr : \Sigma \to [0, 1]$ (the unit interval) is a function satisfying these two axioms:

(P0) $\Pr(\Omega) = 1$.

(P1) If $A_1, A_2, \ldots$ is a countable sequence of pairwise disjoint events then $\Pr(\cup_{i\ge1} A_i) = \sum_{i\ge1}\Pr(A_i)$.

We simply call Σ the probability space when Pr is understood. The number $\Pr(A)$ is the **probability** of A. A **null event** is one with zero probability. Clearly, the empty set $\emptyset$ is a null event. It is important to realize that when Ω is infinite, we typically have null events that are different from $\emptyset$. We deduce that $\Pr(\Omega \setminus A) = 1 - \Pr(A)$ and if $A \subseteq B$ are events then $\Pr(A) \le \Pr(B)$.

If Ω is finite, the probability function $\Pr : \Sigma \to [0,1]$ has the form $\Pr(A) = \sum_{\omega\in A} D(\omega)$ for some function

$$D : \Omega \to [0,1]. \tag{5}$$

with the property $1 = \sum_{\omega\in\Omega} D(\omega)$. Conversely, any such D will induce a probability function on any sample space over Ω. We often call such a function D a **probability distribution** over Ω.

128 The student should learn to set up the probabilistic space underlying any probabilistic analysis. Whenever there is discussion of probability, you should ask: what is Ω, what is Σ? This is important since probabilists tend not to show you these spaces this explicitly.

¶5. Probability in the Running Examples. Recall that we have specified Σ for each of our running examples (E1)–(E4). We now assign probabilities to events.

In example (E1), we choose $\Pr(H) = p$ for some $0 \le p \le 1$. Hence $\Pr(T) = 1 - p$. If $p = 1/2$, we say the coin is **fair**. In example (E2), the probability of an elementary event is $1/6$ in the case of a fair dice.

When Ω is a finite set, and $\Pr(A) = |A|/|\Omega|$ for all $A \in \Sigma$, then we see that the probabilistic framework is simply a convenient language for counting: the size of the set event $A \in \Sigma$ is equal to $|\Omega|$ times its probability. The space $(\Omega, 2^\Omega, \Pr)$ where $\Pr(\omega) = 1/|\Omega|$ for all $\omega \in \Omega$ is called the **counting probability model** or **uniform probability model** for Ω.

For (E3), $\Omega = \mathbb{N}$ and we may choose $\Pr(i) = p_i \ge 0$ $(i \in \Omega = \mathbb{N})$ subject to

$$\sum_{i=0}^{\infty} p_i = 1.$$

An explicit example is illustrated by $p_i = 2^{-(i+1)}$, since $\sum_{i=0}^\infty p_i = 2^{-1}\sum_{i=0}^\infty 2^{-i} = 1$. Observe that if $p_i = 0$ for all but for finitely many exceptions, we are back to the case of finite sample spaces. So the situation is only interesting when there are infinitely many non-zero p_i's. In that case, there is no concept of a "uniform probability" function.

For (E4), the event space is the Euclidean Borel field $(\Omega, \Sigma) = (\mathbb{R}, B^1)$ from ¶3. Defining a probability space here can become intricate, but we ought to recognize an easy case: it is too harder than (E3) when the probability function is not continuous, but "concentrated" on a countable subset of $\mathbb{R}$. Such probability functions on B^1 are said to be **discrete**.

To illustrate discrete probability functions, consider the function $\Pr : B^1 \to [0,1]$ where $\Pr(a) = 1/2$, $\Pr(b) = 1/4$ and $\Pr(c) = 1/4$ for some $a, b, c \in \mathbb{R}$. This defines a probability function on B^1 where the probability of any event $A \subseteq \mathbb{R}$ is just the probability of the finite set $A \cap \{a, b, c\}$. This example generalizes to the case where Pr is non-zero on only a countable set $\{a_1, a_2, \ldots\} \subseteq \mathbb{R}$ and $1 = \sum_{i \geq 1} \Pr\{a_i\}$.

To define continuous probability functions on B^1, we need tools from analysis. There is simple but important subcase, when $\Omega = [a, b] \subseteq \mathbb{R}$ is a finite interval. Its sample space, denoted

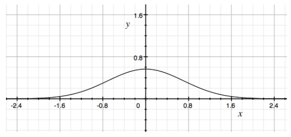
$$B^1[a, b] = ([a, b], \Sigma \cap 2^{[a,b]}),$$

can be obtained from the Borel field B^1 by the subspace construction. More explicitly, $B^1[a, b]$ is generated by the intervals $[a, r] = H_r \cap [a, b]$, for all $c \leq r \leq b$. The simplest continuous probability function for $B^1[a, b]$ is the **uniform probability function** given by

$$\Pr([r, b]) := (b - r)/(b - a) \tag{6}$$

for all generators $[r, b]$ of $B^1[a, b]$. It is not hard to see that $\Pr(A) = 0$ for every countable $A \in \Sigma$. Thus all countable sets are null events.

Continuous Probability Functions for B^1. There is no analogue of the uniform probability function for $B^1[a, b]$. But a most common way to construct a probability space on B^1 is via a continuous function $f : \mathbb{R} \to \mathbb{R}$ with the property that $f(x) \geq 0$, the integral $\int_{-\infty}^{a} f(x)dx$ is defined for all $a \in \mathbb{R}$, and $\int_{-\infty}^{+\infty} f(x)dx = 1$. We call such a function a **density function**. Now define $\Pr(H_a) = \int_{-\infty}^{a} f(x)dx$. For instance, consider the following **Gaussian density function**



$$\phi(x) := \frac{1}{\sqrt{\pi}} e^{-x^2}. \tag{7}$$

Let us verify that it is a density function. Clearly, $\phi(x) \geq 0$. The easiest way to evaluate $\int_{-\infty}^{+\infty} \phi(x)dx$ is to evaluate its square,

$$\begin{aligned}
\left(\int_{-\infty}^{+\infty} \phi(x)dx\right)\left(\int_{-\infty}^{+\infty} \phi(y)dy\right) &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \phi(x)\phi(y)dxdy \\
&= \frac{1}{\pi}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} e^{-(x^2+y^2)}dxdy.
\end{aligned}$$

Next make a change of variable to polar coordinates, $x = r\cos\theta, y = r\sin\theta$. The Jacobian of the change of variable is given by

$$J = \det\frac{\partial(x, y)}{\partial(r, \theta)} = \det\begin{bmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{bmatrix} = r$$

and therefore

$$dxdy = Jdrd\theta = rdrd\theta.$$

We finally get

$$\begin{aligned}
\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} e^{-(x^2+y^2)}dxdy &= \int_0^{2\pi}\int_0^{+\infty} e^{-r^2} rdrd\theta \\
&= \int_0^{2\pi}\left[-\frac{e^{-r^2}}{2}\right]_0^{\infty} d\theta \\
&= \int_0^{2\pi}\left[\tfrac{1}{2}\right] d\theta \\
&= 2\pi\left[\tfrac{1}{2}\right] = \pi
\end{aligned}$$

This proves that our original integral is 1. The probability of an arbitrary interval $\Pr[a, b] = \int_a^b \phi(x)dx$ cannot be expressed as an[2] elementary functions. It can, of course, be numerically approximated to any desired precision. It is also closely related to the error function, $\mathrm{erf}(x) := 2\int_0^x \phi(t)dt$.

[2]That is, a univariate complex function obtained from the composition of exponentials, logarithms, nth roots, and the four rational operations $(+, -, \times, \div)$.

¶6. Probability Spaces from Constructions. We now consider how to assign probabilities to the event spaces constructed using our five basic methods: power set, generator sets, subspaces, disjoint union or product spaces. In the following, assume the probabilistic spaces

$$(\Omega_i, \Sigma_i, \Pr_i) \qquad (i = 1, 2, \ldots). \tag{8}$$

(i) We hinted above that the discrete event space $\Sigma = 2^\Omega$ in (2) may be problematic when Ω is infinite. The reason is that such a space admits many events for which it is unclear how to assign probabilities. This issue is severe when Ω is uncountable. On the other hand, for finite Ω the method of distribution functions (5) is always acceptable.

(ii) Suppose we have an event space formed by disjoint union $\Sigma_1 \uplus \Sigma_2$. We can define probability functions for this event space using existing probability functions $\Pr_1, \Pr_2$ as follows: let $\alpha_1 + \alpha_2 = 1$, $\alpha_i \geq 0$. Construct the probability function denoted

$$\Pr := \alpha_1 \Pr_1 + \alpha_2 \Pr_2$$

where $\Pr : \Sigma_1 \uplus \Sigma_2 \to [0,1]$ is given by

$$\Pr(A_1 \cup A_2) := \alpha_1 \Pr_1(A_1) + \alpha_2 \Pr_2(A_2).$$

(iii) Consider the product event space $\Sigma_1 \otimes \Sigma_2$ from (8). We construct a probability function

$$\Pr_1 \times \Pr_2$$

given by

$$(\Pr_1 \times \Pr_2)(A_1 \times A_2) := \Pr_1(A_1) \Pr_2(A)$$

for generator elements $A_1 \times A_2$. We leave it as an exercise to show this $\Pr_1 \times \Pr_2$ can be extended into a probability function for the product event space.

(iv) Let $\Sigma_1 \cap 2^A$ be the subspace induced by $A \in \Sigma_1$. The induced probability function Pr for this subspace is defined by $\Pr(B \cap A) := \Pr_1(B \cap A)/\Pr_1(A)$ for all $B \in \Sigma_1$.

(v) Finally, we want to give a probability function from generator sets. This is a much deeper result[3] based on Carathéory's Extension Theorem, a result that is outside our scope.

Constantin Carathéodory (1873 – 1950)

¶7. Decision Tree Models. An important type of sample space is based on "sequential decision trees". Assuming a finite tree. The sample space Ω is the set of leaves of the tree and the event space is 2^Ω. How do we assign probabilities to leaves? At each internal node u, we assign a probability to each of its outgoing edges, with the requirement that the probabilities of the edges going out of u sum to 1. For instance, if node u has degree $d \geq 1$ and v is a child, we could assign a probability of $1/d$ to the edge u–v. E.g., the probability of each leaf in Figure 1 is calculated using this uniform model. Intuitively, each internal node u represents a decision and its children represents the outcomes of that decision. Finally, the probability of a leaf is just the product of the probability of the edges along the path from the root to the leaf.

Since a randomized algorithm can be viewed as making a sequence of randomized decisions, such decision tree models are important for algorithmics.

Let us use a decision tree model to clarify an interesting puzzle. In a popular TV game

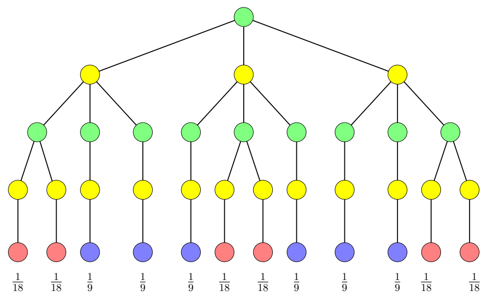


Figure 1: Decision Tree Sample Model

show[4] there are three veiled stages. A prize car is placed on a stage behind one of these veils. You are a contestant in this game, and you win the car if you pick the stage containing the car. The rules of the game are as follows: you initially pick one of the stages. Then the game master selects one of the other two stages to be unveiled – this unveiled stage is inevitably car-less. The game master now gives you the chance to switch your original pick. There are two strategies to be analyzed: *always-switch* or *never-switch*. The never-switch strategy is easily analyzed: you have 1/3 chance of winning. Here are three conflicting claims about the always-switch strategy:

the game is "Let's Make a Deal"

CLAIM (I): Your chance of winning is 1/3, since your chance is the same as the never-switch strategy, as nothing has changed since the start of the game.

CLAIM (II): Your chance of winning is 1/2, since the car is behind one of the two veiled stages.

CLAIM (III): Your chance of winning is 2/3, since it is the complement of the never-switch strategy.

Which of these claims (if any) is correct? What is the flaw in at least two of these claims? To solve this puzzle, we set up a tree model for this problem.

The tree model shown in Figure 2 amounts to a sequence of four choices: it alternates between the game master's choice and your choice. First the game master chooses the veiled stage to place the car (either A, B or C). Then you guess a stage (either A, B or C). Then the game master chooses a stage to unveil. Note that if your guess was wrong, the game master has no choice about which stage to unveil. Otherwise, he has two choices. Finally, you choose to switch or not. The figure only shows the always-switch strategy. Using the uniform probability model, the probability of each leaf is either 1/9 or 1/18, as shown. Each leaf is colored blue if you win, and colored pink if you lose. Clearly, the probability of winning (blue leaves) is 2/3. This proves that CLAIM (III) has the correct probability. Indeed, if we use the never-switch strategy, then the pink leaves would be blue and the blue leaves would be pink. This agrees with CLAIM (III) remark that "the always-switch strategy is the complement of the never-switch strategy."

[3]We can easily extend the generator set G into a "ring" of sets which is closed under finite union and complement. A measure μ on G assigns to each $A \in G$ a non-negative value such that if A_i's are pairwise disjoint then $\mu(\bigcup_i A_i) = \sum_i \mu(A_i)$. Moreover, μ is **σ-finite** in the sense that each $A \in G$ is the countable union of $A_i \in G$ such that $\mu(A_i) < \infty$. Then Carathéodory's Extension Theorem says that μ can be extended to a measure on $\overline{G}$. If this measure is finite, we can turn it into a probability measure.

[4]This problem generated some public interest, including angry letters by professional mathematicians to the New York Times claiming that there ought to be no difference in the two strategies described in the problem.

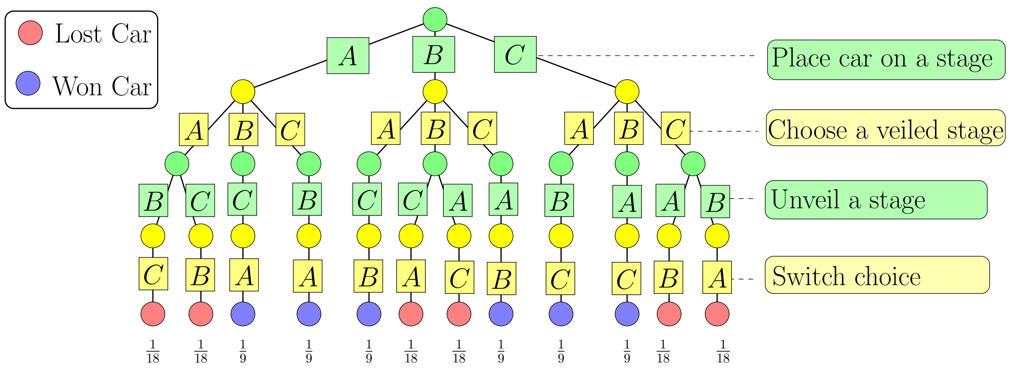


Figure 2: Tree Model for "Let's Make A Deal"

Now we see why CLAIM (I) is wrong is saying "nothing has changed since the start of the game". By revealing one stage, you now know that the car is confined to only two stages. Your original choice has 1/3 chance of being correct; by switching to the "correct one" among the other two choices, you actually have 2/3 chance of being correct. Similarly CLAIM (II) is wrong in saying that "the two remaining veiled stages have equal probability of having the car". We agreed that the one you originally chose has 1/3 chance of having the car. That means the other has 2/3 chance of having the car.

Note that the decision tree in this problem has the property that the decisions at each levels are alternatively made by the two players in this game: the game master makes the decision at the even levels, while you make the decisions at the odd levels. Let us probe a bit further. Do we need the assumption that whenever the game master has a choice of two stages to unveil, both are picked with equal probability? Not at all. In fact the game master can use any rule, including deterministic ones such as choosing to unveil stage A whenever that is possible. Is it essential that the game master place the car on the 3 stages with equal probability? Again, not at all. You as the player must, however, choose the first stage in a truly random manner.

¶8. Intransitive Spinners. Consider three spinners, each with three possible outcomes. Spinner S_0 has (possible) outcomes $\{1, 6, 8\}$, spinner S_1 has outcomes $\{3, 5, 7\}$, and spinner S_2 has outcomes $\{2, 4, 9\}$. The probability that S_0 beats S_1 is 5/9 because $\Pr(S_0 \text{beats } S_1) = \Pr(S_0 = 6 \wedge S_1 = 3 \text{ or } 3) + \Pr(S_0 = 8) = 2/9 + 1/3 = 5/9$. Continuing this way, we check that probability that Spinner S_i beats Spinner S_{i+1} is 5/9 for $i = 0, 1, 2$ (assuming $S_3 = S_0$). So, the relation "spinner X beats spinner Y" is not a transitive relation among spinners. This seems to be surprising.

Let us consider the underlying probability spaces. The relation between two spinners is modeled by the space $S \times S = S^2$ where $S = \{1, 2, 3\}$ where $i \in S$ corresponds to the ith smallest outcome in a spinner. If $i \in S$, let $S_j[i]$ be the ith smallest outcome in spinner S_j. E.g., $S_0[1] = 1, S_0[2] = 6, S_0[3] = 8$. For instance, the event "S_0 beats S_1" is given by $A = \{(2, 1), (2, 2), (3, 1), (3, 2), (3, 3)\} \subseteq S^2$. Thus, $(2, 1) \in A$ because $S_0[2] > S_1[1]$ (i.e., $6 > 3$). But suppose we consider all three spinners at once: the space is $S^3 = \{1, 2, 3\}^3$. At any sample point, we do have a total ordering (and hence transitivity) on the three spinners. Thus the binary relation on spinners is a projection from this larger space. More generally, we can consider various relationships among the pairwise projections of the space S^n. We may not have transitivity, but what other constraints hold?

Exercises

Exercise 1.1: Show that the method of assigning (uniform) probability to events in $B^1[a,b]$ is well-defined. ◇

Exercise 1.2: Let $(\Omega_i, \Sigma_i, \Pr_i)$ be a probability space ($i = 1, 2$). Recall the product construction for these two spaces.
(a) Show that the event space $\overline{\Sigma_1 \times \Sigma_2}$ is not simply the Cartesian product $\Sigma_1 \times \Sigma_2$. Can you give this a simple description?
(b) Show that the probability function $\Pr = \Pr_1 \times \Pr_2$ is well-defined. ◇

Exercise 1.3: The always-switch and never-switch strategies in "Let's Make a Deal" are deterministic. We now consider a randomized strategy: *flip a coin, and to switch only if it is heads*. So this is a mixed strategy – half the time we switch, and half the time we stay put. Analyze this randomized strategy. ◇

Exercise 1.4: In "Let's Make a Deal", the game master's probability for initially placing the car, and the probability of unveiling the stage with no car is not under consideration. The game master might even have some deterministic rule for these decisions. Why is this? ◇

Exercise 1.5: Let us generalize "Let's Make a Deal". The game begins with a car hidden behind one of $m \geq 4$ possible stages. After you make your choice, the game master unveils all but two stages. Of course, the unveiled stages are all empty, and the two veiled stages always include one you picked.
(a) Analyze the always-switch strategy under the assumption that the game master randomly picks the other stage.
(b) Suppose you want to assume the game master is really trying to work against you. How does your analysis change? ◇

Exercise 1.6: (Intransitive Cubes) Consider 4 cubes whose sides have the following numbers:
C_0: $4, 4, 4, 4, 0, 0$
C_1: $3, 3, 3, 3, 3, 3$
C_2: $6, 6, 2, 2, 2, 2$
C_3: $5, 5, 5, 1, 1, 1$
(a) What is the the probability that C_i beats C_{i+1} for all $i = 0, \ldots, 3$? ($C_4 = C_0$)
(b) Let $p \in (0, 1)$ and there exists an assignment of numbers to the faces of the cubes so that the probability that C_i beats C_{i+1} is p for all $i = 0, \ldots, 3$. Show that there exists a biggest value of p. E.g., it is easy to see that $p < 1$. Determine this maximum value of p. ◇

Exercise 1.7: Consider the three spinners A_0, A_1, A_2 in the text where A_i beats A_{i+1} with probability $5/9$ ($i = 0, 1, 2$).
(a) Suppose we spin all three spinners at once: what is the probability p_i of A_i beating the others?
(b) Can you design spinners so that the p_i's are equal ($p_0 = p_1 = p_2$) and retain the original property that A_i beats A_{i+1} with a fixed probability q? (Originally $q = 5/9$ but you may design a different q.) ◇

Exercise 1.8: (Winkler) Six dice are rolled simultaneously, and the number N of distinct numbers that appear is determined. For example, if the dice show $4, 3, 1, 6, 5, 6$ then $N = 5$, and if $1, 3, 1, 3, 3, 3$ then $N = 2$. What is the probability of $N = 4$? ♢

Exercise 1.9: (Winkler) A single die is rolled repeatedly. Event A happens once all six die faces appear at least once. Event B happens once some face (any face) appear four times. If Alice is wishing for event A and Bob for event B, the winner is the one who gets his or her wish first. For example, if the rolls are $2, 5, 4, 5, 3, 6, 6, 5, 1$ then Alice wins. But Bob wins if the rolls are $2, 5, 4, 5, 3, 6, 6, 5, 5$. What is the maximum number of rolls needed to determine a winner? Who is more likely to win? This can be worked out with only a little arithmetic if you are clever. ♢

End Exercises

§2. Independence and Conditioning

Intuitively, the outcomes of two tosses of a coin ought to be "independent" of each other. Conversely, the probability that the sum of two dice rolls is 8 must surely be "conditioned by" the knowledge that one of the dice rolled a 1. In fact, the problem of the sum being 8 is zero. We formalize these two ideas of independence and conditioning.

Don't forget that B is non-null in "$\Pr(A|B)$"

¶9. Conditional Probability. Let $B \in \Sigma$ be any non-null event, *i.e.*, $\Pr(B) > 0$. Such an event B **induces** a probability space which we denote by $\Sigma|B$. The sample space of $\Sigma|B$ is B and event space is $\{A \cap B : A \in \Sigma\}$. The probability function $\Pr_B$ of the induced space is given by

$$\Pr_B(A \cap B) = \frac{\Pr(A \cap B)}{\Pr(B)}.$$

It is conventional to write

$$\Pr(A|B)$$

instead of $\Pr_B(A \cap B)$, and call it the **conditional probability of A given B**.

Two events $A, B \in \Sigma$ are **independent** if $\Pr(AB) = \Pr(A)\Pr(B)$. For example,

A_{ny} : `It is raining in New York`,

A_{sh} : `I go to the park in Shanghai`.

Clearly these are independent events, and so $\Pr(A_{ny}A_{sh}) = \Pr(A_{ny})\Pr(A_{sh})$.

For the first time, we have multiplied two probabilities! The product of two probabilities has an interpretation as the probability of the intersection of two corresponding events, but *only under an independence assumption.* Until now, we have only added probabilities, $\Pr(A) + \Pr(B)$. The sum of two probabilities has an interpretation as the probability of the union of two corresponding events, but *only under an disjointness requirement.* The combination of adding and multiplying probabilities therefore brings a ring-like structure (involving $+, \times$) into play.

It follows that if A, B are independent then $\Pr(A|B) = \Pr(A)$. More generally, a set $S \subseteq \Sigma$ of events is **k-wise independent** if for every $m \le k$, each m-subset $\{A_1, \ldots, A_m\} \subseteq S$ is independent:

$$\Pr(A_1 A_2 \cdots A_m) = \prod_{i=1}^{m} \Pr(A_i).$$

If $k = 2$, we say S is **pairwise independent**. Finally, say S is **completely independent** if it is k-independent for any k.

Clearly, if S is completely independent, then it is k-wise independent. The converse may not hold, as shown in this example: Let $\Omega = \{a, b, c, d\}$. With the counting probability model (¶5) for Ω, consider the events $A = \{a, d\}$, $B = \{b, d\}$, $C = \{c, d\}$. Then we see that $\Pr(A) = \Pr(B) = \Pr(C) = 1/2$ and $\Pr(AB) = \Pr(AC) = \Pr(BC) = 1/4$. Since $\Pr(EF) = \Pr(E)\Pr(F)$ for all events $E \ne F$, we conclude that S is pairwise independent. However S is not independent because $\Pr(ABC) = 1/4 \ne 1/8 = \Pr(A)Pr(B)Pr(C)$. Luckily, in many algorithmic applications, we do not complete independence: just k-wise independence for small values of k (e.g., $k = 2, 3$).

¶10. Bayes' Formula. The starting point for Bayes' formula is a "partition formula". Suppose $A_1, \ldots, A_n$ is a partion of Ω, i.e., $\Omega = \biguplus_{i=1}^n A_i$. Then for any event B, we have

$$\Pr(B) = \Pr(\biguplus_{i=1}^{n} B \cap A_i) = \sum_{i=1}^{n} \Pr(B|A_i)\Pr(A_i). \tag{9}$$

For instance, let $n = 3$ where

$$\begin{array}{rcll}
A_1 & = & \mathtt{sun}: & \text{“}\mathtt{It\ is\ sunny}\text{”}, \\
A_2 & = & \mathtt{rain}: & \text{“}\mathtt{It\ is\ raining}\text{”}, \\
A_3 & = & \mathtt{cloud}: & \text{“}\mathtt{It\ is\ cloudy}\text{”}, \\
B & = & \mathtt{park}: & \text{“}\mathtt{I\ visit\ the\ park}\text{”}. \\
\hline
 & & \multicolumn{2}{l}{\Pr(\mathtt{park}\,|\,\mathtt{rain}) = 0,} \\
 & & \multicolumn{2}{l}{\Pr(\mathtt{park}\,|\,\mathtt{sun}) = 0.5,} \\
 & & \multicolumn{2}{l}{\Pr(\mathtt{park}\,|\,\mathtt{cloud}) = 0.1.}
\end{array}$$

Assume the A_i's are mutually exclusive with $P(A_i) = 1/3$ for each i. So the partition formula (9) tells us

$$\Pr(\mathtt{park}) = \frac{1}{3}(\Pr(\mathtt{park}\,|\,\mathtt{rain}) + \Pr(\mathtt{park}\,|\,\mathtt{sun}) + \Pr(\mathtt{park}\,|\,\mathtt{cloud})) = 0.2.$$

From the partition formula, we now derive

$$\begin{array}{rcll}
\Pr(A_j|B) & = & \frac{\Pr(BA_j)}{\Pr(B)} & (\text{definition of } \Pr(A_j|B)) \\
 & = & \frac{\Pr(B|A_j)\Pr(A_j)}{\Pr(B)} & (\text{definition of } \Pr(B|A_j)) \\
 & = & \frac{\Pr(B|A_j)\Pr(A_j)}{\sum_{i=1}^n \Pr(B|A_i)\Pr(A_i)}. & (\text{by the partition formula (9)}).
\end{array}$$

This result is called **Bayes' formula**:

Thomas Bayes (1701–1761)

$$\Pr(A_j|B) = \frac{\Pr(B|A_j)\Pr(A_j)}{\sum_{i=1}^n \Pr(B|A_i)\Pr(A_i)}. \tag{10}$$

The conditional probabilities $\Pr(B|A_i)$'s on the right hand side of Bayes' formula is interesting: it is an "inversion" of the probability of interest, $\Pr(A_j|B)$.

But first, let us apply this to our park example:

$$\Pr(\mathtt{rain}\,|\,\mathtt{park}) = \frac{\Pr(\mathtt{park}\,|\,\mathtt{rain})\Pr(\mathtt{rain})}{\Pr(\mathtt{park}\,|\,\mathtt{rain})\Pr(\mathtt{rain}) + \Pr(\mathtt{park}\,|\,\mathtt{sun})\Pr(\mathtt{sun}) + \Pr(\mathtt{park}\,|\,\mathtt{cloud})\Pr(\mathtt{cloud})}. = \frac{\Pr(\mathtt{park}\,|\,\mathtt{rain})}{\Pr(\mathtt{park}\,|\,\mathtt{rain}) + \Pr(\mathtt{park}\,|\,\mathtt{sun}) +}$$

Since $\Pr(\texttt{park|rain}) = 0$, we conclude that $\Pr(\texttt{rain|park}) = 0$. In words: the probability of raining is zero, given that you went to the park. So Bayes' formula "predicted" a non-rainy day from the fact that I went to the park. Similarly, $\Pr(\texttt{sun|park}) = 5/6$ and $\Pr(\texttt{cloud|park}) = 1/6$

Bayes' formula may be viewed as an inversion of conditional probability: given that B has occurred, you can determine the probability of any of the mutually exclusive A_j's, *assuming you know* $\Pr(B|A_i)$ *for all* i.

This formula is the starting point for Bayesian probability, the empirical or predictive approach mentioned in the introduction. The goal of Bayesian probability is to use observations to predict the future.

Exercises

Purely for object lesson, of course.

Exercise 2.1: Professor X likes to play a dice game in his probability class, but has no dice. To simulate a dice roll, he asks three students to each toss a fair coin, yielding a binary number between 0 and 7. If 0 or 7 are tossed, the three coins are tossed again. The process is repeated until a number between 1 and 6 is tossed. Prove that this process simulates a fair dice. What is the expected number of individual coin tosses needed to get a dice roll? (Clearly, the number is > 3.) ♢

Exercise 2.2: (K.L. Chung) Recall the Morse code signals of Chapter V. The dot ($\cdot$) and dash ($-$) signals are sent in the proportion 3:4. Because of random signal noise, a dot becomes a dash with probability $1/4$ (3 becomes a 4) and a dash becomes a dot with probability $1/3$ (a 4 becomes a 3). Suppose that the probability of sending a dot or a dash is the same.
(a) If a dot is received, what is the probability that it was a dash?
(b) Only four characters in the Morse Code has a 2-glyph (digraph) encodings:

$$\texttt{A}: \cdot\, - \,, \quad \texttt{I}: \cdot\,\cdot, \quad \texttt{M}: -\,- \,, \quad \texttt{N}: -\,\cdot$$

Under our dot/dash kind of errors, these four characters can be confused with each other. Compute the probability that the character `A` is sent, given that `A` is received. ♢

End Exercises

§3. Random Number Generation and Applications

Randomized algorithms need a source of randomness. Although our main interest in these lectures is randomized algorithms, such a source has many other applications: in the simulation of natural phenomena (computer graphics effects, weather, etc), testing of systems for defects, sampling of populations, decision making and in recreation (dice, card games, etc). One way to get randomness is through physical devices: thermal noise, Geiger counters, Zener Diodes, etc. But such sources may be slow, non-reproducible and may not be unbiased in unknown ways.

¶11. Pseudo-Random Number Generators. In conventional programming languages such as `Java`, `C`, `C++`, etc, the source of randomness is a function which we call `random()` found in the standard libraries of such languages. Each call to `random()` returns some "unpredictable" machine-representable floating point number in the half-open interval $[0, 1)$. Mathematically, what we want is a **random number generator** which returns a real number that is uniformly distributed over the unit interval. This distribution is denoted $U_{[0,1]}$. The function `random()` provides a discrete approximation to $U_{[0,1]}$.

Technically, `random()` is called a **pseudo-random number generator** (PRNG) because it is not truly random. Each PRNG is a parametrized family of number sequences in the unit interval: for each integer parameter s, the PRNG defines an infinite sequence

$$X_s = (X_s(0), X_s(1), X_s(3), \ldots)$$

of integers in some range $[0, m)$. This can be interpreted as a sequence over $[0, 1)$ if we divide each $X_s(i)$ by m. The i-th call to `random()` returns the nearest machine double approximation to $X_s(i)/m$. The sequence X_s is periodic and deterministically generated. The parameter s is freely picked by the user before the initial call to `random()`. We call s the **seed** for this particular instantiation of `random()`. In some sense, there is nothing random about X_s. On the other hand, the sequence X_s satisfies a battery of statistical tests for randomness; this may be sufficient for many applications. The fact that X_s is a deterministic sequence is often important for reproducibility in experiments.

¶12. Linear Congruential Sequences. Perhaps the simplest PRNG is the **linear congruential sequences** determined by the choice of three integer parameters: m (the modulus), a (the multiplier) and c (the increment). These parameters, together with seed s, determine the sequence

$$X_s(i) = \begin{cases} s & \text{if } i = 0, \\ (aX_s(i-1) + c) \,\mathbf{mod}\, m & \text{if } i \ge 1. \end{cases}$$

In some PRNG, `random()` returns $E(X_s(i))$ instead of $X_s(i)$ where E is a "bit-extraction routine' that views $X_s(i)$ as a sequence of bits in its binary notation. For example, in `Java`'s `java.util.Random`, we have

According to Wikipedia...

$$m = 2^{48}, \quad a = 25,214,903,917, \quad c = 11.$$

Thus, each $X_s(i)$ is an 48-bit integer. The bit-extraction routine of `Java` returns the highest order 32 bits, namely bits 47 to 16:

$$E(X_s(i)) = X_s(i)[47 : 16].$$

On the other hand, the standard library `glibc` used by the compiler `GCC` uses

$$m = 2^{31}, \quad a = 1,103,515,245, \quad c = 12345.$$

Thus each $X_s(i)$ is a 31-bit integer, but the bit extraction routine is a no-op (it returns all 31-bits). The theory of linear congruential sequences provide some easily checkable guidelines for how to choose of (m, a, c).

¶13. Chain Rule for Joint Events. We looked at the problem of generating all permutations of n symbols in §V.8. We now look at the problem of generating a random element from this list. This primitive is essential in many random algorithms. Below, we see its use in the a random binary search tree data structure called treaps. The `random()` function above will provide the randomness needed to generate a random permutation. But first, we need a useful formula for computing the probability of a joint event, known as the **chain rule** for joint probability.

From the definition of conditional probability, we have

$$\Pr(AB) = \Pr(A)\Pr(B|A).$$

Then

$$\begin{aligned} \Pr(ABC) &= \Pr(AB)\Pr(C|AB) \\ &= \Pr(A)\Pr(B|A)\Pr(C|AB). \\ \Pr(ABCD) &= \Pr(ABC)\Pr(D|ABC) \\ &= \Pr(A)\Pr(B|A)\Pr(C|AB)\Pr(D|ABC). \end{aligned}$$

More generally,

$$\Pr(A_1A_2\cdots A_n) = \prod_{i=1}^{n} \Pr(A_i|A_1A_2\cdots A_{i-1}). \tag{11}$$

In proof, simply expand the ith factor as $\Pr(A_1A_2,\ldots,A_i)/\Pr(A_1A_2,\ldots,A_{i-1})$, and cancel common factors in the numerator and denominator. This formula is "extensible" in that the formula for $\Pr(A_1\cdots A_n)$ is derived from formula for $\Pr(A_1\cdots A_{n-1})$ just by appending an extra factor involving A_n.

¶14. Random Permutations. Fix a natural number $n \ge 2$. Let S_n denote the set of permutations on $[1..n]$. Our problem is to construct a uniformly random element of S_n using a random number generator.

Let us represent a permutation $\pi \in S_n$ by an array $A[1..n]$ where $A[i] = \pi(i)$ $(i = 1,\ldots,n)$. Here is a simple algorithm from Moses and Oakford (see [12, p. 139]).

```
RandomPermutation
    Input: an array $A[1..n]$.
    Output: A random permutation of $S_n$ stored in $A[1..n]$.
    1.  for $i = 1$ to $n$ do             ◁ Initialize array A
    2.          $A[i] = i$.
    3.  for $i = n$ downto 2 do           ◁ Main Loop
    4.          $X \leftarrow 1 + \lfloor i \cdot \mathtt{random}() \rfloor$.
    5.          Exchange contents of $A[i]$ and $A[X]$.
```

This algorithm takes linear time; it makes $n-1$ calls to the random number generator and makes $n-1$ exchanges of a pair of contents in the array. Here is the correctness assertion for this algorithm:

Lemma 1 *Every permutation of $[1..n]$ is equally likely to be generated.*

Proof. The proof is as simple as the algorithm. Pick any permutation σ of $[1..n]$. Let A' be the array A at the end of running this algorithm. It is enough to prove that

$$\Pr\{A' = \sigma\} = \frac{1}{n!}.$$

Let E_i be the event $\{A'[i] = \sigma(i)\}$, for $i = 1, \ldots, n$. Thus

$$\Pr\{A' = \sigma\} = \Pr(E_1 E_2 E_3 \cdots E_{n-1} E_n).$$

First, note that $\Pr(E_n) = 1/n$. Also, $\Pr(E_{n-1}|E_n) = 1/(n-1)$. In general, we see that

$$\Pr(E_i | E_n E_{n-1} \cdots E_{i+1}) = \frac{1}{i}.$$

The lemma now follows from an application of (11) which shows $\Pr(E_1 E_2 E_3 \cdots E_{n-1} E_n) = 1/n!$. **Q.E.D.**

Note that the conclusion of the lemma holds even if we initialize the array A with any permutation of $[1..n]$. This fact is useful if we need to compute another random permutation in the same array A.

It is instructive to ask what is the underlying probability space? Basically, if A' is the value of the array at the end of the algorithm, then A' is a random permutation in the sense of §3. That is,

$$A' : \Omega \to S_n$$

where Ω is a suitable probability space and S_n is the set of n-permutations. We can view Ω as the set $\prod_{i=2}^{n}[0,1)$ where a typical $\omega \in \Omega = (x_2, x_3, \ldots, x_n)$ tells us the sequence of values returned by the $n-1$ calls to the `random()` function.

Remarks: Random number generation is an extensively studied topic: Knuth [12] is a basic reference. The concept of randomness is by no means easily pinned down. From the complexity viewpoint, there is a very fruitful approach to randomness called Kolmogorov Complexity. A comprehensive treatment is found in Li and Vitányi [14].

Exercises

Exercise 3.1: Student Joe Quick suggests the following algorithm for computing a random permutation of $\{1, \ldots, n\}$:

```
RandomPermutation
    Input: an array $A[1..n]$.
    Output: A random permutation of $S_n$ stored in $A[1..n]$.
    1.  for $i = 1$ to $n$ do
    2.      $A[i] = 0$.            ◁ Initially, each entry is “empty”
    3.  for $i = n$ downto 1 do        ◁ Main Loop
    4.      $j \leftarrow 1 + \lfloor i \cdot \texttt{random}() \rfloor$.
    5.      “Put $i$ into the $j$-th empty slot of array $A$.”
```

Line 5 has the obvious interpretation:

```
PUT($i, j$):
    ▷ To put $i$ into the $j$-th empty slot
    for $k = 1$ to $n$
        If ($A[k] = 0$)
            $j$--
            If ($j = 0$)
                $A[k] \leftarrow i$
                Return.
```

Prove that J.Quick's intuition about this algorithm is correct. ♢

End Exercises

§4. Random Variables

The concepts so far have not risen much above the level of "gambling and parlor games" (the pedigree of our subject). Probability theory really takes off after we introduce the concept of random variables. Intuitively, a random variable is a function that assigns a real value to each point in a sample space.

¶15. Random Variables – the simple case. As introductory example, consider a discrete event space $(\Omega, 2^\Omega)$ for a finite set Ω. Then a random variable is just any function of the form $X : \Omega \to \mathbb{R}$. The event

$$X^{-1}(c) = \{\omega \in \Omega : X(\omega) = c\}$$

will be written suggestively as "$\{X = c\}$" with probability "$\Pr\{X = c\}$". Clearly, for all but finitely many $c \in \mathbb{R}$, we have $\Pr\{X = c\} = 0$.

Using our running example (E1) where $\Omega = \{H, T\}$, consider the random variable X given by the assignment $X(H) = 100$ and $X(T) = 0$. Assume a fair coin, $\Pr\{X = 100\} = \Pr\{X = 0\} = 0.5$. This random variable might represent a game in which you win \$100 if the coin toss is a head, and you win nothing if a tail. In informal language, we would say that "on average, you expect to win \$50". This may be borne out experimentally if you play the game many times. More generally, if the probabilities might be skewed, say $\Pr\{X = 100\} = p$ and $\Pr\{X = 0\} = q = 1 - p$ for some $0 < p < 1$ In this case, we say that your "expected win" is $\Pr\{X = 100\} \cdot 100 + \Pr\{X = 0\} \cdot 0 = p \cdot 100 + q \cdot 0 = 100p$. I.e., in this game, you expect to win \$$100p$. Thus we see that a basic thing we do with random variables is to compute their expected values.

An actual coin-toss experiment (Shanghai, 24 Nov 2020): `T,T,H,H,T,T,T,H,T,H.` Average win \$40.

¶16. Random Variables – the general case. In the above example, it was easy to assign a probability to events such as $\{X = c\}$. But if Ω is infinite, this is trickier. The solution relies on the concept of Borel fields.

A **random variable** (abbreviated as r.v.) in a probability space $(\Omega, \Sigma, \Pr)$ is a real function

$$X : \Omega \to \mathbb{R}$$

such that for all $r \in \mathbb{R}$,

$$X^{-1}(H_r) = \{\omega \in \Omega : X(\omega) \le r\} \qquad (12)$$

belongs to Σ, where H_r is a generator of the Euclidean Borel field B^1 (see (4)). Sometimes the range of X is the extended reals $\mathbb{R} \cup \{\pm\infty\}$.

Why do we need (12)? It allows us to assign probabilities to the event "X is less than r" (equivalently, $X(\omega) \in H_r$). Since these H_r events generate all other events in B^1, we then have a meaningful way to talk about "$X \in A$" for any Euclidean Borel set $A \in B^1$. By analogy with (12), the event "$X \in A$" is the set

$$X^{-1}(A) = \{\omega \in \Omega \,:\, X(\omega) \in A\}. \tag{13}$$

We ask the student to verify the claim that $X^{-1}(A)$ is an event. Probabilists will denote the event (13) using the set-like notation

$$\{X \in A\}. \tag{14}$$

Convention. Writing (14) for (13) illustrates the habit of probabilists to avoid explicitly mentioning sample points. More generally, probabilists will specify events by writing $\{\ldots X \ldots Y \ldots\}$ where "$\ldots X \ldots Y \ldots$" is some predicate on r.v.'s X, Y, etc. This really denotes the event $\{\omega \in \Omega : \ldots X(\omega) \ldots Y(\omega) \ldots\}$. For instance, $\{X \le 5, X+Y > 3\}$ refers to the event $\{\omega \in \Omega : X(\omega) \le 5, X(\omega)+Y(\omega) > 3\}$. Moreover, instead of writing $\Pr(\{\ldots\})$, we simply write $\Pr\{\ldots\}$, where the curly brackets remind us that $\{\ldots\}$ is a set (which happens to be an event).

probability is hard, like learning any new language ...

An important property about r.v.'s is their closure property under the usual numerical operations: if X, Y are r.v.'s then so are

$$\min(X,Y), \quad \max(X,Y), \quad X \pm Y, \quad XY, \quad X^Y, \quad X/Y.$$

In the last case, we require $Y^{-1}(0) = \emptyset$.

¶17. Two Types of Random Variables.

All random variables in probability theory are either discrete or continuous. Random variables over finite sample spaces are easy to understand, and falls under the discrete case. However, the discrete case covers a bit more.

A r.v. X is **discrete** if the range of X,

$$X(\Omega) := \{X(\omega) : \omega \in \Omega\}$$

is countable. Clearly if Ω is countable, then X is automatically discrete. The simple but important case is when $X(\Omega)$ has size 2; we then call X a **Bernoulli r.v.**. Typically, $X(\Omega) = \{0,1\}$ as in our introductory example of coin tossing (that is why coin tossing experiments are often called Bernoulli trials). When $X(\Omega) = \{0,1\}$, we also call X the **indicator function** of the event $\{X = 1\}$. Thus X is the indicator function for the "head event" for the coin tossing example. In discrepancy theory, the Bernoulli r.v. typically have $X(\Omega) = \{+1, -1\}$.

Suppose the probability space is the Borel space $B^1[a,b]$ (where $-\infty \le a < b \le +\infty$) from ¶3. A r.v. X is **continuous** if there exists a nonnegative function $f(x)$ defined for all $x \in \mathbb{R}$ such that for any Euclidean Borel set $A \in B^1$, the probability is given by an integral:

$$\Pr\{X \in A\} = \int_A f(x)dx$$

(cf. (14)). In particular, for an interval $A = [c,d] \subseteq [a,b]$, we have $\Pr\{X \in A\} = \Pr\{c \le X \le d\} = \int_c^d f(x)dx$. For instance, this implies $\Pr\{X = c\} = 0$ for all c. We call $f(x)$ the **density function** of X. It is easy to see that density functions must be non-negative $f(x) \ge 0$ and $\int_a^b f(x)dx = 1$.

¶18. Expected Values. In our introductory example, we noted that a basic application of r.v.'s is to compute their average or expected values. If X is a discrete r.v. whose range is

$$\{a_1, a_2, a_3 \ldots\} \tag{15}$$

then its **expectation** (or, **mean**) $\mathtt{E}[X]$ is defined to be

$$\mathtt{E}[X] := \sum_{i \ge 1} a_i \Pr\{X = a_i\}. \tag{16}$$

This is well-defined provided the series converges absolutely, *i.e.*, $\sum_{i\ge1} |a_1| \Pr\{X = a_i\}$ converges. If X is a continuous r.v., we must replace the countable sum in (16) by an integral. If the density function of X is $f(x)$ then its expectation is defined as

$$\mathtt{E}[X] := \int_{-\infty}^{\infty} uf(u)du. \tag{17}$$

Note that if X is the indicator variable for an event A then

$$\mathtt{E}[X] = \Pr(A).$$

As examples of random variables, suppose in running example (E1), if we define $X(H) = 1, X(T) = 0$ then X is the indicator function of the "head event". For (E2), let us define $X(i) = i$ for all $i = 1, \ldots, 6$. If we have a game in which a player is paid i dollars whenever the player rolls an outcome of i, then X represents "payoff function".

¶19. How long do I wait for success? An infinite sequence of Bernoulli r.v.s,

$$X_1, X_2, X_3, \ldots \tag{18}$$

is called a **Bernoulli process** if the X_i's are independent and having the same probability distribution. Each X_i is called a **trial**. So the sample space is $\Omega = \{0,1\}^\infty$. Here are two sample points: $\omega_1 = (1,1,1,1,1,\ldots)$ (all 1's) and $\omega_2 = (0,0,1,0,0,1,0\ldots)$ (two 0's followed by a 1, repeated in this way). Let us suppose $\Pr\{X = 1\} = p$ and $\Pr\{X = 0\} = q := 1 - p$. This defines a probability function $\Pr : \Sigma \to [0,1]$. For instance, the event $\{X_1 = 1, X_3 = 0\}$ has probability $p(1-p)$. Let us interpret the event $\{X = 1\}$ as "success" and the event $\{X = 0\}$ as "failure". Let Y denote the random variable indicating the number of trials until we encounter the first success. For instance, $Y(\omega_1) = 1$ and $Y(\omega_2) = 3$ in the earlier sample points. We can now ask: what is the expected value of Y? The answer is very simple: $\mathtt{E}[Y] = 1/p$. For example, if you keep tossing a fair coin, your expect number of tosses before you see a head is $1/p = 2$. Let us prove this.

$$\begin{array}{rcl} \mathtt{E}[Y] & = & \sum_{i=1}^{\infty} i \cdot \Pr\{Y = i\} \\ & = & \sum_{i=1}^{\infty} iq^{i-1}p \\ & = & p\sum_{i=1}^{\infty} iq^{i-1} \\ & = & p\frac{1}{(1-q)^2} \\ & = & p\frac{1}{p^2} = 1/p. \end{array}$$

Exercises

Exercise 4.1: On a trip to Las Vegas, you stop over at a casino to play a game of chance. You have \$10, and each play costs \$2. You have 1/3 chance to win \$5, and 2/3 chance to win nothing. You just want to play as long as you have money. How many games do you expect to play? ♢

Exercise 4.2: (Craps Principle) Prove that if A, B are mutually exclusive events, then $\Pr(A|A \cup B) = \frac{\Pr(A)}{\Pr(A)+\Pr(B)}$. Note that $A \cup B$ need not be exhaustive, i.e., we do not require $\Pr(A \cup B) = 1$. ♢

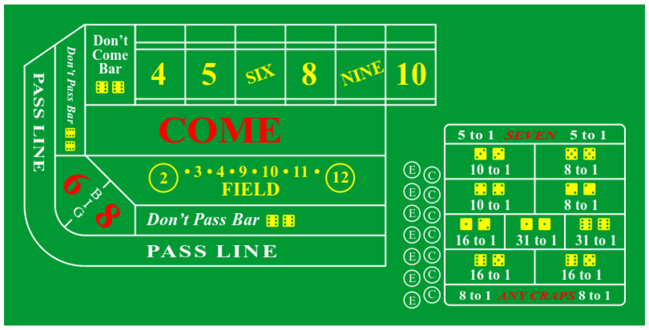


Figure 3: Craps Table Layout in Casinos

Exercise 4.3: The dice game of craps is played as follows: there are two phases. In the "Come-out Phase", the player (called the "shooter") throws two dice and if the sum is two, three or twelve, then he loses ("craps"). If the sum is seven or eleven, then he wins. If the sum is anything else (namely, $4, 5, 6, 8, 9, 10$), this establishes the "point", and begins the "Point Phase". Now the shooter keeps throwing dice until he either throws the "point" again (in which case he wins) or he throws a seven (and loses). Note that to win the point, he does not need to repeat the original combination of the point. E.g., he may initially establish the point of 6 with a roll of $(3, 3)$, but later win the point with a roll of $(1, 5)$ or $(2, 4)$. Calculate the probability of the shooter winning. Note that 7 has the most probable outcome, $\Pr(7) = 1/6$. HINT: the Craps Principle is useful here, $\Pr(A|A \cup B) = \Pr(A)/\Pr(A \cup B)$. ♢

Exercise 4.4: Describe the probability space for the Bernoulli process (18). Justify that it is indeed a probability space. ♢

Exercise 4.5: Consider the following randomized process, which is a sequence of steps. At each step, we roll a dice that has one of six possible outcomes: $1, 2, 3, 4, 5, 6$. In the i-th step, if the outcome is less than i, we stop. Otherwise, we go to the next step. For the first step, $i = 1$, and it is impossible to stop at this step. Moreover, we surely stop by the 7-th step. Let T be the random variable corresponding to the number of steps.
(a) Set up the sample space, event space, and probability function for T.
(b) Compute the expected value of T. ♢

Exercise 4.6: (R. Morris) Let C be a binary counter which is initially 0, you can perform the operation `inc`(C) to increments its value by 1. To save space, we want to **probabilistic counting**.

(a) Here is a warmup: each time you call inc(C), it flips a fair coin, and increments the counter C if you get a head. If you wish to query the counter, you call a routine look(C) which will return *twice* the current value of C. Let C_n denote the value of C C after n calls to inc(C). So C_n is a random variable. Show that $E[\texttt{look}(C_n)]$ is n. NOTE: So the expected value of look(C) is what we expect from a counter. What have we gained? We saved 1 bit of space since the counter value C_n is only $n/2$ (in expectation).
(b) Extend the previous idea to save k bits of space (for any k).
(c) The ultimate is this: define inc(C) as follows: flip a fair coin C times, and increment the value of C only if we see C heads. Define look(C) to return $2^C - 2$. E.g., if $C = 5$, then inc(C) increments C to 7 with probability 2^{-5}. And look(C) returns $2^5 - 2 = 30$. Prove that $\texttt{E}[\texttt{look}(C_n)] = n$. What have you gained in using this counter.
(d) Compute the variance of 2^C. ♢

End Exercises

§5. Random Objects

We seek to generalize the concept of random variables. Let D be a set and $(\Omega, \Sigma, \Pr)$ a probability space. A **random function over** D is a function

$$f : \Omega \to D$$

such that for each $x \in D$, the set $f^{-1}(x)$ is an event. Here, $(\Omega, \Sigma, \Pr)$ is the **underlying probability space** of f.

We may speak of the **probability** of $x \in D$, *viz.*, $\Pr(f^{-1}(x)) = \Pr\{f = x\}$. We say f is **uniformly distributed on** D if $\Pr(f^{-1}(x)) = \Pr(f^{-1}(y))$ for all $x, y \in D$. We often also use bold fonts ($\mathbf{f}$ instead of f, etc) to denote random functions. Here are some common choices of D:

- If $D = \mathbb{R}$, then f is a **random number**. Of course, f is a random variable as defined earlier. But the definition of random number has weaker properties than random variables because its definition does not refer to the Borel field (so we may not able to do as much with it).
- If D is some set of graphs we call f a **random graph**. E.g., D is the set of bigraphs on $\{1, 2, \ldots, n\}$.
- For any finite set S, we call f a **random k-set** of S if $D = \binom{S}{k}$.
- If D is the set of permutations of S, then f is a **random permutation** of S.
- For any set D, we may call f a **random D-element**.

In each example, the elements of D are objects of some category T (numbers, graphs, k-sets, D-objects). Then f is called a **random T object**. If D is a finite set, we also say f is **uniformly random** if $\Pr\{f = d\} = 1/|D|$ for all $d \in D$.

The power of random objects is that they are composites of the individual objects of D. For many purposes, these objects are as good as the honest-to-goodness objects in D. Another view of this phenomenon is the philosophical idea of alternative or possible worlds. Each $\omega \in \Omega$ is a possible world. Then $f(\omega)$ is just the particular incarnation of f in the world ω.

Good, ω means <u>w</u>orld!

¶20. Example: (Random Points in Finite Fields) Consider the uniform probability space on $\Omega = F^2$ where F is any finite field. The simplest examples of such fields are $\mathbb{Z}_p$ (integers mod p for some prime p). For each $x \in F$, consider the random function

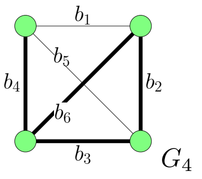
$$\begin{aligned} \mathbf{h}_x &: \Omega \to F, \\ \mathbf{h}_x(\langle a, b\rangle) &= ax + b, \qquad (\langle a, b\rangle \in \Omega). \end{aligned}$$

We claim that $\mathbf{h}_x$ is a uniform random element of F, *i.e.*, $\Pr\{\mathbf{h}_x = i\} = 1/|F|$ for each $i \in F$. This amounts to saying that there are exactly $|F|$ sample points $\langle a, b\rangle = \omega$ such that $\mathbf{h}_x(\omega) = i$. To see this, consider two cases: (1) If $x = 0$ then $h_0(\langle a, b\rangle) = i$ implies $b = i$. But a can be any element in F, so this shows that $|h_0^{-1}(i)| = |F|$, as desired. (2) If $x \neq 0$, then for any choice of b, there is unique choice of a, namely $a = (i - b)x^{-1}$.

¶21. Example: (Random Graphs) Fix $0 \le p \le 1$ and $n \ge 2$. Consider the probability space where $\Omega = \{0, 1\}^m$, $m = \binom{n}{2}$, $\Sigma = 2^\Omega$ and for $(b_1, \ldots, b_m) \in \Omega$, $\Pr(b_1, \ldots, b_m) = p^k(1 - p)^{m-k}$ where k is the number of 1's in $(b_1, \ldots, b_m)$. Once checks that Pr as defined is a probability function. Let K_n be the complete bigraph on n vertices whose edges are labelled with the integers $1, \ldots, m$. Consider the "random graph" which is just the function

$$G_{n,p} : \Omega \to \{\text{subgraphs of } K_n\} \tag{19}$$

where $G_{n,p}(b_1, \ldots, b_m)$ is the subgraph of K_n such that the ith edge is in $G_{n,p}(b_1, \ldots, b_m)$ iff $b_i = 1$. Let us be concrete: let $n = 4$, $m = \binom{4}{2} = 6$, $p = 1/3$ and G_4 be the graph in the margin. Then $\Pr(G_4) = (1/3)^4(2/3)^2 = 4/3^6$. Of course, $4/3^6$ is the probability of any subgraph of K_4 with 4 edges. $\blacksquare$



¶22. Random Statistics. Random variables arise from random objects as follows. A function $C : D \to \mathbb{R}$ is called a **statistic** of D where D is some set of objects. If $g : \Omega \to D$ is a random object, we obtain the random variable $C_g : \Omega \to \mathbb{R}$ where

$$C_g(\omega) = C(g(\omega)).$$

Call C_g a **random statistic** of g.

For example, let $g = G_{n,p}$ be the random graph in equation (19). and let C count the number of Hamiltonian cycles in a bigraph. Then the random variable

$$C_g : \Omega \to \mathbb{R} \tag{20}$$

is defined so that $C_g(\omega)$ is the number of Hamiltonian cycles in $g(\omega)$.

¶23. Independent Ensembles. A collection K of random D-objects with a common underlying probability space is called an **ensemble** of D-objects. We extend the concepts of independent events to "independent ensembles".

A finite ensemble $\{X_1, X_2, \ldots, X_n\}$ of n r.v.'s is **independent** if for all m-subset $\{i_1, i_2, \ldots, i_m\} \subseteq \{1, 2, \ldots, n\}$ and all $c_1, \ldots, c_m \in \mathbb{R}$, the events $\{X_{i_1} \le c_1\}, \ldots, \{X_{i_m} \le c_m\}$ are independent. Note that this is well-defined because the r.v.'s share a common probability space. An ensemble (finite or infinite) is **k-wise independent** if for all $m \le k$, each m-subset of the ensemble is independent. As usual, **pairwise independent** means $2-wise$ independent.

Let K be an ensemble of D-objects where D is a finite set. We say K is **k-wise independent** if for any $a_1, \ldots, a_k \in D$, and any k-subset $\{f_1, \ldots, f_k\} \subseteq K$, the events $\{f_1 = a_1\}, \ldots, \{f_k = a_k\}$ are independent.

¶24. Example: (Finite Fields, contd) Recall the finite field space $\Omega = F^2$ above. Consider the ensemble

$$K = \{\mathbf{h}_x : x \in F\} \tag{21}$$

where $\mathbf{h}_x(\langle a, b\rangle) = ax + b$ as before. We had seen that the elements of K are uniformly random. Let us show that the ensemble K is pairwise independent: Fix $x, y, i, j \in F$ and let $n = |F|$. Suppose $x \neq y$ and $\mathbf{h}_x = i$ and $\mathbf{h}_y = j$. This means

$$\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} i \\ j \end{pmatrix}.$$

The 2×2 matrix is invertible and hence (a, b) has a unique solution. Hence

$$\Pr\{\mathbf{h}_x = i, \mathbf{h}_y = j\} = 1/n^2 = \Pr\{\mathbf{h}_x = i\} \Pr\{\mathbf{h}_y = j\},$$

as desired. Algorithmically, constructions of k-wise independent variables over a small sample space (e.g., $|\Omega| = p^2$ here) is important because it allows us to make certain probabilistic constructions effective.

Exercises

Exercise 5.1: Compute the probability of the event $\{C_g = 0\}$ where C_g is given by (20). Do this for $n = 2, 3, 4$. ♢

Exercise 5.2: Let $\Omega = F$ be a finite field, and for $x \in F$, consider the random function $\mathbf{h}_x : \Omega \to F$ where $\mathbf{h}_x(a) = ax$. Consider the ensemble $K = \{\mathbf{h}_x : x \in F \setminus \{0\}\}$. Show that each $\mathbf{h}_x \in K$ is a uniformly random element of F, but K is not pairwise independent. ♢

Exercise 5.3: Let $f_1, \ldots, f_n$ be real functions $f_i : \mathbb{R} \to \mathbb{R}$ and $\{X_1, \ldots, X_n\}$ is an independent ensemble of r.v.'s. If $f_i(X_i)$ are also r.v.'s then $\{f_1(X_1), \ldots, f_n(X_n)\}$ is also an independent ensemble. ♢

Exercise 5.4: Consider the following randomized process, which is a sequence of probabilistic steps. At each step, we roll a dice that has one of six possible outcomes: $1, 2, 3, 4, 5, 6$. In the i-th step, if the outcome is less than i, we stop. Otherwise, we go to the next step. The first step is $i = 1$. For instance, we never stop after first step, and surely stop by the 7-th step. Let T be the random variable corresponding to the number of steps.
(a) Set up the sample space, the event space, and the probability function for T.
(b) Compute the expected value of T. ♢

Exercise 5.5: Let K be an ensemble of random elements in the finite field F (21).
(a) Show that K is not 3-wise independent.
(b) Generalize the example to construct a collection of k-wise independent random functions. ♢

Exercise 5.6: Let $W(n, x)$ (where $n \in \mathbb{N}$ and $x \in \mathbb{Z}_n$) be a "witness" predicate for compositeness: if n is composite, then $W(n, x) = 1$ for at least $n/2$ choices of x; if n is prime, then $W(n, x) = 0$ for all x. Let $W(n)$ be the random variable whose value is determined by a random choice of x. Let $W_t(n)$ be the random variable whose value is obtained as

follows: randomly choose n values $x_1, \ldots, x_n \in \mathbb{Z}_n$ and compute each $W(n, x_i)$. If any $W(n, x_i) = 1$ then $W_t(n) = 1$ but otherwise $W_t(n) = 0$.
(a) If n is composite, what is the probability that $W_t(n) = 1$?
(b) Now we compute $W_t(n)$ using somewhat less randomness: first assume t is prime and larger than n. only randomly choose two values $a, b \in \mathbb{Z}_t$. Then we define $y_i = a \cdot i + b (\bmod t)$. We evaluate $W_t(n)$ as before, except that we use $y_0, \ldots, y_{t-1} (\bmod n)$ instead of the x_i's. Lower bound the probability that $W_t(n) = 1$ in this new setting. $\diamondsuit$

End Exercises

§6. Expectation and Variance

Two important numbers are associated with a random variable: its "average value" and its "variance" (expected deviation from the average value). We had already defined the average value, $\mathtt{E}[X]$. Let us look at some of its properties.

¶25. Two Remarkable Properties of Expectation. We look at two elementary properties of expectation that often yield surprising consequences. The first is called **linearity of expectation**.

Lemma 2 *For all r.v.'s X, Y and $\alpha, \beta \in \mathbb{R}$:*

$$\mathtt{E}[\alpha X + \beta Y] = \alpha \mathtt{E}[X] + \beta \mathtt{E}[Y].$$

Here is the proof in the discrete case: $\mathtt{E}[\alpha X + \beta Y] = \sum_\omega (\alpha X(\omega) + \beta Y(\omega)) \Pr(\omega) = \alpha \mathtt{E}[X] + \beta \mathtt{E}[Y]$. What is remarkable is the fact that X and Y are completely arbitrary — in particular, there are no independence assumptions. In applications, we decompose a r.v. X of interest into *some* linear combination of simpler r.v.s $X_1, X_2, \ldots, X_m$. If we can compute the expectations of each X_i, then by linearity of expectation, we obtain the expectation of X itself. E.g., X may be the running time of a n-step algorithm and X_i is the expected time for the ith step.

The second remarkable property is that, from expectations, we can infer the existence of objects with certain properties.

Lemma 3 *Let X be a discrete r. v. with finite expectation μ. If Ω is finite, then:*

(i) *There exists $\omega_0, \omega_1 \in \Omega$ such that*

$$X(\omega_0) \le \mu \le X(\omega_1). \tag{22}$$

(ii) *If X is non-negative and $c > 0$ then*

$$\Pr\{X \ge c\mu\} \le 1/c. \tag{23}$$

In particular, if $\Pr\{\cdot\}$ is uniform and Ω finite, then more than half of the sample points $\omega \in \Omega$ satisfy $X(\omega) < 2\mu$.

Proof. Since X is discrete, let

$$\mu = \mathtt{E}[X] = \sum_{i=1}^{\infty} a_i \Pr\{X = a_i\}.$$

(i) If there are arbitrarily negative a_i's then clearly ω_0 exists; otherwise choose ω_0 so that $X(\omega_0) = \inf\{X(\omega) : \omega \in \Omega\}$. Likewise if there are arbitrarily large a_i's then ω_1 exists, and otherwise choose ω_1 so that $X(\omega_1) = \sup\{X(\omega) : \omega \in \Omega\}$. In every case, we have chosen ω_0 and ω_1 so that the following inequality confirms our lemma:

$$X(\omega_0) = X(\omega_0) \sum_{\omega\in\Omega} \Pr(\omega) \le \sum_{\omega\in\Omega} \Pr(\omega) X(\omega) \le X(\omega_1) \sum_{\omega\in\Omega} \Pr(\omega) = X(\omega_1).$$

(ii) This is just Markov's inequality (below). We have $c\mu \cdot \Pr\{X \ge c\mu\} \le \mathtt{E}[X] = \mu$, so $\Pr\{X \ge c\mu\} \le \frac{1}{c}$. **Q.E.D.**

We remark that part(ii) in this Lemma is trivial when $c < 1$. Similarly, the existence of ω_0 or ω_1 in part(i) might also be trivial. To apply this lemma, we set up a random D object,

$$g : \Omega \to D$$

and are interested in a certain statistic $C : D \to \mathbb{R}$. Define the random statistic $C_g : \Omega \to \mathbb{R}$ as in (20). Then there exists ω_0 such that

$$C_g(\omega_0) \le \mathtt{E}[C_g]$$

This means that the object $g(\omega_0) \in D$ has the property $C(g(\omega_0)) \le \mathtt{E}[C_g]$.

Linearity of expectation amounts to saying that summing r.v.'s is commutative with taking expectation. What about products of r.v.'s? If X, Y are independent then

$$\mathtt{E}[XY] = \mathtt{E}[X]\mathtt{E}[Y]. \tag{24}$$

The requirement that X, Y be independent is necessary. As noted earlier, all multiplicative properties of probability depends from some form of independence.

The jth **moment** of X is $\mathtt{E}[X^j]$. If $\mathtt{E}[X]$ is finite, then we define the **variance** of X to be

$$\mathtt{Var}(X) := \mathtt{E}[(X - \mathtt{E}[X])^2].$$

Note that $X - \mathtt{E}[X]$ is the deviation of X from its mean. It is easy to see that

$$\mathtt{Var}(X) = \mathtt{E}[X^2] - \mathtt{E}[X]^2.$$

The positive square-root of $\mathtt{Var}(X)$ is called its **standard deviation** and denoted $\sigma(X)$ (so $\mathtt{Var}(X)$ is also written $\sigma^2(X)$). If X, Y are independent, then summing r.v.'s also commutes with taking variances. More generally:

Lemma 4 *Let X_i $(i = 1, \ldots, n)$ be pairwise independent random variables with finite variances. Then*

$$\mathtt{Var}(\sum_{i}^{n} X_i) = \sum_{i=1}^{n} \mathtt{Var}(X_i).$$

This is a straightforward computation, using the fact that $\mathtt{E}[X_iX_j] = \mathtt{E}[X_i]\mathtt{E}[X_j]$ for $i \ne j$ since X_i and X_j are independent.

¶26. Distribution and Density. For any r.v. X, we define its **distribution function** to be $F_X : \mathbb{R} \to [0,1]$ where

$$F_X(c) := \Pr\{X \le c\}, \qquad c \in \mathbb{R}.$$

The importance of distribution functions stems from the fact that the basic properties of random variables can be studied from their distribution function alone.

Two r.v.'s X, Y can be related as follows: we say X **stochastically dominates** Y, written

$$X \succeq Y$$

if $F_X(c) \le F_Y(c)$ for all c. This implies (Exercise) $\mathtt{E}[X] \ge \mathtt{E}[Y]$ if X stochastically dominates Y. If $X \succeq Y$ and $Y \succeq X$ then we say they are **identically distributed**, denoted

$$X \sim Y.$$

A common probabilistic setting is an independent ensemble K of r.v.'s that all share the same distribution. We then say K is **independent and identically distributed** (abbrev. i.i.d) ensemble. For instance, when X_i is the outcome of the ith toss of some fixed coin, then $K = \{X_i : i = 1, 2, \ldots\}$ is an i.i.d. ensemble.

In general, a **distribution function** [5] $F(x)$ is a monotone non-decreasing real function such that $F(-\infty) = 0$ and $F(+\infty) = 1$. Sometimes, a distribution function $F(x)$ is defined via a **density function** $f(u) \ge 0$, where

$$F(x) = \int_{-\infty}^{x} f(u)du.$$

In case X is discrete, the density function $f_X(u)$ (of its distribution function F_X) is zero at all but countably many values of u. As defined above, a continuous r.v. X is specified by its density function.

¶27. Conditional Expectation. This concept is useful for computing expectation. If A is an event, define the **conditional expectation** $\mathtt{E}[X|A]$ of X to be $\sum_{i\ge1} a_i \Pr\{X = a_i|A\}$. In the discrete event space, we get

$$\mathtt{E}[X|A] = \frac{\sum_{\omega\in A} X(\omega)\Pr(\omega)}{\Pr(A)}.$$

If B is the complement of A, then

$$\mathtt{E}[X] = \mathtt{E}[X|A]\Pr(A) + \mathtt{E}[X|B]\Pr(B).$$

Next, if Y is another r.v., we define a new r.v. $Z := \mathtt{E}[X|Y]$ where $Z(\omega) = \mathtt{E}[X|Y = Y(\omega)]$ for any $\omega \in \Omega$. We can compute the expectation of X using the formula

$$\begin{aligned} \mathtt{E}[X] &= \mathtt{E}[\mathtt{E}[X|Y]] && (25)\\ &= \sum_{a\in\mathbb{R}} \mathtt{E}[X|Y=a]\Pr\{Y=a\}. && (26) \end{aligned}$$

[5]Careful: $D : \Omega \to [0,1]$ was also called a probability distribution function. But the settings are usually different enough to avoid confusion: Ω is finite in the case of D.

For example, let X_i's be i.i.d., and N be a non-negative integer r.v. independent of the X_i's. What is the expected value of $\sum_{i=1}^N X_i$?

$$\begin{aligned} \mathtt{E}[\sum_{i=1}^N X_i] &= \mathtt{E}[\mathtt{E}[\sum_{i=1}^N X_i | N]] \\ &= \sum_{n\in\mathbb{N}} \mathtt{E}[\sum_{i=1}^N X_i | N=n] \Pr\{N=n\} \\ &= \sum_{n\in\mathbb{N}} n\mathtt{E}[X_1] \Pr\{N=n\} \\ &= \mathtt{E}[X_1]\mathtt{E}[N]. \end{aligned}$$

We can also use conditioning in computing variance, since $\mathtt{E}[X^2] = \mathtt{E}[\mathtt{E}[X^2|Y]]$.

Exercises

Exercise 6.1: Answer YES or NO to the following question. A correct answer is worth 5 points, but a wrong answer gets you -3 points. Of course, if you do not answer, you get 0 points. "In a True/False question, you get 5 points for correct answer, 0 points for not attempting the question and -3 points for an incorrect guess. Suppose have NO idea what the answer might be. Should you attempt to answer the question?". ♢

Exercise 6.2: You face a multiple-choice question with 4 possible choices. If you answer the question, you get 6 points if correct and -3 if wrong. If you do not attempt the question, you get -1 point. Should you attempt to answer the question if you have no clue as to what the question is about? ♢

this is not a multiple choice question.

Exercise 6.3: (2 Points)
You are designing a quiz with multiple choice questions. For each question, student must pick one out of 5 possible choices, and only one choice is correct. A correct answer gets a points, and a wrong answer gets $-b$ points. Both a and b are positive numbers. How do you choose a and b such that (i) if a student has no clue, then the expected score is -1 points and (ii) if a student could eliminate one out of the 5 choices, the expected score is 0 points. NOTE: Student are not allowed to ignore questions (so if they don't answer a question, they get $-b$ points). ♢

Exercise 6.4: Compute the expected value of the r.v. C_g in equation (20) for small values of n ($n = 2, 3, 4, 5$). ♢

Exercise 6.5: Simple dice game: you are charged c dollars for rolling a dice, and if your roll has outcome i, you win i dollars. What is the fair value of c? HINT: what is your expected win per roll? ♢

Exercise 6.6: (a) Professor Vegas introduces a game of dice in class (strictly for "object lesson" of course). Anyone in class can play. To play the game, you pay \$12 and roll a pair of dice. If the product of the rolled values on the dice is n, then Professor Vegas pays you \$ n. For instance, if you rolled the numbers 5 and 6 then you make a profit of $\$18 = 30 - 12$. Student Smart would not play, claiming: *the probability of losing money is more than the*

probability of winning money.
(a) What is right and wrong with Student Smart's claim?
(b) Would you play this game? Justify. ♢

Exercise 6.7: One day, Professor Vegas forgot to bring his pair of dice. He stills wants to play the game in the previous exercise. Professor Vegas decides to simulate the dice by tossing a fair coin 6 times. Interpreting heads as 1 and tails as zero, this gives 6 bits which can be viewed as two binary numbers $x = x_2x_1x_0$ and $y = y_2y_1y_0$. So x and y are between 0 and 7. If x or y is either 0 or 7 then the Professor returns your \$12 (the game is off). Otherwise, this is like the dice game in (a). What is the expected profit of this game? ♢

Exercise 6.8: In the previous question, we "simulate" rolling a dice by tossing three fair coins. Unfortunately, if the value of the tosses is 0 or 7, we call off the game. Now, we want to continue tossing coins until we get a value between 1 and 6.
(a) An obvious strategy is this: each time you get 0 or 7, you toss another three coins. This is repeated as many times as needed. What is the expected number of coin tosses to "simulate" a dice roll using this method?
(b) Modify the above strategy to simulate a dice roll with fewer coin tosses. You need to (i) justify that your new strategy simulates a fair dice and (ii) compute the expected number of coin tosses.
(c) Can you show what the optimum strategy is? ♢

Exercise 6.9: In the dice game of the previous exercise, Student Smart decided to do another computation. He sets up a sample space

$$S = \{11, 12, \ldots, 16, 22, 23, \ldots, 26, 33, \ldots, 36, 44, 45, 46, 55, 56, 66\}.$$

So $|S| = 21$. Then he defines the r.v. X where $X(ij) = i \times j$ and computes the expectation of X where using $\Pr(ij) = 1/21$. What is wrong? Can you correct his mistake without changing his choice of sample space? What is the alternative sample space? In what sense is Smart's choice of S is better? ♢

Exercise 6.10: In this game, you begin with a pot of 0 dollars. Before each roll of the dice, you can decide to stop or to continue. If you decide to stop, you keep the pot. If you decide to play, and if you roll any value less than 6, that many dollars are added to the pot. But if you roll a 6, you lose the pot and the game ends. What is your optimal strategy? What if you must pay Z dollars to play? ♢

Exercise 6.11: Let X, Y takes on finitely many values $a_1, \ldots, a_n$. (So they are discrete r.v.'s) Prove if $X \succeq Y$ then $\mathtt{E}[X] \geq \mathtt{E}[Y]$ and equality holds iff $X \sim Y$. ♢

Exercise 6.12: (a) Show that in any graph with n vertices and e edges, there exists a bipartite subgraph with $e/2$ edges. In addition, the bipartite subgraph have $\lfloor n \rfloor$ vertices on one side and $\lceil n \rceil$ of the other. Remark: depending on your approach, you may not be able to fulfill the additional requirement.
(b) Obtain the same result constructively (*i.e.*, give a randomized algorithm). ♢

Exercise 6.13: (Cauchy-Schwartz Inequality) Show that $\mathtt{E}[XY]^2 \leq \mathtt{E}[X^2]\mathtt{E}[Y^2]$ assuming X, Y have finite variances. ♢

Exercise 6.14: (Law of Unconscious Statistician) If X is a discrete r.v. with probability mass function $f_X(u)$, and g is a real function then

$$\mathtt{E}[g(X)] = \sum_{u:f_X(u)>0} g(u) f_X(u).$$

$\diamondsuit$

Exercise 6.15: If $X_1, X_2, \ldots$ are i.i.d. and $N \ge 0$ is an independent r.v. that is integer-valued then $\mathtt{E}[\sum_{i=1}^{N} X_i] = \mathtt{E}[N]\mathtt{E}[X_1]$ and $\mathtt{Var}(\sum_{i=1}^{N} X_i) = \mathtt{E}[N]\mathtt{Var}(X_1) + \mathtt{E}[X]^2\mathtt{Var}(N)$. $\diamondsuit$

Exercise 6.16: Suppose we have a fair game in which you can bet any dollar amount. If you bet \$x, and you win, you receive \$x; and otherwise you lose \$x.
(a) A well-known "gambling technique" is to begin by betting \$1. Each time you lose, you double the amount of the bet (to \$2, \$4, etc). You stop at the first time you win. What is wrong with this scenario?
(b) Suppose you have a limited amount of dollars, and you want to devise a strategy in which the *probability* of your winning is as big as possible. (We are not talking about your "expected win".) How would you achieve this? $\diamondsuit$

Exercise 6.17: [Amer. Math. Monthly] A set consisting of n men and n women are partitioned at random into n disjoint pairs of people. Let X be the number of male-female couples that result. What is the expected value and variance of X? HINT: let X_i be the indicator variable for the event that the ith man is paired with a woman. To compute the variance, first compute $\mathtt{E}[X_i^2]$ and $\mathtt{E}[X_iX_j]$ for $i \ne j$. $\diamondsuit$

Exercise 6.18: [Mean and Variance of a geometric distribution] Let X be the number of coin tosses needed until the first head appears. Assume the probability of coming up heads is p. Use conditional probability (25) to compute $\mathtt{E}[X]$ and $\mathtt{Var}(X)$. HINT: let $Y = 1$ if the first toss is a head, and $Y = 0$ else. $\diamondsuit$

End Exercises

§7. Again, Analysis of QuickSort

In Chapter II, we gave an analysis of Quicksort by solving a recurrence. We now re-visit the analysis, using a probabilistic language.

Assume the input is an array $A[1..n]$ holding n numbers. Recall that Quicksort picks a random $r \in \{1, \ldots, n\}$ and uses the value $A[r]$ to partition the numbers in $A[1..n]$ into those that are (resp.) greater than, and those that are not less than $A[r]$. We then recursively sort these two sets. The main result is this:

Theorem 5 *The expected number of comparisons is* $< 2nH_n$.

We first describe an elegant partition subroutine that requires no extra array storage beyond the input array A:

```
PARTITION($A, i_0, j_0, e$):
Input: Array $A[1..n]$ and $1 \le i_0 < j_0 \le n$
Output: Index $k \in \{i_0, \ldots, j_0\}$ and a rearranged subarray $A[i_0..j_0]$ satisfying
        $A[i_0..k] \le e$ and $A[k+1..j_0] > e$.
    $i \leftarrow i_0$; $j \leftarrow j_0$
    While $(j - i \ge 1)$
        While $(i < j$ and $A[i] \le e)$ $i$++;    ◁ Invariant: $A[i_0..i-1] \le e$
        While $(i < j$ and $A[j] > e)$ $j$−−;    ◁ Invariant: $A[j+1..j_0] > e$
        If $(i < j)$    ◁ either $i = j$ or else $(A[i] > e$ & $A[j] \le e)$

¿   Swap $A[i] \leftrightarrow A[j]$;    $i$++;    $j$−−
    ▷ At this point, $i = j$ or $j + 1$
    If $(j = i)$
        If $(A[i] \le e)$ Return $(i)$
        else Return $(i - 1)$
    else
        Return $(j)$
```

To sort the array $A[1..n]$, we invoke QuickSort$(A, 1, n)$, where QuickSort is the following recursive procedure:

```
QUICKSORT($A, i, j$):
Input: Array $A[1..n]$ and $1 \le i \le j \le n$.
Output: The subarray $A[i..j]$ is sorted in non-decreasing order.
    If $i = j$, Return
    Randomly pick a $r \in \{i, \ldots, j\}$
    Swap $A[r] \leftrightarrow A[i]$
    $k \leftarrow$PARTITION$(A, i, j, A[i])$
    Swap $A[i] \leftrightarrow A[k]$
    If $(i < k)$ QUICKSORT$(A, i, k-1)$
    If $(k < j)$ QUICKSORT$(A, k+1, j)$
```

To simplify the analysis, assume the input numbers in $A[1..n]$ are distinct, and is equal to

$$Z := \{z_1, \ldots, z_n\}$$

where $z_1 < z_2 < \cdots < z_n$. In the following, let $1 \le i < j \le n$, and

$$E_{ij} = \{z_i \text{ is compared to } z_j\}$$

denote the event that that z_i and z_j are compared. Let X_{ij} be the indicator function for E_{ij}. If X is the random variable for the number of comparisons in Quicksort of $A[1..n]$, then we have

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}.$$

The key observation is:

Lemma 6 $\Pr(E_{ij}) = \frac{2}{j-i+1}$

Theorem 5 follows easily from this lemma:

$$\mathtt{E}[X] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathtt{E}[X_{ij}] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\Pr(E_{ij}) < \sum_{i=1}^{n-1} 2H_n < 2nH_n.$$

Before proving the lemma, let us verify it for two special cases:

$$\Pr(E_{i,i+1}) = 1, \qquad \Pr(E_{1,n}) = 2/n. \tag{27}$$

The first case follows from the fact that any correct sorting algorithm must make the comparison $z_i : z_{i+1}$. The second case is also clear because the comparison $z_1 : z_n$ is made iff the random pivot p is chosen to be 1 or n, and this has probability $2/n$.

Proof of Lemma: Consider the event A_{ij} that the initial random pivot lies in the range $[z_i, z_j]$. Clearly $\Pr(A_{ij}) = \frac{j-i+1}{n}$. But we don't really need to know what this is, and we denote it by α. Then $\Pr(\overline{A}_{ij}) = (1-\alpha)$ and

$$\Pr(E_{ij}) = \Pr(E_{ij}|A_{ij})\cdot\alpha + \Pr(E_{ij}|\overline{A}_{ij})\cdot(1-\alpha). \tag{28}$$

It is easy to see that

$$\Pr(E_{ij}|A_{ij}) = \frac{2}{j-i+1}. \tag{29}$$

More interestingly, we also claim

$$\Pr(E_{ij}|\overline{A}_{ij}) = \frac{2}{j-i+1}. \tag{30}$$

Our lemma follows when we plug (29) and (30) into (28).

Let us prove the claim (30). We use induction on the value of $m := n-(j-i+1)$. When $m=0$, this amounts to $\Pr(E_{1,n}) = 2/n$, which we already observed above. If $m > 0$, the event $\overline{A}_{ij}$ can be written as

$$\{p \in \{z_1, \ldots, z_{i-1}\} \cup \{z_{j+1}, \ldots, z_n\}\}$$

where p is the initial pivot element in Quicksort. Each choice of p reduces to an instance of Quicksort of size $n(p)$ where $(j-i+1) \le n(p) < n$. Therefore the value of $n(p)-(j-i+1)$ ranges from 0 to $m-1$. By induction hypothesis, the probability of E_{ij} in each of these instances is $\frac{2}{j-i+1}$. This implies our claim (30) It easily follows that the probability of $\Pr(E_{ij}|\overline{A}_{ij})$ $\frac{2}{j-i+1}$ as

¶28. What is the Sample Space of Quicksort? As we said in the beginning of this chapter – if you distrust your intuition about a probabilistic situation, you should always provide a rigorous construction of the underlying probability space. Let us see this sample space for Quicksort. It will be a generalization of the decision tree model in ¶7.

We introduce the concept of **AND-OR trees**: these are finite rooted trees in which each node is labeled as AND-node or OR-node. The decision tree in ¶7 can be viewed an AND-OR tree with no AND-nodes.

Suppose T and S are an AND-OR trees. We call S an **AND-OR subtree** of T, written $S \subseteq T$, if S is a subtree of T in the usual sense, with the labels of nodes in S are induced from the labels in T. The subtree S is a **sample subtree** of T if it has these additional properties:

- S and T shares a common root.
- If $u \in S$ is an OR-node, then u has degree ≤ 1. The degree of u is 0 iff u is a leaf of T.
- If $u \in S$ is an AND-node, then u has the same degree as u in T.

Let $\Omega(T)$ denote the set of sample subtrees of T, and $\Sigma(T) := 2^{\Omega(T)}$ denote the discrete event space.

E.g., We describe T_n as the AND-OR tree for Quicksort of n elements, $z_1 < z_2 < \cdots < z_n$. In general, a node of T_n is an OR-node iff it's depth is even. Thus the root is an OR-node and its children are AND-nodes. Each OR-node has an integer weight, where the root's weight is n. Each OR-node of weight m is a leaf if $m \le 2$ and otherwise it has degree m. Each AND-node has a pair of integers (m, i) as weight, with $1 \le i \le m$ and m is the weight of its parent. Also, the weights of siblings of an AND-node are distinct. The AND-node of degree (m, i) has degree 2, and its two children have weights $i - 1$ and $m - i$, respectively. This completely determines the AND-OR tree.

Next, we must describe a probability function on $\Sigma(T)$. Call T a **probabilistic AND-OR tree** if the edges u–v (where v is a child of u) of T are assigned probabilities with these properties: if u is an AND-node, then the probability is 1 and if u is an OR-node, then the probabilities of all edges out of u must sum to 1. The **uniform probability** case is when $\Pr(u–v) = 1/d$ whenever u has degree d. Suppose S is a sample subtree of T. We recursively assign a probability $\Pr(u|S)$ to each node $u \in S$:

- If u is a leaf, the probability is 1.
- If u is an AND-node with children v_1, v_1, then $\Pr(u|S) = \Pr(v_1|S)\Pr(v_2|S)$.
- If u is an OR-node of weight d and v is its child, the $\Pr(u|S) = \Pr(u–v)\Pr(v|S)$.

Finally, the probability $\Pr(S)$ is just the probability of its root.

E.g., consider the sample subtree S in Figure 4: It is a sample tree of T_{10}, representing the Quicksort of the elements $z_1, \ldots, z_{10}$. We may check that $\Pr(S) = \frac{1}{10}(\frac{1}{4} \times \frac{1}{5}(\frac{1}{4})) = \frac{1}{800}$.

Lemma 7 *Let T be a probabilistic AND-OR tree, and $\Pr_T : \Omega(T) \to [0, 1]$ denote the probability function defined for sample subtrees of T. Then $\Pr_T$ is a probabilistic distribution on $\Omega(T)$.*

Proof. For each node v of T, the subtree T_v rooted at v is an AND-OR tree and therefore defines a probability distribution $\Pr_v : \Omega(T_v) \to [0, 1]$ on a sample space $\Omega(T_v)$. Recall $\Omega(T_v)$ is the set of all sample subtrees of T_v. If v is a leaf, $\Omega(T_v)$ has just one subtree v and $\Pr_v(v) = 1$. Inductively, suppose the root of T is u and it has children $u_1, \ldots, u_d$. There are two cases: If u is an AND-node, we define the product probability distribution

$$\Pr_u : \otimes_{i=1}^{d} \Omega(T_{u_i}) \to [0, 1].$$

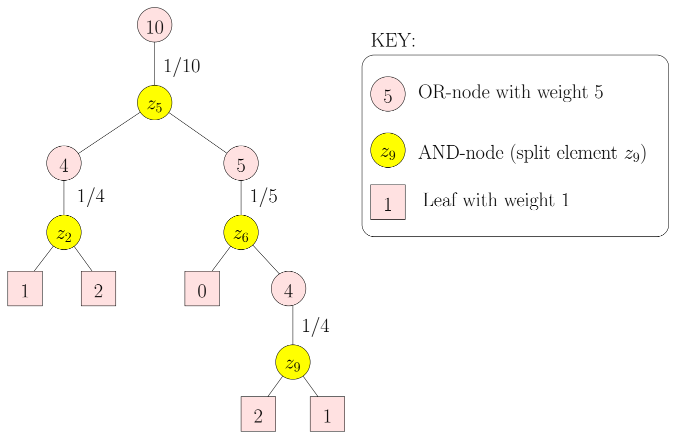


Figure 4: Sample tree for Quicksort for $n = 10$ with probability 1/800.

If u is an OR-node, the probability of all the edges $u{-}u_i$ must sum to one: $\sum_{i=1}^d \Pr(u{-}u_i) = 1$. Then we define the disjoint union probability distribution

$$\Pr_u = \oplus_{i=1}^d \Pr(u{-}u_i) \cdot \Pr_{u_i}.$$

We verify that this definition of $\Pr_u$ agrees with our earlier assignment of probabilities to sample subtrees of T. **Q.E.D.**

Exercises

Exercise 7.1: J. Quick proposes to define the sample space for Quicksort as the set of all permutations on the n input numbers. The probability of each permutation in S_n is $1/n!$. What is wrong with this suggestion? ♢

Exercise 7.2: Determine the size the sample space for Quicksort on n numbers. Use the tree model discribed in the text (the base case of the recursion is when $n \le 2$). ♢

Exercise 7.3: Let us estimate the size $C(n)$ of the sample space for Quicksort of n numbers.
(a) Show that $C(n)$ satisfy the recurrence $C(n) = \sum_{i=1}^n C(i-1)C(n-i)$ for $n \ge 1$ and $C(0) = 1$.
(b) If $G(x) = \sum_{n=0}^\infty C(n)x^n$ is the generating function of the $C(n)$'s, show that $G(x) = (1 - \sqrt{1-4x})/2x$. HINT: What is the connection between $G(x)$ and $G(x)^2$?
(c) Use the Taylor expansion of $\sqrt{1-4x})/2x$. at $x = 0$ to show

$$C(n) = \frac{1}{n+1}\binom{2n}{n}. \tag{31}$$

Then use Stirling's approximation (Chapter II) to give tight bounds on $C(n)$. ♢

End Exercises

§8. Ensemble of Random Variables

Ensembles of random variables appear in two common situations:
(i) An ensemble of i.i.d. r.v.'s.
(ii) An ensemble $\{X_t : t \in T\}$ of r.v.'s where $T \subseteq \mathbb{R}$ is the index set. We think of T as time and X_t as describing the behavior of a stochastic phenomenon evolving over time. Such a family is called a **stochastic process**. Usually, either $T = \mathbb{R}$ (continuous time) or $T = \mathbb{N}$ (discrete time).

We state two fundamental results of probability theory. Both relate to i.i.d. ensembles. Let $X_1, X_2, X_3, \ldots,$ be a countable i.i.d. ensemble of Bernoulli r.v.'s. Let

$$S_n := \sum_{i=1}^n X_i, \quad p := \Pr\{X_1 = 1\}, \quad \sigma := \sqrt{\mathtt{Var}(X_i)}.$$

It is intuitively clear that S_n approaches np as $n \to \infty$.

Theorem 8 (Strong Law of Large Numbers) *For any $\varepsilon > 0$, with probability 1, there are only finitely many sample points in the event*

$$\{|S_n - np| > \varepsilon\}$$

Theorem 9 (Central Limit Theorem)

$$\Pr\left\{\frac{S_n - np}{\sigma\sqrt{n}} \le t\right\} \to \frac{1}{\sqrt{2\pi}} \int_{-\infty}^t e^{-x^2/2} dx$$

as $n \to \infty$.

Alternatively, the Central Limit Theorem says that the distribution of $\frac{S_n - np}{\sigma\sqrt{n}}$ tends to the standard normal distribution (see below) as $n \to \infty$.

¶29. Some probability distributions. The above theorems do not make any assumptions about the underlying distributions of the r.v.'s (therein lies their power). However, certain probability distributions are quite common and it is important to recognize them. Below we list some of them. In each case, we only need to describe the corresponding density functions $f(u)$. In the discrete case, it suffices to specify $f(u)$ at those elementary events u where $f(u) > 0$.

- **Binomial distribution** $B(n, p)$, with parameters $n \ge 1$ and $0 < p < 1$:

$$f(i) := \binom{n}{i} p^i (1-p)^{n-i}, \qquad (i = 0, 1, \ldots, n).$$

Sometimes $f(i)$ is also written $B_i(n, p)$ and corresponds to the probability of i successes out of n Bernoulli trials. In case $n = 1$, this is also called the Bernoulli distribution. If X has such a distribution, then

$$\mathtt{E}[X] = np, \quad \mathtt{Var}(X) = npq$$

where $q = 1 - p$.

- **Geometric distribution** with parameter p, $0 < p < 1$:

$$f(i) := p(1-p)^{i-1} = pq^{i-1}, \qquad (i = 1, 2, \ldots).$$

Thus $f(i)$ may be interpreted as the probability of the first success occurring at the ith Bernoulli trial. If X has such a distribution, then $\mathtt{E}[X] = 1/p$ and $\mathtt{Var}(X) = q/p^2$.

- **Poisson distribution** with parameter $\lambda > 0$:

$$f(i) := e^{-\lambda}\frac{\lambda^i}{i!}, \qquad (i = 0, 1, \ldots).$$

We may view $f(i)$ as the limiting case of $B_i(n, p)$ where $n \to \infty$ and $np = \lambda$. If X has such a distribution, then $\mathtt{E}[X] = \mathtt{Var}(X) = \lambda$.

- **Uniform distribution** over the real interval $[a, b]$:

$$f(u) := \begin{cases} \frac{1}{b-a} & a < u < b \\ 0 & \text{else.} \end{cases}$$

- **Exponential distribution** with parameter $\lambda > 0$:

$$f(u) = \begin{cases} \lambda e^{-\lambda u} & u \ge 0 \\ 0 & \text{else.} \end{cases}$$

- **Normal distribution** with mean μ and variance σ^2:

$$f(u) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left[-\frac{1}{2}\left(\frac{u-\mu}{\sigma}\right)^2\right].$$

In case $\mu = 0$ and $\sigma^2 = 1$, we call this the unit normal distribution.

Exercises

Exercise 8.1: Verify the values of $\mathtt{E}[X]$ and $\mathtt{Var}(X)$ asserted for the various distributions of X.

◇

Exercise 8.2: Show that the density functions $f(u)$ above truly define distribution functions: $f(u) \ge 0$ and $\int_{-\infty}^{\infty} f(u)du = 1$. Determine the distribution function in each case. ◇

End Exercises

§9. Estimates and Inequalities

In probabilistic analysis, we need to estimating probabilities because they are often too intricate to determine exactly. Here are some useful inequalities and estimation techniques.

¶30. Approximating the binomial coefficients. Recall Stirling's approximation in Lecture II.2. Using such bounds, we can show [17] that for $0 < \lambda < 1$ and $\mu = 1 - \lambda$,

$$G(\lambda, n)e^{-\frac{1}{12\lambda n}-\frac{1}{12\mu n}} < \binom{n}{\lambda n} < G(\lambda, n) \tag{32}$$

where

$$G(\lambda, n) := \frac{1}{\sqrt{2\pi\lambda\mu n}}\lambda^{-\lambda n}\mu^{-\mu n}.$$

¶31. Tail of the binomial distribution. The "tail" of the distribution $B(n, p)$ is the following sum

$$\sum_{i=\lambda n}^{n}\binom{n}{i}p^i q^{n-i}.$$

It is easy to see the following inequality:

$$\sum_{i=\lambda n}^{n}\binom{n}{i}p^i q^{n-i} \le \binom{n}{\lambda n}p^{\lambda n}.$$

To see this, note that LHS is the probability of the event $A =$\{There are at least λn successes in n coin tosses\}. For any choice x of λn out of n coin tosses, let B_x be the event that the chosen coin tosses are successes. Then the RHS is the sum of the probability of B_x, over all x. Clearly $A = \cup_x B_x$. Since the RHS may be an over count because the events B_x need not be disjoint, we only have an upper bound. We have the following sharper bound [7]:

$$\sum_{i=\lambda n}^{n}\binom{n}{i}p^i q^{n-i} < \frac{\lambda q}{\lambda - p}\binom{n}{\lambda n}p^{\lambda n}q^{\mu n} \tag{33}$$

where $\lambda > p$ and $q = 1 - p$. This specializes to

$$\sum_{i=\lambda n}^{n}\binom{n}{i} < \frac{\lambda}{2\lambda - 1}\binom{n}{\lambda n}$$

where $\lambda > p = q = 1/2$.

¶32. Markov Inequality. Let X be a non-negative random variable. For any $c > 0$, we have the trivial bound

$$\mathtt{E}[X] \ge c \cdot \Pr\{X \ge c\}$$

since X is non negative. This can be rewritten in a form

$$\Pr\{X \ge c\} \le \frac{\mathtt{E}[X]}{c} \tag{34}$$

or

$$\Pr\{X \ge c \cdot \mathtt{E}[X]\} \le \frac{1}{c}. \tag{35}$$

In particular, with $c = 1$,

$$\Pr\{X \ge 1\} \le \mathtt{E}[X] \tag{36}$$

Any of these inequalities (34, 35, 36) may be called a **Markov inequality** because they imply each other.

Markov inequalities provide *upper bounds* on the probability of events of the form $\{X \ge c\}$ or $\{X \ge c\mathtt{E}[X]\}$, i.e., events defined by *lower bounds* on X. Of course, this is equivalent to *lower bounds* on events defined by *upper bounds* on X. E.g., $\Pr\{X \le c\mathtt{E}[X]\} \ge 1-(1/c) = (c-1)/c$. To get upper bounds on events defined by lower bounds on X, we must use the Chernoff technique (Exercise).

Another proof of (36) uses the **Heaviside function** $H(x)$ that is the 0-1 function given by $H(x) = 1$ if and only if $x > 0$. We have the trivial inequality $H(X-c) \le \frac{X}{c}$ $(c > 0)$. Taking expectations on both sides yields the Markov inequality since $\mathtt{E}[H(X-c)] = \Pr\{X \ge c\}$.

We can even write $\Pr\{X = c\} \le \mathtt{E}[X]/c$. For instance, we infer that the probability that X is twice the expected value is at most $1/2$. Observe that the Markov inequalities becomes trivial when $\mathtt{E}[X] = \infty$, or $c \le \mathtt{E}[X]$ in (34), or $c \le 1$ in (35),

Despite the trivial nature of the Markov inequality, it is the basis for less trivial inequalities to be introduced next.

¶33. Chebyshev Inequality. With any real $c > 0$, we have

$$\Pr\{|X| \ge c\} = \Pr\{X^2 \ge c^2\} \le \frac{\mathtt{E}[X^2]}{c^2} \tag{37}$$

by an application of Markov inequality. Equation (37) is called the Chebyshev inequality. It is also called the Chebyshev-Bienaymé inequality since it originally appeared in a paper of Bienaymé in 1853 [11, p. 73]. In contrast to Markov's inequality, the random variable X is not necessarily non-negative. But in order to reduce it to the Markov case, we must square X or take the absolute value of X. So this bound slightly "less elementary". Another form of this inequality (derived in exactly the same way) is

$$\Pr\{|X - \mathtt{E}[X]| \ge c\} = \Pr\{(X - \mathtt{E}[X])^2 \ge c^2\} \le \frac{\mathtt{Var}(X)}{c^2}. \tag{38}$$

Sometimes $\Pr\{|X - \mathtt{E}[X]| \ge c\}$ is called the **tail probability** of X. By a trivial transformation of parameters, equation (38) can also written as

$$\Pr\{|X - \mathtt{E}[X]| \ge c\sqrt{\mathtt{Var}(X)}\} \le \frac{1}{c^2}. \tag{39}$$

This form is useful in statistics because it bounds the probability of X deviating from its mean by some fraction of the standard deviation, $\sqrt{\mathtt{Var}(X)}$.

Let us give an application of Chebyshev's inequality:

Lemma 10 *Let X be a r.v. with mean $\mathtt{E}[X] = \mu \ge 0$.*
(a) Then

$$\Pr\{X = 0\} \le \frac{\mathtt{Var}(X)}{\mu^2}.$$

(b) Suppose $X = \sum_{i=1}^n X_i$ where the X_i's are pairwise independent Bernoulli r.v.s with $\mathtt{E}[X_i] = p$ (and $q = 1-p$) then

$$\Pr\{X = 0\} \le \frac{q}{np}.$$

Proof. (a) Since $\{X = 0\} \subseteq \{|X - \mu| \geq \mu\}$, we have

$$\Pr\{X = 0\} \leq \Pr\{|X - \mu| \geq \mu\} \leq \frac{\mathtt{Var} X}{\mu^2}$$

by Chebyshev.
(b) It is easy to check that $\mathtt{Var}(X_i) = pq$. Since the X_i's are independent, we have $\mathtt{Var}(X) = npq$. Also $\mathtt{E}[X] = \mu = np$. Plugging into the formula in (a) yields the claimed bound on $\Pr\{X = 0\}$. **Q.E.D.**

Part (b) is useful in reducing the error probability in a certain class of randomized algorithms called "*RP*-algorithms". The outcome of an *RP*-algorithm A may be regarded as a Bernoulli r.v. X which has value 1 or 0. If $X = 1$, then the algorithm has no error. If $X = 0$, then the probability of error is at most p $(0 \leq p < 1)$. We can reduce the error probability in *RP*-algorithms by repeating its computation n times and output 0 iff each of the n repeated computations output 0. Then part(b) bounds the error probability of the iterated computation. For instance, there are RP-algorithms for primality testing (see §XIX.2).

¶34. Jensen's Inequality. A function $f : D \to \mathbb{R}$ defined on a convex domain $D \subseteq \mathbb{R}^d$ is **convex** if the set[6] $\{(x, f(x)) \in D \times \mathbb{R}\}$ is a convex set. Alternatively, if $q = \sum_{i=1}^n \alpha_i p_i$ is a convex combination of any $n \geq 2$ points $p_1, \ldots, p_n \in \mathbb{R}^d$, then

$$f(q) \leq \sum_{i=1}^n \alpha_i f(p_i). \tag{40}$$

Note that q is a convex combination means the α_i's are non-negative and sum to 1, $\sum_{i=1}^n \alpha_i = 1$. This is equivalent to the case where $n = 2$. Examples of convex functions are $f(x) = e^x$ and $f(x) = x^2$. Note that $f(x) = x^r$ is convex iff $r \geq 1$.

If X is a random variable then so is $f(X)$. Jensen's inequality says that if f is a convex function then

$$f(\mathtt{E}[X]) \leq \mathtt{E}[f(X)]. \tag{41}$$

Let us prove this for the case where X is discrete, i.e., it takes on only countably many values x_i of positive probability p_i. Then $\mathtt{E}[X] = \sum_i p_i x_i$ and

$$f(\mathtt{E}[X]) = f(\sum_i p_i x_i) \leq \sum_i p_i f(x_i) = \mathtt{E}[f(X)].$$

For instance, if $r \geq 1$ then $\mathtt{E}[|X|^r] \geq (\mathtt{E}[|X|])^r$.

Exercises

Exercise 9.1: Verify the equation (32). ◊

Exercise 9.2: Let $0 < \lambda < 1$ and $\mu = 1 - \lambda$. Show that

$$\sum_{i=\lambda n}^n \binom{n}{i} p^i q^{n-i} < (p/\lambda)^{\lambda n} (q/\mu)^{\mu n}$$

[6] Known as the epigraph of f.

and

$$\sum_{i=n/2}^{n} \binom{n}{i} p^i q^{n-i} < (4pq)^{n/2}.$$

Remark: If $(pq)/(\lambda\mu) < 1$ if $p \le \lambda \le \frac{1}{2}$. $\diamondsuit$

Exercise 9.3: Let us explore the Markov-type inequalities:
(a) Show that for any $c > 0$, there are non-negative r.v. X and $p \in [0,1]$ such that $\Pr\{X \le c\} = p$.
(b) Describe non-negative random variables X for which Markov's inequality is tight: $\Pr\{X \ge c\} = \mathtt{E}[X]/c$ $\diamondsuit$

Exercise 9.4: The text showed that (36) follows from (34). Show the converse implication. This justifies our calling both "Markov inequalities". $\diamondsuit$

Exercise 9.5: Chebyshev's inequality is the best possible. In particular, show an X such that $\Pr\{|X - \mathtt{E}[X]| > e\} = \mathtt{Var}(X)/e^2$. $\diamondsuit$

§10. Chernoff Bounds

Suppose we wish an upper bound on the probability $\Pr\{X \ge c\}$ where X is an arbitrary r.v.. To apply Markov's inequality, we need to convert X to a non-negative r.v. One way is to use the r.v. X^2, as in the proof of Chebyshev's inequality. The technique of Chernoff converts X to the Markov situation by using

$$\Pr\{X \ge c\} = \Pr\{e^X \ge e^c\}.$$

Since e^X is a non-negative r.v., Markov's inequality bounds the right-hand side by $e^{-1}\mathtt{E}[e^X]$. This yields,

$$\Pr\{X \ge c\} \le e^{-c}\mathtt{E}[e^X]. \tag{42}$$

We can exploit this trick further: for any positive number $t > 0$, we have $\Pr\{X \ge c\} = \Pr\{tX \ge tc\}$, and proceeding as before, we obtain

$$\begin{aligned}\Pr\{X \ge c\} &\le e^{-ct}\mathtt{E}[e^{tX}] \\ &= \mathtt{E}[e^{t(X-c)}].\end{aligned}$$

We then choose t to minimize the right-hand side of this inequality:

Lemma 11 (Chernoff Bound) *For any r.v. X and real c,*

$$\Pr\{X \ge c\} \le m(c). \tag{43}$$

where

$$m(c) = m_X(c) := \inf_{t>0} \mathtt{E}[e^{t(X-c)}]. \tag{44}$$

The name "Chernoff bound" [5] is applied to any inequality derived from (43). We now derive such Chernoff bounds under various scenarios.

Let $X_1, \ldots, X_n$ be independent and

$$S = X_1 + \cdots + X_n.$$

It is easily verified that then $e^{tX_1}, \ldots, e^{tX_n}$ (for any constant t) are also independent. Then equation (24) implies

$$\mathtt{E}[e^{tS}] = \mathtt{E}[\prod_{i=1}^n e^{tX_i}] = \prod_{i=1}^n \mathtt{E}[e^{tX_i}]$$

963 where the last equality follows by independence.

(A) Suppose that, in addition, the $X_1, \ldots, X_n$ are i.i.d., and $m(c)$ is defined as in (44). This shows

$$\begin{array}{rcll} \Pr\{S \geq nc\} & \leq & e^{-nct}\mathtt{E}[e^{tS}] & (\forall\ t > 0) \\ & = & \prod_{i=1}^n e^{-ct}\mathtt{E}[e^{tX_i}] & \\ & = & \prod_{i=1}^n \mathtt{E}[e^{t(X_i-c)}]. & \\ \Pr\{S \geq nc\} & \leq & \prod_{i=1}^n m(c) & (\text{choose } t = \operatorname{argmin}_{t'>0}\left\{\mathtt{E}[e^{t'(X_i-c)}]\right\}). \\ & = & [m(c)]^n. & \end{array}$$

964 This is a generalization of (43).

965 (B) Assume S has the distribution $B(n,p)$. It is not hard to compute that

$$m(c) = \left(\frac{p}{c}\right)^c \left(\frac{1-p}{1-c}\right)^{1-c}. \tag{45}$$

966 Then for any $0 < \varepsilon < 1$, setting $c = (1-\varepsilon)p$,

$$\Pr\{S \geq (1-\varepsilon)np\} \leq \left(\frac{1}{1-\varepsilon}\right)^{(1-\varepsilon)np} \left(\frac{1-p}{1-(1-\varepsilon)p}\right)^{n-(1+\varepsilon)np}. \tag{46}$$

(C) Now suppose the X_i's are [7] independent Bernoulli variables with $\Pr\{X_i = 1\} = p_i$ $(0 \leq p_i \leq 1)$ and $\Pr\{X_i = 0\} = 1 - p_i$ for each i. Then

$$\mathtt{E}[X_i] = p_i, \qquad \mu := \mathtt{E}[S] = \sum_{i=1}^n p_i.$$

967 Fix any $\delta > 0$. Then

$$\begin{array}{rcl} \Pr\{S \geq (1+\delta)\mu\} & \leq & m((1+\delta)\mu) \\ & = & \inf_{t>0} \mathtt{E}[e^{t(X-(1+\delta)\mu)}] \\ & = & \inf_{t>0} \frac{\mathtt{E}[e^{tX}]}{e^{(1+\delta)\mu}}. \end{array}$$

968 We leave as an exercise to optimize the t to obtain

$$\Pr\{S \geq (1+\delta)\mu\} \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu. \tag{47}$$

[7] These are sometimes known as Poisson trials. If all the p_i's are the same, we have the special case of Bernoulli trials.

We can also use this technique to bound the probability that S is at most $(1-\delta)\mu$. This ought to sound surprising, because the event $\{S \le \dots\}$ defined by an upper bound on S. We never see this in the Markov situation, but thanks to the Chernoff transformation, this could be bounded:

$$\Pr\{S \le (1-\delta)\mu\} = \Pr\{-S \ge -(1-\delta)\mu\} = \Pr\left\{e^{-tS} \ge e^{-t(1-\delta)\mu}\right\} \le e^{t(1-\delta)\mu}\mathtt{E}[e^{-tS}].$$

After some further manipulation (Exercise), we obtain

$$\Pr\{S \le (1-\delta)\mu\} \le e^{-\mu\delta^2/2}. \tag{48}$$

¶35. Estimating a Probability and Hoeffding Bound. Consider the natural problem of estimating p $(0 < p < 1)$ where p is the probability that a given coin will show up heads in a toss. The obvious solution is to choose some reasonably large n, toss this coin n times, and estimate p by the ratio h/n where h is the number of times we see heads in the n coin tosses.

This problem is not well-defined yet because we have no criterion for choosing n. But suppose we are given $\delta > 0$ and $\varepsilon < 1$, and we want to determine the $n = n(\delta, \varepsilon)$ such that

$$\Pr\{|p - (h/n)| > \delta\} \le \varepsilon. \tag{49}$$

This seems to be well-defined since, intuitively, when n is large enough, you can achieve the bound (49). This problem may appear to be solvable by the Chernoff bounds in (46), where $S = X_1 + \cdots X_n$ is now interpreted to be h. But a moment's reflection shows the inherent barrier: the p that we are estimating appears on the right hand side of (46). Instead, we want only δ and n to appear. What we need are bounds of the form:

$$\begin{aligned}
\Pr\{S - np > \delta\} &\le \exp(-n\delta^2/2) && (50)\\
\Pr\{S - np < -\delta\} &\le \exp(-n\delta^2/2) && (51)\\
\Pr\{|S - np| > \delta\} &\le 2\exp(-n\delta^2/2) && (52)
\end{aligned}$$

These are called **Hoeffding Bounds**. Before we proving these bounds, let us use them to answer our original problem ((49)) of estimating p:

$$\begin{aligned}
\Pr\{|p - (h/n)| > \delta\} &= \Pr\{|np - S| > n\delta\} && (S = h)\\
&\le 2\exp(-n^3\delta^2/2) && \text{by (52)}\\
&\le \varepsilon.
\end{aligned}$$

Taking logs, see that if n is chosen to exceed $\sqrt[3]{-2\ln(\varepsilon/2)/\delta^2}$, then our goal (49) is achieved. Thus, we will make n coin tosses and if the number of heads is h, we will use $\widetilde{p} = h/n$ as estimate for p. E.g., if we want $\Pr\{|\widetilde{p} - p| < 0.01\}$ to be less than 0.01, it suffices to make 38 coin tosses because $\sqrt[3]{-2\ln(0.005)/0.0001} \approx 37.56$.

Comparing the bounds of Chernoff and Hoeffding, we see that the former bound the relative error in the estimate while the latter bounds absolute error.

We proceed with the proof.

For a survey of Chernoff Bounds, see T. Hagerub and C. Rüb, "A guided tour of Chernoff Bounds", **Information Processing Letters** 33(1990)305–308.

Exercises

Exercise 10.1: Verify the equation (45). ◇

Exercise 10.2: Recall that the Markov inequalities only give lower bounds on the probability of events of the form $\{X \le c\}$. Give an upper bound on $\Pr\{X \le c\}$ by using Chernoff's technique. ◇

Exercise 10.3: Show the following:
i) Bonferroni's inequality,

$$\Pr(AB) \ge \Pr(A) + \Pr(B) - 1.$$

ii) Boole's inequality,

$$\Pr(\cup_{i=1}^n A_i) \le \sum_{i=1}^n \Pr(A_i).$$

(This is trivial, and usually used without acknowledgment.) iii) For all real x, $e^{-x} \ge 1-x$ with equality only if $x = 0$.
iv) $1 + x < e^x < 1 + x + x^2$ which is valid for $|x| < 1$.
◇

Exercise 10.4: Kolmogorov's inequality: let $X_1, \ldots, X_n$ be mutually independent with expectation $\mathtt{E}[X_i] = m_i$ and variance $\mathtt{Var}(X_i) = v_i$. Let $S_i = X_1 + \cdots + X_i$, $M_i = \mathtt{E}[S_i] = m_1 + \cdots + m_i$ and $V_i = \mathtt{Var}(S_i) = v_1 + \cdots + v_i$. Then for any $t > 0$, the probability that the n inequalities

$$|S_i - M_i| < tV_n, \qquad i = 1, \ldots, n,$$

holds simultaneously is at least $1 - t^{-2}$. ◇

Exercise 10.5: (Motwani-Raghavan) (a) Prove the Chernoff bound (47). HINT: Show that $\mathtt{E}[e^{tX_i}] = 1 + p_i(e^t - 1)$ and $1 + x < e^x$ $(x = p_i(e^t - 1))$ implies $\mathtt{E}[e^{tS}] = e^{(e^t-1)\mu}$. Choose $t = \ln(1 + \delta)$ to optimize.
(b) Prove the Chernoff bound (47). HINT: Reduce to the previous situation using $\Pr\{X \le c\} = \Pr\{-X \ge -c\}$. Also, $(1-\delta)^{1-\delta} > e^{-\delta+\delta^2/2}$. ◇

Exercise 10.6: Consider the following problem: Let n be given. Starting with initially empty binary search tree T, I will make n attempts to insert integer keys chosen uniformly from the range $[0, n)$ into T (no duplicate keys allowed). Next I will make n attempts to delete randomly chosen integer keys from the range $[0, n)$ from T. What is the expected size of T after the n attempted deletions? ◇

Exercise 10.7: We want to process a sequence of **requests** on a single (initially empty) list. Each request is either an insertion of a key or the lookup on a key. The probability that any request is an insertion is p, $0 < p < 1$. The cost of an insertion is 1 and the cost of a lookup is m if the current list has m keys. After an insertion, the current list contains one more key.

(a) Compute the expected cost to process a sequence of n requests.
(b) What is the *approximate* expected cost to process the n requests if we use a binary search tree instead? Assume that the cost of insertion, as well as of lookup, is $\log_2(1+m)$ where m is the number of keys in the current tree. NOTE: If L is a random variable (say, representing the length of the current list), assume that $\mathtt{E}[\log_2 L] \approx \log_2 \mathtt{E}[L]$, (*i.e.*, the expected value of the log is approximately the log of the expected value).
(c) Let p be fixed, n varying. Describe a rule for choosing between the two data structures. Assuming $n \gg 1 \gg p$, give some rough estimates (assume $\ln(n!)$ is approximately $n \ln n$ for instance).
(d) Justify the approximation $\mathtt{E}[\log_2 L] \approx \log_2 \mathtt{E}[L]$ as reasonable. $\diamondsuit$

End Exercises

§11. Generating Functions

In this section, we assume that our r.v.'s are discrete with range $\mathbb{N} = \{0, 1, 2, \ldots\}$.

This powerful tool of probabilistic analysis was introduced by Euler (1707-1783). If $a_0, a_1, \ldots,$ is a denumerable sequence of numbers, then its **(ordinary) generating function** is the power series

$$G(t) := a_0 + a_1 t + a_2 t^2 + \cdots = \sum_{i=0}^{\infty} a_i t^i.$$

If $a_i = \Pr\{X = i\}$ for $i \ge 0$, we also call $G(t) = G_X(t)$ the **generating function of** X. We will treat $G(t)$ purely formally, although under certain circumstances, we can view it as defining a real (or complex) function of t. For instance, if $G(t)$ is a generating function of a r.v. X then $\sum_{i\ge 0} a_i = 1$ and the power series converges for all $|t| \le 1$. The power of generating functions comes from the fact that we have a compact packaging of a potentially infinite series, facilitating otherwise messy manipulations. Differentiating (formally),

$$\begin{aligned} G'(t) &= \sum_{i=1}^{\infty} i a_i t^{i-1}, \\ G''(t) &= \sum_{i=2}^{\infty} i(i-1) a_i t^{i-2}. \end{aligned}$$

If $G(t)$ is the generating function of X, then

$$G'(1) = \mathtt{E}[X], \qquad G''(1) = \mathtt{E}[X^2] - \mathtt{E}[X].$$

It is easy to see that if $G_1(t) = \sum_{i\ge 0} a_i t^i$ and $G_2(t) = \sum_{i\ge 0} b_i t^i$ are the generating functions of independent r.v.'s X and Y then

$$G_1(t)G_2(t) = \sum_{i\ge 0} t^i \sum_{j=0}^{i} a_j b_{i-j} = \sum_{i\ge 0} t^i c_i$$

where $c_i = \Pr\{X + Y = i\}$. Thus we have: the product of the generating functions of two independent random variables X and Y is equal to the generating function of their sum $X+Y$.

This can be generalized to any finite number of independent random variables. In particular, if $X_1, \ldots, X_n$ are n independent coin tosses (running example (E1)), then the generating function of X_i is $G_i(t) = q + pt$ where $q := 1 - p$. So the generating function of the r.v. $S_n := X_1 + X_2 + \cdots + X_n$ is

$$(q + pt)^n = \sum_{i=0}^{n} \binom{n}{i} p^i q^{n-i} t^i.$$

Thus, $\Pr\{S_n = i\} = \binom{n}{i} p^i q^{n-i}$ and S_n has the binomial distribution $B(n, p)$.

¶36. Moment generating function. The **moment generating function** of X is defined to be

$$\phi_X(t) := \mathtt{E}[e^{tX}] = \sum_{i \geq 0} a_i e^{it}.$$

This is sometimes more convenient then the ordinary generating function. Differentiating n times, we see $\phi_X^{(n)}(t) = \mathtt{E}[X^n e^{tX}]$ so $\phi^{(n)}(0)$ is the nth moment of X. For instance, if X is $B(n, p)$ distributed then $\phi_X(t) = (pe^t + q)^n$.

Exercises

Exercise 11.1:
(a) What is the generating function of the r.v. X where $\{X = i\}$ is the event that a pair of independent dice roll yields a sum of i $(i = 2, \ldots, 12)$?
(b) What is the generating function of $c_0, c_1, \ldots$ where $c_i = 1$ for all i? Where $c_i = i$ for all i? ♢

Exercise 11.2: What is the generating function for $G(t) = \sum_{i \geq 0} F_i t^i$ where F_i is the Fibonacci numbers $F_0 = 0, F_1 = 1, F_2 = 1, \ldots$? ♢

Exercise 11.3: (a) If $G(t)$ is the generating function of a sequence $a_0, a_1, \ldots$, what is the generating function of the sequence $b_0, b_1, \ldots$ where $b_i = \sum_{j=0}^{i} a_i$?
(b) What is the generating function of the Harmonic numbers $H_n = \sum_{i=1}^{n} 1/i$? ♢

Exercise 11.4: Determine the generating functions of the following probability distributions: (a) binomial, (b) geometric, (c) poisson. ♢

Exercise 11.5: Compute the mean and variance of the binomial distributed, exponential distributed and Poisson distributed r.v.'s using generating functions. ♢

End Exercises

§12. Simple Randomized Algorithms

Probabilistic thinking turns out to be uncannily effective for proving the existence of combinatorial objects. Such proofs can often be converted into randomized algorithms. There exist efficient randomized algorithms for problems that are not known to have efficient deterministic solutions. Even when both deterministic as well as randomized algorithms are available for a problem, the randomized algorithm is usually simpler. This fact may be enough to favor the randomized algorithm in practice. We will produce two kinds of randomized algorithms, called **Las Vegas Algorithms** and **Monte Carlo Algorithms**. The former always produce a correct output, and its running time is only bounded in expectation. In particular, we do not guarantee any worst case time bounds on Las Vegas Algorithms. The latter has a worst case complexity bound, but it may produce an incorrect output with a small probability. Typically, we can convert a Las Vegas Algorithm into a Monte Carlo Algorithm just by cutting off the computation after the expected time has elapsed.

Sometimes, the route to deterministic solution is via a randomized one: after a randomized algorithm has been discovered, we may be able to remove the use of randomness. We will illustrate such "derandomization" techniques. For further reference, see Alon, Spencer and Erdös [3].

We use a toy problem to illustrate probabilistic thinking in algorithm design. Suppose we want to color the edges of K_n, the[8] complete graph on n vertices with one of two colors, either red or blue. For instance, we can color all the edges red. This is not a good choice if our goal is to be "as equitable as possible" in choosing the two colors. Let us take this to mean to have "as few monochromatic triangles as possible". A triangle (i, j, k) is monochromatic if all its edges have the same color. For instance, Figure 5(a,b) show two 2-colorings of K_4, having (a) one and (b) zero monochromatic triangles, respectively. We regard (b) to be more equitable. Figure 5(c,d) illustrates colorings of K_5 and K_6 with 0 and 2 monochromatic triangles. In general, we wish to minimize the number of monochromatic triangles. But this may be too hard. So we want to a "good coloring" that has not too may monochromatic triangles. Of course, instead of monochromatic triangles, we could have ask for no monochromatic quadrilateral, pentagons, etc.

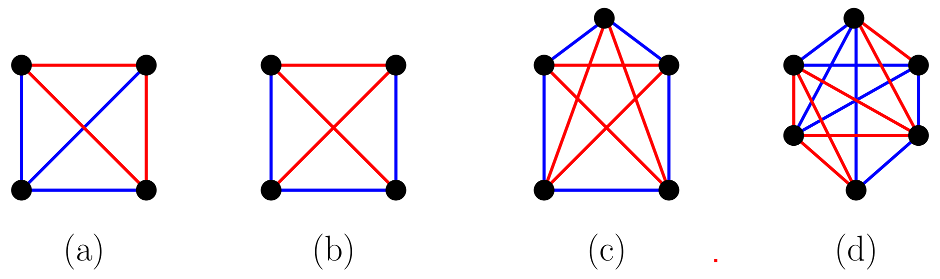


Figure 5: 2-colorings

A *k-coloring* of the edges of a graph $G = (V, E)$ is an assignment $C : E \to \{1, \ldots, k\}$. There are $k^{|E|}$ such colorings. A *random k-coloring* is one that is equal to any of the $k^{|E|}$ colorings with equal probability. Alternatively, a random k-coloring is one that assigns each edge to any of the k colors with probability $1/k$. *When $k = 2$, we assume the 2 colors are blue and red.*

Lemma 12 *Let c and n be positive integers. In the random 2-coloring of the edges of the*

[8]We only consider bigraphs in this section.

complete graph K_n, the expected number of copies of K_c that are monochromatic is

$$\binom{n}{c}2^{1-\binom{c}{2}}$$

Proof. Let X count the number of monochromatic copies of K_c in a random edge 2-coloring of K_n. If V is the vertex set of K_n then

$$X = \sum_U X_U \tag{53}$$

where $U \in \binom{V}{c}$ and X_U is the indicator function of the event that the subgraph of K_n restricted to U is monochromatic. There are $\binom{c}{2}$ edges in U, and we can color U monochromatic in one of two ways: by coloring all the edges blue or all red. Thus the probability of that U is monochromatic in a random coloring is

$$\Pr\{X_U = 1\} = 2^{1-\binom{c}{2}}.$$

But $\mathtt{E}[X_U] = \Pr\{X_U = 1\}$ since X_U is an indicator function. Since there are $\binom{n}{c}$ choices of U, we obtain $\mathtt{E}[X] = \sum_U \mathtt{E}[X_U] = \binom{n}{c}2^{1-\binom{c}{2}}$, by linearity of expectation. **Q.E.D.**

By one of the remarkable properties of expectation (§6), we conclude:

Corollary 13 *There exists an edge 2-coloring of K_n such that the number of monochromatic copies of K_c is at most*

$$\binom{n}{c}2^{1-\binom{c}{2}}$$

In our applications below, we will fix c. For instance, if $c = 3$, this result says that there is a 2-coloring with at most $\frac{1}{4}\binom{n}{3}$ many monochromatic triangles (K_3). If $n = 5$, this $\frac{1}{4}\binom{5}{3} = 2.5$. If $n = 6$, this $\frac{1}{4}\binom{5}{3} = 5$.

¶37. Simple Monte Carlo and Las Vegas Algorithms for good colorings. For any $f > 0$, we say a 2-coloring of K_n is **f-good** if the number of monochromatic K_c is at most $f \cdot \binom{n}{c}2^{1-\binom{c}{2}}$. If $f = 1$, then we simply say the coloring is "good".

E.g., for $c = 3$, and $f = 4/3$, then a $(4/3)$-good coloring has at most $\frac{1}{3}\binom{n}{c}$ monochromatic triangles.

There is a trivial randomized algorithm if we are willing to settle for an f-good coloring for some $f > 1$:

```
RANDOMIZED COLORING ALGORITHM:
Input: $n$ and $f > 1$
Output: a $f$-good 2-coloring of $K_n$
    repeat forever:
    1.  Randomly color the edges of $K_n$ blue or red.
    2.  Count the number $X$ of monochromatic $K_c$'s.
    3.  If $X < f \cdot \binom{n}{c}2^{1-\binom{c}{2}}$, return the random coloring.
```

If the program halts, the random coloring has the desired property. What is the probability of halting? We claim that the probability that a random coloring is not f-good is at most $1/f$, i.e., $\Pr\left\{X \ge f \cdot \binom{n}{c} 2^{1-\binom{c}{2}}\right\} \le 1/f$, by an application of Markov's inequality. From this claim, the probability of repeating the loop *at least once* is at most $1/f$. If $T(n)$ is the time to do the loop once and $\widetilde{T}(n)$ is the expected time of the algorithm, then

Markov: $\Pr\{X \ge t\} \le \mathtt{E}[X]/t$.

$$\widetilde{T}(n) \le T(n) + \frac{1}{f}\widetilde{T}(n) \tag{54}$$

which implies $\widetilde{T}(n) \le \frac{f}{f-1} \cdot T(n)$. But note that

$$T = \mathcal{O}(n^c),$$

since there are $\mathcal{O}(n^c)$ copies of K_c to check. Thus the expected running time is $\widetilde{T} = \mathcal{O}(\frac{f}{f-1} \cdot n^c) = \mathcal{O}(n^c)$. For instance, if $f = 4/3$ then $\widetilde{T}(n) \le 4T(n)$. If $f = 1.001$ the $\widetilde{T}(n) = 1001 \cdot T(n)$.

Las Vegas!

Note that our randomized algorithm has unbounded worst-case running time. Nevertheless, the probability that the algorithm halts is 1. Otherwise, if there is a positive probability $\epsilon > 0$ of not halting, then expected running time becomes unbounded ($\ge \epsilon \times \infty$), which is a contradiction. Alternatively, the probability of not halting is at most $(1/f)^\infty = 0$.

The above algorithm always give the correct answer and has *expected* polynomial running time. But if you are willing to accept a non-zero probability of error, you can get a *worst case* polynomial time. Here is such an algorithm: just return the first random coloring you get! Clearly, it is now worst case $O(n^c)$ but there is a $1/f$ probability that the output is wrong (i.e., more than $1/f$ the expected number of monochromatic K_c's). If you don't like this probability of error, you can reduce it to any $\epsilon > 0$ that you like. E.g., suppose $f = 4/3$. To get an error probability of at most $\epsilon = 0.001$, we repeat the loop 25 times and returning the coloring with the minimum number of monochromatic K_c's among the 25 runs. The probability that this answer is f-bad is less than $(3/4)^{25} < 0.00076 < \epsilon$.

Monte Carlo!

Note that we have assumed $f > 1$ in these algorithms. What if $f = 1$? Let $t = \binom{n}{c} 2^{1-\binom{c}{2}}$, and say[9] that X is "good" if $X \le t$. Thus, the probability that X is not good is $\Pr\{X \ge t+1\}$, which, by Markov's inequality is at most $t/(1+t)$. The expected time for our Las Vegas Algorithm is now $\widetilde{T}(n) \le T(n) + \frac{t}{1+t}\widetilde{T}(n)$ or $\widetilde{T}(n) \le (1+t)T(n)$. For $c = 4$, $\widetilde{T}(n) = O(n^8)$. So you pay a big price for letting f to be 1. We can similarly calculate (Exercise) the cost to achieve a Monte Carlo algorithm with probaility of error ε.

¶38. Sharp Bounds for Monochromatic triangles. For the special case of $c = 3$, we actually have a trivial deterministic algorithm to compute a good 2-coloring of K_n, i.e., a coloring with at most $\frac{1}{4}\binom{n}{3}$ monochromatic triangles (see Corollary 13).

Here is the algorithm: partition the vertices into two sets of sizes $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$, respectively. Color edges within each set red, and edges between the sets blue. We check that this coloring is good.

Can we get substantially better? To answer this, we exploit an elegant formula of Goodman for counting the number of monochromatic triangles in any 2-coloring of K_n. For each node v of K_n, let the number of red (resp., blue) edges incident at v be $d_R(v)$ (resp., $d_B(v)$). Note that $d_R(v) + d_B(v) = n - 1$.

[9]Why can't we define X to be "good" if $X < t$?

Lemma 14 *(Goodman) The number $\Delta(n)$ of monochromatic triangles in a given 2-coloring of K_n is given by the formula*

$$\Delta(n) = \tfrac{1}{2}\left(\sum_v \binom{d_R(v)}{2} + \sum_v \binom{d_B(v)}{2} - \binom{n}{3}\right)$$

For instance, $\Delta(5) \ge 0$ and $\Delta(6) = 2$. We know that both these bounds are sharp.

Proof. $\Delta = \Delta(n)$ counts the contribution from each triangle. We must show a contribution of 1 from monochromatic triangles, and a contribution of 0 from the others:

- Each red triangle will contribute 3 to $\sum_v \binom{d_R(v)}{2}$, contribute 0 to $\sum_v \binom{d_B(v)}{2}$, and 1 to $\binom{n}{3}$. Thus it contributes 1 to Δ. Similarly, each blue triangle contributes 1 to Δ.
- Each triangle with 2 red and 1 blue edge will contribute 1 to $\sum_v \binom{d_R(v)}{2}$, contribute 0 to $\sum_v \binom{d_B(v)}{2}$, and 1 to $\binom{n}{3}$. Thus it contributes 0 to Δ. Similarly, each triangle with 2 blue and 1 red edge will contribute 0 to Δ.

Q.E.D.

This formula for Δ yields a lower bound on the number monochromatic triangle in any 2-coloring:

Lemma 15

$$\Delta(n) \ge \begin{cases} \frac{n(n-1)(n-5)}{24} & \textit{if } n = \textit{odd}, \\ \frac{n(n-2)(n-4)}{24} & \textit{if } n = \textit{even}. \end{cases}$$

Proof. Goodman's formula can be rewritten

$$\Delta = \tfrac{1}{2}\left(\sum_v \left\{\binom{d_R(v)}{2} + \binom{d_B(v)}{2}\right\} - \binom{n}{3}\right)$$

The sum $\binom{d_R(v)}{2} + \binom{d_B(v)}{2}$ is minimized when $d_R(v) = n/2$ (for n even), and $d_R(v) = (n-1)/2$ (for n odd). Since $d_B(v) = n - 1 - d_R(v)$, this implies $d_B(v) = (n-2)/2$ (for n even), and $d_B(v) = (n-1)/2$ (for n odd). Thus for n odd,

$$\begin{aligned} \Delta &\ge \tfrac{1}{2}\left(\sum_v 2\binom{(n-1)/2}{2} - \binom{n}{3}\right) \\ &= \tfrac{1}{2}\left(2n\binom{(n-1)/2}{2} - \binom{n}{3}\right) \\ &= \frac{n(n-1)(n-5)}{24}. \end{aligned}$$

For n even,

$$\begin{aligned} \Delta &\ge \tfrac{1}{2}\left(\sum_v \left\{\binom{n/2}{2} + \binom{(n-2)/2}{2}\right\} - \binom{n}{3}\right) \\ &= \tfrac{1}{2}\left(n\left\{\binom{n/2}{2} + \binom{(n-2)/2}{2}\right\} - \binom{n}{3}\right) \\ &= \frac{n(n-2)(n-4)}{24}. \end{aligned}$$

Q.E.D.

Exercises

Exercise 12.1: (Erdös) In a social gathering with $n \geq 6$ people, there is a groups of three people that are mutual friends or mutual enemies. Assume that every pair are either friends or enemies. $\diamondsuit$

Exercise 12.2: (a) What is the number of monochromatic triangles using the coloring scheme of K_n described in the text: split the vertices into two sets as equal as possible, and color edges within each set red, and the rest blue.
(b) For what values of n is this method optimal? $\diamondsuit$

Exercise 12.3: What is the complexity of a Monte Carlo algorithm to 2-color K_n with at most $\binom{n}{c}2^{1-\binom{c}{2}}$ monochromatic copies of K_c in a graph. We want to ensure that the error probability is at most some given ε. $\diamondsuit$

Exercise 12.4: Find deterministic 2-coloring of K_n that are f-good. The case $c = 3$ was given in the text. What about $c \geq 4$? $\diamondsuit$

Exercise 12.5: Let $C(n)$ be the minimum number of monochromatic triangles in an optimal 2-coloring of K_n. We know that $C(n) = 0$ for $n \leq 5$.
(a) What is the smallest n such that $C(n) > 0$?
(b) What is the smallest n such that $C(n) \geq n$?
(c) What is the largest n you can compute? (Open) $\diamondsuit$

Exercise 12.6: For small values of n, it seems easy to find 2-colorings with fewer than $\frac{1}{4}\binom{n}{3}$ monochromatic triangles. For instance, when $n = 5$, we can color any 5-cycle red and the non-cycle edges blue, then there are no monochromatic triangles (the theorem only gave a bound o 2 monochromatic triangles). Consider the following simple deterministic algorithm to 2-color K_n: pick any T tour of K_n. A tour is an n-cycle that visits every vertex of K_n exactly once and returns to the starting point. Color the n edges in T red, and the rest blue. Prove that this algorithm does not guarantee at most $\frac{1}{4}\binom{n}{3}$ monochrome triangles. For which values of n does this algorithm give fewer than $\frac{1}{4}\binom{n}{3}$ monochrome triangles? $\diamondsuit$

Exercise 12.7: Fixed a bigraph G on n nodes. Show that a random bigraph on $2\log n$ nodes does not occur as an induced subgraph of G. $\diamondsuit$

Exercise 12.8:
(i) What is the role of "3/4" in the first randomized algorithm?
(ii) Give another proof that the probability of halting is 1, by lower bounding the probability of halting at the ith iteration.
(iii) Modify the algorithm into one that has a probability $\varepsilon > 0$ of not finding the desired coloring, and whose worst case running time is $\mathcal{O}_\varepsilon(n^3)$.
(v) Can you improve $T(n)$ to $o(n^3)$? $\diamondsuit$

Exercise 12.9:
(a) Construct a deterministic algorithm to 2-color K_n so that there are at most $2\binom{n/2}{3}$ monochromatic triangles. HINT: use divide and conquer.
(b) Generalize this construction to giving a bound on the number of monochromatic K_c for any constant $c \ge 3$. Compare this bound with the original probabilistic bound. ♢

Exercise 12.10: Let m, n, a, b be positive integers.
(a) Show that in a random 2-coloring of the edges of the complete bipartite graph $K_{m,n}$, the expected number of copies of $K_{a,b}$ that are monochromatic is

$$C(m,n,a,b) = \binom{m}{a}\binom{n}{b}2^{1-ab} + \binom{m}{b}\binom{n}{a}2^{1-ab}.$$

(b) Give a polynomial-time randomized algorithm to 2-color the edges of $K_{m,n}$ so that there are at most $C(m,n,a,b)/2$ copies of monochromatic $K_{2,2}$. What is the expected running time of your algorithm? ♢

End Exercises

§13. Tracking a Random Object, Deterministically

We began with an existence a coloring of K_n that is "good" in the sense of not having many monochromatic triangles. From this, we derive randomized algorithms to find such colorings. What if we wanted deterministic algorithms? We now introduce a method of Raghavan and Spencer to convert existence proofs into efficient, deterministic algorithms. The method amounts to using conditional probability to "track" some good coloring whose existence is known. Hence, the tracking method is often called the **method of conditional probabilities**. The tracking framework is rather like the framework[10] for greedy algorithms: we view our computation as searching for a "good object" by making a sequence of $m \ge 1$ decisions. For example, to find a good coloring of K_n, it amounts to deciding a color for each of the $m = \binom{n}{2}$ edges. Before making the ith decision, we already know the previous $i-1$ decisions. We must make the ith decision in such a way, among the various ways to make the remaining $m-i$ decisions, there exists a good coloring.

Let us return to our problem of 2-coloring of K_n. The edges of K_n are arbitrarily listed as $(e_1, e_2, \ldots, e_m)$ where $m = \binom{n}{2}$. Each i-th decision amounts to choosing a color for e_i. A **partial coloring** χ is an assignment of colors to $e_1, e_2, \ldots, e_i$ where $0 \le i \le m$. When $i = m$, χ is a **complete coloring**. Recall that a complete coloring is good if there are at most $\frac{1}{4}\binom{n}{3}$ monochromatic triangles. We say a partial coloring χ is **good** if the uniformly random complete coloring that extends χ is expected to be good. The tracking method takes a good χ for the first $i-1$ edges, and colors e_i so that the extended partial coloring remains good.

Let us see how to color e_{i+1}, given a partial coloring χ_i of the edges $e_1, \ldots, e_i$. Define $W(\chi_i)$ to be the expected number of monochromatic triangles if the remaining edges $e_{i+1}, e_{i+2}, \ldots, e_m$ are randomly colored red or blue, with equal probability. If χ_0 is the empty partial coloring (no edge is colored), then we already know that

$$W(\chi_0) = \frac{1}{4}\binom{n}{3}.$$

[10]See introduction of Chapter V.

Below, we will show how to compute $W(\chi_{i+1})$ from $W(\chi_i)$. Denote by χ_i^{red} the extension of χ_i in which e_{i+1} is colored red. Similarly for χ_i^{blue}. Note that

$$W(\chi_i) = \frac{W(\chi_i^{red}) + W(\chi_i^{blue})}{2}.$$

We choose to color e_i red iff $W(\chi_i^{red}) \le W(\chi_i^{blue})$. Here then is the deterministic tracking algorithm algorithm to compute a good 2-coloring of K_n.

```
DETERMINISTIC COLORING ALGORITHM:
Input: $K_n$
Output: a good 2-coloring of $K_n$
    1. Let the edges of $K_n$ be $e_1, e_2, \ldots, e_m$ where $m = \binom{n}{2}$.
           Let $\chi_0$ be the empty partial coloring.
    2. For $i = 0$ to $m-1$:
           ▷ Let $\chi_i$ be a partial coloring of $e_1, \ldots, e_i$.
           2.1. Compute $W(\chi_i^{red})$ and $W(\chi_i^{blue})$.
           2.2. Extend $\chi_i$ to $\chi_{i+1}$ by coloring $e_{i+1}$ red iff $W(\chi_i^{red}) \le W(\chi_i^{blue})$.
    3. Return the complete coloring $\chi_m$.
```

randomized greediness!

¶39. Correctness. We claim that the final coloring has at most $\frac{1}{4}\binom{n}{3}$ monochromatic triangles. Clearly,

$$W(\chi_i) = \frac{W(\chi_i^{red}) + W(\chi_i^{blue})}{2} \ge \min\left\{W(\chi_i^{red}), W(\chi_i^{blue})\right\} = W(\chi_{i+1}).$$

It follows that $W(\chi_0) \ge W(\chi_1) \ge \cdots \ge W(\chi_m)$. But χ_m corresponds to a complete coloring of K_n. So $W(\chi_m)$ is equal to the number of monochromatic triangles under χ_m. Thus χ_m is good since

$$W(\chi_m) \le W(\chi_0) = \frac{1}{4}\binom{n}{3}.$$

¶40. Complexity. How can we compute $W(\chi_i)$? By exploiting linearity of expectation! We already saw this technique in the proof of Lemma 12: for each $U \in \binom{V}{3}$, for any triple U of vertices in K_n, let $X_{i,U}$ be the indicator function for the event that U will be monochromatic if the remaining edges $e_{i+1}, \ldots, e_m$ are randomly colored. Then

$$W(\chi_i) = \sum_U \mathtt{E}[X_{i,U}]$$

where the sum ranges over all U. But $\mathtt{E}[X_{i,U}]$ is just the probability that U will become monochromatic:

$$\mathtt{E}[X_{i,U}] = \begin{cases} 2 \cdot 2^{-3} & \text{if no edge of } U \text{ has been colored,} \\ 2^{-3+i} & \text{if } i = 1,2,3 \text{ edges of } U \text{ has been colored with one color,} \\ 0 & \text{the edges of } U \text{ have been given both colors.} \end{cases}$$

Clearly, each $W(\chi_i^{red})$ and $W(\chi_i^{blue})$ can be computed in $\mathcal{O}(n^3)$ time. This leads to an $\mathcal{O}(n^5)$ time algorithm.

But we can improve this algorithm by a slight reorganization of the algorithm. Initially, compute $W(\chi_0)$ in $O(n^3)$ time by summing over all triangles. Then for $i \geq 0$, we can compute $W(\chi_i^{red})$ and $W(\chi_i^{blue})$ in linear time by using the known value of $W(\chi_i)$. This is because each edge affects the status of $n-2$ triangles. Moreover, $W(\chi_{i+1})$ is updated to either $W(\chi_i^{red})$ or $W(\chi_i^{blue})$. Since there are $\mathcal{O}(n^2)$ iterations, the total time for iteration is $\mathcal{O}(n^3)$ time (same as the initialization). So the overall time is $O(n^3)$.

¶41. Framework for deterministic tracking. It is instructive to see the above algorithm in a general framework. Let D be a set of objects and $\chi : D \to \mathbb{R}$ is a real function. For instance, D is the set of 2-colorings of K_n and χ counts the number of monochromatic triangles. In general, think of $\chi(d)$ as computing some statistic of $d \in D$. Call an object d "good" if $\chi(d) \leq k$ (for some fixed k); and "bad" otherwise. Our problem is to find a good object $d \in D$. First we introduce a sequence *tracking variables* $X_1, \ldots, X_m$ in some probability space $(\Omega, 2^\Omega, \Pr)$. Each X_i is an independent r.v. where $\Pr\{X_i = +1\} = \Pr\{X_i = -1\} = \frac{1}{2}$. We want these variables to be "complete" the sense that for any $\epsilon_1, \ldots, \epsilon_m \in \{\pm 1\}$, the event

$$\{X_1 = \epsilon_1, \ldots, X_m = \epsilon_m\}$$

is an elementary event. For instance, we can simply let $\Omega = \{\pm 1\}^m$ and

$$X_i(\epsilon_1, \epsilon_2, \ldots, \epsilon_m) = \epsilon_i$$

for $(\epsilon_1, \ldots, \epsilon_m) \in \Omega$. Given a random object $g : \Omega \to D$, we obtain the random variable $\chi_g : \Omega \to \mathbb{R}$ defined by $\chi_g(\omega) = \chi(g(\omega))$. We say g is good if

$$\mathtt{E}[\chi_g] \leq k \tag{55}$$

This implies that there is *some* sample point ω such that $g(\omega)$ is good. Write W_i as shorthand for $W(\epsilon_1, \ldots, \epsilon_{i-1})$ where $W(\epsilon_1, \ldots, \epsilon_{i-1})$ is the conditional expectation

$$W(\epsilon_1, \ldots, \epsilon_{i-1}) := \mathtt{E}[\chi_g | X_1 = \epsilon_1, \ldots, X_{i-1} = \epsilon_{i-1}].$$

Hence, our assumption (55) above amounts to $W_0 \leq k$. Inductively, suppose we have determined $\epsilon_1, \ldots, \epsilon_{i-1}$ so that

$$W_{i-1} = W(\epsilon_1, \ldots, \epsilon_{i-1}) \leq k.$$

To extend this hypothesis to W_i, observe that

$$\begin{aligned} W_{i-1} &= \frac{W(\epsilon_1, \ldots, \epsilon_{i-1}, +1) + W(\epsilon_1, \ldots, \epsilon_{i-1}, -1)}{2} \\ &\geq \min\{W(\epsilon_1, \ldots, \epsilon_{i-1}, +1), W(\epsilon_1, \ldots, \epsilon_{i-1}, -1)\}. \end{aligned}$$

Hence, if we can efficiently compute the value $W(\epsilon'_1, \ldots, \epsilon'_i)$ for any choice of ϵ'_j's, we may choose ϵ_i so that $W_i \leq W_{i-1}$, thus extending our inductive hypothesis. The hypothesis $W_i \leq k$ implies there is a sample point $\omega \in \{X_1 = \epsilon_1, \ldots, X_i = \epsilon_i\}$ such that $g(\omega)$ is good. In particular, the inequality $W_m \leq k$ implies that $g(\omega)$ is good, where $\omega = (\epsilon_1, \ldots, \epsilon_m)$. Thus, after m steps, we have successfully "tracked" down a good object $g(\omega)$. Note that this is reminiscent of the greedy approach.

The use of the conditional expectations W_i is clearly central to this method. A special case of conditional expectation is when W_i are conditional probabilities. If we cannot efficiently compute the W_i's, some estimates must be used. This will be our next illustration.

Exercises

Exercise 13.1: Generalize the above derandomized algorithm 2-coloring K_n while avoiding too many monochromatic K_c, for any $c \geq 4$. What is the complexity of the algorithm? $\diamondsuit$

Exercise 13.2:
(i) If the edges of K_4 (the complete graph on 4 vertices) are 2-colored so that two edges e, e' have one color and the other 4 edges have a different color, and moreover e, e' have no vertices in common, then we call this coloring of K_4 a "kite". What is the expected

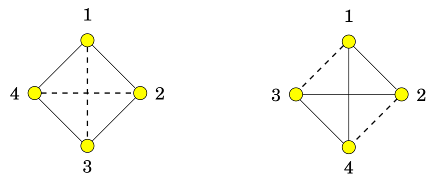


Figure 6: A kite drawn in two ways: edges (1,3) and (2,4) are red, the rest blue.

number of kites in a random 2-coloring of the edges of K_n?
(ii) Devise a deterministic tracking algorithm to compute a 2-coloring which achieves *at least* (not at most) this expected number, and analyze its complexity.
(iii) Can you give a direct $\mathcal{O}(n^2)$ algorithm to do the same? $\diamondsuit$

End Exercises

§14. Derandomization in Small Sample Spaces

The tracking method above is an example of a **derandomization technique**, i.e., a technique to convert randomized algorithms into a deterministic one. We now provide another technique based on small sample spaces. This germ of this idea goes back to Joffe [8].

We continue to use the problem of 2-coloring K_n. Our Monte Carlo algorithm for 2-coloring uniformly and randomly chooses some $\omega \in \Omega$. Each ω corresponds to a specific 2-coloring of K_n. If we want to make this deterministic, a brute force way is to take the 2-coloring χ_ω corresponding to $\omega \in \Omega$, and check whether χ_ω is good, accepting if it is. Termination is certain since we know a good χ_ω exists. Since $|\Omega| = 2^{\binom{n}{2}}$, this brute force method is takes time at least $2^{\binom{n}{2}}$. Can we do better?

Let us step back a moment, to see the probabilistic setting of this brute force algorithm. It amounts to searching through a sample space Ω. The exponential behaviour comes from the fact that $|\Omega|$ is exponential. But if we have a polynomial size sample space $\overline{\Omega}$, then the same brute force search becomes polynomial-time. What do we need of the space $\overline{\Omega}$?

- We endow $\overline{\Omega}$ with the uniform discrete probability space, $(\overline{\Omega}, 2^{\overline{\Omega}}, \Pr)$.
- We define over this probabilistic space an ensemble of $\binom{n}{2}$ indicator variables X_{ij} $(1 \leq i < j \leq n)$. Each $X_{ij} = 1$ iff $(i{-}j) \in \binom{V_n}{2}$ is colored blue.

- The ensemble must be 3-wise independent. This implies that for each triangle $\{i, j, k\} \in \binom{V_n}{3}$, the coloring of any edge in $\{i, j, k\}$ are independent of the other edges.
- Given $\omega \in \overline{\Omega}$, we can determine the value of $X_{ij}(\omega)$ and hence count the number of monochromatic triangles in ω.

With this property, our algorithm goes as follows: for each $\omega \in \overline{\Omega}$, we compute $X_{ij}(\omega)$ $(1 \le i \le j \le n)$ in $O(n^2)$ time. For each triangle (i, j, k), we determine if it is monochromatic, i.e., $X_{ij}(\omega) = X_{jk}(\omega) = X_{k,i}(\omega)$. Checking this for all triangles takes $O(n^3)$ time. If the coloring is good, we output the coloring. Otherwise, we try another ω. This algorithm runs in time $O(n^3|\overline{\Omega}|)$.

A question arise: what is the best coloring we can get from this brute force search through the sample space? Observe that the best coloring obtained this way is not necessarily the optimal 2-coloring.

¶42. Constructing a d-Independent Ensemble over a Small Sample Space. The method of derandomization over a small sample space reduces to the following basic problem:

> (P) Given n and d, we want to construct a "small" sample space Ω that has a d-wise independent ensemble $\{X_1, \ldots, X_n\}$ where each X_i is a Bernoulli r.v. that assumes the value 1 with probability 1/2.

i.e., $X_i \sim B(1, \frac{1}{2})$

"Small" means $|\Omega|$ is polynomial in n and single exponential in d.

The ensemble $\{X_1, \ldots, X_n\}$ required by Problem (P) can be represented by a Boolean matrix X with $|\Omega|$ rows and n columns where

$$X_j(\omega) = X(\omega, j)$$

assuming the rows of X are indexed by sample points $\omega \in \Omega$. Thus each r.v. X_j can be identified with the j-th column of X. The d-wise independence of the X_j amounts to the following: for each set $J \subseteq \{1, \ldots, n\}$ of size a $(1 \le a \le d)$, the submatrix X_J obtained from the a columns $\{X_j : j \in J\}$ has the following property:

$$\text{Each row vector } r \in \{0,1\}^a \text{ appears exactly } |\Omega|/2^a \text{ times among the rows of the matrix } X_J. \tag{56}$$

In particular if $a = 1$, then X_J is a column vector X_j and (56) implies that $\Pr X_j = 1 = 1/2$. Hence our goal is construct such a matrix with $|\Omega| = O(n^{\lfloor d/2 \rfloor})$. The following solution is from Alon, Babai and Itai [1]:

Theorem 16 (Alon, Babai, Itai (1986)) *Given n and d, there exists a sample space Ω of size $O(n^{\lfloor d/2 \rfloor})$ that satisfies the requirement of (P).*

The construction is quite remarkable, requiring rather special properties of $GF(2)$ and $GF(2^k)$ and their interaction. We follow the proof in Alon, Spencer and Erdos [3]. The root of this construction comes from the theory of binary BCH codes which is a family of cyclic error-correcting codes.

Our input parameters are n and d. But we will work with two derived integer parameters k and t where $k \geq \lg(n+1)$ and $t \geq (d-1)/2$. However, for simplicity, we shall assume $k = \lg(n+1)$ (so $n+1$ is a power of 2) and $t = (d-1)/2$ (so d is odd) in the following discussion. It is clear that our construction is easily modified when n and d are not of this form.

We start with the field $GF(2^k)$ and suppose its non-zero elements are

$$x_1, \ldots, x_n \quad (n = 2^k - 1).$$

First form a matrix H

$$H := \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^3 & x_2^3 & \cdots & x_n^3 \\ x_1^5 & x_2^5 & \cdots & x_n^5 \\ \vdots & & \ddots & \\ x_1^{2t-1} & x_2^{2t-1} & \cdots & x_n^{2t-1} \end{bmatrix} \tag{57}$$

We view H in two ways:

(A) First as a $(1+t) \times n$ matrix over $GF(2^k)$. To emphasize this view of H, we will denote it by H^A.

(B) Second, as a $(1+kt) \times n$ matrix over $GF(2)$. To emphasize this view of H as a matrix of *bits*, we denote it by H^B.

For the second view, each $x_j^i \in GF(2^k)$ is represented by a column k-vector of bits. Thus the "row" $(x_1^i, x_2^i, \ldots, x_n^i)$ in the first view becomes a $k \times n$ submatrix of bits. There are t such submatrices (for $i = 1, 3, 5, \ldots, 2t-1$). The first row of 1's is regarded as a single row of n bits. Thus H becomes a Boolean matrix with $1 + kt$ rows in the second view.

In the following derivation, we will regard $GF(2^k)$ as a **k-dimensional algebra** over $GF(2)$. This means:

(i) $GF(2^k)$ is a k-dimensional vector space over the scalar field $GF(2)$. Vector addition in $GF(2^k)$ is just addition in $GF(2^k)$. But notice that we view elements of $GF(2^k)$ as k-vectors, vector addition can be interpreted as the usual component-wise addition of bit vectors mod 2.

(ii) $GF(2^k)$ has a bilinear multiplication, namely: for all $x, y, x', y' \in GF(2^k)$ and $a, b \in GF(2)$, we have

$$(x+y)(x'+y') = xx' + xy' + yx' + yy', \qquad (ax)(by) = (ab)(xy).$$

In our case, this multiplication is associative and commutative ($x(yz) = (xy)z$ and $xy = yx$), because $GF(2^k)$ is actually a field.

We will exploit two special properties of $GF(2)$: when we square the k-algebra expression $ax + by$ ($a, b \in GF(2)$ and $x, y \in GF(2^k)$) we get

$$(ax+by)^2 = (ax)^2 + 2(ax)(by) + (bx)^2 = ax^2 + by^2$$

using the fact that $a^2 = a$ in $GF(2)$, and the cross-term $2(ax)(by)$ of the product vanishes since $2 = 0$ in $GF(2)$. More generally, we have

$$(\sum_i a_i x_i)^2 = \sum_i a_i (x_i)^2. \tag{58}$$

Lemma 17 *Every $2t+1$ columns of H^A (viewing H as a matrix over $GF(2^k)$) is linearly independent over $GF(2)$.*

Proof. Let $J \subseteq \{1, \ldots, n\}$ be any subset of size $|J| = 2t+1$, and let H_J denote the Boolean matrix of size $(1+t) \times (2t+1)$ comprising the columns of H that are selected by J. To show that the columns of H_J are linearly independent over $GF(2)$, suppose $c_1, \ldots, c_{2t+1} \in GF(2)$ and

$$0 = \sum_{j\in J} c_j x_j^i \tag{59}$$

for $i = 0$ and for each odd $i = 1, 3, 5, \ldots, 2t-1$. To show linear independence, we must show that each c_i must be 0. Squaring the equation (59), we get

$$\begin{array}{rcll} 0 & = & (\sum_{j\in J} c_j x_j^i)^2 & \text{(by (59))} \\ & = & \sum_{j\in J} c_j (x_j^i)^2 & \text{(by (58))} \\ & = & \sum_{j\in J} c_j x_j^{2i}. & \end{array}$$

Repeating this k times, we obtain

$$0 = \sum_{j\in J} c_j x_j^{2^k i}.$$

Since every even integer ℓ in the range $1, 2, 3, \ldots, 2t$ has the form $\ell = 2^k i$ for some odd $i = 1, 3, 5, \ldots, 2t-1$, we conclude that the equation (59) actually holds for *all* $i = 0, 1, \ldots, 2t$. But this means that the c_j's is a linear combination of the columns of a Vandermonde matrix of dimension $2t+1$:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & & \ddots & \\ x_1^{2t} & x_2^{2t} & \cdots & x_n^{2t} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{2t} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{60}$$

Since the Vandermonde matrix in non-singular, this can only mean that the c_j's are all zeros.
Q.E.D.

Corollary 18 *Every $2t+1$ columns of H^B (viewing H as a matrix over $GF(2)$) is linearly independent over $GF(2)$.*

Proof. This follows from the fact that the Boolean combination of the $2t+1$ columns of H^A is the same as the Boolean combination of corresponding $2t+1$ columns of H^B.

Precisely: let $J \in \binom{n}{2t+1}$ and $c_j \in GF(2)$ for $j \in J$. Let H_j^A and H_j^B denote the jth column of H^A, H^B. Also let $\overline{x}$ denote the k-bit column vector that represents $x \in GF(2^k)$.

$$\begin{array}{rcll} \sum_{j\in J} c_i H_j^A = 0 & \Leftrightarrow & (\forall i = 0, \ldots, 2t)[\sum_{j\in J} c_j \underline{x_j^i} = 0] & \\ & \Leftrightarrow & (\forall i = 0, \ldots, 2t)[\sum_{j\in J} c_j \overline{x_j^i} = 0] & (*) \\ & \Leftrightarrow & \sum_{j\in J} c_j H_j^B = 0] & \end{array}$$

where (*) is exploits the fact that vector operations on $\overline{x}$ over $GF(2)$ is the correspond k-dimensional algebra vector operations over $GF(2)$. Intuitively, it is because the k-bit vectors $\overline{x}$ are interpreted as polynomials in $GF(2)[X]$ (modulo some irreducible polynomial $I(X)$). **Q.E.D.**

To conclude the proof of Theorem 16, we now construct the Boolean matrix X of dimension $|\Omega| \times n$ required by Problem (P). Let the sample space be

$$\Omega = \left\{1, 2, \ldots, 2^{1+kt}\right\},$$

endowed with the counting probability function. We interpret each $\omega \in \Omega$ as specifying a subset of $\{1, 2, \ldots, 1+kt\}$, and in turn, this specifies a subset of the $1+kt$ rows of the Boolean matrix $H = H^B$. Then the ω-th row of X is just linear combination of the rows of H^B specified by ω. E.g., if ω selects rows $2, 8, 13$ of H^B, then $X_j(\omega) = X(\omega, j) = H_{2,j} + H_{8,j} + H_{13,j}$ (operation in $GF(2)$), where $H_{i,j}$ denote the (i,j)-th bit in H^B.

Let $J \subseteq \{1, \ldots, n\}$ be any set of a $(1 \le a \le 2t+1)$. Let X_J and H_J denote (resp.) the submatrix of X and H formed from the columns specified by J. It remains to prove this claim:

$$\text{For any } r = (r_1, \ldots, r_a) \in \{0,1\}^a, \text{ exactly } 2^{1+kt-a} \text{ rows of } X_J \text{ are equal to } r. \qquad (61)$$

In proof, let $\boldsymbol{b} := (b_1, b_2, \ldots, b_{1+kt})$ be Boolean variables satisfying the equation

$$(b_1, b_2, \ldots, b_{1+kt}) \cdot H_J = (r_1, \ldots, r_a) = r. \qquad (62)$$

Each solution to $\boldsymbol{b}$ in this equation yields a corresponding row in X_J that is equal to r. Thus our claim (61) is equivalent to saying that (62) has exactly 2^{1+kt-a} solutions for $\boldsymbol{b}$. Wlog, assume that the first a rows of H_J are linearly independent. Let us choose arbitrary values for the Boolean variables $b_{a+1}, b_{a+2}, \ldots, b_{1+kt}$ in the equation (62). Then the resulting equation involving only the variables $b_1, \ldots, b_a$ will now have a unique solution. This proves that (62) has exactly 2^{1+kt-a} solutions. This concludes the proof of Theorem 16.

¶43. Remarks Geometry is one of the areas where randomized algorithms has been uniquely successful. Consequently, it is also a realm where derandomization has great success. Instead of the k-wise independent r.v.s over small sample space, we might allow some small bias or almost k-wise independent r.v.s [1] [16, 2].

§15. Arithmetic on Finite Fields.

The sample space construction of Alon, Itai and Babai is typical of this area: we need to reduce many computations to operations on finite fields. Hence we take a detour to review some basic facts about finite fields.

Every finite field has size p^k for some prime p and $k \ge 1$. Up to isomorphism, these fields are uniquely determined by p^k and so they are commonly denoted $GF(p^k)$. The case $k = 1$ is easy: $GF(p)$ is just $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ with arithmetic operations modulo p. For $k \ge 2$, the elements of $GF(p^k)$ may be taken to be $GF(p)^k$, a k vector in $GF(p)$. But arithmetic on $GF(p)^k$ is not the obvious one. The obvious attempt to define a field operations $\circ'$ in $GF(p)^k$ is by componentwise operations in $GF(p)$: $(x_1, \ldots, x_k) \circ' (y_1, \ldots, y_k) = (x_1 \circ y_1, \ldots, x_k \circ y_k)$. where $\circ$ is the corresponding field operation in $GF(p)$. Unfortunately, this does not result in a field. For instance, any element $(x_1, \ldots, x_k)$ which has a 0-component would become a

GF stands for Galois Field

zero-divisor. The correct way to define the field operations in $GF(p)^k$ is to take an irreducible polynomial $I(X)$ of degree k and coefficients in $GF(p)$. Each element of $GF(p)^k$ is interpreted as a polynomial in X of degree $k-1$. Now the field operations are taken to be corresponding field operation on polynomials, but modulo $I(X)$.

Take the simplest case of $p = 2$. Suppose $k = 3$ and let $I(X) = X^3 + X + 1$. Each $(a, b, c) \in GF(2)^3$ is viewed as a polynomial $aX^2 + bX + c$. Addition is performed componentwise: $(a, b, c) + (a', b', c') = (a + a', b + b', c + c')$. But remember $a + a'$ is arithmetic modulo 2: in particular, $1 + 1 = 0$ and $-a = a$. But for multiplication, we may perform the ordinary polynomial multiplication followed by reduction modulo $I(X)$. For instance, X^3 is reduced $X + 1$ (as $X^3 + X + 1 \equiv 0$ implies $X^3 \equiv -(X + 1) \equiv X + 1$. Likewise, $X^4 \equiv X(X^3) \equiv X(X + 1) = X^2 + X$. Thus the product $(a, b, c) \cdot (a', b', c')$ results in a polynomial

$$(aa')X^4 + (ab' + ba')X^3 + (ac' + bb' + ca')X^2 + (bc' + b'c)X + (cc')$$

which reduces to a polynomial of degree ≤ 2,

$$aa'(X^2 + X) + (ab' + a'b)(X + 1) + (ac' + bb' + ca')X^2 + (bc' + b'c)X + cc'$$

E.g., Let us compute the product $(1, 0, 1) \cdot (1, 1, 1)$ using the latter formula. We have

$$(1, 0, 1) \cdot (1, 1, 1) = (1, 1, 0) + (0, 1, 1) + 0 + (0, 1, 0) + (0, 0, 1) = (1, 1, 0).$$

Since $GF(p^k)$ is a field, we can do division. This can be reduced to computing multiplicative inverses. Given $P(X) \equiv 0$, suppose its inverse is $Q(X)$, i.e., $P(X)Q(X) \equiv 1$ modulo $I(X)$. This means

$$P(X)Q(X) + I(X)J(X) = 1$$

for some $J(X)$. We view this equation for polynomials in $GF(2)[X]$. Since $I(X)$ is irreducible and $P(X)$ is not a multiple of $I(X)$, the GCD of $I(X)$ and $P(X)$ is 1. Students may recall that if we compute the GCD of $P(X)$ and $I(X)$ using the **extended Euclidean** algorithm, we can produce the polynomials $Q(X)$ and $J(X)$. Briefly, the standard Euclidean algorithm starting with $a_0 = I(X)$ and $a_1 = P(X)$, we compute a sequence of remainders

$$(a_0, a_1, a_2, \ldots, a_k)$$

where $a_{i+1} = a_{i-1} \,\mathbf{mod}\, a_i$ $(i = 1, \ldots, k)$ and k is determined by the condition $a_{k+1} = 0$. In general, a_k is the GCD of a_0, a_1. In our case, $a_k = 1$ since a_0, a_1 are relatively prime. Let us write

$$a_{i+1} = a_{i-1} - q_i a_i$$

where q_i is the quotient of a_{i-1} divided by a_i. In the extended Euclidean algorithm, we produce two parallel sequences

$$(s_0, s_1, s_2, \ldots, s_k), \quad (t_0, t_1, t_2, \ldots, t_k)$$

where $(s_0, s_1) = (1, 0)$ and $(t_0, t_1) = (0, 1)$, and for $i \ge 1$,

$$s_{i+1} = s_{i-1} - q_i s_i, \quad t_{i+1} = t_{i-1} - q_i t_i.$$

It is easily checked by induction on i that we have

$$a_i = s_i a_0 + t_i a_1.$$

In particular, for $i = k$, this shows $1 = s_k I(X) + t_k P(X)$, i.e., $t_k P(X) \equiv 1(\mathrm{mod}\, I(X))$, or, t_k is the inverse of $P(X)$ modulo $I(X)$.

For instance, dividing $I(X) = X^3 + X + 1$ by $P(X) = X + 1$ over $GF(2)[X]$, we obtain the quotient $Q(X) = X^2 + X$ with remainder of 1. Thus $P(X) \cdot Q(X) = 1 (\bmod I(X))$. Hence the inverse of $P(X)$ is $Q(X) = X^2 + X$. In terms of bit vectors, this is the relation

$$(0, 1, 1) \cdot (1, 1, 0) = (0, 0, 1).$$

In Table 1, we list the polynomials in $GF(2)[X]$ of degrees up to 3. For each polynomial, we either indicate that it is irreducible, or give its factorization. Thus $p_7 = X^2 + X + 1$ is the only irreducible polynomial of degree 2.

Name	Polynomial	Irreducible?
p_0	0	✓
p_1	1	✓
p_2	X	✓
p_3	$X+1$	✓
p_4	X^2	p_0^2
p_5	X^2+1	p_3^2
p_6	X^2+X	p_2p_3
p_7	X^2+X+1	✓
p_8	X^3	p_2p_4
p_9	X^3+1	p_3p_7
p_{10}	X^3+X	p_2p_5
p_{11}	X^3+X^2	p_2p_6
p_{12}	X^3+X+1	✓
p_{13}	X^3+X^2+1	✓
p_{14}	X^3+X^2+X	p_2p_7
p_{16}	X^3+X^2+X+1	p_3p_5

Table 1: Irreducible polynomials up to degree 3 over $GF(2)$

Modulo an irreducible polynomial $I(X)$ of degree k, we can construct a multiplication table for elements $GF(p^k)$, and list their inverses.

Exercises

Exercise 15.1: Give the multiplication and inverse tables for $GF(2^3)$ using $I(X) = p_7$ in Table 1.
(a) Working out these tables by hand. But describe a systematic method of how you do it.
(b) How would you combine these two tables to produce a table for division? ♢

Exercise 15.2: Extend the previous Exercise to constructing a multiplication table and inverse table for $GF(2^n)$ for a general n. Assume that we have an irreducible polynomial $I(X)$ of degree n.
(a) Program the solution using your favorite programming language.
(b) Produce the table for $GF(2^8)$ using the irreducible polynomial $I(X) = X^8 + X^4 + X^3 + X + 1$. ♢

Exercise 15.3: Implement the extended Euclidean algorithm for computing inverses in $GF(2^k)$ modulo an irreducible polynomial $I(X)$ of degree k. ♢

Exercise 15.4: Although division can be reduced to inverses and multiplication, it would be more efficient to do this directly. Adapt the extended GCD algorithm of the previous exercise to produce a direct division algorithm. ♢

Exercise 15.5: Extend Table 1 is a table of complete factorization of polynomials in $GF(2)[X]$ up to degree 3. Explain a systematic way extend this table all polynomials of $\le n$. Using your method, extend the table to $n = 5$. You may program or do this by hand. ♢

Exercise 15.6: Let us consider arithmetic in $GF(2^n)$, viewed as polynomial arithmetic modulo a irreducible polynomial $I(n)$ of degree n. There are many choices for $I(n)$, and algebraically, they are equivalent. But complexity-wise, polynomials that are sparse (few non-zero coefficients) are easier to manipulate. The number of non-zero coefficients is also called the **weight** of the polynomial. Ideally, we like trinomials like p_7, p_{12} of weight 3 in Table 1. There are cases when there are no trinomials, but pentanomials (weight 5). It is an open question whether irreducible polynomials of weights at most 5 for all n.
(a) What is the complexity of reducing a polynomial of degree $2n$ modulo $I(X)$, as a function of w and n.
(b) Among the irreducible polynomials of a given weight, which would be more favorable complexity-wise? ♢

End Exercises

§16. Maximum Satisfiability

In the classic Satisfiability Problem (SAT), we are given a Boolean formula F over some set $\{x_1, \ldots, x_n\}$ of Boolean variables, and we have to decide if F is satisfiable by some assignment of truth values to the x_i's. We may assume F is given as the conjunction of of clauses $F = \bigwedge_{i=1}^m C_i$ $(m \ge 1)$ and each clause C_i is a disjunction of one or more Boolean **literals** (a literal is either a variable x_i or its negation $\overline{x_i}$). Sometimes, it is more convenient to view the formula F as a set of clauses, and a clause as a set literals:

$$F_0 = \{C_1, C_2, C_3, C_4\}, \quad C_1 = \{\overline{x}\}, C_2 = \{x, \overline{y}\}, C_3 = \{y, \overline{z}\}, C_4 = \{x, y, z\} \tag{63}$$

where the Boolean variables are x, y, z. It is easy to see that F_0 is not satisfiable here.

The **Maximum Satisfiability Problem** (MAXSAT) is variant of SAT in which we just want to maximize the number of satisfied clauses in F. It is well-known that SAT is NP-complete, and clearly MAXSAT remains NP-hard. But MAXSAT is now amenable to an approximation interpretation: to find an assignment that satisfies at least a constant fraction of the optimal number of satisfiable clauses. For the unsatisfiable formula F_0 in (63), we see that we can satisfy 3 out of the four clauses by setting x, y, z to 1. Unlike SAT, the presense of clauses of size 1 cannot be automatically "eliminated" in MAXSAT.

If A is an algorithm for MAXSAT, let $A(F)$ denote the number of clauses satisfied by this algorithm on input F. Also, let $A^*(F)$ denote the maximum satisfiable number of clauses for input F (think of A^* as the optimal algorithm). Fix a constant $0 \le \alpha \le 1$. We call A an **α-approximation algorithm** for MAXSAT if $A(F)/A^*(F) \ge \alpha$ for all input F. This definition extends to the case where A is a randomized algorithm. In this case, we replace $A(F)$ by its expected value $\mathtt{E}[A(F)]$.

In this section, we want to illustrate another technique called **random rounding**. The idea is to first give an optimal "fractional solution" in which each Boolean variable are initially assigned a fraction between 0 and 1. We then randomly round these fractional values to 0 or 1; we would expect the rounded result to achieve an provably good objective value. Note that is a randomized algorithm.

¶44. Initial Algorithm A_0. Before we proceed deeper, consider the following simple algorithm A_0: on input $F = \{C_1, \ldots, C_m\}$, we set each Boolean variable to 0 or 1 with equal probability. Let X_i be the indicator r.v. for the event that C_i is satisfied. Then $\mathtt{E}[X_i] = \Pr\{X_i = 1\} = 1 - 2^{-k}$ where k is the number of literals in C_i. Since $k \ge 1$, we get $\mathtt{E}[X_i] \ge 1/2$. The expected number of clauses satisfied is therefore equal to

$$\mathtt{E}[A_0(F)] = \sum_{i=1}^{m} \mathtt{E}[X_i] = m/2.$$

This proves that A_0 is a 1/2-approximation algorithm for MAXSAT (since $A^*(F) \le m$). This is a randomized algorithm.

Using the "tracking framework" in ¶41, we can easily derandomize the simple algorithm into a deterministic 1/2-approximation algorithm (Exercise). Researchers have noticed that an older greedy algorithm of Johnson (1974) for MAXSAT can be interpreted as this de-randomized algorithm. By careful analysis, J. Chen, D. Friesen and H. Zheng (1990) has further shown that the Johnson algorithm is actually a 2/3-approximation.

Of course, the bound $\alpha = 1/2$ can be improved if each clause has more than 1 literal. For instance, if each $|C_i| \ge k$, then our algorithm A_0 gives a $(1 - 2^{-k})$-approximation. For $k = 2$, this is a 3/4-approximation algorithm for MAXSAT. However, we next show that a 3/4-approximation can be achieved without any assumptions on the input.

¶45. The Random Rounding Algorithm A_1. The next idea is to turn MAXSAT into a linear programming problem, from Goemans and Williamson (1994). This transformation is quite standard: we introduce the real variables $c_1, \ldots, c_m$ (c_i corresponding to clause C_i), and $v_1, \ldots, v_n$ (v_j corresponding to variable x_j). Intuitively, v_j is the truth value assigned to x_j, and $c_i = 1$ if C_i is satisfied. However, as a linear programming problem, we can only write write a set of linear constraints on these variables:

$$0 \le v_j \le 1, \quad 0 \le c_i \le 1. \tag{64}$$

Although we intend the v_j's and c_i's to be either 0 or 1, these inequalities only constrain them to lie between 0 and 1. To connect these two set of variables, we look at each clause C_i and write the inequality

$$c_i \le \left(\sum_{j : x_j \in C_i} v_j \right) + \left(\sum_{j : \overline{x_j} \in C_i} (1 - v_j) \right). \tag{65}$$

The objective of the linear programming problem is to maximize the sum

$$c_1 + \cdots + c_m \tag{66}$$

subject to the preceding inequalities. Let us verify that the inequalities (65) ($i = 1, \ldots, m$) would capture the MAXSAT problem correctly if the variables are truly $0-1$. Suppose $v_1, \ldots, v_n$ are $0-1$ values representing an assignment. If C_i is satisfied by this assignment, then at least one of the two sums on the righthand side of (65) is at least 1, and so (65) is clearly satisfied. Moreover, we should set $c_i = 1$ in order to maximize the objective function (66). Conversely, if C_i is not satisfied, then the righthand side of (65) is zero, and we must have $c_i = 0$. This proves that our linear program exactly captures the MAXSAT problem with the proviso *the variables v_j assume integer values.*

Let $\widetilde{A}(F)$ denote the maximum value (66) of our linear program. Since our linear programming solution is maximized over possibly non-integer values of v_j's (thus failing our proviso), we conclude that

$$\widetilde{A}(F) \geq A^*(F). \tag{67}$$

We now describe a **random rounding algorithm**, A_1: on input F, we set up the linear program corresponding to F, and solve it. Let this linear program have solution

$$(v'_1, v'_2, \ldots, v'_n; c'_1, \ldots, c'_m).$$

Thus each v'_j and c'_i is a value between 0 and 1. From (65), we see that

$$c'_i = \min\left\{1, \left(\sum_{j:x_j\in C_i} v'_j\right) + \left(\sum_{j:\overline{x_j}\in C_i} (1-v'_j)\right)\right\}. \tag{68}$$

Since the v'_js lie between 0 and 1, we can interpret them as probabilities. Our algorithm will assign x_j to the value 1 with probability v'_j, and to the value 0 otherwise. This completes our description of A_1.

Our algorithm A_1 calls some linear program solver as a subroutine. Polynomial time algorithms for such solvers are known[11] and are based on the famous interior-point method.

Consider $0-1$ solution computed by A_1: what is the probability that it fails to satisfy a clause C_i? This probability is seen to be

$$p_i = p(C_i) := \left(\prod_{j:x_j\in C_i} (1-v'_j)\right) \cdot \left(\prod_{j:\overline{x_j}\in C_i} v'_j\right). \tag{69}$$

Therefore the probability that C_i is satisfied is $1-p_i$. To lower bound $1-p_i$ over all C_i, we can upper bound p_i instead:

Lemma 19 *If $|C_i| = k$ then*

$$\max_{v'_j} p(C_i) = \begin{cases} (1-k^{-1})^k & \textit{if } C_i \textit{ contains no negated variable,} \\ 1 & \text{else.} \end{cases} \tag{70}$$

The maximization in (70) is subject to v'_j's and c'_i's satisfying (64) and (65).

Proof. First, suppose no negated variables occurs in C_i. Then we have

$$p(C_i) = \prod_{j:x_j\in C_i} (1-v'_j)$$

[11] Interestingly, if the number of variables is bounded by some constant, then linear programming has rather simple randomized algorithms. But in this application, we cannot bound the number of variables by a constant.

subject to the constraint

$$\sum_{j:x_j\in C_i} v'_j \ge c'_i.$$

In the maximization in (70), we can choose $v'_j = 1/k$ if there are k such values of x_j. Note that the constraint is satisfied by this choice, and the corresponding probability is $p(C_i) = (1-1/k)^k$.

Next, suppose at least one negated variable $\overline{x_j}$ occurs in C_i. Corresponding to such a $\overline{x_j} \in C_i$, the term v'_j appears in the formula for $p(C_i)$ (see (69)). In the maximization of (70), we choose $v'_j = 1$ for all such negated variables. The v'_j's corresponding to the non-negated variables $x_j \in C_i$ may be set to 0. This yields $p(C_i) = 1$.

Note that we are not claiming that this maximization for individual C_i's can be simultaneously achieved. **Q.E.D.**

This lemma shows that $p(C_i) \le (1-k^{-1})^k$ is always true. Observe that $e^x > 1+x$ for all non-zero x. Therefore $e^{-1/k} > 1-k^{-1}$ or $e^{-1} > (1-k^{-1})^k \ge p(C^i)$. It follows that the probability of satisfying C_i is least $1-(1-k^{-1})^k > 1-e^{-1} = 0.632121$. Hence the expected number of clauses to be satisfied is at least 0.63 using random rounding.

We have therefore improved upon the method of random assignment.

¶46. Hybrid Algorithm $\max\{A_0, A_1\}$**.** It is instructive to look at the relative probabilities of satisfying a clause using the random assignment A_0 and using random rounding A_1. If the clause has k literals, then the probability of satisfying it is $= 1-2^{-k}$ if we use random assignment, and is $\ge 1-(1-k^{-1})^k$ if we use random rounding. Note that the limit of $1-(1-k^{-1})^k$ as $k\to\infty$ is $1-e^{-1}$. Numerically, $1-e^{-1} > 0.63212$. These probabilities for small k are shown in Table 2:

k	$A_0 : 1-2^{-k}$	$A_1 : 1-(1-k^{-1})^k$
1	0.5	1
2	0.75	0.75
3	0.875	0.704
4	0.9375	0.684
$\vdots$		
∞	1	> 0.6321

Table 2: Probability of satisfying a clause with k literals

Note that, as clause sizes increase, A_0 becomes better while A_1 becomes worse. The cross-over between the two methods occurs relatively quickly: when $k = 2$, both probabilities are 0.75. Thus it is not surprising that when we take the better output from these two algorithms, we ensure that at least 3/4 of the clauses is satisfied:

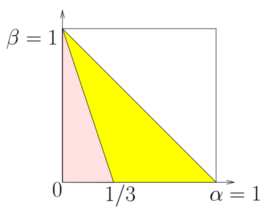
Theorem 20 *Let $A_i(F)$ denote the fraction of clauses in F that is satisfied by algorithm A_i $(i = 0, 1)$. For all formulas F,*

$$\max(A_0(F), A_1(F)) \ge 0.75.$$

Proof. Suppose the fraction $\alpha \geq 0$ of the clauses in F have exactly one literal, and $\beta \geq 0$ of the clauses have exactly two literals. From Table 2, we see that

$$\begin{aligned} A_0(F) &\geq 0.5\alpha + 0.75\beta + 0.875(1-\alpha-\beta) \\ &= 0.875 - 0.375\alpha - 0.125\beta, \\ A_1(F) &\geq \alpha + 0.75\beta + 0.632(1-\alpha-\beta) \\ &= 0.632 + 0.368\alpha + 0.118\beta. \end{aligned}$$

Consider the line $\beta + 3\alpha = 1$ in the (α, β)-plane. This is illustrated in see margin figure: the feasible (α, β) values are shaded either yellow or pink on different sides of this line Then we verify that $A_0(F)$ is identically 0.75 on this line, and $A_1(F) = 0.75 + 0.014\alpha \geq 0.75$. Thus along this line we have verified that $\max\{A_0(F), A_1(F)\} \geq 0.75$. It is easy to see that if $\beta + 3\alpha < 1$ (pink region in the figure), then $A_0(F) > 0.75$. Similarly, we can see that if $\beta + 3\alpha > 1$ (yellow region) then $A_1(F) > 0.75$. We conclude that the $\max\{A_0(F), A_1(F)\} \geq 0.75$ for all α, β. **Q.E.D.**



This proof yields some useful information: you need not run both A_0 and A_1 to get the 0.75 guarantee. Just check whether the fractions α, β of your formula F falls under the case $\beta + 3\alpha \leq 1$ or the case $\beta + 3\alpha > 1$. In the former, call A_0 and otherwise, call A_1.

¶47. Generalization to Weighted Clauses. We could generalize the problem by assigning a weight w_i to the ith clause in a CNF formula $F = \bigwedge_{i=1}^m C_1$. Our goal is now to find an assignment I such that the total weight of all the satisfied clause is maximized.

Here is a simple greedy algorithm from Johnson (1974): let the variables in F be $x_1, \ldots, x_n$. We incrementally construct the I_i that assigns a Boolean value to $x_1, \ldots, x_i$. Let F/I_i denote the set of clauses of F that are still unsatisfied by I_i; also, each clauses C in F, let C/I_i be the clause pruned of the literals whose indices are $\leq i$. E.g., consider F_0 in (63). Let the variables x, y, z be renamed x_1, x_2, x_3, and let I_1 assign x_1 to false. Then $F_0/I_1 = \{C_2, C_3, C_4\}$. Moreover, $C_2/I_1 = \{\{\overline{y}\}\}$, $C_3/I_1 = \{y, \overline{z}\}$, and $C_4/I_1 = \{y, z\}$.

For $i = 0, \ldots, n-1$, suppose that I_i has been computed. We can extend I_{i+1} by setting x_{i+1} to a Boolean value $b \in \{0, 1\}$. Let $I_{i,b}$ denote the extension with $x_{i+1} = b$. For each clause C_j in F, let

$$W(C_j, I_i, b) = \begin{cases} w_j/2^{|C_j/I_i|} & \text{if } (b = 1 \text{and } x_{i+1} \in C_j) \text{ or } (b = 0 \text{and } \overline{x}_{i+1} \in C_j) \\ 0 & \text{else.} \end{cases}$$

Moreover, $W(F, I_i, b)$ is the sum of $W(C_j, I_i, b)$ as C_j range over F/I_i. The Johnson algorithm choose I_{i+1} to be $I_{i,1}$ iff

$$W(F, I_i, 1) \geq W(F, I_i, 0).$$

This algorithm compute an assignment I_n that is at least 1/2 the optimal weight (Exercise). The algorithm can be interpreted as an application of the method of conditional probabilities. J. Chen, D. Friesen and H. Zheng showed that the Johnson algorithm actually achieves 2/3 of the optimum. More sophisticated algorithms by Yannakakis, and also Goemans and Williamson, are able to push this value to 3/4 of optimum.

Exercises

Exercise 16.1: Give an deterministic algorithm to construct an assignment to a formula F which satisfies at least the expected value of a random assignment. Describe the necessary data structures. What is the complexity of your algorithm for a formula with n variables and m clauses? ♢

Exercise 16.2: Fix a formula F over the Boolean variables $x_1, \ldots, x_n$. For $a = (a_1, \ldots, a_n) \in [0,1]^n$, let I_a denote the random assignment where $I_a(x_i) = a_i$, and let E_a be the expected number of clauses that are satisfied in F by I_a. Given a and b, show a strategy to achieve expected value of $(E_a + E_b)/2$. ♢

Exercise 16.3: Recall the Weighted Maximum Satisfiability problem. Prove that the Johnson Algorithm finds an assignment with weight at least $1/2$ of the optimum. ♢

Exercise 16.4: The 4LIN problem consists of n Boolean variables $x_1, x_2, \ldots, x_n$ and m equations where each equation has the form:

$$x_i \oplus x_j \oplus x_k \oplus x_\ell = 1, \qquad 1 \le i < j < k < \ell \le n.$$

Here $\oplus$ denotes the xor operation.

(a) If the variables in some fixed equation are assigned $\{0,1\}$ values uniformly and independently, what is the probability that the equation is satisfied? Justify.

(b) Show that there is an assignment to the n variables that satisfies at least $\frac{m}{2}$ equations.

(c) Now assume that there exists an assignment that satisfies all the equations. Design a polynomial time algorithm to find such an assignment.

♢

End Exercises

§17. Discrepancy Problems

¶48. Discrepancy Random Variables. A typical "discrepancy problem" is this: *given real numbers $a_1, \ldots, a_n$, choose signs $\epsilon_1, \ldots, \epsilon_n \in \{\pm 1\}$ so as to minimize the absolute value of the sum*

$$S = \sum_{i=1}^{n} \epsilon_i a_i.$$

The minimum value of $|S|$ is the *discrepancy* of $(a_1, \ldots, a_n)$. Call[12] a random variable X a *discrepancy r.v.* if the range of X is ± 1; it is *random* if, in addition,

$$\Pr\{X = +1\} = \Pr\{X = -1\} = \tfrac{1}{2}.$$

[12]You may think of this as another name for a Bernoulli r.v..

The *hyperbolic cosine function* $\cosh(x) = (e^x + e^{-x})/2$ arises naturally in discrepancy random variables. If X_i are random discrepancy r.v.'s, then

$$\begin{aligned}
\mathtt{E}[e^{a_iX_i}] &= \frac{e^{a_i}+e^{-a_i}}{2}\\
&= \cosh(a_i)\\
\mathtt{E}[e^{a_1X_1+a_2X_2}] &= \frac{e^{a_1+a_2}+e^{-a_1-a_2}+e^{a_1-a_2}+e^{-a_1+a_2}}{4}\\
&= \frac{\cosh(a_1+a_2)+\cosh(a_1-a_2)}{2}.
\end{aligned}$$

Using the fact that

$$2\cosh(a_1)\cosh(a_2) = \cosh(a_1+a_2)+\cosh(a_1-a_2),$$

we conclude that $\mathtt{E}[e^{a_1X_1+a_2X_2}] = \cosh(a_1)\cosh(a_2)$. In general, with $S = \sum_{i=1}^n a_iX_i$, we get

$$\mathtt{E}[e^S] = \prod_{i=1}^n \cosh(a_i). \tag{71}$$

A useful inequality in this connection is

$$\cosh(x) \le e^{x^2/2}, \qquad x \in \mathbb{R}, \tag{72}$$

with equality iff $x = 0$. This can be easily deduced from the standard power series for e^x.

¶49. A Matrix Discrepancy Problem. Raghavan considered a discrepancy problem in which we need to estimate the conditional probabilities. Let $A = (a_{ij})$ be an $n \times n$ input matrix with $|a_{ij}| \le 1$. Let $\Omega = \{\pm 1\}^n$. Our goal want to find $\overline{\epsilon} = (\epsilon_1, \ldots, \epsilon_n) \in \Omega$, such that for each $i = 1, \ldots, n$,

$$\left|\sum_{j=1}^n \epsilon_j a_{ij}\right| \le \alpha n$$

where

$$\alpha := \sqrt{\frac{2\ln(2n)}{n}}. \tag{73}$$

This choice of α will fall out from the method, so it is best to treat it as a yet-to-be-chosen constant (n is fixed during this derivation). Using the method of deterministic tracking, we introduce random discrepancy r.v.'s $X_1, \ldots, X_n$ such that $X_i(\overline{\epsilon}) = \epsilon_i$ for all i. Also introduce the r.v.'s

$$S_i = \sum_{j=1}^n X_j a_{ij}, \qquad i = 1, \ldots, n.$$

Suppose the values $\epsilon_1, \ldots, \epsilon_\ell$ have been chosen. Consider the event

$$C^\ell := \{X_1 = \epsilon_1, \ldots, X_\ell = \epsilon_\ell\}$$

and the conditional "bad" event

$$B_i^\ell := \{\,|S_i| > \alpha n \,|\, C^\ell\}.$$

To carry out the deterministic tracking method above, we would like to compute the probability $\Pr(B_i^\ell)$. Unfortunately we do not know how to do this efficiently. We therefore replace $\Pr(B_i^\ell)$

by an easy to compute upper estimate, as follows:

$$\begin{aligned}
\Pr(B_i^\ell) &= \Pr\{|S_i| > \alpha n \;\big|C^\ell\} \\
&= \Pr\{e^{\alpha S_i} > e^{\alpha^2 n} \big|C^\ell\} + \Pr\{e^{-\alpha S_i} > e^{\alpha^2 n} \big|C^\ell\} \\
&\le e^{-\alpha^2 n}\mathtt{E}[e^{\alpha S_i} + e^{-\alpha S_i} \;\big|C^\ell] \quad \text{(Markov inequality)} \\
&= e^{-\alpha^2 n} W_i^\ell,
\end{aligned}$$

where the last equation defines W_i^ℓ. Thus we use W_i^ℓ as surrogate for $\Pr(B_i^\ell)$. We do it because we can easily compute W_i^ℓ as follows:

$$\begin{aligned}
W_i^\ell &= \mathtt{E}[e^{\alpha S_i} + e^{-\alpha S_i} | C^\ell] \\
&= \mathtt{E}[\exp\left(\alpha \sum_{j=1}^{\ell} \epsilon_j a_{ij}\right) \exp\left(\alpha \sum_{j=\ell+1}^{n} X_j a_{ij}\right)] + \mathtt{E}[\exp\left(-\alpha \sum_{j=1}^{\ell} \epsilon_j a_{ij}\right) \exp\left(-\alpha \sum_{j=\ell+1}^{n} X_j a_{ij}\right)] \\
&= \exp\left(\alpha \sum_{j=1}^{\ell} \epsilon_j a_{ij}\right) \prod_{j=\ell+1}^{n} \cosh(\alpha a_{ij}) + \exp\left(-\alpha \sum_{j=1}^{\ell} \epsilon_j a_{ij}\right) \prod_{j=\ell+1}^{n} \cosh(\alpha a_{ij}) \\
&= 2\cosh\left(\alpha \sum_{j=1}^{\ell} \epsilon_j a_{ij}\right) \prod_{j=\ell+1}^{n} \cosh(\alpha a_{ij}).
\end{aligned}$$

In particular, for $\ell = 0$,

$$\begin{aligned}
\sum_{i=1}^{n} W_i^0 &= 2\sum_{i=1}^{n} \prod_{j=1}^{n} \cosh(\alpha a_{ij}) \\
&\le 2\sum_{i=1}^{n} \prod_{j=1}^{n} e^{a_{ij}^2 \alpha^2/2} \\
&< 2ne^{n\alpha^2/2},
\end{aligned}$$

where the last inequality is strict since we will assume no row of A is all zero. So the probability that a random choice of $\overline{\epsilon}$ is bad is at most

$$\begin{aligned}
\sum_{i=1}^{n} \Pr(B_i^0) &\le e^{-n\alpha^2} \sum_{i=1}^{n} W_i^0 \\
&< 2ne^{-n\alpha^2/2}.
\end{aligned}$$

We choose α so that the last expression is equal to 1; this is precisely the α in (73). This proves that there is a sample point $\overline{\epsilon}$ where none of the n bad events B_i^0 occur. The problem now is to track down this sample point. In the usual fashion, we show that if $\epsilon_1, \ldots, \epsilon_\ell$ have been chosen then we can choose $\epsilon_{\ell+1}$ such that

$$\sum_{i=1}^{n} W_i^\ell \ge \sum_{i=1}^{n} W_i^{\ell+1}.$$

This can be done as follows: let C_+^ℓ denote the event $C^\ell \cap \{X_{\ell+1} = +1\}$ and similarly let $C_-^\ell := C^\ell \cap \{X_{\ell+1} = -1\}$. Then

$$\begin{aligned}
\sum_{i=1}^n W_i^\ell &= \sum_{i=1}^n \mathtt{E}[e^{\alpha S_i} + e^{-\alpha S_i} | C^\ell] \\
&= \sum_{i=1}^n \frac{\mathtt{E}[e^{\alpha S_i} + e^{-\alpha S_i} | C_\ell^+] + \mathtt{E}[e^{\alpha S_i} + e^{-\alpha S_i} | C_\ell^-]}{2} \\
&\geq \min\{\sum_{i=1}^n \mathtt{E}[e^{\alpha S_i} + e^{-\alpha S_i} | C_\ell^+], \sum_{i=1}^n \mathtt{E}[e^{\alpha S_i} + e^{-\alpha S_i} | C_\ell^+]\} \\
&= \sum_{i=1}^n \mathtt{E}[e^{\alpha S_i} + e^{-\alpha S_i} | C^{\ell+1}]
\end{aligned}$$

provided we choose $\epsilon_{\ell+1}$ to make the last equation hold. But the final expression (74) is $\sum_{i=1}^n W_i^{\ell+1}$. This proves $\sum_{i=1}^n W_i^\ell \geq \sum_{i=1}^n W_i^{\ell+1}$. After n steps, we have

$$\sum_{i=1}^n \Pr(B_i^n) \leq e^{-\alpha^2 n} \sum_{i=1}^n W_i^n < 1. \tag{74}$$

But B_i^n is the probability that $|S_i| > \alpha n$, conditioned on the event C^n. As $C^n = \{(\epsilon_1, \ldots, \epsilon_n)\}$ is an elementary event, the probability of any event conditioned on C^n is either 0 or 1. Thus equation (74) implies that $\Pr(B_i^n) = 0$ for all i. Hence C^n is a solution to the discrepancy problem.

We remark that computing W_i is considered easy because the exponential function e^x can be computed relatively efficiently to any desired degree of accuracy (see Exercise).

Exercises

Exercise 17.1:
(i) Verify equation (72).
(ii) Show the bound $\Pr\{X_1 + \cdots + X_n > a\} < e^{-a^2/2n}$, where X_i are random discrepancy r.v.'s and $a > 0$. ♢

Exercise 17.2: What is the bit-complexity of Raghavan's algorithm? Assume that e^x (for x in any fixed interval $[a, b]$) can be computed to n-bits of relative precision in $\mathcal{O}(M(n) \log n)$ time where $M(n)$ is the complexity of binary number multiplication. The inputs numbers a_{ij} are in floating point notation, *i.e.*, a_{ij} is represented as a pair (e_{ij}, f_{ij}) of binary integers so that

$$a_{ij} = 2^{e_{ij}} f_{ij}$$

and e_{ij}, f_{ij} are at most m-bit numbers. ♢

End Exercises

§18. Random Search Trees

Recall the concept of binary search trees from Lecture III. This lecture focuses on a class of random binary search trees called **random treaps**. Basically, treaps are search trees whose shape is determined by the assignment of **priorities** to each key. If the priorities are chosen randomly, the result is a random treap with expected height of $\Theta(\log n)$. Why is expected $\Theta(\log n)$ height interesting when worst case $\Theta(\log n)$ is achievable? One reason is that randomized algorithms are usually simpler to implement. Roughly speaking, they do not have to "work too hard" to achieve balance, but can rely on randomness as an ally.

The analysis of random binary search trees has a long history. The analysis of binary search trees under random insertions only was known for a long time. It had been an open problem to analyze the expected behavior of binary search trees under insertions *and deletions*. A moment reflection will indicate the difficulty of formulating such a model. As an indication of the state of affairs, a paper *A trivial algorithm whose analysis isn't* by Jonassen and Knuth [9] analyzed the random insertion and deletion of a binary tree containing no more than 3 items at any time. The currently accepted probabilistic model for random search trees is due to Aragon and Seidel [4] who introduced the treap data-structure.[13] Prior to the treap solution, Pugh [18] introduced a simple but important data structure called **skip list** whose analysis is similar to treaps. The real breakthrough of the treap solution lies in its introduction of a new model for randomized search trees, thus changing [14] the ground rules for analyzing random trees.

¶50. Notations. We write $[i..j]$ for the set $\{i, i+1, \ldots, j-1, j\}$ where $i \le j$ are integers. Usually, the set of keys is $[1..n]$. If u is an item, we write u.`key` and u.`priority` for the key and priorities associated with u. Since we normally confuse an item with the node it is stored in, we may also write u.`parent` or u.`leftChild`. By convention, u is the root iff u.`parent` $= u$.

§19. Skip Lists

It is instructive to first look at the skip list data structure. Pugh [18] gave experimental evidence that its performance is comparable to non-recursive AVL trees algorithms, and superior to splay trees (the experiments use 2^{16} integer keys). Note that AVL trees are harder to implement than splay trees or skip lists.

Suppose we want to maintain an ordered list L_0 of keys $x_1 < x_2 < \cdots < x_n$. We shall construct a **hierarchy** of sublists

$$L_0 \supseteq L_1 \supseteq \cdots \supseteq L_{m-1} \supseteq L_m$$

where L_m is only empty list in this hierarchy. This is illustrated in Figure 7 where $m = 5$.

Each L_{i+1} is a *random sample* of L_i in the following sense: each key of L_i is put into L_{i+1} with probability 1/2. The hierarchy stops the first time the list becomes empty. In storing these lists, we shall add an artificial key $-\infty$ at the head of each list. Let L'_i be the list L_i augmented with $-\infty$. It is sufficient to store the lists L'_i as a linked list with $-\infty$ as the head (a singly-linked list is sufficient). In addition, corresponding keys in two consecutive levels shares a two-way link (called the up- and down-links, respectively.

Clearly, the expected length of L_i is $\mathtt{E}[|L_i|] = n2^{-i}$. Let us compute the expected height $\mathtt{E}[m]$. The probability of any key of L_0 appearing in level $i \ge 0$ is $1/2^i$. Now $m \ge i+1$ iff some

[13] The name "treap" comes from the fact that these are binary search trees with the heap property. It was originally coined by McCreight for a different data structure.

[14] Alexander the Great's solution for the Gordan Knot is similar.

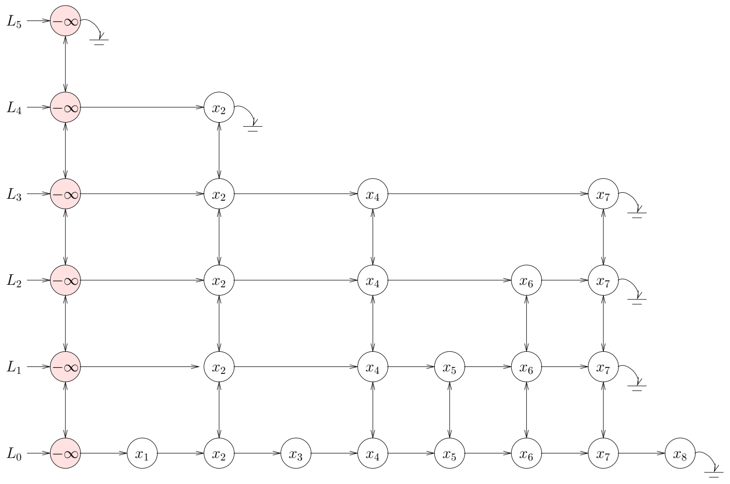


Figure 7: A skip list on 8 keys.

key x_j appears level i. Since there are n choices for j, we have

$$\Pr\{m \ge i+1\} \le \frac{n}{2^i}.$$

Hence

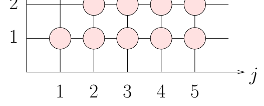
$$\begin{array}{rcll}
\mathtt{E}[m] &=& \sum_{j\ge 1} j\cdot \Pr\{m=j\} & \\
&=& \sum_{i\ge 1}\Pr\{m\ge i\} & \text{(standard trick)}\\
&=& \sum_{i\le\lceil \lg n\rceil}\Pr\{m\ge i\} + \sum_{j>\lceil \lg n\rceil}\Pr\{m\ge j\} & \text{(another one!)}\\
&\le& \lceil \lg n\rceil + \sum_{j>\lceil \lg n\rceil}\frac{n}{2^{j-1}} & \\
&\le& \lceil \lg n\rceil + 2. &
\end{array}$$

Exchanging order of summation. The above "standard trick" of equating $\sum_{j\ge 1} j\cdot\Pr\{m=j\}$ to $\sum_{i\ge 1}\Pr\{m\ge i\}$ is well-known in combinatorics. It is amounts to exchanging the order of summation in a double summation, i.e., replacing $S_1=\sum_{j\in J}\sum_{i\in I_j} A_{ij}$ by $S_2=\sum_{i\in I}\sum_{j\in J_i} A_{ij}$. Here, J, I_j, I, J_i are suitable index sets. The particular exchange we are witnessing is visualized by the margin figure, where the (i,j)th circle in the infinite matrix stores the values A_{ij} which we are trying to sum. The index sets in this case are:

$$J = I = \{1,2,3,\ldots\}, I_j = \{1,2,\ldots,j\}, J_i = \{i, i+1, i+2, \ldots\}.$$

1641 In S_1, we are first sum the A_{ij}'s by columns (indexed by increasing j) and in each column, we sum by rows (indexed by increasing i). In S_2, reverse this order (first sum by rows, then by columns). Evidently, $S_1 = S_2$ (assuming absolute convergence of the sums). To apply the $S_1 = S_2$ identity to our probabilistic example, we must rewrite $\mathtt{E}[m]$ as a double summation:

$$\begin{array}{rcl}
\mathtt{E}[m] = \sum_{j\ge 1} j\cdot\Pr\{m=j\} &=& \sum_{j\ge 1}\sum_{i=1}^{j}\Pr\{m=j\}\\
&=& \sum_{i\ge 1}\sum_{j\ge i}\Pr\{m=j\}\\
&=& \sum_{i\ge 1}\Pr\{m\ge i\}.
\end{array}$$



Have we really achieved anything special with skip lists? You may think: why not just **deterministically** omit every other item in L_i to form L_{i+1}? Then $m \le \lceil \lg n \rceil$ with less fuss! The reason why the probabilistic solution is superior is because *the analysis will hold up even in the presence of insertion or deletion of arbitrary items.* The deterministic solution may look very bad for particular sequence of deletions (how?) The critical idea is that the probabilistic behavior of each item x is *independent* of the other items in the list: its presence in each sublists is determined by its own sequence of coin tosses!

why we do this randomly...

¶51. Analysis of Lookup. Our algorithm to lookup a key k in a skip list returns the largest key k_0 in L_0 such that $k_0 \le k$. If k is less than the smallest key x_1 in L_0, we return the special value $k_0 = -\infty$.

You may also be interested in finding the smallest key k_1 in L_0 such that $k_1 \ge k$. But k_1 is either k_0 or the successor of k_0 in L_0. If k_0 has no successor, then we imagine "$+\infty$" as its successor.

In general, define k_i $(i = 0, \ldots, m)$ to be the largest key in level L'_i that is less than or equal to k. So $k_m = -\infty$. Given key k_{i+1} in level $i+1$, we can find k_i by following a down-link and then "scanning forward" in search of k_i. The search ends after the scan in list L_0: we end up with the element k_0. Let us call the sequence of nodes visited from k_m to k_0 the **scan-path**. For example, if $k = x_5$ in Figure 7, then the scan path would be:

$$k_5 = -\infty, k_4 = x_2, k_3 = x_4, k_2 = x_4, k_1 = x_5, k_0 = x_5.$$

[SUGGEST: modify Figure 7 to be more interesting for the concept of "scan-path"!!!] Let s be the random variable denoting the length of the scan-path. If s_i is the number of nodes of the scan-path that that lies in L'_i, then we have

$$s = \sum_{i=0}^{m} s_i.$$

The expected cost of looking up key k is $\Theta(\mathtt{E}[s])$.

Lemma 21 $\mathtt{E}[s] < 2\lceil \lg n \rceil + 4$.

Before presenting the proof, it is instructive to give a wrong argument: since $\mathtt{E}[m] \le \ln n + 3$, and $\mathtt{E}[s_i] < 2$ (why?), we have $\mathtt{E}[s] \le 2(\ln n + 2)$. This would be correct if each s_i is i.i.d. and also independent of m (see §VII.5). Unfortunately, s_i is not independent of m (e.g., $s_i = 0$ iff $m < i$).

Proof of lemma 21. Following Pugh, we introduce the random variable

$$s(\ell) = \sum_{i=0}^{\ell-1} s_i$$

where $\ell \ge 0$ is any constant (the threshold level). By definition, $s(0) = 0$. We note that

$$s \le s(\ell) + |L_\ell| + (m - \ell + 1).$$

To see this, note that the edges in the scan-path at levels ℓ and above involve at most $|L_\ell|$ horizontal edges, and at most $m - \ell$ vertical (down-link) edges.

Choose $\ell = \lceil \lg n \rceil$. Then $\mathtt{E}[|L_\ell|] = n/2^\ell \le 1$. Also $\mathtt{E}[m - \ell + 1] \le 3$ since $\mathtt{E}[m] \le \lceil \lg n \rceil + 2$. The next lemma will imply $\mathtt{E}[s(\ell)] \le 2\lceil \lg n \rceil$. Combining these bounds yields lemma 21.

¶52. A Random Walk. To bound $s(\ell)$, we introduce a random walk on the lattice $\mathbb{N} \times \mathbb{N}$.

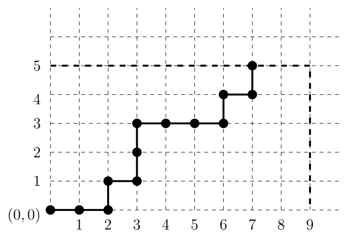


Figure 8: Random walk with $X_{9,5} = 12$.

Let u_t denote the position at time $t \in \mathbb{N}$. We begin with $u_0 = (0,0)$, and we terminate at time t if $u_t = (x,y)$ with either $x = n$ or $y = \ell$. At each discrete time step, we can move from the current position (x,y) to $(x, y+1)$ with probability p $(0 < p < 1)$ or to $(x+1, y)$ with probability $q = 1 - p$. So p denotes the probability of "promotion" to the next level. Let $X_{n,\ell}$ denoting the number of steps in this random walk at termination. For $p = 1/2$, we obtain the following connection with $s(\ell)$:

$$\mathtt{E}[X_{n,\ell}] \ge \mathtt{E}[s(\ell)]. \tag{75}$$

To see this, imagine walking along the scan-path of a lookup, but in the reverse direction. We start from key k_0. At each step, we get to move up to the next level with probability p or to scan backwards one step with probability $q = 1 - p$. There is a difference, however, between a horizontal step in our random walk and a backwards scan of the scan-path. In the random walk, each step is unit length, but the backwards scan may take a step of larger lengths with some non-zero probability. But this only means that $X_{n,\ell} \succeq s(\ell)$ (denoting stochastic domination, §VII.3). Hence (75) is an inequality instead of an equality. For example, in the skip list of figure 7, if $k_0 = x_8$ then $s(\ell) = s(4) = 7$ while $X_{n,\ell} = X_{8,4} = 10$.

Lemma 22 *For all $\ell \ge 0$, $\mathtt{E}[X_{n,\ell}] \le \ell/p$.*

Proof. Note that this bound removes the dependence on n. Clearly $\mathtt{E}[X_{n,0}] = 0$, so assume $\ell \ge 1$. Consider what happens after the first step: with probability p, the random walk moves to the next level, and the number of steps in rest of the random walk is distributed as $X_{n,\ell-1}$; similarly, with probability $1-p$, the random walk remain in the same level and the number of steps in rest of the random walk is distributed as $X_{n-1,\ell}$. This gives the recurrence

$$\mathtt{E}[X_{n,\ell}] = 1 + p \cdot \mathtt{E}[X_{n,\ell-1}] + q \cdot \mathtt{E}[X_{n-1,\ell}].$$

Since $\mathtt{E}[X_{n-1,\ell}] \le \mathtt{E}[X_{n,\ell}]$, we have

$$\mathtt{E}[X_{n,\ell}] \le 1 + p \cdot \mathtt{E}[X_{n,\ell-1}] + q \cdot \mathtt{E}[X_{n,\ell}].$$

This simplifies to

$$\mathtt{E}[X_{n,\ell}] \le \frac{1}{p} + \mathtt{E}[X_{n,\ell-1}].$$

This implies $\mathtt{E}[X_{n,\ell}] \le \frac{\ell}{p}$ since $\mathtt{E}[X_{n,0}] = 0$. **Q.E.D.**

If $\ell = \lceil \lg n \rceil$ and $p = 1/2$ then

$$\mathtt{E}[s(\ell)] \le \mathtt{E}[X_{n,\ell}] \le 2 \lceil \lg n \rceil ,$$

as noted in the proof of lemma 21.

We leave as an exercise to the reader to specify, and to analyze, algorithms for insertion and deletion in skip lists.

Exercises

Exercise 19.1:
(i) The expected space for a skip list on n keys is $\mathcal{O}(n)$.
(ii) Describe an algorithm for insertion and analyze its expected cost.
(iii) Describe an algorithm for deletion and analyze its expected cost. ♢

Exercise 19.2: The above (abstract) description of skip lists allows the possibility that $L_{i+1} = L_i$. So, the height m can be arbitrarily large.
(i) Pugh suggests some á priori maximum bound on the height (say, $m \le k \lg n$) for some small constant k. Discuss the modifications needed to the algorithms and analysis in this case.
(ii) Consider another simple method to bound m: if $L_{i+1} = L_i$ should happen, we simply omit the last key in L_i from L_{i+1}. This implies $m \le n$. Re-work the above algorithms and complexity analysis for this approach. ♢

Exercise 19.3: Determine the exact formula for $\mathtt{E}[X_{n,\ell}]$. ♢

Exercise 19.4: We have described skip lists in which each key is promoted to the next level with probability $p = 1/2$. Give reasons for choosing some other $p \ne 1/2$. ♢

§20. Treaps

In our treatment of treaps, we assume that each item is associated with a *priority* as well as the usual *key*. A *treap* T on a set of items is a binary search tree with respect to the keys of items, and a max-heap with respect to the priorities of items. Recall that a tree is a max-heap with respect to the priorities if the priority at any node is greater than the priorities in its descendants.

Suppose the set of keys and set of priorities are both $[1..n]$. Let the priority of key k be denoted $\sigma(k)$. We assume σ is a permutation of $[1..n]$. We explicitly describe σ by listing the keys, in order of decreasing priority:

$$\sigma = (\sigma^{-1}(n), \sigma^{-1}(n-1), \ldots, \sigma^{-1}(1)). \tag{76}$$

We sometimes call σ a *list* in reference to this representation. In general, if the range of σ is not in $[1..n]$, we may define analogues of (76): a list of all the keys is called a **σ-list** if the priorities of the keys in the list are non-increasing. So σ-lists are unique if σ assigns unique keys.

Example. Let $\sigma = (5, 2, 1, 7, 3, 6, 4)$. So the key 5 has highest priority of 7 and key 4 has priority 1. The corresponding treap is given by figure 9 where the key values are written in the nodes.

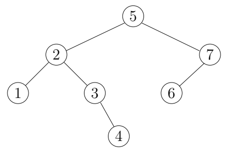


Figure 9: Treap $T_7(\sigma)$ where $\sigma = (5, 2, 1, 7, 3, 6, 4)$.

■

There is no particular relation between the key and the priority of an item. Informally, a *random treap* is a treap whose items are assigned priorities in some random manner.

Fact 1 *For any set S of items, there is a treap whose node set is S. If the keys and priorities are unique for each item, the treap is unique.*

*Proof.*The proof is constructive: given S, pick the item u_0 with highest priority to be the root of a treap. The remaining items are put into two subsets that make up the left and right subtrees, respectively. The construction proceeds inductively. If keys and priorities are unique then u_0 and the partition into two subsets are unique. **Q.E.D.**

¶53. Left paths and left spines. The *left path* at a node u_0 is the path $(u_0, \ldots, u_m)$ where each u_{i+1} is the left child of u_i and the last node u_m has no left child. The node u_m is called the *tip* of the left path. The *left spine* at a node u_0 is the path $(u_0, u_1, \ldots, u_m)$ where $(u_1, \ldots, u_m)$ is the right path of the left child of u_0. If u_0 has no left child, the left spine is the trivial path (u_0) of length 0. The *right path* and *right spine* is similarly defined. The *spine height* of u is the sum of the lengths of the left and right spines of u. As usual, by identifying the root of a tree with the tree, we may speak of the left path, right spine, etc, of a tree. Note that the smallest item in a binary search tree is at the tip of the left path.

EXAMPLE: In figure 9, the left and right paths at node 2 are $(2, 1)$ and $(2, 3, 4)$, respectively. The left spine of the tree $(5, 2, 3, 4)$. The spine height of the tree is $3 + 2 = 5$.

§21. Almost-treap and Treapification

We define a binary search tree T to be an *almost-treap at* u $(u \in T)$ if T is not a treap, but we can adjust the priority of u so that T becomes a treap. Roughly speaking, this means the heap property is satisfied everywhere except at u. Thus if we start with a treap T, we can change the priority of any node u to get an almost-treap at u. We call an almost-treap an *under-treap* if the priority of u must be adjusted upwards to get a treap, and an *over-treap* otherwise.

¶54. Defects. Suppose T is an over-treap at u. Then there is a unique maximal prefix π of the path from u to the root such that the priority of each node in π is less than the priority of u. The number of nodes in π is called the *defect* in T. Note that u is not counted in this prefix.

Next suppose T is an under-treap at u. Then there are unique maximal prefixes of the left and right spines at u such that the priority of each node in these prefixes is greater than the priority of u. The total number of nodes in these two prefixes is called the *defect* in T.

A pair (u, v) of nodes is a *violation* if either u is an ancestor of v but u has lower priority than v, or u is a descendent of v but with higher priority. It is not hard to check that that number of violations in an over-treap is equal to the defect. But this is not true in the case of an under-treap at u: this is because we only count violations along the two spines of u. We make some simple observations.

Fact 2 *Let T be an almost-treap at u with $k \geq 1$ defects. Let v be a child of u with priority at least that of its sibling (if any).*
(a) If T is an over-treap then rotating at u produces[15] *an over-treap with $k-1$ defects.*
(b) If T is an over-treap, v is defined and and $v.\texttt{priority} < u.\texttt{priority}$ then $\texttt{rotate}(v)$ produces an over-treap with $k+1$ defects.
(c) If T be an under-treap and v is defined then a rotation at v results in an under-treap at u with $k-1$ defects.

We now give a simple algorithm to convert an almost-treap into a treap. The argument to the algorithm is the node u if the tree is an almost-treap at u. It is easy to determine from u whether the tree is a treap or an under-treap or an over-treap.

```
Algorithm Treapify($u, T$):
Input: $T$ is an almost-treap at $u$.
Output: a treap $T$.
    If $u$.priority $>$ $u$.parent.priority
        then Violation=OverTreap else Violation=UnderTreap.
    switch(Violation)
        case OverTreap:
            Do rotate($u$) until
                $u$.priority $\leq$ $u$.parent.priority.
        case UnderTreap:
            Let $v$ be the child of $u$ with the largest priority.
            While $u$.priority $<$ $v$.priority DO
                rotate($v$).
                $v \leftarrow$ the child of $u$ with largest priority.
```

We let the reader verify the following observation:

Fact 3 *Let T be an almost-treap at u.*
a) (Correctness) Treapify(u) converts T into a treap.

[15] If $k = 1$, the result is actually a treap. But we shall accept the terminology "almost-treap with 0 defects" as a designation for a treap.

b) If T is an under-treap, then number of rotations is at most the spine height of u.
c) If T is an over-treap, the number of rotations is at most the depth of u.

Exercises

Exercise 21.1: Let $n = 5$ and $\sigma = (3, 1, 5, 2, 4)$ *i.e.*, $\sigma(3) = 5$ and $\sigma(4) = 1$. Draw the treap. Next, change the priority of key 4 to 10 (from 1) and treapify. Next, change the priority of key 3 to 0 (from 5) and treapify. ♢

Exercise 21.2: Start with a treap T. Compare treapifying at u after we increase the priority of u by two different amounts. Is it true that the treapifying work is of the smaller increase is no more than the work for the larger increase? What if we decrease the priority of u instead? ♢

§22. Operations on Treaps

We show the remarkable fact that all the usual operations on binary search trees can be unified within a simple framework based on treapification.

In particular, we show how implement the following binary search tree operations:

(a) Lookup(Key,Tree)→Item,
(b) Insert(Item,Tree),
(c) Delete(Node,Tree),
(d) Successor(Item,Tree)→Item,
(e) Min(Tree)→Item,
(f) DeleteMin(Tree)→Item,
(g) Split(Tree1,Key)→Tree2,
(h) Join(Tree1,Tree2).

The meaning of these operations are defined as in the case of binary search trees. So the only twist is to simultaneously maintain the properties of a max-heap with respect to the priorities. First note that the usual binary tree operations of Lookup(Key,Tree), Successor(Item,Tree), Min(Tree) and Max(Tree) can be extended to treaps without change since these do not depend modify the tree structure. Next consider insertion and deletion.

(i) Insert(u, T): we first insert in T the item u ignoring its priority. The result is an almost-treap at u. We now treapify T at u.

(ii) Delete(v, T): assuming no item in T has priority $-\infty$, we first change the priority of the item at node v into $-\infty$. Then we treapify the resulting under-treap at v. As a result, v is now a leaf which we can delete directly.

We can implement DeleteMin(Tree) using Min(Tree) and Delete(item,Tree). It is interesting to observe that this deletion algorithm for treaps translates into a new deletion algorithm for ordinary binary search trees: to delete node u, just treapify at u by pretending that the tree is an under-treap at u until u has at most one child. Since there are no priorities in ordinary binary search trees, we can rotate at any child v of u while treapifying.

¶55. Split and join of treaps. Let T be a treap on a set S of items. We can implement these two operations easily using insert and deletes.

(i) *To split T at k:* We insert a new item $u = (k, \infty)$ (*i.e.*, item u has key k and infinite priority). This yields a new treap with root u, and we can return the left subtree and right subtree to be the desired T_L, T_R.

(ii) *To join T_L, T_R:* first perform a $\mathrm{Min}(T_R)$ to obtain the minimum key k^* in T_R. Form the almost-treap whose root is the artificial node $(k^*, -\infty)$ and whose left and right subtrees are T_L and T_R. Finally, treapify at $(k^*, -\infty)$ and delete it.

The beauty of treap algorithms is that they are reduced to two simple subroutines: binary insertion and treapification.

Exercises

Exercise 22.1: For any binary search tree T and node $u \in T$, we can assign priorities to items such that T is a treap and a deletion at u takes a number of rotation equal to the the spine height of u. $\diamondsuit$

Exercise 22.2: Modify the definition of the split operation so that all the keys in S_R is strictly greater than the key k. Show how to implement this version. $\diamondsuit$

Exercise 22.3: Define "almost-treap at U" where U is a set of nodes. Try to generalize all the preceding results. $\diamondsuit$

Exercise 22.4: Alternative deletion algorithm: above, we delete a node u by making it a leaf. Describe an alternative algorithm where we step the rotations as soon as u has only one child. We then delete u by replacing u by its in-order successor or predecessor in the tree. Discuss the pros and cons of this approach. $\diamondsuit$

§23. Searching in a Random Treap

We analyze the expected cost of searching for a key in a random treap. Let us set up the random model. Assume the set of keys in the treap is $[1..n]$ and S_n the set of permutations on $[1..n]$. The event space is $(S_n, 2^{S_n})$ with a uniform probability function $\Pr_n$. Each permutation $\sigma \in S_n$ represents a choice of priority for the keys $[1..n]$ where $\sigma(k)$ is the priority for key k. Recall our "list" convention (76) (§1) for writing σ.

The *random treap* on the keys $[1..n]$ is T_n defined as follows: for $\sigma \in S_n$, $T_n(\sigma)$ is the treap on keys $[1..n]$ where the priorities of the keys are specified by σ as described above. There are

three ways in which such a sample space arise.
• We randomly generate σ directly in a uniform manner. In section §6 we describe how to do this.
• We have a fixed but otherwise arbitrary continuous distribution function $F : \mathbb{R} \to [0, 1]$ and the priority of key i is a r.v. $X_i \in [0, 1]$ with distribution F. The set of r.v.'s $X_1, \ldots, X_n$ are i.i.d.. In the continuous case, the probability that $X_i = X_j$ is negligible. In practice, we can approximate the range of F by a suitably large finite set of values to achieve the same effect.
• The r.v.'s $X_1, \ldots, X_n$ could be given a uniform probability distribution if successive bits of each X_i is obtained by tossing a fair coin. Moreover, we will generate as many bits of X_i as are needed by our algorithm when comparing priorities. It can be shown that the expected number of bits needed is a small constant.

We frequently relate a r.v. of $\Pr_n$ to another r.v. of $\Pr_k$ for $k \neq n$. The following simple lemma illustrates a typical setting.

Lemma 23 *Let X, Y be r.v.'s of $\Pr_n, \Pr_k$ (respectively) where $1 \le k < n$. Suppose there is a map $\mu : S_n \to S_k$ such that*
(i) for each $\sigma' \in S_k$, the set $\mu^{-1}(\sigma')$ of inverse images has size $n!/k!$, and
(ii) for each $\sigma \in S_n$, $X(\sigma) = Y(\mu(\sigma))$.
Then $\mathtt{E}[X] = \mathtt{E}[Y]$.

Proof.
$$\mathtt{E}[X] = \frac{\sum_{\sigma \in S_n} X(\sigma)}{n!} = \frac{\sum_{\sigma' \in S_k} (n!/k!) Y(\sigma')}{n!} = \frac{\sum_{\sigma' \in S_k} Y(\sigma')}{k!} = \mathtt{E}[Y].$$
Q.E.D.

The map μ that "erases" from the string σ all keys greater than k has the properties stated in the lemma. E.g., if $n = 5$, $k = 3$ then $\sigma(3, 1, 4, 2, 5) = (3, 1, 2)$ and $\sigma^{-1}(3, 1, 2) = 4 \cdot 5 = 20$.

Recalling our example in figure 9: notice that if we insert the keys [1..7] into an initially empty binary search tree in order of decreasing priority, (i.e., first insert key 5, then key 2, then key 1, etc), the result is the desired treap. This illustrates the following lemma:

Lemma 24 *Suppose we repeatedly insert into a binary search tree the following sequence of keys $\sigma^{-1}(n), \sigma^{-1}(n-1), \ldots, \sigma^{-1}(1)$, in this order. If the initial binary tree is empty, the final binary tree is in fact the treap $T_n(\sigma)$.*

*Proof.*The proposed sequence of insertions results in a binary search tree, by definition. We only have to check that the heap property. This is seen by induction: the first insertion clearly results in a heap. If the ith insertion resulted in a heap, then the $(i+1)$st insertion is an almost heap. Since this newly inserted node is a leaf, and its rank is less than all the other nodes already in the tree, it cannot cause any violation of the heap property. So the almost heap is really a heap. Since treaps are unique for unique keys and ranks, this tree must be equal to T_σ.
Q.E.D.

The above lemma suggests that a random treap can be regarded as a binary search tree that arises from random insertions.

¶56. Random ancestor sets A_k. We are going to fix $k \in [1..n]$ as the key we are searching for in the random treap T_n. To analyze the expected cost of this search, we define the random set $A_k : S_n \to 2^{[1..n]}$ where

$$A_k(\sigma) := \{j : j \text{ is an ancestor of } k \text{ in } T_n(\sigma)\}.$$

By definition, $k \in A_k(\sigma)$. Clearly, the cost of LookUp(T_n, k) is equal to $\Theta(|A_k|)$. Hence our goal is to determine $\mathtt{E}[|A_k|]$. To do this, we split the random variable $|A_k|$ into two parts:

$$|A_k| = |A_k \cap [1..k]| + |A_k \cap [k..n]| - 1.$$

By the linearity of expectation, we can determine the expectations of the two parts separately.

¶57. Running k-maximas of σ. We give a descriptive name to elements of the set

$$A_k(\sigma) \cap [1..k].$$

A key j is called a *running k-maxima of σ* if $j \in [1..k]$ and for all $i \in [1..k]$,

$$\sigma(i) > \sigma(j) \Rightarrow i < j.$$

Of course, $\sigma(i) > \sigma(j)$ just means that i appears before j in the list

$$(\sigma^{-1}(n), \ldots, \sigma^{-1}(1)).$$

In other words, as we run down this list, by the time we come to j, it must be bigger than any other elements from the set $[1..k]$ that we have seen so far. Hence we call j a "running maxima".

EXAMPLE. Taking $\sigma = (4, 3, 1, 6, 2, 7, 5)$ as in figure 9, the running 7-maximas of σ are 4, 6 and 7. Note that no running k-maximas appears after k in the list σ.

Lemma 25 *$j \in A_k(\sigma) \cap [1..k]$ iff j is a running k-maxima of σ.*

Proof.($\Rightarrow$) Suppose $j \in A_k(\sigma) \cap [1..k]$. We want to show that j is a running k-maxima. That is, for all $i \in [1..k]$ and $\sigma(i) > \sigma(j)$, we must show that $i < j$. Look at the path π_j from the root to j that is traced while inserting j. It will initially retrace the corresponding path π_i for i (which was inserted earlier). Let i' be the last node that is common to π_i and π_j (so i' is the least common ancestor of i and j). We allow the possibility $i = i'$ (but surely $j \neq i'$ since $j = i'$ would make j an ancestor of i, violating the max-heap property). There are two cases: j is either (a) in the right subtree of i' or (b) in the left subtree of i'. We claim that it cannot be (b). To see this, consider the path π_k traced by inserting k. Since j is an ancestor of k, π_j must be a prefix of π_k. If j is a left descendent of i', then so is k. Since i is either i' or lies in the left subtree of i', this means $i > k$, contradiction. Hence (a) holds and we have $j > i' \geq i$.

($\Leftarrow$) Conversely, suppose j is a running k-maxima in σ. It suffices to show that $j \in A_k(\sigma)$. This is equivalent to showing

$$A_j(\sigma) \subseteq A_k(\sigma).$$

View $T_n(\sigma)$ as the result of inserting a sequence of keys by decreasing priority. At the moment of inserting k, key j has already been inserted. We just have to make sure that the search algorithm for k follows the path π_j from the root to j. This is seen inductively: clearly k visits the root. Inductively, suppose k is visiting a key i on the path π_j. If $i > k$ then surely both j and k next visit the left child of i. If $i < k$ then $j > i$ (since j is a running maxima) and hence again both j and k next visit the right child of i. This proves that k eventually visits j.

Q.E.D.

¶58. Running k-minimas of σ. Define a key j to be a *running k-minima of σ* if $j \in [k..n]$ and for all $i \in [k..n]$,

$$\sigma(i) > \sigma(j) \Rightarrow j < i.$$

With $\sigma = (4,3,1,6,2,7,5)$ as before, the running 2-minimas of σ are 4, 3 and 2. As for running k-maximas, no running k-minimas appear after k in the list σ. We similarly have:

Lemma 26 *$A_k(\sigma) \cap [k..n]$ is the set of running k-minimas of σ.*

Define the random variable U_n where $U_n(\sigma)$ is the number of running n-maximas in $\sigma \in S_n$.

We will show that U_k has the same expected value as $|A_k \cap [1..k]|$. Note that U_k is a r.v. of $\Pr_k$ while $|A_k \cap [1..k]|$ is a r.v. of $\Pr_n$. Similarly, define the r.v. V_n where $V_n(\sigma)$ is the number of running 1-minimas in $\sigma \in S_n$.

Lemma 27
(a) $\mathtt{E}[|A_k \cap [1..k]|] = \mathtt{E}[U_k]$.
(b) $\mathtt{E}[|A_k \cap [k..n]|] = \mathtt{E}[V_{n-k+1}]$.

*Proof.*Note that the r.v. $|A_k \cap [1..k]|$ is related to the r.v. U_k exactly as described in lemma 23. This is because in each $\sigma \in S_n$, the keys in $[k+1..n]$ are clearly irrelevant to the number we are counting. Hence (a) is a direct application of lemma 23. A similar remark applies to (b) because the keys in $[1..k-1]$ are irrelevant in the running k-minimas. **Q.E.D.**

We give a recurrence for the expected number of running k-maximas.

Lemma 28
(a) $\mathtt{E}[U_1] = 1$ *and* $\mathtt{E}[U_k] = \mathtt{E}[U_{k-1}] + \frac{1}{k}$ *for* $k \geq 2$. *Hence* $\mathtt{E}[U_k] = H_k$
(b) $\mathtt{E}[V_1] = 1$ *and* $\mathtt{E}[V_k] = \mathtt{E}[V_{k-1}] + \frac{1}{k}$ *for* $k \geq 2$. *Hence* $\mathtt{E}[V_k] = H_k$

*Proof.*We consider (a) only, since (b) is similar. Consider the map taking $\sigma \in S_k$ to $\sigma' \in S_{k-1}$ where we delete 1 from the sequence $(\sigma^{-1}(k), \ldots, \sigma^{-1}(1))$ and replace each remaining key i by $i-1$, and take this sequence as the permutation σ'. For instance, $\sigma = (4,3,1,5,2)$ becomes $\sigma' = (3,2,4,1)$. Note that $U_5(\sigma) = 2 = U_4(\sigma')$. On the other hand, if $\sigma = (1,4,3,5,2)$ then $\sigma' = (3,2,4,1)$. and $U_5(\sigma) = 3$ and $U_4(\sigma') = 2$. In general, we have

$$U_k(\sigma) = U_{k-1}(\sigma') + \delta(\sigma)$$

where $\delta(\sigma) = 1$ or 0 depending on whether or not $\sigma(1) = k$. For each $\sigma' \in S_{k-1}$, that there are exactly k permutations $\sigma \in S_k$ that maps to σ'; moreover, of these k permutations, exactly one has $\delta(\sigma) = 1$. Thus

$$\begin{aligned}
\mathtt{E}[U_k] &= \frac{\sum_{\sigma\in S_k} U_k(\sigma)}{k!}\\
&= \frac{\sum_{\sigma\in S_k} (U_{k-1}(\sigma') + \delta(\sigma))}{k!}\\
&= \frac{\sum_{\sigma'\in S_{k-1}} (1 + kU_{k-1}(\sigma'))}{k!}\\
&= \frac{1}{k} + \frac{\sum_{\sigma'\in S_{k-1}} U_{k-1}(\sigma')}{(k-1)!}\\
&= \frac{1}{k} + \mathtt{E}[U_{k-1}].
\end{aligned}$$

Clearly, the solution to $\mathtt{E}[U_k]$ is the kth harmonic number $H_k = \sum_{i=1}^{k} 1/i$. **Q.E.D.**

The depth of k in the treap $T_n(\sigma)$ is

$$|A_k(\sigma)| - 1 = |A_k(\sigma) \cap [1..k]| + |A_k(\sigma) \cap [k..n]| - 2.$$

Since the cost of LookUp(k) is proportional to 1 plus the depth of k, we obtain:

Corollary 29 *The expected cost of LookUp(k) in a random treap of n elements is proportional to*

$$\mathtt{E}[U_k] + \mathtt{E}[V_{n-k+1}] - 1 = H_k + H_{n-k+1} - 1 \le 1 + 2\ln n.$$

Exercises

Exercise 23.1: (i) Prove lemma 26. (ii) Minimize $H_k + H_{n-k+1}$ for k ranging over $[1..n]$. ◊

Exercise 23.2:
(i) Show that the variance of U_n is $\mathcal{O}(\log n)$. In fact, $\mathtt{Var}(U_n) < H_n$.
(ii) Use Chebyshev's inequality to show that the probability that U_n is more than twice its expected value is $\mathcal{O}(\frac{1}{\log n})$. ◊

§24. Insertion and Deletion

A treap insertion has two phases:
Insertion Phase. This puts the item in a leaf position, resulting in an almost-treap.
Rotation Phase. This performs a sequence of rotations to bring the item to its final position.
The insertion phase has the same cost as searching for the successor or predecessor of the item to be inserted. By the previous section, this work is expected to be $\Theta(\log n)$. What about the rotation phase?

Lemma 30 *For any set S of items and $u \in S$, there is an almost-treap T at u in which u is a leaf and the set of nodes is S. Moreover, if the keys and priorities in S are unique, then T is unique.*

*Proof.*To see the existence of T, we first construct the treap T' on $S - \{u\}$. Then we insert u into T' to get an almost-treap at u. For uniqueness, we note that when the keys and priorities are unique and u is a leaf of an almost-treap T, then the deletion of u from T gives a unique treap T'. On the other hand, T is uniquely determined from u and T'. **Q.E.D.**

This lemma shows that insertion and deletion are inverses in a very strong sense. Recall that for deletion, we assume that we are given a node u in the treap to be deleted. Then there are two parts again:

Rotation Phase. This performs a sequence of rotations to bring the node to a leaf position.

Deletion Step. This simply deletes the leaf.

We now clarify the precise sense in which rotation and deletion are inverses of each other.

Corollary 31 *Let T be a treap containing a node u and T' be the treap after deleting u from T. Let T_+ be the almost-treap just before the rotation phase while inserting u into T'. Let T_- be the almost-treap just after the rotation phase while deleting u from T.*

(i) T_+ and T_- are identical.

(ii) The intermediate almost-treaps obtained in the deletion rotations are the same as the intermediate almost-treaps obtained in the insertion rotations (but in the opposite order).

In particular, the costs of the rotation phase in deletion and in insertion are equal. Hence it suffices to analyze the cost of rotations in when deleting a key k. This cost is $\mathcal{O}(L_k + R_k)$ where L_k and R_k are the lengths of the left and right spines of k. These lengths are at most the depth of the tree, and we already proved that the expected depth is $\mathcal{O}(\log n)$. Hence we conclude: *the expected time to insert into or delete from a random treap is $\mathcal{O}(\log n)$.*

In this section, we give a sharper bound: the expected number of rotations during an insertion or deletion is at most 2.

¶59. Random left-spine sets B_k. Let us fix the key $k \in [1..n]$ to be deleted. We want to determine the expected spine height of k. We compute the left and right spine heights separately. For any $\sigma \in S_n$ and any key $k \in [1..n]$, define the set

$$B_k(\sigma) := \{j : j \text{ is in the left spine of } k\}.$$

¶60. Triggered running k-maxima. Again, we can give a descriptive name for elements in the random set B_k. Define j to be a *triggered running k-maxima of σ* if $j \in [1..k-1]$ and $\sigma(k) > \sigma(j)$ and for all $i \in [1..k-1]$,

$$\sigma(i) > \sigma(j) \Rightarrow i < j.$$

In other words, as we run down the list

$$(\sigma^{-1}(n), \ldots, k, \ldots, j, \ldots, \sigma^{-1}(1)),$$

by the time we come to j, we must have seen k (this is the trigger) and j must be bigger than any other elements in $[1..k-1]$ that we have seen so far. Note that the other elements in $[1..k-1]$ may occur before k in the list (*i.e.*, they need not be triggered).

With $\sigma = (4, 3, 1, 6, 2, 7, 5)$ as in figure 9, the triggered running 3-maximas of σ are 1 and 2. The triggered running 6-maximas of σ is comprised of just 7; note that 2 is not a 6-maxima because 2 is less than some keys in [1..5] which occurred earlier than 2.

Lemma 32 *(Characterization of left-spine) Then $j \in B_k(\sigma)$ iff j is a triggered running k-maxima of σ.*

Proof.(⇒) Suppose $j \in B_k(\sigma)$. We want to show that
(i) $j < k$,
(ii) $\sigma(k) > \sigma(j)$, and
(iii) for all $i \in [1..k-1]$, $\sigma(i) > \sigma(j)$ implies $i < j$.
But (i) holds because of the binary search tree property, and (ii) holds by the heap property. What about (iii)? Let $i \in [1..k-1]$ and $\sigma(i) > \sigma(j)$. We want to prove that $i < j$. *First assume i lies in the path from root to j.* There are two cases.

CASE (1): k is a descendent of i. Then we see that $k > i$ and j is a descendent of k implies $j > i$.

CASE (2): i is a descendent of k. But in a left-spine, the items have increasing keys as they lie further from the root. In particular, $i < j$, as desired.
Now consider the case where i does not lie in the path from the root to j. Let i' be the least common ancestor of i and j. Clearly either

$$i < i' < j \qquad \text{or} \qquad j < i' < i$$

must hold. So it suffices to prove that $i' < j$ (and so the first case hold). Note that $i' \neq k$. If i' is a descendent of k then essentially CASE (2) above applies, and we are done. So assume k is a descendent of i'. Since j is also a descendent of k, it follows that i and k lie in different (left or right) subtrees of i'. By definition $i < k$. Hence i lies in the left subtree of i' and k lies in the right subtree of i'. Hence j lies in the right subtree of i', i.e., $j > i'$, as desired. This proves (iii).
(⇐) Suppose j is a triggered running k-maxima. We want to show that $j \in B_k(\sigma)$. We view $T_n(\sigma)$ as the result of inserting items according to priority (§3): assume that we are about to insert key j. Note that k is already in the tree. We must show that j ends up in the "tip" of the left-spine of k. For each key i along the path from the root to the tip of the left-spine of k, we consider two cases.

CASE (3): $i > k$. Then i is an ancestor of k and (by (i)) we have $i > k > j$. This means j will move into the subtree of i which contains k.

CASE (4): $i < k$. Then by (iii), $i < j$. Again, this means that if i is an ancestor of k, j will move into the subtree of i that contains k, and otherwise i is in the left-spine of k and j will move into the right subtree of i (and thus remain in the left-spine of k). **Q.E.D.**

It is evident that the keys in $[k+1..n]$ is irrelevant to the number of triggered running k-maximas. Hence we may as well assume that $\sigma \in S_k$ (instead of S_n). To capture this, let us define the r.v. T_k that counts the number of triggered running k-maximas in case $n = k$.

Lemma 33 *Consider the r.v. $|B_k|$ of $\Pr_n$. It is related to the r.v. T_k of $\Pr_k$ via $\mathtt{E}[|B_k|] = \mathtt{E}[T_k]$.*

*Proof.*For each $\sigma \in S_n$, let us define its *image σ' in S_k* obtained by taking the sequence $(\sigma^{-1}(n), \ldots, \sigma^{-1}(1))$, deleting all keys of $[k+1..n]$ from this sequence, and considering the result as a permutation $\sigma' \in S_k$. Note that $T_k(\sigma') = |B_k(\sigma)|$. It is not hard to see that there are exactly $n!/k!$ choices of $\sigma \in S_k$ whose images equal a given $\sigma' \in S_k$. Hence lemma 23 implies $\mathtt{E}[|B_k|] = \mathtt{E}[T_k]$. **Q.E.D.**

Lemma 34 *We have $\mathtt{E}[T_1] = 0$. For $k \geq 2$,*

$$\mathtt{E}[T_k] = \mathtt{E}[T_{k-1}] + \frac{1}{k(k-1)}$$

and hence $\mathtt{E}[T_k] = \frac{k-1}{k}$.

*Proof.*It is immediate that $\mathtt{E}[T_1] = 0$. So assume $k \ge 2$. Again consider the map from $\sigma \in S_k$ to a corresponding $\sigma' \in S_{k-1}$ where σ' is the sequence obtained from $(\sigma^{-1}(k), \ldots, \sigma^{-1}(1))$ by deleting the key 1 and subtracting 1 from each of the remaining keys. Note that

• each $\sigma' \in S_{k-1}$ is the image of exactly k distinct $\sigma \in S_k$, and

• for each $j \in [2..k-1]$, j is a triggered running k-maxima in σ iff $j-1$ is a triggered running $(k-1)$-maxima in σ'.

• key 1 is a triggered running k-maxima in σ iff $\sigma(k) = k$ and $\sigma(1) = k-1$.

This proves
$$T_k(\sigma) = T_{k-1}(\sigma') + \delta(\sigma)$$
where $\delta(\sigma) = 1$ or 0 depending on whether or not $\sigma = (k, 1, \sigma^{-1}(k-2), \ldots, \sigma^{-1}(1))$. Now, there are exactly $(k-2)!$ choices of $\sigma \in S_k$ such that $\delta(\sigma) = 1$. Hence
$$\begin{aligned}
\mathtt{E}[T_k] &= \frac{\sum_{\sigma \in S_k} T_k(\sigma)}{k!} \\
&= \frac{\sum_{\sigma \in S_k} (T_{k-1}(\sigma') + \delta(\sigma))}{k!} \\
&= \frac{\sum_{\sigma \in S_k} T_{k-1}(\sigma')}{k!} + \frac{(k-2)!}{k!} \\
&= \frac{\sum_{\sigma' \in S_{k-1}} kT_{k-1}(\sigma')}{k!} + \frac{1}{k(k-2)} \\
&= \mathtt{E}[T_{k-1}] + \frac{1}{k(k-2)}.
\end{aligned}$$
It is immediate that
$$\mathtt{E}[T_k] = \sum_{i=2}^{k} \frac{1}{i(i-1)},$$
using the fact that $\mathtt{E}[T_1] = 0$. This is a telescoping sum in disguise because
$$\frac{1}{i(i-1)} = \frac{1}{i-1} - \frac{1}{i}.$$
The solution follows at once. **Q.E.D.**

¶61. Right spine. Using symmetry, the expected length for the right spine of k is easily obtained. We define j to be a *triggered running k-minimas in σ* if (i) $j > k$, (ii) $\sigma(j) < \sigma(k)$, (iii) for all $i \in [k+1..n]$, $\sigma(i) > \sigma(j)$ implies $j < i$. We leave as an exercise to show:

Lemma 35 *j is in the right spine of k iff j is a triggered running k-minima of σ.*

The keys in $[1..k-1]$ are irrelevant for this counting. Hence we may define the random variable
$$R_k(\sigma)$$
which counts the number of triggered running 1-minimas in $\sigma \in S_k$. We have:

Lemma 36
(a) The expected length of the right spine is given by $\mathtt{E}[R_{n-k+1}]$.
(b) For $k \ge 2$, $\mathtt{E}[R_k] = \mathtt{E}[R_{k-1}] + \frac{1}{k(k-1)}$. *The solution is* $\mathtt{E}[R_k] = (k-1)/k$.

Corollary 37 *The expected length of the right spine of k is*

$$\mathtt{E}[R_{n-k+1}] = \frac{n-k}{n-k+1}.$$

Hence the expected lengths of the left and right spines of a key k is less than 1 each. This leads to the surprising result:

Corollary 38 *The expected number of rotations in deleting or inserting a key from the random treap T_n is less than 2.*

This is true because each key is likely to be a leaf or near one. So this result is perhaps not surprising since more than half of the keys are leaves.

Exercises

Exercise 24.1: Show lemma 35. ◇

§25. Cost of a Sequence of Requests

We have shown that for single requests, it takes expected $\mathcal{O}(\log n)$ time to search or insert, and $\mathcal{O}(1)$ time to delete. Does this bound apply to a sequence of such operations? The answer is not obvious because it is unclear whether the operations may upset our randomness model. How do we ensure that inserts or deletes to a random treap yields another random treap?

We formalize the problem as follows: fix any sequence

$$p_1, \ldots, p_m$$

of treap requests on the set $[1..n]$ of keys. In other words, each p_i ($k \in [1..n]$ has one of the forms

$$\text{LookUp}(k), \text{Insert}(k), \text{Delete}(k).$$

We emphasize that that the above sequence of requests is arbitrary, not "random" in any probabilistic sense. Also note that we assume that deletion operation is based on a key value, not on a "node" as we normally postulate (§4). This variation should not be a problem in most applications.

Our probability space is again

$$(S_n, 2^{S_n}, \Pr_n).$$

For each $i = 1, \ldots, m$ and $\sigma \in S_n$, let $T_i(\sigma)$ be the treap obtained after the sequence of operations $p_1, \ldots, p_i$, where σ specifies the priorities assigned to keys and $T_0(\sigma)$ is the initial empty treap. Define X_i to be the r.v. such that $X_i(\sigma)$ is the cost of performing the request p_i on the treap $T_{i-1}(\sigma)$, resulting in $T_i(\sigma)$. The expected cost of operation p_i is simply $\mathtt{E}[X_i]$.

Lemma 39

$$\mathtt{E}[X_i] = \mathcal{O}(\log i).$$

*Proof.*Fix $i = 1, \ldots, m$. Let $K_i \subseteq [1..n]$ denote the set of keys that are in $T_i(\sigma)$. A key observation is the dependence of $T_{i-1}(\sigma)$ on $p_1, \ldots, p_{i-1}$ amounts only to the fact that these $i-1$ operations determine K_i. Let $|K_i| = n_i$. Then there is a natural map taking $\sigma \in S_n$ to $\sigma' \in S_{n_i}$ (*i.e.*, the map deletes from the sequence $(\sigma^{-1}(n), \ldots, \sigma^{-1}(1))$ all elements not in K_i, and finally replacing each remaining key $k \in K_i$ by a corresponding key $\pi_i(k) \in [1..n_i]$ which preserves key-order). Each σ' is the image of $n!/(n_i)!$ distinct $\sigma \in S_n$. Moreover, for any key $k \in K_i$, its relevant statistics (the depth of k and length of left/right spines of k) in $T_i(\sigma)$ is the same as that of its replacement key $\pi_i(k)$ in the treap on $[1..k]$. As shown above, the expected values of these statistics is the same as the expected values of these statistics on the uniform probability space on S_{n_i}. But we have proven that the expected depth of any $k \in [1..n_i]$ is $\mathcal{O}(\log n_i)$, and the expected length of the spines of k is at most 1. This proves that when the update operation is based on a key $k \in K_i$, $\mathtt{E}[X_i] = \mathcal{O}(\log n_i) = \mathcal{O}(\log i)$, since $n_i \le i$.

What if the operation p_i is an unsuccessful search, i.e., we do a LookUp(k) where $k \notin K_i$? In this case, the length of the path that the algorithm traverses is bounded by the depth of either the successor or predecessor of k in K_i. Again, the expected lengths in $\mathcal{O}(\log i)$. This concludes our proof. **Q.E.D.**

¶62. Application to Sorting. We give a randomized sorting algorithm as follows: on input a set of n keys, we assign them a random priority (by putting them in an array in decreasing order of priority). Then we do a sequence of n inserts in order of this priority. Finally we do a sequence of n DeleteMins. This gives an expected $\mathcal{O}(n \log n)$ algorithm for sorting.

NOTES: Galli and Simon (1996) suggested an alternative approach to treaps which avoid the use of random priorities. Instead, they choose the root of the binary search tree randomly. Mulmuley introduced several abstract probabilistic "game models" that corresponds to our analysis of running (triggered) maximas or minimas.

Exercises

Exercise 25.1: What is the worst case and best case key for searching and insertion in our model? ♢

End Exercises

References

[1] Alon, Itai, and Babai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7:567–583, 1986.

[2] N. Alon, O. Goldreich, J. Hastad, and R. Peralta. Simple constructions of almost k-wise independent random variables. In *Proc. 31st IEEE Symposium on Foundations of Computer Science*, page 544–553, 1990.

[3] N. Alon, J. H. Spencer, and P. Erdös. *The probabilistic method.* John Wiley and Sons, Inc, New York, 1992.

[4] C. R. Aragon and R. G. Seidel. Randomized search trees. *IEEE Foundations of Comp. Sci.*, 30:540–545, 1989.

[5] H. Chernoff. A measure of asymptotic efficiency for tests of hypothesis based on sum of observations. *Ann. of Math. Stat.*, 23:493–507, 1952.

[6] K. L. Chung. *Elementary Probability Theory with Stochastic Processes.* Springer-Verlag, New York, 1979.

[7] W. Feller. *An introduction to Probability Theory and its Applications.* Wiley, New York, 2nd edition edition, 1957. (Volumes 1 and 2).

[8] A. Joffe. On a set of almost deterministic k-independent random variables. *The Annals of Probability*, 2(1):161–162, 1974.

[9] A. T. Jonassen and D. E. Knuth. A trivial algorithm whose analysis isn't. *J. Computer and System Sciences*, 16:301–322, 1978.

[10] R. M. Karp. Probabilistic recurrence relations. *J. ACM*, 41(6):1136–1150, 1994.

[11] D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, Boston, 2nd edition edition, 1975.

[12] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Boston, 2nd edition edition, 1981.

[13] A. N. Kolmogorov. *Foundations of the theory of probability.* Chelsea Publishing Co., New York, 1956. Second English Edition.

[14] M. Li and P. M. B. Vitányi. *An introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag, second edition, 1997.

[15] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[16] J. Naor and M. Naor. Small-bias probability spaces: efficient constructions and applications. *SIAM J. Computing*, 22:838–856, 1993.

[17] W. W. Peterson and J. E. J. Weldon. *Error-Correcting Codes.* MIT Press, 1975. 2nd Edition.

[18] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.