10.5. Exercises

Exercise 10.1: Formally prove Theorem 10.1.

Exercise 10.2: Another method for approximating π using Monte Carlo techniques is based on Buffon's needle experiment. Research and explain Buffon's needle experiment, and further explain how it can be used to obtain an approximation for π.

Exercise 10.3: Show that the following alternative definition is equivalent to the definition of an FPRAS given in the chapter: A *fully polynomial randomized approximation scheme* (*FPRAS*) for a problem is a randomized algorithm for which, given an input x and any parameter ε with $0 < \varepsilon < 1$, the algorithm outputs an $(\varepsilon, 1/4)$-approximation in time that is polynomial in $1/\varepsilon$ and the size of the input x. (*Hint:* To boost the probability of success from 3/4 to $1 - \delta$, consider the *median* of several *independent* runs of the algorithm. Why is the median a better choice than the mean?)

Exercise 10.4: Suppose we have a class of instances of the DNF satisfiability problem, each with $\alpha(n)$ satisfying truth assignments for some polynomial α. Suppose we apply the naïve approach of sampling assignments and checking whether they satisfy the formula. Show that, after sampling $2^{n/2}$ assignments, the probability of finding even a single satisfying assignment for a given instance is exponentially small in n.

Exercise 10.5: **(a)** Let $S_1, S_2, \ldots, S_m$ be subsets of a finite universe U. We know $|S_i|$ for $1 \leq i \leq m$. We wish to obtain an (ε, δ)-approximation to the size of the set

$$S = \bigcup_{i=1}^{m} S_i.$$

We have available a procedure that can, in one step, choose an element uniformly at random from a set S_i. Also, given an element $x \in U$, we can determine the number of sets S_i for which $x \in S_i$. We call this number $c(x)$.

Define p_i to be

$$p_i = \frac{|S_i|}{\sum_{j=1}^{m} |S_j|}.$$

The jth trial consists of the following steps. We choose a set S_j, where the probability of each set S_i being chosen is p_i, and then we choose an element x_j uniformly at random from S_j. In each trial the random choices are independent of all other trials. After t trials, we estimate $|S|$ by

$$\left(\frac{1}{t}\sum_{j=1}^{t}\frac{1}{c(x_j)}\right)\left(\sum_{i=1}^{m}|S_i|\right).$$

Determine – as a function of m, ε, and δ – the number of trials needed to obtain an (ε, δ)-approximation to $|S|$.

(b) Explain how to use your results from part (a) to obtain an alternative approximation algorithm for counting the number of solutions to a DNF formula.

Exercise 10.6: The problem of counting the number of solutions to a knapsack instance can be defined as follows: Given items with sizes $a_1, a_2, \ldots, a_n > 0$ and an integer $b > 0$, find the number of vectors $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ such that $\sum_{i=1}^n a_i x_i \leq b$. The number b can be thought of as the size of a knapsack, and the x_i denote whether or not each item is put into the knapsack. Counting solutions corresponds to counting the number of different sets of items that can be placed in the knapsack without exceeding its capacity.

(a) A naïve way of counting the number of solutions to this problem is to repeatedly choose $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ uniformly at random, and return the 2^n times the fraction of samples that yield valid solutions. Argue why this is not a good strategy in general; in particular, argue that it will work poorly when each a_i is 1 and $b = \sqrt{n}$.

(b) Consider a Markov chain $X_0, X_1, \ldots$ on vectors $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$. Suppose X_j is $(x_1, x_2, \ldots, x_n)$. At each step, the Markov chain chooses $i \in [1, n]$ uniformly at random. If $x_i = 1$, then X_{j+1} is obtained from X_j by setting x_i to 0. If $x_i = 0$, then X_{j+1} is obtained from X_j by setting x_i to 1 if doing so maintains the restriction $\sum_{i=1}^n a_i x_i \leq b$. Otherwise, $X_{j+1} = X_j$.

Argue that this Markov chain has a uniform stationary distribution whenever $\sum_{i=1}^n a_i > b$. Be sure to argue that the chain is irreducible and aperiodic.

(c) Argue that, if we have an FPAUS for the knapsack problem, then we can derive an FPRAS for the problem. To set the problem up properly, assume without loss of generality that $a_1 \leq a_2 \leq \cdots \leq a_n$. Let $b_0 = 0$ and $b_i = \sum_{j=1}^i a_i$. Let $\Omega(b_i)$ be the set of vectors $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ that satisfy $\sum_{i=1}^n a_i x_i \leq b_i$. Let k be the smallest integer such that $b_k \geq b$. Consider the equation

$$|\Omega(b)| = \frac{|\Omega(b)|}{|\Omega(b_{k-1})|} \times \frac{|\Omega(b_{k-1})|}{|\Omega(b_{k-2})|} \times \cdots \times \frac{|\Omega(b_1)|}{|\Omega(b_0)|} \times |\Omega(b_0)|.$$

You will need to argue that $|\Omega(b_{i-1})|/|\Omega(b_i)|$ is not too small. Specifically, argue that $|\Omega(b_i)| \leq (n+1)|\Omega(b_{i-1})|$.

Exercise 10.7: An alternative definition for an ε-uniform sample of Ω is as follows: A sampling algorithm generates an ε-uniform sample w if, for all $x \in \Omega$,

$$\frac{|\Pr(w = x) - 1/|\Omega||}{1/|\Omega|} \leq \varepsilon.$$

Show that an ε-uniform sample under this definition yields an ε-uniform sample as given in Definition 10.3

Exercise 10.8: Let $S = \sum_{i=1}^{\infty} i^{-2} = \pi^2/6$. Design a Markov chain based on the Metropolis algorithm on the positive integers such that, in the stationary distribution, $\pi_i = 1/Si^2$. The neighbors of any integer $i > 1$ for your chain should be only $i - 1$ and $i + 1$, and the only neighbor of 1 should be the integer 2.

Exercise 10.9: Recall the Bubblesort algorithm of Exercise 2.22. Suppose we have n cards labeled 1 through n. The order of the cards X can be the state of a Markov chain. Let $f(X)$ be the number of Bubblesort moves necessary to put the cards in increasing sorted order. Design a Markov chain based on the Metropolis algorithm such that, in the stationary distribution, the probability of an order X is proportional to $\lambda^{f(X)}$ for a given constant $\lambda > 0$. Pairs of states of the chain are connected if they correspond to pairs of orderings that can be obtained by interchanging at most two cards.

Exercise 10.10: A Δ-coloring C of an undirected graph $G = (V, E)$ is an assignment labeling each vertex with a number, representing a color, from the set $\{1, 2, \ldots, \Delta\}$. An edge (u, v) is *improper* if both u and v are assigned the same color. Let $I(C)$ be the number of improper edges of a coloring C. Design a Markov chain based on the Metropolis algorithm such that, in the stationary distribution, the probability of a coloring C is proportional to $\lambda^{I(C)}$ for a given constant $\lambda > 0$. Pairs of states of the chain are connected if they correspond to pairs of colorings that differ in just one vertex.

Exercise 10.11: In Section 10.4.1 we constructed a Markov chain on the independent sets of a graph where, in the stationary distribution, $\pi_x = \lambda^{|I_x|}/B$. Here I_x is the independent set corresponding to state x and $B = \sum_x \lambda^{|I_x|}$. Using a similar approach, construct a Markov chain on the independent sets of a graph *excluding the empty set*, where $\pi_x = |I_x|/B$ for a constant B. Because the chain excludes the empty set, you should first design a neighborhood structure that ensures the state space is connected.

Exercise 10.12: The following generalization of the Metropolis algorithm is due to Hastings. Suppose that we have a Markov chain on a state space Ω given by the transition matrix $\mathbf{Q}$ and that we want to construct a Markov chain on this state space with a stationary distribution $\pi_x = b(x)/B$, where for all $x \in \Omega$, $b(x) > 0$ and $B = \sum_{x \in \Omega} b(x)$ is finite. Define a new Markov chain as follows. When $X_n = x$, generate a random variable Y with $\Pr(Y = y) = Q_{x,y}$. Notice that Y can be generated by simulating one step of the original Markov chain. Set X_{n+1} to Y with probability

$$\min\left(\frac{\pi_y Q_{y,x}}{\pi_x Q_{x,y}}, 1\right),$$

and otherwise set X_{n+1} to X_n. Argue that, if this chain is aperiodic and irreducible, then it is also time reversible and has a stationary distribution given by the π_x.

Exercise 10.13: Suppose we have a program that takes as input a number x on the real interval $[0, 1]$ and outputs $f(x)$ for some bounded function f taking on values in the range $[1, b]$. We want to estimate

$$\int_{x=0}^{1} f(x)\, dx.$$

Assume that we have a random number generator that can generate independent uniform random variables $X_1, X_2, \ldots$. Show that

$$\sum_{i=1}^{m} \frac{f(X_i)}{m}$$

gives an (ε, δ)-approximation for the integral for a suitable value of m.

10.6. An Exploratory Assignment on Minimum Spanning Trees

Consider a complete, undirected graph with $\binom{n}{2}$ edges. Each edge has a weight, which is a real number chosen uniformly at random on $[0, 1]$.

Your goal is to estimate how the expected weight of the minimum spanning tree grows as a function of n for such graphs. This will require implementing a minimum spanning tree algorithm as well as procedures that generate the appropriate random graphs. (You should check to see what sorts of random number generators are available on your system and determine how to seed them – say, with a value from the machine's clock.)

Depending on the algorithm you use and your implementation, you may find that your program uses too much memory when n is large. To reduce memory when n is large, we suggest the following approach. In this setting, the minimum spanning tree is extremely unlikely to use any edge of weight greater than $k(n)$ for some function $k(n)$. We can first estimate $k(n)$ by using repeated runs for small values of n and then throw away edges of weight larger than $k(n)$ when n is large. If you use this approach, be sure to explain why throwing away edges in this manner will not lead to a situation where the program finds a spanning tree that is not actually minimal.

Run your program for $n = 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192$, and larger values, if your program runs fast enough. Run your program at least five times for each value of n and take the average. (Make sure you re-seed the random number generator appropriately!) You should present a table listing the average tree size for the values of n that your program runs successfully. What seems to be happening to the average size of the minimum spanning tree as n grows?

In addition, you should write one or two pages discussing your experiments in more depth. The discussion should reflect what you have learned from this assignment and might address the following topics.

- What minimum spanning tree algorithm did you use, and why?
- What is the running time of your algorithm?
- If you chose to throw away edges, how did you determine $k(n)$, and how effective was this approach?
- Can you give a rough explanation for your results? (The limiting behavior as n grows large can be proven rigorously, but it is very difficult; you need not attempt to prove any exact result.)
- Did you have any interesting experiences with the random number generator? Do you trust it?