Chapter 37

Randomized Algorithms

By Sariel Har-Peled, September 30, 2009[1] Version: 1.0

This chapter include problems on randomized algorithms

37.1 Randomized algorithms

Exercise 37.1.1 (Find kth smallest number.) [20 Points]

This question asks you to to design and analyze a *randomized incremental* algorithm to select the kth smallest element from a given set of n elements (from a universe with a linear order).

In an *incremental* algorithm, the input consists of a sequence of elements $x_1, x_2, \ldots, x_n$. After any prefix $x_1, \ldots, x_{i-1}$ has been considered, the algorithm has computed the kth smallest element in $x_1, \ldots, x_{i-1}$ (which is undefined if $i \leq k$), or if appropriate, some other invariant from which the kth smallest element could be determined. This invariant is updated as the next element x_i is considered.

Any incremental algorithm can be *randomized* by first randomly permuting the input sequence, with each permutation equally likely.

1. **[5 Points]** Describe an incremental algorithm for computing the kth smallest element.

2. **[5 Points]** How many comparisons does your algorithm perform in the worst case?

3. **[10 Points]** What is the expected number (over all permutations) of comparisons performed by the randomized version of your algorithm? (Hint: When considering x_i, what is the probability that x_i is smaller than the kth smallest so far?) You should aim for a bound of at most $n + O(k \log(n/k))$. Revise (a) if necessary in order to achieve this.

Exercise 37.1.2 (Minimum Cut Festival) [20 Points]

1. Given a multigraph $G(V, E)$, show that an edge can be selected uniform at random from E in time $O(n)$, given access to a source of random bits.

2. For any $\alpha \geq 1$, define an α approximate cut in a multigraph G as any cut whose cardinality is within a multiplicative factor α of the cardinality of the min-cut in G. Determine the probability that a single iteration of the randomized algorithm for cuts will produce as output some α-approximate cut in G.

3. Using the analysis of the randomized min-cut algorithm, show that the number of distinct min-cuts in a multigraph G cannot exceed $n(n-1)/2$, where n is the number of vertices in G.

4. Formulate and prove a similar result of the number of α -approximate cuts in a multigraph G.

Exercise 37.1.3 (Adapt min-cut) [20 Points]

Consider adapting the min-cut algorithm to the problem of finding an *s-t min-cut* in an undirected graph. In this problem, we are given an undirected graph G together with two distinguished vertices s and t. An s-t min-cut is a set of edges whose removal disconnects s from t; we seek an edge set of minimum cardinality. As the algorithm proceeds, the vertex s may get amalgamated into a new vertex as the result of an edge being contracted; we call this vertex the s-vertex (initially s itself). Similarly, we have a t-vertex. As we run the contraction algorithm, we ensure that we never contract an edge between the s-vertex and the t-vertex.

1. **[10 Points]** Show that there are graphs in which the probability that this algorithm finds an s-t min-cut is exponentially small.

2. **[10 Points]** How large can the number of s-t min-cuts in an instance be?

Exercise 37.1.4 (Majority tree) [20 Points]

Consider a uniform rooted tree of height h (every leaf is at distance h from the root). The root, as well as any internal node, has 3 children. Each leaf has a boolean value associated with it. Each internal node returns the value returned by the majority of its children. The evaluation problem consists of determining the value of the root; at each step, an algorithm can choose one leaf whose value it wishes to read.

(a) Show that for any deterministic algorithm, there is an instance (a set of boolean values for the leaves) that forces it to read all $n = 3^h$ leaves. (hint: Consider an adversary argument, where you provide the algorithm with the minimal amount of information as it request bits from you. In particular, one can devise such an adversary algorithm.).

(b) Consider the recursive randomized algorithm that evaluates two subtrees of the root chosen at random. If the values returned disagree, it proceeds to evaluate the third sub-tree. If they agree, it returns the value they agree on. Show the expected number of leaves read by the algorithm on any instance is at most $n^{0.9}$.

Exercise 37.1.5 (Hashing to Victory) [20 Points]

In this question we will investigate the construction of hash table for a set W, where W is static, provided in advance, and we care only for search operations.

1. **[2 Points]** Let $U = \{1, \ldots, m\}$, and $p = m + 1$ is a prime.

   Let $W \subseteq U$, such that $n = |W|$, and s an integer number larger than n. Let $g_k(x, s) = (kx \mod p) \mod s$.

   Let $\beta(k, j, s) = \left|\left\{x \;\middle|\; x \in W, g_k(x, s) = j\right\}\right|$. Prove that

$$\sum_{k=1}^{p-1} \sum_{j=1}^{s} \binom{\beta(k, j, s)}{2} < \frac{(p-1)n^2}{s}.$$

2. **[2 Points]** Prove that there exists $k \in U$, such that

$$\sum_{j=1}^{s} \binom{\beta(k, j, s)}{2} < \frac{n^2}{s}.$$

3. **[2 Points]** Prove that $\sum_{j=1}^{n} \beta(k, j, n) = |W| = n$.

4. **[3 Points]** Prove that there exists a $k \in U$ such that $\sum_{j=1}^{n} (\beta(k, j, n))^2 < 3n$.

5. **[3 Points]** Prove that there exists a $k' \in U$, such that the function $h(x) = (k'x \mod p) \mod n^2$ is one-to-one when restricted to W.

6. **[3 Points]** Conclude, that one can construct a hash-table for W, of $O(n^2)$, such that there are no collisions, and a search operation can be performed in $O(1)$ time (note that the time here is worst case, also note that the construction time here is quite bad - ignore it).

7. **[3 Points]** Using (d) and (f), conclude that one can build a two-level hash-table that uses $O(n)$ space, and perform a lookup operation in $O(1)$ time (worst case).

Exercise 37.1.6 (Sorting Random Numbers) [20 Points]

Suppose we pick a real number x_i at random (uniformly) from the unit interval, for $i = 1, \ldots, n$.

1. **[5 Points]** Describe an algorithm with an expected linear running time that sorts $x_1, \ldots, x_n$.

To make this question more interesting, assume that we are going to use some standard sorting algorithm instead (say merge sort), which compares the numbers directly. The binary representation of each x_i can be generated as a potentially infinite series of bits that are the outcome of unbiased coin flips. The idea is to generate only as many bits in this sequence as is necessary for resolving comparisons between different numbers as we sort them. Suppose we have only generated some prefixes of the binary representations of the numbers. Now, when comparing two numbers x_i and x_j, if their current partial binary representation can resolve the comparison, then we are done. Otherwise, the have the same partial binary representations (upto the length of the shorter of the two) and we keep generating more bits for each until they first differ.

1. **[10 Points]** Compute a tight upper bound on the expected number of coin flips or random bits needed for a single comparison.

2. **[5 Points]** Generating bits one at a time like this is probably a bad idea in practice. Give a more practical scheme that generates the numbers in advance, using a small number of random bits, given an upper bound n on the input size. Describe a scheme that works correctly with probability $\geq 1 - n^{-c}$, where c is a prespecified constant.