CS 161 Fall 2017: Section 4/Midterm Review Solutions

Randomized Algorithms: Matrix Multiplication Verification

We have three matrices, A, B and C, each of size $n \times n$. We want to verify if $AB = C$. We could do so by multiplying and verifying them in $O(n^3)$ (or faster using some divide-and-conquer tricks!), but we want to develop a faster randomized algorithm here.

(a) Let $\mathbf{v}$ be a vector with n elements, each 0 or 1 with probability $\frac{1}{2}$. If we have another nonzero vector $\mathbf{u}$, show that $\mathbf{uv} = 0$ with probability $\leq \frac{1}{2}$. (Hint: express $\mathbf{uv}$ as $\mathbf{u}_k\mathbf{v}_k + x$ where $\mathbf{u}_k \neq 0$.)

Using the hint, we set $z = (\mathbf{uv})_k = \mathbf{u}_k\mathbf{v}_k + x$ where $\mathbf{u}_k \neq 0$. We know we can find some $\mathbf{u}_k \neq 0$ because $\mathbf{u}$ is a nonzero vector. Now, we break into separate cases based on $\mathbf{v}_k$, which has a $\frac{1}{2}$ chance of being 0.

$$P[z = 0] = P[z = 0 \mid x = 0]P[x = 0] + P[z = 0 \mid x \neq 0]P[x \neq 0]$$

We know that

$$P[z = 0 \mid x = 0] = P[v_k = 0] = \frac{1}{2}$$

On the next term,

$$P[z = 0 \mid x \neq 0] = P[v_k = 1 \wedge \mathbf{u}_k = -x] \leq P[v_k = 1] = \frac{1}{2}$$

(b) Note that the above extends to a matrix M: if M is nonzero, we have that $M\mathbf{v} = 0$ with probability $\leq \frac{1}{2}$. Given this, show that $P[AB\mathbf{v} = C\mathbf{v}] \leq \frac{1}{2}$

Set $M = AB - C$.

(c) Using the previous parts, develop a randomized algorithm to check if $AB = C$.

Generate random $\mathbf{v}$, and check if $AB\mathbf{v} = C\mathbf{v}$. If not, we know that $AB \neq C$. Otherwise, there is a possibility of equality. We repeat with a new $\mathbf{v}$ for k times.

(d) Is your above algorithm a Las Vegas or Monte Carlo algorithm?

Monte Carlo. This algorithm can return an incorrect solution, regardless of how many times we iterate.

(e) What is the expected runtime of your algorithm? What is the probability of returning a correct answer?

For k iterations, this will take $O(kn^2)$ time. ($O(n^2)$ for each iteration to do the matrix vector products.) This algorithm returns a correct answer with probability $1 - 2^{-k}$.

Selection Potpurri

(a) You are implementing the SELECT, and your friend is writing the pivot selection portion using median of medians. However, it turns out that your partner left some bugs, so the only guarantee you have is

that pivot selection will return some element of the array. Given that you implemented your portion of SELECT correctly, will your program still work? What is the worst-case runtime?

SELECT should still work because random pivots also work. However, the worst-case runtime becomes $O(n^2)$, as we may only reduce our problem size by 1 at each recursive call.

(b) Instead of using median of medians, you decide to use the mean μ of the array. (To guarantee an element in the array, assume that this is the smallest element in the array $\geq \mu$.) What is the worst-case runtime of SELECT?

$O(n^2)$. If we have an array of the form $[1, 2, 4, 8 \dots 2^n]$ and then shuffle it, the mean is essentially at the largest element of the array, which would only reduce the problem size by 1 at each recursive step.

(c) You have a set S of n integers, and you're given integer $1 \leq k < n$. Design an algorithm to verify that

$$\forall T \subset S, |T| = k, \sum_{t \in T} t \geq k$$

In other words, for every subset of S of size k, verify that the sum of elements in that subset is $\geq k$.

Run SELECT with median of medians to find the kth largest element. Then, partition the array such that elements smaller than k appear to the left of k. We then check the sum of these elements in linear time.

Number Distances

We are given an unsorted array A with n numbers between 1 and M, where M is a large (but constant) positive integer. We want to find if there exist two elements of the array that are within T of one another.

(a) Design a simple algorithm that solves this in $O(n^2)$.

Compare all pairs of elements to see if they are within T. There are $O(n^2)$ pairs.

(b) Design an algorithm that solves this in $O(n \log n)$.

Sort the elements in $O(n \log n)$. Then, compare neighboring elements in linear time.

(c) How could you solve this in $O(n)$? (Hint: modify bucket sort!)

Create buckets of size $T + 1$ (divide up the range up to M with these buckets). Drop each element of the array into its corresponding bucket, meaning that an element with value T would go into the first bucket, while an element with value $T + 2$ would go into the second. Then, iterate through each bucket with the following cases:

(a) Bucket with no elements: continue onto next bucket

(b) Bucket with multiple elements: we have found two elements that are within T of one another.

(c) Bucket has exactly one element: compare the value in this bucket to values in neighboring buckets, if they exist.