6.046
PROBLEM SET 4

JOHN WANG

Collaborators:

1. Problem 5-1: High Probability Bounds on Randomized Select

1.1. Problem A.

Problem 1.1. *Let b be a real number such that $1/2 < b < 1$ and let A_i be the array in the ith recursion. Define a bad pivot choice in the ith recursion as the one that results in $|A_{i+1}| > b|A_i|$. Give a lower bound on the probability of having k bad pivot choices in a row.*

Solution First, let us imagine the A_i in sorted order. In order for a pivot to be bad so that $|A_{i+1}| > b|A_i|$, then we need either the smaller or larger secondary array to be larger than $b|A_i|$. This occurs if we pick a pivot with rank smaller than bn or with rank greater than $n-(bn)$. If we select a pivot uniformly at random, then there is a $(1-b)$ chance of selecting a pivot in one of these regions. Since there are two regions, then we have a probability of $2(1-b)$ of selecting a bad pivot on the ith trial. Since each one of these trials is independent, the probability of selecting k bad pivots in a row is then at least $(2(1-b))^k$. □

1.2. Problem B.

Problem 1.2. *If our initial array size is n, then after one bad pivot choice, the next array is of size at least bn. Recall that the running time at each recursive call requires time equal to at least the size of the array. Give a precise lower bound on the total running time of k recursive calls to* `Randomized-Select` *if in every recursive call we choose a bad pivot.*

Solution Let T be the running time of k recursive calls. We know that T is bounded by the following:

$$T > \sum_{i=0}^{k-1} nb^i \tag{1.1}$$

This follows because at the ith step in the recursion, we have an array of size at least nb^i. Therefore, since the running time at each recursive call requires time equal at least to the size of the array, we must have T be greater than the sum of all the k recursive calls. We can evaluate this sum to find:

$$T > n\sum_{i=0}^{k-1} b^i \tag{1.2}$$

$$T > n\left(\frac{1-b^{k-1}}{1-b}\right) \tag{1.3}$$

Where we have used the formula for a partial geometric series. This gives us a lower bound on the running time of k recursive calls if we choose a bad pivot each time. □

1.3. Problem C.

Problem 1.3. *Use a proof by contradiction to disprove the following statement: Let $T(n)$ be the running time of* `Randomized-Select` *on an input of size n. Then there exist integers $n_0, c \geq 1$ such that for all $n \geq n_0$, $P(T(n) > cn) \leq 1/n$.*

Solution Suppose this statement were true. Then select n_0, c such that they satisfy the conditions of the statement. Now let us examine the event E defined as the event where `Randomized-Select` chooses n

bad pivots consecutively. Now let us choose $b = 1 - \frac{1}{2c}$ for a bad pivot. We know that the running time is bounded by the following:

$$\begin{aligned}
T(E) &> n\left(\frac{1-b^{n-1}}{1-b}\right) && (1.4)\\
&= n\left(\frac{1-(1-\frac{1}{2c})^{n-1}}{\frac{1}{2c}}\right) && (1.5)\\
&= 2nc\left(1-\left(\frac{2c-1}{2c}\right)^{n-1}\right) && (1.6)
\end{aligned}$$

Now let us assume that $T(E) > cn$ for some n. Then we must have the following hold:

$$\begin{aligned}
2nc\left(1-\left(\frac{2c-1}{2c}\right)^{n-1}\right) &> cn && (1.7)\\
1-\left(\frac{2c-1}{2c}\right)^{n-1} &> \frac{1}{2} && (1.8)\\
\left(\frac{2c-1}{2c}\right)^{n-1} &< \frac{1}{2} && (1.9)
\end{aligned}$$

Since we know that $\frac{2c-1}{2c} < 1$ (as $c > 1$ by assumption), we can take the log base $\frac{2c-1}{2c}$ of both sides but must switch the inequality. This yields:

$$\begin{aligned}
n-1 &> \log_{\frac{2c-1}{2c}}\left(\frac{1}{2}\right) && (1.10)\\
n &> 1-\log_{\frac{2c-1}{2c}}(2) && (1.11)
\end{aligned}$$

However, we know that the following holds true:

$$0 < \frac{1}{c} < \frac{2c-1}{2c} < 1 \qquad (1.12)$$

This means that $\log_{\frac{2c-1}{2c}}(x) < \log_{\frac{1}{c}}(x) < 0$ for $x > 1$. Therefore, we can give another bound for n with:

$$n > 1-\log_{\frac{1}{c}}(2) \qquad (1.13)$$

This means that if $T(E) > cn$, we must have $n > 1 - \log_{\frac{1}{c}}(2)$. Notice that since $\frac{1}{c} < 1$, we must have $\log_{\frac{1}{c}}(2) < 0$, which means that $n > 1$. Now, if this is the case, let us examine the probability that we obtain n bad recursive calls. This can be given by our solution to problem A:

$$\begin{aligned}
P[E] &> (2(1-b))^n && (1.14)\\
&> \left(2\left(1-\left(1-\frac{1}{2c}\right)\right)\right)^{1-\log_{\frac{1}{c}}(2)} && (1.15)\\
&= \left(\frac{1}{c}\right)^{1-\log_{\frac{1}{c}}(2)} && (1.16)\\
&= \left(\frac{1}{c}\right)\frac{1}{2} && (1.17)\\
&= \frac{1}{2c} && (1.18)
\end{aligned}$$

Therefore, we know that $P[E] > 1/(2c)$. However, notice that this is invariant with n. Therefore, if we select n large enough such that $n > 1 - \log_{\frac{1}{c}}(2)$ and such that $1/n > 1/(2c)$, we we see that $P[E] \not\leq 1/n$. Therefore, we can select $n > 2c+(1-\log_{\frac{1}{c}}(2))+n_0$ such that we are sure that $n > n_0$, $\frac{1}{n} > \frac{1}{2c}$, and $T[E] > cn$. Since we can choose n arbitrarily large to satisfy this, we have found an n such that $P(T(n) > cn) \not\leq \frac{1}{n}$, which is a contradiction of our statement. This completes the proof. $\square$

2. Problem 5-2: Random Vectors and Matrices

2.1. **Problem A.**

Problem 2.1. *You are given a non-zero vector $\vec{u} \in \mathrm{Z}_p^n$ and some number $c \in \mathrm{Z}_p$. Prove that if another vector $\vec{v} \in \mathrm{Z}_p^n$ has each element chosen independently and uniformly at random from Z_p, then the probability that $\vec{v} \cdot \vec{u} = c$ is $1/p$.*

Solution First, we shall show it works for the base case of $n = 1$, then use induction on n. Therefore, we first will show that if $\vec{u} \in \mathrm{Z}_p^1$ is an integer $u_1 \in \mathrm{Z}_p$, and we pick another number $v_1 \in \mathrm{Z}_p$ uniformly at random, the the probability that $u_1 \cdot v_1 \equiv c \pmod p$ is $1/p$. To show this, we prove the following lemma:

Lemma 2.1. *For any number $u \in \mathrm{Z}_p$, there exists a unique integer $\bar{u}_c \in \mathrm{Z}_p$ such that $u \cdot \bar{u}_c \equiv c \pmod p$.*

Proof. First we will show existence. Let g be a primitive root modulo p, which exists because p is a prime. Next, we know that $1, g, g^2, \ldots, g^{p-2}$ is a reordering of $1, 2, 3, \ldots, p-1$ modulo p by the definition of primitive root. We can express $u = g^k$ for some $k \in \{1, 2, \ldots, p-2\}$ and $c = g^m$ for some $m \in \{1, 2, \ldots, p-2\}$. Now if $m > k$, we can simply let $\bar{u}_c = g^{m-k}$ so that $u \cdot \bar{u}_c = g^k g^{m-k} = g^m \equiv c \pmod m$. If $m = k$, we let $\bar{u}_c = 1$. If $m < k$, then we choose $\bar{u}_c = g^{m+(p-1)-k}$ so that $u \cdot \bar{u}_c = g^k g^{m+(p-1)-k} = g^{m+(p-1)} \equiv g^m \pmod p$ so that $u \cdot \bar{u}_c \equiv c \pmod p$. This proves existence for all cases.

We will now show uniqueness. Suppose there are two such numbers $\bar{u}_{c,1}$ and $\bar{u}_{c,2}$. Both have the property that $u \cdot \bar{u}_{c,1} \equiv c \pmod p$ and $u \cdot \bar{u}_{c,2} \equiv c \pmod p$. This means that $u \cdot \bar{u}_{c,1} - u \cdot \bar{u}_{c,2} \equiv 0 \pmod p$, which implies $u(\bar{u}_{c,1} - \bar{u}_{c,2}) \equiv 0 \pmod p$. By the definition of modulus, we know that $p | u(\bar{u}_{c,1} - \bar{u}_{c,2})$. However, since u and p are coprime since p is a prime, we know that $p \nmid u$. This means that $p | (\bar{u}_{c,1} - \bar{u}_{c,2})$. However, this implies that $\bar{u}_{c,1} \equiv \bar{u}_{c,2} \pmod p$, which shows uniqueness. □

Now that we have shown there exists a unique integer$\bar{u}_c$ modulo p such that $u \cdot \bar{u}_c = c$, we know there is probability $1/p$ of selecting that integer uniformly at random from Z_p. This proves the base case for the induction. Next, we will assume that our hypothesis holds up to vectors of length n. We must show it holds for vectors of length $n + 1$.

Consider the vector $\vec{u}$ of size $n+1$. It consists of a first element v_1, and another n elements which can be thought of as a vector $\vec{u}_n$. The probability that a vector $\vec{v}$ chosen uniformly at random has $\vec{u} \cdot \vec{v} = c$ is the probability that $v_1 \cdot u_1 + \vec{u}_n \cdot \vec{v}_n = c$. Now notice that if $\vec{u}_n \cdot \vec{v}_n \equiv a \pmod p$, then we must have $v_1 \cdot u_1 \equiv c - a \pmod p$. It is clear that there is exactly one such element $c - a \in \mathrm{Z}_p$, which can be chosen with probability $1/p$. Therefore, the probability that $\vec{u} \cdot \vec{v} = c$ is given by:

$$\sum_{i=0}^{p-1} \left(\frac{1}{p}\right)\left(\frac{1}{p}\right) = \sum_{i=0}^{p-1} \left(\frac{1}{p}\right)^2 = \frac{1}{p} \tag{2.2}$$

The equation follows because there are p possibilities for the result of $\vec{u}_n \cdot \vec{v}_n = a$, namely $\{0, 1, \ldots, p-1\}$. Each of these have a probability of $1/p$ of occuring by our inductive hypothesis, and $v_1 \cdot v_2 = c - a$ also has a probability $1/p$ of occuring. This shows that the probability that $\vec{u} \cdot \vec{v} = c$ is $1/p$, which proves the theorem for arbitrary n. □

2.2. Problem B.

Problem 2.2. *You are given two vectors $\vec{x}, \vec{y} \in \mathrm{Z}_p^n$ such that $\vec{x} \neq \vec{y}$. Using the result from part (a), show that if $\vec{v}$ is a random vector as before, then $P[\vec{v} \cdot \vec{x} = \vec{v} \cdot \vec{y}] = 1/p$.*

Solution Since v is a random vector, we know that the probability that $\vec{v} \cdot \vec{x} = c_x$ is $1/p$ for all $c_x \in \{0, 1, \ldots, p-1\}$. Moreover, the probability that $\vec{v} \cdot \vec{y} = c_y$ is $1/p$ for all $c_y \in \{0, 1, \ldots, p-1\}$. Now, we would like to find the probability that $c_x = c_y$. This is just the probability across i in the set $\{0, 1, 2, \ldots, p-1\}$ that both c_x and c_y are equal to i. This is given by:

$$\sum_{i=0}^{p-1} \left(\frac{1}{p}\right)^2 = \frac{1}{p} \tag{2.3}$$

This follows from problem (a) because there is a $1/p$ probability that $c_x = i$ and also a $1/p$ probability that $c_y = i$. Since c_x and c_y are independent, we can multiply probabilities, and sum across all $i \in \{0, 1, \ldots, p-1\}$. This shows that $P[\vec{v} \cdot \vec{x} = \vec{v} \cdot \vec{y}] = 1/p$. □

2.3. Problem C.

Problem 2.3. *Using the result from part (b), show that if A is an $m \times n$ matrix with each element chosen independently at random from Z_p, then $P[A\vec{x} = A\vec{y}] = 1/p^m$.*

Solution First, let us denote $A = [\vec{A}_1, \vec{A}_2, \dots, \vec{A}_m]^T$ as a column vector consisting of $\vec{A}_i$, which is the ith row in A. We know from linear algebra that multiplication of matrices can simply be thought of as multiplication of individual row vectors in A with $\vec{x}$. Thus, we know:

$$A\vec{x} = \begin{bmatrix} \vec{A}_1 \\ \vec{A}_2 \\ \vdots \\ \vec{A}_m \end{bmatrix} \vec{x} = \begin{bmatrix} \vec{A}_1 \cdot \vec{x} \\ \vec{A}_2 \cdot \vec{x} \\ \vdots \\ \vec{A}_m \cdot \vec{x} \end{bmatrix} \tag{2.4}$$

Thus, in order for $A \cdot \vec{x} = A \cdot \vec{y}$, we must have $\vec{A}_i \cdot \vec{x} = \vec{A}_i \cdot \vec{y}$ for all $i \in \{1, 2, \dots, m\}$. We know from before that $P[\vec{A}_i \cdot \vec{x} = \vec{A}_i \cdot \vec{y}] = 1/p$, since $\vec{A}_i$ are all vectors chosen uniformly at random. Moreover, since we know that the vectors $\vec{A}_i$s are chosen independently by assumption, we can multiply the probabilities together. Since there are m vectors that must be equal, we find that $P[A\vec{x} = A\vec{y}] = (1/p)^m = (1/p^m)$. □

2.4. Problem D.

Problem 2.4. *Conclude that the family $\mathcal{H}$ of all such functions $h_A(\vec{x}) = A\vec{x}$ where A is an $m \times n$ matrix with elements in $\mathbb{Z}_p$ is universal.*

Solution In order for a hash family $\mathcal{H}$ to be universal, we must have for $\vec{x} \neq \vec{y}$ that $P[h_A(\vec{x}) = h_A(\vec{y})] < 1/n$ for all $h \in \mathcal{H}$ where n is the number of different possible vectors one can hash to. In our case, we know that the result of $A\vec{x}$ has m rows, each with p different possibilities. Thus, there are a total of mp different possible vectors in our hash function's range. Therefore, we must show that $P[h_A(\vec{x}) = h_A(\vec{y})] < 1/(mp)$. However, we know from part (c) that $P[h_A(\vec{x}) = h_A(\vec{y})] = P[A\vec{x} = A\vec{y}] = 1/p^m$. Thus, we must have $1/p^m < 1/(mp)$. This occurs when $p^m > mp$, or equivalently, when $p^{m-1} > m$.

We note that if $p = 2$, then $m > 4$ will result in $p^{m-1} = 2^3 = 8 > 4$. This is because $m = O(p^{m-1})$ and that $\lim_{m\to\infty} \frac{m}{p^{m-1}} = 0$. Since p^{m-1} is an increasing function in p, we know that $p^{m-1} > m$ for all $p \geq 2$ if $m > 4$. Therefore, it is only necessary to choose $m \geq 4$ and any prime $p \geq 2$ in order for $\mathcal{H}$ to be a universal hash family. □

2.5. Problem E.

Problem 2.5. *Using the resut from part (a), devise a randomized algorithm to determine if $C = B \cdot A$. Show that your algorithm is correct with probability at least 90%.*

Solution First, we know that $A = (m \times n)$, $B = (k \times m)$, and $C = (k \times n)$ are the dimensions of the matrices. Knowing this, we can derive an algorithm as follows. We will create a random vector $\vec{x}$ of dimension n by selecting $x_i \in \mathbb{Z}_2$ uniformly at random and independently. Then, we will compute $B \cdot A \cdot \vec{x}$ by first computing $A \cdot \vec{x}$, then computing $B \cdot (A \cdot \vec{x})$. We will also compute $C \cdot \vec{x}$. If $B \cdot A \cdot \vec{x} \neq C \cdot \vec{x}$, then return false. Otherwise, continue this procedure four times. If the result passes all four times, return true.

First, we will analyze running time. Let us first analyze a single loop through the algorithm. Generating a random $n \times 1$ vector requires $O(n)$ time. Multiplying A, which is an $(m \times n)$ matrix, by $\vec{x}$ requires $O(mn)$ time, because each dot product requires n multiplications and additions, and we must do this m times. Thus, $A \cdot \vec{x}$ results in an $(m \times 1)$ sized vector. Computing $B \cdot (A \cdot \vec{x})$ will therefore require $O(km)$ time, since B is a $(k \times m)$ sized matrix. Multiplying B with an $(m \times 1)$ sized vector requires m multiplications and additions for each entry in the result. Since there are k entires, the running time is $O(km)$. Next, computing $C \cdot \vec{x}$ requires $O(kn)$ time because C is a $(k \times n)$ matrix. Checking $B \cdot A \cdot \vec{x} = C \cdot \vec{x}$ requires $O(k)$ time because both the right hand side and left hand side are $(k \times 1)$ sized vectors. The total running time per loop is therefore $O(n + mn + km + kn + k) = O(mn + km + kn)$. Going through this loop at maximum four times only adds a constant factor to the run time. Therefore, the algorithm runs in $O(mn + km + kn)$ time. Thus, if C is an $(n \times n)$ matrix, this algorithm runs in $O(n^2)$ time, which is very fast.

Now we shall show that the algorithm is correct at least 90% of the time. First, if $B \cdot A = C$, then we see the algorithm will always be correct. This is because $B \cdot A \cdot \vec{x}$ will always be equal to $C \cdot \vec{x}$ in this case, and therefore the algorithm will go through four loops and return true. Next, if $B \cdot A \neq C$, we will show that the algorithm is correct at least 90% of the time, which will show it is always correct at least 90% of the time.

Let $D = B \cdot A$ and assume that $D \neq C$. We know from part B that $P[\vec{x} \cdot \vec{u} = \vec{x} \cdot \vec{v}] \leq 1/p$ where $\vec{x}$ is a random vector and $\vec{u} \neq \vec{v}$. Then the probability that the ith element in the output vector of $D \cdot \vec{x}$ differs from the ith element in the output vector of $C \cdot \vec{x}$ is $1/p$. The probability that any element differs is therefore $1 - (1 - 1/p)^k = 1 - (\frac{p-1}{p})^k$. Since $p = 2$, we know the probability that any element differs, and thus that the algorithm is incorrect, is given by $1 - (\frac{1}{2})^k$. Since k is the dimension of a matrix, and we know that

$k \geq 1$, we know that this probability is bounded by $\frac{1}{2}$. The probability that we run the algorithm 4 times, and it says that $D \cdot \vec{x} = C \cdot \vec{x}$ for all four randomized vectors is bounded above by $(\frac{1}{2})^4 = 1/16$. Thus, the probability that the algorithm returns the correct answer is bounded below by $1 - (\frac{1}{2})^4 = 15/16 \approx 0.93$. This shows that the algorithm is correct at least 90% of the time.

Finally, we shall show an example of the algorithm at work. Let us pick the following matrices:

$$B = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{2.5}$$

We see immediately that $B \cdot A \neq C$ because the bottom left element of $B \cdot A$ is 0, whereas the bottom left element of C is 1. Let us run our algorithm to check this. First, we pick a random $n \times 1$ vector, where $n = 2$ in this case. Let us pick $\vec{x} = [0, 1]$. Then we first find $A \cdot \vec{x}$:

$$A \cdot \vec{x} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.6}$$

Then we compute $B \cdot (A \cdot \vec{x})$:

$$B \cdot (A \cdot \vec{x}) = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.7}$$

Now we copmare this result of $C \cdot \vec{x}$:

$$C \cdot \vec{x} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.8}$$

The algorithm passes the first test. Therefore, we go onto the second loop. We pick a new random vector $\vec{x} = [1, 1]$ and check if $B \cdot (A \cdot \vec{x}) = C \cdot \vec{x}$ again. This time, we find that:

$$B \cdot (A \cdot \vec{x}) = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{2.9}$$

$$C \cdot \vec{x} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \tag{2.10}$$

Since $B \cdot (A \cdot \vec{x}) \neq C \cdot \vec{x}$, the algorithm returns false, and it would be correct in this case. $\square$