CS 170 HW 13

Due **2021-12-08, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

2 One-Sided Error and Las Vegas Algorithms

An RP algorithm is a randomized algorithm that runs in polynomial time and always gives the correct answer when the correct answer is 'NO', but only gives the correct answer with probability greater than 1/2 when the correct answer is 'YES'.

(a) Prove that every problem in RP is in NP (i.e., show that $RP \subseteq NP$). *Hint: it may be helpful to view a randomized algorithm $R(x)$ as a deterministic algorithm $A(x, r)$ where x is the input and r is the result of the 'coin flips' which the algorithm uses for its randomness.*

(b) In lecture, we saw an example of a *Las Vegas Algorithm*: a random algorithm which always gives the right solution, but whose runtime is random. A ZPP algorithm is a Las Vegas algorithm which runs in expected polynomial time (ZPP stands for Zero-Error Probabilistic Polytime). Prove that if a problem has a ZPP algorithm, then it has an RP algorithm. *Hint: Use Markov's inequality.*

Solution:

(a) Assume we have some problem that is in RP and let A be an algorithm that solves this problem. If A is given an input x such that the correct answer given input x is 'YES', then A will return 'YES' will probability greater than 1/2. We want to use A to construct an NP algorithm for x.

Randomized algorithms run like regular algorithms, but will occasionally use random coin-flips to make decisions. For any randomized algorithm B, we can build a deterministic algorithm B' that takes the outcome of those coin-flips beforehand and runs the same computation as B. Let A' be that deterministic algorithm for A.

Since A is an RP algorithm, there must be a poly-size sequence of coin-flip outcomes $\vec{b} = b_1, \dots, b_k \in \{0, 1\}^k$ such that A returns 'YES' given input x and outcomes $\vec{b}$. If we treat $\vec{b}$ as the 'solution' to x, we can see that A' is an NP algorithm for the given problem. Thus every problem in RP is in NP.

(b) Let A be any ZPP algorithm and assume A runs in expected time at most n^k for some constant k. We construct an RP algorithm A' that given input x of size n, does the following:

- Run A for $2n^k$ steps, then stop.
- If A returns 'YES' in that time, return 'YES'
- If A returns 'NO' in that time, return 'NO'
- Otherwise, return 'NO'

How do we know A' is an RP algorithm? First of all, if the correct answer on input x is 'NO', then either A will stop and A' will say 'NO' or A will not stop and A' will still say 'NO'. So A' will always give the right answer when the right answer is 'NO'.

No assume the correct answer is 'YES'. If A stops in time, A' will give the correct answer. But if A does not stop in time, then A' will give the wrong answer. So we want to bound the probability that A does not stop in time.

Let X be a random variable representing the number of steps A will take on input x. We know that $E[X] \leq n^k$. By Markov's inequality this gives us

$$Pr[X \geq a] \leq \frac{n^k}{a}$$

If we set $a = 2n^k$, this gives us

$$Pr[X \geq 2n^k] \leq \frac{n^k}{2n^k} = \frac{1}{2}$$

Thus if the correct answer on input x is 'YES', A' will give the correct answer with probability greater than 1/2.

So A' is an RP algorithm.

3 QuickSelect

Let A be an array of n integers, and let k be an integer. A and k are given as input. Let $\boldsymbol{X}_{i,j}$ be an indicator random variable for the event that the i-th smallest number is ever compared with the j-th smallest in $\mathsf{QuickSelect}(A, k)$.

(a) Write an exact expression for $\mathbf{E}[\boldsymbol{X}_{i,j}]$. *Hint: think about what elements will ever be chosen as pivots.*

Solution:

Since $\boldsymbol{X}_{i,j}$ is defined the way it is, we want to essentially determine what is the probability that the i-th smallest number is ever compared with the j-th smallest number.

We notice that where k is with respect to i and j actually affects our probability because in the algorithm of QuickSelect we only keep the side which contains k. This means i and j are ever compared if and only if one is chosen as a pivot and the other has not yet been discarded. So, i and j are not compared if and only if a different number in the range $[\min(k, i), \max(j, k)]$ is chosen as a pivot first; if a different number is chosen first, at least one of i and j will be discarded. Notice how there are $\max(j, k) - \min(k, i) + 1$ elements in this range that could be chosen as pivots.

Without loss of generality, let $i < j$ and assume the list is $1, 2, \ldots, n$. We will break up

the indicator into 3 cases: $k \geq \{i, j\}$, $k \leq \{i, j\}$, and $i < k < j$.
If $k \geq i, j$, then $\max(j, k) = k$ so the probability is

$$\frac{2}{k - i + 1}.$$

If $k \leq i, j$, then $\min(i, k) = k$ so the probability is

$$\frac{2}{j - k + 1}.$$

If $i < k < j$, then $\max(j, k) = j$ and $\min(i, k) = i$ so the probability is

$$\frac{2}{j - i + 1}.$$

(b) Show that the expected runtime of QuickSelect(A, k) is $O(n)$. *Hint: use part (a) to bound the expected number of comparisons done by the algorithm.*

Solution:

Our approach to computing the expected number of comparisons will be to upper bound

$$\mathbf{E}\left[\sum_{i<j} \boldsymbol{X}_{i,j}\right] = \sum_{i<j} \mathbf{E}\boldsymbol{X}_{i,j}.$$

Since our indicator is divided into three cases, let's also divide our expectation into three separate sums by linearity of expectations. We can simplify each case separately and then combine them together in the end.

Case 1 $k \geq i, j$:
The expectation of comparisons of pairs of (i, j) such that $k \geq j$ is:

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \boldsymbol{X}_{i,j}$$

Simplifying we get:

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \boldsymbol{X}_{i,j} = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{2}{k - i + 1} = \sum_{i=1}^{k-1} \frac{2(k - i)}{k - i + 1} \leq 2(k - 1) \leq 2n$$

Case 2 $k \leq i, j$:
This is similar to case 1.

$$\sum_{j=k+1}^{n} \sum_{i=k}^{j-1}$$

Simplifying we get:

$$\sum_{j=k+1}^{n} \sum_{i=k}^{j-1} \boldsymbol{X}_{i,j} = \sum_{j=k+1}^{n} \sum_{i=k}^{j-1} \frac{2}{j - k + 1} = \sum_{j=k+1}^{n} \frac{2(j - k)}{j - k + 1} \leq 2(n - k) \leq 2n$$

Case 3 $i < k < j$:
We will break this case up into length, specifically $d := j - i$. We notice that there are at most $j - i - 1$ pairs of (i, j) pairs which has the same distance. Thus we get the summation Thus our expectation becomes

$$\sum_{d=2}^{n} \sum_{i=\max(1,k+1-d)}^{\max(k-1,1)} \boldsymbol{X}_{i,j}$$

Simplifying we get:

$$\sum_{d=2}^{n} \sum_{i=\max(1,k+1-d)}^{\max(k-1,1)} \boldsymbol{X}_{i,j} = \sum_{d=2}^{n} \sum_{i=\max(1,k+1-d)}^{\max(k-1,1)} \frac{2}{j-i+1} \leq \sum_{d=2}^{n} \frac{2(d-1)}{d+1} \leq 2n$$

Since all three summations are upperbounded by $2n$, we getour overall runtime to be upperbounded by $6n$ thus proving that $\mathsf{QuickSelect}(A, k)$ is $O(n)$.

4 Pairwise Independent Hashing

Let $\mathcal{H}$ be a class of hash functions in which each $h \in \mathcal{H}$ maps the universe $\mathcal{U}$ of keys to $\{0, 1, \ldots, m-1\}$. Recall that $\mathcal{H}$ is *universal* if for any $x \neq y \in \mathcal{U}$, $\Pr_{h\in\mathcal{H}}[h(x) = h(y)] \leq 1/m$.

We say that $\mathcal{H}$ is pairwise independent if, for every fixed pair (x, y) of keys where $x \neq y$, and for any h chosen uniformly at random from $\mathcal{H}$, the pair $(h(x), h(y))$ is equally likely to be any of the m^2 pairs of elements from $\{0, 1, \ldots, m-1\}$. (The probability is taken only over the random choice of the hash function.) As an example, it turns out that $h_{a,b}(x) = ax+b \bmod m$ is a pairwise independent hash family (when the random choice is over a and b and m is prime). We will not prove this here.

(a) Show that if $\mathcal{H}$ consists of strictly fewer than m^2 functions, it cannot be pairwise independent.

(b) Show that, if $\mathcal{H}$ is pairwise independent, then it is universal.

(c) Suppose that you choose a hash function $h \in \mathcal{H}$ uniformly at random. Your friend, who knows $\mathcal{H}$ but does not know which hash function you picked, tells you a key x, and you tell her $h(x)$. Can your friend tell you $y \neq x$ such that $h(x) = h(y)$ with probability greater than $1/m$ (over your choice of h) if:

(i) $\mathcal{H}$ is universal?

(ii) $\mathcal{H}$ is pairwise independent?

In each case, either give a choice of $\mathcal{H}$ which allows your friend to find a collision, or prove that they cannot for any choice of $\mathcal{H}$.

Solution:

(a) Pick any two elements $x, y \in \mathcal{U}$; in order for $\mathcal{H}$ to be pairwise independent, $(h(x), h(y))$ must take all m^2 possible values in $[m] \times [m]$ with equal probability $1/m^2$; but if there are fewer than m^2 functions in $\mathcal{H}$, there must (by the pigeonhole principle) be at least one pair $(a, b) \in [m] \times [m]$ which does not equal under $(h(x), h(y))$ under any choice of h, and so occurs with probability zero. This is a contradiction.

(b) If $\mathcal{H}$ is pairwise independent, then the tuple $(h(x), h(y))$ takes on all m^2 possible values with equal probability, and in m of these values we have $h(x) = h(y)$. So $\Pr[h(x) = h(y)] = m/m^2 = 1/m$ as desired.

(c) (i) We can construct a scenario where the adversary can force a collision. On a universe $\mathcal{U} = \{x, y, z\}$, consider the following family $\mathcal{H}$:

	x	y	z
h_1	0	0	1
h_2	1	0	1

$\mathcal{H}$ is a universal hash family: x and y collide with probability $1/2$, x and z collide with probability $1/2$, and y and z collide with probability $0 < 1/2$.

The adversary can determine whether we have selected h_1 or h_2 by giving us x to hash. If $h(x) = 0$, then we have chosen h_1, and the adversary then gives us y. Otherwise, if $h(x) = 1$, we have chosen h_2 and the adversary gives us z.

(ii) Since the pair $(h(x), h(y))$ is equally likely to be any of the m^2 possibilities, the marginal distributions of any two $h(x), h(y)$ are (pairwise) independent and uniformly distributed, so for any y your friend guesses, the chance $h(x) = h(y)$ is $1/m$.

5 Two-level Hashing

In lecture we saw a data structure for the static dictionary problem using $O(n^2)$ memory with $O(1)$ worst case query time (not just expected), and $O(n^2)$ expected preprocessing time. Here by preprocessing time, we mean the time it takes to create the data structure given the database (recall we picked $O(1)$ random hash functions in expectation from a universal hash family until we found one that causes no collisions, and for any hash function choice we could check whether there are any collisions and build the hash table in $O(n^2)$ time).

In this problem, we will develop a static dictionary data structure with $O(n)$ memory, $O(1)$ worst case query time, and $O(n)$ expected preprocessing time. The idea will be to use *two-level hashing*. As in lecture, we assume a database of size n with keys in the domain $[U] := \{0, \ldots, U - 1\}$.

(a) Consider picking a hash function $h : [U] \to [m]$ randomly from a 2-wise independent family. Let X_i then be the number of database keys x such that $h(x) = i$. Write down an exact expression for $\mathbb{E}(\sum_{i=0}^{m-1} X_i^2)$ in terms of only m and n.

(b) Give a static dictionary data structure with $O(n)$ memory, $O(1)$ worst case query time, and $O(n)$ expected preprocessing time. **Hint:** Pick a 'good' h using part (a) with $m = n$

and consider using universal hash functions $h_1, \ldots, h_m$ with $h_i : [U] \to [X_i^2]$ with the solution from class.

Solution:

(a) For any fixed i, let Z_j be an indicator random variable for the jth key in the dataset mapping to i under h. Then $X_i = \sum_{j=1}^n Z_j$, so that

$$\begin{aligned}
\mathbb{E}X_i^2 &= \mathbb{E}(\sum_j Z_j)^2 \\
&= \sum_j \mathbb{E}Z_j^2 + \sum_{j \neq j'} \mathbb{E}(Z_j Z_{j'}) \\
&= \sum_j \mathbb{E}Z_j + \sum_{j \neq j'} \mathbb{E}(Z_j Z_{j'}) \qquad \text{(since } Z_j^2 = Z_j\text{)} \\
&= \frac{n}{m} + \frac{n(n-1)}{m^2}
\end{aligned}$$

Therefore $\mathbb{E}(\sum_{i=1}^m X_i^2) = n + n(n-1)/m$.

(b) We set $m = n$ and use part (a), which says the expected sum of X_i^2 is less than $2n$. Thus the probability that it is bigger than $4n$ is less than $1/2$ by Markov's inequality. We repeatedly pick h's until we find an h which gives $\sum_i X_i^2 \leq 4n$, which means we have to try at most $2 = O(1)$ h's in expectation and thus such an h can be found in $O(n)$ time in expectation. Then for each i, for the items j with $h(j) = i$ we build a static dictionary on them using the quadratic space data structure from class. The expected preprocessing time is $O(n)$, and the query time is $O(1)$ since we just have to evaluate two hash functions and do a single array lookup.