Claim

CS 6150: HW4 – Graphs, Randomized algorithms

Submission date: Wednesday, Nov 10, 2021 (11:59 PM)

This assignment has 5 questions, for a total of 50 points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Question	Points	Score
QuickSelect	6	
Sampling from a stream	6	
Walking on a path	12	
Birthdays and applications	12	
Checking matrix multiplication	14	
Total:	50	

Instructions. For all problems in which you are asked to develop an algorithm, write down the pseudocode, along with a rough argument for correctness and an analysis of the running time (unless specified otherwise). Failure to do this may result in a penalty. If you are unsure how much detail to provide, please contact the instructors on Piazza.

Question 1: QuickSelect ..... **[6]**

Recall that given an (unsorted) array of **distinct** integers $A[0, 1, \ldots, n-1]$ and a parameter $1 \leq k \leq n$, the Selection problem asks to find the kth smallest entry of A. In class, we saw an algorithm that used a randomized implementation of ApproximateMedian, and showed that it leads to an $O(n)$ time algorithm. Let us now consider a different procedure, that is similar to QuickSort.

PROCEDURE QUICKSELECT(A, k)

1. If $|A| = 1$, return the only element
2. Select x from A uniformly at random
3. Form arrays B and C, containing the elements of A that are $< x$ and $> x$ respectively
4. If $|B| = (k-1)$, return x, else if $|B| < (k-1)$, return QUICKSELECT$(C, k - |B| - 1)$, else return QUICKSELECT(B, k)

Let $T(n)$ be defined as the **expected running time** of QuickSelect on an array of length n. Using the law of conditional expectation, prove that

$$T(n) \leq n + \sum_{j=1}^{n} \frac{1}{n} \max\{T(j-1), T(n-j)\}.$$

Using this along with $T(1) = 1$, prove that $T(n) \leq 4n$. Write down a description of all the events you use when you use conditional expectation.

(For the purposes of this question, you may ignore the additional $O(1)$ time for steps (1-2) and (4) of the procedure above.) [*Hint:* Follow the analysis for QuickSort seen in class, use induction.]

Side note. It is interesting to see that the constant term (the 4 in $4n$) above is much better than what we had for the deterministic algorithm we saw before. It turns out that there's a way of improving the constant further: instead of choosing x uniformly at random, we pick a small sample from the array and pick the sample median.

Solution.

We define X_n = running time of Quickselect on array of length n. We define events E_j = j'th smallest element is chosen as pivot at first step. We define events $T(n) = \mathbb{E}[X_n]$. Then we have

$$X_n | E_j \leq n + \max\{X_{n-j}, X_{j-1}\} \tag{1}$$

When we apply the law of conditional expectation,

$$\begin{aligned} \mathbb{E}[X_n] &\leq \sum_{j=1}^{n} \frac{1}{n}(n + \max\{\mathbb{E}[X_{n-j}], \mathbb{E}[X_{j-1}]\}) \\ T(n) &\leq n + \sum_{j=1}^{n} \frac{1}{n}(\max\{T(n-j), T(j-1)\}) \end{aligned} \tag{2}$$

We can prove that $T(n) \leq 4n$ by induction. We assume that it is true for number less than n.

$$\begin{aligned} T(n) &\leq n + \sum_{j=1}^{n} \frac{1}{n}(\max\{T(n-j), T(j-1)\}) \\ &\leq n + \sum_{j=n/2}^{n} \frac{2}{n} T(j) \\ &\leq n + 4 * \frac{2}{n} \frac{n/2+n}{2} n/2 \\ &\leq 4n \end{aligned} \tag{3}$$

Question 2: Sampling from a stream ..... **[6]**

If you have an array of n elements, sampling one at random is easy: you choose an index i at random in $\{0, 1, \ldots, n-1\}$ and return the ith element. Now suppose you have a *stream* of elements $a_1, a_2, \ldots$ (suppose they are all distinct for simplicity), and you don't know how many will arrive beforehand. Your goal is the following: at the end of the stream, you should output a random element from the stream.

The trivial algorithm is to store all the elements in an array (say a dynamic array), and in the end, output a random element. But it turns out that this can be done with very little memory.

Consider the following procedure: we maintain a special variable x, initialized to the first element of the array. At time t, upon seeing a_t, we set $x = a_t$ with probability $1/t$, otherwise we keep x unchanged.

Prove that in the end, the variable x stores a uniformly random sample from the stream. (In other words, if the stream had N elements, $\Pr[x = a_i] = 1/N$ for all i.)

[*Hint:* try doing a direct computation.]

Solution.

If we select a_i, it means that we set $x = a_t$ and we won't set x later.

$$Pr[x = a_i] = \frac{1}{t} \cdot \frac{t}{t+1} \cdot \frac{t+1}{t+2} \cdots \frac{N-1}{N} = \frac{1}{N} \tag{4}$$

Question 3: Walking on a path ..... **[12]**

Consider a path of length n comprising vertices $v_0, v_1, \ldots, v_n$. A particle starts at v_0 at $t = 0$, and in each time step, it moves to a **uniformly random neighbor** of the current vertex. Thus if it is at v_s at time t for some $s > 0$, then at time $(t+1)$, it moves to v_{s+1} or v_{s-1} with probability $1/2$ each. (If it is at v_0, the only neighbor is v_1 and so it moves there.) The particle gets "absorbed" once it reaches v_n and the walk stops.

Define $T(i)$ as the expected number of time steps taken by a particle *starting at* i to reach v_n. By definition, $T(n) = 0$.

(a) **[5]** Prove that $T(0) = 1 + T(1)$, and further, that for any $0 < s < n$, $T(s) = 1 + \frac{T(s-1)+T(s+1)}{2}$.

(b) **[5]** Use this to prove that $T(s) = (2s+1) + T(s+1)$ for all $0 \leq s < n$, and then find a closed form for $T(0)$. [*Hint:* Use induction.]

(c) **[2]** Give an upper bound for the probability that the particle walks for $> 4n^2$ steps without getting absorbed.

Solution.

(a) We define X_s = running time taken from node s to reach v_n. We define events E = left or right as the particle choose to move to left node or right node. Then

$$\begin{aligned} X_0|[E = right] &= 1 + X_1 \\ X_0 = X_0|[E = right] &= 1 + X_1 \\ T(0) &= 1 + T(1) \end{aligned} \tag{5}$$

Similarly, for starting from node s,

$$\begin{aligned} X_s|[E = right] &= 1 + X_{s+1} \\ X_s|[E = left] &= 1 + X_{s-1} \\ X_s &= \frac{1}{2}(X_s|[E = right] + X_s|[E = left]) \\ T(s) &= 1 + \frac{T(s-1) + T(s+1)}{2} \end{aligned} \tag{6}$$

(b) It is true when $s = 0$. We assume it is true for $j \leq s - 1$, then we would prove it with induction,

$$\begin{aligned} T(s) &= 1 + \frac{T(s-1) + T(s+1)}{2} \\ T(s) &= 1 + \frac{2s - 1 + T(s) + T(s+1)}{2} \\ T(s) &= 2s + 1 + T(s+1) \end{aligned} \tag{7}$$

So it is also true for $j = s$. For $T(0)$, we have,

$$\begin{aligned} T(0) = 2 * 0 + 1 + T(1) = 2 * 0 + 1 + 2 * 1 + 1 + T(2) = \ldots &= \sum_{j=0}^{n-1}(2 * j + 1) + T(n) \\ &= n^2 \end{aligned} \tag{8}$$

(c) $\mu = n^2$

$$Pr(T > 4\mu) < \frac{1}{4} \tag{9}$$

Question 4: Birthdays and applications ..... **[12]**

Suppose we have n people, each of whom has their birthday on a random day of the year. Suppose **there are** m **days in a year**, and let us pretend that this is some parameter.

(a) **[5]** What is the expected *number of pairs* (i, j) with $i < j$ such that person i and person j have the same birthday? For what value of n (as a function of m) does this number become 1?

(b) **[7]** This idea has some nice applications in CS, one of which is in estimating the "support" of a distribution. Suppose we have a radio station that claims to have a library of one million songs, and suppose that the radio station plays these songs by picking, at each

step a uniformly random song from its library (with replacement), playing it, then picking the next song, and so on.

Suppose we have a listener who started listening when the station began, and noticed that among the first 200 songs, there was a repetition (i.e., a song played twice). Prove that the probability of this happening (conditioned on the library size being a million songs) is < 0.05. Note that this gives us "reasonable doubt" about the station's claim that its library has a million songs.

Hint: Compute the probability of the complementary event —that all songs would be distinct— and prove that it must be large. You may use the inequality $(1-x)^n \geq 1-nx$ (for $x > 0$ and a positive integer n) without proof.

[This idea has many applications in CS, for estimating the size of sets without actually enumerating them.]

Solution.

(a) We define X_i is a binary random variable of 1 or 0 which denotes the persons in pair i have the same birthday. We define $X = \sum X_i$. The number of pairs is,

$$N = \binom{n}{2} = \frac{n(n-1)}{2} \tag{10}$$

The expectation of X_i is,

$$\begin{aligned} \mathbb{E}[X_i] = 0 * Pr(X_i = 0) + 1 * Pr(X_i = 1) &= Pr(X_i = 1) \\ &= \binom{m}{1}(\frac{1}{m})^2 \\ &= \frac{1}{m} \end{aligned} \tag{11}$$

Based on the linearity of expectation, we can yield that,

$$\mathbb{E}[X] = \sum \mathbb{E}[X_i] = \frac{n(n-1)}{2} \cdot \frac{1}{m} = \frac{n(n-1)}{2m} \tag{12}$$

When $n = \frac{1+\sqrt{8m+1}}{2}$, the expectation becomes 1.

(b) We set $n = 10^6$ and $m = 200$, we have the following inequality,

$$Pr(X \geq t\,\mathbb{E}[x]) \leq \frac{1}{t} \tag{13}$$

By setting $t = 50$, $t\,\mathbb{E}[x] < 1$, we could have,

$$Pr(X \geq 1) \leq Pr(X \geq 50\,\mathbb{E}[x]) \leq \frac{1}{t} = 0.02 \tag{14}$$

So we could prove that the probability is < 0.05.

Question 5: Checking matrix multiplication ..... **[14]**

Matrix multiplication is one of the classic algorithmic problems. Consider the problem of multiplying two $n \times n$ matrices over the field $\mathbf{F}_2$ (i.e., we have matrices with entries 0/1, and we perform all computations modulo 2; e.g., 0*0 + 1*1 + 1*1 + 1*0 = 0).

The best known algorithms here are messy and take time $O(n^{2.36\cdots})$. However, the point of this exercise is to prove a simpler statement. Suppose someone gives a matrix C and claims that $C = AB$, can we *quickly verify* if the claim is true?

(a) [**5**] First prove a warm-up statement: suppose a and b are any two 0/1 vectors of length n, and suppose that $a \neq b$. Then, for a random binary vector $x \in \{0,1\}^n$ (one in which each coordinate is chosen uniformly at random), prove that $\Pr[\langle a, x\rangle \neq \langle b, x\rangle \pmod 2] = 1/2$. [In other words, with a probability 1/2, we can "catch" the fact that $a \neq b$.]

(b) [**6**] Now, design an $O(n^2)$ time algorithm that tests if $C = AB$ and has a success probability $\geq 1/2$. (You need to bound both the running time and probability.)

(c) [**3**] Show how to improve the success probability to 7/8 while still having running time $O(n^2)$.

Solution.

(a) Suppose $a_i \neq b_i$. Let $\alpha = a_i x_i$, $\beta = b_i x_i$, $\bar{\alpha} = \sum_{i \neq j} a_j x_j \pmod 2$, $\bar{\beta} = \sum_{i \neq j} b_j x_j \pmod 2$, We assume all following calculation will apply modulo 2 implicitly.

$$\begin{aligned}\Pr[\langle a, x\rangle \neq \langle b, x\rangle \pmod 2] = {} & \Pr[\bar{\alpha} = \bar{\beta} | \alpha \neq \beta] \Pr[\alpha \neq \beta] \\ & + \Pr[\bar{\alpha} \neq \bar{\beta} | \alpha \neq \beta] \Pr[\alpha = \beta]\end{aligned} \tag{15}$$

We know $\alpha = \beta$ if $x_i = 0$ and $\alpha \neq \beta$ if $x_i = 1$. So $\Pr[\alpha \neq \beta] = \Pr[\alpha = \beta] = \frac{1}{2}$. Besides, $\bar{\alpha}$ is independent of α and $\bar{\beta}$ is independent of β. So $\Pr[\bar{\alpha} \neq \bar{\beta} | \alpha \neq \beta] = \Pr[\bar{\alpha} \neq \bar{\beta}]$. Then we would have,

$$\begin{aligned}\Pr[\langle a, x\rangle \neq \langle b, x\rangle \pmod 2] &= \Pr[\bar{\alpha} = \bar{\beta}] \Pr[\alpha \neq \beta] + \Pr[\bar{\alpha} \neq \bar{\beta}] \Pr[\alpha = \beta] \\ &= \frac{1}{2}(\Pr[\bar{\alpha} \neq \bar{\beta}] + \Pr[\bar{\alpha} = \bar{\beta}]) \\ &= \frac{1}{2}\end{aligned} \tag{16}$$

(b) The algorithm is in Algorithm 1.

Running time This algorithm take n^2 time as as we $y = Bx$ first and then do $T1 = Ay$. Each matrix-vector product takes only $O(n^2)$ time.

Success probability. If $C = AB$, this algorithm will always output the correct answer. If $C \neq AB$, at least one row in C and AB are different. This algorithm will have 0.5 probability to be successful.

$$\Pr[success] = 1 * \Pr[C = AB] + \frac{1}{2}\Pr[C \neq AB] = \frac{1}{2} + \Pr[C = AB] \geq \frac{1}{2} \tag{17}$$

(c) We can boost the performance by repeating. Each time the error probability is $\leq \frac{1}{2}$, and we can repeat the algorithm 3 times, the error probability is less than $\frac{1}{2}^3 = \frac{1}{8}$ and the success probability is greater than $\frac{7}{8}$.

Algorithm 1 Matrix comparison

Input A , B
Output Matrix-comparison(A, B)
function MATRIX-COMPARISON(A, B)
    Pick a vector $x \in_R \{0,1\}^n$
    $T1 = ABx$, $T2 = Cx$ ▷ $T1 = ABx$ takes n^2 time as we $y = Bx$ first and then do $T1 = Ay$
    if T1==T2 **then**
        return True
    else
        return False