CS 6150: HW4 – Graphs, Randomized algorithms
Collaborators: Thomas Burr, Craig Butler, Trey Lavery, Alex Carpenteri, Yo Office Hours, Frost Office Hours

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.

Part 1

From the quicksort proof in the notes we take the following: $f(n) = E[x] = \sum_{i=1}^{n} \frac{1}{n} E[X|F_i]$ For any i,$E[x|T_i] = T(j-1)) + T(n-j) + O(n)$ because as stated in the problem $(i-1)$ and $(n-j)$ are the size of the recursive sub-problems, and we do $O(n)$ additional work. From the algorithm, assume the value k that we are looking for is always located in the bigger half of the two blocks $j-1$ and $n-j$. Then since we are looking for the expected running time which is the worst case behavior for the algorithm, we take the max of the two partition halves as shown in the equation.

Thus we get the following above equation for the runtime of the quickselect algorithm in the question.

(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.]

Part 2

$\rightarrow T(n) \leq 4n$

$\rightarrow T(n) \leq n + \sum_{j=1}^{n} \frac{1}{n} \max 4 \cdot T(j-1), 4 \cdot T(n-j).$

$\rightarrow n + 4 \cdot \frac{1}{n} \cdot \sum_{j=1}^{n} \max T(j-1), T(n-j)$

$\rightarrow n + 4 \cdot \frac{1}{n} \cdot \sum_{j=1}^{\frac{n}{2}} T(n-j) + \sum_{j=\frac{n}{2}}^{n} T(j-1)$

Using the symmetric property ofthe summation and wolfram alpha to find a closed form of the

first summation we get the following assuming the pivot is the midpoint
$\rightarrow n + 4 \cdot \frac{1}{n} \cdot 2(\frac{1}{8}n \cdot (3n-2))$
$\rightarrow T(n) \leq 4n - 2$ which is less than $4n$

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.

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 $\{0, 1, \dots, n-1\}$ and return the ith element. Now suppose you have a *stream* of elements $a_1, a_2, \dots$ (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.)
$\rightarrow t = 1, x = a_1, prob = 1$
$\rightarrow t = 2, x = a_2, prob = \frac{1}{2}$
$\rightarrow t = 3, x = a_3, prob = \frac{1}{3}$
$\rightarrow t = n, x = a_n, prob = \frac{1}{n}$

Let us consider an example where we are trying to determine the probability for selecting a_2 from the stream.
We apply the rule for multiplying probabilities because the events are independent where we select the probability for picking a_2 and not selecting any of the other elements. We then get the following:
$\rightarrow \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} \cdot \frac{4}{5} \cdot \frac{5}{6} \cdot \dots \cdot \frac{n-1}{n}$
We simplify this equation in terms of some variable k :
$\rightarrow \frac{1}{k} \cdot \frac{k}{k+1} \cdot \frac{k+1}{k+2} \cdot \frac{k+2}{k+3} \cdot \dots \cdot \frac{n-1}{n}$. In this example, $k = 2$.
Cancelling like terms because of multiplication we get $\frac{1}{n}$. Using the defintion of uniform distribution, we have found that for any term a_k where k represents the term we want to select from the stream, the probability of selecting that term will be $\frac{1}{n}$ because we have shown that selecting any term from the stream will be $\frac{1}{n}$ which satisfies the definition of a uniform distribution.

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

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

Consider a path of length n comprising vertices $v_0, v_1, \dots, v_n$. 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 $0 < s < n$, $T(s) = 1 + \frac{T(s-1)+T(s+1)}{2}$.

To prove the first part that $T(0) = 1 + T(1)$ the only way for a particle to get back to $T(0)$ is if is at a step greater than $T(0)$. The problem states that a particle can either move forward or backward so the only way to move backwards to $T(0)$ is if we are already at $T(1)$ since we have the option of moving forward to $T(2)$ or backward $T(1)$ with the equation $1 + T(1)$.

To proved the second part we can use linearity of expectation. In the problem it is stated that the particle moving v_{s+1} or v_{s-1} is $\frac{1}{2}$. Let us define the event X as the particle moving forward and Y as the event of the particle moving backward. Then the expectation for the total number of steps is the following:

$\rightarrow T(s) = E(X|Forward) \cdot P(Forward) + E(Y|Backward) \cdot P(Backward)$

$\rightarrow T(s) = (1 + T(s+1)) \cdot \frac{1}{2} + (1 + T(s-1)) \cdot \frac{1}{2}$

$\rightarrow T(s) = 1 + \frac{T(s-1)+T(s+1)}{2}$

(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.]

$\rightarrow T(s) = 2s + 1 + T(s+1)$

$\rightarrow T(s) - T(s+1) = 2s + 1$

$T(0) = 2s + 1 + T(1)$

$T(1) = 2s + 1 + T(2)$

$T(n-1) = 2s + 1 + T(n)$

$T(0) = T(0) - T(1) + T(1) - T(2) + T(2) - T(3) + \cdots + (T(n-1) - T(n))$

$\rightarrow \sum_{i=0}^{n-1} 2i + 1$

$\rightarrow n^2$

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

We can apply Markov's rule that we learned in class to this problem.

$\rightarrow P(x \geq 4n^2) \leq \frac{E(0)}{4n^2}$

$\rightarrow \frac{1}{4}$

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?

Part 1

Let **p1** represent the probability for the first person to select their birthday. Since no birthdays have been selected then *p1* has m choices for their birthday. Let **p2** represent the probability for the second person to have the same birthday as *p1*. The probability for this event occuring is $\frac{1}{m}$. Using the law of linear expectation to calculate the expectation for calculating the probability that two people share a common birthday we get the

following:
The number of unique pairs (i, j) with the same birthday is $\binom{n}{2}$. Dividing that value by the total number of days m we get the following value for the expected total number of pairs: $\frac{n^2-n}{2m}$.
Part 2
Using the function calculated in part 1 we get the following: $\frac{n^2-n}{2m} = 1$.
Then simplifying this we get the following: $n^2 - n = 2m$
$\rightarrow n^2 - n - 2m = 0$
$\rightarrow n = \frac{1+-\sqrt{1+8m}}{2}$ from applying the quadratic equation to the previous step.
$\rightarrow n = \frac{1+\sqrt{1+8m}}{2}$ because we can never have a negative number for n if we assume that m will always be a positive value.

(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.]

For this part, we can model this problem similar to the birthday problem. The number of songs that are duplicates is $\binom{200}{2}$ which can be thought of as $\binom{n}{2}$ from the birthday problem. The number of days m in this problem is represented with the number of tracks which in this case is 10^6. Representing the probability of a unique song being played twice we get the following: $\frac{\binom{200}{2}}{10^6}$. Simplifying this fraction we get a probability of *0.0199* which is less than *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 $a \neq b$.]
If we take the dot product of $< a[1...n-1], x[1...n-1] >$ then we get some vector y. The last value in the vector can either be a 0 or 1. Now let us consider the dot product of the following vectors:
$< a, x >$ then we get some vector z. Notice the difference between vectors a and x in this dot product contain all the entries from 1 to n compared to the first two vectors that produce vector y. The probability of vector y and vector z being the same is $\frac{1}{2}$ if the last entry of vector y and vector z match.

(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.) // For this problem we can apply Freivald's algorithm.
Algorithm

Algorithm 1 Matrix Checking

```
    Input: Matrix C, Matrix A, Matrix B
    Output: A boolean containing the result of C = AB with successful probability of $\frac{1}{2}$
1: Generate a random $n$ x 1 vector $x$ with each entry $x_i \in F_2$
2: Compute vector $Cx$ = Matrix C · vector $x$
3: Compute vector $Bx$ = Matrix B · vector $x$
4: Compute vector $D$ = Matrix A · vector $Bx$
5: for $i$ in range 0 to len(vector D) do
6:     if D[i] - Cx[i] $\neq$ 0 then return False
7:     end if
8: end for
9: return True
```

The runtime of the algorithm is $O(n^2)$ because the time taken to load the matrix is $O(n^2)$ and to compute the dot product between a matrix and a vector we have to check every single entry in the matrix for the computation which takes $O(n^2)$. Using the results from part a, we can bound the probability of the algorithm to be $\frac{1}{2}$

(c) [**3**] Show how to improve the success probability to 7/8 while still having running time $O(n^2)$.
The probability of running the algorithm and having an error that is not detected and produces the right result is $\frac{1}{2^k}$. If we run the algorithm more than once, we lower the probability that this scenario will occur. To get a success rate of $\frac{7}{8}$, we need to run the algorithm 3 times.