Problem Set 2 Solutions

This problem set is due **at 10:00pm** on **Wednesday, February 19, 2020**.

EXERCISES (NOT TO BE TURNED IN)

Divide and Conquer Algorithms

- Do Exercise 4.2-3 in CLRS on page 82.
- Do Exercise 9.3-1 in CLRS on page 223.

Amortized Analysis

- Do Exercise 17.1-3 in CLRS on page 456.
- Do Exercise 17.2-3 in CLRS on page 459.
- Do Exercise 17.3-7 in CLRS on page 463.

Union-Find

- Do Exercise 21.3-3 in CLRS on page 572.
- Do Exercise 21.3-5 in CLRS on page 572.

Problem 2-1. WageRank [45 points] After another successful year of Googol, Parry Lage prepares to negotiate his salary with the board of Counting Numbers Inc. As the co-developer of the WageRank algorithm, Parry believes himself capable of modeling the behavior of the board members and predicting his salary even before the meeting. The procedure, as he understands it, is that each of the 100 board members has drawn an optimal salary uniformly at random from the range $[0, 1000000]$ in dollars, not necessarily independently. Parry can propose any salary he desires, and each board member has one vote, which will be cast in favor of the proposal if the proposed value is no greater than this board member's optimal salary and will be cast against the proposal otherwise. If at least 51 of the 100 board members cast votes in favor of Parry's proposed salary, then this salary will be approved. Otherwise, Parry risks being fired! Parry wonders how likely it is that he can secure a proposal of at least $\$200000$, an *extremely* modest salary given all of his hard work.

(a) [8 points] Use Markov's Inequality to provide a bound on the probability that Parry will fail in his proposal of a salary of $\$200000$. In addition, solve for the highest salary that Parry can propose while keeping his probability of failure at most $\frac{1}{2}$.

Solution: First, we define the indicator random variables X_i, which equal 1 if the i^{th} board member denies a given salary and 0 otherwise, for each i, $1 \leq i \leq 100$. Now, we define

$$X = \sum_{i=1}^{100} X_i.$$

We would like to find the probability $\Pr(X \geq 50)$. Note that for a proposed salary of 200000, each X_i is 1 with probability $\frac{1}{5}$ and 0 with probability $\frac{4}{5}$. Thus, we have $\mathbb{E}[X_i] = \frac{1}{5}$ for each i. Thus, by the linearity of expectation, we have

$$\mathbb{E}[X] = \sum_{i=1}^{100} \mathbb{E}[X_i] = 100 \cdot \frac{1}{5} = 20.$$

Now, by Markov's Inequality, we have

$$\Pr(X \geq 50) \leq \frac{20}{50} = \frac{2}{5}$$

Moreover, following the same logic, we see that if Parry proposes a salary of s dollars, then each X_i is 1 with probability $\frac{s}{1000000}$ and 0 with probability $(1 - \frac{s}{1000000})$. Thus, we have $\mathbb{E}[X_i] = \frac{s}{1000000}$, and by linearity of expectation, this gives

$$\mathbb{E}[X] = \sum_{i=1}^{100} \mathbb{E}[X_i] = 100 \cdot \frac{s}{1000000} = \frac{s}{10000}.$$

Now, by Markov's Inequality, we have

$$\Pr(X \geq 50) \leq \frac{s}{500000},$$

so to guarantee a probability of failure of at most $\frac{1}{2}$, we should take $\frac{s}{500000} \leq \frac{1}{2}$, or $s \leq 250000$, meaning that Parry can propose a salary of at most $\$250000$.

(b) [7 points] Now, one of Parry's most trusted advisors has come to Parry and shared an important piece of information. Specifically, she informs Parry that the board members have chosen the salaries pairwise independently. Use Chebyshev's Inequality to provide a tighter bound on the probability that Parry will fail in his proposal of a salary of $\$200000$.

Hint: Recall that two random variables X and Y are pairwise independent if, and only if, $\Pr(X = x, Y = y) = \Pr(X = x) \cdot \Pr(Y = y)$.

Solution: We define X_i and X as before, and we again have $\mathbb{E}[X] = 20$. This time, we also have the additional fact that the X_i are pairwise independent.

Now, to find the variance of X, we first find the variance of each X_i. Recall that for a proposed salary of $\$200000$, each X_i is 1 with probability $\frac{1}{5}$ and 0 with probability $\frac{4}{5}$, giving each X_i an expectation of $\mathbb{E}[X_i] = \frac{1}{5}$.

Thus, we have

$$\text{Var}[X_i] = \frac{4}{5} \cdot \left(0 - \frac{1}{5}\right)^2 + \frac{1}{5} \cdot \left(1 - \frac{1}{5}\right)^2 = \frac{4}{25}.$$

This is useful because we would like to find the variance of X, which is the sum of the random variables X_i. The variance of pairwise-independent random variables is linear, so we have

$$\text{Var}[X] = \sum_{i=1}^{100} \text{Var}[X_i],$$

which will allow us to calculate the variance of the random variable X.

We can formally prove the linearity of the variance for these pairwise-independent random variables using the fact that the expectation is multiplicative for pairwise-independent random variables. We do so as follows:

$$\begin{aligned}
\text{Var}[X] &= \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{100} X_i\right)^2\right] - \left(\sum_{i=1}^{100} \mathbb{E}[X_i]\right)^2 \\
&= \left(\mathbb{E}\left[\sum_{i=1}^{100} X_i^2 + \sum_{\substack{i,j=1 \\ i\neq j}}^{100} X_i X_j\right]\right) - \left(\sum_{i=1}^{100} (\mathbb{E}[X_i])^2 + \sum_{\substack{i,j=1 \\ i\neq j}}^{100} \mathbb{E}[X_i]\mathbb{E}[X_j]\right) \\
&= \left(\sum_{i=1}^{100} \mathbb{E}[X_i^2] + \sum_{\substack{i,j=1 \\ i\neq j}}^{100} \mathbb{E}[X_i X_j]\right) - \left(\sum_{i=1}^{100} (\mathbb{E}[X_i])^2 + \sum_{\substack{i,j=1 \\ i\neq j}}^{100} \mathbb{E}[X_i]\mathbb{E}[X_j]\right) \\
&= \sum_{i=1}^{100} \mathbb{E}[X_i^2] + \sum_{\substack{i,j=1 \\ i\neq j}}^{100} \mathbb{E}[X_i]\mathbb{E}[X_j] - \sum_{i=1}^{100} (\mathbb{E}[X_i])^2 - \sum_{\substack{i,j=1 \\ i\neq j}}^{100} \mathbb{E}[X_i]\mathbb{E}[X_j] \\
&= \sum_{i=1}^{100} \mathbb{E}[X_i^2] - \sum_{i=1}^{100} (\mathbb{E}[X_i])^2 \\
&= \sum_{i=1}^{100} \left(\mathbb{E}[X_i^2] - (\mathbb{E}[X_i])^2\right) \\
&= \sum_{i=1}^{100} \text{Var}[X_i],
\end{aligned}$$

where we invoke the pairwise-independence of the random variables in the fourth-to-last line when we write that

$$\mathbb{E}[X_i X_j] = \mathbb{E}[X_i]\mathbb{E}[X_j].$$

Thus, we have

$$\text{Var}[X] = \sum_{i=1}^{100} \text{Var}[X_i] = 100 \cdot \frac{4}{25} = 16,$$

as desired.

Substituting into Chebyshev's Inequality, we have

$$\Pr(|X - 20| \geq 30) \leq \frac{16}{30^2} = \frac{4}{225},$$

so it follows that

$$\Pr(X \geq 50) \leq \Pr(|X - 20| \geq 30) \leq \frac{4}{225},$$

as desired.

Note that Parry can increase his proposed salary to $\$429986$ while keeping the probability of failure at most $\frac{1}{2}$ with these conditions, as he can determine from Chebyshev's inequality.

(c) [5 points] Now, Parry, having co-developed WageRank, insists that he can make an even stronger assumption about the board members—namely, that they all select the salaries mutually independently. With this new assumption, use the Chernoff Bound to provide a yet tighter bound on the probability that Parry will fail in his proposal of a salary of $\$200000$.

Solution: We define X_i and X as before, and we again have $\mu = \mathbb{E}[X] = 20$.

Because the X_i are mutually independent (since Parry now assumes that the board members' choices are mutually independent) and each is an indicator variable in the set $\{0, 1\}$, the conditions are met to apply the Chernoff bound. Specifically, setting $\beta = \frac{49-20}{20} = 1.45 > 1$, we fall into the second case of the Chernoff bound, for which we have

$$\Pr(X > 49) = \Pr(X > (1 + 1.45) \cdot 20) \leq e^{-(1.45)(20)/3} = e^{-29/3} \approx 6.3 \cdot 10^{-5}$$

Now, Parry is *fairly* convinced that he can succeed with his proposal of $\$200000$.

(d) [25 points] Just as Parry believes that he's begun to work out the right salary to propose, he is informed by one of his advisors that the format of the board meeting is not what he believed. The board size is not fixed at 100 but rather includes n board members, and Parry can only propose one salary to the board, so he needs to choose very wisely. Additionally, the power dynamics on the board have shifted, and some board members have more votes than others. Specifically, suppose the board members $x_1, x_2 ... x_n$ each have $w_1, w_2, ... w_n$ votes respectively. You may assume the total number of votes, W, is odd.

Parry, understanding the intricacies of WageRank, has held a private meeting with every board member to ask the board member what a reasonable CEO salary is. He knows that board member x_i thinks s_i is the ideal salary for Parry. A board member will use all their votes to vote against Parry's proposed salary if it is higher than their wishes, and the board member will use all their votes in favor of the proposed salary if the proposal is less than or equal to their wishes. You may assume the s_i are distinct.

Devise an algorithm to determine the highest salary Parry can propose. For full credit, your algorithm should run in $\mathcal{O}(n)$. Partial credit will be given

to an $\mathcal{O}(n \log n)$ solution. You may cite algorithms shown in lecture without proof.

Approaching the solution: For a problem like this one in which each element of some collection can have its own weight, it is often helpful to imagine the special case in which each element's weight is identical. In the context of this problem, taking all of the weights to be identical converts this problem to a simple median-finding problem, since Parry needs at least half of the board members to support him and can achieve this goal by choosing a salary just below the median salary among the board members. This motivates us to consider how we can apply the idea of median-finding, even when the board members might have distinct weights.

The basic idea in the algorithm for the median is to find a balanced pivot. Luckily, in our case, we can use the median algorithm as a subroutine!

Solution: Let $W = \sum_i w_i$. We assume that all weights are strictly positive. We want to find the largest s_i such that, the total sum of weights of all board members which think s_i is reasonable is at least $W/2$. That is, if we could sort the numbers into $s_1 \leq s_2 \leq \cdots \leq s_n$ (which we can't because of time limitations), the answer would be the value corresponding to the largest index i such that

$$\sum_{j \geq i} w_j \geq W/2$$

Consider the following version of the *weighted median* problem. Given a sequence $s_1, \ldots, s_n$, with respective weights $w_1, \ldots, w_n$, find the *largest* value s_i such that

$$\sum_{j \text{ s.t. } s_j \geq s_i} w_j \geq W/2$$

Let s^* be the weighted median. We claim that s^* is the highest salary Melon can propose. First, by the definition of s^*, there are enough votes by members that think a reasonable salary is at least s^* to accept the proposal. If he proposes a value less than s^*, then his proposal would be accepted but it would be sub-optimal. If he proposes a higher value, then, again by the definition of s^*, there would only be strictly less than $W/2$ accepting members and the proposal would be rejected.

Hence, we wish to find the weighted median. As with the regular median algorithm, we generalize the task. Define $WM(s, w, t)$ as the problem of, given values $s = s_1, \ldots, s_n$ and weights $w = w_1, \ldots, w_n$, finding the *largest* value s_i such that

$$\sum_{j \text{ s.t. } s_j \geq s_i} w_j \geq t$$

Clearly, the weighted median is just $WM(s, w, (\sum w)/2)$. We use the following algorithm which heavily uses the unweighted median algorithm as a subroutine.

1. If $n < 5$, find the weighted median "by hand". We could sort the elements and scan from right to left while the sum of weights is at least $W/2$.
2. Find the unweighted median m of the values in s.
3. Divide s into s_R and s_L by placing the value s_i in s_R if $s_i \geq m$. Otherwise, place s_i in s_L. Similarly, divide w into two halves w_R and w_L according to the value of the corresponding s_i. Compute $w_r = \sum_{w \in w_R} w$.
4. If $w_r = t$, return m.
5. If $w_r > t$, return $WM(s_R, w_R, t)$.
6. If $w_r < t$, return $WM(s_L, w_L, t - w_r)$.

For runtime, note that step 1 takes $\mathcal{O}(1)$ time, step 2 takes $\mathcal{O}(n)$ due to the median-finding algorithm from class and step 3 takes $\mathcal{O}(n)$ time. Step 4-6 recurses on a subarray of half size (due to the median). So our recurrence is

$$T(n) = T(n/2) + \mathcal{O}(n)$$

And hence, by the master theorem (case 3), the runtime is $\mathcal{O}(n)$.

Now we prove correctness. That is, we prove that the algorithm returns the weighted median. Clearly, the algorithm works for $n \leq 5$. We now argue that it also works regardless of which step (4-6) returns the answer. Note that, to show that a given value v is correct, we need to prove that (i) the sum of weights of elements greater or equal to v is at least t and (ii) v is the largest such value.

First, assume that we return through step 4. In this case, we claim that m satisfies both conditions. Condition (i) follows from the fact that $w_r = t$. Condition (ii) follows by the assumption that all weights are strictly positive.

The case where we return through step 5 is also easy. In this case, the sum of weights of values larger than m is strictly greater than t. This means that m is potentially violating condition (ii). Let m' be the value returned by the recursive call. Conditions (i) and (ii) follow easily from the fact that all elelments in L are smaller than m and therefore smaller than m'. Indeed, if w' is the sum of weights of values in R greater than m', then w' is also the sum of weights of values in the whole set larger than m'.

Finally we treat the last case where we return through step 6. In this case, m is too large as it violates condition (i). Let m' be the value returned by the recursive call and let w' be the sum of weights of elements in L greater than or equal to m'. By the definition of WM, $w' \geq t - w_r$ and m' is the largest such value. Note that all values in R are also larger than m' (because m is

the median of the whole set). Therefore, the sum of weights of values in the whole set which are larger than m' is $m' + w_r \geq t$ and condition (i) is satisified. Condition (ii) follows from the fact that m' is the largest value in L with the property that the sum of weights of values in L greater than m' is at least $t - w_r$.

Problem 2-2. Zark's Dynamic Photos [45 points]

Zark Muckerburg, having just purchased the photo-sharing app Delaypound (DP), realizes that with so many users creating and deleting content, he'll need a more adaptable storage array. Zark wants to support two operations: appending image data at the end of the array and deleting the image data from the end of the array when a photo is removed.

In particular, he begins with an array of length one and inserts the first element into the one free space in this array. In general, if at any point Zark has a full array of length n and wants to append an element, he creates a new empty array of length $n+1$, copies the n elements from the length-n array to the first n entries of this new array, and then inserts the new element into the last space. If Zark wants to delete the last element from an array of length n, he creates a new empty array of length $n-1$ and copies the first $n-1$ elements from the length-n array to the $n-1$ entries of this new array.

Suppose that copying each element when resizing the array has a cost of 1, as does adding a single element, or deleting a single element *when the array is not being resized*.

(a) [10 points] Calculate the cost of performing n insertions followed by n deletions. Given this cost, find the amortized cost per operation.

Solution: The first insertion has a cost of 1 because we add one element without changing the array size. The second insertion has a cost of 2 because we copy one element and add another element. In general, the i^{th} insertion has a cost of i. Thus, the total cost of the insertions is

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

For the deletions, the first deletion has a cost of $n-1$, since $n-1$ elements need to be copied to a new array of length $n-1$. The second deletion has a cost of $n-2$, since $n-2$ elements need to be copied to a new array of length $n-2$. In general, the i^{th} deletion has a cost of $n-i$. Thus, the total cost of the deletions is

$$\sum_{i=1}^{n}(n-i) = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}.$$

Summing these costs yields a total cost of

$$\frac{n(n+1)}{2} + \frac{n(n-1)}{2} = n^2$$

and dividing by a total of $2n$ operations gives an amortized cost of $\frac{n}{2}$ per operation.

(b) [10 points] Zark is unsatisfied by this naïve storage method, so he puts his top engineers on the problem, and they propose an alternative solution. They suggest that, whenever the array reaches capacity, rather than being copied to a new array that is one unit longer, it can be copied to one that is *twice* as long. In particular, to append an element to a full length-n array, Zark can create a new empty array of length $2n$, copy the n elements from the length-n array to the first n entries of this new array, and then insert the new element into the $(n+1)$th space.

To handle deletions, the engineers suggest that an array can be shrunk if the number of elements reduces to one-half the length of the array. In particular, if the array has length n but is only populated by elements in the first $\frac{n}{2}+1$ slots and Zark wants to delete an element, then he can create a new empty array of length $\frac{n}{2}$ and copy the first $\frac{n}{2}$ elements from the length-n array to the $\frac{n}{2}$ entries of this new array.

The engineers purport that their algorithm achieves the desired amortized $\mathcal{O}(1)$ cost per operation. Propose an adversarial sequence of operations in which the amortized cost per operation is $\Omega(n)$.

Solution: Suppose that we take n to be a power of 2. Then one such sequence is performing $\frac{n}{2}$ inserts and then alternating between inserts and deletes. The total cost of the first $\frac{n}{2}$ appends is

$$\sum_{i=0}^{\lg(n/2)} 2^i = n - 1.$$

Then the cost of the next append is $\frac{n}{2}+1$ (because all $\frac{n}{2}$ elements need to be copied and another element added when the length of the array doubles), and the cost of the next delete is $\frac{n}{2}$ (because the first $\frac{n}{2}$ elements need to be copied when the length of the array halves). This proceeds for all of the remaining $\frac{n}{2}$ operations, for a total cost of $\frac{n}{4} \cdot (n+1)$.

Finally, summing the total cost yields

$$(n-1) + \frac{n}{4} \cdot (n+1) = \frac{1}{4}n^2 + \frac{5}{4}n - 1$$

and the amortized cost per operation is $\Omega(n)$, as desired.

(c) [25 points] Zark sees some potential to the solution that his top engineers propose, but is still unsatisfied, so he recruits you to see if an amortized $\mathcal{O}(1)$ cost per operation can truly be achieved. You, being a hardworking algorithms student, realize that although the doubling scheme for appending elements is spot-on, the halving scheme is a bit ineffective. Instead, you propose that Zark

should only halve the size of the array if the number of elements reduces to one-quarter of the length of the array. In particular, if the array has length n but is only populated by elements in the first $\frac{n}{4}+1$ slots and one element is to be deleted, then he can create a new empty array of length $\frac{n}{2}$ and copy the first $\frac{n}{4}$ elements from the length-n array to the first $\frac{n}{4}$ entries of this new array. Use the potential method to demonstrate that this implementation provides an amortized cost of $\mathcal{O}(1)$ per operation, as desired.

Hint: use the potential function $\Phi(m,n) = \max\{2m-n, \frac{n}{2}-m\}$, where m is the current number of stored elements and n is the current capacity (length) of the array.

Solution: First, we check that the potential function is non-negative. We can split this into two cases. Note that $0 \leq m \leq n$.

- When $m \geq \frac{n}{2}$, we have $2m-n \geq 0$, so $\Phi(m,n) \geq 0$.
- When $m \leq \frac{n}{2}$, we have $\frac{n}{2}-m \geq 0$, so $\Phi(m,n) \geq 0$.

Thus, in either case, $\Phi(m,n) = \max\{2m-n, \frac{n}{2}-m\} \geq 0$, as desired. We also assume that, at the beginning, both $m = n = 0$ so that $\Phi(0,0) = 0$.

Next, we would like to demonstrate that this potential function guarantees an amortized cost of $\mathcal{O}(1)$ per operation.

For all insertions and deletions that don't require any changes to the array, the cost is 1. For insertions, m increases by 1 so $\Delta\Phi \leq 2$. For deletions, m decreases by 1 so $\Delta\Phi \leq 1$. So the amortized cost is at most 3.

Suppose that the array is at max capacity, i.e. $m = n$, and Zark wants to append a new image. The cost of this operation is $n+1$, to copy over all n elements as well as insert the newest one. Before the operation, the potential is

$$\Phi = \Phi(n,n) = \max\left\{2n-n, \frac{n}{2}-n\right\} = n.$$

After the operation, the array has length $2n$ and stores $n+1$ elements, so the potential is

$$\Phi' = \Phi(n+1,2n) = \max\left\{2(n+1)-2n, \frac{2n}{2}-(n+1)\right\} = 2.$$

Thus, we get that the amortized cost of doubling is

$$\begin{aligned}\text{amortized cost} &= \text{actual cost} + \Delta\Phi \\ &= (n+1) + 2 - n \\ &= 3.\end{aligned}$$

Suppose instead that the array is only populated by the first $\frac{n}{4}+1$ slots, i.e. $m = \frac{n}{4}+1$, and Zark wants to delete an image. The cost of this operation is

$\frac{n}{4}+1$ to copy the $\frac{n}{4}$ elements to be kept into a new array. Before the operation, the potential is

$$\Phi = \Phi\left(\frac{n}{4}+1, n\right) = \max\left\{2\left(\frac{n}{4}+1\right) - n, \frac{n}{2} - \left(\frac{n}{4}+1\right)\right\} = \frac{n}{4} - 1.$$

After the operation, the array has length $\frac{n}{2}$ and stores $\frac{n}{4}$ elements, so the potential is

$$\Phi' = \Phi\left(\frac{n}{4}, \frac{n}{2}\right) = \max\left\{2\left(\frac{n}{4}\right) - \frac{n}{2}, \frac{n}{4} - \frac{n}{4}\right\} = 0.$$

Thus, we get that the amortized cost of halving is

$$\begin{aligned}\text{amortized cost} &= \text{actual cost} + \Delta\Phi \\ &= \frac{n}{4} + 0 - \left(\frac{n}{4} - 1\right) \\ &= 1.\end{aligned}$$

If we start with no images and an empty array, then $\Phi = 0$ initially. We have shown that Φ is always non-negative, and the cost of k operations is bounded by $3k$. Thus, this procedure achieves constant amortized cost per operation, as desired.

Problem 2-3. Feedback Form [10 points] Please fill out a feedback form about this problem set at

`https://forms.gle/2j2iGmkDKqDiXLM89`.

It should not take more than a few minutes and will greatly help us improve teaching and material for future semesters!