Problem Set 3 Solutions

This problem set is due **at 10:00pm** on **Wednesday, February 26, 2020**.

EXERCISES (NOT TO BE TURNED IN)

Union-Find

- Do Exercise 21.3-4 in CLRS on page 572.

Problem 3-1. Cam's Books [60 points]

Cam Petitive and Opal T. Mull have to do readings for a class, but they dread going to the library!

In typical MIT style, the library owns m books which students refer to by number, from 1 to m. Their professor gives the name of a book, b, each time she assigns a reading. If a student has the book checked out, they complete the reading swiftly and happily (cost 0). Otherwise, they have to walk across campus to check the book out from the library (cost 1). The library has lots of copies of each book, so students don't have to worry about reserving books or book shortages. The problem is, each student is only allowed to have k books checked out at any time, so students who have k books checked out must return one of their previously checked-out books when checking out a new one.

Opal has somehow managed to figure out not only the n readings, but also the optimal sequence of books to return, to minimize trips to the library. Cam hasn't done the same, but he thinks he can compete with Opal without knowing the readings or doing such complicated calculations.

Note that the professor seems to enjoy repetition as a teaching tool, and doesn't hesitate to assign old readings. At the beginning of the semester, both Cam and Opal have zero books checked out.

(a) [15 points] The library doesn't set explicit due dates, so Cam thinks he can exploit this by holding on to his books for as long as possible. He works out the following strategy for each reading b :

- If b is already checked out, he does not go to the library, and completes the reading swiftly and happily.
- Otherwise, he checks out b from the library as follows:
  - If Cam hasn't reached his limit of k books checked out yet, he doesn't return any books.
  - Otherwise, he returns the book with the *most recent* check out date.

Prove that Cam's strategy is not competitive with Opal's if $k \geq 2$. That is, prove that there are input sequences of length n such that the ratio of Cam's to Opal's library trips tends to ∞ as n tends to ∞.

Solution: Note that the problem corresponds to the paging problem from recitation.

In order for Cam's strategy to be α-competitive, for any sequence B of books, we need

$$C_{CAM}(B) \leq \alpha \cdot C_{OPT}(B) + c$$

for some constants α and c. Following the instructions, we'll construct an adversarial example of length n such that the ratio of Cam's to Opal's library

trips tends to ∞ as n tends to ∞. As a result, there are no α and c for which Cam's strategy is α-competitive.

For any $n > k$ consider reading assignments of length n of the following form:

- The first k readings are books $1, 2, \ldots, k$.
- After this, the readings alternate between $k+1$ and k.

After the first k readings, Cam would always replace the books k and $k+1$ alternatively at each reading assignment, and his cost on the sequence would be n.

Opal, on the other hand, after the first k requests, would return one of the books other than k or $k+1$, and would never have to visit the library again on the sequence. So her cost would be $k+1$.

The competitive ratio is hence $\frac{n}{k+1}$, which tends to ∞ as n does.

Cam's strategy here corresponds to the Last-In-First-Out (LIFO) paging algorithm.

(b) [25 points] The semester has progressed (both Opal and Cam have the same k books checked out), and now Cam thinks he has a better idea. Cam keeps his k library books stacked up on his desk, to remember the last time each was used for a reading. Whenever a reading is assigned, if Cam has the book already, he slides it out, does the reading, and places it on the top of the stack. If Cam has to go to the library, he slides out the bottom book from the stack, returns that one, and puts his newly checked-out book on the top of the stack when he gets back.

Use the potential method to prove that Cam's strategy is k-competitive with Opal's. That is, for any sequence of reading assignments of length n, the ratio of Cam's to Opal's library trips is no more than k.

Hint: At any time let S_{CAM} be the set of books that Cam has checked out, and let S_{OPT} be the set of books that Opal has checked out. Use the potential function

$$\Phi = \sum_{b \in S} w(b)$$

where $S = S_{CAM} \setminus S_{OPT}$ is the set of books that Cam has which Opal doesn't, and $w(b)$ is how high up book b is on Cam's shelf, with the bottom book's height $w(b_{bottom}) = 1$ and the top book's height $w(b_{top}) = k$.

Solution:

Note that Cam's strategy in part (b) corresponds to the Least Recently Used (LRU) paging algorithm.

To prove Cam's strategy is k-competitive, we need to show that for any sequence of books B,

$$C_{CAM}(B) \leq k \cdot C_{OPT} + c.$$

Let $C(i)$ represent the cost of an algorithm at step i. Our strategy will be to show Cam's amortized cost at each step is k-competitive:

$$C_{CAM}(i) + \Phi(i) - \Phi(i-1) \leq k \cdot C_{OPT}(i), \tag{1}$$

Then we can add up the amortized cost over all steps to show the overall cost is k-competitive.

As in the hint, let $\Phi = \sum_{b \in S} w(b)$.

Now consider an arbitrary reading b. To prove (1), we'll look at the different cases on whether Opal or Cam has the book checked out already.

- Suppose that Cam has b checked out and Opal does not. Opal's cost is 1. We will show that the potential increases by at most k. First, note that, whereas prior to the request, the book b contributed to Φ (as Cam had it checked out but not Opal), after the request b *does not* contribute to the sum. In addition, after reading b, Cam places it on top of his stack. Because of this, any books in $S_{CAM} \setminus S_{OPT}$ which were previously above b will drop places by one, decreasing the summation. However, Opal must return at least one book. So the set $S_{CAM} \setminus S_{OPT}$ can increase by one. Therefore, we may need to add a term to the sum defining Φ. This term can be at most k.
- Suppose that Opal has b checked out and Cam does not. In this case, Opal's cost is 0 while Cam's cost is 1. We will show that $\Delta\Phi \leq -1$. The set $S_{CAM} \setminus S_{OPT}$ has at least 1 book (this is because Opal has b checked out, so there must be at least one book checked out by Cam and not by Opal). When Cam returns the book at the bottom of the stack, every book's place drops by 1. So the summation decreases by at least 1!
- Suppose that both Opal and Cam have b checked out already. Then $C_{CAM}(i) = C_{OPT}(i) = 0$. Still, Opal may decide to go to the library. But if so, Opal will incur a cost of 1 and, as in the first case, the potential increases by at most k.
- Suppose that neither Opal nor Cam have b checked out yet. Then $C_{CAM}(i) = C_{OPT}(i) = 1$. When Cam returns the bottom book, the potential can only decrease. If Opal returns a book that Cam has, then we need to add a term to Φ but, as before, this term can be at most k. We get that $\Delta\Phi \leq k$.

In all cases $C_{CAM}(i) + \Phi(i) - \Phi(i-1) \leq k * C_{OPT}(i)$. Therefore, when the costs and potentials are summed across the whole sequence, all the $\Phi(i)$ are cancelled out except for the first and last terms, leaving

$$\sum_{i=1}^{n} C_{CAM}(i) + \Phi(n) - \Phi(0) \leq k \cdot \sum_{i=1}^{n} C_{OPT}(i).$$

Since the potential starts at 0 (Cam and Opal start with the same set of books, so $S_{CAM} \setminus S_{OPT}$ is empty), and the potential function is always positive,

$$\sum_{i=1}^{n} C_{CAM}(i) \leq k \cdot \sum_{i=1}^{n} C_{OPT}(i)$$

and hence Cam's strategy is k-competitive with Opal's.

(c) [5 points] Which aspect of the proof for part (b) does not work if you were to try to prove Cam's original strategy is k-competitive with the same potential function?

Solution: If we try to prove Cam's original strategy is k-competitive, and, like before, consider each possible case upon receiving book b, we run into trouble in the case in which Opal has b checked out and Cam doesn't. In this case, Opal's cost is zero while Cam's cost is 1. However, the potential function, unlike before, does not decrease—when we remove the top book from the stack, the positions of the remaining books in the stack stay the same. So $C_{CAM} + \Delta\Phi > 0$. So, in equation (1) above, the left hand side is positive and the right hand side is 0. Notice that this is exactly what happens with the sequence from part (a). Note that the failure to apply this potential method alone does not prove this strategy is not k-competitive. Luckily, we already proved this in part (a).

(d) [15 points] Cam makes a friend in the class, Amy Ortez. Amy is intrigued to see that her book-return strategy is very similar to Cam's. Amy keeps her books on a shelf instead of in a stack. If she needs to go to the library, she slides out the rightmost book from the shelf, returns it, and inserts her newly checked out book to the left of her other books upon returning. Whenever a book is assigned which she already has checked out, Amy slides it out, does the reading, and then places it back where it was located on the shelf.

Use the potential method to prove that Amy's strategy is also k-competitive with Opal's.

Solution: Since Amy's strategy is very similar to Cam's, we try to use the same potential function as before. This works! In fact, the argument is mostly the same. Of the four cases analyzed previously, the only place where the behavior of Amy's strategy differs from Cam's is when Amy/Cam do not have to return a book, so they have the book in their stack. So we only need to check those cases.

- In the case where Amy has the book checked out but Opal doesn't, the analysis is almost the same as before. The only difference is that this time,

no books in $S_{AMY} \setminus S_{OPT}$ shift downwards as a result of moving b to the top of the stack. Still, the potential increases at most k because of Opal returning a book.

- In the case where both Amy and Opal have b checked out, the analysis is the same as before.

Amy's strategy corresponds to the First-In-First-Out (FIFO) paging algorithm.

Problem 3-2. Catering Challenge [30 points]

App Bonatite is catering for pavilion festivities at MIT. They know the total amount of food they will bring, and have a list of all the students who are attending, but they aren't sure how much food to portion for each student. So they reach out to some contacts at the dining halls, who report back relative amounts of what students eat. For instance, it might be observed that student A eats 1.5 as much as student B. No one keeps record of specific quantities of food eaten; only *ratios* between how much different students eat are observed. App Bonatite wants to know when they will have enough information to set up all the food for the pavilion festivities, or at least be able to quickly report how much information they currently have.

More formally: suppose we have some student s. Let $f(s)$ denote the amount of food s should get (assume some consistent unit of food throughout this problem). Devise a data structure which can handle the following operations.

- *Update*(s_1, s_2, x): Given two students s_1 and s_2 such that $f(s_1)/f(s_2) = x$, update the data structure with this knowledge.
- *GetRatio*(s_1, s_2): Given two students s_1 and s_2, return their relative food ratio $f(s_1)/f(s_2)$ or "insufficient information" if this information is not yet known.
- *SetupFood*(s, f): Given a student s, and the amount of food f they eat, report whether or not there is sufficient information to set up all the food. (In particular, if App Bonatite knows the ratios between all students, then they can set up all the food).

Design and describe a data structure that can handle the *Update*, *GetRatio* and *SetupFood* operations in amortized $O(\alpha(n))$ time.

Solution: The key is that we can use transitivity of knowledge about relative quantities of information. That is, for students s_1, s_2, and s_3, if $f(s_1)/f(s_2) = x$ and $f(s_2)/f(s_3) = y$, then $f(s_1)/f(s_3) = xy$. Consider the graph on n vertices where vertices i and j are connected by an edge if Update(s_i, s_j, x) has been called at some point. Then for two additional vertices k and l, it is possible to calculate the relative food ratio $f(s_k)/f(s_l)$ if and only if vertices k and l are in the same connected component of the graph. Furthermore, *SetupFood*() is equivalent to detecting whether or not this graph of students is connected. This motivates us to use the union-find data structure (with path compressions), and augment it in some way to store the food ratios.

For each node s, store the attributes $s.size$ and $s.ratio$. Define $s.size$ to equal the size of the subtree rooted at s. Define $s.ratio$ to equal $f(s.parent)/f(s)$ (for nodes without a parent, set $s.ratio = 1$). Initially, there are n disjoint students each with size 1 and ratio 1. (i.e. there are n calls to MAKE-SET).

Next, we must demonstrate how to augment path compression in order to update the values of $s.size$ and $s.ratio$ for each of the nodes that are modified. If we compress some path $s_1, s_2, \ldots, s_k$ (e.g. as would occur during some FIND operation), then we want to

update each pointer so that $s_1.parent = s_k, s_2.parent = s_k, \ldots, s_{k-1}.parent = s_k$. At the same time, we also update each size attribute so that $i_\ell.size = i_\ell.size - i_{\ell-1}.size$ for $\ell \in \{k-1, \ldots, 2\}$. Furthermore, we update the ratio attribute as $s_\ell.ratio = s_{\ell+1}.ratio \cdot s_\ell.ratio$ for $\ell \in \{k-1, \ldots, 1\}$ so that the ratio of each child to its parent is maintained. Note that i_k is without a parent and, therefore, $i_k.ratio = 1$. Both operations add only constant extra time for each node in the path, and that the size of i_k does not change. Therefore the amortized runtime of our algorithm does not change.

Finally, we describe how to implement each of the above 3 operations:

Now, after a FIND(s) operation, s is pointing to its representative r and, therefore, we know the ratio $f(r)/f(s)$.

- **Update**(s_1, s_2, x): Call FIND on s_1 and s_2 to get their representatives r_1 and r_2, respectively. If $r_1 = r_2$, then we already have the ratio information. Otherwise, choose the new root to be the one with larger size, say r_1 without loss of generality. Then, $r_2.parent = r_1$ and set

$$r_1.size = r_1.size + r_2.size$$

$$r_2.ratio = \frac{f(r_1)}{f(r_2)} = \frac{f(r_1)}{f(s_1)} \cdot \frac{f(s_2)}{f(r_2)} \cdot \frac{f(s_1)}{f(s_2)} = \frac{s_1.ratio}{s_2.ratio} \cdot x.$$

- **GetRatio**(s_1, s_2): Call FIND on s_1 and s_2 to get their representatives r_1 and r_2, respectively. If $r_1 = r_2 = r$, we have that

$$\frac{f(s_1)}{f(s_2)} = \frac{f(s_1)}{f(r)} \cdot \frac{f(r)}{f(s_2)} = \frac{s_2.ratio}{s_1.ratio},$$

  so we simply return $s_2.ratio/s_1.ratio$. Otherwise, $r_1 \neq r_2$ and we return "insufficient information."

- **SetupFood**(s, f): Call FIND on s to get its representative r. If $r.size = n$, then return *true*; otherwise, return *false*.

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

`https://forms.gle/2BN8VGMChPYFBpwX7`.

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