

Solution Set 7

Problem 1.

1.a This is the famous *Belady's anomaly* [1]. Heres one bad sequence:

step#	request	FIFO ₃	miss?	FIFO ₄	miss?
1	A	A	•	A	•
2	B	BA	•	BA	•
3	C	CBA	•	CBA	•
4	D	DCB	•	DCBA	•
5	E	EDC	•	EDCB	•
6	B	BED	•	EDCB	
7	C	CBE	•	EDCB	
8	A	ACB	•	AEDC	•
9	B	ACB		BAED	•
10	C	ACB		CBAE	•
11	F	FAC	•	FCBA	•
12	G	GFA	•	GFCB	•
13	A	GFA		AGFC	•

Here, FIFO with 3 slots faults 10 times and FIFO with 4 slots faults 11 times (note that 7 faults are unavoidable in this case, as there are 7 different pages in the request sequence).

2.a The monotonicity property of LRU is proven by the following observation.

Claim: Let $k < k'$. Then at any time, the contents of the fast memory under LRU_k is a subset of the contents of the fast memory under $\text{LRU}_{k'}$.

Proof: Fix a request sequence. Define, for each page p , at each step t , its *last index*, denoted $\text{last}_t(p)$, to be the number of steps since p was used before t . For example, at $t = 13$ for the sequence tabulated above, we have $\text{last}_{13}(G) = 1$ since it was last used before step 13 at step 12, $\text{last}_{13}(F) = 2$ since it was last used at step 11, $\text{last}_{13}(C) = 3$ etc. Clearly, $\text{last}_t(p)$ does not depend on the algorithm: it depends only on the request sequence. Now, consider the pages, at each step t , in increasing order of their last_t index. It is immediate from definitions that the pages in the fast memory of LRU_k are exactly the top k pages in that sorted list, because a $\text{last}_t(p) < \text{last}_t(q)$ if and only if p was used more recently than q (note that ties are impossible). It therefore follows that if $k < k'$, then at time t the fast memory of LRU_k holds the pages with the k smallest last indices, while $\text{LRU}_{k'}$ holds the pages with the k' smallest last indices.

Problem 2. Suppose that the cache size is $k = 2$ and it is initially empty. Consider the request sequence 1, 2, 3, 2, 3, 2, 3, LIFO evicts 2 at each request of 3, and evicts 3 at each but the first request of 2. If the request sequence length is n , then LIFO faults n times, but the optimal algorithm (which coincides with LRU and with FIFO for this sequence) faults only at the first three requests.

Problem 3. If we reduce the weight of the erring experts by a factor x , then every error we make reduces the total weight from w_i to $w_{i+1} \leq w_i(1 - x/2)$, and if we denote the number of errors made by the algorithm by M , we have that

$$w_t \leq w_0 \left(1 - \frac{x}{2}\right)^M = n \left(1 - \frac{x}{2}\right)^M . \quad (1)$$

Since we know that there is at least one expert who makes no more than m errors, that expert's weight is at least $(1 - x)^m$. Therefore the total weight of experts satisfies

$$w_t \geq (1 - x)^m . \quad (2)$$

Combining Eq. (1) and Eq. (2) we obtain $(1 - x)^m \leq n \left(1 - \frac{x}{2}\right)^M$, which is equivalent to

$$M \leq \frac{1}{\log \frac{2}{2-x}} \left(\log n + m \log \frac{1}{1-x} \right) .$$

Problem 4. We use Yao's Lemma. To this end, we define a probability distribution over the inputs: Given d , we let the treasure be at distance d in any of the k roads with equal probability $1/k$. Let us now consider a deterministic algorithm. Any such algorithm can be specified by a sequence of pairs (i, s) , where $1 \leq i \leq k$ is the road to take, and s is the distance from the junction which the algorithm advances on road i . In words, the algorithm is "take road i_1 to distance s_1 ; if the treasure not found, turn back and take road i_2 to distance s_2 , etc."

With this model, the lower bound is easy. Fix any deterministic algorithm. For each direction $1 \leq i \leq k$, consider only the first pair (i, s) which $s \geq d$. Clearly the treasure will be found in one of them: the question is which. If the right direction is the j th in this sequence, the algorithm clearly pays at least $j \cdot d$ units. Since the direction is chosen randomly, we can bound the expected cost from below by

$$\sum_{i=1}^k \frac{1}{k} (i \cdot d) = \frac{d}{k} \cdot \frac{k(k+1)}{2} = d \cdot \frac{k+1}{2} .$$

On the other hand, the expected optimal cost is d , namely the competitive ratio, on average over the inputs, is $\frac{k+1}{2}$. Applying Yao's Lemma, we conclude that any randomized algorithm has an instance, for every $d > 0$, in which the expected cost of the algorithm is at least $\frac{d(k+1)}{2}$ whereas the optimal cost is d . It follows that the competitive ratio for the k -way treasure hunt is $\Omega(k)$.

References

- [1] L. A. Belady, R. A. Nelson, and G. S. Shedler. An anomaly in space-time characteristics of certain programs running in a paging machine. *Communications of the ACM*, 12:349–353, June 1969.