ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2013 Prof. R. Wattenhofer / K.-T. Foerster, T. Langner, J. Seidel

Discrete Event Systems

Solution to Exercise Sheet 12

1 Bin Packing

The algorithm mentioned in the exercise is 2-competitive. The proof works as follows: Consider the bins in the order in which they were closed. Consider two consecutive bins i and $i+1$. Assume that the algorithm fills bin i up to level $x \leq 1$. The next item (the first to be put into bin $i+1$) must be of size *larger* than $1-x$. Otherwise, the algorithm would not have opened a new bin. Therefore, any *two consecutive bins* must have total size strictly more than 1. Because the optimum must also use at least one bin to store this size, the algorithm requires at most twice as many bins.

To show that there is indeed a sequence which the optimum can serve using half the number of bins as the algorithm, assume the following input sequence:

$$I = \underbrace{\left(1, \frac{2}{n}, 1, \frac{2}{n}, 1, \frac{2}{n}, \ldots, 1, \frac{2}{n}\right)}_{n \text{ items}}.$$

Clearly, our algorithm ALG needs to open a new bin after every item. That is, the number of bins opened by ALG is $\text{cost}_{\text{ALG}}(I) = n$. On the other hand, the optimal algorithm OPT can put all $n/2$ objects of size 1 into one bin each and all the $n/2$ objects of size $2/n$ together into one bin. Hence, $\text{cost}_{\text{OPT}}(I) = n/2 + 1$. The competitive ratio c for input sequence I is therefore

$$c \geq \frac{\text{cost}_{\text{ALG}}(I)}{\text{cost}_{\text{OPT}}(I)} = \frac{n}{\frac{n}{2}+1} = \frac{2n}{n+2}.$$

For large n, this tends to 2.

2 Paging

a) (i) FIFO *(First-in/First-out): Replace the page that has been in the cache longest.*

The FIFO strategy is 3-*competitive*. In order to prove this, observe that on any consecutive input subsequence containing three or fewer distinct page references, FIFO incurs three or fewer page faults. Now, consider a 3-*phase partition* of the input sequence I. A 3-phase partition is defined as follows: Phase 0 is the empty sequence. For every $i \geq 1$, phase i is the maximal sequence following phase $i-1$ that contains at most three distinct page requests; that is, if it exists phase $i+1$ begins on the request that constitutes the fourth distinct page request since the start of the i-th phase.

By the above observation, it is clear that FIFO incurs at most three page faults for any phase $i \geq 1$, because it cannot fault twice on the same page. For any $i \geq 1$, let q be the first request of phase i and consider the input sequence starting with the second request of phase i up to and including the first request of phase $i+1$. The

optimal algorithm OPT can only have two additional pages cached (on top of q), and there are three different requests (except if phase i is the last phase) in this sequence not counting request q. Hence, OPT must incur at least one page fault in this phase. Combining the two bounds, we have

$$\text{cost}_{\text{FIFO}}(I) \leq 3 \cdot \text{cost}_{\text{OPT}}(I) + \alpha,$$

where $\alpha \leq 3$ is the maximal number of page faults incurred by FIFO in the last phase. A sequence for which FIFO actually yields three times as many page faults as OPT is given by

$$I = (p_1,\ p_2,\ p_3,\ p_4,\ p_1,\ p_2,\ \ldots)\ .$$

(ii) LFU *(Least Frequently Used): Replace the page that has been requested the smallest number of times since entering the fast memory.*
The LFU strategy is *not competitive*. Consider the following request sequence:

$$I = (p_1,\ p_1,\ p_2,\ p_2,\ p_3,\ p_4,\ p_3,\ p_4,\ p_3,\ \ldots)$$

In this sequence, LFU keeps on exchanging p_3 and p_4 ad infinitum, while the optimum can keep these two pages in the cache.

(iii) LIFO *(Last-in/First-out): Replace the page most recently moved to the cache.*
For the same reason as LFU, the LIFO strategy is also *not competitive*. Consider the following request sequence:

$$I = (p_1,\ p_2,\ p_3,\ p_4,\ p_3,\ p_4,\ p_3,\ \ldots)$$

In this sequence, LIFO keeps on exchanging p_3 and p_4 ad forever. It is therefore not competitive.

(iv) LRU *(Least Recently Used): When eviction is necessary, replace the page whose most recent request was the earliest.*
Like the FIFO strategy, LRU is 3-*competitive*. The reason is that (like FIFO), LRU has the property that on any consecutive input subsequence containing three or fewer distinct page references, it incurs at most three page faults. The remainder of the proof is then equivalent to the FIFO case, including the worst-case input sequence.

(v) FWF *(Flush When Full): Whenever there is a page fault and there is no space left in the cache, evict* all *pages currently in the cache.*
The FWF algorithm is also 3-*competitive*. Consider the first phase, i.e., the first consecutive input subsequence containing three distinct page references. Clearly, FWF does not operate a flush. Now, consider the subsequent phase. In this phase, there can be at most one flush and hence, three page faults. Similarly, in every subsequent phase, there can be at most one flush and three page faults. From this observation, the proof follows like in the FIFO case, again including the worst-case input sequence for the FIFO algorithm.

b) We prove the following theorem:

Theorem. *There exists no deterministic online paging algorithm* ALG *with a competitive ratio better than* 3.

Proof. Assume that there are four pages, $p_1, \ldots, p_4$. We prove that there is an arbitrarily long request sequence I for which $|I| = \text{cost}_{\text{ALG}}(I) \geq 3 \cdot \text{cost}_{\text{OPT}}(I)$. Without loss of generality, assume that ALG initially holds p_1, p_2, and p_3 in its cache. We define a "cruel" request sequence $I = (r_1, r_2, r_3, \ldots)$ inductively: $r_1 = p_4$, and r_{i+1} is defined to be the unique page that is not in ALG's cache just after serving the request sequence $r_1, \ldots, r_i$. In other words, the adversary always requests the one page that ALG does not have in its cache. Clearly, I can be made arbitrarily long and ALG has a page fault on each request in I, hence $|I| = \text{cost}_{\text{ALG}}(I)$.

We now show, however, that it is possible to serve every request sequence I with at most $|I|/3$ page faults. Specifically, consider the offline algorithm NRL that knows the complete request sequence in advance and always evicts the one page from the cache whose *next request is latest*. Suppose that for serving the i-th request r_i, NRL evicts the page p. By the definition of NRL, and since there are four pages in total, it must be that all the pages in the cache (except perhaps r_i) must be requested prior to the next request of p. Hence, NRL has a page fault at most once every three requests.

Clearly, it holds that

$$\text{cost}_{\text{OPT}}(I) \leq \text{cost}_{\text{NRL}}(I) \leq \frac{|I|}{3} = \frac{\text{cost}_{\text{ALG}}(I)}{3},$$

which finishes the proof of the theorem. □

3 Memory

a) Turn over every card once in n moves. Then collect all n pairs. This is 2–competitive.

b) There exists a $(2n-1)$–competitive strategy and $(2n-1)$–competitive is also a lower bound for any deterministic strategy:

(i) No strategy can guarantee to collect all pairs in a solitaire Memory game in less than $2n-1$ moves: We can assume that if no uncollected pairs are known to the player, then she will turn over a new card in the first step of her move. We can also assume that if the first card has no known matching partner, that then the player will turn over another unknown card as the second step of her move. Deviating from these rules would not decrease the number of needed moves. In a similar fashion, we can assume that if a pair of cards is known and not collected after a move, that the player will collect them at the end – doing it at an earlier point would incur the same costs of one move per pair.

The cards could be positioned in such a way, that the player will open the cards in the following order:

$$(1,2),(3,1),(4,2),(5,3),\ldots,$$
$$(n-2,n-4),(n-1,n-3),(n,n-2),(n,n-1)\ .$$

From the first $\{(1,2)\}$ to the $(n-1)$th move $\{(n,n-2)\}$, the first card that is turned over is always an unknown card that has no known matching partner. Therefore, the only viable option is to turn over a second card during that move. From the second to the $(n-1)$th move, this reveals a card that has a known matching partner from a previous move. Also in the first move, no pair is collected. Therefore, the player needs at least $(n-1)$ moves where no pair is made. Since collecting the pairs needs n additional moves at the end, each strategy needs at least $(n-1)+n = 2n-1$ moves to finish collecting all pairs

(ii) There exists a strategy that can guarantee to collect all pairs in a solitaire Memory game in at most $2n-1$ moves. We use the following strategy: First the player turns over $(2n-2)$ cards in $(n-1)$ moves. Due to the pigeonhole principle, the player now knows the location of at least $(n-2)$ pairs (or has collected some of them already), since only two cards were not turned over yet. In the worst case, we can now collect these $(n-2)$ pairs with another $(n-2)$ moves, needing $(n-1)+(n-2) = 2n-3$ moves in total so far. The player now turns over one of the last two remaining cards. If the players knows the location of the matching partner, she can collect this pair. Else, since only one card was not turned over yet, this last card is the matching partner. Since now only 2 cards are not collected on the table, they must be a pair. Adding these two moves to $2n-3$ previous moves results in total in $2n-1$ moves.

c) Assume that there are still i pairs left and let X_i be a random variable indicating the number of moves until the next pair is found. After the first card has been picked, there are still $2i-1$ other cards on the table. So the probability to pick the matching card in the current move is $p = \frac{1}{2i-1}$. As X_i is geometrically distributed with parameter p, we have $E[X_i] = \frac{1}{p} = 2i - 1$.

Let X be the number of moves to find all pairs and we have $X = \sum_{i=1}^{n} X_i$. By linearity of expectation, we calculate the expected number of moves to find all pairs as

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} (2i-1) = n^2 \ .$$