Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
M. Ahmadi, A. R. Molla, O. Saukh January 28, 2016

Algorithm Theory, Winter Term 2015/16
Problem Set 13

hand in (hard copy or electronically) by 10:15, Thursday February 04, 2016, tutorial session will be on February 08, 2016

Exercise 1: LRU with Potential Function (5 points)

When studying online algorithm, the total (average) cost for serving a sequence of requests can often be analyzed using amortized analysis. In the following, we will apply this to the paging problem and you will analyze the competitive ratio of the LRU algorithm by using a *potential function*. Recall that a potential function assigns a non-negative real value to each system state. In the context of online algorithms, we think of running an optimal offline algorithm and an online algorithm side by side and the system state is given by the combined states of both algorithms.
Consider the LRU paging algorithm, i.e., the online paging algorithm that always replaces the page that has been used least recently. Let $\sigma = (\sigma(1), \sigma(2), \dots, \sigma(m))$ be an arbitrary request sequence of pages. Let OPT be some optimal offline algorithm. You can assume that OPT evicts at most one page in each step (e.g., think of OPT as the LFD algorithm). At any time t (i.e., after serving requests $\sigma(1), \dots, \sigma(t)$), let $S_{\text{LRU}}(t)$ be the set of pages in LRU's fast memory and let $S_{\text{OPT}}(t)$ be the set of pages contained in OPT's fast memory. We define $S(t) := S_{\text{LRU}}(t) \setminus S_{\text{OPT}}(t)$. At each time t, we further assign an integer weight $w(p,t)$ from the range $\{1, \dots, k\}$ to each page $p \in S_{\text{LRU}}(t)$ such that for any two pages $p, q \in S_{\text{LRU}}(t), w(p,t) < w(q,t)$ iff the last request to p occurred before the last request to q (i.e., the requests in S_{LRU} are numbered from $1, \dots, k$ according to times of their last occurrences). Recall that we use k to denote the size of the fast memory. We define the potential function at time t to be

$$\Phi(t) := \sum_{p \in S(t)} w(p,t).$$

We define the amortized cost $a_{\text{LRU}}(t)$ for serving request $\sigma(t)$ as

$$a_{\text{LRU}}(t) := c_{\text{LRU}}(t) + \Phi(t) - \Phi(t-1),$$

where $c_{\text{LRU}}(t)$ is the actual cost for serving request $\sigma(t)$. Note that $c_{\text{LRU}}(t) = 1$ if a page fault for algorithm LRU occurs when serving request $\sigma(t)$ and $c_{\text{LRU}}(t) = 0$ otherwise. Similarly, we define $c_{\text{OPT}}(t)$ to be the actual cost of the optimal offline algorithm for serving request $\sigma(t)$. Again, $c_{\text{OPT}}(t) = 1$ if OPT encounters a page fault in step t and $c_{\text{OPT}}(t) = 0$ otherwise. In order to show that the competitive ratio of the algorithm is at most k, you need to show that for every request $\sigma(t)$,

$$a_{\text{LRU}}(t) \leq k \cdot c_{\text{OPT}}(t).$$

Exercise 2: (Generalized) Online Load Balancing (7 points)

In the lecture, a simple greedy algorithm was presented for an instance of the load balancing problem where the *speeds* of all machines are assumed to be equal. This algorithm gives a factor 2 approximation for the minimum makespan. As we have seen, it is not necessary to sort the jobs at the beginning of the algorithm and we can always arbitrarily pick up a job and assign it to the machine with the smallest load. Hence, the greedy algorithm can also be used as an online algorithm for the corresponding online load balancing problem where the jobs arrive in an online fashion one by one. The algorithm then achieves a competitive ratio of at most 2.

Now we are interested to find an online algorithm with constant competitive ratio for an generalization of the above online load balancing problem where the *speeds* of machines are different; as before jobs arrive one at a time in an online fashion. Assume that there are m machines and that for a given job j, $t_i(j)$ is the processing time when running job j on machine i. For each $1 \leq i \leq m$ and $1 \leq j \leq n$, we assume that

$$t_i(j) := \frac{w(j)}{v(i)},$$

where *weight* $w(j)$ depends only on job j and *speed* $v(i)$ depends only on the machine i.
smallskip
Consider the following procedure we run after job j arrives: assume that the machines are indexed according to **increasing speed**. Let $\vec{T}(j) := (T_1(j), T_2(j), \ldots, T_m(j))$ denote the loads of the m machines after job j is assigned.

```
Procedure Assign($\vec{T}(j-1), \lambda$)
    /* $\lambda$ is a given parameter.
    Let $S := \{i : T_i(j-1) + t_i(j) \leq 2\lambda\}$;
    if $S = \emptyset$ then
        $b :=$ fail
    else
        $k := \min\{i : i \in S\}$;
        $T_k(j) := T_k(j-1) + t_k(j)$;
        $b :=$ success
    end
    return $(\vec{T}(j), b)$.
```

Hence, in `Assign(`$\vec{T}$`, `λ`)`, job j is assigned to the slowest machine such that the load on this machine is still below 2λ after the assignment.

(a) (4 points) Show that if $\lambda \geq T^*$ (where T^* is the optimal makespan), the procedure `Assign` always succeeds (i.e., it never returns **fail**). As a consequence, if we knew the optimal makespan T^*, choosing $\lambda = T^*$ would lead to a competitive ratio of 2.

Hint: *By way of contradiction, assume that some job t cannot be assigned. Then show that there must exist some job s (which was assigned earlier than t) which could have been assigned to a slower machine.*

(b) (3 points) Using the result of (a), devise an online algorithm with $\mathcal{O}(1)$ competitive ratio (for the case when the optimal makespan is not known). using `Assign` repeatedly (with changing λ) for solving an instance of online load balancing where the speed of machines are not necessarily equal.

Hint: *Use `Assign` repeatedly (with changing λ). Define phases such that at the beginning of the first phase, λ is set to be equal to the load (processing time) generated by the first arrival job on the fastest machine.*

Remark: *Note that if you could not solve (a), you can still try to solve (b).*