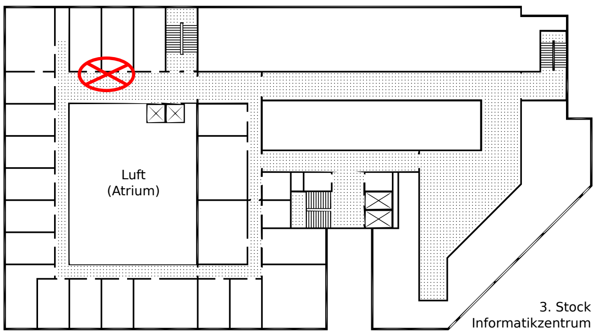
Abteilung Algorithmik **Summer term 2018**
Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig

Prof. Dr. Sándor P. Fekete
Phillip Keldenich

Online Algorithms
2nd Homework Assignment, 7th of May 2018

Solutions are due Monday, the 28th of May 2018, until 1:15 PM in the homework cupboard. You can also hand in your solution in person before the small tutorial begins. If you cannot hand in the homework in person, you can also hand it in via e-mail to both `j.heroldt@tu-bs.de` *and* `keldenich@ibr.cs.tu-bs.de`. Please clearly label your solutions using your name and matriculation number.



Due to the excursion week, you have more time than usual to hand in your solution.

Exercise 1 (The List Update Problem I): In this exercise, we consider the List Update Problem. Suppose that we are storing a set $S = \{s_1, \ldots, s_n\}$ of values as unsorted linked list. Our input sequence consists of queries $s_i \in S$ that ask us to find a certain element in our list. In order to find an element, we iterate through the list starting from the front and return the element once we find it. Because iterating through our list takes time, we incur a cost of 1 for each element that we touch during the search. For example, searching for 2 in the list $[1, 2, 3]$ costs two units, and searching for 2 in $[2, 1, 3]$ costs just one unit.

After each query for an item s_i, we are allowed to move the item s_i to any point closer to the front of the list; this exchange does not cost us anything. For example, if we search for 2 in the list $[1, 4, 2, 3]$, we can change the list to $[2, 1, 4, 3]$, $[1, 2, 4, 3]$ or $[1, 4, 2, 3]$ after finding 2 for a total cost of three units.

Our goal is to devise an online algorithm for this problem; after each request, the algorithm has to decide where to move the requested item. Intuitively speaking, we want our algorithm to maintain its list such that frequently requested elements are closer to the front than less frequently requested elements.

a) After each request, the algorithm Transpose swaps the requested element with the preceeding element in the list. For example, after searching 2 in the list $[1, 4, 2, 3]$, the list becomes $[1, 2, 4, 3]$; searching for 2 again results in the list $[2, 1, 4, 3]$. Prove that Transpose is not c-competitive for any constant c.

b) The algorithm FrequencyCount maintains a frequency count for each element that is incremented each time the element is requested. After each request, it moves the requested item such that the list is in nonincreasing order of frequency count. Prove that FrequencyCount is not c-competitive for any constant c.

(5+5 points)

Exercise 2 (The List Update Problem II: Move to Front): In this exercise, we consider the MoveToFront algorithm for the List Update Problem. After each request s_i, the algorithm moves the requested item s_i to the front of the list. For example, after searching for 2 in $[1, 4, 2, 3]$, the list becomes $[2, 1, 4, 3]$.

a) Prove that there is no constant $c < 2$ such that Move To Front is c-competitive.

b) Prove that Move To Front is 2-competitive.

   Hint: Use the number of inversions in Move To Front's list w.r.t. OPT's list after request i as a potential function $\phi(i)$. An inversion is a pair x, y of elements such that x comes before y in Move To Front's list, but after y in OPT's list.

   In order to prove $c_{\text{MTF}}(i) + \phi(i) - \phi(i-1) \leq 2c_{\text{OPT}}(i)$ for a request s_i, consider the number of items k that come before s_i in both OPT's and Move To Front's list and the number of items ℓ that come before s_i in Move To Front's list but after s_i in OPT's list.

(10+17 points)

Exercise 3 (Directed Graph Exploration): In this exercise, we consider the problem of exploring a strongly connected directed graph. Each vertex is labeled by a natural number $\{1, \dots, n\}$. We do not know the incoming and outgoing edges of a vertex before visiting it for the first time; each time we visit a vertex, we learn its outgoing edges. We start at vertex 1 and initially know the outgoing edges of 1. The goal is to visit each vertex at least once and return to the start by following the directed edges according to their direction. Traversing each edge incurs a cost of 1.

a) Prove that no deterministic online algorithm has a competitive ratio better than $\frac{n+1}{2} - \frac{1}{n}$.

b) Devise a strategy that has competitive ratio $\frac{n+1}{2} - \frac{1}{n}$ and prove this upper bound.

(8+15 points)