Data Structures and Programming
Spring 2019
Programming Assignment #1
Due: see class web page

The purpose of this assignment is to evaluate four on-line heuristic algorithms, namely, *move-to-front* (MTF), *bit*, *transpose*, and *frequency count* (FC) on singly linked lists. The problem is as follows. Given a list L consisting of n distinct elements $x_1, x_2, ..., x_n$, and a sequence of *requests* $y_1, y_2, ..., y_k$ (where $y_1, y_2, ..., y_k \in \{x_1, x_2, ..., x_n\}$), the problem is to design a singly linked list data structure for $x_1, x_2, ..., x_n$ so as to minimize the number of *comparisons*. (For example, in Fig.1 it takes 4 time units to find key 8.) It can be shown that in an off-line environment (i.e., $y_1, y_2, ..., y_k$ are given beforehand), the optimal strategy is to arrange $x_1, x_2, ..., x_n$ in a descending order according to their frequencies of occurrences in the request sequence. Now suppose requests $y_1, y_2, ..., y_k$ are given one at a time (i.e., in an on-line fashion). *Move-to-front*, *transpose*, *BIT*, and *FC* are four on-line algorithms that have been proposed in the literature to cope with the above on-line version of the list access problem. (See Figs 2 and 3 for example.) For FC (Frequency Count), for each item maintain a frequency count of the number of accesses to it - when an element is accessed increase its frequency count and reorder the list in the decreasing order of frequencies. Algorithm BIT works as follows. Associated with each element of the list is a bit which is complemented whenever that item is accessed. We let $b(x)$ denote the bit corresponding to an item x. If an access causes a bit to change to 1 the accessed item is moved to the front, otherwise the list remains unchanged. The n bits are initialized uniformly at random. Roughly speaking BIT is *"move-to-front every other access."* Notice that BIT uses n random bits regardless of the length of the request sequence.

In this assignment, L is assumed to be $\{0, 1, ..., 9\}$. Write a program to perform the following:

1. Consider a sequence of requests of length 1000 (roughly) given on the website.
2. Calculate the cost of the optimal off-line algorithm described above. (To this end, first you have to find the the frequency of occurrences for each key in the generated request sequence.)
3. Assuming that L is $0 \to 1 \to \cdots \to 9$ initially, calculate the cost with respect to the request sequence generated in Step 1 under the move-to-front heuristic.
4. Repeat Step 3 for the transpose heuristic.
5. Repeat Step 3 for BIT. (Note that you need to generate a random sequence of 10 binary bits before the list access starts.
6. Repeat Step 3 for FC.
7. Output the following table (the exact format will be announced later)

Cumulated # comparisons	Optimal	MTF	Transpose	BIT	FC
i=50					
i=100					
...					
i=1000					

8. (*** **Bonus** ***) Can you think of a fourth (fifth, ...) on-line heuristic algorithm? If so, repeat Steps 1-6 for your algorithm(s). (You are encouraged to give this a try; do not have to be concerned with the performance of your algorithms(s), although you should try your best to come up with a "good" design. It is the effort that counts! However, do not turn in trivial algorithms such as 'DO NOTHING', which **trivially** will not improve your grade.)

   Another possible extension is to try other request sequences. For instance,

   (a) a random sequence of requests of length 1000, 10000, ...

   (b) a random sequence of requests of length 1000, 10000, ... so that the relative frequencies of occurrences for 0, 1, ..., 9 (roughly) follow the following ratios: 1 : 2 : ... :10.

<u>**Notes**</u>
You should follow the programming practices recommended in Programming Languages or equivalent courses.

TURN IN: TBA

IMPORTANT NOTES:

- You are strongly advised to start on this project immediately.
- **You are not allowed to collaborate with anyone in any way on this project. This rule will be strictly enforced.**

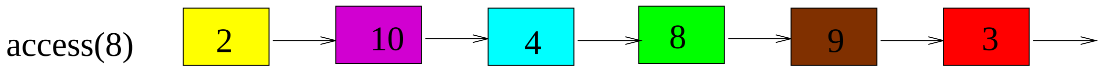

Fig. 1

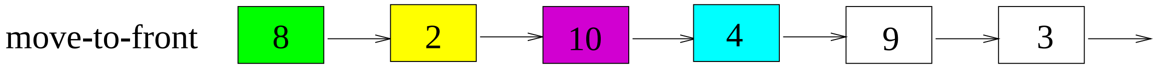

Fig. 2

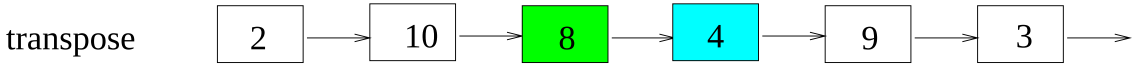

Fig. 3