CS 624: Analysis of Algorithms

Assignment 6

Due: Monday, March 22, 2021

1. Exercise 3.1 in the Lecture 11 handout[1].

2. Exercise 3.2 in the Lecture 11 handout.

   You need to think carefully about this problem. Here are some sequences of A's and B's, all having limiting frequencies for the symbol "A". Study this list of examples carefully before you start:

sequence	limiting frequency of A's
A, A, A, A, A, A, ...	1
A, B, A, B, A, B, ...	1/2
A, B, B, A, B, B, A, B, B, A, ...	1/3
A, B, A, B, B, A, B, A B, B, A, B, A, B, B, ...	2/5
A, B, A, B, B, A, B, B, B, A, B, B, B, B, A, ...	0
B, B, B, B, B, B, ...	0

   The problem is asking you to produce a sequence (and it can be a sequence consisting of just A's and B's) for which there is **no** limiting frequency for the entry "A". You might think this can't be done. But it can, and it's not really all that hard. Once you get the idea, it's very simple.

   Be careful: to say that there is "no limiting frequency" does **not** mean that the limiting frequency is 0. The table above in fact shows two sequences in which the limiting frequency is 0. Neither one of them has "no limiting frequency".

   And just to give you an idea of the kind of thing you might do: you don't necessarily have to write down a simple formula. You could write something like this: "If $1 \leq n \leq 10$, then $\langle$something$\rangle$," and continue in that sort of manner.

3. Consider the MOVE-TO-FRONT algorithm applied to a sequence of 4 elements whose initial state is $\{A, B, C, D\}$. Suppose the sequence of accesses is as follows:

$$B, C, A, C, C, A, D, D, A, B, A, C, B, C$$

   - Show that the cost using MOVE-TO-FRONT with this sequence of accesses is 56, and show the final state of the list.
   - What can you say (based on the Sleator-Tarjan theorem) about the cost of the best possible algorithm applied in the same situation?
   - Could there actually be an algorithm that was that good?

[1] Be careful: "Show that ..." really means "Prove that ...".

4. (a) Read Section 10.1. This should be very easy, since I assume you already know about stacks and queues. The text has an implementation of a queue in terms of an array of fixed size, in which the operations ENQUEUE and DEQUEUE each have cost $O(1)$. (Remember that this is just a fancy way of saying that the cost is uniformly bounded by some constant.) You don't have to write anything—just read this and understand it. Let me know if you find anything confusing.

   (b) The only problem with that implementation of queues is that the size of a queue is bounded by the size of the array. It would be nicer to have a queue which, like a stack, had no upper bound on its size. (Now of course in practice, a stack *does* have an upper bound on its size—that's why we talk about "stack overflow". But also in practice we have ways of dealing with this so that except in very unusual circumstances, stacks act as if their size was not constrained.)

   So assuming that we had an implementation for stacks in which their size was not constrained, and in which the operations PUSH and POP each have cost 1, I first want you to do Exercise 10.1-6 (page 236). The idea of this exercise is to provide an implementation of a queue in which the size of the queue is not constrained. In particular, you should derive the worst-case costs of ENQUEUE and DEQUEUE, and it should be clear that the cost of DEQUEUE is *not* $O(1)$. Presumably, that's the price we pay for having a queue of unlimited size.

   (c) Then finally, do Exercise 17.3-6 (page 463). The idea of this exercise is show that in the implementation of the queue in Exercise 10.1-6 (which you just did), the *amortized* cost of both ENQUEUE and DEQUEUE for this data structure *is* $O(1)$. So things really are not so bad—in fact, they're pretty good.