Lab09-AmortizedAnalysis

Exercises for Algorithms by Xiaofeng Gao, 2016 Spring Semester

Name:_____ Student ID:_____ Email: _____

1. We can implement a FIFO queue Q using two stacks S_1 and S_2 as follows:

   - enqueue(x): PUSH x onto stack S_1.
   - dequeue(): if stack S_2 is not empty, pop it. Otherwise, we POP every element of S_1 and PUSH it into S_2 in their respective order (so while $|S_1| \neq 0$, PUSH(POP(S_1),S_2)). Next simply POP from S_2 and return the result.

   Cost model: assume that a PUSH and a POP each has a cost of 1.

   (a) Using the accounting method, find the amortized costs of each operation in Q.

   (b) Using the potential method, find the amortized costs of each operation in Q.

2. Here is a "dictionary" data structure that supports efficient insert and lookup operations. It consists of a collection of arrays, where array i has size 2^i. Each array is either empty or full, and each is in sorted order. The answer of which arrays are full and which are empty is based on the binary representation of the number of items to be stored. For example, if we had 13 items (where $13 = 1 + 4 + 8$), then the arrays of size $1, 4$, and 8 would be full and the rest empty. It may look like this:

   A0: [5]
   A1: empty
   A2: [1,4,7,19]
   A3: [2,6,9,12,13,16,20,25]

   (a) What is the time complexity of the lookup operation in terms of O notation?

   (b) How about the insertion? We do it like mergesort. To insert the number 10, we first create an array of size 1 that just has this single number in it. We then look to see if $A0$ is empty. If it is, we make this array be $A0$. If not, we merge this array with $A0$ to create a new array (which is $[5, 10]$ here) and look to see if $A1$ is empty, and so on. For the example above, the insertion will stop at $A1$.

   Cost model: assume that creating the initial array of size 1 costs 1, and merging two arrays of size m costs $2m$ (merging is a linear-time step).

   Using aggregate analysis, find the amortized costs of the insertion.

3. You have seen amortized analysis of the binary counter in class. Here, we introduce the redundant counter, where each digit may be $-1, 0$, or 1. The underlying base is still 2. For example, we can represent a decimal number 3 as 11_2, or $10(-1)_2$.
   The increment operation of such a counter is analogous to that on binary counter. We add 1 to the low order bit (the "bit" here has 3 optional values). If the result is 2 then it is changed to 0, and a carry of 1 is propagated to the next bit. We repeat this procedure until no carry results. The decrement operation is similar. We subtract 1 from the low order bit. If it becomes -2 then it is changed to 0, and a carry of -1 is propagated.
   Cost model: assume that the cost of changing the value of a bit is 1.

   (a) Using the accounting method, find the amortized costs of the two operations.

   (b) Using the potential method, find the amortized costs of the two operations.