ShanghaiTech University

CS240 Algorithm Design and Analysis

Fall 2021

Problem Set 4

Due: 23:59, Dec.17, 2021

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" of Gradescope, set your FULL NAME to your Chinese name and enter your STUDENT ID correctly.
3. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
4. When submitting your homework, match each of your solution to the corresponding problem number.

Problem 1:

To implement a TABLE-DELETE operation, it is simple enough to remove the specified item from the table. It is often desirable, however, to contract the table when the load factor(number of elements in a table/table size) of the table becomes too small, so that the wasted space is not exorbitant. Table contraction is analogous to table expansion: when the number of items in the table drops too low, we allocate a new, smaller table and then copy the items from the old table into the new one. Similar to table doubling, we halving its size when its load factor drops below 1/4 in TABLE-DELETE, but in this problem,we contract a table by multiplying its size by 2/3 when its load factor drops below 1/3. Using the potential function

$$\Phi(\mathrm{T}) =\mid 2 * T.num - T.\text{ size } \mid$$

show that the amortized cost of a TABLE-DELETE that uses this strategy is bounded above by a constant.

Problem 2:

Assume you are creating an array data structure that has a fixed size of n. You want to backup this array after every so many insertion operations. Unfortunately, the backup operation is quite expensive, it takes n time to do the backup. Insertions without a backup just take 1 time unit. How frequently can you do a backup and still guarantee that the amortized cost of insertion is O(1) ? Prove that you can do backups in O(1) amortized time. Use the potential method for your proof.

Problem 3:

There is a way to implement a queue Q with two stacks S_1 and S_2. To insert an element e , we simply push it into S_1. To delete an element , we pop each element in S_1 and then push it into S_2 when S_2 is empty. If S_2 is not empty, we pop the element in S_2 directly. Assume that the cost of push and pop is 1. Here is an example of insert a, insert b, delete.

	$S_1 =$ []	$S_2 =$ []	
INSERT(a)	$S_1 =$ [a]	$S_2 =$ []	
INSERT(b)	$S_1 =$ [b a]	$S_2 =$ []	
DELETE()	$S_1 =$ []	$S_2 =$ [a b]	"dump"
	$S_1 =$ []	$S_2 =$ [b]	"pop" (returns a)

Figure 1: Example of Q3

Start with no element in the queue, give the best upper bound you can on the amortized cost using accounting method when there are arbitrary insert and delete .

Problem 4:

Suppose we have a device that generates a sequence of independent fair random coin-flips, but what we want is a six-sided dice that generates the values 1,2,3,4,5,6 with equal probability. Give an algorithm that does so using $\frac{11}{3}$ coin-flips on average and explain the expected coin-flips your algorithm need to use.

Problem 5:

Consider a 3-coloring problem in which you need to maximize the number of satisfied edges. In a graph $G = (V, E)$, edge (u, v) is *satisfied* if the node u and v are assigned with different colors. Design a randomized algorithm that the expected number of edges it satisfies is at least $\frac{2}{3}$ of the optimal number and prove it.

Problem 6:

a) Design a scheme to generate unbiased coin flips using a biased coin (the coin has a probability p to generate a HEAD and you do not know the p). You are required to generate an unbiased result with the expected number of flipping the coin up to $\frac{1}{p(1-p)}$.

b) Modify your scheme to make it more efficient, i.e., try to get more bits every time you toss the coin (in expectation).