York University
EECS 4101/5101, Winter 2023
Assignment 1

Due Date: January 31st, at 23:59

In the midst of winter, I found there was, within me, an invincible summer. And that makes me happy. For it says that no matter how hard the world pushes against me, within me, there is something stronger - something better, pushing right back ...

Albert Camus

All problems are written problems; submit your solutions electronically **only via Crowdmark**. You are welcome to discuss the general idea of the problems with other students. However, you must write your answers individually and mention your peers (with whom you discussed the problems) in your solution. Please refer to the course webpage for guidelines on academic integrity.

Problem 1 Alternative Dynamic Arrays [6 + 6 + 6 = 18 marks]

Consider a variant of dynamic arrays in which when an array becomes full, instead of doubling the size of the array, we triple the size of the array. This way, array sizes will be powers of 3. For example, on the 28th operation (insertion), the size of the array is changed from 27 to 81.

a) Use the aggregate-cost method to find an upper bound of 2.5 for the amortized cost of each operation. **Answer:** The total cost will be m (for inserting new items) plus $\sum_{j=0}^{\lfloor \log_3(m-1) \rfloor} 3^j$ (for copying old items to new arrays). It will be at most $m + \frac{3^{\log_3(m-1)+1}-1}{2} = m + 3(m-1)/2 - 1/2 = 5m/2 - 2$. So, the amortized cost per operation is roughly $5/2 \in \Theta(1)$. You will get the full mark as long as you show the amortized cost is a constant.

b) Use the accounting method to prove that the amortized cost per operation is at most 2.5. **Answer:** There is a deposit of 2.5\$ for each operation. An inexpensive operation spends 1\$ (for inserting the item) and saves 1.5\$ from the deposit. The number of consecutive inexpensive operations before an expensive one is $2m/3 - 1$, i.e., there is a saving of $3/2 \times (2m/3 - 1) = m - 3/2$ before the expensive operation. On the other hand, an expensive operation on the mth request deposits 2.5\$ and spends m\$, i.e., it spends $m - 2.5$\$. So, the balance after the expensive operation is $(m - 3/2) - (m - 5/2) = 1$, i.e., the balance remains positive and hence the claimed amortized cost of at most 2.5 holds.

c) Use the potential-function method to prove an upper bound of 2.5 for the amortized cost of each operation. **Answer:** Assume $\Phi(i) = 1.5i - 0.5a_i$ where a_i is the size of the array. For an inexpensive operation, the size of array does not change. The difference in potential will be $1.5i - 1.5(i-1) = 1.5$. The amortized cost will be 1+1.5 = 2.5 (1 for inserting the item and 1.5 for difference in potential). For an expensive operation, the size of the array changes from $(i-1)$ to $3(i-1)$. The difference in potential will be

$$(1.5i - 0.5 \cdot 3(i-1)) - (1.5(i-1) - 0.5(i-1)) = 1.5i - 1.5i + 1.5 - 1.5i + 1.5 + 0.5i - 0.5 = -i + 2.5$$

The amortized cost will be i (for inserting/moving items) plus $-i + 2.5$ (difference in potential) which sums up to 2.5.

Problem 2 Special Stack Variant [8 marks]

In the class, we saw a special type of stack in which each operation involves popping $n \geq 0$ items followed by pushing exactly one item. Consider a variant in which each operation involves popping one item followed by pushing $n \geq 0$ items. Assume the stack is implemented using an array of fixed size C, and the number of items in the stack is never more than C. Use the potential function method to show the amortized cost of each operation is at most 2.

Answer: Define the potential to be the number of empty cells in the array. For an operation involving n pushes, the actual cost is $n + 1$ and for each push, the potential is decreased by 1. So, the difference in potential will be $-n$ for pushed items and $+1$ for the pop, i.e., a total difference of $-n + 1$. The amortized cost will be $(n + 1) + (-n + 1) = 2$.

Problem 3 Ski-rental & Randomization [10 marks]

Consider the following algorithm for the ski-rental problem: flip two fair coins at the beginning, if both are tails, buy the equipment at the beginning; if any of the coins is a head, always rent and never buy. Assume the cost of buying is b and the cost of renting is 1 per day; let x denote the number of days that the the player goes skiing.

a) What is the expected cost of the algorithm in terms of b and x? (the expectation is taken over the random choice made by the algorithm).

b) Specify whether the competitive ratio of the algorithm in independent of x or not. Here, the competitive ratio is the maximum ratio between the expected cost of the algorithm and the optimal cost (where the maximum is taken over all inputs). You need to either prove a constant bound for the competitive ratio (which is independent of x) OR show via an adversarial input that the competitive ratio is not constant.

Answer: a) The cost of the algorithm is b when both coins are tails and x otherwise. On expectation, it is $b/4 + 3x/4 = (b + 3x)/4$.

b) Despite using randomization, the competitive ratio is not independent of x. Consider the following adversarial input: the adversary sets $x = 2^b$, where b is also a large number. The expected cost of the algorithm is $(b + 3x)/4 = (b + 3 \cdot 2^b)/4 > 2^{b-1}$. The cost of OPT is b (it knows x is pretty large and buys at the beginning). The competitive ratio will be at least $\frac{2^{b-1}}{b}$ which is unbounded for large values of b.

Problem 4 The Path-cow Problem [10+8 marks]

I) Consider the following algorithm for the path-cow problem. The cow starts at the origin and moves $x = 1$ unit to the right. If the target is not found, the cow returns to the origin and goes $x = 1$ unit to the left. If the target is not found, the cow returns to the origin and repeats this procedure with $x = 2, 4, \ldots, 2^i, \ldots$ until the target is found.

a) Assume the cow finds the hole at distance u from the origin on its left. Assume the largest power of 2 which is smaller than u is 2^k. What is the total distance moved by the cow?

b) Where does the adversary place the hole in order to harm the algorithm?

c) What is the competitive ratio of this algorithm?

Answer: a) The total moved distance would be:

$$4(1 + 2 + \ldots + 2^k) + 2 \cdot 2^{k+1} + u = 6 \cdot 2^{k+1} + u - 4$$

.

b) Adversary places the hole at distance $u = 2^k + \epsilon$ on the left. The cost of OPT will be $u = 2^k + \epsilon$.

c) The competitive ratio would be: $\frac{6 \cdot 2^{k+1} + u - 4}{u} = 1 + \frac{6 \cdot 2^{k+1} - 4}{2^k + \epsilon} \approx 1 + 6\frac{2^{k+1}}{2^k} = 13$ (to get this ratio, the adversary chooses small ϵ and large u).

II) Assume instead of a path, we have a binary tree with each edge having a length of 1. Initially, the cow is located at the root. First, she moves to the left child; if the target is not found, she returns to the root and then moves to the right child. If the target is still not found, she returns to the root and visits nodes at depth 2 of the tree (i.e., at a distance 2 from the root on the left). After checking these nodes, the cow returns to the root and repeats the same for nodes of depth 3. This procedure is repeated until at some point the target is found. In a nutshell, the algorithm works in rounds, where at round i, she visits all vertices of depth i.

What is the competitive ratio of this algorithm? To answer, assume the target is at depth k and write the competitive ratio in terms of k. As before, you must indicate where the adversary places the target and deduce the competitive ratio accordingly.

Answer: At step i, nodes of depth i are visited. The induced binary subtree is formed by $1+2+\ldots+2^i = 2^{i+1}-1$ vertices and hence $2^{i+1}-2$ edges. Each of these edges are visited 2 times in a round. So, the cost of the algorithm at round i is $2^{i+2}-4$. In the worst case, the target is located at depth k and is the last vertex checked by the algorithm at round k. In this case, the distance moved in round k is $2^{k+2}-4-k$ (the last deduction is because the algorithm does not return to the root after finding the target). On the other hand, the cost of Opt is k. The cost of the algorithm will be

$$(2^{1+2}-4)+(2^{2+2}-4)+\ldots+(2^{k+1}-4)+(2^{k+2}-4-k) = (2^{k+3}-1)-4k-7$$

The competitive ratio will be $\frac{(2^{k+3}-1)-4k-7}{k}$.