CS3230 Semester 1 2024/2025

Design and Analysis of Algorithms

Tutorial 08
Amortized Analysis
For Week 09

Document is last modified on: July 25, 2024

1 Lecture Review: Amortized Analysis

Amortized analysis is a strategy for analyzing a sequence of operation**s** (notice plural; one operation repeated several times or several operations intermixed) to show that the average cost per operation is small, even though a single operation within the sequence might be expensive.

Amortized analysis is **not** about calculating expected runtime and thus no probability theory involved. Amortized analysis guarantees the **average performance of each operation in the worst case**.

There are three common amortization arguments, and we will use two of them in this tutorial:

1. **Aggregate** method
2. **Accounting** (or Banker's) method
3. **Potential** method

1.1 Aggregate Method

Aggregate method is the most straightforward method (recall binary counter increment analysis from the lecture). In aggregate method, we try to show that for all n, a sequence of n operations takes $T(n)$ in the worst-case. The average, or amortized cost, per operation is therefore $\frac{T(n)}{n}$ in the worst-case. This method might be hard to apply in general. We will generally not use this method going forward.

1.2 Accounting Method

We charge the i-th operation a fictitious (larger) **amortized cost** $C(i)$ than its true cost $T(i)$, assuming that \$1 pays for 1 unit of work (time).

Every time we perform an operation, we consume this \$1 fee, but any amount that is not immediately consumed is stored in the (virtual) **bank** to be used by subsequent (i.e., more expensive) operations.

The idea is to impose **an extra charge on inexpensive** (and usually many) operations and **use it to pay for expensive** (and usually rare) operations later on.

At any given time, the bank balance **must not go negative**, i.e., we want to ensure that $\forall Q, \sum_{i=1}^{Q} T(i) \leq \sum_{i=1}^{Q} C(i)$. This way, the total amortized costs provides an upper bound on the total true costs.

1.3 Potential Method

To use potential method, we need to determine a suitable potential function ϕ that satisfies two properties: $\phi(0) = 0$ (initially, the potential is zero) and $\phi(i) \geq 0, \forall i > 0$ (the potential never goes negative). A good heuristic for determining ϕ is to find "something which decreases a lot after performing the expensive operations!". If we can find this suitable ϕ, then the amortized cost is $C(i) = T(i) + (\phi(i) - \phi(i-1))$ (the actual cost plus the difference of potential).

2 Tutorial 08 Questions

Note: For Questions Q1+Q2 involving **Dynamic Table**, go to `https://visualgo.net/en/array?mode=array` (so we do not trigger the sorted array mode), click 'Create M, N', set $N = M$ (or $M = N$, either way is fine), click 'Random' (to create a full array with $N = M$ random integers), then click 'Insert(v)', then click 'Append' (to append any value at the back of this full array, hence triggering the Dynamic Table animation).

Q1). Which one of the following statements is **incorrect**?

1. The amortized cost $C(i)$ for the insertion in Dynamic Table is $\Theta(1)$
   Also see Q2).
2. In the accounting method, $C(i)$ is always greater than the actual cost $T(i)$ of an operation
3. $\forall Q, \sum_{i=1}^{Q} C(i) - \sum_{i=1}^{Q} T(i) \geq 0$

Q2). is hidden, but it is related to **Dynamic Table** that we have discussed in Lecture.

Note: For Questions Q3+Q4 involving **Queue** ADT with MultiEnqueue and MultiDequeue capabilities, go to `https://visualgo.net/en/list?mode=Queue` (so we trigger Queue mode).

Q3). You are given a data structure based on the **Queue** ADT with the following operations:

- ENQUEUE(x): Put element x to the back of the queue
  Just set $v = x$ (a single integer) in the 'Enqueue' menu

- DEQUEUE(): Remove one element from the front of the queue
  Just select '1x' in the 'Dequeue' menu
- DELETE(k): Remove the first k elements from the queue
  Set the value of $K = k$ before selecting 'Kx' in the 'Dequeue' menu
- ADD(A): Put all elements in array A to the queue
  Set $v = A$ (put A as comma separated integers) in the 'Enqueue' menu

Prove the following claims using **Accounting Method** (stating the amortized costs/the bank charges of each operation), assuming that the queue is initially empty: "All operations run in amortized $O(1)$ time, except ADD which runs in amortized $O(|A|)$ time".

Q4). is hidden, but it is related to the same **Queue** ADT above.

Q5). Consider another version of **Dynamic Table** with only deletion of the last element (POP()), i.e., there is no insertion involved. The pseudo-code:

```
POP():
    N = N-1;
    if (N == 0):
        free(T)
   else if (N <= size(T) / 2):
        T' = createTable(N)
        copy(T, T')
        free(T)
        T = T'
```

An example ASCII art:

```
0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F, SIZE(T) = 2*N = 2*8 = 16 before POP()
x|x|x|x|x|x|x|x|-|-|-|-|-|-|-|-, there are N+1 = 8+1 = 9 elements before POP()

0|1|2|3|4|5|6|7|, SIZE(T) = N = 8 after POP()
x|x|x|x|x|x|x|x|, there are N = 8 elements after POP()
```

Prove using **Potential Method** that POP() runs in amortized $O(1)$.
State your potential function.
Assume N is initially equal to $SIZE(T)$.