21120491 Advanced Data Structure and Algorithm Analysis **Spring-Summer 2023**

Lecture 1: Amortized Analysis

Lecturer: Deshi Ye *Scribe: D. Ye*

1.1 Overview

A binary search tree can perform three basic operations: searching, insertion, and deletion. Note that we have shown that searching a node in an n-node AVL tree costs $O(\log n)$. Here $O(\log n)$ is the running time of the Searching operation for BST in the worst-case. Now, suppose that we would like to repeat M times to search the same node x in an AVL tree. What is the running time for these M searches? The answer is $O(M \log n)$ because each search costs $O(\log n)$.

In the above example, we have two concepts. One is the worst-case running time for one operation, say, $O(\log n)$ for one search. The other is the total running time for many operations, say $O(M \log n)$ for M searches.

Amortized analysis is a technique for analyzing the running time of a sequence of operations. It is straightforward to find the total running time of a sequence of operations by providing the product of the number of operations and the worst-case running time for one operation. However, it is too pessimistic. Can we get a better upper bound on overall operations? Amortized analysis is to provide an upper bound on an arbitrary sequence of operations.

The worst-case analysis is concerned with one operation. The amortized analysis is concerned with the overall cost of a sequence of operations. An analysis of the worst-case cost of a **sequence** of operations is called an amortized analysis.

The amortized analysis differs from the average-case analysis that relies on probabilistic assumptions about the data structures and operations in order to compute an expected running time of an algorithm.

In sum, we list different analysis of the running time as below.

- Worst-case analysis: the analysis is valid for all input (even for the "worst" input).
- Average-case analysis: the analysis is valid for input with a given distribution (we get the expected performance).
- Amortized analysis: for problems in which one must perform a series of operations, and the goal is to analyze the time per operation for any sequence of operations and for all input of the problem.

The key idea of amortized analysis is to find relations between expensive operations and cheap operations, for example, it would be nice if one expensive operation will be accompanied with a larger number of cheap operations.

Actual cost vs Amortized cost per operation Suppose we perform a series of operations $O_1, O_2, \ldots, O_M$, and the amount of time taken to execute operation O_i is denoted by $c(O_i)$.

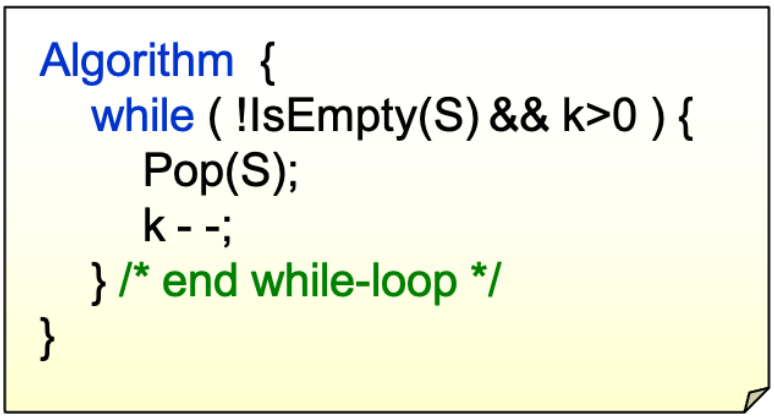

Figure 1.1: Algorithms for Multi-Pop

For each operation O_i, the goal of amortized analysis is to find a value $\hat{c}(O_i)$, called the amortized cost of O_i, such that

$$\sum_{i=1}^{M} c(O_i) \leq \sum_{i=1}^{M} \hat{c}(O_i).$$

Here, for each operation O_i, $c(O_i)$ is called the **actual cost**, and $\hat{c}(O_i)$ is called the **amortized cost**.

Instead of bounding the cost of the sequence of operations by bounding the actual cost of **each operation** separately, an amortized analysis provides **an upper bound** on the actual cost of the **entire sequence**.

Definition 1.1 *The* **amortized cost per operation** *is the total cost of a sequence of M operations divided by M.*

There are several methods to achieve amortized analysis: the aggregate analysis, accounting method, and potential method [1].

1.2 Aggregate analysis

The aggregate analysis is to find an upper bound on n operations, and the amortized cost per operation will be the total cost divided by n. The aggregate analysis is directly to find the upper bound of $\sum_{i=1}^{M} c(O_i)$, and then let $\hat{c}(O_i) = \sum_{i=1}^{M} c(O_i)/M$ for all i.

Example: The Multi-Pop algorithm is shown in Fig. 1.1. Consider a sequence of n Push, Pop, and MultiPop operations on an initially empty stack, what is the total cost of these n operations?

In this example, the actual cost $c(O_i)$ can be 1 or $T = \min(sizeof(S), k)$, where O_i can be PUSH, POP, or MULTIPOP.

In the following, we want to find an upper on the total actual costs. In a stack of size n, we have at most n PUSH operations. To pop an object from the stack only if we have pushed it onto the stack. Therefore, the number of times that POP can be called on a nonempty stack, including calls within MULTIPOP,

is at most the number of PUSH operations, which is at most n. For any value of n, any sequence of n PUSH, POP, and MULTIPOP operations takes a total of $O(n)$ time. The amortized cost of any operation is $\hat{c}(O_i) = O(n)/n = O(1)$ for any i.

1.3 Accounting method

As the definition of actual cost $c(O_i)$ and the amortized cost $\hat{c}(O_i)$, we define the credit of O_i to be $\hat{c}(O_i) - c(O_i)$. The Accounting method is also called the banker's method, operations can place credits on the data structure or spend credits that have already been placed. In details, when we do the operation O_i, we spend costs $\hat{c}(O_i)$, so the total cost we have spent is $\sum_{i=1}^{M} \hat{c}(O_i)$. However, regarding the operation O_i, the actual cost is $c(O_i)$, that means we save $\hat{c}(O_i) - c(O_i)$ costs if $\hat{c}(O_i) - c(O_i) \geq 0$. In case that $\hat{c}(O_i) - c(O_i) < 0$, we need to spend the costs (i.e. credits) that have been saved in the bank. The assignment of the amortized $\hat{c}(O_i)$ is successful if we always have enough credits in the bank when we need them.

In the example of Multi-Pop stacks, we assign the amortized cost $\hat{c}(O_i) = 2$ if O_i is a PUSH operation, and $\hat{c}(O_i) = 0$ otherwise. When it is a PUSH operation, we will save 1 credit in the bank. When we POP one object, we need to draw 1 credit from the bank, and we just use the exact credit saved by this object. That means there are exactly k credits in the bank if the stack has k objects. Thus, the saved credits are sufficient for future POP and MULTIPOP operations.

The key part of accounting method is to guarantee that our account will never go negative, *i.e.*, it always satisfies $\sum_{i=1}^{M} c(O_i) \leq \sum_{i=1}^{M} \hat{c}(O_i)$ for any M.

1.4 Potential method

The Potential method is similar to the banker's method. Let D_i be the data structure at time i (after applying the operation O_i), and the potential function $\Phi(Di)$ maps D_i onto a real value. For simplification, we let $\hat{c}_i = \hat{c}(O_i)$ and $c_i = c(O_i)$ for an operation O_i, respectively. In the potential method, the amortized cost of operation O_i is equal to the actual cost plus the increase in potential due to the operation:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \tag{1.1}$$

By Equation 1.1, the total amortized cost is

$$\sum_{i=1}^{M} \hat{c}_i = \sum_{i=1}^{M} c_i + \sum_{i=1}^{M} (\Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{M} c_i + \Phi(D_M) - \Phi(D_0).$$

In general, it is a good potential function if $\Phi(D_M) - \Phi(D_0) \geq 0$. To ensure that the total amortized cost is an upper bound on the total actual cost, for a sequence of any length, we merely must ensure that $\Phi(D_i) \geq \Phi(D_0)$ for all i.

In the example of STACK, let D_i be the stack that results after the i-th operation. The potential $\Phi(D_i)$ is the number of objects in the stack D_i. Note that $\Phi(D_0) = 0$ since it starts from empty stack, and $\Phi(D_i) \geq 0$.

In case of PUSH, the actual cost is $c_i = 1$. The amortized cost $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 2$ since stack D_i has one more object than the stack D_{i-1}.

For the operation POP, the actual cost $c_i = 1$, the potential decreased by 1 after POP one object, i.e. $\Phi(D_i) - \Phi(D_{i-1}) = -1$, hence, the amortized cost for POP is $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 0$.

For the operation MULTIPOP, the actual cost $c_i = T$, where $T = \min(sizeof(S), k)$. Note that the potential will decrease T, whereas $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = T - T = 0$.

One can easily get an upper bound on the total of the actual cost

$$O(M) = \sum_{i=1}^{M} \hat{c}_i \geq \sum_{i=1}^{M} c_i.$$

The amortized cost per operation is $\sum_{i=1}^{M} c_i/M = O(1)$ for a serial of PUSH, POP, MULTIPOP operations.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.