3-3 Amortized Analysis

Jun Ma

majun@nju.edu.cn

October 10, 2020

TC 17.1-3 I

Suppose we perform a sequence of n operations on a data structure in which the cost of the i-th operation is

$$c_i = \begin{cases} i & \text{if } i = 2^k \\ 1 & \text{otherwise} \end{cases}$$

Use **aggregate analysis** to determine the amortized cost per operation.

Aggregate analysis

- Let $C = \{i | i <= n, i = 2^k \text{ for some } k\}$, $|C| = \lfloor \log n \rfloor + 1$

$$\sum_{1 \le i \le n} c_i = \sum_{i \in C} c_i + \sum_{i \notin C} c_i$$

Aggregate analysis

- Let $C = \{i | i <= n, i = 2^k \text{ for some } k\}$, $|C| = \lfloor \log n \rfloor + 1$

$$
\begin{aligned}
\sum_{1 \le i \le n} c_i &= \sum_{i \in C} c_i + \sum_{i \notin C} c_i \\
&= (1 + 2 + \cdots + 2^{\lfloor \log n \rfloor}) + (n - \lfloor \log n \rfloor - 1)
\end{aligned}
$$

Aggregate analysis

- Let $C = \{i | i <= n, i = 2^k \text{ for some } k\}$, $|C| = \lfloor \log n \rfloor + 1$

$$\begin{aligned}\sum_{1 \le i \le n} c_i &= \sum_{i \in C} c_i + \sum_{i \notin C} c_i \\ &= (1 + 2 + \cdots + 2^{\lfloor \log n \rfloor}) + (n - \lfloor \log n \rfloor - 1) \\ &= 2^{\lfloor \log n \rfloor + 1} + (n - \lfloor \log n \rfloor - 1)\end{aligned}$$

Aggregate analysis

- Let $C = \{i | i <= n, i = 2^k \text{ for some } k\}$, $|C| = \lfloor \log n \rfloor + 1$

$$\begin{aligned}
\sum_{1 \le i \le n} c_i &= \sum_{i \in C} c_i + \sum_{i \notin C} c_i \\
&= (1 + 2 + \cdots + 2^{\lfloor \log n \rfloor}) + (n - \lfloor \log n \rfloor - 1) \\
&= 2^{\lfloor \log n \rfloor + 1} + (n - \lfloor \log n \rfloor - 1) \\
&= O(n)
\end{aligned}$$

Aggregate analysis

- Let $C = \{i | i <= n, i = 2^k \text{ for some } k\}$, $|C| = \lfloor \log n \rfloor + 1$

$$\begin{aligned} \sum_{1 \le i \le n} c_i &= \sum_{i \in C} c_i + \sum_{i \notin C} c_i \\ &= (1 + 2 + \cdots + 2^{\lfloor \log n \rfloor}) + (n - \lfloor \log n \rfloor - 1) \\ &= 2^{\lfloor \log n \rfloor + 1} + (n - \lfloor \log n \rfloor - 1) \\ &= O(n) \end{aligned}$$

- So $O(1)$ cost per each operation.

TC 17.2-2

Suppose we perform a sequence of n operations on a data structure in which the cost of the i-th operation is

$$c_i = \begin{cases} i & \text{if } i = 2^k \\ 1 & \text{otherwise} \end{cases}$$

Use **accounting method** to determine the amortized cost per operation.

Accounting method

$$c'_i = \begin{cases} 2 & \text{if } i = 2^k \\ 3 & \text{otherwise} \end{cases}$$

We still have to show $\forall n(\sum_{i \le n} c_i \le \sum_{i \le n} c'_i)$

- Let $C = \{i | i <= n, i = 2^k \text{ for some } k\}$, $|C| = \lfloor \log n \rfloor + 1$
  - $$\begin{aligned} \sum_{1 \le i \le n} c_i &= \sum_{i \in C} c_i + \sum_{i \notin C} c_i \\ &= (1 + 2 + \cdots + 2^{\lfloor \log n \rfloor}) + (n - \lfloor \log n \rfloor - 1) \\ &= 2^{\lfloor \log n \rfloor + 1} + (n - \lfloor \log n \rfloor - 1) \\ &\le 3n - \lfloor \log n \rfloor - 1 \end{aligned}$$
  - $$\sum_{1 \le i \le n} c'_i = \sum_{i \in C} c'_i + \sum_{i \notin C} c'_i$$

Accounting method

$$c_i' = \begin{cases} 2 & \text{if } i = 2^k \\ 3 & \text{otherwise} \end{cases}$$

We still have to show $\forall n (\sum_{i \le n} c_i \le \sum_{i \le n} c_i')$

- Let $C = \{i | i <= n, i = 2^k \text{ for some } k\}$, $|C| = \lfloor \log n \rfloor + 1$
  - $$\begin{aligned} \sum_{1 \le i \le n} c_i &= \sum_{i \in C} c_i + \sum_{i \notin C} c_i \\ &= (1 + 2 + \cdots + 2^{\lfloor \log n \rfloor}) + (n - \lfloor \log n \rfloor - 1) \\ &= 2^{\lfloor \log n \rfloor + 1} + (n - \lfloor \log n \rfloor - 1) \\ &\le 3n - \lfloor \log n \rfloor - 1 \end{aligned}$$
  - $$\begin{aligned} \sum_{1 \le i \le n} c_i' &= \sum_{i \in C} c_i' + \sum_{i \notin C} c_i' \\ &= 2(\lfloor \log n \rfloor + 1) + 3(n - \lfloor \log n \rfloor - 1) \end{aligned}$$

Accounting method

$$c'_i = \begin{cases} 2 & \text{if } i = 2^k \\ 3 & \text{otherwise} \end{cases}$$

We still have to show $\forall n (\sum_{i \le n} c_i \le \sum_{i \le n} c'_i)$

- Let $C = \{i | i <= n, i = 2^k \text{ for some } k\}$, $|C| = \lfloor \log n \rfloor + 1$
  - $$\begin{aligned} \sum_{1 \le i \le n} c_i &= \sum_{i \in C} c_i + \sum_{i \notin C} c_i \\ &= (1 + 2 + \cdots + 2^{\lfloor \log n \rfloor}) + (n - \lfloor \log n \rfloor - 1) \\ &= 2^{\lfloor \log n \rfloor + 1} + (n - \lfloor \log n \rfloor - 1) \\ &\le 3n - \lfloor \log n \rfloor - 1 \end{aligned}$$
  - $$\begin{aligned} \sum_{1 \le i \le n} c'_i &= \sum_{i \in C} c'_i + \sum_{i \notin C} c'_i \\ &= 2(\lfloor \log n \rfloor + 1) + 3(n - \lfloor \log n \rfloor - 1) \\ &= 3n - \lfloor \log n \rfloor - 1 \end{aligned}$$

TC 17.4-1

Suppose that we wish to implement a dynamic, open-address hash table.

- Why might we consider the table to be full when its load factor reaches some value α that is strictly less than 1?
- Describe briefly how to make insertion into a dynamic, open-address hash table run in such a way that the expected value of the amortized cost per insertion is $O(1)$.

TC 17.4-1

Suppose that we wish to implement a dynamic, open-address hash table.

- Why might we consider the table to be full when its load factor reaches some value α that is strictly less than 1?
- Describe briefly how to make insertion into a dynamic, open-address hash table run in such a way that the expected value of the amortized cost per insertion is $O(1)$.

Corollary 11.7
Inserting an element into an open-address hash table with load factor α requires at most $1/(1-\alpha)$ probes on average, assuming uniform hashing.

Answer

- **Expanding** when $\alpha \geq 0.75$.
- Contracting when $\alpha \leq 0.25$.

Answer

- **Expanding** when $\alpha \geq 0.75$.
- Contracting when $\alpha \leq 0.25$.
- Potential function:

$$\Phi_i = \begin{cases} \frac{8}{3}num_i - size_i & \text{if table is at least half full} \\ \frac{1}{2}size_i - num_i & \text{if table is less than half full} \end{cases}$$

Answer

- **Expanding** when $\alpha \geq 0.75$.
- Contracting when $\alpha \leq 0.25$.
- Potential function:

$$\Phi_i = \begin{cases} \frac{8}{3}num_i - size_i & \text{if table is at least half full} \\ \frac{1}{2}size_i - num_i & \text{if table is less than half full} \end{cases}$$

- If the i-th insertion does not lead to expanding

$$num_i = num_{i-1} + 1$$
$$size_i = size_{i-1}$$

Answer

- **Expanding** when $\alpha \geq 0.75$.
- Contracting when $\alpha \leq 0.25$.
- Potential function:

$$\Phi_i = \begin{cases} \frac{8}{3} num_i - size_i & \text{if table is at least half full} \\ \frac{1}{2} size_i - num_i & \text{if table is less than half full} \end{cases}$$

- If the i-th insertion does not lead to expanding

$$\begin{aligned} num_i &= num_{i-1} + 1 \\ size_i &= size_{i-1} \end{aligned}$$

$$\begin{aligned} E(c_i') &= E(c_i + \Phi_i - \Phi_{i-1}) \\ &= \begin{cases} E(c_i) + 8/3 \leq 4 + 8/3 & \text{if table is at least half full} \\ E(c_i) - 1 \leq 4 - 1 = 3 & \text{if table is less than half full} \end{cases} \end{aligned}$$

- If the i-th insertion leads to expanding

- If the i-th insertion leads to expanding

$$num_i = num_{i-1} + 1$$
$$size_i = 2size_{i-1}$$
$$num_{i-1} = 3/4size_{i-1}$$

- If the i-th insertion leads to expanding

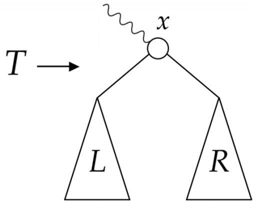
$$
\begin{aligned}
num_i &= num_{i-1} + 1 \\
size_i &= 2size_{i-1} \\
num_{i-1} &= 3/4size_{i-1}
\end{aligned}
$$

$$
\begin{aligned}
E(c_i') &= E(c_i + \Phi_i - \Phi_{i-1}) \\
&= E(c_i) + (1/2size_i - num_i) - (8/3num_{i-1} - size_{i-1}) \\
&= E(c_i) - num_i \\
&\le 4 + num_i - num_i = 4
\end{aligned}
$$

TC Problem 17.3 (Amortized weight-balanced trees)

Consider an ordinary binary search tree augmented by adding to each node x the attribute $x.size$ giving the number of keys stored in the subtree rooted at x. Let α be a constant in the range $1/2 \leq \alpha < 1$. We say that a given node x is α-balanced if $x.left.size \leq \alpha x.size$ and $x.right.size \leq x.size$. The tree as a whole is α-balanced if every node in the tree is α-balanced.

G. Varghese first introduced the amortized approach for maintaining weight-balanced trees (Cormen, Leiserson, Rivest, and Stein, 2009, p. 473).



Q(a)

Given a node x in an arbitrary binary search tree, show how to rebuild the subtree rooted at x so that it becomes 1/2-balanced.
Your algorithm should run in time $\Theta(x.size)$, and it can use $O(x.size)$ auxiliary storage.

Q(a)

Given a node x in an arbitrary binary search tree, show how to rebuild the subtree rooted at x so that it becomes 1/2-balanced.
Your algorithm should run in time $\Theta(x.size)$, and it can use $O(x.size)$ auxiliary storage.

- Performing an inorder traversal on the subtree and store elements increasingly in array A
- Recursively reconstruct a BST from $A[a, b]$, initially $a = 1, b = x.size$
  - Select the **median** element of A as the root.
  - Handle recursively $A[a...m-1]$ and $A[m+1, ..., b]$

Q(b)

Show that performing a search in an n-node α-balanced binary search tree takes $O(\lg n)$ worst-case time.

Q(b)

Show that performing a search in an n-node α-balanced binary search tree takes $O(\lg n)$ worst-case time.

- Show the height of an n-node α-balanced binary search tree is at most $c \lg n$ for some positive constant c
- $$\begin{aligned} h(x) &= 1 + \max\{h(x.left), h(x.right)\} \\ &\leq 1 + c \lg \alpha n \\ &= 1 + c \lg \alpha + c \lg n \end{aligned}$$
- When $1 + c \lg \alpha \leq 0$, we have $h(x) \leq c \lg n$
- So, we choose $c \geq \frac{-1}{\lg \alpha}$

For the remainder of this problem

- Assume the constant $\alpha > 1/2$.
- Suppose that we implement INSERT and DELETE as usual for an n-node binary search tree
- After every such operation, if any node in the tree is no longer α-balanced, then we “rebuild” the subtree rooted at the **highest** such node in the tree so that it becomes 1/2-balanced.
- For a node x in a binary search tree T , we define

$$\Delta(x) = |x.left.size - y.left.size|$$

$$\Phi(T) = c \sum_{x \in T : \Delta(x) \geq 2} \Delta x$$

where c is a sufficiently large constant that depends on α.

Q(c)

Argue that any binary search tree has nonnegative potential and that a 1/2- balanced tree has potential 0.

Q(c)

Argue that any binary search tree has nonnegative potential and that a 1/2- balanced tree has potential 0.

We show this by showing that $\Delta(x) \leq 1$ for every node x in T. Prove by contradiction.

- Assume $\exists x \in T$, s.t. $\Delta(x) = x.left.size - x.right.size \geq 2$
- $x.size = x.left.size + x.right.size + 1 \leq x.left.size + (x.left.size - 2) + 1$
- So,$x.left.size \geq (x.size + 1)/2$, contradiction!

Q(d)

Suppose that m units of potential can pay for rebuilding an m-node subtree. How large must c be in terms of α, in order for it to take $O(1)$ amortized time to rebuild a subtree that is not α-balanced?

Q(d)

Suppose that m units of potential can pay for rebuilding an m-node subtree. How large must c be in terms of α, in order for it to take $O(1)$ amortized time to rebuild a subtree that is not α-balanced?

- Let x be the **highest unbalanced node**, and T its subtree. Without loss of generality, $|L| > \alpha|T|$ and $|L| \geq |R| + 2$

Q(d)

Suppose that m units of potential can pay for rebuilding an m-node subtree. How large must c be in terms of α, in order for it to take $O(1)$ amortized time to rebuild a subtree that is not α-balanced?

- Let x be the **highest** **unbalanced node**, and T its subtree. Without loss of generality, $|L| > \alpha|T|$ and $|L| \geq |R| + 2$
- $|R| = |T| - |L| - 1 < (1 - \alpha)|T| - 1$

Q(d)

Suppose that m units of potential can pay for rebuilding an m-node subtree. How large must c be in terms of α, in order for it to take $O(1)$ amortized time to rebuild a subtree that is not α-balanced?

- Let x be the **highest** **unbalanced node**, and T its subtree. Without loss of generality, $|L| > \alpha|T|$ and $|L| \geq |R| + 2$
- $|R| = |T| - |L| - 1 < (1 - \alpha)|T| - 1$
- So, $|L| - |R| > (2\alpha - 1)|T| + 1$

Q(d)

Suppose that m units of potential can pay for rebuilding an m-node subtree. How large must c be in terms of α, in order for it to take $O(1)$ amortized time to rebuild a subtree that is not α-balanced?

- Let x be the highest **unbalanced node**, and T its subtree. Without loss of generality, $|L| > \alpha|T|$ and $|L| \geq |R| + 2$
- $|R| = |T| - |L| - 1 < (1 - \alpha)|T| - 1$
- So, $|L| - |R| > (2\alpha - 1)|T| + 1$
- $\Phi(T) = c \sum_{x \in T : \Delta(x) \geq 2} \Delta x \geq c(|L| - |R|)$
- $$\begin{aligned} \Phi(T_{rebuiled}) - \Phi(T) &= 0 - \Phi(T) \\ &\leq -c(|L| - |R|) \\ &< -c((2\alpha - 1)|T| + 1) \\ &< -c(2\alpha - 1)|T| \end{aligned}$$

Q(d)

Suppose that m units of potential can pay for rebuilding an m-node subtree. How large must c be in terms of α, in order for it to take $O(1)$ amortized time to rebuild a subtree that is not α-balanced?

- Let x be the **highest unbalanced node**, and T its subtree. Without loss of generality, $|L| > \alpha|T|$ and $|L| \geq |R| + 2$
- $|R| = |T| - |L| - 1 < (1-\alpha)|T| - 1$
- So, $|L| - |R| > (2\alpha - 1)|T| + 1$
- $\Phi(T) = c \sum_{x \in T : \Delta(x) \geq 2} \Delta x \geq c(|L| - |R|)$
- $$\begin{aligned} \Phi(T_{rebuiled}) - \Phi(T) &= 0 - \Phi(T) \\ &\leq -c(|L| - |R|) \\ &< -c((2\alpha - 1)|T| + 1) \\ &< -c(2\alpha - 1)|T| \end{aligned}$$
- $\hat{c_{rb}} = c_{rb} + \Phi(T_{rebuiled}) - \Phi(T) < |T| - c(2\alpha - 1)|T| \leq 0$

Q(d)

Suppose that m units of potential can pay for rebuilding an m-node subtree. How large must c be in terms of α, in order for it to take $O(1)$ amortized time to rebuild a subtree that is not α-balanced?

- Let x be the **highest unbalanced node**, and T its subtree. Without loss of generality, $|L| > \alpha|T|$ and $|L| \geq |R| + 2$
- $|R| = |T| - |L| - 1 < (1 - \alpha)|T| - 1$
- So, $|L| - |R| > (2\alpha - 1)|T| + 1$
- $\Phi(T) = c \sum_{x \in T : \Delta(x) \geq 2} \Delta x \geq c(|L| - |R|)$
-
$$\begin{aligned} \Phi(T_{rebuiled}) - \Phi(T) &= 0 - \Phi(T) \\ &\leq -c(|L| - |R|) \\ &< -c((2\alpha - 1)|T| + 1) \\ &< -c(2\alpha - 1)|T| \end{aligned}$$
- $\hat{c_{rb}} = c_{rb} + \Phi(T_{rebuiled}) - \Phi(T) < |T| - c(2\alpha - 1)|T| \leq 0$
- So, $c \geq 1/(2\alpha - 1)$


Thank
You!