Decision Trees, Adversary Argument and Amortized Analysis

Hengfeng Wei

hfwei@nju.edu.cn

April 11 $\sim$ April 12, 2017

Decision Trees, Adversary Argument and Amortized Analysis

Decision Trees, Adversary Argument and Amortized Analysis

Amortized analysis

Amortized analysis is
an algorithm analysis technique for
analyzing a sequence of operations
irrespective of the input to show that
the average cost per operation is small, even though
a single operation within the sequence might be expensive.

Methods for amortized analysis: the summation method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

Methods for amortized analysis: the summation method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

$$(\sum_{i=1}^{n} c_i)/n$$

Summation method: array doubling revisited

On any sequence of n INSERT ops on an initially empty array.

Summation method: array doubling revisited

On any sequence of n Insert ops on an initially empty array.

$$\begin{array}{rcccccccccc} o_i: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ c_i: & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 8 & 1 \end{array}$$

Summation method: array doubling revisited

On any sequence of n INSERT ops on an initially empty array.

$$
\begin{array}{ccccccccccc}
o_i: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
c_i: & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 8 & 1
\end{array}
$$

$$
c_i = \begin{cases} (i-1)+1 = i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{o.w.} \end{cases}
$$

Summation method: array doubling revisited

On any sequence of n INSERT ops on an initially empty array.

$$\begin{array}{lcccccccccc} o_i: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ c_i: & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 8 & 1 \end{array}$$

$$c_i = \begin{cases} (i-1)+1 = i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{o.w.} \end{cases}$$

$$\sum_{i=1}^{n} c_i \le n + \sum_{j=0}^{\lceil \lg n \rceil - 1} 2^j = n + (2^{\lceil \lg n \rceil} - 1) \le n + 2n = 3n$$

Summation method: array doubling revisited

On any sequence of n INSERT ops on an initially empty array.

$$\begin{array}{lcccccccccc} o_i: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ c_i: & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 8 & 1 \end{array}$$

$$c_i = \begin{cases} (i-1)+1 = i & \text{if } i-1 \text{ is an exact power of } 2 \\ 1 & \text{o.w.} \end{cases}$$

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lceil \lg n \rceil - 1} 2^j = n + (2^{\lceil \lg n \rceil} - 1) \leq n + 2n = 3n$$

$$\forall i, \hat{c}_i = 3$$

Methods for amortized analysis: the accounting method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

$$a_1, a_2, \ldots, a_n$$

Methods for amortized analysis: the accounting method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

$$a_1, a_2, \ldots, a_n$$

$$\hat{c_i} = c_i + a_i, a_i >=< 0.$$

Methods for amortized analysis: the accounting method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

$$a_1, a_2, \ldots, a_n$$

$$\hat{c_i} = c_i + a_i, a_i >=< 0.$$

$$\forall n, \sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c_i}$$

Methods for amortized analysis: the accounting method

$$o_1, o_2, \dots, o_n$$

$$c_1, c_2, \dots, c_n$$

$$a_1, a_2, \dots, a_n$$

$$\hat{c}_i = c_i + a_i, a_i >=< 0.$$

$$\forall n, \sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \implies \forall n, \sum_{i=1}^{n} a_i \ge 0$$

Methods for amortized analysis: the accounting method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

$$a_1, a_2, \ldots, a_n$$

$$\hat{c}_i = c_i + a_i, a_i >=< 0.$$

$$\forall n, \sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \implies \forall n, \sum_{i=1}^{n} a_i \ge 0$$

Key way of thinking:

Put the accounting cost on specific objects.

Accounting method: array doubling revisited

$$\hat{c}_i = 3 \textit{ vs. } \hat{c}_i = 2$$

Accounting method: array doubling revisited

$$\hat{c}_i = 3 \textit{ vs. } \hat{c}_i = 2$$

$$\hat{c}_i = 3 = \underbrace{1}_{\text{insert}} + \underbrace{1}_{\text{move itself}} + \underbrace{1}_{\text{help move another}}$$

Accounting method: array doubling revisited

$$\hat{c}_i = 3 \textit{ vs. } \hat{c}_i = 2$$

$$\hat{c}_i = 3 = \underbrace{1}_{\text{insert}} + \underbrace{1}_{\text{move itself}} + \underbrace{1}_{\text{help move another}}$$

	$\hat{c}_i$	c_i (actual cost)	a_i (accounting cost)
INSERT (normal)	3	1	2
INSERT (expansion)	3	$1 + t$	$-t + 2$

Array merging (Problem 4.13): the summation method

$\textsc{Create}\ (1);\ \textsc{Merge}\ (2m)$

Array merging (Problem 4.13): the summation method

$$\textsc{Create}\ (1);\ \textsc{Merge}\ (2m)$$

i	c_i
1	1
2	$1+2$
3	1
4	$1+2+4$
5	1
6	$1+2$
7	1
8	$1+2+4$
$\vdots$	$\ldots$

Array merging (Problem 4.13): the summation method

$$\textsc{Create}\ (1);\ \textsc{Merge}\ (2m)$$

i	c_i
1	1
2	$1+2$
3	1
4	$1+2+4$
5	1
6	$1+2$
7	1
8	$1+2+4$
$\vdots$	$\ldots$

$$\sum_{i}^{n} c_i = \sum_{i=1}^{\lfloor \log n \rfloor} \lfloor \frac{n}{2^i} \rfloor 2^i = n \log n$$

Array merging (Problem 4.13): the summation method

$$\textsc{Create}\ (1);\ \textsc{Merge}\ (2m)$$

i	c_i
1	1
2	$1+2$
3	1
4	$1+2+4$
5	1
6	$1+2$
7	1
8	$1+2+4$
$\vdots$	$\dots$

$$\sum_{i}^{n} c_i = \sum_{i=1}^{\lfloor \log n \rfloor} \lfloor \frac{n}{2^i} \rfloor 2^i = n \log n$$

$$\forall i, \hat{c}_i = \log n$$

Array merging (Problem 4.13): the accounting method

Array merging (Problem 4.13)

Decision Trees, Adversary Argument and Amortized Analysis