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

Algorithm analysis

- Given a problem P
- Design algorithms $A, A', \ldots$
- Input space $\mathcal{X}_n$: inputs of size n

Algorithm analysis

- Given a problem P
- Design algorithms $A, A', \ldots$
- Input space $\mathcal{X}_n$: inputs of size n

$$T_A(n) \triangleq \max_{X \in \mathcal{X}_n} T_A(X)$$

Algorithm analysis

- Given a problem P
- Design algorithms $A, A', \dots$
- Input space $\mathcal{X}_n$: inputs of size n

$$T_A(n) \triangleq \max_{X \in \mathcal{X}_n} T_A(X)$$

$$T_P(n) \triangleq \min_{A \text{ solves } P} T_A(n)$$

Algorithm analysis

- Given a problem P
- Design algorithms $A, A', \dots$
- Input space $\mathcal{X}_n$: inputs of size n

$$T_A(n) \triangleq \max_{X \in \mathcal{X}_n} T_A(X)$$

$$T_P(n) \triangleq \min_{A \text{ solves } P} T_A(n) = \min_{A \text{ solves } P} \max_{X \in \mathcal{X}_n} T_A(X)$$

Algorithm analysis

Selection (median):

Sorting:

Algorithm analysis

Selection (median):

Sorting:

$n!$

Algorithm analysis

Selection (median):

Sorting:

$$n!$$

$$\Longrightarrow n^2$$

Algorithm analysis

Selection (median):

Sorting:

$$n! \Longrightarrow n^2 \Longrightarrow n \log n$$

Algorithm analysis

Selection (median):

Sorting:

$$n!$$
$$\Longrightarrow n^2$$
$$\Longrightarrow n \log n$$

$$n$$

Algorithm analysis

Sorting:

$$
\begin{aligned}
& n! \\
& \quad \Longrightarrow n^2 \\
& \quad \Longrightarrow n \log n \\
n \log n \Longleftarrow & \\
& n
\end{aligned}
$$

Selection (median):

Algorithm analysis

Selection (median):

n^2

Sorting:

$n!$

$\implies n^2$

$\implies n \log n$

$n \log n \impliedby$

n

Algorithm analysis

Sorting:

$$
\begin{aligned}
&n! \\
&\quad\Longrightarrow n^2 \\
&\quad\Longrightarrow n \log n \\
n \log n \Longleftarrow & \\
&n
\end{aligned}
$$

Selection (median):

$$
\begin{aligned}
&n^2 \\
&\quad\Longrightarrow n \log n
\end{aligned}
$$

Algorithm analysis

Sorting:

$$n!$$
$$\Longrightarrow \quad n^2$$
$$\Longrightarrow \quad n \log n$$
$$n \log n \Longleftarrow$$
$$n$$

Selection (median):

$$n^2$$
$$\Longrightarrow \quad n \log n$$
$$\Longrightarrow \quad 16n$$

Algorithm analysis

Sorting:

$$n!$$

$$\Longrightarrow n^2$$

$$\Longrightarrow n \log n$$

$$n \log n \Longleftarrow$$

$$n$$

Selection (median):

$$n^2$$

$$\Longrightarrow n \log n$$

$$\Longrightarrow 16n$$

$$\Longrightarrow 2.95n$$

Algorithm analysis

Sorting:

$$n!$$

$$\Longrightarrow \quad n^2$$

$$\Longrightarrow \quad n \log n$$

$$n \log n \Longleftarrow$$

$$n$$

Selection (median):

$$n^2$$

$$\Longrightarrow \quad n \log n$$

$$\Longrightarrow \quad 16n$$

$$\Longrightarrow \quad 2.95n$$

$$n$$

Algorithm analysis

Sorting:

$$n! \Longrightarrow n^2 \Longrightarrow n \log n$$

$$n \log n \Longleftarrow n$$

Selection (median):

$$n^2 \Longrightarrow n \log n \Longrightarrow 16n \Longrightarrow 2.95n$$

$$\frac{3n}{2} - \frac{3}{2} \log n \Longleftarrow n$$

Algorithm analysis

Sorting:

$$n!$$

$$\Longrightarrow \; n^2$$

$$\Longrightarrow \; n \log n$$

$$n \log n \Longleftarrow$$

$$n$$

Selection (median):

$$n^2$$

$$\Longrightarrow \; n \log n$$

$$\Longrightarrow \; 16n$$

$$\Longrightarrow \; 2.95n$$

$$2n \Longleftarrow$$

$$\frac{3n}{2} - \frac{3}{2} \log n \Longleftarrow$$

$$n$$

Decision Trees, Adversary Argument and Amortized Analysis

Horse Racing

Example (Horse Racing)

- 25 horses
- Round: ≤ 5 horses race
- Goal: Find $\#1, \#2, \#3$ fastest.

Horse Racing

Example (Horse Racing)

- 25 horses
- Round: ≤ 5 horses race
- Goal: Find $\#1, \#2, \#3$ fastest.

$$8$$

Horse Racing

Example (Horse Racing)

- 25 horses
- Round: ≤ 5 horses race
- Goal: Find $\#1, \#2, \#3$ fastest.

$$8 \implies 7$$

Horse Racing

Example (Horse Racing)

- 25 horses
- Round: ≤ 5 horses race
- Goal: Find $\#1, \#2, \#3$ fastest.

$$8 \implies 7 \quad (a_2, a_3, b_1, b_2, c_1)$$

Horse Racing

Example (Horse Racing)

- 25 horses
- Round: ≤ 5 horses race
- Goal: Find $\#1, \#2, \#3$ fastest.

$$8 \implies 7 \quad (a_2, a_3, b_1, b_2, c_1)$$

(< 5)

Horse Racing

Example (Horse Racing)

- 25 horses
- Round: ≤ 5 horses race
- Goal: Find $\#1, \#2, \#3$ fastest.

$$8 \implies 7 \quad (a_2, a_3, b_1, b_2, c_1)$$

$$(< 5) \implies (= 5)$$

Horse Racing

Example (Horse Racing)

- 25 horses
- Round: ≤ 5 horses race
- Goal: Find $\#1, \#2, \#3$ fastest.

$$8 \implies 7 \quad (a_2, a_3, b_1, b_2, c_1)$$

$$(< 5) \implies (= 5) \quad (\#1)$$

Horse Racing

Example (Horse Racing)

- 25 horses
- Round: ≤ 5 horses race
- Goal: Find $\#1, \#2, \#3$ fastest.

$$8 \implies 7 \quad (a_2, a_3, b_1, b_2, c_1)$$

$$(< 5) \implies (= 5) \quad (\#1) \implies (= 6)$$

Horse Racing

Example (Horse Racing)

- 25 horses
- Round: ≤ 5 horses race
- Goal: Find $\#1, \#2, \#3$ fastest.

$$8 \implies 7 \quad (a_2, a_3, b_1, b_2, c_1)$$

$$(< 5) \implies (= 5) \quad (\#1) \implies (= 6) \quad (\#2)$$

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 $\textsc{Insert}$ ops on an initially empty array.

Summation method: array doubling revisited

On any sequence of n $\textsc{Insert}$ ops on an initially empty array.

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

Summation method: array doubling revisited

On any sequence of n $\textsc{Insert}$ ops on an initially empty array.

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

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

Summation method: array doubling revisited

On any sequence of n $\textsc{Insert}$ 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.

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

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

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

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

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

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

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

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

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

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

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

$$\text{CREATE } (1); \text{ MERGE } (2m)$$

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

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	$1+2+4$
$\vdots$	$\dots$

$$\sum_{i=1}^{n} c_i = \sum_{j=0}^{\lfloor \log n \rfloor} \lfloor \frac{n}{2^j} \rfloor 2^j \le n(\lfloor \log n \rfloor + 1)$$

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	$1+2+4$
$\vdots$	$\dots$

$$\sum_{i=1}^{n} c_i = \sum_{j=0}^{\lfloor \log n \rfloor} \lfloor \frac{n}{2^j} \rfloor 2^j \le n(\lfloor \log n \rfloor + 1)$$

$$\forall i, \hat{c_i} = 1 + \lfloor \log n \rfloor$$

Array merging (Problem 4.13): the accounting method

$$\hat{c}_i = 1 + \lfloor \log n \rfloor$$

Array merging (Problem 4.13): the accounting method

$$\hat{c}_i = 1 + \lfloor \log n \rfloor$$

What does it mean?

Array merging (Problem 4.13): the accounting method

$$\hat{c}_i = 1 + \lfloor \log n \rfloor$$

What does it mean?

$$\forall n, \sum_{i=1}^{n} a_i \geq 0$$

Two stacks, one queue (Problem 4.14)

Algorithm 1 Simulating a queue using two stacks S_1, S_2.

procedure $\textsc{Enq}(x)$
  Push(S_1, x)

procedure $\textsc{Deq}()$
  if $S_2 = \emptyset$ **then**
    while $S_1 \neq \emptyset$ **do**
      Push$(S_2,$ *Pop*$(S_1))$
  Pop(S_2)

Two stacks, one queue (Problem 4.14)

Algorithm 2 Simulating a queue using two stacks S_1, S_2.

```
procedure ENQ(x)
    Push(S_1, x)

procedure DEQ()
    if S_2 = ∅ then
        while S_1 ≠ ∅ do
            Push(S_2, Pop(S_1))
    Pop(S_2)
```

Costs of “Push” and “Pop”.

Two stacks, one queue: the summation method

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

Two stacks, one queue: the summation method

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

The operation sequence is NOT known.

Two stacks, one queue: the accounting method

item:	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
	1	1	1	1

Two stacks, one queue: the accounting method

item:	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 3$$
$$\hat{c}_{\text{DEQ}} = 1$$

Two stacks, one queue: the accounting method

item:	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 3$$
$$\hat{c}_{\text{DEQ}} = 1$$

$$\sum_{i=1}^{n} a_i \geq 0$$

Two stacks, one queue: the accounting method

item:	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 3$$
$$\hat{c}_{\text{DEQ}} = 1$$

$$\sum_{i=1}^{n} a_i \geq 0 \Longleftarrow \sum_{i=1}^{n} a_i = \#S_1 \times 2$$