2-4 Recurrences

魏恒峰

hfwei@nju.edu.cn

2018 年 04 月 09 日

Maximum-sum subarray

Maximum-sum subarray (Google Interview)

- Array $A[1 \cdots n], a_i >=< 0$
- To find (the sum of) a maximum-sum subarray of A

$$A[-2, 1, -3, 4, -1, 2, 1, -5, 4] \implies [4, -1, 2, 1]$$

Maximum-sum subarray

Maximum-sum subarray (Google Interview)

- Array $A[1 \cdots n], a_i >=< 0$
- To find (the sum of) a maximum-sum subarray of A

$$A[-2, 1, -3, 4, -1, 2, 1, -5, 4] \implies [4, -1, 2, 1]$$

Subproblem: MSS$[i]$: sum of an MS$[i]$ of $A[1 \cdots i]$

Goal: mss $=$ MSS$[n]$

Maximum-sum subarray

Maximum-sum subarray (Google Interview)

- Array $A[1 \cdots n], a_i >=< 0$
- To find (the sum of) a maximum-sum subarray of A

$$A[-2, 1, -3, 4, -1, 2, 1, -5, 4] \implies [4, -1, 2, 1]$$

Subproblem: $\textsf{MSS}[i]$: sum of an $\textsf{MS}[i]$ of $A[1 \cdots i]$

Goal: $\textsf{mss} = \textsf{MSS}[n]$

Make choice: Is $a_i \in \textsf{MS}[i]$?

Recurrence:

$$\textsf{MSS}[i] = \max\{\textsf{MSS}[i-1], ???\}$$

Maximum-sum subarray

Subproblem: $\mathsf{MSS}[i]$: sum of an $\mathsf{MS}[i]$ *ending with* a_i

Goal: $\mathsf{mss} = \max_{1 \le i \le n} \mathsf{MSS}[i]$

Maximum-sum subarray

Subproblem: MSS$[i]$: sum of an MS$[i]$ *ending with* a_i

Goal: mss $= \max_{1 \le i \le n}$ MSS$[i]$

Make choice: Where does the MS$[i]$ start?

Recurrence:

$$\text{MSS}[i] = \max\{\text{MSS}[i-1] + a_i, a_i\} \text{ (Proof!)}$$

Maximum-sum subarray

Subproblem: MSS$[i]$: sum of an MS$[i]$ *ending with* a_i

Goal: mss $= \max\limits_{1 \le i \le n}$ MSS$[i]$

Make choice: Where does the MS$[i]$ start?

Recurrence:

$$\text{MSS}[i] = \max\{\text{MSS}[i-1] + a_i, a_i\} \text{ (Proof!)}$$

Init:

$$\text{MSS}[0] = 0$$

Maximum-sum subarray

Subproblem: MSS$[i]$: sum of an MS$[i]$ *ending with* a_i

Goal: mss $= \max_{1 \le i \le n}$ MSS$[i]$

Make choice: Where does the MS$[i]$ start?

Recurrence:

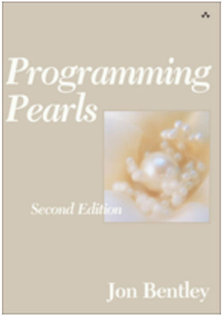
$$\text{MSS}[i] = \max\{\text{MSS}[i-1] + a_i, a_i\} \text{ (Proof!)}$$

Init:

$$\text{MSS}[0] = 0$$

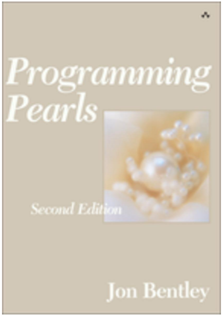
Time: $\Theta(n)$

Maximum-sum subarray



Ulf Grenander $O(n^3) \implies O(n^2)$

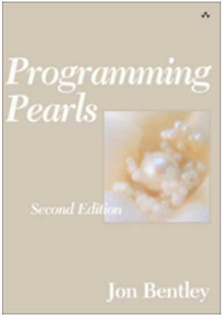
Maximum-sum subarray



Ulf Grenander $O(n^3) \implies O(n^2)$

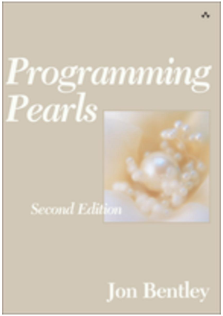
Michael Shamos $O(n \log n)$, onenight

Maximum-sum subarray



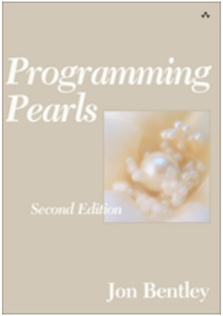
Ulf Grenander	$O(n^3) \implies O(n^2)$
Michael Shamos	$O(n \log n)$, onenight
Jon Bentley	Conjecture: $\Omega(n \log n)$

Maximum-sum subarray



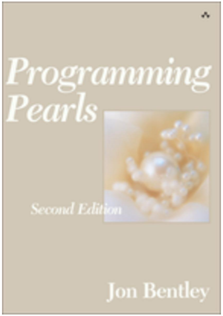
Ulf Grenander	$O(n^3) \implies O(n^2)$
Michael Shamos	$O(n \log n)$, onenight
Jon Bentley	Conjecture: $\Omega(n \log n)$
Michael Shamos	Carnegie Mellon seminar

Maximum-sum subarray



Ulf Grenander	$O(n^3) \implies O(n^2)$
Michael Shamos	$O(n \log n)$, onenight
Jon Bentley	Conjecture: $\Omega(n \log n)$
Michael Shamos	Carnegie Mellon seminar
Jay Kadane	$O(n)$,

Maximum-sum subarray



Ulf Grenander $O(n^3) \implies O(n^2)$

Michael Shamos $O(n \log n)$, onenight

Jon Bentley Conjecture: $\Omega(n \log n)$

Michael Shamos Carnegie Mellon seminar

Jay Kadane $O(n)$, ≤ 1 minute

Maximum-product subarray

Maximum-product subarray (Problem 7.4)

- Array $A[1 \ldots n]$
- Find maximum-product subarray of A

(1) $a_i \in \mathbb{N}$

(2) $a_i \in \mathbb{Z}$

(3) $a_i \in \mathbb{R}$

Maximum-product subarray

Maximum-product subarray (Problem 7.4)

- Array $A[1 \dots n]$
- Find maximum-product subarray of A

(1) $a_i \in \mathbb{N}$
(2) $a_i \in \mathbb{Z}$
(3) $a_i \in \mathbb{R}$

sum *vs.* product

Maximum-product subarray

Subproblem: MaxP$[i]$, MinP$[i]$

		$\frac{1}{2}$	4	-2	5	$-\frac{1}{5}$	8
MaxP$[i]$	1	$\frac{1}{2}$	4	-2	5	8	64
MinP$[i]$	1	$\frac{1}{2}$	2	-8	-40	-1	-8

$$\text{MaxP}[i] = \max\{\text{MaxP}[i-1] \cdot a_i, \text{MinP}[i-1] \cdot a_i, a_i\}$$
$$\text{MinP}[i] = \min\{\text{MaxP}[i-1] \cdot a_i, \text{MinP}[i-1] \cdot a_i, a_i\}$$

2d

Binary Search (CLRS $4.5-3$)

$$T(n) = 2T(n/2) + \Theta(1)$$

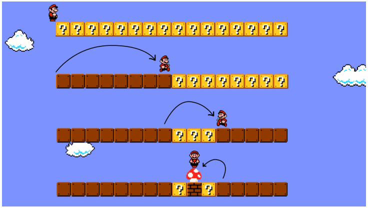
$$T(n) = \Theta(n \lg n)$$

People who analyze algorithms have double happiness.

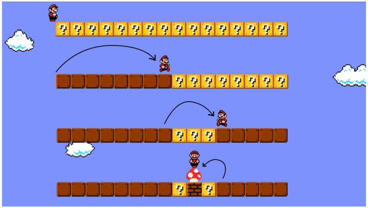
First of all they experience the sheer beauty of elegant mathematical patterns that surround elegant computational procedures.

Then they receive a practical payoff when their theories make it possible to get other jobs done more quickly and more economically.

— Donald E. Knuth (1995)



$$T(n) = \begin{cases} \max\left\{T(\lfloor \frac{n-1}{2} \rfloor), T(\lceil \frac{n-1}{2} \rceil)\right\} + 1, & n > 2 \\ 1, & n = 1 \end{cases}$$



$$T(n) = \begin{cases} \max\left\{T(\lfloor \frac{n-1}{2} \rfloor), T(\lceil \frac{n-1}{2} \rceil)\right\} + 1, & n > 2 \\ 1, & n = 1 \end{cases}$$

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + 1, & n > 2 \\ 1, & n = 1 \end{cases}$$

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + 1, & n > 2 \\ 1, & n = 1 \end{cases}$$

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + 1, & n > 2 \\ 1, & n = 1 \end{cases}$$

$$n = 2^k \implies T(n) = k + 1$$

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + 1, & n > 2 \\ 1, & n = 1 \end{cases}$$

$$n = 2^k \implies T(n) = k + 1$$

$$2^k \leq n < 2^{k+1} \implies T(n) = k + 1$$

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + 1, & n > 2 \\ 1, & n = 1 \end{cases}$$

$$n = 2^k \implies T(n) = k + 1$$

$$2^k \le n < 2^{k+1} \implies T(n) = k + 1$$

$$T(n) = \lfloor \lg n \rfloor + 1$$

Theorem

The worst case time complexity (# of comparisons) of $\textsc{BinarySearch}$ *on an input size of* n $=$ *# of bits in the binary representation of* n.


TREASURE
HUNT
on foot


Hints
&
Tips


Help
Wanted

Analysis of the Mergesort in Section $2.3.1$ of CLRS ($\#$ of Comparisions; $a_i : \infty$ not Counted)

(a) Analyze the worst case ($W(n)$) and the best case ($B(n)$) time complexity of mergesort *as accurately as possible*.
Plot them and explain what you observe.

(b) Analyze the average case ($A(n)$) time complexity of mergesort.
Plot it and explain what you observe.

(c) Prove that: The minimum number of comparisons needed to merge two sorted arrays of equal size m is $2m - 1$.

Analysis of the Mergesort in Section 2.3.1 of CLRS (# of Comparisions; $a_i : \infty$ not Counted)

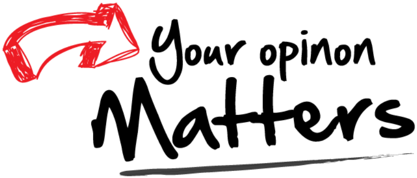
(a) Analyze the worst case ($W(n)$) and the best case ($B(n)$) time complexity of mergesort *as accurately as possible*.
Plot them and explain what you observe.

(b) Analyze the average case ($A(n)$) time complexity of mergesort.
Plot it and explain what you observe.

(c) Prove that: The minimum number of comparisons needed to merge two sorted arrays of equal size m is $2m - 1$.

$W(n)$: Consider $W(n+1)$


Thank You!



Office 302

Mailbox: H016

hfwei@nju.edu.cn