

Disc Week 6

Agenda

1. DC - Finding k^{th} smallest in L
2. DP - Subsequences

1 Selection (Divide & Conquer)

Input: List L and number k

Output: " k^{th} smallest element of L " = $\text{Select}(L, k)$

- Note: 1. $\text{Select}(L, \frac{|L|}{2})$ returns median of L
2. Sorting L and returning k^{th} element gives $O(n \log n)$ algorithm. Let's do better!

- Idea: Pick v and Split L into

$$L_{<v}, L_{=v}, L_{>v}$$

Then

if $k \leq |L_{<v}|$ look in left: $\text{Select}(L_{<v}, k)$

if $k > |L_{<v}| + |L_{=v}|$ look in right: $\text{Select}(L_{>v}, k - |L_{<v}| - |L_{=v}|)$

if $|L_{<v}| < k \leq |L_{<v}| + |L_{=v}|$ return v

- Notice that if we can efficiently choose v so that $|L_{<v}|, |L_{>v}| \approx \frac{|L|}{2}$

$$T(n) \leq T\left(\frac{n}{2}\right) + O(n) \Rightarrow O(n)$$

$\nearrow$ recurse on 1 half $\nearrow$ split into 3 groups $\uparrow$ master's

- How do we pick v in $O(n)$ time?

Ans: Randomly! (Randomly from L that is)

Note: Worst case is $O(n^2)$ if we continually pick v to be the smallest or largest element of L .

- Best case is $O(n)$ if we pick the middle every time. What's the average case? Ans: $O(n)$, why?

Call v "good" if it lies between the 25th and 75th percentile. v then has a 50% chance of being "good." On average we need to choose 2 v 's per iteration to find a good v which gives

$$T(n) \leq T\left(\frac{3n}{4}\right) + O\left(\overset{O(n)}{2 \cdot n}\right) = O(n)!$$

$\nearrow$ Expected running time

[2] Dynamic Programming - Contiguous Subsequence

6.1

Input: List of numbers $a_1, \dots, a_n$

Output: ^{Value of} Contiguous Subsequence (CSS) of maximum Sum.

Ex:

5, 15, -30, 10, -5, 40, 10

cont. SS

• 5, 15, 40 → NOT a CSS

• 10, -5, 40, 10 → Max CSS

How to solve DPs?

- Find a relation that tells you how to solve subproblems given the answers to smaller subproblems.

Idea: Let $T(j) =$ "Max CSS of a_1 through a_j using a_j " (possibly of length zero)

Want: $\max_j T(j)$

Recurrence: $T(j) = \max \{0, a_j + T(j-1)\}$

~~0~~ $\uparrow$
length 0
CSS.

$\uparrow$
use a_j so
put on max CSS
of a_1 through a_{j-1} that uses
 a_{j-1} i.e. $T(j-1)$

Base: $T(0) = 0$

Alg

MaxCSS($a_1, \dots, a_n$)

1. $T(0) = 0$

2. For $i = 1$ to n

$$T(i) = \max \{ 0, a_i + T(i-1) \}$$

3. return $\max_j T(j)$

Runtime: $O(n)$

Correctness:

Claim: $T(j) =$ "Max CSS of a_1 through a_j using a_j (and possibly of zero length)"

If we are able to show the claim, then step 3 returns the $\max_j T(j)$ which we know will be the overall max CSS since $\uparrow$ must end at some a_j , and $\therefore$ be the max CSS that ends ^{exactly} at that a_j .

Let's prove the claim by induction. The base case

$T(0) = 0$ is true since we only have the zero length CSS to consider. Assume $T(j) =$ "Max CSS ..." for all $j < k$.

We want to show $T(k) =$ "Max CSS ...". There

are two options: (1) a_k can be included in some positive CSS or (2) ~~otherwise~~ there are no pos CSS's which a_k is a part of. In case (2) the 0-length CSS must be the max and in case 1 $a_k + T(k-1)$ must be the max CSS (since $T(k-1)$ is a max CSS ending at a_{k-1}). This is exactly what we do at step 2. $\therefore$ The claim is shown.