CS 341

Michael Sung Lee

May 3, 2016

Contents

1. Lecture 1

1.1 Bentley Problem

Given numbers $a_1, ..., a_n$,
find a contiguous subsequence (”block”)
$a_i, a_{i+1}, ..., a_j$, with the largest sum
example: 1, -6, 3, -1, 4, 2, -3, 2

1.1.1 Algorithm 0

Idea - Brute Force

```
1. ans = -∞
2. for i = 1 to n do
3.     for j = i to n do
4.         sum = 0
5.         for k=i to j do
6.             sum = sum + a_k
7.         ans = max{ans, sum}
8. return ans
```

Analysis: How fast is it?
Observe from the inside out

line 5-6: $c(j-i+1) \leq cn$ (c is a machine/compiler dependent constant)
line 4-7: $\leq cn + c'$
line 3-7: $\leq (n-i+1) * (cn + c')$
$\leq cn^2 + c'n$

total: $\leq n(cn^2 + c'n) + c''$
$= O(n^3)$

1.1.2 Algorithm 1

Idea - Don't re-compute sum from scratch

1. ans = -∞
2. for i = 1 to n do
3.   sum = 0
4.   for j = i to n do
5.   sum = sum + a_j
6.   ans = max{ans, sum}
7. return ans

Analysis: $O(n^2)$

1.1.3 Algorithm 2

Idea - Divide and conquer

1. if n == 1
2.   return a_1
3. ans = max{ solve($a_1, ..., a_{\lfloor n/2 \rfloor}$),
   solve($a_{\lfloor n/2 \rfloor + 1}, ..., a_n$) }
4. ans_L = sum = 0
5. for i = $\lfloor n/2 \rfloor$ to 1 do
6.   sum = sum + a_i
7. ans_L = max{ans, sum}
8. ans_R = sum = 0
9. for i = $\lfloor n/2 \rfloor + 1$ to n do
10.   sum = sum + a_i
11. ans_R = max{ans_R, sum}
12. return max{ans, ans_L + ans_R}

Analysis

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{else} \end{cases}$$

Note that this is the same as the mergesort recurrence.

= O($nlog(n)$)

1.1.4 Algorithm 3

Idea - Dynamic programming (solve intermediate sub problems).
Let b_j = max sum over all blocks that end at j.

1, -6, 3, -1, 4, 2, -3, 2

$b_5 = 6$
$b_6 = 8$

Next idea - don't recompute b_j from scratch by computing b_j from b_{j-1} use the equation:

$b_j = \max\{b_{j-1} + a_j,\ a_j\}$

1. b = ans = 0
2. for j = 1 to n
3. $\quad b = \max\{b + a_j,\ a_j\}$
4. $\quad$ ans = max{ans, b}
5. return ans

Analysis: O(n) time