COMPSCI 311: Introduction to Algorithms

Lecture 10: Divide and Conquer

Dan Sheldon

University of Massachusetts Amherst

Divide and Conquer: Recipe

- Divide problem into several parts
- Solve each part recursively
- Combine solutions to sub-problems into overall solution

Learning Goals

	Greedy	Divide and Conquer
Formulate problem		
Design algorithm		✓
Prove correctness	✓	
Analyze running time		✓
Specific algorithms	Dijkstra, MST	

Motivating Problem: Maximum Subsequence Sum (MSS)

- **Input**: array A of n numbers, e.g.

$$A = 4, -3, 5, -2, -1, 2, 6, -2$$

- **Find**: value of the largest subsequence sum

$$A[i] + A[i+1] + \ldots + A[j]$$

  - (empty subsequence allowed and has sum zero)
- MSS in example? 11 (first 7 elements)

Clicker

Which of the following is true for a maximum-sum subsequence?

A. It has more positive than negative numbers

B. It does not start or end with a negative number

C. Any maximal sequence of negative numbers is bordered by a sequence of positive numbers with sum larger in absolute value

A Simple MSS Algorithm

Brute force in $\Theta(n^2)$ (c.f K&T Chapter 2, Exercise 6)

MSS(A)
  Initialize all entries of $n \times n$ array B to zero
  for $i = 1$ to n **do**
    sum = 0
    for $j = i$ to n **do**
      compute sum of $A[i]$ ... $A[j]$
      $B[i, j]$ = sum
  Return maximum value among all $B[i, j]$

Running time? $O(n^2)$. Can we do better?

Divide-and-conquer for MSS

- Recursive solution for MSS
- **Idea**:
  - Find MSS L in left half of array
  - Find MSS R in right half of array
  - Find MSS M for sequence that crosses the midpoint

$$A = \underbrace{4, -3, 5}_{L=6}, \overbrace{-2, -1, \underbrace{2, 6}_{R=8}}^{M=11}, -2$$

- Return $\max(L, R, M)$
- Change one entry to make MSS=R. $-2 \rightarrow -10$
- How to find L, R, M?

MSS(A, left, right)
  if right $-$ left $\leq$ 2 **then** ▷ Base case
    Solve directly and return MSS

  mid $= \lfloor \frac{\text{left+right}}{2} \rfloor$ ▷ Recurse on left and right halves
  L = MSS(A, left, mid)
  R = MSS(A, mid+1, right)

  Set sum $= 0$ and $L' = 0$ ▷ Compute L' (left part of M)
  for i = mid down to 1 **do**
    sum += $A[i]$
    $L' = \max(L', \text{sum})$

  Set sum $= 0$ and $R' = 0$ ▷ Compute R' (right part of M)
  for i mid+1 to right **do**
    sum += $A[i]$
    $R' = \max(R', \text{sum})$
  $M = L' + R'$ ▷ Compute M

  return $\max(L, R, M)$ ▷ Return max

MSS(A, left, right)
  if right $-$ left $\leq$ 2 **then**
    Solve directly and return MSS

  mid $= \lfloor \frac{\text{left+right}}{2} \rfloor$
  L = MSS(A, left, mid)
  R = MSS(A, mid+1, right)

  Set sum $= 0$ and $L' = 0$
  for i = mid down to 1 **do**
    sum += $A[i]$
    $L' = \max(L', \text{sum})$

  Set sum $= 0$ and $R' = 0$
  for i mid+1 to right **do**
    sum += $A[i]$
    $R' = \max(R', \text{sum})$
  $M = L' + R'$

  return $\max(L, R, M)$

Running time?

- Let $T(n)$ be running time of MSS on array of size n
- Two recursive calls on arrays of size $n/2$: $2T(n/2)$
- Work outside of recursive calls: $O(n)$
- Running time

$$T(n) = 2T(n/2) + O(n)$$

Recurrence

- Recurrence

$$T(n) = 2T(n/2) + O(n)$$

- sequence $T(0), T(1), T(2), \ldots$
- $T(n)$ defined in terms of smaller values
- For running time, choose any convenient base case: $T(1) = O(1)$, $T(2) = O(1)$

Goal: "solve" the recurrence = find simple expression for $T(n)$ for all n

---

Recurrence

- Recurrence (with convenient base case)

$$T(n) = 2T(n/2) + O(n)$$
$$T(1) = O(1)$$

- First, let's use definition of Big-O:

$$T(n) \leq 2T(n/2) + cn$$
$$T(1) \leq c$$

Clicker

$$T(n) = 2T(n/2) + O(n) \qquad T(n) \leq 2T(n/2) + cn$$
$$T(1) = O(1) \qquad T(1) \leq c$$

Why is it OK to use the same value of c in both instances of the big-O definition?

A. It's not OK. You just took a shortcut. (By the way, you forgot about n_0.)

B. Take $c = \min\{c_1, c_2\}$ where c_1 and c_2 are the values from each instance.

C. Take $c = \max\{c_1, c_2\}$ where c_1 and c_2 are the values from each instance.

Recurrence

- Same recurrence with change of variable

$$T(m) \le 2T(m/2) + cm, \quad m \ge 2$$
$$T(1) \le c$$

  - no difference, but sometimes helpful conceptually
  - n = original input size, m = generic input size

- Three approaches to solve it

  1. Unrolling
  2. Recursion tree (another version of unrolling)
  3. Guess and verify (proof by induction)

Recurrence Solving (1): Unrolling

- **Idea 1**: "unroll" the recurrence

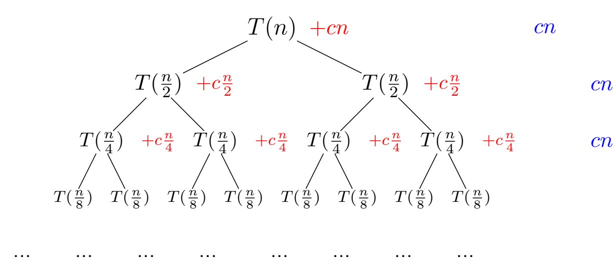
$$
\begin{aligned}
T(n) &\le 2T(n/2) + cn && m = n \\
&\le 2\Big[2T(n/4) + c(n/2)\Big] + cn && m = n/2 \\
&= 4T(n/4) + 2cn \\
&\le 4\Big[2T(n/8) + c(n/4)\Big] + 2cn && m = n/4 \\
&= 8T(n/8) + 3cn \\
&\le \dots \\
&\dots \\
&\le nT(1) + \log_2(n) \cdot cn = O(n \log n)
\end{aligned}
$$

Clicker

Suppose we have the recurrence $T(n) = T(n/2) + T(n/3)$. What do we get after two unrollings?

A. $T(n/4) + T(n/9)$
B. $T(n/4) + 2T(n/6) + T(n/9)$
C. $2T(n/6)$
D. $T(n/4) + T(n/6) + T(n/9)$

Recurrence Solving (2): Recursion Tree



- $\log_2(n)$ levels $\times$ cn work per level
- **Conclusion:** $T(n) \le cn \log n$

Recurrence Solving (3): Guess and Verify

$$T(n) \le 2T(n/2) + cn$$
$$T(2) \le c$$

- Guess solution. $T(n) \le cn \log n$. Prove by (strong) induction.
- Base case

$$T(2) \le c < c \cdot 2 = c \cdot 2 \log 2 \quad \checkmark$$

Induction step

Strong induction:

- Assume $T(m) \le c \cdot m \log m$ for all $m < n$
- Prove $T(n) \le c \cdot n \log n.$

$$
\begin{aligned}
T(n) &\le 2T(n/2) + cn \\
&\le 2c(n/2)\log(n/2) + cn \quad \text{by ind. hyp. } m = n/2 \\
&= cn(\log n - 1) + cn \\
&= cn \log n
\end{aligned}
$$

Summary

Three approaches to solve first recurrence:

1. Unrolling ✓
2. Recursion tree ✓
3. Guess and verify (proof by induction) ✓

Next: other recurrences!