CH-231-A

Algorithms and Data Structures

ADS

Lecture 11

Dr. Kinga Lipskoch

Spring 2020

Fibonacci Numbers (1)

Recursive definition:

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

Sequence: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$

$\qquad n$ 0 1 2 3 4 5 ...

Output: return the n-th Fibonacci number

Fibonacci Numbers (2)

Naive algorithm:

Implement the recursion as in the definition.

Fibonacci(n)
1 **if** (n < 2)
2   **return** n
3 **else**
4   **return** Fibonacci (n - 1) + Fibonacci (n - 2)

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

Fibonacci Numbers (3)

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

Lower bound:
$T(n-1) \approx T(n-2)$
$T(n) = 2T(n-2) + \Theta(1)$
$T(n) = 4T(n-4) + \Theta(1)$
$T(n) = 8T(n-6) + \Theta(1)$
...
$T(n) = 2^k T(n-2k) + \Theta(1) \Longrightarrow n - 2k = 0 \Longrightarrow k = n/2$
$T(n) = 2^{n/2} T(0) + \Theta(1)$
$\Longrightarrow T(n) = \Omega(2^{n/2})$, i.e., exponential time

Fibonacci Numbers (4)

Side note: A tighter lower bound is the following

$$T(n) = \Omega(\Phi^n),$$

because #leaves in rec. tree = Fibonacci(n) $\times \Theta(1)$
where Φ is the golden ratio

$$\phi = (1 + \sqrt{5})/2$$

The closed form for Fibonacci(n) explains the time complexity
from above.

Fibonacci Numbers (5)

Bottom up approach:

Avoid recursion, i.e.,
compute $F_0, F_1, F_2, ..., F_n$ in the given order instead, forming each
number by summing the two previous.

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

Fibonacci Numbers (6)

Closed form (rounded to next integer):
$F_n = \Phi^n/\sqrt{5}$ where $\Phi = (1 + \sqrt{5})/2$
(proof by induction).

Compute by "Power of a number" recursion.
$T(n) = \Theta(\lg n)$
But: numerical problems may occur (floating-point arithmetic).

Fibonacci Numbers (7)

Matrix representation:

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$
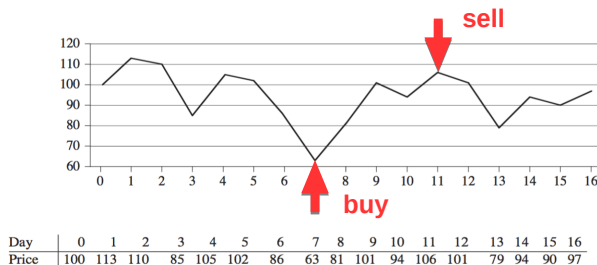
(proof by induction).

Compute by "Power of a number" recursion
(using a generalization to 2x2 matrices).
$T(n) = \Theta(\lg n)$
And: uses integers only (no floating-point errors).

Maximum-Subarray Problem (1)

▶ Motivation scenario: buy & sell stock

▶ Input: a sequence of numbers

▶ Output: subsequence that results in the highest profit

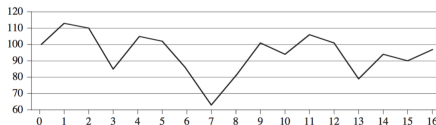| Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97

Maximum-Subarray Problem (2)

Brute-Force algorithm:

▶ Try every pair of days.

Number of pairs:

$$\binom{n}{k} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2} = \Theta(n^2)$$

Maximum-Subarray Problem (3)

▶ Transformation:
   ▶ Consider daily change in price instead

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

▶ Maximum-subarray problem:
   ▶ Input: a sequence of numbers $< a_1, a_2, ..., a_n >$ of positive and negative numbers (otherwise it does not make sense)
   ▶ Output: contiguous, non-empty subsequence
   $< a_i, a_{i+1}, ..., a_j >$ with $i \geq 1, j \leq n$,

   such that $\displaystyle\sum_{k=i}^{j} a_k$ is maximized
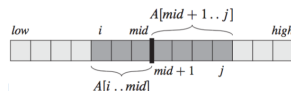
Maximum-Subarray Problem (4)

Divide & Conquer:

▶ Idea: Divide A into 2 pieces $\implies$ maximum subarray is either
  ▶ entirely in the subarray $A[low...mid]$, or
  ▶ entirely in the subarray $A[mid + 1...high]$, or
  ▶ crossing the midpoint
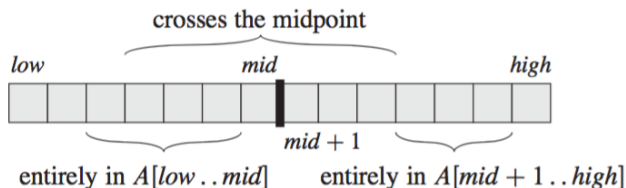
Maximum-Subarray Problem (5)

FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)

```
 1  left-sum = -∞
 2  sum = 0
 3  for i = mid downto low
 4      sum = sum + A[i]
 5      if sum > left-sum
 6          left-sum = sum
 7          max-left = i
 8  right-sum = -∞
 9  sum = 0
10  for j = mid + 1 to high
11      sum = sum + A[j]
12      if sum > right-sum
13          right-sum = sum
14          max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

Time complexity: $\Theta(n)$

Maximum-Subarray Problem (6)

FIND-MAXIMUM-SUBARRAY($A, low, high$)

$\Theta(1)$

1   **if** $high == low$
2   **return** ($low, high, A[low]$)   // base case: only one element

$2T(n/2)$

3   **else** $mid = \lfloor (low + high)/2 \rfloor$
4   ($left\text{-}low, left\text{-}high, left\text{-}sum$) =
          FIND-MAXIMUM-SUBARRAY(A, low, mid)
5   ($right\text{-}low, right\text{-}high, right\text{-}sum$) =
          FIND-MAXIMUM-SUBARRAY($A, mid + 1, high$)

$f(n)$

6   ($cross\text{-}low, cross\text{-}high, cross\text{-}sum$) =
          FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)
7   **if** $left\text{-}sum \geq right\text{-}sum$ and $left\text{-}sum \geq cross\text{-}sum$
8   **return** ($left\text{-}low, left\text{-}high, left\text{-}sum$)
9   **elseif** $right\text{-}sum \geq left\text{-}sum$ and $right\text{-}sum \geq cross\text{-}sum$
10   **return** ($right\text{-}low, right\text{-}high, right\text{-}sum$)
11   **else return** ($cross\text{-}low, cross\text{-}high, cross\text{-}sum$)

▶ Recurrence:
$$T(n) = 2T(n/2) + \Theta(n)$$

▶ Solution:
$a = 2, b = 2, n^{\log_b a} = n, f(n) = \Theta(n) \Longrightarrow$ Case 2
Thus, $T(n) = \Theta(n \lg n)$

Matrix Multiplication

Problem: Input A, B, Output C

$$\left. \begin{array}{l} A = [a_{ij}], B = [b_{ij}]. \\ C = [c_{ij}] = A \cdot B. \end{array} \right\} \quad i, j = 1, 2, \dots, n.$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Matrix Multiplication: Standard Algorithm

Standard algorithm:

```
1    for i = 1 to n do
2      for j = 1 to n do
3        c[i][j] = 0
4        for k = 1 to n do
5          c[i][j] = c[i][j] + a[i][k] * b[k][j]
```

Time complexity: $T(n) = \Theta(n^3)$

Matrix Multiplication: Divide & Conquer (1)

Divide & Conquer:

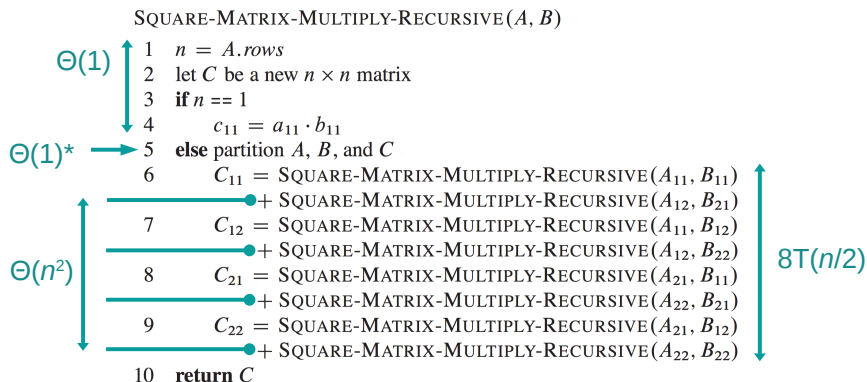Idea: $n \times n$ matrix $= 2 \times 2$ matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ \hline t & u \end{bmatrix} = \begin{bmatrix} a & b \\ \hline c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ \hline g & h \end{bmatrix}$$

$$C \quad = \quad A \quad \cdot \quad B$$

Combining subproblem solutions:

$$\left. \begin{array}{l} r = ae + bg \\ s = af + bh \\ t = ce + dg \\ u = cf + dh \end{array} \right\}$$ 8 mults of $(n/2) \times (n/2)$ submatrices
4 adds of $(n/2) \times (n/2)$ submatrices

Matrix Multiplication: Divide & Conquer (2)

SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)

$\Theta(1)$

1　　$n = A.rows$
2　　let C be a new $n \times n$ matrix
3　　**if** $n == 1$
4　　　　$c_{11} = a_{11} \cdot b_{11}$

$\Theta(1)^*$ →

5　　**else** partition A, B, and C
6　　　　$C_{11} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{11}, B_{11})
　　　　　　　$+$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{12}, B_{21})
7　　　　$C_{12} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{11}, B_{12})
　　　　　　　$+$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{12}, B_{22})
8　　　　$C_{21} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{21}, B_{11})
　　　　　　　$+$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{22}, B_{21})
9　　　　$C_{22} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{21}, B_{12})
　　　　　　　$+$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{22}, B_{22})
10　　**return** C

$\Theta(n^2)$　　　　　　　　　　　　　　　　　　　　　　　　　　　　$8T(n/2)$

* with index calculations,
otherwise $\Theta(n^2)$ for copying entries

Matrix Multiplication: Divide & Conquer (3)

Recurrence:

$$T(n) = 8T(n/2) + \Theta(n^2)$$

#subproblems

subproblem size

work dividing
and combining

$n^{log_b a} = n^{log_2 8} = n^3, f(n) = \Theta(n^2)$
meaning that $f(n) = \Theta(n^{3-\epsilon})$, where $\epsilon = 1$
Case 1: $T(n) = \Theta(n^3)$.

Not better than the standard algorithm.

Matrix Multiplication: Conclusions

- ▶ Lessons learned:
    - ▶ #additions goes away
      (constant factor)
    - ▶ #multiplications not ⇒ recursive case
      (they make the tree "bushy")
- ▶ What to do?
    - ▶ Try to reduce #multiplications
    - ▶ It is ok to have more additions