CH08-320201

Algorithms and Data Structures

ADS

Lecture 5

Dr. Kinga Lipskoch

Spring 2019

Recall: Divide & Conquer

Design paradigm:

1. Divide the problem (instance) into subproblems.
2. Conquer the subproblems by solving them recursively.
3. Combine subproblem solutions.

Recall: Merge Sort

1. Divide: Trivial
2. Conquer: Recursively sort 2 subarrays
3. Combine: Linear-time merge

$$T(n) = \underbrace{2}_{\text{\#subproblems}}T(\underbrace{n/2}_{\text{subproblem size}}) + \underbrace{\Theta(n)}_{\text{work dividing and combining}}$$

1

Master Method on Merge Sort

$T(n) = 2T(n/2) + n$
$a = 2,\ b = 2$

$n^{log_b a} = n$
$f(n) = n$

Case 2:
$f(n) = \Theta(n)$,
Thus, $T(n) = \Theta(n \lg n)$.

Power of a Number

- Problem:
  - Input: numbers $a \in \mathbb{R}$ and $n \in \mathbb{N}$.
  - Output: a^n
- Naive algorithm:
  - $T(n) = \Theta(n)$
- Divide & Conquer:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

- Recurrence:
  - $T(n) = T(n/2) + \Theta(1)$
- Solution:
  - $a = 1, b = 2, n^{log_b a} = 1, f(n) = \Theta(1) \Longrightarrow$ Case 2
  - Thus, $T(n) = \Theta(\lg n)$

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

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)
if (n < 2)
    return n
else
    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:

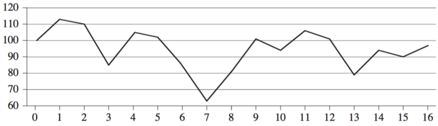
- Try every pair of days.

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

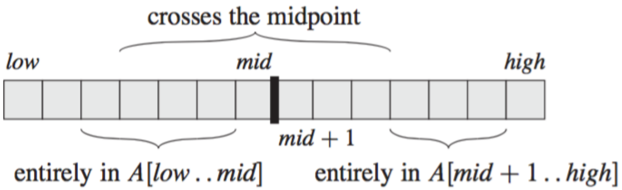
- Maximum-subarray problem:
  - Input: a sequence of numbers $< a_1, a_2, ..., a_n >$ of positive and negative numbers (otherwise it does not make sense)
  - Output: contiguous, non-empty subsequence

    $< a_i, a_{i+1}, ..., a_j >$ with $i \geq 1, j \leq n$, so that $\sum_{k=i}^{j}$ is maximized

Maximum-Subarray Problem (4)

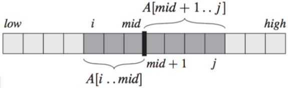
Divide & Conquer:

- Idea: Divide A into 2 pieces $\Longrightarrow$ 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)
left-sum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```



Time complexity: $\Theta(n)$

Maximum-Subarray Problem (6)

$\Theta(1)$

$2T(n/2)$

$f(n)$

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])            // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)
    elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-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 (1)

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 (2)

Standard algorithm:

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

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

Matrix Multiplication (3)

Divide & Conquer:

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

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

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

Combining subproblem solutions:

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