Data Structures and Algorithms: Lecture 3

Barbara Morawska

August 7, 2018

Divide and conquer algorithms

Analysis of the running time of a "divide and conquer" algorithm:

- Recursive calls to itself
- "bottoms-up" with the smallest problem (base case)

Example: MERGE-SORT

Recurrence

Define a recurrence

A recurrence (recursive equation) is an equation or inequality that describes a function in terms of its value on smaller inputs.

Example: The worst case of MERGE-SORT is defined by recurrence:

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

The solution for this equation is $O(n \lg n)$ (or $T(n) = O(n \lg n)$).

The problem: how to compute the solution?

Technicalities

- If an input of $\textsc{Merge-sort}$ is an array of odd length ($A.length$ is odd), then

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1 \end{cases}$$

  - We ignore ceilings and floors.(They do not change asymptotic values.)
  - We ignore base case $T(n) = \Theta(1)$ for small n.
- Hence the recurrence for $\textsc{Merge-sort}$ is:

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

Recurrence

Examples of recurrences:

- Unequal sizes of subproblems:
  $T(n) = T(2/3n) + T(1/3n) + \Theta(n)$
  What is the solution?
- Only one smaller subproblem.
  Recursive version of LINEAR SEARCH:
  $T(n) = T(n-1) + \Theta(1)$
  What is the solution?

Solving recurrence

3 methods to solve recurrences:

- substitution method
- recursion tree method
- master method

Maximum-subarray problem

Maximum-subarray problem:

INPUT: array of numbers

OUTPUT: contiguous subarray whose values have the greatest sum.

Changes in the stock prices

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

Note: for the problem to be non-trivial some values in the input array must be negative!

Example of the problem

When to buy and sell to get the greatest 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

43

First solution:

- Compare prices for each pair (i, j), $i < j$).
- Output the pair with the maximal difference.
- Complexity: n^2 comparisons. $\Theta(n^2)$.
- Use MAXIMUM-SUBARRAY algorithm to get complexity in $o(n^2)$.
  (Running time much less than n^2.)

Problem: how to use MAXIMUM-SUBARRAY algorithm to solve this problem? How to get a fast MAXIMUM-SUBARRAY algorithm?

MAXIMUM-SUBARRAY algorithm

How to use MAXIMUM-SUBARRAY algorithm to solve maximal profit problem?

Use array of changes in the prices as the input for MAXIMUM-SUBARRAY algorithm :

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

Brute force algorithm:

- Check all subarrays: $\binom{n-1}{2}$
- Complexity: $\Theta(n^2)$

Use divide-and-conquer algorithm

INPUT: $A[low..high]$
OUTPUT: max subarray

- Divide A into two halves: $A[low..mid], A[mid..high]$
- Maximal subarray can be located in:
  - $A[low..mid]$
  - $A[mid..high]$
  - crossing the mid-point

Idea:

- Find max-subarray for $A[low..mid]$ (recursive call)
- Find max-subarray for $A[mid..high]$ (recursive call)
- Find max-subarray crossing the mid-point
- Compare them and output the one with the largest sum.

Notice: finding max-subarray crossing the mid-point is not an instance of the same problem.

Algorithm for finding max-subarray crossing the mid-point

Procedure Find-max-crossing-subarray(A, low, mid, $high$)

```
left−sum = −∞
sum = 0
for i = mid down to low do
    sum = sum + A[i]
    if sum > left−sum then
        left−sum = sum
        max−left = i
right−sum = −∞
sum = 0
for j = mid + 1 to high do
    sum = sum + A[j]
    if sum > right−sum then
        right−sum = sum
        max−right = j
return (max−left, max−right, left−sum + right−sum)
```

Running time of Find-max-crossing-subarray

Cost of two loops:

- Two loops:
  lines 3 – 7
  lines 10–14
- Each iteration costs $\Theta(1)$. How many interations?
  lines 3–7: at most $mid - low + 1$
  lines 10–14: at most $high - mid$
  Hence: n times.

Hence the algorithm runs in $\Theta(n)$ time.

Example

Run the algorithm on the array:

$$A = (13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7)$$

FIND-MAXIMUM-SUBARRAY

```
Procedure Find-maximum-subarray(A, low, high)
if high = low then
    return (low, high, A[low])
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 then
    return (left−low, left−high, left−sum)
else if right−sum ≥ left−sum and right−sum ≥ cross−sum then
    return (right−low, right−high, right−sum)
else
    return (cross−low, cross−high, cross−sum)
```

Analysis of running time

Find a recurrence

- If $n = 1$ (base case), lines 1, 2: $T(1) = \Theta(1)$
- If $n > 1$
  - line 1 – 4 computing *mid*, constant time $\Theta(1)$
  - line 5,6 recursive calls $2T(n/2)$
  - line 7 – $\Theta(n)$
  - lines 8 –9 constant: $\Theta(1)$.

Hence:

$$\begin{aligned} T(n) &= \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1) \\ &= 2T(n/2) + \Theta(n) \end{aligned}$$

The same complexity as MERGE-SORT!

Algorithms for matrix multiplication

INPUT: two square $n \times n$-matices $A = (a_{ij})$ and $B = (b_{ij})$
OUTPUT: a square $n \times n$-matrix $C = A \cdot B$, $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$
There are n^2 entries in C, each is computed as a sum of n values.

Procedure Square-matrix-multiply(A, B)

```
n = A.rows
Let C be a new n × n-matrix
for i = 1 to n do
    for j = 1 to n do
        c_ij = 0
        for k = 1 to n do
            c_ij = c_ij + a_ik b_kj
return C
```

Running time is $\Theta(n^3)$.
We look for $o(n^3)$ algorithm (much faster than n^3 for big values of n).

Divide-and-conquer matrix multiplication

- Divide A and B:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

- Then C is computed as:

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

- $C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$
  $C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$
  $C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$
  $C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$
- Computing C_{ij} takes computing two multipications of $n/2 \times n/2$-matrices.

Pseudocode

Procedure Square-matrix-multiply-recursive(A, B)

$n = A.rows$
Let C be a new $n \times n$-matrix
if $n == 1$ **then**
    $c_{11} = a_{11} \cdot b_{11}$
else
    partition A, B, C as explained above
    $C_{11} = \textsc{Square-matrix-multiply-recursive}(A_{11}, B_{11}) + \textsc{Square-matrix-multiply-recursive}(A_{12}, B_{21})$
    $C_{12} = \textsc{Square-matrix-multiply-recursive}(A_{11}, B_{12}) + \textsc{Square-matrix-multiply-recursive}(A_{12}, B_{22})$
    $C_{21} = \textsc{Square-matrix-multiply-recursive}(A_{21}, B_{11}) + \textsc{Square-matrix-multiply-recursive}(A_{22}, B_{21})$
    $C_{22} = \textsc{Square-matrix-multiply-recursive}(A_{21}, B_{12}) + \textsc{Square-matrix-multiply-recursive}(A_{22}, B_{22})$
return C

Running time of recursive matrix multiplication

Define a recurrence for the algorithm

- line 4 (base case) $\Theta(1)$ (The remaining cases are for $n > 1$)
- line 6: copying the matices would take $\Theta(n^2)$.
  One can divide them by redefining indexes (using pointers) in $\Theta(1)$.
- lines 7 – 10: $8 \cdot T(n/2)$ plus time for 4 additions.
  Addition takes $4\Theta((n/2)^2) = 4\Theta(n^2) = \Theta(n^2)$

Hence the recurrence is:

$$\begin{aligned} T(n) &= \Theta(1) + 8T(n/2) + \Theta(n^2) \\ &= 8 \cdot T(n/2) + \Theta(n^2) \end{aligned}$$

We will see that the solution for such recurrence is $\Theta(n^3)$

Strassen's method for multiplication of matrices

Try to replace multiplications with additions.

Idea

- Divide the input matrices A, B as before in $\Theta(1)$ time.
- Create 10 $n/2 \times n/2$-matrices: $S_1, S_2, \ldots, S_{10}$ using only addition on the small matrices. – $\Theta(n^2)$
- recursively call the procedure to obtain 7 $n/2 \times n/2$-matrices: $P_1, P_2, \ldots, P_7$
- Use addition on the obtained matrices to obtain $C_{11}, C_{12}, C_{21}, C_{22}$. – $\Theta(n^2)$.

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if} \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases}$$

$$T(n) = \Theta(n^{\lg 7})$$

Additional matrices

$$
\begin{array}{l|l}
S_1 = B_{12} - B_{22} & P_1 = A_{11} \cdot S_1 \\
S_2 = A_{11} + A_{12} & P_2 = S_2 \cdot B_{22} \\
S_3 = A_{21} + A_{22} & P_3 = S_3 \cdot B_{11} \\
S_4 = B_{21} - B_{11} & P_4 = A_{22} \cdot S_4 \\
S_5 = A_{11} + A_{22} & P_5 = S_5 \cdot S_6 \\
S_6 = B_{11} + B_{22} & P_6 = S_7 \cdot S_8 \\
S_7 = A_{12} - A_{22} & P_7 = S_7 \cdot S_8 \\
S_8 = B_{21} + B_{22} & \\
S_9 = A_{11} - A_{21} & \\
S_{10} = B_{11} + B_{12} &
\end{array}
$$

Computing C

$C_{11} = P_5 + P_4 - P_2 + P_6$,
$C_{12} = P_1 + P_2$, $C_{21} = P_3 + P_4$
$C_{22} = P_5 + P_1 - P_3 - P_7$