Advanced Core in Algorithm Design #4
算法設計要論 第 4 回

Yasushi Kawase
河瀬 康志

Oct. 25th, 2022

last update: 9:37am, October 30, 2022

Schedule

Lec. #	Date	Topics
1	10/4	Introduction, Stable matching
2	10/11	Basics of Algorithm Analysis, Greedy Algorithms (1/2)
3	10/18	Greedy Algorithms (2/2)
4	10/25	Divide and Conquer (1/2)
5	11/1	Divide and Conquer (2/2)
6	11/8	Dynamic Programming (1/2)
7	11/15	Dynamic Programming (2/2)
—	11/22	Thursday Classes
8	11/29	Network Flow (1/2)
9	12/6	Network Flow (2/2)
10	12/13	NP and Computational Intractability
11	12/20	Approximation Algorithms (1/2)
12	12/27	Approximation Algorithms (2/2)
13	1/10	Randomized Algorithms

Outline

Divide-and-Conquer

- Divide up problem into several subproblems
  divide problem of size n into a subproblems of size n/b
- Solve each subproblems recursively
- Combine solutions to subproblems into overall solution
  combine in $f(n)$ time

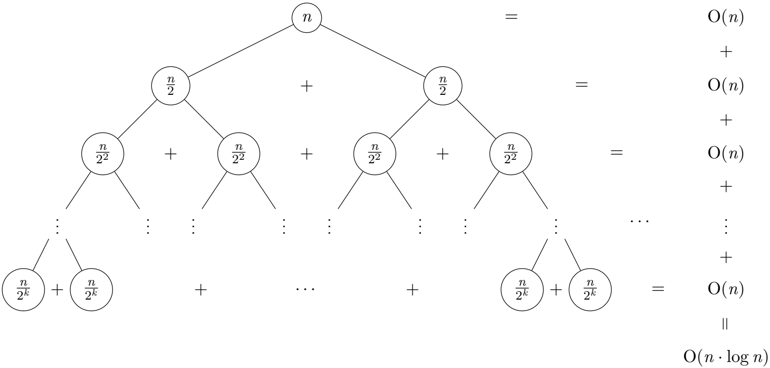
- Total computational time for a problem of size n satisfies

$$T(n) = aT(n/b) + f(n)$$

- $T(n) = \mathrm{O}(1)$ when n is less than some bound

Typical Example

$$T(n) = 2 \cdot T(n/2) + \mathrm{O}(n) \longrightarrow T(n) = \mathrm{O}(n \log n)$$



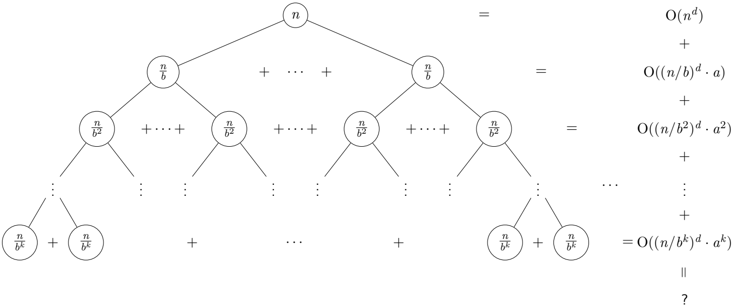
Reccurence relations

Recurrence relations	Computational time
$T(n) = T(n/2) + \mathrm{O}(1)$	$T(n) = \mathrm{O}(\log n)$
$T(n) = 2 \cdot T(n/2) + \mathrm{O}(1)$	$T(n) = \mathrm{O}(n)$
$T(n) = 2 \cdot T(n/2) + \mathrm{O}(n)$	$T(n) = \mathrm{O}(n \log n)$
$T(n) = 3 \cdot T(n/2) + \mathrm{O}(n)$	$T(n) = \mathrm{O}(n^{\log_2 3})$
$T(n) = aT(n/b) + \mathrm{O}(n^d)$	$T(n) = \begin{cases} \mathrm{O}(n^d) & \text{if } d > \log_b a \\ \mathrm{O}(n^d \log n) & \text{if } d = \log_b a \\ \mathrm{O}(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$

$a > 0, b > 1, d \geq 0$

Proof sketch

$$T(n) = a \cdot T(n/b) + \mathrm{O}(n^d) \longrightarrow T(n) = \begin{cases} \mathrm{O}(n^d) & \text{if } d > \log_b a \\ \mathrm{O}(n^d \log n) & \text{if } d = \log_b a \\ \mathrm{O}(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$



Quiz

Which is the most appropriate computational time for the following?

$$T(n) = 4 \cdot T(n/2) + \mathrm{O}(n)$$

1. $T(n) = \mathrm{O}(n^{1/2})$
2. $T(n) = \mathrm{O}(n)$
3. $T(n) = \mathrm{O}(n \log n)$
4. $T(n) = \mathrm{O}(n^2)$
5. $T(n) = \mathrm{O}(2^n)$

Outline

1. Basics of Divide-and-Conquer
2. Sorting
3. Matrix multiplication
4. Closest Pair of Points

Sorting problem

Problem

- Input: a list L of n elements from a totally ordered universe
- Goal: rearrange them in ascending order

Examples

- $[2, 3, 1] \longrightarrow [1, 2, 3]$
- $[4, 2, 8, 5, 7] \longrightarrow [2, 4, 5, 7, 8]$
- ["s","o","r","t"] $\longrightarrow$ ["o","r","s","t"]

Merge sort

$\texttt{MergeSort}(L)$

if $|L| \leq 1$ **then Return** L;
Divide L into equal-sized sublists A and B;
$A \leftarrow \texttt{MergeSort}(A)$;
$B \leftarrow \texttt{MergeSort}(B)$;
$L \leftarrow \texttt{Merge}(A, B)$;
Return L;

- $\texttt{Merge}(A, B)$ can be computed in $\mathrm{O}(|A| + |B|)$ times

  $\texttt{Merge}([3, 7, 12, 18], [2, 11, 15, 23]) \to [2, 3, 7, 11, 12, 15, 18, 23]$

- the total computational time is $T(n) = 2T(n/2) + \mathrm{O}(n)$

  → $T(n) = \mathrm{O}(n \log n)$

Merge sort

MergeSort(L)

if $|L| \leq 1$ **then Return** L;
Divide L into equal-sized sublists A and B;
$A \leftarrow$ MergeSort(A);
$B \leftarrow$ MergeSort(B);
$L \leftarrow$ Merge(A, B);
Return L;

- Merge(A, B) can be computed in $\mathrm{O}(|A| + |B|)$ times

  Merge($[3, 7, 12, 18], [2, 11, 15, 23]$) $\rightarrow [2, 3, 7, 11, 12, 15, 18, 23]$

- the total computational time is $T(n) = 2T(n/2) + \mathrm{O}(n)$

  → $T(n) = \mathrm{O}(n \log n)$

Lower bound of comparisons

Theorem

Comparison sorting requires $\Omega(n \log n)$ comparisons

- there are $n!$ possible orderings
- if an algorithm always completes after at most k comparisons, it cannot distinguish more than 2^k cases

→ $2^k \geq n! \implies k = \Omega(n \log n)$

Outline

Matrix multiplication

Problem

Input Given two $n \times n$ matrices A and B

Goal output their product $C = AB$

naive algorithm: $\Theta(n^3)$ time ($\because c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$)

→ improve it to $\mathrm{O}(n^{2.81})$

Example $(n = 3)$

$$\underbrace{\begin{pmatrix} 2 & 3 & -1 \\ 1 & 4 & 3 \\ 2 & 1 & 5 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} 3 & 1 & 2 \\ 2 & -4 & 2 \\ -2 & 3 & 1 \end{pmatrix}}_{B} = \underbrace{\begin{pmatrix} 14 & -13 & 9 \\ 5 & -6 & 13 \\ -2 & 13 & 11 \end{pmatrix}}_{C}$$

Approach

- partition A and B into $\frac{n}{2} \times \frac{n}{2}$ blocks

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

- the product C is

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

- straightforward application of divide-and-conquer
  $T(n) = 8\,T(n/2) + \mathrm{O}(n^2)$ → $T(n) = \mathrm{O}(n^3)$ (not improved)

- Can we reduce the number of multiplications?

Approach

- partition A and B into $\frac{n}{2} \times \frac{n}{2}$ blocks

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

- the product C is

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

- straightforward application of divide-and-conquer
  $T(n) = 8\,T(n/2) + \mathrm{O}(n^2)$ ➞ $T(n) = \mathrm{O}(n^3)$ (not improved)

- Can we reduce the number of multiplications?
  YES! $8 \to 7$ is possible

Strassen's trick

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11}(B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12})B_{22} \\ P_3 &= (A_{21} + A_{22})B_{11} \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21})(B_{11} + B_{12}) \end{aligned}$$

$$T(n) = 7\,T(n/2) + \mathrm{O}(n^2) \longrightarrow T(n) = \mathrm{O}(n^{\log_2 7}) = \mathrm{O}(n^{2.81})$$

$\log_2 7 = 2.80735\ldots$

Strassen's Algorithm

$\texttt{Strassen}(n, A, B)$ (assume n is a power of 2)

if $n = 1$ **then Return** AB;
$P_1 \leftarrow \texttt{Strassen}(n/2, A_{11}, B_{12} - B_{22})$;
$P_2 \leftarrow \texttt{Strassen}(n/2, A_{11} + A_{12}, B_{22})$;
$P_3 \leftarrow \texttt{Strassen}(n/2, A_{21} + A_{22}, B_{11})$;
$P_4 \leftarrow \texttt{Strassen}(n/2, A_{22}, (B_{21} - B_{11})$;
$P_5 \leftarrow \texttt{Strassen}(n/2, A_{11} + A_{22}, B_{11} + B_{22})$;
$P_6 \leftarrow \texttt{Strassen}(n/2, A_{12} - A_{22}, B_{21} + B_{22})$;
$P_7 \leftarrow \texttt{Strassen}(n/2, A_{11} - A_{21}, B_{11} + B_{12})$;
$C_{11} \leftarrow P_5 + P_4 - P_2 + P_6$;
$C_{12} \leftarrow P_1 + P_2$;
$C_{21} \leftarrow P_3 + P_4$;
$C_{22} \leftarrow P_1 + P_5 - P_3 - P_7$;
Return C;

Theorem

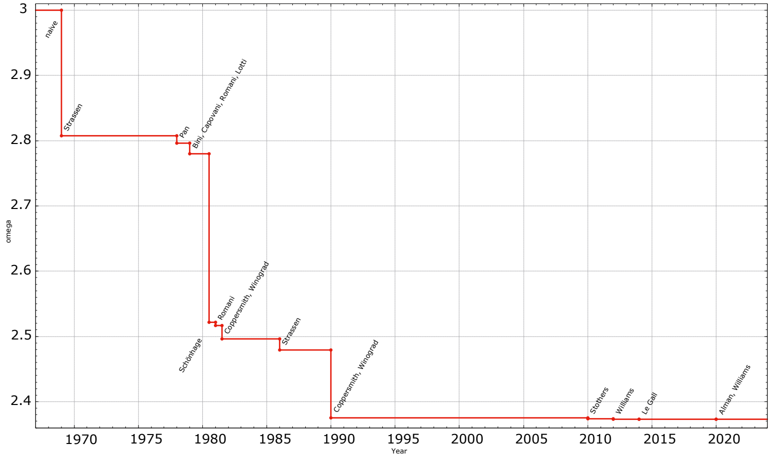
The running time of Strassen's algorithm is $\mathrm{O}(n^{\log_2 7}) = \mathrm{O}(n^{2.81})$

State of the art

- Upper bound: $O(n^{2.3728596})$ [Alman and Williams 2020]

  $O(n^{2.37188})$? [Duan, Wu, Zhou 2022+]

- Lower bound: $\Omega(n^2)$



https://en.wikipedia.org/wiki/Matrix_multiplication

Outline

Closest pair of points problem

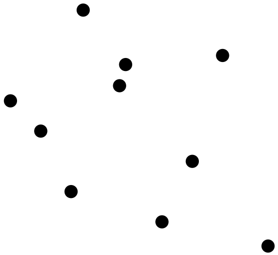
Problem

Input $p_1, p_2, \dots, p_n \in \mathbb{R}^2$ $(p_i = (x_i, y_i))$

$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Goal find a pair (p_i, p_j) that minimizes the distance $d(p_i, p_j)$

- naive algorithm (check all pairs): $\Theta(n^2)$ time
- divide-and-conquer based algorithm: $\mathrm{O}(n \log n)$ time



Closest pair of points problem

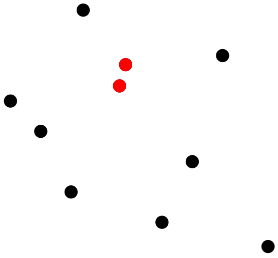
Problem

Input $p_1, p_2, \ldots, p_n \in \mathbb{R}^2$ $(p_i = (x_i, y_i))$

$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Goal find a pair (p_i, p_j) that minimizes the distance $d(p_i, p_j)$

- naive algorithm (check all pairs): $\Theta(n^2)$ time
- divide-and-conquer based algorithm: $\mathrm{O}(n \log n)$ time



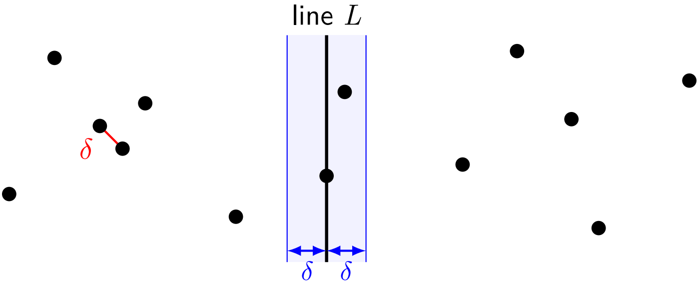
Divide-and-Conquer

Algorithm Overview

1. Sort by x-coordinate and divide into two halves (left and right);
2. Recursively solve the problem;
3. Outputs the closest pair of left–left, right–right, left–right;

Obs.: the closest pair is left–right $\Rightarrow$ they lies within a distance δ of L

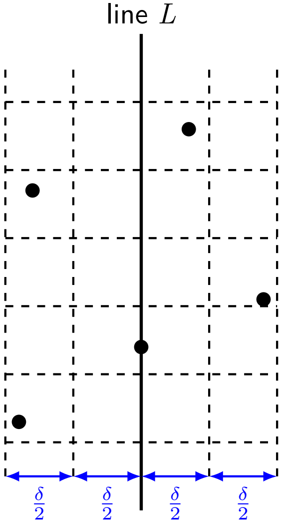
min of left–left and right–right



Check left–right points pair

- partition the strip into boxes of $\delta/2$ per side
- each box can contain at most one point
- sort the points in the strip by y-coordinate
  $\mathrm{O}(n)$ time by sorting whole points in advance
- for each point, it is sufficient to check
  its distance to each of the next 15 points

→ $\mathrm{O}(n)$ time



Running time

- P_x: list of points P sorted by x-coordinate
- P_y: list of points P sorted by y-coordinate

`ClosestPair`(P_x, P_y)

if $|P| \leq 3$ **then** return a closest pair by naive algorithm;
Divide into two halves and construct Q_x, Q_y, R_x, R_y;
$\delta \leftarrow \min\{d(\texttt{ClosestPair}(Q_x, Q_y)), d(\texttt{ClosestPair}(R_x, R_y))\}$;
Extract points in the stripe and construct S_y;
Find the closest pair of P by checking the strip;

The total computational time is $T(n) = 2\,T(n/2) + \mathrm{O}(n)$

Theorem

The running time of the algorithm is $\mathrm{O}(n \log n)$