算法设计与分析
Algorithms Design & Analysis

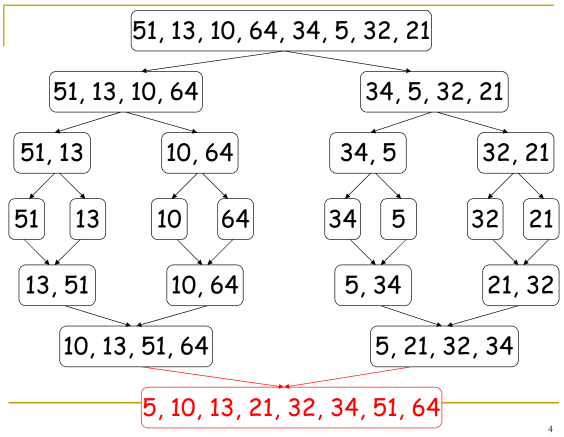
第五讲：分治法

- 孙 子 曰 ： 凡 治 众 如 治 寡， 分 数是 也。

  分 数 ： 分、 数 指 军 队 之 组 织、 编制。编 制 严 密， 人 多 少 均 同 样 指 挥。

(回顾：合并排序)Review：Merge sort

- divide the sequence of n numbers into two halves(分割)
- **recursively** sort the two halves(递归处理)
- **merge** the two sort halves into a single sorted sequence(合并)



分治法(Divide and Conquer)

- Our first design strategy: Divide and Conquer(算法设计策略)
- Often recursive, at least in definition(通常是递归形式的)
- Strategy(策略思想):
  - Break a problem into 1 or more smaller subproblems that are identical in nature to the original problem(将一个问题分割成几个规模小、性质相同的独立的子问题)
  - Solve these subproblems (recursively)(通常通过递归方法解决子问题)
  - Combine the results for the subproblems (somehow) to produce a solution to original problem(合并每个子问题的解得到整个问题的解)
- Note the assumption(前提假设):
  - We can solve original problem given subproblems' solutions(原始问题的解能够通过子问题的求解获得)

分治法(Divide and Conquer)

- It is often easier to solve several small instances of a problem than one large one.(子问题要能够更容易的求解)
  - **divide** the problem into smaller instances of the same problem(分割)
  - solve (**conquer**) the smaller instances recursively(递归求解)
  - **combine** the solutions to obtain the solution for original input(合并)
  - Must be able to solve one or more small inputs **directly(子问题的解可以直接获得)**

分治法的算法描述

- Solve(I)

```
n = size(I)
if (n <= smallsize)
   solution = directlySolve(I);
else
   divide I into I1, …, Ik.
   for each i in {1, …, k}
      Si = solve(Ii);
   solution = combine(S1, …, Sk);
return solution;
```

7

为什么要采用分治法?(Why Divide and Conquer?)

- Sometimes it's the simplest approach(简单)
- Divide and Conquer is often more efficient than “obvious” approaches(通常更有效)
  - E.g. Mergesort, Quicksort
- But, not necessarily efficient(但不是一定)
  - Might be the same or worse than another approach(与其它方法的效率可能相同\甚至更坏)
- Must analyze cost(必须分析分治算法的代价)
- Note: divide and conquer may or may not be implemented recursively(也不肯定是递归的)

8

分治法开销(Cost for a Divide and Conquer Algorithm)

- Perhaps there is…(分治法开销有)
  - A cost for dividing into sub problems(分割开销)
  - A cost for solving each of several subproblems(子问题解决开销)
  - A cost to combine results(合并开销)
- So (for *n > smallSize*)(开销函数的形式)
  $T(n) = D(n) + Sum[\ T(size(I_i))\ ] + C(n)$
  - often rewritten as(更一般化)
    $T(n) = a\,T(n/b) + f(n)$
- These formulas are *recurrence relations(递归关系)*

9

二分搜索(Binary Search)

- Input: a sequence of n **sorted** numbers $a_0, a_1, \ldots, a_{n-1}$; and a number X(输入:排序好的序列$a_0, a_1, \ldots, a_{n-1}$和数值X)
- Idea of algorithm:
  - compare X with number in the **middle(与中值比较)**
  - then focus on only the first **half** or the second half (depend on whether X is smaller or greater than the middle number)(左、右比较)
  - reduce the amount of numbers to be searched by half(规模逐半减小)

10

Binary Search (2)

we first work on n numbers, from a[0]..a[n-1]

3 7 11 12 15 19 24 33 41 55

24

then we work on n/2 numbers, from a[n/2]..a[n-1]

19 24 33 41 55

24

further reduce by half

19 24

24

11

二分搜索的时间复杂性(Time complexity)

- Let T(n) denote the time complexity of binary search algorithm on n numbers.

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

12

矩阵乘法(Matrix Multiplication)

- Given $n \times n$ matrices X and Y, wish to compute the product $Z=XY$.
- Formula for doing this is(公式)

$$Z_{ij} = \sum_{k=0}^{n-1} X_{ik} Y_{kj}$$

- This runs in $O(n^3)$ time(时间开销函数)
  - In fact, multiplying an $n \times m$ by an $m \times q$ takes nmq operations(对于非阵情况)

矩阵乘法(Matrix Multiplication)

$$\begin{bmatrix} I & J \\ K & L \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$I = AE + BG$$
$$J = AF + BH$$
$$K = CE + DG$$
$$L = CF + DH$$

矩阵乘法(Matrix Multiplication)

- Using the decomposition on previous slide, we can computer Z using 8 recursively computed (n/2)x(n/2) matrices plus 4 additions that can be done in $O(n^2)$ time
- (分割成8个(n/2)x(n/2)相乘和另外4个矩阵加)
- Thus T(n) = 8T(n/2) + bn^2(开销函数)
- Still gives T(n) is $\Theta(n^3)$

矩阵乘法Strassen's 算法

- If we define the matrices S_1 through S_7 as follows(如下定义S_1 - S_7)

$$S_1 = A(F - H)$$
$$S_2 = (A + B)H$$
$$S_3 = (C + D)E$$
$$S_4 = D(G - E)$$
$$S_5 = (A + D)(E + H)$$
$$S_6 = (B - D)(G + H)$$
$$S_7 = (A - C)(E + F)$$

矩阵乘法Strassen's 算法

- Then we get the following(可以得到):

$$I = S_5 + S_6 + S_4 - S_2$$
$$J = S_1 + S_2$$
$$K = S_3 + S_4$$
$$L = S_1 - S_7 - S_3 + S_5$$

- So now we can compute $Z=XY$ using only 7 recursive multiplications(7个递归相乘)

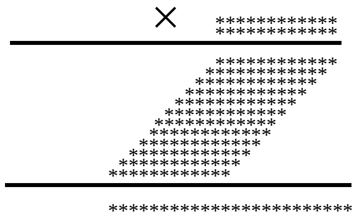
矩阵乘法Strassen's 算法

- This gives the relation T(n)=7T(n/2)+bn^2 for some b>0. (开销函数)
- By the Master Theorem, we can thus multiply two $n \times n$ matrices in $\Theta(n^{\log 7})$ time, which is approximately $\Theta(n^{2.808})$.(主方式方法求解)
  - May not seem like much, but if you're multiplying two 100×100 matrices:(实例比较)
    - n^3 is 1,000,000
    - $n^{2.808}$ is 413,048
- With added complexity, there are algorithms to multiply matrices in as little as $\Theta(n^{2.376})$ time(有更优秀的算法)
  - Reduces figures above to 56,494

整数相乘(Integer Multiplication)

- When do we need to multiply two very large numbers?(大整数相乘的情况)
  - In Cryptography and Network Security(密码学和网络安全)
  - encryption and decryption need to multiply numbers(加密和解密)

一般乘法(How to multiply 2 n-bit numbers)



$\Theta(n^2)$ bit operations

分治法(Can We do Better?)

- Divide and Conquer

$X =$	a	b
$Y =$	c	d

$$X = a2^{n/2} + b,\ Y = c2^{n/2} + d$$

$$XY = ac2^{n} + (ad + bc)2^{n/2} + bd$$

- MULT(X,Y)
  - **if** |X| = |Y| = 1 **then do return** XY
  - **else return**

$$\mathrm{MULT}(a,c)2^{n} + (\mathrm{MULT}(a,d) + \mathrm{MULT}(b,c))2^{n/2} + \mathrm{MULT}(b,d)$$

开销函数

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

- By the Master Theorem(主方式法):

$$T(n) = \Theta\left(n^2\right)$$

更好的方法(Karatsuba 1962)

- Gauss Equation

$$ad + bc = (a+b)(c+d) - ac - bd$$

- MULT(X,Y)
  - **if** |X| = |Y| = 1 **then do return** XY
  - **else**
    - A_1 = MULT(a,c); A_2 = MULT(b,d);
    - A_3 = MULT((a+b)(c+d));

$$A_1 2^{n} + (A_3 - A_1 - A_2)2^{n/2} + A_2$$

开销函数

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

- By the Master Theorem:

$$T(n) = \Theta\left(n^{\log_2 3}\right) = \Theta\left(n^{1.58}\right)$$

最近点对问题(Closest Pair Problems)

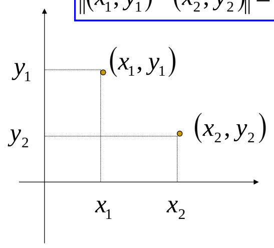
- Input:
  - A set of points $P = \{p_1, \ldots, p_n\}$ in two dimensions(二维空间上的点)
- Output:
  - The pair of points p_i, p_j that minimize the Euclidean distance between them.(距离最小的点对)

25

Distances

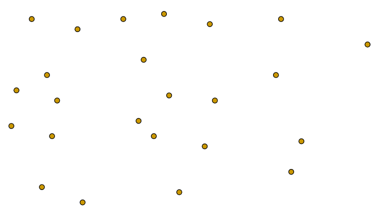
- Euclidean distance

$$\boxed{\left\|(x_1, y_1) - (x_2, y_2)\right\| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$



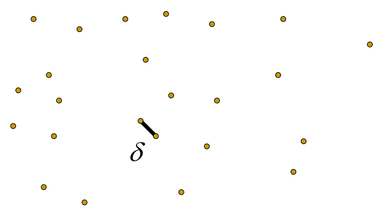
26

最近点对问题(Closest Pair Problems)



27

最近点对问题(Closest Pair Problems)



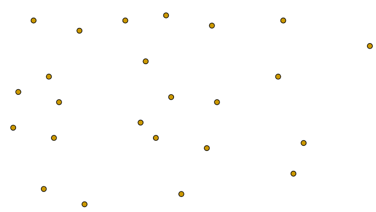
28

分治解决办法

- O(n²) time algorithm is easy(蛮力法:计算出两两之间的距离然后比较的方法)
- Assumptions:
  - No two points have the same x-coordinates (x坐标互异)
  - No two points have the same y-coordinates (y坐标互异)
- How do we solve this problem in 1 dimension? (如何在1维空间上求解?)
  - Sort the number and walk from left to right to find minimum gap (排列,从左到右求最小距离)

29

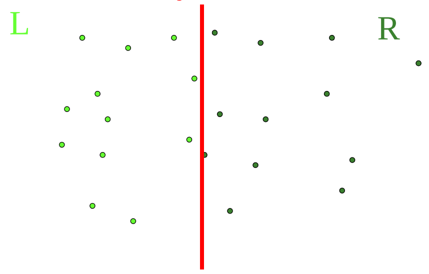
分治解决办法



30

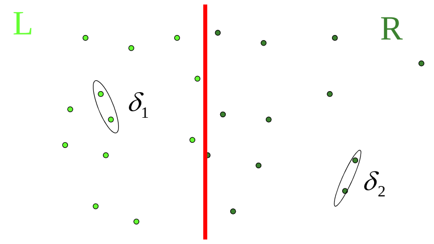
分割(Divide)

Divide by x-median



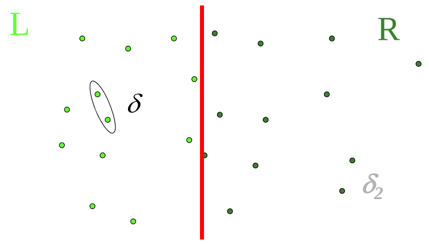
递归解决(Conquer)

Conquer: Recursively solve L and R



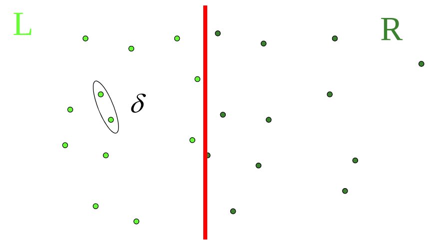
整合(Combination I)

Takes the smaller one of δ_1, δ_2: $\delta = \min(\delta_1, \delta_2)$



整合(Combination II)

Is there a point in L and a point in R whose distance is smaller than δ?(L和R中是否各存在一点,它们之间的距离小于δ?)



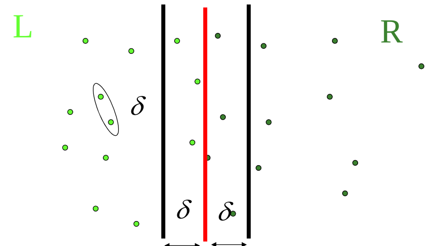
Takes the smaller one of δ_1, δ_2: $\delta = \min(\delta_1, \delta_2)$

整合(Combination II)

- If the answer is “no” then we are done!!!(如果没有,结束)
- If the answer is “yes” then the closest such pair forms the closest pair for the entire set(如果有,此类点对中最距离最近的点即是整个问题的解)
- How do we determine this?(如何确定)

整合(Combination II)

Is there a point in L and a point in R whose distance is smaller than δ?

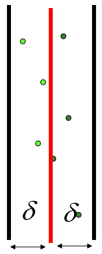


Takes the smaller one of δ_1, δ_2: $\delta = \min(\delta_1, \delta_2)$

Combination II

Is there a point in L and a point in R whose distance is smaller than δ?

L　　R



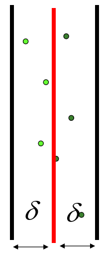
Need only to consider the narrow band

O(n) time(只需考虑此带条内的点的距离)

Combination II

Is there a point in L and a point in R whose distance is smaller than δ?

L　　R



观察结论:S中的每个点最多需要和S中的7个点进行比较

Denote this set by S(用S表示该带条)

δ_L: minimum distance of P_L

δ_R: minimum distance of P_R

δ=min{δ_L, δ_R}.

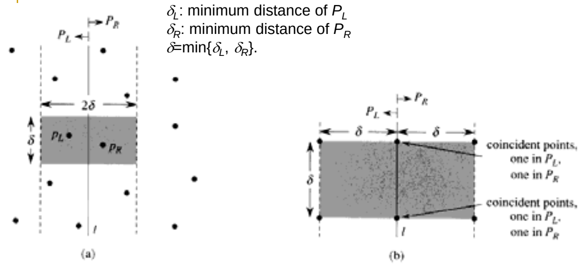


Figure 33.11 Key concepts in the proof that the closest-pair algorithm needs to check only 7 points following each point in the array Y'. **(a)** If $p_L \in P_L$ and $p_R \in P_R$ are less than δ units apart, they must reside within a $\delta \times 2\delta$ rectangle centered at line l. **(b)** How 4 points that are pairwise at least δ units apart can all reside within a $\delta \times \delta$ square. On the left are 4 points in P_L, and on the right are 4 points in P_R. There can be 8 points in the $\delta \times 2\delta$ rectangle if the points shown on line l are actually pairs of coincident points with one point in P_L and one in P_R.

Closest-Pair

- **Closest-pair(P)**
  - **Preprocessing: (预处理,递增排列)**
    - **Construct P_x and P_y as sorted-list by x- and y-coordinates**
  - **Divide (分割)**
    - **Construct L, L_x , L_y and R, R_x , R_y**
  - **Conquer (治理)**
    - **Let δ_1= Closest-Pair(L, L_x , L_y)**
    - **Let δ_2= Closest-Pair(R, R_x , R_y)**
  - **Combination (整合)**
    - **Let δ = min(δ_1 , δ_2)**
    - **Construct S and S_y (按y坐标排序)**
    - **For each point in S_y, check each of its next 7points down the list（检查与后续7个点之间的距离）**
    - **If the distance is less than δ, update the δ as this smaller distance（如果存在小于δ的距离，则为距离最短点对）**

Complexity Analysis

- Preprocessing takes O(n lg n) time
- Divide takes O(n) time
- Conquer takes 2 T(n/2) time
- Combination takes O(n) time

- **So totally takes O(n lg n) time**