Divide and Conquer

2021年9月1日 22:06

1.Merge Sort

- 问题描述

  将一个长度为n的数组进行排序

- 算法思想

  归并排序的思想是将长度为n的数组分成长度基本一致的两段，不断进行下去，知道数组长度不超过某一个常数为止，对这些长度小于等于某一个常数的数组排序是容易解决的。最后将结果逐层合并。

- 算法分析

  假设数组长度为n时算法复杂度为T(n)，则处理分成的两段数组时算法时间复杂度为$T(\frac{n}{2})$，将处理好的这两段合并的时间复杂度为$\Theta(n)$，所以递推式为

$$T(n) \leq 2T\left(\frac{n}{2}\right) + \Theta(n)$$

  通过主定理或者递推树可知该算法时间复杂度为$\Theta(n\log n)$

2.Counting Inversions

- 算法思想

  计算一个序列中逆序对可以在归并排序的过程中进行，具体来说实在归并排序算法中合并的那一步进行的。

- 算法分析

  该算法复杂度与归并排序一致，为$\Theta(n\log n)$

3.Finding the Closest Pair of Point

- 问题描述

  给定n个点在坐标系中的坐标，求出n个点中距离最近的两个点

- 问题分析

  解决这个问题最朴素的想法是计算n各点两两之间的距离，然后找出最小值，这样操作算法的时间复杂度$T(n) = \Theta(n^2)$。那么我们可不可以对这个算法进行改进呢？我们可以考虑一种最简单的情形——即这n各点在一条直线，也就是先讨论一维的情形。

  对于一维情形，我们不妨假设这n个点都在x轴上，将这n个点的坐标进行排序最快要$\Theta(n\log n)$，接下来将n各点从左向右计算每相邻两个点的距离来求出最小值，这一步算法的时间复杂度为$\Theta(n)$。所以对于一维情形算法的时间复杂度为$\Theta(n\log n) + \Theta(n) = \Theta(n\log n)$。我们思考对于二维的情形是否也能够用类似的方法处理，即将平面上n个点按照横坐标（纵坐标）排序然后找出横坐标（纵坐标）距离最近的两个点。但我们很快发现这种算法是错误的，因为很容易找出反例。对于二维情形，我们还是考虑分治与递归的方法。

  首先假设这n各点横纵坐标都不相同（若存在相同则一定可以将坐标系旋转某个角度使之符合我们的要求），我们假设经过处理后的点集为$P = \{p_1, p_2, \ldots, p_n\}$，每个点$p_i$的坐标为$(x_i, y_i)$，对任意属于P的两个点$p_i, p_j$，它们之间的欧式距离为$d(p_i, p_j)$。现在将集合P中的点分别按x坐标和y坐标升序排列，得到序列P_x和P_y。接下来定义Q是在序列P_x中前$\lceil \frac{n}{2} \rceil$个点组成的点集，定义R是在序列$P_x$中后$\lfloor \frac{n}{2} \rfloor$个点组成的点集。为了直观表示，在集合P中用一条线l将Q和R分开，如下图所示。

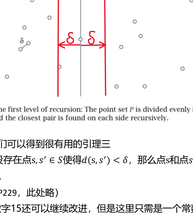

Figure 5.6 The first level of recursion: The point set P is divided evenly into Q and R by the line L, and the closest pair is found on each side recursively.

在$O(n)$的时间复杂度内我们可以创造如下4个序列：将Q中的点按x坐标升序排成的点集Q_x，将Q中的点按y坐标升序排成的点集Q_y；将R中的点按x坐标升序排成的点集R_x，将R中的点按y坐标升序排成的点集R_y。现在我们假设递归后Q中距离最近的两个点是q_0*和q_1*，R中距离最近的两个点是r_0*和r_1*。假设$\delta = \min\{d(q_0^*, q_1^*), d(r_0^*, r_1^*)\}$，现在的问题是：是否存在点q ∈ Q和点r ∈ R使得$d(q,r) < \delta$？

如果不存在这样的q和r，那么距离最近的两个点已经找到，否则距离最近的点对就是(q, r)

我们容易发现若存在点$q \in Q$和点$r \in R$使得$d(q,r) < \delta$那么点q和点r分布在距离l.很近的范围内，因此有如下引理一

引理一 若存在点$q \in Q$和点$r \in R$使得$d(q,r) < \delta$，则点q和点r距离l的距离都不超过δ

（证明见书P228，此处略）

由引理一，很容易得到如下引理二

引理二 存在点$q \in Q$和点$r \in R$使得$d(q,r) < \delta$当且仅当存在点$s, s' \in S$使得$d(s,s') < \delta$

引理一可以让我们的搜索限制在距离为2δ的窄条形区域（如下图所示），假设集合S是P中落在这个窄条形区域中的点组成的集合，通过遍历P_y，可以用$\Theta(n)$的时间复杂度构建将集合S按y坐标升序排列的序列S_y。

Figure 5.6 The first level of recursion: The point set P is divided evenly into Q and R by the line L, and the closest pair is found on each side recursively.

进一步，我们可以得到很有用的引理三

引理三 假设存在点$s, s' \in S$使得$d(s,s') < \delta$，那么点s和点s'都在S_y中彼此相邻的15个位置之中的一个。

（证明见书P229，此处略）

其实数字15还可以继续改进，但是这里只需是一个常数就可以了。接下来进行如下操作：依次遍历S_y，对每个S_y中的点，计算和在序列S_y中紧接着这个点的后面15个点到这个点的距离。这样遍历下来，得到的最小距离为δ'，如果$\delta' < \delta$，那么最小距离就是δ'，否则就是递归得到的δ。至此，完整的分治与递归过程就完成了。

- 算法思想

  将上面的问题分析过程进行可以得到如下算法步骤：

  （1）分治：用分割线l将P_x分成数量大致相等的左右两部分，这两部分在进行分治到求得距离最近的点对

  （2）递归：也就是将分治的两部分进行合并的过程。容易知道距离最小点对的分布只可能是如下两种情形：（i）最小距离点对分布在l的一侧（ii)最小距离点对分布在l的两侧。现在分布在l一侧的最小距离已经通过递归得到，现在的主要问题主要是分布在l.两侧的点的最小距离。为此，通过引理三知道，我们只需遍历S_y中的点并求每个点到之后相邻的15个点的距离以得到在l.两侧距离最短的点对和最小距离，并将这个距离和递归得到的距离进行比较，以得到最终的最小距离。

- 伪代码

  该算法的伪代码如下：

```
Closest-Pair(P)
  Construct P_x and P_y  (O(n log n) time)
  (q_0^*, p_1^*) = Closest-Pair-Rec(P_x, P_y)

Closest-Pair-Rec(P_x, P_y)
  If |P| ≤ 3 then
    find closest pair by measuring all pairwise distances
  Endif

  Construct Q_x, Q_y, R_x, R_y (O(n) time)
  (q_0^*,q_1^*) = Closest-Pair-Rec(Q_x, Q_y)
  (r_0^*,r_1^*) = Closest-Pair-Rec(R_x, R_y)

  δ = min(d(q_0^*,q_1^*), d(r_0^*,r_1^*))
  x* = maximum x-coordinate of a point in set Q
  L = {(x,y) : x = x*}
  S = points in P within distance δ of L.

  Construct S_y (O(n) time)
  For each point s ∈ S_y, compute distance from s
     to each of next 15 points in S_y
     Let s, s' be pair achieving minimum of these distances
     (O(n) time)

  If d(s,s') < δ then
     Return (s,s')
  Else if d(q_0^*,q_1^*) < d(r_0^*,r_1^*) then
     Return (q_0^*,q_1^*)
  Else
     Return (r_0^*,r_1^*)
  Endif
```

- 算法分析

  假设计算n个点中最短距离的时间复杂度为T(n)，则计算左右两部分的点集中的点的最短距离的时间复杂度为$2T(\frac{n}{2})$，从集合S_y中找出最短距离的过程中每一步都要计算15次距离，计算次数为常数，由于S_y中点的数量小于等于n，因此这一步的时间复杂度小于等于$\Theta(n)$，因此有$T(n) \leq 2T\left(\frac{n}{2}\right) + \Theta(n)$，因此$T(n) \leq \Theta(n\log n)$，相比最简单粗暴的计算两两之间的距离的算法快了不少。

4.Integer Multiplication

- 问题描述

  计算两个n位数字x和y相乘的结果

- 问题分析

  关于这个问题的一种比较简单的方法就是竖式相乘，两个n位数字相乘，列出竖式的时间复杂度为$\Theta(n \times n) = \Theta(n^2)$，将竖式中的结果相加的时间时间复杂度为$\Theta(n)$，因此总的时间复杂度为$\Theta(n^2)$。下面我们考虑用分治与递归来解决这个问题。我们不妨假设这两个n位数是二进制的，将x写成$x_1 \cdot 2^{n/2} + x_0$的形式（也就是说x_1对应高n/2位，x_0对应低n/2位）类似的，将y写成$y_1 \cdot 2^{n/2} + y_0$的形式。从而，有$xy = \left(x_1 \cdot 2^{n/2} + x_0\right)\left(y_1 \cdot 2^{n/2} + y_0\right) = x_1y_1 \cdot 2^n + (x_1y_0 + x_0y_1) \cdot 2^{n/2} + x_0y_0$

  上式将n位二进制数相乘的情形转化为4个n/2位二进制数相乘的情形。从而分治与递归的时间复杂度可以按如下方法计算：假设n位二进制数相乘的情形时间复杂度为T(n)，则4个n/2位二进制数相乘的情形时间复杂度为$4T(n/2)$，将结果合并（相加）的时间复杂度为$\Theta(n)$，因此有$T(n) = 4T(n/2) + \Theta(n)$。从而由主定理可得$T(n) = \Theta\left(n^{\log_2 4}\right) = \Theta(n^2)$。从而结果并没有改进，所以我们尝试进一步改进，算法分析如下。

- 算法思想

  我们发现交叉项$x_1y_0 + x_0y_1 = (x_1 + x_0)(y_1 + y_0) - x_1y_1 - x_0y_0$，从而我们现在只需计算$x_1y_1$、$x_0y_0$和$(x_1 + x_0)(y_1 + y_0)$这三项，也就是进行3次n/2位二进制数相乘。

- 伪代码

  该算法的伪代码如下：

```
Recursive-Multiply(x,y):
  Write x = x_1 · 2^{n/2} + x_0
        y = y_1 · 2^{n/2} + y_0
  Compute x_1 + x_0 and y_1 + y_0
  p = Recursive-Multiply(x_1 + x_0, y_1 + y_0)
  x_1y_1 = Recursive-Multiply(x_1, y_1)
  x_0y_0 = Recursive-Multiply(x_0, y_0)
  Return x_1y_1 · 2^n + (p - x_1y_1 - x_0y_0) · 2^{n/2} + x_0y_0
```

- 算法分析

  由于只计算3次n/2位二进制数相乘的情形，从而有递推式$T(n) = 4T(n/2) + \Theta(n)$，根据主定理得到时间复杂度$T(n) = \Theta\left(n^{\log_2 3}\right) = \Theta(n^{1.59})$相比一开始有了一些改进。

5.Convolutions and the Fast Fourier Transform

见书P234~242，此处略