دانشکده‌ی علوم ریاضی

آنالیز الگوریتم‌ها — ۲۴ بهمن ۱۳۹۱

# جلسه‌ی ۳: نزدیک‌ترین زوج نقاط


*مدرّس: دکتر شهرام خزائی* — *نگارنده: امیر سیوانی اصل*


## ۱ پیدا کردن نزدیک‌ترین زوج نقطه

فرض می‌کنیم، $n$ نقطه داریم و می‌خواهیم در بین این نقاط نزدیک‌ترین زوج‌نقطه را بیابیم. منظور از فاصله‌ی نقاط $P = (x_1, y_1)$ و $Q = (x_2, y_2)$، که از رابطه $\mathrm{dist}(P, Q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ محاسبه می‌شود، همان فاصله اقلیدسی است.
اولّین و ساده‌ترین راه حل این است که همه‌ی زوج نقاط روی صفحه را در نظر بگیریم و کوچک‌ترین فاصله‌ی این زوج نقاط را با استفاده از جستجوی کامل[1] بیابیم. شبه کد این الگوریتم در زیر آمده است.

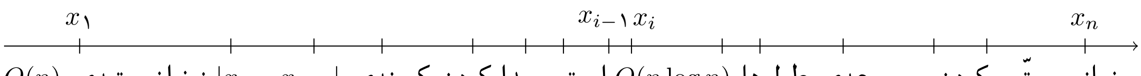
```
function CLOSEST-PAIR-BRUTE-FORCE(P_1, ..., P_n)
    minDist ← ∞
    for every pair (P_i, P_j) with 1 ≤ i < j ≤ n do
        d ← dist(P_i, P_j)
        if d < minDist then
            minDist ← d
            closestPair ← (P_i, P_j)
    return closestPair
```

این الگوریتم از مرتبه‌ی $\Theta(n^2)$ است.

### ۱.۱ حالت یک‌بعدی

#### ۱.۱.۱ روش عادی

در این حالت ورودی مجموعه‌ای از اعداد مانند $(x_1, \ldots, x_n)$ است که بیانگر مؤلّفه‌ی طول این $n$ نقطه در روی محور افقی هستند. این الگوریتم به این صورت کار می‌کند که ابتدا نقاط را بر حسب مؤلّفه‌ی طول‌شان مرتّب کرده و در آرایه‌ی مرتّب شده‌ی $(x_1, \ldots, x_n)$ قرار می‌دهد. سپس کم‌ترین $|x_i - x_{i-1}|$ را یافته و به عنوان جواب بازمی‌گرداند.




مرتبه‌ی زمانی مرتّب کردن مجموعه‌ی طول‌ها $O(n \log n)$ است. پیدا کردن کمینه‌ی $|x_i - x_{i-1}|$ نیز از مرتبه‌ی $O(n)$ زمان می‌برد زیرا نیازمند یک‌بار پیمایش آرایه هستیم. پس مرتبه‌ی زمانی کلّی الگوریتم $O(n \log n)$ خواهدبود.

[1] brute force

### ۲.۱.۱ تقسیم و حل

راه حل فوق در دل خود از رهیافت تقسیم و حل استفاده می‌کند (در قسمت مرتب‌سازی آرایه ورودی)، اما تعمیم روش برای حالت دو بعدی سرراست به نظر نمی‌رسد. ما به دنبال یک روشی مبتنی بر تقسیم و حل هستیم که بتوان آن‌را تعمیم داد.

فرض می‌کنیم که آرایه ورودی $(x_1, \ldots, x_n)$ مرتّب باشد. الگوریتم زیر را در نظر بگیرید.

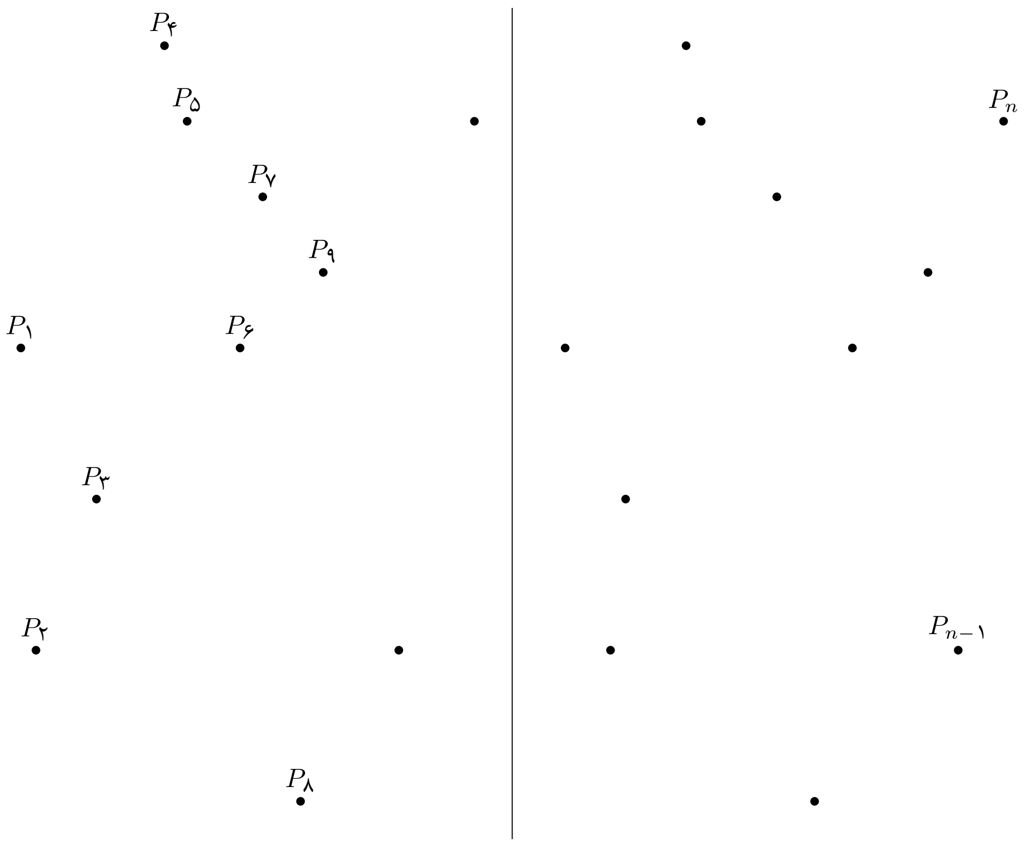
با استفاده از روش تقسیم و حل، $n$ نقطه ورودی را به دو دسته نقطه‌ی $L$ و $R$ که هر یک شامل $n/2$ نقطه است تقسیم می‌کنیم به طوری همه‌ی نقاط $L$ در سمت چپ میانه و همه‌ی نقاط $R$ در سمت راست آن باشند. دقّت کنید که چون آرایه ورودی مرتّب است، این کار را در $O(n)$ می‌توان انجام داد. حال برای هر دسته مسأله را به صورت بازگشتی حل می‌کنیم. از هردسته زوج نقطه‌ای به عنوان پاسخ خواهیم داشت. کمینه‌ی این دو فاصله لزوماً نزدیک‌ترین زوج نقطه را نمایش نمی‌دهد. زیرا ممکن است زوج نقطه‌ای در دو طرف میانه قرار داشته‌باشند که به هم نزدیک‌تر باشند. بنابراین باید سمت راست‌ترین نقطه‌ی دسته‌ی سمت چپ و سمت چپ‌ترین نقطه‌ی دسته‌ی سمت راست (یعنی نقاط دوطرف میانه) را نیز محاسبه کنیم. در نهایت کمینه‌ی نزدیک‌ترین زوج‌نقاط در سمت چپ، سمت راست و دوطرف میانه را به عنوان پاسخ بازمی‌گردانیم. شبه‌کد این الگوریتم را در زیر مشاهده می‌کنید.

**function** CLOSEST-PAIR-1D$(x_1, \ldots, x_n)$
    [*assumes input array is sorted and $n$ is a power of two*]
    **if** $n = 2$ **then**
        $minDist = |x_1 - x_2|$
    **else**
        $minDistL \leftarrow$ CLOSEST-PAIR-1D$(x_1, \ldots, x_{n/2})$ [*i.e., min dist of the left half points*]
        $minDistR \leftarrow$ CLOSEST-PAIR-1D$(x_{n/2+1}, \ldots, x_n)$ [*i.e., min dist of the right half points*]
        $minDistS \leftarrow x_{n/2} - x_{n/2+1}$ [*i.e., min dist of of the split points*]
    **return** $\min\{minDistL, minDistR, minDistS\}$

با فرض مرتبب بودن آرایه ورودی صحت الگوریتم فوق روشن است و زمان اجرای آن $O(n \log n)$ است. با توجه به اینکه آرایه اولیه را نیز در زمان $O(n \log n)$ می‌توان مرتب کرد، نزدیک‌ترین زوج نقاط را می‌توان در زمان $O(n \log n)$ پیدا نمود. اضافه کردن قابلیت پیدا کردن نزدیک‌ترین نقاط (علاوه بر فاصله آنها) به الگوریتم فوق نیز سرراست است.

## ۲.۱ حالت دوبعدی

در راستای تعمیم راه حل یک‌بعدی به دوبعدی، دقت کنید که تقسیم صفحه به چهار قسمت به گونه‌ای که هر قسمت شامل یک‌چهارم نقاط صفحه باشد میسر نیست. فرض می‌کنیم که نقاط داده شده بر حسب مؤلّفه‌ی $x$شان مرتّب هستند. این‌کار را در زمان $O(n \log n)$ می‌توان انجام داد. راه حل دیگری که به ذهن می‌رسد تقسیم نقاط به دو قسمت چپ و راست بر مبنای مؤلّفه‌ی طولشان و استفاده از رهیافت تقسیم و حل است.




**function** CLOSEST-PAIR-2D($P_1, \ldots, P_n$)
  [*assumes* $P_i = (x_i, y_i)$ *where* $(x_1, \ldots, x_n)$ *is sorted and* $n$ *is a power of two*]
  **if** $n = 2$ **then**
    $minDist = \mathrm{dist}(P_1, P_2)$
  **else**
    $minDistL \leftarrow$ CLOSEST-PAIR-2D($P_1, \ldots, P_{n/2}$) [*i.e., min dist of the left half points*]
    $minDistR \leftarrow$ CLOSEST-PAIR-2D($P_{n/2+1}, \ldots, P_n$) [*i.e., min dist of the right half points*]
    $minDistS \leftarrow$ CLOSEST-PAIR-SPLIT($P_1, \ldots, P_n$) [*i.e., min dist of of the split points*]
  **return** $\min\{minDistL, minDistR, minDistS\}$

## پیدا کردن نزدیک‌ترین نقاط جدا از هم

الگوریتم فوق بر روی $n$ نقطه ورودی که برحسب مؤلّفه‌ی طول‌شان مرتب هستند اعمال می شود. با استفاده از رهیافت تقسیم و حل ابتدا نقاط ورودی را به دو دسته نقطه‌ی چپ و راست که هر کدام شامل $\frac{n}{۲}$ نقطه است تقسیم می‌شود. سپس به صورت بازگشتی نزدیک‌ترین فاصله میان نقاط هر دو دسته محاسبه می‌شود. در نهایت لازم است که نزدیک‌ترین فاصله میان نقاط جدا از هم نیز محاسبه شود و جواب‌ها با هم ادغام گردند.
ما به دنبال یک الگوریتم سریع برای پیاده‌سازی پیدا کردن نزدیک‌ترین نقاط جدا از هم هستیم. ساده‌ترین راه‌حل، مقایسه‌ی همه‌ی نقاط موجود در یک دسته با نقاط دسته دیگر است، که دارای مرتبه‌ی زمانی آن $O(n^۲)$ می‌باشد. اما

با استفاده از الگوریتم غیر بدیهی زیر می‌توان این مسأله را در زمان $O(n \log n)$ حل کرد.

**function** CLOSEST-PAIR-SPLIT($P_1, \ldots, P_n$)

  [*assumes* $P_i = (x_i, y_i)$ *where* $(x_1, \ldots, x_n)$ *is sorted*]

  [*assumes minDistL and minDistR have already been computed*]

  $\delta = \min(minDistL, minDistR)$

  Let $\hat{x}$ be the median of $x$ coordinates

  Let $S$ contanins all thos those points $P_i = (x_i, y_i)$ with $|x_i - \hat{x}| \leq \delta$

  Sort points in $S$ by their $y$ coordinate and call them $P_1, \ldots, P_m$

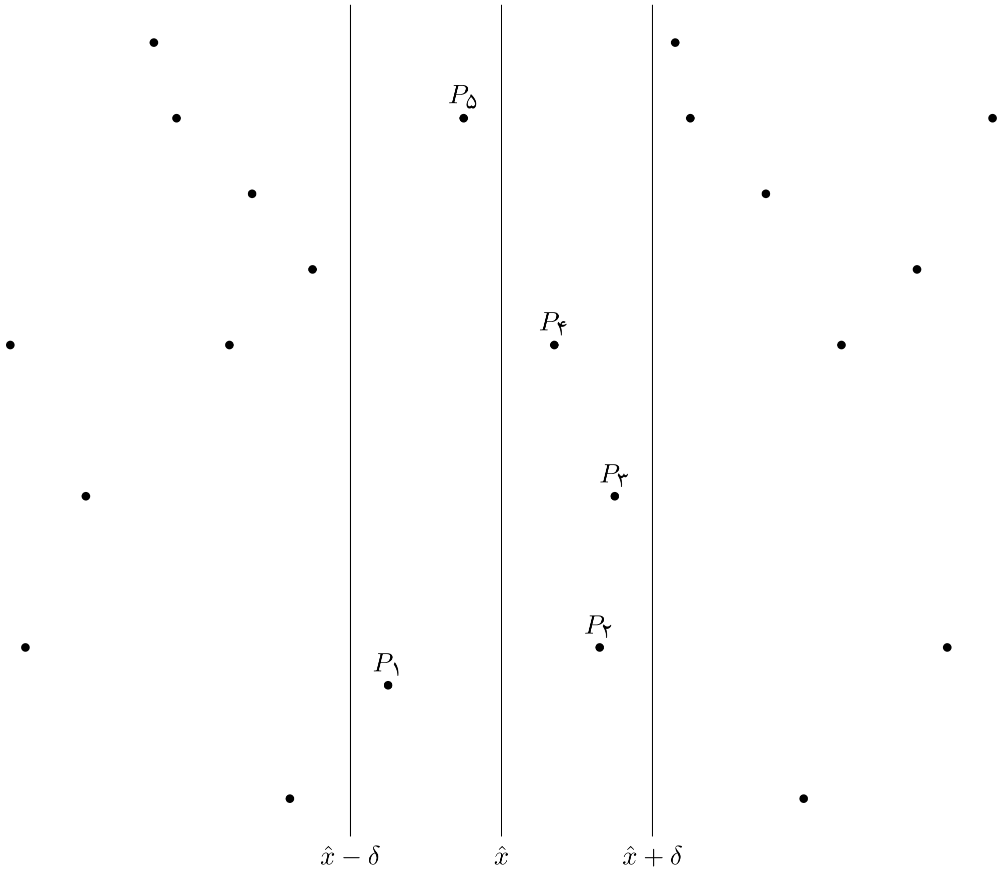
  $minDistS \leftarrow \delta$

  **for** everay pair $P_i, P_j \in S$ with $|i - j| \leq 7$ **do**

    **if** $\text{dist}(P_i, P_j) < minDistS$ **then**

      $minDistS = \text{dist}(P_i, P_j)$

  **return** $minDistS$




الگوریتم فوق ابتدا $\delta = min(\text{minDistL}, \text{minDistR})$ و $\hat{x}$، میانه‌ی مؤلّفه‌های‌ی $x$ نقاط، را محاسبه می‌کند و نقاطی را که طولشان در بطه‌ی $|x - \hat{x}| < \delta$ صدق می‌کنند در مجموعه‌ی $S$ قرار می‌دهد. سپس، مجموعه‌ی نقاط را بر حسب عرض‌شان مرتّب می‌کند و آن‌ها را به‌ترتیب $P_۱(x_۱, y_۱), ..., P_m(x_m, y_m)$ می‌نامد. آنگاه نزدیک زوج نقطه $(P_i, P_j)$ که $|i - j| \leq ۷$ محاسبه می‌شود. اگر فاصله آنها کمتر از $\delta$ باشد، $minDistS$ برابر این فاصله قرار داده‌می‌شود.

## محاسبه‌ی پیچیدگی الگوریتم

به فرض اینکه نقاط ورودی بر حسب مؤلّفه طولشان مرتب شده باشند، پیچیدگی الگوریتم از رابطه بازگشتی $T(n) = 2T(\frac{n}{2}) + O(n \log n)$ محاسبه می‌شود که جواب آن $T(n) = O(n \log^2 n)$ است. با توجه به اینکه مرتّب سازی اولیه نقاط بر حسب مؤلّفه طولشان از مرتبه‌ی زمانی $O(n \log n)$ است، پیچیدگی الگوریتم $O(n \log^2 n)$ است.
البتّه با پیش‌پردازش، به گونه‌ای که $x$ و $y$ ها مرتّب‌شده باشند، می‌توان یک الگوریتم با پیچیدگی $O(n \log n)$ ارائه کرد که به عنوان تمرین به خواننده واگذار می‌گردد.

## صحت الگوریتم

در ادامه $L$ مجموعه نقاط سمت چپ، $R$ مجموعه نقاط سمت راست و $\delta$ نزدیک‌ترین فاصله بین نقاطی که هر دو در یک سمت می‌باشند است.

**ادعا ۱** *اگر* $P = (x_1, y_1) \in L$ *و* $Q = (x_2, y_2) \in R$ *و* $\operatorname{dist}(P, Q) < \delta$ *آن‌گاه* $P, Q \in S$.

**برهان.**

$$P \notin S \Rightarrow x_1 \leq \overline{x} - \delta \tag{1}$$

$$Q \notin S \Rightarrow x_2 \geq \overline{x} + \delta \tag{2}$$

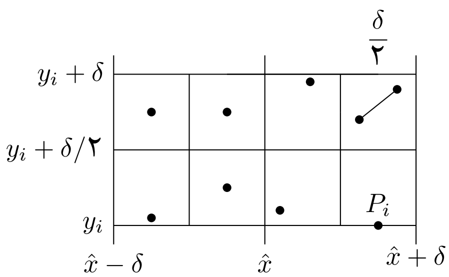
از ۱ و ۲ داریم:

$$\operatorname{dist}(P, Q) \geq |x_1 - x_2| \geq 2\delta \tag{3}$$

■

**ادعا ۲** *بعد از مرتّب کردن نقاط برحسب مؤلّفه عرضشان، اگر* $|i - j| > 7$ *آنگاه* $\operatorname{dist}(P_i, P_j) > \delta$.

**برهان.** بدون از دست دادن کلیت مسأله فرض کنید $i < j$ و $P_i = (x_i, y_i)$، $P_j = (x_j, y_j)$. فرض خلف کنید که $\operatorname{dist}(P_i, P_j) \leq \delta$ بنابراین داریم $y_j - y_i \leq \delta$. حال یک مستطیل با ابعاد $2\delta \times \delta$ به صورت زیر در نظر بگیرید. همه $j - i + 1$ نقطه $P_i, P_{i+1}, \cdots, P_j$ (که تعداد آنها حداقل ۹ تاست) در درون یا بر روی اضلاع این مستطیل جای دارند. مستطیل را به ۸ مربع با ابعاد $\delta/2 \times \delta/2$ همانگونه که در شکل نشان داده شده است تقسیم کنید. طبق اصل لانه کبوتری حداقل دو نقطه از این نقاط در درون یا روی اضلاع یکی از مربع‌ها قرار خواهد گرفت. اما در این صورت فاصله این دونقطه حداکثر برابر قطر مربع، یعنی $\delta/\sqrt{2}$، خواهد بود که خلاف این فرض است که هیچ دونقطه‌ای در یک سمت با فاصله کمتر از $\delta$ وجود ندارد.




■