۱ مسئله پیدا کردن نزدیکترین زوج نقطه

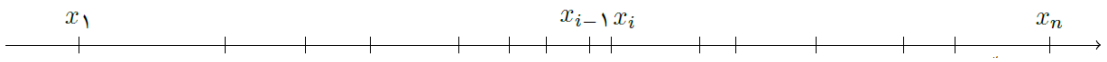
فرض میکنیم، n نقطه داریم و میخواهیم در بین این نقاط نزدیکترین زوج نقطه را بیابیم. منظور از فاصله ی نقاط $P = (x1, y2)$ و $Q = (x2, y2)$ که از رابطه $dist(P, Q) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$ محاسبه می‌شود، همان فاصله اقلیدسی است. ساده ترین راه حل این است که همه ی زوج نقاط روی صفحه را در نظر بگیریم و کوچکترین فاصله ی این زوج نقاط را با استفاده از جستجوی کامل بیابیم. نمونه کد این الگوریتم در زیر آمده است که مرتبه زمانی این الگوریتم $\Theta(n^2)$ 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
```

۲ حالت یک بعدی

۱.۲ روش عادی

در این حالت ورودی مجموعه‌ای از اعداد مانند $(x_1, \ldots, x_n)$ است که بیانگر مؤلفه‌ی طول این n نقطه در روی محور افقی هستند. این الگوریتم به این صورت کار می‌کند که ابتدا نقاط را بر حسب مؤلفه‌ی طولشان مرتب کرده و در آرایه‌ی مرتب شده ی $(x_1, \ldots, x_n)$ قرار می‌دهد. سپس کمترین $|x_i - x_{i-1}|$ را به عنوان جواب باز می‌گرداند. مرتبه‌ی زمانی مرتب کردن مجموعه‌ی طولها $O(n \log n)$ است. پیدا کردن کمترین $|x_i - x_{i-1}|$ هم از مرتبه‌ی $O(n)$ است زیرا باید یکبار آرایه را طی کنیم که در نتیجه مرتبه کلی از $O(n \log n)$ است.



۲.۲ تقسیم و غلبه

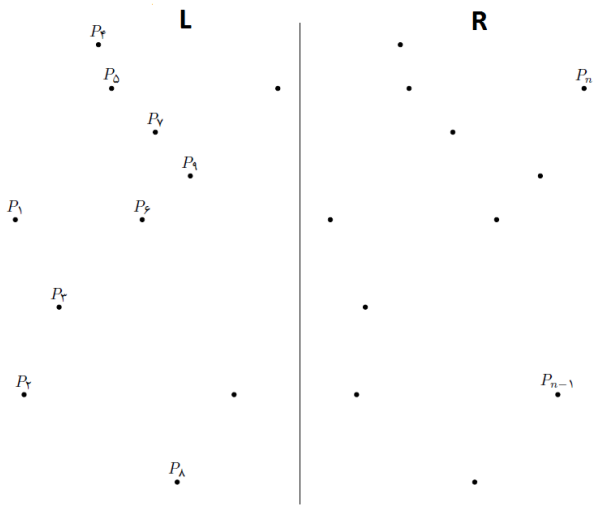
اگر چه حالت یک بعدی نیاز به طول و تفصیل ندارد و با روشی که گفته شد، نزدیکترین زوج نقاط با یک پیمایش آرایه مرتب قابل محاسبه است، اما بد نیست بدانیم که ایده اصلی روش تقسیم و غلبه در حالت یک بعدی نیز قابل مشاهده است.

فرض کنید که آرایه ورودی $(x_1, ..., x_n)$ مرتب است. الگوریتم زیر را در نظر بگیرید. با استفاده از روش تقسیم و غلبه، n نقطه ورودی را به دو دسته نقاط L و R که هرکدام شامل $\frac{n}{2}$ نقطه است تقسیم می‌کنیم به طوری که همه ی نقاط L در سمت چپ میانه و همه ی نقاط R در سمت راست آن باشند. چون آرایه ورودی مرتب است، این کار را در $O(n)$ می‌توان انجام داد. حال برای هر دسته مسأله را به صورت بازگشتی حل می‌کنیم. یعنی از هر دسته زوج نقطه ای به عنوان پاسخ خواهیم داشت. کمترین فاصله ی این دو لزوماً نزدیکترین زوج نقطه رانمایش نمی‌دهد. چون ممکن است زوج نقطه ای در دو طرف میانه قرار داشته باشند که به هم نزدیکتر باشند. پس باید سمت راست ترین نقطه ی دسته ی سمت چپ و سمت چپ ترین نقطه ی دسته ی سمت راست (یعنی نقاط دوطرف میانه) را نیز محاسبه کنیم. در پایان کمترین مقدار نزدیکترین زوج نقاط در سمت چپ، سمت راست و دوطرف میانه را به عنوان پاسخ بازمی‌گردانیم. نمونه کد 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 $\min\{minDistL, minDistR, minDistS\}$

۳ حالت دو بعدی

در راستای تعمیم راه حل یک بعدی به دو بعدی، تقسیم صفحه به چهار قسمت به طوری که هر قسمت شامل یک چهارم نقاط صفحه باشد میسر نیست. فرض میکنیم نقاط داده شده بر حسب مؤلفه ی x شان مرتب هستند. اینکار را در زمان $O(n \log n)$ می‌توان انجام داد. ایده دیگری که میتوان بررسی کرد استفاده از ایده تقسیم و غلبه است به طوری که نقاط را به دو قسمت چپ L و راست R بر مبنای مؤلفه ی x تقسیم می‌کنیم.



الگوریتم زیر را در نظر بگیرید که بر روی n نقطه ورودی که برحسب مؤلفه ی x مرتب هستند اعمال می‌شود.

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**
    $minDist = \text{dist}(P_1, P_2)$
  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 $\min\{minDistL, minDistR, minDistS\}$

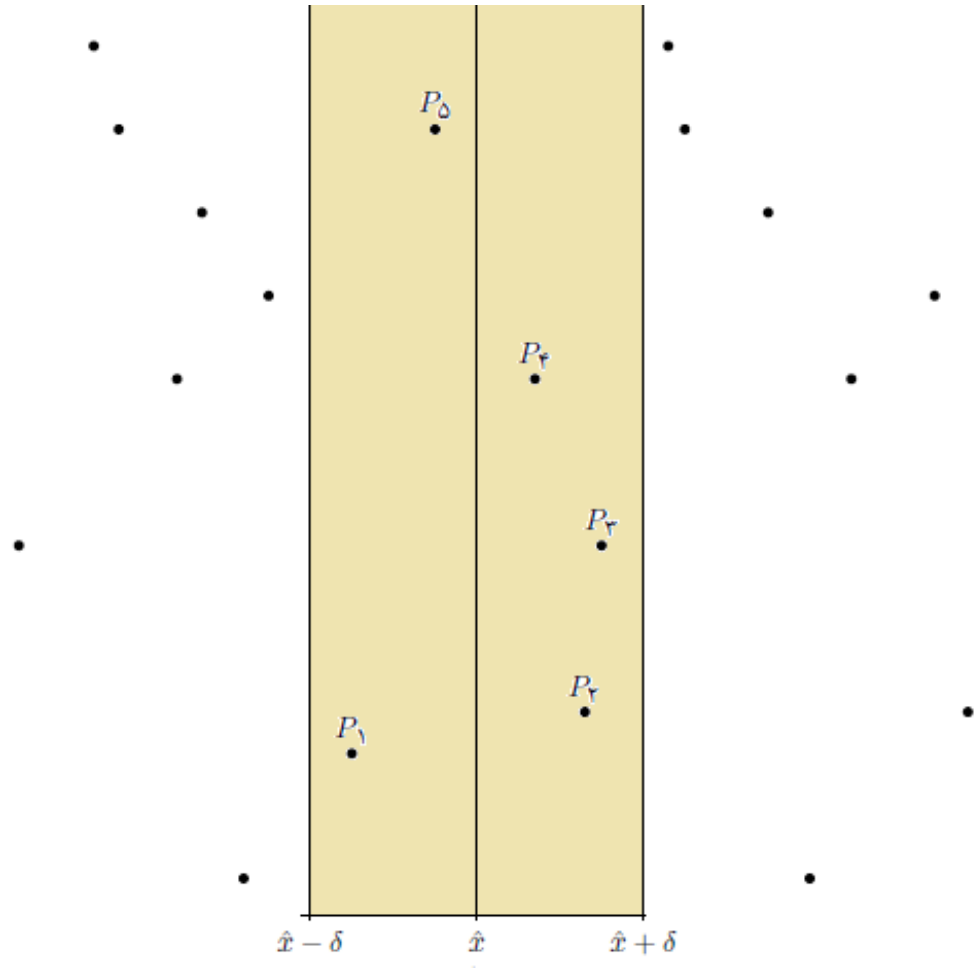
با استفاده از ایده تقسیم و غلبه ابتدا نقاط ورودی را به دو دسته نقطه ی چپ و راست که هر کدام شامل $\frac{n}{2}$ نقطه است تقسیم می‌شود. سپس به صورت بازگشتی نزدیکترین فاصله میان نقاط هر دو دسته محاسبه می‌شود. در نهایت لازم است که نزدیک‌ترین فاصله میان نقاط L و R نیز محاسبه شود و جواب نهایی از مقایسه این سه نتیجه بدست آید.

۱.۳ پیدا کردن نزدیکترین فاصله بین L و R

ما به دنبال یک الگوریتم سریع برای پیاده‌سازی پیدا کردن نزدیکترین نقاط که یکی شان در L و دیگری در R است هستیم. ساده‌ترین راه حل، مقایسه ی همه ی نقاط موجود در یک دسته با نقاط دسته دیگر است، که

دارای مرتبه ی زمانی $O(n^2)$ می‌باشد. اما با استفاده از الگوریتم غیر بدیهی زیر می‌توان این مسأله را در زمان $O(n \log n)$ حل کرد.

الگوریتم ابتدا $\delta = \min(\text{minDistL}, \text{minDistR})$ و $\hat{x}$، میانه ی مؤلفه x نقاط را محاسبه می‌کند و نقاطی راکه مولفه x شان در $|x - \hat{x}| < \delta$ صدق می‌کنند در مجموعه ی S قرار می‌دهد. مانند شکل زیر. نوار تیره رنگ شامل نقاط داخل S است.



سپس، مجموعه ی نقاط S را بر حسب مولفه y شان مرتب می‌کند و آنها را به ترتیب $P_1, \ldots, P_m$ می‌نامد. آنگاه نزدیک‌ترین زوج نقطه (P_i, P_j) که $|i - j| \leq 7$ محاسبه می‌شود. اگر فاصله آنها کمتر از δ باشد، minDistS برابر این فاصله قرار داده می‌شود. الگوریتم مربوطه در شکل زیر نشان داده شده است.

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 $minDistS$

۲.۳ پیچیدگی الگوریتم

به فرض اینکه نقاط ورودی بر حسب مؤلفه x شان 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 مرتب سازی اولیه نقاط بر حسب مؤلفه y از مرتبه ی زمانی $O(n \log n)$ است، پیچیدگی الگوریتم $O(n \log^2 n)$ است.

خوشبختانه یک ایده برای تسریع بیشتر الگوریتم وجود دارد. دقت کنید که در ابتدای کار نقاط را بر اساس مولفه y مرتب می‌کنیم. برای محاسبه مجموعه S و مرتب کردنش (بر اساس مولفه y)، حال کافی است دو لیست مرتب را در هم ادغام کنیم. این از مرتب سازی از نو نقاط داخل S بهتر است. لذا رابطه بازگشتی بصورت $T(n) = 2T(\frac{n}{2}) + O(n)$ در می‌آید و زمان الگوریتم در نهایت $O(n \log n)$ خواهد شد.

۳.۳ صحت الگوریتم

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

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

اثبات: (آ) $P \notin S \Rightarrow x_1 \leq x - \delta$ (ب) $Q \notin S \Rightarrow x_2 \geq x + \delta$

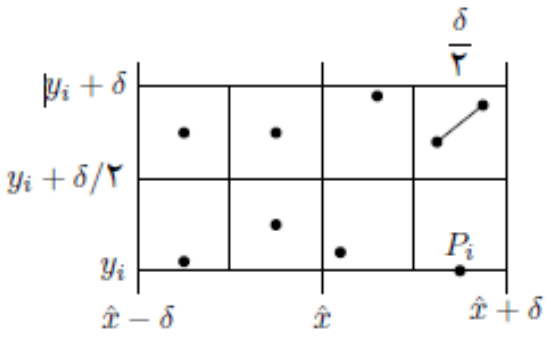
با توجه به (آ) و (ب) داریم:

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

حال ادعا می‌کنیم بعد از مرتب کردن نقاط براساس مؤلفه عرض، اگر $|i - j| > 7$ آنگاه باید داشته باشیم $\text{dist}(P_i, P_j) > \delta$.

اثبات: فرض کنید $i > j$ و $(P_i = (x_i, y_i), P_j = (x_j, y_j))$. به برهان خلف فرض کنید که $\text{dist}(P_i, P_j) \leq \delta$پس داریم $y_j - y_i \leq \delta$. حال یک مستطیل با ابعاد $2\delta \times \delta$ به صورت شکل ۲ در

نظر بگیرید. همه $1 + j - i$ نقطه $P_i, P_{i+1}, \ldots, P_j$ که تعداد آنها حداقل ۹ تاست درون یا بر روی اضلاع این مستطیل هستند. مستطیل را به ۸ مربع با ابعاد $\frac{\delta}{2} \times \frac{\delta}{2}$ به طوری که در شکل نشان داده شده است تقسیم میکنیم. 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 دو نقطه حداکثر برابر قطر مربع یعنی $\frac{\delta}{\sqrt{2}}$، خواهد بود که تناقض دارد با این که هیچ دو نقطه‌ای در یک سمت با فاصله کمتر از δ وجود ندارد.



شکل ۱: مستطیل با ابعاد $2\delta \times \delta$