

PAA - Aula 05

Encontrar o Par Mais Próximo

Definição do problema de encontrar o par mais próximo.

Entrada: um conjunto $P = \{p_1, p_2, \dots, p_n\}$ de n pontos no plano \mathbb{R}^2

- sendo \mathbb{R} o conjunto dos números reais.

Distância Euclidiana: a distância entre dois pontos $p_i = (x_i, y_i)$ e $p_j = (x_j, y_j)$ é

$$d(p_i, p_j) = \left((x_i - x_j)^2 + (y_i - y_j)^2 \right)^{1/2}$$

Solução: um par (p, q) de pontos em P que

- minimiza a distância $d(p, q)$ sobre todos os pares de pontos em P



Vale notar que, um algoritmo por busca exaustiva,

- composto por dois laços aninhados, pode comparar
 - a distância entre todos os $\binom{n}{2} = C_2^n = \frac{n \cdot (n-1)}{2}$ pares de pontos.
- Como cada comparação de distâncias leva tempo constante,
 - esse algoritmo encontra o par mais próximo em tempo $O(n^2)$

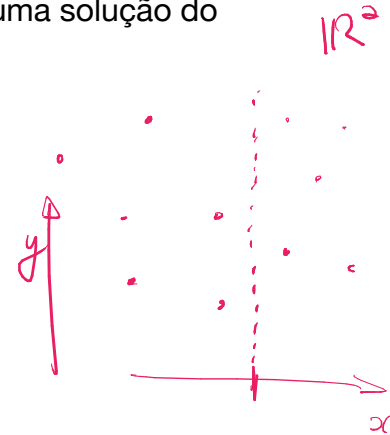
Será que conseguimos fazer melhor?

Vamos tentar desenvolver um algoritmo mais eficiente usando a abordagem de projeto por divisão e conquista.

- Dividir: o problema original é dividido em subproblemas do mesmo tipo.
- Conquistar: os subproblemas são resolvidos recursivamente, sendo que os subproblemas pequenos são caso base.
- Combinar: as soluções dos subproblemas são combinadas numa solução do problema original.

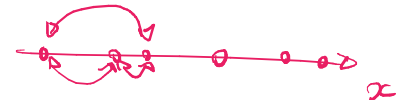
Para colocar alguma estrutura na instância do problema

- criamos duas cópias de P , chamadas P_x e P_y
- P_x é ordenada pelas coordenadas x dos pontos
- e P_y é ordenada pelas coordenadas y
- Note que esse procedimento leva tempo $O(n \lg n)$



Para intuir porque ordenar os pontos é uma boa ideia,

- observem que a versão deste problema na linha (dimensão \mathbb{R})
 - ainda tem $\binom{n}{2} = O(n^2)$ pares para comparar,
- mas pode ser resolvida em tempo linear,
 - uma vez que os pares estejam ordenados.
- Quiz1: por que?



Obs: ordenar elementos da entrada é uma boa estratégia em diversos problemas.

Algoritmo recursivo simples

(p, q) par Mais Próximo (P_x, P_y) :

se o # de pontos for menor que 4: // caso base
devolve o par mais próximo diretamente

divida P_x em L_x e R_x (e P_y em L_y e R_y)
de modo que a metade dos pontos mais à esquerda
fique nos conjuntos L , e os mais à direita nos R .

Obs: os vetores/conjuntos continuam ordenados

Ques 2: essa divisão deve ser feita em tempo linear. (Cons?)

$(p_l, q_l) = \text{Par Mais Próximo}(L_x, L_y)$

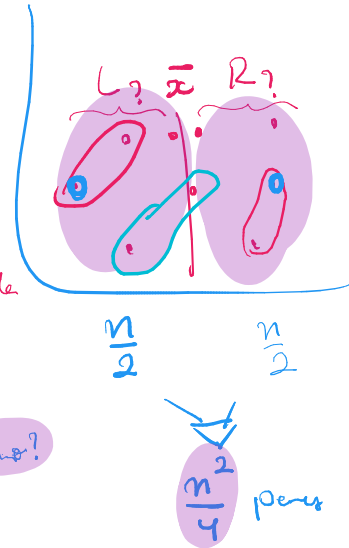
$(p_r, q_r) = \text{Par Mais Próximo}(R_x, R_y)$

$\Rightarrow (p_d, q_d) = \text{Par Mais Próximo Dividido}(P_x, P_y) \Rightarrow$ ainda temos $O(n^2)$ pares divididos

devolve o melhor entre (p_l, q_l) , (p_r, q_r) , (p_d, q_d)

Sendo (p_d, q_d) o par mais próximo dividido, um detalhe crucial é que

- só precisamos saber o par dividido se ele for menor que o não dividido.
 - Essa observação permite fortalecer o algoritmo.

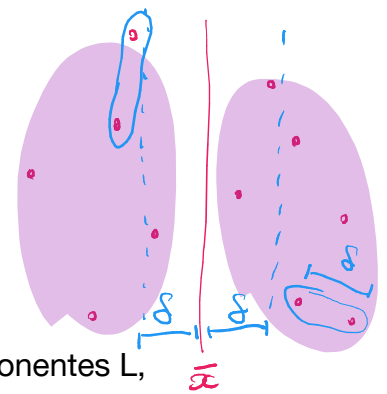


Algoritmo recursivo melhorado

(p, q) parMaisProximo(Px, Py):

$O(1)$ — (se o número de pontos for menor que 4: # caso base
encontre o par mais próximo diretamente e devolva-o

$O(n)$ — (→ divida Px em Lx e Rx (e Py em Ly e Ry) de modo que
a metade dos pontos mais a esquerda fique nos componentes L,
a metade mais a direita fique nos componentes R,
e os subconjuntos continuem ordenados.



2 chamadas
recursivas

(pl, ql) = parMaisProximo(Lx, Ly) *subprob. esq.*
(pr, qr) = parMaisProximo(Rx, Ry) *subprob. dir.*

$O(1)$ — $\delta = \min \{ d(p_l, q_l), d(p_r, q_r) \}$
(pd, qd) = parMaisProximoDividido (Px, Py, δ) — $O(n)$

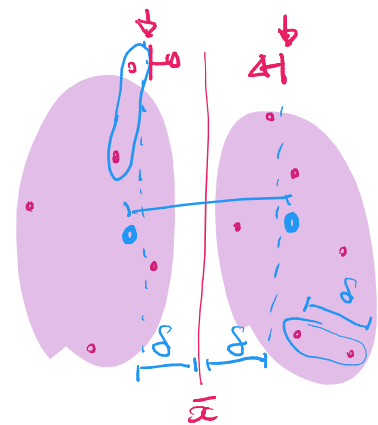
devolva o melhor entre (pl, ql), (pr, qr), (pd, qd)

Agora o parMaisProximoDividido pode usar o δ para melhorar sua busca.

Algoritmo que analisa os pares divididos

(pd, qd) parMaisProximoDividido(Px, Py, delta):

$O(n)$ $\left\{ \begin{array}{l} \bar{x} \text{ é uma coordenada } x \text{ que divide } P \text{ ao meio} \\ S_y \text{ é a restrição do vetor ordenado } P_y \\ \text{que só tem pontos com coordenada} \\ x \text{ entre } \bar{x} - \delta \text{ e } \bar{x} + \delta. \end{array} \right.$



Q3: por que isto é seguro?

$O(1)$ - $d_{min} = \delta$, $p = q = Null$

$O(n)$ $\left(\begin{array}{l} \text{para } i = 1 \text{ até } |S_y| - 1: \\ \quad \left(\begin{array}{l} \text{para } k = 1 \text{ até } \min\{7, |S_y| - i\}: \\ \quad \text{se } d(S_y[i], S_y[i+k]) < d_{min}: \\ \quad \quad p = S_y[i], q = S_y[i+k], d_{min} = d(p, q) \end{array} \right. \end{array} \right.$

Q4: por que só preciso olhar p/q at 7 seguintes ao i ?

devolva (p, q)

Qual a eficiência do algoritmo parMaisProximoDividido?

- $O(n)$, pois o laço interno executa um número constante (≤ 7) vezes.
no máximo

O que isso implica para a eficiência do algoritmo parMaisProximo?

- Cada chamada dele vai desencadear até **2** chamadas recursivas,
 - de subproblemas com **metade** do número de pontos,
 - e o trabalho local é **linear** no número de pontos.
- Assim, temos a recorrência $T(n) = 2T(n/2) + cn$
 - que é a mesma recorrência do **mergeSort**.
- Portanto, sabemos que ela corresponde a uma função de tempo $O(n \lg n)$

Por que o algoritmo parMaisProximoDividido está correto?

- Vamos ver uma demonstração em duas partes para sua corretude.

A base da demonstração é perceber que o par mais próximo dividido

- só nos interessa se sua distância for **menor** que delta,
 - já que já conhecemos pares não divididos com essa distância.

Para mostrar que o algoritmo está correto, primeiro vamos provar o seguinte lema.

Lema 1 - Se dois pontos p e q estão a menos que δ de distância,

- então $|x_p - x_q| < \delta$ e $|y_p - y_q| < \delta$

Prova: $d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$

$$(y_p - y_q)^2 \geq 0 \Rightarrow \sqrt{(x_p - x_q)^2} = |x_p - x_q|$$

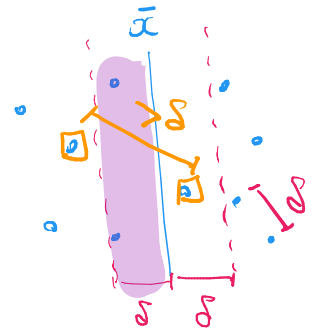
Portanto, se $d(p, q) \leq \delta$ então $|x_p - x_q| \leq d(p, q) \leq \delta$

Obs: Note que uma demonstração equivalente vale para a coordenada y .

$$a \Rightarrow b \Leftrightarrow \sim b \Rightarrow \sim a$$

A contrapositiva do Lema 1 diz que $|x_p - x_q| > \delta \Rightarrow d(p, q) > \delta$

- Como qualquer par dividido com
 - um ponto fora da faixa $[\bar{x} - \delta, \bar{x} + \delta]$
- está separado por uma faixa de largura δ
 - esse par não pode estar
 - a distância menor que δ



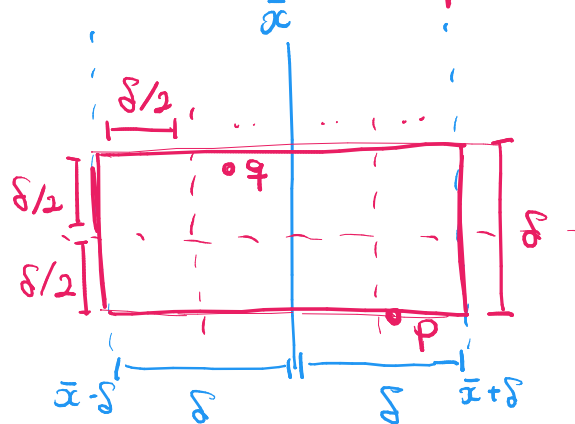
Vamos mostrar que qualquer par dividido (p, q) com distância menor que δ

- está a no máximo 8 posições sequenciais no vetor ordenado S_y

- Note que mostrar isso implica a corretude do algoritmo,
 - pois ele compara cada ponto em S_y com 7 pontos que o sucedem.

Pelo Lema 1, se um par dividido (p, q) tem distância menor que δ

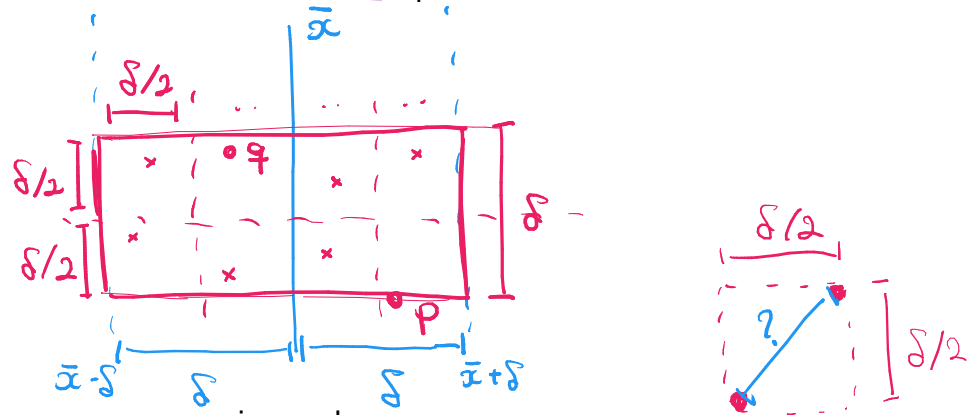
- a diferença entre suas coordenadas y deve ser menor que δ
- Portanto, podemos focar numa região com 2δ de largura,
 - e um δ de altura a partir da menor coordenada y entre p e q
- Sem perda de generalidade, suponha que p vem antes de q em S_y



Nos resta mostrar que não podem existir mais que 8 pontos nessa região.

- Faremos isso dividindo a região em 8 quadrados de lado $\delta/2$

É possível existir dois pontos a e b num mesmo quadrado?



- Por absurdo, vamos supor que sim, e observar que
 - (a, b) não é um par dividido, e que $d(a, b) \leq \sqrt{2} \delta/2 < \delta$
- Assim, se a e b estão no mesmo quadrado, eles estão
 - a uma distância menor que δ contrariando a definição do δ
- Disso concluímos que existe no máximo 1 ponto por quadrado.
 - Como são 8 quadrados, temos que p e q
 - estão a no máximo 8 posições sequenciais em S_y

Portanto, quando o algoritmo examina

- os próximos 7 pontos a partir do ponto corrente,
- se houver um par mais próximo dividido envolvendo o ponto corrente,
 - este par será encontrado.