PAA - Aula ~~05~~ 06
Encontrar o Par Mais Próximo

Definição do problema de encontrar o par mais próximo.

Entrada: um conjunto $P = \{p_1, p_2, \ldots, p_n\}$ 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

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

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 $\binom{n}{2} = C_2^n = \frac{n.(n-1)}{2}$ pares de pontos.
- Como cada comparação de distâncias leva tempo constante,
  - esse algoritmo encontra o par mais próximo em tempo $O(n^2)$

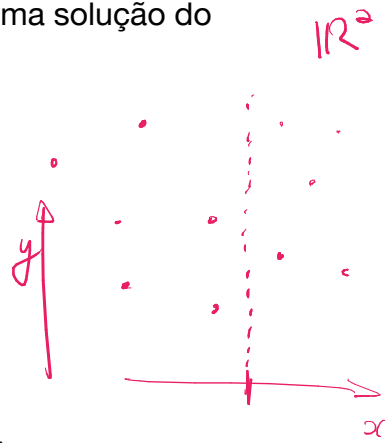
Será que conseguimos fazer melhor?

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.

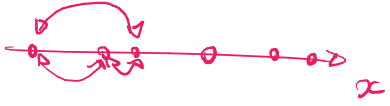
Para colocar alguma estrutura na instância do problema

- 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?



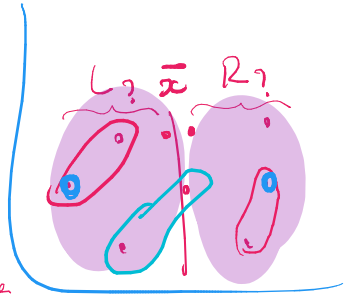
Obs: ordenar elementos da entrada é uma boa estratégia em diversos problemas.

Algoritmo recursivo simples

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

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

divida Px em Lx e Rx (e Py em Ly e Ry)
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
Quiz 2: essa divisão deve ser feita em tempo linear. Como?



$\frac{n}{2}$ $\frac{n}{2}$

$\frac{n^2}{4}$ pares

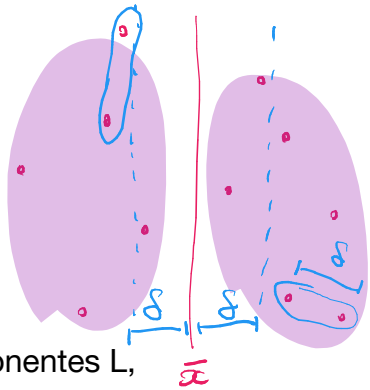
(pe, qe) = parMaisProximo (Lx, Ly)
(pd, qd) = parMaisProximo (Rx, Ry)
⇒ (pd, qd) = parMaisProximoDivido (Px, Py) ⇒ ainda temos $O(n^2)$ pares divididos

devolva o melhor entre (pe, qe), (pd, qd), (pd, qd)

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 compontentes 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) — $\delta = \min\{d(p_l, q_l), d(p_r, q_r)\}$

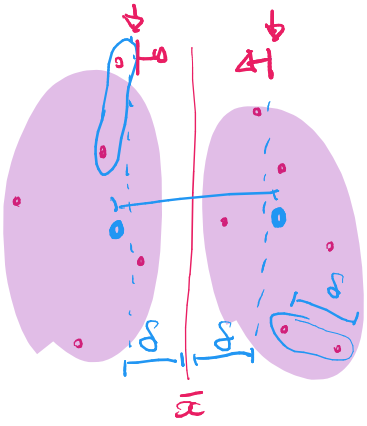
(p_d, q_d) = parMaisProximoDividido (P_x, P_y, δ) — O(n)

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) { $\bar{x}$ é uma coordenada x que divide P ao meio

Sy é a restrição do vetor ordenado Py que só tem pontos com coordenada x entre $\bar{x} - \delta$ e $\bar{x} + \delta$. Quiz 3: por que isso é seguro?

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

O(n) — para $i = 1$ até $|Sy| - 1$:

  O(1) { para $k = 1$ até $\min\{7, |Sy| - i\}$:

    se $d(Sy[i], Sy[i+k]) < d_{min}$:

      $p = Sy[i]$, $q = Sy[i+k]$, $d_{min} = d(p, q)$

Quiz 4: por que só preciso olhar p/ os 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: $d(p,q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$

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

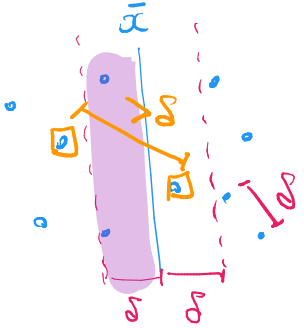
Portanto, se $d(p,q) < \delta$ então $|x_p - x_q| \leq d(p,q) < \delta$

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

$a \Rightarrow b \iff \sim b \Rightarrow \sim a$

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

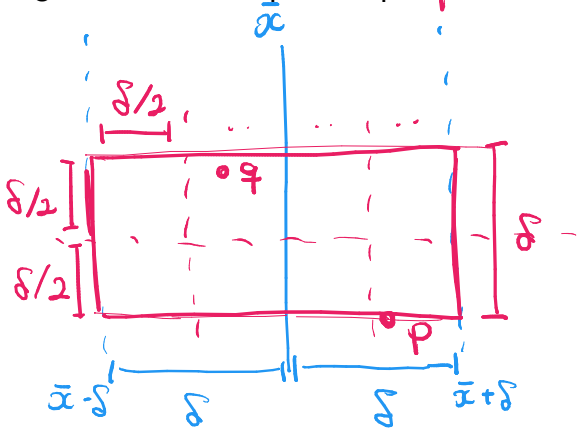


Vamos mostrar que qualquer par dividido (p, q) com distância menor que δ
- está a no máximo 8 posições sequenciais no vetor ordenado S_y

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

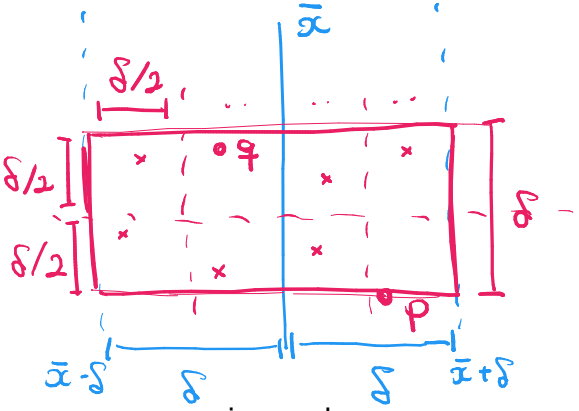
Pelo Lema 1, se um par dividido (p, q) tem distância menor que δ
- a diferença entre suas coordenadas y deve ser menor que δ

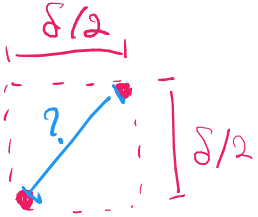
- Portanto, podemos focar numa região com 2δ de largura,
  - e um δ de altura a partir da menor coordenada y entre p e q
- Sem perda de generalidade, suponha que p vem antes de q em S_y



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.