PAA - Aula 05
Encontrar o Par Mais Próximo

Definição do problema de encontrar o par mais próximo.

Entrada: um conjunto P = {p1, p2, ..., pn} de n pontos no plano R^2,

- sendo R o conjunto dos números reais.

Distância Euclidiana: a distância entre dois pontos pi = (xi, yi) e pj = (xj, yj) é

- d(pi, pj) = ((xi - xj)^2 + (yi - yj)^2)^(½).
    - Note que x^(½) = sqrt(x) = raíz quadrada de x.

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 (n escolhe 2) = 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 menores 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 Px e Py.
- Px é ordenada pelas coordenadas x dos pontos
  - e Py é ordenada pelas coordenadas y.
  - Note que esse procedimento leva tempo O(n log n).

Para intuir porque ordenar os pontos é uma boa ideia,

- observem que a versão deste problema na linha (dimensão R)
  - ainda tem (n escolhe 2) pares para comparar,
- mas pode ser resolvida em tempo linear,
  - uma vez que os pares estejam ordenados (Quiz: por que?)

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

Algoritmo recursivo simples

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

  Se o número de pontos for menor que 4 (caso base)

    encontre o par mais próximo diretamente e devolva-o

  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
    (Quiz: isso deve ser feito em tempo linear. Como?)

  (pl, ql) parMaisProximo(Lx, Ly)
  (pr, qr) parMaisProximo(Rx, Ry)
  (pd, qd) parMaisProximoDividido(Px, Py)

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

Sendo (pd, qd) 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 nos permite fortalecer o algoritmo.

Algoritmo recursivo melhorado

Usando a observação anterior, temos o seguinte algoritmo recursivo melhorado.

```
(p, q) parMaisProximo(Px, Py):
    Se o número de pontos for menor que 4 (caso base)
        encontre o par mais próximo diretamente e devolva-o

    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.

    (pl, ql) parMaisProximo(Lx, Ly)
    (pr, qr) parMaisProximo(Rx, Ry)
    delta = min{d(pl, ql), d(pr, qr)}
    (pd, qd) parMaisProximoDividido(Px, Py, delta)

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

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

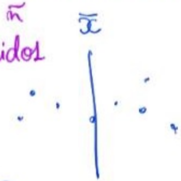
Pseudocódigo do algoritmo que analisa os pares divididos:

pontos ordenados por x

pontos ordenados por y

(p,q) ParMaisPróximoDividido(P_x, P_y, δ)

menor distância entre pares ñ divididos



- seja $\bar{x}$ uma coordenada x que divide o conj. de pontos ao meio

- seja S_y a restrição do vetor ordenado P_y que só considera pontos com coordenada x entre $\bar{x} - \delta$ e $\bar{x} + \delta$ (por que isso é seguro?)
- $d_{min} = \delta$, $p = q = Null$
- for $(i = 1;\ i < |S_y|;\ i{+}{+})$

    for $(k = 1;\ k \leq \min\{7, |S_y| - i\};\ k{+}{+})$ (por que só precisamos explorar os 7 próximos pontos?)

        if $(d(S_y[i], S_y[i+k]) < d_{min})$

            $p = S_y[i]$

            $q = S_y[i+k]$

            $d_{min} = d(p, q)$
- return (p, q)

Qual a eficiência do algoritmo parMaisProximoDividido?

- O(n), pois o laço interno executa um número constante (<= 7) vezes.

O que isso implica para a eficiência do algoritmo parMaisProximo?

- Cada chamada dele vai desencadear 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 $T(n) = 2 T(n/2) + cn$,
    - que é a mesma recorrência do mergeSort.
- Portanto, sabemos que ela corresponde a uma função de tempo $O(n \log n)$.

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.

Primeiro, vamos verificar que se dois pontos estão mais próximos que delta,

- então a diferença entre suas coordenadas x deve ser menor que delta.
    - Vale o mesmo para a diferença e entre suas coordenadas y.

Lema 1 - Se dois pontos p e q estão a menos que δ de distância, então o módulo da diferença entre suas coordenadas x (ou y) também é menor que δ

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

desigualdade vale pois $(y_p - y_q)^2 \geq 0$

$$\geq \sqrt{(x_p - x_q)^2}$$

$$= \boxed{|x_p - x_q|}$$

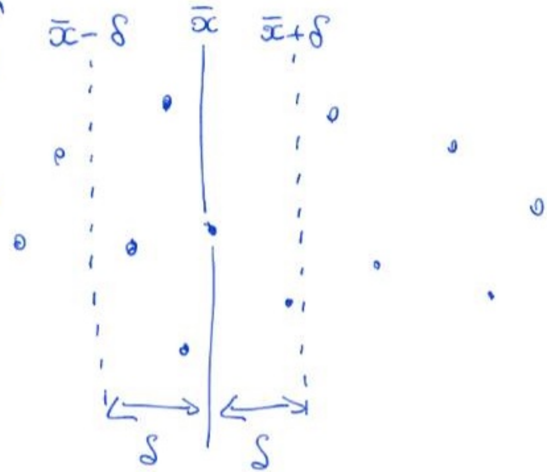
⊛ observe que a mesma demonstração vale para a coordenada y

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

O Lema 1 implica que qualquer par dividido com pelo menos um ponto fora da faixa $[\bar{x}-\delta, \bar{x}+\delta]$ não pode estar a distância menor que δ. Basta considerar a contrapositiva do lema, i.e.,

$$|x_p - x_q| > \delta \Rightarrow d(p,q) > \delta$$

e observar que os pontos estarão separados por uma faixa de largura δ.



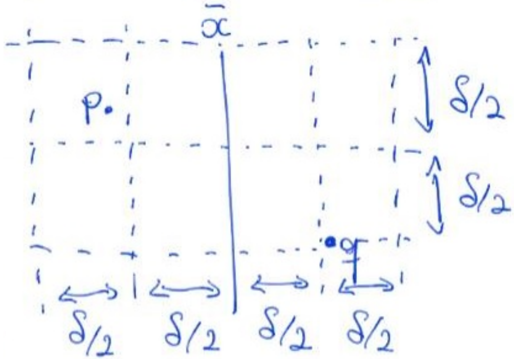
Assim, se um par dividido (p, q) tem distância menor que delta,

- suas coordenadas x devem estar entre (x' - delta) e (x' + delta),
    - sendo x' a coordenada x que dividiu os pontos ao meio.
- A diferença entre suas coordenadas y também deve ser menor que delta.
- Portanto, podemos focar numa região com dois deltas de largura, em torno de x', e um delta de altura a partir da menor coordenada y entre p e q.

com distância menor que δ

Agora vamos mostrar que qualquer par dividido está a no máximo 8 posições de distância no vetor ordenado S_y. Note que mostrar isso implica a corretude do alg., já que ele compara cada ponto em S_y com os 7 pontos que o sucedem. Começamos tomando um par dividido mais próximo p, q em S_y. Sem perda de generalidade, suponha que q está antes de p em S_y. Note que, pelo Lema 1 o ponto p deve estar a no máximo δ mais para cima que q. Nos resta mostrar que não podem existir mais do que 6 pontos em S_y com coordenada y entre y_q e y_p. Faremos isso dividindo a região em 8 quadrados de lado $\delta/2$ como mostra a figura.

Por absurdo, suponha que existem dois pontos a e b no mesmo quadrado. Observe que a e b não são um par dividido e $d(a,b) \leq \sqrt{2}\,\delta/2 < \delta$. Portanto, a e b 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 de distância 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.