Divisão e Conquista

O Problema da Menor Distância Entre Dois Pontos

2.Motivação

2.1Definição do problema

Resolver o problema da menor distância entre dois pontos possui uma série de aplicações em diversos algoritmos usados em diversos assuntos, desde a Física até a Computação Gráfica.

2.Motivação

2.1.2Definição formal do problema

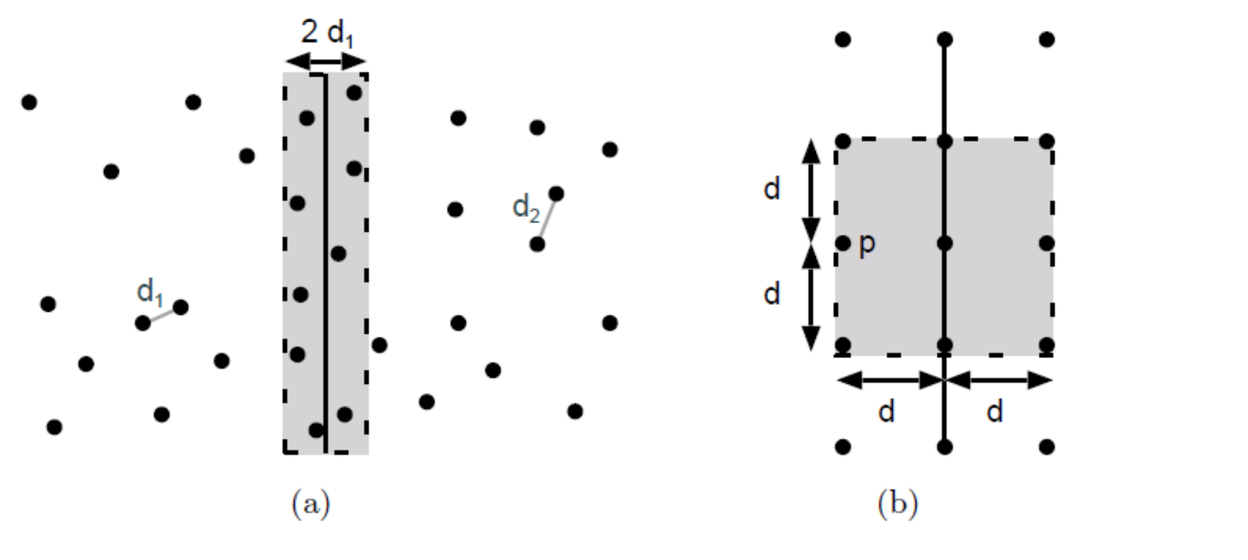
-Entrada: um conjunto de n pontos de um conjunto S num plano

-Questão:Como podemos encontrar a menor distância entre dois pontos de S

-Saída:Par de pontos mais próximo

2.Motivação

2.2Exemplos de instâncias do problema



3.Algoritmo

3.1Descrição da ideia do algoritmo.
Um algoritmo ingênuo faria a comparação das distâncias entre todos os pontos, o que levaria um tempo da complexidade de O (n^2).
Considerando os pontos ordenados em relação a x e y, traçamos uma reta r

3.1Descrição da ideia do algoritmo

dividindo S em S1 e S2, feito isto, resolvemos o problema recursivamente achando em cada subconjunto a menor distância entre dois pontos, e consideremos a menor distância das duas, temos então d = min(d1,d2).

3.1Descrição da ideia do algoritmo

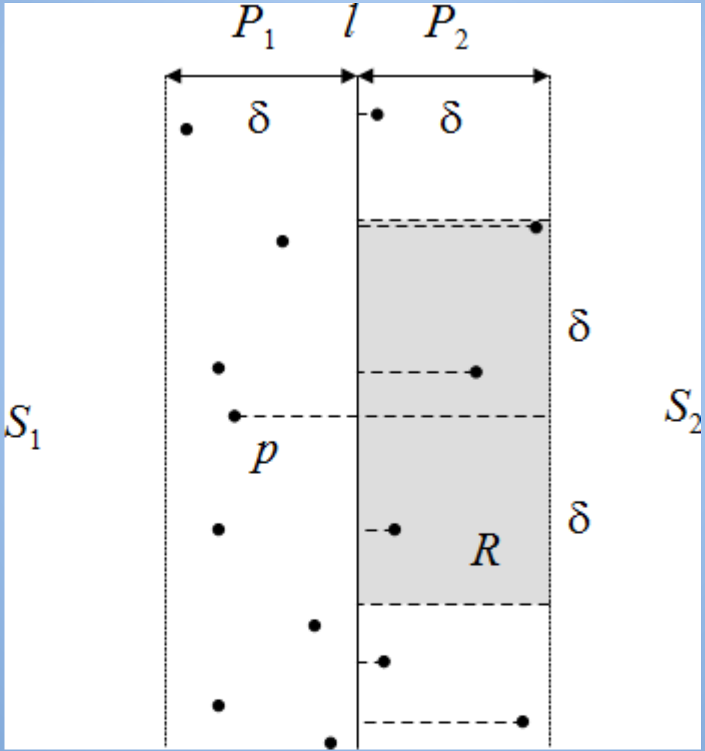
Tendo d, basta solucionarmos o problema para S1 U S2.

3.Algoritmo

3.2Apresentar solução para instância apresentada

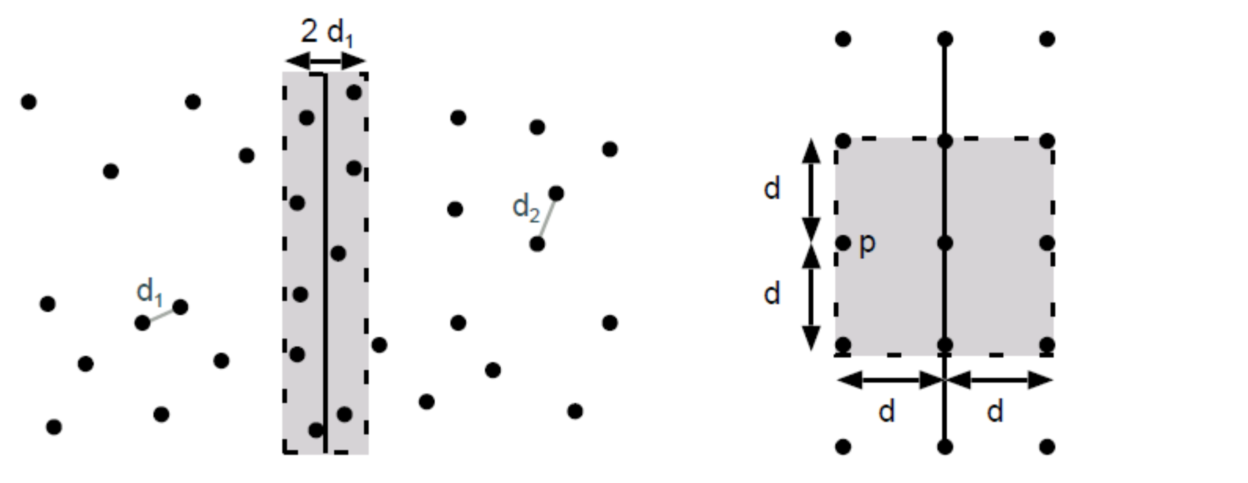
Partindo do princípio que temos d = min (d1,d2), buscamos para cada ponto p em S1 numa região conveniente, a distância entre p e pontos de S2 numa área bem definida e vice-versa.

3.2 Apresentar solução de instância



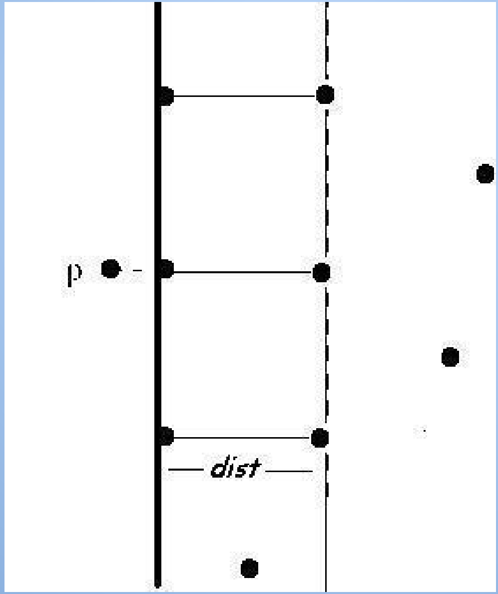
3.2 Apresentar solução de instância

Podemos garantir que apenas 6 pontos devem ser verificados



(a) (b)

3.2 Apresentar solução de instância



3.2Descrição formar do algoritmo

```
parDePontosMaisProximos of (xP, yP)
          onde xP is P(1) .. P(N) distribuido pela coordenada x, e
                 yP is P(1) .. P(N) distribuido pela coordenada y (ordem ascendente)
se N ≤ 3 entao
  retorna  pontos mais proximos de xP usando algoritmo de força bruta
else
  xL ← pontos de xP de 1 até ⌈N/2⌉
  xR ← pontos de xP de ⌈N/2⌉+1 até N
  xm ← xP(⌈N/2⌉)_x
  yL ← { p ∈ yP : p_x ≤ xm }
  yR ← { p ∈ yP : p_x > xm }                          inicializações
  (dL, pairL) ← parDePontosMaisProximos of (xL, yL)              menor distância em S1
  (dR, pairR) ← parDePontosMaisProximos of (xR, yR)              menor distância em S2
  (dmin, pairMin) ← (dR, pairR)
  se dL < dR entao
    (dmin, pairMin) ← (dL, pairL)                     d = min(d1,d2)
  endif
  yS ← { p ∈ yP : |xm - p_x| < dmin }                 região conveniente
  nS ← number of points in yS                         número de elementos nessa região
  (maisProximos, parDePontosMaisProximos) ← (dmin, pairMin)          inicialização
```

```
  for i from 1 to nS - 1
    k ← i + 1
    while k ≤ nS and yS(k)_y - yS(i)_y < dmin
      if |yS(k) - yS(i)| < maisProximos then
        (maisProximos, parDePontosMaisProximos) ← (|yS(k) - yS(i)|, {yS(k), yS(i)})  verificações de distância para a região
conveniente
      endif
      k ← k + 1
    endwhile
  endfor
  return closest, closestPair
endif
```

4. Análise do Algoritmo

4.1Prova de corretude

Para qualquer conjunto S de pontos o algoritmo o divide em dois subconjuntos, S1 e S2, verifica em cada um deles a menor distância recursivamente entre dois de seus pontos, de posse da menor distância verifica se há alguma menor em S1 U S2.

4.1 Prova de corretude

Assim, verificado S1, S2 e por último S1 U S2, fica garantido que sempre terminaremos achando a menor distância entre dois pontos de S.

4.2 Demonstração da complexidade

Para analisarmos a complexidade do tempo deste algortimo, construímos a pela recorrência,

$T(n) = 2T(n/2) + f(n)$

ao analisarmos S1 U S2 chegamos a f(n) que inicialmente seria $O(n^2)$ $(O(n/2) * O(n/2))$

mas, como vimos anteriormente é necessário apenas o cálculo para 6 pontos o que nos leva a $6 * O(n/2) = O(3n)$, ou seja, tempo linear $O(n)$.

Assim, pelo Teorema Mestre temos

$T(n) = 2T(n/2) + O(n) = n * \log n$

5. Conclusões e Discussões

5.1 Vantagens e desvantagens

O mesmo algoritmo não se aplica se falarmos em dimensões com um número de eixos maior que 2. No entanto, é um tempo bem razoável comparado ao inicial $O(n^2)$.

5.2 É possível um algoritmo melhor

Não, dado o fato que foram consideradas as devidas ordenações iniciais mas, diferentes definições de distância por exemplo, poderiam influenciar sua performance.

6.Referências

[1]http://pt.wikipedia.org/wiki/Dist%C3%A2ncia

[2]http://rosettacode.org/wiki/Closest-pair_problem

[3]http://en.wikipedia.org/wiki/Closest_pair_of_points_problem