Geometria Computacional

Aula 14

Par de Pontos Mais Próximos

Algoritmo de linha de varredura

Linha varre o plano da esquerda para a direita.

Ordene os pontos por x-coordenada.

Guardamos em δ a menor distância vista até o momento.

Manteremos numa ABBB todos os pontos à esquerda da linha, que estejam à distância menor que δ da linha.

Inicializações:

δ começa com a distância dos dois pontos mais à esquerda; ABBB começa vazia.

Que eventos podem causar alteração no δ ou na ABBB?

Possíveis eventos

Dois tipos de eventos:

- Evento de entrada:
  a linha atinge um ponto p da coleção.
- Evento de saída:
  um ponto p que está na ABBB fica à distância δ da linha.

Possíveis eventos

Dois tipos de eventos:

- Evento de entrada:
  a linha atinge um ponto p da coleção.
- Evento de saída:
  um ponto p que está na ABBB fica à distância δ da linha.

Evento de entrada:

p é inserido na ABBB e, se necessário, atualiza-se δ.

Possíveis eventos

Dois tipos de eventos:

- Evento de entrada:
  a linha atinge um ponto p da coleção.
- Evento de saída:
  um ponto p que está na ABBB fica à distância δ da linha.

Evento de entrada:

p é inserido na ABBB e, se necessário, atualiza-se δ.

Evento de saída:

p é removido da ABBB.

Possíveis eventos

Dois tipos de eventos:

- Evento de entrada:
  a linha atinge um ponto p da coleção.
- Evento de saída:
  um ponto p que está na ABBB fica à distância δ da linha.

Evento de entrada:

p é inserido na ABBB e, se necessário, atualiza-se δ.

Evento de saída:

p é removido da ABBB.

Como organizar a fila dos eventos?

Fila dos eventos

Ordenamos os pontos da coleção pela x-coordenada.

Fila dos eventos

Ordenamos os pontos da coleção pela x-coordenada.

Manteremos dois apontadores para este vetor ordenado:

- i: próximo ponto a entrar;
- j: próximo ponto a sair.

Fila dos eventos

Ordenamos os pontos da coleção pela x-coordenada.

Manteremos dois apontadores para este vetor ordenado:

- i: próximo ponto a entrar;
- j: próximo ponto a sair.

No início de cada iteração, avançamos a linha para o ponto i:
removemos da ABBB eventuais pontos que
estejam à distância $\geq \delta$ da linha, incluímos i.

$x \leftarrow P[i].x$
enquanto $j \leq i$ e $x - P[j].x \geq \delta$ **faça**
  remova o ponto j da ABBB
  $j \leftarrow j + 1$
insira $P[i]$ na ABBB
$i \leftarrow i + 1$

Como atualizar o valor de δ

Fora isso, ao inserirmos o ponto i,

se necessário, devemos atualizamos o valor de δ.

Como atualizar o valor de δ

Fora isso, ao inserirmos o ponto i,
se necessário, devemos atualizamos o valor de δ.

Como encontrar os pontos próximos do ponto i na ABBB?

Como atualizar o valor de δ

Fora isso, ao inserirmos o ponto i,
se necessário, devemos atualizamos o valor de δ.

Como encontrar os pontos próximos do ponto i na ABBB?

Em que ordem guardamos os pontos na ABBB?

Como atualizar o valor de δ

Fora isso, ao inserirmos o ponto i,
se necessário, devemos atualizamos o valor de δ.

Como encontrar os pontos próximos do ponto i na ABBB?

Em que ordem guardamos os pontos na ABBB?

Ordenados pela sua y-coordenada.

Como atualizar o valor de δ

Fora isso, ao inserirmos o ponto i,
se necessário, devemos atualizamos o valor de δ.

Como encontrar os pontos próximos do ponto i na ABBB?

Em que ordem guardamos os pontos na ABBB?

Ordenados pela sua y-coordenada.

Como isso nos ajuda?

Com que pontos queremos calcular a distância com i?

Como atualizar o valor de δ

Fora isso, ao inserirmos o ponto i,
se necessário, devemos atualizamos o valor de δ.

Como encontrar os pontos próximos do ponto i na ABBB?

Em que ordem guardamos os pontos na ABBB?

Ordenados pela sua y-coordenada.

Como isso nos ajuda?

Com que pontos queremos calcular a distância com i?

Com predecessores e sucessores
cuja y-coordenada difira de menos que δ.

Algoritmo de linha de varredura

Distância-SweepLine(P, n)

1 MergeSort(P, n) ▷ pontos ordenados pela x-coordenada
2 $\delta \leftarrow dist(P[1], P[2])$
3 CrieABBB(T) Insira$(T, 1)$ Insira$(T, 2)$
4 $j \leftarrow 1$

Algoritmo de linha de varredura

```
Distância-SweepLine(P, n)
 1  MergeSort(P, n)                ▷ pontos ordenados pela x-coordenada
 2  δ ← dist(P[1], P[2])
 3  CrieABBB(T)   Insira(T, 1)   Insira(T, 2)
 4  j ← 1
 5  para i ← 3 até n faça
 6      x ← P[i].x
 7      enquanto j < i e x − P[j].x ≥ δ faça
 8          Remova(T, j)
 9          j ← j + 1
10      Insira(T, i)
11      Atualiza(P, T, i, δ)
12  devolva δ
```

Algoritmo de linha de varredura

```
Distância-SweepLine(P, n)
 1  MergeSort(P, n)            ▷ pontos ordenados pela x-coordenada
 2  δ ← dist(P[1], P[2])
 3  CrieABBB(T)   Insira(T, 1)   Insira(T, 2)
 4  j ← 1
 5  para i ← 3 até n faça
 6      x ← P[i].x
 7      enquanto j < i e x − P[j].x ≥ δ faça
 8          Remova(T, j)
 9          j ← j + 1
10      Insira(T, i)
11      Atualiza(P, T, i, δ)
12  devolva δ
```

Consumo de tempo: $O(n \lg n)$ se o Atualiza(P, T, i, δ) for $O(\lg n)$.

Atualização do δ

```
Atualiza(P, T, i, δ)
1  j ← Predecessor(T, i)
2  enquanto j ≠ NIL e P[i].y − P[j].y < δ faça
3      d ← dist(P[i], P[j])
4      se d < δ então δ ← d
5      j ← Predecessor(T, j)
```

Atualização do δ

```
Atualiza(P, T, i, δ)
 1  j ← Predecessor(T, i)
 2  enquanto j ≠ NIL e P[i].y − P[j].y < δ faça
 3      d ← dist(P[i], P[j])
 4      se d < δ então δ ← d
 5      j ← Predecessor(T, j)
 6  j ← Sucessor(T, i)
 7  enquanto j ≠ NIL e P[j].y − P[i].y < δ faça
 8      d ← dist(P[i], P[j])
 9      se d < δ então δ ← d
10      j ← Sucessor(T, j)
```

Atualização do δ

```
Atualiza(P, T, i, δ)
 1  j ← Predecessor(T, i)
 2  enquanto j ≠ NIL e P[i].y − P[j].y < δ faça
 3      d ← dist(P[i], P[j])
 4      se d < δ então δ ← d
 5      j ← Predecessor(T, j)
 6  j ← Sucessor(T, i)
 7  enquanto j ≠ NIL e P[j].y − P[i].y < δ faça
 8      d ← dist(P[i], P[j])
 9      se d < δ então δ ← d
10      j ← Sucessor(T, j)
```

Qual é o consumo de tempo?

Atualização do δ

```
Atualiza(P, T, i, δ)
 1  j ← Predecessor(T, i)
 2  enquanto j ≠ NIL e P[i].y − P[j].y < δ faça
 3      d ← dist(P[i], P[j])
 4      se d < δ então δ ← d
 5      j ← Predecessor(T, j)
 6  j ← Sucessor(T, i)
 7  enquanto j ≠ NIL e P[j].y − P[i].y < δ faça
 8      d ← dist(P[i], P[j])
 9      se d < δ então δ ← d
10      j ← Sucessor(T, j)
```

Qual é o consumo de tempo? $\mathrm{O}(\lg n)$

pois há no máximo 4 predecessores e 4 sucessores a serem testados.

Geometria Computacional

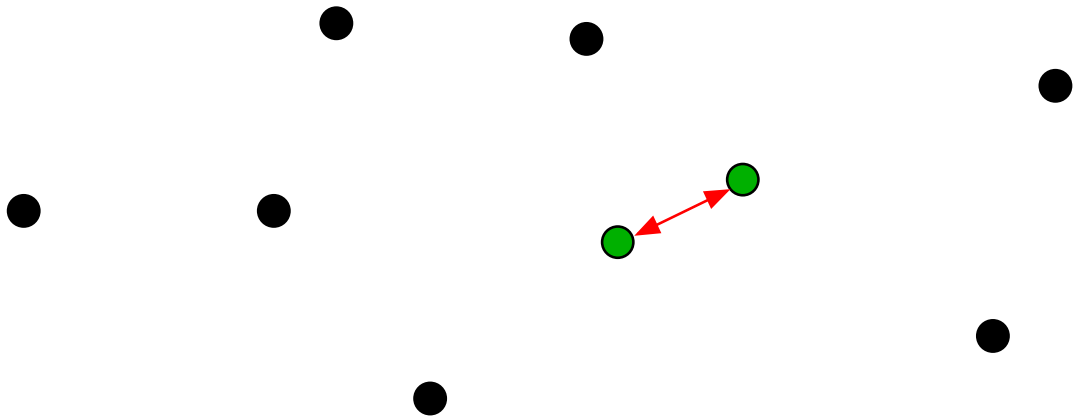
Aula 14

Par de Pontos Mais Próximos

Sec 13.7 do livro de Kleinberg e Tardos

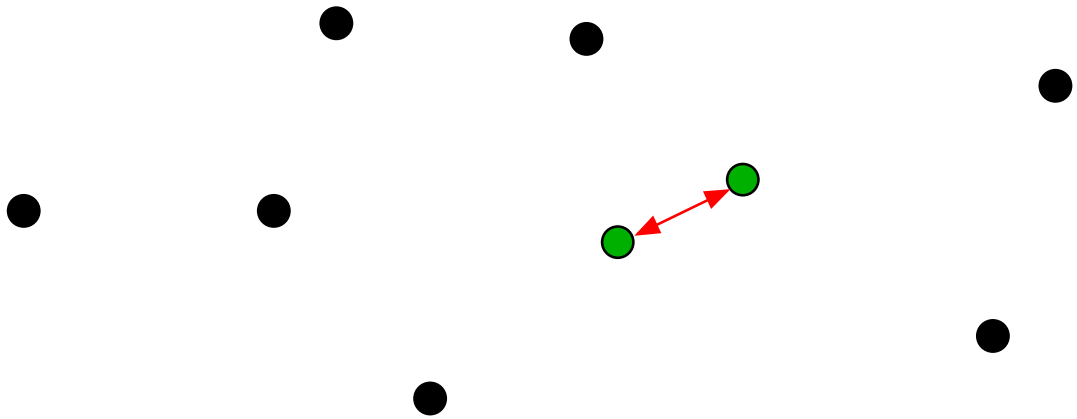
Par de pontos mais próximos

Problema: Dados n pontos no plano,
determinar dois deles que estão à distância mínima.



Par de pontos mais próximos

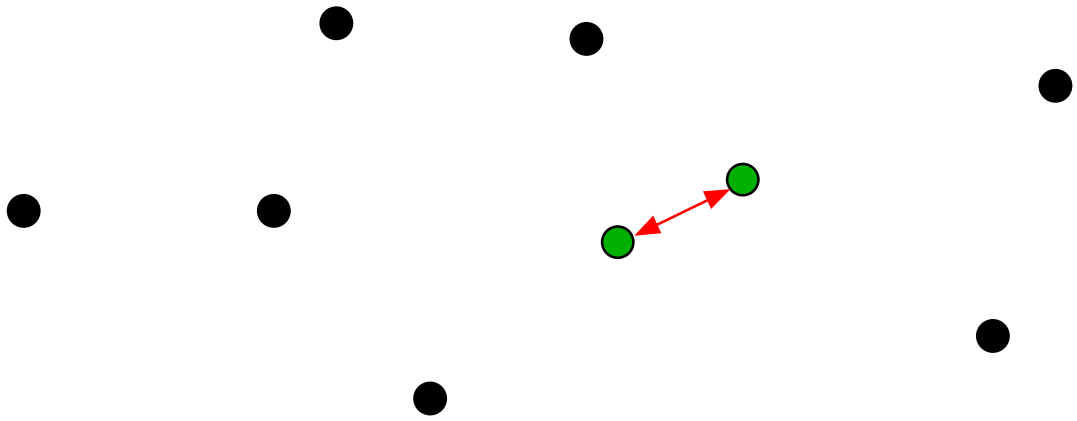
Problema: Dados n pontos no plano,
determinar dois deles que estão à distância mínima.



Esta aula: algoritmo aleatorizado cujo
consumo esperado de tempo é $\mathrm{O}(n)$.

Par de pontos mais próximos

Problema: Dados n pontos no plano,
determinar dois deles que estão à distância mínima.



Esta aula: algoritmo aleatorizado cujo
consumo esperado de tempo é $\mathrm{O}(n)$.

Hipótese simplificadora:
todos os pontos estão no quadrado $[0,1] \times [0,1]$.

Esboço do algoritmo

Problema: Dados n pontos no quadrado $[0,1] \times [0,1]$, determinar dois deles que estão à distância mínima.

Esboço do algoritmo

Problema: Dados n pontos no quadrado $[0,1] \times [0,1]$, determinar dois deles que estão à distância mínima.

Dividimos o quadrado em quadradinhos $\frac{\delta}{2} \times \frac{\delta}{2}$:
Seja $\delta = \mathit{dist}(p_1, p_2)$ e $Q_{i,k} = \left(\frac{i\delta}{2}, \frac{(i+1)\delta}{2}\right) \times \left(\frac{k\delta}{2}, \frac{(k+1)\delta}{2}\right)$, para $i, k = 0, \ldots, N$, onde $N = \lceil 2/\delta \rceil$.

Esboço do algoritmo

Problema: Dados n pontos no quadrado $[0,1] \times [0,1]$, determinar dois deles que estão à distância mínima.

Dividimos o quadrado em quadradinhos $\frac{\delta}{2} \times \frac{\delta}{2}$:
Seja $\delta = \mathit{dist}(p_1, p_2)$ e $Q_{i,k} = \left(\frac{i\delta}{2}, \frac{(i+1)\delta}{2}\right) \times \left(\frac{k\delta}{2}, \frac{(k+1)\delta}{2}\right)$,
para $i, k = 0, \ldots, N$, onde $N = \lceil 2/\delta \rceil$.

Para $j \leftarrow 3$ até n faça:
  Seja S o conjunto de pares (i, k) tais que
  existe um ponto p_ℓ com $\ell < j$ no quadrado $Q_{i,k}$.

Esboço do algoritmo

Problema: Dados n pontos no quadrado $[0,1] \times [0,1]$, determinar dois deles que estão à distância mínima.

Dividimos o quadrado em quadradinhos $\frac{\delta}{2} \times \frac{\delta}{2}$:
Seja $\delta = \mathit{dist}(p_1, p_2)$ e $Q_{i,k} = \left(\frac{i\delta}{2}, \frac{(i+1)\delta}{2}\right) \times \left(\frac{k\delta}{2}, \frac{(k+1)\delta}{2}\right)$, para $i, k = 0, \ldots, N$, onde $N = \lceil 2/\delta \rceil$.

Para $j \leftarrow 3$ até n faça:

Seja S o conjunto de pares (i, k) tais que existe um ponto p_ℓ com $\ell < j$ no quadrado $Q_{i,k}$.

Calcule (r, s) tal que p_j está em $Q_{r,s}$.

Para cada (i, k) em S tal que $|r - i| \leq 2$ e $|s - k| \leq 2$
calcule $\mathit{dist}(p_j, p_\ell)$ onde $\ell < j$ e $p_\ell \in Q_{i,k}$
e atualize δ quando necessário.

Esboço do algoritmo

Problema: Dados n pontos no quadrado $[0,1] \times [0,1]$, determinar dois deles que estão à distância mínima.

Dividimos o quadrado em quadradinhos $\frac{\delta}{2} \times \frac{\delta}{2}$:
Seja $\delta = \mathit{dist}(p_1, p_2)$ e $Q_{i,k} = \left(\frac{i\delta}{2}, \frac{(i+1)\delta}{2}\right) \times \left(\frac{k\delta}{2}, \frac{(k+1)\delta}{2}\right)$,
para $i, k = 0, \ldots, N$, onde $N = \lceil 2/\delta \rceil$.

Para $j \leftarrow 3$ até n faça:
  Seja S o conjunto de pares (i, k) tais que
  existe um ponto p_ℓ com $\ell < j$ no quadrado $Q_{i,k}$.

  Calcule (r, s) tal que p_j está em $Q_{r,s}$.

  Para cada (i, k) em S tal que $|r - i| \leq 2$ e $|s - k| \leq 2$
    calcule $\mathit{dist}(p_j, p_\ell)$ onde $\ell < j$ e $p_\ell \in Q_{i,k}$
    e atualize δ quando necessário.

Devolva δ.

Consumo de tempo estimado

Como fazer para o consumo esperado de tempo ser $\mathrm{O}(n)$?

Consumo de tempo estimado

Como fazer para o consumo esperado de tempo ser $\mathrm{O}(n)$?

Que ED usar para armazenar S?

Consumo de tempo estimado

Como fazer para o consumo esperado de tempo ser $\mathrm{O}(n)$?

Que ED usar para armazenar S?

Que operações sofre S?

Consumo de tempo estimado

Como fazer para o consumo esperado de tempo ser $\mathrm{O}(n)$?

Que ED usar para armazenar S?

Que operações sofre S?

Por iteração, uma inserção e algumas consultas.

Em algumas iterações, S muda totalmente...

Consumo de tempo estimado

Como fazer para o consumo esperado de tempo ser $\mathrm{O}(n)$?

Que ED usar para armazenar S?

Que operações sofre S?

Por iteração, uma inserção e algumas consultas.

Em algumas iterações, S muda totalmente...

Seria bom se...
inserções e buscas consumissem tempo (esperado) $\mathrm{O}(1)$.

Consumo de tempo estimado

Como fazer para o consumo esperado de tempo ser $\mathrm{O}(n)$?

Que ED usar para armazenar S?

Que operações sofre S?

Por iteração, uma inserção e algumas consultas.

Em algumas iterações, S muda totalmente...

Seria bom se...
inserções e buscas consumissem tempo (esperado) $\mathrm{O}(1)$.

Que ED atinge isso?

Consumo de tempo estimado

Como fazer para o consumo esperado de tempo ser $\mathrm{O}(n)$?

Que ED usar para armazenar S?

Que operações sofre S?

Por iteração, uma inserção e algumas consultas.

Em algumas iterações, S muda totalmente...

Seria bom se...
inserções e buscas consumissem tempo (esperado) $\mathrm{O}(1)$.

Que ED atinge isso? Hashing!

Primeira tentativa

Distância(P, n)

1 $\delta \leftarrow dist(P[1], P[2])$

2 seja (i_ℓ, k_ℓ) tal que $P[\ell] \in Q_{i_\ell, k_\ell}$ para $\ell = 1, 2$

3 Crie-Hashing(H) Insira($H, i_1, k_1, P[1]$) Insira($H, i_2, k_2, P[2]$)

Primeira tentativa

Distância(P, n)

1 $\delta \leftarrow dist(P[1], P[2])$
2 seja (i_ℓ, k_ℓ) tal que $P[\ell] \in Q_{i_\ell, k_\ell}$ para $\ell = 1, 2$
3 Crie-Hashing(H) Insira$(H, i_1, k_1, P[1])$ Insira$(H, i_2, k_2, P[2])$
4 **para** $j \leftarrow 3$ **até** n **faça**
5 seja (r, s) tal que $P[j] \in Q_{r,s}$
6 **para** $t \leftarrow -2$ **até** 2 **faça**
7 **para** $u \leftarrow -2$ **até** 2 **faça**
8 **se** Pertence$(H, r + t, s + u)$

Primeira tentativa

Distância(P, n)

1 $\delta \leftarrow dist(P[1], P[2])$
2 seja (i_ℓ, k_ℓ) tal que $P[\ell] \in Q_{i_\ell, k_\ell}$ para $\ell = 1, 2$
3 Crie-Hashing(H) Insira($H, i_1, k_1, P[1]$) Insira($H, i_2, k_2, P[2]$)
4 **para** $j \leftarrow 3$ **até** n **faça**
5 seja (r, s) tal que $P[j] \in Q_{r,s}$
6 **para** t, $u \leftarrow -2$ **até** 2 **faça**
7 **se** Pertence($H, r + t, s + u$)

Primeira tentativa

Distância(P, n)

1 $\delta \leftarrow dist(P[1], P[2])$
2 seja (i_ℓ, k_ℓ) tal que $P[\ell] \in Q_{i_\ell,k_\ell}$ para $\ell = 1, 2$
3 Crie-Hashing(H) Insira($H, i_1, k_1, P[1]$) Insira($H, i_2, k_2, P[2]$)
4 **para** $j \leftarrow 3$ **até** n **faça**
5 seja (r, s) tal que $P[j] \in Q_{r,s}$
6 **para** t, $u \leftarrow -2$ **até** 2 **faça**
7 **se** Pertence($H, r + t, s + u$)
8 **então** seja $P[\ell] \in Q_{r+t,s+u}$ com $\ell < j$
9 **se** $\delta > dist(P[j], P[\ell])$ **então** $\delta \leftarrow dist(P[j], P[\ell])$

Primeira tentativa

Distância(P, n)

1 $\delta \leftarrow dist(P[1], P[2])$
2 seja (i_ℓ, k_ℓ) tal que $P[\ell] \in Q_{i_\ell,k_\ell}$ para $\ell = 1, 2$
3 Crie-Hashing(H) Insira$(H, i_1, k_1, P[1])$ Insira$(H, i_2, k_2, P[2])$
4 **para** $j \leftarrow 3$ **até** n **faça**
5 seja (r, s) tal que $P[j] \in Q_{r,s}$
6 **para** t, $u \leftarrow -2$ **até** 2 **faça**
7 **se** Pertence$(H, r + t, s + u)$
8 **então** seja $P[\ell] \in Q_{r+t,s+u}$ com $\ell < j$
9 **se** $\delta > dist(P[j], P[\ell])$ **então** $\delta \leftarrow dist(P[j], P[\ell])$
10 **se** δ foi alterado nessa iteração
11 **então** Reconstrua(H, P, j)
12 **senão** Insira$(H, r, s, P[j])$
13 **devolva** δ

Primeira tentativa

Distância(P, n)

1 $\delta \leftarrow dist(P[1], P[2])$
2 seja (i_ℓ, k_ℓ) tal que $P[\ell] \in Q_{i_\ell,k_\ell}$ para $\ell = 1, 2$
3 Crie-Hashing(H) Insira($H, i_1, k_1, P[1]$) Insira($H, i_2, k_2, P[2]$)
4 **para** $j \leftarrow 3$ **até** n **faça**
5 seja (r, s) tal que $P[j] \in Q_{r,s}$
6 **para** t, $u \leftarrow -2$ **até** 2 **faça**
7 **se** Pertence($H, r + t, s + u$)
8 **então** seja $P[\ell] \in Q_{r+t,s+u}$ com $\ell < j$
9 **se** $\delta > dist(P[j], P[\ell])$ **então** $\delta \leftarrow dist(P[j], P[\ell])$
10 **se** δ foi alterado nessa iteração
11 **então** Reconstrua(H, P, j)
12 **senão** Insira($H, r, s, P[j]$)
13 **devolva** δ

Consumo de tempo esperado: $O(n)$ exceto pela linha 11.

Versão final

```
Distância (P, n)
 1  Embaralhe(P, n)      ▷ permutação aleatória dos pontos dados
 2  δ ← dist(P[1], P[2])
 3  seja (i_ℓ, k_ℓ) tal que P[ℓ] ∈ Q_{i_ℓ,k_ℓ} para ℓ = 1, 2
 4  Crie-Hashing(H)    Insira(H, i_1, k_1, P[1])    Insira(H, i_2, k_2, P[2])
 5  para j ← 3 até n faça
 6      seja (r, s) tal que P[j] ∈ Q_{r,s}
 7      para t, u ← −1 até 1 faça
 8          se Pertence(H, r + t, s + u)
 9              então seja P[ℓ] ∈ Q_{r+t,s+u} com ℓ < j
10                      se δ < dist(P[j], P[ℓ]) então δ ← dist(P[j], P[ℓ])
11      se δ foi alterado nessa iteração
12          então Reconstrua(H, p, j)
13          senão Insira(H, r, s, P[j])
14  devolva δ
```

Versão final

Distância (P, n)

```
 1  Embaralhe(P, n)      ▷ permutação aleatória dos pontos dados
 2  δ ← dist(P[1], P[2])
 3  seja (i_ℓ, k_ℓ) tal que P[ℓ] ∈ Q_{i_ℓ,k_ℓ} para ℓ = 1, 2
 4  Crie-Hashing(H)    Insira(H, i_1, k_1, P[1])    Insira(H, i_2, k_2, P[2])
 5  para j ← 3 até n faça
 6      seja (r, s) tal que P[j] ∈ Q_{r,s}
 7      para t, u ← −1 até 1 faça
 8          se Pertence(H, r + t, s + u)
 9              então seja P[ℓ] ∈ Q_{r+t,s+u} com ℓ < j
10                      se δ < dist(P[j], P[ℓ]) então δ ← dist(P[j], P[ℓ])
11      se δ foi alterado nessa iteração
12          então Reconstrua(H, p, j)
13          senão Insira(H, r, s, P[j])
14  devolva δ
```

Qual é o consumo de tempo esperado?

Consumo de tempo

Seja X o número de inserções em H.

O consumo de tempo é proporcional a $\mathrm{E}[X]$.

Consumo de tempo

Seja X o número de inserções em H.

O consumo de tempo é proporcional a $\mathrm{E}[X]$.

Seja X_j a variável binária que vale 1 sse
a linha 12 foi executada na iteração j.

$$X = \sum_{j=1}^{n}(1 + (j-1)X_j) \leq n + \sum_{j=1}^{n} j\, X_j$$

Consumo de tempo

Seja X o número de inserções em H.

O consumo de tempo é proporcional a $\mathrm{E}[X]$.

Seja X_j a variável binária que vale 1 sse
a linha 12 foi executada na iteração j.

$$X = \sum_{j=1}^{n}(1 + (j-1)X_j) \leq n + \sum_{j=1}^{n} j\, X_j$$

Mas então

$$\mathrm{E}[X] \leq n + \sum_{j=1}^{n} j\, \mathrm{E}[X_j] = n + \sum_{j=1}^{n} j\, \Pr\{X_j = 1\}.$$

Consumo de tempo

Seja X o número de inserções em H.

O consumo de tempo é proporcional a $\mathrm{E}[X]$.

Seja X_j a variável binária que vale 1 sse
a linha 12 foi executada na iteração j.

$$X \;=\; \sum_{j=1}^{n}(1+(j-1)X_j) \;\le\; n+\sum_{j=1}^{n} j\,X_j$$

Mas então

$$\mathrm{E}[X] \;\le\; n+\sum_{j=1}^{n} j\,\mathrm{E}[X_j] \;=\; n+\sum_{j=1}^{n} j\,\Pr\{X_j=1\}.$$

Note que $\Pr\{X_j=1\} \le 2/j$, logo $\mathrm{E}[X] \le n+2n = 3n$.

Variância

Em posição geral, não há
dois pares distintos de pontos da coleção a uma mesma distância.

Neste caso, $\Pr\{X_j = 1\} = 2/j$ e $\mathrm{E}[X] = 3n$.

$$\begin{aligned} \mathrm{Var}(X) &= \mathrm{E}[(X - 3n)^2] \\ &= \mathrm{E}[X^2 - 3nX + 9n^2] \\ &= \mathrm{E}[X^2] - 3n\mathrm{E}[X] + 9n^2 \\ &= \mathrm{E}[X^2]. \end{aligned}$$

Lembre-se que

$$X = \sum_{j=1}^{n}(1 + (j-1)X_j) \leq n + \sum_{j=1}^{n} j\,X_j.$$

Logo $\mathrm{E}[X^2] = \mathrm{E}[(n + \sum_{j=1}^{n} j\,X_j)^2]$.

Variância

$$\begin{aligned}
\mathrm{E}[X^2] &\leq \mathrm{E}[(n + \sum_{j=1}^{n} j\, X_j)^2] \\
&= \mathrm{E}[n^2 + 2n \sum_{j=1}^{n} j\, X_j + (\sum_{j=1}^{n} j\, X_j)^2] \\
&= n^2 + 4n^2 + \mathrm{E}[(\sum_{j=1}^{n} j\, X_j)^2].
\end{aligned}$$

Note que $X_j^2 = X_j$ e que X_j e X_i são independentes se $i \neq j$. Portanto,

$$\begin{aligned}
\mathrm{E}[X^2] &\leq 5n^2 + \sum_{j=1}^{n} \mathrm{E}[j^2 X_j] + 2 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} ij\, \mathrm{E}[X_i X_j] \\
&= 5n^2 + \sum_{j=1}^{n} j^2 \mathrm{E}[X_j] + 2 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} ij\, \mathrm{E}[X_i]\mathrm{E}[X_j].
\end{aligned}$$

Variância

$$
\begin{aligned}
\mathrm{E}[X^2] &\leq 5n^2 + \sum_{j=1}^{n} j^2 \Pr\{X_j = 1\} \\
&\quad + 2\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} ij \Pr\{X_i = 1\} \Pr\{X_j = 1\} \\
&= 5n^2 + \sum_{j=1}^{n} (2j) + 2\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 4 \\
&= 5n^2 + 2\sum_{j=1}^{n} j + 4\frac{n}{n-1} \\
&= 5n^2 + n(n+1) + 4\frac{n}{n-1} \\
&= 10n^2 - 3n.
\end{aligned}
$$