Universidad de Buenos Aires (FIUBA)

Maestría en Inteligencia Artificial

Trabajo Práctico 2

Closest Pair of Points (Divide y Vencerás)

Materia: Computación, Algoritmos y Estructuras de Datos
Docente: Dr. Lic. Camilo Argoty
Alumna: Esp. Lic. Noelia Qualindi
SIU: a1411

Repositorio del trabajo:

`https://github.com/noequalindi/computing_algorithms/tree/main/tp2`

1. Enunciado elegido

Se resuelve el punto 3 del TP2: dado un conjunto de puntos $P \subset \mathbb{R}^2$, encontrar una pareja (p, q) que minimice la distancia euclídea $d(p, q)$, con complejidad $O(n \log n)$.

2. Idea del algoritmo (Divide y Vencerás)

El enfoque clásico consiste en:

1. Ordenar los puntos por coordenada x (lista P_x) y por coordenada y (lista P_y).
2. Dividir el conjunto en dos mitades por una recta vertical $x = x_{mid}$: izquierda Q y derecha R.
3. Resolver recursivamente el par más cercano en Q y en R, obteniendo d_L y d_R. Sea $d = \text{mín}(d_L, d_R)$.
4. Considerar la *franja* (strip) de puntos a distancia menor que d de la recta divisoria. En esa franja, ordenar por y (ya está ordenada si se filtra desde P_y).
5. Verificar, para cada punto en la franja, solo los siguientes puntos en el orden por y (a lo sumo 7 vecinos). Esto es suficiente por un argumento geométrico estándar.

3. Correctitud (intuición)

El par más cercano global es:

- o bien está completamente en la mitad izquierda,
- o bien está completamente en la mitad derecha,
- o bien *cruza* la recta divisoria.

Los dos primeros casos quedan cubiertos por la recursión. Para el caso cruzado, si la distancia óptima es menor que d, entonces ambos puntos deben estar dentro de la franja de ancho $2d$ alrededor de x_{mid}; además, al ordenar por y, solo hace falta comparar con un número constante de vecinos.

4. Problema

Dado un conjunto $P = \{\bar{p}_1, \ldots, \bar{p}_n\} \subset \mathbb{R}^2$, encontrar $p, q \in P$ tales que la distancia sea mínima. Se utilizó distancia euclídea

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

5. Complejidad

5.1. Tiempo

- Ordenamientos iniciales: $O(n \log n)$.
- Recurrencia: $T(n) = 2T(n/2) + O(n)$ (filtrado y chequeo de franja).

Por Teorema Maestro, $T(n) = O(n \log n)$.

5.2. Memoria

Se utiliza memoria adicional $O(n)$ para mantener listas ordenadas por y y la franja.

6. Implementación

Se implementó el algoritmo en Python, siguiendo exactamente el esquema anterior.

6.1. Cómo correr

```
cd tp2
python tp2_closest_pair.py --examples
```

Esto ejecuta los 3 conjuntos de ejemplo requeridos (cada uno con 10 puntos) y muestra el par más cercano y la distancia mínima.

7. Evidencia de ejecución (local)

Se ejecutó la implementación en Python con tres conjuntos de puntos (cada uno de tamaño $n = 10$), tal como solicita la consigna. A continuación se incluye la salida de consola.

7.1. Ejemplo 1

```
python tp2_closest_pair.py --examples

=== Example 1 ===
Points: (0, 0), (2, 3), (3, 4), (5, 1), (1, 1), (4, 4), (7, 2), (6, 6), (8, 5),
(9, 1)
Closest pair: (3, 4) and (4, 4)
Minimum distance: 1.000000
```

7.2. Ejemplo 2

```
=== Example 2 ===
Points: (-5, -4), (-3, 2), (0, 0), (1, 1), (2, 2), (10, 10), (10, 11), (11, 10),
(3, -1), (4, -2)
Closest pair: (10, 10) and (10, 11)
Minimum distance: 1.000000
```

7.3. Ejemplo 3

```
=== Example 3 ===
Points: (0.1, 0.1), (0.2, 0.2), (0.3, 0.3), (5, 5), (6, 6), (7, 7), (8, 8),
(9, 9), (1, 1.01), (1.02, 1.03)
Closest pair: (1, 1.01) and (1.02, 1.03)
Minimum distance: 0.028284
```

7.4. Pruebas complementarias

Además de los tres conjuntos requeridos (n=10), se realizaron pruebas adicionales para verificar el comportamiento del programa en escenarios reproducibles y en un caso manual pequeño.

7.4.1. Caso aleatorio reproducible (seed=42)

```
python tp2_closest_pair.py --random 10 --seed 42

=== Random n=10 seed=42 ===
Points: (31, -36), (-47, 44), (-15, -19), (-22, -33), (44, -37), (36, 44), (19, -39), (25, 4)
Closest pair: (19, -39) and (31, -36)
Minimum distance: 12.369317
```

7.4.2. Caso manual (n=3)

```
python tp2_closest_pair.py --points "0,0" "1,2" "3,4"

=== Manual ===
Points: (0, 0), (1, 2), (3, 4)
Closest pair: (0, 0) and (1, 2)
Minimum distance: 2.236068
```

7.5. Repositorio

El código fuente y salidas reproducibles se entregan vía repositorio. Reemplazar el link:

`https://github.com/noequalindi/computing_algorithms/tree/main/tp2`

8. Bibliografía

Referencias

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009. (Closest pair of points / Divide y Vencerás).

[2] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008. (Closest pair of points en $O(n \log n)$).