FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1 - Greedy
Maximización del Emparejamiento

[75.29] Teoría de Algoritmos I
Primer cuatrimestre de 2023

Alumno:	DI MATTEO, CAROLINA
Número de padrón:	103963
Email:	cdimatteo@fi.uba.ar
Fecha de Entrega:	30-03-2023

Índice

1. Introducción

El presente informe reune la documentación de la solución al problema propuesto para el primer trabajo práctico de la materia Teoría de Algoritmos I que consiste en maximizar el emparejamiento de tuplas entre dos pares de conjuntos A y B utilizando una metodología de resolución de tipo Greedy.

2. Supuestos

Respecto del uso del programa, se supone que el usuario:

- envía 3 parámetros, donde el primero es el tamaño de ambos conjuntos, y el segundo y el tercero son las rutas de los archivos que contienen la información de los conjuntos A y B respectivamente
- verificó previamente que los archivos existen y las rutas son válidas
- verificó previamente que la información que contienen los archivos es válida
- verificó previamente que los tamaños de los conjuntos son equivalentes

Respecto de la resolución del problema, se suponen todos números reales y de dimensión 2.

3. Solución

3.1. Pseudocódigo

```
Ordeno A según componente x ascendiente.
Ordeno B según componente x ascendiente.

Por cada tupla en A:
    Por cada tupla en B:
        Si la componente x de la tupla de B es mayor a la componente x de la tupla de A:
            Corto el ciclo
        Si la componente y de la tupla de B es mayor a la componente y de la tupla de A:
            Continúo a la siguiente tupla de A
        Si la tupla de B es la más cercana a A respecto del eje y:
            Guardo la selección
    Devuelvo la selección final como la tupla de B que más cerca está de la tupla de A

    Si no se seleccionó ninguna tupla en B: Elimino la tupla en A y continúo
    Si no: Formo el match y elimino ambas tuplas de los conjuntos
```

3.2. Complejidad Temporal

Veamos que ordenar ambos conjuntos tiene complejidad $O(n \log n)$ y que, en el peor de los casos, por cada tupla en A se debe recorrer todo el conjunto B al menos una vez (pues el hecho de ir quitando elementos del mismo a medida que se realizan matches, disminuye su tamaño y por ende la complejidad de recorrerlo), que resulta en $O(n^2)$.

Finalmente, la complejidad temporal tiende a $O(n^2)$.

3.3. Complejidad Espacial

A priori, almacenar cada uno de los conjuntos resultaría en $2 * O(n)$. Ahora bien, veamos que para cada tupla en A, si no se forma un match la misma será eliminada del conjunto. En otro caso, se agrega a un nuevo conjunto de matches y las tuplas se eliminan de sus conjuntos de origen. Finalmente, una misma tupla no existe en dos lugares a la vez, de forma repetida. Por el contrario, se traslada a un nuevo conjunto o bien es eliminada de memoria, resultando así una complejidad espacial que tiende a $O(n)$.

3.4. Optimalidad

Llamaremos a la solución Greedy M, y a los elementos del match $m_1, ..., m_r$.
Imaginemos que existe una solcuión óptima O, cuyos elementos son $o_1, ..., o_s$.
Si $|M| = r$, $|O| = s$ y $s > r$, entonces:

1. o existe al menos un par de tuplas (donde una domina a la otra) que no fueron seleccionadas por el algoritmo Greedy,
2. o existe al menos una selección de tuplas hecha por Greedy que no lleva al óptimo global.

Veamos que (1) no puede ocurrir pues la heurística programada no lo permite: si una tupla domina a otra, se forma un match. De modo que esta situación se torna en un absurdo y queda descartada del abanico de posibilidades.

Ahora bien, para estudiar qué ocurre con (2), pensemos en las siguientes pares de tuplas: a_r, b_r, a_s, b_s; donde la solución óptima resulta en (a_r, b_s) y (a_s, b_r) y la solución Greedy en (a_r, b_r).
De la solución óptima vemos que:

- $a_r >= b_s$
- $a_s >= b_r$
- $a_r.x <= a_s.x$
- b_s es el elemento más cercano a a_r respecto al eje y
- b_r es el elemento más cercano a a_s respecto al eje y

Mientras que, de la solución Greedy podemos observar:

- $a_r >= b_r$
- $a_s < b_s$
- $a_r.x <= a_s.x$
- b_r es el elemento más cercano a a_r respecto al eje $y \rightarrow$ no existe otro elemento más hacia la izquierda cercano a $a_r \rightarrow$ ABSURDO!

Por naturaleza, la heurística del algoritmo Greedy propuesto, toma el elemento más lejano en el eje de las x, y más cercano en el eje de las y para realizar el match entre las tuplas de A y B. Finalmente, el algoritmo es óptimo.

4. Implementación

Para el desarollo de la solución propuesta, se crearon y utilizaron dos clases con el objeto de modelizar y modularizar los elementos *coordenadas* y *conjunto* (de coordenadas, es decir: conjunto de puntos).

La clase `coord` modela la estructura de una coordenada, de componentes x e y; modifica el *built-in* `__str__` (para facilitar el formateo de la estructura al momento de exportar la información);

e implementa el método `y_distance` que se encarga de calcular la distancia sobre el eje y de la coordenada respecto a otra que fuera enviada por parámetro.

La clase `conj` modela la estructura de un conjunto de puntos, o coordenadas. El atributo `conj` almacena -en una estructura de lista- todas las coordenadas que le sean asignadas, y el atributo `tam` contiene el valor del tamaño total del conjunto (con la única finalidad de evitar recorrer la lista de coordenadas cada vez que se quiera acceder a este valor). Esta clase implementa métodos tales como `push`, `pop`, `sort` y `empty` que se encargan de realizar las operaciones básicas y comunes que se esperarían de una lista, con alguna que otra particularidad amoldada al problema en cuestión. Cabe destacar que el método de ordenamiento especifica que el mismo debe hacerse respecto a la componente x de cada coordenada, tal y como se comentó previamente en este mismo informe. Finalmente, se encuentra desarrollada la función `y_closest` que -podríamos decir- es la que contiene más lógica puesto que es la encargada de entregar la tupla del conjunto que se encuentre más cercana respecto al eje de las y a un punto dado. Notemos que, como ambos conjuntos serán ordenados por componente x creciente, el valor entregado por este método será la tupla *más a la izquierda*, más próxima al punto verticalmente.

Finalmente, la lógica principal del programa se llama en la función `main`, donde no sólo se opera sobre los archivos y se procesan las coordenadas y los conjuntos, si no que además se solicita el procesamiento del matcheo entre conjuntos. Devuelve el resultado de matches según la disposición del enunciado.

5. Ejemplos

5.1. Primer Ejemplo

Para los puntos: $A = \{(26, 16), (22, 12), (29, 3), (30, 16), (21, 1)\}$ y $B = \{(27, 17), (17, 10), (3, 28), (5, 4), (24, 1)\}$ se encontraron los siguientes matches:

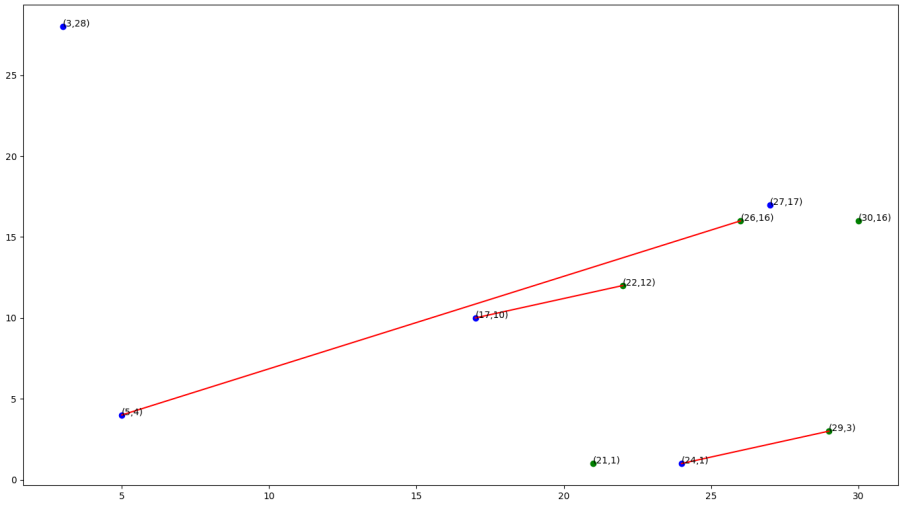

Figura 1: 5 Tuplas, 3 Matches

5.2. Segundo Ejemplo

Para los puntos: $A = \{(16, 19), (16, 11), (30, 27), (8, 19), (5, 5), (25, 30), (11, 30), (11, 17)\}$ y $B = \{(27, 28), (10, 29), (9, 13), (19, 17), (27, 2), (13, 30), (27, 25), (1, 28)\}$ se encontraron los siguientes matches:

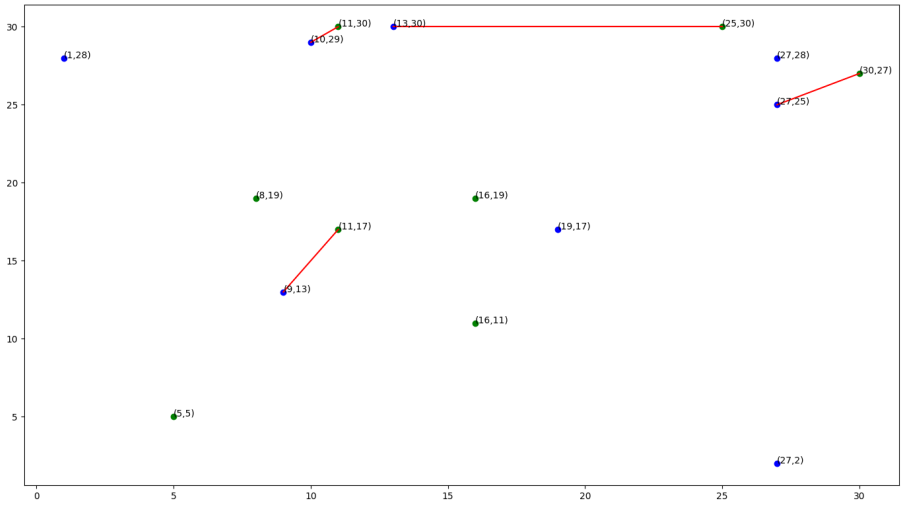

Figura 2: 8 Tuplas, 4 Matches

6. Conclusiones

Finalmente, podemos ver que el algoritmo propuesto en el presente informe es de tipo Greedy puesto que en cada paso busca soluciones óptimas locales (encontrar la tupla más cercana según el criterio previamente descripto) para alcanzar la solución óptima global. Podemos adicionar que el emparejamiento realizado tiene como objetivo barrer todas las tuplas *de izquierda a derecha*. Por otra parte, creo que son posibles otros tipos de emparejamientos pero que estos no pueden ser resueltos con metodologías de tipo Greedy por sus restricciones debido a la selección de tuplas de emparejamiento.