Trabajo práctico 1:
Vecinos más cercanos

Sistemas de Comunicaciones

Instituto La Salle Florida

Julio de 2021

1. Objetivo

El presente trabajo práctico tiene el objetivo de poner en práctica los conceptos de algoritmos recursivos y complejidad computacional vistos en clase. Para ello, se propone una situación problemática muy similar a un caso de aplicación real, como motivación, el cual se debe resolver bajo un marco acotado pero amplio de conceptos troncales para la materia.

2. Motivación

Una empresa de comunicación masiva decidió lanzar una campaña de publicidad para fomentar la descarga e instalación de una APP de descuentos en dispositivos móviles.

Esta APP, al ser descargada, registra las coordenadas de posición del dispositivo con el fin de poder hacer análisis de geolocalización.

Esta empresa, para estimular la descarga de la APP, decidió ofrecer crédito a favor de los usuarios que la instalen. Sin embargo, al ver que con este estímulo no era suficiente, se decidió sumar otro estímulo que genere que las mismas personas que ya descargaron la APP, fomenten a que sus allegados también la descarguen.

Este segundo estímulo consta de un premio de USD 100K a aquellas 2 personas que cuando descargaron la APP, estaban mas cerca físicamente.

Este segundo estímulo se lanzó a mitad del mes anterior y finalizó hoy, con la promesa de dar conocimiento en los medios a aquellos ganadores del premio monetario en 3 días.

Sorprendentemente, la cantidad de descargas de la APP ascendió a 1 millón de personas dejándole un problema a la empresa con poco tiempo para resolver, pues en solo 3 días, deben poder procesar 1 millón de coordenadas en el plano y determinar cuales son las que están más próximas.

Los programadores más capacitados que tiene contratados la empresa le indican al gerente de Marketing que no hay manera de correr un algoritmo que en 3 días encuentre las 2 coordenadas más próximas, pues se deben computar las distancias que tiene cada coordenada con cada una de las otras para elegir así la menor, y ese es un problema de resolución que tiene complejidad $O(n^2)$, donde con las computadoras que se manejan, ese algoritmo podría tardar hasta 4 meses en terminar de ejecutarse.

Dado esto, el gerente de Marketing con mucha presión de tener que mostrar un resultado en los medios, decide contratarte a vos, con el único objetivo de poder resolver este problema, en menos de 3 días.

3. Desarrollo

Como se mencionó anteriormente, el objetivo del trabajo es encontrar las 2 coordenadas más próximas en un conjunto de 1 millón de coordenadas con un algoritmo que tarde menos de 3 días en correr.

Como para poder simplificar y entender bien el objetivo, se muestra en la figura 1 un ejemplo con 8 pares de coordenadas o puntos, donde se puede ver a simple vista que los puntos P1 y P2 son los vecinos más cercanos, pues están a 1 unidad de distancia en el plano. Entonces, la idea básicamente es poder desarrollar un algoritmo que logre procesar todas las coordenadas y encuentre aquellos 2 pares de coordenadas que están más próximos.

No obstante, es un problema bastante complejo de resolver sin hacer pasos previos y analizando bien en detalle la mejor estrategia de resolución. Por lo tanto, para tener éxito, vamos a seguir una serie de pasos que van a ayudar en la resolución del problema.

Junto con este enunciado, se proveen los siguientes 3 archivos:

- coordenadas1.csv: contiene solo 10 mil coordenadas para probar el algoritmo.
- coordenadas2.csv: contiene 100 mil coordenadas para también probar el algoritmo.

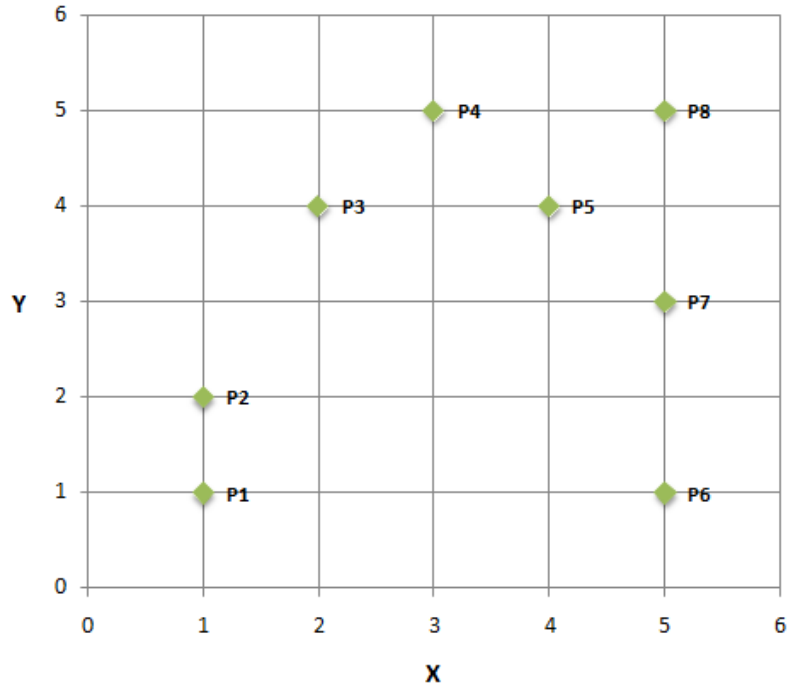

Figura 1: Ejemplo introductorio del problema de vecinos más cercanos

- coordenadas totales.csv: contiene el millón de coordenadas totales que recolectó la empresa.

Algunas consideraciones a descatar:

- Las coordenadas son todas variables numéricas de punto flotante y positivas para simplificar la resolución del problema.
- Las coordenadas son euclídeas y no geo-espaciales (latitud y longitud) para simplificar el problema.
- Las coordenadas se encuentran medidas en metros.

Antes de comenzar a escribir código para intentar resolver el problema, debemos tratar de armar un framework de trabajo que nos permita ir desarrollando un algoritmo superador al algoritmo de fuerza bruta, pero que de alguna forma debemos validar que funcione bien. Por lo tanto, a continuación, se sugieren los siguientes pasos distribuidos en distintas secciones donde en cada una, se listan una serie de **TO-DOs** que sirven de guía para lograr resolver el problema.

3.1. Funciones auxiliares

Se recomienda mucho poder dejar una parte del código destinada a escribir funciones auxiliares que sirvan en la resolución del problema. Por ejemplo, una función que sin dudas va a necesitar usarse es calcular la distancia entre 2 coordenadas.

Dados 2 puntos en el plano $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, la distancia entre los 2 puntos queda definida como:

$$d(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

TO-DO: escribir una función que reciba 2 pares de coordenadas en forma de tupla cada una y calcule la distancia entre ellas.

Otra función que puede ser interesante escribir es aquella que se ocupe de abrir un archivo .csv y deje la información de las coordenadas en una lista de tuplas. Por ejemplo, si tuvieramos que procesar un archivo que contenga las coordenadas de los 8 puntos mostrados en al figura 1, el resultado debería ser la siguiente lista:

$$L = [(1,1), (1,2), (3,5), (2,4), (5,1), (4,4), (5,3), (5,5)]$$

TO-DO: escribir una función que reciba el nombre de un archivo de coordenadas y devuelva por el nombre una lista de tuplas con las coordenadas leídas del archivo[1].

3.2. Algoritmo por fuerza bruta

Si quisieramos resolver el problema de encontrar los 2 puntos (vecinos) más cercanos, siempre podemos recurrir a la técnica por fuerza bruta, la cual básicamente se basa en, dado N puntos, computar las distancias para cada punto contra cada uno de los otros puntos y luego elegir aquellos 2 puntos que registran la mínima distancia.

Si por ejemplo, volvemos a revisar la figura 1, que tiene 8 puntos, deberíamos computar 8 x 8 = 64 distancias y así elegir la mínima [2].

Por ejemplo, si vemos la figura 1, veríamos que P1 y P2, están a 1 unidad de distancia, mientras que P1 y P6 están a 4 unidades, así como P3 y P5 están a 2 unidades.

TO-DO: escribir una función que dada una lista de tuplas que contiene las coordenadas de N puntos, busque aquellos puntos 2 puntos que están a la distancia mas chica del to-

[1] Para resolver esto, va a ser necesario que investigues en internet como leer un archivo en Python.

[2] En rigor de verdad, podemos ahorrarnos calcular la distancia de un punto consigo mismo y así no calcular 8 de las 64 distancias, dando como resultado 8 x 7 distancias, pero que igualmente resulta con una complejidad $O(n^2)$

tal y devuelva por el nombre en una lista 3 elementos: los 2 puntos y la distancia entre ellos.

Se recomienda probar el algoritmo con una lista de pocos puntos primero, por ejemplo la misma de la figura 1, para asegurarse que el algoritmo este bien escrito. Luego, probar algoritmo con el archivo coordenadas1.csv a fin de verificar que la distancia mínima se da en los puntos:

- P_a = (7727.420256, 8352.613949)
- P_b = (7727.504707, 8353.293855)
- $d(P_a, P_b) = 0.6851307468185359$

¿Cúantos segundos demoro el agoritmo en encontrar la solución?

Finalmente, a modo de reflexión, te invito a que te imagines, dado que tenemos 1 millón de puntos y tendríamos que procesar N x N distancias, cuantos días puede tardar el algoritmo en finalizar, si para procesar cada distancia, la PC tarda solo $10\mu s$.

3.3. Algoritmo recursivo

Como ya te habrás imaginado, el camino de resolver el problema usando el algoritmo iterativo de fuerza bruta no sirve dado que podemos tardar un cuatrimestre en llegar a tener el resultado.

Dado esto, es que necesitamos desafiarnos y pensar un algoritmo que tenga una mejor performance y pueda resolverse en un tiempo de ejecución menor.

¿Se te ocurre algún algoritmo recursivo, donde utilizando la técnica de Divide & Conquer, logre resolver el problema con una complejidad menor a $O(n^2)$?

Si bien pueden existir varias alternativas de solución, vamos a proponer un algoritmo que logre esto. Para ello, vamos a enunciar en modo texto, que debe hacer el algoritmo:

1. **Ordenar la lista por el eje X:** el primer paso va a constar de ordenar la lista de N elementos en orden ascendente y de acuerdo únicamente a la coordenada X de cada punto. Sin embargo, usar un método de ordenamiento que tenga una complejidad $O(n^2)$ no nos va a servir puesto que, si lo usáramos, ya habríamos consumido el mismo tiempo que si buscabamos resolver el problema por fuerza bruta. Por lo tanto, para que esto funcione, sugiero usar un algoritmo como el Merge Sort, pero adaptándolo para que ordene una lista de tuplas, de acuerdo a su coordenada X, de forma ascedente.

TO-DO: escribir una función que dada una lista de tuplas que contiene las coordenadas de N puntos las ordene en forma ascendente de acuerdo a la coordenada X de los puntos y devuelva la lista ordenada por el nombre.

2. **Casos base:** como bien sabes, todo algoritmo recursivo debe tener un caso base, donde la función no vuelva a invocarse a si misma y directamente devuelva un resultado. Para este algoritmo, los casos base pueden ser 2 situaciones distintas:

   - N = 2, en este caso, al tener 2 puntos, solo debemos devolver los 2 puntos y la distancia entre ellos.
   - N = 1, si tenemos un solo punto, no tenemos forma de computar ninguna distancia, por lo tanto, debemos devolver el único punto que tenemos y una distancia infinita o bien, un número negativo como distancia que sea interpretado como infinito.

3. **Divide:** en caso de no estar en algún caso base, vamos a tener una lista de más de 2 puntos, lo cual nos permite dividir el problema en 2 listas de mitad de tamaño. En este paso, debemos hacer eso y crear 2 sub-listas llamadas L1 y L2.

4. **Llamados recursivos:** invocar a la función recursiva 2 veces; una con la lista L1 y otra con la lista L2. A cada sub-solución llamarlas res_1 y res_2.

5. **Conquer:** este es el paso clave. Acá es donde debemos ingeniarnolas para que, teniendo las 2 listas L1 y L2, y las 2 sub-soluciones que el mismo algoritmo recursivo me provee en el paso anterior, lograr encontrar combinar estas soluciones en una solución final. Para ello se plantean los siguientes pasos:

   a) Elegir, entre res_1 y res_2 la mejor solución (aquella que tiene la menor distancia). Llamar a esa solucion res_12.

   b) Encontrar las coordenadas de las 2 rectas verticales y divisorias de las listas L1 y L2. Esto sería viendo la coordenada X el último punto de L1 y la coordenada X del primer punto L2. Por ejemplo, mirando la figura 1, si separamos en 2 grupos los puntos (desde P1 a P4 en un grupo y desde P5 a P8 el otro), las rectas verticales divisorías serían X = 3 y X = 4.

   c) Buscar todos los puntos de L1 que estén a una menor distancia que lo hallado en la solución res_12 de la recta divisoria de la lista L2. Hacer lo mismo con todos los puntos de L2 y la recta divisoria de L1. Recordar que, al estar ordenadas las listas L1 y L2, podemos hacer la búsqueda en forma inteligente para no tener que recorrer ambas listas, sino cortar cuando ya veamos que los puntos siguientes van a estar mas lejos a la recta divisoria que lo encontrado en el paso anterior.

   d) Con los puntos encontrados en el paso anterior, por fuerza bruta, calcular distancia entre ellos y elegir aquel par de puntos que tengan la menor distancia.

e) Comparar la distancia encontrada en el paso anterior, con la solución res_12, y elegir aquella que sea mínima para devolver por el nombre

TO-DO: ejecutar el algoritmo descripto anteriormente a mano con lapiz y papel, usando el ejemplo mostrado en la figura 1. Incluir esta ejecución en el informe de este TP.

TO-DO: escribir la función recursiva usando el algoritmo planteado anteriormente y probar su funcionamiento con el ejemplo de la figura 1 y con el archivo de coordenadas1.csv.

¿Cúantos segundos demoro el agoritmo en encontrar la solución con el archivo coordenadas1.csv?

Es importante no seguir desarrollando el TP hasta no tener funcionando este algoritmo, dado que todo lo que sigue, requiere que no haya errores en el algoritmo recursivo.

3.4. Análisis complejidad computacional

TO-DO: Analizar en forma teórica la complejidad computacional tanto del algoritmo iterativo por fuerza bruta y el algoritmo recursivo que usa la estrategia de Divide & Conquer.

3.5. Evaluación a posteriori

TO-DO: Escribir un programa que genere listas de tuplas aleatorias de tamaño creciente (se recomienda desde N = 2 y hasta N = 2000), y que para cada lista, busque los vecinos más cercanos con ambos algoritmos, compute los tiempos de ejecución y realice un gráfico compartativo para evaluar la complejidad de los algoritmos a posteriori.

3.6. Resolución final

TO-DO: Utilizar el algoritmo recursivo para resolver el problema de vecinos más cercanos con los 2 archivos restantes.

La solución a las coordenadas del archivo coordenadas2.csv debería ser:

- P_a = (58.28449453, 9978.423861)
- P_b = (58.2383031, 9978.47222)
- $d(P_a, P_b) = 0.06687481653416626$

TO-DO: Reflexione y piense por qué cree que el mismo problema, con una solución correcta, puede tener tanta diferencia de tiempos de ejecución. ¿Hay algo malo en el algoritmo recursivo? ¿Cómo es posible esto? [3]

[3] Quizás sirva investigar el concepto de heurística.