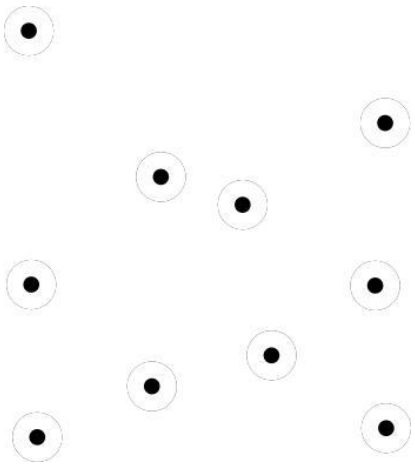
Práctica 2 de AMC: Algoritmos DYV y Voraces

Parte 1 Análisis de Divide y Vencerás.

Problema del trío de puntos más cercanos

El problema es: "dado un conjunto de puntos P = { (x_1,y_1) (x_2,y_2) … (x_n,y_n) } hallar el **trío** de puntos más cercanos" teniendo en cuenta que la distancia entre dos puntos i y j es sqrt$[(x_i-x_j)^2+(y_i-y_j)^2]$.

Una primera solución podría ser mirar todos los tríos de puntos y quedarse con el más pequeño; al haber n·(n-1)/2·(n-2)/3 tríos de puntos, el tiempo de ejecución es de **O(n^3)**. El programa resultante es corto y muy rápido para casos pequeños, pero a medida que aumenta el tamaño del conjunto de puntos el tiempo de ejecución va creciendo de forma exponencial, al incrementarse exponencialmente el número de combinaciones de puntos que forman un trío.

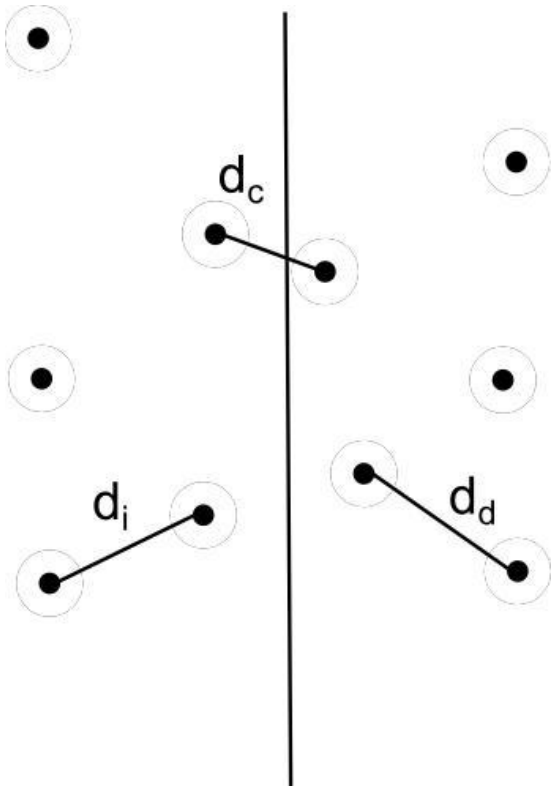


Esta primera solución plantea una búsqueda exhaustiva, pero, ¿existe un algoritmo más rápido para solucionar el problema? Una solución basada en la técnica **Divide y Vencerás** puede ser la siguiente:

Supongamos que ordenamos los puntos según la coordenada x; esto supondría un tiempo de **O(n·log n)**, por lo que el algoritmo tarda como mínimo eso (es una cota inferior para el algoritmo completo). Ahora que se tiene el conjunto ordenado, se puede trazar una línea vertical, $x = x_m$, que divida al conjunto de puntos en dos: P_i y P_d. Ahora, o el trío más cercano está en P_i, o está en P_d, o uno o dos puntos del trío está en P_i y el resto en P_d. Si los tres estuvieran en P_i o en P_d, se hallaría recursivamente, subdividiendo más el problema. El caso más complejo, por tanto, se reduce al tercer caso, cuando alguno de los puntos del trío se encuentra en la otra zona.

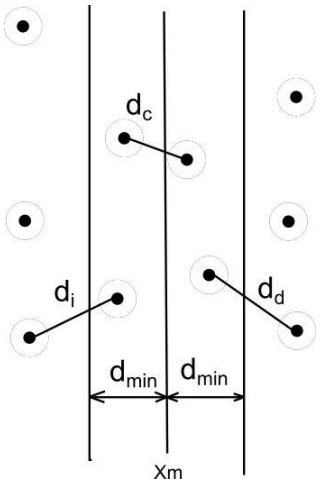
Para ayudaros a resolver el problema a continuación os explico cómo solucionarlo para el caso en que quisiera hallarlo para sólo 2 puntos (en lugar de 3). Una vez entendido cómo se ha resuelto lo extrapoláis al caso de 3 puntos.

En el caso de hallar la distancia mínima de un par de puntos con la técnica **Divide y Vencerás**, al dividir recursivamente el conjunto de puntos en dos partes podría darse los 3 casos antes comentados:



Como podemos observar en la figura, el par más cercano puede estar en la zona izquierda de la figura (zona P_i), o estar en la zona derecha de la figura (zona P_d), o bien uno de los puntos puede estar en la zona izquierda (zona P_i) y el otro en la derecha (zona P_d).

Llamemos d_i, d_d y d_c a las mínimas distancias en el primer caso, en el segundo, y en el tercero, respectivamente, y d_{min} al menor de d_i y d_d. Para resolver el tercer caso, sólo hace falta mirar los puntos cuya coordenada x esté entre x_m-d_{min} y x_m+d_{min}. Para grandes conjuntos de puntos distribuidos uniformemente, el número de puntos que caen en esa franja es sqrt(n), así que con una búsqueda exhaustiva el tiempo de ejecución sería de **O(n)**, y tendríamos el problema resuelto. El tiempo de ejecución sería, según lo dicho en el otro apartado, **O(n·log n)**.



Pero si los puntos no están uniformemente distribuidos, la cosa cambia. En el peor de los casos, todos los puntos están en la franja, así que la fuerza bruta no siempre funciona en tiempo lineal. Para ello, se puede recurrir a ordenar los puntos de la franja según la coordenada, lo que supone un tiempo de ejecución de **O(n·log n)**. Opcional (2 puntos): encontrar un método para que el coste de este último paso sea de **O(n·log n)** en cualquier caso, con lo que mantendríamos el tiempo de ejecución anterior.

Desarrollo

Se pide al alumno que realice una codificación exhaustiva (la propuesta en el apartado anterior u otra diferente) y la codificación divide y vencerás explicada en el lenguaje de programación que el alumno prefiera (preferiblemente Java o C++).

El alumno deberá entregar un estudio teórico realizado sobre el pseudocódigo simplificado de ambos algoritmos donde se calcule el mejor y el peor caso de manera razonada. En el mismo documento se deben realizar **10 ejecuciones** representativas de casos para cada uno de los algoritmos (mejor caso, peor caso, ejemplos de casos limites, y aleatorios con muchos elementos).

Se deben mostrar gráficamente alguno de los ejemplos límites y explicar su resolución. Finalmente se pide comparar los resultados esperados del estudio teórico con los experimentales. Para ello se elegirán grupos aleatorios (el mismo grupo para cada algoritmo) de (como mínimo) los siguientes tamaños: 200, 500, 1500, 5000. Con el fin de poder comprobar la efectividad de la implementación Divide y Vencerás, el alumno deberá elegir un tamaño de la nube de puntos cuyo tiempo de ejecución esté entre 2 y 3 minutos. Se comparará la ejecución de este tamaño en las dos versiones. Si se considera útil para la representación gráfica se pueden representar valores adicionales (nuevas ejecuciones).

Las representaciones gráficas del modelo teórico y práctico se superpondrán en una sola gráfica de líneas para cada algoritmo (dos gráficas en total).

Aplicación Práctica

Para comprobar el correcto funcionamiento el alumno deberá además añadirle al programa una funcionalidad que permita calcular las 2 ciudades más cercanas entre un conjunto de ciudades, cuyas coordenadas están en un fichero.

Para ello el programa deberá permitir cargar cualquier fichero de la biblioteca TSPLIB (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/) cuyo formato sea el mismo que el indicado en la Parte 2 de la práctica.