Práctica 1: Estrategias Algorítmicas

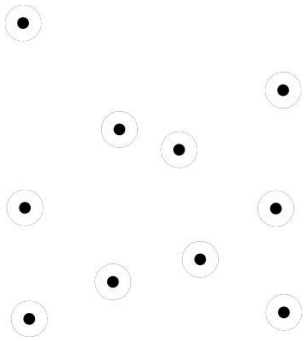
Objetivo.

El objetivo de esta práctica es desarrollar algoritmos para resolver dos problemas clásicos de búsqueda sobre conjuntos de puntos: la búsqueda del punto más cercano a otros dos y la búsqueda del árbol de conexiones mínimas. Para ello se plantearán estrategias de búsqueda exhaustiva, de Divide y Vencerás y de búsqueda voraz.

Parte 1 Análisis de algoritmos exhaustivos y Divide y Vencerás

El Problema del punto más cercano a otros dos.

Dado un conjunto de puntos situados en un plano P = { (x_1,y_1), (x_2,y_2), …, (x_n,y_n) } este problema consiste en encontrar (teniendo en cuenta que la distancia entre dos puntos i y j es $sqrt[(x_i-x_j)^2+(y_i-y_j)^2]$) el **punto** p_i **más cercano a otros dos** P = { p_j, p_k } = (x_i,y_i) tal que distancia((x_i,y_i), (x_j,y_j)) + distancia ((x_i,y_i), (x_k,y_k)) sea mínima, es decir, se debe encontrar de entre todos los puntos posibles cual es el que está más cerca de otros dos puntos (trío de puntos con la menor distancia entre sí).



Una primera solución es realizar una búsqueda exhaustiva analizando todas las combinaciones de 3 puntos posibles y seleccionando aquel con distancia mínima a los otros dos. Para un conjunto de n puntos existen n·(n-1)/2·(n-2)/3 combinaciones de 3 puntos, por lo que 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.

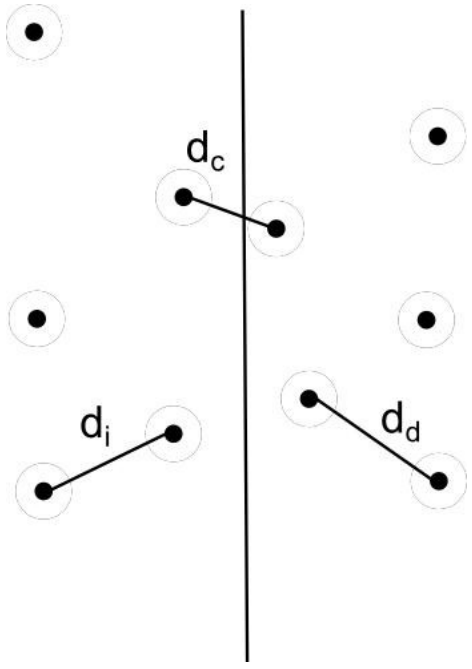
Esta primera solución plantea una búsqueda exhaustiva, pero existen algoritmos más rápidos para solucionar el problema, uno de los cuales aplica la técnica **Divide y Vencerás.**

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 punto más cercano p_s a otros dos está en P_i, o está en P_d, pero puede que uno o dos de los puntos más cercanos a dicho p_s esté en la zona contraria. 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 más cercano al buscado se encuentra en la otra zona.

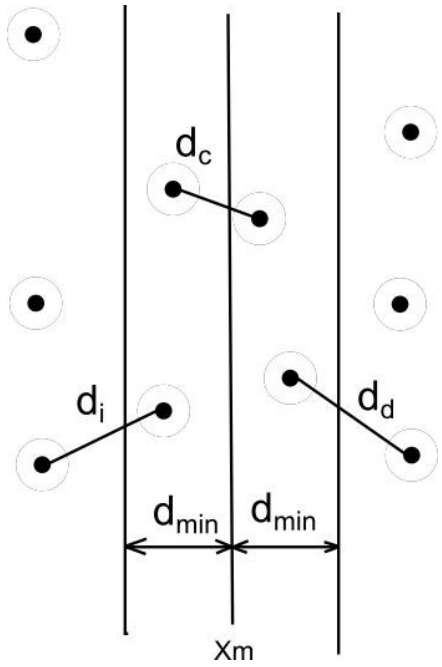
A continuación, se explica la solución para la búsqueda del punto más cercano a otro (distancia mínima de 2 puntos). Una vez entendido cómo se ha resuelto lo extrapoláis al caso del punto más cercano a otros dos (distancia mínima entre 3 puntos).

En el caso de hallar la distancia mínima de 2 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 punto más cercano a otro 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 y (en lugar de la x), lo que supone un tiempo de ejecución de **O(n·log n)**.

Ficheros de entrada

Para ejecutar los algoritmos propuestos se podrá utilizar tanto grupos de puntos aleatorios como grupos de puntos leídos de un fichero externo.

El programa deberá permitir cargar cualquier fichero de la biblioteca TSPLIB (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/) del repositorio TSP para problemas simétricos: *Symmetric traveling salesman problem* (TSP) (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/), los cuales presentan el mismo formato, una lista con dos valores para cada ciudad que representan sus coordenadas en el plano.

A continuación, se ofrece un ejemplo del formato de estos ficheros (en concreto, el fichero *burma14.tsp*).

```
NAME: burma14
TYPE: TSP
COMMENT: 14-Staedte in Burma (Zaw Win)
DIMENSION: 14
EDGE_WEIGHT_TYPE: GEO
EDGE_WEIGHT_FORMAT: FUNCTION
DISPLAY_DATA_TYPE: COORD_DISPLAY
NODE_COORD_SECTION
   1  16.47       96.10
   2  16.47       94.44
   3  20.09       92.54
   4  22.39       93.37
   5  25.23       97.24
   6  22.00       96.05
   7  20.47       97.02
   8  17.20       96.29
   9  16.30       97.38
  10  14.05       98.12
  11  16.53       97.38
  12  21.52       95.59
  13  19.41       97.13
  14  20.09       94.55
EOF
```

Desarrollo de la práctica

El alumno deberá desarrollar un proyecto en lenguaje JAVA que resuelva el problema de encontrar la ciudad más cercana a otras dos entre un conjunto de ciudades, dadas sus coordenadas en el mapa (las 3 ciudades más cercanas entre sí) mediante los algoritmos de búsqueda exhaustiva y de Divide y Vencerás explicados anteriormente.

El programa desarrollado deberá permitir seleccionar como entrada un conjunto de datos generados de forma aleatoria, o bien un conjunto de datos importados mediante la carga de cualquier fichero de la biblioteca TSPLIB. Para ello el programa debe proporcionar una opción de menú o una ventana de diálogo que permita al usuario indicar la ruta del fichero que desea abrir.

El resultado del proceso de búsqueda se debe mostrar en modo texto (tres campos que indiquen cuales son los puntos encontrados: el punto más cercano a otros dos) **y en modo gráfico** (por medio de un panel en el que se dibujen los puntos del data set y la línea que delimita la distancia del punto buscado a los otros dos).

Para desarrollar el algoritmo de ordenación de los puntos se deberá implementar o bien el algoritmo ***HeapSort o*** el algoritmo ***QuickSort*.**

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. Se comparará la ejecución de estos tamaños en las dos versiones (exhaustiva y divide y vencerás). 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).