Memoria Práctica 1

Alberto de la Torre Solis

Tabla de contenido

Divide Y Vencerás, Par más cercano

Explicación sobre la implementación

Para la realización del problema he decidido dividir el problema en 2 grupos con el mismo número de puntos. El motivo es que de esta forma garantizamos que para cada llamada recursiva el tamaño del subproblema sea de n/2, haciendo que el algoritmo tenga la máxima eficiencia.

Análisis Teórico

Para el análisis teórico se presenta el pseudocódigo de cada método y para cada trozo de código aparece el número de operaciones elementales que conlleva justo en la parte de arriba. Se ha tomado como una operación elemental los siguientes elementos:

- Asignación
- Comparación (aritmética o lógica)
- Return
- Llamada a función (loe más oe que ejecute la función propiamente dicha)
- Acceso a índices
- Operaciones aritméticas y lógicas
- Operaciones triviales del api de java

Las comparaciones con el índice del bucle, aunque no estén explicitas, existen y se contabilizan. Otro punto a tener en cuenta en los análisis es que para los bucles se calcula su coste en oe pero se le añade además el coste de la condición de parada, que es la comparación que hace en la última vuelta que ya no entra en el cuerpo del bucle. Estas oe se encuentran fuera del bucle correspondiente como es lógico ya que sólo se realiza una vez, no en cada pasada, sólo en la final. Al igual se contabiliza fuera del bucle la inicialización.

Comenzaremos analizando el método de búsqueda exhaustiva.

Calcular Perímetro → $T(n) \in \Theta(1)$

Búsqueda exhaustiva → $T(n) \in \Theta(n^3)$

Comparar Mínimo → $T(n) \in \Theta(1)$

Menor → $T(n) \in \Theta(1)$

Calcular Perímetro → $T(n) \in \Theta(1)$

Divide Y Vencerás → $T(n) \in \Theta(n \log n)$

Estrategia Voraz

Explicación sobre la implementación

En esta parte de la práctica no hay muchos aspectos relevantes de la implementación. Comentar que para la búsqueda exhaustiva se ha empleado un método recursivo para generar todas las posibles permutaciones del conjunto de ciudades y calcular las distancias, y para la búsqueda voraz se ha utilizado la estrategia de buscar siempre la ciudad no visitada más cercana.
Se ha implementado un array de ciudades visitadas para saber si una cuidad ha sido visitada en orden constante y no tener que revisar la secuencia. Si no el orden de la búsqueda voraz sería O(n3) en vez de O(n2) al tener que comparar 'n' elementos, y el tiempo de la búsqueda exhaustiva crecería mucho ya que sería O(n*n!) al tener que comparar 'n' elementos en cada permutación.
Cuando creamos la matriz le pasamos el número de ciudades que queremos para el problema y luego al pasársela a los algoritmos nos olvidamos de límites, porque la matriz está construida para las ciudades que queremos. En la matriz y en el array de visitados la posiciones 0 se ignoran para evitar errores al indexar, como las ciudades comienzan en 1 utilizaremos desde 1 a 'n', no desde 0 a 'n-1'.
En esta parte de la práctica no se ha incluido el coste de crear la matriz de distancias debido a que estos algoritmos se suponen que se ejecutan sobre grafos etiquetados con el coste entre ciudades donde no es necesario crear la matriz de distancias. Se ha supuesto que en esta práctica el paso de crear la matriz de distancias es exclusivamente para simular un grafo etiquetado. Aunque se hubiera incluido este coste el orden final de los dos algoritmos no variaría puesto que son iguale o mayores de O(n2), sólo que harían más operaciones.

Análisis de Complejidad

Prim → $T(n) \in \Theta(n^2)$

Distancia → $T(n) \in \Theta(1)$

Comparativa de resultados teóricos y experimentales, DyV

En este apartado vamos a mostrar las ejecuciones de los algoritmos para los 10 casos propuestos y a continuación se realizará la comparación de las gráficas para los resultados teóricos y experimentales. Utilizaremos para mostrar los resultados la salida del programa, donde BE será búsqueda exhaustiva y DYV será divide y vencerás. La ordenación de los puntos por el eje 'x' inicial sólo se contabiliza para medir el tiempo del algoritmo DYV ya que en el BE no es necesario ordenar ya que sería un sobrecoste innecesario extra debido a que realizará las mismas comparaciones estén o no ordenados. En cambio, en el DYV es requisito necesario ordenarlos.
El tiempo viene expresado en nanosegundos.

Caso 1: Límite

Punto 1 (5.0,8.0) Punto 2 (7.0.7.0)
BE > dist = 2.236068, tiempo = 19849891
DYV > dist = 2.236068, tiempo = 19009694
Se puede observar que apenas hay diferencia entre ambos algoritmos, aunque el DyV es ligeramente más rápido que la Búsqueda Exhaustiva.

Caso 2: Mejor

Punto 1 (0.0,5.0) Punto 2 (2.0,5.0)
BE > dist = 2.0, tiempo = 115067
DYV > dist = 2.0, tiempo = 92393
Al haber muy pocos puntos y estar ordenados, la BE da un resultado mejor que el DyV ya que no tiene que entrar en los bucles y nuestra Franja es de orden constante.

Caso 3: Peor

Punto 1 (4.0,0.0) Punto 2 (5.0,3.0)
BE > dist = 3.1622777, tiempo = 45515
DYV > dist = 3.1622777, tiempo = 121748
Al igual que antes, en tamaños pequeños la BE es más rapida que el DyV.

Caso 4: Aleatorio con 200 puntos

Punto 1 (902290.0,325320.0) Punto 2 (903794.0,328232.0)
BE > dist = 3277.4624, tiempo = 18727978
DYV > dist = 3277.4624, tiempo = 5651082
A medida que vamos agrandando la nube de puntos, el DyV será más eficiente que la BE.
En este caso, DyV es 2,79 veces más rápido que la BE.

Caso 5: Aleatorio con 500 puntos

Punto 1 (701246.0,634774.0) Punto 2 (702111.0,634175.0)
BE > dist = 1052.1531, tiempo = 24509629
DYV > dist = 1052.1531, tiempo = 5650305
Aquí es 4,33 veces más rápido que BE.

Caso 6: Aleatorio con 1500 puntos

Punto 1 (424891.0,976574.0) Punto 2 (425200.0,976250.0)
BE > dist = 447.72424, tiempo = 49957037
DYV > dist = 447.72424, tiempo = 9360212
Aquí es 5,33 veces más rápido que la BE.

Caso 7: Aleatorio con 5000 puntos

Punto 1 (905935.0,285009.0) Punto 2 (905959.0,285061.0)
BE > dist = 57.271286, tiempo = 251330323
DYV > dist = 57.271286, tiempo = 22695242
Aquí es 11,07 veces más rápido que la BE.

Caso 8: Aleatorio con 7000 puntos

Punto 1 (636068.0,149232.0) Punto 2 (635988.0,149288.0)
BE > dist = 97.65244, tiempo = 477098454
DYV > dist = 97.65244, tiempo = 10233277
Aquí es 46,62 veces más rápido que la BE.

Caso 9: Aleatorio con 10000 puntos

Punto 1 (354348.0,576611.0) Punto 2 (354331.0,576624.0)
BE > dist = 21.400934, tiempo = 970263180
DYV > dist = 21.400934, tiempo = 16607819
Aquí es 58,42 veces más rápido que la BE.

Caso 10: Aleatorio con 50000 puntos

Punto 1 (67914.0,411137.0) Punto 2 (67917.0,411145.0)
BE > dist = 8.5440035, tiempo = 24078467459
DYV > dist = 8.5440035, tiempo = 57925137
Aquí es 415,68 veces más rápido que la BE.

Caso 11: Aleatorio con 100000 puntos

Punto 1 (340134.0,62234.0) Punto 2 (340132.0,62233.0)
BE > dist = 2.236068, tiempo = 96954440711
DYV > dist = 2.236068, tiempo = 93118860
Aquí es 1041,19 veces más rápido que la BE.
En este caso mi ordenador se pasó unos minutos para poder terminar de realizar la operación.

Tabla de tiempos

En esta tabla se muestran los resultados obtenidos de las ejecuciones experimentales con los tamaños que se indicaban en el enunciado.

Puntos	Búsqueda Exhaustiva	Búsqueda DyV
100	109192	98893
1000	9739517	860427
10000	957112248	10134669

Conclusiones

Si no se hubiera realizado la mejora de la franja el DyV sería mucho más rápido que el BE, pero con la mejora introducida hemos conseguido que la franja sea de orden O(n*log(n)) en vez de orden $O(n^2)$. Para incrementos muy grandes, el BE experimenta un crecimiento muy elevado mientras que el DyV ni lo nota prácticamente. De ello podemos deducir que para tamaños más grandes más se notará la diferencia de rendimiento entre algoritmos como hemos podido observar en el caso de 100000 en el que el DyV es 1041 veces más rapido que el BE.

Comparativa de resultados teóricos y experimentales, Voraz

Tabla de tiempos

En esta tabla se muestran los resultados obtenidos de las ejecuciones. Adicionalmente a las del enunciado se han añadido 2 ejecuciones exhaustivas más para poder generar también una gráfica comparativa.

Ciudades	Búsqueda Exhaustiva(ns)	Distancia mínima
4	306403	638
8	23891670	1810
12	88505375758	2395

Ciudades	Búsqueda Voraz(ns)	Distancia mínima
130	720091	7511
280	3931870	3203
654	5117049	43395

Tabla de tiempos

En la Búsqueda Exhaustiva como ya se imaginaba el aumento de un caso a otro es muy grande, apenas visible aquí debido a que para tamaños mayores no es computable en un tiempo aceptable (como vemos para 12 ciudades tarda 1 min con 47seg, mientras que para 8 sólo eran 23 milisegundos). Si se fuera incrementando el tamaño al final se tendría que la curva formada sería casi una recta vertical (en una gráfica).
Para la Búsqueda voraz se formaría una curva exponencial cuadrática no muy pronunciada.