Algorítmica 2019/20
Práctica 3:
Greedy

Índice:

Realizado por:
Jorge Medina Romero
Alberto Robles Hernández
Ahmed Brek Prieto
Mohammed Lahssaini Nouijah

1-. TSP

1.1-.Planteamiento del problema.

El problema del viajante de comercio consiste en, dado un conjunto de ciudades, un viajante debe recorrer todas ellas exactamente una vez, regresando al punto de partida, de forma tal que la distancia recorrida sea mínima. Más formalmente, dado un grafo G, conexo y ponderado, se trata de hallar el ciclo Hamiltoniano de mínimo peso de ese grafo. Una solución para TSP es una permutación del conjunto de ciudades que indica el orden en que se deben recorrer. Para el cálculo de la longitud del ciclo no debemos olvidar sumar la distancia que existe entre la última ciudad y la primera (hay que cerrar el ciclo).

Candidatos = Ciudades situadas en las coordenadas x, y,
Usados = Conjunto de ciudades ya añadidas a la solución.
Solución = Vector con todas las ciudades anteriormente en candidatos, ordenadas
Criterio factibilidad = Que todas las ciudades candidatas estén en solución
Función objetivo = La distancia del camino que forma la solución.

1.2-.Algoritmo Greedy por cercanía.

Definición del algoritmo

Criterio selección = Se escoge la ciudad más cercana a la última ciudad de solución.

```
FUNCIÓN cercania(Ciudades)
        Solucion = Ciudades[0]
        MIENTRAS(Ciudades tenga elementos)
                Ciudad = BuscaCiudadMasCercana(Solucion[ultima],Ciudades)
                Solucion.Añadir(Ciudad)
                Ciudades.Elimina(Ciudad)
        END
        DEVOLVER(Solucion)
END
```

Utilizando el fichero de datos ulysses16.tsp, podemos observar que utilizando el algoritmo
basado en cercanía, no obtenemos una solución óptima.
La solución óptima es : 1 14 13 12 7 6 15 5 11 9 10 16 3 2 4 8
Basado en cercanía: 1 8 4 2 3 16 12 13 14 6 7 10 9 5 15 11

```
Camino
1 :: 38.24 :: 20.42
8 :: 37.52 :: 20.44
4 :: 36.26 :: 23.12
2 :: 39.57 :: 26.15
3 :: 40.56 :: 25.32
16 :: 39.36 :: 19.56
12 :: 38.47 :: 15.13
13 :: 38.15 :: 15.35
14 :: 37.51 :: 15.17
6 :: 37.56 :: 12.19
7 :: 38.42 :: 13.11
10 :: 41.17 :: 13.05
9 :: 41.23 :: 9.1
5 :: 33.48 :: 10.54
15 :: 35.49 :: 14.32
11 :: 36.08 :: -5.21
```

Eficiencia teórica.

En la función “calculaCaminoCercania”, que es la propia de este algoritmo, podemos observar 3 bucles. Dos de ellos están contenidos en un while, pero no anidados entre ellos. De ahí deducimos que la eficiencia es $O(n^2)$.

1.3-.Algoritmo Greedy por inserción.

Definición del algoritmo

Criterio selección = Se escoge la ciudad que al introducirla en la solución ya existente, en la posición que menos distancia añade al camino, aumenta menos el tamaño de la solución, y se inserta en dicha posición.

```
FUNCIÓN insercion(Ciudades)
        Solucion = CrearRecorridoParcial(Ciudades)
        MIENTRAS(Ciudades tenga elementos)
                        <Ciudad,Indice> = CiudadQueAñadeMenosRecorridoyDonde
                        AñadeMenosRecorrido(Solucion,Ciudades)
                Solucion.Insertar(Ciudad,Indice)
                Ciudades.Elimina(Ciudad)
        END
        DEVOLVER(Solucion)
END
```

Utilizando el fichero de datos ulysses16.tsp, podemos observar que utilizando el algoritmo
basado en inserción, no obtenemos una solución óptima.
La solución óptima es : 1 14 13 12 7 6 15 5 11 9 10 16 3 2 4 8
Basado en cercanía: 1 4 8 3 14 2 12 11 7 5 13 9 10 6 16 15

```
Camino
1 :: 38.24 :: 20.42
4 :: 36.26 :: 23.12
8 :: 37.52 :: 20.44
3 :: 40.56 :: 25.32
14 :: 37.51 :: 15.17
2 :: 39.57 :: 26.15
12 :: 38.47 :: 15.13
11 :: 36.08 :: -5.21
7 :: 38.42 :: 13.11
5 :: 33.48 :: 10.54
13 :: 38.15 :: 15.35
9 :: 41.23 :: 9.1
10 :: 41.17 :: 13.05
6 :: 37.56 :: 12.19
16 :: 39.36 :: 19.56
15 :: 35.49 :: 14.32
```

Eficiencia teórica.

En este caso, debemos estudiar la función “calculaCaminoInsercion”. En ella encontramos un bucle while que realizará n iteraciones, pues en cada una elimina una ciudad hasta que no queda ninguna. Este bucle realiza llamadas a insertar y eliminar ciudades (eficiencia O(n)), además de llamar a la función “siguienteCiudad”. Estudiando esta última, encontramos otro bucle con n iteraciones donde se llama a “encontrarMejorCamino”, que ejecuta un bucle de n iteraciones nuevamente donde todas sus sentencias son elementales, es decir, con eficiencia O(1). Con toda esta información, vemos que es un código equivalente a 3 bucles anidados, por lo que concluimos que tiene una eficiencia cúbica, $O(n^3)$.

1.4-.Algoritmo Greedy propio.

Definición del algoritmo

En este algoritmo, se calcula una solución según el criterio de cercanía. Ahora, realizamos una corrección sobre la solución obtenida. Buscamos la ciudad más aislada, es decir, la que tenga sus vecinos más alejados y la eliminamos del camino. A continuación, se estudia en qué posición del camino, su inserción supone un menor aumento de la distancia total del camino y se inserta en esa posición. Esta ciudad se marca como usada y continuamos con la siguiente, para no repetir la misma.
Criterio selección = Se escoge la ciudad peor situada (La que añade más distancia a la solución), y se inserta en la posición en la que añade menos distancia a la solución.

```
FUNCIÓN propio(D:Ciudades)
        Solucion = cercania(D)
        Usados = {false}

        FOR(i := 0 MIENTRAS i < numCiudades)
                Ciudad = BuscaCiudadMasAislada(Solucion,
numCiudades, usados)
                insertaPeorCiudad(Solucion, numCiudades, Ciudad)
        END
        DEVOLVER(Solucion)
END
```

Utilizando el fichero de datos ulysses16.tsp, podemos observar que utilizando el algoritmo basado en inserción, no obtenemos una solución óptima.
La solución óptima es : 1 14 13 12 7 6 15 5 11 9 10 16 3 2 4 8
Basado en cercanía: 1 8 16 4 2 3 5 15 13 14 12 6 7 10 9 11

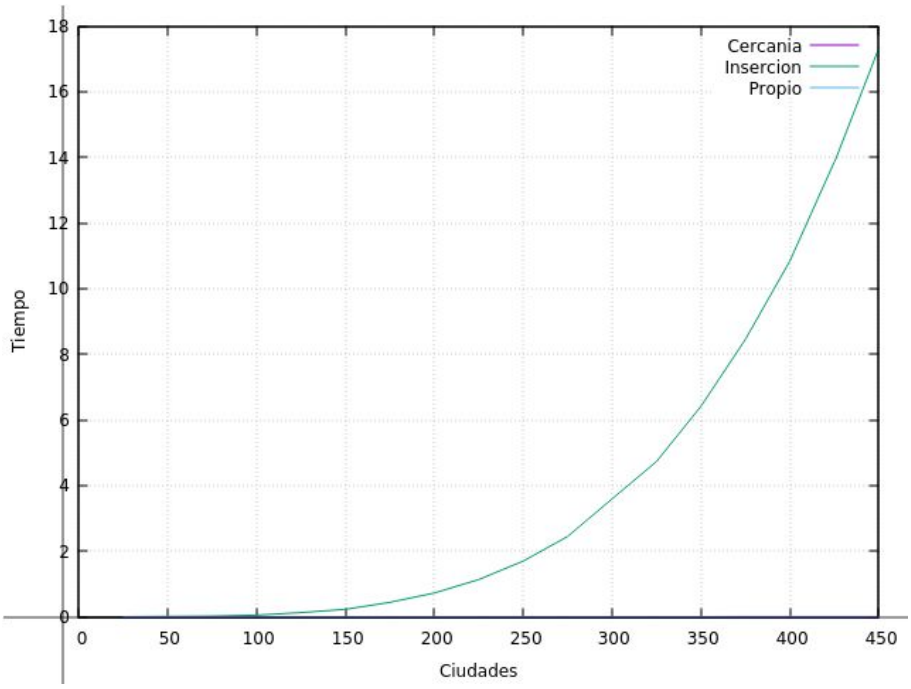
```
Camino
1 :: 38.24 :: 20.42
8 :: 37.52 :: 20.44
16 :: 39.36 :: 19.56
4 :: 36.26 :: 23.12
2 :: 39.57 :: 26.15
3 :: 40.56 :: 25.32
5 :: 33.48 :: 10.54
15 :: 35.49 :: 14.32
13 :: 38.15 :: 15.35
14 :: 37.51 :: 15.17
12 :: 38.47 :: 15.13
6 :: 37.56 :: 12.19
7 :: 38.42 :: 13.11
10 :: 41.17 :: 13.05
9 :: 41.23 :: 9.1
11 :: 36.08 :: -5.21
```

Eficiencia teórica.

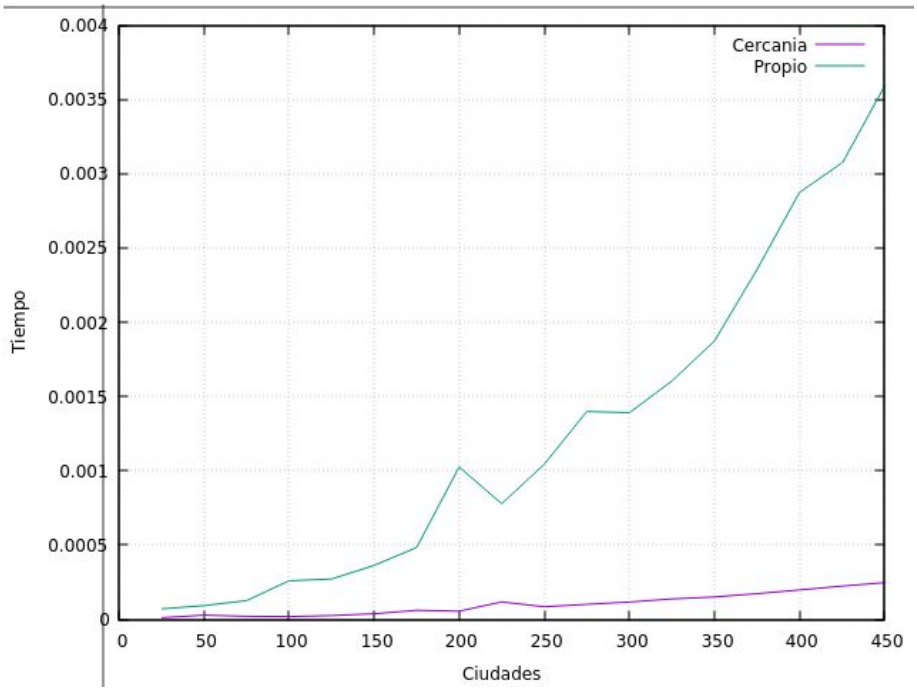
La función que implementa el algoritmo es “calculaCaminoPropio”. Si la analizamos, observamos que, además de algunas operaciones elementales, hay una llamada a “calculaCaminoCercania” y un bucle for por otro lado. Esta primera función, es la que calcula un camino por cercanía que, como hemos comprobado, es de orden cuadrático. Vayamos con el bucle. Vemos que en el se llama a “buscaCiudadMasAislada” y “insertaPeorCiudad”. Estas dos tienen un bucle for con n iteraciones; y en el caso de la segunda función, también se llama a dos funciones fuera del bucle pero ambas son de orden lineal. Con ello podemos deducimos que la eficiencia de este algoritmo es cuadrática, $O(n^2)$.

1.5-.Comparación de algoritmos.

Eficiencia empírica de los algoritmos.



Debido a la baja eficiencia del algoritmo de inserción no se pueden apreciar los otros dos, ya que el basado en inserción es $O(n^3)$ y los otros dos $O(n^2)$ por lo que al tomar valores grandes el algoritmo de inserción crece más rápidamente que el de cercanía y el propio.



Aquí podemos ver la comparación de las eficiencias empíricas del algoritmo de cercanía y el algoritmo propio. Como podemos observar, el algoritmo basado en cercanía es más rápido que nuestro algoritmo propio, pero la eficiencia asintótica es la misma en los dos, $O(n^2)$, por lo que la diferencia entre los dos algoritmos se basa en constantes multiplicativas.

2-. Contenedores en un barco

2.1-.Planteamiento del problema

Definición del algoritmo

El problema propuesto nos pide diseñar un algoritmo que nos permita elegir qué combinación de contenedores debemos de almacenar en un barco para maximizar alguna propiedad: ya sea el número de contenedores o el número de toneladas totales cargadas.

Candidatos = Contenedores disponibles para ser seleccionados, cada uno con un peso propio.

Usados = Conjunto de contenedores ya elegidos para ser cargados.
Solución = Vector con todos los contenedores que han sido seleccionados y ordenados para ser cargados.
Criterio factibilidad = Que no se supere el límite de peso que puede cargar un barco, ni quepan más contenedores que cumplan el criterio elegido.
Función objetivo$_1$ = Número de contenedores cargados.
Función objetivo$_2$ = Número de toneladas cargadas.
El criterio de selección utilizado depende del algoritmo en el que nos encontremos.

2.2-.Maximizar el número de contenedores

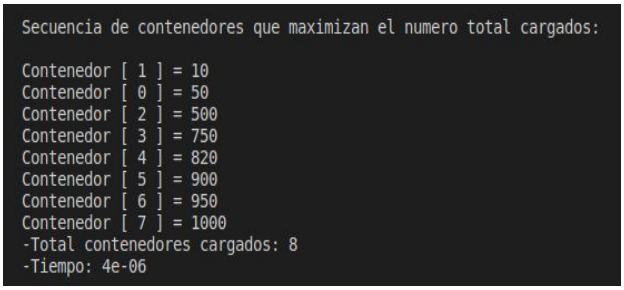
Definición del algoritmo

Criterio selección = Se escoge el contenedor menos pesado de entre los disponibles.

```
FUNCIÓN maximizaNumContenedores(Contenedores)
        Solucion = Contenedores[0]
        MIENTRAS(Contenedores tenga elementos)
                Contenedor = BuscaContenedorMenosPesado(Contenedores)
                Solucion.Añadir(Contenedor)
                Contenedores.Elimina(Contenedor)
        END
        DEVOLVER(Solucion)
END
```

El criterio elegido para este algoritmo es seleccionar el contenedor de menor peso de entre los disponibles en cada momento, para que sea así un algoritmo Greedy. Esto nos garantiza una solución óptima, pues si en cada momento introducimos el contenedor de menor tamaño, dejamos el mayor espacio posible para el resto de contenedores y, cuando no quede espacio para el siguiente contenedor de menor tamaño, sabremos que hemos introducido todos los más pequeños y por tanto el mayor número posible de estos.

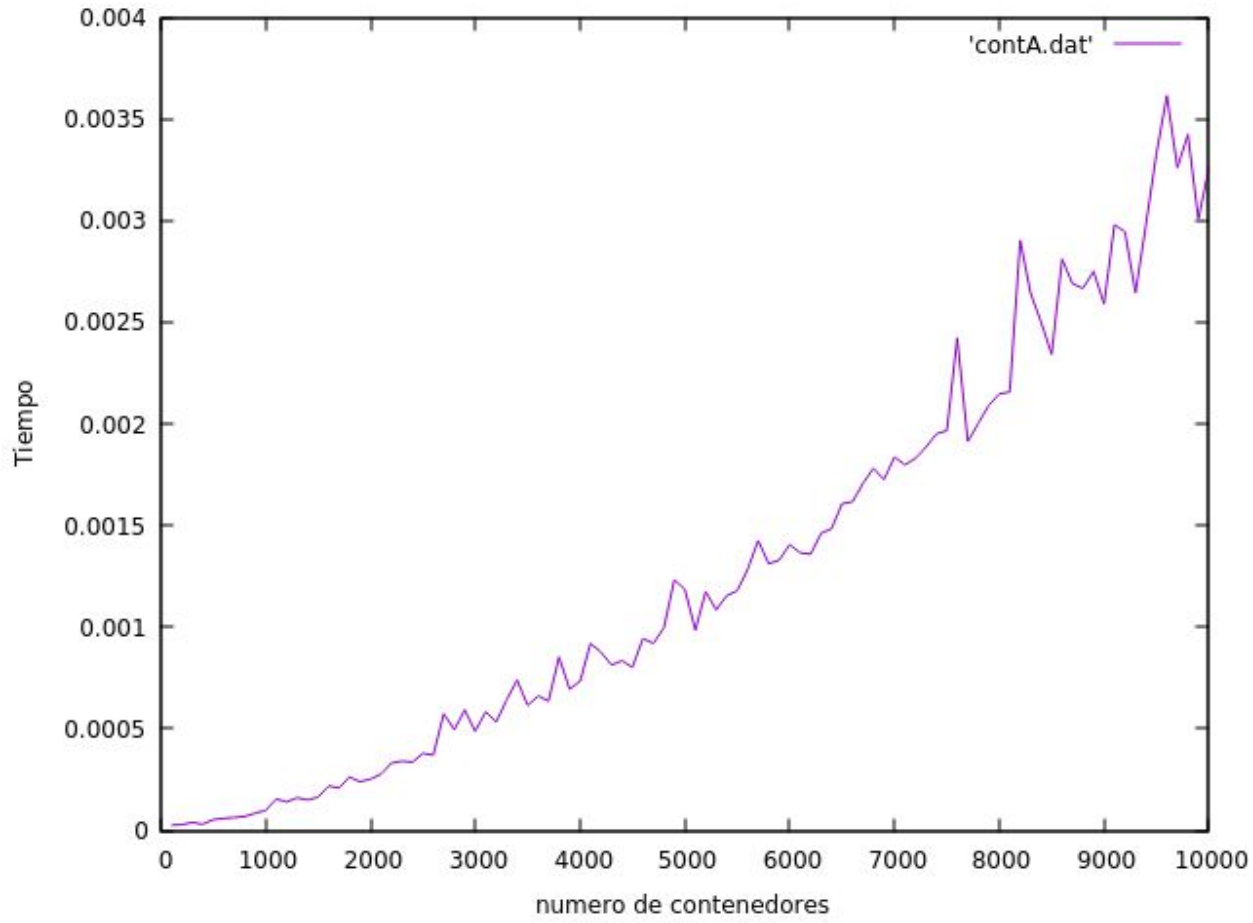
Utilizando el fichero de datos “filedata”, podemos observar que el algoritmo devuelve:
Índices: 1 0 2 3 4 5 6 7
Total contenedores: 8



Eficiencia teórica

El estudio de la eficiencia teórica del algoritmo implementado es sencillo. Si observamos el código, encontramos que en la función “SolucionCaminoContenedores”, que es la que implementa el algoritmo en sí, no hay llamadas a funciones externas y que todo son operaciones elementales. Según el criterio que se elija para maximizar un el número de contenedores o toneladas, se ejecutará una parte del código u otra, aunque podemos ver que su estructura es exactamente igual, variando el criterio de comparación. Por tanto, basta con estudiar una de ellas y podremos suponer que ambas eficiencias son iguales. Pues bien, analizando el código encontramos dos bucles for anidados, que en el peor de los casos ejecutarán n iteraciones cada uno. Deducimos de forma trivial que la eficiencia es de orden cuadrático, orden $O(n^2)$.

Eficiencia empírica



2.3-.Maximizar el número de toneladas

Definición del algoritmo

Criterio selección = Se escoge el contenedor más pesado de entre los disponibles.

```
FUNCIÓN maximizaNumToneladas(Contenedores)
        Solucion = Contenedores[0]
        MIENTRAS(Contenedores tenga elementos)
                Contenedor = BuscaContenedorMasPesado(Contenedores)
                Solucion.Añadir(Contenedor)
                Contenedores.Elimina(Contenedor)
        END
        DEVOLVER(Solucion)
END
```

Este problema presenta diferencias sustanciales sobre el anterior. Ahora, la propiedad que se intenta maximizar es el número de toneladas que carga el barco. El criterio utilizado es introducir en cada momento el contenedor con mayor peso de entre los disponibles, ya que es la forma en la que más rápidamente nos acercaremos al peso máximo, es decir, el objetivo. Cabe destacar que este criterio no nos garantiza la solución óptima. Esto se demuestra fácilmente con un contraejemplo. Supongamos una capacidad del barco de 100 toneladas; 2 contenedores de 50 toneladas y otro de 70. Según nuestro criterio, se elegiría el de mayor peso, es decir, el de 70 toneladas, y la búsqueda acabaría ya que no caben más contenedores de entre los disponibles. Observamos que la solución óptima no es la que devuelve el algoritmo, sino que sería tomar los dos contenedores de 50 toneladas.

Llegados a este punto podríamos plantearnos si quizás el criterio a seguir más adecuado podría ser el mismo que para el problema anterior: tomar el contenedor de menor peso en cada caso. De nuevo, observamos que no devolvería la solución óptima en el caso de que un barco pueda cargar con 100 toneladas, y se dispongan de 2 contenedores de 10 toneladas y uno de 90.

Por tanto, hemos optado por el primer criterio por la mera razón de que se acerca antes al objetivo, como ya se ha explicado, teniendo así cierta actitud “voraz”, propia de los algoritmos Greedy. Y en caso de que no dé la solución óptima, es mucho más probable que la solución que devuelva sea mejor que si tomamos el criterio de elegir el contenedor menos pesado.

Utilizando el fichero de datos “filedata”, podemos observar que el algoritmo devuelve:
Índices: 12 7
Total contenedores: 2

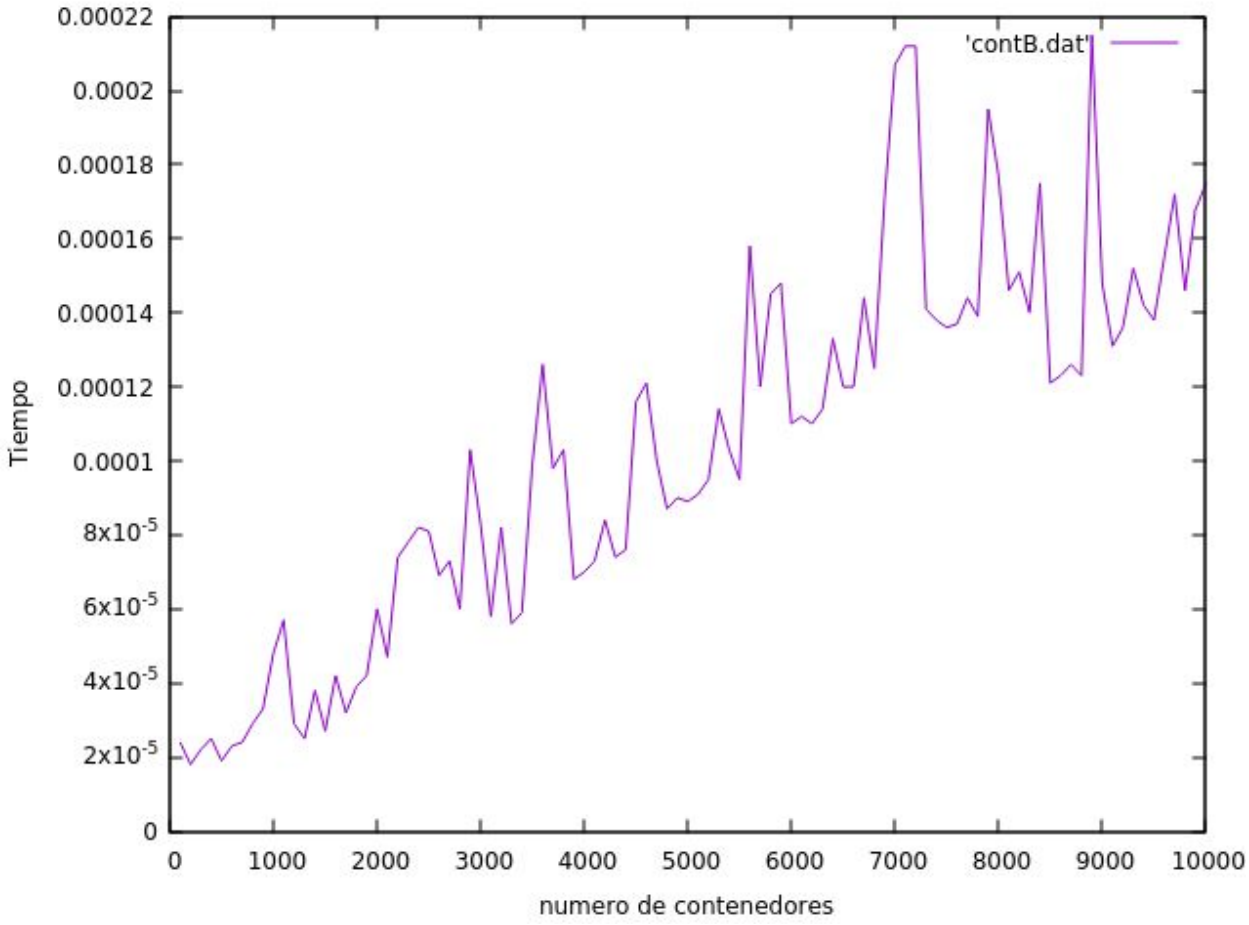
```
Secuencia de contenedores que maximizan el tonelaje total cargado:

Contenedor [ 12 ] = 4000
Contenedor [ 7 ] = 1000
-Total contenedores cargados: 2
-Tiempo: 2e-06
```

Eficiencia teórica

Como se ha explicado anteriormente, la eficiencia de los dos algoritmos es la misma, por lo que ya sabemos que es de orden $O(n^2)$, según hemos estudiado.

Eficiencia empírica



En la gráfica se aprecian numerosos picos, que podrían considerarse irregularidades, aunque son perfectamente normales. Se deben a la aleatoriedad en la generación del tamaño de los contenedores. Si en un problema con un barco se generan mil contenedores y uno de ellos ocupa el 99% del tamaño de un barco, por ejemplo, el problema terminará más pronto que si los contenedores generados son más pequeños. Aún así, se observa que, por lo general, el tiempo es directamente proporcional al número de contenedores utilizados.