Proyecto Algoritmia

Índice

Implementación voraz

La implementación voraz se ha realizado sobre el problema de distribución, que consiste en distribuir un componente desde dos fábricas (A y B) hasta *N* fábricas teniendo en cuenta el coste de envío, que varían dependiendo el almacén de origen y la fábrica destino.

Funcionamiento del algoritmo

El algoritmo se basa en ordenar los objetos de la forma que sea mas optimos empezar a cogerlos, de esta forma intentamos minimizar el coste de envío del almacén a sus fábricas respectivas.

Para minimizar ordenamos los elementos dependiendo de su número * su coste de envío, tenemos en cuenta cual es el almacén con mayor coste y ordenamos a partir de ese.

Solución encontrada

La solución encontrada es a partir de la toma de decisiones que hace el algoritmo no se puede garantizar que siempre encuentre la solución óptima, y tampoco sabemos cómo de lejos estamos de la solución óptima, por lo tanto este algoritmo obtendría una solución heurística ya que solo encuentra una.

Coste espacial y temporal

Coste temporal como el temporal en este caso son de O(|N| lg|N|), los dos tienen el mismo coste.

El lg |N| lo obtenemos de la ordenación que hacemos para poder sacar una solución más óptima, y el |N| es el resultante de recorrer el vector para hacer los cálculos necesarios para obtener la solución.

Implementación Búsqueda con retroceso

La implementación de la búsqueda con retroceso se basa en el problema de inversión, que consiste en tomar una decisiones para obtener unas ganancias en N meses que nos dice el enunciado.

Conjunto de soluciones factibles

El conjunto de soluciones factibles es la elección de una de las 3 combinaciones que podemos elegir, siempre sin rebasar N meses que tenemos. La opción A no implica ninguna inversión y tiene una duración de 1 mes, la opción B implica una inversión y tiene una duración de 1 mes, y la opción C tiene una inversión más segura y una duración de 6 meses. La solución final el dinero obtenido en total después de las inversiones y la tupla de las elecciones que hemos realizado.

$$\times = \{(x_0, x_1, ..., x_{n-1}) \varepsilon (A, B, C)\} \mid \sum_{1 \le i \le n} (k - T_i) B_i = Q$$

El Conjunto de estado son los diferentes tipo de inversión que puedan hacer.

K simboliza el dinero que tenemos
T simboliza la tasa que se le aplica a la elección tomada
B simboliza el beneficio que de invertir en ese mes

Funcionamiento del algoritmo

El algoritmo resuelve el problema mediante el `BacktrackingOptSolver`, existen diferentes tipos de Backtracking para resolver el problema, he elegido el Opt, por que una vez realizado varias pruebas se ha demostrado midiendo el tiempo que su tiempo para resolver el problema es menor respecto a los demás, además sólo devuelve los valores positivos, desde 1 hasta la solución obtenida.

El método *successors* realiza una comprobación básica antes de empezar, para asegurarse que aún no ha terminado. Después se toman las 3 decisiones mediante el yield , que se invierte en el opción A , seguido la B y para invertir en la C se tiene en cuenta si hay suficientes meses disponibles porque la C tiene una duración de 6 meses

Coste espacial y temporal de successors

EL coste temporal del método successors es O(|1|) es un método sencillo que comprueba las 3 opciones que hay. En Cambio por otro lado el coste espacial es exponencial porque comprueba todas las combinaciones posibles, y en cada una el vector de soluciones puede llegar a alcanzar un tamaño igual al número de meses.

Implementación Programación Dinámica

La implementación de programación dinámica se basa en resolver el problema de subsecuencia ascendente. Este problema consiste en devolver una secuencia de posiciones de *L* encontradas en posiciones crecientes sin repetirlas.

Conjunto de soluciones factibles, función objetivo y función recursiva

Disponemos de dos listas de enteros y queremos encontrar la subsecuencia que hay entre ellas, X lista original, Y lista ordenada de menor a mayor.

Conjunto de soluciones factibles

Los índices $i_1, i_2, ..., i_{n-1}$ *hacen referencia a los enteros de la lista X* y los índices $j_1, j_2, ..., j_{n-1}$ *hacen referencia a los enteros de la lista Y*

- $1 \leq i_l \leq m \; y \; 1 \leq j_l \leq n$, *para todo* $1 \leq l \leq k$,
- $i_l \leq i_{l+1} \; y \; j_l < j_{l+1}$ *para todo* $1 \leq l \leq ky$
- $x_{il} = y_{il}$ *para todo* $1 \leq l \leq k$.

Función objetivo

$\times = \{(x_0, x_1, ..., x_{n-1}) \; \varepsilon \; (Z^{\geq 0})^N\} : max(x_0, x_1, ..., x_{n-1})$

El objetivo es obtener la solución que tenga una mayor cantidad de enteros, es decir, la secuencia más larga

Función recursiva

$$solution(i,j) = \begin{cases} 0, & si \; i = 0 \; o \; j = 0 \\ max\{solution(i-1,j), Solution(i-j-1\},) & si \; i > 0, \; j > 0 \; y \; x_i \neq y_i \\ solution(i-1,j-1)+1, & si \; i > 0, \; j > 0 \; y \; x_i = y_i \end{cases}$$

Funcionamiento del algoritmo

El algoritmo se basa en la obtención de una secuencia ascendente, para esto he llevado a cabo la comparación de dos listas, la primera es la que obtenemos del fichero que leemos que es la base, y la segunda es la primera lista que hemos obtenido pero ordenada mediante la función sorted(). De esta forma al proporcionar dos listas intentó resolver el problema obteniendo una subsecuencia común entre ambas listas, utilizando de base y devolviendo los índices que la primera.

El algoritmo además de conseguir la subsecuencia más larga, tenemos que obtener el camino de recuperación, para devolver los índices. El método ***subsecuencia*** es quien realiza la búsqueda de la secuencia. Tenemos dos matrices una para comprobar los que vamos visitando y cuantos hemos encontrado actualmente, se llama visitados, y la otra matriz que es para almacenar en que posiciones hemos encontrado la coincidencia, esta segunda matriz está formada por tuplas, que nos indican en esta tupla a la próxima posición a la que debemos visitar y guardar el índice *i-1* que es la posición del vector inicial que queremos guardar. He utilizado una matriz para almacenar la recuperación de camino porque al ser *NxM* puede llegar al caso que sea muy grande y con un diccionario para almacenar valores, en las pruebas realizadas su tiempo se incrementa por el uso de memoria, por ello he utilizado una matriz para resolverlo.

Para obtener las subsecuencia recorremos el vector *X* y vamos haciendo las 3 comprobaciones pertinentes:
El if es cuando encontramos el valor deseados, que incrementamos el valor y guardamos la posición. El elif,else es para saber cual es mayor porque no podemos hacer uso de la función max, porque tenemos que almacenar el anterior en la matiz histórico.

```
if x[i-1] == y[j-1]:
      visitados[i][j], historico[i][j] = visitados[i-1][j-1] + 1, (i-1, j-1)

elif visitados[i][j-1] > visitados[i-1][j]:
      visitados[i][j], historico[i][j] = visitados[i][j-1], (i,j-1)

else:
      visitados[i][j], historico[i][j] = visitados[i-1][j], (i-1,j)
```

Coste espacial y temporal del algoritmo

El coste espacial como temporal del algoritmo es O(|M| |N|) por el simple hecho que resolvemos el problema mediante una matriz de tamaño M x N, así que como mayor sea la lista de enteros que tenemos que tratar mayor será su coste. En este caso en concreto ambas cadenas tendrán el mismo tamaño puesto que una es tal cual la del fichero y la otra es la misma pero ordenada, entonces podemos decir que su coste realmente es O(|M| |M|) o lo que es lo mismo O $(|M^2|)$.

Bibliografía

- Diapositivas del Tema 3, Algoritmos voraces
- Diapositivas del Tema 4, Búsqueda en retroceso
- Diapositivas del Tema 7, Programación dinámica
- Libro de algoritmia, Capítulo 8, Programación Dinámica