Laboratorio 1: Fuerza Bruta

Javier Arredondo Contreras

Universidad de Santiago de Chile
javier.arredondo.c@usach.cl

14 de noviembre de 2018

Resumen

El presente artículo muestra la implementación de un algoritmo presentado por el cuerpo docente de la asignatura Algoritmos Avanzados de la Universidad de Santiago de Chile. Para dicho problema se aplican métodos y técnicas utilizadas por fuerza bruta para la optimización de la problemática. El objetivo del estudio de este documento se basa principalmente en analizar los resultados obtenidos para luego profundizar en la eficacia y eficiencia de la solución propuesta.

I. Introducción

La asignatura de Algoritmos Avanzados, impartido por el Departamento de Ingeniería Informática tiene como objetivo que los estudiantes sean capaces de analizar en profundidad distintos tipos de algoritmos. Dentro de esta primera experiencia se busca aplicar el diseño algorítmico de soluciones que brinda *Fuerza Bruta*; el cual dice que se deben buscar todas las posibles soluciones, evaluar cada una de ellas y finalmente escoger la solución que se adecue a una serie de requisitos del problema.

En base a este documento se ha presentado un problema por parte del cuerpo docente; el cual es brevemente explicado a lo largo del documento.

i. Objetivo principal

- Desarrollar una algoritmo que sea capaz de resolver el problema planteado, a través del uso de fuerza bruta.

ii. Objetivos secundarios

- Obtener el listado de inversiones que maximice los beneficios para la empresa *Clovers Inversions*.
- Responder a las preguntas de análisis de un algoritmo: ¿Se detiene?, cuando se detiene ¿entrega una solución correcta?, ¿es eficiente?, ¿se puede mejorar? y ¿existe otro método?.
- Para obtener la eficiencia de la solución, se debe estimar el orden de complejidad del algoritmo desarrollado utilizando la notación $O()$.

iii. Estructura del documento

En primer lugar se presenta la descripción y contexto del problema, para luego exponer un breve marco teórico que ayuda a comprende los conceptos claves del documento. Posteriormente se realiza un análisis para el algoritmo, respondiendo las preguntas planteadas para así estudiar la eficiencia de dicho algoritmo.

II. Descripción del problema

El problema propuesto por el cuerpo docente corresponde en optimizar los beneficios de las inversiones de la empresa *Clovers Inversions*, esta problemática nace debido a que en este último tiempo han nacido una gran cantidad de nuevas empresas y por consecuencia la lista de inversiones a crecido de forma exponencial. El problema radica en que dado un capital inicial, se pueda maximizar la utilidad de las inversiones; para esto se requiere el listado de inversiones y la cantidad de inversiones que la

empresa está dispuesta a realizar.

La solución del problema está centrada en buscar la canasta de inversiones que logren maximizar los beneficios de las inversiones.

III. Marco teórico

A continuación se describen conceptos utilizados a lo largo del documento:

- Algoritmo: Cierta cantidad de instrucciones que permiten encontrar la solución a alguna problemática.
- Fuerza bruta: Resolución de problemas mediante encontrar todos los posibles candidatos para la solución. Después de encontrar todas las posibles combinaciones, se procede a buscar la que cumpla ciertos requisitos dados por el problema.
- Orden de complejidad: Estimación del tiempo de ejecución de un algoritmo.
- Eficiencia de un algoritmo: Serie de propiedades de un algoritmo, que están relacionados con el tiempo de ejecución y la utilización de recursos que este haga.
- Problema de la mochila: En la teoría de algoritmos, este problema es recurrente y consiste en la optimización de una combinatoria de elementos. Principalmente busca la mejor solución entre un conjunto finito de posibles soluciones.

IV. Descripción de la solución

Idea: Generar todas las posibles combinaciones de inversiones posibles, posteriormente en el conjunto solución se escoge la canasta de inversiones óptima.

1400000	
5	
250000	1000
600000	6000
450000	4000
300000	2000
100000	4500

Cuadro 1: *Ejemplo de archivo de entrada.*

En el contexto del problema, se entrega como entrada un archivo que posee el capital inicial de la empresa *Clovers Inversion*, la cantidad de inversiones que esta quiere hacer y un listado de las posibles inversiones que se pueden realizar, tal como se muestra en el Cuadro 3.

En primera instancia se separó en pequeños subproblemas cada componente del problema general, a continuación se explica cada subproblema encontrado y su respectiva solución.

I. Lectura del archivo

Para la lectura del archivo se implemento una estructura de datos; una lista. Para así poder almacenar la información de la entrada en dicha estructura y posteriormente procesarlos.

Operaciones implementadas para la lista:

- *append()*: Agrega un elemento al final de la lista.
- *pop()*: Quita el último elemento de la lista.
- *showList()*: Muestra cada elemento de la lista.

II. Encontrar canasta de inversiones óptima

Este subproblema sigue siendo muy general, por lo cual se tuvo que dividir en otros subproblemas que sean abordables y se logren resolver.

1. **Implementación de estructura *knapsack***

   Para una mayor abstracción del problema se ha creado la estructura de datos que corresponde a una mochila (*knapsack* en inglés), con esta estructura se logra manipular el problema, ya que se puede crear todas las posibles combinaciones de elementos que se encuentran en una lista y cada combinación se guarda en una mochila. Finalmente a partir de las restricciones del problema escogemos la mochila más adecuada. Cada mochila tiene los siguientes atributos: capital invertido, capacidad restante, cantidad de elementos contenidos, beneficio total y listado de inversiones. Las operaciones implementadas para ésta estructura de datos son:

- *add()*: Agrega un elemento a la mochila. Se actualiza: capital invertido, beneficio total, capacidad y tamaño de la mochila.
- *showKnapsack()*: Muestra cada elemento en la mochila y los atributos de la mochila.

2. **Generar todas la combinaciones posibles**

   En primera instancia se tuvo que calcular la cantidad de combinaciones posibles, con la expresión 2^n; donde 2 corresponde a las posibilidades de que el elemento esté en la mochila; en lenguaje científico 0 significa que el elemento no esté y 1 que el elemento esté (solo existen estas 2 posibilidades), por otro lado n corresponde a las inversiones disponibles.

   Luego, se generan las 2^n combinaciones posibles; para esto se ha utilizado el sistema binario, en donde se ha convertido en número binario desde 0 hasta 2^n, esto para representar todas las combinaciones posibles. Por ejemplo con un $n = 3$, las combinaciones posibles serían las que se muestran en el Cuadro 2, donde 0 y 1 representan si el elemento está presente está o no.

   Finalmente, con estas combinaciones posibles se crean todas las mochilas correspondientes, por lo cual se van agregando todos los 1 presentes en la combinación; esta forma garantiza que se creen todas las mochilas posibles, desde la mochila vacía hasta la mochila que tiene todas las inversiones.

3. **Encontrar mochila óptima**

   Cuando se está generando cada mochila posible, se busca la que cumpla con los requisitos de entrada; en esta búsqueda la estructura *knapsack* es de gran ayuda, ya que se busca la que tenga atributos adecuados y no se calcula nada nuevo. Se almacena en una variable auxiliar la mochila parcialmente óptima. Al finalizar este ciclo se obtiene la mochila óptima que cumple con los requisitos. Finalmente, la mochila óptima es escrita en un archivo de salida.

n	Combinación
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Cuadro 2: *Combinaciones posibles con $n = 3$.*

V. Análisis de la solución y resultados

1. **¿El algoritmo se detiene?** Efectivamente se detiene, ya que cuando genera todas las posibles combinaciones, se establece la mochila óptima y finaliza con el almacenamiento de esta en un archivo.

2. **¿Resuelve el problema?** Si resuelve el problema, ya que al terminar de generar las posibles mochilas, se escoge la que optimice los beneficios de las inversiones.

3. **¿Es eficiente?** No, esto debido a que se generan combinaciones que no sirven, como por ejemplo: que gasten más del capital inicial, conjunto vacío, entre otros. Esto podría obviar y generar menos combinaciones, ya que al generar todas las combinaciones posibles aumenta el orden de complejidad.

   Para este algoritmo, se tiene el siguiente orden de complejidad por funcionalidad:

   - ***getKnapsack()***: $O(1)$, ya que lee desde el archivo de forma imperativa las condiciones a respetar.
   - ***getInversion()***: Lectura de cada inversión del archivo de entrada.
     - *initList()*: $O(1)$, ya que solicita memoria a la máquina.

- resto de *getInversion()*: $O(n)$, ya que lee n líneas del archivo de entrada.
- Total: $O(1) + O(n) \sim O(n)$

- ***bruteForce()***: Algoritmo de fuerza bruta para el problema.
  - *myPow()*: $O(n)$, realiza la potencia, multiplica n veces el término de la base.
  - *makeCombinatory()*: $O(n^2)$, convierte el n números en binario, donde el costo de transformar cada número es m.
  - resto de *bruteForce()*: $O(n^2)$, ya que para cada combinación de los n números binarios generados, se verifica dígito a dígito m si es un 1, de ser así se agrega dicho elemento a la mochila. Luego están las instrucciones para verificar si la mochila generada cumple con los requisitos óptimos, para así ser el óptimo parcial.
  - Total: $O(n) + O(n^2) + O(n^2) \sim O(n^2)$
- ***writeOutput()***: $O(n)$, ya que se escriben n líneas en el archivo de salida.
- **Total algoritmo**: $O(1) + O(n) + O(n^2) + O(n) \sim O(n^2)$

4. **¿Se puede mejorar?** Si, por ejemplo no generando las soluciones que no cumplan con los requisitos; el conjunto vacío y las que generen un capital invertido negativo.

5. **¿Existen otros métodos?** Si, utilizando alguna otra estrategia de resolución, como por ejemplo *backtracking*.

VI. Traza

Sea un capital inicial de 500 unidades monetarias, se requieren 2 inversiones y se tienen las inversiones presentadas en Cuadro 3.

Inversión	200	150	450
Beneficio	4	7	10

Cuadro 3: *Entrada para traza.*

Sea n la canasta analizada. I1, I2 e I3 las inversiones previstas, B el beneficio total, K el capital invertido en dicha canasta y O si es óptimo o no. Se realiza la traza de las canastas en Cuadro 4.

n	I1	I2	I3	B	K	O
0	0	0	0	0	0	No
1	0	0	1	10	450	No
2	0	1	0	7	150	No
3	0	1	1	17	600	No
4	1	0	0	4	200	No
5	1	0	1	14	650	No
6	1	1	0	11	350	Si
7	1	1	1	21	800	No

Cuadro 4: *Traza para 3 inversiones.*

VII. Conclusiones

El algoritmo encuentra la solución buscada por la empresa *Clovers Inversions*, es decir, se encuentra la canasta de inversiones que optimice el beneficio dado un listado de inversiones. Además se cumplen con los objetivos planteados por el cuerpo docente, ya que se ha utilizado fuerza bruta para la composición de la solución del problema.

El algoritmo propuesto responde las preguntas de análisis de algoritmo vistas en cátedra, en cuanto a la eficiencia del algoritmo, se puede decir que el algoritmo es eficiente, dado que la complejidad de este es polinómica $O(n^2)$. Aún así el algoritmo tiende a demorarse mucho más cuando se tiene un mayor n; dado que este crece de manera cuadrática. Esto se debe a que el algoritmo genera las combinaciones que de cierta manera sirven y los que no sirven.

Una de las formas de mejorar el algoritmo sería básicamente no generar soluciones inútiles, ya que como se ha mencionado en el análisis, estas no sirven para el conjunto solución.

Con esta experiencia, se ha podido evidenciar que fuerza bruta cumple con encontrar soluciones óptimas, pero en cierto rigor no es

recomendable, ya que requiere un cómputo exhaustivo e innecesario, sin embargo muchas veces resulta más fácil de implementar que otras estrategias más eficientes.

Referencias

[M. Villanueva, 2002] Algoritmos: Teoría y Aplicaciones