Como obtener las inversiones que garantizan el máximo de utilidad con programación dinámica.

Alberto Rodríguez Zamorano
Alberto.rodriguez.z@usach.cl
https://github.com/Albertorz31

Algorithm design: Dynamic Programming.

RESUMEN: El siguiente documento consiste en la resolución de un problema por medio de la implementación de Programación dinámica. El problema que trata el documento es el poder encontrar las inversiones con tal de garantizar un máximo de utilidad para una empresa de inversiones. El documento trata el problema mediante la creación de un algoritmo que recibe de entrada un archivo, el cual contiene la capital inicial, el número de inversiones disponible, y las líneas siguientes muestra el costo y la utilidad de cada inversión. Por esto se debe entregar un archivo de texto en el que muestre el capital invertido, junto al beneficio máximo que se puede obtener, tambien se muestran las inversiones en las que se invirtió y se logró ese beneficio. Para su solución, se crea una lista que contiene las inversiones, después para aplicar la programación dinámica, se utiliza un árbol binario, en el cual en cada nivel se va comparando cuál de los dos hijos es el de mayor beneficio hasta llegar al nodo raíz que verá cuál de los dos hijos tiene mayor beneficio y se creará una lista con esas inversiones. El algoritmo es recursivo y se descompone en 2 subproblemas y el tiempo para definir el subproblema es O(1), el algoritmo tiene una complejidad de O(2^n). Al no ser la complejidad un polinomio se puede decir que el algoritmo no es eficiente, aunque para este caso, sería el algoritmo más eficiente para obtener la solución óptima de este problema.

PALABRAS CLAVE: Lista, algoritmo, capital, inversión, nodo, costo, utilidad.

ABSTRACT: The following document consists in the resolution of a problem through the implementation of Dynamic Programming. The problem dealt with in the document is to be able to find the investments in order to guarantee maximum utility for an investment company. The document addresses the problem by creating an algorithm that receives input from a file, which contains the initial capital, the number of available investments, and the following lines shows the cost and utility of each investment. For this reason, a text file must be submitted showing the capital invested, along with the maximum benefit that can be obtained, as well as the investments invested, and that benefit was achieved. For its solution, a list containing the investments is created, then to apply dynamic programming, a binary tree is used, in which at each level it is compared which of the two children is the most beneficial until reaching the root node You will see which of the two children has the greatest benefit and a list of those investments will be created. The algorithm is recursive and is broken down into 2 subproblems and the time to define the subproblem is O (1), the algorithm has a complexity of O (2 ^ n). Since the complexity is not a polynomial, it can be said that the algorithm is not efficient, although for this case, it would be the most efficient algorithm to obtain the optimal solution for this problem.

KEYWORDS: List, algorithm, capital, investment, node, cost, utility.

1 INTRODUCCIÓN

En la asignatura de Algoritmos Avanzados, la temática que se trata es la creación de algoritmos de poca complejidad para resolver problemas. El documento presente busca la resolución de un problema por medio de Programación Dinámica, cuyo problema trata de garantizar el máximo de utilidad comprando una cantidad determinada de inversiones. Lo que se espera de la aplicación de Programación Dinámica es que sea poco compleja como solución, dejándola para posibles comparaciones en el futuro con otras formas de solución, de tal forma que se busque cual tiene un orden de complejidad menor.

2 DESCRIPCIÓN DEL PROBLEMA

La empresa de inversiones internacional *clover inversions* está preocupada, ya que, han nacido muchas nuevas empresas y con ello la lista de inversiones ha crecido de forma exponencial. Dado esto se pide que, dado un capital inicial, se pueda comprar las inversiones con tal de garantizar el máximo de utilidad. Para esto la entrada debe ser un archivo de texto que contenga la capital inicial, el número de inversiones disponibles, y las líneas siguientes muestran el costo y la utilidad de cada inversión.

Entrada.in

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

Figura 1: ejemplo de entrada.

Y la salida debe ser un archivo de texto que contenga el capital invertido y el beneficio total, además de la lista de inversiones compradas.

Salida.out

```
1400000 15500
100000
600000
450000
250000
```

Figura 2: ejemplo de salida.

Además, el algoritmo debe incluir las funcionalidades dinamica() y printCurrent(). La función dinamica() debe ser una función que genere el recorrido nodo por nodo y al mismo tiempo encuentre la solución óptima. La función printCurrent() debe ser una función que imprima las inversiones actuales que se tiene, es decir, las inversiones de beneficio máximo que se tiene mientras se van recorriendo los nodos del arbol:

```
printCurrent(...) {

        #ifdef DEBUG
        printf("enter para continuar...\n")
        while(getchar() != '\n');
        /*
        en esta parte debe escribir su código para imprimir
        lo que sea necesario para mostrar el estado actual
        del nodo.
        */
        #endif
    }
```

Figura 3: Parte del algoritmo printCurrent()

3 MARCO TEÓRICO

Para comprender los conceptos utilizados en el presente documento, se entiende lo siguiente: La Programación Dinámica, en informática, es un método para reducir el tiempo de ejecución de un algoritmo mediante la utilización de subproblemas superpuestos y subestructuras óptimas, en otras palabras, se utiliza para optimizar problemas complejos que pueden ser discretizados y secuenciados.

Una subestructura óptima significa que se pueden usar soluciones óptimas de subproblemas para encontrar la solución óptima del problema en su conjunto. Los

subproblemas se resuelven a su vez dividiéndolos en subproblemas más pequeños hasta que se alcance el caso fácil.

Para este laboratorio el problema se divide en 2 subproblemas, recordando las soluciones anteriores, esto gracias a los estados pendientes.

La estructura de datos utilizada en este laboratorio donde se aplicará la Programación Dinámica es un árbol binario. Este consiste en un árbol en el cual cada nodo puede tener un hijo izquierdo y un hijo derecho nada más, no puede tener más hijos. Para este trabajo todos los nodos deben tener 2 hijos (ni más ni menos). Si uno de los hijos está vacío o es *null*, o si no cumple con las restricciones, igual se considera como un hijo del nodo y para este caso este puede tener hijos. Cada nodo no puede tener más de dos subárboles.

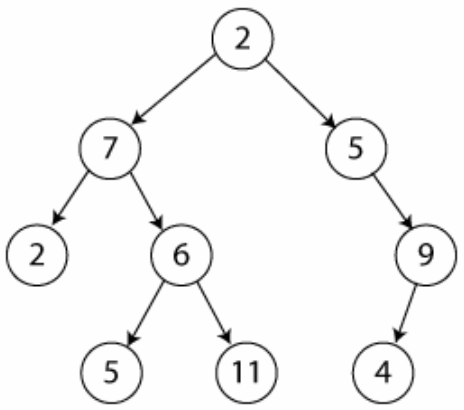

Figura 4: Ejemplo de árbol binario.

Existen muchas formas de representar o implementar un árbol, la usada en este trabajo es mediante lista enlazadas y una función que realiza recursión con estados pendientes.

4 DESCRIPCIÓN DE LA SOLUCIÓN

4.1 IDEA

Para resolver este problema por medio de Programación Dinamica, se tiene pensado el uso de listas enlazadas, arboles binarios. En el archivo de entrada, la primera línea que representa la capital disponible, con esta se podrá ir acotando las ramas del árbol y no entrar en subárboles que no cumplan con el capital disponible. Las inversiones de las siguientes líneas se guardan en una lista enlazada, donde en cada nodo se guarda el costo y la utilidad de cada una. Después a esta lista con la inversión se le ejecuta el método de Programación Dinámica.

Ya con la lista de inversiones lista, se realiza la Programación Dinámica mediante un árbol binario. Donde el valor de cada nodo será la lista de inversiones de uno de los dos hijos que tiene, el criterio para elegir uno de los dos hijos, es que el nodo que tenga mayor utilidad que el otro es el que se utiliza como nodo del padre. Cuando uno de los hijos nos cumple con la restricción que si al agregar tal inversión supera al capital disponible no se considerá ese nodo y se quedará solo con el otro. Realizará esto recursivamente hasta que encuentre el óptimo de utilidad entre las inversiones disponibles.

Lo anterior mencionado es similar al método de *Backtracking* con acotamiento, pero este es Programación Dinámica ya que, no se repite ningún nodo que ya haya sido revisado, es decir, por ejemplo, si en un subárbol se comparó la inversión 1 con la 3. En el otro subárbol del otro hijo del nodo no se compara el 1 con el 3 de nuevo, esta es la optimización del algoritmo, ya que, de forma “inteligente” compara solo los que le falta comparar.

Después de revisar todos y comparar los hijos izquierdo y derechos del nodo raíz, se entrega como lista final de inversiones la que tenga la mayor utilidad.

4.2 REPRESENTACIÓN Y ESTRUCTURA

Los datos del archivo de texto de la entrada inicialmente son representados como strings,

pero luego son transformados a enteros para ser trabajados. Solo se va a trabajas con una estructura, la cual es la lista enlazada. Esta va a poseer 3 tipos de datos para almacenar; el primero es un tipo de dato entero que representa el costo de ese nodo, el otro tambien es de tipo de dato entero que representa la utilidad de ese nodo y otra llamada *siguiente* para ver el siguiente nodo en la lista. Se van creando muchas listas dentro del árbol, pero se van guardando las que tiene por el momento la utilidad máxima. Si después una de esta ya no es la óptima se eliminar esta lista.

El árbol binario se representa en la función *dinámica*, donde cada llamada de la función representa un nodo de este y las listas representan los hijos de cada nodo.

4.3 PSEUDOCÓDIGO

Para el tipo de dato lista, posee alguna de sus funcionalidades básicas de Crear_lista(lista, entero, entero), obtener(lista, entero) y insertar(lista, lista, entero). La lectura del archivo se hace en la función principal, la cual también crea la lista enlazada a la que se le realiza la Programación Dinámica para después obtener la lista con las inversiones. Pero la función más importante es la que realiza el método de Programación Dinámica, la cual es la que realiza el recorrido del árbol binario, usando el criterio de máxima utilidad.

Para el pseudocódigo se mostrará el método de Programación Dinamica que se realiza en el código de forma general.

```
#listaInv: representa la lista con las inversiones disponibles

dinamica(listaInv,capital,lista): lista

  Si(listaInv->siguiente!=NULO):
    listaInv=listaInv->siguiente
    lista1=dinámica(listaInv,capital,lista)
    Si(listaInv->costo < capital):
      Capital=capital-listaInv->costo
      agregarNodo(lista,listaInv->costo)
      lista2=dinámica(listaInv,capital,lista)
  Sino:
    Si(lista=NULO y listaInv->costo < capital):
      CrearLista(lista)
    Sino:
      Si(listaInv->costo <= capital):
         AgregarNodo(lista,listaInv)
         Devolver lista
      Sino:
         Devolver lista

  Si(lista1.utilidad > lista2.utilidad):
    listaFinal=lista1
  Sino:
    listaFinal=lista2

  Devolver listaFinal
```

Realizará la recursión hasta que ya haya revisado todas las posibles combinaciones de inversiones que se pueda.

4.4 TRAZA

Para la traza de dará un caso ejemplo, en cómo se recorre el arbol nodo por nodo, y las inversiones que se van acumulando a medida que se recorre (en cual se mostrará en la siguiente imagen). En esta sección mostraremos el funcionamiento de la función printCurrent, la cual irá mostrando las inversiones con la utilidad máxima de cada recursión, esta irá cambiando a medida que se encuentre otra mejor.

- Lista inicial: Muestra cual es actualmente el conjunto de inversiones que tiene utilidad máxima, en las primeras ramas que revisó.

```
Inversiones de mayor beneficio actuales
costo: 300000 utilidad: 2000
costo: 100000 utilidad: 4500
```

- Segunda Lista: después, encuentra otra lista que tiene mayor utilidad que la anterior.

```
Inversiones de mayor beneficio actuales
costo: 450000 utilidad: 4000
costo: 300000 utilidad: 2000
costo: 100000 utilidad: 4500
```

- Caso final: Después de recorrer todos los nodos del árbol se muestra al final la lista de inversiones que da el mayor beneficio.

```
Inversiones de mayor beneficio actuales
costo: 250000 utilidad: 1000
costo: 600000 utilidad: 6000
costo: 450000 utilidad: 4000
costo: 100000 utilidad: 4500
```

4.5 ORDEN DE COMPLEJIDAD

Anteriormente en el resumen se explicó que el algoritmo es recursivo, ya que, se descompone en 2 subproblemas de tamaño n-1, el tiempo para definir el subproblema es de 1, entonces el tiempo es de:

$$T(n) = 2T(n-1) + O(1)$$

Entonces luego se utiliza la fórmula para algoritmos recursivos por sustracción, El algoritmo finalmente tiene un orden de:

$$\sim O(2^n)$$

Para todas las funciones básicas de la lista enlazada tiene complejidad de orden O(n), debido a que n corresponde a la cantidad de veces que se recorre la lista.

Para el algoritmo de printCurrent su complejidad tambien es O(n) debido a que imprime todos los elementos de una lista.

5 ANÁLISIS DE LA SOLUCIÓN

5.1 ANÁLISIS DE IMPLEMENTACIÓN

Se puede comprobar que el algoritmo se detiene, entregando el resultado correcto, si el archivo ingresado es correcto con los requerimientos, por ejemplo, las inversiones deben ser números enteros, al igual que la capital. Esto quiere decir que el algoritmo se detiene cuando entrega el resultado operando con los parámetros correctos, que sería un archivo de texto que contenga los datos necesarios.

El algoritmo no es eficiente, ya que, la complejidad no es un polinomio, aunque, por el momento es el mejor algoritmo para resolver este problema similar al conocido problema de la *Mochila*, por el hecho de que se obtiene la solución óptima y se realiza una división de los subproblemas de forma “Inteligente”, ya que, no repite estados en otras partes del arbol.

Al utilizar Programación Dinámica, se pudo reducir considerablemente los estados del árbol, logrando mejoras en temas de tiempo y espacio en la memoria, por el hecho de ser un algoritmo recursivos, disminuye la cantidad de estados pendientes que se tiene.

Otro método que también hubiese funcionado es utilizar *Backtracking*, pero como se mencionó anteriormente, esta Programación Dinámica es una optimización de este, por lo que este método es el mejor para obtener la

solución óptima. Si se quiere disminuir el tiempo de ejecución se podía usar el método Goloso, pero este no garantiza obtener la solución óptima.

Se realizaron 2 pruebas en el programa, en una se encuentran 5 inversiones disponibles para invertir y en la otra 8 inversión. En el primer caso, se generaron 32 nodos del árbol, el algoritmo se demoró 0,4 milisegundos en entregar el archivo de salida. Mientras que en el segundo caso se generaron 256 nodos del árbol de demoró aproximadamente 0,8 milisegundos. Con esto se podría decir que no varia tanto el tiempo cuando se agregan mas inversiones a comprobar.

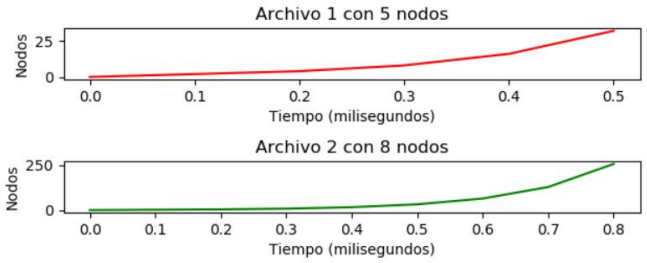

Figura 6: Gráfico de nodo vs tiempo.

5.2 EJECUCIÓN

El algoritmo fue implementado en el lenguaje de programación C. Además, se trabajó en el sistema operativo Windows 10, por lo tanto, para la compilación del programa se debe agregar en la consola lo siguiente:

```
Avanzados\Lab4>gcc dinamica.c -o ejecutable.exe
```

En el caso de que se quiere compilar implementando el DEBUG, se agrega lo siguiente:

```
Avanzados\Lab4>gcc dinamica.c -DDEBUG -o ejecutable.exe
```

Por ultimo, para ambos casos de compilación, la ejecución es la siguiente, en la cual se deben ingresar el archivo de entrada y el nombre del archivo de salida:

```
Avanzados\Lab4>ejecutable.exe entrada.in solucion.txt
```

6 CONCLUSIONES

Tras el desarrollo y análisis del algoritmo, se puede extraer que tiene dificultad implementar el código, ya que, poder optimizar el algoritmo usando Programación Dinámica requiere mucho análisis y entender muy bien paso a paso como se desarrolla el algoritmo. Tambien se logra de buena forma la implementación del método de Programación Dinámica.

Se logra la solución óptima para el problema dado con una complejidad exponencial. Por lo tanto, se puede concluir que en temas de eficiencia el algoritmo no es eficiente, pero si es el algoritmo con el mejor tiempo de ejecución.

En conclusión, se puede decir que el método de Programación Dinámica es el más recomendable de todos los demás métodos vistos anteriormente en los otros trabajos, ya que, si la prioridad es obtener la solución óptima. Este es el que en menor tiempo lo resuelve. Si se quiere priorizar complejidad y eficiencia antes de la exactitud de la solución, es recomendable usar el método Goloso.

7 REFERENCIAS

Enlaces de donde se ha basado para el desarrollo del documento. El formato debe ser

- “Algoritmos: Teoría y aplicaciones” , Mónica Villanueva ,2002 , disponible en: http://www.udesantiagovirtual.cl/moodle2/pluginfile.php?file=%2F117134%2Fmod_resource%2Fcontent%2F1%2FAlgoritmos-apuntes.pdf