Universidad de Santiago de Chile
Departamento ingeniería informática
Ingeniería civil informática
Algoritmos Avanzados
Laboratorio

2-2018

Generando el máximo beneficio

Valentina Ligueño
e-mail: valentina.ligueno@gmail.com

Generating the maximum benefit

RESUMEN: *El siguiente documento va a describir el funcionamiento de un algoritmo con retroceso, que tiene como finalidad encontrar la combinación de un conjunto de elementos que genera la utilidad máxima que se obtiene a partir de una restricción limitante.*

PALABRAS CLAVE: problema de la mochila, algoritmo recursivo, árbol binario, backtracking, algoritmo con retroceso.

ABSTRACT: *The following document, it's going to describe how it works an backtracking algorithm whose purpose is to find the combination from an set of elements which generates the maximum utility that is obtained from a limiting constraint.*

KEYWORDS: Knapsack problem, recursive algorithm, binary tree, backtracking algorithm.

1 INTRODUCCIÓN

El presente documento, tiene como enfoque principal describir cómo funciona y como fue implementado un algoritmo con retroceso o también conocido como backtracking, para la resolución de un problema predeterminado, por lo tanto primero se va a describir el problema, luego se presentará un marco teórico para la mejor comprensión de lo que se va a desarrollar, se va a describir como fue solucionado el problema, es decir se va a entregar una descripción detallada del algoritmo implementado y finalmente se va a analizar dicha solución y se va a concluir acerca de ella.

2 DESCRIPCIÓN DEL PROBLEMA

Como problema, se tiene inicialmente un presupuesto o capital fijo disponibles y además conjunto de posibles inversiones, las cuales tienen asociadas un costo y un beneficio o utilidad, la idea es encontrar un subconjunto que englobe la máxima cantidad de inversiones posibles, cuyo costo no supere capital inicial y que generen el máximo beneficio posible.

3 MARCO TEÓRICO

Un algoritmo con retroceso es una técnica de resolución de problemas que consiste básicamente en tomar decisiones acotadas por ciertas restricciones, con el fin de encontrar todos los caminos o combinaciones válidas que puedan resolver un determinado problema. Es decir avanza en torno a una solución y si dicha solución no satisface la solución del problema, retrocede hasta la última buena decisión que tomó y sigue otro camino, cuando finalmente ha generado todos los caminos posibles que resuelven el problema, se detiene.

Normalmente para generar el algoritmo con retroceso se necesita de árboles binarios, estos son estructuras de datos, se compone de nodos que tienen hijos, uno izquierdo y uno derecho, con hijos se hace referencia a que se enlazan a otro nodo, por lo tanto cada nodo tiene como máximo dos hijos, el backtracking lo utiliza porque al ir creando este árbol a partir de un conjunto de elementos se generan las decisiones, siendo por ejemplo hijo izquierdo no, e hijo izquierdo sí, este concepto quedará más claro cuando se explique el desarrollo de la solución.

.

4 DESCRIPCIÓN DE LA SOLUCIÓN

Para la resolución del problema se tienen los siguientes supuestos:

- El archivo de entrada siempre estará en el formato correcto, es decir, en su primera línea estará el capital máximo

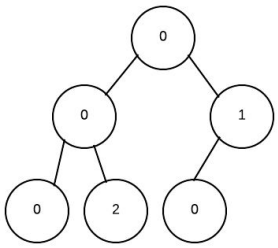
que se puede gastar, en la segunda la cantidad de inversiones existentes y en las siguientes estarán los nodos y luego se listan las inversiones, en donde aparece su costo y el beneficio asociado.

- El número de inversiones siempre corresponderá correctamente con las inversiones entregadas

4.1 IDEA

El problema planteado en el punto 2 se va a abordar con backtracking, se va crear una inversión vacía, la cual no tiene costo ni beneficio y se va a tomar como el primer elemento escogido, luego se va a insertar la primera inversión, se va a insertar de forma de generar un árbol binario, por lo tanto a la izquierda del nodo inicial se va a insertar una inversión vacía y a la derecha la primera inversión, luego para estos dos nodos que quedaron sin hijos, se va a hacer la segunda inserción de la segunda inversión siguiendo la misma idea, izquierda vacía, y derecha la inversión seleccionada en el momento, la idea es ir haciendo inserciones por nivel, a todos los nodos que no tengan hijos, en cada inserción se va a guardar el beneficio y capital acumulado hasta ese nodo, también en cada nodo se almacena que camino o combinación se siguió para llegar hasta allí; al insertar un nuevo elemento hay que verificar no se vaya a superar el capital límite disponible así solo se harán las inserciones que sí concluyan en una solución real y válida para el problema. Y finamente para que se entregue de inmediato la respuesta al terminar de generar el árbol, se guardará el máximo beneficio que se encontró, cuanto capital se gastó y que camino se necesitó para llegar allí.

Por ejemplo, si se tiene un capital de 2, las inversiones con costo 1 y 2, con beneficios 4 y 3 respectivamente, se genera el siguiente árbol.



Donde si se sigue cada rama y se ignora la inversión 0 se llega a las combinaciones { [ ], [2], [1] }. Como se ve en la imagen, la combinación [1,2] no se generó porque en total hacen un costo de 3, superando al capital disponible. el mayor beneficio se encuentra en la combinación [1], con un beneficio de 3, por lo tanto al terminar de crear este árbol se entrega que para un beneficio máximo con un capital de 2, la mejor opción es la inversión en 1.

4.2 REPRESENTACIÓN Y ESTRUCTURA

Los datos del archivo de entrada se almacenan en un arreglo de estructuras, en donde cada estructura almacena una inversión, su costo y su beneficio.

Luego las inversiones se van insertando en nodos que forman árbol binario como se explica en 4.1. de este arbol un nodo va a ser una estructura que contiene el costo, el beneficio y el camino acumulado hasta ese momento.

Finalmente el camino se representa con un string que escribe la inversiones utilizadas separadas por un guión (ejemplo: 1-2-4). Dichas inversiones son representadas por su índice en el arreglo de estructuras.

4.3 PSEUDOCÓDIGO

```
BENEFICIO_MAX = 0
CAPITAL_UTILIZADO = 0
CAMINO_FINAL = “”

backTracking(nodoActual,costo,beneficio,indice)
 si(nodoActual->izquierdo != Vacío)
   backTracking(nodoActual->izquierdo, costo,
   beneficio, indice)

 si(nodoActual->derecho != Vacío)
   backTracking(nodoActual->derecho,costo,
   beneficio, indice)
```

```
si(nodoActual->derecho == Vacío y
nodoActual->izquierdo == Vacío)

  costoAcumulado = nodoActual->costoAcumulado
  + costo;

  si (costoAcumulado <= CAPITAL_LIMITE)
     beneficioAcumulado = nodoActual->
     beneficioAcumulado + beneficio
     caminoAcumulado = nodoActual->camino +
     indice

     si (beneficioAcumulado > BENEFICIO_MAX)
        BENEFICIO_MAX = beneficioAcumulado
        CAPITAL_UTILIZADO = capitalAcumulado
        CAMINO_FINAL = caminoAcumulado

  nodoActual->izquierdo = nodoActual #Toma los
  valores de nodo actual;

  nodoActual->derecho =
  crearNodo(costoAcumulado,
  beneficioAcumulado, caminoAcumulado);


crearNodo(costo, beneficio, camino): nodoNuevo
   nodoNuevo = nodo vacío
   nodoNuevo-> costoAcumulado = costo;
   nodoNuevo-> beneficioAcumulado = beneficio;
   nodoNuevo-> camino =  camino + "-"
      costo = 0
      para j desde 0 hasta largo(permut)-1
        costo = costo + matrizAdy[j][j+1]

      si (costo < minimo)
        minimo = costo
        indice = indiceDe(permut)

   retornar nodoNuevo;

#Finalmente la solución está contenida en las
variables  BENEFICIO_MAX, CAPITAL_UTILIZADO Y
CAMINO_FINAL que entrega el máximo beneficio,
el capital que se invirtió para llegar a ese
beneficio y los índices de las inversiones que
se tomaron
```

4.4 TRAZA

Considere como entradas las siguientes inversiones costos siguientes 5 7, 8 6, 7 5, con un capital de 15

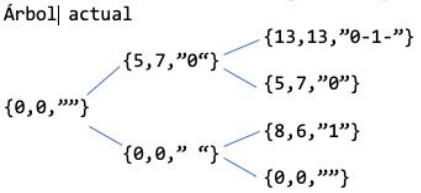
```
raíz = crearNodo (0,0,"")
raiz = {0,0,"-"}
backTracking(raíz, 5, 7, 0)
   costoAcumulado = 0+5 = 5
   BeneficioAcumulado = 0 + 7 = 7
   caminoAcumulado = "-" + "0" = "- 0"

   BENEFICIO_MAX = 7
   CAPITAL_UTILIZADO = 5
   CAMINO_FINAL = - 0
   nodoActual->izquierdo = crearNodo (0,0,"-")
   nodoActual->izquierdo = {0,0,"- -"}
   nodoActual->derecho = crearNodo (5,7,"-0")
   nodoActual->derecho = {5,7,"- 0 -"}
Fin inserción 1 (índice 0, costo 5, beneficio 7)
Árbol actual
         {5,7,"- 0 –"}
{0,0,""}
           {0,0," "}

backTracking(raíz, 8, 6, 1)
  #como raiz->izq no es nula cae en el primer if
  backTracking({0,0,"- -"},8,6,1)
    #Como ambos hijos son nulos cae en el tercer if
    costoAcumulado = 0+8 = 8
    BeneficioAcumulado = 0 + 6 = 6
    caminoAcumulado = "- -" + "1" = "- - 1"
    #Datos de solución no cambian
    nodoActual->izquierdo = crearNodo (0,0,"-")
    nodoActual->izquierdo = {0,0,"- - -"}
    nodoActual->derecho = crearNodo (8,6,"- - 1")
    nodoActual->derecho = {8,6,"-  - 1 -"}

  #Como raiz->der no es nula cae en el segundo if
  backTracking({5,7,"- 0 -"},8,6,1)
    #Como ambos hijos son nulos cae en el tercer if
    costoAcumulado = 5+8 = 13
    BeneficioAcumulado = 7 + 6 = 13
    caminoAcumulado = "- 0 -" + "1" = "- 0 - 1"
    #Datos de solución cambian porque hay nuevo
máximo
    BENEFICIO_MAX = 13
    CAPITAL_UTILIZADO = 13
    CAMINO_FINAL = "- 0 – 1"

    nodoActual->izquierdo = crearNodo (5,7,"- 0 -")
    nodoActual->izquierdo = {5,7,"- 0 - -"}
    nodoActual->derecho = crearNodo (13,13,"- 0 - 1")
    nodoActual->derecho = {13,13,"-  0 - 1 -"}
Fin inserción 2 (índice 1, costo 8, beneficio 6)
Árbol| actual
```

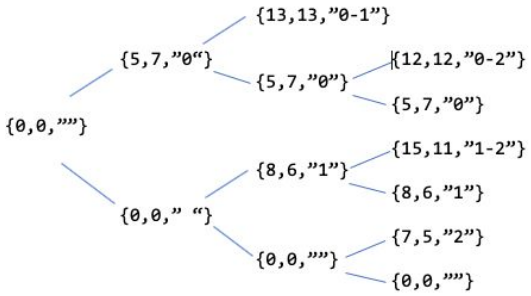



```
backTracking(raíz, 7, 5, 2)
  #raiz->izq = {0,0,"- -"} no nulo cae en el 1º if
  backTracking({0,0,"- -"}, 7, 5, 2)
    #{0,0,"- -"}->izq = {0,0,"- - -"}, no nulo 1º if
    backTracking({0,0,"- - -"}, 7, 5, 2)
      #Como ambos hijos son nulos cae en el tercer if
      costoAcumulado = 0+7 = 7
      BeneficioAcumulado = 0 + 5 = 5
      caminoAcumulado = "- - -" + "2" = "- - -2"
      #Datos de solución no cambian
      nodoActual->izquierdo = crearNodo (0,0,"- - -")
      nodoActual->izquierdo = {0,0,"- - - -"}
```

```
      nodoActual->derecho = crearNodo (7,5,"- - -2")
      nodoActual->derecho = {7,5,"- - - 2 -"}

    #{0,0,"- -"}->der = {8,6,"- -1- "} no nulo, 2º if
    backTracking({8,6,"- - 1 -"}, 7, 5, 2)
      #Como ambos hijos son nulos cae en el tercer if
      costoAcumulado = 8+7 = 15
      BeneficioAcumulado = 6 + 5 = 11
      caminoAcumulado = "- - 1 -" + "2" = "- -1-2"
      #Datos de solución no cambian
      nodoActual->izquierdo = crearNodo(8,6,"- -1-")
      nodoActual->izquierdo = {8,6,"- -1- -"}
      nodoActual->derecho = crearNodo(15,11,"- -1-2")
      nodoActual->derecho = {15,11,"- -1-2-"}

  #como raiz->der = {5,7,"- 0 -"} no nulo 2º if
  backTracking({5,7,"- 0 -"}, 7, 5, 2)
    #{5,7,"-0-"}->izq = {13,13,"0-1-"} no nulo 1º if
    backTracking({13,13,"0 - 1 -"}, 7, 5, 2)
      #Como ambos hijos son nulos cae en el tercer if
      costoAcumulado = 13+7 = 20
      #20 supera al capital inicial, no se crean
nodos
    #{5,7,"-0-"}->der = {5,7,"-0- -"} no nulo 2º if
    backTracking({5,7,"-0-  -"}, 7, 5, 2)
      #Como ambos hijos son nulos cae en el tercer if
      costoAcumulado = 5+7 = 12
      BeneficioAcumulado = 7 + 5 = 12
      caminoAcumulado = "-0- -" + "2" = "-0- -2"
      #Datos de solución no cambian
      nodoActual->izquierdo = crearNodo(5,7,"-0- -")
      nodoActual->izquierdo = {5,7,"-0- - -"}
      nodoActual->derecho = crearNodo(12,12,"-0- -2")
      nodoActual->derecho = {12,12,"-0- -2-"}
Fin inserción 2 (índice 2, costo 5, beneficio 7)
```




```
Beneficio máximo: 13
Capital utilizado: 13
Camino 0 - 1 -> 5 7 - 8 6
```

4.5 ORDEN DE COMPLEJIDAD

Para backTracking, como es un algoritmo recursivo se tiene un tiempo de

$$T(n) = 2T(n-1) + (n^0)$$

Queda lo anterior porque sólo realizan 2 llamadas recursivas, cada una con un elemento o nodo menos, y la parte no recursiva es constante. Entonces si luego se utiliza la fórmula para algoritmos recursivos por sustracción. El algoritmo finalmente tiene un orden de :

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

5 ANÁLISIS DE LA SOLUCIÓN

La solución implementada trabaja muy bien, ha sido probada hasta con 15 inversiones y entrega los resultados correctos en un tiempo muy corto. En todos los casos que se han probado funciona correctamente, incluso cuando todas las inversiones superan el capital.

5.1 ANÁLISIS DE IMPLEMENTACIÓN

¿Se detiene el algoritmo?
Sí, hablando específicamente de backTracking se detiene una vez que ha insertado todos los nodos posibles en las posiciones correctas, esta función es llamada por cada inversión que se quiere insertar.

¿Se detiene con la solución?
Si, ya que mientras se insertan las inversiones, al mismo tiempo se comprueba si aparece o no un nuevo beneficio máximo, y si es que lo hay se guardan dichos datos para que al terminar las inserciones se entregue la solución.

¿Es eficiente?
No es eficiente en términos de tiempo porque no se tiene una cota superior de orden polinómica.
¿Se puede mejorar?
Se podría buscar una manera más rápida de llegar a los nodos donde se van a hacer las inserciones, sin tener que realizar tanto llamado recursivo.

¿Otra idea?

Otra idea sería utilizar otro método de resolución de problemas, tales como fuerza bruta, goloso o programación dinámica

5.2 EJECUCIÓN

Formato de archivo de entrada:

```
Capital                    15
Número de nodos            3
Inversión1 Beneficio       9 7
Inversión2 Beneficio       8 6
```

Para compilar el programa:

```
$ gcc principal.c funciones.c -o programa
```

para ejecutar el programa:

```
$ ./programa
```

Y para entrar al modo DEBUG del programa:

```
$ gcc principal.c funciones.c -DDEBUG -o programa
```

Importante aclarar que para que el programa compile, el archivo de entrada debe ser tal cual como se explica en el formato. Si no se sigue el formato lo más probable es que o el programa no funcione o entrege una solución incorrecta.
Cuando el programa comience a ejecutarse, le pedirá que ingrese el nombre del archivo, debe ser muy preciso, si el programa no encuentra el archivo que usted indicó le volverá a pedir el ingreso de nombre hasta que le escriba un nombre de un archivo existente. Solo basta con esto y el programa indicará cuando termine la ejecución y escribirá el resultado en un archivo llamado salida.out

6 CONCLUSIONES

El algoritmo a pesar de no tener una complejidad eficiente, por no ser acotado polinomialmente, con la cantidad de nodos o inversiones que ha sido probado funciona muy rápido. Y en dichos casos ha entregado las soluciones correctas. Backtracking, en comparación con fuerza bruta es mucho más rápido, ya que en esta última había que generar el árbol completo que era complejidad ~ $O(2^n)$ y luego recorrerlo que también es de complejidad ~ $O(2^n)$ para descartar las soluciones que no eran válidas y encontrar la solución correcta. por lo que el algoritmo trabaja el doble de lo que lo hace ahora. Por lo que se concluye que backtracking es un método mucho más adecuado para enfrentar este problema.

7 REFERENCIAS

- "Complejidad de algoritmos recursivos", Alberto Castro, 2011, disponible en: http://btocastro.blogspot.com/2011/07/complejidad-de-algoritmos-recursivos.html
- "Backtracking", Jesús Lázaro García, s.f, disponible en: ftp://www.cc.uah.es/pub/Alumnos/G_Ing_Informatica/Algoritmia_y_Complejidad/anteriores/Apuntes/08_Backtracking.pdf