Solución de un problema mediante el algoritmo Backtracking

Universidad de Murcia
2º Grado en Ingeniería Informática
Asignatura: Algoritmos y estructuras de datos II
Curso 2017/2018

Proyecto realizado por:
Nicolás Enrique Linares La Barba
Miguel Ángel Lucas Barceló

Índice

1. Problema

Dado un conjunto C con N tipos de monedas y con un número inagotable de ejemplares de cada tipo, hay que conseguir, si se puede, formar la cantidad M empleando el mínimo número de ejemplares de monedas.

2. Diseño

Para el diseño de este problema hemos partido del esquema general de un algoritmo de backtracking dedicado a la optimización, proporcionado en las diapositivas del tema 5 de la asignatura. Que tiene la siguiente forma:

```
Backtracking (var s: TuplaSolución)
    nivel:= 1
    s:= s_INICIAL
    voa:= -∞; soa:= Ø
    repetir
        Generar (nivel, s)
        si Solución (nivel, s)  AND Valor(s) > voa entonces
            voa:= Valor(s); soa:= s
        si Criterio (nivel, s) entonces
            nivel:= nivel + 1
        sino
            mientras NOT MasHermanos (nivel, s) AND (nivel>0)
                hacer Retroceder (nivel, s)
        finsi
    hasta  nivel==0
```

Se ha modificado ligeramente el algoritmo ya que en el problema dado hay que calcular la solución con más peso y con menos, lo que quiere decir que hay que hacer un problema de optimización y minimización a la vez.

Pero en general, se ha respetado tanto la estructura, como los parámetros pasados a la función de backtracking y la utilización de las funciones y procedimientos empleados por el esquema para su mayor comprensión y semejanza.

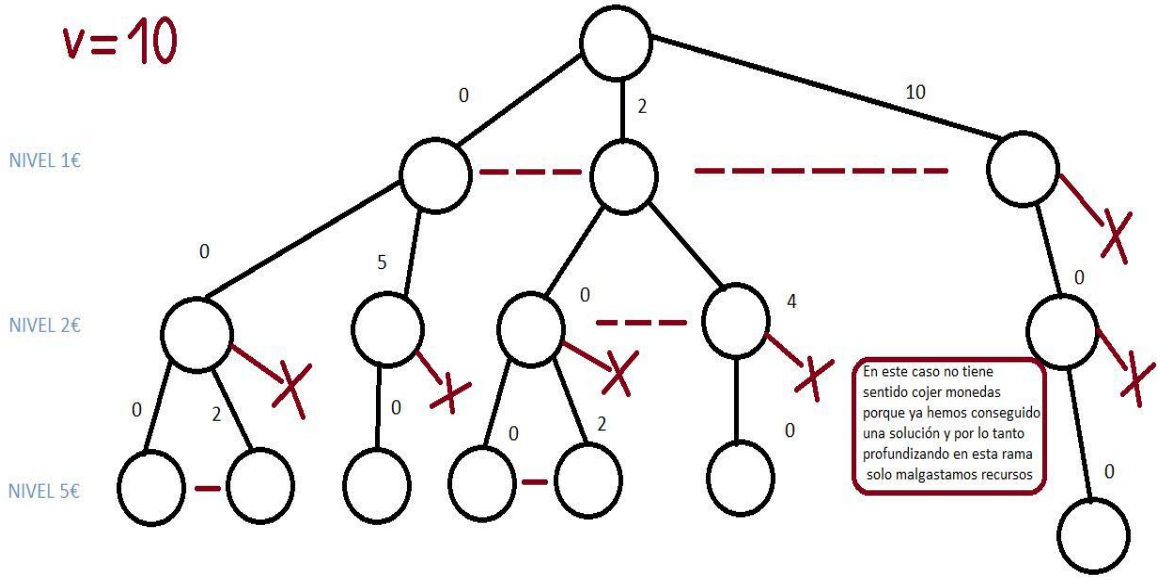
Para representar la ejecución, se ha escogido un árbol permutacional donde los niveles representan cada tipo de moneda y cada hermano el número de monedas escogidas de ese valor. En el conjunto solución se guarda el número de monedas escogidas de cada nivel.

Cada nivel tiene un número de hermanos diferente y cada hermano también tiene un número distinto de hijos. Esto se debe a que, el objetivo es conseguir un valor V con unas determinadas monedas que se pueden repetir ilimitadamente. Por lo que hay que asignar un número de hermanos a cada nivel acorde con la moneda y el valor que lleva acumulado.

Por ejemplo, si hay monedas de 1€, 2€ y 5€, y el valor a conseguir son 10€ no tiene sentido que se pueda poner más de 10 monedas de 1€, o 5 monedas de 2€. Si hay 2 monedas de 1€ no tiene sentido que se pueda poner más de 4 monedas de 2€ o 1 de 5€ ya que se pasaría del dinero que queremos conseguir, por lo que estaría calculando algo que no va a tener solución.

La fórmula sería la siguiente:

número de hermanos = (v - dineroActual) / valorMoneda(nivel).



Este diseño consiste en que, para cada caso y una vez leída la entrada, se avanza los niveles (tipo de moneda) del árbol que proporciona “generar”, y se comprueba en cada nodo generado si tiene “solución”. Como es un problema de minimización/maximización, si tiene solución y es mejor que la solución anterior entonces se guarda, pero se sigue ejecutando el algoritmo.

Cuando no se pueda seguir avanzando porque no se cumple “criterio”, se mira los hermanos (número de monedas) del mismo nivel, en el caso de que no haya ”más hermanos” retrocede un nivel y repite el proceso hasta que no pueda retroceder más, por estar en el primer nivel.

3. Explicación de las variables

Las variables utilizadas son “ncasos” que son los casos que se van a analizar, “numMonedas” que como su nombre indica son el número de monedas que se proporcionan. “V” es el valor que hay que conseguir con las monedas; los valores de cada moneda se guardan en el array “valores” mientras que los pesos se guardan en “pesos”.

La variable “s” es la tupla solución que indica cuantas monedas de cada tipo se han cogido, la tupla es igual al numMonedas.

La variable “nivel” indica el tipo de moneda que se analizan en un momento actual. La solución con menos peso la almacenas en “soaMin”, y el peso en “voaMin”; igual que para la solución más pesada en “soaMax” y “voaMax”.

4. Explicación de las funciones

Funciones y procedimientos principales:

- Backtracking(s), es la función principal del programa encargada de seguir el esquema y resolver el problema.

- Generar(nivel, s) proporciona el siguiente hermano o el primero del nivel que se le pasa como parámetro, siempre dará un hermano posible porque se comprueba antes de llamar a esta función.

- Pesar/valorar(nivel, s), calcula el peso o valor que hay en la tupla solución.

- MasHermanos(nivel, s), se encarga de calcular si quedan más hermanos posibles para un nivel o no. Aquí utiliza la ecuación antes mencionada:

  Número de hermanos = (v - dineroActual) / valorMoneda[nivel]

  Y comprueba si el número de hermanos que teníamos almacenado en S[nivel] es menor que el número de hermanos de dicho nivel.

- Criterio(nivel, s), comprueba si se podría conseguir una solución avanzando nivel, es decir, si puede seguir profundizando (no ha llegado a comprobar todas los tipos de monedas, nodo hoja) o si es factible, es decir si no ha superado el valor V. Si se ha superado se poda esa rama, para no malgastar recursos, ya que no podrá conseguir nada.

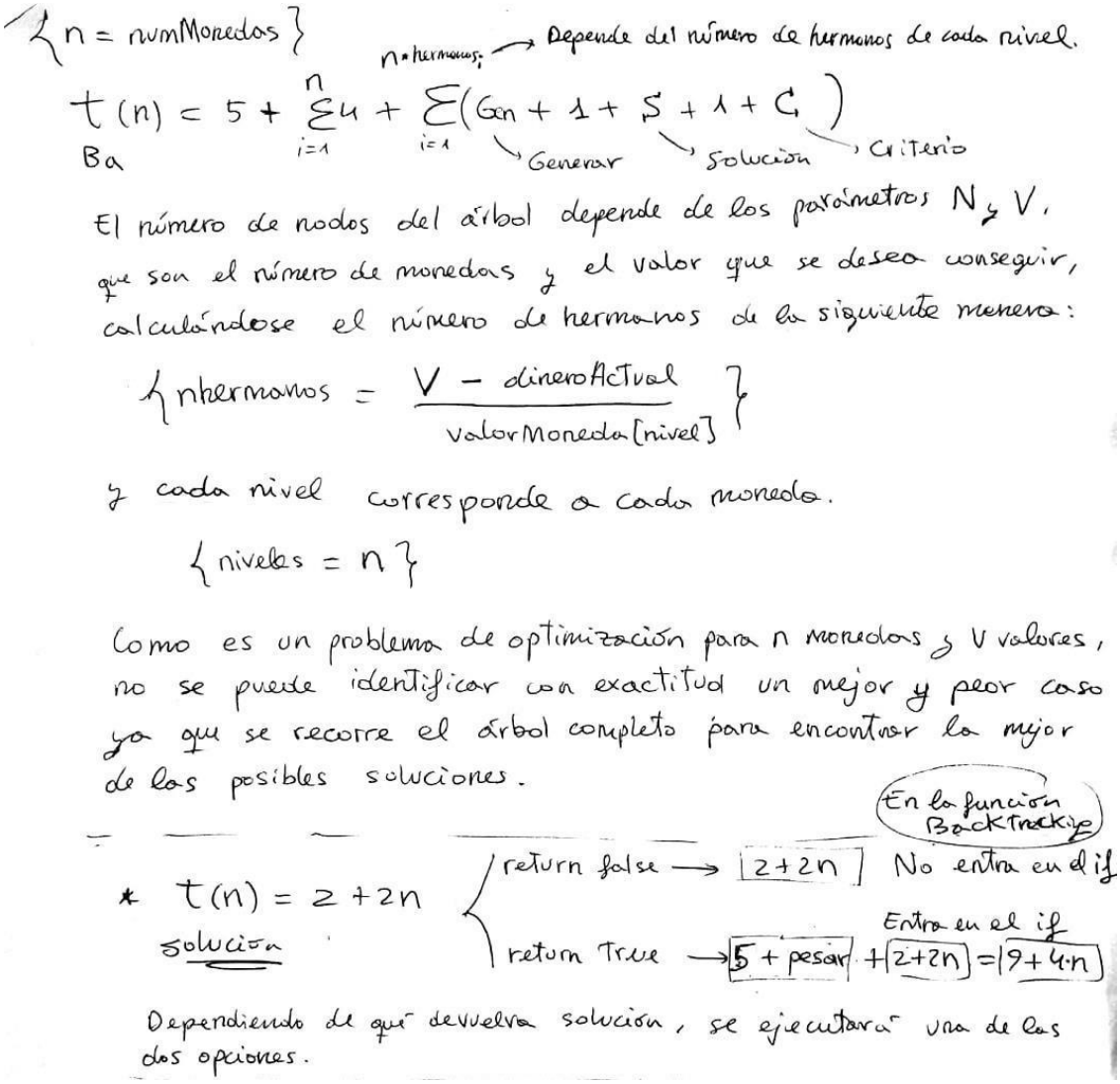
- Solucion(nivel, s), calcula si la tupla solución contiene una solución válida o no, es decir, si el valor de “s” corresponde a V y ha evaluado todos los tipos de monedas (estaría en un nodo hoja).

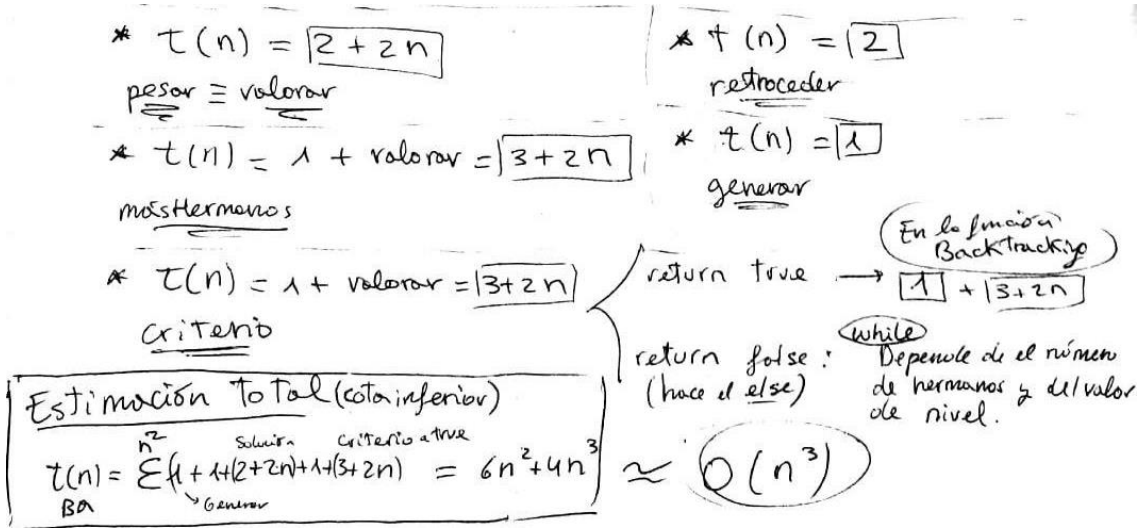
Otras funciones y procedimientos utilizados:

- *LeerEntrada()* : almacenará los datos proporcionados para cada problema, inicializando los valores de las matrices “amistad” y “trabajo”.

-
- ImprimirSol(s): recorre el conjunto solución imprimiendo las monedas que se han escogido y cuantas.

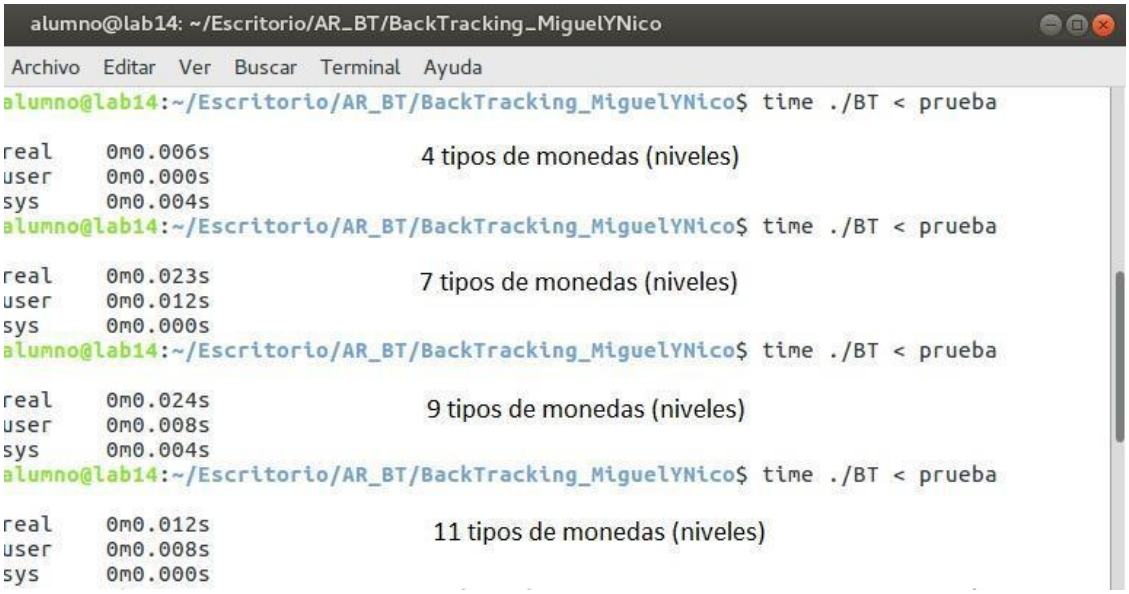
5. Estudio teórico del tiempo de ejecución





6. Estudio experimental

Para el análisis experimental se va a estudiar cuatro casos diferentes. Cada problema trabaja con unos datos de entrada distintos, que varían en el número de monedas que hay que emparejar y los valores de cada moneda valor y peso. Para obtener mejor resultado se ha quitado la salida del programa donde muestra los resultados de la solución. Podemos verlo en la siguiente imagen.



7. Contraste del estudio teórico y experimental

Se ha obervado que conforme aumenta el número de monedas empeora el tiempo de ejecución, sobre todo se puede comprobar que para 4 monedas tarda 0.006s y para 7 monedas 0.023s. Aunque ese caso nos coincida con el análisis teórico, los siguientes muestran discrepancia respecto a aumentar el número de monedas y mantener o incluso bajar el tiempo, esto es debido a los valores de las monedas, ya que gracias a las podas se puede ahorrar visitar muchos nodos y por lo tanto, mejorar el tiempo.