Instituto Tecnológico de Costa Rica

Curso Análisis de algoritmos

Profesor Mauricio Rojas Fernández

Estudiante: Daniel Montoya Aguilar

Carnet 201106385

Problemas resueltos con programación dinámica

Problema de la mochila con programación dinámica

Sea n objetos no fraccionables de pesos pi y beneficios bi. El peso máximo que puede llevar la mochila es C. Queremos llenar la **mochila** con objetos, de forma que se maximice el beneficio.

Datos

n objetos con pesos pi y beneficios bi asociados a cada objeto.
No se pueden fraccionar los objetos (se toman o no se toman).
Se define un problema más general en función del número de objetos y la capacidad C de la **mochila**: mochila(k,l,C).
Resolver el **problema** consiste en obtener: mochila(1,n,C) .

Resolución mediante Programación Dinámica

Para resolver el **problema de la mochila** necesitamos realizar ciertas acciones:

Ver que se cumple el principio de optimalidad de Bellman.
Buscar ecuaciones recurrentes para el **problema**.
Construir una tabla de valores a partir de las ecuaciones.

Principio de optimalidad de Bellman

Cualquier subsecuencia de decisiones de una secuencia óptima de decisiones que resuelve un **problema** también debe ser óptima respecto al subproblema que se resuelve.
Sea y1,…,yn una secuencia óptima de valores 0-1 para x1,…,xn.
o Si y1=0, entonces y2,…,yn forman una secuencia óptima para el **problema mochila**(2, n, C).
o Si y1=1, entonces y2,…,yn forman una secuencia óptima para el **problema mochila**(2, n, C - p1).

Demostración: Si existe una solución mejor para el **problema** correspondiente, entonces es mejor que para el **problema mochila**(1, n, C), en contra de la hipótesis. Lo mismo se cumple en cualquier etapa de decisión.

Ecuaciones recurrentes para el problema

a) Ecuación hacia delante

Supongamos que gi(C) es el beneficio acumulado para la solución óptima del **problema mochila**(j,n,C), entonces:

$g_j(C) = \max \{g_{j+1}(C), g_{j+1}(C-p_j)+b_j\}$

cuyo significado:

gj+1(C) -> no se toma el objeto j
gj+1(C-pj)+bj -> se toma el objeto j

El caso trivial se da cuando j vale n+1, y en este caso:

gn+1(C) = 0

Luego el cálculo de **mochila**(1,n,C) se reduce a la aplicación de las ecuaciones:

gj(C) = max {gj+1(C), gj+1(C-pj)+bj} si j<>n+1
gj(C) = 0 si j = n+1

b) Ecuación hacia atrás

Supongamos que gj(C) es el beneficio acumulado para la solución óptima del **problema mochila**(1,j,C), entonces:

gj(C)= max {gj-1(C), gj-1(C-pj)+bj}

cuyo significado:

gj-1(C) -> no se toma el objeto j
gj-1(C-pj)+bj -> se toma el objeto j

El caso trivial se da cuando j vale 0, y en este caso:

g0(C) = 0

Luego el cálculo de **mochila**(1,n,C) se reduce a la aplicación de las ecuaciones:

gj(C) = max {gj-1(C), gj-1(C-pj)+bj} si j<>0
gj(C) = 0 si j = 0

```
--Implementación del problema de la mochila con programación dinámica--
Introduzaca el valor del objeto: 1

4
Introduzaca el valor del objeto: 2

2
Introduzaca el valor del objeto: 3

2
Introduzaca el valor del objeto: 4

1
Introduzaca el valor del objeto: 5

10
Introduzaca el peso del objeto: 1

12
Introduzaca el peso del objeto: 2

2
Introduzaca el peso del objeto: 3

1
Introduzaca el peso del objeto: 4

1
Introduzaca el peso del objeto: 5

4
15
Matriz de solución:
0 0 0 0 0 0 0 0 0 0 0 0 0 4 4 4 4
0 0 2 2 2 2 2 2 2 2 2 2 4 4 6 6
0 2 2 4 4 4 4 4 4 4 4 4 4 6 6 8
0 2 3 4 5 5 5 5 5 5 5 5 5 6 7 8
0 2 3 4 10 12 13 14 15 15 15 15 15 15 15 15
```

Problema de las monedas con programación dinámica

Para el problema de las monedas con **programación dinámica** se necesita crear un algoritmo que permita a una máquina expendedora devolver el cambio mediante el menor número de monedas posible. Mediante la programación dinámica se solucionará el caso en el que el número de monedas de cada tipo es ilimitado. En el **problema de las monedas** mediante el algoritmo voraz el que el número de monedas es ilimitado.

Descripción

Supongamos que se tienen monedas de valor 1, 4 y 6 y que se debe devolver una cantidad correspondiente al valor 8. Siguiendo el método de la **programación dinámica**, se rellenará una tabla con las filas correspondientes a cada valor para las monedas y las columnas con valores desde el 1 hasta el 8. Cada posición (i, j) de la tabla nos indica el número mínimo de monedas requeridas para devolver la cantidad j con monedas con valor menor o igual al de i:

____ 1 2 3 4 5 6 7 8
m1=1 1 2 3 4 5 6 7 8
m2=4 1 2 3 1 2 3 4 2
m3=6 1 2 3 1 2 1 2 2

Ejemplo para la posición i = 2 y j = 7, se requiere una moneda tipo 2 con valor 4 y tres monedas de tipo 1 con valor uno, por lo tanto en la tabla el número de monedas en la posición (2,7) sera 1 + 3 = 4.

Algoritmo

1. Para cada casilla de la tabla hacer:
2. Si el valor de la moneda actual es mayor que la cantidad, se paga con el resto de monedas, es decir, se toma el resultado de la casilla superior.
3. Si el valor de la moneda actual es menor o igual que la cantidad, se toma el mínimo entre:
1. Pagar con el resto de monedas, tomando el resultado de la casilla superior.
2. Pagar con una moneda del tipo actual y el resto con el resultado que se hubiera obtenido al pagar la cantidad actual a la que se le ha restado el valor de la moneda actual.
4. Tomar como resultado el valor de la última celda.

```
--Implementación del problema de las monedas con programación dinámica--
Monedas:
1 4 6
El número mínimo de monedas requeridas para devolver la cantidad 8 es: 2
```

Bibliografía:

Unidad V Programación dinámica. Consultado el 23 de octubre del 2014 de http://es.slideshare.net/elmergabrielchanpech/programacin-dinmica-15433493

Programación dinámica. Consultado el 23 de octubre del 2014 de http://es.wikipedia.org/wiki/Programaci%C3%B3n_din%C3%A1mica

Universidad Nacional de Colombia. Investigación de operaciones: programación dinámica. Consultado el 23 de octubre del 2014 de http://www.virtual.unal.edu.co/cursos/sedes/medellin/3007324/und_5/html/tema_01/contenido_03.html