Desempeño de la Programación Dinámica

Jeffrey Alfaro, 2015069856, *Escuela de Ingeniería en Computación, Instituto Tecnológico de Costa Rica*
Kevin Alpizar, 2016123044, *Escuela de Ingeniería en Computación, Instituto Tecnológico de Costa Rica*

I. Introducción

EL presente documento consiste en mostrar los resultados que se hicieron para evaluar el desempeño de la programación dinámica versus fuerza bruta con dos problemas típicos de programación dinámica como lo son el problema de la ”mina de oro” y el problema del ”contenedor” o mejor conocido como el problema de la ”mochila“, también se pretende mostrar la complejidad temporal de cada uno de ellos como manera de formalizar el desempeño de cada uno de los problemas.

II. Metodología

A continuación, se mostrarán los resultados obtenidos al realizar pruebas de comparación entre algoritmos de fuerza bruta y programación dinámica para los problemas de la del Contenedor y Mina de Oro. Para realizar los experimentos fueron ambos problemas fueron programados en Backtracking y programación dinámica en el lenguaje de programación python versión 3.

A. Problema 1: Mina de oro

Una mina de oro de dimensiones N*M. Cada campo de la mina contienen un número entero positivo el cual es la cantidad de oro en toneladas. Inicialmente un minero puede estar en la primera columna pero en cualquier fila y puede mover a la derecha, diagonal arriba a la derecha o diagonal abajo a la derecha de una celda dada. El programa debe retornar la máxima cantidad de oro que el minero puede recolectar llegando hasta el límite derecho de la mina, y las casillas (camino) seleccionadas.]

Parámetros de entrada del programa:

mina.py (generar/-g/g) N M oroMax iteraciones
Donde:

1) mina.py: es el archivo que se va a ejecutar.
2) En el segundo parámetro se puede ingresar ya sea generar, -g o g indicando al programa que va a generar un experimento con los parámetros restantes.
3) N: es la cantidad de filas que va a tener la mina.
4) M: es la cantidad de columnas que va a tener la mina.
5) oroMax: donde oroMax la cantidad máxima para generar oro por cada campo, siendo valores entre 1 y oroMax.
6) Iteraciones: la cantidad de veces que se debe correr el sistema.

Para cada experimento con el problema del contenedor, se utilizaron como parámetros de entrada: mochila.py -g N M 50 50, donde N la cantidad defilas y M la cantidad de columnas, en este caso se probaron con N y M = 11, 12 y 13, oroMax = 50 e iteraciones = 50. En la figura “X” se muestran los resultados segundos como unidad de tiempo, como la duración promedio por cada experimento.

Fig. 1. Comparación de tiempos para la Mina de Oro

Tabla de Resultados en segundos:

NxM	Dinámica	Backtracking
11x11	0.0005612188	2.0856872262
12x12	0.0013044848	7.2537790924
13x13	0.0054998008	19.308314106

B. Problema 2: Contenedor

Se tienen n elementos distintos, y un contenedor que soporta una cantidad específica de peso W. Cada elemento i tiene un peso w_i, un valor o beneficio asociado dado por b_i, y un cantidad dada por c_i. El problema consiste en agregar elementos al contenedor de forma que se maximice el beneficio de los elementos que contiene sin superar el peso máximo que soporta. La solución debe ser dada con la lista de los elementos que se ingresaron y el valor del beneficio máximo obtenido.

Parámetros de entrada del programa:

mochila.py (genarar/-g/g) W N pesoMax beneficioMax cantidadMax iteraciones
Donde:

1) Mochila.py: es el archivo que se va a ejecutar.
2) En el segundo parámetro se puede ingresar ya sea generar, -g o g indicando al programa que va a generar un experimento con los parámetros restantes.
3) W: el tamaño del contenedor
4) N: la cantidad de elementos

5) pesoMax: el máximo valor para generar el peso
6) beneficioMax: el máximo valor para los beneficios
7) cantidadMax: el máximo valor para las cantidades
8) Iteraciones: la cantidad de veces que se debe correr el sistema

Los valores peso, beneficios, cantidad se deben generar aleatoriamente desde 1 hasta su máximo(pesoMax, beneficioMax, cantidadMax respectivamente).Los datos que se generan son los datos con que se corre la cantidad de iteraciones especificadas en el parámetro número ocho.

Para cada experimento con el problema del contenedor, se utilizaron como parámetros de entrada: mochila.py -g 1000 N 20 300 25 50, donde 1000 sería el tamaño del contenedor, N la cantidad de elementos, en este caso se probaron con N = 20, 22, 24, 26, 28, 30, pesoMax = 20, beneficioMax = 300, cantidadMax = 25 e iteraciones = 50. En la figura 2 se muestran los resultados segundos como unidad de tiempo, como la duración promedio por cada experimento.

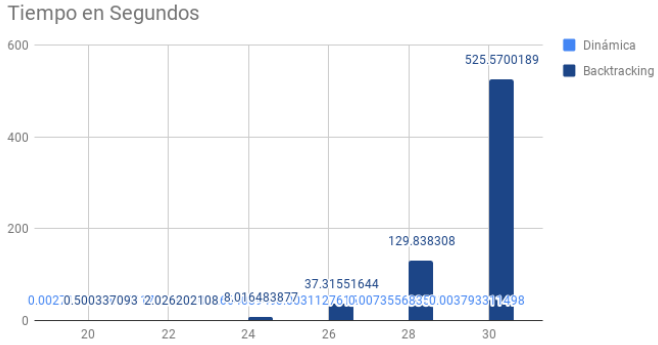

Fig. 2. Comparación de tiempos para el contenedor

Tabla de Resultados en segundos:

N	Dinámica	Backtracking
20	0.002719247503	0.500337093
22	0.0117923025	2.026202108
24	0.002664089498	8.016483877
26	0.0031127615	37.31551644
28	0.007355683501	129.838308

III. Análisis

A. Problema 1: Mina de oro

La complejidad temporal del algoritmo de fuerza bruta para este problema es de:

$$O(3^n)$$

.
La complejidad temporal del algoritmo de programación dinámica para este problema es de:

$$O(nlog(n))$$

B. Problema 2: Contenedor

La complejidad temporal del algoritmo de fuerza bruta para este problema es de:

$$O(2^n)$$

.
La complejidad temporal del algoritmo de programación dinámica para este problema es de:

$$O(nlog(n))$$

IV. Conclusión

Es claro que este tipo de problemas que se pueden aplicar programación dinámica son mucho mejores en tiempo de ejecución que aplicando una algoritmo de fuerza bruta, y según los experimentos hay una diferencia muy significativa de tiempo entre ambos algoritmos, se concluye que en estos casos es mucho mejor utilizar programación dinámica para resolver este tipo de problemas. También concluimos que la complejidad de tiempo de programación dinámica es mucho más rápido que el de backtracking.