**Análisis y diseño de algoritmos II**

**Trabajo Practico Especial**

**Algoritmo Heurístico para la resolución del problema de la mochila**

**Integrante 1:** Leandro Salias

* **E-Mail:** leandrosalias42@gmail.com

**Integrante 2:** Gonzalo Clementi

* **E-Mail:** gonzacl26@gmail.com

Índice

**Introducción………………………………………………………………………………….…………3**

* Introducción del problema que se implementa…………………………….…**3**

**Decisiones de diseño de la implementación……………………………………….……3**

* Diseño e implementación…………………………………………………………………**3**

**Análisis de la complejidad…………………………………………………………………………4**

* Análisis de la complejidad de los algoritmos implementados……………**4**

**Conclusión……………………………………………………………………………………..………….4**

* Análisis del comportamiento de los algoritmos

según los casos de entrada……………..….………………………………….….…….**4**

* Ejemplo 1……………………………………………………………………………………………………**4**
* Ejemplo 2……………………………………………………………………………………………………**5**
* Mejoras propuestas……………………………………..……………………………….….**6**

**Referencias………………………………………………………………………………………….……6**

Introducción

El problema de la mochila, comúnmente abreviado por KP (del inglés Knapsack problem) es un problema de optimización combinatoria, es decir, que busca la mejor solución entre un conjunto finito de posibles soluciones a un problema. Lo que se busca resolver en este problema es cómo llenar una mochila que tiene una capacidad determinada, introduciendo objetos los cuales tienen un peso y un valor de beneficio específicos, de manera tal que se maximice el beneficio total al introducir los elementos sin sobrepasar la capacidad máxima de la mochila.

De manera más formal podemos decir que teniendo n elementos, cada elemento i tiene asignado un valor v de beneficio y p de peso. Si para cada i se cumple que vi > 0 y pi > 0, y siendo C la capacidad máxima de la mochila, la solución al problema vendrá dado por la secuencia de variables x1, x2, ..., xn donde el valor de xi será 1 si el elemento se introduce en la mochila o de lo contrario será 0. Luego nuestro objetivo es:

maximizar

tal que

Para este trabajo se nos requiere implementar un algoritmo heurístico para resolver este problema, tal que podamos obtener una solución cercana a la óptima o incluso la óptima para ciertos casos.

Diseño e implementación

Para resolver el problema decidimos implementar un algoritmo basado en la técnica de programación Greedy. Lo que éste hace es introducir los elementos uno a uno, hasta que alguno de éstos exceda la capacidad de la mochila, si esto pasa el procedimiento termina. Para mejorar esto implementamos un algoritmo heurístico de ordenamiento, el cual ordena los elementos dada la relación beneficio/peso para que primero ingresen a la mochila los elementos que tengan mayor beneficio y menor peso, y así maximizar el beneficio de la mochila.

Decidimos utilizar como estructuras dos arreglos: uno que guarda los objetos (cada uno con su respectivo beneficio y peso) y otro que guarda la solución, es decir los objetos que fueron introducidos en la mochila.

Análisis de la complejidad

Para determinar la complejidad temporal, analizaremos el siguiente pseudocódigo:

Siendo C la capacidad máxima de la mochila, n la cantidad de objetos, y obj el arreglo de objetos.

**Problema de la Mochila(obj[], C, n)**

peso = 0

beneficio = 0

i = 0

mergeSort(arr,0,n-1) ----> **O(n.log(n))**

mientras (peso + obj[i].peso <= C) hacer ----> **O(n)**

peso = peso + obj[i].peso

beneficio = beneficio + obj[i].beneficio

i = i+1

retornar beneficio

Como puede observarse el ordenamiento por Mergesort tiene complejidad O(n.log(n)) siendo n la cantidad de elementos, y luego el while tiene complejidad O(n) ya que en el peor caso inserta n elementos en la mochila. Por lo tanto como nos quedamos con el mayor entre ambos la complejidad temporal del algoritmo para el problema de la mochila termina siendo O(n.log(n))

Conclusión

Análisis del comportamiento de los algoritmos

Para verificar que el algoritmo y la heurística sean buenos o no, analizaremos algunos casos de prueba:

Caso 1:

Tomaremos como capacidad de la mochila C = 10

Luego los objetos serán:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Objeto** | **A** | **B** | **C** | **D** | **E** |
| **Beneficio** | 6 | 3 | 10 | 4 | 5 |
| **Peso** | 1 | 4 | 1 | 2 | 5 |
| **Relación (b/p)** | 6 | 0,75 | 10 | 2 | 1 |

Después de ordenar por Mergesort:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Objeto** | **C** | **A** | **D** | **E** | **B** |
| **Beneficio** | 10 | 6 | 4 | 5 | 3 |
| **Peso** | 1 | 1 | 2 | 5 | 4 |
| **Relación (b/p)** | 10 | 6 | 2 | 1 | 0,75 |

La solución óptima sería ingresar los elementos C, A, D y E, lo cual nos resulta en un beneficio total de 25 y un peso ingresado de 9.

Introduciendo los elementos tal como dice el algoritmo, ingresaríamos los siguientes elementos: C-A-D-E

El elemento B no ingresa ya que excede la capacidad máxima de la mochila que era 10. El peso que se ingresa es de 9 y el beneficio total es de 25. Para este caso se pudo llegar a la solución óptima.

Caso 2:

Tomaremos como capacidad de la mochila C = 110

Luego los objetos serán:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Objeto** | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** |
| **Beneficio** | 11 | 21 | 31 | 33 | 43 | 53 | 55 | 65 |
| **Peso** | 1 | 11 | 21 | 23 | 33 | 43 | 45 | 55 |
| **Relación(b/p)** | 11 | 1,90 | 1,47 | 1,43 | 1,30 | 1,23 | 1,22 | 1,18 |

Como ya están ordenados por la relación beneficio/peso el Mergesort deja todo como está. Luego la solución óptima es introducir los objetos A, B, C, E y F con un beneficio total de 159 y un peso óptimo de 109.

Ahora el algoritmo introduce los objetos en el siguiente orden: A-B-C-D-E

Hasta este punto el beneficio es de 139 y el peso ingresado es 89. Cuando quiere ingresar el objeto F si se lo agrega se sobrepasa la capacidad de la mochila, por lo tanto, termina de ejecutarse el algoritmo y retorna como beneficio 139, lo cual no es el óptimo.

Analizando los dos casos podemos concluir que esta heurística encuentra la solución óptima para ciertos casos, y para otros encuentra una solución que no es la óptima, pero es una solución factible al problema, cumpliendo con la definición de algoritmo heurístico.

Mejoras propuestas

Para este caso del problema de la mochila en el cual ingresan o no ingresan objetos, en ningún momento se fragmentan, una posible alternativa puede ser la implementación del algoritmo utilizando programación dinámica. El siguiente pseudocódigo da una idea de cómo funciona:

Siendo C la capacidad máxima de la mochila, n la cantidad de objetos, b el vector de beneficios, y p el vector de pesos. La matriz c va guardando el beneficio óptimo.

**Problema de la Mochila (b, p, n, C)**

para col = 0 hasta C hacer ----> **O(C)**

c[0, col] = 0

para fil = 1 hasta n hacer ----> **O(n)**

c[fil, 0] = 0

para col = 1 hasta C hacer ----> **O(C)**

si p[fil] ≤ col entonces

c[fil][col] = max(b[fil] + c[fil – 1][col – p[fil-1], c[fil-1][col]

sino

c[fil, col] = c[fil-1, col]

retornar c[n][C]

El lugar c[n][C] de la matriz es el beneficio óptimo que se puede obtener. Luego para encontrar los elementos que se introdujeron se comienza desde la posición c[n][C] hacia atrás, si c[fil][col] = c[fil – 1][col] entonces el elemento fil no pertenece a la solución y se continúa buscando con c[fil – 1][col]. De lo contrario, fil es parte de la solución y se continúa buscando con c[fil – 1][col – C].

La mejora que presenta esta implementación en cuanto a la hecha por nuestro grupo, es que la solución que ofrece es siempre la óptima, a diferencia de la nuestra que da una solución óptima para ciertos casos. Pero al encontrar siempre la solución óptima se sacrifica complejidad temporal, ya que el tiempo que tarda este algoritmo de programación dinámica es de O(nC) siendo n la cantidad de objetos y C la capacidad de la mochila, lo cual es mayor al tiempo O(n.log(n)) del algoritmo Greedy que implementamos.

También investigamos que existen otros tipos de problema de la mochila. Uno de ellos, el problema de la mochila fraccional, puede resolverse llegando a la solución óptima utilizando el método Greedy que implementamos. Esto se debe a que, al ingresar el último elemento a la mochila, si éste excede la capacidad, se fracciona de manera tal que pueda entrar y al tener los elementos ordenados con la heurística implementada se estaría consiguiendo el beneficio total óptimo.

Referencias

* https://es.wikipedia.org/wiki/Problema\_de\_la\_mochila