Actividad 1

Heurísticas Voraces:

Problema de la Mochila

Diego Andrés Figueroa Peart
A01660987

Curso:

Análisis y diseño de algoritmos avanzados
(TC2038.652)

Profesor:

Cristhian Alejandro Ávila Sánchez

Fecha de entrega:

8 de octubre de 2023

Actividad 1. Heurísticas Voraces: Problema de la Mochila

Planteamiento del problema

El problema de la mochila (en inglés, Knapsack problem) es un problema de algoritmos computacionales establecido por Richard M. Karp en su artículo “Reducibility Among Combinatorial Problems”, publicado por primera vez en el año 1972.

El problema plantea una mochila con una capacidad máxima de peso P_{max} (la versión a resolver añade adicionalmente una capacidad máxima de volumen, V_{max}) y una lista de N artículos, cada uno con un valor de utilidad U, peso P y volumen V. Se desea incluir en la mochila un subconjunto de M artículos, maximizando la suma de sus utilidades sin rebasar los límites de peso y volumen.

Para resolver este problema, utilicé una estrategia voraz, es decir, en cada paso individual busqué el artículo con mayor utilidad que todavía cupiera en la mochila, con el objetivo de tener al final un conjunto de los artículos de mayor utilidad.

Datos de entrada:

- n = Número de artículos.
- a = Lista de identificadores de los artículos (enumerados de 1 a n).
- p = Lista de pesos de los artículos (cada uno entre 1 y 100).
- v = Lista de volúmenes de los artículos (cada uno entre 1 y 100).
- u = Lista de valores de utilidad de los artículos (cada uno entre 1 y 100).
- pmax = Peso máximo que soporta la mochila (10 veces el número de artículos).
- vmax = Volumen máximo que soporta la mochila (10 veces el número de artículos).

Datos de salida:

- m = Número de artículos en la mochila.
- articulos_mochila = Lista de artículos incluidos en la mochila.
- u_total = Suma de las utilidades de los artículos incluidos.

Funcionamiento:

1. Se define el número de artículos en las listas y estos se generan con números enteros aleatorios. Se imprimen los mismos en la consola, con los valores de pmax y vmax.

2. Se llama a una función llamada “mochila” con los parámetros de entrada (n, a, p, v, u, pmax y vmax).
3. Se termina la ejecución de la función mochila si n, pmax o vmax son iguales a 0.
4. Se obtiene el artículo de la lista con mayor utilidad y su índice dentro de la lista.
5. Si el artículo cabe en la mochila, es decir, si su peso y volumen no exceden los límites anteriormente establecidos, este se incluye en la mochila. Se añade su identificador a artículos_mochila, se suma 1 a m, y se suma su valor de utilidad a u_total.
6. Se resta el peso y volumen del artículo a los límites de la mochila, para usar estos nuevos valores en la llamada recursiva.
7. Haya cabido o no el artículo en la mochila, este se elimina de las listas a, p, v y u, y se resta 1 de n.
8. Se realiza una llamada recursiva a la función mochila con las listas actualizadas sin el artículo anteriormente procesado, los nuevos valores de pmax y vmax, y los valores acumulados de u_total, m y artículos_mochila. Se regresa al paso 3.
9. Una vez que se termine la ejecución, se imprimirán en la consola el número de artículos incluidos en la mochila, la lista de los mismos, y la utilidad máxima que se obtuvo.

Pseudocódigo:

```
function mochila(n, a, p, v, u, pmax, vmax, u_total, m, articulos_mochila)
    if n, pmax or vma = 0 then
        return u_total, m, articulos_mochila
    end if
    max_u = maximum(u)
    max_index = index(max_u)
    if pmax - p[max_index] >= 0 or vmax - v[max_index] >= 0 then
        m + 1
        append a[max_index] to articulos_mochila
        u_total + max_u
        pmax - p[max_index]
        vmax - v[max_index]
    end if
    remove max_index from p
    remove max_index from v
    remove max_index from u
    remove max_index from a
    n - 1
    mochila(n, a, p, v, u, pmax, vmax, u_total, m, articulos_mochila)
end function
```

Complejidad:

Mejor caso: O(1)

Caso promedio: $O(n^2)$

Peor caso: $O(n^2)$

El programa que realicé tiene una complejidad de mejor caso de O(1), representando aquel caso donde n sea igual a 0 y la función mochila cese inmediatamente. La complejidad de caso promedio y peor caso de $O(n^2)$ se debe a que en promedio deberá haber una llamada recursiva por cada artículo en la lista (excluyendo casos en que los pesos y/o volúmenes de los artículos incluidos en la mochila sea exactamente igual a los límites, en cuyo caso la función cesará en cuanto la sumatoria llegue a este valor). Dentro de cada llamada recursiva, utilizo las funciones max() y list.index(), las cuales tienen complejidad de O(n). Multiplicando la complejidad de la función recursiva por la complejidad de las funciones que contiene, es decir, O(n) x O(n), obtenemos la complejidad de $O(n^2)$.

Programa:

```
import random as r
import sys

sys.setrecursionlimit(100000)

def mochila(n, a, p, v, u, pmax, vmax, u_total, m, articulos_mochila):
    if not n or not pmax or not vmax:
        return u_total, m, articulos_mochila
    max_u = max(u)
    max_index = u.index(max_u)
    if (pmax - p[max_index]) >= 0 or (vmax - v[max_index]) >= 0:
        m += 1
        articulos_mochila.append(a[max_index])
        u_total += max_u
        pmax -= p[max_index]
        vmax -= v[max_index]
    p.pop(max_index)
    v.pop(max_index)
    u.pop(max_index)
    a.pop(max_index)
    n -= 1
    return mochila(n, a, p, v, u, pmax, vmax, u_total, m, articulos_mochila)

def main():
    n = 100
    a = [i for i in range(1, n + 1)]
    p = [r.randint(1, 100) for i in range(n)]
    v = [r.randint(1, 100) for i in range(n)]
    u = [r.randint(1, 100) for i in range(n)]
    pmax = n*10
    vmax = n*10

    print("Número de artículos:", n)
    print("Lista de artículos:", a)
    print("Lista de pesos:", p)
    print("Lista de volúmenes:", v)
    print("Lista de utilidades:", u)
    print("Peso máximo:", pmax)
    print("Volumen máximo:", vmax)

    u_total, m, articulos_mochila = mochila(n, a, p, v, u, pmax, vmax, 0, 0, [])
```

```
    print("Número de artículos en la mochila:", m)
    print("Lista de artículos en la mochila:", articulos_mochila)
    print("Utilidad máxima obtenida:", u_total)

if __name__ == "__main__":
    main()
```