Práctica 2: Programación dinámica y método voraz para problema de la mochila y cambio.

Acevedo Juárez Sebastián, Ávila Mejía Daniel Said

15 de Octubre de 2021

1. Introducción

En el estudio de la programación, nos encontramos con varios problemas a los cuales les tenemos que encontrar una solución para obtener un resultado óptimo. Estos problemas pueden ser empleados en la vida diaria en otros aspectos de la vida. En esta práctica, compararemos distintos métodos para resolver dos problemas bien conocidos como son el problema de la mochila(Knapsack problem) y el problema del cambio.

2. Marco teórico

La programación dinámica es un método en el que un problema es dividido en sub-problemas para que en conjunto, se encuentre una solución. Tiene como finalidad encontrar una solución de un problema de optimización en forma secuencial. Del mismo modo, se le considera como un método de optimización que se puede emplear para la solución de problemas que involucran me temática aplicada.

La solución de problemas mediante la programación dinámica se basa en el llamado principio de óptimo, enunciado por Bellman(1957) y que dice: "En una secuencia de decisiones óptima toda sub-secuencia ha de ser también óptima".

Hay muchos factores a considerar si deseamos emplear programación dinámica para alguna problemática:

- La capacidad del problema para ser dividido en etapas.
- Tener etapas asociadas
- La decisión óptima de cada etapa depende únicamente del estado actual.
- La decisión tomada en una etapa, determina el estado de la siguiente.

Cuando nos referimos a un algoritmo voraz, del mismo modo que la programación dinámica, estamos hablando de la resolución de un problema de optimización. Tomas de decisiones óptimas locales en cada paso, de manera muy eficiente, con la esperanza de poder encontrar un óptimo global tras una serie de pasos.

Este método es empleado por pasos:

1. Se inicia con una solución vacía y de un conjunto de candidatos que formarán parter de la solución.
2. Durante cada paso, se escoge el siguiente elemento para añadir a las solución, entre los candidatos.(Una vez tomada la decisión no se podrá deshacer).
3. El proceso finaliza cuando el conjunto de elementos seleccionados son una solución.

A diferencia de la programación dinámica, no siempre arrojan la solución más óptima, sin embargo, son más sencillos y rápidos de implementar.

El problema de devolver el cambio consiste en desarrollar un algoritmo para pagar una cierta cantidad de dinero a un cliente, empelando el menor número posible de monedas.

El problema de la mochila (o en inglés Knapsack Problem) indica lo siguiente: Nos dan n objetos y una mochila. Para $i = 1, 2, \ldots, n$, el objeto i tiene un peso positivo w_i y valor positivo v_i. La mochila puede llevar un peso que no sobrepase W. Nuestro objetivo es llenar la mochila de tal manera que se maximice el valor de los objetos transportados, respetando la limitación de capacidad de la mochila. En una versión del problema se puede suponer que se puede tomar un trozo de los objetos, de manera que podamos decidir llevar solo una fracción x de estos (la versión del problema que no permite fraccionar objetos es más compleja). En este caso el objeto i contribuye en x, w, al peso total de la mochila, y en, x, v al valor de la carga. Matemáticamente el problema esta definido por:

$$maximizar \sum_{i=1}^{n} x_i v_i \tag{1}$$

sujeto a

$$\sum_{i=1}^{n} x_i w_i \leq W \tag{2}$$

Donde $v_i > 0$, $w_i > 0$ y $0 \leq x_i \leq 1$ para $1 \leq i \leq n$.

3. Pruebas

Para todos los problemas, se implmentó un archivo .txt donde vienen los datos necesarios para la ejecución de los algoritmos. Aunado a esto, se utilizaron las librerías *timeit* y *matplotlib* para tomar el tiempo de ejecución y realizar las gráficas correspondientes, respectivamente.

3.1. Problema de la mochila (Knapsack Problem)

Para el caso de la programación dinámica, se implementó de la siguiente manera:

```
def mochila(p_mochila, p_articulos, valor, n):
    K = [[0 for w in range(p_mochila + 1)]for i in range(n + 1)]
    for i in range(n + 1):
        for j in range(p_mochila + 1):
            if i == 0 or j == 0:
                K[i][j] = 0
            elif p_articulos[i - 1] <= j:
                K[i][j] = max(valor[i-1]+K[i-1][j-p_articulos[i-1]],K[i-1][j])
            else:
                K[i][j] = K[i - 1][j]
    res = K[n][p_mochila]
    print("Cantidad de ganancia maxima: ",res)

    j = p_mochila
    for i in range(n, 0, -1):
        if res <= 0:
            break
        if res == K[i - 1][j]:
            continue
        else:
            print("Peso del articulo: ", p_articulos[i - 1], end=" || ")
            print("Valor del articulo: ",valor[i - 1])
            res = res - valor[i - 1]
            j = j - p_articulos[i - 1]
```

En el caso de método voraz, se implementó de la siguiente manera:

```
def greedy_knapsack(objetos, p_mochila,n):
    objetos = sorted(objetos, key=lambda objetos:objetos[0], reverse=True)
    new_objetos = objetos
    res = 0
    for i in range(0,n):
        p_articulos,valor = objetos[i]
        cantidad = (p_mochila-p_mochila%p_articulos)/p_articulos
        new_objetos[i] = (cantidad,valor)
        p_mochila = p_mochila%p_articulos
        res += cantidad*valor
    print("Cantidad de ganancia m xima:",round(res,2))
    for j in range (0,n):
        if new_objetos[j][0] != 0:
            print("Cantidad",new_objetos[j][0], end=" || ")
            print("Valor",new_objetos[j][1])
```

Para ambos métodos, se emplearon los siguientes datos(en los artículos, el primer dígito representa el peso, y el segundo el valor):

- Peso Máximo: 11
- Articulo: 1 1
- Articulo: 2 6
- Articulo: 5 18
- Articulo: 6 22
- Articulo: 7 28

Se realizaron 5 pruebas para poder llegar a una conclusión más precisa.Al realizar las pruebas, nuestro programa arrojó la siguiente gráfica:

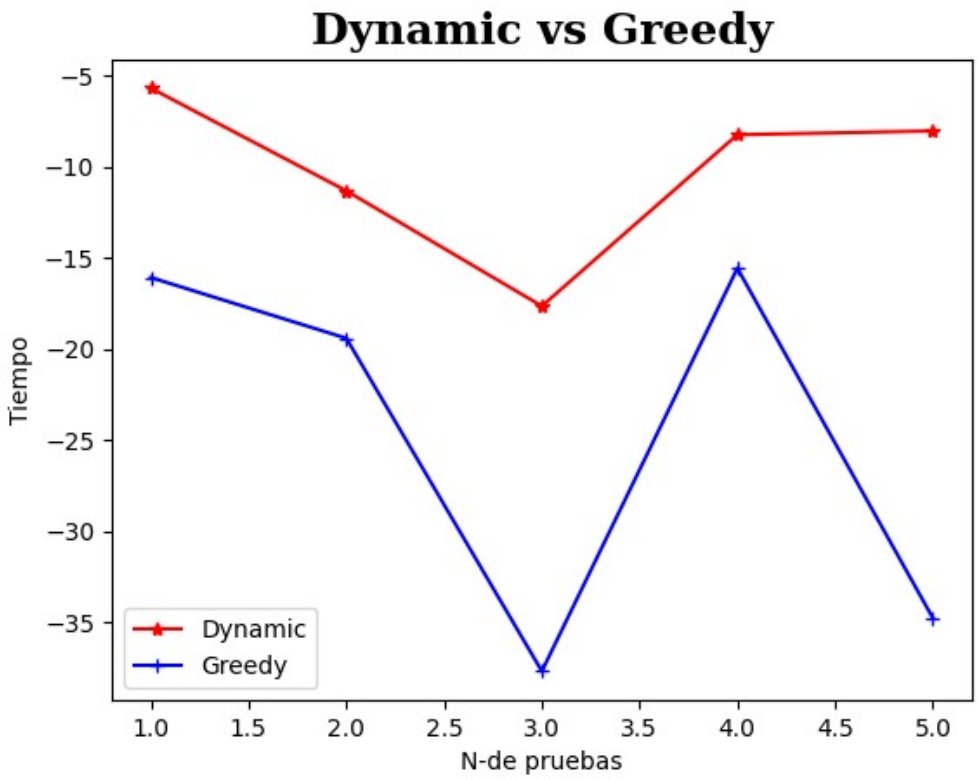

Figura 1: Gráfica comparativa entre programación dinámica y algoritmo voraz para el problema de la mochila.

3.2. Problema del cambio

Para el caso de la programación dinámica, se implementó de la siguiente manera:

```
for denomination in d:
  sol[denomination]=0
acc = 0
i = len(d)-1
while acc<N:
  if (N-acc) >= d[i]:
    acc += d[i]
    sol[d[i]] += 1
  else:
    i -= 1
m = np.zeros( (len(d), N+1) )
def cambioDyn(k):
  for col in range (1,N+1):
    m[0][col]=col
  for row in range (1, len(d)):
    for col in range (1, N+1):
      if d[row]>col:
        m[row][col] = m[row-1][col]
      else:
        m[row][col] = min( m[row-1][col], m[row][col-d[row]]+1 )
```

En el caso de método voraz, se implementó de la siguiente manera:

```
def cambioVora(V,deno):
    n = len(deno)
    res = []
    i = n - 1
    while(i >= 0):
        while (V >= deno[i]):
            V -= deno[i]
            res.append(deno[i])
        i -= 1
    for i in range(len(res)):
        print(res[i], end = " ")
```

Para ambos métodos, se emplearon los siguientes datos:

- Valor a encontrar: 15
- Denominaciones:
  - $1
  - $2
  - $5
  - $10

Se realizaron 5 pruebas para poder llegar a una conclusión más precisa .Al realizar las pruebas, nuestro programa arrojó la siguiente gráfica:

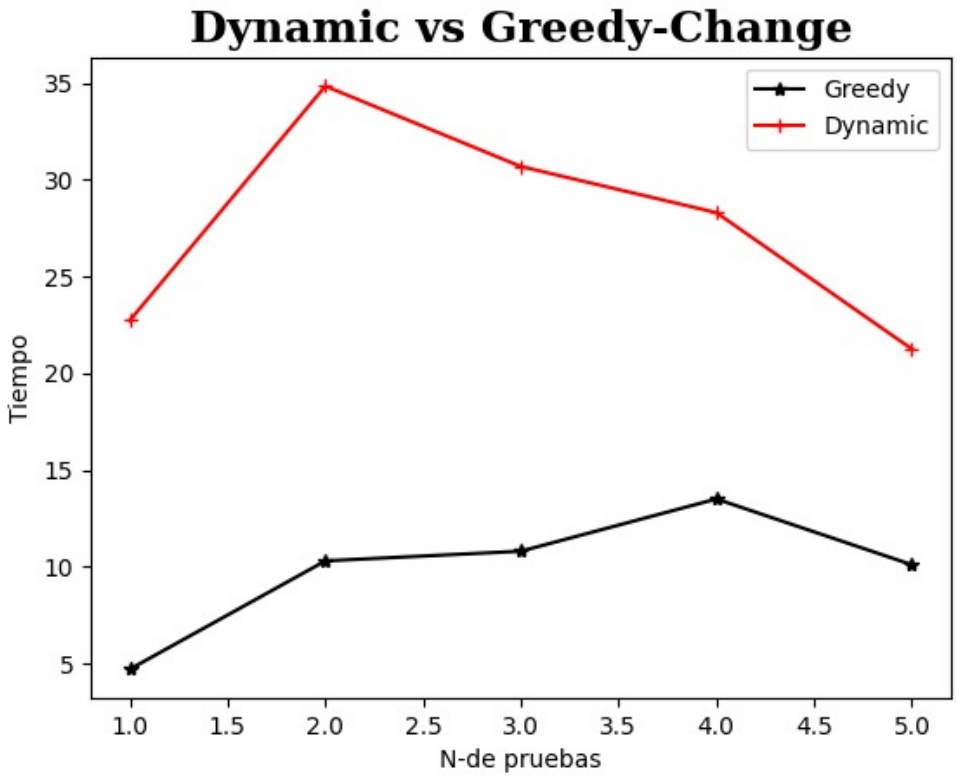

Figura 2: Gráfica comparativa entre programación dinámica y algoritmo voraz para el problema del cambio.

4. Conclusiones

A lo largo de esta practica se dio a la tarea de analizar la implementación de algoritmos dinámicos vs voraces; esto fue mediante el planteamiento de dos problemas, "Problema de la Mochila" y "Problema del cambio". Como se menciono en el marco teórico ambos buscan la solución más óptima en análisis de datos. Nos dimos cuenta de que, pese a que el planteamiento es muy diferente, las soluciones son muy similares; en términos de la implementación tuvimos momentos en el que la programación de uno parecía prima del otro, tanto en el voraz como en el dinámico. Tras implementarlos y ponerlos a trabar con los mismos datos si se encontraron discrepancias en los resultados, ya que los algoritmos dinámicos tendían a obtener resultados más rápidos. Con esto podemos determinar que para estos dos problemas en particular el método de programación Dinámico resulto ser mas eficiente, es importante resaltar que esta conclusión fue bajo un cierto lenguaje de programación (Python) y valores dados.

Referencias

UADEH (2019). Problema de la mochila (Knaspack problem). Universidad Autónoma del Estado de Hidalgo. https://www.uaeh.edu.mx/scige/boletin/tlahuelilpan/n6/e2.html

Algoritmos Voraces — Aprende Programación Competitiva. (2019). Aprende Programación Competitiva. https://aprende.olimpiada-informatica.org/algoritmia-voraz

UNAM, Flores, I. (2015). Programación Dinámica. https://www.ingenieria.unam.mx/sistemas/PDF/Sesion6$_I daliaFlores_2 0abr15.pdf$