Análisis y Diseño de Algoritmos

Programación Dinámica

Francisco Javier Mercader Martínez
Pedro Alarcón Fuentes

Introducción

En esta memoria se presenta la resolución del problema **"La ruta de la tapUPCT"** mediante la aplicación de la técnica de Programación Dinámica (PD). El objetivo es diseñar un algoritmo que permite a un estudiante maximizar el número total de tenedores obtenidos al consumir tapas en diferentes bares, sin superar el presupuesto disponible de `E` euros y respetando la restricción de consumir como máximo 3 tapas por bar.

1. Ecuación de recurrencia

Sea $dp[i][e]$ el máximo número de tenedores que se pueden obtener utilizando los primeros i bares con un presupuesto total e e euros.

Casos base:

- Para todo e entre 0 y E:
  - $dp[0][e] = 0$ (no se han considerado bares, por lo tanto, no se pueden obtener tenedores).

Recurrencia:

Para i desde 1 hasta N (cada bar):

- Para e desde 0 hasta E (cada presupuesto posible):
  - $dp[i][e] = dp[i-1][e]$ (inicialmente, no se toman del bar i).
  - Para k desde 1 hasta 3 (número de tapas que podemos tomar del bar i):
    - Si $e \geq k \cdot p_i$: $dp[i][e] = \max(dp[i-1][e], dp[i-1][e - k \cdot p_i] + k \cdot t_i)$.
    - Si no, se rompe el bucle (no podemos permitirnos más tapas en este bar con el presupuesto e).

Casos imposibles:

- $e < 0$, $dp[i][e]$ no está definido, ya que no puede obtener un presupuesto negativo.
- No se consideran $k > 3$ ya que el límite es de 3 tapas por bar.

A continuación, se presenta el pseudocódigo que describe la solución al problema utilizando programación dinámica. Este pseudocódigo nos proporcionará una visión general de los pasos necesarios antes de intentar resolver el problema implementando Python.

Definiciones previas:

- `N:` Número total de bares.
- `E:` Presupuesto total en euros
- `p[1..N]:` Arreglo que contiene el precio de una tapa de cada bar.
- `t[1..N]:` Arreglo que contiene el número de tenedores por tapa en cada bar.
- `K:` Número máximo de tapas por bar (en este caso, 3).
- `dp[0..N][0..E]:` Matriz de programación dinámica para almacenar el máximo número de tenedores.

Construcción de la tabla `dp`:

Procedimiento CONSTRUIR_TABLA(N, E, p, t)
Crear una matriz dp de dimensiones $(N+1) \times (E+1)$ inicializada en cero.

Para $i \leftarrow 1..N$ **hacer**
Para $e \leftarrow 0..E$ **hacer**
$dp[i][e] \leftarrow dp[i-1][e]$ // No consumir tapas en el bar i

Para $k \leftarrow 1..K$ **hacer**
$coste \leftarrow k \times p[i]$
Si $e \geq coste$ **entonces**
$valor \leftarrow dp[i-1][e-coste] + k \times t[i]$
Si $valor > dp[i][e]$ **entonces**
$dp[i][e] \leftarrow valor$
Fin Si
Fin Si
Fin Para
Fin Para
Fin Para
Retornar dp
Fin Procedimiento

Reconstrucción de la solución óptima:

Procedimiento RECONSTRUIR_SOLUCION(N, E, p, t)
Crear una arreglo $tapas\_por\_bar[1 \ldots N]$ inicializado en cero
$e \leftarrow E$ // Presupuesto restante

Para $i \leftarrow N$ **hasta** 1 **hacer**
Si $dp[i][e] \neq dp[i-1][e]$ **entonces**
Para $k \leftarrow 1$ **hasta** K **hacer**
$coste \leftarrow k \times p[i]$
Si $e \geq coste$ **y** $dp[i][e] = dp[i-1][e-coste] + k \times t[i]$ **entonces**
$tapas\_por\_bar[i] \leftarrow k$
$e \leftarrow e - coste$
Romper el bucle
Fin Si
Fin Para

Fin Si
Fin Para
Retornar *tapas_por_bar*
Fin Procedimiento

Descripción del pseudocódigo:

- **Procedimiento `contruir_tabla`:**
  - **Objetivo:** Construir la tabla **`dp`** que almacena el máximo número de tenedores que se pueden obtener para cada combinación de bares y presupuesto.
  - **Proceso:**
    - Se inicializa la matriz **`dp`** en cero.
    - Para cada bar **`i`** y cada posible presupuesto **`e`**, se considera:
      - **No consumir tapas en el bar `i`:** Se copia el valor de **`dp[i - 1][e]`**.
      - **Consumir `k` tapas en el bar `i`:** Se calcula el nuevo valor y se actualiza **`dp[i][e]`** si es mayor que el valor actual.
- **Procedimiento `reconstruir_solucion`:**
  - **Objetivo:** Determinar cuántas tapas se han consumido en cada bar para alcanzar el máximo número de tenedores.
  - **Proceso:**
    - Se inicializa el arreglo **`tapas_por_bar`** en ceros.
    - Se recorre la matriz **`dp`** de atrás hacia adelante (desde **`N`** hasta **`i`**).
    - Si el valor actual es diferente al de la fila anterior, significa que se han consumido tapas en ese bar.
    - Se determina cuántas tapas se consumieron probando valores de **`k`** que satisfagan la condición en **`dp`**.

---

2. Implementación del algoritmo de construcción (ascendente) de la tabla

```python
def construir_tabla(N, E, p, t):
    """
    Construye la tabla de programación dinámica para maximizar el número de tenedores.

    :param N: Número de bares
    :param E: Presupuesto total
    :param p: Lista de precios de cada bar (índice desde 0)
    :param t: Lista de tenedores de cada bar (índice desde 0)
    :return: Tabla dp con los valores máximos de tenedores para cada subproblema.
    """

    # Inicializar la tabla dp con ceros
    dp = [[0] * (E + 1) for _ in range(N + 1)]
```

```
    # Construcción de la tabla dp
    for i in range(1, N + 1):
        pi = p[i - 1]
        ti = t[i - 1]
        for e in range(E + 1):
            dp[i][e] = dp[i - 1][e] # No tomar tapas del bar i
            for k in range(1, 4):   # Tomar hasta 3 tapas del bar i
                coste = k * pi
                if e >= coste:
                    valor = dp[i - 1][e - coste] + k * ti
                    if valor > dp[i][e]:
                        dp[i][e] = valor
                else:
                    break   # No podemos permitirnos más tapas en este bar

    return dp
```

Descripción General:

La función **construir_tabla** implementa la parte principal del algoritmo de programación dinámica para resolver el problema de maximizar el número de tenedores obtenidos al consumir tapas en diferentes bares, respetando las restricciones de presupuesto y límite de tapas por bar.

Parámetros:

- **N:** Número total de bares disponibles en la guía tapUPCT.
- **E:** Presupuesto total disponible en euros para gastar en tapas.
- **p:** Lista que contiene el precio de una tapa en cada bar. El índice **i** corresponde al bar número **i + 1** . Por ejemplo, **p[0]** es el precio de una tapa en el bar 1.
- **t:** Lista que contiene el número de tenedores que otorga cada tapa en cada bar. De la misma forma, **t[0]** es el número de tenedores de una tapa en el bar 1.

Resultado:

- **dp:** Matriz bidimensional de tamaño **(N + 1) x (E + 1)** que representa la tabla de programación dinámica. En esta tabla, **dp[i][e]** almacena el máximo número de tenedores que se pueden obtener al considerar los primeros **i** bares con un presupuesto total de **e** euros.

---

3. Implementación del algoritmo de reconstrucción de la solución a partir de la tabla calculada por el algoritmo del paso 2

```
def reconstruir_solucion(N, E, p, t, dp):
    """
    Reconstruye la solución óptima a partir de la tabla dp.

    :param N: Número de bares.
    :param E: Presupuesto total.
```

```
    :param p: Lista de precios de cada bar.
    :param t: Lista de tenedores de cada bar.
    :param dp: Tabla dp construida previamente
    :return: Lista con el número de tapas tomadas en cada bar
    """
    # Inicializar la lista de tapas por bar
    tapas_por_bar = [0] * N
    e = E
    for i in range(N, 0, -1):
        if dp[i][e] != dp[i - 1][e]:
            # Tomamos alguna tapa del bar ir
            pi = p[i - 1]
            ti = t[i - 1]
            for k in range(1, 4):
                coste = k * pi
                if e >= coste and dp[i][e] == dp[i - 1][e - coste] + k * ti:
                    tapas_por_bar[i - 1] = k
                    e -= coste
                    break
    return tapas_por_bar
```

Descripción General:

La función **reconstruir_solucion** utiliza la tabla **dp** previamente calculada para determinar exactamente cuántas tapas se deben consumir en cada bar para lograr el máximo número de tenedores obtenido.

Parámetros:

- **N:** Número total de bares.
- **E:** Presupuesto total disponible en euros.
- **p:** Lista que contiene el precio de una tapa en cada bar.
- **t:** Lista que contiene el número de tenedores que otorga cada tapa en cada bar.
- **dp:** Tabla de programación dinámica obtenida de **construir_tabla**.

Resultado:

- **tapas_por_bar:** Lista donde cada elemento **tapas_por_bar[i]** indica el número de tapas que se deben consumir en el bar **i+1** para lograr la solución óptima. Por ejemplo, **tapas_por_bar[0]** es el número de tapas en el bar 1.

---

4. Validación mediante ejemplos sencillos

Ejemplo 1:

- **Presupuesto (E):** 10 euros

- **Número de bares (`N`):** 2
- **Bar 1:**
  - Precio por tapa ($p_1 = 2$ euros)
  - Tenedores por tapa ($t_1 = 3$)
- **Bar 2:**
  - Precio por tapa ($p_2 = 3$ euros)
  - Tenedores por tapa ($t_2 = 4$)

Aplicación del algoritmo:

```
N = 2
E = 10
p = [2, 3]
t = [3, 4]

dp = construir_tabla(N, E, p, t)
tapas_por_bar = reconstruir_solucion(N, E, p, t, dp)

print("Máximo número de tenedores:", dp[N][E])
```

```
## Máximo número de tenedores: 14
```

```
print("Tapas por bar:", tapas_por_bar)
```

```
## Tapas por bar: [2, 2]
```

Análisis:

- **Bar 1:** 2 tapas $\times$ 2 euros = 4 euros, 2 tapas $\times$ 3 tenedores = 6 tenedores
- **Bar 2:** 2 tapas $\times$ 3 euros = 6 euros, 2 tapas $\times$ 4 tenedores = 8 tenedores
- **Coste total:** $4 + 6 = 10$ euros
- **Tenedores totales:** $6 + 8 = 14$ tenedores

Justificación de la óptimalidad:

- No es posible obtener más de 14 tenedores sin exceder el presupuesto.
- Cualquier otra combinación resultaría en menos tenedores o excedería el presupuesto.

Muestra del algoritmo con variables aleatorias para mostrar que el algoritmo funciona para varios casos:

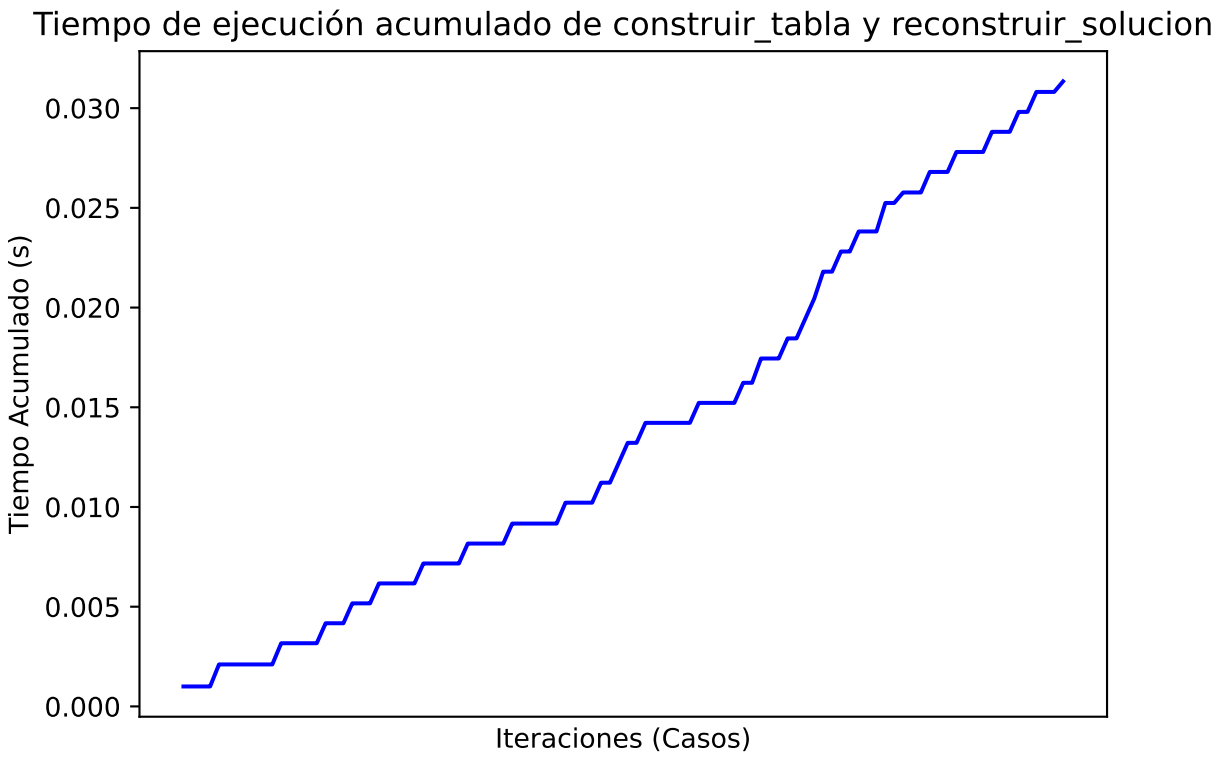
```python
for i in range(10):
    print(f"-- Prueba {i + 1} --")
    N = random.randint(1, 10)  # Número de bares entre 1 y 10
    E = random.randint(10, 100)  # Presupuesto entre 10 y 100
    p = [random.randint(1, 10) for _ in range(N)]  # Precios aleatorios entre 1 y 10
    t = [random.randint(1, 5) for _ in range(N)]  # Tenedores aleatorios entre 1 y 5

    dp = construir_tabla(N, E, p, t)
    tapas_por_bar = reconstruir_solucion(N, E, p, t, dp)

    print(f"Número de bares: {N}")
    print(f"Presupuesto total: {E} euros")
    print(f"Precios por bar: {p}")
    print(f"Tenedores por bar: {t}")
    print(f"Tapas por bar: {tapas_por_bar}")
    print(f"Máximo número de tenedores: {dp[N][E]}")
    print()
```

```
## -- Prueba 1 --
## Número de bares: 6
## Presupuesto total: 74 euros
## Precios por bar: [6, 8, 7, 5, 1, 9]
## Tenedores por bar: [4, 3, 4, 2, 5, 4]
## Tapas por bar: [3, 0, 3, 1, 3, 3]
## Máximo número de tenedores: 53
##
## -- Prueba 2 --
## Número de bares: 9
## Presupuesto total: 46 euros
## Precios por bar: [1, 3, 5, 5, 7, 1, 5, 9, 4]
## Tenedores por bar: [5, 4, 5, 3, 5, 2, 3, 3, 4]
## Tapas por bar: [3, 3, 3, 0, 1, 3, 0, 0, 2]
## Máximo número de tenedores: 61
##
## -- Prueba 3 --
## Número de bares: 5
## Presupuesto total: 98 euros
## Precios por bar: [4, 4, 9, 9, 9]
## Tenedores por bar: [5, 5, 1, 5, 4]
## Tapas por bar: [3, 3, 2, 3, 3]
## Máximo número de tenedores: 59
##
## -- Prueba 4 --
## Número de bares: 8
## Presupuesto total: 99 euros
## Precios por bar: [2, 1, 10, 10, 9, 8, 6, 8]
## Tenedores por bar: [5, 2, 1, 2, 2, 5, 2, 1]
## Tapas por bar: [3, 3, 0, 3, 2, 3, 3, 0]
## Máximo número de tenedores: 52
##
## -- Prueba 5 --
## Número de bares: 6
```

```
## Presupuesto total: 50 euros
## Precios por bar: [2, 2, 7, 3, 1, 5]
## Tenedores por bar: [3, 2, 2, 3, 1, 5]
## Tapas por bar: [3, 3, 1, 3, 3, 3]
## Máximo número de tenedores: 44
##
## -- Prueba 6 --
## Número de bares: 2
## Presupuesto total: 29 euros
## Precios por bar: [1, 4]
## Tenedores por bar: [3, 3]
## Tapas por bar: [3, 3]
## Máximo número de tenedores: 18
##
## -- Prueba 7 --
## Número de bares: 7
## Presupuesto total: 93 euros
## Precios por bar: [8, 2, 2, 9, 10, 4, 8]
## Tenedores por bar: [1, 3, 4, 5, 1, 3, 3]
## Tapas por bar: [2, 3, 3, 3, 0, 3, 3]
## Máximo número de tenedores: 56
##
## -- Prueba 8 --
## Número de bares: 2
## Presupuesto total: 65 euros
## Precios por bar: [2, 4]
## Tenedores por bar: [3, 5]
## Tapas por bar: [3, 3]
## Máximo número de tenedores: 24
##
## -- Prueba 9 --
## Número de bares: 5
## Presupuesto total: 60 euros
## Precios por bar: [6, 3, 3, 7, 9]
## Tenedores por bar: [3, 3, 5, 4, 1]
## Tapas por bar: [3, 3, 3, 3, 0]
## Máximo número de tenedores: 45
##
## -- Prueba 10 --
## Número de bares: 2
## Presupuesto total: 17 euros
## Precios por bar: [9, 1]
## Tenedores por bar: [4, 4]
## Tapas por bar: [1, 3]
## Máximo número de tenedores: 16
```



---

5. Estudio teórico del tiempo de ejecución

Orden de complejidad temporal:

- **Construcción de la tabla `dp`:** $O(N \times E \times K)$, donde:
  - N es el número de bares.
  - E es el presupuesto total.
  - K es el número de tapas por bar (constante, $K = 3$)

  K es número constante por lo que no entra en el orden de ejecución y como N y E son dos variable determinadas, cota superior tiene un orden de $O(n^2)$.

- **Reconstrucción de la solución:** $O(N \times K)$, pero dado que K es constante, es $O(N)$.

Casos peor, mejor y exacto:

- **Caso peor:** $O(n^2)$
- **Caso mejor:** $O(n^2)$
- **Caso exacto:** $O(n^2)$