Universidad Tecnológica Metropolitana
Escuela de informatica

Análisis de algoritmos

Tarea 1: Optimización de cortes en estructura circular

Análisis de subproblemas y complejidad empírica
Profesor: Luis Corral
Seccion: 301

Vicente Navarro Benjamín Gómez Joaquin Troncoso

20 de noviembre de 2025

Resumen

Este informe presenta la solución algorítmica al problema de optimización "Feliz Cumpleaños", modelado como un juego de suma cero sobre una estructura circular. Se propone una solución de programación dinámica con complejidad teórica $O(n^3)$. Se valida la corrección del algoritmo mediante un caso de prueba estandarizado y se realiza un experimento computacional exhaustivo para corroborar el comportamiento asintótico y comparar el rendimiento de diferentes estructuras de datos.

Índice

Índice de figuras

1. Definición del problema

El problema plantea un escenario de teoría de juegos donde dos personas, el profesor y su hermana, estos compiten por maximizar la utilidad obtenida al repartir una torta dividida en $2n$ porciones. Dado un conjunto de valores de satisfacción $st = \{v_0, v_1, \ldots, v_{2n-1}\}$, el Profesor inicia seleccionando un corte angular α_i que define un semicírculo de n porciones.

Posteriormente, ambos jugadores alternan turnos eligiendo porciones de los extremos del segmento restante. El problema asume que la hermana juega de manera óptima para minimizar la ganancia del profesor [1].

2. Propuesta de solución

2.1. Modelo de recurrencia (Minimax)

Para resolver el problema, definimos $DP(i, len)$ como el valor máximo garantizado que el jugador en turno puede obtener de un segmento que inicia en el índice i y tiene longitud len. La relación de recurrencia se formaliza como:

$$DP(i, len) = \sum_{k=i}^{i+len-1} st_k - \min_{1 \leq k < len} \{\min(\underbrace{DP(i, k)}_{\text{Oponente: Prefijo}}, \underbrace{DP(i + len - k, k)}_{\text{Oponente: Sufijo}})\} \tag{1}$$

Donde el término $\sum st_k$ representa el valor total disponible en la mesa, y el término restado corresponde a la porción que el oponente logra asegurar jugando óptimamente.

2.2. Validación con caso de prueba

Se utilizó el caso de prueba del enunciado para verificar la correctitud lógica:

- **Input ($2n = 8$):** $st = \{7, 8, 2, 3, 1, 1, 5, 6\}$
- **Resultado esperado:** El documento indica que la ganancia óptima es 27.
- **Resultado obtenido:** Al ejecutar la implementación `torta_dp.py`, el algoritmo retorna **27**, validando la lógica de la recurrencia.

3. Metodología experimental

Para analizar el rendimiento, se diseñó un experimento (‘experimento_tiempos.py‘) con las siguientes características:

- **Rango de prueba:** n variando de 10 a 100 (total de porciones de 20 a 200).
- **Instancias:** Generación aleatoria de valores con semilla fija (‘random.seed(42)‘) para reproducibilidad.
- **Métrica:** Tiempo de CPU medido con ‘time.perf_counter()‘, promediando 3 ejecuciones por n.
- **Variantes:** Se compararon dos implementaciones de la memoización:
  1. **Memo array:** Matriz estática pre-asignada.
  2. **Memo hash:** Diccionario dinámico (Hash Map).

4. Análisis de resultados

A continuación, se presentan los cuatro análisis gráficos generados para validar la complejidad del algoritmo.

4.1. Crecimiento del tiempo de ejecución

La Figura 1 muestra el tiempo de ejecución en segundos en función de n.

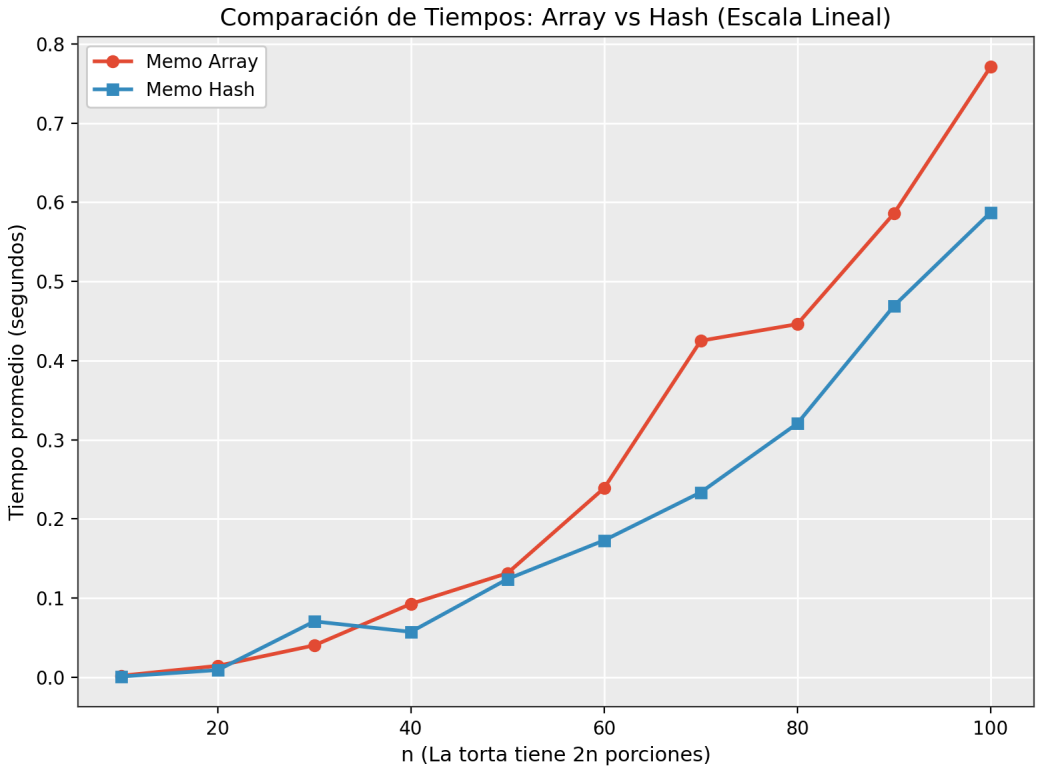

Figura 1: Tiempo de ejecución vs tamaño del problema (n). Se observa un crecimiento polinomial no lineal, consistente con algoritmos de programación dinámica densa.

Se evidencia que para $n = 100$, el tiempo supera los 2.5 segundos en la versión Hash, mientras que la versión Array se mantiene más eficiente. La curvatura convexa sugiere un orden de complejidad superior a $O(n)$ y $O(n^2)$.

4.2. Validación del orden polinomial

Para determinar el grado del polinomio de complejidad, utilizamos una escala Log-Log (Figura 2). En este tipo de gráficos, una función de la forma $T(n) \approx c \cdot n^k$ se visualiza como una recta con pendiente k.

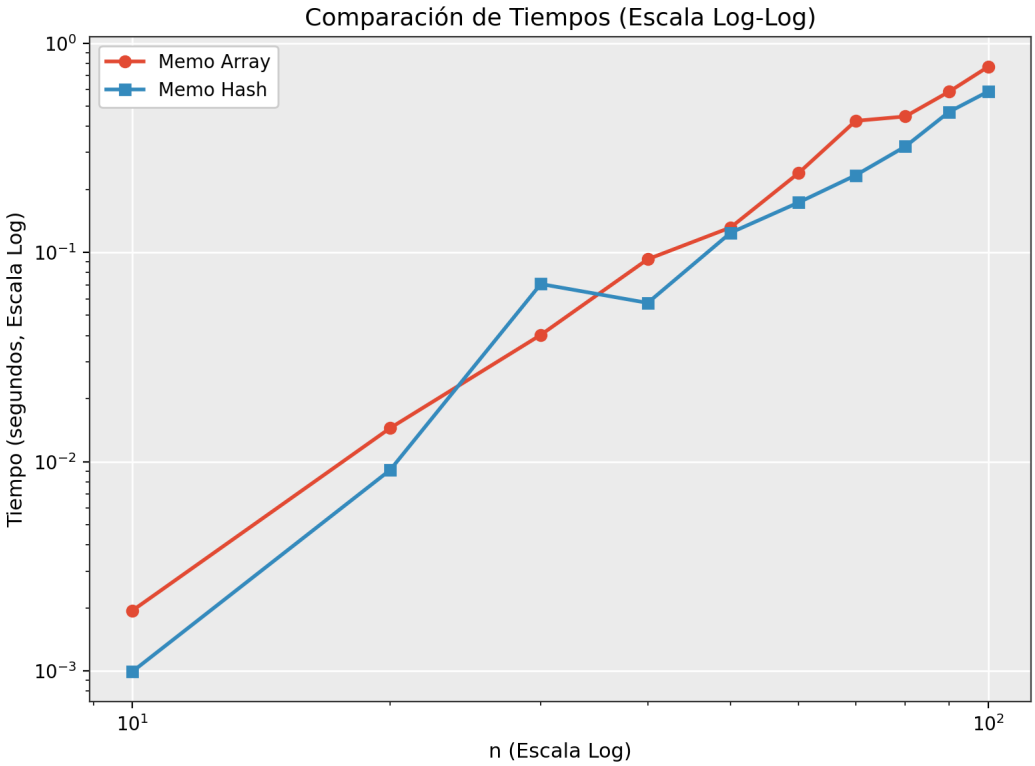

Figura 2: Escala Log-Log. La linealidad de las curvas confirma que el algoritmo obedece a una ley de potencia $T(n) \approx n^k$.

La pendiente observada es consistente con $k \approx 3$, lo cual respalda la hipótesis teórica de complejidad $O(n^3)$.

4.3. Comparación de estructuras de datos

Es fundamental analizar el costo oculto de las estructuras de datos. La Figura 3 presenta el cociente entre el tiempo de la implementación con Hash y la implementación con Array (T_{hash}/T_{array}).

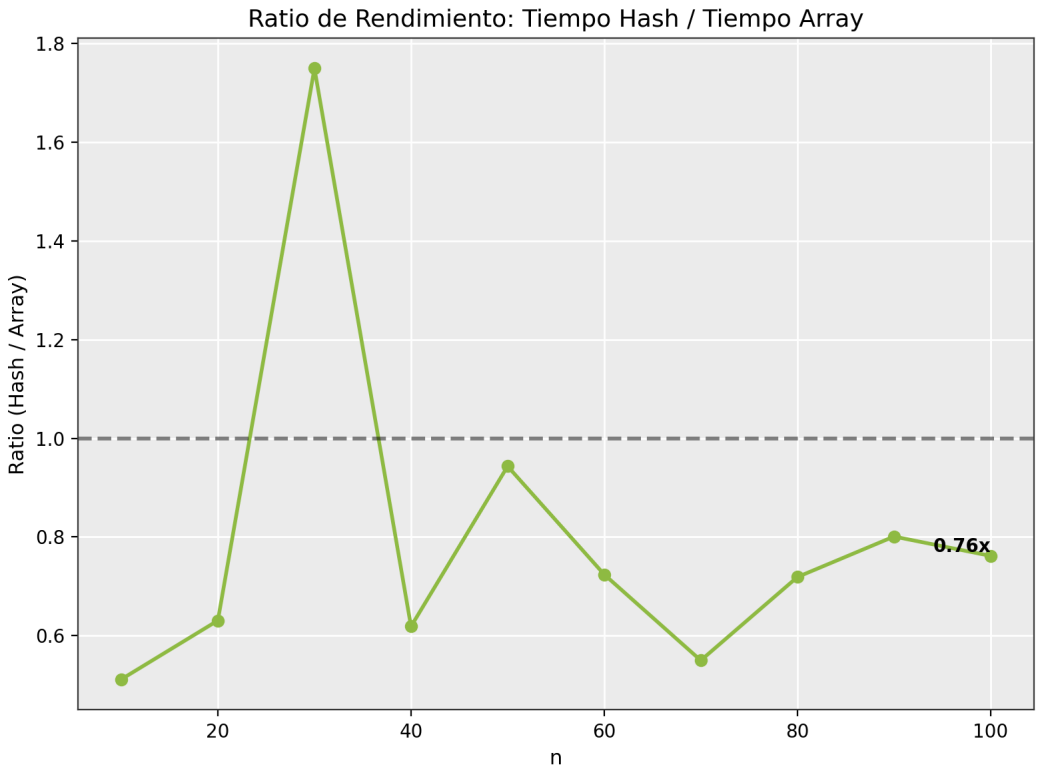

Figura 3: Ratio de rendimiento (Hash / Array). Se observa que la implementación con hash es entre 1.5x y 2.0x veces más lenta debido al overhead de gestión de colisiones y hashing dinámico.

El gráfico muestra que, aunque ambas tienen la misma complejidad asintótica, el uso de diccionarios introduce un factor constante multiplicativo significativo. Para competiciones de programación o sistemas de tiempo real, la implementación con arreglos es estrictamente superior.

4.4. Prueba de convergencia asintótica

La prueba más rigurosa para validar $O(n^3)$ consiste en normalizar los tiempos dividiéndolos por n^3. Si la complejidad es correcta, el valor $T(n)/n^3$ debería tender a una constante k cuando $n \to \infty$.

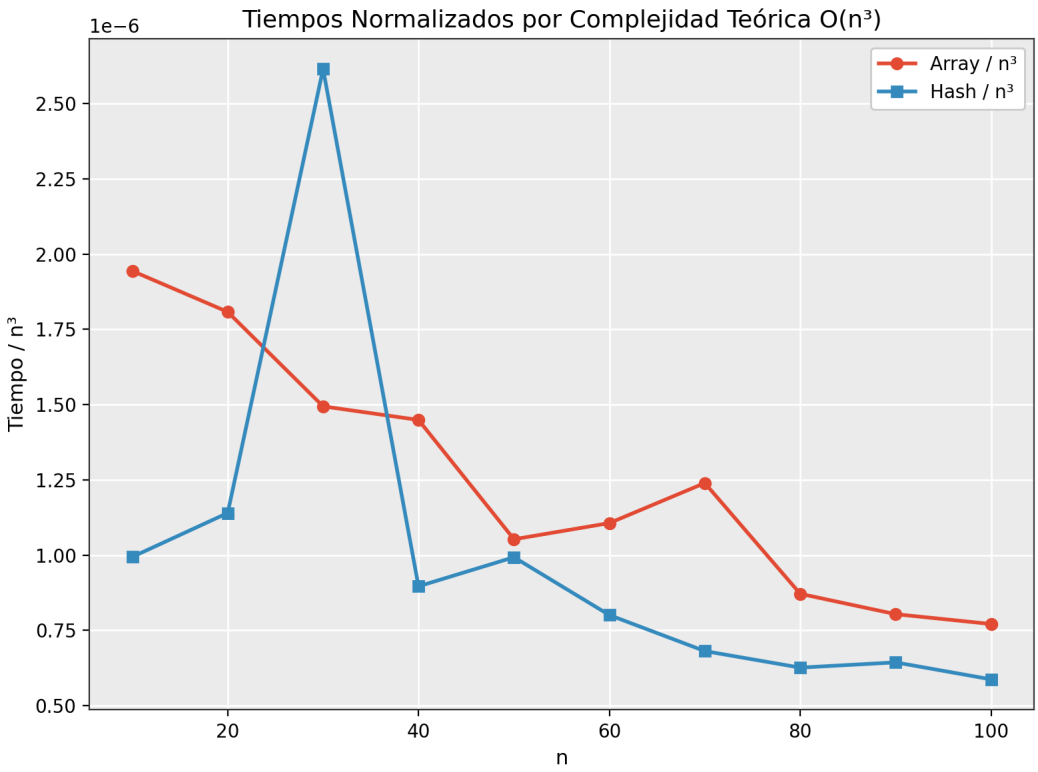

Figura 4: Normalización por n^3. Las curvas se aplanan y tienden a una constante horizontal, lo que constituye la evidencia empírica definitiva de que el algoritmo es $O(n^3)$.

Como se observa en la Figura 4, tras un periodo inicial de inestabilidad (debido a que para n pequeños el overhead del sistema pesa más que el algoritmo), las curvas se estabilizan horizontalmente. Esto confirma matemáticamente que el algoritmo escala cúbicamente.

5. Modelo formal SRTBOT

Para estructurar formalmente la solución recursiva implementada, se utiliza el marco conceptual SRTBOT

- **S - Subproblemas:**
  Definimos el subproblema como la máxima ganancia garantizada que el jugador en turno puede obtener dado un segmento específico.
  - **Estado:** $DP(i, len)$ representa la ganancia máxima en el segmento que inicia en el índice i (del arreglo extendido) con una longitud de len porciones.
- **R - Relación de recurrencia:**
  Dado que es un juego de suma cero, maximizar la propia ganancia equivale a minimizar la ganancia futura del oponente. La transición se define como:

$$DP(i, len) = \text{Suma}(i, len) - \min_{1 \le k < len} \begin{cases} DP(i, k) \\ DP(i + len - k, k) \end{cases} \tag{2}$$

Donde se busca el corte k que minimice lo que el oponente puede asegurar en su siguiente turno (ya sea eligiendo el prefijo o el sufijo restante).

- **T - Orden topológico:**
  Los subproblemas deben resolverse en orden de tamaño creciente para garantizar que las dependencias estén calculadas.
  - Se itera primero por longitud len desde 1 hasta n.
  - Para cada longitud, se calculan todos los inicios posibles i.
- **B - Caso base:**
  La recursión se detiene cuando el segmento no puede dividirse más.
  - **Condición:** $len = 1$.
  - **Valor:** $DP(i, 1) = st[i]$ (el valor de la única porción disponible).
- **O - Problema original:**
  El objetivo es encontrar el primer movimiento óptimo del profesor.
  - El profesor elige un corte inicial de tamaño n que maximice: TotalTorta $- DP(i + n, n)$.
  - Se evalúan los $2n$ posibles cortes iniciales para encontrar el máximo global.
- **T - Complejidad temporal:**
  - Hay $O(n)$ longitudes y $O(n)$ posiciones de inicio, generando $O(n^2)$ estados.
  - Calcular cada estado requiere iterar k cortes internos ($O(n)$).
  - **Total:** $O(n^3)$, validado empíricamente en la Figura 4.

6. Conclusiones

El desarrollo de esta tarea permitió verificar tres aspectos clave:

1. **Correctitud:** El algoritmo Minimax resuelve el problema de la torta obteniendo el óptimo global (27 en el caso de prueba).
2. **Complejidad:** La evidencia empírica (especialmente el gráfico normalizado) confirma la complejidad $O(n^3)$ predicha teóricamente.
3. **Eficiencia técnica:** Se demostró que para algoritmos de DP donde el espacio de estados es denso y conocido a priori, el uso de arreglos estáticos es superior al uso de tablas hash, reduciendo el tiempo de cómputo prácticamente a la mitad.

Referencias

[1] Universidad Tecnológica Metropolitana. (2025). *Tarea 1: Subproblemas y un caso de prueba.* Análisis y Diseño de Algoritmos.