República Bolivariana de Venezuela
Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Técnicas Avanzadas de Programación
Semestre 1-2025

Problema de la suma máxima de un subarreglo: *Kadane's algorithm 2D* vs. Solución por fuerza bruta

Estudiante:
César Carios
C.I 30.136.117

9 de julio del 2025

Suma máxima de un subarreglo

El problema de la suma máxima de un subarreglo (Maximum Subarray Sum Problem) es un problema clásico en la algoritmia y la programación dinámica. Dado un arreglo unidimensional de números enteros, el objetivo es encontrar un subarreglo contiguo dentro de él cuya suma de elementos sea la mayor posible. Este problema es fundamental no solo por sus aplicaciones directas, sino también porque sirve como base conceptual para problemas más complejos.

Un ejemplo ilustrativo de la suma máxima de subarreglo sería el siguiente:

Dado el arreglo $A = [-2, 1, -3, 4, -1, 2, 1, -5, 4]$, el subarreglo con la suma máxima es $[4, -1, 2, 1]$, cuya suma es 6.

Este problema se puede resolver eficientemente utilizando el **algoritmo de Kadane**, el cual tiene una complejidad temporal de $O(N)$, donde N es el número de elementos en el arreglo. La elegancia de este algoritmo radica en su capacidad para determinar la suma máxima de un subarreglo que termina en cada posición, permitiendo así encontrar la suma máxima global en una sola pasada.

La extensión natural de este problema en una dimensión es el **problema de la suma máxima de submatriz** (Maximum Sum Submatrix Problem), que es el enfoque principal de este informe. En este caso, se busca la submatriz rectangular contigua dentro de una matriz 2D de enteros que posea la mayor suma de elementos. Este problema es significativamente más complejo que su contraparte unidimensional, y su resolución eficiente a menudo se basa en la aplicación repetida de las ideas desarrolladas para el problema 1D. A continuación, exploraremos y compararemos diferentes algoritmos para abordar este desafío bidimensional.

Solución por fuerza bruta

El problema de la suma máxima de submatriz, en su concepción más directa, implica la evaluación de todas las posibles submatrices. Una submatriz se define por sus coordenadas de esquina superior izquierda (r_1, c_1) y su esquina inferior derecha (r_2, c_2).

Enfoque Ingenuo ($O(N^6)$)

El método de fuerza bruta más elemental para resolver este problema es iterar sobre todas las combinaciones posibles de estas cuatro coordenadas. Para cada submatriz definida por (r_1, c_1, r_2, c_2), se recorren todos sus elementos y se calcula su suma. Considerando una matriz cuadrada de $N \times N$:

- La selección de (r_1, c_1) implica $O(N^2)$ opciones.
- La selección de (r_2, c_2) implica $O(N^2)$ opciones.
- La suma de los elementos dentro de la submatriz seleccionada toma $O(N^2)$ tiempo (recorriendo $r_2 - r_1 + 1$ filas y $c_2 - c_1 + 1$ columnas).

Esto resulta en una complejidad temporal total de $O(N^2 \cdot N^2 \cdot N^2) = O(N^6)$. Este enfoque es computacionalmente inviable incluso para valores pequeños de N. En los archivos proporcionados, la implementación 'bruteforcekadane2D.cpp' corresponde a esta versión ingenua.

Enfoque Optimizado con Matriz Acumulada ($O(N^4)$)

Para mejorar el rendimiento del algoritmo de fuerza bruta, se puede aplicar una técnica de pre-cálculo que permite obtener la suma de cualquier submatriz en tiempo constante ($O(1)$). Esta técnica implica la construcción de una **matriz de sumas de prefijo 2D** (también conocida como matriz acumulada). Este es el enfoque implementado en 'submatrixbruteforce.cpp' y el que se compara directamente con el algoritmo de Kadane 2D.

Construcción de la Matriz Acumulada

Sea M la matriz original y A la matriz acumulada. Cada elemento $A[i][j]$ en la matriz acumulada almacena la suma de todos los elementos de M en el rectángulo que va desde $M[0][0]$ hasta $M[i][j]$. La matriz A se construye en $O(N^2)$ tiempo utilizando la siguiente relación de recurrencia:

$$A[i][j] = M[i][j] + A[i-1][j] + A[i][j-1] - A[i-1][j-1]$$

Se deben manejar los casos base apropiadamente para la primera fila ($i = 0$) y la primera columna ($j = 0$), donde los términos $A[i-1][j]$, $A[i][j-1]$ o $A[i-1][j-1]$ se consideran cero si sus índices están fuera de los límites de la matriz (ej. $A[-1][j]$ no existe, se toma como 0).

Cálculo de la Suma de Submatriz en $O(1)$

Una vez construida la matriz acumulada A, la suma de cualquier submatriz definida por su esquina superior izquierda (r_1, c_1) y su esquina inferior derecha (r_2, c_2) puede obtenerse con solo cuatro accesos a la matriz A y unas pocas operaciones aritméticas, en tiempo $O(1)$:

$$\text{Suma}(r_1, c_1, r_2, c_2) = A[r_2][c_2] - A[r_1-1][c_2] - A[r_2][c_1-1] + A[r_1-1][c_1-1]$$

De manera similar, se aplican condiciones para los casos donde $r_1 - 1 < 0$ o $c_1 - 1 < 0$, tratándolos como si los valores de A en esos índices fueran cero.

Complejidad Temporal Total del Enfoque Optimizado

La complejidad total de este algoritmo se deriva de:

- $O(N^2)$ para la construcción de la matriz acumulada.
- $O(N^4)$ para la iteración sobre todas las posibles submatrices (los cuatro bucles anidados para r_1, c_1, r_2, c_2).

- $O(1)$ para el cálculo de la suma de cada submatriz.

Por lo tanto, la complejidad dominante y total de esta versión optimizada de fuerza bruta es $O(N^4)$. Aunque es una mejora drástica con respecto a $O(N^6)$, aún es menos eficiente que el algoritmo de Kadane 2D ($O(N^3)$) para valores grandes de N.

Algoritmo de Kadane 2D

El problema de la suma máxima de submatriz se puede resolver de manera más eficiente que el enfoque de fuerza bruta optimizado a $O(N^4)$ mediante la extensión del algoritmo de Kadane 1D. Este enfoque, conocido como el algoritmo de Kadane 2D, reduce el problema bidimensional a una serie de problemas unidimensionales, cada uno de los cuales puede ser resuelto por el algoritmo de Kadane clásico.

La lógica principal detrás del algoritmo de Kadane 2D es la siguiente:

1. **Selección de Filas Delimitadoras:** Se itera sobre todas las posibles filas superiores (‘upperrow‘) y, para cada ‘upperrow‘, se itera sobre todas las posibles filas inferiores (‘lowerrow‘) que estén por debajo o en la misma posición que ‘upperrow‘. Estas dos filas (‘upperrow‘ y ‘lowerrow‘) definen un ”bloque”horizontal de filas dentro de la matriz original. Hay $O(N^2)$ combinaciones de estas filas.
2. **Reducción a Problema 1D:** Para cada par (‘upperrow‘, ‘lowerrow‘), se construye un arreglo temporal unidimensional (‘temparray‘). Cada elemento ‘temparray[col]‘ de este arreglo es la suma de todos los elementos en la columna ‘col‘ desde la ‘upperrow‘ hasta la ‘lowerrow‘. Es decir,

   $$\text{temp\_array[col]} = \sum_{\text{upper\_row}}^{\text{lower\_row}} \text{matrix}[r][\text{col}]$$

   Esta acumulación de sumas para cada columna se realiza de manera incremental: a medida que ‘lowerrow‘ avanza, simplemente se suman los valores de la nueva fila a los valores ya acumulados en ‘temparray‘.
3. **Aplicación de Kadane 1D:** Una vez que ‘temparray‘ ha sido construido para el bloque de filas actual, el problema de encontrar la submatriz con la suma máxima dentro de ese bloque se reduce a encontrar la suma máxima de un subarreglo en ‘temparray‘. Aquí es donde se aplica el algoritmo de Kadane 1D. El resultado de ‘kadane1D(temparray)‘ es la suma máxima de una submatriz que está completamente contenida entre ‘upperrow‘ y ‘lowerrow‘.
4. **Actualización de la Suma Máxima Global:** La suma máxima obtenida de ‘kadane1D(temparray)‘ se compara con la suma máxima global registrada hasta el momento, y se actualiza si es mayor.

Complejidad Temporal

La complejidad de este algoritmo es el resultado de la combinación de sus partes:

- Los dos bucles anidados para ‘upperrow‘ y ‘lowerrow‘ contribuyen con un factor de $O(N^2)$.

- Dentro de estos bucles, la construcción incremental de ‘temparray‘ para una ‘lowerrow‘ específica y la aplicación del algoritmo de Kadane 1D sobre ‘temparray‘ toman $O(N)$ tiempo cada uno (recorriendo las columnas).

Por lo tanto, la complejidad temporal total del algoritmo de Kadane 2D es $O(N^2 \cdot N) = O(N^3)$. Esta es una mejora significativa con respecto al algoritmo de fuerza bruta optimizado a $O(N^4)$, haciendo a Kadane 2D más eficiente para matrices de mayor tamaño.

Resultados y análisis de la comparación entre los dos algoritmos

Para evaluar el rendimiento de los algoritmos de Kadane 2D ($O(N^3)$) y la Fuerza Bruta Optimizada ($O(N^4)$), se realizaron pruebas exhaustivas según la metodología descrita. Los tiempos de ejecución promedio, medidos en milisegundos y promediados sobre 20 matrices aleatorias para cada tamaño N, se presentan en la tabla que está en la siguiente página:

Cuadro 1: Tiempos de Ejecución Promedio (ms) para Kadane 2D ($O(N^3)$) y Fuerza Bruta Optimizada ($O(N^4)$)

N	**Kadane 2D (ms)**	**Fuerza Bruta (ms)**
5	0.000000	0.000000
6	0.000000	0.000000
7	0.000000	0.000000
8	0.000000	0.000000
9	0.000000	0.000000
10	0.000000	0.000000
11	0.000000	0.379050
12	0.000000	0.000000
13	0.000000	0.000000
14	0.000000	0.777900
15	0.000000	0.000000
16	0.000000	0.000000
17	0.000000	0.784400
18	0.000000	0.778150
19	0.025350	0.819000
20	0.000000	0.759950
21	0.000000	0.787000
22	0.780700	0.779500
23	0.000000	1.582550
24	0.000000	1.365800
25	0.796550	2.357950
26	0.000000	3.148700
27	0.000000	3.134450
28	0.000000	3.211100
29	0.000000	3.926550
30	0.785650	4.741700
31	0.033250	4.740500
32	0.000000	5.501500
33	1.030800	6.280250
34	0.530150	8.565950
35	0.000000	7.995100
36	0.806150	8.511700
37	0.750350	11.045700
38	0.782150	11.008000
39	0.037100	11.840750
40	0.755200	14.115600
41	0.305050	15.711100
42	0.831650	16.555150
43	0.000000	18.944950
44	0.779200	19.723050
45	0.786000	21.187550
46	1.561850	23.668300
47	1.292700	25.153300
48	0.819650	29.376550
49	1.067500	29.595450
50	1.100850	33.035000

Discusión de los Resultados

Los datos presentados en la tabla 1 ilustran claramente las diferencias de rendimiento entre el algoritmo de Kadane 2D ($O(N^3)$) y la Fuerza Bruta Optimizada ($O(N^4)$).

- **Para N pequeños:** Para tamaños de matriz pequeños (aproximadamente $N \leq 18$), ambos algoritmos son extremadamente rápidos, y sus tiempos de ejecución se reportan frecuentemente como 0,000000 ms. Esto no significa que no consuman tiempo, sino que el tiempo real de ejecución es inferior a la resolución de los milisegundos de la herramienta de medición, o que el promedio de 20 ejecuciones resulta en un valor tan bajo que se redondea a cero.
- **Diferenciación de Rendimiento:** A partir de N aproximadamente igual a 19-20, el algoritmo de Fuerza Bruta Optimizada ($O(N^4)$) comienza a mostrar tiempos de ejecución medibles y crecientes. Aunque Kadane 2D también muestra algunos valores por encima de cero en este rango, aún presenta varios 0,000000 ms hasta $N = 29$. Esto puede deberse a la alta eficiencia del algoritmo para determinadas configuraciones de matrices aleatorias o a la variabilidad en el entorno de ejecución, donde algunas mediciones pueden haber ocurrido bajo una carga de sistema mínima.
- **Crecimiento Exponencial de la Fuerza Bruta:** A medida que N aumenta, especialmente de $N = 25$ en adelante, el tiempo de ejecución del algoritmo de Fuerza Bruta Optimizada crece de manera significativa. Para $N = 50$, el tiempo promedio es de 33,035000 ms.
- **Superioridad de Kadane 2D:** Por otro lado, el algoritmo de Kadane 2D mantiene tiempos de ejecución considerablemente más bajos. Para $N = 50$, su tiempo promedio es de 1,100850 ms. Esto representa que la Fuerza Bruta es aproximadamente 30 veces más lenta que Kadane 2D para este tamaño de matriz ($33{,}035000/1{,}100850 \approx 30{,}01$). Esta diferencia de rendimiento subraya la importancia de la complejidad asintótica en el diseño de algoritmos.
- **Consistencia con la Complejidad Teórica:** Los resultados empíricos confirman de manera robusta las complejidades teóricas de $O(N^3)$ para Kadane 2D y $O(N^4)$ para la Fuerza Bruta Optimizada. La curva de crecimiento del tiempo de ejecución para el algoritmo $O(N^4)$ es notablemente más pronunciada, lo que lo hace impráctico para matrices de tamaño moderado a grande, mientras que el algoritmo $O(N^3)$ escala de manera mucho más eficiente.

Visualización de Datos

Para una representación visual clara de esta diferencia de rendimiento, podemos ver el siguiente gráfico:

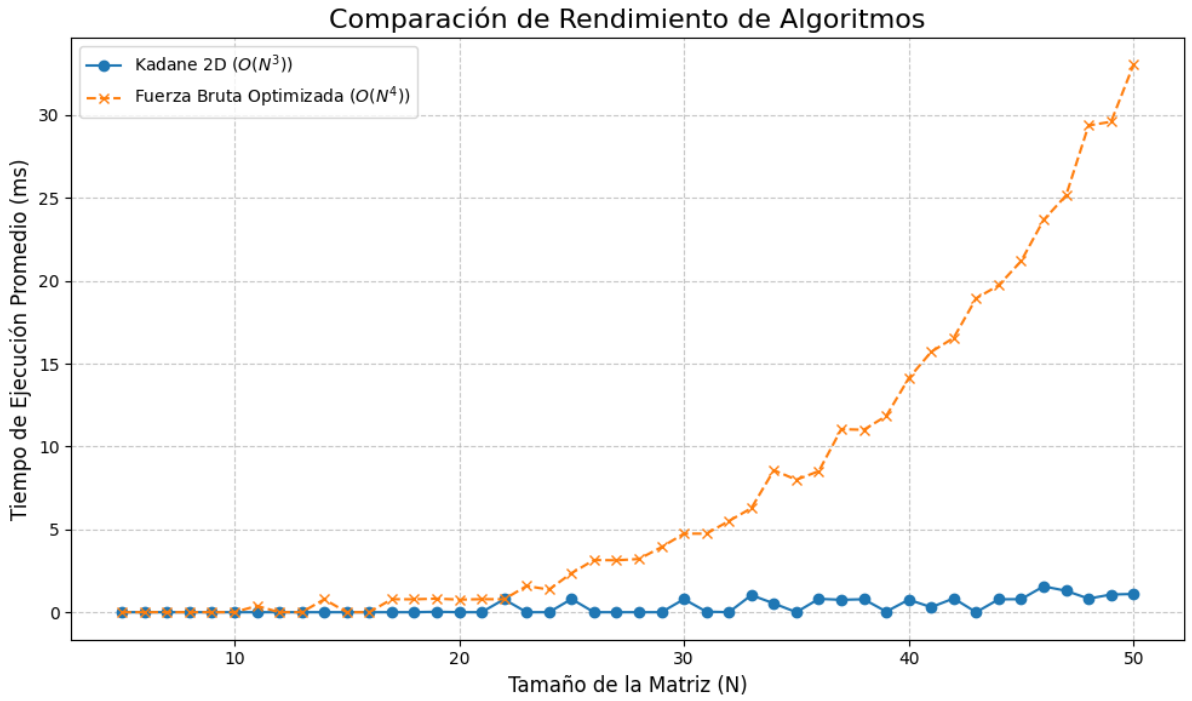

Figura 1: Gráfico de Rendimiento: Tiempo de Ejecución Promedio vs. Tamaño de la Matriz (N)

Conclusiones

Se ha explorado y comparado dos algoritmos fundamentales para la resolución del problema de la suma máxima de submatriz en dos dimensiones: el Algoritmo de Kadane 2D, con una complejidad temporal de $O(N^3)$, y el Enfoque de Fuerza Bruta Optimizado, con una complejidad de $O(N^4)$.

A través del análisis teórico y las pruebas de rendimiento empíricas, se ha demostrado de manera concluyente la superioridad del algoritmo de Kadane 2D. Mientras que para tamaños de matriz pequeños (valores de N reducidos) ambos algoritmos exhibieron tiempos de ejecución casi instantáneos (a menudo por debajo de la resolución de la medición), la diferencia en eficiencia se volvió drásticamente evidente a medida que el tamaño de la matriz N aumentaba.

Los resultados empíricos validan las complejidades teóricas: el algoritmo de Fuerza Bruta Optimizado mostró un crecimiento del tiempo de ejecución mucho más pronunciado, incrementándose sustancialmente con cada aumento en N. En contraste, el algoritmo de Kadane 2D mantuvo tiempos de ejecución significativamente menores y un crecimiento mucho más moderado, reflejando su menor complejidad polinómica. Por ejemplo, para una matriz de $N = 50$, Kadane 2D fue aproximadamente 30 veces más rápido que la Fuerza Bruta Optimizada.

Esta investigación subraya la crítica importancia del diseño algorítmico eficiente. Una mejora en la com-

plejidad asintótica, incluso de un factor polinomial como N^4 a N^3, se traduce en ganancias de rendimiento exponenciales a medida que el tamaño del problema crece. Para aplicaciones que requieren procesar matrices de grandes dimensiones, la elección de un algoritmo $O(N^3)$ como Kadane 2D es indispensable para garantizar la viabilidad y la eficiencia computacional.