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

Asignación #1: Backtracking

Problema de las N-Reinas

Estudiante:
Erimar Reis CI 29.743.464

Caracas, 9 de Mayo de 2025

Introducción

El problema de las N reinas es un desafío clásico que consiste en ubicar una cantidad N de reinas en un tablero de ajedrez de dimensiones N x N, de manera que ninguna de ellas pueda atacar a la otra. Dado el comportamiento de esta pieza entendemos que ninguna debe compartir la misma fila, columna ni diagonal. Este problema tiene una naturaleza combinacional, por ende su complejidad aumenta considerablemente según el valor de N elegido, esto lo convierte en un reto interesante para ser optimizado y estudiado.

Para este estudio nos basamos en la técnica de backtracking y comparamos tres enfoques algorítmicos para resolver el problema:

1. **Algoritmo clásico mejorado**: toma el método de búsqueda tradicional por combinaciones, pero restringiendo la selección libre únicamente a columnas.
2. **Algoritmo Optimizado**: mejora la estructura de datos y la estrategia de búsqueda mediante el uso de permutaciones.
3. **Algoritmo probabilístico**: basándose en el algoritmo clásico mejorado, selecciona aleatoriamente un subconjunto de columnas válidas para cada fila, reduciendo la cantidad de posibilidades.

En este informe, analizamos los tiempos de ejecución de cada método, prestando especial atención a cómo la aproximación probabilística afecta o mejora la capacidad de encontrar soluciones. Para evaluar el rendimiento, se midieron los diferentes algoritmos en microsegundos, considerando $N \in \{8, 16, 32, 64\}$ y ajustando las iteraciones según la complejidad del problema.

Implementaciones

El **Backtracking** es el enfoque tradicional para resolver el problema de las N reinas, esta técnica explora todas las posibles combinaciones del tablero hasta encontrar una solución válida. La manera más sencilla de implementarlo consiste en añadir una reina por cada casilla vacía en el tablero, comprobando que cumpla las condiciones de validez; si no las satisface se coloca en la siguiente casilla vacía. Este enfoque es equivalente a probar todas las permutaciones posibles, pero es posible que se eliminen potenciales soluciones al verificar las condiciones de validez. A partir de este problema surge la primera optimización:

Algoritmo Clásico Mejorado

En este método se coloca una única reina por cada fila y se iteran todas las columnas, esto permite reducir las condiciones de validez ya que se garantiza que no existan dos reinas por fila. Es decir, solo es necesario verificar que no coincidan en la misma columna o diagonal.
Con base en esto podemos decir que su complejidad es $O(N!)$, pues en la primera fila hay N opciones y en la segunda fila hasta $N - 1$. También se debe considerar que, al validar la posición de cada reina, es necesario marcar columnas y diagonales. Estas operaciones de marcado y desmarcado son $O(N)$ cada una, pero se realizan por cada elección de columna. Aunque este enfoque puede ser más eficiente porque descarta opciones inválidas temprano, la complejidad sigue siendo $O(N!)$ con constantes altas debido a las operaciones de marcado y desmarcado.

Algoritmo Optimizado

Para mejorar la eficiencia del algoritmo clásico introducimos una representación basada en arreglos, donde cada posición del arreglo indica la columna en la que está ubicada la reina en la fila correspondiente. Con esta implementación se eliminan las verificaciones explícitas de filas, dado que cada fila contiene una única reina, además se optimiza la comprobación de diagonales utilizando operaciones matemáticas simples en lugar de otras estructuras de datos.
Al analizar teóricamente el algoritmo, observamos que mantiene una complejidad $O(N!)$, esto debido a que se generan permutaciones de columnas, es decir $N!$ posibilidades. Para cada permutación se verifica conflictos en diagonales, sin embargo, basta con verificar si la distancia entre filas y columnas de dos reinas es igual para saber que ambas colisionan. Esta verificación es una simple operación matemática $O(1)$ que se realiza para todas las reinas, es decir $O(N)$. Así, sabemos que para hallar una solución, en el peor caso se deben verificar todas las permutaciones del arreglo, lo que resulta en una complejidad $O(N!)$.

Algoritmo Probabilístico

Como alternativa a los métodos determinísticos se propone un enfoque probabilístico en el que se selecciona aleatoriamente un subconjunto del 30 % de columnas por fila. Esta simple acción reduce significativamente el espacio de búsqueda ya que estamos considerando únicamente un número limitado de posiciones en cada paso. Adicional a esto, incluímos una aleatorización del orden de exploración mezclando el arreglo que contiene las posibles columnas a visitar, esto ayuda a diversificar la búsqueda y evitar patrones repetitivos.
Aunque la selección aleatoria puede llevar a encontrar una solución más rápido, al igual que en el primer algoritmo esta solución tiene complejidad $O(N!)$. Al limitar el número de columnas probadas se reduce el número de opciones por fila; sin embargo, en el peor caso, si se deben probar varias columnas, la complejidad sigue siendo $O(N!)$ aunque con constantes menores.
Es importante resaltar que este método no garantiza encontrar una solución válida. Su verdadera ventaja radica en la reducción del tiempo de ejecución, ya que al explorar menos combinaciones puede proporcionar una respuesta más rápida para valores grandes de N.

Análisis de resultados

Para evaluar el rendimiento de los algoritmos implementados se realizaron pruebas controladas en las que se midieron los tiempos de ejecución en microsegundos. Para facilitar la comprensión de los resultados, se convirtión la unidad de medida de microsegundos a segundos. A continuación se resume los resultados obtenidos:

N	Iteraciones	Clásico	Optimizado	Probabilístico
8	1000	45,799	12,189	246,666667
16	1000	2101,36	1956,031	48233,466667
32	10	29823027,5	22675307,1	16041330,1999999
64	1	n/a	n/a	68114068,0
64	2	n/a	n/a	1613313040,5

Cuadro 1: Mediciones en Microsegundos

N	Iteraciones	Clásico	Optimizado	Probabilístico
8	1000	0,000045799	0,000012189	0,000246666667
16	1000	0,00210136	0,001956031	0,04823346667
32	10	29,8230275	22,6753071	16,0413302
64	1	n/a	n/a	68,114068
64	2	n/a	n/a	1613,313041

Cuadro 2: Mediciones en Segundos

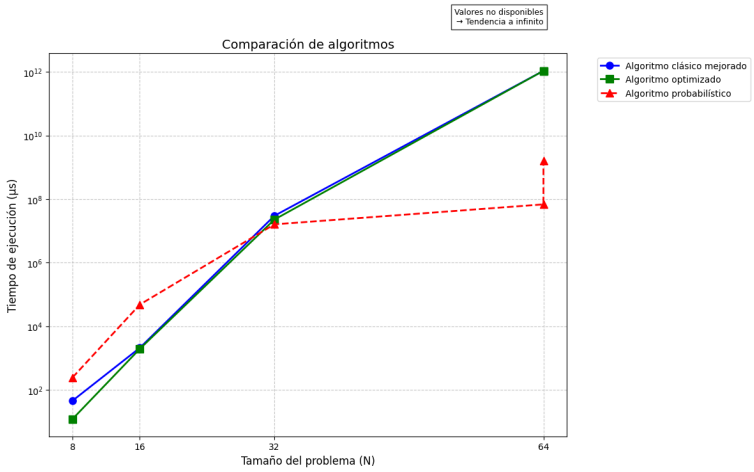

Figura 1: Comparación de los algoritmos

Tendencia Observada

- Para $N = 8$ y $N = 16$, el método probabilístico mucho más lento que las optimizaciones clásicas. Esto se debe a que en un tablero 8x8 o 16x16, la cantidad de columnas disponibles ya es bastante reducida, por ende al inhabilitar casillas se reduce la capacidad de encontrar la primera solución eficientemente.
- Para $N = 32$ tanto el algoritmo clásico mejorado como el optimizado comienzan a volverse inviables, alcanzando un promedio de 25 segundos para obtener una solución.
- Las mediciones de $N = 64$ para el algoritmo clásico y optimizado no se pudieron completar satisfactoriamente; se dejó en prueba aproximadamente 3 horas sin recibir respuesta, esto demuestra su alta complejidad temporal. Mientras tanto, el algoritmo Probabilístico sí logró encontrar una solución.
  En este método el tiempo de ejecución varía considerablemente: para algunas iteraciones se obtuvo un tiempo aproximado de 68 segundos $\approx$ 1,13 minutos mientras que para otras superó en promedio los $1613,31$ segundos $\approx$ 26,89 minutos. Su eficiencia depende completamente de la selección aleatoria de columnas y de la manera en la que se mezcla este vector, sin embargo, en general, siempre logrará proporcionar una solución en menor tiempo que los algoritmos mencionados anteriormente.
- Mientras que los algoritmos clásicos presentan un crecimiento exponencial, el método probabilístico escala de manera lineal o polinomial esto permite resolver instancias grandes del problema de forma eficiente.

Complejidad Práctica vs. Teórica

Los resultados obtenidos muestran diferencias significativas entre los enfoques. A continuación se analiza brevemente cada caso:

1. **Algoritmo clásico mejorado**: A pesar de que su complejidad teórica es $O(N!)$, en la práctica se observa un crecimiento exponencial $(O(2^n))$
2. **Algoritmo Optimizado**: La eliminación de verificaciones redundantes mejora ligeramente la eficiencia, reduciendo las operaciones sin alterar la complejidad global.
3. **Algoritmo probabilístico**: Aunque su complejidad teórica es $O(N!)$, en la práctica observamos que se comporta similar a una función polinómica, lo que permitiría compararlo con una complejidad $O(N^2)$

Probabilidad de Éxito del Método probabilistico

La efectividad del método depende de la capacidad que tenga para encontrar una solución válida dentro de un número determinado de iteraciones. Con los datos obtenidos y presentados anteriores junto con la cantidad de soluciones registradas en cada caso, es posible evaluar la probabilidad de éxito para los diferentes valores de N:

- Para $N = 8$ se encontraron **6 soluciones** en **1000 iteraciones**, por lo que la probabilidad de éxito es $P = \frac{6}{1000} = 0{,}6\,\%$
- Para $N = 16$ se encontraron **60 soluciones** en **1000 iteraciones**, lo que indica una la probabilidad de éxito de $P = \frac{60}{1000} = 6\,\%$
- Para $N = 32$ se encontraron **10 soluciones** en **10 iteraciones**, así la probabilidad de éxito es $P = \frac{10}{10} = 100\,\%$
- Para $N = 64$ se encontró **1 solución** en **1 iteración**, lo que implica una probabilidad de $100\,\%$ en este caso, aunque se requiere una validación estadística más extensa y detallada para respaldar esta afirmación.

Estos resultados muestran que, a medida que N crece, la probabilidad de éxito del método aumenta drásticamente, ya que explora menos combinaciones dentro de un amplio espacio de posibilidades, reduciendo los errores. La reducción del tiempo de ejecución compensa considerablemente el riesgo de múltiples intentos de este método, haciendo de este método una alternativa viable para problemas de gran escala.

Conclusiones

Es claro que el enfoque clásico para resolver este problema presenta limitaciones importantes en términos de escalabilidad. A medida que N crece, el tiempo de ejecución se vuelve exponencial, haciendo que el método sea inviable para $N = 64$. Además, el consumo de memoria aumenta considerablemente al tener que almacenar múltiples estados del tablero completo durante la recursión.

Por otro lado, la optimización probabilística reduce el espacio de búsqueda, aunque no garantiza que se encuentre una solución. Para valores pequeños este algoritmo puede requerir múltiples ejecuciones antes de hallar una solución válida. También se debe considerar que el proceso de selección aleatoria puede introducir un sesgo en la selección de columnas, aumentando el total de iteraciones. A pesar de sus limitaciones, el enfoque probabilístico demuestra ser una alternativa viable para resolver instancias grandes del problema, sacrificando la cantidad de soluciones encontradas a favor de eficiencia.

Así concluye este informe, resaltando la importancia de explorar enfoques híbridos que equilibren precisión y rendimiento para la resolución de problemas combinatorios como este. La búsqueda de estrategias más eficientes no solo optimiza resultados, también permite abordar instancias de mayor complejidad de manera viable y efectiva.

Referencias

- N Queen Problem - Geeksforgeeks
- N Queen Problem - Medium
- Chrono in C++ - GeeksforGeeks
- DeepSeek - AI Chat