Ejercicio 2: Backtracking - Posicionamiento de las m piezas en un tablero de nxn (revisado mayo 2018)

Muñoz Plaza. Alejandro

Abstract—En este ejercicio se pone en uso la técnica de backtracking en el caso práctico de colocar varias piezas de ajedrez en un tablero de dimensión n. Al iniciar el programa se nos pide la dimensión del tablero y el número de piezas a colocar, a continuación, pulsamos iniciar y si encuentra una solución nos la muestra en una nueva ventana y el tiempo que ha tardado en encontrarla o nos informa de que esa combinación que se ha introducido no tiene solución posible.

1 Introduction

Este problema deriva del problema de las 8 damas planteado por el alemán "Max Bezzel" en la revista "Berliner Schachzeitung" en 1848. Consiste en colocar 8 damas en un tablero de ajedrez de tamaño 8x8. Al analizar este programa se descubrió que existían 92 soluciones posibles a este problema que fueron encontradas por primera vez por "Franz Nauck" en 1850. La diferencia con el problema que se nos plantea es que además de damas podemos colocar otras piezas, en este caso alfiles, torres y dos piezas inventadas, y la dimensión del tablero puede ser definida.

Para resolver este problema usaremos la técnica de programación backtracking reduciendo el tiempo máximo de ejecución del problema de $O(n^2)$ a $O(n \log n)$.

2 Guía de Usuario

2.1 Campos de datos

IMPORTANTE: Para pasar de un proyecto de Intellij a netbeans se tendría que crear un nuevo proyecto netbeans y sustituir la carpeta "src" por la "src" del proyecto de Intellij.

Descripción de la Fig.1:

El primer campo indica el tamaño del tablero, el tamaño del tablero tiene un mínimo de cuatro y no tiene límite superior.

Los siguientes indica la cantidad de piezas de ese tipo que se van a intentar colocar en el tablero.

Al pulsar el botón aparecerá una ventana con la primera solución encontrada y encima del botón el tiempo que ha tardado en encontrar la solución. Si no se ha encontrado aparecerá la ventana informando de ello.

Fig. 1. Ventana principal del programa.

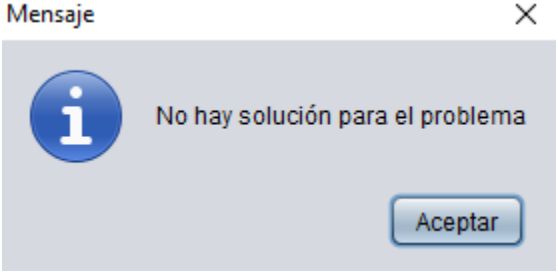

Fig. 2. Ventana que aparece al no encontrar una solución.

Fig. 3. Ventana que aparece cuando se ha encontrado una solución.

3 Piezas nuevas

Se han implementado dos piezas inventadas que tienen cada una su propia manera de comer otras piezas:

1. Lancero: Come cualquier pieza en la horizontal y vertical a dos casillas de distancia.

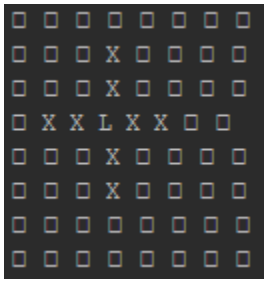

Fig.3. Zona de marcado del Lancero.

2. Halcón:

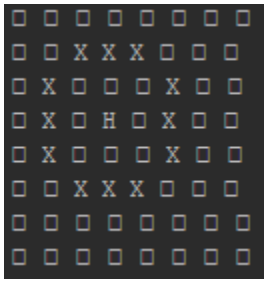

Fig.4. Zona de marcado del Halcón.

4 Coste computacional

Para obtener el coste computacional de la función que obtiene la solución hay que analizar el coste de cada una de las líneas del código de la solución.

Después de este análisis observamos que la operación que mas coste tiene es la que obtiene las posiciones libres del tablero, ya que, revisa todas las posiciones del tablero dando un coste de $O(n^2)$. Aunque con este calculo nos ahorramos recorrer ramas del árbol donde no hay solución, el caso de colocar una pieza en una posición ya amenazada por otra.

Por ello, podemos decir que aplicando este sobre coste para casos relativamente pequeños es mucho mas rápido que la solución de fuerza bruta.

5 Conclusiones

Para este problema el uso backtracking permite reducir el tiempo considerablemente en la mayoría de casos.

El uso de backtracking para reducir el coste de ejecución no es una reducción directa del coste, ya que, depende de la disposición inicial en la que se encuentre el problema y de las decisiones que tome el programador para recorrer el árbol de posibilidades. Por lo tanto, podemos decir que el backtracking es una implementación de un problema y dependiendo del análisis del problema funcionara mejor o peor.

References

[1] https://elpais.com/elpais/2016/03/10/ciencia/1457629302_411493.html