Raimundo Herrera Sufán - 14632152

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos — 1' 2017

Tarea 1

Análisis de Problema

Necesidad de backtracking

En general la necesidad de backtracking viene dada por un problema de múltiples caminos, donde cada decisión, genera árboles nuevos. En este caso, el hecho de elegir una pieza sobre otra, o bien una rotación sobre otra, va presentando los múltiples caminos y muestra la razón. Si se analiza matemáticamente, es un problema que por cada nodo se expande combinatorialmente, por lo que es necesario probar todas (o al menos todas aquellas razonables) las posibilidades de caminos, hasta encontrar uno válido, en este caso, un puzle resuelto.

Poda

La poda propuesta corresponde a no rotar las fichas cuando estas presenten simetría, si todos los lados son iguales, entonces rotar la pieza no tiene sentido en absoluto, por lo que 3 árboles que podrían expandirse ampliamente son eliminados de inmediato. Luego si hay simetría en algun eje (vertical u horizontal) sucede lo mismo, aunque con 2 árboles eliminados.

El costo es reducido, pues se implementa con una función muy pequeña que lo que hace es a priori ver la simetría de la pieza, y luego al ejecutar el backtracking, ver si hay o no que rotar, por lo que es constante de tamaño igual al número de comparaciones realizadas, que es alrededor de 5. No creo que una estructura ayudase a mejorar esta poda, porque lo que hace es eliminar caminos, y acorta los for, sin importar la estructura.

Heurística

La heurística propuesta corresponde en probar primero con aquellos espacios vacios del puzle que presenten menos libertad, es decir esten más determinados, viendo esto a través de la cantidad de bordes disponibles para calzarles piezas. Si la libertad es menor, entonces hay menos piezas que calzaran y se va achicando el árbol.

En tiempo es bastante pequeño porque consiste en recorrer la grilla para encontrarlos, y es tiempo constante asociado al tamaño de la grilla, sin embargo podría ser posible optimizarlo agregando un heap que tuviera en la cabeza a aquel elemento con más bordes cerrados, y así al sacarlo porque se utilizó se utilizacen operaciones de heap, que son muy eficientes, en vez de recorrer cada vez la grilla.