Informe Tarea 1: Algoritmo A* y Puzzle de 15

Víctor Duarte Arce
Departamento de Ingeniería Informática
Universidad de Santiago de Chile, Santiago, Chile
victor.duarte.a@usach.cl

I. Introducción

I-A. Objetivos

1. Implementación de A*: Desarrollar una implementación eficiente del algoritmo A* para resolver el problema del 15-Puzzle.
2. Evaluación del rendimiento: Evaluar el desempeño del algoritmo A* en términos de tiempo de ejecución y capacidad para encontrar soluciones óptimas.
3. Exploración de mejoras: Investigar posibles mejoras o extensiones al algoritmo A* para aumentar su eficiencia en la resolución del 15-Puzzle.
4. Presentación de resultados: Presentar los resultados obtenidos y ofrecer conclusiones sobre la efectividad y limitaciones del enfoque propuesto.

I-B. Algoritmo A* [1]

El algoritmo A* (Ver algoritmo 2) se basa en los *estados*, de los cuales hay dos conjuntos, *abiertos* y *todos* en su búsqueda. Inicialmente, el conjunto de estados abiertos contiene únicamente el estado inicial del rompecabezas, mientras que el conjunto de estados cerrados está vacío. El algoritmo A* utiliza una función heurística para estimar el costo o la distancia desde cada estado hasta el estado objetivo. Esta función guía al algoritmo para explorar primero los estados más prometedores, reduciendo así la cantidad de estados que necesita examinar.

I-C. Problema a resolver: 15-Puzzle

El problema del 15-Puzzle, se trata de un problema del tipo NP completo, el cual consiste en un rompecabezas deslizante que consta de 15 fichas numeradas del 1 al 15 colocadas en una cuadrícula 4x4 con un espacio vacío. El objetivo es reorganizar las fichas desde una disposición inicial desordenada (Ver figura 1) a una disposición ordenada (Ver figura 2), moviendo una ficha a la vez hacia el espacio vacío hasta lograr ordenar todas las fichas de forma ascendente, de izquierda a derecha y de arriba hacia abajo.

II. Antecedentes

II-A. Imágenes

Figura 1: (Una configuración del Puzzle 15 [2])

Figura 2: (Puzzle 15 Ordenado [2])

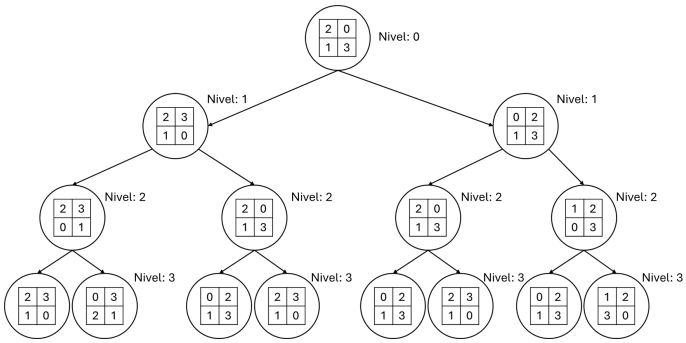

Figura 3: (Caso simple búsqueda de solución)

Figura 4: (Caso de 2×2)

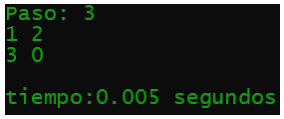

Figura 5: (Caso de 2×2)

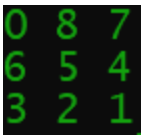

Figura 6: (Caso de 3×3)

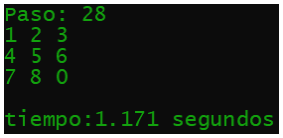

Figura 7: (Caso resuelto de 3×3)

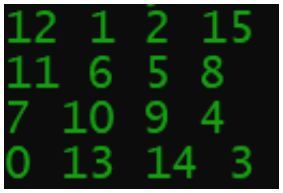

Figura 8: (Caso de 4×4)

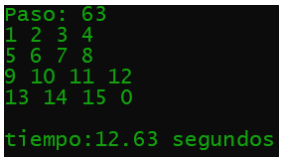

Figura 9: (Caso resuelto de 4×4)

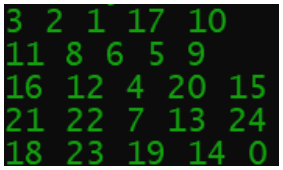

Figura 10: (Caso de 5×5)

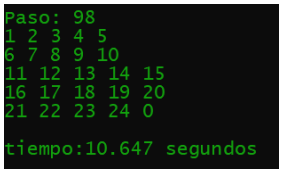

Figura 11: (Resultado de un caso de 5×5)

II-B. Fórmulas

II-B1. Transformación matriz entero: Para la transformación de matriz a entero se usaron las siguientes fórmulas

1. Sea una transformación $T : M(\mathbb{N}_0)_{n\times n} \to \mathbb{N}_0$

$$T(M) = \sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij} \cdot n^{2k} \tag{1}$$

donde $k \in \{l \in \mathbb{N}_0 |\ l \leq n\}$ es un número que se incrementa en cada ciclo.

II-B2. Distancia de Manhattan: La distancia de Manhattan se define como [3]:

$$d_1(p,q) = \sum_{i=1}^{n} |p_i - q_i| \tag{2}$$

para $p = (p_1, p_2, ..., p_n)$ y $q = (q_1, q_2, ..., q_n)$

II-B3. Asignación de valores de la heurística:

$$H = \begin{cases} \frac{0{,}0182 \cdot depth + 0{,}1 \cdot distance + 0{,}12 \cdot misses}{size} & si \quad n > 4 \\ \frac{0{,}0182 \cdot depth + distance + 0{,}12 \cdot misses}{size} & si \quad n = 4 \\ depth & si \quad n < 4 \end{cases} \tag{3}$$

donde $n \in \mathbb{N}$ es el tamaño del *puzzle*.

II-C. Algoritmos

Algoritmo 1 Transformación de naturales a matriz

```
aux ← id
aux_1 ← id_1
board ← 0_{n×n}                                   ▷ Matiz nula
base ← size^2
k ← 0
for i ← size down to 1 do
    for j ← size down to 1 do
        if base es impar then
            if k > ⌊base/2⌋ then
                board_ij ← id  mód base
                id ← ⌊id / base⌋
            else
                board_ij ← id_1  mód base
                id_1 ← ⌊id_1 / base⌋
            end if
            k ← k + 1
        else
            if k ≥ ⌊base/2⌋ then
                board_ij ← id  mód base
                id ← ⌊id / base⌋
            else
                board_ij ← id_1  mód base
                id_1 ← ⌊id_1 / base⌋
            end if
            k ← k + 1
        end if
    end for
end for
id ← aux
id_1 ← aux_1
return board
```

Algoritmo 2 Algoritmo A*

```
A ← ∅
T ← ∅
e ← estado inicial
A ← A ∪ {e}
T ← T ∪ {e}
while A ≠ ∅ do
    e ← A.pop()
    if e es solución then
        imprimir "Encontramos la solución"
        imprimir e
        retornar
    end if
    expandir_estado(e)
end while
imprimir "No se encontró solución"
```

III. Métodos

III-A. Representación del tablero

Para el rápido acceso a los estados a través de los árboles y las tablas *Hash* se decidió implementar una representación a través de dos enteros los cuales serían una identificación del tablero. Nótese que es importante que identifiquen el tablero y no el estado, por eso se eligió la transformación 1, la cual obtiene solo información de la matriz que representa el tablero.

Sobre la ecuación 1, esta se basó en representar dígito a dígito la el tablero, pero para que el entero no se desbordara tanto, se decidió usar una representación dígito a dígito pero con una base variable, así nos aseguramos que cada par de identificadores sea único.

Además para estos dos identificadores se usó la estructura *long long unsigned int*, ya que una estructura más pequeña se desbordaría en los casos de 6×6.

III-B. Transformación inversa de los enteros al tablero

Se usó el algoritmo 1 para pasar de los dos identificaciones al tablero original.

III-C. Clases

Para la implementación del algoritmo A* se implementaron las siguientes clases:

III-C1. Clase $State$: La clase *State* contiene una representación de una tablero, un puntero al estado padre (de donde viene (ver figura 3)), las coordenadas del lugar donde se encuentra el espacio vacío y el valor de su heurística.

III-C2. Clase AVL: Para el rápido acceso a los tableros, se implementó una estructura del tipo *Árbol binario ordenado balanceado*, ya que su tiempo promedio de acceso a un elemento es de $O(log(n))$.

III-C3. Clase $Hash$: De nuevo, para el rápido acceso a los tableros, se hizo una *Tabla Hash* con *AVLs* para el manejo de colisiones, ya que así el tiempo de acceso a cada tablero será de $O(log(\frac{n}{m}))$ donde n es la cantidad de tableros guardados y m es el largo de la *Tabla Hash*, la cual, en este caso se usó una hash de tamaño 1000.

III-D. Clase $Heap$

Para el conjunto de los estados abiertos, se implementó una estructura del estilo *Heap* para poder ordenarlos por algún criterio, en este caso una heurística.

III-E. Heurística

Para la heurística se utilizaron tres criterios

1. **Profundidad**: Se utilizó la profundidad para determinar la prioridad de los casos abiertos a revisar, este criterio es muy importante, ya que cada nivel de profundidad aumenta los casos desde $O(2^n)$ hasta $O(4^n)$ dependiendo del caso (Ver figura 3 de un caso simple de 2×2).
2. **Fallos**: Los *Fallos* son la cantidad de casillas que no están en el lugar "correcto", es decir, el lugar en el que estaría si el *puzzle* estuviera resuelto.
3. **Distancia de Manhattan**: Para este criterio se usó la suma entre todas las distancias de Manhattan (Fórmula 2) de cada casilla y el lugar en el que debería estar si el *puzzle* estuviera resuelto.

Por último se le asigna un atributo a la clase $State$ el cual contiene todos los valores de la heurística (Ver ecuación 3).

IV. Resultados

IV-A. Caso 2×2

Para el caso de la figura 4 el tiempo de ejecución y la cantidad de pasos se muestra en la figura 5 y además se encontró la solución con menos pasos posibles por la heurística implementada en 3.

IV-B. Caso 3×3

Para el caso de la figura 6 el tiempo de ejecución y la cantidad de pasos se muestra en la figura 7 y además se encontró la solución con menos pasos posibles por la heurística implementada en 3.

IV-C. Caso 4×4

Para el caso de la figura 8 el tiempo de ejecución y la cantidad de pasos se muestra en la figura 9.

IV-D. Caso 5×5

Para el caso de la figura 10 el tiempo de ejecución y la cantidad de pasos se muestra en la figura 11.

Para este tamaño ya se empieza a notar el uso de la memoria *RAM*, el cual llega a ser de 440 Mega Bytes aproximadamente.

IV-E. Caso 6×6 o más

Para casos de mayor tamaño no se ha realizado una prueba, porque se necesita de más memoria principal o *RAM*, la cual aumenta en un orden $O(4^n)$ por cada paso que se necesita para resolver el *puzzle*.

V. Conclusiones

El trabajo realizado fue en su mayor parte exitoso, ya que se logró implementar una solución relativamente eficiente a un problema del tipo NP completo a través de heurísticas y estructuras de datos de rápido acceso, como tablas *Hash* y *Árboles binarios ordenados balanceados*.

Algunos aspectos a mejorar pueden ser los siguientes:

1. **Heurística**: La implementación de la heurística, a pesar de existir es muy vaga, ya que se usaron dos criterios demasiado simples.
2. **Heap**: La implementación del *Heap* es muy vaga, ya que para mejorar la eficiencia se centró en mejorar la estructura que utiliza el conjunto *Todos*.

Referencias

[1] Wikipedia contributors. (2024) A* search algorithm — Wikipedia, the free encyclopedia. [Online; accessed 22-April-2024]. [Online]. Available: https://en.wikipedia.org/wiki/A*_search_algorithm

[2] ——. (2024) 15 puzzle — Wikipedia, the free encyclopedia. [Online; accessed 22-April-2024]. [Online]. Available: https://en.wikipedia.org/wiki/15_Puzzle

[3] ——, "Geometría del taxista — Wikipedia, la enciclopedia libre," 2024, [Online; accessed 22-April-2024]. [Online]. Available: https://es.wikipedia.org/wiki/Geometra_del_taxista