Resolución de juego Puzzle 8 con algoritmo de retroceso.

Carlos Alvarez Silva
e-mail: carlos.alvarez.s@usach.cl

Solving Puzzle 8 game with backward algorithm.

RESUMEN: *Se hace uso del algoritmo de retroceso para resolver un puzzle de la forma más rápida posible. Se basa en una función recursiva implementada en lenguaje C, que construye todas las combinaciones de movimientos válidas en el puzzle, y al terminar selecciona aquella combinación con el mínimo número de movimientos necesarios para resolverlo.*

PALABRAS CLAVE: Algoritmo de retroceso, Juego de rompecabezas, Puzzle-8.

ABSTRACT. *The backward algorithm is used to solve a puzzle in the quickest way possible. The solution is based on a recursive function implemented in the programming language in C, which constructs every possible combination and delivers the minimum number of movements necessary to solve it.*

Keywords. *Backward Algorithm, Puzzle Game, Puzzle-8.*

1 INTRODUCCIÓN

En la clase de algoritmos avanzados se muestran los algoritmos más conocidos para problemas comunes de ordenamiento, búsqueda y clasificación.

Entre los algoritmos de búsqueda impartidos se encuentra el algoritmo con retroceso, un algoritmo basado en el algoritmo de fuerza bruta que resulta en una idea sencilla y de fácil implementación.

Sobre dicho algoritmo se estudiará su idea y se indicará su implementación para la resolución en un problema de búsqueda, para luego evaluar su desempeño, eficacia y eficiencia durante la ejecución.

2 DESCRIPCIÓN DEL PROBLEMA

Para la experiencia se busca la resolución del juego puzzle 8, un juego de lógica que consiste en armar un rompecabezas compuesto por 8 piezas deslizables en la menor cantidad de movimientos posibles.

Con el propósito de encontrar la menor combinación de movimientos que consigan tal objetivo, se utilizará el algoritmo de retroceso para generar las combinaciones de movimientos posibles que resuelven el puzzle y después seleccionar la menor de ellas. Se considera combinación de estados como todos los movimientos realizados necesarios para modificar el puzzle desde un estado inicial a un nuevo estado cualquiera sin repetir estados nuevos en el proceso.

3 MARCO TEÓRICO

a) **ALGORITMO DE FUERZA BRUTA:** Es un algoritmo de búsqueda que consiste en la formación de todas las combinaciones posibles que pueden o no llegar a la solución de un problema.
Cuando todas las combinaciones sean hechas, se separan las combinaciones exitosas del resto de combinaciones, y luego se elige la mejor combinación en base a lo que se busca.

b) **ALGORITMO CON RETROCESO:**
También conocido como *backtracking*, es un algoritmo de búsqueda de características similares al algoritmo de fuerza bruta.
Consiste en la generación de todos los estados válidos que derivan de un estado inicial dado, si encuentra un estado incorrecto o bien no quedan estados por revisar, la búsqueda retrocede al estado anterior y se toma la siguiente alternativa.
Cuando la búsqueda finaliza se genera un árbol implícito, cuya rama de menor longitud (o mayor según corresponda) es también la solución al problema.
Comparativamente es más rápido con respecto al método de fuerza bruta, ya que requiere de una menor cantidad de operaciones, y se puede convertir una implementación del algoritmo de retroceso a uno que implemente fuerza bruta eliminando la selección de estados inválidos.

4 DESCRIPCIÓN DE LA SOLUCIÓN

La idea de la solución consiste en una función que recursivamente va generando todos los estados posibles de un puzzle en base a los movimientos posibles desde un estado inicial (arriba, abajo, izquierda y derecha).

Si la configuración actual de un puzzle coincide con la del puzzle objetivo, se registra el número de movimientos realizados en una lista de casos exitosos para su futura comparación y obtención del mínimo valor, el cual coincide con el valor resultante.

Si la configuración de un estado del puzzle ya ha sido revisada, no se sigue generando más

combinaciones desde ese estado, en cambio se continúa con un estado sin analizar, el cual se registra en un listado para futuras comparaciones.

El algoritmo termina cuando no quedan más estados de puzzle por revisar.

Para la implementación de la solución se utilizó el lenguaje de programación C, sobre el cual se hace uso de 3 estructuras de datos:

1. **Puzzle:** Estructura de datos que contiene un arreglo de caracteres, el cual representa una matriz de 3x3 elementos, el cual a su vez representa el estado actual de un puzzle en un determinado momento. Es la estructura más importante para la implementación de la solución, puesto que se trabaja sobre las mismas en todo momento.

2. **ListaPuzzle:** Estructura que representa un listado de estados de un determinado puzzle. Se utiliza para el registro y comparación de estados pasados.

3. **Lista:** Estructura que lista números de forma dinámica. Se utiliza para registrar el número de movimientos ejecutados para cada combinación exitosa durante la ejecución de la solución.

El pseudocódigo del programa solución es el siguiente:

```
--INICIO DE ALGORITMO--

estadoInicial = Puzzle
estadoObjetivo = Puzzle
listaEstados = ListaPuzzle vacia
listaExitos = Lista Vacia
nMovimientos = 0

--INICIO DE FUNCION--
Puzzle8Backtracking
    (estadoInicial,
     estadoObjetivo,
     listaEstados,
     listaExitos,
     nMovimientos):

Si      son_iguales(estadoInicial,
estadoObjetivo):
        listaExitos.Append(nMovimientos)
        --FIN DE FUNCION--

Si existe_en(estadoActual,listaEstados)
        --FIN DE FUNCION--
Sino:
        listaEstados.Append(estadoActual)

Para cada dirección posible:
        estadoNuevo=
        mover_pieza(estadoActual,DIRECCION)
        Puzzle8Backtracking
             (estadoNuevo,
             estadoObjetivo,
             listaEstados,
             listaExitos
             nMovimientos + 1)

--FIN DE FUNCION--

Resultado = obtenerMenor(listaExitos)

--FIN DE ALGORITMO--
```

En base al seudocódigo señalado anteriormente se indica que el algoritmo generado tiene un orden de 4^n.

5 ANÁLISIS DE LOS RESULTADOS

Si bien la solución bajo este algoritmo tarda una cantidad de tiempo menor que al ser implementado bajo fuerza bruta, la diferencia no es alta, considerando que aún se deben generar cientos de miles de movimientos para generar todos los movimientos de todas las combinaciones a evaluar, por lo que no es menor la situación en la que se somete el computador en términos de memoria durante la ejecución de la implementación del algoritmo.

De lo anterior se concluye que la solución bajo el algoritmo con retroceso no asegura que este termine con éxito en términos de memoria utilizada, que por tanto la solución no es eficiente, sin embargo y suponiendo que el algoritmo tenga el espacio suficiente en su implementación, esta eventualmente entrega la solución correcta al problema, por lo que se puede decir que el algoritmo es eficaz en retornar la solución.

No se puede mejorar el programa solución bajo el algoritmo utilizado, a menos que se pueda reducir el almacenamiento utilizado por las estructuras utilizadas.

Dado que la problemática implica una cantidad enorme de combinaciones totales y posibles, se requiere de un nuevo algoritmo que no implique generar cada combinación posible para alcanzar la solución en menos tiempo.

6 TRAZA DE LA SOLUCIÓN

A continuación, se muestran los estados iniciales y finales del puzzle en forma de matrices para el análisis del algoritmo. Tómese en cuenta que el espacio vacío se representará con una 'X'.

$Puzzle_{Inicial}$			$Puzzle_{Objetivo}$		
1	2	X	1	2	3
4	5	3	4	5	6
7	8	6	7	8	X

Tabla 1. Estados inicial y objetivo.

Es necesario mencionar que la configuración sobre la cual se hará la traza solo requiere de dos movimientos para lograr el estado objetivo.

N_i	N_{i-1}	$Puzzle_{Actual}$	$Exitos$	$Estados$	N_{Mov}
0	--	1 2 X 4 5 3 7 8 6	{}	{}	0
1	0	1 2 3 4 5 X 7 8 6	{}	{0}	1
2	0	1 X 2 4 5 3 7 8 6	{}	{0,1}	1
3	1	1 2 3 4 5 6 7 8 X	{2}	{0,1,2}	2
4	1	1 2 X 4 5 3 7 8 6	{2}	{0,1,2,3}	2
5	1	1 2 3 4 X 5 7 8 6	{2}	{0,1,2,3}	2
6	2	1 5 2 4 X 3 7 8 6	{2}	{0,1,2,3, 5}	3
7	2	1 2 X 4 5 3 7 8 6	{2}	{0,1,2,3, 5,6}	3
8	2	X 1 2 4 5 3 7 8 6	{2}	{0,1,2,3, 5,6}	3
9	3	1 2 3 4 5 X 7 8 6	{2}	{0,1,2,3, 5,6,8}	3
…	…	……	…….	……	….

Tabla 2. Traza de las primeras 10 llamadas a la función.

Nótese que, si bien se llega al resultado mínimo en la cuarta llamada a la función, el algoritmo sigue generando más combinaciones hasta formar cada combinación posible.

7 CONCLUSIONES

El programa solución ejecuta correctamente, termina su ejecución de forma exitosa entregando la respuesta adecuada al ejercicio, sin embargo, el algoritmo utilizado no asegura eficiencia para el caso específico tratado, por lo que se sugiere cambiar el algoritmo utilizado por alguno que consuma una menor cantidad de recursos del sistema.

REFERENCIAS

Robert Sedgewick. *Algoritmos en C++*. 4° Edición. 1995. Libro disponible en: https://books.google.cl/books?redir_esc=y&id=8OBlquzq83oC&q=

Tutorial Horizon. *Introduction To Backtracking Programming.* Página revisada el día 17 de Mayo del 2018. Página disponible en: https://algorithms.tutorialhorizon.com/introduction-to-backtracking-programming/