Universidad de Santiago de Chile
Departamento de Ingeniería Informática
Curso de Algoritmos Avanzados

Semestre: 1
Año: 2018

Resolución de juego Puzzle 8 con algoritmo de fuerza bruta.

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

Solving Puzzle 8 game with brute force algorithm.

RESUMEN: *Se hace uso del algoritmo de fuerza bruta para resolver un puzzle de la forma más rápida posible. Se basa en una función recursiva que construye cada combinación posible y entrega el mínimo número de movimientos.*

PALABRAS CLAVE: Algoritmo, Juego, Fuerza bruta, Puzzle-8.

ABSTRACT. *The brute force algorithm is used to solve a puzzle in the quickest way possible. The solution is based on a recursive function that constructs every possible combination and delivers the minimum number of movements necessary to solve it.*

Keywords. *Algorithm, game, brute force, 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 de fuerza bruta, un algoritmo de idea sencilla y 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 fuerza bruta para generar las combinaciones posibles que resuelven el puzzle y después seleccionar la menor de ellas.

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.

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--
Puzzle8FuerzaBruta
    (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)

estadoNuevo = mover_pieza(estadoActual,ARRIBA)
Puzzle8FuerzaBruta
    (estadoNuevo,
     estadoObjetivo,
     listaEstados,
     listaExitos
     nMovimientos + 1)

estadoNuevo = mover_pieza(estadoActual,ABAJO)
Puzzle8FuerzaBruta
    (estadoNuevo,
     estadoObjetivo,
     listaEstados,
     listaExitos
     nMovimientos + 1)

estadoNuevo = mover_pieza(estadoActual,IZQ)
Puzzle8FuerzaBruta
    (estadoNuevo,
     estadoObjetivo,
     listaEstados,
     listaExitos
     nMovimientos + 1)

estadoNuevo = mover_pieza(estadoActual,DER)
Puzzle8FuerzaBruta
    (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

El primer asunto a discutir es la peligrosa situación en la que se somete el computador durante la ejecución del programa en términos de memoria: la cantidad de combinaciones que genera el algoritmo es tal que es fácil saturar la memoria del computador durante la ejecución del programa, lo que obliga a crear un número límite de los movimientos por combinación para lograr disminuir el gasto de memoria.

De lo anterior se concluye que la solución bajo el algoritmo de fuerza bruta 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 paradigma utilizado, a menos que se pueda reducir el almacenamiento utilizado de la estructura puzzle.

Dado que la solución bajo el algoritmo de fuerza bruta es poco eficiente, se debiera proceder a utilizar otro algoritmo que reduzca el número de combinaciones

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'.

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

Tabla 1. Estados inicial y objetivo.

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

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