Algoritmos Avanzados Laboratorio I

NICOLÁS ROMERO FLORES

Profesora: Mónica Villanueva

Ayudante: Gerardo Zúñiga

Fecha: May 4, 2018

Compiled May 4, 2018

En el presente informe se muestra el problema y la solución correspondiente al primer laboratorio del curso de Algoritmos Avanzados. La solución se elabora usando la técnica de fuerza bruta. Se realiza un análisis de los resultados obtenidos, viendo la eficacia y eficiencia de la solución.

1. INTRODUCCIÓN

El primer laboratorio de algoritmos avanzados consiste en la aplicación de "fuerza bruta" como técnica de resolución de problemas, y de esta manera desarrollar un algoritmo que resuelva el problema planteado en el enunciado, el cual corresponde a encontrar la menor cantidad de pasos posibles para resolver un puzzle 8. El problema sera abordado con mas detalles en secciones posteriores. Finalmente el algoritmo desarrollado es analizado, es decir, se ve la eficiencia y eficacia del algoritmo propuesto, además de opciones que nos permitan mejorar el algoritmo. En el caso de la eficiencia, se analizaran el tiempo de ejecución y orden de complejidad del algoritmo, de esta manera determinar efectivamente lo eficiente del algoritmo.

A. Objetivo Principal

1. Desarrollar un algoritmo que, mediante fuerza bruta resuelva el problema planteado, el cual consiste en resolver un puzzle 8 en la menor cantidad de pasos posibles. El puzzle sera entregado en un fichero de texto, y así mismo, la respuesta sera entregada por otro fichero de texto mediante el formato especificado en el enunciado.

B. Objetivos Secundarios

1. Obtener la cantidad mínima de pasos necesarios para resolver el puzzle.
2. Responder las preguntas principales de un algoritmo: ¿Se detiene?, Cuando se detiene ¿entrega la solución correcta?, ¿Es eficiente?, ¿Se puede mejorar?, ¿Existen otros métodos?
3. Para el caso de la eficiencia, estimar el orden de complejidad del algoritmo desarrollado usando la cota superior asintótica (notación O grande).

C. Estructura del informe

En el informe se presentan la descripción del problema , luego un marco teórico que define conceptos clave que se utilizaran a lo largo del informe, se procede a describir y analizar la solución obtenida para el problema para finalmente, hacer un análisis de esta respondiendo las preguntas anteriormente planteadas, haciendo énfasis en la eficiencia del algoritmo.

2. DESCRIPCIÓN DEL PROBLEMA

Luchito asiste al cumpleaños de un amigo. Todos los invitados reciben un Puzzle 8, el cual es un juego de piezas móviles en el cual es necesario mover estas hasta reconstruir una imagen predeterminada. A pesar de su corta edad, Luchito es muy competitivo y quiere resolver el puzzle mas rápido que todos sus amigos. Para esto es necesario construir un programa escrito en lenguaje de programación C utilizando la técnica de fuerza bruta que entregue la cantidad mínima de movimientos necesarios para solucionar el puzzle, esto se debe indicar en un archivo de salida.

3. MARCO TEÓRICO

Se describirán algunos de los conceptos usados a lo largo del informe:

1. **Algoritmo:** Secuencia de instrucciones finita que permite encontrar la solución a un determinado problema.
2. **Fuerza bruta:** Consiste en una forma de resolución de problemas en la cual se deben encontrar todos los candidatos posibles a solución, y luego dentro de ese conjunto encontrar el subconjunto que cumpla las condiciones dadas.

3. **Cola:** Es una lista de elementos en la cual solo se le pueden ingresar elementos al final de ella (encolar) y solo se pueden sacar elementos desde el inicio de ella (desencolar). También se les conoce como listas FIFO (First In, First Out).

4. **Eficiencia de un algoritmo:** Corresponde a las propiedades del algoritmo que nos permite identificar cuanto tiempo tarda un algoritmo en completarse y cuantos recursos usara. Por lo tanto una mayor eficiencia significara que el algoritmo usara menos recursos y tardara menos tiempo en resolver un problema.

5. **Tiempo de ejecución:** Expresión algebraica que indica el tiempo en que demora ejecutar un algoritmo, en función de la entrada de este. Las instrucciones mas básicas poseen un tiempo de ejecución constante 1.

6. **Orden de complejidad:** Corresponde a la estimación del tiempo de ejecución usando la cota superior asintótica de esta.

7. **Lenguaje de programación C:** Lenguaje de programación imperativo-procedural lanzado en 1972. Es un lenguaje de nivel medio que destaca por el manejo manual de memoria, poder crear tipos de datos compuestos y estructuras, además de las características propias de su paradigma.

4. DESCRIPCIÓN DE LA SOLUCIÓN

El puzzle entregado en el archivo de entrada fue modelado como un arreglo bidimensional de 3x3 (matriz), que contiene números del 1 al 8 y una "x" en el espacio vacío. De esta forma, la solución del puzzle 8 correspondería a la siguiente matriz.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & x \end{bmatrix} \quad \textbf{(1)}$$

A. Idea

Para encontrar la cantidad mínima de pasos, se toma el primer estado (la matriz inicial) y se hace uso de búsqueda en anchura, es decir que tomando el estado actual se mueven las piezas a los 4 estados posibles (mover el espacio arriba, abajo, izquierda o derecha si es posible), y se van encolando estos estados en una cola junto a la profundidad de este (la cantidad de pasos necesarios para llegar al estado), y así luego se desencola un estado y se repite el proceso. Además, para no repetir los estados (y tener una condición de termino), se va guardando cada estado en una lista de visitados, de esta manera cada vez que se desencola un estado se comprueba primero si este se encuentra en la lista de visitados y es descartado en caso de que si se encuentre.

B. Funciones y procedimientos

Las principales funciones, procedimientos y algoritmos utilizados para resolver el problema serán descritos en esta sección:

- **swap():** Funcion que intercambia de posicion dos elementos y retorna una nueva matriz con los elementos intercambiados.
- **Algoritmo solution():** Corresponde al algoritmo que buscara todos los caminos posibles junto a sus profundidades (cantidad de pasos necesarios) para llegar al estado solución y retornara un arreglo con todas las profundidades encontradas. Este algoritmo sera explicado en Pseudocódigo. **Entrada:** Puzzle original. **Salida:** Arreglo con todas las profundidades de las caminos a la solución encontradas.

```
1. dx = [0,0,-1,1]
2. dy = [-1,1,0,0]
3. cola = []
4. visitados = []
5. profundidades = []
6. cola.encolar([PuzzleOriginal,0])
7. visitados.anadir(PuzzleOriginal)
8. Mientras cola no este vacia:
9.     puzzle,profundidad=cola.desencolar()
10.    posicion = buscarElemento('x',puzzle)
11.    Para i desde 0 hasta 3:
13.        Si (posicion.x + dx[i] < 3 y posicion.x + dx[i] >= 0 y
14.        posicion.y + dy[i] < 3 y posicion.y + dy[i] >= 0):
15.            puzzleAux=swap(puzzle,posicion.x,posicion.y,dx[i],dy[i])
16.            Si puzzleAux no se encuentra en visitados:
17.                visitados.anadir(puzzleAux)
18.                cola.encolar([puzzleAux, profundidad + 1])
19.                Si puzzleAux es igual a puzzleSolucion:
20.                    profundidades.anadir(profundidad + 1)
21. retornar profundidades
```

- **min():** Función que dado un arreglo, entrega el menor de sus elementos. Es usado para que, luego de tener el arreglo con las profundidades, se calcule la menor de estas.

5. ANÁLISIS DE LA SOLUCIÓN Y RESULTADOS

1. **¿El algoritmo se detiene?** Si se detiene, pues cuando ya se hayan visitados todos los elementos no quedaran elementos en la cola.

2. **¿Resuelve el problema?** Si resuelve el problema, ya que luego de obtener todos los posibles caminos, elige el que tenga la menor cantidad de pasos.

3. **¿Es eficiente?** No, ya que al utilizar fuerza bruta se revisan estados que podrían ser descartados de raíz inmediatamente, haciendo que el orden de complejidad crezca.

4. **¿Se puede mejorar?** Si, cualquier otro método que no sea fuerza bruta sera mas eficiente en encontrar la respuesta.

5. **¿Existen otros métodos?** Si, por ejemplo usando la misma búsqueda en anchura, pero detener el algoritmo apenas se encuentre la primera solución, pues ya que es búsqueda en anchura, el primero en ser encontrado sera efectivamente el camino mas corto.

A. Complejidad de la solución

La complejidad del algoritmo dependerá del tamaño de la entrada de este, en este caso el tamaño de la entrada corresponde a los 9 elementos del puzzle.

1. **createMatrix():** Se asigna espacio para una matriz de NxN, dado que solo se usa un ciclo para asignar la memoria el orden de complejidad es:

$$O(n) \quad \textbf{(2)}$$

2. **readFile():** Se lee el archivo de entrada correspondiente a los 9 elementos, y se guardan en una matriz de 3x3. Por lo tantos su orden de complejidad es:

$$O(n) \quad \textbf{(3)}$$

3. **solution():** Dado que se trabaja con todos los estados posibles del puzzle, cada ciclo donde se trabaje con los estados sera de orden O(n!), por lo tanto el ciclo que se encuentra

en la linea 8 sera de O(n!), luego la condicional que se encuentra en la linea 16 comprueba que el estado actual no haya sido visitado anteriormente, es decir, que acotando superiormente también seria de orden O(n!). Añadir y desencolar elementos es de orden O(n). Finalmente dado que la condición de comprobar elementos se encuentra dentro del otro ciclo, el orden de complejidad seria:

$$O((n!)^2) \quad \textbf{(4)}$$

4. **min():** Es un algoritmo de búsqueda lineal, por lo tanto el orden es O(n).

$$O(n) \quad \textbf{(5)}$$

Por lo tanto, acotando superiormente, el orden de complejidad de la solución propuesta es de:

$$O((n!)^2) \quad \textbf{(6)}$$

6. TRAZA DE LA SOLUCIÓN

En la figura 1 se muestra un extracto de la traza de la solución dado un archivo de entrada.

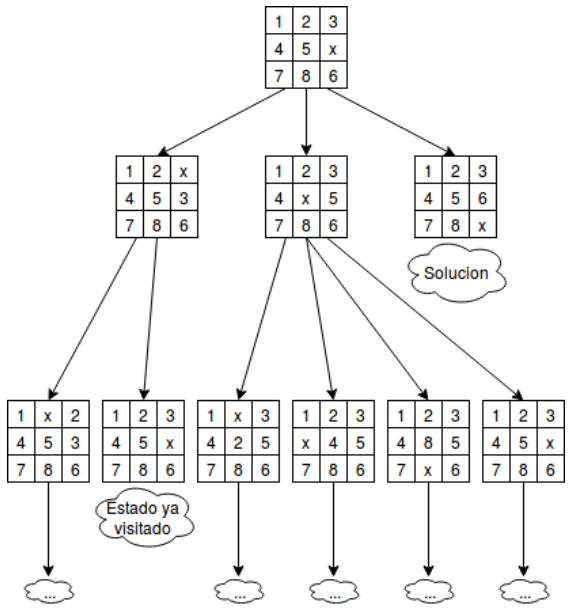

Fig. 1. Traza de la solución.

7. CONCLUSIONES

La solución entregada cumple efectivamente con lo pedido en el enunciado, es decir que este entrega la cantidad de pasos mínimos para poder resolver el puzzle, además de cumplir con el requisito de usar fuerza bruta para encontrar la solución.
El algoritmo propuesto responde correctamente a las preguntas de análisis del algoritmo. En cuanto a la eficiencia de este, el algoritmo tiene un orden de complejidad bastante alto, el cual es el indicado es la indicada en la ecuacion 7, lo que nos indica lo poco eficiente del algoritmo, ya que tarda bastante en entregar el resultado esperado. Esto se debe a que en primera instancia el algoritmo busca todos los candidatos a solucion, es decir todos los caminos posibles que nos lleven a resolver el puzzle, y luego de entre estos se elige el que tome menor cantidad de pasos.

$$O((n!)^2) \quad \textbf{(7)}$$

Una forma de mejorar este algoritmo seria, por ejemplo, detener el algoritmo apenas encuentre la primera solución, ya que con la búsqueda en anchura, la primera solución encontrada sera efectivamente la mas corta.
Con esto se puede ver que la fuerza bruta es eficaz en encontrar la solución, pero no es nada eficiente en hacerlo. Sin embargo, muchas veces resulta mas fácil de implementar la solución usando fuerza bruta, además que puede servir en casos en que específicamente se necesite obtener todos los candidatos a solución posibles.