8-Puzzle

Ivan Dario Solano Velasquez
Moises Bernardo Suarez Gamez
Diego Armando Velasquez Vargas

AGENDA

01 DFS

02 Iterartive DFS

03 A* (Manhattan)

04 Tiempo de ejecución

DFS

El Algoritmo de búsqueda DFS explora los bordes del vértice descubierto más recientemente que todavía tiene bordes inexplorados que nacen de él. Una vez que se han explorado todos los bordes, la búsqueda "retrocede" para explorar los bordes dejando el vértice desde el cual se descubrió. [1]

DFS

```java
public static int DFS(Board initial, Board solution){
        LinkedList<Board> list  = new LinkedList<>();
        HashSet<Board> visited = new HashSet<>();
        list.add(initial);
        int nodes = 0;
        while( !list.isEmpty() ){
            nodes ++;
            Board board = list.removeFirst();
            if(!visited.contains(board))
                for( Board b : nextMovements(board) ){
                    if( b.equals(solution) )
                        return nodes;
                    list.addFirst(b);
                }
            visited.add(board);
        }
        return nodes;
    }
```

Iterative DFS

Es una versión limitada en profundidad del algoritmo BFS primero repetidamente con límites de profundidad crecientes hasta que se encuentra el objetivo, este usa mucha menos memoria; en cada iteración, visita los nodos en el árbol de búsqueda en el mismo orden que la búsqueda de profundidad,

IDDFS

```
public static int DFSIteration(Board initial, Board solution, int maxDepth){
        LinkedList<Board> list  = new LinkedList<>();
        HashSet<Board> visited = new HashSet<>();
        list.add(initial);
        int nodes = 0;
        while( !list.isEmpty() ){
            nodes ++;
            Board board = list.removeFirst();
            if(!visited.contains(board) && board.getDistance() < maxDepth)
                for( Board b : nextMovements(board) ){
                    if( b.equals(solution) ){
                        res = true;
                        return nodes;
                    }
                    list.addFirst(b);
                }
            visited.add(board);
        }
        return nodes;
    }
```

A* (Manhattan)

El algoritmo A* es un algoritmo de búsqueda que puede ser empleado para el cálculo de caminos. Es un algoritmo heurístico, ya que una de sus principales características es que hará uso de una función de evaluación heurística.

En este caso usaremos la distancia Manhattan que en este caso la definiremos como la distancia que está una pieza de estar en su lugar correcto

A*

```java
public static int aEstrella(Board initial, Board solution) {
        PriorityQueue<Board> list = new PriorityQueue<>();
        list.add(initial);
        int nodes = 0;
        while (!list.isEmpty()){
            nodes++;
            for( Board b : nextMovements(list.poll()) ){
                if( b.equals(solution) )
                    return nodes;
                b.setDistanceTotal(b.getDistance()+b.mannhattan());
                list.add(b);
            }
        }
        return nodes;
    }
```

Tiempo

Algoritmo	Tiempo en Milisegundos				
	1ra vez	2da vez	3ra Vez	4ta Vez	5ta Vez
DFS	77453	77186	70937	68914	64285
INTERACTIVE DFS	234425	227249	217057	221312	225612
A* (Manhantan)	201	224	247	1943	222
A* (Piezas fuera de Lugar)	193	247	195	223	280

Tiempo

Algoritmo	Tiempo en Milisegundos
DFS	71755
INTERACTIVE DFS	225131
A* (Manhantan)	567,4
A* (Piezas fuera de Lugar)	227,6

THANKS