8 puzzle

Alex José Alberto Barreto Cajicá
Daniel Arturo Moreno Rincón

Comparativa de las búsquedas

```
def solve():
    init = [[1,2,3],[4,5,6],[7,8,0]]
    puzzle1 = Board(init)
    puzzle1.shuffle()
    #Crea copias del tablero para todas las búsquedas
    puzzle2, puzzle3, puzzle4 = puzzle1.copy()
    a1 = a_star(puzzle1)
    a2 = a_star2(puzzle2)
    d1 = bfs(puzzle3)
    d2 = i_dfs(puzzle4)
    print(a1,a2,d1,d2)
```

El Tablero

```
class Board:
    def __init__(self,board):
        self.board = board
        #posición del hueco en el tablero
        self.i = 2
        self.j = 2

    def action(): movimiento
    def shuffle(): revolver el tablero
    def copy(): hacer una copia del tablero
    def isSolved(): el tablero está resuelto
```

Movimiento

```
# 0 Arriba, 1 Derecha, 2 Abajo, 3 Izquierda
def action(self,move):
     #Arriba
     if(move==0):
         if(i-1>=0):
             board[i][j] = board[i-1][j]
             board[i-1][j] = 0
             i -= 1
         #Si no se puede mover, escoja el siguiente movimiento
         else:
             self.action((move+1)%4)
```

Desordenamiento

```
def shuffle(self):
    prevMove = 0
    for i in range(14):
        move = random.randrange(0,4)
        #checkea que la nueva jugada no deshaga la anterior
        if((move+2)%4 == prevMove):
            move = (move+1)%4
        self.action(move)
        prevMove = move
```

El Nodo

```
class Node:
    def __init__(self, board, move, parent):
        self.board = board
        self.move = move
        self.parent = parent
        if(self.parent=None):
            self.g = 0
        else:
            self.g = self.parent.g + 1

    def f1(): self.g + self.board.manhattan
    def f2(): self.g + self.board.misplaced
    def id(): identificador único del nodo
```

Primera Heurística: Manhattan

```
def manhattan(self):
     distance = 0
     for i in range(3):
         for j in range(3):
             if(self.board[i][j]!=0):
                 x = (self.board[i][j]-1)/3
                 y = (self.board[i][j]-1)%3
                 distance += abs(i-x) + abs(j-y)
     return distance
```

Segunda Heurística: Piezas sin colocar

```
def misplaced(self):
     count = 0
     for i in range(3):
         for j in range(3):
             if(self.board[i][j]!=(i*3)+j+1):
                 count += 1
     return count
```

A*

```
def a_star1(board):
    root = Node(board,-1,None)
    pq = priority_queue([root], key = f1)
    visited = set()
    visited.add(root.id)
    while pq:
      node = pq.pop()
      if node.board.solved:
           return result
      for move in range(4):
           childBoard = node.board.copy()
           childBoard.action(move)
           child = Node(childBoard, move, node)

           if child.id not in visited:
                pq.push(child)
                visited.add(child.id)
```

Búsqueda en profundidad limitada

```
def l_dfs(node,visited,depth):
    if node.id in visited or node.g > depth:
        return False
    if node.board.solved:
        return True

    visited.add(node.id)

    for move in range(4):
        childBoard = node.board.copy()
        childBoard.action(move)
        child = Node(childBoard, move, node)
        l_dfs(child,visited)
```

Búsqueda en profundidad iterativa

```
def i_dfs(board):
    root = Node(board,-1,None)
    visited = set()
    for depth in range(25):
     if l_dfs(root,visited,depth):
         return True
```

Resultados en tiempo

search.py
Tiempo

BFS	IDFS	A*(h2)	A*(h1)
3.2495	5.5858	4.9359	2.6439
7.6090	3.9070	3.1320	1.9900
5.1630	3.3320	9.8320	5.6879
2.0876	4.9179	3.0570	1.9360
4.5370	3.4139	2.4570	1.8480
4.1469	3.2309	2.2339	1.9019
3.8230	3.1139	2.2269	1.7629
3.9990	3.1679	2.2519	1.8079
3.8889	3.1730	2.2669	1.7760
4.0259	3.2110	2.2019	1.7749
4.0009	3.2789	2.2640	1.7370
3.9370	3.2449	2.2699	1.7389
3.9620	3.2599	2.1419	1.7989
3.8669	3.1600	2.1590	1.7319
3.8409	3.2599	2.1779	1.7589
3.9339	3.2660	2.1799	1.7480
3.9300	3.3810	2.2139	1.7479
3.9489	3.1370	2.1029	1.7699
3.9950	3.1660	2.2320	1.7839
3.9979	3.0989	2.1470	1.7280
3.9450	3.1029	2.1489	1.7920
3.9519	3.1300	2.1750	1.7959
3.8770	3.1380	2.1579	1.7599
4.0170	3.1000	2.2309	1.7749
3.9010	3.1390	2.1640	1.7970

Muchas Gracias