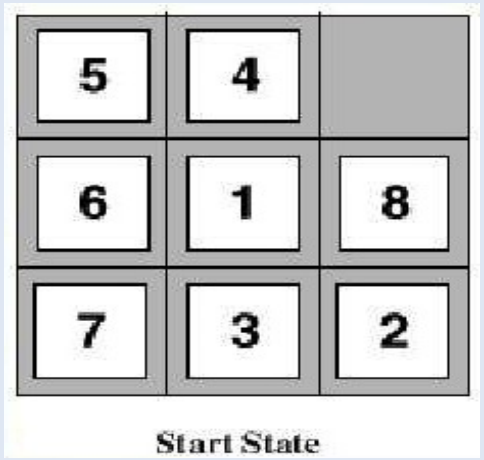
8 Puzzle

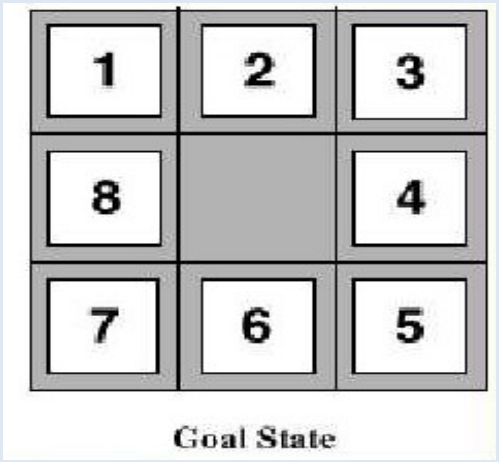
Alberto Acevedo García

Indice

El juego del 8 Puzzle

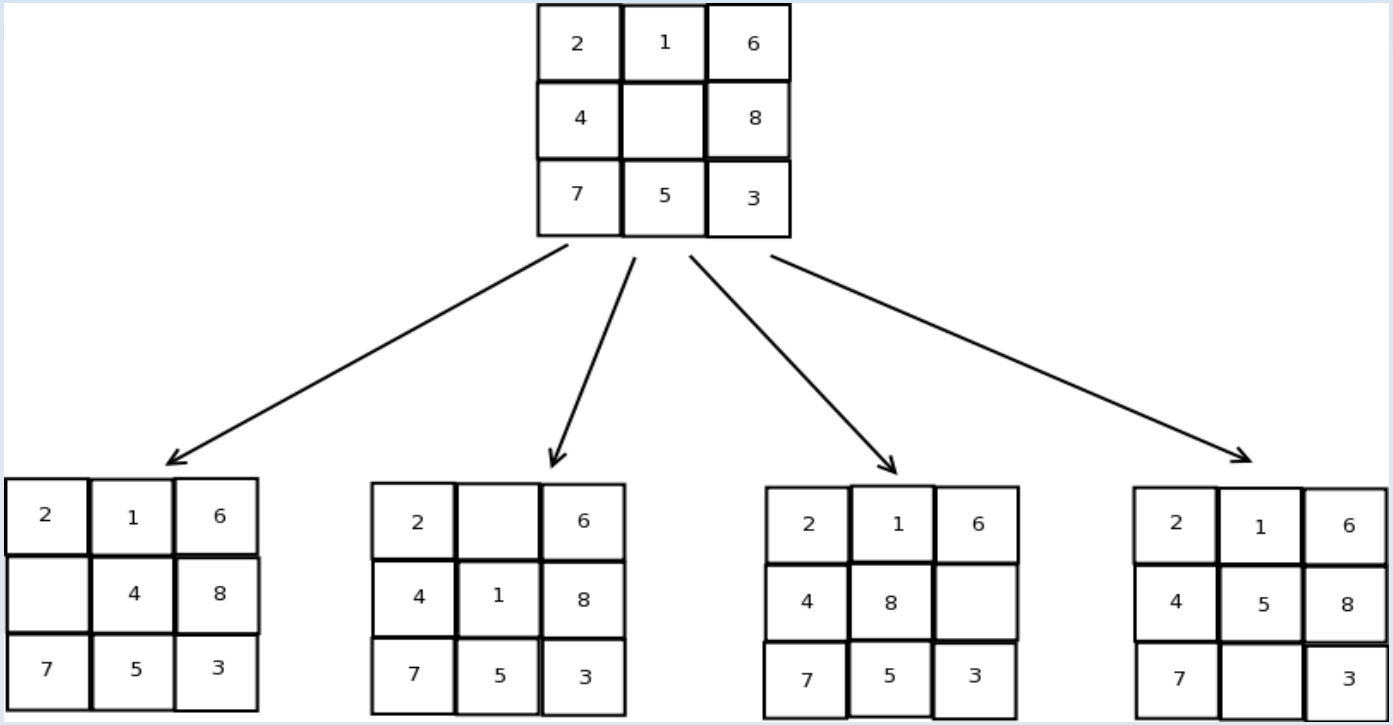
- El objetivo es llegar de un estado inicial cualquiera a el estado final:





El juego de el 8 Puzzle

- Para ello solo se podrán mover las fichas contiguas a el espacio:



El algoritmo A*

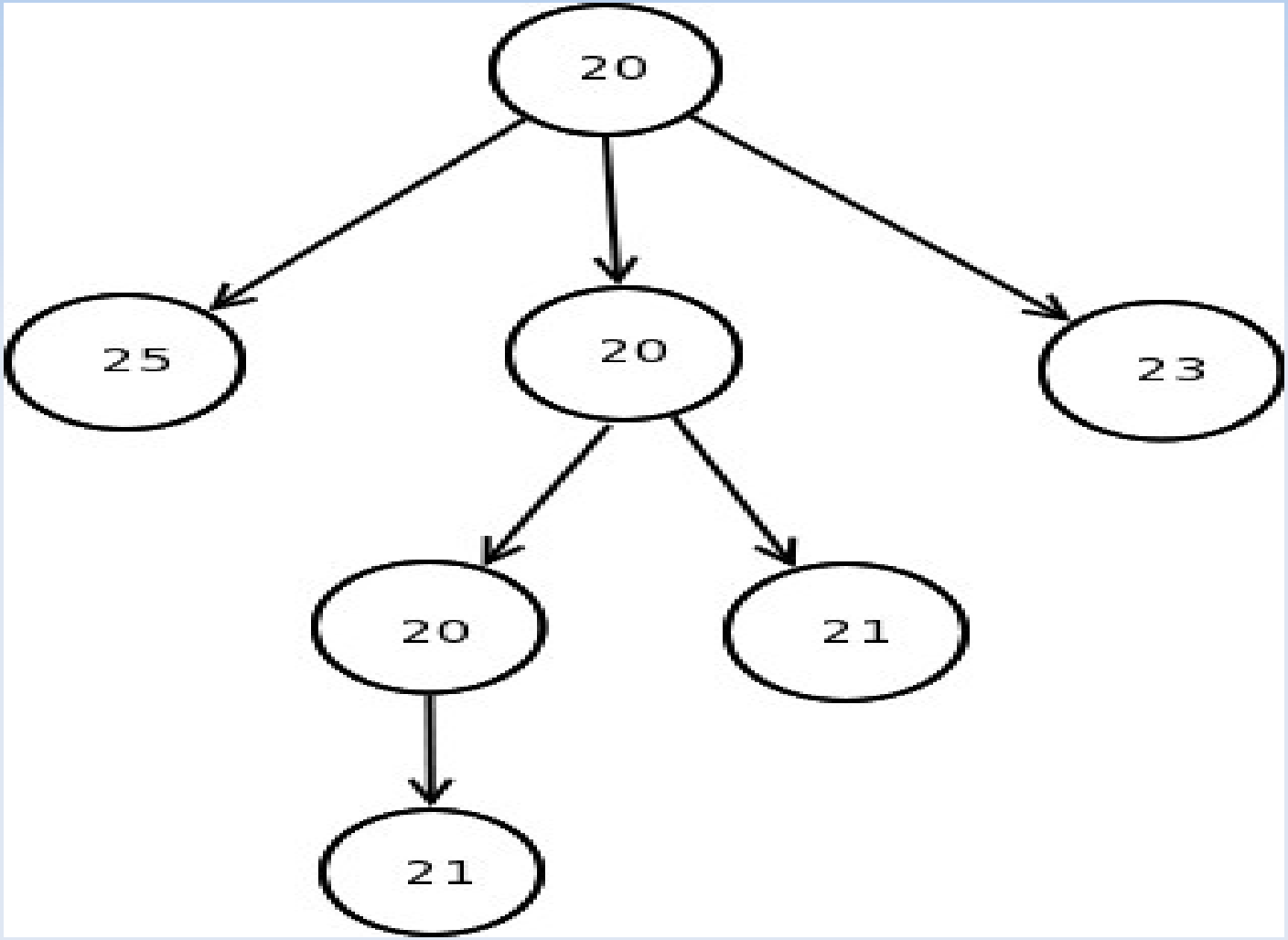
- Presentado por primera vez en 1968 por Peter E. Hart, Nils J. Nilsson y Bertram Raphael.
- Es un algoritmo de búsqueda en grafos.
- A* Utiliza una función de evaluación para cada nodo:

$$f(n) = g(n) + h(n)$$

El Algoritmo A*

- Cuando h(n) es admisible, asegura que la solución encontrada es la optima.
- Utiliza dos estructuras auxiliares:
  - Abiertos : es una cola de prioridad para los nodos no visitados.
  - Cerrados : es el conjunto de los nodos visitados.

El algoritmo A*



Heurística de Manhattan

- Consiste en la suma de la distancia de Manhattan de todas las casillas.

- La distancia de Manhattan viene definida :

$$|(x - x')| + |(y - y')|$$

Suma de Secuencias

- Se basa en la secuencia que debe seguir el estado final.

- Para cada ficha del tablero

  si la ficha es distinta del 0 :

  si esta en el centro → 1

  si no esta en el centro y su sucesor en la secuencia no es el correspondiente → 2

Heurística de Nilsson's

- Heuristica creada por Nilsson.

- Combina las dos anteriores:

$$HeuristicaManhattan + 3 * sumaSequencias$$

Implementación en Java

- Representación del Tablero:

```
private int estado[][];
private int blancoi;
private int blancoj;
```

- Ejemplo de movimientos :

```
public void moverarriba() throws MovimientonovalidoException{
    if(blancoi!=0){
        int i=estado[blancoi-1][blancoj];
        estado[blancoi][blancoj]=i;
        estado[blancoi-1][blancoj]=0;
        blancoi=blancoi-1;
    }else{
        throw new MovimientonovalidoException();
    }
}
```

Implementación en Java

- La interfaz Nodo:

```
public interface Nodo {

    public int getValorg();
    public void setValorg(int valorg);
    public int getValorh();
    public void setValorh(int valorh);
    public int getValorf();
    public void setValorf(int valorf);
    public Set<Nodo> Hijos();
    public boolean Solucion();
    public void Mostrar();

}
```

Implementación Java

```
Nodo actual=null;
boolean solucion=false;
while(!solucion&&!abiertos.isEmpty()){

    actual=abiertos.first();

    if(!actual.Solucion()){ //Si solución

        abiertos.remove(actual);
        cerrados.add(actual);

        Set<Nodo> nuevos=actual.Hijos(); ///Serian el conjunto de actual salir
        for(Nodo o:nuevos){
            //o.Mostrar();
            if(!cerrados.contains(o)){

                int nuevo_g_escore=actual.getValorg() +1; //El gescore deberia ser de actual
                o.setValorg(nuevo_g_escore);
                o.setValorf(nuevo_g_escore+o.getValorh());
                if(!abiertos.contains(o)){
                    abiertos.add(o);
                    camino.put(o, actual);
                }else if(nuevo_g_escore<CogerElemento(o).getValorg()){  //Si el camino nuevo es mejor
                    abiertos.remove(o);
                    abiertos.add(o);
                    camino.remove(o);
                    camino.put(o, actual);
                }
            }
        }
    }else{
        solucion=true;
    }
```

Implementación Haskell

```haskell
--Representacion del 8 Puzzle
type Puzzle = [Int] --Lista de longitud 9
```

```haskell
--Mover hueco (0) hacia arriba
moverarriba [x,y,z,0,a,b,c,d,e] = [0,y,z,x,a,b,c,d,e]
moverarriba [x,y,z,a,0,b,c,d,e] = [x,0,z,a,y,b,c,d,e]
moverarriba [x,y,z,a,b,0,c,d,e] = [x,y,0,a,b,z,c,d,e]
moverarriba [x,y,z,a,b,c,0,d,e] = [x,y,z,0,b,c,a,d,e]
moverarriba [x,y,z,a,b,c,d,0,e] = [x,y,z,a,0,c,d,b,e]
moverarriba [x,y,z,a,b,c,d,e,0] = [x,y,z,a,b,0,d,e,c]
moverarriba [x,y,z,a,b,c,d,e,f] = []
```

Implementación Haskell

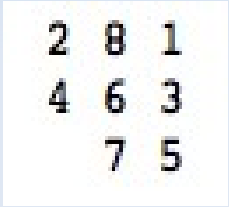
```haskell
-- Nodo para A estrella
data Nodo = Unodo { estado ::Puzzle,  ---  Estado del puzzle
                    g  :: Int,        ---  Profundidad a la que se encuentra el nodo
                    h  :: Int         ---  Valor heuristica del nodo
                  } deriving Show

valorNodo :: Nodo -> Int
valorNodo x = (g x) + (h x)

newNodo :: Puzzle -> Int -> Int -> Nodo
newNodo x y z = Unodo{estado=x,
                      g=y,
                      h=z
                      }

instance Eq Nodo where
        x == y = estado x == estado y
instance Ord Nodo where
        x <= y = valorNodo x <= valorNodo y
```

Implementación Haskell

```
--Aestrella

aEstrella x heuristica = aEstrellaExpandido (heuristica) [(newNodo x 0 (heuristica x))] [] [(introducirInicial (newNodo x 0 (heuristicaManhatan x)))]

aEstrellaExpandido he [] y camino= []
aEstrellaExpandido he ls@(x:xs) y z
        | x == (newNodo final 0 0) = listaSolucion z x [(estado x)]
        |otherwise = aEstrellaExpandido (he) (pongoHijos (he) (hijos (estado x))  xs (g x) y) (x:y) (cambiarCamino (he) (hijos (estado x)) x xs z y)

--Funciones auxiliares Para AExtrella

--Añade los hijos a la lista de abiertos si no estan incluidos o si su camino es peor
pongoHijos  he [] y z cerrados= y
pongoHijos he ls@(x:xs) y z cerrados
        |x==[] = pongoHijos (he) xs y z cerrados
        |(esta (newNodo x (z+1) (he x)) cerrados) = pongoHijos (he) xs y z cerrados
        |not (esta (newNodo x (z+1)(he x)) y ) = pongoHijos (he) xs (insertar (newNodo x (z+1) (he x)) y) z cerrados
        |mejor (newNodo x (z+1) (he x)) y = pongoHijos (he) xs (insertar (newNodo x (z+1) (he x)) (eliminar (newNodo x (z+1) (he x)) y)) z cerrados
        |otherwise = pongoHijos (he) xs y z cerrados
```

Resultados

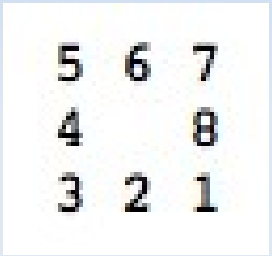
- Comparativa por Heurísticas en Java:



Heurística de Manhattan = 0,009s seg
Heurística de Nilsson's = 0,004 seg
Suma de Secuencias = 0,220 min

Resultados

- Comparativa por Heurísticas en Java:



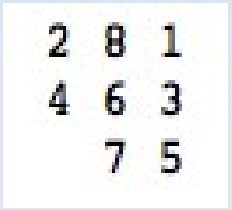
Heurística de Manhattan = 0,770 seg

Heurística de Nilsson's = 0,620 seg

Suma de Secuencias = 15 min

Resultados en Haskell

- Comparativa de Heurísticas en Haskell:



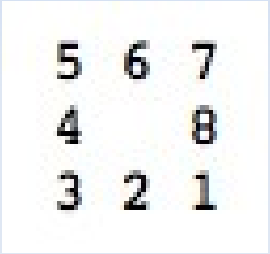
Heurística de Manhattan = 0,014 seg

Heurística de Nilsson's = 0,014 seg

Suma de Secuencias = 0,015 seg

Resultados

- Comparativa de Heurísticas en Haskell:



Heurística de Manhattan = 0,442 seg

Heurística de Nilsson's = 0,030 seg

Suma de Secuencias = 10 min

Resultados

- Haskell vs Java

Tablero 1	Haskell	Java
Manhattan	0,014 seg	0,009 seg
Nilsson's	0,014 seg	0,004 seg
Secuencias	0,015 min	0,220 min

Tablero 2	Haskell	Java
Manhattan	0,442 seg	0,770 seg
Nilsson's	0,030 seg	0,620 seg
Secuencias	10 min	15 min

Alternativa en Haskell

- Representación

```
type Position = (Int,Int)
data Board = Board (Array Int Position) deriving Eq
```

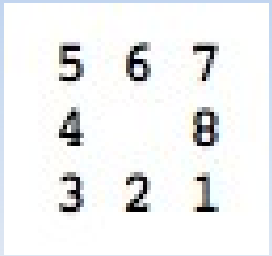
- Heurística y movimientos

```
-- Manhattan distance from b to goal
manhattan :: Board -> Int
manhattan b = sum $ zipWith dist (positions b) (positions goal)
 where
  positions (Board b) = [ b!i | i <- [0..8] ]

-- Boards obtained after doing a single move
oneMove :: Board -> [Board]
oneMove (Board b) = map Board [ b//[(0,b!i),(i,b!0)]
                              | i<-[1..8], dist (b!0) (b!i) == 1
                              ]
```

Alternativa en Haskell

```
module EightPuzzle where

import Graph
import PriorityFirstSearch
import EightBoard

-- heuristic to use
instance Ord Board where
 b <= b' = manhattan b <= manhattan b'

-- graph for 8-puzzle
eightPuzzleGraph :: Graph Board
eightPuzzleGraph = mkGraphSuc undefined oneMove

-- find path to goal
solve :: Board -> [Board]
solve b = head . filter ((== goal) . last) $ pftPaths eightPuzzleGraph b

initial :: Board
initial = readBoard "2 4316758"

initial2 :: Board
initial2 = readBoard "7245 6831"

main :: IO ()
main = mapM_ print (solve initial)
```

Comparativa entre ambos programas en Haskell



Tiempo Haskell A* = 0.380 seg

Tiempo Haskell Grafos = 0.160 seg

Bibliografía y Referencias

- Principles of Artificial Intelligence, Nils J. Nilsson,Ed. Morgan Kaufmann,1982
- http://en.wikipedia.org/wiki/A*_search_algorithm
- http://heuristicswiki.wikispaces.com/N+-+Puzzle
- http://www.8puzzle.com/8_puzzle_problem.html