# Relatório - Busca Heurística (8-puzzle)

Alunos:

- Caio Cargnin Cardoso
- Diego Marzarotto

---

## Implementação

O programa consiste de um script python, que utiliza 3 classes ao realizar a busca A* para encontrar a solução do jogo 8-puzzle.

Além das bibliotecas nativas do python, o script usa também o *numpy*, portanto, antes de rodar o script é necessário instalar a biblioteca. Caso esteja usado o *pip* para gerenciar as dependências, basta rodar:

```
sudo pip install -r requirements.txt
```

Caso contrário, instale o pip antes. No Ubuntu basta executar:

```
sudo apt-get install python-pip
```

Para instruções em outros sistemas operacionais, consulte o google. :) Para utilizar o script, ele deve ser executado em um terminal.

```
./eightpuzzle.py
```

Ao ser executado, o script apresenta a seguinte saída para o usuário:

```
### 8-puzzle ###

Entre com um estado do tabuleiro,
com os números separados por espaços,
representando o espaço vazio com um X:

Ex: 1 2 3 4 5 6 7 8 X = equivalente a |1 2 3|
                                      |4 5 6|
                                      |7 8 X|

>
```

Que deve então entrar com um estado válido do tabuleiro:

```
> 1 2 3 4 5 6 7 X 8
```

Em seguida é mostrado o caminho até a solução:

```
|1 2 3|
|4 5 6|
|7 X 8|
   |
   v
|1 2 3|
|4 5 6|
|7 8 X|

## Número de nodos visitados: 2
## Maior quantidade de nodos na fronteira: 3
## Número de passos: 1
## Heurística utilizada: h2
```

---

Podem ser passadas as diferentes heurísticas implementadas como parâmetros para o script.

```
./eighpuzzle.py h1
./eighpuzzle.py h2
./eighpuzzle.py h3
```

---

A seguir, uma breve descrição das classes existentes, a assinatura e documentação dos principais métodos:

---

**Classe Nodo**

Responsável por representar um estado do tabuleiro.

Esta classe também é capaz de gerar seus estados sucessores,
além de saber determinar se é ou não um nodo objetivo.
O cálculo do custo e das heurísticas também é responsabilidade do Nodo.

O cálculo do custo é representado através da função f:

- **f**: número de movimentos anteriores até se atingir o estado atual;

Três heurísticas distintas podem ser utilizadas:

- **h1**: número de peças fora do lugar;
- **h2**: distância manhattan de cada peça em relação ao seu lugar original;
- **h3**: média simples entre h1 e h2;

*Referência: Russel; Norvig, 2010*

---

```
def f(self):
    '''Cálculo do custo, baseado na quantidade movimentos até se atingir o estado atual.'''
    ...

def h1(self):
    '''Heurística baseada no número de peças fora do lugar.'''
    ...

def h2(self):
    '''Heurística baseada na distância manhattan de cada peça'''
    ...

def h3(self):
    '''Heurística baseada na média simples entre h1 e h2.'''
    ...
```

**Classe BuscaHeuristica8Puzzle**

---

Responsável por realizar a busca A* da solução para o jogo 8-puzzle.

Implementa a lógica de busca da solução para o jogo 8-puzzle, utilizando para isso o algoritmo A*, no qual utiliza-se uma heurística para estimar o custo de cada estado do tabuleiro em relação ao estado final do jogo.

*Referência: Russel; Norvig, 2010*

---

```python
def buscar(self, estado_atual):
    '''Método responsável pela busca propriamente dita.

    Para armazenar os nodos presentes na fronteira, durante a busca,
    foi utilizada uma lista de prioridade, que apesar de ser uma lista simples do python,
    é manipulada utilizando a biblioteca 'heapq', nativa da linguagem,
    que permite que operações de inserção e remoção ('pop') mantenham
    as propriedades de ordenação da lista de prioridade.

    Dessa forma, garante-se que o próximo elemento a ser retirado
    através da operação "pop" seja o Nodo com menor custo estimado;

    Para auxiliar na verificação de Nodos já presentes na fronteira,
    é também utilizada uma tabela de hash ('dict' do python),
    na qual a chave é uma função hash baseada apenas no estado de cada Nodo.

    '''
    ...


def existe_solucao(self, estado_atual):
    '''Método responsável por verificar se existe solução para um estado.

    A verificação implementada foi com base no número de inversões do tabuleiro.
    Inversões são ocorrências de um número maior aparecendo antes de outro número menor.
    Caso o número de inversões no tabuleiro seja ímpar, não existe solução possível.
    '''
    ...
```

---

**Classe Interface**

Responsável por interagir com o usuário e apresentar o resultado da busca.

Além de mostrar o caminho percorrido até a solução,
apresenta também o número de passos,
o tamanho máximo da fronteira durante a busca
e o número de nodos visitados até a solução.

---

## Exemplos

**Um exemplo mínimo, utilizando a heurística h2, com dois passos de distância em relação ao objetivo:**

---

```
|1 2 3|
|4 5 6|
|X 7 8|
   |
   v
|1 2 3|
|4 5 6|
|7 X 8|
   |
   v
|1 2 3|
|4 5 6|
|7 8 X|

## Número de nodos visitados: 3
## Maior quantidade de nodos na fronteira: 3
## Número de passos: 2
## Heurística utilizada: h2
```

---

**Um exemplo maior, exemplificando a diferença entre as heurísticas:**

---

```
busca = BuscaHeuristica8Puzzle(heuristica='h1')
nodo_final = busca.buscar(np.array([
    [1    , 2    , 3    ],
    [4    , 5    , None ],
    [6    , 7    , 8    ]
]))
Interface().mostrar_solucao(busca, nodo_final, resumido=True)

## Número de nodos visitados: 165
## Maior quantidade de nodos na fronteira: 105
## Número de passos: 13
## Heurística utilizada: h1
```

```
busca = BuscaHeuristica8Puzzle(heuristica='h2')
nodo_final = busca.buscar(np.array([
    [1    , 2    , 3    ],
    [4    , 5    , None ],
    [6    , 7    , 8    ]
]))
Interface().mostrar_solucao(busca, nodo_final, resumido=True)
```

```
## Número de nodos visitados: 81
## Maior quantidade de nodos na fronteira: 56
## Número de passos: 13
## Heurística utilizada: h2
```

```
busca = BuscaHeuristica8Puzzle(heuristica='h3')
nodo_final = busca.buscar(np.array([
    [1    , 2    , 3    ],
    [4    , 5    , None ],
    [6    , 7    , 8    ]
]))
Interface().mostrar_solucao(busca, nodo_final, resumido=True)
```

```
## Número de nodos visitados: 102
## Maior quantidade de nodos na fronteira: 64
## Número de passos: 13
## Heurística utilizada: h3
```

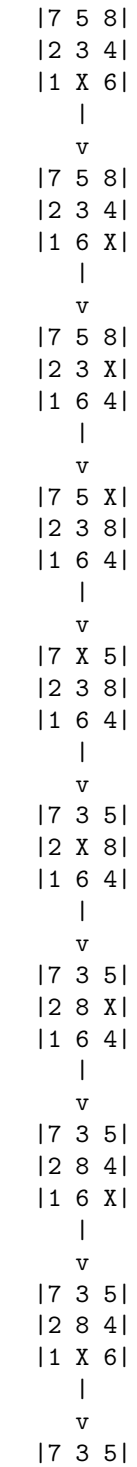
## Problemas / Limitações

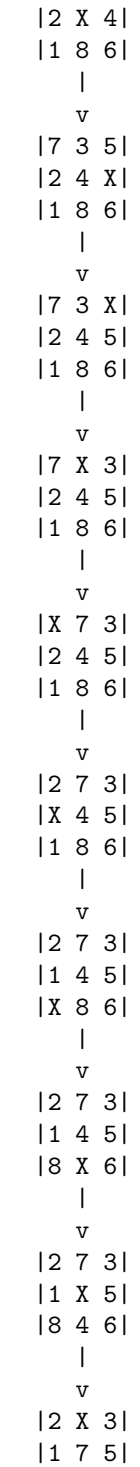
Como sugerido na apresentação junto ao professor, foi implementada posteriormente uma verificação de estados do tabuleiro que não são possíveis de serem resolvidos.

A verificação implementada foi com base no número de inversões do tabuleiro. Inversões são ocorrências de um número maior aparecendo antes de outro número menor. Caso o número de inversões no tabuleiro seja ímpar, não existe solução possível.

O exemplo com maior número de passos que conseguimos resolver foi
com um estado distante 2 passos da solução, utilizando a heurística h2.
As outras heurísticas não conseguiram executar uma busca com tantos passos.

---


|7 5 8|
|2 3 4|
|1 X 6|
|
v
|7 5 8|
|2 3 4|
|1 6 X|
|
v
|7 5 8|
|2 3 X|
|1 6 4|
|
v
|7 5 X|
|2 3 8|
|1 6 4|
|
v
|7 X 5|
|2 3 8|
|1 6 4|
|
v
|7 3 5|
|2 X 8|
|1 6 4|
|
v
|7 3 5|
|2 8 X|
|1 6 4|
|
v
|7 3 5|
|2 8 4|
|1 6 X|
|
v
|7 3 5|
|2 8 4|
|1 X 6|
|
v
|7 3 5|


|2 X 4|
|1 8 6|
|
v
|7 3 5|
|2 4 X|
|1 8 6|
|
v
|7 3 X|
|2 4 5|
|1 8 6|
|
v
|7 X 3|
|2 4 5|
|1 8 6|
|
v
|X 7 3|
|2 4 5|
|1 8 6|
|
v
|2 7 3|
|X 4 5|
|1 8 6|
|
v
|2 7 3|
|1 4 5|
|X 8 6|
|
v
|2 7 3|
|1 4 5|
|8 X 6|
|
v
|2 7 3|
|1 X 5|
|8 4 6|
|
v
|2 X 3|
|1 7 5|

```
|8 4 6|
   |
   v
|X 2 3|
|1 7 5|
|8 4 6|
   |
   v
|1 2 3|
|X 7 5|
|8 4 6|
   |
   v
|1 2 3|
|7 X 5|
|8 4 6|
   |
   v
|1 2 3|
|7 4 5|
|8 X 6|
   |
   v
|1 2 3|
|7 4 5|
|X 8 6|
   |
   v
|1 2 3|
|X 4 5|
|7 8 6|
   |
   v
|1 2 3|
|4 X 5|
|7 8 6|
   |
   v
|1 2 3|
|4 5 X|
|7 8 6|
   |
   v
|1 2 3|
|4 5 6|
|7 8 X|
```

```
## Número de nodos visitados: 4792
## Maior quantidade de nodos na fronteira: 2595
## Número de passos: 27
## Heurística utilizada: h2
```