UNIVERSIDADE FEDERAL DE SÃO CARLOS
Disciplina de Inteligência Artificial - Turma A

Bianca Garcia Martins
RA: 606723

Rebecca Fernandes Santos
RA: 726584

Para cada exercícios foi criado um arquivo.R (Cidade.R e QuebraCabeca.R), baseado no arquivo modelo de estados Estado.R, e um exemplo.R (exemploCidade.R e exemploQuebraCabeca.R). Para ambos modificou-se por completo o arquivo exemplo.R, definindo os estado inicial e final,e chamando os algoritmos correspondentes para solucionar o exercício, e modificou- se as funções que definiam a heurística e a geração de filhos no arquivo Estado.R.

Desconsiderou-se os algoritmos de busca desinformada, uma vez que serão apenas utilizados os algoritmos greedy e A*, que são buscas informadas.

O algoritmo guloso (greedy) utiliza apenas a função heurística, e o algoritmo A* usa a função heurística somada ao custo, que resulta na função avaliativa.

1. Quebra Cabeça de 8 peças

A representação dos estados foi feita através de matrizes 3x3. Cada peça tem um número único, os quais foram retirados do slide utilizado em aula (Imagem 1.1). O arquivo exemploQuebraCabeca.R tem definido as matrizes inicial e objetivo, as quais determinam as posições iniciais e finais das peças (Imagens 1.2 e 1.3).

2	8	3
1	6	4
7		5

Estado Inicial

1	2	3
8		4
7	6	5

Estado Final

Imagem 1.1 - Estado inicial e final do quebra-cabeça.
Fonte: Slide disponibilizado pela professora Heloisa.

```r
### Matriz inicial
inicial = matrix(
  c(2, 8, 3, 1, 6, 4, 7, 0, 5),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)

### Matriz objetivo
final = matrix(
  c(1, 2, 3, 8, 0, 4, 7, 6, 5),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)
```

Imagem 1.2 - Declaração das matrizes inicial e objetivo.

```
> print(inicial)
     [,1] [,2] [,3]
[1,]    2    8    3
[2,]    1    6    4
[3,]    7    0    5
```

```
> print(final)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    8    0    4
[3,]    7    6    5
```

Imagem 1.3 - Print das matrizes inicial e final.

Os operadores aplicáveis aos estados são as movimentações do espaço sem peça, representado pelo número 0. São eles: Ir para cima, ir para baixo, ir para a esquerda e ir para a direita. Todos de custo unitário. No entanto, dependendo a posição atual do espaço sem peça, ele não pode realizar alguns movimentos (Imagem 1.4), por exemplo,quando ele se localiza nas margens do tabuleiro (coluna e linha iguais a 1 ou 3).

```
81    ### Troca com o de cima
82    if (rowZero > 1) {
83      newDesc = matrixCopy(desc)
84      newDesc[rowZero - 1, colZero] = desc[rowZero, colZero]
85      newDesc[rowZero, colZero] = desc[rowZero - 1, colZero]
86      filhosDesc = c(filhosDesc, list(newDesc))
87    }
88    ### Troca com o de baixo
89    if (rowZero < 3) {
90      newDesc = matrixCopy(desc)
91      newDesc[rowZero + 1, colZero] = desc[rowZero, colZero]
92      newDesc[rowZero, colZero] = desc[rowZero + 1, colZero]
93      filhosDesc = c(filhosDesc, list(newDesc))
94    }
95    ### Troca com o da esquerda
96    if (colZero > 1) {
97      newDesc = matrixCopy(desc)
98      newDesc[rowZero, colZero - 1] = desc[rowZero, colZero]
99      newDesc[rowZero, colZero] = desc[rowZero, colZero - 1]
100     filhosDesc = c(filhosDesc, list(newDesc))
101   }
102   ### Troca com o da direita
103   if (colZero < 3) {
104     newDesc = matrixCopy(desc)
105     newDesc[rowZero, colZero + 1] = desc[rowZero, colZero]
106     newDesc[rowZero, colZero] = desc[rowZero, colZero + 1]
107     filhosDesc = c(filhosDesc, list(newDesc))
108   }
109
110   ## gera os objetos QuebraCabeca para os filhos
111   for(filhoDesc in filhosDesc){
112     filho <- QuebraCabeca(desc = filhoDesc, pai = obj)
113     filho$h <- heuristica(filho)
114     filho$g <- obj$g + 1
115     filhos <- c(filhos, list(filho))
116   }
```

Imagem 1.4 - Troca de posição entre o espaço em branco e peças adjacentes, se possível.

A função heurística que foi implementada é a soma das distâncias de cada peça em relação ao seu lugar final, chamada distância de Manhattan (Imagem 1.5).

Distância de Manhattan: | x1 - x2 | + | y1 - y2 |, onde
x1 e y1 são, respectivamente, a linha e a coluna onde se encontra a peça atualmente, e x2 e y2 a linha e coluna da posição objetivo da peça.

O filho escolhido como próximo a ser gerado é aquele que tiver a menor heurística.

```r
  h = 0
  for (i in 1:nrow(matriz)) {
    for (j in 1:ncol(matriz)) {
      valor = matriz[i, j]
      pos1 = c(row = i, col = j)
      pos2 = which(objetivo == valor, arr.ind = T)
      h = h + abs(pos1[1] - pos2[1]) + abs(pos1[2] - pos2[2]) # Cálculo da distância
    }
  }
  return(h)
}
```

Imagem 1.5 - Implementação do cálculo da heurística.

Também foi impresso diversas matrizes mostrando o cada movimento do espaço sem peça, tanto para a busca gulosa (Imagem 1.6) quanto para o algoritmo A* (Imagem 1.7).

```
> print(buscaBestFirst(inicial, objetivo, "Gulosa"))
[[1]]
     [,1] [,2] [,3]
[1,]    2    8    3
[2,]    1    6    4
[3,]    7    0    5
G(n):  0   H(n):  Inf   F(n):  Inf
[[2]]
     [,1] [,2] [,3]
[1,]    2    8    3
[2,]    1    0    4
[3,]    7    6    5
G(n):  1   H(n):  4   F(n):  4
[[3]]
     [,1] [,2] [,3]
[1,]    2    0    3
[2,]    1    8    4
[3,]    7    6    5
G(n):  2   H(n):  4   F(n):  4
[[4]]
     [,1] [,2] [,3]
[1,]    0    2    3
[2,]    1    8    4
[3,]    7    6    5
G(n):  3   H(n):  4   F(n):  4
[[5]]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    0    8    4
[3,]    7    6    5
G(n):  4   H(n):  2   F(n):  2
[[6]]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    8    0    4
[3,]    7    6    5
G(n):  5   H(n):  0   F(n):  0
```

Imagem 1.6 - Impressão das matrizes representando cada troca entre o espaço vazio e uma peça adjacente, usando o algoritmo guloso.

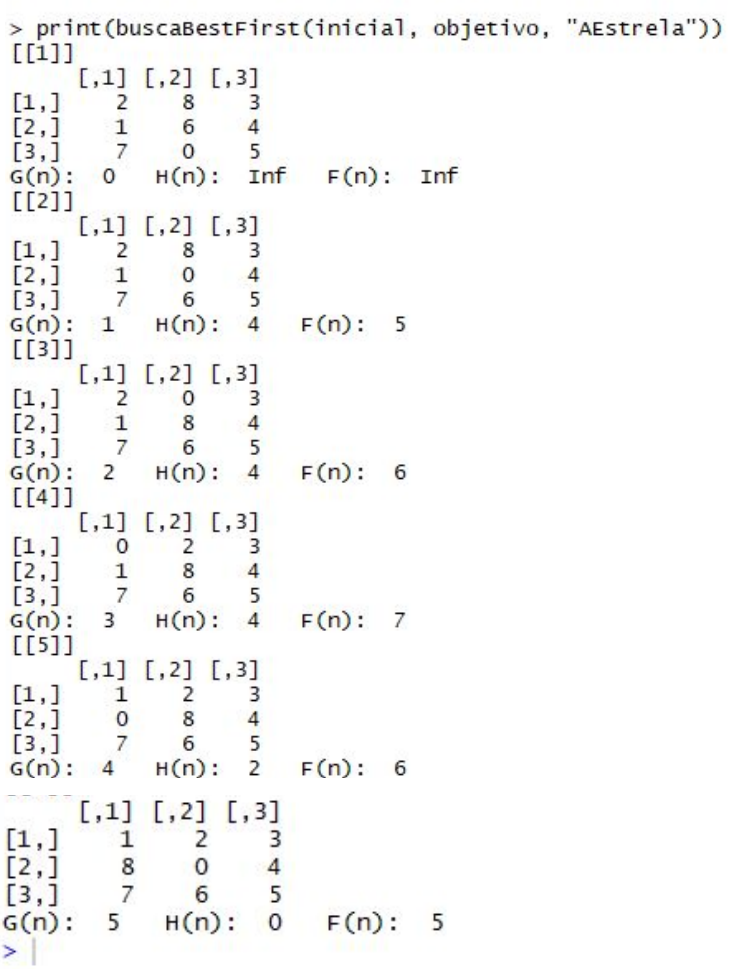

Imagem 1.7 - Impressão das matrizes representando cada troca entre o espaço vazio e uma peça adjacente, usando o algoritmo A*.

2. Trajeto entre duas cidades

Os estados foram representados em uma lista de cidades no arquivo exemploCidade.R, onde cada uma é um estado (Imagem 2.1). O ponto inicial é a cidade A, e o ponto final é a cidade B. As cidades E, H, I, N, V foram ignoradas já que não é possível que chegar de A até B pela primeira vez, passando por essas cidades. Os exemplos de cidades foram retirados do mapa de cidades da Romenia, dado em aula.

```
4  cidades <- c("A", "B", "C", "D","F", "G", "L", "M","O", "P", "R", "S", "T", "U", "Z")
```

Imagem 2.1 - Lista de cidades que serão os estados.

O operador aplicável é partir de uma cidade para a próxima.

Para se definir os custos foi criada uma matriz que mapeava as distância em quilômetros entre as cidades. Os valores foram obtidos do material apresentado pela professora.

Caso deseja-se verificar esta matriz, basta descomentar a linha 47 do código exemploCidade.R correspondente ao seguinte código: print(distanciakm()) (Imagem 2.2). As linhas e as colunas representam as cidades.

```
> print(distanciakm())
    A   B   C   D   F  G   L  M   O   P   R   S   T  U  Z
A   0   0   0   0   0  0   0  0   0   0   0 140 118  0 75
B   0   0   0   0 211 90   0  0   0 101   0   0   0 85  0
C   0   0   0 120   0  0   0  0   0 138 146   0   0  0  0
D   0   0 120   0   0  0   0 75   0   0   0   0   0  0  0
F   0 211   0   0   0  0   0  0   0   0   0  99   0  0  0
G   0  90   0   0   0  0   0  0   0   0   0   0   0  0  0
L   0   0   0   0   0  0   0 70   0   0   0   0 111  0  0
M   0   0   0  75   0  0  70  0   0   0   0   0   0  0  0
O   0   0   0   0   0  0   0  0   0   0   0 151   0  0 71
P   0 101 138   0   0  0   0  0   0   0  97   0   0  0  0
R   0   0 146   0   0  0   0  0   0  97   0  80   0  0  0
S 140   0   0   0  99  0   0  0 151   0  80   0   0  0  0
T 118   0   0   0   0  0 111  0   0   0   0   0   0  0  0
U   0   0  85   0   0  0   0  0   0   0   0   0   0  0  0
Z  75   0   0   0   0  0   0  0  71   0   0   0   0  0  0
>
```

Imagem 2.2 - Matriz com a distância em quilômetros entre as cidades.

Foi criada outra matriz de heurísticas, com valores já definidos no exemplo do material citado anteriormente, onde cada valor representa a distância em linha reta da cidade em questão até a cidade B. Caso deseja-se visualizar esta matriz, basta descomentar a linha 19 do código exemploCidade.R correspondente ao seguinte código: print(distanciareta()) (Imagem 2.3).

```
> print(distanciareta())
  Distância em linha reta até B
A                           366
B                             0
C                           160
D                           242
F                           178
G                            77
L                           244
M                           241
O                           380
P                            98
R                           193
S                           253
T                           329
U                            80
Z                           374
>
```

Imagem 2.3 - Matriz com as heurísticas de cada cidade.

Os valores de custo e heurística foram definidos no arquivo de exemplo. Entretando a heurística foi definida novamente no arquivo Cidade.R, na função que define um valor à ela. Nessa mesma função, foi definido novamente a lista de cidades (Imagem 2.1) para que se pudesse nomear as linhas da matriz de heurísticas.

Em Cidade.R, foi modificado a função construtora Cidade. Passa-se como parâmetro a matriz de custo, utilizado na função de geração de filhos.

Foi impresso o resultado de best-first gerado pelo algoritmo guloso e pelo algoritmo A* (Imagem 2.4).

```
====    Busca Best-First (Gulosa)       =====

> print(buscaBestFirst(inicial, objetivo, "Gulosa"))
[[1]]
Cidade:  A
G(n):  0   H(n):  Inf   F(n):  Inf
[[2]]
Cidade:  S
G(n):  140   H(n):  253   F(n):  253
[[3]]
Cidade:  F
G(n):  239   H(n):  178   F(n):  178
[[4]]
Cidade:  B
G(n):  450   H(n):  0   F(n):  0

> ##Será utilizado o algoritmo A*, onde é utilizado a função de custo e heurística.
> cat("====\tBusca Best-First (A*)\t=====\n")
====    Busca Best-First (A*)   =====

> print(buscaBestFirst(inicial, objetivo, "AEstrela"))
[[1]]
Cidade:  A
G(n):  0   H(n):  Inf   F(n):  Inf
[[2]]
Cidade:  S
G(n):  140   H(n):  253   F(n):  393
[[3]]
Cidade:  R
G(n):  220   H(n):  193   F(n):  413
[[4]]
Cidade:  P
G(n):  317   H(n):  98   F(n):  415
[[5]]
Cidade:  B
G(n):  418   H(n):  0   F(n):  418
>
```

Imagem 2.4 - Impressão dos resultados de ambos os algoritmos.