Relatório do primeiro trabalho

Inteligência Artificial

Turma - B

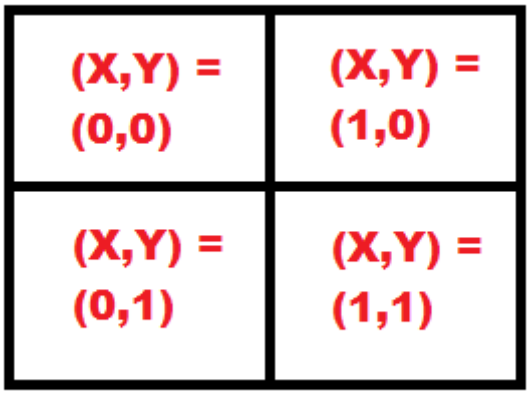
Algoritmos de Busca

Igor Bianchi – 558400

Felipe Chinen – 496235

Descrição do algoritmo de tratamento do problema

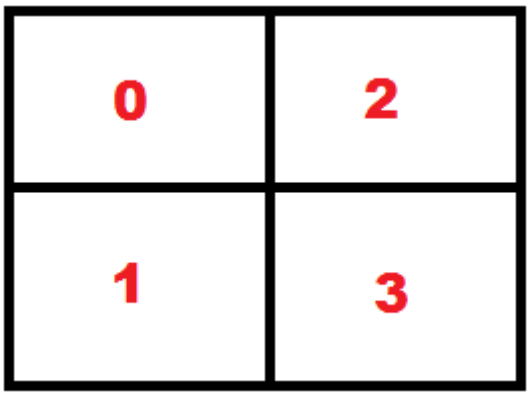
A nossa formulação da resolução do problema partiu da descrição do espaço em que ele está inserido. Trata-se de um "tabuleiro" 2x2 em que cada "casa" foi determinada por coordenadas X e Y, variando de 0 a 1.



Para que fosse mais fácil a utilização destas coordenadas no nosso algoritmo, aplicamos uma fórmula simples que transforma as duas coordenadas em apenas um número.

$$doisXMaisY(X,Y) = 2 * X + Y$$

Então, o nosso cenário fica da seguinte forma:



A partir desta transformação, também configuramos uma forma de representar os quadrados sujos utilizando apenas uma variável de entrada. É uma generalização de um vetor, a variável pode ter 4 dígitos e cada um representa um quadrado do cenário. Se o dígito que representa o quadrado estiver ativo, ou seja, com valor 1, significa que ele está sujo. O caso contrário é quando ele não está ativo, ou seja, valor 0.

Ex: Todos quadrados sujos: 1111. Todos limpos: 0. Quadrado 0 sujo: 1000. Quadrado 3 sujo: 0001.

Como entrada para resolução do problema, o usuário deve informar as coordenadas do quadrado de partida e os que estão sujos.

Ex: *inicial <- Aspirador(desc = c(X = 1, Y = 1, 1110))*

Neste exemplo o quadrado de partida é o 3 e os sujos são 0, 1 e 2.

O **estado inicial** pode variar dentro do intervalo permitido.

O **estado objetivo** da solução é deixar todos os quadrados sujos limpos, ou seja, com valor 0, não importando as coordenadas em que o aspirador terminou. Isto foi feito com a sobrecarga do operador "=="

```
Ops.Aspirador = function(obj1, obj2){
    if(.Generic == "=="){
        return(all(obj1$desc[3] == obj2$desc[1]))
    }
```

```
}
```

Exemplo de inicialização do objetivo:

```
objetivo <- Aspirador()
objetivo$desc <- c(0)
```

Dessa forma, no objetivo não se passa a informação das coordenadas.

A **descrição dos estados** do problema foi feita detectando onde o aspirador está atualmente. Por exemplo, se estiver no quadrado 0, ele pode se mover para baixo ou para direita, além de poder aspirar se houver sujeira.

Em cada quadrado ele opera com os mesmos movimentos de outro, mover-se para cima ou baixo, esquerda ou direita, permanecer no mesmo quadrado. Porém, mais adiante é verificado se as coordenadas estão fora do intervalo permitido (X,Y) = (0~1,0~1), se estiverem este filho é descartado da lista de filhos.

Exemplo da **representação do operador** do quadrado 0:

```
if(doisXMaisY == 0){
    operadores <- list(c(0,0,0), c(0,0,1000), c(-1,0,0), c(0,-1,0), c(1,0,0), c(0,1,0))
    filhosDesc <- lapply(operadores, function(op) desc-op)
```

São cinco operadores possíveis para cada quadrado e podem ser entre 3 e 4 estados válidos dependendo da posição e dos quadrados sujos.

O operador para aspirar o quadrado é a subtração do valor correspondente ao que representa. Por exemplo, o quadrado 0 tem como representação de sujeira o número 1000, então será subtraído este número em um dos operadores. Para tratamento de estados inválidos foi criada uma função que verifica se o número resultante é binário, se não for, significa que o quadrado já estava limpo e então este filho é descartado.

Função que verifica os quadrados sujos válidos:

```
verificaBinario <- function(valor){
  copia <- valor
  while(copia != 0){
    if(copia %% 10 > 1)
      return (FALSE)
    copia <- copia %/% 10
  }
  return (TRUE)
}
```

O tratamento de filhos válidos, como dito anteriormente, leva em consideração se a posição e o número de quadrados sujos são válidos. Se não for um filho válido, este é dito incompatível e retirado da lista de filhos.

```
incompativeis <- sapply(1:length(filhosDesc),
                        function(i) {
                          fDesc <- filhosDesc[[i]]
                          verificacaoBinario <- verificaBinario(fDesc[3])
                          if((any(fDesc[1:2] < 0)) ||     ##    x ou y menor que 0
                             (any(fDesc[1:2] > 1)) ||   ##    x ou y maior que 1
                             (!verificacaoBinario))     ## valor dos quadrados sujos válido
                            i ## é incompatível: retorna índice
                          else
                            0 ## senão é compatível
                        })
## mantém no vetor apenas os que são incompatíveis
incompativeis <- incompativeis[incompativeis != 0]
## remove estados filhos incompatíveis
filhosDesc <- filhosDesc[-incompativeis]
```

A **função heurística** que escolhemos é bem simples porém relevante, ela descreve a quantidade de quadrados sujos restantes do problema.

```
heuristica.Aspirador <- function(atual){
  if(is.null(atual$desc[3]))
    return(Inf)
  ## h(obj) = quantidade de quadrados sujos restantes
  return(sapply(strsplit(as.character(atual$desc[3]), ""), function (x) sum(x == 1)))
}
```

Ela soma todos os dígitos do campo de quadrados sujos que estejam representados por 1. Por exemplo, se o valor for 1001, a função transformará em uma lista de apenas 11 e somará seus dígitos, resultando em 2.

O **custo** de cada ação é determinado utilizando o estado em que estava o pai e o que aconteceu para se chegar no filho, já que o custo é varia de acordo com a posição ou se o quadrado foi aspirado.

```
for(filhoDesc in filhosDesc){
    filho <- Aspirador(desc = filhoDesc, pai = obj)
    filho$h <- heuristica(filho)
    ## verifica de qual o estado o filho surgiu
    if(obj$desc[1] != filho$desc[1]) ## filho tem coordenada X diferente do pai
      filho$g <- obj$g + 1          ## então filho recebe um custo + 1
    if(obj$desc[2] != filho$desc[2]) ## filho tem coordenda y diferente do pai
      filho$g <- obj$g + 3          ## então filho recebe um custo + 3
    if(obj$desc[3] != filho$desc[3]) ## filho tem quadrados sujos diferentes do pai
      filho$g <- obj$g + 2          ## então filho recebe um custo + 2
    filhos <- c(filhos, list(filho))
  }
```

Saída dos algoritmos de busca

Entrada de exemplo:

```
debugSource("Aspirador.R")
debugSource("Canibais.R")
debugSource("buscaDesinformada.R")
debugSource("buscaInformada.R")

 inicial <- Aspirador(desc = c(X = 1, Y = 1, 1110))
objetivo <- Aspirador()
objetivo$desc <- c(0)
```

Busca em largura

```
[[1]]
(X Y QuadradosSujos): ( 1 1 1110 )
G(n):  0
H(n):  Inf
F(n):  Inf

[[2]]
(X Y QuadradosSujos): ( 0 1 1110 )
G(n):  1
```

H(n): 3
F(n): Inf

[[3]]
(X Y QuadradosSujos): (0 1 1010)
G(n): 3
H(n): 2
F(n): Inf

[[4]]
(X Y QuadradosSujos): (0 0 1010)
G(n): 6
H(n): 2
F(n): Inf

[[5]]
(X Y QuadradosSujos): (0 0 10)
G(n): 8
H(n): 1
F(n): Inf

[[6]]
(X Y QuadradosSujos): (1 0 10)
G(n): 9
H(n): 1
F(n): Inf

[[7]]
(X Y QuadradosSujos): (1 0 0)
G(n): 11
H(n): 0
F(n): Inf

Busca em profundidade

```
[[1]]
(X Y QuadradosSujos): ( 1 1 1110 )
G(n):  0
H(n):  Inf
F(n):  Inf

[[2]]
(X Y QuadradosSujos): ( 0 1 1110 )
G(n):  1
H(n):  3
F(n):  Inf

[[3]]
(X Y QuadradosSujos): ( 0 1 1010 )
G(n):  3
H(n):  2
F(n):  Inf

[[4]]
(X Y QuadradosSujos): ( 1 1 1010 )
G(n):  4
H(n):  2
F(n):  Inf
```

```
[[5]]
(X Y QuadradosSujos): ( 1 0 1010 )
G(n):  7
H(n):  2
F(n):  Inf

[[6]]
(X Y QuadradosSujos): ( 1 0 1000 )
G(n):  9
H(n):  1
F(n):  Inf

[[7]]
(X Y QuadradosSujos): ( 0 0 1000 )
G(n):  10
H(n):  1
F(n):  Inf

[[8]]
(X Y QuadradosSujos): ( 0 0 0 )
G(n):  12
H(n):  0
F(n):  Inf
```

Busca de custo uniforme

```
[[1]]
(X Y QuadradosSujos): ( 1 1 1110 )
G(n):  0
H(n):  Inf
F(n):  Inf

[[2]]
(X Y QuadradosSujos): ( 0 1 1110 )
G(n):  1
H(n):  3
F(n):  Inf

[[3]]
(X Y QuadradosSujos): ( 0 1 1010 )
G(n):  3
H(n):  2
F(n):  Inf

[[4]]
(X Y QuadradosSujos): ( 0 0 1010 )
G(n):  6
H(n):  2
F(n):  Inf

[[5]]
(X Y QuadradosSujos): ( 0 0 10 )
G(n):  8
H(n):  1
F(n):  Inf

[[6]]
(X Y QuadradosSujos): ( 1 0 10 )
G(n):  9
H(n):  1
F(n):  Inf

[[7]]
(X Y QuadradosSujos): ( 1 0 0 )
G(n):  11
```

```
H(n):  0
F(n):  Inf
```

Busca best-first

```
[[1]]
(X Y QuadradosSujos): ( 1 1 1110 )
G(n):  0
H(n):  Inf
F(n):  Inf

[[2]]
(X Y QuadradosSujos): ( 0 1 1110 )
G(n):  1
H(n):  3
F(n):  4

[[3]]
(X Y QuadradosSujos): ( 0 1 1010 )
G(n):  3
H(n):  2
F(n):  5

[[4]]
(X Y QuadradosSujos): ( 0 0 1010 )
G(n):  6
H(n):  2
F(n):  8

[[5]]
(X Y QuadradosSujos): ( 0 0 10 )
G(n):  8
H(n):  1
F(n):  9

[[6]]
(X Y QuadradosSujos): ( 1 0 10 )
G(n):  9
H(n):  1
F(n):  10

[[7]]
(X Y QuadradosSujos): ( 1 0 0 )
G(n):  11
H(n):  0
F(n):  11
```