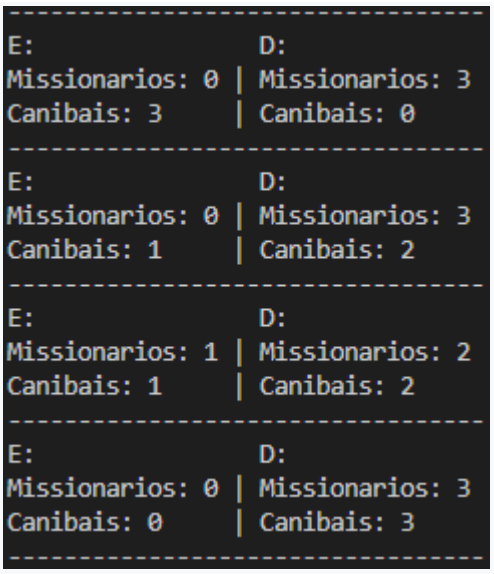
Missionários Canibais:

Consiste em transportar, de uma margem à outra de um rio, um grupo de 3 missionários e 3 canibais, de forma que, em nenhum momento, o número de canibais seja maior que o número de missionários. Existe um único barco que pode ser usado para transportar no máximo dois passageiros.transportar, de uma margem à outra de um rio, um grupo de 3 missionários e 3 canibais, de forma que, em nenhum momento, o número de canibais seja maior que o número de missionários. Existe um único barco que pode ser usado para transportar no máximo dois passageiros.

2)O algoritmo de busca utilizado foi o de busca em largura.

3)




```
class Estado():

    #inicializa as variáveis
    def __init__(self, missionarios_esq, missionarios_dir, canibais_esq, canibais_dir,
lado_rio):

        self.missionarios_esq = missionarios_esq
        self.missionarios_dir = missionarios_dir
        self.canibais_esq = canibais_esq
        self.canibais_dir = canibais_dir
        self.lado_rio = lado_rio
        self.pai = None
        self.filhos = []

    def __str__(self):

        #formatação para a saída
        return 'E:\t        D:\nMissionarios: {}\t| Missionarios: {}\nCanibais: {}\t| Canibais:
{}'.format(
            self.missionarios_esq, self.missionarios_dir, self.canibais_esq, self.canibais_dir
        )

    def estado_valido(self):
        #Verifica se são estados válidos para serem criados
        if ((self.missionarios_esq < 0) or (self.missionarios_dir < 0)
            or (self.canibais_esq < 0) or (self.canibais_dir < 0)):
            return False
        #Verifica se a quantidade de missionários é maior que de canibais
        return ((self.missionarios_esq == 0 or self.missionarios_esq >= self.canibais_esq)
and
```

```
            (self.missionarios_dir == 0 or self.missionarios_dir >= self.canibais_dir))


    def estado_final(self):
        #retorna os estados finais de ambos os lados do rio
        resultado_esq = self.missionarios_esq == self.canibais_esq == 0
        resultado_dir = self.missionarios_dir == self.canibais_dir == 3
        return resultado_esq and resultado_dir

    def gerar_filhos(self):
        novo_lado_rio = 'dir' if self.lado_rio == 'esq' else 'esq'
        movimentos = [
            #todos os movimentos possiveis do transporte
            {'missionarios': 2, 'canibais': 0},
            {'missionarios': 1, 'canibais': 0},
            {'missionarios': 1, 'canibais': 1},
            {'missionarios': 0, 'canibais': 1},
            {'missionarios': 0, 'canibais': 2},
        ]

        for movimento in movimentos:
            if self.lado_rio == 'esq':
                #vai atualizando os valores do esquerdo quando os missionários ou canibais
para o lado direito
                missionarios_esq = self.missionarios_esq - movimento['missionarios']
                missionarios_dir = self.missionarios_dir + movimento['missionarios']
                canibais_esq = self.canibais_esq - movimento['canibais']
                canibais_dir = self.canibais_dir + movimento['canibais']
            else:
                #vai atualizando os valores do direito quando os missionários ou canibais para
o lado esquerdo
                missionarios_dir = self.missionarios_dir - movimento['missionarios']
                missionarios_esq = self.missionarios_esq + movimento['missionarios']
                canibais_dir = self.canibais_dir - movimento['canibais']
                canibais_esq = self.canibais_esq + movimento['canibais']
            #inica os possíveis filhos
            filho = Estado(missionarios_esq, missionarios_dir, canibais_esq,
                           canibais_dir, novo_lado_rio)
            filho.pai = self
            if filho.estado_valido():
                self.filhos.append(filho)

from Estados.Estado import Estado
```

```
class Missionarios_Canibais():


    def __init__(self):
        #inicia os estados iniciais
        self.fila_execucao = [Estado(3, 0, 3, 0, 'esq')]
        self.solucao = None

    def gerar_solucao(self):
        #inicia a BFS

        for elemento in self.fila_execucao:
            if elemento.estado_final():

                self.solucao = [elemento]
                while elemento.pai:
                    self.solucao.insert(0, elemento.pai)
                    elemento = elemento.pai
                break

            elemento.gerar_filhos()
            self.fila_execucao.extend(elemento.filhos)

from bfs.bfs import Missionarios_Canibais


def main():
    #inicia os possíveis caminhos e a BFS
    problema = Missionarios_Canibais()
    problema.gerar_solucao()

    for estado in problema.solucao:
        #apenas formata a resposta
        print (estado)
        print (34 * '-')

if __name__ == '__main__':
    main()
```

Referências:

[Problema Missionários e Canibais com busca em largura]