Tiago Koji Castro Shibata (8988730)

Inteligência Artificial

# Relatório:
# Problema do pastor, lobo, ovelha e repolho

- **Introdução**

Em aula, estudamos métodos de busca cega/não informada. O problema do pastor, lobo, ovelha e repolho foi dado como tarefa. Formulei o enunciado segundo o paradigma de IA, construí o espaço de estados e construí a árvore de busca por um método de busca não informada.

- **Formulação do enunciado**

Nesse problema clássico, um pastor, lobo, ovelha e repolho se encontram em uma margem do rio. Todos devem mover-se para a outra margem utilizando um barco, que é guiado pelo pastor, e aceita um passageiro. Em nenhum momento o pastor deve abandonar uma margem deixando o lobo e a ovelha juntos, senão o lobo comerá a ovelha. O mesmo se aplica à ovelha com o repolho.

Por exemplo, a partir do estado inicial, o único movimento válido é mover o pastor com a ovelha para a outra margem. Em qualquer outro movimento (pastor sozinho, pastor com lobo ou pastor com repolho) o lobo ficará com a ovelha ou a ovelha com o repolho, que é uma situação inválida.

Segundo o paradigma de IA, é importante definir para esse problema:

- Estado: Representei o estado como uma lista de valores booleanos, indicando se dado personagem (pastor, lobo, ovelha, repolho) está na margem desejada.
- Estado inicial: O estado inicial, que também é o único elemento da fronteira inicial, é [False, False, False, False].
- Expansão e função de inserção: Na expansão, o primeiro elemento do vetor (posição do pastor e do bote) é invertido, e são gerados novos estados para cada personagem que estava na mesma margem do pastor e pode ser movido para o outro lado. Estados válidos (sem ataque do lobo ou da ovelha) são inseridos na fronteira, que insere em ordem crescente de custo usando uma heap binária.
- Teste de término: O término ocorre ao encontrar um estado [True, True, True, True].
- Custo: Cada viagem do pastor com o bote adiciona um ao custo do estado.

Implementei uma busca de custo uniforme utilizando uma heap binária para sempre expandir o nó da fronteira com menor custo de caminho. Nesse problema, como o

custo para o próximo estado sempre incrementa de 1, a busca por custo uniforme dá o mesmo resultado que uma busca por largura.

- **Implementação**

Realizei implementação da solução em Python. Na função de inserção, decidi verificar se o elemento já foi encontrado no passado e não o inserir novamente na fronteira, para reduzir um pouco a árvore de busca.

```
import heapq


class State:
    def __init__(self, state, cost, parent):
        self.state = state
        self.cost = cost
        self.parent = parent

    def __lt__(self, other):
        return self.cost < other.cost


def bfs(state, sucessor_f, final_test_f):
    visited_states = {state}
    heap = [State(state, 0, None)]
    while heap:
        next_expansion = heapq.heappop(heap)
        print('State {} with cost {} and parent {}'.format(
            Next_expansion.state,
            next_expansion.cost,
            next_expansion.parent and next_expansion.parent.state))
        if final_test_f(next_expansion.state):
            return next_expansion
        next_states = sucessor_f(next_expansion.state)
        for state in next_states:
            if state not in visited_states:
                heapq.heappush(heap, State(state, next_expansion.cost + 1,
                                           next_expansion))
                visited_states.add(state)


def sucessor_f(state):
    boat_move = (not state[0], *state[1:])  # move boat to opposite margin
    successors = []
    for i, margin in enumerate(state):
        new_state = list(boat_move)
        new_state[i] = not new_state[i]
        new_state[0] = not state[0]
```

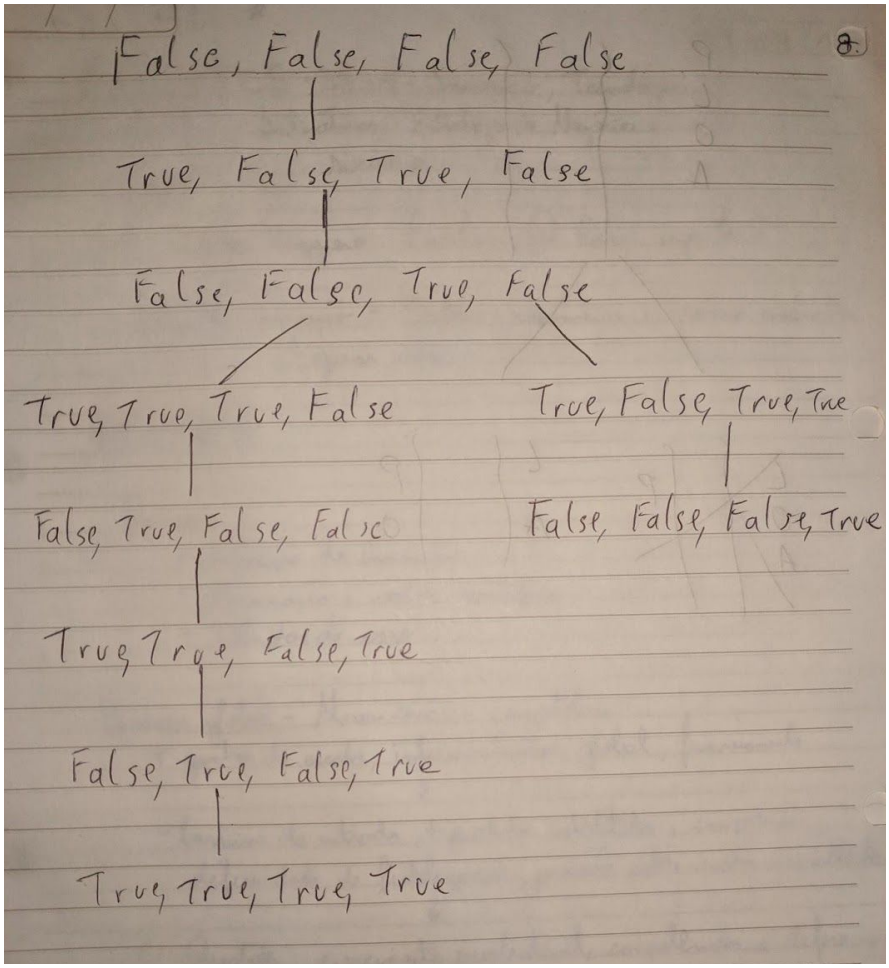
```
        if ((new_state[0] == new_state[1] or new_state[1] != new_state[2]) and
            (new_state[0] == new_state[2] or new_state[2] != new_state[3])):
            successors.append(tuple(new_state))
    return successors

# State: booleans representing whether the boat, wolf, sheep and cabbage are on the
right margin
solution = bfs((False, False, False, False), sucessor_f, all)
print('Solution:')
while solution:
    print(solution.cost, solution.state)
    solution = solution.parent
```

Executar o programa gera:

```
State (False, False, False, False) with cost 0 and parent None
State (True, False, True, False) with cost 1 and parent (False, False, False, False)
State (False, False, True, False) with cost 2 and parent (True, False, True, False)
State (True, True, True, False) with cost 3 and parent (False, False, True, False)
State (True, False, True, True) with cost 3 and parent (False, False, True, False)
State (False, True, False, False) with cost 4 and parent (True, True, True, False)
State (False, False, False, True) with cost 4 and parent (True, False, True, True)
State (True, True, False, True) with cost 5 and parent (False, True, False, False)
State (False, True, False, True) with cost 6 and parent (True, True, False, True)
State (True, True, True, True) with cost 7 and parent (False, True, False, True)
Solution:
7 (True, True, True, True)
6 (False, True, False, True)
5 (True, True, False, True)
4 (False, True, False, False)
3 (True, True, True, False)
2 (False, False, True, False)
1 (True, False, True, False)
0 (False, False, False, False)
```

Vemos que o pastor levou a ovelha, retornou, levou o lobo, retornou com a ovelha, levou o repolho, retorna, e leva o lobo, realizando 7 movimentos.




Pela árvore, vemos também que há outra solução ótima: do estado [False, False, False, True] podemos passar para [True, True, False, True] e também resolver o problema com 7 movimentos. No entanto, como não estamos inserindo elementos duplicados na fronteira, essa expansão não apareceu na árvore.

Se removermos a condição de reinserção de estados já visitados, a árvore de busca é bem maior, já que se retorna muitos estados duplicados:

```
State (False, False, False, False) with cost 0 and parent None
State (True, False, True, False) with cost 1 and parent (False, False, False, False)
State (False, False, True, False) with cost 2 and parent (True, False, True, False)
State (False, False, False, False) with cost 2 and parent (True, False, True, False)
State (True, True, True, False) with cost 3 and parent (False, False, True, False)
State (True, False, True, True) with cost 3 and parent (False, False, True, False)
State (True, False, True, False) with cost 3 and parent (False, False, False, False)
State (True, False, True, False) with cost 3 and parent (False, False, True, False)
State (False, True, False, False) with cost 4 and parent (True, True, True, False)
State (False, False, True, False) with cost 4 and parent (True, False, True, False)
State (False, False, True, False) with cost 4 and parent (True, False, True, False)
```

State (False, False, False, False) with cost 4 and parent (True, False, True, False)
State (False, False, False, False) with cost 4 and parent (True, False, True, False)
State (False, False, False, True) with cost 4 and parent (True, False, True, True)
State (False, False, True, False) with cost 4 and parent (True, False, True, True)
State (False, False, True, False) with cost 4 and parent (True, True, True, False)
State (True, True, False, True) with cost 5 and parent (False, True, False, False)
State (True, False, True, True) with cost 5 and parent (False, False, True, False)
State (True, True, True, False) with cost 5 and parent (False, False, True, False)
State (True, False, True, True) with cost 5 and parent (False, False, True, False)
State (True, False, True, True) with cost 5 and parent (False, False, True, False)
State (True, False, True, False) with cost 5 and parent (False, False, True, False)
State (True, False, True, True) with cost 5 and parent (False, False, True, False)
State (True, True, False, True) with cost 5 and parent (False, False, False, True)
State (True, True, True, False) with cost 5 and parent (False, False, True, False)
State (True, False, True, False) with cost 5 and parent (False, False, True, False)
State (True, False, True, False) with cost 5 and parent (False, False, False, False)
State (True, False, True, True) with cost 5 and parent (False, False, False, True)
State (True, True, True, False) with cost 5 and parent (False, False, True, False)
State (True, False, True, False) with cost 5 and parent (False, False, True, False)
State (True, False, True, False) with cost 5 and parent (False, False, False, False)
State (True, True, True, False) with cost 5 and parent (False, True, False, False)
State (True, True, True, False) with cost 5 and parent (False, False, True, False)
State (True, False, True, False) with cost 5 and parent (False, False, True, False)
State (False, True, False, False) with cost 6 and parent (True, True, False, True)
State (False, True, False, False) with cost 6 and parent (True, True, True, False)
State (False, False, True, False) with cost 6 and parent (True, False, True, True)
State (False, False, False, True) with cost 6 and parent (True, False, True, True)
State (False, True, False, False) with cost 6 and parent (True, True, True, False)
State (False, False, False, False) with cost 6 and parent (True, False, True, False)
State (False, False, True, False) with cost 6 and parent (True, False, True, False)
State (False, False, True, False) with cost 6 and parent (True, False, True, True)
State (False, False, True, False) with cost 6 and parent (True, False, True, False)
State (False, False, True, False) with cost 6 and parent (True, True, True, False)
State (False, False, True, False) with cost 6 and parent (True, False, True, True)
State (False, False, False, False) with cost 6 and parent (True, False, True, False)
State (False, False, False, True) with cost 6 and parent (True, True, False, True)
State (False, True, False, True) with cost 6 and parent (True, True, False, True)
State (False, False, True, False) with cost 6 and parent (True, False, True, True)
State (False, False, False, True) with cost 6 and parent (True, False, True, True)
State (False, False, True, False) with cost 6 and parent (True, False, True, False)
State (False, False, False, True) with cost 6 and parent (True, False, True, True)
State (False, True, False, False) with cost 6 and parent (True, True, False, True)
State (False, False, False, False) with cost 6 and parent (True, False, True, False)
State (False, True, False, False) with cost 6 and parent (True, True, True, False)
State (False, False, False, True) with cost 6 and parent (True, False, True, True)
State (False, False, True, False) with cost 6 and parent (True, True, True, False)
State (False, True, False, False) with cost 6 and parent (True, True, True, False)
State (False, False, False, True) with cost 6 and parent (True, False, True, True)
State (False, False, False, True) with cost 6 and parent (True, True, False, True)
State (False, False, False, False) with cost 6 and parent (True, False, True, False)
State (False, False, False, False) with cost 6 and parent (True, False, True, False)
State (False, True, False, True) with cost 6 and parent (True, True, False, True)
State (False, False, True, False) with cost 6 and parent (True, False, True, False)
State (False, False, True, False) with cost 6 and parent (True, True, True, False)
State (False, False, True, False) with cost 6 and parent (True, False, True, True)
State (False, False, False, False) with cost 6 and parent (True, False, True, False)
State (False, False, True, False) with cost 6 and parent (True, True, True, False)

State (False, False, True, False) with cost 6 and parent (True, False, True, False)
State (False, False, True, False) with cost 6 and parent (True, True, True, False)
State (False, False, True, False) with cost 6 and parent (True, False, True, False)
State (False, True, False, False) with cost 6 and parent (True, True, True, False)
State (True, True, False, True) with cost 7 and parent (False, True, False, False)
State (True, False, True, True) with cost 7 and parent (False, False, True, False)
State (True, False, True, True) with cost 7 and parent (False, False, True, False)
State (True, False, True, True) with cost 7 and parent (False, False, True, False)
State (True, True, True, False) with cost 7 and parent (False, True, False, False)
State (True, False, True, True) with cost 7 and parent (False, False, True, False)
State (True, True, False, True) with cost 7 and parent (False, True, False, False)
State (True, True, True, False) with cost 7 and parent (False, True, False, False)
State (True, True, False, True) with cost 7 and parent (False, True, False, False)
State (True, True, False, True) with cost 7 and parent (False, False, False, True)
State (True, True, True, False) with cost 7 and parent (False, False, True, False)
State (True, False, True, True) with cost 7 and parent (False, False, False, True)
State (True, False, True, False) with cost 7 and parent (False, False, False, False)
State (True, False, True, False) with cost 7 and parent (False, False, True, False)
State (True, True, False, True) with cost 7 and parent (False, False, False, True)
State (True, True, False, True) with cost 7 and parent (False, True, False, False)
State (True, True, True, False) with cost 7 and parent (False, False, True, False)
State (True, False, True, False) with cost 7 and parent (False, False, True, False)
State (True, False, True, True) with cost 7 and parent (False, False, True, False)
State (True, False, True, True) with cost 7 and parent (False, False, True, False)
State (True, True, False, True) with cost 7 and parent (False, True, False, True)
State (True, True, True, True) with cost 7 and parent (False, True, False, True)
Solution:
7 (True, True, True, True)
6 (False, True, False, True)
5 (True, True, False, True)
4 (False, False, False, True)
3 (True, False, True, True)
2 (False, False, True, False)
1 (True, False, True, False)
0 (False, False, False, False)