UNIVERSIDADE FEDERAL FLUMINENSE
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
TCC00297 - INTELIGÊNCIA ARTIFICIAL

Trabalho de Implementação: Problema dos Jarros D'Água

BEATRIZ DE OLIVEIRA PIEDADE
JULIA DOS SANTOS SPEZANI
LEONARDO AMORIM MENDES

NITERÓI
2024

Contents

1 Problema

São dados 2 recipientes, R1 e R2, um com capacidade de 4 litros e outro com capacidade de 3 litros. Os recipientes não têm marcas de medidas. Deve ser colocado exatamente 2 litros no recipiente R1.

1.1 Espaço de Busca

Espaço cartesiano composto pelo conjunto de pares ordenados de inteiros (x,y) tal que x $\in \{0, 1, 2, 3, 4\}$, y $\in \{0, 1, 2, 3\}$

1.2 Estados Finais

Pares ordenados de inteiros (x,y) tal que x $\in \{2\}$, y $\in \{0, 1, 2, 3\}$

2 Solução

Foi utilizado o algoritmo A*, que consiste na busca do caminho mais promissor levando em consideração a avaliação (soma dos litros de água nos jarros) e o custo (número de trocas entre os jarros).

2.1 Fatos

O estado meta da busca:

```
objetivo(2, _).
```

O estado máximo de cada jarro:

```
maximo(jarro1, 4).
maximo(jarro2, 3).
```

2.2 Algoritmo de busca

Primeiro checa se o primeiro elemento do caminho atual é o estado meta. Caso seja, o caminho é invertido para que o movimento mais antigo vire o primeiro da lista e o estado mais recente vire o último.

```
busca([[_, _,  _, [JarroA1, JarroA2]|Estados]|_], Solucao) :-
    objetivo(JarroA1, JarroA2),
    reverse([[JarroA1, JarroA2]|Estados], Solucao).
```

Caso contrário, cria uma lista com todos os vizinhos do estado atual. A lista gerada é concatenada à lista de caminhos visitados e essa junção é ordenada de forma que o caminho mais promissor seja o primeiro elemento.

```
busca([CaminhoAtual|CaminhosVisitados], Solucao) :-
    estende(CaminhoAtual, NovosCaminhos),
    append(CaminhosVisitados, NovosCaminhos, CaminhosPossiveis),
    ordena(CaminhosPossiveis, CaminhosOrdenados),
    busca(CaminhosOrdenados, Solucao).
```

2.3 Busca por vizinhos

A busca por vizinhos amplia a lista de possibilidades a partir do último estado atingido no caminho atual. Ao alterar o estado, calcula-se a nova métrica de decisão.

```
estende([CustoA, _, _, [JarroA1, JarroA2]|Estados], NovosCaminhos) :-
    findall(
        [CustoN, AvaliacaoN, MetricaN, [JarroN1, JarroN2], [JarroA1, JarroA2]
                                                            |Estados],
        (altera([JarroA1, JarroA2], [JarroN1, JarroN2]),
            not(member([JarroN1, JarroN2], [[JarroA1, JarroA2]|Estados])),
            CustoN is CustoA + 1,
            AvaliacaoN is JarroN1 + JarroN2,
            MetricaN is CustoN + AvaliacaoN),
        NovosCaminhos).
```

A nova métrica para decisão *f(n)* é dada pela soma do custo *g(n)* com a avaliação *h(n)*, onde:

$$f(novo) = g(novo) + h(novo)$$
$$g(novo) = g(antigo) + 1$$
$$h(novo) = valorJarro1 + valorJarro2$$

2.4 Estados possíveis

Um dos jarros vazio e o outro mantendo o conteúdo.

```
altera([_, Ja2], [0, Ja2]).
altera([Ja1, _], [Ja1, 0]).
```

Um dos jarros cheio ao máximo e o outro mantendo o conteúdo.

```
altera([_, Ja2], [Jn1, Ja2]) :- maximo(jarro1, Jn1).
altera([Ja1, _], [Ja1, Jn2]) :- maximo(jarro2, Jn2).
```

Conteúdo de um transferido para o outro.

```
capacidade(Jarro, Valor, Dif) :-
    maximo(Jarro, Max),
    Dif is Max - Valor.

altera([Ja1, Ja2], [Jn1, Jn2]) :-
    capacidade(jarro2, Ja2, Capacidade),
    Ja1 >= Capacidade,
    Jn1 is Ja1 - Capacidade,
    Jn2 is Capacidade + Ja2. %valorMaximo
altera([Ja1, Ja2], [0, Jn2]) :-
    capacidade(jarro2, Ja2, Capacidade),
```

```
    Ja1 < Capacidade,
    Jn2 is Ja1 + Ja2.

altera([Ja1, Ja2], [Jn1, Jn2]) :-
    capacidade(jarro1, Ja1, Capacidade),
    Ja2 >= Capacidade,
    Jn1 is Capacidade + Ja1, %valorMaximo
    Jn2 is Ja2 - Capacidade.
altera([Ja1, Ja2], [Jn1, 0]) :-
    capacidade(jarro1, Ja1, Capacidade),
    Ja2 < Capacidade,
    Jn1 is Ja1 + Ja2.
```

2.5 Ordenação

Foi utilizado o algoritmo quicksort para ordenar a lista de caminhos:

```
%PARTICIONAMENTO
divide(_, [], [], []).

divide([CP, EP, Pivo|EstadosP], [[CS, ES, Soma|EstadosS]|Cauda],
                                [[CS, ES, Soma|EstadosS]|Menores], Maiores) :-
    Soma < Pivo,
    divide([CP, EP, Pivo|EstadosP], Cauda, Menores, Maiores),
    !.

divide([CP, EP, Pivo|EstadosP], [[CS, ES, Soma|EstadosS]|Cauda],
                                Menores, [[CS, ES, Soma|EstadosS]|Maiores]) :-
    Soma >= Pivo,
    divide([CP, EP, Pivo|EstadosP], Cauda, Menores, Maiores).

%ORDENACAO
ordena([], []).

ordena([Pivo|Cauda], Final) :-
    divide(Pivo, Cauda, Menores, Maiores),
    ordena(Menores, FinalMenores),
    ordena(Maiores, FinalMaiores),
    append(FinalMenores, [Pivo|FinalMaiores], Final).
```

2.6 Soluções

Para cada estado inicial em *busca([[0, 0, 0, [x, y]]], Solucao).*, a solução é:

(0, 0): [[0, 0], [0, 3], [3, 0], [3, 3], [4, 2], [0, 2], [2, 0]]

(2, 0): [[2, 0]]

(0, 2): [[0, 2], [2, 0]]

(3, 0): [[3, 0], [3, 3], [4, 2], [0, 2], [2, 0]]

(0, 1): [[0, 1], [4, 1], [2, 3]]