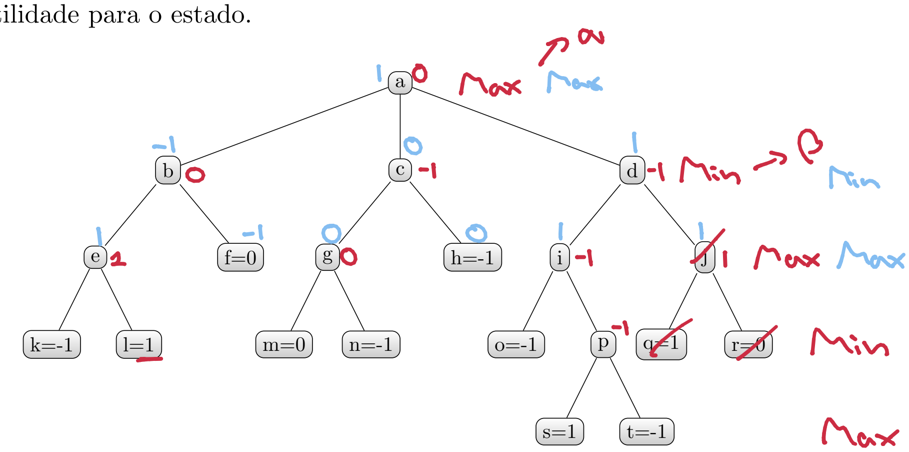
1^{o}Teste de Inteligência Artificial
16/5/2022 - 10-12 horas

Grupo 1 Considere a árvore da figura acima, que representa o espaço de estados de um jogo de dois jogadores, o valor nas folhas indica o valor da função de utilidade para o estado.



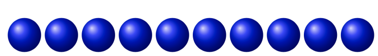
1. Indique o valor dos nós não terminais da árvore (a, b, c, d, e, g, i, j e p) de acordo com o algoritmo minmax e a jogada perfeita no estado 'a'.

2. Indique os nós que não precisam de ser avaliados com o corte $\alpha - \beta$.

3. Indique a jogada perfeita com o minimax e um cuttof= 2 para a seguinte função de avaliação:

a = 1	b = -1	c =0	d=1	e=1	f=-1	g=0
h=0	i=1	j=1	k=0	l=1	m=1	n=-1
o=-1	p=0	q=1	r=0	s=1	t=-1	

4. Considere o seguinte Jogo:



Há 10 peças em linha. Cada jogador pode retirar uma ou duas peças da esquerda para a direita. O jogador que retirar a última peça perde.

(a) Represente o espaço de estados para este jogo e defina os operadores de transição de estados em Prolog.

(b) Com a sua definição de estado e operadores qual é profundidade máxima da árvore minmax na primeira jogada (com 10 peças)?

(c) Com a sua definição de estado e operadores quantas jogadas pode fazer um jogador após quatro jogadas (duas para cada jogador)?

Grupo I

1. a = 0   g = 0

b = 0   i = -1

c = -1   j = 1

d = -1   p = -1

e = 1

A jogada perfeita no estado 'a' é ir para 'b'.

2. $d = -1 \rightarrow \alpha = 0$ e $\beta = -1$

Então para $d=-1$ temos $\alpha = 0$ e $\beta = -1$, $\alpha \geq \beta$, logo neste situação podemos cortar o ramo a partir de d.

Ou seja, cortamos j, q, r.

3. A jogada perfeita para a função de avaliação dada é ir para 'd'.

4. a) %estado(NdePecasRestantes)

```
estado_inicial(10)
estado_final(0)
%op( estado_atual, operador, estado_seguinte, custo)
op( e(N), tira(1), e(N1), 1 ):- N1 is N-1,
                                N>=1.
op( e(N), tira(2), e(N1), 1) :- N1 is N-2,
                                N>=2.
```

b) A profundidade máxima é dada quando os jogadores tiram sempre apenas 1 peça.

c) Após 4 jogadas:

Peças retiradas entre 4 e 8

Logo, peças restantes entre 2 e 6

Sendo o número de peças restantes ≥ 2, então existirão garantidamente 2 jogadas possíveis.

d) % Estado([ListaPeçasRestantes])

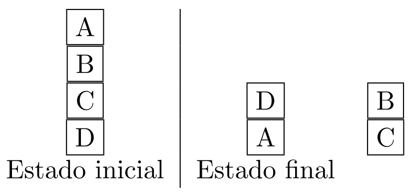
```
Estado_Inicial([ p1, p2, p3, p4, p5, p6, p7, p8, p9, p10 ]).
Estado_final([ ])
```

```
op(e([X|L]), tira(1, X), e(L), 1).   Caso base (1 peça)
op(e([Y|L]), tira(1,X), e([Y|L1]), 1) :- Y \= X,
                                        op(e(L), tira(1,X), e(L1), 1).
op(e([X,_|L]), tira(2,X), e(L), 1).  Caso base (2 peça)
op(e([Y|L]), tira(2,X), e([Y,L1]), 1) :- (Y \= X; (L=[]; L=[H|_],
                                                       H \= Y)),
                                        op(e(L), tira(2,X), e(L1), 1)
```

(d) Se se alterassem as regras do jogo para: em cada jogada, um jogador pode ou retirar uma peça qualquer ou retirar duas peças quaisquer se estiverem juntas, perde o jogador que tirar a última peça.

Represente, em prolog, o espaço de estados para o jogo com as novas regras. E indique quantas Jogadas diferentes pode um jogador fazer no estado inicial.

Grupo 2 Considere o problema dos blocos (A,B,C e D). Os blocos podem ser empilhados e desempilhados um a um. Considere que tem um robot com duas mãos que pode segurar um bloco em cada mão.



1. Indique o vocabulário (condições, fluentes e ações) que considera adequado para resolver este problema usando um planeador.

2. Descreva este problema na notação STRIPS, para cada ação indique: a lista de pré condições, a addList e a DelList.

3. Represente o estado inicial e o estado final deste problema com o seu vocabulário.

4. Indique uma sequência de ações para ir do estado inicial ao final.

5. Como é que um pop (planeador de ordem parcial) resolveria o problema da alínea anterior:

   (a) Indique os passos do plano para ir do estado inicial ao estado final: para cada passo indique o identificador e o nome da ação com as variáveis instanciadas.

   (b) Indique 5 links do plano: os identificadores dos passos e a condição.

   (c) Exemplifique duas ameaças diferentes: em cada exemplo, indique um passo e o link que pode ser ameaçado pelo passo.

Grupo II

1. Vocabulario:

~~Condições~~ Fluentes

livre(M) → O robô tem a mão M livre.

porCima(X, Y) → O bloco X está por cima do bloco Y.

mao(B, M) → O bloco B está na mão M.

possivel(B,P) → Podemos colocar um bloco sobre o bloco B na pilha P.

pilha(B, P) → Bloco B está na pilha P.

Ações

pegar(B, P, M) → pega o bloco B com a mão M da pilha P

colocar(B, P, M) → coloca o bloco B na pilha P com a mão M

2. pegar(B, P, M):

Pré-condições: possivel(B,P), livre(M), porCima(B, B1), pilha(B, P)

Add-List: mao(B, M), possivel(B1, P)

Del-List: possivel(B,P), livre(M), pilha(B, P), porCima(B, B1)

colocar(B, P, M):

Pré-condições: mao(B,M), possivel(B, P)

Add-List: porCima(B,B1), livre(M), possivel(B,P)

Del-List: mao(B, M), possivel(B1, P)

```
% acao(Nome, Precondicoes, ADDList, DELList)
    acao(pegar(B,P,M), [livre(M), possivel(B,P), porCima(B,B1), pilha(B,P)],
        [mao(B,M), possivel(B1,P)], [possivel(B,P), livre(M), pilha(B,P), porCima(B,B1)]).
    acao(colocar(B,P,M), [mao(B,M), possivel(B,P)], [porCima(B,B1), livre(M), possivel(B,P)]
        , [mao(B,M), possivel(B1,P)]).
```

3. estado_inicial([livre(E), livre(D), porCima(C, D), porCima(B, C), porCima(A, B), possivel(A, 1),
pilha(A, 1), pilha(B, 1), pilha(C, 1), pilha(D, 2)]).

estado_final([livre(E), livre(D), porCima(D, A), pilha(A, 1), pilha(D, 1), possivel(D, 1), porCima(B, C),
pilha(B, 2), pilha(C, 2), possivel(B, 2)]).

4. passos([pegar(^{2}A, 1, E), pegar(^{2}B, 1, D), colocar(^{3}B, 2, D), pegar(^{4}C, 1, D), colocar(^{5}C, 2, D),
pegar(^{6}D, 1, D), colocar(^{7}A, 1, E), colocar(^{8}D, 1, D), pegar(^{9}C, 2, E), pegar(^{10}B, 2, D),
colocar(^{11}C, 2, E), colocar(^{12}B, 2, D)]).

5. a) s0 → estado_inicial

s1 → estado_final

P1 = colocar(B, 2, M)

P2 = colocar(C, 2, M)

P3 = colocar(A, 1, M)

P4 = colocar(D, 1, M) P2 < P1 P3 < P4 P4 < P2

Para P1:

P5 = pegar(B, 1, M) P5 < P1

Para P5:

P6 = pegar(A, 1, M) P6 < P5

Para P2:

P7 = colocar(B, 2, M) P7 < P8

P8 = pegar(C, 1, M) P8 < P2

Para P4:

P9 = pegar(D, 2, M) P9 < P4

Para P9:

P10 = colocar(C, 2, M) P10 < P9