Grupo 1.

> Considere o problema de num tabuleiro de xadrez 8X8, ir da casa (1,1) até à casa (4,5) com um cavalo (move-se em L, 2 casas numa direcção mais uma casa na perpendicular a essa direcção).

1. Represente este problema como um problema de pesquisa no espaço de estados em Prolog (estado inicial e final, e os operadores de transição de estado).

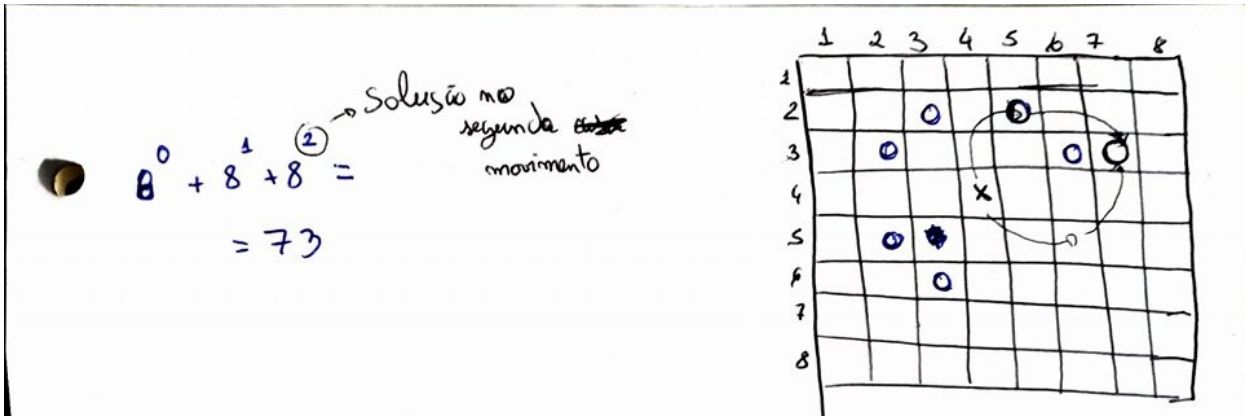
```
estado_inicial(e(tab([(1,1), (1,2), ..., (8,8)]),
                cavalo(1,1) )).
estado_final(e(tab(...),
                cavalo(4,5) )).

tr
(
    e(Tab, cavalo(X,Y)),
    cima_direita,
    e(Tab, cavalo(X1,Y1))
):-
    member((X,Y), Tab),
    X1 is  X  - 2, Y1 is Y + 1,
    member((X1,Y1), Tab).
tr
(
    e(Tab, cavalo(X,Y)),
    direita_cima,
    e(Tab, cavalo(X1,Y1))
):-
    member((X,Y), Tab),
    X1 is  X  - 1, Y1 is Y + 2,
    member((X1,Y1), Tab).
tr
(
    e(Tab, cavalo(X,Y)),
    cima_esquerda,
    e(Tab, cavalo(X1,Y1))
):-
    member((X,Y), Tab),
    X1 is  X  - 2, Y1 is Y - 1,
    member((X1,Y1), Tab).
tr
(
    e(Tab, cavalo(X,Y)),
    esquerda_cima,
    e(Tab, cavalo(X1,Y1))
):-
    member((X,Y), Tab),
    X1 is  X  - 1, Y1 is Y - 2,
    member((X1,Y1), Tab).
```

```
tr
(
    e(Tab, cavalo(X,Y)),
    esquerda_baixo,
    e(Tab, cavalo(X1,Y1))
):-
    member((X,Y), Tab),
    X1 is  X  + 1, Y1 is Y - 2,
    member((X1,Y1), Tab).
tr
(
    e(Tab, cavalo(X,Y)),
    baixo_esquerda,
    e(Tab, cavalo(X1,Y1))
):-
    member((X,Y), Tab),
    X1 is  X  + 2, Y1 is Y - 1,
    member((X1,Y1), Tab).
tr
(
    e(Tab, cavalo(X,Y)),
    baixo_direita,
    e(Tab, cavalo(X1,Y1))
):-
    member((X,Y), Tab),
    X1 is  X  + 2, Y1 is Y + 1,
    member((X1,Y1), Tab).
tr
(
    e(Tab, cavalo(X,Y)),
    direita_baixo,
    e(Tab, cavalo(X1,Y1))
):-
    member((X,Y), Tab),
    X1 is  X  + 1, Y1 is Y + 2,
    member((X1,Y1), Tab).
```

---

2. Numa pesquisa em largura quantos estados são expandidos se o cavalo estiver na casa (4,4), e o quisermos colocar na casa (7,3).

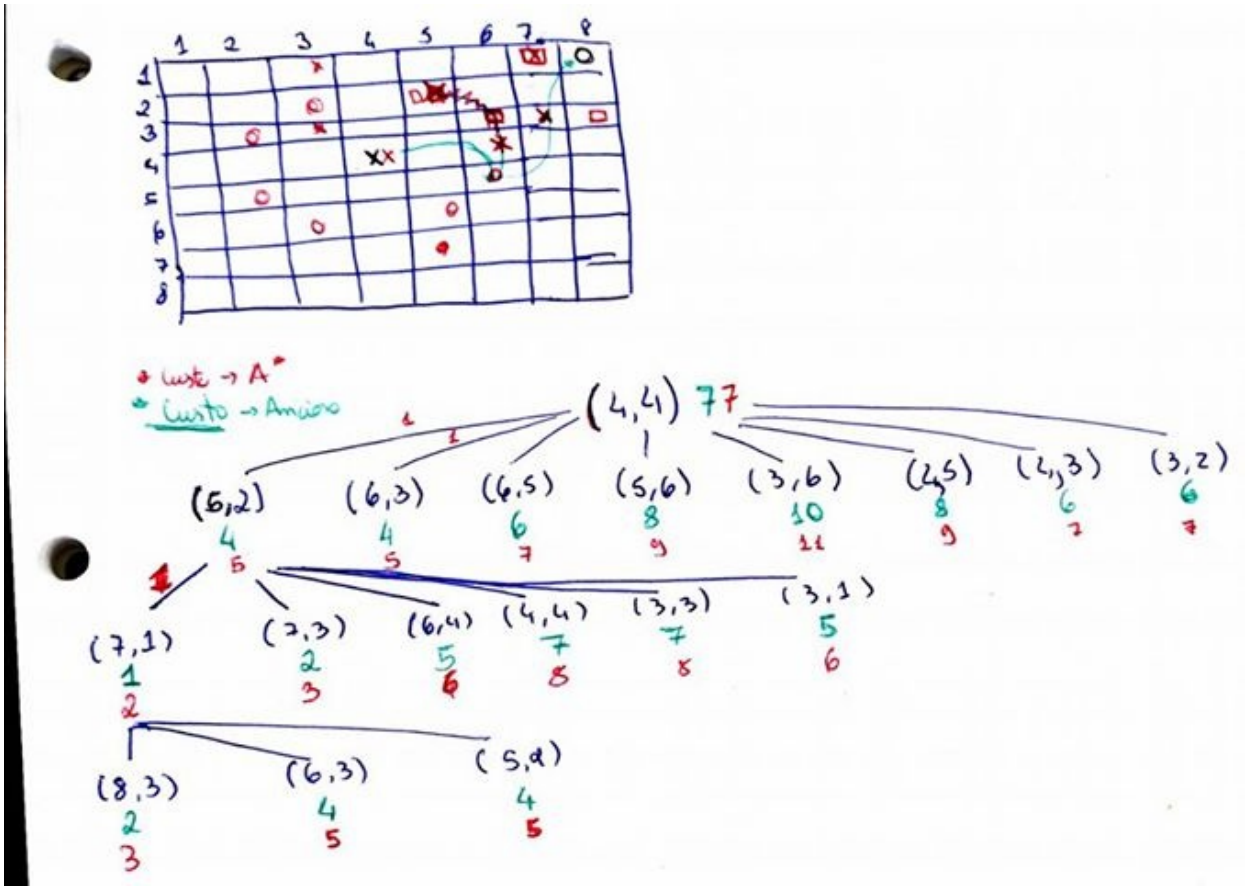


3. Considere a seguinte função de heurística que estima a distância entre duas casas:

```
f((X,Y),(W,Z),N):-  modulo_dif(X,W),M),
                    modulo_dif(Y,Z),O),
                    N is M+O.

modulo_dif(A,B,C):-  A>B, C is A-B.
modulo_dif(A,B,C):-  C is B-A
```

a) Desenhe a´árvore de pesquisa do espaço de estados deste problema (quando o estado inicial´e a casa (4,4) e o estado final´é a casa (8,1) indicando a ordem de visita dos nós com o algoritmo A* e com o Ansioso, usando esta função para estimar a distância entre estados. (expanda a´árvore só até ao nível 3. (Justifique a ordem de visita anotando nos nós expandidos o custo estimado).



b) Para este problema esta heurística´e admissível? Justifique.

Não é admissivel, apesar da heuristica ter um valor pequeno (como por exemplo 1), o cavalo não consegue movimentar apenas uma casa para o lado. Assim sendo para o cavalo ir para o sitio certo tem de ter uma heuristica maior e não a mais pequena.

Grupo 2.

> Considere o problema de colocar 4 cavalos num tabuleiro de 4x4
> sem se atacarem.

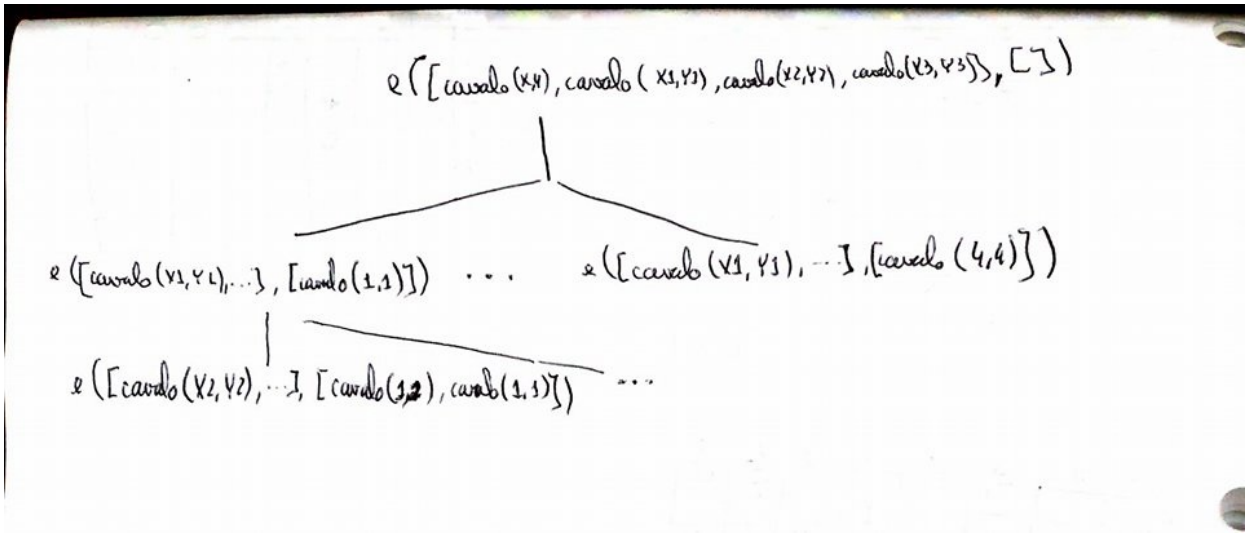
> Este problema pode ser representado e resolvido como um problema de satisfção de restrições(CSP).

1. Proponha uma uma representa¸c˜ao em Prolog para os estados do problema. Exemplifique representando o estado inicial deste problema.

```
estado_inicial(
    e(
    [cavalo(_), cavalo(_), cavalo(_), cavalo(_)], % NonAffect
    [] % Affect
)).
```

2. Defina o predicado sucessor(Ei,Es) que sucede para todo Ei, Es em que Es´é um sucessor de Ei. E expanda a árvore do espaço de estados até ao nível dois.

```
sucessor(e([cavalo(X,Y)|NonAffect], Affect),
         e(NonAffect, [cavalo(X,Y)|Affect])):-
             member((X,Y),Tabuleiro), % Tabuleiro - estado com as posições válidas
             delete(Tabuleiro, (X,Y), Tab),
             Tabuleiro is Tab.
```



3. Represente em Prolog as restrições deste problema. E defina o predicado verifica

restrições (Estado) que sucede quando um estado verifica todas as restrições.

```
ve_restricoes(e(_, Affect)):-
    ve_cima_direita(Affect),
    ve_cima_esquerda(Affect),
    ve_baixo_direita(Affect),
    ve_baixo_esquerda(Affect),
    ve_esquerda_cima(Affect),
    ve_esquerda_baixo(Affect),
    ve_direita_cima(Affect),
    ve_direita_baixo(Affect).

ve_cima_direita([cavalo(X,Y)|Affect]):-
    X1 is X - 2,
    Y1 is Y + 1,
    X1 >= 1, Y1 =< 4,
    ! member((X1,Y1), Tabuleiro).

% Repetir o processo para os outros ...
```