UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Lucas Machado da Palma 12200640
Luiz Henrique Urias 12200647

Prolog – Trabalho 3A

– Arquivo PL –

```
matrix([[255,10,30],
            [10,20,40]]).

matriz([[1,0,0],
        [0,0,1]]).
coordLine([], _, _, []).

coordLine([H|T], Lin, Col, [(Lin,Col,H)|Tm]) :-
    Col1 is Col + 1,
    coordLine(T, Lin, Col1, Tm).

coordAux([], _, _, []) :- !.

coordAux([H|T], Lin, Col, [Hm|Tm]) :-
    Lin1 is Lin + 1,
    coordLine(H, Lin1, Col, Hm),
    coordAux(T, Lin1, Col, Tm).

coord2coord([], C, C).

coord2coord([H|T], C, Coord) :-
    append(C,H,Cx),
    coord2coord(T, Cx, Coord).

coord(Mat, Coord) :-
    coordAux(Mat, -1, 0, CoordMat),
    coord2coord(CoordMat, [], Coord).

zerosAuxLine((X,_), S, S) :-
    X < 0,
    !.
zerosAuxLine((_,X), S, S) :-
    X =< 0,
    !.
zerosAuxLine((H,W), Sa, S) :-
    Wa is W - 1,
    zerosAuxLine((H,Wa), [(H,Wa,0)|Sa], S).

zerosAuxSet((X,_), S, S) :-
    X < 0,
    !.
zerosAuxSet((_,X), S, S) :-
    X < 0,
    !.
zerosAuxSet((H,W), Sa, S) :-
    Ha is H - 1,
    zerosAuxLine((Ha,W), [], Sb),
    append(Sb, Sa, Sc),
    zerosAuxSet((Ha,W), Sc, S).

zeros((H,W), S) :-
    zerosAuxSet((H,W), [], S).

putPixel(_, [], []) :-
    !.
```

```
putPixel((A,B,V), [(A,B,_)|T1], [(A,B,V)|T2]) :-
    putPixel((A,B,V), T1, T2),
    !.
putPixel((A,B,V), [(Ax,Bx,Vx)|T1], [(Ax,Bx,Vx)|T2]) :-
    Ax \= A,
    putPixel((A,B,V), T1, T2).
putPixel((A,B,V), [(Ax,Bx,Vx)|T1], [(Ax,Bx,Vx)|T2]) :-
    Bx \= B,
    putPixel((A,B,V), T1, T2).

% NEIGHBORHOOD
% ------------------------

above(S, (X,Y,_), (Xa,Y,V)) :-
    X > 0,
    Xa is X - 1,
    getPixel(S, (Xa,Y,V)).

below(S, (X,Y,_), (Xa,Y,V)) :-
    Xa is X + 1,
    getPixel(S, (Xa,Y,V)).

left(S, (X,Y,_), (X,Ya,V)) :-
    Y > 0,
    Ya is Y - 1,
    getPixel(S, (X,Ya,V)).

right(S, (X,Y,_), (X,Ya,V)) :-
    Ya is Y + 1,
    getPixel(S, (X,Ya,V)).

neighbor(S, (X,Y,V), E) :-
    above(S, (X,Y,V), E).
neighbor(S, (X,Y,V), E) :-
    below(S, (X,Y,V), E).
neighbor(S, (X,Y,V), E) :-
    left(S, (X,Y,V), E).
neighbor(S, (X,Y,V), E) :-
    right(S, (X,Y,V), E).

n4(S, (X,Y,V), N) :-
    findall(E, neighbor(S, (X,Y,V), E), N).
```

% Problema 1

Limiarização (thresholding) : dado um valor T como argumento, para cada intensidade I < T na imagem de entrada, o pixel correspondente na imagem resultante se torna zero; para I >= T, a saída se torna um (produz -se uma imagem binária).

```
retorna([(X,Y,I)|C], X, Y, I, C).
zerar(Matriz, T, Saida):- coord(Matriz, Lista), limi(Lista,T,Saida).
limi([],_, []).
limi(Lista,T, [(X,Y,Is)|N]):-retorna(Lista, X, Y, I, C), ((I<T, Is is 0);(I>=T, Is is 1)),
limi(C, T, N).
```

```
?- matrix(M), zerar(M, 60, Saida).
M = [[255, 10, 30], [10, 20, 40]],
Saida = [ (0, 0, 1), (0, 1, 0), (0, 2, 0), (1, 0, 0), (1, 1, 0), (1, 2, 0)] ;
false.

?-
```

% Problema 2

Negativa: para cada intensidade I na imagem de entrada, produz-se 255 – I na imagem de saída; se a entrada for binária, a subtração passa a ser 1 – I.

```
if(X, Z):- ((X==1; X==0) -> Z is 1; Z is 0).
negAux1([],[]).
negAux2([],[]).
verificaB([],_).
verificaB(Lista, Z):- retorna(Lista, X, Y, I, C), if(I,Z), verificaB(C,_).

neg(Matriz, Resultado, Verdade):-coord(Matriz, Lista), verificaB(Lista, Verdade), ((Verdade==1,
negAux2(Lista, Resultado));(Verdade==0, negAux1(Lista, Resultado))).

negAux2(Lista, [(X,Y,Is)|N]):- retorna(Lista, X, Y, I, C), Is is (1 - I), negAux2(C, N).
negAux1(Lista, [(X,Y,Is)|N]):- retorna(Lista, X, Y, I, C), Is is (255-I), negAux1(C, N).
```

```
?- matrix(M), neg(M, Resultado, Verdade).
M = [[255, 10, 30], [10, 20, 40]],
Resultado = [ (0, 0, 0), (0, 1, 245), (0, 2, 225), (1, 0, 245), (1, 1, 235), (1, 2, 215)],
Verdade = 0 ;
false.

?- matriz(M), neg(M, Resultado, Verdade).
M = [[1, 0, 0], [0, 0, 1]],
Resultado = [ (0, 0, 0), (0, 1, 1), (0, 2, 1), (1, 0, 1), (1, 1, 1), (1, 2, 0)],
Verdade = 1 ;
false.

?-
```

% Problema 3

Soma de constante: dado um valor K, para cada intensidade I na imagem de entrada, produz-se I + K na imagem resultante; no entanto, se (I + K) > 255, o valor de soma deve se tornar 255; se K < 0 e (I + K) < 0, então o valor da soma deve se tornar 0.

```
% Problema 3
somaAux([],_,[]).
soma(Matriz, K, Resultado):- coord(Matriz, Lista), somaAux(Lista, K, Resultado).
verificador(X,Z):-(X>255 -> Z is 255; Z is X).
```

```
verificador2(X,Z):-(X<0 -> Z is 0; Z is X).
% somaAux(Lista, K, [(X,Y,A)|N]):- retorna(Lista, X, Y, I, C), Is is (I + K), verificador(Is, Z),verificador2(Is, W), (A is Z; A is W),somaAux(C, K,  N).
somaAux(Lista, K, [(X, Y, Z)|N]):- retorna(Lista, X, Y, I, C), Is is (I + K), ((Is>0, verificador(Is, Z));(Is=<0, verificador2(Is, Z))), somaAux(C, K, N).
```

```
?- matrix(M), soma(M, 50, Resultado).
M = [[255, 10, 30], [10, 20, 40]],
Resultado = [ (0, 0, 255), (0, 1, 60), (0, 2, 80), (1, 0, 60), (1, 1, 70), (1, 2, 90)] ;
false.

?- matrix(M), soma(M, -20, Resultado).
M = [[255, 10, 30], [10, 20, 40]],
Resultado = [ (0, 0, 235), (0, 1, 0), (0, 2, 10), (1, 0, 0), (1, 1, 0), (1, 2, 20)] ;
false.

?- matrix(M), soma(M, -260, Resultado).
M = [[255, 10, 30], [10, 20, 40]],
Resultado = [ (0, 0, 0), (0, 1, 0), (0, 2, 0), (1, 0, 0), (1, 1, 0), (1, 2, 0)] ;
false.

?-
```

% Problema 4

Cada pixel da imagem resultante é obtido pela soma dos pixel correspondentes de duas imagens de entrada com as mesmas dimensões (observar a saturação em 255).

```
mm(M1, M2, Resultado):- coord(M1, L1), coord(M2, L2), mmAux(L1, L2, Resultado).
mmAux([],[],[]).
mmAux(L1, L2, [(X, Y, Z)|N]):- retorna(L1, X, Y, I1, C1), retorna(L2, X, Y, I2, C2), Is is (I1 + I2), verificador(Is, Z), mmAux(C1, C2, N).
```

```
?- matriz(M1), matrix(M2), mm(M1, M2, Resultado).
M1 = [[1, 0, 0], [0, 0, 1]],
M2 = [[255, 10, 30], [10, 20, 40]],
Resultado = [ (0, 0, 255), (0, 1, 10), (0, 2, 30), (1, 0, 10), (1, 1, 20), (1, 2, 41)] ;
false.

?-
```

% **Problemas 5 e 6**

Infelizmente não conseguimos utilizar a implementação de vizinhos disponibilizada pelo professor. Segue o código da nossa tentativa de cumprir com o objetivo de questão de numero 5.

```
% Problema 5
% retornaPixel([H|C], H, C).
% retornaTripla((X,Y,I), X,Y,I).
% isolado(Matriz, Isolados):-coord(Matriz, Lista), isoladoAux(Lista).
% verificaI(Pixel, Vizinho):- retornaTripla(Pixel, _,_,I), retornaTripla(Vizinho,_,_,P), I<P.
% isoladoAux([],[],_).
% isoladoAux(Lista, [(X,Y,Z)|N], Pixel, Abaixo):- retorna(Lista,X,Y,Z,C), retornaPixel(Lista,Pixel,_), retornaTripla(Pixel, P1,P2,P3), below(Lista,Pixel,Abaixo), verificaI(Pixel,Abaixo), isoladoAux(C,N,_,_).
```

Para o problema 5, não foi possível testar todos os pixel. Foi testado apenas para o vizinho abaixo, mas isso se aplica aos demais. Uma vez que estamos usando recursão para resolver, quando o pixel escolhido for o da última linha por exemplo, ele não terá vizinho abaixo, logo, o resultado será falso, e isso acaba gerando false para toda a regra. Ou seja, gerou um false no decorrer da regra, retornando false e terminando.

% **Problema 7**

Uma vez que se tem duas matrizes M1 e M2, deseja-se ter como resultado uma lista de pixel contendo a diferença entre as intensidades de cada pixel.

```
subImg(M1, M2, Resultado):- coord(M1, L1), coord(M2, L2), mmAux2(L1, L2, Resultado).
mmAux2([],[],[]).
mmAux2(L1, L2, [(X, Y, Z)|N]):- retorna(L1, X, Y, I1, C1), retorna(L2, X, Y, I2, C2), Is is (I1 - I2),
verificador2(Is, Z), mmAux2(C1, C2, N).
```

```
?- matriz(M1), matrix(M2), subImg(M1, M2, Resultado).
M1 = [[1, 0, 0], [0, 0, 1]],
M2 = [[255, 10, 30], [10, 20, 40]],
Resultado = [ (0, 0, 0), (0, 1, 0), (0, 2, 0), (1, 0, 0), (1, 1, 0), (1, 2, 0)] ;
false.

?- matriz(M1), matrix(M2), subImg(M2, M1, Resultado).
M1 = [[1, 0, 0], [0, 0, 1]],
M2 = [[255, 10, 30], [10, 20, 40]],
Resultado = [ (0, 0, 254), (0, 1, 10), (0, 2, 30), (1, 0, 10), (1, 1, 20), (1, 2, 39)] ;
false.

?-
```

% **Problema 8**

Dada duas matrizes de pixel M1 e M2, representando 2 imagens ou instantes diferentes de uma gravação, deseja-se calcular a soma entre as diferenças de cada um dos pixels (intensidade) na posição ij das matrizes. De forma simplificada representada pela formula Σ (M1ij – M2ij).