

# Filtros de Imagens

Uma imagem digital é a projeção de uma matriz, que representa a imagem, em uma tela. Cada posição  $(x,y)$  da matriz, onde  $x$  é o número da linha e  $y$  é o número da coluna, representa um pixel (*picture element*)  $p(x,y)$ . O valor de  $p(x,y)$  indica a cor que deve ser projetada na posição correspondente da imagem.

Nas imagens em tons de cinza PGM (Portable Graymap Format), esse valor descreve a intensidade do ponto projetado que varia de 0 (preto) a 255 (branco). As coordenadas da matriz seguem a ordem *raster* (de cima para baixo, da esquerda para direita). Abaixo temos um exemplo do conteúdo de um arquivo PGM.

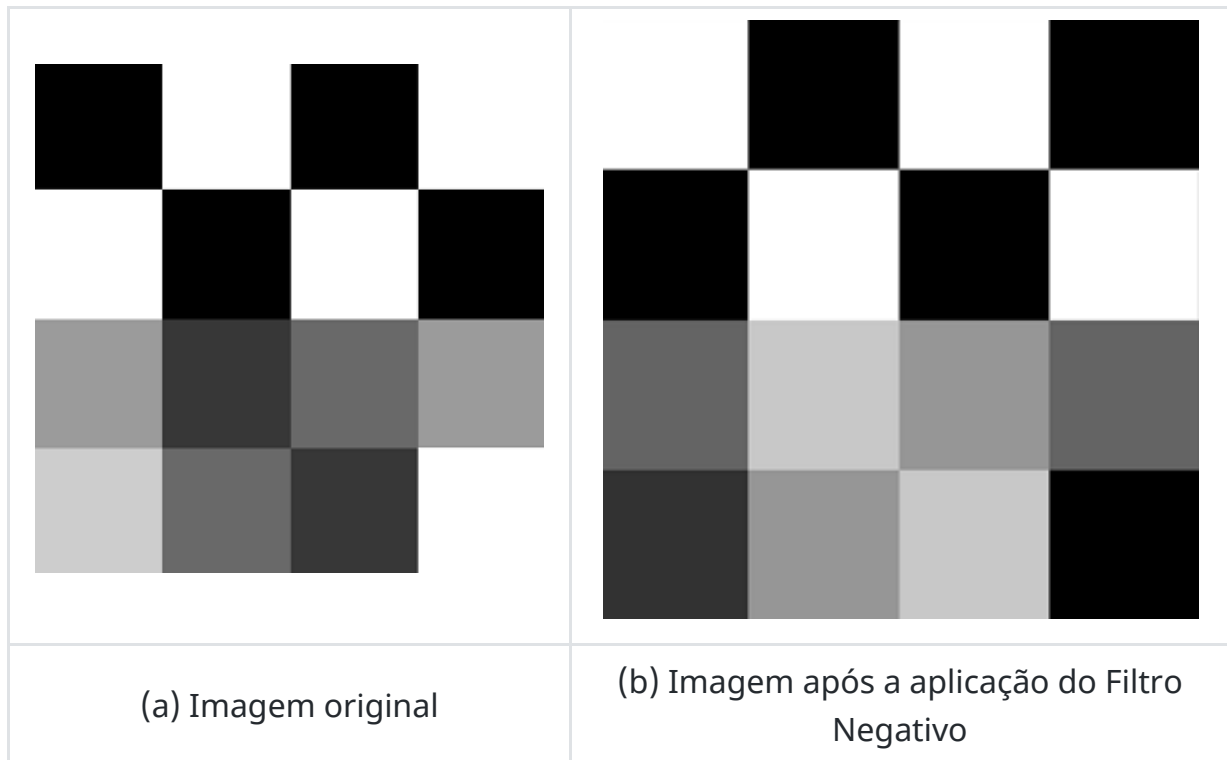
```
P2
24 7
255
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 51 51 51 51 0 0 51 51 51 51 0 0 119 119 119 119 0 0 119 119 119 119 0
0 51 0 0 0 0 0 51 0 0 0 0 0 119 0 0 0 0 0 119 0 0 119 0
0 51 51 51 0 0 0 51 51 51 0 0 0 119 119 119 0 0 0 119 119 119 119 0
0 51 0 0 0 0 0 51 0 0 0 0 0 119 0 0 0 0 0 119 0 0 0 0
0 51 0 0 0 0 0 51 51 51 51 0 0 119 119 119 119 0 0 119 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

O cabeçalho do arquivo é formado pelas três primeiras linhas, sendo que a primeira linha possui o valor `P2`, o qual indica o formato do arquivo; a segunda linha possui dois valores inteiros  $m$  e  $n$  que representam o número de colunas e linhas, respectivamente; e a terceira linha possui o valor máximo que um pixel pode assumir, no nosso caso esse valor sempre será `255`. Depois do cabeçalho, o arquivo possui  $n$  linhas, onde cada linha possui  $m$  valores inteiros separados por um espaço em branco. Cada um desses valores representa um pixel da imagem em escala de cinza. Uma imagem no formato PGM pode ser visualizada utilizando este [site](https://susy.ic.unicamp.br:9999/mc102/12/enunciado.html).

Programas de edição e processamento de imagens possuem diversos filtros que são utilizados para manipulação de imagens, como redução de ruído, detecção de arestas, nitidez, etc. Neste laboratório você deve implementar um programa que realiza a aplicação dos seguintes filtros: filtro negativo, filtro da mediana, e filtros implementados por convolução. Cada um desses filtros é descrito a seguir.

## Filtro Negativo

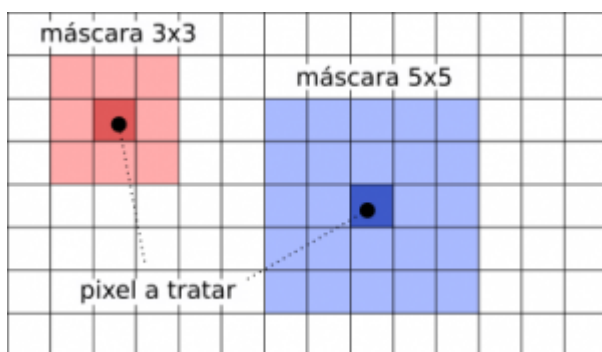
O filtro negativo consiste em converter cada pixel em seu complemento, ou seja, cada valor  $p(x,y)$  deve ser substituído por  $255-p(x,y)$ .



**Figura 1** Exemplo da aplicação do filtro negativo em uma imagem. Fonte: [www.geeksforgeeks.org](http://www.geeksforgeeks.org).

## Filtro da Mediana

Essa operação reduz o nível de ruído na imagem. No filtro da mediana, cada pixel  $p(x,y)$  é substituído pelo valor da mediana da matriz que circunda  $p(x,y)$  (**máscara**). A Figura 2 apresenta dois exemplos de máscara com dimensões distintas. Iremos considerar que o tamanho da máscara é 3x3, ou seja, a máscara com valor central  $p(x,y)$  é formada pelos pixels  $p(x-1,y-1)$ ,  $p(x-1,y)$ ,  $p(x-1,y+1)$ ,  $p(x,y-1)$ ,  $p(x,y)$ ,  $p(x,y+1)$ ,  $p(x+1,y-1)$ ,  $p(x+1,y)$ ,  $p(x+1,y+1)$ .



**Figura 2** Exemplos de máscaras com dimensões 3x3 e 5x5. Fonte: <http://wiki.inf.ufpr.br/maziero>.

Para pixels localizados nas bordas da imagem, algumas posições da máscara são inválidas (possuem valores que ultrapassam os limites da matriz) e devem ser desconsideradas no cálculo da mediana. Por exemplo, para o pixel  $p(0,0)$ , a máscara a ser considerada para o cálculo da mediana é formada pelos pixels  $p(0,0)$ ,  $p(0,1)$ ,  $p(1,0)$  e  $p(1,1)$ .

## Convolução

A operação de convolução permite a realização de vários filtros, como *blur* (perda de foco), *edge detect* (detecção de arestas), e *sharpen* (nitidez). Nessa operação, além da matriz correspondente a imagem de entrada, a operação de convolução possui uma matriz núcleo  $M$  e um divisor inteiro  $D$ .

Consideramos que a matriz núcleo tem a seguinte forma, onde cada letra representa um valor inteiro:





$$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Cada pixel  $p(x,y)$  da imagem de entrada é substituído pelo pixel  $p'(x,y)$ . O valor de  $p'(x,y)$  é calculado de acordo com a fórmula abaixo:

$$\begin{aligned} l_1 &= a \cdot p(x-1, y-1) + b \cdot p(x-1, y) + c \cdot p(x-1, y+1) \\ l_2 &= d \cdot p(x, y-1) + e \cdot p(x, y) + f \cdot p(x, y+1) \\ l_3 &= g \cdot p(x+1, y-1) + h \cdot p(x+1, y) + i \cdot p(x+1, y+1) \\ p'(x, y) &= \left\lfloor \frac{l_1 + l_2 + l_3}{D} \right\rfloor \end{aligned}$$

Note que o resultado da fórmula acima é equivalente a considerar a parte inteira da divisão  $(l_1 + l_2 + l_3)/D$ . Para pixels na borda da imagem, alguns valores da fórmula são inválidos e, portanto, ela não poderá ser aplicada. Sendo assim, na implementação da convolução, você deverá remover os pixels da borda, ou seja, os pixels na borda da imagem original não são incluídos na imagem resultante, que terá  $n-2$  linhas e  $m-2$  colunas. Todos os pixels devem estar no intervalo  $[0, 255]$  e, portanto, valores negativos devem ser substituídos por 0 e valores maiores que 255 devem ser substituídos por 255.

A Figura 3 apresenta exemplos da aplicação de filtros usando a operação de convolução, sendo que cada filtro possui matriz núcleo  $M$  e valor de  $D$  específicos.

|  |   |
|--|---|
|   |    |
| (a) Imagem original  | <p>(b) Aplicação do filtro <i>Sharpen</i>:</p> $M = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \text{ e } D = 1$         |
|    |    |
| <p>(c) Aplicação do filtro <i>blur</i>:</p> $M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \text{ e } D = 9$ | <p>(d) Aplicação do filtro <i>edge detect</i>:</p> $M = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \text{ e } D = 1$ |

**Figura 3** Exemplo de aplicação de filtros usando a operação de convolução.

**Observação:** Para os últimos dois filtros (mediana e convolução), você deve guardar os valores da imagem original, já que os valores dos vizinhos de um pixel são definidos com os valores da imagem original.

## Entrada

O seu programa receberá como entrada as seguintes linhas (em ordem): uma linha com o nome do filtro a ser aplicado ( *negativo* , *mediana* , *blur* , *sharpen* , *edge-detect* ); uma linha contendo o valor  $p_2$  , que indica o formato do arquivo e deve ser desconsiderada; uma linha contendo dois números  $m$  e  $n$  , que indicam o número de colunas e linhas, respectivamente; uma linha que contém o valor máximo que um pixel pode assumir (no nosso caso, esse valor sempre será

255 ); e, por último,  $n$  linhas que possuem  $m$  valores inteiros separados por espaços, representando os valores de cada pixel da imagem em escala de cinza, na ordem *raster* (de cima para baixo, da esquerda para direita). Exceto pela primeira linha que descreve o filtro a ser aplicado, a entrada constitui uma imagem no formato PGM.

## Saída

---

A saída deverá ser impressa no formato PGM, contendo a imagem resultante da aplicação do filtro. As primeiras três linhas representam o cabeçalho da imagem gerada: a primeira linha deverá conter apenas a string `P2`, que indica o formato do arquivo; a segunda linha conterá dois números inteiros indicando o número de colunas e o número de linhas da imagem, respectivamente; e a terceira linha indicará o valor máximo que um pixel da imagem pode conter (que sempre será 255). As linhas seguintes deverão conter os valores (números inteiros) dos pixels resultantes da aplicação do filtro, esses valores são separados por um único caractere em branco.

Você pode visualizar as imagens no formato PGM usando este [site](#). Para entender o efeito de cada filtro, visualize todas imagens dos casos de [testes abertos](#), tanto as imagens de entrada (arquivos \*.in), quanto as de saídas (arquivos \*.out).

## Código Base

---

No arquivo auxiliar `lab12.py` você irá encontrar um código base para dar início ao processo de elaboração deste laboratório. Para facilitar a implementação do seu programa, no código base existem os cabeçalhos de funções com a descrição do que deve ser desenvolvido em cada uma delas. Além disso, disponibilizamos uma função que calcula a mediana de uma lista de números e uma função que imprime uma imagem no formato PGM. Cada função desempenha uma tarefa bastante específica. Usando essas funções é possível obter uma solução para o problema. Os cabeçalhos das funções a serem implementadas são apresentados a seguir.

```
def filtro_negativo(imagem_original):  
    ...  
  
def filtro_mediana(imagem_original):  
    ...  
  
def convolucao(imagem_original, M, D):  
    ...
```

Exemplos de entradas e saídas esperadas pelo seu programa:

## Teste 01

### Entrada

```
negativo
P2
24 7
255
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 30 30 30 30 0 0 30 30 30 30 0 0 100 100 100 100 0 0 180 180 180 180 0
0 30 0 0 0 0 0 30 0 0 0 0 0 100 0 0 0 0 180 0 0 180 0
0 30 30 30 0 0 0 30 30 30 0 0 0 100 100 100 0 0 0 180 180 180 180 0
0 30 0 0 0 0 0 30 0 0 0 0 0 100 0 0 0 0 180 0 0 0 0
0 30 0 0 0 0 0 30 30 30 30 0 0 100 100 100 100 0 0 180 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

### Saída

```
P2
24 7
255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 225 225 225 225 255 255 225 225 225 225 255 255 155 155 155 155 255 255 75 7
255 225 255 255 255 255 255 225 255 255 255 255 155 255 255 255 255 255 75 2
255 225 225 225 255 255 255 225 225 225 255 255 155 155 155 255 255 255 75 7
255 225 255 255 255 255 255 225 255 255 255 255 155 255 255 255 255 255 75 2

255 225 255 255 255 255 255 225 225 225 225 255 255 155 155 155 155 255 255 75 2
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

## Teste 02

### Entrada

```
mediana
P2
24 7
255
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 30 30 30 30 0 0 30 30 30 30 0 0 100 100 100 100 0 0 180 180 180 180 0
0 30 0 0 0 0 0 30 0 0 0 0 0 100 0 0 0 0 180 0 0 180 0
0 30 30 30 0 0 0 30 30 30 0 0 0 100 100 100 0 0 0 180 180 180 180 0
0 30 0 0 0 0 0 30 0 0 0 0 0 100 0 0 0 0 180 0 0 0 0
0 30 0 0 0 0 0 30 30 30 30 0 0 100 100 100 100 0 0 180 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

## Saída

```

P2
24 7
255
0 0 15 15 0 0 0 0 15 15 0 0 0 0 50 50 0 0 0 0 90 90 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
15 30 30 30 0 0 0 30 30 30 0 0 0 100 100 100 0 0 0 180 180 180 180 90
15 0 30 0 0 0 0 0 30 0 0 0 0 0 100 0 0 0 0 0 180 0 0 0
15 0 30 0 0 0 0 30 30 30 0 0 0 100 100 100 0 0 0 0 180 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 15 15 0 0 0 0 50 50 0 0 0 0 0 0 0 0

```

## Teste 03

## Entrada

```

edge-detect
P2
24 7
255
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 30 30 30 30 0 0 30 30 30 30 0 0 100 100 100 100 0 0 180 180 180 180 0
0 30 0 0 0 0 0 30 0 0 0 0 0 100 0 0 0 0 0 180 0 0 180 0
0 30 30 30 0 0 0 30 30 30 0 0 0 100 100 100 0 0 0 180 180 180 180 0
0 30 0 0 0 0 0 30 0 0 0 0 0 100 0 0 0 0 0 180 0 0 0 0
0 30 0 0 0 0 0 30 30 30 30 0 0 100 100 100 100 0 0 180 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

## Saída

```

P2
22 5
255
180 150 180 210 0 0 180 150 180 210 0 0 255 255 255 255 0 0 255 255 255 255
120 0 0 0 0 0 120 0 0 0 0 0 255 0 0 0 0 255 0 0 255
150 120 210 0 0 0 150 120 210 0 0 0 255 255 255 0 0 0 255 255 255 255
150 0 0 0 0 0 120 0 0 0 0 0 255 0 0 0 0 255 0 0 0
210 0 0 0 0 0 180 150 180 210 0 0 255 255 255 255 0 0 255 0 0 0

```

## Orientações

- Veja [aqui](#) a página de submissão da tarefa.
- O arquivo a ser submetido deve se chamar lab12.py.
- No link "Arquivos auxiliares" há um arquivo compactado (aux12.zip) que contém todos os arquivos de testes abertos (entradas e saídas esperadas).

- O laboratório é composto de 10 testes abertos e 10 testes fechados.
- O limite máximo será de 20 submissões.
- Acesse o sistema SuSy com seu RA (apenas números) e a senha que você utiliza para fazer acesso ao sistema da DAC.
- Você deve seguir as instruções de submissão descritas no enunciado.
- Serão considerados apenas os resultados da última submissão.
- Esta tarefa tem peso 3.
- O prazo final para submissão é dia 27/12/2020 (domingo).