UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
DEPARTAMENTO DE COMPUTAÇÃO - DC
CENTRO DE CIÊNCIAS DA NATUREZA - CCN
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA: PROCESSAMENTO DIGITAL DE IMAGENS
PROFESSORES: Kelson R. T. Aires PERÍODO: 2024.1

Relatório

Aluno: Járdesson Ribeiro

1. Criar uma imagem contendo apenas os pontos da fronteira do objeto usando:

Figura 1 - Imagem Base - “aviao.png”

Imagens Processadas

Figura 2 - Adjacência 4

Figura 3 - Adjacência 8

Figura 4 - Adjacência M

2. Execute o mesmo procedimento da questão anterior para a imagem “folha.png”.

Figura 5 - Imagem Base - “folha.png”

Imagens Processadas

		
Figura 6 - Adjacência 4	**Figura 7 - Adjacência 8**	**Figura 8 - Adjacência M**

3. Para cada uma das imagens resultantes das questões 1 e 2, gere uma imagem para cada Vizinhos - 4, 8 e d para cada imagem anterior:
    a. Com base na **Figura 2 - Adjacência 4**

		
Figura 9 - Vizinhos - 4	**Figura 10 - Vizinhos - 8**	**Figura 11 - Vizinhos - d**

    b. Com base na **Figura 3 - Adjacência 8**

		
Figura 12 - Vizinhos - 4	**Figura 13 - Vizinhos - 8**	**Figura 14 - Vizinhos - d**

c. Com base na **Figura 4 - Adjacência M**

		
Figura 15 - Vizinhos - 4	**Figura 16 - Vizinhos - 8**	**Figura 17 - Vizinhos - d**

d. Com base na **Figura 6 - Adjacência 4**

		
Figura 18 - Vizinhos - 4	**Figura 19 - Vizinhos - 8**	**Figura 20 - Vizinhos - d**

e. Com base na **Figura 7 - Adjacência 8**

		
Figura 21 - Vizinhos - 4	**Figura 22 - Vizinhos - 8**	**Figura 23 - Vizinhos - d**

f. Com base na **Figura 8 - Adjacência M**

		
Figura 24 - Vizinhos - 4	**Figura 25 - Vizinhos - 8**	**Figura 26 - Vizinhos - d**

4. Análise
   a. **Tópico 1**:

Com base na **Figura 1 - "aviao.png"** foram criadas outras 3 imagens representadas nas figuras **Figura 2, Figura 3 e Figura 4** para os 3 tipos de adjacências, **Adjacência 4**, **Adjacência 8** e **Adjacência M**, respectivamente. O intuito era criar imagens apenas com a forma da figura presente na **Figura 1**, de acordo com a fronteira, aplicando os conceitos de adjacência entre pixels.

É possível notar que a **Figura 3** que usou **Adjacência 8** possui uma borda um pouco mais grossa que as outras figuras representadas por **Figura 2** e **Figura 4**. Isso ocorre tendo em vista o algoritmo que define o que é a adjacência para essa imagem. Nesse exemplo, a Adjacência 8 utiliza 8 pixels que estão ao redor do pixel atual, buscando pelo menos um pixels com diferença de intensidade. Logo, ela usa tanto os pixels da **Adjacência 4** quanto **Adjacência d**, que aumenta a chance de um pixel ser enquadrado como fronteira. Por consequência, a borda criada na imagem **Figura 3** é mais espessa que as duas outras figuras.

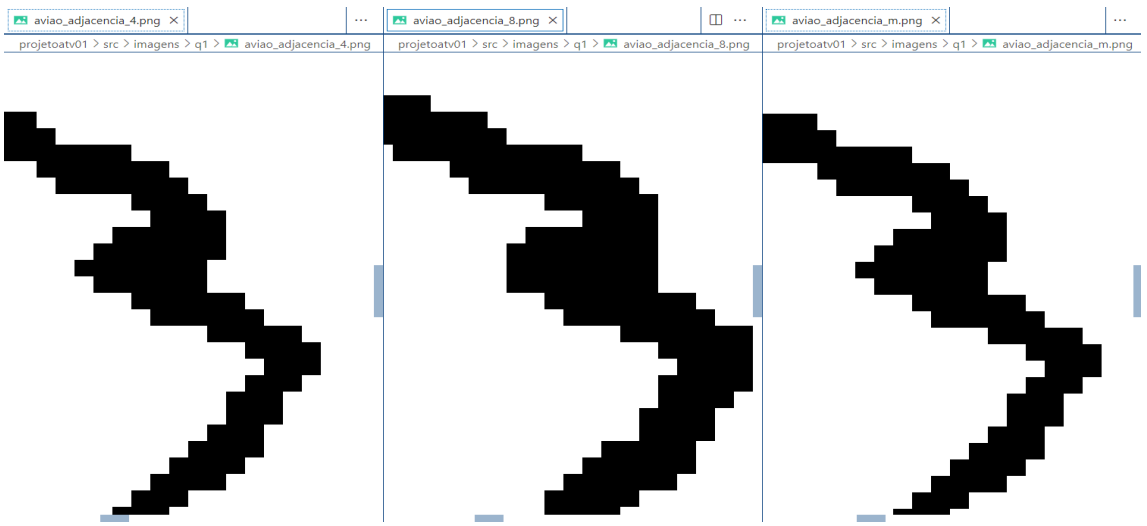

Figura 27 - Comparação usando Zoom em região específica

Na **Figura 27** podemos notar que a **Figura 3**, no centro, ocupa mais pixels em relação às duas outras figuras, pois como salientado ele utiliza **Adjacência 8** o que permite que um pixel tenha mais chance de ser colocado como fronteira. Outro detalhe, que a **Figura 2** e **Figura 4** possuem basicamente a mesma espessura na região ampliada. Isso deve-se ao fato de que a **Adjacência M** utilizar também a **Adjacência 4**, apenas tendo outro detalhe de haver uma verificação de interseção entre pixels quando buscados por **Adjacência 4 Diagonal.**

```
def is_intensidade_permitida(self, pixelAtual, pixelVizinho) -> bool:
    r1, g1, b1, a1 = pixelAtual
    r2, g2, b2, a2 = pixelVizinho

    intensidadePixelAtual = (r1 + g1 + b1) / 3
    intensidadePixelVizinho = (r2 + g2 + b2) / 3

    diferencaIntensidade = abs(intensidadePixelAtual - intensidadePixelVizinho)

    return self.intensidade > diferencaIntensidade
```

Figura 28 - Código para cálculo de diferença de intensidade

```
def verificar_adjacencia_4(self, x: int, y: int, matriz_imagem) -> bool:
    dx = [0, 1, 0, -1]
    dy = [-1, 0, 1, 0]
    is_fronteira = False
    for i in range(0, 4):
        linha = x + dx[i]
        coluna = y + dy[i]
        if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
            pixelVizinho = matriz_imagem[linha, coluna]
            if not self.is_intensidade_permitida(matriz_imagem[x, y], pixelVizinho):
                is_fronteira = True
                break

    return is_fronteira
```

Figura 29 - Código de verificação de Adjacência 4

```
def verificar_adjacencia_8(self, x: int, y: int, matriz_imagem) -> bool:
    dx = [0, 1, 0, -1, -1, 1, -1, 1]
    dy = [-1, 0, 1, 0, -1, -1, 1, 1]
    is_fronteira = False
    for i in range(0, 8):
        linha = x + dx[i]
        coluna = y + dy[i]
        if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
            pixelVizinho = matriz_imagem[linha, coluna]
            if not self.is_intensidade_permitida(matriz_imagem[x, y], pixelVizinho):
                is_fronteira = True
                break
    return is_fronteira
```

Figura 30 - Código de verificação de Adjacência 8

O código presente na **Figura 28** calcula a diferença de intensidade da cor do pixel, por se tratar de uma imagem em tons de cinza o preto com o valor de RGB(0, 0, 0) e o branco em RGB(255, 255, 255) podemos aplicar uma variação de intensidade igual a 1.0 que é o suficiente para marcar notar a diferença. Na **Figura 28** apenas “visita” os vizinhos 4 do pixel atual e verifica se há diferença de intensidade.

Na **Figura 30** já utilizamos tanto **Adjacência 4** normal quanto a **Adjacência Diagonal** o que forma a **Adjacência 8**

b. **Tópico 2**:

Realizamos o mesmo procedimento a com a **Figura 5,** que é constituída de tons de cinza, mesmo tendo a figura definida com o fundo branco e o fundo da imagem sendo branco, porém a forma de definir as fronteiras é o mesmo processo.

Analisando as figuras **Figura 6**, **Figura 7** e **Figura 8** podemos observar os mesmos resultados da **Figura 1,** no qual a **Adjacência 8** possui mais espessura que a **Adjacência 4** e **Adjacência M**, que por sua vez possuem a mesma espessura de borda.

c. **Tópico 3**:

A ideia central é pegar cada imagem gerada nos **Tópicos 1** e **2** e para cada uma dessas imagens definir os **vizinhos - 4, 8 e d** e marcar como pertencente à fronteira da figura.

Primeiramente, vamos pegar as figuras **Figura 9**, **Figura 10** e **Figura 11** para analisarmos usando uma ampliação em uma região específica para termos mais ou menos uma ideia de como fica quando marcamos usando **Vizinhos-4, Vizinhos-8 e Vizinhos-d** para as figuras mencionadas respectivamente.

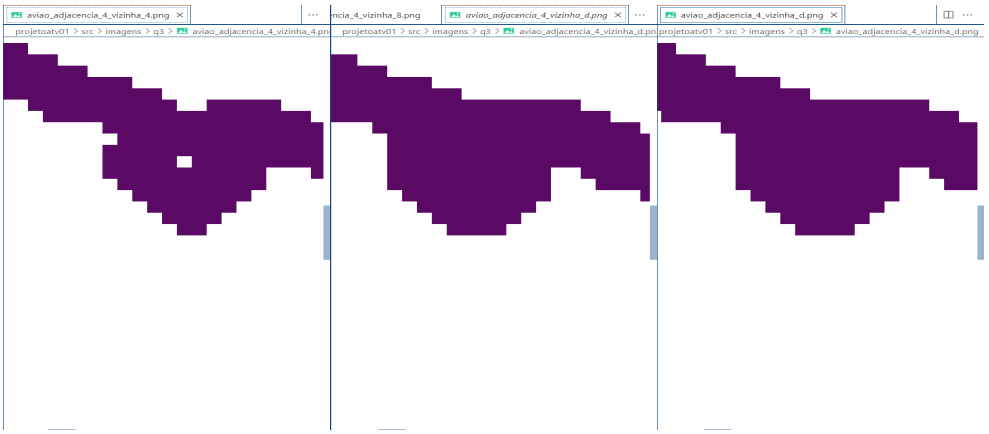

Figura 31 - Ampliação com vizinhos-4, 8 e d respectivamente.

Podemos observar que ao marcar os vizinhos de uma fronteira, na **Figura 2**, podemos notar que a **Figura 9** apresenta uma espessura de pixels menor que os **vizinhos 8 e d**. O que é diferente da abordagem anterior, na qual **Adjacência 8** era mais espessa que **Adjacência 4** e **Adjacência M.**

```python
def colorir_vizinho_4(self, x: int, y: int, matriz_imagem_final):
    dx = [0, 1, 0, -1]
    dy = [-1, 0, 1, 0]
    for i in range(0, 4):
        linha = x + dx[i]
        coluna = y + dy[i]
        if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
            matriz_imagem_final[linha, coluna] = self.cor_pixel_vizinho

def colorir_vizinho_d(self, x: int, y: int, matriz_imagem_final):
    dx = [-1, 1, -1, 1]
    dy = [-1, -1, 1, 1]
    for i in range(0, 4):
        linha = x + dx[i]
        coluna = y + dy[i]
        if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
            matriz_imagem_final[linha, coluna] = self.cor_pixel_vizinho

def colorir_vizinho_8(self, x: int, y: int, matriz_imagem_final):
    dx = [0, 1, 0, -1, -1, 1, -1, 1]
    dy = [-1, 0, 1, 0, -1, -1, 1, 1]
    for i in range(0, 8):
        linha = x + dx[i]
        coluna = y + dy[i]
        if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
            matriz_imagem_final[linha, coluna] = self.cor_pixel_vizinho
```

Figura 32 - Código para coloração de vizinhos da fronteira

A **Figura 32** mostra o código para colorir cada pixel vizinho do pixel da fronteira. Nesse passo foi definido uma cor, RGB `(94, 14, 102, 255)` para colorir os pixels vizinhos, o que possibilitou identificar que os pixels da fronteira também são vizinhos de algum pixel da fronteira.

```python
def is_pixel_fronteira(self, pixel) -> bool:
    r, g, b = pixel
    r2, g2, b2, alpha = self.cor_pixels_fronteira

    return (r == r2 and g == g2 and b == b2)
```

Figura 33 - Código para identificar pixel de fronteira

A **Figura 33** mostra o código que define os pixels da fronteira, tendo em vista que a fronteira tem apenas uma coloração diferente de cor de fundo, logo o código apenas verifica se há diferença entre as cores dos pixels.

Em conclusão, podemos observar que a Adjacência 4 consegue marcar menos pixels como adjacentes, tendo em vista que ele pega apenas os pixels de cima, baixo, esquerda e direita, e esses devem ter uma variação de intensidade para serem marcados como adjacentes. Por outro lado, quando utilizamos **Adjacência 8** e **Adjacência M** temos uma maior chance de colocar como adjacente, pois usa tanto a **Adjacência 4 e Adjacência 4 Diagona**l. Ademais, a **Adjacência M** já utiliza a mesma **Adjacência 4 Diagonal** com uma restrição de interseção entre os conjuntos dos pixels adjacentes entre os pixels em **N4(**p**)** e **N4(**q**)** onde q é um pixel pertencente a **Nd(p).**

Ainda em conclusão, podemos notar que ao marcar os **vizinhos 4, 8** e **d** podemos notar que **vizinhos 8** e **vizinhos d** apresentam a mesma espessura marcando os mesmos vizinhos da fronteira. Notamos ainda, que na **Adjacência 8,** todos os vizinhos apresentam basicamente a mesma espessura de vizinhos na fronteira, representados nas **Figura 12, 13 e 14.**

Códigos Completos em Anexo

main.py

```
from PIL import Image
import os

class IdentificadorFronteira:
    cor_fundo = (255, 255, 255)
    cor_figura = (0, 0, 0)
    cor_figura_rgba = (0, 0, 0, 255)
    def __init__(self, imagem: str, valorIntensidade: float) -> None:
        self.nome_imagem = imagem
        self.imagemParaProcessamento = self.obter_arquivo_imagem()
        self.largura = self.imagemParaProcessamento.size[0]
        self.comprimento = self.imagemParaProcessamento.size[1]
        self.imagemFinal = Image.new("RGB", (self.largura, self.comprimento), self.cor_fundo)
        self.intensidade = valorIntensidade

    def obter_arquivo_imagem(self) -> Image :
        diretorio_atual = os.path.dirname(os.path.abspath(__file__))
        caminho_imagem = os.path.join(diretorio_atual, "imagens", self.nome_imagem)

        return Image.open(caminho_imagem)

    def is_intensidade_permitida(self, pixelAtual, pixelVizinho) -> bool:
        r1, g1, b1, a1 = pixelAtual
        r2, g2, b2, a2 = pixelVizinho

        intensidadePixelAtual = (r1 + g1 + b1) / 3
        intensidadePixelVizinho = (r2 + g2 + b2) / 3

        diferencaIntensidade = abs(intensidadePixelAtual - intensidadePixelVizinho)

        return self.intensidade > diferencaIntensidade

    def verificar_adjacencia_4(self, x: int, y: int, matriz_imagem) -> bool:
        dx = [0, 1, 0, -1]
        dy = [-1, 0, 1, 0]
        is_fronteira = False
        for i in range(0, 4):
           linha = x + dx[i]
           coluna = y + dy[i]
           if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
               pixelVizinho = matriz_imagem[linha, coluna]
               if not self.is_intensidade_permitida(matriz_imagem[x, y], pixelVizinho):
                   is_fronteira = True
                   break

        return is_fronteira

    def definir_adjacencia_4_para_pixel_diagonal(self, x: int, y: int, matriz_imagem):
        dx4 = [0, 1, 0, -1]
        dy4 = [-1, 0, 1, 0]
        adjacencia4 = []
        for i in range(0, 4):
           linha = x + dx4[i]
           coluna = y + dy4[i]
           if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
               adjacencia4.append(matriz_imagem[linha, coluna])

        return adjacencia4

    def verificar_adjacencia_m(self, x: int, y: int, matriz_imagem) -> bool:
        dx4 = [0, 1, 0, -1]
        dy4 = [-1, 0, 1, 0]
```

```
                is_fronteira = False
                adjacencia4_pixel_principal = []
                for i in range(0, 4):
                   linha = x + dx4[i]
                   coluna = y + dy4[i]
                   if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
                       pixelVizinho = matriz_imagem[linha, coluna]
                       adjacencia4_pixel_principal.append(pixelVizinho)
                       if not self.is_intensidade_permitida(matriz_imagem[x, y], pixelVizinho):
                           is_fronteira = True
                           break

                dxd = [-1, 1, -1, 1]
                dyd = [-1, -1, 1, 1]
                for i in range(0, 4):
                    linha = x + dxd[i]
                    coluna = y + dyd[i]
                    if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento
:
                        pixel_vizinho_adjacentes_4 = self.definir_adjacencia_4_para_pixel_diagonal(linha,
coluna, matriz_imagem)
                        for pixel_principal in adjacencia4_pixel_principal:
                            if pixel_principal in pixel_vizinho_adjacentes_4:
                                if not self.is_intensidade_permitida(matriz_imagem[x, y],
pixel_principal):
                                    is_fronteira = True
                                    break

                return is_fronteira


            def definir_fronteira_imagem_adjacencia_m(self, nome_imagem: str) -> Image:
                matriz_imagem = self.imagemParaProcessamento.load()
                matriz_imagem_processada = self.imagemFinal.load()
                for x in range(self.largura):
                    for y in range(self.comprimento):
                        is_fronteira = self.verificar_adjacencia_m(x, y, matriz_imagem)
                        if is_fronteira:
                            matriz_imagem_processada[x, y] = self.cor_figura_rgba
                self.salvar_imagem(nome_imagem)

            def definir_fronteira_imagem_adjacencia_4(self, nome_imagem: str) -> Image:
                matriz_imagem = self.imagemParaProcessamento.load()
                matriz_imagem_processada = self.imagemFinal.load()
                for x in range(self.largura):
                    for y in range(self.comprimento):
                        is_fronteira = self.verificar_adjacencia_4(x, y, matriz_imagem)
                        if is_fronteira:
                            matriz_imagem_processada[x, y] = self.cor_figura_rgba
                self.salvar_imagem(nome_imagem)

            def salvar_imagem(self, nome_imagem: str):
                diretorio_atual = os.path.dirname(os.path.abspath(__file__))
                caminho_imagem = os.path.join(diretorio_atual, "imagens\\q1", nome_imagem)

                self.imagemFinal.save(caminho_imagem)

            def verificar_adjacencia_8(self, x: int, y: int, matriz_imagem) -> bool:
                dx = [0, 1, 0, -1, -1, 1, -1, 1]
                dy = [-1, 0, 1, 0, -1, -1, 1, 1]
                is_fronteira = False
                for i in range(0, 8):
                   linha = x + dx[i]
                   coluna = y + dy[i]
                   if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
```

```
                        pixelVizinho = matriz_imagem[linha, coluna]
                        if not self.is_intensidade_permitida(matriz_imagem[x, y], pixelVizinho):
                            is_fronteira = True
                            break
                return is_fronteira

            def definir_fronteira_imagem_adjacencia_8(self, nome_imagem: str) -> Image:
                matriz_imagem = self.imagemParaProcessamento.load()
                matriz_imagem_processada = self.imagemFinal.load()
                for x in range(self.largura):
                    for y in range(self.comprimento):
                        is_fronteira = self.verificar_adjacencia_8(x, y, matriz_imagem)
                        if is_fronteira:
                            matriz_imagem_processada[x, y] = self.cor_figura_rgba
                self.salvar_imagem(nome_imagem)


        def main():
            identificadorFronteiraAdjacencia4 = IdentificadorFronteira("aviao.png", 1.0)
            identificadorFronteiraAdjacencia8 = IdentificadorFronteira("aviao.png", 1.0)
            identificadorFronteiraAdjacenciaM = IdentificadorFronteira("aviao.png", 1.0)


identificadorFronteiraAdjacencia4.definir_fronteira_imagem_adjacencia_4("aviao_adjacencia_4.png")

identificadorFronteiraAdjacencia8.definir_fronteira_imagem_adjacencia_8("aviao_adjacencia_8.png")

identificadorFronteiraAdjacenciaM.definir_fronteira_imagem_adjacencia_m("aviao_adjacencia_m.png")


            identificadorFronteiraAdjacencia4 = IdentificadorFronteira("folha.png", 1.0)
            identificadorFronteiraAdjacencia8 = IdentificadorFronteira("folha.png", 1.0)
            identificadorFronteiraAdjacenciaM = IdentificadorFronteira("folha.png", 1.0)


identificadorFronteiraAdjacencia4.definir_fronteira_imagem_adjacencia_4("folha_adjacencia_4.png")

identificadorFronteiraAdjacencia8.definir_fronteira_imagem_adjacencia_8("folha_adjacencia_8.png")

identificadorFronteiraAdjacenciaM.definir_fronteira_imagem_adjacencia_m("folha_adjacencia_m.png")

        if __name__ == "__main__":
            main()
```

identificador_vizinhaca.py

```
        import os
        from PIL import Image

        class IdentificaroVizinhaca:
            cor_pixels_fronteira = (0, 0, 0, 255)
            cor_fundo = (255, 255, 255, 255)
            cor_pixel_vizinho = (94, 14, 102, 255)

            def __init__(self, nome_imagem: str) -> None:
                self.nome_imagem = nome_imagem
                self.imagem_para_processamento: Image = self.obter_arquivo_imagem()
```

```
        self.largura = self.imagem_para_processamento.size[0]
        self.comprimento = self.imagem_para_processamento.size[1]
        self.imagem_final = Image.new("RGB", (self.largura, self.comprimento), self.cor_fundo)

    def obter_arquivo_imagem(self) -> Image :
        diretorio_atual = os.path.dirname(os.path.abspath(__file__))
        caminho_imagem = os.path.join(diretorio_atual, "imagens\\q1", self.nome_imagem)

        return Image.open(caminho_imagem)

    def salvar_imagem(self, nome_imagem: str):
        diretorio_atual = os.path.dirname(os.path.abspath(__file__))
        caminho_imagem = os.path.join(diretorio_atual, "imagens\\q3", nome_imagem)

        self.imagem_final.save(caminho_imagem)

    def is_pixel_fronteira(self, pixel) -> bool:
        r, g, b = pixel
        r2, g2, b2, alpha = self.cor_pixels_fronteira

        return (r == r2 and g == g2 and b == b2)

    def colorir_vizinho_4(self, x: int, y: int, matriz_imagem_final):
        dx = [0, 1, 0, -1]
        dy = [-1, 0, 1, 0]
        for i in range(0, 4):
            linha = x + dx[i]
            coluna = y + dy[i]
            if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
                matriz_imagem_final[linha, coluna] = self.cor_pixel_vizinho

    def colorir_vizinho_d(self, x: int, y: int, matriz_imagem_final):
        dx = [-1, 1, -1, 1]
        dy = [-1, -1, 1, 1]
        for i in range(0, 4):
            linha = x + dx[i]
            coluna = y + dy[i]
            if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
                matriz_imagem_final[linha, coluna] = self.cor_pixel_vizinho

    def colorir_vizinho_8(self, x: int, y: int, matriz_imagem_final):
        dx = [0, 1, 0, -1, -1, 1, -1, 1]
        dy = [-1, 0, 1, 0, -1, -1, 1, 1]
        for i in range(0, 8):
            linha = x + dx[i]
            coluna = y + dy[i]
            if coluna >= 0 and coluna < self.largura and linha >= 0 and linha < self.comprimento :
                matriz_imagem_final[linha, coluna] = self.cor_pixel_vizinho

    def definir_vizinhos_k(self, nome_imagem_final, tipo: str):
        matriz_pixels_imagem_final = self.imagem_final.load()
        matriz_pixels_imagem_processamento = self.imagem_para_processamento.load()
        for x in range(0, self.largura):
            for y in range(0, self.comprimento):
                pixel_atual = matriz_pixels_imagem_processamento[x, y]
                if self.is_pixel_fronteira(pixel_atual):
                    if tipo == '4':
                        self.colorir_vizinho_4(x, y, matriz_pixels_imagem_final)
                    elif tipo == 'd':
                        self.colorir_vizinho_d(x, y, matriz_pixels_imagem_final)
                    elif tipo == '8':
                        self.colorir_vizinho_8(x, y, matriz_pixels_imagem_final)

        self.salvar_imagem(nome_imagem_final)
```

```
def main():
    imagens = ["aviao_adjacencia_4.png", "aviao_adjacencia_8.png", "aviao_adjacencia_m.png",
"folha_adjacencia_4.png", "folha_adjacencia_8.png", "folha_adjacencia_m.png"]
    vizinhacas = ["4", "8", "d"]

    for imagem in imagens:
        for vizinhaca in vizinhacas:
            split_nome_imagem = imagem.split(".")
            identificaroVizinhaca = IdentificaroVizinhaca(imagem)
            identificaroVizinhaca.definir_vizinhos_k(split_nome_imagem[0]+"_vizinha_"+vizinhaca+".png",
vizinhaca)


if __name__ == "__main__":
    main()
```