Processamento Digital de Imagens – Trabalho 3
Detecção de Componentes Conexos com Flood Fill

Guilherme L. Salomão, João Victor M. Freire,
Martin Heckmann, Renan D. Pasquantonio

14 de Outubro de 2020

1 Introdução

Existem diversas formas de transformarmos uma imagem em escala de cinza em uma imagem binária, isto é, em uma imagem que só tem dois valores de intensidade. Essas são bastante úteis no processamento digital de imagens, pois é uma forma de simplificar a informação contida em tal imagem. Em particular, essa técnica é útil no processo de segmentação de objetos.

Em uma imagem binária, podemos considerar valores não nulos como objetos e, assim, é interessante descobrir o conjunto de coordenadas que o representa. Para isso utilizamos técnicas de detecção de componentes conexos.

Um componente conexo é definido como um conjunto de pixels conectados. Ou seja, definida uma vizinhança, é possível sair de um ponto (x, y) do objeto e atingir todos os outros pontos dele passando apenas por vizinhos de mesma intensidade. Uma possível é a vizinhança-4, que considera como vizinhos os pixels acima, abaixo, à esquerda e à direita. A vizinhança-8 adiciona os quatro pixels da diagonal aos considerados pela 4.

2 Motivação

Existem métodos de extração de componentes conexos que percorrem toda a imagem categorizando todos os componentes existentes com um rótulo diferente. No entanto, esse é um algoritmo que percorre a imagem toda mais de uma vez. Em determinadas aplicações, não é necessário encontrarmos todos os componentes conexos, mas apenas encontrar um objeto inteiro.

Um exemplo de aplicação que tem essa necessidade é a ferramenta *balde de tinta*, muito comum em programas de edição de imagens. Essa ferramenta, ao selecionar um pixel, altera a cor de todos os pixels adjacentes que possuem a mesma intensidade do selecionado.

O algoritmo que desempenha essa função de encontrar os pontos de um objeto, a partir de uma origem, é chamado de *Flood Fill*. Sua implementação e teste será o objeto de estudo neste terceiro trabalho prático.

3 Explicação do Método Implementado

3.1 Implementação com Busca em Largura

Uma das formas de implementar o Flood Fill é através de uma busca em largura na imagem. A seguir, está uma implementação utilizando BFS e vizinhança-4.

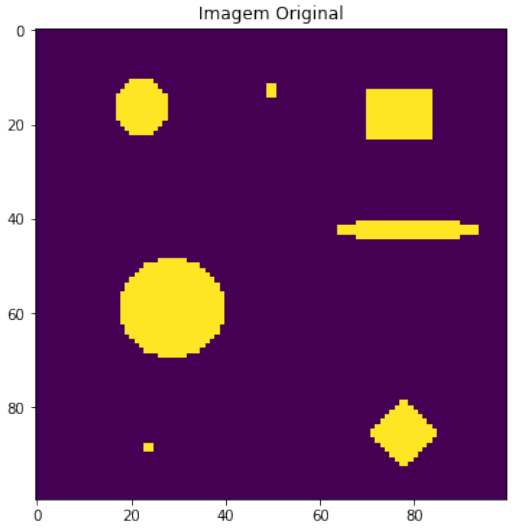
```
def flood_fill(img, origin):
    obj = []              # Lista com coordenadas do objeto
    visited = set()       # Conjunto dos pontos visitados
    queue = deque()       # Fila de pontos a serem visitados

    queue.append(origin)
    while queue:

        row, col = queue.popleft()

        if (row, col) not in visited:
            obj.append((row, col))
            visited.add((row, col))

            up    = (row-1, col)  # Vizinho de cima
            right = (row, col+1)  # Vizinho da direita
            down  = (row+1, col)  # Vizinho de baixo
            left  = (row, col-1)  # Vizinho da esquerda

            if row != 0:
                if (up not in visited) and \
                   (img[up] == img[origin]):
                    queue.append(up)

            if col != img.shape[1]-1:
                if (right not in visited) and \
                   (img[right] == img[origin]):
                    queue.append(right)

            if row != img.shape[0]-1:
                if (down not in visited) and \
                   (img[down] == img[origin]):
                    queue.append(down)

            if col != 0:
                if (left not in visited) and \
                   (img[left] == img[origin]):
                    queue.append(left)

    return obj
```

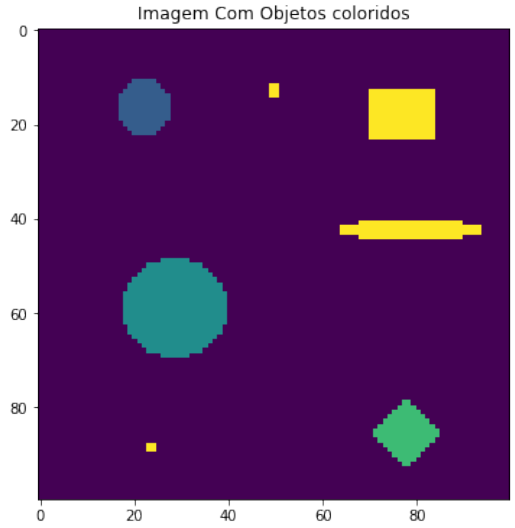
A função `flood_fill()` recebe a imagem que contém o objeto e um ponto dele, chamado `origin`. A variável `obj` armazenará as coordenadas encontradas do objeto. As estruturas auxiliares `visited` e `queue` irão armazenar os pontos já visitados e uma fila de pontos que ainda devemos visitar, respectivamente.

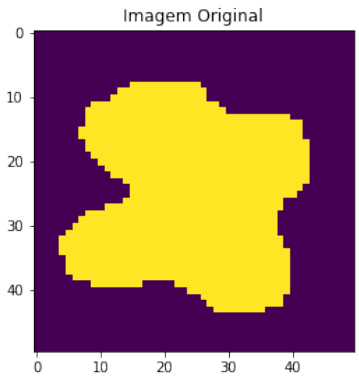
Inicialmente colocamos a origem na fila e entramos no laço principal. Enquanto ainda houverem pontos nela, selecionamos o da frente e, caso ainda não tenha sido visitado, adicionamos ao objeto. Na primeira iteração essa operação faz sentido já que a origem é um ponto do objeto, mas para as demais devemos garantir que só pontos do objeto entrem na fila.

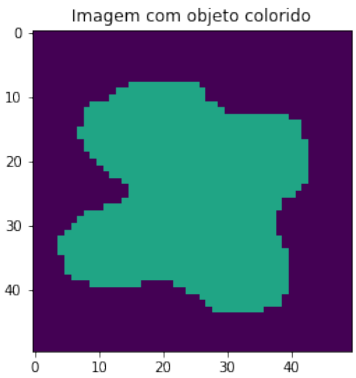
Para cada ponto da fila, olhamos quatro vizinhos, com coordenadas `up`, `down`, `left`, e `right`. É preciso levar em consideração pontos que podem estar na borda da imagem e que, portanto, têm apenas dois ou três vizinhos. Para isso verificamos se `(row, col)` tem valores `0`, `img_shape[0]-1`, ou `img_shape[1]-1`. Para os vizinhos válidos, devemos coloca-los na fila se ainda não foram visitados, e se tiverem a mesma intensidade da origem. Essa última verificação nos garante que todos os pontos colocados na fila cumprem os critérios para serem parte do objeto.

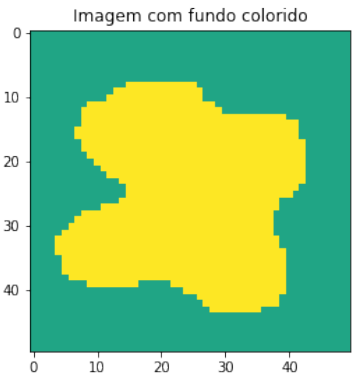
A seguir estão alguns testes realizados em imagens vistas em aula.











3.2 Implementação com Busca em Largura

É possível implementarmos o Flood Fill utilizando também uma busca em profundidade. Basta substituir a fila na implementação anterior por uma pilha ou utilizar a pilha da recursão. Decidimos, então, implementar uma versão recursiva do código com o objetivo de mudar a cor de um objeto, simulando o efeito *balde de tinta*.

```
def flood_fill_rec(img, x, y, replacement):
    target = img[x][y]
    flood_fill_recursao(img, x, y, target, replacement)
    return

def flood_fill_recursao(img, x, y, target, replacement):
    row,cols = img.shape
    if img[x][y] != target:
        return
    elif img[x][y] == replacement:
        return
    else:
        img[x][y] = replacement
        if (x-1)>=0:
            flood_fill_recursao(img, x-1, y, target, replacement)
        if(y-1)>=0:
            flood_fill_recursao(img, x, y-1, target, replacement)
        if(x+1<row):
            flood_fill_recursao(img, x+1, y, target, replacement)
        if(y+1<cols):
            flood_fill_recursao(img, x, y+1, target, replacement)
```

Aqui, a função recebe um ponto de origem (`origin`), a imagem e uma cor alternativa usada para substituir a cor original do objeto (`replacement`). Então, o algoritmo acessa o ponto de origem, verifica se a cor é igual a do objeto ou não é a desejada. Caso atenda essas condições, então tem sua cor trocada e a função é chamada recursivamente para seus quatro vizinhos, novamente respeitando as bordas da imagem.

Referências

[1] Comin, C. H. Processamento Digital de Imagens – Aula 21: Limiarização e Morfologia, 2020.

[2] Comin, C. H. Processamento Digital de Imagens – Aula 22: Operações Morfológicas e Descritores de Forma, 2020.

[3] Wikipedia. Flood Fill. `https://en.wikipedia.org/wiki/Flood_fill`.