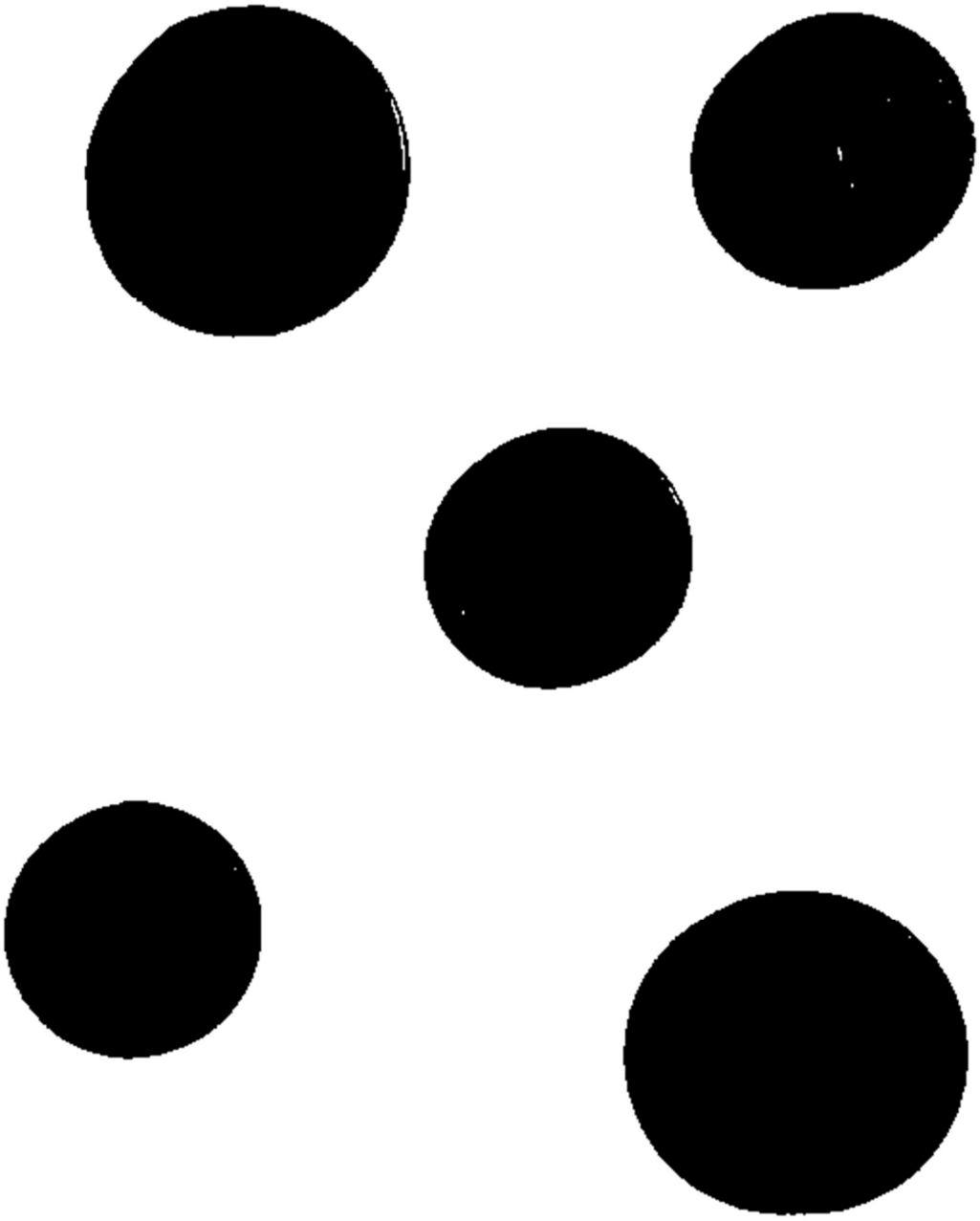
Relatório atividade moedas PIM 2022/2

Eduardo Pandini

```python
def main():

    moedas = Image.open("imagem.jpg").convert("L")
    pixels = moedas.load()
```

Código se inicia abrindo a imagem, convertendo a imagem para grayscale e carregando a matriz de pixels.



```
limiar = (0,150)
limiar_f(moedas, pixels, limiar)
moedas.show()
```

```python
def limiar_f(imagem, pixels, limiar:tuple):
    xsize, ysize = imagem.size

    for i in range(xsize):
        for j in range(ysize):
            if pixels[i,j] >= limiar[0] and pixels[i,j] <= limiar[1]:
                pixels[i,j] = 0
            else:
                pixels[i,j] = 255

    return pixels
```

É definido então o limiar e feita a conversão de todos os pixels com valor superior a 150 para 255, e os inferiores a 150 para 0 na grayscale

```python
todasM = localizar(moedas, pixels)
```

```python
def localizar(imagem, pixels):
    xsize, ysize = imagem.size

    objetos = []

    print("achando dinheiros hehe")
    for i in range(xsize):
        for j in range(ysize):
            if pixels[i,j] == 0 and not ignore_sublist(objetos, (i,j)):
                print("\tBFS")
                objeto = BFS(imagem, pixels, (i,j))
```

É enviada a matriz para a função “localizar”, primeiramente cria um vetor para guardar as moedas localizadas, após isso, caso identifique um pixel preto, irá invocar a função BFS, passando como parâmetro a imagem, a matriz de pixels e a localização do pixel preto na matriz

```python
def BFS(imagem, pixels, start):
    xsize, ysize = imagem.size
    moves = [(0,1),(0,2),(1,0),(2,0),(0,-1),(0,-2),(-1,0),(-2,0), (1,1), (1,-1),(-1,-1),(-1,1)]
    pretos = []
    visitar = [start]
    visitados = []
```

Na função BFS, é definida a movimentação em conectividade 4, são criados vetores para armazenar os pixels pretos, a matriz dos pixels a serem visitados, iniciando com o pixel original passado para a função, e a matriz dos pixels já visitados

```
ignorePixel = lambda x,y: x <= 0 or y <= 0 or x >= xsize or y >= ysize or (x,y) in visitados or (x,y) in pretos

while len(visitar) > 0:
    prox = visitar.pop(0)
    visitados.append(prox)

    vizinhos = []
    for move in moves:
        x,y = move[0]+prox[0], move[1]+prox[1]
        if ignorePixel(x,y):
            continue
        vizinhos.append((x,y))
    for vizinho in vizinhos:
        vx, vy = vizinho
        if pixels[vx,vy] == 0:
            pretos.append(vizinho)
            visitar.append(vizinho)
return pretos
```

A função então percorre a lista dos pixels a serem visitados, remove o pixel atual da lista e o coloca na lista de já visitados, então checa os seus vizinhos, que caso pretos, são colocados na lista “visitar”. Durante este processo também é feita a contagem de quantos vizinhos conectados estão no grupo.

```
    print("achando dinheiros hehe")
    for i in range(xsize):
        for j in range(ysize):
            if pixels[i,j] == 0 and not ignore_sublist(dinheiros, (i,j)):
                print("\tBFS")
                objeto = BFS(imagem, pixels, (i,j))
                dinheiros.append(objeto)
                print(f"\t{len(objeto)}")
    dinheiros = [din for din in dinheiros if len(din) >= 500]
    return dinheiros
```

A moeda é colocada então no vetor e tem seu tamanho printado, e descartamos todos os agrupamentos menores que 500 px

```
    todasM = localizar(moedas, pixels)

    contagemP = [len(moeda) for moeda in todasM]
    contagemP.sort()
    maiorMoeda = max(contagemP)
    moeda1r = []
    moeda10c = []
```

É criada uma lista com as moedas encontradas, e estas são ordenadas segundo sua contagem de px. Criamos então 2 vetores para armazenar as moedas de 1 real e 10 centavos

```
moeda1r = []
moeda10c = []
for tamanho in contagemP:
    if in_tuple(tamanho, (maiorMoeda-((1/10)*tamanho), maiorMoeda+((1/10)*tamanho))):
        moeda1r.append(tamanho)
    else:
        moeda10c.append(tamanho)
```

Para cada moeda, observamos se ela está com o tamanho próximo a maior moeda, foi utilizado uma variação de até 10% no tamanho, caso esteja, consideramos ela como uma moeda de 1 real, caso contrário, consideramos como uma moeda de 10 centavos

```
valor = len(moeda1r) + (0.1*len(moeda10c))

print("Dinheiros: R${:.2f}".format(valor))
```

É então somado o valor das moedas identificadas e o resultado é printado.

Considerações

O algoritmo é bastante lento, identifico como sendo um problema na análise dos vizinhos, que pode ter sido mal otimizada, caso for fazer o teste, mantenha isto em mente, ou utilize uma imagem com menor contagem de px. Outro problema foi eu ter percebido muito tarde a minha completa falta de moedas de 1 real, utilizei no lugar as de 25 centavos, que por terem o tamanho similar, funcionaram como substitutas, o algoritmo também funcionaria com moedas de 1 real, eu espero.