MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE CIÊNCIAS DA TERRA
Departamento de Geomática

Disciplina: PROCESSAMENTO DIGITAL DE IMAGENS II
Código: GA144

CH Total:45 h

CH Semanal 03 h

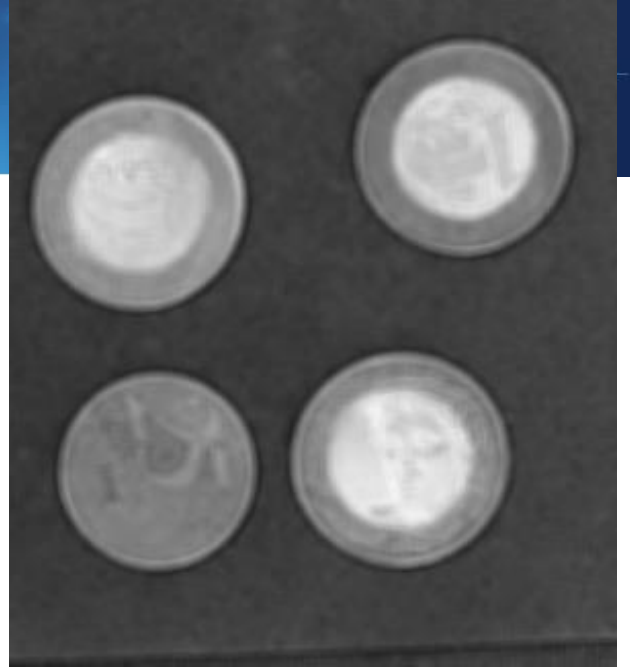
Componentes conexos

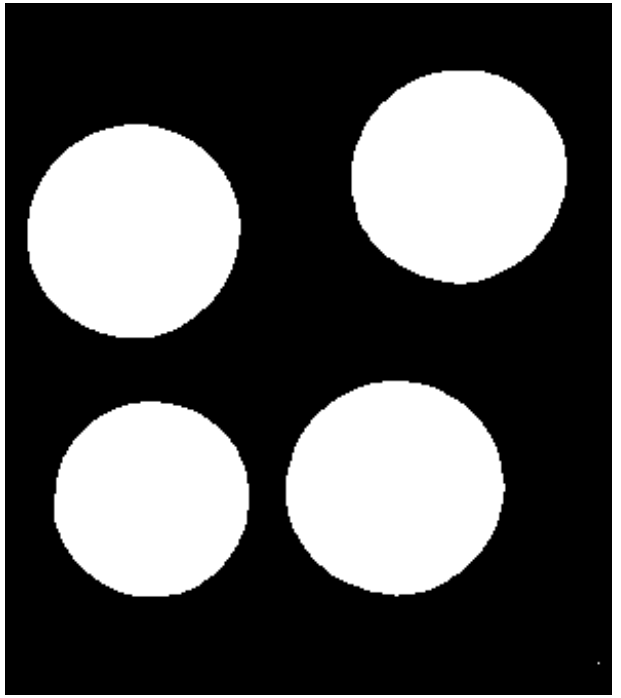
Aplicar suavização para uniformizar as regiões

E depois, bnarizar

```
im = cv2.blur(src=I, ksize=(5,5))
 cv2_imshow(im)

limiar=90
maximo=255
th, K = cv2.threshold(im, limiar, maximo, cv2.THRESH_BINARY)
cv2_imshow(K)
```





MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE CIÊNCIAS DA TERRA
Departamento de Geomática

Disciplina: PROCESSAMENTO DIGITAL DE IMAGENS II
Código: GA144

CH Total:45 h

CH Semanal 03 h

Método de OTSU

OTSU

Método de OTSU

Trata o Histograma da imagem como uma Função Densidade de Probabilidade Discreta:

$$p(x)=H(x)/N$$

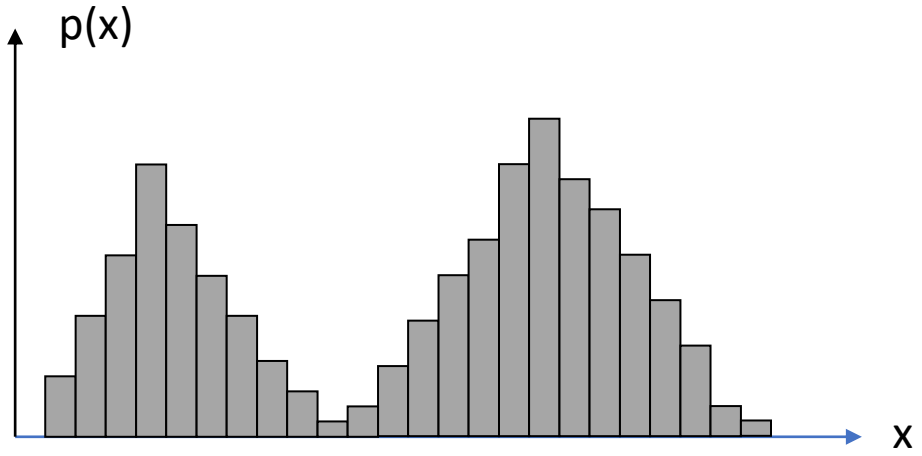
Onde:

x= valor digital, com q= 0,1,2,……255 (*pode ser outro valor máximo)

H(x) = número de pixels com valor digital x

N = número total de pixels na imagem

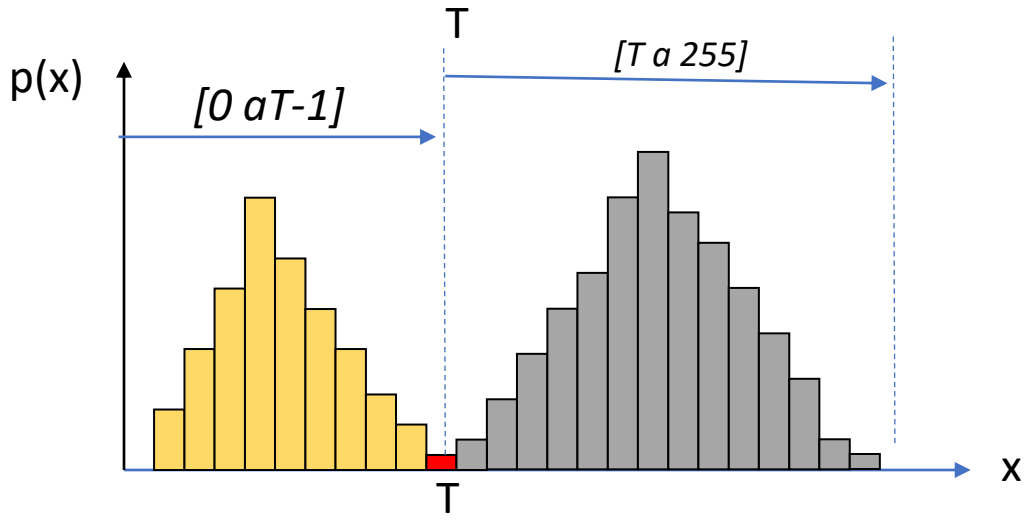
Qual valor é mais provável?
quele mais frequentemente!



OTSU

Usando o Limiar (T) pode-se separar a imagem em duas classes, dois grupos:

- A = pixels com valores entre [0, T-1] e
- B= pixels com valores entre [T, 255]

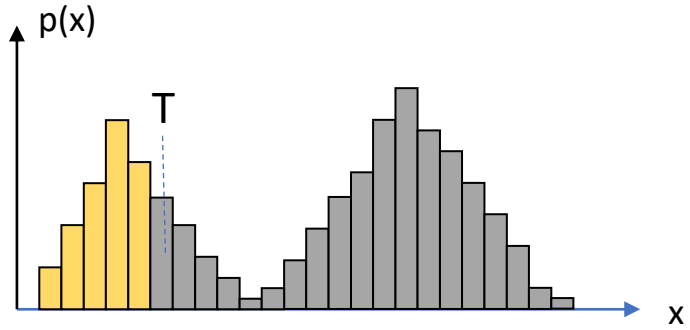



```
Limiar=100
if I[x,y] < Limiar:
  J[x,y] = 0
else:
  J[x,y] = 255
```

Cada grupo é descrito por:
soma das probabilidades das classes ,

a área de cada grupo (w)

- valor x médio (m)
- variância (S)



$$w_1 = \sum_{o}^{T-1} p(x) \qquad w_2 = \sum_{T}^{255} p(x)$$

$$m_1 = \sum_{0}^{T-1} x * p\,(x)/w_1 \qquad m_2 = \sum_{T}^{255} x * p\,(x)/w_1$$

$$S_1 = \sum_{0}^{T-1} (x - m_1)^2 * p\,(x)/w_1 \qquad S_2 = \sum_{T}^{255} (x - m_1)^2 * p\,(x)/w_2$$

Variância (S) mínima

O Limiar ótimo separa os pixels em dois grupos uniformes.

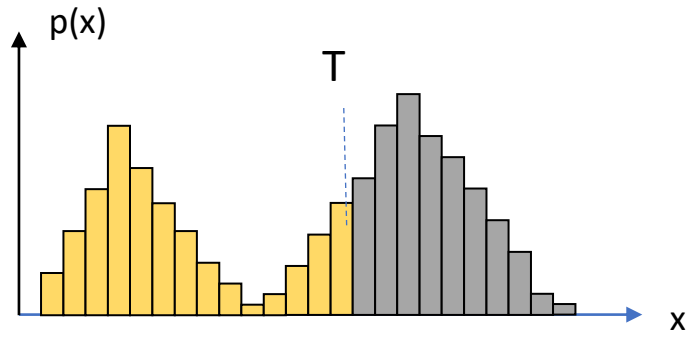
Grupos uniformes tem variância baixa.

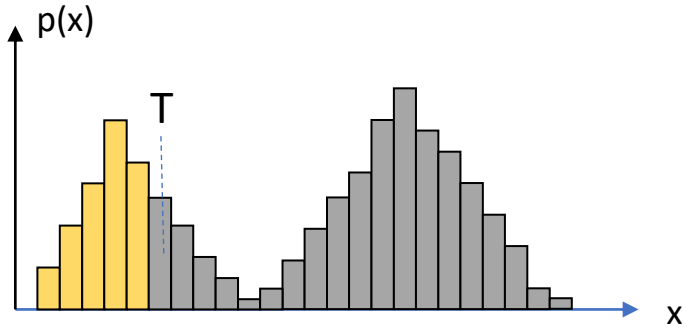
Logo, a variância combinada dos dois grupos deve ser baixa

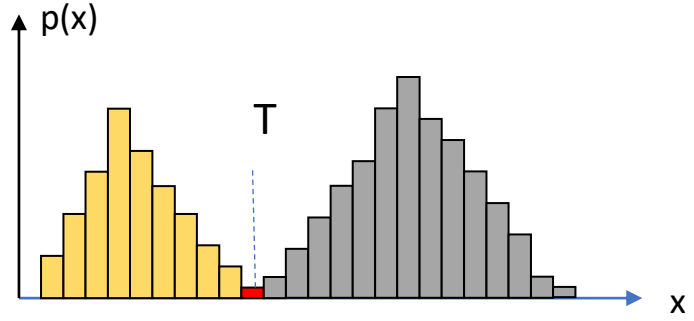
A variância combinada é dada pela soma (ponderada) das variâncias

$$S^2(T) = a_1(T)\ S_1^2(T) + a_2(T)\ S_2^2(T)$$

a= fator de ponderação







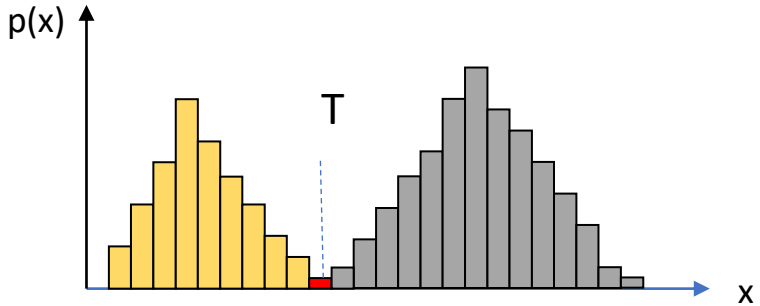
Variância (S) mínima

Busca-se o Limiar “T” que minimiza a variância combinada (dentro dos grupos).
Como “peso”, usa-se a soma das probabilidades das classes (a área).
Classe mais frequente “pesa” mais.

$$S^2(T) = w_1(T)\ S_1^2(T) + w_2(T)\ S_2^2(T)$$

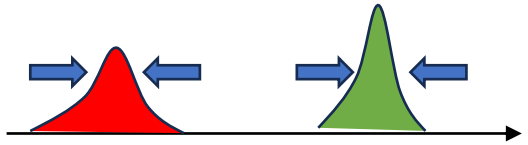
$$w_1(T) = \sum_{0}^{T-1} p(x)$$

$$w_2(T) = \sum_{T}^{255} p(x)$$



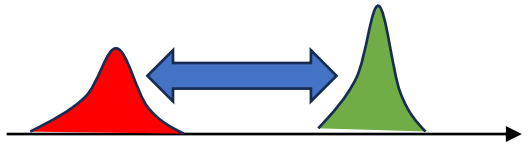
Minimizar a variância dentro das classes equivale a

Maximizar a variância entre classes:



$$S^2_{entre}(T) = S^2 - S^2_{dentro}(T)$$

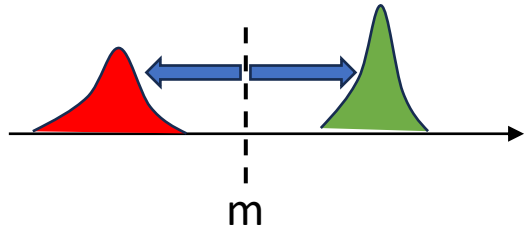
A variância entre classes pode ser calculada com ajuda da distância das médias dos grupos à média de toda a imagem



média da imagem

$$m = w_1 m_1 + w_2 m_2$$

$$S_{entre}{}^2 = w_1(m_1 - m)^2 + w_2(m_2 - m)^2$$



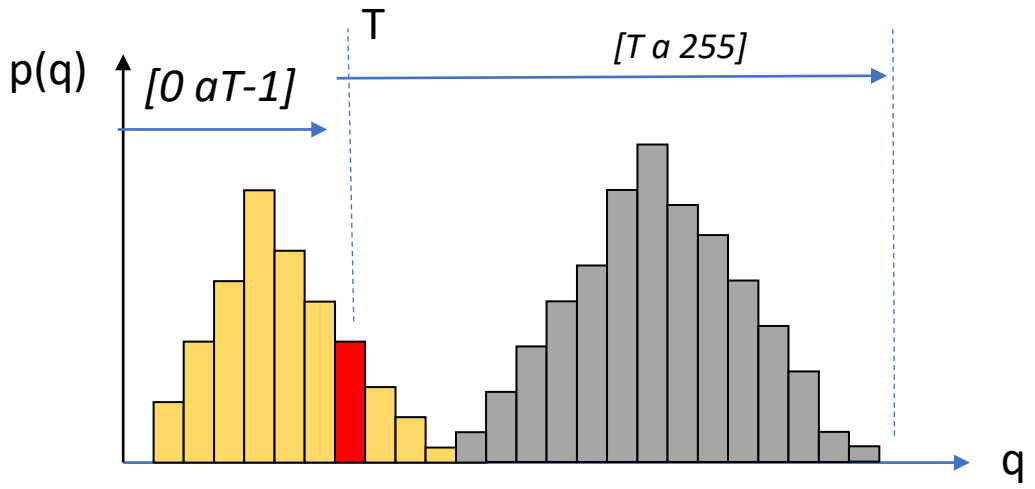
Ou, desenvolvendo e agrupando...

$$S_{entre}{}^2 = w_1 w_2 (m_1 - m_2)^2$$

OTSU

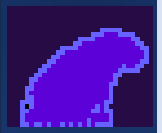
Como fazer?

Varrer todos os possíveis valores e calcular a variância para achar o máximo



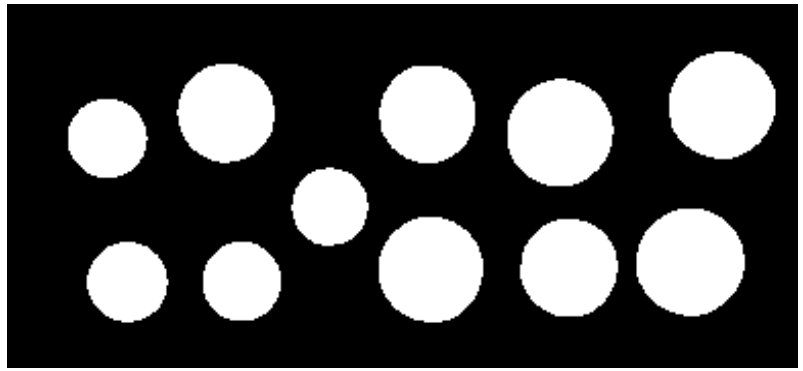
Consulte:Otsu N.,"*A Threshold Selection Method from Gray-level Histograms*", IEEE Transactions on Systems, Man and Cybernetics, v. SMC 9, no 1, pp.62-66, 1979.

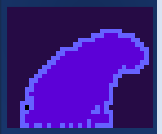
OTSU



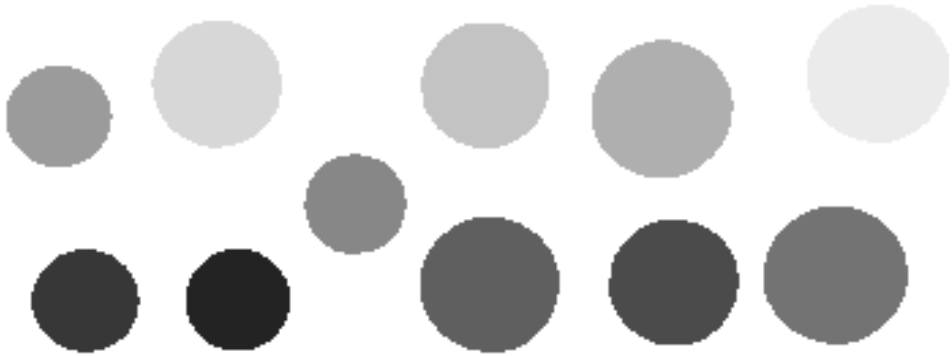
```
# Gaussian Blur (7x7)
im = cv2.GaussianBlur(I, (7, 7), 0)


#   threshold com OTSU
th, threshold = cv2.threshold(im, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
cv2_imshow(K)
print(th)  # veja o valor do limiar que foi calculado automaticamente
```





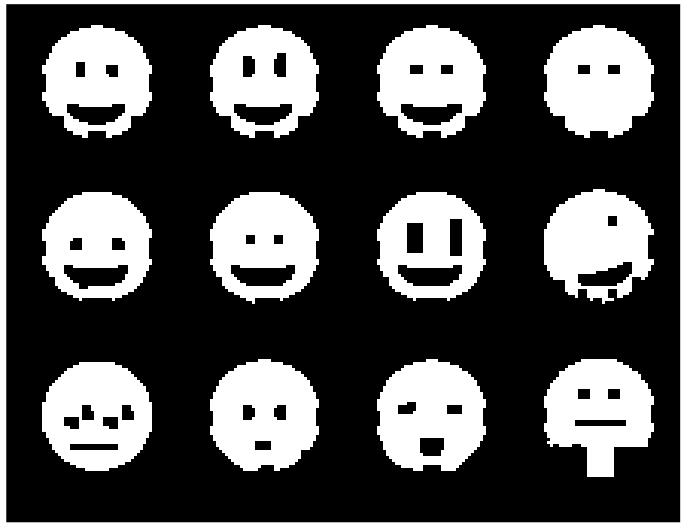
Componentes conexos

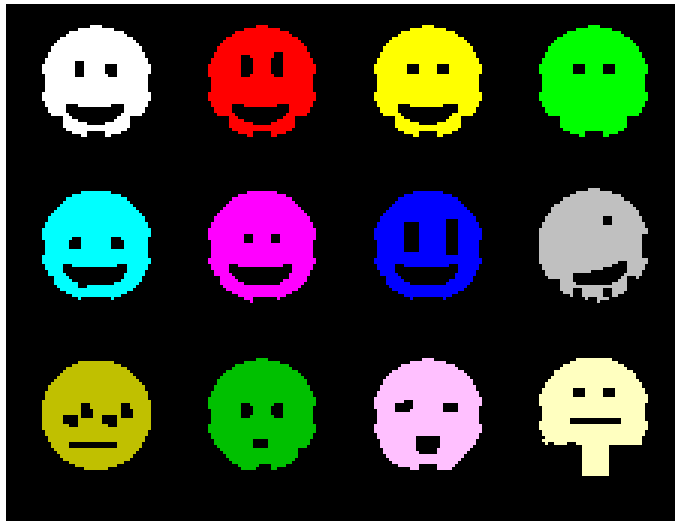


Componentes conexos

- Objetivo

Dada uma imagem contendo regiões conexas de mesma cor, identificar cada grupo de pixels (conexos) como uma região única.





Algoritmo: Componentes conexos

Um grupo conexo é um conjunto de pixels vizinhos (conectados) com a mesma propriedade. Por exemplo, com o mesmo valor digital =1.

Imagem binária

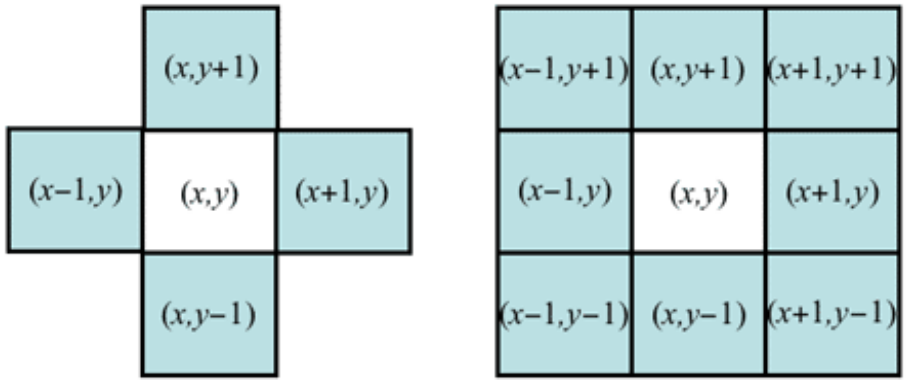
0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	1	0
0	1	1	0	0	0	0	1	0
0	1	1	0	0	0	0	1	0
0	0	0	0	1	1	0	0	0
0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Regiões conexas

0	0	0	0	0	0	0	0	0
0	1	1	0	0	3	3	3	0
0	1	1	0	0	0	0	3	0
0	1	1	0	0	0	0	3	0
0	0	0	0	2	2	0	0	0
0	0	0	0	2	2	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Conectividade

- Definir Conceito de conectividade
- Quando podemos considerar que dois pixels são “vizinhos”?
- Verificar a conectividade 4 ou 8 vizinhos



Vizinhança 4 Vizinhança 8

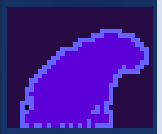
Dois pixels estão conectados se eles satisfazem uma relação de adjacência

0	0	0	0	0	0	0	0	0
0		D	0	0				0
0			0	0	0	0		0
0	A	B	0	0	0	0		0
0	0	0	0	C		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

A e B estão conectados

B e C não

.



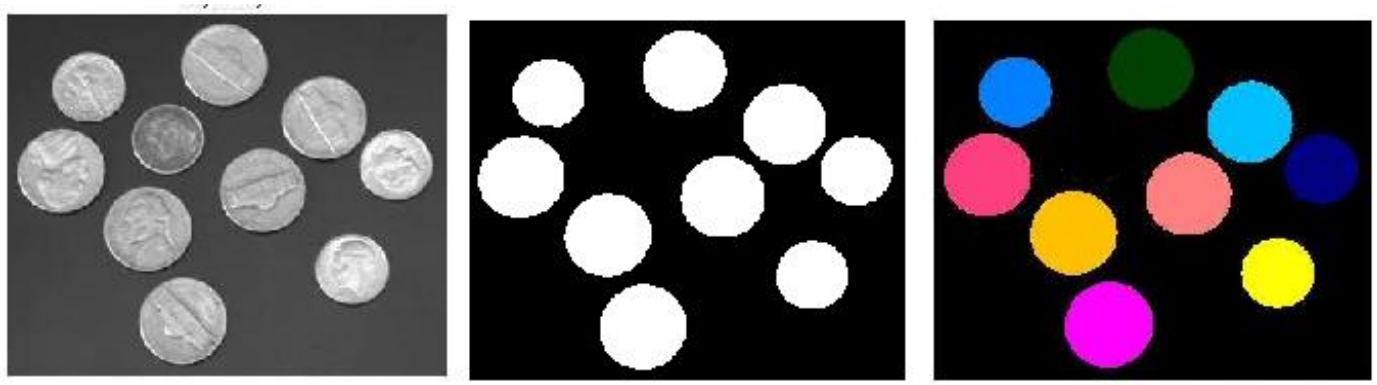
Quantos grupos conexos de Pixels “claros” Existem..
Considerando a vizinhança 4
E
Considerando a vizinhança 8 ?

0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	1	0
0	1	1	0	0	0	0	1	0
0	1	1	0	0	0	0	1	0
0	0	0	0	1	1	1	0	0
0	0	0	0	1	1	1	0	0
0	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Rotular pixels conexos

Problema: dada uma imagem com pixels rotulados com o mesmo valor e o fundo:

- Encontrar quantos grupos existem e
- Rotular todos os pixels de um mesmo grupo com o mesmo valor



Algoritmo

1) Numerar todos os pixels “ativos” começando no canto superior esquerdo até o canto inferior direito.

2) Analisar a vizinhança dos pixels numerados:

- se existe um pixel “ativo” com valor menor que o valor do pixel central, o pixel central adota esse menor valor.

3) Repetir isto até não ocorrer mais mudanças

4) Rotular os grupos resultantes com base no histograma.

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	1	0
0	0	1	0	0	1	0
0	0	0	0	1	1	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	0	2	0	0	0
0	3	0	4	0	5	0
0	6	7	8	0	9	0
0	0	10	0	0	11	0
0	0	0	0	12	13	0
0	0	0	0	0	0	0

Algoritmo

2) Analisar a vizinhança dos pixels numerados:

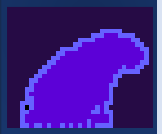
se existe um pixel “ativo” com valor menor que o valor do pixel central, o pixel central adota esse menor valor.

0	0	0	0	0	0	0
0	1	0	2	0	0	0
0	3	0	4	0	5	0
0	6	7	8	0	9	0
0	0	10	0	0	11	0
0	0	0	0	12	13	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	0	2	0	0	0
0	1	0	2	0	5	0
0	6	7	8	0	**9**	0
0	0	10	0	0	11	0
0	0	0	0	12	13	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	0	2	0	0	0
0	1	0	2	0	5	0
0	1	1	1	0	5	0
0	0	1	0	0	5	0
0	0	0	0	5	5	0
0	0	0	0	0	0	0

Algoritmo



1) Numerar todos os pixels “ativos” começando no canto superior esquerdo até o canto inferior direito.
2) Analisar a vizinhança dos pixels numerados:
se existe um pixel “ativo” com valor menor que o valor do pixel central, o pixel central adota esse menor valor.

3) Repetir isto até não ocorrer mais mudanças

.

0	0	0	0	0	0	0
0	1	0	2	0	0	0
0	1	0	2	0	5	0
0	1	1	1	0	5	0
0	0	1	0	0	5	0
0	0	0	0	5	5	0
0	0	0	0	0	0	0

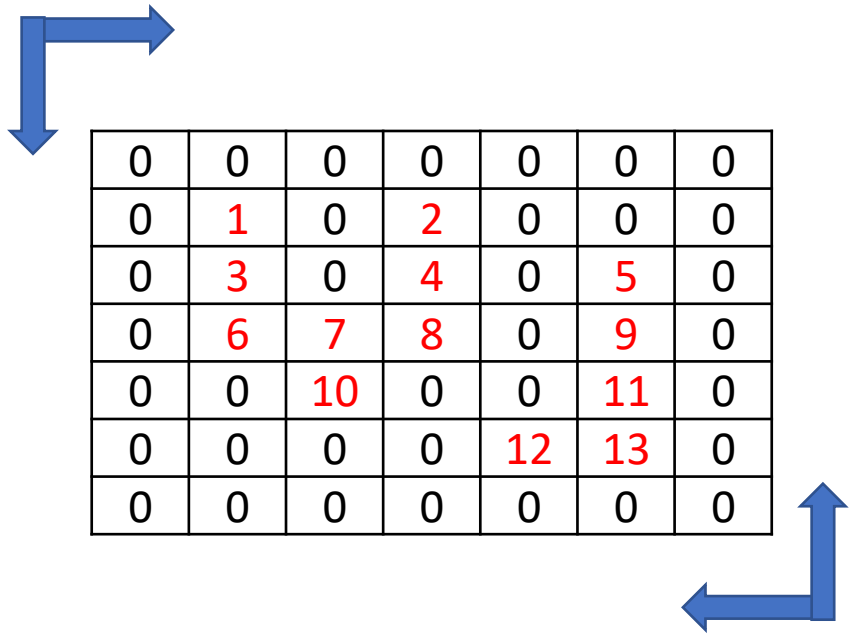
0	0	0	0	0	0	0
0	1	0	2	0	0	0
0	1	0	1	0	5	0
0	1	1	1	0	5	0
0	0	1	0	0	5	0
0	0	0	0	5	5	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	5	0
0	1	1	1	0	5	0
0	0	1	0	0	5	0
0	0	0	0	5	5	0
0	0	0	0	0	0	0

acelerar

- Alternando varreduras de : cima para baixo, esquerda para direita
- Baixo para cima, direita para esquerda

O processo pode ser mais rápido

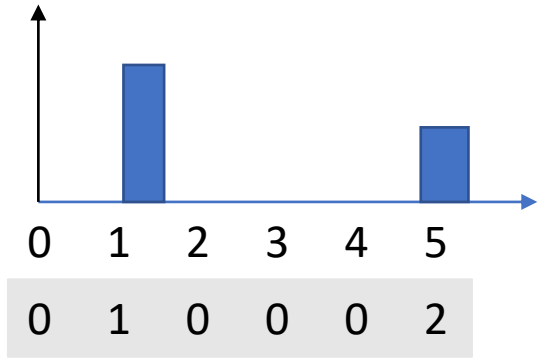


0	0	0	0	0	0	0
0	1	0	2	0	0	0
0	3	0	4	0	5	0
0	6	7	8	0	9	0
0	0	10	0	0	11	0
0	0	0	0	12	13	0
0	0	0	0	0	0	0

4) Rotular os grupos resultantes com base no histograma.

- Calcular o Histograma
- Numerar apenas as posições do histograma que possuem valores acima de zero.
- Aplicar esta nova tabela para mudar os números na imagem

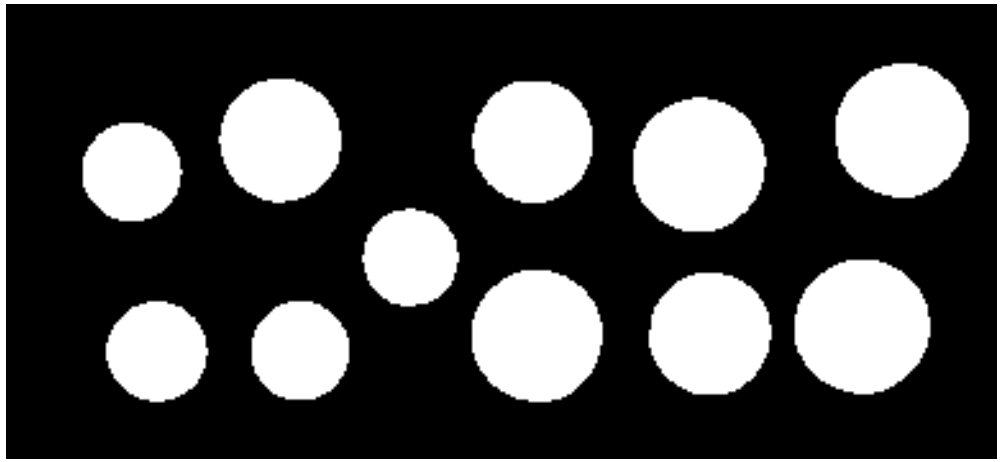
0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	5	0
0	1	1	1	0	5	0
0	0	1	0	0	5	0
0	0	0	0	5	5	0
0	0	0	0	0	0	0

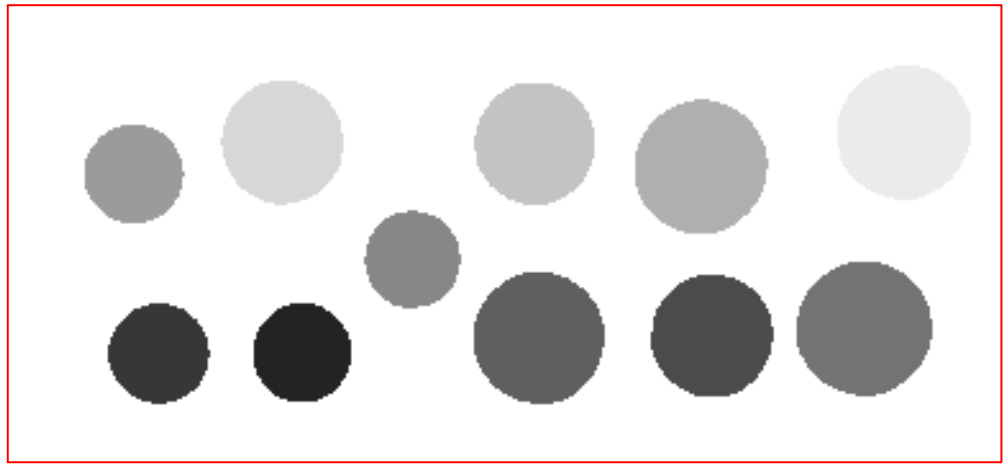


0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	2	0
0	1	1	1	0	2	0
0	0	1	0	0	2	0
0	0	0	0	2	2	0
0	0	0	0	0	0	0

OpenCV

```
ret, ICC = cv2.connectedComponents(K)
print(ret)  # quantas regiões achou?
cv2_imshow(255-ICC*20)
```





- OpenCV também pode calcular algumas propriedades das regiões como

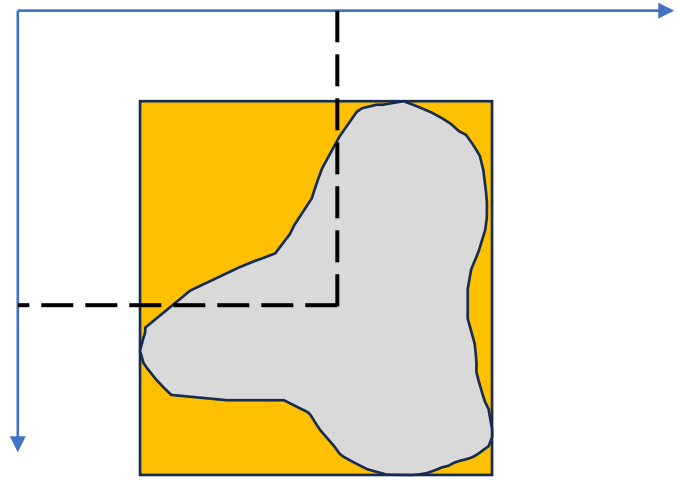
a) número de regiões

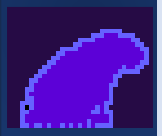
b) Coordenadas do canto superior esquerdo do retângulo envolvente

c) Dimensões do retângulo envolvente

d) Área

e) Coordenadas do centróide





```
Calcular as componentes conexas com suas propriedades
props = cv2.connectedComponentsWithStats(K, 4,  cv2.CV_32S)
```

- ```
  recuperar as informações em variáveis separadas
  (totalLabels, label_ids, valores, centroide) = props
  ```

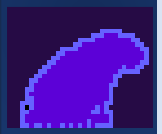
- totalLabels: número de regiões (a primeira é a imagem toda)
- Valores: tabela contendo: coordenadas do canto superior esquerdo, largura, altura do bounding box, e área [x0, y0, dx, dy]
- Centroide: coordenadas do centroide [Xc, Yc]
  ```

exemplo

- Desenhe o bounding box e o centroide de cada região (conected componente) identificada

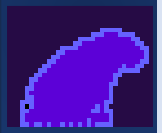
- Na tabela VALORES temos [x0, y0, dx, dy, área]
- Estas informações podem ser diretamente lidas usando índices de coluna (0,1,2,3,4), por exemplo, a coluna 4 tem a área...
- Ou com os índices de OpenCV equivalentes

```
cv2.CC_STAT_LEFT     =0    X0
cv2.CC_STAT_TOP      =1    Y0
cv2.CC_STAT_WIDTH    =2    largura
cv2.CC_STAT_HEIGHT   =3    altura
cv2.CC_STAT_AREA]    = 4   Area
```



```
L = np.zeros(img.shape, dtype="uint8") # crie imagem vazia com dimensões =imagem original
RGB
for i in range(1, totalLabels):   # variar as componentes I(regiões), desconsidere zero
    x1 = valores[i, cv2.CC_STAT_LEFT]     # ler coordenadas do bounding box
    y1 = valores[i, cv2.CC_STAT_TOP]
    w  = valores[i, cv2.CC_STAT_WIDTH]
    h  = valores[i, cv2.CC_STAT_HEIGHT]
    pt1 = (x1, y1)                # canto superior esquerdo
    pt2 = (x1+ w, y1+ h)          # calcular o canto inferior direito
    # desenhar o retângulo envolvente com função cv2.rectangle
    cv2.rectangle(L, pt1,pt2,     (0, 255, 0), 1)  # desenha retângulo verde, linha=1

    (X, Y) = centroid[i]  # recupera coordenadas do centroide
    cv2.circle(L, (int(X), int(Y)), 2, (0, 0, 255), -1)   # desenha cengtroide em vermelho
    area = values[i, cv2.CC_STAT_AREA] # recupera a Area   e mostra área e centroide
    print(area, X,Y)
cv2_imshow(L)  # mostre imagem final
```



- Usando estas informações

- Conte quantas moedas tem de cada valor!