3.5. Etiquetado de componentes conexas mediante códigos de cadena

El algoritmo de etiquetado de componentes conectados o CCL (etiquetado de componentes conectados) asigna un código de identificación único a cada grupo de píxeles conectados y similares en una imagen. El algoritmo utiliza la forma binaria $b(x, y)$ y analiza la relación entre los píxeles de cuatro vecinos (N4), los pixeles de ocho vecinos (N8) y, finalmente, los dos píxeles p y q tendrán un componente. Todos están conectados, considerando el frente o la parte posterior, y hay un camino de píxeles del mismo color entre ellos, entonces la ecuación (3.47) se cumple y S es la fracción de píxeles de la forma binaria b.

$$p\ conectado\ a\ q \Leftrightarrow \{s_i \in S | s_1 = p, s_{n+1} = q, s_{i+1} \in N(s_i), i = 1,2, \dots, n\} \qquad (3.47)$$

Debido a la importancia del algoritmo CCL, se han realizado muchos esfuerzos para mejorarlo y desarrollarlo.

Algoritmos de etiquetado:

- **De una pasada** (**one-scan**)
  Se recorre la imagen solo una vez. El acceso aleatorio y aleatorio a la estructura de datos que contiene la imagen, la señal dada y el problema de predicción en el proceso son todos problemas de este tipo de algoritmo, como el algoritmo de seguimiento de algoritmo propuesto por Chang en 2003.

  Estructuras de Datos

  Matriz de etiquetas (label): Misma dimensión que la imagen original, inicializada en 0.
  Etiqueta actual (next_label): Contador inicial en 1 para asignar nuevas etiquetas.
  Arreglo de equivalencias (equivalence_table): Un arreglo o mapa que resuelve fusiones de etiquetas (e.g., etiqueta provisional i equivalente a j). Se puede implementar con un arreglo donde equivalence[i] = i inicialmente, y se actualiza para apuntar a la raíz de un árbol.
  Lista de equivalencias pendientes: Para resolver fusiones después del escaneo inicial (opcional en implementaciones optimizadas).

Pasos del Algoritmo

1. **Inicialización**:
   - Crea la matriz de etiquetas vacía (todos 0).
   - next_label = 1.
   - Inicializa la tabla de equivalencias (cada etiqueta apunta a sí misma).
2. **Escaneo de la Imagen (Una Pasada)**:
   - Recorre cada píxel (i, j) de la imagen de fila en fila (i de 0 a M-1), y dentro de cada fila, de columna en columna (j de 0 a N-1).
   - Si el píxel original es 0 (fondo), asigna label[i][j] = 0 y continúa.
   - Si el píxel original es 1 (objeto):
     - Examina los vecinos relevantes (para 8-conectividad: arriba (i-1,j), arriba-izquierda (i-1,j-1), arriba-derecha (i-1,j+1), izquierda (i,j-1)).

        - Encuentra las etiquetas no cero entre estos vecinos. Si hay múltiples etiquetas diferentes, identifica la menor (o usa una convención para resolver conflictos).
        - Casos:
            - **No hay vecinos etiquetados**: Asigna una nueva etiqueta: label[i][j] = next_label, y next_label += 1. Actualiza equivalence[next_label-1] = next_label-1.
            - **Un vecino etiquetado**: Asigna la etiqueta de ese vecino a label[i][j]. (Si es de la fila anterior, verifica equivalencias previas).
            - **Múltiples vecinos con la misma etiqueta**: Asigna esa etiqueta común.
            - **Múltiples vecinos con etiquetas diferentes**: Asigna la etiqueta más pequeña (o la primera encontrada), y registra una equivalencia entre las etiquetas conflictivas en la tabla (e.g., une las raíces en union-find: equivalence[root_a] = root_b).
3. **Resolución de Equivalencias**:
    - Después del escaneo, itera sobre todas las entradas en la tabla de equivalencias para comprimir caminos (path compression) y asignar a cada etiqueta provisional su etiqueta final (raíz del árbol).
    - Opcionalmente, realiza una segunda pasada ligera solo sobre los píxeles etiquetados para reemplazar etiquetas provisionales por las finales, pero en la versión estricta de "una pasada", esto se integra o se hace en paralelo si es posible.
4. **Asignación de Etiquetas Finales**:
    - Renumera las etiquetas equivalentes para que sean consecutivas (e.g., etiqueta 1, 2, ..., k donde k es el número de componentes únicos).
    - La matriz de etiquetas ahora representa cada componente conexa con un ID único.

Ejemplo Simple

Supongamos una imagen binaria 3x3:

0 1 1
1 1 0
0 1 0

- Escaneo:
    - (0,1): Nuevo, label=1.
    - (0,2): Vecino izquierda=1, label=1.
    - (1,0): Nuevo (no vecinos), label=2.

    - (1,1): Vecinos arriba=1, izquierda=2 → Conflicto, asigna min(1,2)=1, une 1 y 2 (equivalence[2]=1).
    - (1,2): Fondo.
    - (2,1): Vecino arriba=1, label=1.
- Resolución: Etiqueta 2 se fusiona a 1. Componente final: Todo el objeto tiene label=1.

Ventajas y Consideraciones

- **Eficiencia**: Una sola pasada principal, ideal para imágenes grandes.
- **Implementación**: En código (e.g., Python con NumPy), usa estructuras como scipy.ndimage.label para una versión built-in, pero el algoritmo manual usa bucles anidados.
- **Variantes**:
    - 4-conectividad: Solo vecinos arriba e izquierda.
    - Optimizaciones: Usa run-length encoding para saltar fondos.
- **Limitaciones**: Puede requerir memoria extra para la tabla de equivalencias en imágenes con muchos componentes.

- **De múltiples pasadas** (**multi-scan**)
  La imagen se barre de muchas maneras (invertida, de arriba hacia abajo, de izquierda a derecha, de abajo hacia arriba, de derecha a izquierda) y siempre se utiliza mucha memoria. El tiempo de procesamiento depende de la disposición de los píxeles de cada imagen, por lo que no es posible establecer el tiempo de procesamiento en todos los casos. El software y el hardware empleados para múltiples pasadas son fáciles de instalar. Estos incluyen los servicios prestados por Haralic en 1981 y Suzuki en 2000-2003.

Algoritmo de Etiquetado de Componentes Conexas de Múltiples Pasadas

(Multi-Pass Connected Component Labeling)

Este es un enfoque **clásico y más simple** (aunque menos eficiente que el de una pasada) para etiquetar componentes conexas en imágenes binarias. Se basa en **varias pasadas completas** sobre la imagen hasta que no haya más cambios en las etiquetas.

Es especialmente útil para entender los conceptos básicos de conectividad y propagación de etiquetas, y se usa frecuentemente en explicaciones educativas o implementaciones didácticas.

Suposiciones

- Imagen binaria: 1 = objeto (foreground), 0 = fondo (background).
- Conectividad: **4-conectividad** (arriba, abajo, izquierda, derecha) o **8-conectividad** (más diagonal).
- Etiquetas: enteros positivos, empezando desde 1.
- La imagen se representa como matriz 2D img[M][N].

Estructuras de Datos

- label[M][N]: Matriz de etiquetas, inicializada en 0.
- next_label: Contador de etiquetas (inicia en 1).
- (Opcional) Lista de equivalencias si se fusionan componentes.

Pasos del Algoritmo (Múltiples Pasadas)

Fase 1: Primera pasada (asignación provisional de etiquetas)

Escanea la imagen **de arriba a abajo, izquierda a derecha**.

Para cada píxel (i,j) con img[i][j] == 1:

1. Mira los **vecinos ya procesados**:
    - Para **4-conectividad**: (i-1,j) (arriba), (i,j-1) (izquierda).
    - Para **8-conectividad**: además (i-1,j-1), (i-1,j+1).
2. **Casos**:
    - **Ningún vecino etiquetado** → Asigna nueva etiqueta: label[i][j] = next_label++
    - **Un vecino etiquetado** → Copia su etiqueta: label[i][j] = label_vecino
    - **Varios vecinos con etiquetas distintas** → Asigna la **menor** de ellas (o cualquiera), y **registra equivalencia** entre todas las etiquetas encontradas.

Ejemplo: si arriba tiene 3 e izquierda tiene 5 ⟶ asigna 3, y marca 5 ≡ 3.

Fase 2: Resolución de equivalencias (opcional, pero común)

- Usa una tabla de equivalencias (o **Union-Find**) para fusionar etiquetas que pertenecen al mismo componente.

- Al final, cada etiqueta provisional se mapea a una **etiqueta raíz** (representante del componente).

Fase 3: Pasadas de propagación (hasta estabilización)

Repite lo siguiente **hasta que no haya cambios**:

Pasada hacia abajo y derecha (forward pass):

- Recorre de (0,0) a (M-1,N-1)
- Para cada píxel objeto:
    - Actualiza label[i][j] con el **mínimo** de:
        - su etiqueta actual
        - etiquetas de vecinos ya procesados (arriba, izquierda, etc.)

Pasada hacia arriba y izquierda (backward pass):

- Recorre de (M-1,N-1) a (0,0)
- Actualiza con **mínimo** de vecinos ya procesados en esta dirección (abajo, derecha, etc.)

Se alternan estas dos pasadas hasta que **ningún píxel cambie de etiqueta**.

Fase 4: Renumeración final (opcional)

- Asigna etiquetas consecutivas: 1, 2, 3, ... a cada componente único.
- Actualiza la matriz label reemplazando raíces por nuevas etiquetas.

Ejemplo Práctico (4-conectividad)

Imagen binaria 4x5:

text

```
0 1 1 0 1
1 1 0 0 1
0 1 0 1 0
1 0 1 0 1
```

Primera pasada (izq→der, arr→aba):

Pos	Vecinos	Acción	Etiqueta
(0,1)	ninguno	nueva	1
(0,2)	izq=1	copia	1
(0,4)	ninguno	nueva	2
(1,0)	ninguno	nueva	3
(1,1)	arriba=1, izq=3	min(1,3) → 1, 3≡1	1
(1,4)	arriba=2	copia	2
(2,1)	arriba=1	copia	1
(2,3)	ninguno	nueva	4
(3,0)	ninguno	nueva	5
(3,2)	ninguno	nueva	6
(3,4)	ninguno	nueva	7

Matriz de etiquetas provisional:

text

```
0 1 1 0 2
3 1 0 0 2
0 1 0 4 0
5 0 6 0 7
```

Equivalencias: 3 → 1

Pasadas de propagación (hasta convergencia)

Se realizan **múltiples pasadas** alternando dirección, propagando el **mínimo**:

Después de varias iteraciones, todas las etiquetas se unifican por propagación:

text

```
0 1 1 0 2
1 1 0 0 2
0 1 0 1 0
1 0 1 0 2
```

Componentes finales:

- Etiqueta 1: gran componente en el centro-izquierda.
- Etiqueta 2: componente en la derecha.

Ventajas y Desventajas

Aspecto	Detalle
Ventajas	- Muy fácil de entender e implementar. - No requiere estructuras complejas como Union-Find en la primera versión. - Garantiza convergencia.
Desventajas	- **Lento**: puede requerir hasta O(M+N) pasadas en el peor caso (cadenas largas). - Ineficiente para imágenes grandes. - Requiere múltiples accesos a memoria.

Comparación con el Algoritmo de Una Pasada

Característica	Una Pasada	Múltiples Pasadas
Nº de pasadas	1 principal + resolución	1 + muchas propagaciones
Estructura	Union-Find, tabla equivalencias	Simple, solo min()
Eficiencia	O(n)	O(n × (M+N)) peor caso
Memoria	+tabla equivalencias	Solo matriz etiquetas
Complejidad implementación	Media-Alta	Baja

Pseudocódigo (Múltiples Pasadas, 4-conectividad)

python

```python
def multi_pass_labeling(binary_image):
    M, N = len(binary_image), len(binary_image[0])
    label = [[0]*N for _ in range(M)]
    next_label = 1

    # Primera pasada: asignar etiquetas provisionales
    for i in range(M):
        for j in range(N):
            if binary_image[i][j] == 1:
                neighbors = []
```

```
            if i > 0: neighbors.append(label[i-1][j])
            if j > 0: neighbors.append(label[i][j-1])
            neighbors = [l for l in neighbors if l > 0]

            if not neighbors:
                label[i][j] = next_label
                next_label += 1
            else:
                label[i][j] = min(neighbors)

# Múltiples pasadas hasta estabilización
changed = True
while changed:
    changed = False

    # Forward pass (arriba→abajo, izq→der)
    for i in range(M):
        for j in range(N):
            if binary_image[i][j] == 1:
                min_val = label[i][j]
                if i > 0: min_val = min(min_val, label[i-1][j])
                if j > 0: min_val = min(min_val, label[i][j-1])
                if min_val < label[i][j]:
                    label[i][j] = min_val
                    changed = True

    # Backward pass (abajo→arriba, der→izq)
    for i in range(M-1, -1, -1):
        for j in range(N-1, -1, -1):
            if binary_image[i][j] == 1:
                min_val = label[i][j]
                if i < M-1: min_val = min(min_val, label[i+1][j])
                if j < N-1: min_val = min(min_val, label[i][j+1])
                if min_val < label[i][j]:
                    label[i][j] = min_val
                    changed = True
```

```
    # Renumeración final (opcional)
    # ... (mapear etiquetas a 1,2,3...)

    return label
```

Conclusión

El **algoritmo de múltiples pasadas** es:

- **Didáctico y robusto**
- **Ineficiente en tiempo**
- Ideal para **prototipado, enseñanza o imágenes pequeñas**

Para aplicaciones reales, se prefiere el **algoritmo de una pasada con Union-Find**.

- **De dos pasadas** (**doble-scan**)
  La imagen se escaneará dos veces. El primero puede escribir en poco tiempo, el segundo puede colocar múltiples puntos en cada píxel. El acceso a la memoria normalmente utiliza una o más tablas para almacenar correspondencias entre diferentes etiquetas que están asignadas temporalmente a la misma ubicación. Las estructuras de datos (desde matrices de adyacencia, estructuras de n-tuplas hasta vectores y tablas relacionales) se utilizan para almacenar estas similitudes entre objetos de la imagen. Estos algoritmos son más difíciles de implementar en hardware que los algoritmos alternativos, sus tiempos de integración son elevados y dependiendo de la complejidad de la imagen a procesar pueden presentar superposiciones y caídas de señales.

Algoritmo de Etiquetado de Componentes Conexas de Doble Pasada

(Two-Pass Connected Component Labeling)

Este es el **algoritmo estándar más eficiente y ampliamente utilizado** en procesamiento de imágenes para etiquetar componentes conexas en imágenes binarias.

Se llama **"doble pasada"** porque:

1. **Primera pasada**: Escaneo de la imagen (asignación provisional de etiquetas + detección de equivalencias).
2. **Segunda pasada**: Reemplazo de etiquetas provisionales por las finales (usando resolución de equivalencias).

Ventaja clave: Solo **dos pasadas completas** sobre la imagen, independientemente del tamaño o forma de los componentes. **Eficiencia**: O(n) en tiempo y espacio (donde n = M×N).

Suposiciones

- Imagen binaria: 1 = objeto, 0 = fondo.
- Conectividad: **8-conectividad** (más común), pero adaptable a 4-conectividad.
- Vecinos considerados en primera pasada: (i-1,j-1), (i-1,j), (i-1,j+1), (i,j-1) → los 4 vecinos **superiores e izquierdos**.
- Etiquetas: enteros positivos desde 1.

Estructuras de Datos

Estructura	Descripción
label[M][N]	Matriz de etiquetas (inicialmente 0)
next_label	Contador de etiquetas provisionales (1, 2, ...)
parent[]	Arreglo para **Union-Find** (resolución de equivalencias)
rank[]	Opcional: para optimizar unión por rango

Algoritmo Paso a Paso

Primera Pasada: Asignación Provisional + Unión de Etiquetas

Recorre la imagen **de arriba a abajo, de izquierda a derecha**.

Para cada píxel (i,j) con img[i][j] == 1:

python

```
vecinos = []
if i > 0: vecinos.append(label[i-1][j])           # arriba
if i > 0 and j > 0: vecinos.append(label[i-1][j-1])  # arriba-izq
if i > 0 and j < N-1: vecinos.append(label[i-1][j+1]) # arriba-der
if j > 0: vecinos.append(label[i][j-1])           # izquierda
```

Filtra solo etiquetas **no cero** (píxeles ya etiquetados):

python

```
etiquetas_vecinas = {l for l in vecinos if l > 0}
```

Casos:

Caso	Acción
No hay etiquetas vecinas	label[i][j] = next_label parent[next_label] = next_label next_label += 1
Una sola etiqueta vecina	label[i][j] = etiqueta_vecina
Varias etiquetas distintas	Asigna la **menor**: label[i][j] = min(etiquetas_vecinas) Y **une todas** con union(a, b)

Union-Find (con Path Compression y Union by Rank)

python

```
def find(x):
    if parent[x] != x:
        parent[x] = find(parent[x])  # path compression
    return parent[x]

def union(x, y):
    px, py = find(x), find(y)
    if px != py:
        if rank[px] > rank[py]:
            parent[py] = px
        elif rank[px] < rank[py]:
            parent[px] = py
        else:
            parent[py] = px
            rank[px] += 1
```

Segunda Pasada: Reemplazo por Etiqueta Final

Recorre **toda la imagen nuevamente**:

python

```
for i in range(M):
    for j in range(N):
        if label[i][j] > 0:
```

```
            label[i][j] = find(label[i][j])  # etiqueta raíz
```

(Opcional) Renumeración Consecutiva

Para etiquetas 1, 2, 3, ...:

python

```
unique_labels = sorted(set(find(l) for row in label for l in row if l > 0))
map_old_to_new = {old: new for new, old in enumerate(unique_labels, 1)}

for i in range(M):
    for j in range(N):
        if label[i][j] > 0:
            label[i][j] = map_old_to_new[label[i][j]]
```

Ejemplo Completo (8-conectividad)

Imagen binaria 4x6:

text

```
0 1 1 0 1 1
1 1 0 0 0 1
0 1 0 1 0 0
1 0 1 0 1 0
```

Primera pasada:

(i,j)	Vecinos etiquetados	Acción	label[i][j]
(0,1)	—	nueva	1
(0,2)	1	copia	1
(0,4)	—	nueva	2
(0,5)	2	copia	2
(1,0)	—	nueva	3
(1,1)	1,3	min=1, union(1,3)	1
(1,5)	2	copia	2
(2,1)	1	copia	1
(2,3)	—	nueva	4

(i,j)	Vecinos etiquetados	Acción	label[i][j]
(3,0)	—	nueva	5
(3,2)	1	copia	1
(3,4)	—	nueva	6

Matriz provisional:

text

```
0 1 1 0 2 2
3 1 0 0 0 2
0 1 0 4 0 0
5 0 1 0 6 0
```

Equivalencias detectadas: union(1,3)

Segunda pasada (find):

- find(3) → 1
- find(5), find(4), find(6) quedan como están.

Resultado final (sin renumerar):

text

```
0 1 1 0 2 2
1 1 0 0 0 2
0 1 0 4 0 0
5 0 1 0 6 0
```

Con renumeración: 1→1, 2→2, 4→3, 5→4, 6→5 → etiquetas 1 a 5.

Pseudocódigo Completo

```
def two_pass_labeling(binary_img):
    M, N = len(binary_img), len(binary_img[0])
    label = [[0] * N for _ in range(M)]
    parent = []
    rank = []
```

```
next_label = 1

# --- PRIMERA PASADA ---
for i in range(M):
    for j in range(N):
        if binary_img[i][j] == 0:
            continue

        neighbors = []
        if i > 0: neighbors.append(label[i-1][j])
        if i > 0 and j > 0: neighbors.append(label[i-1][j-1])
        if i > 0 and j < N-1: neighbors.append(label[i-1][j+1])
        if j > 0: neighbors.append(label[i][j-1])

        labeled_neighbors = [l for l in neighbors if l > 0]

        if not labeled_neighbors:
            label[i][j] = next_label
            parent.append(next_label)
            rank.append(0)
            next_label += 1
        else:
            min_label = min(labeled_neighbors)
            label[i][j] = min_label
            for other in labeled_neighbors:
                if other != min_label:
                    union(min_label, other, parent, rank)

# --- SEGUNDA PASADA ---
for i in range(M):
    for j in range(N):
        if label[i][j] > 0:
            label[i][j] = find(label[i][j], parent)

# --- Renumeración ---
roots = {find(l, parent) for row in label for l in row if l > 0}
```

```
    root_to_id = {root: idx+1 for idx, root in enumerate(sorted(roots))}
    for i in range(M):
        for j in range(N):
            if label[i][j] > 0:
                label[i][j] = root_to_id[find(label[i][j], parent)]

    return label
```

Comparación con Otros Algoritmos

Algoritmo	Nº Pasadas	Eficiencia	Memoria Extra	Complejidad
Múltiples pasadas	Hasta M+N	Baja	Solo matriz	Fácil
Doble pasada	**2**	**Alta**	parent[], rank[]	Media
Una pasada (optimizada)	1 + resolución	Muy alta	Union-Find	Alta

Nota: El de "doble pasada" **es** el estándar de "una pasada + resolución", pero se llama **two-pass** porque técnicamente hay dos escaneos completos.

Ventajas

- Solo **2 pasadas** → predecible y rápido.
- Escalable a imágenes grandes.
- Base de bibliotecas como **OpenCV**, **scikit-image**, **SciPy**.

Conclusión

El **algoritmo de doble pasada** es el **estándar de oro** para etiquetado de componentes conexas:

"Asigna etiquetas provisionales en una pasada, resuelve equivalencias con Union-Find, y corrige en una segunda pasada."

Algoritmo de etiquetado de regiones conexas

Para etiquetar, se asigna un símbolo único (etiqueta) a cada parte conectada de la imagen según la conexión correspondiente.

Algoritmo de Rosenfeld y Pfaltz:

1. Propagar las etiquetas
2. Buscar clases de equivalencias
3. Asignar etiquetas empleando clases de equivalencias

La propagación de etiquetas se define para $N_4(x)$ como se muestra en la figura 3.52

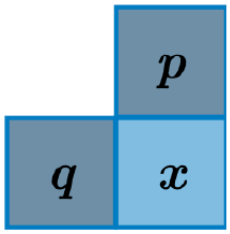

Figura 3.52 Propagación de etiquetas para $N_4(x)$

En código de C++ o Java

```
for(int y=2; y<=N; y++) {
        for(int x=2; x<=M; x++) {
                if(b[y][x]==1) {
                        lp = b[y-1][x];
                        lq = b[y][x-1];
                        if(lp==0 && lq==0) {
                                etiqueta++;
                                lx = etiqueta;
                                }
                        else
                        if((lp!=lq) && (lp!=0) && (lq!=0)) {
                                registrarEquivalencia(lp, lq);
                                lx = lq;
                                }
                        else
                        if(lq!=0) {
                                lx = lq;
                                }
                        else
                        if(lp!=0) {
                                lx = lp;
                                }
                                b[y][x] = lx;
                        }
                }
        }
```

La distribución de puntos de forma *b* de tamaño *M* × *N*, donde *lp*, *lq* y *lx* son las letras asignadas a *p*, *q* y *x*. Por defecto, etiqueta = 0. La Figura 3.53 muestra la secuencia de métodos para propagar puntos para vincular campos.

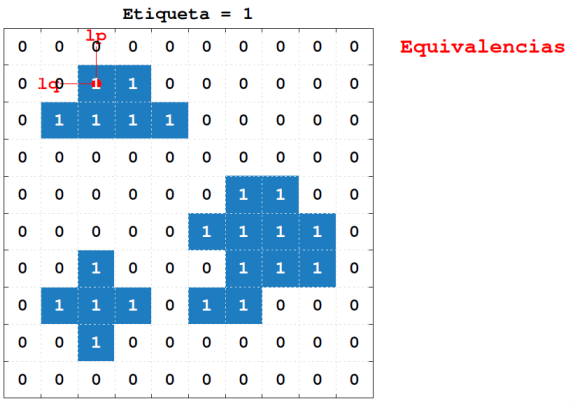

Figura 3.53 Secuencia de pasos del algoritmo.

1) Construya la matriz binaria *B* conectadas de manera similar, asegurándose de que sean consistentes y fáciles de usar.

$$B = B_0 \cup B_0^T \cup I_n \tag{3.48}$$

donde B_0 es la matriz de equivalencias original, *T* es la transpuesta, e I_n es una matriz identidad de tamaño *n* × *n*.

2) Calcule la matriz B^+ empleando el algoritmo de Warshall.

3) Defina la clase correspondiente. Si $B^+(i,j) = 1$, entonces el símbolo correspondiente a j corresponde al símbolo correspondiente a i. Como se muestra en la figura 3.54.

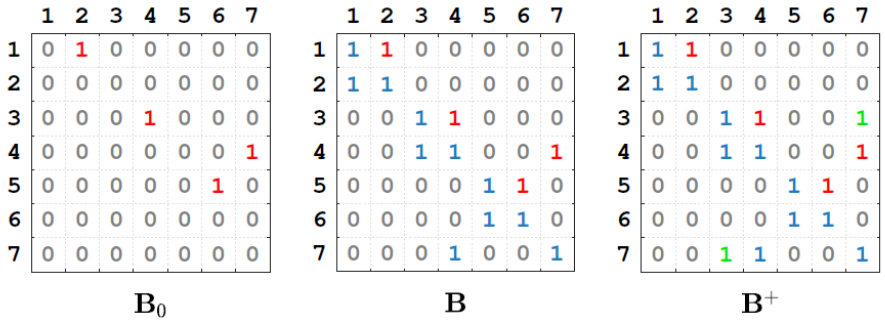

Figura 3.54 Matrices de relación de equivalencia.

4) Construya una segunda pasada sobre la imagen utilizando las etiquetas publicadas, reemplazando cada etiqueta con una etiqueta asignada a la misma categoría, como se muestra en la Figura 3.55

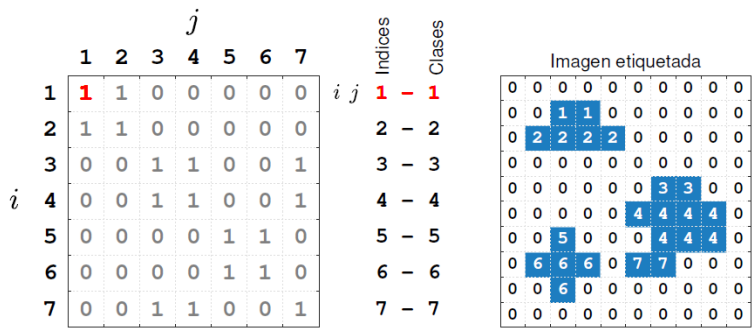

Figura 3.55 Secuencia de pasos para asignar clases de equivalencia.

El ejemplo del algoritmo se muestra en la figura 3.56

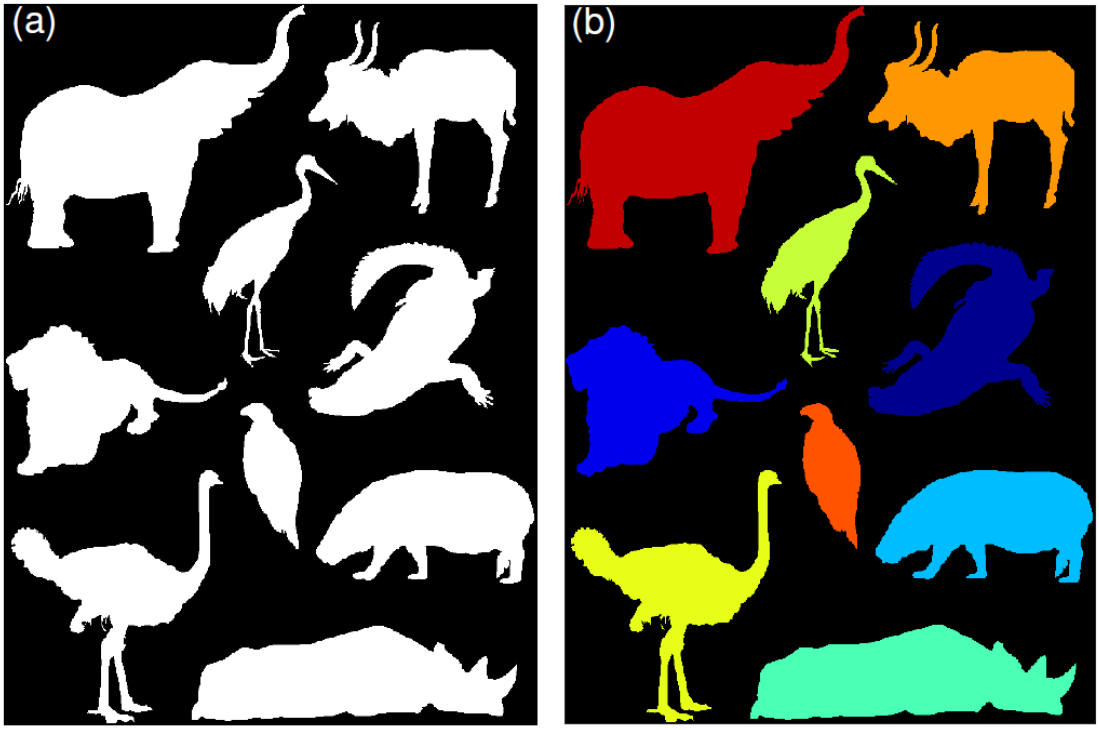

Figura 3.56 (**a**) Imagen binaria, (**b**) Etiquetado de componentes conexas.