5 Etichetarea componentelor conexe

5.1 Introducere

În această lucrare de laborator se vor prezenta algoritmi pentru etichetarea obiectelor distincte din imagini alb-negru. Ca rezultat, fiecare obiect va avea un număr unic. Acest număr, numit etichetă, va putea fi folosit pentru analiza fiecărui obiect în parte.

5.2 Fundamente teoretice

Vom prezenta doi algoritmi pentru etichetare. Intrarea acestor algoritmi este imaginea binară, iar ieșirea este o matrice de etichete, de aceeași dimensiune ca imaginea de intrare. Această matrice trebuie să aibă celula de un tip numeric capabil să rețină numărul total de etichete.

În imaginea binară de intrare, obiectele sunt reprezentate ca și componente conexe de culoare neagră (0), iar fundalul are culoarea albă (255). Pentru a defini noțiunea de componentă conexă, trebuie să definim tipurile de vecinătate.

Vecinătatea de 4 a unei poziții **(i,j)** este definită ca pozițiile:

$$N_4(i,j)=\{(i-1,j),\ (i,j-1),\ (i+1,j),\ (i,j+1)\},$$

i.e. vecinii de sus, stânga, jos și dreapta.

Vecinătatea de 8 conține toate pozițiile care diferă de poziția curentă cu maxim 1 unitate pe o una sau ambele coordonate:

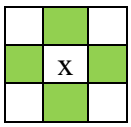
$$N_8(i,j) = \{(k,l) \mid |k-i|\leq 1,\ |l-j|\leq 1,\ (k,l)\neq(i,j)\},$$

Deci include vecinătatea de 4 și vecinii de pe diagonale.

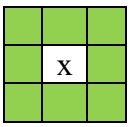
La parcurgerea imaginii într-o direcție oarecare, putem defini vecinii anteriori raportat la această direcție. Vecinii anteriori ai unei parcurgeri sus-jos, stânga-dreapta sunt:

$$Np(i,j)=\{(i,j-1),\ (i-1,j-1),\ (i-1,j),\ (i-1,j+1)\}.$$

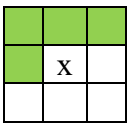
Aceste definiții sunt ilustrate în figura de mai jos.



a) Vecinătate de 4



b) Vecinătate de 8



c) Vecinii anteriori

Vom defini un graf format de imaginea binară. Vârfurile grafului sunt pozițiile pixelilor obiect, iar vecinătatea dintre pixeli reprezintă muchiile grafului. Două vârfuri sunt vecine dacă unul dintre ele aparține vecinătății celuilalt. Vom utiliza vecinătatea de 4 sau de 8, astfel că graful generat este neorientat. O componentă conexă este o mulțime de vârfuri pentru care există o cale între oricare două elemente ale acestei mulțimi.

5.2.1 Algoritm 1 – Traversarea în lățime

Vom începe cu o metodă directă pentru etichetare, care se bazează pe traversarea în lățime a grafului format de imaginea binară. Primul pas este inițializarea matricei de etichete cu valoarea zero pentru toți pixelii, indicând faptul că inițial totul este neetichetat. Apoi, algoritmul va căuta un pixel de tip obiect care este neetichetat. Dacă acest punct este găsit, el va primi o etichetă nouă, pe care o va propaga vecinilor lui. Vom repeta acest proces până când toți pixelii obiect vor primi o etichetă.

Algoritmul este descris de următorul pseudocod:

```
label = 0
labels = zeros(height, width)   //matrice height x width, cu valori 0
for i = 0:height-1
    for j = 0:width-1
        if img(i,j)==0 and labels(i,j)==0
            label++
            Q = queue()
            labels(i,j) = label
            Q.push( (i,j) )
            while Q not empty
                q = Q.pop()
                for each neighbor in N8(q)
                    if img(neighbor)==0 and labels(neighbor)==0
                        labels(neighbor) = label
                        Q.push( neighbor )
```

Algoritm 1 – Traversare în lățime pentru etichetarea componentelor conexe

Structura de date de tip coadă menține lista punctelor care trebuie etichetate cu eticheta curentă ”label”. Deoarece este o structură FIFO, se va obține traversarea în lățime. Vom marca nodurile vizitate setând eticheta pentru poziția lor, în matricea de etichete. Dacă structura de date se schimbă într-o stivă, se va obține o traversare în adâncime.

5.2.2 Algoritm 2 – Două treceri cu clase de echivalență

Etichetarea poate fi realizată prin două parcurgeri liniare pe imagine, plus o procesare suplimentară pe un graf mult mai mic. Această abordare folosește mai puțină memorie. Deoarece în algoritmul anterior aveam nevoie de memorarea unei liste de puncte, în cazul în care o componentă conexă este foarte mare dimensiunea listei poate fi comparabilă cu dimensiunea imaginii.

Algoritmul al doilea va face o primă parcurgere și va genera etichete inițiale pentru pixelii obiect. Pentru fiecare pixel vom lua în considerare pixelii deja vizitați și etichetați, deci vom folosi vecinătatea pixelilor anteriori definită mai sus, N_p. După inspectarea etichetelor pixelilor deja vizitați, avem următoarele cazuri:

- Dacă nu avem vecin anterior etichetat, vom crea o etichetă nouă
- Dacă avem vecini etichetați, vom selecta minimul acestor etichete (notat cu x). După aceea, vom marca fiecare etichetă y din vecinătate, diferită de x, ca echivalentă cu x.

Vom asigna eticheta găsită în pasul anterior pentru poziția curentă. După prima trecere, există câte o etichetă pentru fiecare poziție din imagine. Totuși, vor exista etichete diferite pentru același obiect, care sunt etichete echivalente, deci va trebui să fie înlocuite cu o etichetă nouă, care reprezintă o clasă de echivalență.

Relațiile de echivalență definesc un graf neorientat, având ca noduri etichetele inițiale. Acest graf este de obicei mult mai mic decât graful inițial al pixelilor obiect, deoarece numărul de noduri este numărul de etichete generate la prima parcurgere. Muchiile grafului sunt relațiile de echivalență. Putem aplica algoritmul 1 pe acest graf mai mic, pentru a obține o nouă listă de etichete. Toate etichetele echivalente cu 1 se re-etichetează cu 1, următoarea componentă conexă ne-echivalentă cu 1 se re-etichetează cu 2, și așa mai departe. O nouă trecere prin matricea de etichete inițiale va realiza re-etichetarea.

```
label = 0
labels = zeros(height, width)
vector<vector<int>> edges(1000)
for i = 0:height-1
    for j = 0:width-1
         if img(i,j)==0 and labels(i,j)==0
              L = vector()
              for each neighbor in Np(i,j)
                   if labels(neighbor)>0
                        L.push_back(labels(neighbor))
              if L.size() == 0        //assign new label
                   label++
                   labels(i,j) = label
              else                    //assign smallest neighbor
                   x = min(L)
                   labels(i,j) = x
                   for each y from L
                      if (y <> x)
                        edges[x].push_back(y)
                        edges[y].push_back(x)

newlabel = 0
newlabels = zeros(label+1)   //sir de zero, de dimensiune label+1
for i = 1:label
    if newlabels[i]==0
         newlabel++
         Q = queue()
         newlabels[i] = newlabel
         Q.push( i )
         while Q not empty
              x = Q.pop()
              for each y in edges[x]
                   if newlabels[y] == 0
                        newlabels[y] = newlabel
                        Q.push( y )


for i = 0:height-1
    for j = 0:width-1
         labels(i,j) = newlabels[labels(i,j)]
```

Algoritm 2 – Etichetarea componentelor conexe prin două treceri

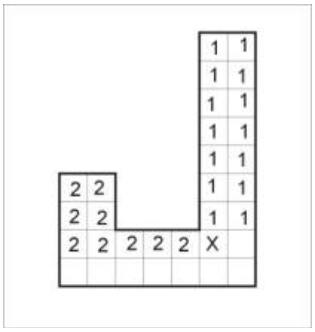


Fig. 5.1 Exemplu de caz unde vecinii anteriori au etichete diferite.
Etichetele 1 și 2 sunt marcate ca echivalente.

5.3 Detalii de implementare

Codul următor ilustrează cum se poate parcurge vecinătatea de 4 a unui pixel. Acest cod se poate modifica ușor pentru a obține o vecinătate de 8, sau pentru a lua în considerare doar vecinii de sus și din stânga.

```
int di[4] = {-1,0,1,0};
int dj[4] = {0,-1,0,1};
uchar neighbors[4];
for(int k=0; k<4; k++)
    neighbors[k] = img.at<uchar>(i+di[k], j+dj[k]);
```

Atenție la limitele imaginii! Nu le depășiți!

Rețineți etichetele într-o matrice capabilă să reprezinte numărul maxim de etichete:

$2^8 = 256$ - **uchar (CV_8UC1)**
$2^{16} = 65536$ - **short (CV_16SC1)**
$2^{32} \sim 2.1e9$ - **int (CV_32SC1)**

Puteți utiliza containerele **std::stack** și **std::queue** pentru a reține punctele pentru Algoritmul 1: stivă pentru DFS, coadă pentru BFS. Punctele se pot stoca sub formă de structuri **pair<int,int>**. Un exemplu de cod pentru inițializarea unei cozi și efectuarea de operații pe aceasta:

```
#include <queue>
queue<pair<int,int>> Q;
Q.push( pair<int,int>(i,j) );
pair<int,int> p = Q.front(); Q.pop();
//se pot accesa coordonatele punctului p astfel
i = p.first; j = p.second;
```

Relațiile de echivalență care definesc muchiile grafului mai mic pot fi memorate folosind liste de adiacență, sub forma **vector<vector<uchar>>**. Exemplu de cod pentru inițializarea listei și inserarea muchiilor:

```
//ne asigurăm că vectorul are o dimensiune suficientă
vector<vector<int>> edges(1000);
//dacă u este echivalent cu v
edges[u].push_back(v);
edges[v].push_back(u);
```

Pentru a afișa matricea de etichete sub forma unei imagini color trebuie să generăm o culoare aleatoare pentru fiecare etichetă. Se poate utiliza generatorul implicit, care este mai bun decât apelul clasic **rand()%256**.

```
#include <random>
default_random_engine gen;
uniform_int_distribution<int> d(0,255);
uchar x = d(gen);
```

5.4 Exemple de etichetare

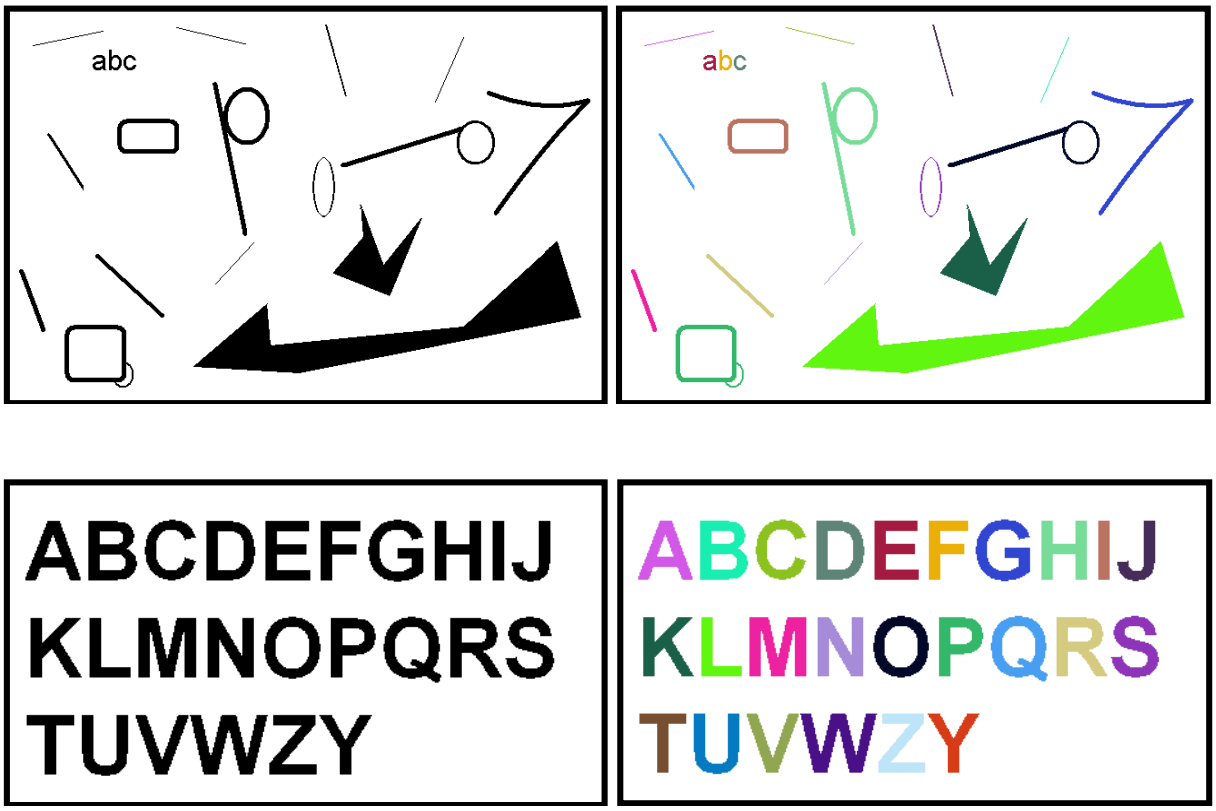


Fig. 5.2 Exemple de etichetare

5.5 Activitate practică

1. Implementați algoritmul de traversare în lățime (Algoritmul 1). Implementați astfel încât să puteți schimba vecinătatea din tipul N4 în tipul N8 și invers.
2. Implementați o funcție care va genera o imagine color pornind de la matricea de etichete. Afișați rezultatele.
3. Implementați algoritmul de etichetare cu două treceri. Afișați rezultatele intermediare după prima parcurgere. Comparați acest rezultat cu rezultatul final și cu rezultatul primului algoritm.
4. Opțional, vizualizați procesul de etichetare prin afișarea rezultatelor intermediare și făcând o pauză după fiecare pas pentru a ilustra ordinea de traversare a punctelor.
5. Opțional, schimbați coada în stivă, pentru a implementa traversarea DFS.
6. **Salvați ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

5.6 Bibliografie

[1]. Umbaugh Scot E, *Computer Vision and Image Processing*, Prentice Hall, NJ, 1998, ISBN 0-13-264599-8
[2]. Robert M. Haralick, Linda G. Shapiro, *Computer and Robot Vision*, Addison-Wesley Publishing Company, 1993.