Technical University of Cluj - Napoca
Computer Science Department

Procesarea Imaginilor

(An 3, Semestrul 2)

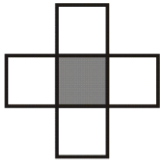
Curs 4: Etichetarea obiectelor. Detecţia conturului obiectelor.

Definiţii

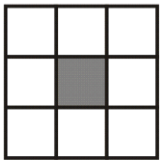
1. Vecini

- Doi pixeli sunt în vecinătate de 4 dacă au o latură comună.
- Doi pixeli sunt în vecinătate de 8 dacă au cel puţin un colţ comun.

N4



N8



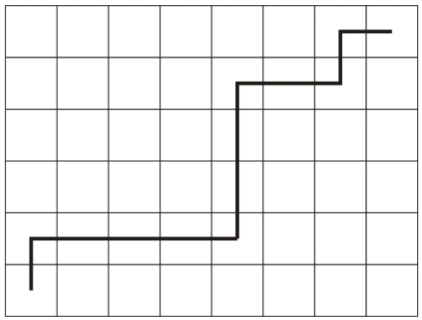
2. Cale (path)

$$\text{Path}\,(p\,[i_0, j_0] \Rightarrow p\,[i_n,j_n]\,) := \{\,[i_0,j_0], [i_1,j_1], \ldots, [i_n,j_n]$$
$$\mid\; [i_k,j_k]\; N_{4/8}\; [i_{k+1},j_{k+1}]\;\forall\; k = 0 \,..\, n\text{-}1\,\}$$

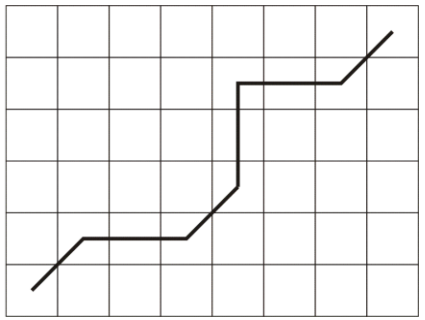
N4 $\Rightarrow$ 4-path

N8 $\Rightarrow$ 8-path

4-path



8-path



Definiţii

3. Obiect

$S := \{ p[i,j] \mid p[i,j] = 1 \}$

4. Conectivitate

$p_S \leftrightarrow q_S$ (conectat) dacă $\exists$ Path $(p \Rightarrow q) \subset S$.

5. Componente conexe

$\{p_i \in S , i = 1 \ldots n \mid p_k \leftrightarrow p_j, \forall (p_k, p_j) \in S, k,j = 1 \ldots n\}$

6. Fundal := mulţimea tuturor componentelor conexe ale mulţimii complement a mulţimii S, C(S), care au puncte pe marginea imaginii. Toate celelalte componente se numesc goluri.

7. Frontieră (boundary)

Boundary (S): = S'=$\{ p \in S \mid \exists q \in N_{4/8}(p), q \in C(S) \}$

C(S) – complement of S

8. Interior

Interior (S) = S - S'

Etichetarea componentelor conexe

Componenta conexa

Mulţimea maximală de puncte conexe

$\{p_i \in S , i = 1 \ldots n \mid p_k \leftrightarrow pj, \forall (p_k, p_j) \in S, \ k,j = 1 \ldots n\}$

- O posibilitate pentru a eticheta obiectele unei imagini binare este să alegem un punct unde b_{ij} =1 şi să ataşăm acestui punct, şi vecinilor acestuia, o etichetă. Apoi, se etichetează vecinii vecinilor, etc.
- Când această procedură recursivă se termină, un obiect este complet etichetat, şi se poate continua prin alegerea unui nou punct de pornire.
- Pentru alegerea unui nou punct, parcurgem sistematic imaginea şi alegem primul punct neetichetat cu b_{ij} =1.

Imaginea ca un graf ne-orientat

- Imaginea binară poate fi considerată un graf ne-orientat. Punctele cu valoare 1 sunt vârfurile grafului, iar vecinătățile sunt arcele.
- Etichetarea: identificarea subgrafurilor conexe
- Algoritmi pentru parcurgerea grafului și identificarea componentelor conexe: explorarea în lățime (Breadth First Search, BFS), sau explorarea în adâncime (Depth First Search, DFS)
- Vom folosi în continuare BFS
- Notații:
  - B(i,j) – valoarea imaginii binare în pozitia (i,j)
  - L(i,j) – valoarea etichetei în pozitia (i,j), inițial zero
  - Neighborhood(i,j), Neighborhood(p) – mulțimea punctelor ce se află în vecinătate de 8 sau 4 față de punctul (i,j), sau față de punctul p.
  - Q – o structura de date de tip coadă
    - Q.empty() - returnează **true** dacă coada este goală
    - Q.enqueue(p) - plasează un punct în coadă
    - p = Q.dequeue() - scoate un punct din coadă

Imaginea ca un graf ne-orientat

- Algoritm de etichetare folosind BFS

```
For i=1 to NLINES
    For j = 1 to NCOLUMNS
        if (B(i,j)==1 and L(i,j)==0)
        L(i,j) = NEWLABEL()
        Q.enqueue( (i,j) )
        while (not Q.empty() )
                p = Q.dequeue()
                For all r in Neighborhood(p)
                        if (B(r.i, r.j)==1 and L(r.i, r.j)==0)
                                L (r.i, r.j) = L (p.i,p.j)
                                Q.enqueue(r)
                        End if
                End for
        End while
    End for
End for
```

Imaginea ca un graf ne-orientat

- NEWLABEL() – funcție care va returna la fiecare apel un număr nou, care va fi folosit ca etichetă pentru grupul de pixeli conectați
- Neighborhood(p) poate fi implementată ca vecinătate de 4, având drept elemente coordonatele (i-1, j), (i, j+1), (i+1, j) și (i, j-1), ca vecinătate de 8, sau se poate utiliza orice altă definiție a vecinătății. De exemplu, se poate defini o vecinătate a tuturor pixelilor care sunt depărtați cu mai puțin de 3 unități de pixelul (i,j).
- Pentru coada Q se poate utiliza un șir obișnuit (array), cu doi indecși, unul care crește la adăugarea unui element nou, și celălalt care crește la extragerea unui element. Coada este vidă atunci când cei doi indecși sunt egali.

Etichetare secvenţială

Algoritmul iterativ (Haralick 1981)

- Nu este necesară memorie suplimentară pentru a produce imaginea etichetată.
- Metodă folositoare atunci când memoria este limitată.

1. Iniţializare – fiecare pixel obiect primește o etichetă unică
2. Repetă:

   Propagarea etichetelor de sus în jos şi de la stânga la dreapta

   Propagarea etichetelor de jos în sus şi de la dreapta la stânga

   Până când nu se mai produc schimbări

```
procedure Iterate;
“Iniţializarea fiecărui pixel obiect cu o etichetă unică”
for L:=1 to NLINES do
        for P:=1 to NCOLUMNS do
                if  I(L,P) =1
                then LABEL(L,P):=NEWLABEL()
                else LABEL(L,P):=0
        end for
end for;
```

Etichetare secvenţială

"**procedure** Iterate – page 2"

```
repeat

"Trecerea sus-jos"
CHANGE:=false;
 for L:=1 to NLINES do
          for P:=1 to NCOLUMNS do
                    if  LABEL(L,P)<>0 then
                              begin
                                        M:=MIN(LABELS(NEIGHBORS(L,P)U(L,P)));
                                        if M<> LABEL(L,P)
                                        then CHANGE:=true;
                                        LABEL(L,P):=M
                              end
          end for
 end for;
```

Etichetare secvenţială

“**procedure** Iterate – page 3”

“Trecerea jos-sus”

```
 for L:= NLINES to 1 by –1 do
          for P:= NCOLUMNS to 1 by –1 do
                    if  LABEL(L,P)<>0 then
                              begin
                                        M:=MIN(LABELS(NEIGHBORS(L,P)U(L,P)));
                                        if M<> LABEL(L,P)
                                        then CHANGE:=true;
                                        LABEL(L,P):=M
                              end
          end for
end for;

until CHANGE==false

end Iterate
```

Etichetare secvenţială

Exemplu (N4 – vecinătate de 4)

	1	1		1	1	
	1	1		1	1	
	1	1	1	1	1	

1. Imagine iniţială

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

2. Iniţializare

	1	1		3	3	
	1	1		3	3	
	1	1	1	1	1	

3. Propagare sus-jos, stânga-dreapta

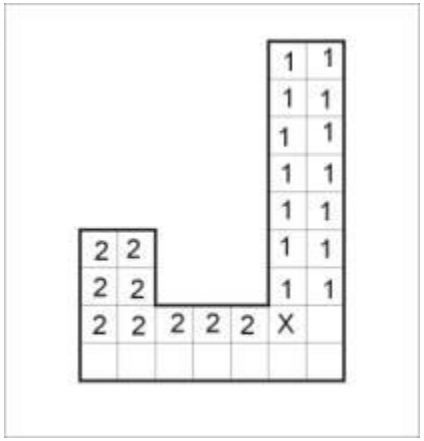
	1	1		1	1	
	1	1		1	1	
	1	1	1	1	1	

4. Propagare jos-sus, dreapta-stânga

Algoritmul clasic

- Se bazează pe algoritmul de găsire a componentelor conexe în grafuri.
- 2 treceri prin imagine, dar necesită o tabelă globală pentru memorarea echivalenţelor.

1. Prima trecere: propagarea etichetelor ca la metoda anterioară.
   - Atunci când două etichete cu valori diferite se pot propaga la acelaşi pixel, se propagă eticheta cu valoarea cea mai mică, iar perechea de etichete se memorează într-o tabelă de echivalenţe (e.g. (1,2) → EqTable).
   - Fiecare poziţie din tabela de echivalenţe este o pereche ce conţine etichete găsite ca echivalente.
   - După prima parcurgere, se calculează clase de echivalenţă.



2. O a doua trecere prin imagine transformă etichetele, dând fiecărei etichete valoarea clasei de echivalenţă din care aceasta face parte.

Algoritmul clasic

Examplu (vecinătate de 4)

1					1	1
		1	1			1
		1				1
		1				1
1	1	1				1
	1	1		1		1
	1	1	1	1		1
				1	1	1

1. Imagine inițială

1					2	2
		3	3			2
		3				2
		3				2
4	4	3				2
	4	3		5		2
	4	3	3	3		2
				3	3	2

2. Imagine etichetată după prima parcurgere

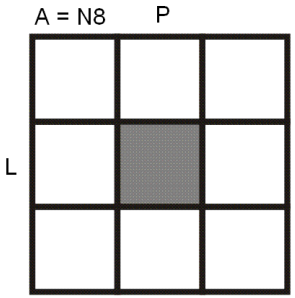
EQTABLE:
(4, 3), (3, 5), (3, 2) …

EQCLASSES:
1: {1}
2: (2, 3, 4, 5}
….

Algoritmul clasic

```
procedure Classical
"Initializarea claselor de echivalenţă"
EQTABLE:=CREATE();

"Parcurgerea 1, de sus în jos"
for L:= 1 to NLINES do
        "Se iniţializează etichetele de pe fiecare linie cu zero"
        for P:= 1 to NCOLUMNS do
                LABEL(L,P):=0
        end for
        "Procesarea liniei"
        for P:=1 to NCOLUMNS do
                if I(L,P):= 1 then
                        begin
                                A:= LABELED_NEIGHBORS((L,P));
                                if ISEMPTY(A)
                                then M:=NEWLABEL()
                                else M:= MIN(LABELS(A));
                                LABEL(L,P):=M;
                                for X in LABELS(A) and X<>M
                                        ADD(X, M, EQTABLE)
                                end for;
                        end
        end for
end for;
```



Algoritmul clasic

“Găsirea claselor de echivalenţă”
EQCLASSES:=Resolve(EQTABLE);

“Parcurgerea 2, de sus în jos”

```
for L:= 1 to NLINES do
        for P:= 1 to NCOLUMNS do
                if I(L,P) = 1
                then LABEL(L,P):=EQCLASSES(LABEL(L,P))
        end for
end for
end Classical
```

- Resolve() – algoritm pentru găsirea componentelor conexe ale structurii de graf definită prin mulţimea echivalenţelor (EQTABLE) găsită la prima parcurgere
- Principala problemă a algoritmului clasic este tabela de echivalenţe, care pentru imagini cu multe regiuni poate deveni foarte mare.

Rezolvarea claselor de echivalență

- Resolve() – algoritm pentru găsirea componentelor conexe ale structurii de graf definită prin mulţimea echivalenţelor (EQTABLE) găsită la prima parcurgere
- EQCLASSES (L) – rezultatul algoritmului Resolve(). Pentru fiecare etichetă L, se va returna o clasă de echivalență.
- Resolve() este un algoritm de **etichetare a etichetelor**
- Etichetele primare formeaza un graf ne-orientat
- Un arc între două etichete este o intrare în tabela EQTABLE
- Vom defini mulțimea vecinătate a unei etichete L

Neighborhood (L) = { M | Există un indice k astfel încât EQTABLE(k).A == L și EQTABLE(k).B == M, sau EQTABLE(k).A == M și EQTABLE(k).B == L }

- Vom utiliza algoritmul BFS, bazat pe o coadă Q

Rezolvarea claselor de echivalență

```
Procedure Resolve()
For L = 1 to MAX_LABELS EQCLASSES(L) = 0 end for
For L = 1 to MAX_LABELS
        if (EQCLASSES(L) == 0)
        EQCLASSES (L) = NEW_EQUIVALENCE_CLASS()
        Q.enqueue( L )
        while (not Q.empty() )
                M = Q.dequeue()
                For all N in Neighborhood(M)
                        if (EQCLASSES(N) == 0)
                                EQCLASSES(N) = EQCLASSES(M)
                                Q.enqueue(N)
                        End if
                End for
        End while
End for
Return EQCLASSES
```

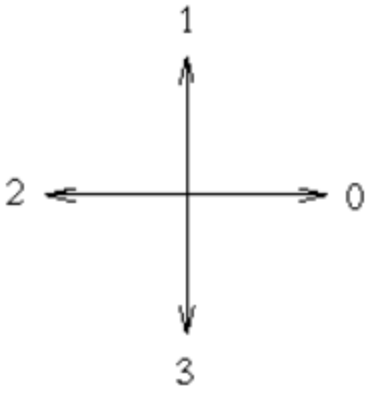
Urmărirea conturului

Frontieră / contur

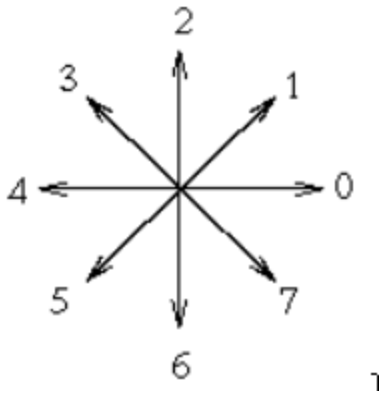
Contur(R) = { p ∈ R | ∃ q ∈ N4/8(p), q ∈ C(R) }

- *Conturul* sau *i-conturul* unei mulţimi de pixeli conectaţi R este definit ca mulţimea tuturor pixelilor din R care au cel puţin un *i-vecin* ce nu aparţine lui R. (i – 4 sau 8)
- *Coduri lanţ – coduri de direcţie* : *c*

  (operaţiile numerice pe *c* sunt modulo 4 sau modulo 8)



4 vecini

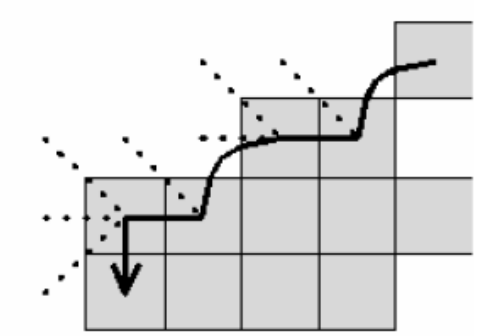


8 vecini

Urmărirea conturului

Algoritmul de trasare a conturului

1. Cauta in imagine (top-down si left-right) pana cand gaseste un pixel de start *P0.* Se defineste o variabila *dir* care stocheaza valoarea directiei ultimei mutari (miscari) de-a lungul conturului, de la pixelul precedent la cel curent. Valorile initiale se asigneaza astfel:
- *dir* = 0 pt. N4
- *dir* = 7 pt. N8



2. Se cauta urmatorul punct de contur intr-o vecinatate de 3x3
In jurul pixelului curent in sens anti-orarar, iincepand cu directia
- (*dir* + 3) *mod* 4 (Fig. 1c)
- (*dir* + 7) *mod* 8 if *dir* este par (Fig. 1d)
- (*dir* + 6) *mod* 8 if *dir* este impar (Fig. 1e)

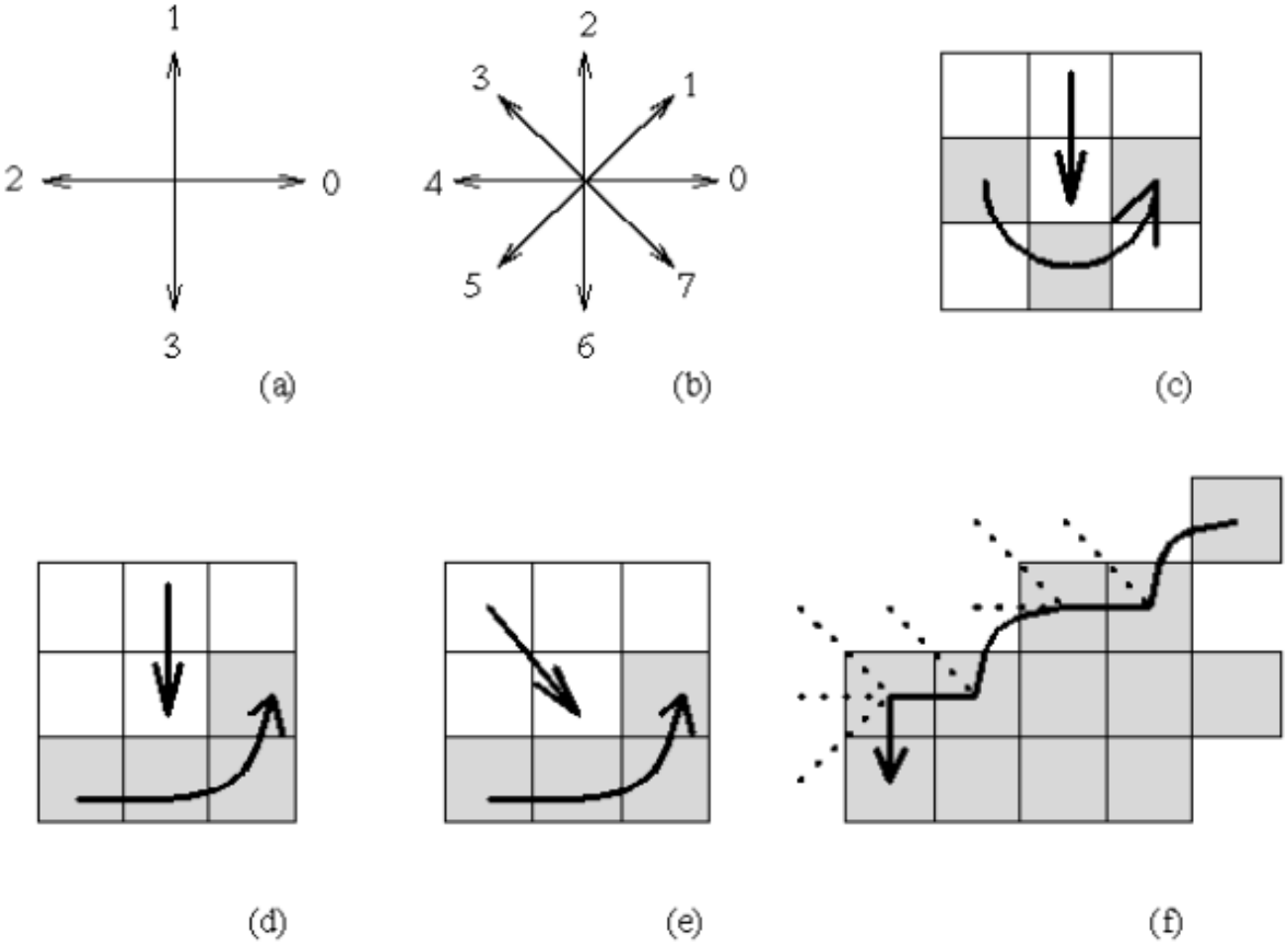
Primul pixel de “1” gasit este noul element de contur curent *Pn*. In acelasi timp se actualizeaza valoarea *dir*.

3. Daca elementul de contur curent *Pn* este egal cu elemntul *P1* si daca elementul *Pn-1* este egal cu *P0*, STOP. Altfel repeta pasul 2.

4. Conturul detectat este reprezentat de multimea: *P0 … Pn-2*.

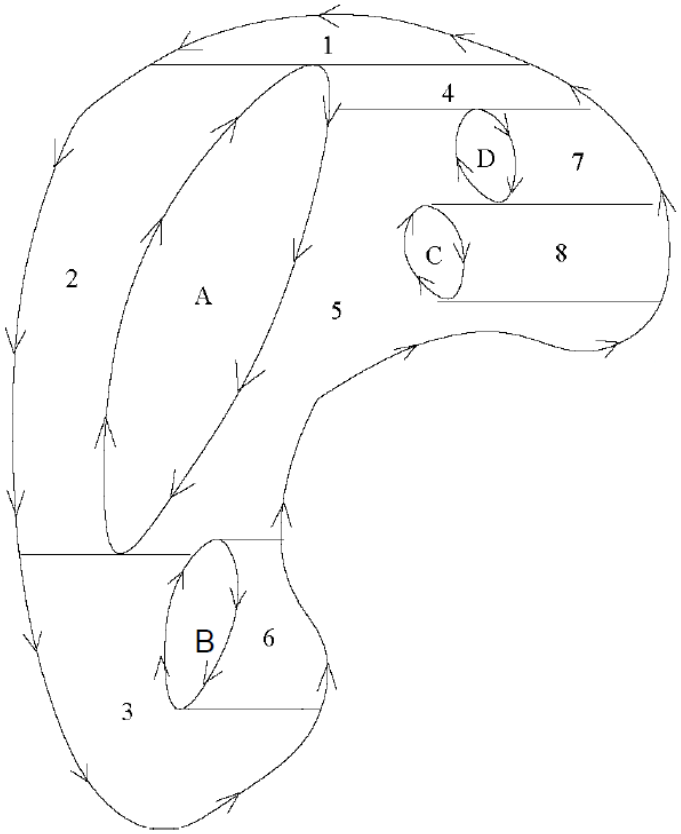
Urmărirea conturului

Algoritmul de trasare a conturului



Urmărirea contururilor multiple

- Pot exista mai multe obiecte – mai multe contururi exterioare
- Pot exista găuri în obiecte – contururi interioare
- Contururile exterioare se parcurg în sens anti-orar
- Contururile interioare se parcurg în sens orar



Urmărirea contururilor multiple

Modificare algoritm de parcurgere:

Pixelii vizitaţi trebuie marcaţi

- Exemplu: pixeli obiect nevizitaţi – valoare 1, pixeli obiect vizitaţi – valoare 2

La finalizarea parcurgerii unui contur, se vor analiza punctele de pe arce descendente, şi se va căuta pe direcţia orizontală spre dreapta, pentru identificarea punctelor de margine de gol – puncte cu valoare 1 (ne-parcurs) care au la dreapta un vecin de valoare 0 (fundal)

- Se va apela algoritmul de parcurgere începând cu punctul găsit

Aproximarea prin poligoane

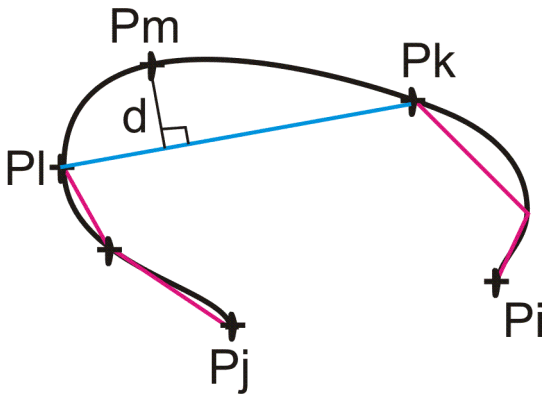
Aproximarea poligonală a contururilor

Curba C: f(x,y)=0 $\Rightarrow$ poligon ce aproximează cât mai bine pe C, având în acelaşi timp un număr cât mai mic de vârfuri posibil**:**

- Orice algoritm de potrivire a unui poligon necesită partiţionarea datelor de intrare în grupuri, fiecare grup fiind aproximat de câte o latură a poligonului.
- O simplificare a problemei este că latura poligonului va fi o linie trasată între punctele extreme ale fiecărui grup, în loc de găsirea soluţiei optime.
- Dacă eroarea de aproximare este prea mare, grupul va fi împărţit în două, şi tot aşa până când eroarea devine acceptabilă.
- Fie Q un contur format din punctele P_i (x_i,y_i) unde i=1,2,…,n, şi ε un prag pentru eroarea de aproximare.

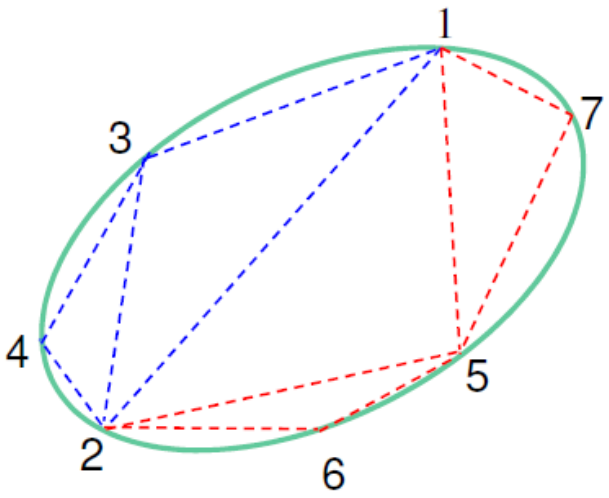
Aproximarea prin poligoane

Procedure POLIGONAL_APROX(Q)
begin
A:=Create_List();
B:=Create_List();
i=Index_of_first_point(Q);
j=Index_of_the_most_far_point(Q);
Insert(j,A); Insert(j,B); // B = (P_j)
Insert(i,A); // A = (P_j, P_i)
while((A!=NULL)
{
Fie *k* şi *l* indecşii ultimelor elemente din listele A şi B;
Fie P_k P_l segmentul generat de cele două puncte;
Fie *m* indexul celui mai îndepărtat punct de segmentul P_kP_l , dintre punctele de contur ce încep din P_k şi se termină în P_l, conturul fiind parcurs în sens opus acelor de ceasornic.
if ((d=Distance(P_m, P_kP_l)> ε)
then Insert(m, A)
else { Delete(k, A)
Insert(k, B); }
}
end



$$\text{distance}(P_1, P_2, (x_0, y_0)) = \frac{|(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2y_1 - y_2x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}.$$

Aproximarea prin poligoane - exemplu



A	B
2	2
1	
3	
4	
2	2
1	4
3	
2	2
1	4
	3
2	2
	4
	3
	1
2	2
5	4
	3
	1

A	B
2	2
5	4
7	3
	1
2	2
5	4
	3
	1
	7
2	2
	4
	3
	1
	7
	5
2	2
6	4
	3
	1
	7
	5

A	B
2	2
	4
	3
	1
	7
	5
	6
	2
	4
	3
	1
	7
	5
	6
	2