Technical University of Cluj - Napoca
Computer Science Department

Procesarea Imaginilor

(An3, semestrul 2)

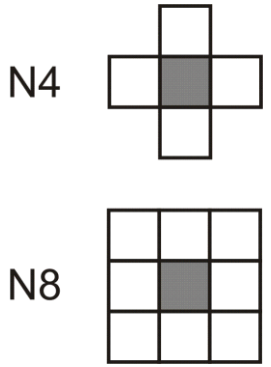
Curs 4: Etichetarea obiectelor. Detectia conturului obiectelor

Definitii

1. Vecini

- Doi pixeli sunt intr-o relatie de vecinatate de tip N4 daca au o frontiera comuna

- Doi pixeli sunt intr-o relatie de vecinatate de tip N8 daca au cel putin un colt comun

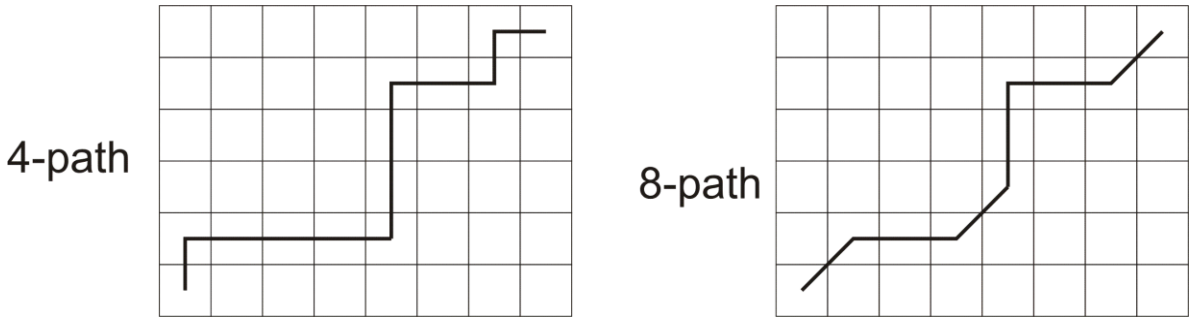


2. Cale (Path)

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

N4 $\Rightarrow$ 4-path

N8 $\Rightarrow$ 8-path



Definitii

3. Obiect (Foreground)

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

4. Conectivitate (Connectivity)

$p_S \leftrightarrow q_S$ (connected) if $\exists$ Path $(p \Rightarrow q) \subset S$.

5. Componente conexe (Connected components) = OBIECTE

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

6. Fundal (Background) := setul tuturor componentelor conexe ale C(S) care au elemente (puncte) pe marginea imaginii. Toate celelalte compnente conexe din C(S) se numesc goluri.

7. Frontiera/Margine (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 (obiect)

Setul maximal de puncte conexe:

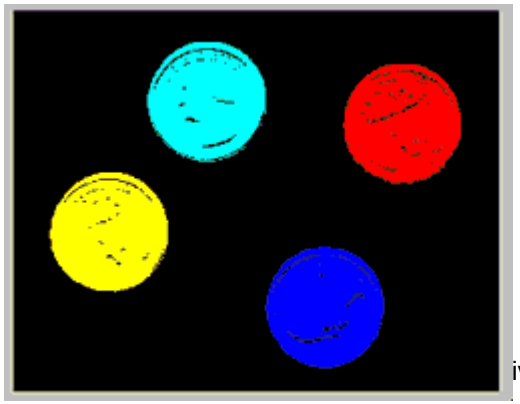
$\{p_i \in S , i = 1 \dots n \mid p_k \leftrightarrow pj, \forall (p_k, p_j) \in S, \ k,j = 1 \dots n\}$

O modalitate de a eticheta obiectele dintr-o imagine digitala binara este de a alege un punct b_{ij} =1 si a asigna o eticheta acestui punct si vecinilor acestuia. Mai departe se vor eticheta toti vecinii vecinilor ….

- Cand procedura recursiva/iterativa se termina, se va obtine o componenta conexa etichetata complet si putem continua alegand alt punct de start (neetichetat inca)
- Pentru a gasi un nou punct de start, se va parcurge imaginea in mod sistematic , incepand o procedura de etichetare de fiecare data cand se gaseste un punct b_{ij} =1.

$\Rightarrow$

Etichetare
(Labeling)



Etichetare secventiala

Algorithmul Iterativ (Haralick 1981)

- Nu necesita spatiu suplimentar de memorie pentru a genera imaginea etichetata din imaginea binara.
- Utila in sisteme cu memorie limitata.

1. Pas de initializare
2. repeat

   propagare top-down & left-right a etichetelor

   propagare bottom-up & right-left a etichetelor

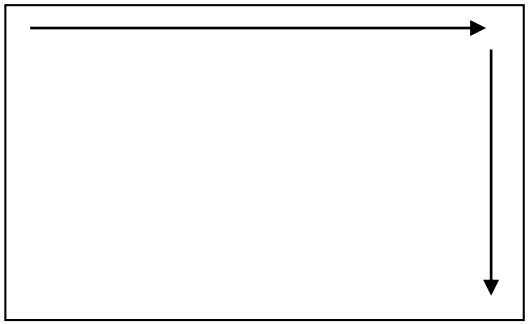
   until "nu mai apare nici o schimbare"

```
procedure Iterate;
// Initializare a fiecarui pixel de "1" cu o eticheta unica
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;
```

LABEL – matrice de etichete

Etichetare secventiala

"**procedure** Iterate – pag. 2"
"Iteratii succesive: top-down si bottom-up"



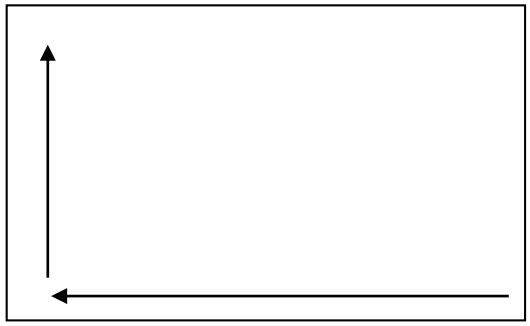
```
repeat

CHANGE:=false;

// Iteratie top-down
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 secventiala

"**procedure** Iterate – pag. 3"



```
// Iteratie bottom-up"
 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 secventiala

Examplu (N4)

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

1. Imagine initiala

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

2. Initializare

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

3. Propagarea etichetelor:

top-down & left-right

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

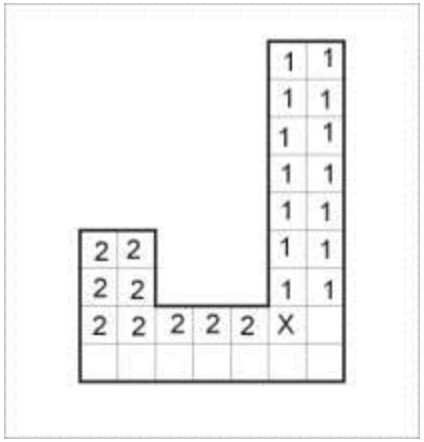
4. Propagarea etichetelor:

bottom-up & right-left

Algoritmul clasic (cu clase de echivalenta)

- Bazat pe algoritmul clasic de gasire a componentelor conexe din grafuri.
- Necesita doar 2 iteratii peste imagine, dar are nevoie de o tabela (mare) pentru inregistrarea echivalentelor .

1. Primul pas: propagarea etichetelor (similar cu algoritmul precedent)
   - De fiecare data cand se intalneste situatia in care doua etichete diferite se pot propaga la acelasi pixel, se va propaga ethicheta cea mai mica si echivalenta gasita se introduce intr-o tabela de echivalente (ex. (1,2) → EqTable).
   - Fiecare intrare din tabela de echivalenta consta intr-o pereche ordonata, valorile continute in fiecare pereche fiind valorile etichetelor gasite echivalente
   - Dupa acest pas se gasesc clasele de chivalenta.
   - Fiecarei clase de echivalenta i se asigneaza o eticheta unica (valoarea minima sau cea mai veche din clasa).



2. Al doilea pas: se parcurge imaginea si se asigneaza fiecarui pixel valoarea etichetei corespunzatoare clasei de chivalenta din care face parte).

Algoritmul Clasic

Examplu (N4)

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 initiala

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 etichetata dupa prima parcurgere Top down (pas 1)

EQTABLE:
(3, 4), (3, 5), (2, 3) …

EQCLASSES:
1: {4, 3, 5, 2}
2: {6,8,9, …}
….
n: {….}

EQLABEL:

1: 2		1:1
2: 6	sau	2:2
…		…
n: x		n:n

Algoritmul Clasic

```
procedure Classical
"Initializare tabela de echivalente globale" si matricea de etichete
EQTABLE:=CREATE(); LABEL:=CREATE();

"Top-down pass 1"
for L:= 1 to NLINES do
        "Initialize all labels on line L to zero"
        for P:= 1 to NCOLUMNS do
                LABEL(L,P):=0
        end for
        "Process the line"
        for P:=1 to NCOLUMNS do
                if I(L,P):= 1 then
                        begin
                                V:= NEIGHBORS((L,P));
                                if ISEMPTY(V)
                                then M:=NEWLABEL()
                                else M:= MIN(LABELS(V));
                                LABEL(L,P):=M;
                                for X in LABELS(V) and X<>M
                                        ADD(X, M, EQTABLE)
                                end for;
                        end
        end for
end for;
```

LABEL – matrice ("imagine") de etichete

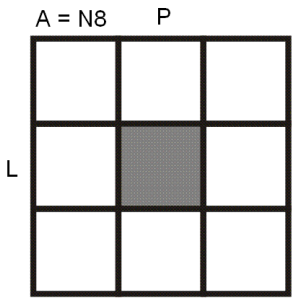


IMAGE PROCESSING

Algoritmul Clasic

```
“Find equivalence classes”
EQCLASSES:=Resolve(EQTABLE);

for E in EQCLASSES
        EQLABEL(E):= min(LABELS(E))
end for;

“Top-down pass 2”
for L:= 1 to NLINES do
        for P:= 1 to NCOLUMNS do
                if I(L,P) = 1
                then LABEL(L,P):=EQLABEL(CLASS(LABEL(L,P)))
        end for
end for
end Classical
```

- Resolve() - algoritm pt. gasirea componentelor conexe ale grafului definit de setul de echivalente (EQTABLE) definit la pasul 1.
- Porblema principala a acestui algoritm: pentru imagini mari cu multe regiuni, tabela de echivalente poate deveni foarte mare

Algoritmul Clasic

Procedura *Resolve( )*

Pentru a modela echivalenţa etichetelor putem folosi un graf neorientat în care nodurile sunt etichetele iar arcele sunt relaţiile binare de echivalenţă stabilite în pasul anterior (o intrare in tabela EQTABLE) $\Rightarrow$ clasele de echivalenţă sunt subgrafurile conexe ale acestui graf.

Subgraf conex $\Rightarrow$ căutare în lăţime (Breadth First Search, BFS), pornind dintr-un prim nod oarecare al subgrafului:

- se pun toţi vecinii primului nod într-o listă şi se marchează ca fiind luaţi în considerare
- pentru fiecare nod din lista respectivă, acesta este extras din listă, şi i se adaugă toţi vecinii (neconsideraţi încă) în lista şi se marchează ca fiind luaţi în considerare.
- procesul se repetă pentru fiecare nod adăugat în listă, până când lista devine goală
- toate nodurile trecute prin această listă vor fi etichetate cu aceeaşi etichetă, care va fi noua etichetă asociată acestei clase de etichete echivalente.

În continuare se caută un alt nod care nu a fost considerat, şi pornind de la acesta se va căuta o nouă clasă de echivalenţă. Procesul se continuă până când sunt considerate toate nodurile grafului. Fiecare nod singular reprezintă câte o clasă de echivalenţă cu un singur element.

Rezolvarea claselor de echivalenta

EQCLASSES:=Resolve(EQTABLE);

Ex. EQTABLE:

x	y
3	4
3	5
2	3
6	8
6	9
7	8
..	..

Vom utiliza algoritmul BFS, bazat pe o coada Q

- Definim multimea vecinatate a unei etichete L

Neighborhood (L) = { M | ∃ k | EQTABLE(k).x == L si EQTABLE(k).y == M,
sau EQTABLE(k).x == M si EQTABLE(k).y == L }

- Q – o structura de date de tip coada (lista FIFO)
  - Q.empty() - returneaza **true** daca coada este goala (top = bottom)
  - Q.enqueue(p) - plaseaza un nod (eticheta) in coada (top++)
  - p = Q.dequeue() - scoate un nod (eticheta) din coada (bottom ++)

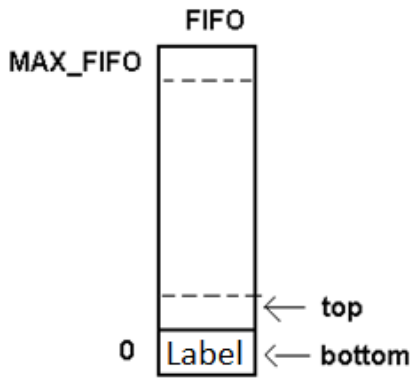


IMAGE PROCESSING

Rezolvarea claselor de echivalenta

```
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
```

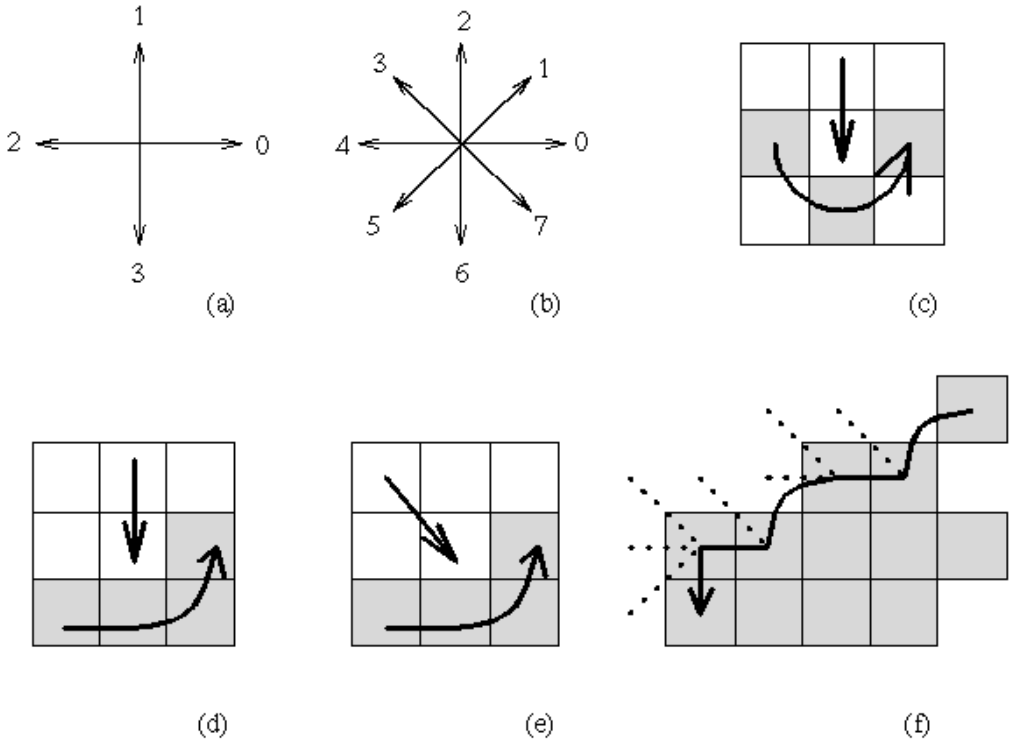
Contour Tracing (detectie contur)

Contur:

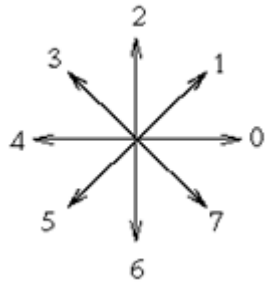
Contour (R) = { p ∈ R | ∃ q ∈ N4/8(p), q ∈ C(R) }

(*chain-code / direction codes*): *c*

(operatiile numerice asupra lui *c* se presupun a fi modulo 4 sau 8)



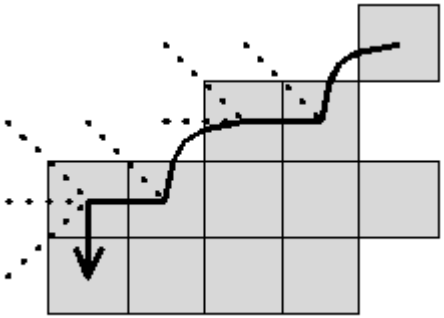
Contour Tracing (detectie contur)



Algoritm trasare contur exterior:

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, secvential (dir++), in sens anti-orarar, incepand cu directia

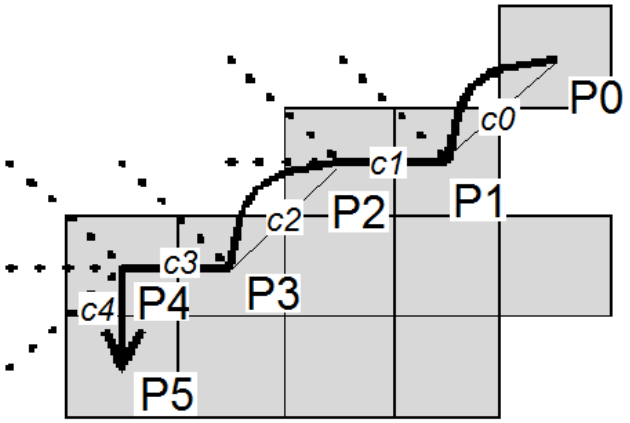
- (*dir* + 3) *mod* 4
- (*dir* + 7) *mod* 8 if *dir* este par
- (*dir* + 6) *mod* 8 if *dir* este impar

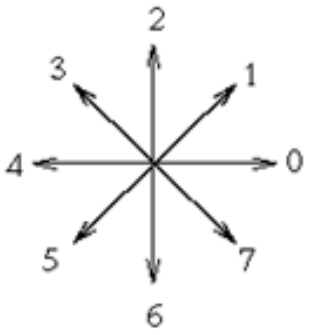
Primul pixel de “1” gasit este noul element de contur curent *Pn*. In acelasi timp se actualizeaza valoarea *dir*.

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*.

Exemplu – reprezentarea conturului (N8)





$c_i \in \{0,1, \dots ,7\}$ - cod de directie

Var.1 – lista de puncte:

$L = \{ P_0(x_0,y_0), P_1(x_1,y_1), \dots , P_{n-2}(x_{n-2},y_{n-2}) \}$

Var .2 – coduri inlantuite:

$P_0(x_0,y_0)$, $c = \{c_0, c_1, \dots , c_{n-2}\}$,

Var. 3 – derivata codului inlantuit (invarianta la rotatie)

$P_0(x_0,y_0)$, $cd = \{cd_0, cd_1, \dots , cd_{n-2}\}$,

unde: $cd_i = (c_i - c_{i-1}) \ mod \ 8$, $cd_0 = c_0 \ mod \ 8$

Aproximarea conturului prin linii poligonale

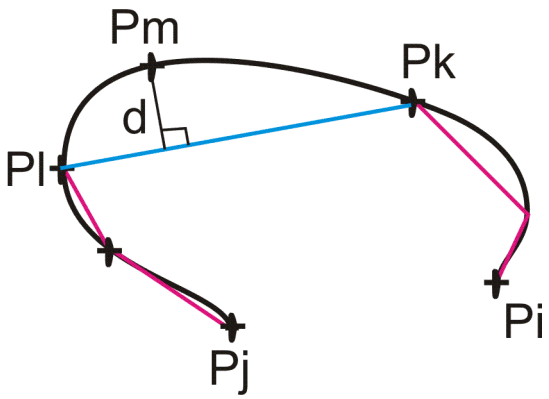
Scop:

Aproximarea curbei C: f(x,y)=0 cu o linie poligonala avand un numar relativ mic de linii (varfuri) care aproximeaza cel mai bine curba C

- Orice algoritm de aproximare poligonala necesita ca punctele (datele) sa fie divizate in grupuri, fiecare dintre acestea fiind aproximate de cate o latura a poligonului
- Prima simplificare a problemei de fitting este de a duce o linie intre capetele fiecarui grup In loc sa cautam solutia optimala.
- Daca eroarea de aproximare este prea mare , grupul se poate sparge in sub-grupuri pana cand eroarea de aproximare devine acceptabila.
- Fie Q un contur alcatuit din punctele Pi (xi,yi) unde i=1,2,…,n, si ε eroarea maxima.

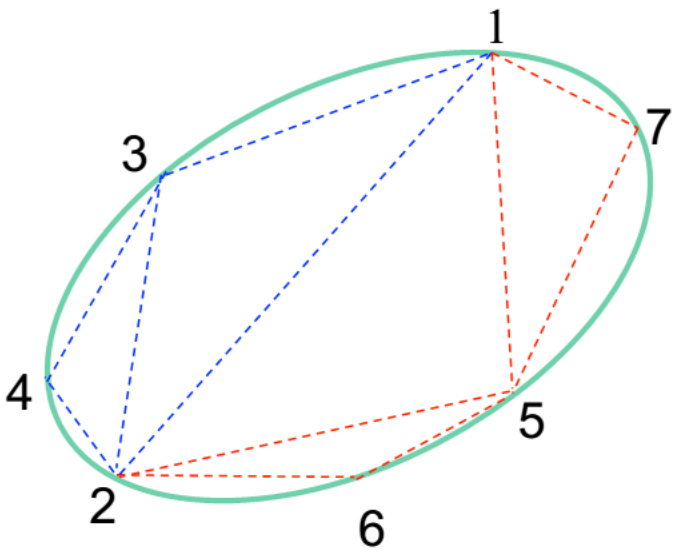
Aproximarea conturului prin linii poligonale

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 conturului prin linii poligonale - 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

IMA