Triangulări - Algoritmul Graham Scan

Marian Ioan MUNTEANU

Al.I.Cuza University of Iasi, Romania
webpage: http://www.math.uaic.ro/∼munteanu

10 decembrie 2012

Cuprins

De ce?

- necesitatea îmbunătățirii tehnicilor de reprezentare a curbelor și a suprafețelor pe calculator în domeniul *CAGD*-ului utilizând inițial elemente de Geometrie Computațională;

- CAGD $\Longleftrightarrow$ Geometrie Computațională

Deși aparent diferite aceste discipline, la Conferința CAGD din 1974 graficianul G.Chaikin a prezentat o metodă de generare a curbelor pornind de la un poligon închis 2D și folosind un proces continuu de retezare a vârfurilor ajungând astfel la o limită apropiată a curbei $\Longrightarrow$ o metodă iterativă de generare a curbelor.

Următorul pas a fost analiza suprafețelor polinomiale bidimensionale care pot fi reprezentate:

1. utilizând produsul tensorial pe un domeniu dreptunghiular;
2. folosind coordonate baricentrice raportate la un domeniu triunghiular;
3. cu ajutorul "peticelor triunghiulare".

De ce?

- necesitatea îmbunătățirii tehnicilor de reprezentare a curbelor și a suprafețelor pe calculator în domeniul *CAGD*-ului utilizând inițial elemente de Geometrie Computațională;
- CAGD $\Longleftrightarrow$ Geometrie Computațională

Deși aparent diferite aceste discipline, la Conferința CAGD din 1974 graficianul G.Chaikin a prezentat o metodă de generare a curbelor pornind de la un poligon închis 2D și folosind un proces continuu de retezare a vârfurilor ajungând astfel la o limită apropiată a curbei $\Longrightarrow$ o metodă iterativă de generare a curbelor.

Următorul pas a fost analiza suprafețelor polinomiale bidimensionale care pot fi reprezentate:

1. utilizând produsul tensorial pe un domeniu dreptunghiular;
2. folosind coordonate baricentrice raportate la un domeniu triunghiular;
3. cu ajutorul "peticelor triunghiulare".

De ce?

- necesitatea îmbunătățirii tehnicilor de reprezentare a curbelor și a suprafețelor pe calculator în domeniul *CAGD*-ului utilizând inițial elemente de Geometrie Computațională;
- CAGD $\Longleftrightarrow$ Geometrie Computațională

Deși aparent diferite aceste discipline, la Conferința CAGD din 1974 graficianul G.Chaikin a prezentat o metodă de generare a curbelor pornind de la un poligon închis 2D și folosind un proces continuu de retezare a vârfurilor ajungând astfel la o limită apropiată a curbei $\Longrightarrow$ o metodă iterativă de generare a curbelor.

Următorul pas a fost analiza suprafețelor polinomiale bidimensionale care pot fi reprezentate:

1. utilizând produsul tensorial pe un domeniu dreptunghiular;
2. folosind coordonate baricentrice raportate la un domeniu triunghiular;
3. cu ajutorul "peticelor triunghiulare".

De ce?

- necesitatea îmbunătățirii tehnicilor de reprezentare a curbelor și a suprafețelor pe calculator în domeniul *CAGD*-ului utilizând inițial elemente de Geometrie Computațională;
- CAGD $\Longleftrightarrow$ Geometrie Computațională

Deși aparent diferite aceste discipline, la Conferința CAGD din 1974 graficianul G.Chaikin a prezentat o metodă de generare a curbelor pornind de la un poligon închis 2D și folosind un proces continuu de retezare a vârfurilor ajungând astfel la o limită apropiată a curbei $\Longrightarrow$ o metodă iterativă de generare a curbelor.

Următorul pas a fost analiza suprafețelor polinomiale bidimensionale care pot fi reprezentate:

1. utilizând produsul tensorial pe un domeniu dreptunghiular;
2. folosind coordonate baricentrice raportate la un domeniu triunghiular;
3. cu ajutorul "peticelor triunghiulare".

Algoritmi de triangulare

Algoritmii de triangulare se ocupă cu determinarea unei mulțimi de triunghiuri având dată o mulțime de puncte 2D drept vârfuri. În general, am putea numi triangulare orice mulțime de triunghiuri în plan. Însă, din motive atât teoretice dar și practice, o mulțime de triunghiuri definește o triangulare dacă:

- nici un triunghi al unei triangulări nu este degenerat;
- interioarele oricăror două triunghiuri nu se intersectează;
- frontierele a două triunghiuri se pot intersecta numai după o latură comună sau un vârf comun;
- reuniunea tuturor triunghiurilor unei triangulări este egală cu domeniul triangulat;
- domeniul de triangulat trebuie să fie convex;
- triangularea nu trebuie să aibă găuri;
- un vârf al frontierei domeniului are exact două laturi de frontieră comune. Această implică faptul că numărul vârfurilor frontierei este egal cu numărul laturilor frontierei.

Algoritmi de triangulare

Algoritmii de triangulare se ocupă cu determinarea unei mulțimi de triunghiuri având dată o mulțime de puncte 2D drept vârfuri. În general, am putea numi triangulare orice mulțime de triunghiuri în plan. Însă, din motive atât teoretice dar și practice, o mulțime de triunghiuri definește o triangulare dacă:

- nici un triunghi al unei triangulări nu este degenerat;
- interioarele oricăror două triunghiuri nu se intersectează;
- frontierele a două triunghiuri se pot intersecta numai în cazul unei muchii comune sau un vârf comun;
- reuniunea tuturor triunghiurilor unei triangulări este egală cu cea a domeniului triangulat;
- domeniul de triangulat trebuie să fie conex;
- triangulările nu trebuie să aibă găuri;
- un vârf al frontierei domeniului trebuie să fie vârf al cel puțin două triunghiuri comune. Aceasta implică faptul că numărul de vârfuri ale frontierei este egal cu numărul laturilor frontierei.

Algoritmi de triangulare

Algoritmii de triangulare se ocupă cu determinarea unei mulțimi de triunghiuri având dată o mulțime de puncte 2D drept vârfuri. În general, am putea numi triangulare orice mulțime de triunghiuri în plan. Însă, din motive atât teoretice dar și practice, o mulțime de triunghiuri definește o triangulare dacă:

- nici un triunghi al unei triangulări nu este degenerat;
- interioarele oricăror două triunghiuri nu se intersectează;
- frontierele a două triunghiuri se pot intersecta numai după o latură comună sau un vârf comun;
- reuniunea tuturor triughiurilor unei triangulări este egală cu domeniul triangulat;
- domeniul de triangulat trebuie să fie conex;
- triangularea nu trebuie să aibă goluri;
- un vârf al frontierei domeniului are exact două laturi de frontieră comune. Aceasta implică faptul că numărul vârfurilor de frontieră este egal cu numărul laturilor frontierei.

Algoritmi de triangulare

Algoritmii de triangulare se ocupă cu determinarea unei mulțimi de triunghiuri având dată o mulțime de puncte 2D drept vârfuri. În general, am putea numi triangulare orice mulțime de triunghiuri în plan. Însă, din motive atât teoretice dar și practice, o mulțime de triunghiuri definește o triangulare dacă:

- nici un triunghi al unei triangulări nu este degenerat;
- interioarele oricăror două triunghiuri nu se intersectează;
- frontierele a două triunghiuri se pot intersecta numai după o latură comună sau un vârf comun;
- reuniunea tuturor triughiurilor unei triangulări este egală cu domeniul triangulat;
- domeniul de triangulat trebuie să fie conex;
- triangularea nu trebuie să aibă goluri;
- un vârf al frontierei domeniului are exact două laturi de frontieră comune. Aceasta implică faptul că numărul vârfurilor de frontieră este egal cu numărul laturilor frontierei;

Algoritmi de triangulare

Algoritmii de triangulare se ocupă cu determinarea unei mulțimi de triunghiuri având dată o mulțime de puncte 2D drept vârfuri. În general, am putea numi triangulare orice mulțime de triunghiuri în plan. Însă, din motive atât teoretice dar și practice, o mulțime de triunghiuri definește o triangulare dacă:

- nici un triunghi al unei triangulări nu este degenerat;
- interioarele oricăror două triunghiuri nu se intersectează;
- frontierele a două triunghiuri se pot intersecta numai după o latură comună sau un vârf comun;
- reuniunea tuturor triughiurilor unei triangulări este egală cu domeniul triangulat;
- domeniul de triangulat trebuie să fie conex;
- triangularea nu trebuie să aibă goluri;
- un vârf al frontierei domeniului are exact două laturi de frontieră comune. Aceasta implică faptul că numărul vârfurilor de frontieră este egal cu numărul laturilor frontierei.

Algoritmi de triangulare

Algoritmii de triangulare se ocupă cu determinarea unei mulțimi de triunghiuri având dată o mulțime de puncte 2D drept vârfuri. În general, am putea numi triangulare orice mulțime de triunghiuri în plan. Însă, din motive atât teoretice dar și practice, o mulțime de triunghiuri definește o triangulare dacă:

- nici un triunghi al unei triangulări nu este degenerat;
- interioarele oricăror două triunghiuri nu se intersectează;
- frontierele a două triunghiuri se pot intersecta numai după o latură comună sau un vârf comun;
- reuniunea tuturor triughiurilor unei triangulări este egală cu domeniul triangulat;
- domeniul de triangulat trebuie să fie conex;
- triangularea nu trebuie să aibă goluri;
- un vârf al frontierei domeniului trebuie să fie vârful a cel puțin două laturi comune. Aceasta implică faptul că numărul triunghiurilor unei triangulări este egal cu numărul punctelor frontierei.

Algoritmi de triangulare

Algoritmii de triangulare se ocupă cu determinarea unei mulțimi de triunghiuri având dată o mulțime de puncte 2D drept vârfuri. În general, am putea numi triangulare orice mulțime de triunghiuri în plan. Însă, din motive atât teoretice dar și practice, o mulțime de triunghiuri definește o triangulare dacă:

- nici un triunghi al unei triangulări nu este degenerat;
- interioarele oricăror două triunghiuri nu se intersectează;
- frontierele a două triunghiuri se pot intersecta numai după o latură comună sau un vârf comun;
- reuniunea tuturor triughiurilor unei triangulări este egală cu domeniul triangulat;
- domeniul de triangulat trebuie să fie conex;
- triangularea nu trebuie să aibă goluri;
- un vârf al frontierei domeniului are exact două laturi de frontieră comune. Aceasta implică faptul că numărul vârfurilor de frontieră este egal cu numărul laturilor frontierei;

Algoritmi de triangulare

Algoritmii de triangulare se ocupă cu determinarea unei mulțimi de triunghiuri având dată o mulțime de puncte 2D drept vârfuri. În general, am putea numi triangulare orice mulțime de triunghiuri în plan. Însă, din motive atât teoretice dar și practice, o mulțime de triunghiuri definește o triangulare dacă:

- nici un triunghi al unei triangulări nu este degenerat;
- interioarele oricăror două triunghiuri nu se intersectează;
- frontierele a două triunghiuri se pot intersecta numai după o latură comună sau un vârf comun;
- reuniunea tuturor triughiurilor unei triangulări este egală cu domeniul triangulat;
- domeniul de triangulat trebuie să fie conex;
- triangularea nu trebuie să aibă goluri;
- un vârf al frontierei domeniului are exact două laturi de frontieră comune. Aceasta implică faptul că numărul vârfurilor de frontieră este egal cu numărul laturilor frontierei;

Algoritmi de triangulare

Algoritmii de triangulare se ocupă cu determinarea unei mulțimi de triunghiuri având dată o mulțime de puncte 2D drept vârfuri. În general, am putea numi triangulare orice mulțime de triunghiuri în plan. Însă, din motive atât teoretice dar și practice, o mulțime de triunghiuri definește o triangulare dacă:

- nici un triunghi al unei triangulări nu este degenerat;
- interioarele oricăror două triunghiuri nu se intersectează;
- frontierele a două triunghiuri se pot intersecta numai după o latură comună sau un vârf comun;
- reuniunea tuturor triughiurilor unei triangulări este egală cu domeniul triangulat;
- domeniul de triangulat trebuie să fie conex;
- triangularea nu trebuie să aibă goluri;
- un vârf al frontierei domeniului are exact două laturi de frontieră comune. Aceasta implică faptul că numărul vârfurilor de frontieră este egal cu numărul laturilor frontierei;

Proprietăți ale triangulărilor

Un aspect important care trebuie cunoscut despre o triangulare, mai ales d.p.d.v. al CG, este dimensiunea triangulării obținute pentru un domeniu dat.

Notații:

- ✓ V- mulțimea de vârfuri, $V = V_I \cup V_B$;
- ✓ E- mulțimea de laturi;
- ✓ T- mulțimea de triunghiuri;

I - componente interioare

B - componente de frontieră

Proprietate (de demonstrat)

$|T| = 2|V_I| + |V_B| - 2$; $|E| = 3|V_I| + 2|V_B| - 3$; $|E_I| = 3|V_I| + |V_B| - 3$.

Proprietate (formula Euler-Poincaré)

$|T| = |E| - |V| + 1$.

Proprietăți ale triangulărilor

Un aspect important care trebuie cunoscut despre o triangulare, mai ales d.p.d.v. al CG, este dimensiunea triangulării obținute pentru un domeniu dat.
Notații:

- ✓ V- mulțimea de vârfuri, $V = V_I \cup V_B$;
- ✓ E- mulțimea de laturi;
- ✓ T- mulțimea de triunghiuri;

I - componente interioare

B - componente de frontieră

Proprietate (de demonstrat)

$|T| = 2|V_I| + |V_B| - 2; |E| = 3|V_I| + 2|V_B| - 3; |E_I| = 3|V_I| + |V_B| - 3.$

Proprietate (formula Euler-Poincaré)

$|T| = |E| - |V| + 1.$

Proprietăți ale triangulărilor

Un aspect important care trebuie cunoscut despre o triangulare, mai ales d.p.d.v. al CG, este dimensiunea triangulării obținute pentru un domeniu dat.
Notații:

- ✓ V- mulțimea de vârfuri, $V = V_I \cup V_B$;
- ✓ E- mulțimea de laturi;
- ✓ T- mulțimea de triunghiuri;

I - componente interioare

B - componente de frontieră

Proprietate (de demonstrat)

$$|T| = 2|V_I| + |V_B| - 2;\ |E| = 3|V_I| + 2|V_B| - 3;\ |E_I| = 3|V_I| + |V_B| - 3.$$

Proprietate (formula Euler-Poincaré)

$$|T| = |E| - |V| + 1.$$

Proprietăți ale triangulărilor

Un aspect important care trebuie cunoscut despre o triangulare, mai ales d.p.d.v. al CG, este dimensiunea triangulării obținute pentru un domeniu dat.
Notații:

- ✓ V- mulțimea de vârfuri, $V = V_I \cup V_B$;
- ✓ E- mulțimea de laturi;
- ✓ T- mulțimea de triunghiuri;

I - componente interioare

B - componente de frontieră

Proprietate (de demonstrat)

$$|T| = 2|V_I| + |V_B| - 2;\ |E| = 3|V_I| + 2|V_B| - 3;\ |E_I| = 3|V_I| + |V_B| - 3.$$

Proprietate (formula Euler-Poincaré)

$$|T| = |E| - |V| + 1.$$

Proprietăți ale triangulărilor

Un aspect important care trebuie cunoscut despre o triangulare, mai ales d.p.d.v. al CG, este dimensiunea triangulării obținute pentru un domeniu dat.
Notații:

- ✓ V- mulțimea de vârfuri, $V = V_I \cup V_B$;
- ✓ E- mulțimea de laturi;
- ✓ T- mulțimea de triunghiuri;

I - componente interioare

B - componente de frontieră

Proprietate (de demonstrat)

$$|T| = 2|V_I| + |V_B| - 2;\ |E| = 3|V_I| + 2|V_B| - 3;\ |E_I| = 3|V_I| + |V_B| - 3.$$

Proprietate (formula Euler-Poincaré)

$$|T| = |E| - |V| + 1.$$

caz special al formulei lui Euler: pentru un graf plan convex are loc

$$m_v - m_e + m_f = 2$$

Proprietăți ale triangulărilor

Un aspect important care trebuie cunoscut despre o triangulare, mai ales d.p.d.v. al CG, este dimensiunea triangulării obținute pentru un domeniu dat.
Notații:

- ✓ V- mulțimea de vârfuri, $V = V_I \cup V_B$;
- ✓ E- mulțimea de laturi;
- ✓ T- mulțimea de triunghiuri;

I - componente interioare

B - componente de frontieră

Proprietate (de demonstrat)

$|T| = 2|V_I| + |V_B| - 2$; $|E| = 3|V_I| + 2|V_B| - 3$; $|E_I| = 3|V_I| + |V_B| - 3$.

Proprietate (formula Euler-Poincaré)

$|T| = |E| - |V| + 1$.

Proprietate (de demonstrat)

$|V| - 2 \leqslant |T| \leqslant 2|V| - 5$; $2|V| - 3 \leqslant |E| \leqslant 3|V| - 6$

Preliminarii

- unul dintre primii algoritmi de triangulare, inițial conceput pentru a calcula înfășurătoarea convexă a unei mulțimi de puncte în plan;
- 1990, Kong, Everret, Toussaint implementează algoritmul folosind tehnica ear cutting pentru triangularea unui poligon simplu $\mathcal{P} = (p_0, ..., p_{n-1})$ cu n vârfuri.
- complexitatea algoritmului este $O(kn)$, unde $k - 1$ reprezintă numărul de vârfuri concave.

Definiția

$\mathcal{P}$ este un poligon simplu dacă oricare două laturi neconsecutive nu se intersectează.

Definiția

Segmentul de dreaptă care unește două vârfuri neconsecutive p_i și p_j se numește diagonală a lui $\mathcal{P}$ dacă aceasta este inclusă în interiorul poligonului.

Preliminarii

- unul dintre primii algoritmi de triangulare, inițial conceput pentru a calcula înfășurătoarea convexă a unei mulțimi de puncte în plan;
- 1990, Kong, Everret, Toussaint implementează algoritmul folosind tehnica ear cutting pentru triangularea unui poligon simplu $\mathcal{P} = (p_0, ..., p_{n-1})$ cu n vârfuri.
- complexitatea algoritmului este $O(kn)$, unde $k-1$ reprezintă numărul de vârfuri concave.

Definiția

$\mathcal{P}$ este un poligon simplu dacă oricare două laturi neconsecutive nu se intersectează.

Definiția

Segmentul de dreaptă care unește două vârfuri neconsecutive p_i și p_j se numește diagonală a lui $\mathcal{P}$ dacă aceasta este inclusă în interiorul poligonului.

Preliminarii

- unul dintre primii algoritmi de triangulare, inițial conceput pentru a calcula înfășurătoarea convexă a unei mulțimi de puncte în plan;
- 1990, Kong, Everret, Toussaint implementează algoritmul folosind tehnica ear cutting pentru triangularea unui poligon simplu $\mathcal{P} = (p_0, ..., p_{n-1})$ cu n vârfuri.
- complexitatea algoritmului este $O(kn)$, unde $k - 1$ reprezintă numărul de vârfuri concave.

Definiția

$\mathcal{P}$ este un poligon simplu dacă oricare două laturi neconsecutive nu se intersectează.

Definiția

Segmentul de dreaptă care unește două vârfuri neconsecutive p_i și p_j se numește diagonală a lui $\mathcal{P}$ dacă aceasta este inclusă în interiorul poligonului.

Preliminarii

- unul dintre primii algoritmi de triangulare, inițial conceput pentru a calcula înfășurătoarea convexă a unei mulțimi de puncte în plan;
- 1990, Kong, Everret, Toussaint implementează algoritmul folosind tehnica ear cutting pentru triangularea unui poligon simplu $\mathcal{P} = (p_0, ..., p_{n-1})$ cu n vârfuri.
- complexitatea algoritmului este $O(kn)$, unde $k - 1$ reprezintă numărul de vârfuri concave.

Definiția

$\mathcal{P}$ este un poligon simplu dacă oricare două laturi neconsecutive nu se intersectează.

Definiția

Segmentul de dreaptă care unește două vârfuri neconsecutive p_i și p_j se numește diagonală a lui $\mathcal{P}$ dacă aceasta este inclusă în interiorul poligonului.

Preliminarii

- unul dintre primii algoritmi de triangulare, inițial conceput pentru a calcula înfășurătoarea convexă a unei mulțimi de puncte în plan;
- 1990, Kong, Everret, Toussaint implementează algoritmul folosind tehnica ear cutting pentru triangularea unui poligon simplu $\mathcal{P} = (p_0, ..., p_{n-1})$ cu n vârfuri.
- complexitatea algoritmului este $O(kn)$, unde $k - 1$ reprezintă numărul de vârfuri concave.

Definiția

$\mathcal{P}$ este un poligon simplu dacă oricare două laturi neconsecutive nu se intersectează.

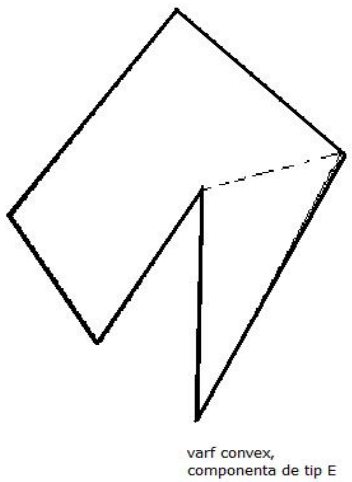
Definiția

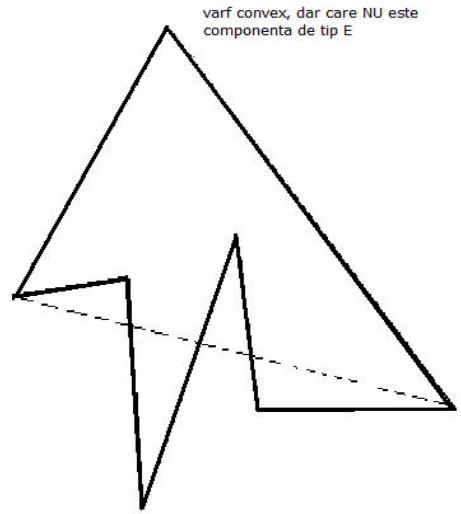
Segmentul de dreaptă care unește două vârfuri neconsecutive p_i și p_j se numește diagonală a lui $\mathcal{P}$ dacă aceasta este inclusă în interiorul poligonului.

Preliminarii: 2E & 1M

Definiția

Un vârf p_i al unui poligon simplu s. n. componentă **E** *dacă segmentul de dreaptă (p_{i-1}, p_{i+1}) este o diagonală. Punctul p_{i+1} s.n. vârful componentei* **E**, p_i.





Teorema fundamentală a algoritmului:

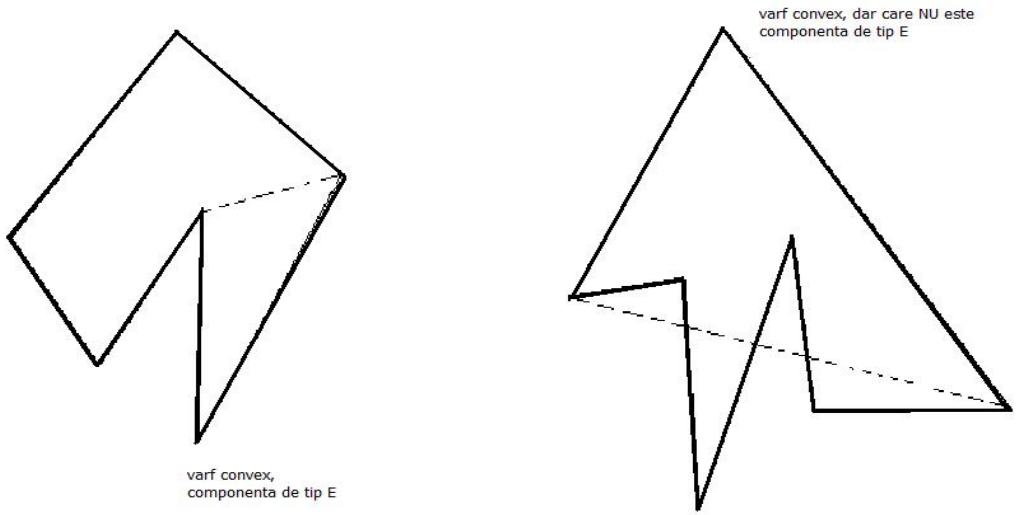
Teorema (2 - E)

Exceptând triunghiurile, orice poligon simplu are cel puțin 2 componente **E** *care nu se suprapun (disjuncte).*

Preliminarii: 2E & 1M

Definiția

Un vârf p_i al unui poligon simplu s. n. componentă **E** *dacă segmentul de dreaptă (p_{i-1}, p_{i+1}) este o diagonală. Punctul p_{i+1} s.n. vârful componentei* **E**, p_i.



Teorema fundamentală a algoritmului:

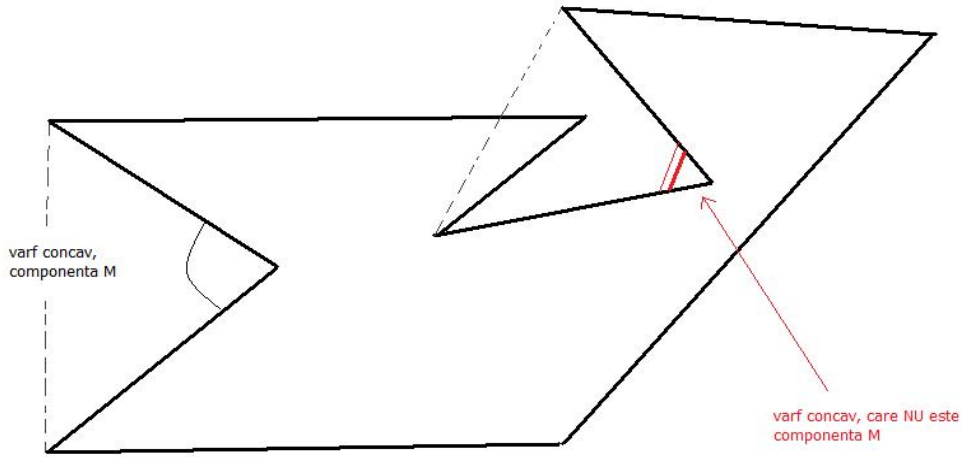
Teorema (2 - E)

Exceptând triunghiurile, orice poligon simplu are cel puțin 2 componente **E** *care nu se suprapun (disjuncte).*

Preliminarii: 2E & 1M

Definiția

Un vârf p_i al unui poligon simplu s. n. componentă **M** *dacă segmentul de dreaptă (p_{i-1}, p_{i+1}) este situat complet în exteriorul poligonului.*



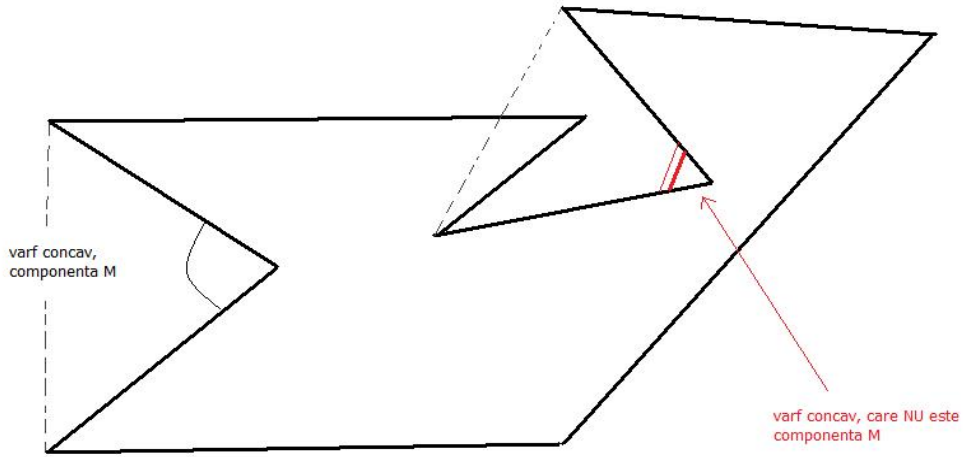
Teorema (1 - M)

Orice poligon concav are cel puțin o componentă **M**.

Preliminarii: 2E & 1M

Definiția

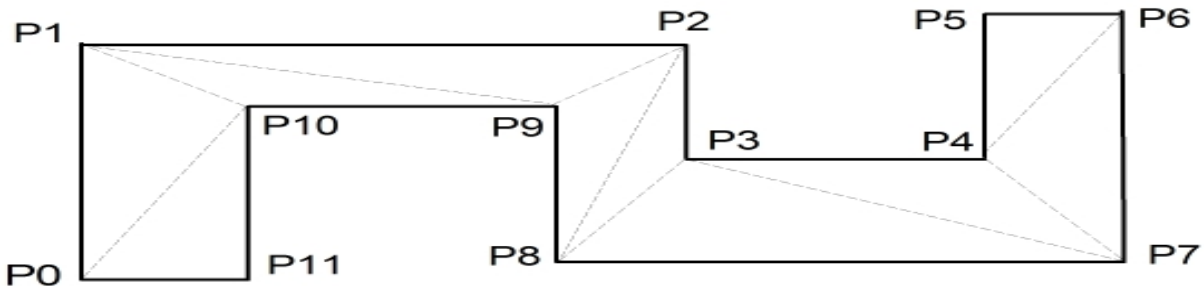
Un vârf p_i al unui poligon simplu s. n. componentă **M** *dacă segmentul de dreaptă (p_{i-1}, p_{i+1}) este situat complet în exteriorul poligonului.*



Teorema (1 - M)

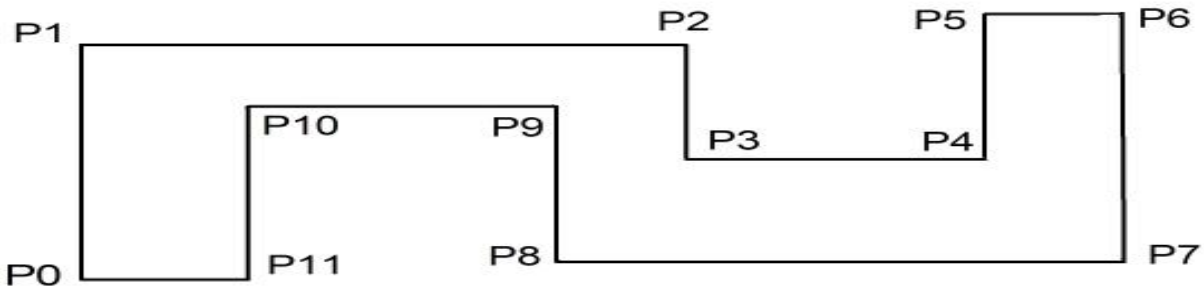
Orice poligon concav are cel puțin o componentă **M**.

Exemplu - aplicarea "ear cutting"



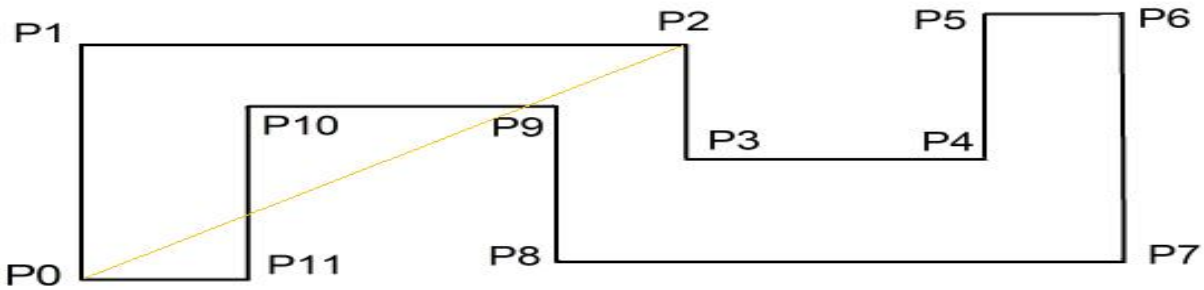
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



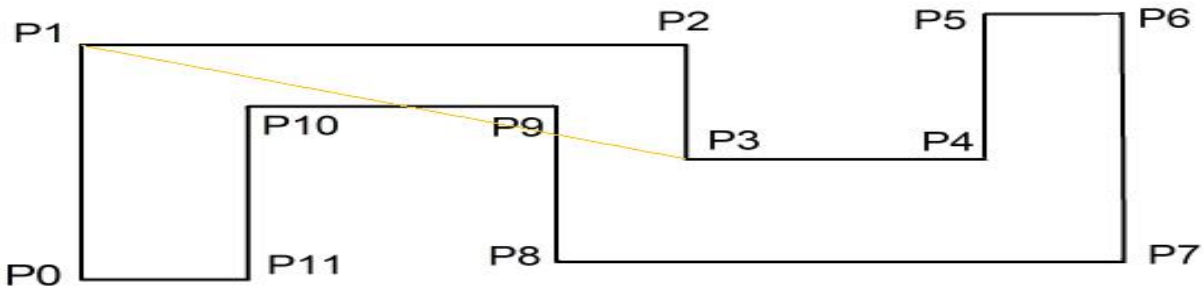
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



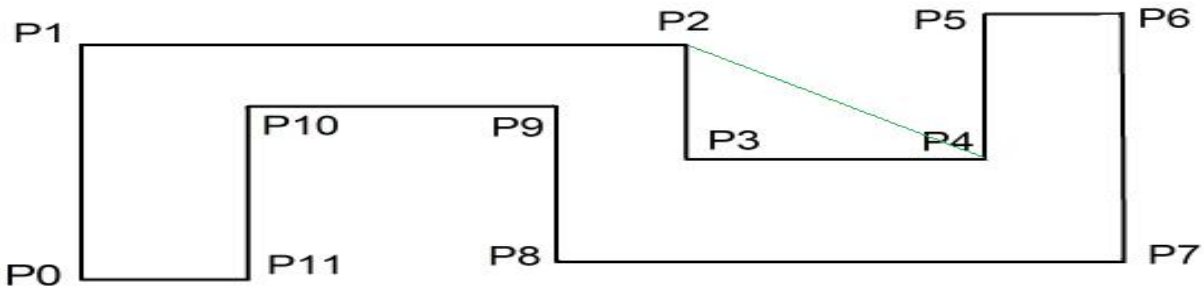
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



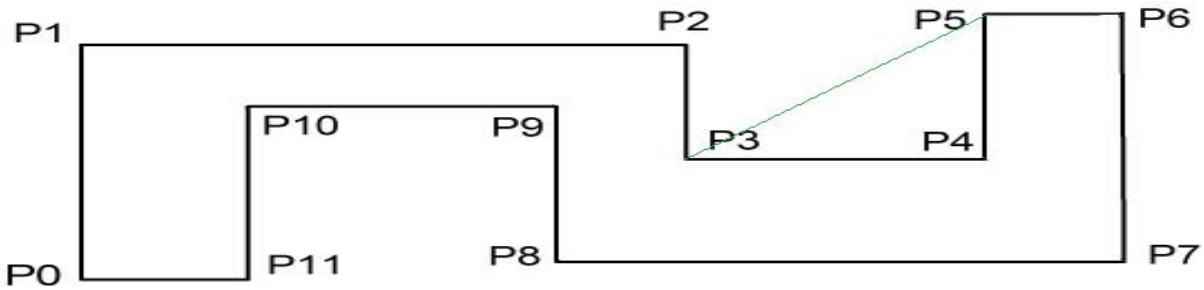
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



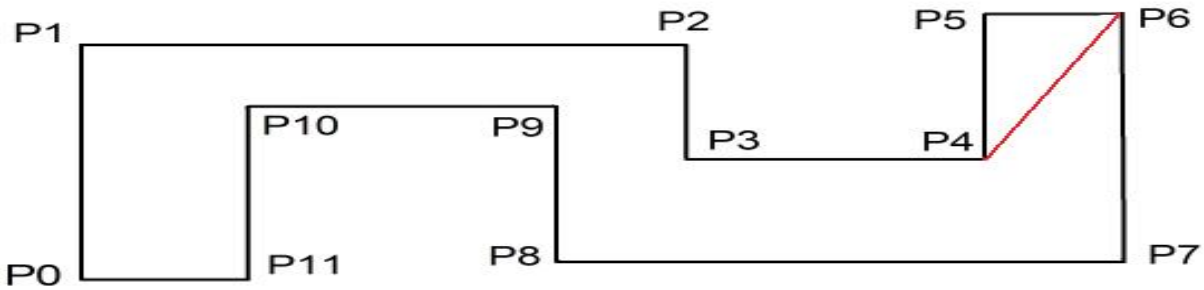
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



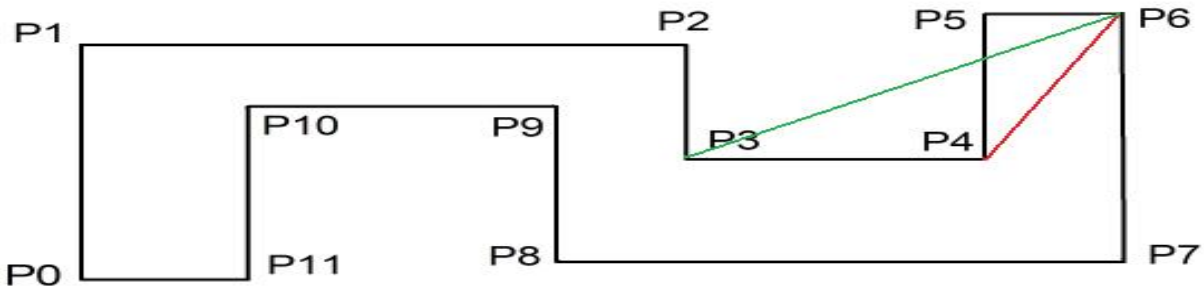
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



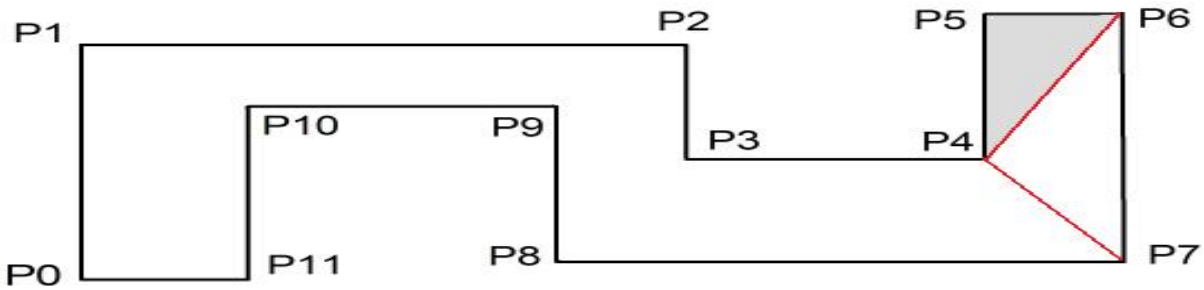
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



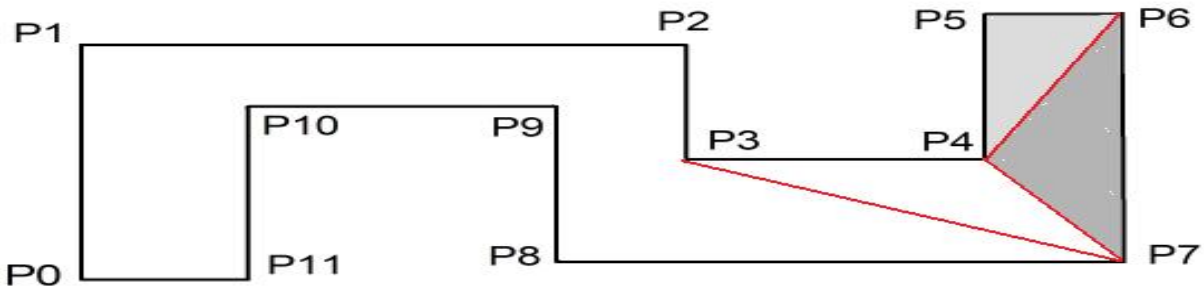
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



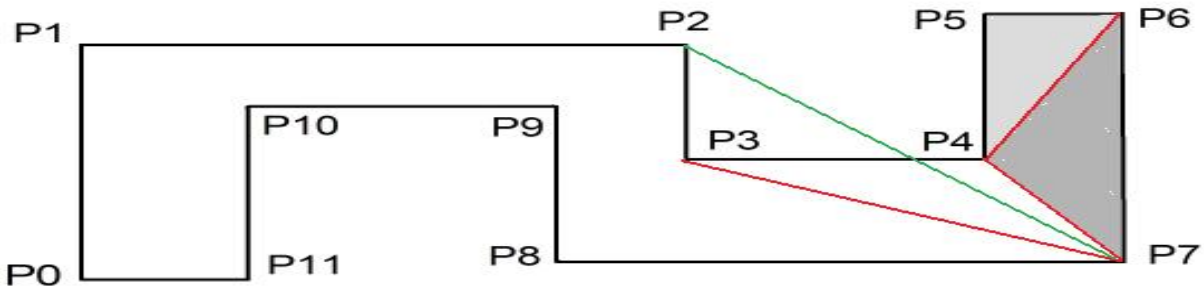
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



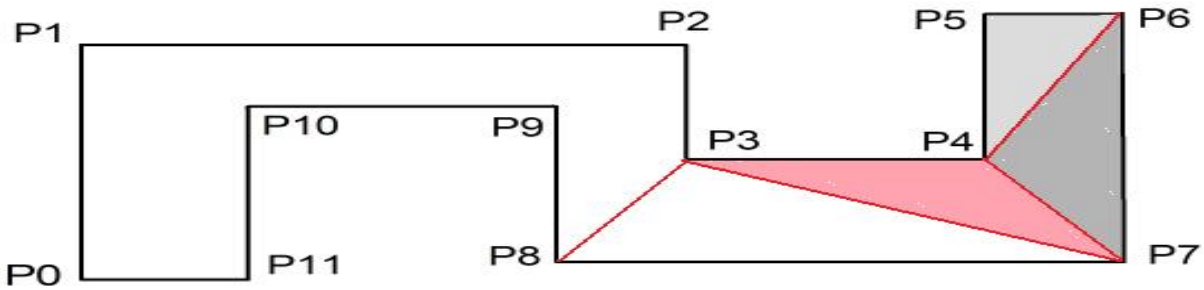
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



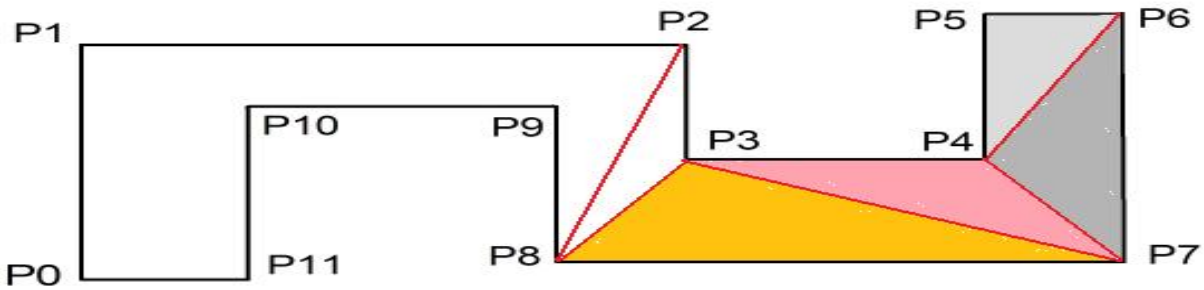
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



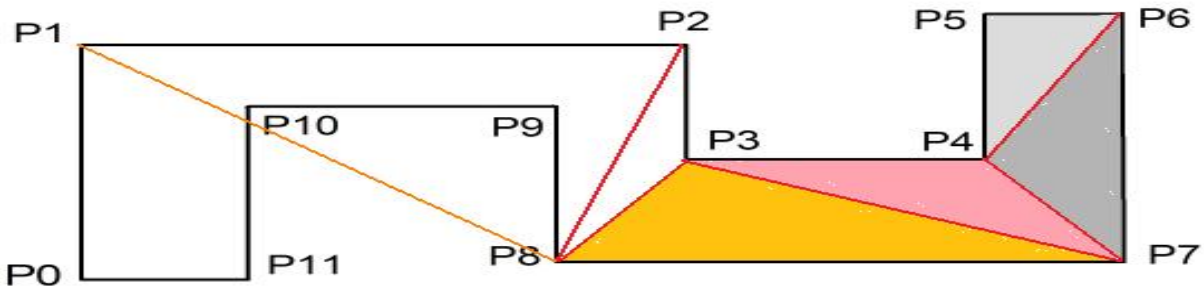
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



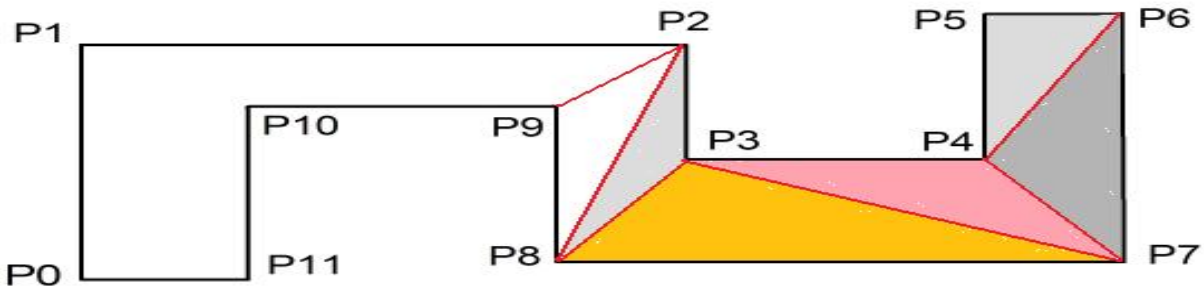
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



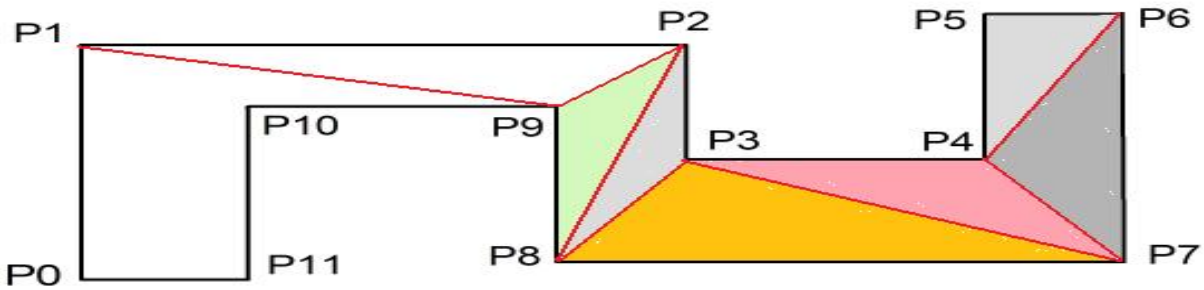
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



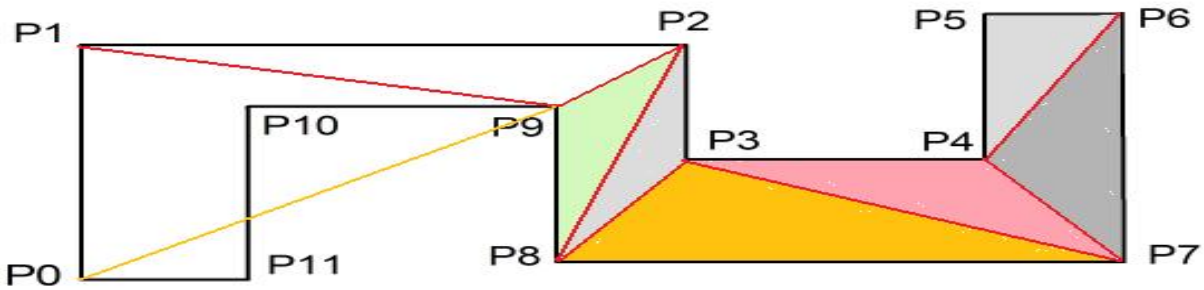
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



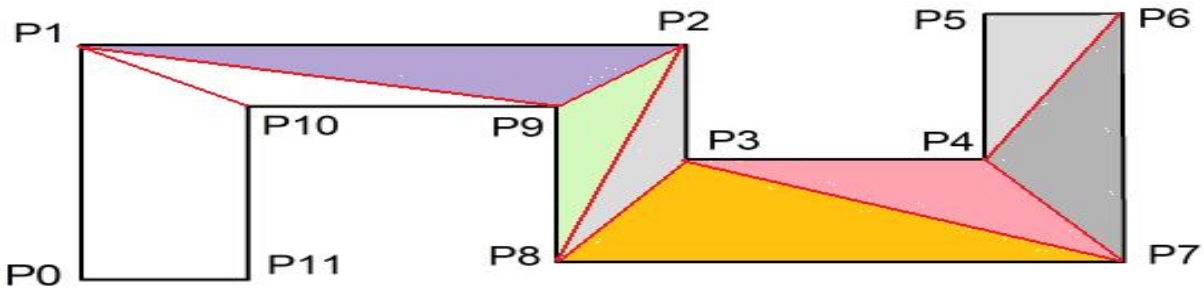
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



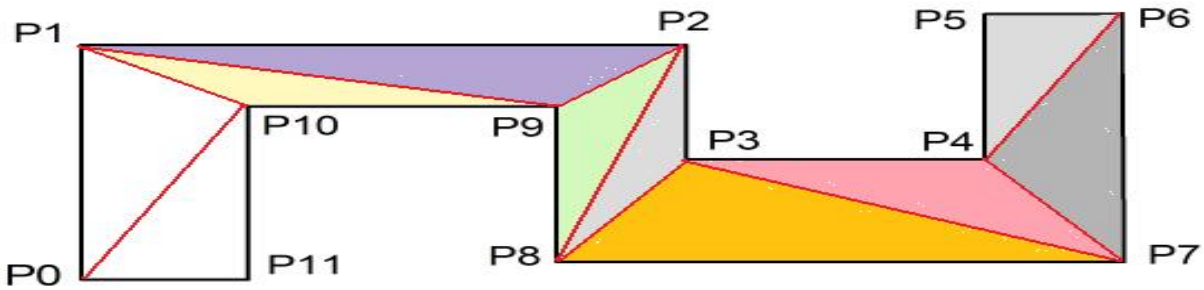
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



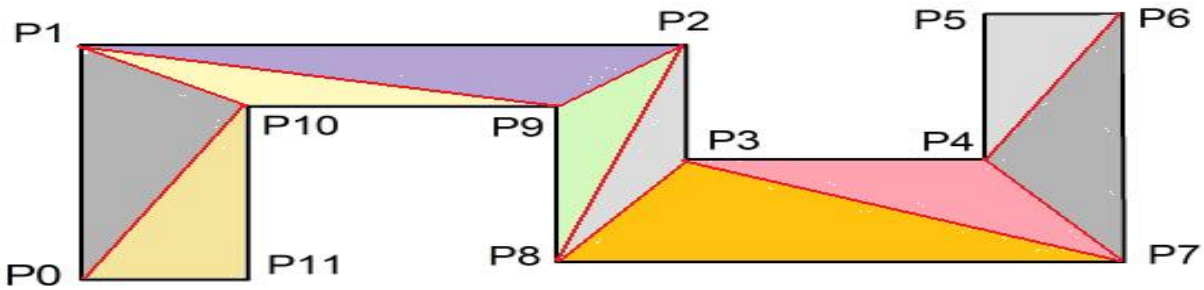
Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Exemplu - aplicarea "ear cutting"



Inițial, algoritmul testează p_1 pentru a determina dacă este sau nu o componentă **E**, fapt echivalent cu a testa dacă p_2 este sau nu vârful unei componente **E**. Scanarea este avansată peste p_2, p_3, p_4 și p_5 când p_5 este determinat ca fiind o componentă **E**. Vârful p_5 este decupat și atunci p_4 este testat, determinându-se că nu este o componentă **E**. Următorul vârf testat este p_6. Este găsit componentă **E** și decupat. Vârful p_4 este testat iar și de această dată este componentă **E** și este decupat.Vârfurile rămase vor fi decupate în ordinea următoare: p_7, p_3, p_8, p_2, p_9, p_1.

Algoritmul de triangulare a poligonului $\mathcal{P}$

Date de intrare : Un poligon simplu $\mathcal{P} = (p_0, p_1, p_2, ..., p_{n-1})$, sortat ca o listă dublu-înlănțuită.

Date de ieșire : Un set de diagonale $\mathcal{D}$ care determină o triangulare a lui $\mathcal{P}$.

Procedurile $SUCC(p_i)$ și $PRED(p_i)$ indică succesorul, respectiv predecesorul vârfului p_i. $\mathcal{R}$ este mulțimea tuturor vârfurilor concave ale lui $\mathcal{P}$.

$IsAnEar(\mathcal{P}, \mathcal{R}, p_i)$ este o funcție care returnează "adevărat" dacă p_i este o componentă **E** în poligonul $\mathcal{P}$ și returnează "fals" în caz contrar.

Triangulare($\mathcal{P}$)

1. $p_2 \rightarrow p_i$;
2. **cât timp** $p_i! = p_0$ **execută**
3. **dacă** $IsAnEar(\mathcal{P}, \mathcal{R}, PRED(p_i))$ și $\mathcal{P}$ nu este triunghi **atunci**
4. $\mathcal{D} := \mathcal{D} \cup (PRED(PRED(p_i)), p_i)$
5. $\mathcal{P} := \mathcal{P} - PRED(p_i)$
6. **dacă** $p_i \in \mathcal{R}$ și p_i este vârf convex, **atunci**
7. $\mathcal{R} := \mathcal{R} - p_i$
8. **dacă** $PRED(p_i) \in \mathcal{R}$ și $PRED(p_i)$ este vârf convex, **atunci**
9. $\mathcal{R} := \mathcal{R} - PRED(p_i)$
10. **dacă** $PRED(p_i) = p_0$ **atunci**
11. $p_i := SUCC(p_i)$
12. **altfel** $p_i := SUCC(p_i)$
13. **sfârșit**

sfârșit Triangulare($\mathcal{P}$)

Algoritmul de triangulare a poligonului $\mathcal{P}$

Date de intrare : Un poligon simplu $\mathcal{P} = (p_0, p_1, p_2, ..., p_{n-1})$, sortat ca o listă dublu-înlănțuită.

Date de ieșire : Un set de diagonale $\mathcal{D}$ care determină o triangulare a lui $\mathcal{P}$.

Procedurile $SUCC(p_i)$ și $PRED(p_i)$ indică succesorul, respectiv predecesorul vârfului p_i. $\mathcal{R}$ este mulțimea tuturor vârfurilor concave ale lui $\mathcal{P}$.

$IsAnEar(\mathcal{P}, \mathcal{R}, p_i)$ este o funcție care returnează "adevărat" dacă p_i este o componentă **E** în poligonul $\mathcal{P}$ și returnează "fals" în caz contrar.

Triangulare($\mathcal{P}$)

1. $p_2 \rightarrow p_i$;
2. **cât timp** $p_i! = p_0$ **execută**
3. **dacă** $IsAnEar(\mathcal{P}, \mathcal{R}, PRED(p_i))$ și $\mathcal{P}$ nu este triunghi **atunci**
4. $\mathcal{D} := \mathcal{D} \cup (PRED(PRED(p_i)), p_i)$
5. $\mathcal{P} := \mathcal{P} - PRED(p_i)$
6. **dacă** $p_i \in \mathcal{R}$ și p_i este vârf convex, **atunci**
7. $\mathcal{R} := \mathcal{R} - p_i$
8. **dacă** $PRED(p_i) \in \mathcal{R}$ și $PRED(p_i)$ este vârf convex, **atunci**
9. $\mathcal{R} := \mathcal{R} - PRED(p_i)$
10. **dacă** $PRED(p_i) = p_0$ **atunci**
11. $p_i := SUCC(p_i)$
12. **altfel** $p_i := SUCC(p_i)$
13. **sfârșit**

sfârșit Triangulare($\mathcal{P}$)

Algoritmul de triangulare a poligonului $\mathcal{P}$

Date de intrare : Un poligon simplu $\mathcal{P} = (p_0, p_1, p_2, ..., p_{n-1})$, sortat ca o listă dublu-înlănțuită.
Date de ieșire : Un set de diagonale $\mathcal{D}$ care determină o triangulare a lui $\mathcal{P}$.
Procedurile $SUCC(p_i)$ și $PRED(p_i)$ indică succesorul, respectiv predecesorul vârfului p_i. $\mathcal{R}$ este mulțimea tuturor vârfurilor concave ale lui $\mathcal{P}$.
$IsAnEar(\mathcal{P}, \mathcal{R}, p_i)$ este o funcție care returnează "adevărat" dacă p_i este o componentă **E** în poligonul $\mathcal{P}$ și returnează "fals" în caz contrar.

Triangulare($\mathcal{P}$)

1. $p_2 \rightarrow p_i$;
2. **cât timp** $p_i != p_0$ **execută**
3. **dacă** $IsAnEar(\mathcal{P}, \mathcal{R}, PRED(p_i))$ și $\mathcal{P}$ nu este triunghi **atunci**
4. $\mathcal{D} := \mathcal{D} \cup (PRED(PRED(p_i)), p_i)$
5. $\mathcal{P} := \mathcal{P} - PRED(p_i)$
6. **dacă** $p_i \in \mathcal{R}$ și p_i este vârf convex, **atunci**
7. $\mathcal{R} := \mathcal{R} - p_i$
8. **dacă** $PRED(p_i) \in \mathcal{R}$ și $PRED(p_i)$ este vârf convex, **atunci**
9. $\mathcal{R} := \mathcal{R} - PRED(p_i)$
10. **dacă** $PRED(p_i) = p_0$ **atunci**
11. $p_i := SUCC(p_i)$
12. **altfel** $p_i := SUCC(p_i)$
13. **sfârșit**

sfârșit Triangulare($\mathcal{P}$)

Algoritmul de triangulare a poligonului $\mathcal{P}$

funcția *IsAnEar*$(\mathcal{P}, \mathcal{R}, p_j)$
1. **dacă** $\mathcal{R} = \varnothing$ **atunci** returnează "adevărat"
2. **altfel dacă** p_j este un vârf convex, **atunci**
3. **dacă** $\triangle(PRED(p_j), p_j), SUCC(p_j)$ nu conține nici un vârf al lui $\mathcal{R}$
4. **atunci** returnează "adevărat"
5. **altfel** returnează "fals"
6. **altfel** returnează "fals"

sfârșit *IsAnEar*

Lema

La fiecare apel al funcției IsAnEar, $\mathcal{R}$ este formată doar din vârfurile concave ale lui $\mathcal{P}$.

Lema

Dacă un vârf convex p_j nu este o componentă **E** *atunci triunghiul $\triangle(p_{j-1}, p_j, p_{j+1})$ conține un vârf concav.*

Algoritmul de triangulare a poligonului $\mathcal{P}$

funcția *IsAnEar*$(\mathcal{P}, \mathcal{R}, p_j)$
1. **dacă** $\mathcal{R} = \varnothing$ **atunci** returnează "adevărat"
2. **altfel dacă** p_j este un vârf convex, **atunci**
3. **dacă** $\triangle(PRED(p_j), p_j), SUCC(p_j)$ nu conține nici un vârf al lui $\mathcal{R}$
4. **atunci** returnează "adevărat"
5. **altfel** returnează "fals"
6. **altfel** returnează "fals"

sfârșit *IsAnEar*

Lema

La fiecare apel al funcției IsAnEar, $\mathcal{R}$ este formată doar din vârfurile concave ale lui $\mathcal{P}$.

Lema

Dacă un vârf convex p_j nu este o componentă **E** *atunci triunghiul $\triangle(p_{j-1}, p_j, p_{j+1})$ conține un vârf concav.*

Algoritmul de triangulare a poligonului $\mathcal{P}$

funcția *IsAnEar*$(\mathcal{P}, \mathcal{R}, p_j)$
1. **dacă** $\mathcal{R} = \varnothing$ **atunci** returnează "adevărat"
2. **altfel dacă** p_j este un vârf convex, **atunci**
3. **dacă** $\triangle(PRED(p_j), p_j), SUCC(p_j)$ nu conține nici un vârf al lui $\mathcal{R}$
4. **atunci** returnează "adevărat"
5. **altfel** returnează "fals"
6. **altfel** returnează "fals"

sfârșit *IsAnEar*

Lema

La fiecare apel al funcției IsAnEar, $\mathcal{R}$ este formată doar din vârfurile concave ale lui $\mathcal{P}$.

Lema

Dacă un vârf convex p_j nu este o componentă **E** *atunci triunghiul $\triangle(p_{j-1}, p_j, p_{j+1})$ conține un vârf concav.*