

Grațiela Cicortăș

Delia Tușe

**TEME DE GEOMETRIE
COMPUTAȚIONALĂ**

Casa Cărții de Știință
Cluj-Napoca, 2010

© Grajela Cicoraș, Delia Tușe, 2010

Descrierea CIP este disponibilă la Biblioteca Națională a României
ISBN 978-973-133-831-6

Director: Mircea Trifu
Fondator: dr. T.A. Codreanu

Tiparul executat la Casa Cărții de Știință
400129 Cluj-Napoca; B-dul Eroilor nr. 6-8
Tel./fax: 0264-431920
www.casacartii.ro; e-mail: editura@casacartii.ro

Introducere

Geometria computațională este un domeniu complex, situat la frontieră dintre matematică și informatică, consacrat studiului unor algoritmi ce pot fi formulați în termeni geometrici. Aparut în anii '70, acest domeniu a atras, încă de la început, interesul unui număr mare de cercetători, datorită interacțiunii puternice cu diferite ramuri ale științei: algoritmi și structuri de date, matematică combinatorială, geometrie euclidiană, optimizare etc.

Scopul geometriei computaționale constă în reprezentarea unor obiecte 2 sau 3-dimensionale (există, însă, probleme care au fost rezolvate, mai nou, în cazul d -dimensional, $d \geq 4$) și dezvoltarea unor algoritmi eficienți pentru manipularea obiectelor respective.

Reușește geometria computațională să satisfacă cerințele algoritmice ale practicenilor? Poate să distingă în domeniile aplicative noi surse de probleme? Geometria computațională se bucură de două calități importante:

- (i) diversitatea sa și potențialul de a simula cele mai multe forme de calcul;
- (ii) fundamentul algoritic solid.

Acestea permit abordarea și rezolvarea satisfăcătoare a unor probleme importante de robotică, grafică pe calculator, astrofizică, modelare moleculară, bioinformatică, vizualizare științifică, sisteme de informații geografice, sisteme de circuite integrate, inginerie asistată de calculator.

Pe de altă parte, există motive că se poate de simple pentru care geometria computațională nu poate să rezolve toate problemele furnizate de astfel de domenii. Un exemplu este natura discretă a geometriei computaționale; orice problemă rezolvată de calculator trebuie exprimată într-o formă discretă, însă în multe aplicații se folosesc aproximări discrete ale unor fenomene caracterizate prin continuitate. O altă restricție apare datorită faptului că obiectele folosite în geometria computațională sunt linii sau figuri "plate"; în multe probleme obiectele ar trebui modelate cu ajutorul unor suprafete curbate. Nu în ultimul rând, dimensiunea spațiului problemei de rezolvat depășește, în anumite situații, 2 sau 3, iar problemele rezolvate în geometria computațională pentru dimensiune mare sunt puține.

În funcție de problemele rezolvate și metodele utilizate, se pot distinge două ramuri importante ale acestui domeniu complex:

- ◊ geometria computațională combinatorială (numită uneori geometrie algoritmică), în care obiectele geometrice de bază sunt puncte, linii, segmente, poligoane, poliedre, iar problemele rezolvate pot fi statice, de căutare sau probleme dinamice;
- ◊ geometria computațională numerică, unde problemele centrale sunt modelarea suprafetelor și reprezentarea lor, iar instrumentele utilizate sunt curbe Ferguson, curbe Bezier, curbe spline, plăci curbe Coons sau plăci curbe Bezier.

Lucrarea de față abordează probleme centrale din Geometria computațională combinatorială precum și algoritmi clasici de rezolvare a acestora. Materialul prezentat reprezintă o versiune amplu dezvoltată și de sine stătătoare a cursului de un semestru adresat studenților secțiilor de Informatică respectiv Matematică informatică.

În primul capitol am inclus câteva noțiuni referitoare la complexitatea algoritmilor, după care am prezentat o primă problema de geometrie computațională - intersecția segmentelor în plan, precum și unele elemente de teoria grafurilor. Cel de-al doilea capitol este dedicat studiului învelitorii convexe a unei mulțimi finite de puncte din plan și a unor algoritmi de determinare a acesteia, printre care algoritmul lui Jarvis, respectiv Graham. Capitolul 3 debutează cu prezentarea problemei galeriei de artă. Se studiază existența triangulării unui poligon simplu, precum și proprietățile acesteia. În încheiere am prezentat metoda triangulării prin eliminarea urechilor. Capitolul 4 este consacrat triangulării poligoanelor monotone. Se studiază, de asemenea, partitioanarea poligoanelor simple în poligoane monotone, precum și câteva rezultate legate de partitioanarea convexă a poligoanelor simple. Capitolul 5 tratează o altă problemă importantă din geometria computațională: intersecția unei familii de semiplane. Aici se studiază, cu ajutorul listei dublu conectate a muchiilor, suprapunerea a două subdiviziuni ale planului, după care se analizează problema intersecției a două regiuni convexe din plan. În capitolul 6 sunt stabilite proprietățile de bază ale diagramei Voronoi, precum și un algoritm de construcție a acesteia. Ultimul capitol al lucrării este consacrat studiului triangulării Delaunay.

Capitolul 1

Preliminarii

1.1 Complexitatea algoritmilor

Analiza complexității unui algoritm are drept scop estimarea volumului de resurse de calcul necesare execuției algoritmului:

- spațiul de memorie necesar stocării datelor ce trebuie prelucrate de algoritm;
- timpul necesar execuției prelucrărilor respective.

Astfel se stabilește dacă un algoritm utilizează un volum acceptabil de resurse în scopul rezolvării unei probleme date. Dacă spațiul de memorie sau timpul de execuție sunt prea mari, algoritmul nu este eficient; în concluzie, nu va fi aplicat în practică.

Această analiză permite compararea diverselor soluții algoritmice pentru aceeași problemă. În general, resursa timp este mai importantă decât resursa spațiu de memorie; sunt însă situații în care se stabilește un compromis între cele două cerințe.

Este clar faptul că volumul resurselor de calcul necesare execuției unui algoritm depinde de dimensiunea problemei de rezolvat, aceasta fiind determinată de volumul datelor de intrare.

Există două metode de măsurare a eficienței unui algoritm:

- analiza matematică a algoritmului (analiza asimptotică);
- analiza empirică, care măsoară timpul exact de rulare pentru date specifice.

Timpul de execuție reprezintă o măsură a numărului de pași pe care îi execută un algoritm (numărul de repetări ale unor operații); el nu este timpul real de execuție, însă este proporțional cu acesta. Prin analiza asimptotică a algoritmului se stabilește rata de creștere a timpului de execuție în funcție de volumul setului de date de intrare; mai exact, aceasta exprimă creșterea timpului de execuție în cazul creșterii spre infinit a volumului setului de date de intrare. Este clar faptul că rata de creștere este determinată de termenul dominant din expresia timpului de execuție. Astfel este suficient să contorizăm operația dominantă, adică operația cea mai costisitoare și mai des executată.

Pentru anumite probleme timpul de execuție depinde nu doar de dimensiunea problemei, ci și de caracteristicile datelor de intrare. Astfel timpul de execuție va fi cuprins între cel obținut în cazul cel mai defavorabil (numărul operațiilor efectuate este cel mai mare) respectiv în cazul cel mai favorabil (numărul de operații efectuate este minim).

Este important faptul că analiza celui mai defavorabil caz furnizează timpul maxim de execuție relativ la orice date de intrare de dimensiune precizată.

În analiza asymptotică a algoritmilor se utilizează notațiile următoare:

- ◊ $O(g(n))$ reprezintă clasa funcțiilor care cresc asymptotic cel mult la fel de repede ca $g(n)$ atunci când $n \rightarrow \infty$:

$$O(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}_+ \mid \exists c \in \mathbb{R}_+^*, n_0 \in \mathbb{N} \text{ astfel încât } 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$$

Altfel spus, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = k$, unde k este o valoare pozitivă, nu neapărat nenulă.

Această clasă de funcții permite descrierea timpului de execuție al unui algoritm în cazul cel mai defavorabil. Fiind importantă comportarea algoritmului pentru date arbitrară de intrare, este suficientă specificarea unei margini superioare a timpului de execuție.

- ◊ $\Omega(g(n))$ exprimă clasa funcțiilor care cresc asymptotic cel puțin la fel de repede ca $g(n)$ atunci când $n \rightarrow \infty$:

$$\Omega(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}_+ \mid \exists c \in \mathbb{R}_+^*, n_0 \in \mathbb{N} \text{ astfel încât } cg(n) \leq f(n), \forall n \geq n_0\}$$

Altfel spus, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \geq k$, unde k este o valoare pozitivă, nu neapărat finită.

Această clasă de funcții este utilă pentru descrierea comportării unui algoritm în cel mai favorabil caz.

- ◊ $\Theta(g(n))$ reprezintă clasa funcțiilor care cresc cu aceeași rată ca și $g(n)$ atunci când $n \rightarrow \infty$:

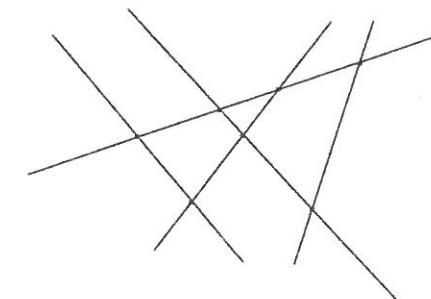
$$\Theta(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}_+ \mid \exists c_1, c_2 \in \mathbb{R}_+^*, n_0 \in \mathbb{N} \text{ a.î. } c_1g(n) \leq f(n) \leq c_2g(n), \forall n \geq n_0\}$$

Dacă $f(n) \in \Theta(g(n))$, spunem că f și g sunt asymptotic echivalente; în acest caz vom avea $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = k$, unde k este o valoare finită strict pozitivă.

Fie \mathcal{A} un algoritm. Vom spune că \mathcal{A} are complexitatea $\mathcal{O}(f(n))$ dacă există o constantă c astfel încât pentru orice $n \in \mathbb{N}^*$ timpul de execuție este cel mult $cf(n)$ pentru toate datele de intrare de dimensiune n .

1.2 O problemă de intersecție

În această secțiune ne propunem să rezolvăm următoarea problemă: se dă în plan o mulțime S de segmente de dreaptă închise și se cere să se determine toate intersecțiile dintre segmentele din S .



Pentru rezolvarea acestei probleme, evident că am putea utiliza soluția grosieră care constă în verificarea pe rând a tuturor segmentelor între ele și dacă se intersecțează, stabilim punctul de intersecție.

Vom descrie însă în continuare un algoritm care rezolvă această problemă și care este mai optim decât soluția grosieră.

Principiul pe care se bazează acest algoritm este un principiu clasic din geometria computațională care stă la baza multor alți algoritmi de altfel.

Pentru început vom defini câteva noțiuni utilizate în descrierea algoritmului. Considerăm punctele planului raportate la un reper cartezian fixat.

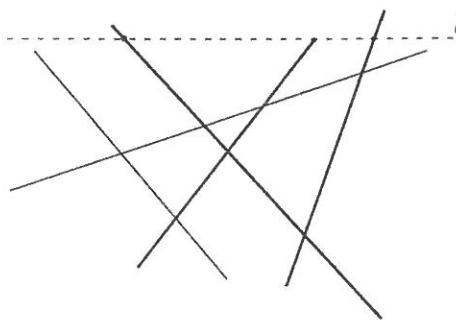
Definiția 1.2.1. Numim **y-intervalul** unui segment proiecția acestuia pe axa Oy .

Observația 1.2.1. Este evident că dacă y-intervalele a două segmente nu se intersecțează, atunci nici segmentele nu se intersecțează. Reciproca acestei afirmații nu este însă adevărată. Un exemplu în acest sens îl constituie segmentele paralele cu axa Oy .

Vom considera o dreaptă orizontală care "mătură" planul, plecând dintr-un punct situat deasupra tuturor segmentelor și vom reține toate intersecțiile dintre această dreaptă și segmentele date. Acest lucru este doar intuitiv, întrucât linia de măsurare nu este reprezentată în mod explicit de către algoritm.

Definiția 1.2.2. Numim această dreaptă orizontală **linie de măsurare** și o notăm cu l .

Definiția 1.2.3. Se numește **stare** a liniei de măsurare l mulțimea de segmente din S care o intersecțează la un moment dat.



Definiția 1.2.4. Punctele în care starea se schimbă se numesc **evenimente**.

Starea liniei de măturare se modifică pe măsură ce dreapta se mișcă. Pentru optimizarea algoritmului, vom considera că dreapta nu se mișcă în mod continuu, adică punct cu punct, ci "sare" doar în pozițiile în care se ajunge la un eveniment.

Este evident că primele evenimente vor fi chiar capetele segmentelor.

Momentele în care dreapta l ajunge la un eveniment sunt singurele în care algoritmul face ceva, și anume actualizează starea și face teste de intersecție între segmentele aflate în stare.

Actualizarea stării constă în adăugarea unui segment la stare atunci când evenimentul este capătul superior al segmentului și respectiv în înlăturarea unui element din stare, atunci când evenimentul este capătul inferior al segmentului. Un segment va fi inserat în stare doar dacă nu există deja. Două evenimente coincid de exemplu dacă două segmente se intersectează în capete. Dacă sunt mai multe segmente care trebuie să fie înlăturate din stare, se elimină cel mai din stânga.

În plus, pentru a optimiza algoritmul, segmentele din stare se vor ordona de la stânga la dreapta în funcție de coordonata x a punctelor de intersecție cu linia de măturare l , testându-se astfel pentru intersecție numai segmentele adiacente din stare și considerând doar punctele de intersecție aflate sub linia de măturare, celelalte fiind deja determinate. În acest caz nu vor fi testate pentru intersecție și segmentele care sunt îndepărțate pe orizontală.

Ca structuri de date atât pentru coada de evenimente, cât și pentru starea algoritmului vom folosi arborii binari de căutare.

În cazul evenimentelor, acesta va fi ordonat după relația de ordine " \prec " definită astfel:

$$p \prec q \Leftrightarrow (p_y > q_y) \text{ sau } (p_y = q_y \text{ și } p_x < q_x),$$

unde p și q sunt două puncte, iar p_x respectiv p_y notează coordonatele carteziene ale acestora în raport cu reperul dat.

Arboarele binar de căutare care reprezintă starea algoritmului va fi ordonat după

relația de ordine " $<$ " definită astfel:

$$s_1 < s_2 \Leftrightarrow i_{1_x} \leq i_{2_x},$$

unde s_1 și s_2 sunt două segmente din S , iar $i_1 = s_1 \cap l$ și $i_2 = s_2 \cap l$.

Prezentăm, în continuare, **algoritmul de măturare plană**:

Se dă: o mulțime S de segmente.

Se cere: intersecțiile segmentelor din S .

- (1) adaugă capetele segmentelor în coada de evenimente
- (2) pentru fiecare eveniment execută
 - (3) actualizează starea algoritmului
 - (4) pentru segmentele adiacente din stare execută
 - (5) dacă se intersectează atunci afișează intersecția
 - (6)

Observația 1.2.2. Complexitatea algoritmului de măturare plană este $\mathcal{O}(n \log n)$.

Observația 1.2.3. Această metodă a fost utilizată, după prezentarea ei de către Shamos și Hoey, pentru a proiecta algoritmi eficienți pentru o serie de probleme, cum ar fi de exemplu construcția diagramei Voronoi (algoritmul lui Fortune), triangularea Delaunay sau operații booleene pe poligoane.

1.3 Preliminarii geometrice

Din punct de vedere intuitiv, un poligon este o regiune a planului delimitată de o mulțime finită de segmente închise de dreapta astfel încât ele formează o "curbă" simplă închisă.

Definiția 1.3.1. Fie v_1, v_2, \dots, v_n n puncte din plan. Fie $e_1 = v_1v_2, e_2 = v_2v_3, \dots, e_i = v_iv_{i+1}, \dots, e_n = v_nv_1$ n segmente care unesc aceste puncte. Spunem că aceste segmente delimită un **poligon simplu** dacă:

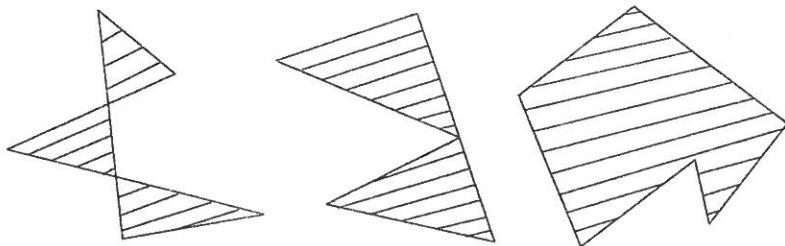
- (i) $e_i \cap e_{i+1} = v_{i+1}$, $i = \overline{1, n}$, cu $e_{n+1} = e_1$ și $v_{n+1} = v_1$ (intersecția oricărei perechi de segmente adiacente în ordonarea ciclică este un punct unic);
- (ii) $e_i \cap e_j = \emptyset$, $i, j = \overline{1, n}$, $i + 1 \neq j$ (segmentele neadiacente nu se intersectează).

Aceste segmente definesc o curbă închisă în sensul că ele au căte două un capăt comun. "Curba" este simplă datorită condiției (ii).

Punctele v_1, \dots, v_n se numesc **vârfurile poligonului**, iar segmentele e_1, \dots, e_n se numesc **laturi**.

Vom folosi notațiile următoare: $P = v_1 \dots v_n$ este poligonul de vârfuri v_1, \dots, v_n și $\partial P = e_1 \dots e_n$ este frontieră poligonului P . Observăm că $\partial P \subseteq P$; orice poligon simplu este reuniunea frontierei sale cu interiorul poligonului.

Exemplul 1.3.1. Primele două figuri de mai jos reprezintă poligoane care nu sunt simple, iar ultima figură reprezintă un poligon simplu.



Definiția 1.3.2. O mulțime $S \subset E_2$ se numește **mulțime convexă** dacă $\forall p, q \in S$, segmentul $pq \subset S$.

Pe baza definiției se demonstrează ușor proprietatea următoare:

Propoziția 1.3.1. *Intersecția a două mulțimi convexe este o mulțime convexă.*

Definiția 1.3.3. Un **poligon convex** este un poligon simplu care este și mulțime convexă.

1.4 Grafuri geometrice în plan

Definiția 1.4.1. Fie $G = (V, E)$ un graf. G se numește **graf planar** dacă el poate fi scufundat în plan fără ca imaginile muchiilor să se intersecteze în puncte interioare.

Definiția 1.4.2. O **scufundare planară** sau **plană** a unui graf este o aplicație care asociază fiecărui vârf un punct din plan și fiecărei muchii un arc de curbă simplă (continuă) din plan cu extremitățile în două imagini de vârfuri ale grafului, astfel încât orice două imagini de muchii se pot intersecta doar la extremități.

Definiția 1.4.3. Imaginea unui graf planar printr-o scufundare plană se numește **graf geometric** sau **subdiviziune a planului**.

Definiția 1.4.4. Dacă imaginile muchiilor sunt segmente de dreaptă, graful planar se numește **liniar**, iar scufundarea plană se numește **liniară**.

Prezentăm, fără demonstrație, următoarea teoremă importantă din teoria grafurilor:

Teorema 1.4.1 (Kuratowski-Wagner). *Orice graf planar are o scufundare plană liniară.*

Definiția 1.4.5. Fie un graf planar G și subdiviziunea planului corespunzătoare lui. Numim **față** sau **regiune** a lui G o porțiune din plan, maximală în raport cu incluziunea, în care oricare două puncte pot fi unite cu un arc astfel încât nici un punct al acestuia nu corespunde unui vârf și nu aparține imaginii unei muchii a lui G . Frontiera unei fețe f este formată din acele puncte x care corespund vârfurilor și muchiilor lui G astfel încât x poate fi unit cu orice punct al lui f printr-un arc de curbă conținut în f .

Observația 1.4.1. Orice graf planar are o unică față nemărginită, numită **față exterioară sau infinită**.

Un rezultat central referitor la grafuri planare este dat de teorema următoare:

Teorema 1.4.2 (Euler). *Fie v, e, f numărul vârfurilor, muchiilor, respectiv fețelor (înclusiv cea exterioară) ale unui graf planar liniar conex. Are loc formula lui Euler:*

$$v - e + f = 2$$

Dacă graful are n componente conexe, atunci formula devine:

$$v - e + f = n + 1.$$

Demonstrație. Pentru început, presupunem că graful este conex. Vom demonstra formula lui Euler prin inducție relativ la numărul fețelor grafului. Dacă $f = 1$, graful are o singură față (infinită). Atunci acesta este un arbore și avem $v = e + 1$. Presupunem că formula de mai sus are loc pentru orice graf având cel mult $f - 1$ fețe. Fie un graf cu f fețe și fie o muchie comună fețelor f și f' . Stergem această muchie; obținem un graf cu $f - 1$ fețe și $e - 1$ muchii. Din ipoteza de inducție, avem $v - (e - 1) + (f - 1) = 2$, adică $v - e + f = 2$.

Presupunem, în continuare, că graful are n componente conexe. Pentru fiecare dintre acestea notăm cu v_i, e_i, f_i numărul vârfurilor, muchiilor respectiv fețelor. Este clar că au loc egalitățile $v_i - e_i + f_i = 2$, $i = \overline{1, n}$. Prin adunarea lor avem: $\sum_{i=1}^n v_i - \sum_{i=1}^n e_i + \sum_{i=1}^n f_i = 2n$. Pe de altă parte, $v = \sum_{i=1}^n v_i$, $e = \sum_{i=1}^n e_i$, $f + (n - 1) = \sum_{i=1}^n f_i$, unde termenul $n - 1$ din ultima egalitate provine din faptul că față infinită trebuie numărată o singură dată. Înlocuind în egalitatea de mai sus rezultă $v - e + f = 2n - (n - 1) = n + 1$. \square

Definiția 1.4.6. Un graf planar se numește **maximal** dacă prin adăugarea unei muchii graful obținut nu ar mai fi planar.

Observația 1.4.2. Pentru un astfel de graf, toate fețele sunt triunghiuri.

Propoziția 1.4.1. *Dacă G este un graf planar maximal cu $v \geq 3$ atunci*

$$e = 3v - 6.$$

Demonstrație. Fie f numărul fețelor lui G .

Deoarece fiecare față are trei muchii și fiecare muchie este frontiera a două fețe, rezultă că are loc egalitatea $3f = 2e$. Pe de altă parte, din egalitățile $3f = 2e$, $v - e + f = 2$ obținem $6 + 3e - 3v = 2e$, adică $e = 3v - 6$. \square

Propoziția 1.4.2. Dacă G este un graf planar, atunci $e \leq 3v - 6$.

Demonstrație. Dacă G este maximal, din Propoziția 1.4.1 obținem egalitatea. Dacă G nu este maximal, adăugăm muchii până obținem un graf maximal G' , pentru care $e' = 3v - 6$. Din $e \leq e'$ rezultă $e \leq 3v - 6$. \square

Propoziția 1.4.3. Orice graf planar conține un vârf de grad cel mult 5.

Demonstrație. Dacă $v \leq 5$, afirmația este adevărată.

Presupunem $v \geq 6$. Observăm că pentru suma gradelor tuturor vârfurilor avem următoarea proprietate:

$$\sum_{i=1}^v d(i) = 2e \leq 6v - 12$$

Dacă presupunem că $d(i) \geq 6$, $\forall i = \overline{1, v}$, atunci obținem

$$2e = \sum_{i=1}^v d(i) \geq 6v.$$

Din ultimele două relații rezultă

$$6v \leq \sum_{i=1}^v d(i) \leq 6v - 12,$$

ceea ce este imposibil. \square

Capitolul 2

Învelitori convexe în plan

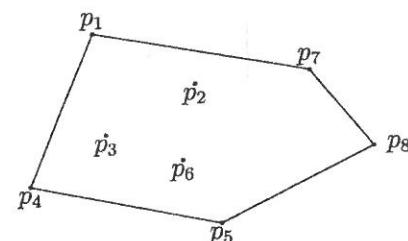
2.1 Învelitoarea convexă a unei mulțimi finite de puncte din plan

Definiția 2.1.1. Învelitoarea convexă a unei mulțimi oarecare $S \subset E_2$ este cea mai mică mulțime convexă din plan care conține mulțimea S .

Vom nota în continuare învelitoarea convexă a mulțimii S cu $CH(S)$, pornind de la denumirea în limba engleză, *convex hull*.

Exemplul 2.1.1. Dacă $S = \{p_1, p_2\}$ este o mulțime de două puncte în plan, atunci învelitoarea convexă a mulțimii S este segmentul închis p_1p_2 . Dacă $S = \{p_1, p_2, p_3\}$ este o mulțime de trei puncte în plan, atunci învelitoarea convexă a mulțimii S este triunghiul $\Delta p_1p_2p_3$ reunit cu interiorul său. Dacă $S = \{p_1, p_2, p_3, p_4\}$ este o mulțime de patru puncte în plan, învelitoarea convexă a mulțimii S nu mai poate fi precizată în mod unic, aceasta depinzând de poziția punctelor.

Exemplul 2.1.2. Învelitoarea convexă a mulțimii reprezentate în figura următoare este $p_1p_7p_8p_5p_4$.



Observația 2.1.1. Pe baza Definiției 2.1.1 și a Propoziției 1.3.1 deducem că învelitoarea convexă a unei mulțimi S reprezintă intersecția tuturor mulțimilor convexe din plan care conțin mulțimea S .

Ne propunem în acest capitol să determinăm învelitoarea convexă a unei mulțimi finite S de puncte din plan.

Teorema 2.1.1. *Învelitoarea convexă a unei mulțimi finite S de puncte din plan este unicul poligon convex ale căruia vârfuri sunt puncte din S și care conține toate punctele din S .*

Demonstrație. Demonstrăm prin reducere la absurd faptul că mulțimea $CH(S)$ este conexă. Astfel, presupunem că $CH(S)$ are mai multe componente conexe și fie p_1, p_2 două puncte din componente conexe distincte ale mulțimii $CH(S)$. Atunci segmentul p_1p_2 nu este conținut în $CH(S)$, deci $CH(S)$ nu este o mulțime convexă, ceea ce reprezintă o contradicție.

Demonstrăm că $CH(S)$ este mulțime mărginită. S fiind mulțime mărginită, ea este conținută într-un disc D de rază suficient de mare. Discul fiind mulțime convexă, rezultă că $CH(S) \subset D$, deoarece $CH(S)$ este cea mai mică mulțime convexă care conține mulțimea S . În concluzie, mulțimea $CH(S)$ este mărginită.

Fie $p_1, p_2 \in S$ astfel încât toate punctele lui S sunt de aceeași parte a dreptei determinată de p_1 și p_2 . Atunci segmentul p_1p_2 este situat pe frontieră lui $CH(S)$, adică $p_1p_2 \subseteq \partial CH(S)$.

Observăm că $p_1p_2 \subset CH(S)$: p_1p_2 este mulțimea tuturor combinațiilor convexe ale punctelor p_1 și p_2 , adică mulțimea elementelor de forma $\alpha p_1 + \beta p_2$, cu $\alpha, \beta \geq 0$, $\alpha + \beta = 1$. Aici, prin $\alpha p_1 + \beta p_2 = q$ înțelegem $\alpha \cdot \overrightarrow{OP_1} + \beta \cdot \overrightarrow{OP_2} = \overrightarrow{OQ}$.

Pe altă parte, $CH(S)$ este mulțimea tuturor combinațiilor convexe ale punctelor din S , adică $\alpha_1 p_1 + \alpha_2 p_2 + \dots + \alpha_n p_n$ cu $\alpha_1, \dots, \alpha_n \geq 0$, $\alpha_1 + \alpha_2 + \dots + \alpha_n = 1$.

Fie H semiplanul închis separat de dreapta p_1p_2 , care conține mulțimea dată S . Deoarece H este mulțime convexă, obținem că are loc inclusiunea $CH(S) \subseteq H$. Atunci în semiplanul opus nu există puncte din S . Rezultă că $p_1p_2 \subseteq \partial CH(S)$.

Toate punctele din $\partial CH(S)$ se află pe segmente de forma p_1p_2 , cu $p_1, p_2 \in S$. Prin reducere la absurd, fie $p \in \partial CH(S)$ astfel încât p nu se află pe un astfel de segment. Atunci putem înlocui $CH(S)$ cu o submulțime mai mică, convexă, care conține S . Atunci $CH(S)$ nu mai este cea mai mică mulțime convexă care conține mulțimea S .

În concluzie, au loc afirmațiile următoare:

I. $\partial CH(S)$ este o reuniune de segmente de dreaptă, cu capetele din S .

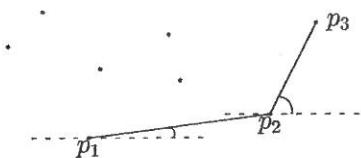
II. Aceste segmente de dreaptă sunt laturile unui poligon convex.

□

2.2 Determinarea învelitorii convexe a unei mulțimi finite din plan

Fie $p_1, p_2 \in S$ astfel încât $p_1p_2 \subset \partial CH(S)$. Presupunem că toate punctele din S se află la stânga vectorului $\overrightarrow{p_1p_2}$; altfel, inversăm numerotarea punctelor.

Rotim dreapta p_1p_2 în jurul punctului p_2 ; ea va întâlni un punct din S ; fie p_3 primul punct întâlnit. Atunci segmentul $p_2p_3 \subset \partial CH(S)$. În plus, nu există puncte $p \in S$, $p \neq p_1, p \neq p_3$ astfel încât $p_2p \subset \partial CH(S)$.



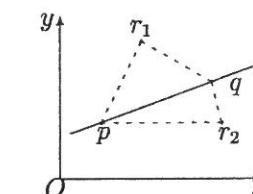
Dacă există trei puncte coliniare, îl luăm pe cel mai îndepărtat. Se repetă procedeul descris mai sus. S fiind o mulțime finită, este clar faptul că la un moment dat acest algoritm de determinare a punctelor situate pe frontieră poligonului se oprește.

Orice punct din $S \setminus \{vârfuri\}$ este în interiorul lui $CH(S)$; pe de altă parte aceste puncte au rămas tot timpul în stânga laturii.

Observația 2.2.1. Cum testăm dacă un punct este în stânga sau în dreapta unui vector, sau a unei laturi, sau a unei drepte?

Fie $p = (p_x, p_y)$, $q = (q_x, q_y)$ și $r = (r_x, r_y)$ puncte date din plan. Ecuția dreptei pq se poate scrie sub formă $\begin{vmatrix} x & y & 1 \\ p_x & p_y & 1 \\ q_x & q_y & 1 \end{vmatrix} = 0$ sau, echivalent, $\begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ x & y & 1 \end{vmatrix} = 0$. Să calculăm

determinantul $\begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} \stackrel{\text{not.}}{=} D(p, q, r)$.



Vom constata faptul că $D(p, q, r) < 0$ dacă Δpqr este parcurs în sens trigonometric și $D(p, q, r) > 0$ dacă Δpqr este parcurs în sens invers trigonometric. În primul caz se spune că se efectuează o întoarcere la stânga, iar în al doilea caz o întoarcere la dreapta.

Observația 2.2.2. Dacă p, q și r sunt coliniare, atunci convenim că r este la dreapta laturii pq dacă r este interior segmentului pq .

Prezentăm în continuare câțiva algoritmi cunoscuți pentru determinarea învelitorii convexe a unei mulțimi finite de puncte din plan.

I. Algoritmul slab de determinare a învelitorii convexe $CH(S)$.

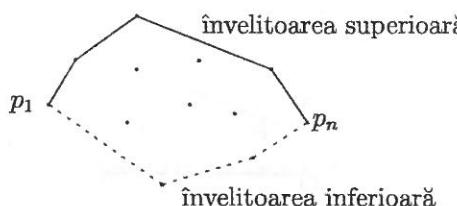
Se dă: o mulțime S de puncte în plan.

Se cere: o listă L care să conțină vârfurile învelitorii convexe $CH(S)$ ordonată în sens trigonometric.

- (1) $E \leftarrow \emptyset$
- (2) pentru $(p, q) \in S \times S$ cu $p \neq q$ execută
- (3) $valid \leftarrow true$
- (4) pentru $r \in S$, cu $r \neq p$ și $r \neq q$ execută
- (5) dacă r este la dreapta laturii pq atunci
- (6) $valid \leftarrow false$
- (7) dacă $valid$ atunci
- (8) adaugă pq la E
- (9) din lista E de laturi construiește lista L de vârfuri, în sens trigonometric

Observația 2.2.3. Se observă faptul că în algoritmul de mai sus au fost testate $n(n - 1)$ perechi de puncte, iar pentru fiecare astfel de pereche au fost testate $n - 2$ puncte. Astfel, algoritmul slab de determinare a învelitorii convexe $CH(S)$ are complexitatea $\mathcal{O}(n^3)$.

II. Algoritmul de determinare a învelitorii convexe $CH(S)$ prin determinarea învelitorii superioare și a învelitorii inferioare.



Se dă: o mulțime S de n puncte în plan.

Se cere: o listă L care să conțină vârfurile învelitorii convexe $CH(S)$ ordonată în sens invers trigonometric.

- (1) sortează punctele după abscisă; fie p_1, \dots, p_n sirul obținut
- (2) trece punctele p_1, p_2 în lista L_{sup} cu p_1 ca prim punct
- (3) pentru $i \leftarrow 3, n$ execută
- (4) adaugă p_i în L_{sup}
- (5) cât timp $|L_{sup}| > 2$ și ultimele trei puncte din L_{sup}
nu efectuează o întoarcere la dreapta execută
- (6) sterge punctul din mijlocul celor trei puncte din L_{sup}
- (7) trece punctele p_n, p_{n-1} în lista L_{inf} cu p_n ca prim punct
- (8) pentru $i \leftarrow n - 2, 1$ execută
- (9) adaugă p_i în L_{inf}
- (10) cât timp $|L_{inf}| > 2$ și ultimele trei puncte din L_{inf}
nu efectuează o întoarcere la dreapta execută
- (11) sterge punctul din mijlocul celor trei puncte din L_{inf}
- (12) scoate primul și ultimul punct din L_{inf} // pentru că apar și în L_{sup}
- (13) adaugă L_{inf} la L_{sup} pentru a obține lista L

Observația 2.2.4. Dacă două puncte au aceeași abscisă, atunci vom ordona punctele lexicografic, și anume, ordonăm punctele după abscisă, iar pentru abscise identice ordonăm punctele după ordonată.

Observația 2.2.5. Algoritmul de determinare a învelitorii convexe $CH(S)$ prin determinarea învelitorii superioare și inferioare are complexitatea $\mathcal{O}(n \lg n)$.

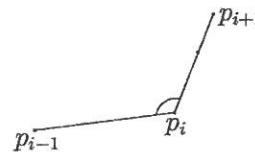
III. Algoritmul lui Jarvis pentru determinarea învelitorii convexe $CH(S)$.

Algoritmul lui Jarvis (1973) se bazează pe demonstrația Teoremei 2.1.1 privind învelitoarea convexă.

Algoritmul constă în următorii pași:

- Se dă $S = \{p_1, p_2, \dots, p_n\}$.
- Se deplasează o linie orizontală l de jos în sus până când întâlneste un punct din S . Dacă sunt mai multe puncte, se consideră cel mai din stânga. Fie acesta p_1 . Se adaugă p_1 la $CH(S)$.
- Se rotește apoi dreapta l în jurul lui p_1 până când întâlneste un alt punct $p_2 \in S$. Dacă sunt mai multe puncte coliniare, se va alege cel mai îndepărtat, iar celelalte puncte intermedii nu vor fi vârfuri ale învelitorii convexe.
- Se rotește în continuare dreapta l în jurul lui p_2 și aşa mai departe, până când se ajunge la primul punct ales p_1 .

Observația 2.2.6. Alegerea punctului p_{i+1} pentru $i \geq 2$ se va face pe baza condiției ca unghiul $\angle p_{i-1}p_ip_{i+1}$ să fie maxim.



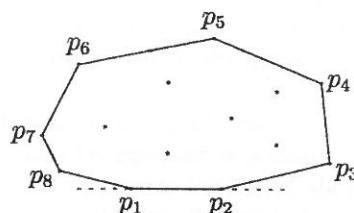
Prezentăm, în continuare, algoritmul lui Jarvis de determinare a învelitorii convexe $CH(S)$ a unei mulțimi S de puncte în plan:

Se dă: o mulțime $S = \{p_1, p_2, \dots, p_n\}$ de n puncte în plan.
Se cere: o listă \mathcal{L} care să conțină vârfurile învelitorii convexe $CH(S)$ ordonată în sens trigonometric.

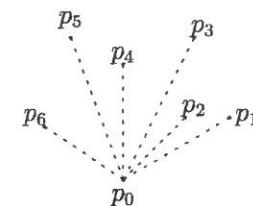
- (1) se determină $p_1 \in S$ cu y_1 minim
- (2) se determină $p_2 \in S$ astfel încât $\angle(p_1p_2, Ox)$ să fie minim
- (3) se adaugă p_1, p_2 la \mathcal{L}
- (4) $i \leftarrow 2$
- (5) cât timp $p_i \neq p_1$ execută
- (6) se determină $p_{i+1} \in S$ astfel încât $\angle p_{i-1}p_ip_{i+1}$ să fie maxim
- (7) *i* $\leftarrow i + 1$
- (8) se adaugă p_i la \mathcal{L}

Observația 2.2.7. Dacă numărul vârfurilor lui $CH(S)$ este k , algoritmul lui Jarvis are complexitatea $\mathcal{O}(kn)$.

Observația 2.2.8. Algoritmul se poate generaliza la cazul mulțimilor de puncte din spațiu, algoritm cunoscut sub denumirea de "algoritm de împachetare a cadoului".



Observația 2.2.9. Unghiul $\angle(p_0p_1, Ox)$ făcut de dreapta p_0p_1 cu direcția pozitivă a axei Ox poartă denumirea de **unghi polar al punctului p_1 relativ la p_0** . De exemplu, punctele p_1, p_2, p_3, p_4, p_5 și p_6 din figura următoare sunt sortate după unghiul polar relativ la vârful p_0 .



Vom utiliza sortarea punctelor cu ajutorul unghiurilor polare în raport cu un vârf dat pentru a descrie un alt algoritm.

IV. Algoritmul de scanare Graham pentru determinarea învelitorii convexe $CH(S)$.

Se dă: o mulțime $S = \{p_0, p_1, \dots, p_{n-1}\}$ de n puncte în plan.
Se cere: învelitoarea convexă $CH(S)$.

- (1) se determină $p_0 \in S$ cu y_0 minim // dacă sunt mai multe, se ia cel mai din stânga
- (2) se ordonează restul punctelor după unghiul polar relativ la p_0
- (3) fie p_1, \dots, p_{n-1} sirul obținut
- (4) Push(St, p_i), $i = \overline{0, 2}$
- (5) pentru $i = \overline{3, n-1}$ execută
- (6) cât timp $Top(St)$ este la stânga dreptei $NextToTop(St)p_i$ execută
- (7) Pop(St)
- (8) Push(St, p_i)
- (9) se tipărește stiva St

Observația 2.2.10. În algoritmul de mai sus s-au folosit următoarele funcții pentru lucrul cu stiva St :

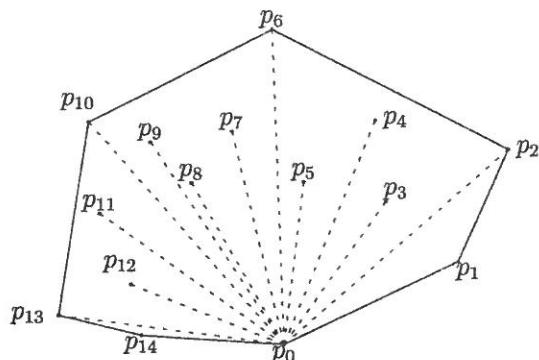
- $Push(St, p_i)$ - adaugă punctul p_i în stivă;
- $Top(St)$ - returnează elementul din vârful stivei;
- $NextToTop(St)$ - returnează elementul următor elementului din vârful stivei;
- $Pop(St)$ - returnează și elimină elementul din vârful stivei.

Observația 2.2.11. Dacă există mai multe puncte coliniare sau, cu alte cuvinte, au același unghi polar relativ la p_0 , se alege cel mai îndepărtat.

Observația 2.2.12. Condiția din linia (6) a algoritmului de mai sus se poate formula și astfel: $\text{NextToTop}(St)$, p_i și $\text{Top}(St)$ sunt în sens trigonometric.

Observația 2.2.13. Algoritmul de scanare Graham are complexitatea $\mathcal{O}(n \lg n)$.

Exemplul 2.2.1. Învelitoarea convexă a mulțimii $S = \{p_0, p_1, \dots, p_{14}\}$ de puncte în plan este redată în figura următoare.



Urmând pașii algoritmului, obținem următoarele:

initial $St = \{p_0, p_1, p_2\}$;
 pentru $i = 3$, $St = \{p_0, p_1, p_2, p_3\}$;
 pentru $i = 4$, $St = \{p_0, p_1, p_2, p_4\}$;
 pentru $i = 5$, $St = \{p_0, p_1, p_2, p_4, p_5\}$;
 pentru $i = 6$, $St = \{p_0, p_1, p_2, p_6\}$;
 pentru $i = 7$, $St = \{p_0, p_1, p_2, p_6, p_7\}$;
 pentru $i = 8$, $St = \{p_0, p_1, p_2, p_6, p_8\}$;
 pentru $i = 9$, $St = \{p_0, p_1, p_2, p_6, p_9\}$;
 pentru $i = 10$, $St = \{p_0, p_1, p_2, p_6, p_{10}\}$;
 pentru $i = 11$, $St = \{p_0, p_1, p_2, p_6, p_{10}, p_{11}\}$;
 pentru $i = 12$, $St = \{p_0, p_1, p_2, p_6, p_{10}, p_{12}\}$;
 pentru $i = 13$, $St = \{p_0, p_1, p_2, p_6, p_{10}, p_{13}\}$;
 pentru $i = 14$, $St = \{p_0, p_1, p_2, p_6, p_{10}, p_{13}, p_{14}\}$.

Capitolul 3

Triangularea unui poligon

3.1 Problema galeriei de artă

În 1973 Klee formulează această problemă sub forma următoare: câte camere video (paznici) sunt necesare pentru a supraveghea o galerie de artă și cum se poate determina poziția acestora?

Podeaua galeriei de artă este un poligon simplu cu n vârfuri. Un paznic este un punct fix care vede în orice direcție (are gradul de vizibilitate 2π).

Un poligon convex necesită evident un singur paznic, însă în cazul unui poligon neconvex soluția nu poate fi determinată la fel de ușor. În acest caz trebuie "descompus" poligonul în "părți ușor de supraveghetă", care nu vor fi altceva decât triunghiuri, de aici termenul de *triangulare*.

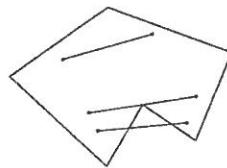
Pentru început precizăm că orice poligon considerat în acest capitol va fi un poligon simplu. De asemenea, menționăm că partitărea poligoanelor poate fi:

- partitărea convexă, care constă în împărțirea poligonului în subpoligoane convexe;
- partitărea în patrulatere, de obicei trapeze;
- triangularea, care se va realiza cu ajutorul diagonalelor.

În acest capitol ne vom ocupa de triangularea poligoanelor simple, în timp ce partitărea în trapeze respectiv partitărea convexă vor fi abordate în secțiunile 4.3 și respectiv 4.4 ale capitolului următor.

Definim acum noțiunea de vizibilitate a punctelor unui poligon simplu.

Definiția 3.1.1. Într-un poligon simplu P , spunem că punctul x poate vedea punctul y (sau y este vizibil din x) dacă segmentul închis xy este conținut în P , adică $xy \subseteq P$. Spunem că punctul x se poate vedea clar din y dacă $xy \subseteq P$ și $xy \cap \partial P \subseteq \{x, y\}$.



Definiția 3.1.2. Un paznic este un punct. O mulțime de paznici acoperă poligonul P dacă orice punct din P este vizibil de către cel puțin un paznic. Doi paznici nu-și blochează unul altuia vizibilitatea.

În continuare vom prezenta formularea matematică a problemei.

Fie $g(P)$ numărul minim de paznici necesari pentru a acoperi poligonul P :

$$g(P) = \min\{\text{card}(S) \mid S \text{ acoperă } P\}$$

Aici S este mulțimea pazniciilor, deci este o mulțime de puncte.

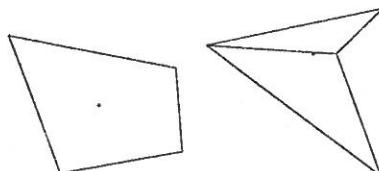
Fie P_n un poligon cu n vârfuri și fie funcția

$$G(n) = \max\{g(P_n) \mid P_n \text{ este poligon cu } n \text{ vârfuri}\}$$

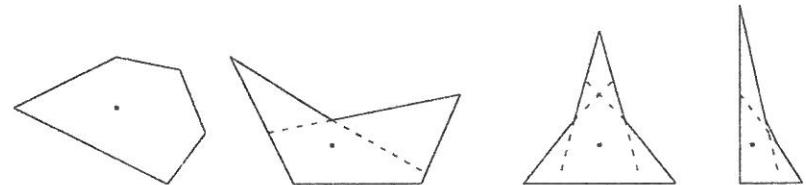
Problema galeriei de artă revine la determinarea funcției $G(n)$, care va reprezenta numărul de paznici suficienți pentru a supraveghea galeria P_n .

Pentru rezolvarea problemei, facem următoarele observații:

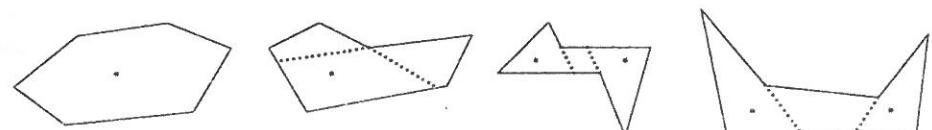
- $1 \leq G(n)$: cel puțin un paznic este necesar
- $G(n) \leq n$: n paznici acoperă orice poligon cu n vârfuri (câte unul în fiecare vârf)
- $G(3) = 1$: orice triunghi necesită un singur paznic
- $G(4) = 1$: cele două figuri de mai jos acoperă toate posibilitățile corespunzătoare patrulaterelor



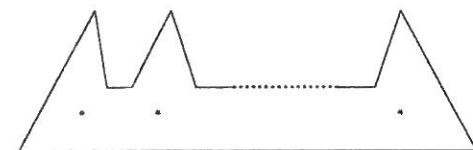
- $G(5) = 1$



- $G(6) = 2$



- $G(n) = \frac{n}{3}$ pentru $n = 3k$



Acesta este un exemplu generic (Chvatal, 1975) pentru a arăta că numărul necesar de paznici nu poate fi mai mic decât o anumită valoare.

Chvatal a arătat că $G(n) = \left\lceil \frac{n}{3} \right\rceil$. Vom prezenta însă o demonstrație mai simplă, aparținând lui Fisk (1978), care se bazează pe triangularea poligonului cu ajutorul diagonalelor.

Definiția 3.1.3. O diagonală a unui poligon P este un segment închis determinat de două vârfuri a și b care sunt reciproc vizibile în mod clar. Două diagonale nu se încrucisează (nu se taie) dacă intersecția lor este submulțime a capetelor lor; altfel spus, nu au puncte interioare comune.

Dacă punem în evidență numărul maxim de diagonale care nu se taie pentru un poligon P , am realizat o partiție a acestuia, care este chiar *triangularea* poligonului.

Observația 3.1.1. Este evident faptul că triangularea nu este unică. Vom demonstra, în secțiunea următoare, existența unei triangulări pentru orice poligon.

Dată fiind o triangulare a poligonului P , dacă așezăm câte un paznic în fiecare triunghi, este clar că mulțimea tuturor paznicilor va acoperi poligonul respectiv. Observăm însă că un paznic poziționat pe latura comună a două triunghiuri poate supraveghea ambele triunghiuri. În final, deducem că este suficient să așezăm câte un paznic în vârful comun a trei triunghiuri.

Determinarea vârfurilor în care trebuie așezați paznicii se face cu ajutorul 3-colorării poligonului sau a grafului asociat triangulării; menționăm că în acest graf nodurile sunt vârfurile poligonului, iar arcele sunt laturile poligonului, precum și acele diagonale ale poligonului care au fost utilizate pentru obținerea triangulării.

Definiția 3.1.4. Numim k -colorare a unui graf o asociere de culori nodurilor grafului astfel încât două noduri unite printr-un arc să nu aibă aceeași culoare.

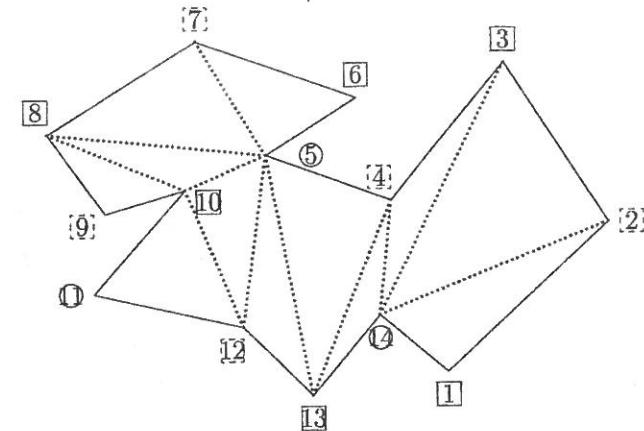
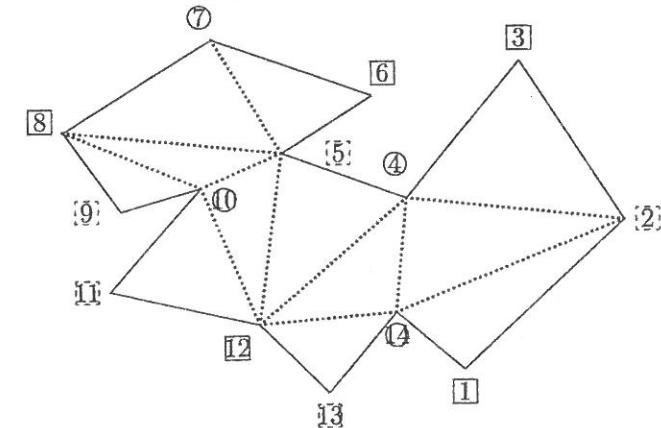
Observația 3.1.2. Vom demonstra, în secțiunile următoare, faptul că graful asociat unei triangulări a unui poligon admite o 3-colorare.

Observația 3.1.3. Plasând câte un paznic în fiecare dintre vârfurile asociate unei culori, se obține o acoperire a poligonului: fiecare triunghi asociat poligonului are câte o culoare în fiecare vârf, deci orice punct al său va fi vizibil din vârful de culoarea alesă.

Observația 3.1.4. Folosim următorul principiu matematic: dacă avem n obiecte care trebuie plasate în k cutii, cel puțin într-o cutie vom avea cel mult $\frac{n}{k}$ obiecte; altfel, numărul total de obiecte va fi mai mare decât $k \cdot \frac{n}{k} = n$, ceea ce este imposibil.

În cazul nostru, obiectele sunt cele n vârfuri, iar $k = 3$ reprezintă numărul culorilor. Atunci una dintre culori va fi folosită de cel mult $\frac{n}{3}$ ori, mai exact de $\lceil \frac{n}{3} \rceil$ ori.

Figurile următoare ilustrează două exemple de 3-colorări pentru un poligon cu 14 vârfuri, obținute cu ajutorul a două triangulări distințe.



Teorema 3.1.1 (Teorema galeriei de artă). Pentru un poligon simplu cu n vârfuri, $\lceil \frac{n}{3} \rceil$ camere video (paznici) sunt, în anumite ocazii, necesare și totdeauna suficiente pentru ca orice punct al poligonului să fie vizibil cel puțin de către o cameră video.

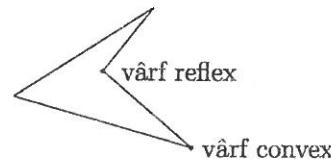
Observația 3.1.5. Fie P un poligon simplu cu n vârfuri. Algoritmul care determină vârfurile în care se așeză cele $\lceil \frac{n}{3} \rceil$ camere video ca în teorema de mai sus are complexitatea $\mathcal{O}(n \lg n)$.

Observația 3.1.6. Este clar faptul că în anumite situații numărul furnizat de teorema de mai sus va fi mai mare decât numărul camerelor video suficiente pentru a supraveghea

galeria de artă. Evident numărul minim nu poate fi precizat în cazul general, el depinzând de forma poligonului ce reprezintă podeaua galeriei. Pe de alta parte, în 1981 Avis și Toussaint au elaborat un algoritm eficient care precizează locația fiecărei camere video; în acest fel se obține, de fapt, numărul exact de camere video necesare pentru a supraveghea galeria de artă.

3.2 Existența unei triangulări

Definiția 3.2.1. Un vârf p_i al unui poligon se numește **convex** dacă $\angle p_{i-1}p_ip_{i+1} \leq \pi$, respectiv p_{i+1} fiind vâfurile adiacente lui p_i ; altfel, el se numește **reflex** (**concav**). Dacă $\angle p_{i-1}p_ip_{i+1} < \pi$ respectiv $\angle p_{i-1}p_ip_{i+1} > \pi$ atunci vârful p_i se numește **strict convex** respectiv **strict reflex**.

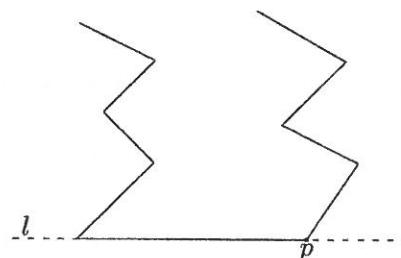


Lema 3.2.1. Orice poligon are cel puțin un vârf strict convex.

Demonstrație. Considerăm vâfurile ordonate în sens trigonometric. Observăm că un vârf p_i este strict convex dacă $p_{i-1}p_ip_{i+1}$ efectuează o întoarcere spre stânga; p_i va fi un vârf reflex dacă $p_{i-1}p_ip_{i+1}$ efectuează o întoarcere spre dreapta.

Fie l o latură care trece prin cel mai de jos vârf p (adică $p_y = \min$); dacă sunt mai multe (adică două), fie p vârful din dreapta.

Atunci în p se va efectua o întoarcere spre stânga, deci p este un vârf convex.



□

Lema 3.2.2. Orice poligon P cu $n \geq 4$ vârfuri are cel puțin o diagonală.

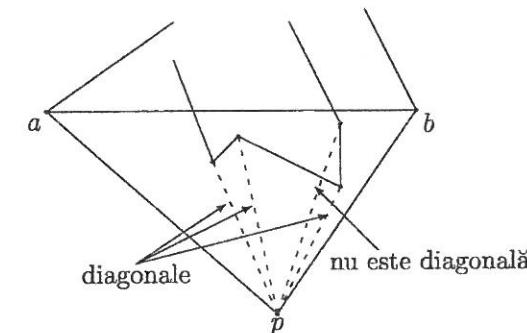
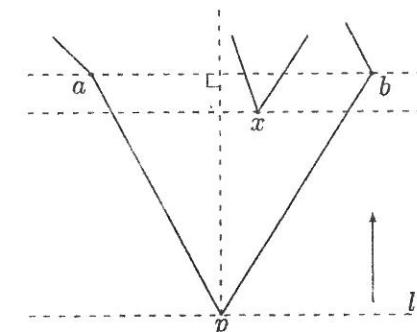
Demonstrație. Fie p un vârf strict convex și fie a, b vâfurile adiacente lui p . Deosebim două cazuri:

I. Dacă ab este diagonală, demonstrația este încheiată.

II. Presupunem că ab nu este diagonală.

În acest caz, ab este fie exterioară lui P (ceea ce este imposibil deoarece p este strict convex), fie intersectează ∂P în alte puncte decât a sau b . Atunci $\triangle apb$ conține cel puțin un al patrulea vârf al poligonului P ; fie x cel mai apropiat de p (adică primul vârf găsit de dreapta $l \parallel ab$ când l pornește din p spre ab).

Atunci px este diagonală a lui P .



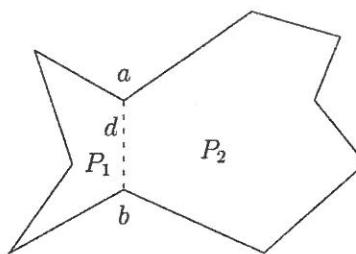
□

Teorema 3.2.1 (Teorema de existență a unei triangulări). *Orice poligon P cu n vârfuri poate fi partitionat în triunghiuri cu ajutorul diagonalelor (zero sau mai multe).*

Demonstrație. Demonstrația se face prin inducție după n .

Dacă $n = 3$, atunci P este un triunghi, caz în care partitioarea nu este necesară.

Fie $n \geq 4$; fie $d = ab$ o diagonală a lui P , conform Lemei 3.2.2. Observăm că d partiziunează P în două subpoligoane P_1 și P_2 , fiecare având d drept muchie și având mai puțin de n vârfuri. Conform ipotezei de inducție, P_1 și P_2 se pot triangula.



□

Lema 3.2.3. *Orice triangulare a unui poligon P cu n vârfuri utilizează $n - 3$ diagonale și are $n - 2$ triunghiuri.*

Demonstrație. Demonstrația se face prin inducție după n .

Cazul $n = 3$ este trivial.

Fie $n \geq 4$; partiziționăm P în două subpoligoane P_1 și P_2 cu ajutorul unei diagonale $d = ab$.

Fie n_1 numărul vârfurilor lui P_1 și n_2 numărul vârfurilor lui P_2 ; avem $n_1 + n_2 = n + 2$, deoarece a și b au fost numărate de două ori. Conform ipotezei de inducție avem:

P_1 are $n_1 - 3$ diagonale și $n_1 - 2$ triunghiuri,

P_2 are $n_2 - 3$ diagonale și $n_2 - 2$ triunghiuri.

În concluzie, numărul de diagonale necesare triangulării poligonului P va fi:

$$(n_1 - 3) + (n_2 - 3) + 1 = n_1 + n_2 - 5 = n + 2 - 5 = n - 3$$

(+1 semnificând diagonala d), în timp ce numărul de triunghiuri formate cu ajutorul acestor diagonale va fi:

$$(n_1 - 2) + (n_2 - 2) = n_1 + n_2 - 4 = n + 2 - 4 = n - 2.$$

□

Consecință 3.2.1. *Suma unghiurilor (interioare) ale unui poligon cu n vârfuri este $(n - 2)\pi$.*

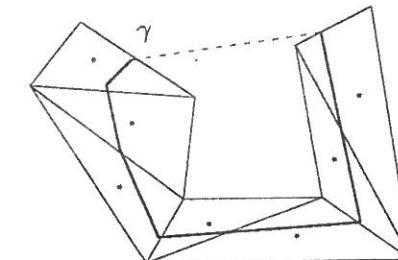
3.3 Proprietățile triangulării

Definiția 3.3.1. Fie P un poligon și fie o triangulare a sa. **Dualul T al triangulării** este un graf care are un nod asociat fiecărui triunghi și un arc între două noduri dacă triunghiurile au drept latură comună o diagonală.

Lema 3.3.1. *Dualul T al unei triangulări este un arbore (graf conex fără cicluri), cu fiecare nod de grad cel mult trei.*

Demonstrație. Orice nod are gradul cel mult 3 deoarece un triunghi poate avea alte trei triunghiuri vecine; altfel spus, se poate forma cu cel mult trei diagonale.

T este arbore pentru că nu are nici un ciclu; în caz contrar, presupunând că T ar avea un ciclu, am putea pune în evidență un drum γ care unește mijloacele diagonalelor comune triunghiurilor specifice, iar acest drum include vârfuri ale poligonului, ceea ce contrazice faptul că P este poligon simplu.



□

Definiția 3.3.2. Vom spune că trei vârfuri consecutive a, b, c ale unui poligon P formează o ureche a lui P dacă ac este diagonală; în acest caz, b se numește **capătul (vârful) urechii**. Două urechi ale lui P se spune că **nu se suprapun** dacă interioarele triunghiurilor definite cu ajutorul urechilor respective sunt disjuncte.

Teorema 3.3.1 (Teorema celor două urechi; Meister). *Orice poligon cu $n \geq 4$ vârfuri are cel puțin două urechi care nu se suprapun.*

Demonstrație. În dualul T al unei triangulări, un nod terminal corespunde unei urechi.

Din Lema 3.2.3 rezultă că T are $n - 2 \geq 2$ noduri, cel puțin două dintre acestea fiind noduri terminale.

Din cele două afirmații obținem că poligonul P are cel puțin două urechi.

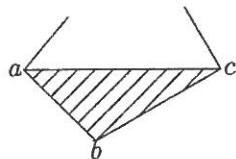
□

Teorema 3.3.2 (3-colorarea). *Graful corespunzător unei triangulări a unui poligon P admite o 3-colorare.*

Demonstrație. Demonstrația se face prin inducție relativ la n .

Cazul $n = 3$ este trivial.

Presupunem $n \geq 4$; atunci P are o ureche abc cu capătul b . Formăm un nou poligon P' astfel: înlocuim secvența abc în ∂P cu secvența ac în $\partial P'$. Atunci P' are $n - 1$ vârfuri. Aplicând ipoteza de inducție, rezultă că P' admite o 3-colorare. Acum "lipim" înapoi abc , punând în b a treia culoare, adică acea culoare care nu e în a sau c .



□

Observația 3.3.1. Metoda folosită aici conduce la triangularea prin eliminarea urechilor sau otectomiei.

3.4 Aria unui poligon

În această secțiune vom prezenta o aplicație a triangulării unui poligon simplu, și anume calculul ariei acestuia.

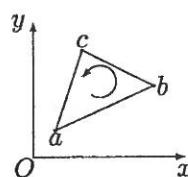
Observația 3.4.1. Fie $\triangle abc$. Atunci

$$\mathcal{A}(a, b, c) = \frac{1}{2} \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix},$$

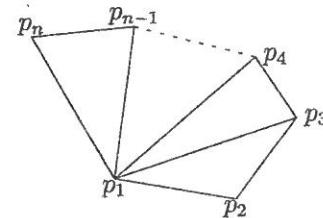
unde abc efectuează o întoarcere spre stânga; cu alte cuvinte, vârfurile a , b și c sunt parcuse în sens trigonometric.

Observăm că relația de mai sus poate fi scrisă și sub forma:

$$2\mathcal{A}(a, b, c) = a_x b_y - a_y b_x + b_x c_y - b_y c_x + c_x a_y - c_y a_x$$

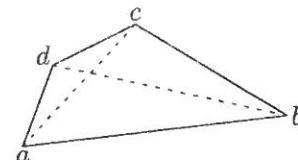


Observația 3.4.2. Aria unui poligon convex poate fi determinată prin triangularea unui poligon pornind dintr-un vârf astfel:



$$\mathcal{A}(P) = \mathcal{A}(p_1, p_2, p_3) + \mathcal{A}(p_1, p_3, p_4) + \dots + \mathcal{A}(p_1, p_{n-1}, p_n)$$

Observația 3.4.3. Aria unui patrulater convex se poate calcula în două moduri, în funcție de triangularea aleasă:



$$\mathcal{A}(P) = \mathcal{A}(a, b, c) + \mathcal{A}(a, c, d) = \mathcal{A}(d, a, b) + \mathcal{A}(d, b, c)$$

Relația de mai sus poate fi scrisă și sub forma:

$$2\mathcal{A}(P) = a_x b_y - a_y b_x + b_x c_y - b_y c_x + c_x a_y - c_y a_x + a_x c_y - a_y c_x + c_x d_y - c_y d_x + d_x a_y - d_y a_x,$$

deci

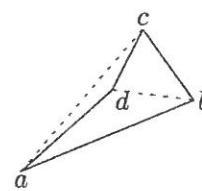
$$2\mathcal{A}(P) = a_x b_y - a_y b_x + b_x c_y - b_y c_x + c_x d_y - c_y d_x + d_x a_y - d_y a_x.$$

Observația 3.4.4. La fel ca în cazul unui patrulater convex, pentru un poligon convex $P = p_1 p_2 \dots p_n$ vom putea exprima aria astfel:

$$2\mathcal{A}(P) = \sum_{i=1}^n (p_{i,x} p_{i+1,y} - p_{i,y} p_{i+1,x}), \text{ unde } p_{n+1} = p_1.$$

Observația 3.4.5. Pentru a calcula aria unui patrulater concav, remarcăm faptul că în acest caz există o singură triangulare. Obținem următoarea relație:

$$\mathcal{A}(P) = \mathcal{A}(a, b, d) + \mathcal{A}(b, c, d)$$

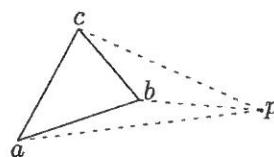


Pe de altă parte, observăm că formula din Observația 3.4.3 rămâne adevărată, adică putem scrie:

$$\mathcal{A}(P) = \mathcal{A}(a, b, c) + \mathcal{A}(a, c, d)$$

deoarece $\mathcal{A}(a, c, d)$ este negativă, vîrfurile a, c și d fiind parcuse în sens invers trigonometric.

Observația 3.4.6. Pe baza Observației 3.4.5, deducem că aria unui triunghi abc poate fi exprimată cu ajutorul unui punct arbitrar p :



$$\mathcal{A}(a, b, c) = \underbrace{\mathcal{A}(p, a, b)}_{-} + \underbrace{\mathcal{A}(p, b, c)}_{-} + \underbrace{\mathcal{A}(p, c, a)}_{+}$$

Spunem că am exprimat aria triunghiului abc dintr-un centru arbitrar p .

Teorema 3.4.1 (Aria unui poligon). Fie P un poligon simplu cu vîrfurile p_1, p_2, \dots, p_n și fie p un punct arbitrar. Atunci are loc formula:

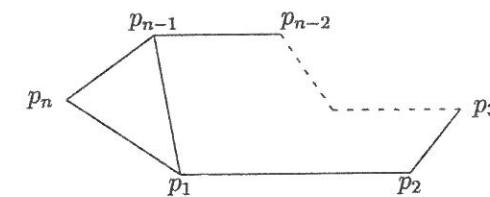
$$\mathcal{A}(P) = \mathcal{A}(p, p_1, p_2) + \mathcal{A}(p, p_2, p_3) + \dots + \mathcal{A}(p, p_{n-1}, p_n) + \mathcal{A}(p, p_n, p_1)$$

Dacă $p_i = (p_{i,x}, p_{i,y})$, $i = \overline{1, n}$, atunci aria se poate scrie sub forma următoare:

$$2\mathcal{A}(P) = \sum_{i=1}^n (p_{i,x}p_{i+1,y} - p_{i,y}p_{i+1,x}), \text{ unde am notat } p_{n+1} = p_1.$$

Demonstrație. Demonstrația se face prin inducție relativ la numărul n al vîrfurilor poligonului P . Cazul $n = 3$ este evident din Observația 3.4.6 și Observația 3.4.1.

Presupunem $n \geq 4$ și presupunem afirmația adevărată pentru $n - 1$. Poligonul P are cel puțin o ureche; fie aceasta $p_{n-1}p_np_1$. Fie P' poligonul având vîrfurile p_1, p_2, \dots, p_{n-1} .



Conform ipotezei de inducție avem:

$$\mathcal{A}(P') = \mathcal{A}(p, p_1, p_2) + \mathcal{A}(p, p_2, p_3) + \dots + \mathcal{A}(p, p_{n-2}, p_{n-1}) + \mathcal{A}(p, p_{n-1}, p_1)$$

$$\mathcal{A}(p_1, p_{n-1}, p_n) = \mathcal{A}(p, p_1, p_{n-1}) + \mathcal{A}(p, p_{n-1}, p_n) + \mathcal{A}(p, p_n, p_1)$$

Deoarece $\mathcal{A}(p, p_{n-1}, p_1) = -\mathcal{A}(p, p_1, p_{n-1})$, obținem:

$$\begin{aligned} \mathcal{A}(P) &= \mathcal{A}(P') + \mathcal{A}(p_1, p_{n-1}, p_n) = \\ &= \mathcal{A}(p, p_1, p_2) + \mathcal{A}(p, p_2, p_3) + \dots + \mathcal{A}(p, p_{n-2}, p_{n-1}) + \mathcal{A}(p, p_{n-1}, p_n) + \mathcal{A}(p, p_n, p_1) \end{aligned}$$

Dacă $p_i = (p_{i,x}, p_{i,y})$, $i = \overline{1, n}$, pe baza ipotezei de inducție și a Observației 3.4.1 putem scrie:

$$2\mathcal{A}(P') = \sum_{i=1}^n (p_{i,x}p_{i+1,y} - p_{i,y}p_{i+1,x}), \quad p_n = p_1$$

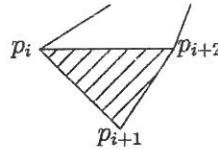
$$2\mathcal{A}(p_{n-1}, p_n, p_1) = p_{n-1,x}p_{n,y} - p_{n-1,y}p_{n,x} + p_{n,x}p_{1,y} - p_{n,y}p_{1,x} + p_{1,x}p_{n-1,y} - p_{1,y}p_{n-1,x}$$

$$\begin{aligned} 2\mathcal{A}(P) &= \underbrace{p_{1,x}p_{2,y} - p_{1,y}p_{2,x}}_{+} + \dots + \underbrace{p_{n-1,x}p_{1,y} - p_{n-1,y}p_{1,x}}_{+} + \\ &+ \underbrace{p_{n-1,x}p_{n,y} - p_{n-1,y}p_{n,x}}_{+} + \underbrace{p_{n,x}p_{1,y} - p_{n,y}p_{1,x}}_{+} + p_{1,x}p_{n-1,y} - p_{1,y}p_{n-1,x} \end{aligned}$$

□

3.5 Triangularea poligoanelor simple prin otectomie

În secțiunea 3.3 am demonstrat teorema lui Meister care afirma că orice poligon simplu cu $n \geq 4$ vârfuri are cel puțin două urechi care nu se suprapun. Pe de altă parte, pentru a pune în evidență o ureche, folosim o formă particulară de diagonale ale poligonului.



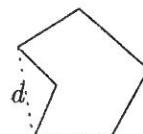
Observația 3.5.1. Este esențial să determinăm o diagonală de forma p_ip_{i+2} a poligonului $P = p_1p_2\dots p_n$. Pentru aceasta, vom rezolva următoarea problemă: **determinarea unei diagonale $d = p_ip_j$** .

Astfel, pornim de la observația că orice diagonală d trebuie să verifice următoarele condiții:

1. d nu intersectează nici o latură neincidentă a poligonului P .



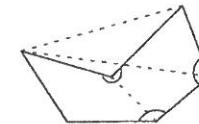
2. $d - \{p_i, p_j\}$ este în interiorul poligonului P .



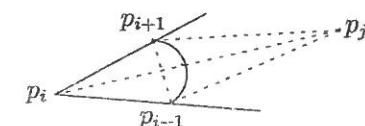
Vom analiza cele două condiții impuse asupra segmentului $d = p_ip_j$.

1. Pentru fiecare latură l a poligonului P care nu este incidentă lui d , se testează dacă aceasta intersectează d . Dacă răspunsul este afirmativ, atunci d nu este diagonală, altfel, se verifică cea de a doua condiție, neputând stabili, în această etapă, dacă d este sau nu diagonală.

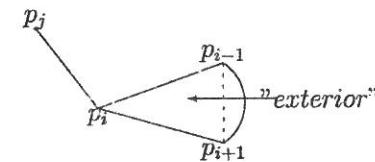
2. Fie $d = p_ip_j$ astfel încât $d \cap l = \emptyset$ și fie vârfurile adiacente lui p_i , adică p_{i-1} și p_{i+1} . Dacă $d - \{p_i, p_j\} \subset \text{int } P$ este interioară poligonului P în vecinătatea vârfurilor p_i și p_j , atunci d este diagonală. Este suficient să testăm un singur vârf deoarece am presupus satisfăcută condiția 1.



- (a) Dacă p_i este vârf convex: $p_ip_jp_{i+1}$ și $p_ip_{i-1}p_j$ efectuează o întoarcere spre stânga.



- (b) Dacă p_i este vârf reflex, este suficient să testăm dacă p_ip_j nu este în exterior.



Observația 3.5.2. Pentru determinarea unei urechi folosim diagonale de forma

$$d = p_ip_{i+2}, \quad i = \overline{1, n}, \quad \text{unde } p_{n+1} = p_1, \quad p_{n+2} = p_2,$$

urechea fiind $p_ip_{i+1}p_{i+2}$.

Avantajele utilizării metodei de eliminare a urechilor în vederea obținerii triangulării unui poligon simplu sunt următoarele:

- se testează relativ puține perechi de vârfuri;
- unul din poligoanele obținute fiind triunghi, rămâne un singur poligon (cu un număr mai mic de vârfuri) de triangulat.

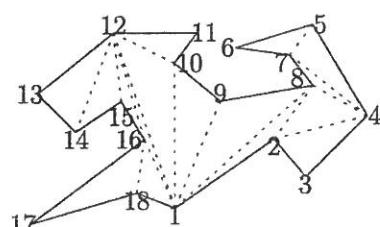
Prezentăm, în continuare, algoritmul de triangulare prin otectomie:

Se dă: un poligon $P = p_1p_2 \cdots p_{n-1}p_n$.
Se cere: o triangulare a poligonului P .

- (1) cât timp $n > 3$ execută
- (2) pentru $i = \overline{1, n}$ execută
- (3) dacă $p_i p_{i+2}$ este diagonală //unde $p_{n+1} = p_1$ și $p_{n+2} = p_2$ atunci
- (4) tipărește diagonala $p_i p_{i+2}$
- (5) șterge vârful p_{i+1}
- (6) $n = n - 1$
- (7) break //instrucțiune care determină ieșirea forțată din
- (8) //instrucțiunea "pentru" în acest caz, reluându-se
- (9) //algoritmul pentru poligonul $P = p_1 \cdots p_i p_{i+2} \cdots p_n$
- (10) //cu $n - 1$ vârfuri

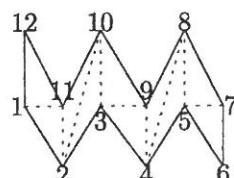
Observația 3.5.3. Complexitatea algoritmului de triangulare prin otectomie este $\mathcal{O}(n^3)$.

Exemplul 3.5.1. Pentru poligonul din figura de mai jos,



prin acest algoritm sunt eliminate urechile din vârfurile: 3, 6, 5, 7, 4, 2, 8, 9, 11, 10, 13, 14, 15, 12, și 17, rămânând triunghiul format de vârfurile 1, 16 și 18.

Observația 3.5.4. Dacă avem vârfuri coliniare, ca de exemplu:



atunci eliminăm urechile din vârfurile: 6, 7, 5, 8, 4, 9, 3, 10, 2, rămânând triunghiul format de vârfurile 1, 11 și 12.

Capitolul 4

Partiționarea unui poligon în poligoane monotone

4.1 Poligoane monotone

Pentru a obține un algoritm de triangulare mai rapid decât cel prin otectomie, un poligon simplu va fi partiționat în poligoane monotone.

Definiția 4.1.1. Fie un poligon simplu $P = p_1p_2 \dots p_n$. Numim lanț poligonal o mulțime $C \subseteq \partial P$ de forma $C = p_i p_{i+1} \cup p_{i+1} p_{i+2} \cup \dots \cup p_{i+k} p_{i+k+1}$.

Definiția 4.1.2. Un lanț poligonal C este strict monoton în raport cu o dreaptă l dacă orice dreaptă $l' \perp l$ intersectează C în cel mult un punct, adică $l' \cap C = \{\cdot\}$ sau $l' \cap C = \emptyset$.

Definiția 4.1.3. Un lanț poligonal C este monoton în raport cu o dreaptă l dacă pentru orice dreaptă $l' \perp l$, $l' \cap C$ este fie \emptyset , fie $\{\cdot\}$, fie un segment.

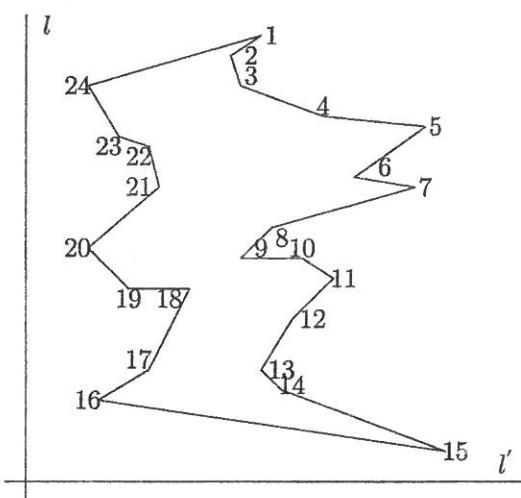
Definiția 4.1.4. Un poligon P se numește monoton în raport cu o dreaptă l dacă ∂P se poate descompune în două lanțuri poligonale A și B monotonе în raport cu l .

Exemplul 4.1.1. Poligonul $P = p_1 \dots p_{24}$ din figura următoare este monoton în raport cu o axă verticală l . Cele două lanțuri poligonale monotonе în raport cu l sunt:

$$A = p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12} p_{13} p_{14} p_{15},$$

$$B = p_1 p_{24} p_{23} p_{22} p_{21} p_{20} p_{19} p_{18} p_{17} p_{16} p_{15}.$$

A și B nu sunt strict monotone pentru că $p_9 p_{10}$ și $p_{19} p_{18}$ sunt perpendiculare pe l .

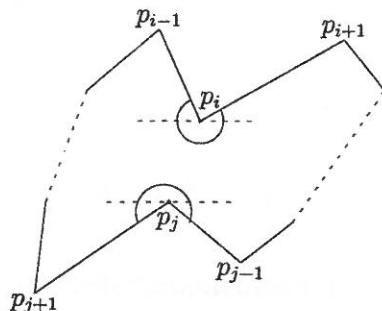


Observația 4.1.1. Este evident faptul că nu toate poligoanele sunt monotone. Pe de altă parte, există poligoane monotone în raport cu mai multe drepte.

Observația 4.1.2. În continuare considerăm că l este axa Oy . Studiul poligoanelor monotone în raport cu Oy este relativ simplu, deoarece vârfurile lanțurilor monotone sunt sortate după ordonata lor.

Vom vedea că un poligon este monoton dacă el este monoton în vecinătatea oricărui vârf.

Definiția 4.1.5. Numim punct de întoarcere al poligonului P un vârf reflex p_i astfel încât p_{i-1} și p_{i+1} sunt fie ambele deasupra lui p_i , fie ambele sub p_i . Spunem că un punct q este deasupra punctului p dacă $q_y > p_y$ sau $q_y = p_y$ și $q_x < p_x$.



Teorema 4.1.1. Un poligon P este monoton dacă și numai dacă nu are puncte de întoarcere.

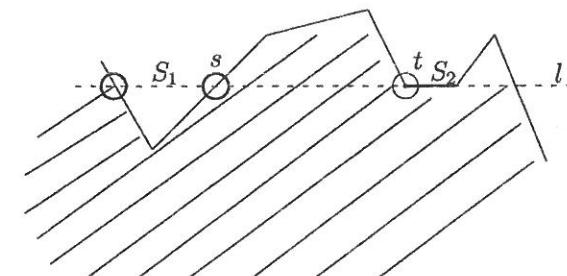
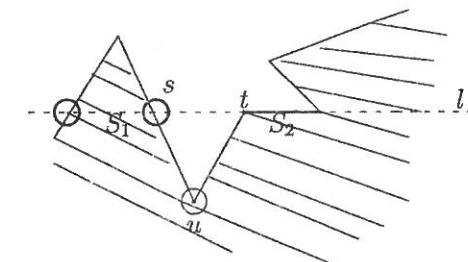
Demonstrație. Demonstrăm că un poligon care nu este monoton are cel puțin un punct de întoarcere.

Presupunem că P nu este monoton. Partiționăm ∂P în lanțurile A și B . Rezultă că cel puțin unul dintre cele două lanțuri nu este monoton. Presupunem că A nu este monoton, adică o dreaptă orizontală l taie A în cel puțin două părți conexe. Fie $l \cap A = \{S_1, S_2\}$. Fie s cel mai din dreapta punct din S_1 și t cel mai din stânga punct din S_2 .

Distingem două cazuri:

1. segmentul st este exterior poligonului P ;
2. segmentul st se află în interiorul poligonului P , adică $st \subset P$.

În ambele cazuri avem cel puțin un punct de întoarcere (în prima figură u , iar în cea de-a doua t).



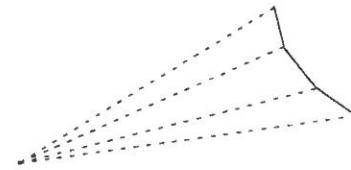
□

4.2 Triangularea unui poligon monoton

Pentru a triangula un poligon monoton P , este necesară parcurgerea următoarelor etape:

- sortarea vârfurilor poligonului P de sus în jos ($p_1 \mapsto y_1 = \max, \dots, p_n \mapsto y_n = \min$);
- considerarea unui vârf p și unirea acestuia prin diagonale cu toate vâfurile vizibile, aflate deasupra lui;
- înălțurarea poligonului astfel obținut, care va fi deja triangulat.

Observația 4.2.1. Vâfurile aflate deasupra lui p vor fi toate într-un lanț, iar lanțul va fi reflex.



Prezentăm, în continuare, algoritmul de triangulare a unui poligon monoton:

Se dă: un poligon monoton P

Se cere: o triangulare a poligonului P

- (1) sortează vâfurile în raport cu ordonata; fie $P = p_1 p_2 \cdots p_n$ poligonul obținut
- (2) adaugă p_1, p_2 în stivă
- (3) pentru $j = 3, n - 1$ execută
 - (4) dacă p_j și elementul din vârful stivei sunt pe lanțuri diferite
 - (5) atunci
 - (6) șterge toate vâfurile din stivă
 - (7) adaugă o diagonală de la p_j la fiecare vârf șters, mai puțin ultimul
 - (8) adaugă p_{j-1} și p_j în stivă
 - (9) altfel
 - (10) șterge elementul din vârful stivei
 - (11) șterge toate vâfurile din stivă care se pot uni cu p_j printr-o diagonală
 - (12) adaugă diagonalele respective
 - (13) adaugă ultimul vârf șters în stivă
 - (14) adaugă p_j în stivă
 - (15) adaugă diagonalele de la p_n la toate vâfurile din stivă în afară de primul și ultimul

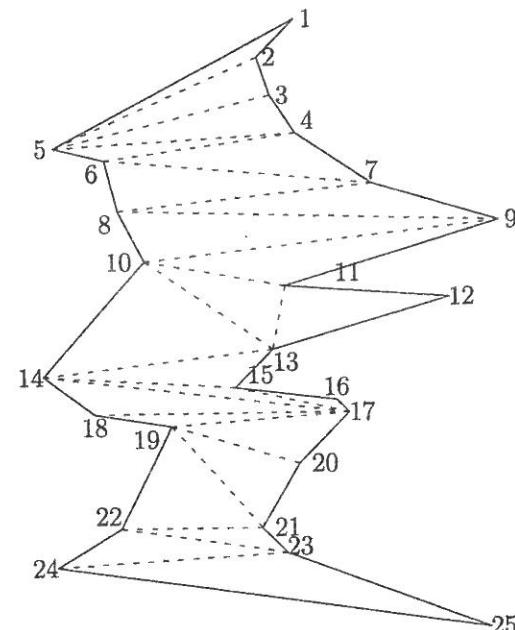
Observația 4.2.2. Complexitatea algoritmului de triangulare a unui poligon monoton este $\mathcal{O}(n)$.

Exemplul 4.2.1. Fie poligonul simplu $P = p_1 p_2 \cdots p_{25}$ din figura următoare. Vâfurile sunt sortate după ordonată. P este monoton deoarece frontieră sa se descompune în două lanțuri monotone:

$$A = p_1 p_5 p_6 p_8 p_{10} p_{14} p_{18} p_{19} p_{22} p_{24} p_{25},$$

$$B = p_1 p_2 p_3 p_4 p_7 p_9 p_{11} p_{12} p_{13} p_{15} p_{16} p_{17} p_{20} p_{21} p_{23} p_{25}.$$

Redăm mai jos etapele obținerii triangulării poligonului dat, urmând pașii algoritmului.



$$St = \{p_1, p_2\}$$

$$p_3 \in B;$$

$$Top(St) = p_2 \in B$$

Șterge p_2

$$St = \{p_1, p_2, p_3\}$$

$$p_4 \in B;$$

$$Top(St) = p_3 \in B$$

Șterge p_3

$$St = \{p_1, p_2, p_3, p_4\}$$

$p_5 \in A;$

$\text{Top}(St) = p_4 \in B$
 Sterge p_4, p_3, p_2, p_1
 Adaugă d_{54}, d_{53}, d_{52}
 $St = \{p_4, p_5\}$

 $p_6 \in A;$

$\text{Top}(St) = p_5 \in A$
 Sterge p_5
 Sterge p_4
 Adaugă d_{64}
 $St = \{p_4, p_6\}$

 $p_7 \in B;$

$\text{Top}(St) = p_6 \in A$
 Sterge p_6, p_4
 Adaugă d_{76}
 $St = \{p_6, p_7\}$

 $p_8 \in A;$

$\text{Top}(St) = p_7 \in B$
 Sterge p_7, p_6
 Adaugă d_{87}
 $St = \{p_7, p_8\}$

 $p_9 \in B;$

$\text{Top}(St) = p_8 \in A$
 Sterge p_8, p_7
 Adaugă d_{98}
 $St = \{p_8, p_9\}$

 $p_{10} \in A;$

$\text{Top}(St) = p_9 \in B$
 Sterge p_9, p_8
 Adaugă $d_{10,9}$
 $St = \{p_9, p_{10}\}$

 $p_{11} \in B;$

$\text{Top}(St) = p_{10} \in A$
 Sterge p_{10}, p_9
 Adaugă $d_{11,10}$
 $St = \{p_{10}, p_{11}\}$

 $p_{12} \in B;$

$\text{Top}(St) = p_{11} \in B$
 Sterge p_{11}
 $St = \{p_{10}, p_{11}, p_{12}\}$

 $p_{13} \in B;$

$\text{Top}(St) = p_{12} \in B$
 Sterge p_{12}
 Sterge p_{11}, p_{10}
 Adaugă $d_{13,11}, d_{13,10}$
 $St = \{p_{10}, p_{13}\}$

 $p_{14} \in A;$

$\text{Top}(St) = p_{13} \in B$
 Sterge p_{13}, p_{10}
 Adaugă $d_{14,13}$
 $St = \{p_{13}, p_{14}\}$

 $p_{15} \in B;$

$\text{Top}(St) = p_{14} \in A$
 Sterge p_{14}, p_{13}
 Adaugă $d_{15,14}$
 $St = \{p_{14}, p_{15}\}$

 $p_{16} \in B;$

$\text{Top}(St) = p_{15} \in B$
 Sterge p_{15}
 $St = \{p_{14}, p_{15}, p_{16}\}$

 $p_{17} \in B;$

$\text{Top}(St) = p_{16} \in B$
 Sterge p_{16}
 Sterge p_{15}, p_{14}
 Adaugă $d_{17,15}, d_{17,14}$
 $St = \{p_{14}, p_{17}\}$

 $p_{18} \in A;$

$\text{Top}(St) = p_{17} \in B$
 Sterge p_{17}, p_{14}
 Adaugă $d_{18,17}$
 $St = \{p_{17}, p_{18}\}$

 $p_{19} \in A;$

$\text{Top}(St) = p_{18} \in A$

Şterge p_{18}
 Șterge p_{17}
 Adaugă $d_{19,17}$
 $St = \{p_{17}, p_{19}\}$

$p_{20} \in B;$

$Top(St) = p_{19} \in A$
 Șterge p_{19}, p_{17}
 Adaugă $d_{20,19}$
 $St = \{p_{19}, p_{20}\}$

$p_{21} \in B;$

$Top(St) = p_{20} \in B$
 Șterge p_{20}
 Șterge p_{19}
 Adaugă $d_{21,19}$
 $St = \{p_{19}, p_{21}\}$

$p_{22} \in A;$

$Top(St) = p_{21} \in B$
 Șterge p_{21}, p_{19}
 Adaugă $d_{22,21}$
 $St = \{p_{21}, p_{22}\}$

$p_{23} \in B;$

$Top(St) = p_{22} \in A$
 Șterge p_{22}, p_{21}
 Adaugă $d_{23,22}$
 $St = \{p_{22}, p_{23}\}$

$p_{24} \in A;$

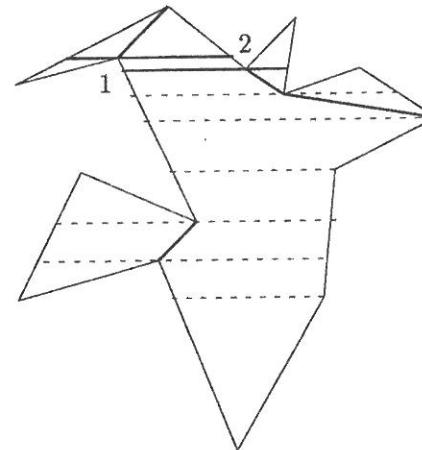
$Top(St) = p_{23} \in B$
 Șterge p_{23}, p_{22}
 Adaugă $d_{24,23}$
 $St = \{p_{23}, p_{24}\}$

p_{25} nu trebuie unit cu nici un vârf din St .

4.3 Partiționarea poligoanelor simple în poligoane monotone

Primul pas în partiționarea poligoanelor simple în poligoane monotone constă în partiționarea acestora în trapeze cu bazele paralele cu axa Ox . O primă observație este aceea că acest tip de partiționare nu se realizează doar cu ajutorul diagonalelor.

Observația 4.3.1. O partiționare în trapeze se obține ducând linii orizontale prin toate vârfurile poligonului (mai exact segmente de lungime maximă care trec prin vârfuri și sunt conținute în $intP$). Triunghiul este privit ca un trapez degenerat, cu o bază de lungime nulă.



Notăm, pe scurt, partiționarea în trapeze prin PT .

Definiția 4.3.1. Vârfurile prin care trec paralelele necesare PT se numesc **vârfuri suport**.

Considerăm un poligon care nu are două vârfuri pe aceeași paralelă. Atunci, într-o PT , orice trapez are exact două vârfuri suport.

Observația 4.3.2. Legătura între PT și poligoanele monotone este următoarea: dacă un vârf suport este punct interior al bazei unui trapez, atunci acel vârf este de întoarcere.

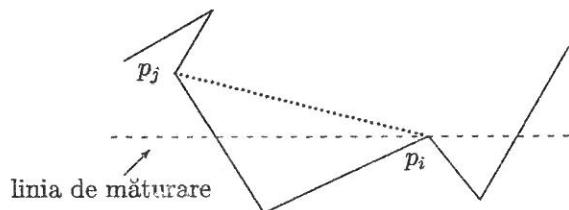
Am demonstrat că un poligon este monoton dacă și numai dacă nu are vârfuri de întoarcere. Rezultă că poligonul din figură nu este monoton pentru că are vârfuri de întoarcere.

Concluzie: Pentru a parta un poligon simplu în poligoane monotone, trebuie să „înlăturăm” vârfurile de întoarcere. Acest lucru se realizează prin adăugarea unor diagonale, în felul următor:

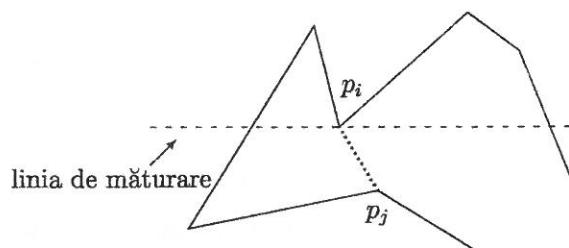
- I. Dacă laturile adiacente sunt sub vârf și interiorul poligonului este local deasupra vârfului, atunci alegem o diagonală cu un capăt în vârf, iar celălalt capăt deasupra lui. Exemplu: vârful 1. Un astfel de vârf se mai numește **vârf de separare**.
- II. Dacă laturile adiacente sunt deasupra vârfului de întoarcere și interiorul poligonului este local sub vârf, atunci alegem o diagonală cu un capăt în vârf, iar celălalt capăt sub vârf. Exemplu: vârful 2. Un astfel de vârf se mai numește **vârf de unire**.

Algoritmul de *PT* folosește ideea de la algoritmul de măturare plană: o linie orizontală parcurge planul, starea acesteia fiind actualizată în fiecare vârf al poligonului. În prealabil, vârfurile trebuie sortate după ordonată.

- I. Presupunem că ajungem într-un vârf de separare p_i . Aceasta trebuie unit cu cel mai de jos vârf situat deasupra liniei de măturare, de exemplu p_j astfel încât p_ip_j este o diagonală.



- II. Presupunem că suntem într-un vârf de unire p_i . Aceasta trebuie unit cu cel mai de sus vârf de sub linia de măturare, fie acesta p_j , astfel încât p_ip_j este o diagonală.

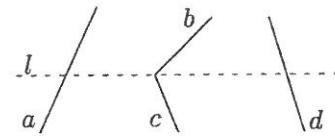


Observația 4.3.3. Trasând aceste diagonale, vârful p_i nu va mai fi vârf de întoarcere în poligoanele obținute.

Observația 4.3.4. Starea liniei de măturare l conține laturile pe care le intersectează la un moment dat.

Ce schimbări pot interveni într-un vârf suport?

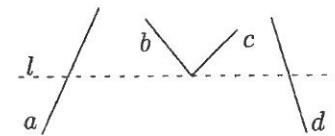
(i)



În acest caz se șterge b și se inserează c .

$$(\dots, a, b, d, \dots) \mapsto (\dots, a, c, d, \dots)$$

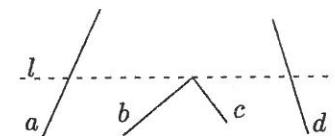
(ii)



În acest caz se șterg b și c .

$$(\dots, a, b, c, d, \dots) \mapsto (\dots, a, d, \dots)$$

(iii)



În acest caz se inserează b și c .

$$(\dots, a, d, \dots) \mapsto (\dots, a, b, c, d, \dots)$$

Prezentăm, în continuare, algoritmul de triangulare a unui poligon simplu prin partiționarea în poligoane monotone:

- (1) sortează vârfurile după ordonată
- (2) construiește PT
- (3) partiționează poligonul în poligoane monotone
- (4) triangulează fiecare poligon monoton

Observația 4.3.5. Complexitatea algoritmului de triangulare a unui poligon simplu prin partiționarea în poligoane monotone este $\mathcal{O}(n \log n)$.

Observația 4.3.6. Există și alți algoritmi de triangulare a poligoanelor simple, de complexitate $\mathcal{O}(n \log r)$, r fiind numărul vârfurilor reflexe, $\mathcal{O}(n \log \log n)$ sau chiar $\mathcal{O}(n)$. Algoritmul de complexitate $\mathcal{O}(n)$ îi aparține lui Chazelle și se bazează pe o noțiune mai sofisticată, numită hartă de vizibilitate, care este o generalizare a PT ; pe de altă parte, acest algoritm este dificil de implementat.

4.4 Partiționarea convexă

Triangularea unui poligon poate fi privită ca un caz particular al partiționării poligonului în subpoligoane convexe.

O astfel de partiționare poate fi optimală într-unul din următoarele sensuri:

- partiționarea într-un număr cât mai mic de poligoane convexe;
- partiționarea într-un timp cât mai scurt.

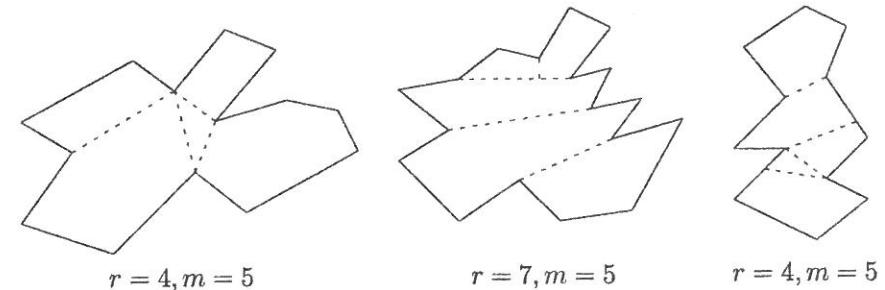
Menționăm faptul că în partiționarea convexă pot fi utilizate atât diagonale ale poligonului, cât și segmente interioare poligonului, acestea putând avea vârfurile pe frontieră sau chiar în interiorul poligonului.

Pentru a evalua eficiența partiționării convexe a poligoanelor, este necesar să avem o estimare a numărului subpoligoanelor convexe ce trebuie utilizate.

Teorema 4.4.1 (Chazelle). *Fie m cel mai mic număr de poligoane convexe în care poate fi partiționat un poligon simplu. Dacă poligonul are r vârfuri reflexe, au loc inegalitățile:*

$$\left\lceil \frac{r}{2} \right\rceil + 1 \leq m \leq r + 1.$$

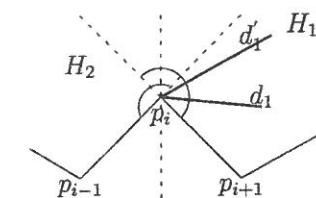
Exemplul 4.4.1. În figurile următoare sunt prezentate câteva exemple în acest sens.



Există un algoritm de partiționare a poligoanelor simple în poligoane convexe utilizând diagonalele (Hertel, Mehlhorn).

Definiția 4.4.1. Într-o partiționare convexă a unui poligon, o diagonală se numește esențială pentru un vârf dacă atunci când o ștergem se obține cel puțin un poligon neconvex. O diagonală care nu este esențială pentru nici un vârf se numește neesențială.

Observația 4.4.1. Vârful pentru care diagonală este esențială trebuie să fie reflex. Pe de altă parte, se demonstrează relativ ușor faptul că există cel mult două diagonale esențiale pentru orice vârf reflex. În figura următoare, în H_1 există cel mult una și la fel și în H_2 .



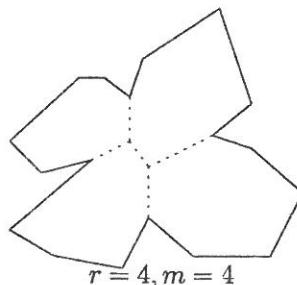
Prezentăm, în continuare, algoritmul de partiționare a poligoanelor simple în poligoane convexe utilizând diagonalele:

- (1) pornim cu o triangulare a poligonului
- (2) ștergem diagonalele neesențiale
- (3) obținem partiționarea poligonului P în poligoane convexe

Problema care se pune este următoarea: cât de departe de soluția optimă vom fi, utilizând acest algoritm?

Teorema 4.4.2. Fie m cel mai mic număr de poligoane convexe în care poate fi partiționat un poligon simplu și fie M numărul poligoanelor convexe în care este partiționat poligonul respectiv utilizând algoritmul de mai sus. Atunci $4m \geq M$.

În finalul acestui capitol, prezentăm un exemplu de partiționare convexă optimală (Chazelle, $\mathcal{O}(n^3)$).



Capitolul 5

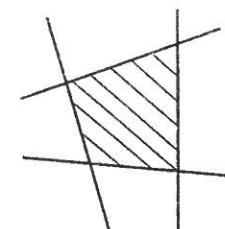
Probleme de intersecție

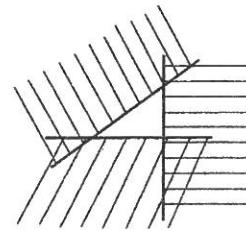
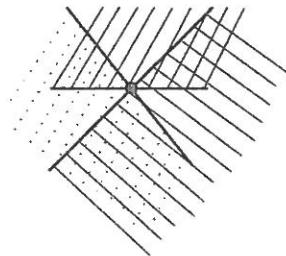
5.1 Intersecții de semiplane

Fie $(h_i) : a_i x + b_i y + c_i \leq 0$, a_i, b_i, c_i constante reale astfel încât $a_i^2 + b_i^2 > 0$, $i = \overline{1, n}$ o mulțime de n semiplane închise. Notăm $H = \{h_1, h_2, \dots, h_n\}$.

În acest capitol vom rezolva următoarea problemă: determinarea punctelor comune semiplanelor h_i , $i = \overline{1, n}$.

Observația 5.1.1. Orice semiplan este o mulțime convexă. Intersecția mulțimilor convexe este o mulțime convexă. Deci intersecția unei familii de semiplane este o regiune convexă în plan, mai exact o regiune poligonală convexă, mărginită de cel mult n laturi conținute de dreptele $(d_i) : a_i x + b_i y + c_i = 0$, $i = \overline{1, n}$. Această regiune poate fi nemărginită. De asemenea, poate degenera într-un punct sau în mulțimea vidă.





Determinarea intersecției unei mulțimi de semiplane se poate face cu ajutorul algoritmului următor:

Algoritmul de intersecție a semiplanelor H: INTERSECT(H)

Se dă: $H = \{h\}$ o mulțime de n semiplane

Se cere: Regiunea poligonlă convexă $C = \bigcap_{h \in H} h$

- (1) dacă $\text{card}(H) = 1$
- (2) atunci
- (3) $C \leftarrow h \in H$ (unicul semiplan)
- (4) altfel
- (5) împarte H în două submulțimi H_1, H_2 de dimensiune $\left[\frac{n}{2}\right]$ (respectiv $\left[\frac{n}{2}\right] + 1$)
- (6) $C_1 \leftarrow \text{INTERSECT}(H_1)$
- (7) $C_2 \leftarrow \text{INTERSECT}(H_2)$
- (8) $C \leftarrow \text{INTERSECT}(C_1, C_2)$

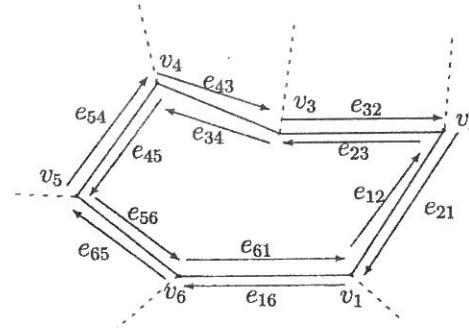
În ultima secțiune a acestui capitol vom descrie modulul $\text{INTERSECT}(C_1, C_2)$ de intersecție a două regiuni convexe din plan.

5.2 Lista dublu conectată a muchiilor

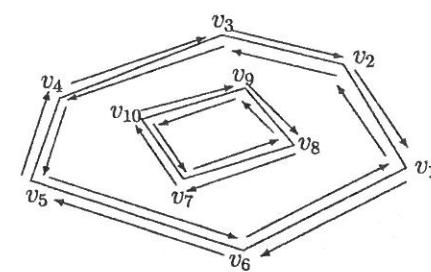
Lista dublu conectată a muchiilor, pe scurt LDCM, va fi o structură de date asociată unui graf planar, cu ajutorul căreia vom putea executa unele operații elementare, cum ar fi:

- parcurgerea frontierei unei fețe;
- vizitarea muchiilor incidente unui vârf;
- accesarea unei fețe de pe o față adiacentă (dacă se dă o muchie comună).

Deoarece fiecare muchie a unui graf planar mărginește exact două fețe, este convenabil să asociem oricărei muchii două semi-muchii, câte una pe frontieră fiecărei fețe, astfel încât fiecare dintre acestea să aibă o singură semi-muchie următoare respectiv precedentă. Cele două semi-muchii se numesc **gemene**.



Altfel spus, putem vorbi despre originea respectiv destinația unei semi-muchii. Orientarea se face astfel încât interiorul feței să fie în stânga celui care parcurge frontieră. Dacă în interiorul unei fețe se află o altă față, spunem că fața respectivă are o **gaură**.



În concluzie, orice LDCM constă în trei tipuri de date:

Vertex: această structură de date este formată din coordonatele vârfurilor și, pentru fiecare vârf, câte o semi-muchie incidentă. Acestea se definesc astfel:

- $\text{Coordinates}(v)$ = coordonatele punctului din plan asociat vârfului v al grafului;
- $\text{IncidentEdge}(v)$ = o semi-muchie care are originea în vârful v .

Face: este o strucură de date ce conține, pentru fiecare față, câte o semi-muchie exterioară și una interioară. Acestea sunt definite în felul următor:

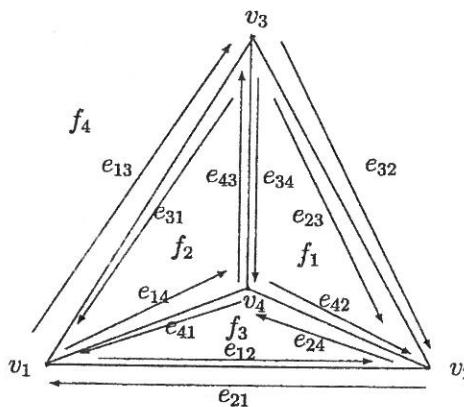
- $\text{OuterComponent}(f)$ = o semi-muchie de pe frontieră exterioară a feței f . Pentru față exterioară se trece nil.

- $\text{InnerComponent}(f)$ = o semi-muchie de pe frontieră interioară a feței f . Dacă f nu are găuri se trece nil.

Half-Edge: această structură de date conține informații despre un vârf incident semi-muchiei, față pe a cărei frontieră interioară se află aceasta, semi-muchia geamănă, cea precedentă respectiv următoare semi-muchiei date, definite după cum urmează:

- $\text{Origin}(\vec{e})$ = vârful în care își are originea semi-muchia \vec{e} ;
- $\text{Twin}(\vec{e})$ = semi-muchia geamănă asociată lui \vec{e} ;
- $\text{IncidentFace}(\vec{e})$ = față situată la stânga semi-muchiei \vec{e} ;
- $\text{Next}(\vec{e})$ = semi-muchia care urmează lui \vec{e} pe frontieră același feță;
- $\text{Prev}(\vec{e})$ = semi-muchia precedentă lui \vec{e} pe frontieră același feță.

Exemplul 5.2.1. Graful plan asociat unui tetraedru



Vertex	Coordinates	IncidentEdge
v_1	(x_1, y_1)	\vec{e}_{14}
v_2	(x_2, y_2)	\vec{e}_{23}
v_3	(x_3, y_3)	\vec{e}_{31}
v_4	(x_4, y_4)	\vec{e}_{42}

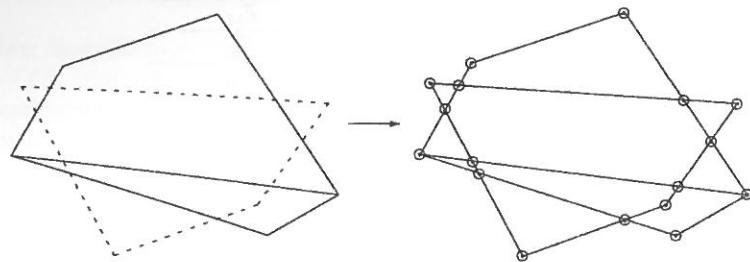
Face	OuterComponent	InnerComponent
f_1	\vec{e}_{34}	nil
f_2	\vec{e}_{31}	nil
f_3	\vec{e}_{41}	nil
f_4	nil	\vec{e}_{32}

Half-Edge	Origin	Twin	IncidentFace	Next	Prev
\vec{e}_{12}	v_1	\vec{e}_{21}	f_3	\vec{e}_{24}	\vec{e}_{41}
\vec{e}_{21}	v_2	\vec{e}_{12}	f_4	\vec{e}_{13}	\vec{e}_{32}
\vec{e}_{13}	v_1	\vec{e}_{31}	f_4	\vec{e}_{32}	\vec{e}_{21}
\vec{e}_{31}	v_3	\vec{e}_{13}	f_2	\vec{e}_{14}	\vec{e}_{43}
\vec{e}_{14}	v_1	\vec{e}_{41}	f_2	\vec{e}_{43}	\vec{e}_{31}
\vec{e}_{41}	v_4	\vec{e}_{14}	f_3	\vec{e}_{12}	\vec{e}_{24}
\vec{e}_{23}	v_2	\vec{e}_{32}	f_1	\vec{e}_{34}	\vec{e}_{42}
\vec{e}_{32}	v_3	\vec{e}_{23}	f_4	\vec{e}_{21}	\vec{e}_{13}
\vec{e}_{24}	v_2	\vec{e}_{42}	f_3	\vec{e}_{41}	\vec{e}_{12}
\vec{e}_{42}	v_4	\vec{e}_{24}	f_1	\vec{e}_{23}	\vec{e}_{34}
\vec{e}_{34}	v_3	\vec{e}_{43}	f_1	\vec{e}_{42}	\vec{e}_{23}
\vec{e}_{43}	v_4	\vec{e}_{34}	f_2	\vec{e}_{31}	\vec{e}_{14}

5.3 Suprapunerea a două subdiviziuni ale planului

Date fiind două subdiviziuni ale planului S_1, S_2 , suprapunerea lor este o nouă subdiviziune a planului, notată $O(S_1, S_2)$ și definită astfel:

- vârfurile sunt vârfuri ale lui S_1, S_2 sau puncte de intersecție dintre muchii (una din S_1 , alta din S_2);
- muchiile sunt muchii ale lui S_1, S_2 sau porțiuni de muchii (determinate de un vârf și o intersecție de muchii sau de două intersecții de muchii);
- fețele sunt submulțimi conexe maximale (în raport cu incluziunea) ale intersecțiilor de forma $f_1 \cap f_2$, unde $f_1 \in S_1$ și $f_2 \in S_2$.



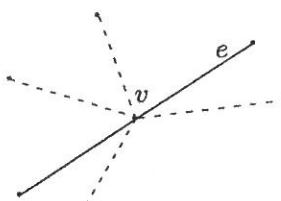
În această secțiune ne vom ocupa de următoarea problemă: Se dau S_1, S_2 prin LDCM. Se cere LDCM pentru $O(S_1, S_2)$. Fiecare față din $O(S_1, S_2)$ trebuie etichetată cu etichetele fețelor din S_1, S_2 care o conțin.

Observația 5.3.1. Această problemă conduce la studiul intersecției segmentelor date de muchiile celor două subdiviziuni ale planului, S_1 și S_2 . Vom utiliza un algoritm de măturare plană.

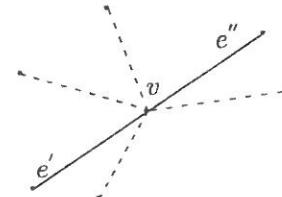
Structurile de date sunt:

- coada de evenimente \mathcal{Q} (formată din capetele segmentelor și punctele de intersecție);
- starea liniei de măturare \mathcal{T} (formată din segmentele care intersectează linia de măturare, ordonate de la stânga la dreapta);
- LDCM obținută din copierea celor două LDCM ale lui S_1, S_2 într-una singură, notată \mathcal{D} . Este căt se poate de clar faptul că inițial \mathcal{D} nu este o LDCM, numai după rularea algoritmului va fi corectă.

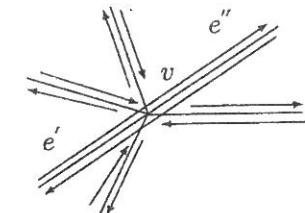
Pentru început, trebuie să stabilim ce se întâmplă într-un eveniment care este o intersecție. De exemplu, să luăm cazul în care o muchie e din S_1 trece printr-un vârf v al lui S_2 .



Muchia e trebuie înlocuită cu două muchii e' și e'' ; fiecare dintre ele dă câte două semi-muchii în LDCM.



Acstea trebuie orientate astfel încât să avem o ordonare ciculară în jurul lui v :



Pentru a stabili modul în care se completează informațiile referitoare la fețe, facem câteva observații.

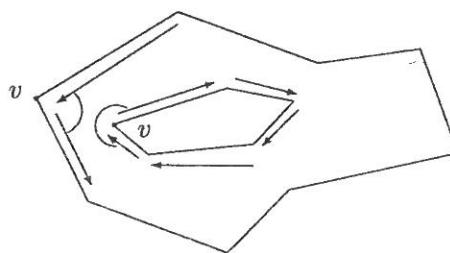
Observația 5.3.2. Numărul fețelor lui $O(S_1, S_2)$ este $1 +$ numărul frontierelor exterioare (fiecare față, cu excepția celei infinite, conține o frontieră exterioară).

Observația 5.3.3. Fiecare frontieră este un ciclu într-un graf.

Așadar, din tabelul LDCM, \mathcal{D} , pentru semi-muchii, putem determina toate ciclurile din $O(S_1, S_2)$.

- Trebuie să facem diferența între frontieră exterioară a unei fețe și frontieră unei găuri:

5.3. Suprapunerea a două subdiviziuni ale planului

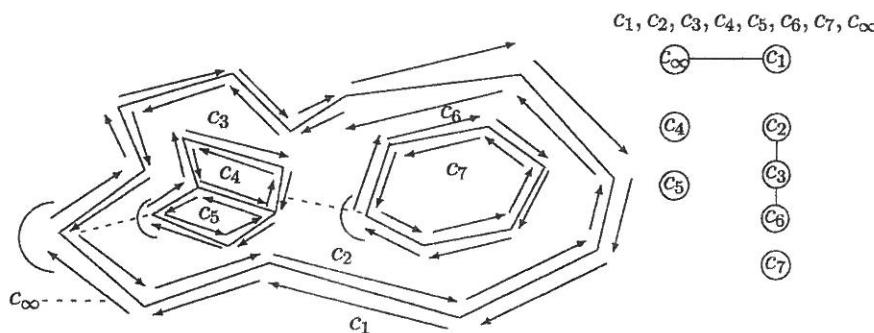


Alegem cel mai din stânga vârf v din ciclu; dacă sunt mai multe, îl alegem pe cel mai de jos. Calculăm unghiul interior al celor două muchii incidente în v . Dacă este mai mic decât π , atunci frontieră ciclului este frontieră exteroară a unei fețe.

- Trebuie să stabilim care sunt ciclurile care mărginesc aceeași față. Construim un graf G care are câte un nod pentru fiecare ciclu, plus un nod pentru fața exteroară c_∞ . Între două noduri există un arc dacă și numai dacă unul dintre cicluri este o gaură, iar celălalt are o semi-muchie imediat în stânga celui mai din stânga vârf al primului ciclu. Dacă nu este nimic la stânga celui mai din stânga vârf al unui ciclu, nodul care îl reprezintă se leagă cu c_∞ .

Prezentăm, în continuare, un exemplu în acest sens.

Exemplul 5.3.1.



Propoziția 5.3.1. Fiecare componentă conexă a grafului G corespunde exact ciclurilor incidente unei fețe.

În concluzie, cu ajutorul grafului G vom putea găsi $IncidentFace(\vec{e})$.

5.4. Aplicații ale algoritmului MAPOVERLAY

Observația 5.3.4. Etapele construirii grafului G sunt următoarele:

- se caută câte un nod pentru fiecare ciclu;
- se consideră cel mai din stânga vârf al unui ciclu care este o gaură și se stabilește dacă trebuie adăugat un arc;
- se etichetează fiecare față a suprapunerii.

Prezentăm, în continuare, algoritmul de suprapunere a două subdiviziuni S_1 și S_2 ale planului, $MAPOVERLAY(S_1, S_2)$:

Se dă: S_1, S_2 date prin LDCM
Se cere: $O(S_1, S_2)$, dată prin LDCM

- (1) copiază cele două LDCM într-o singură \mathcal{D}
- (2) determină intersecțiile muchiilor din S_1, S_2 , cu algoritmul de măturare plană
- (3) actualizează \mathcal{D} în evenimentele care implică muchii din S_1 și S_2
- (4) stochează muchia situată imediat la stânga evenimentului
- (5) dacă evenimentul este vârf al unei subdiviziuni
- (6) atunci
 - (7) determină fața celeilalte subdiviziuni care îl conține
 - (8) // în acest moment \mathcal{D} coincide cu $LDCM(S_1, S_2)$
 - (9) determină ciclurile de frontieră din $O(S_1, S_2)$, din \mathcal{D}
 - (10) construiește G
 - (11) pentru fiecare componentă conexă a lui G execută
 - (12) fie C ciclul corespunzător frontierei exterioare din componentă
 - (13) fie f fața mărginită de ciclu
 - (14) creează $Face(f)$, setează $OuterComponent(f)$ și $InnerComponent(f)$
 - (15) etichetează fiecare față a lui $O(S_1, S_2)$ cu etichetele fețelor care o conțin

Observația 5.3.5. Fie S_1, S_2 subdiviziuni ale planului, cu n_1 și respectiv n_2 vârfuri și fie $n = n_1 + n_2$. Complexitatea algoritmului $MAPOVERLAY(S_1, S_2)$ este $O((n+k)\lg n)$, unde k este numărul vârfurilor lui $O(S_1, S_2)$.

5.4 Aplicații ale algoritmului MAPOVERLAY

Două poligoane P_1, P_2 pot fi privite ca subdiviziuni ale planului ale căror fețe mărginite sunt P_1, P_2 . Putem aplica algoritmul de mai sus astfel:

- Pentru $P_1 \cap P_2$ extragem fețele etichetate cu P_1 și P_2 ;
- Pentru $P_1 \cup P_2$ extragem fețele etichetate cu P_1 sau P_2 ;

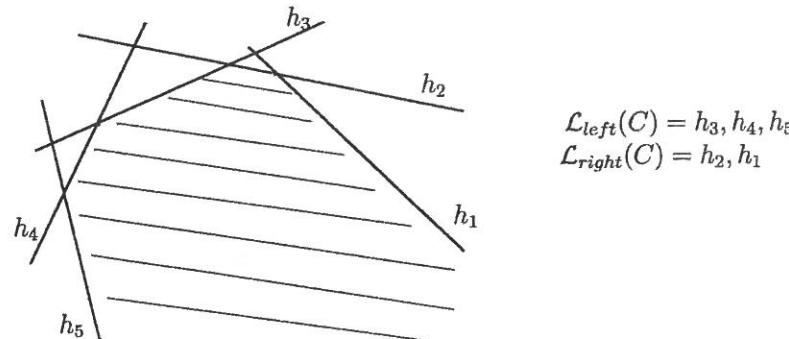
- Pentru $P_1 \setminus P_2$ extragem fețele etichetate cu P_1 , dar nu P_2 .

Observația 5.4.1. Fie P_1, P_2 poligoane cu n_1 respectiv n_2 vârfuri. Atunci complexitatea algoritmului prin care se determină $P_1 \cap P_2$, $P_1 \cup P_2$, $P_1 \setminus P_2$ este $\mathcal{O}(n \log n + k \log n)$, unde $n = n_1 + n_2$ și k este numărul vâfurilor lui $P_1 \cap P_2$ respectiv $P_1 \cup P_2$, $P_1 \setminus P_2$.

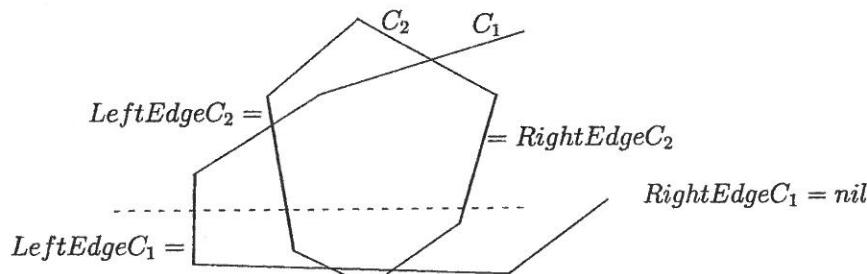
Să revenim la intersecția a două regiuni poligonale convexe. Dacă folosim algoritmul din Capitolul 1, complexitatea acestuia va fi $\mathcal{O}(n \lg n)$, deoarece numărul punctelor de intersecție este mai mic sau egal cu n .

În continuare vom prezenta un algoritm mai eficient decât cel anterior.

Frontiera unei regiuni poligonale convexe C se împarte în $\mathcal{L}_{left}(C)$ și $\mathcal{L}_{right}(C)$, utilizându-se vârful pentru care $y = max$:

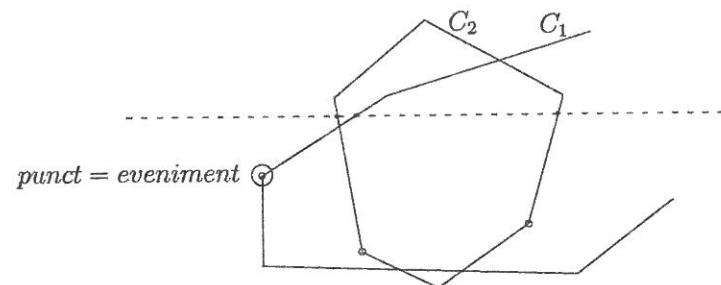


Folosim un algoritm de măturare plană și studiem laturile lui C_1, C_2 care sunt intersectate de linia de măturare și care vor fi notate prin $LeftEdge_{C_1}$, $RightEdge_{C_1}$, $LeftEdge_{C_2}$, $RightEdge_{C_2}$.

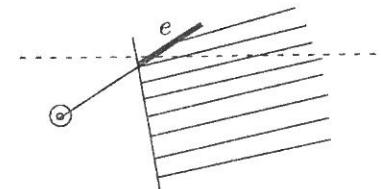


Se pornește de la vârful $v_1 \in C_1$ pentru care $y_1 = max$; eventual $y_1 = \infty$; la fel $v_2 \in C_2$ pentru care $y_2 = max$. Atunci $y_{start} = \min\{y_1, y_2\}$. Studiem $y \leq y_{start}$.

Evenimentele sunt acele puncte în care linia de măturare începe respectiv închetează să intersecteze laturile poligoanelor. Un *punct=eveniment* care determină latura ce trebuie analizată se găsește astfel: se consideră capetele inferioare ale muchiilor intersectate de linia de măturare și se alege acela pentru care $y = maxim$:



Fiecare *punct=eveniment* determină o nouă latură e pe frontiera regiunii convexe căutate:



Acum trebuie să stabilim dacă latura respectivă este pe frontiera poligonului C_1 sau a lui C_2 , după care stabilim dacă este în partea stângă sau în cea din dreapta a frontierei poligonului respectiv.

Presupunem că e se află pe partea din stânga a frontierei lui C_1 și notăm cu p extremitatea lui e având $y = max$. Găsim trei situații distincte:

- p este capătul de sus al unei laturi a lui C , caz în care p se va afla în interiorul lui C_2 (trebuie doar să testăm dacă p este între $LeftEdge_{C_2}$ și $RightEdge_{C_2}$ și să adăugăm semiplanul pe căruia frontiera se află e la $\mathcal{L}_{left}(C)$);

- $e \cap LeftEdge_{C_2}$ este capătul de sus al lui e , caz în care punctul de intersecție va fi un vârf al lui C (stabilim dacă latura lui C ce pornește din acel punct este conținută în e sau în $LeftEdge_{C_2}$ și adăugăm semiplanul corespunzător la $\mathcal{L}_{left}(C)$);

- $e \cap RightEdge_{C_2}$ este capătul de sus al lui e ; fie părți din ambele laturi dau câte

o latură a lui C care pornește din punctul respectiv (adăugăm semiplanul delimitat de e la $\mathcal{L}_{left}(C)$ și semiplanul delimitat de $RightEdgeC_2$ la $\mathcal{L}_{right}(C)$), fie punctul respectiv de intersecție este capătul de jos pentru două astfel de laturi (nu trebuie adăugat nici un semiplan).

Observația 5.4.2. Algoritmul ce determină intersecția a două regiuni poligonale din plan are complexitatea $\mathcal{O}(n)$.

Observația 5.4.3. Algoritmul ce determină intersecția a n semiplane are complexitatea $\mathcal{O}(n \log n)$.

Capitolul 6

Diagrama Voronoi

6.1 Definiție. Proprietăți fundamentale.

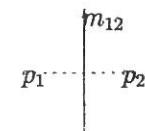
Fie $S = \{p_1, p_2, \dots, p_n\}$ o mulțime de puncte din plan.

Definiția 6.1.1. Diagrama Voronoi a mulțimii S este o partiție a planului în n regiuni V_1, V_2, \dots, V_n astfel încât:

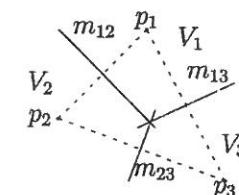
$$\begin{cases} p_i \in V_i, i = \overline{1, n} \\ q \in V_i \Leftrightarrow dist(q, p_i) \leq dist(q, p_j), \forall p_j \in S, j \neq i \end{cases}$$

Notăm $V_i = \mathcal{V}(p_i) = \{x \mid dist(x, p_i) \leq dist(x, p_j), j = \overline{1, n}, j \neq i\}$.

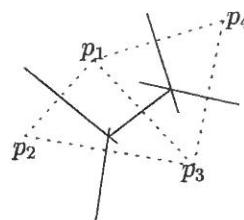
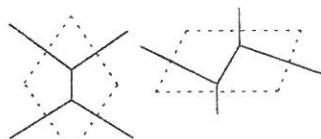
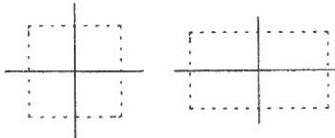
Observația 6.1.1. Dacă $n = 2$, pentru a obține diagrama Voronoi a mulțimii $S = \{p_1, p_2\}$ utilizăm mediatoarea m_{12} a segmentului p_1p_2 .



În cazul în care $n = 3$, diagrama Voronoi se obține cu ajutorul mediatoarelor laturilor triunghiului $p_1p_2p_3$.

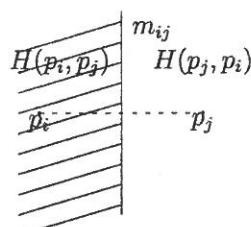


Cazul $n = 4$ este prezentat în figurile următoare:



Observația 6.1.2. V_1, V_2, \dots, V_n sunt regiuni poligonale convexe (posibil nemărginite) delimitate de cel mult $n - 1$ laturi (segmente, semidrepte sau drepte) și având cel mult $n - 1$ vârfuri.

Observația 6.1.3. Fie $p_i, p_j \in S, i \neq j$. Fie m_{ij} mediatoarea segmentului $p_i p_j$ și fie $H(p_i, p_j)$ semiplanul delimitat de m_{ij} , care conține p_i :



$$\text{Avem } V_i = \mathcal{V}(p_i) = \bigcap_{\substack{j=1, n \\ j \neq i}} H(p_i, p_j)$$

Vom nota în continuare cu $\text{Vor}(S)$ diagrama Voronoi a mulțimii S .

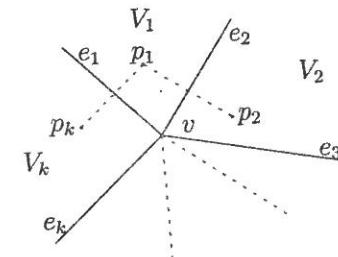
Definiția 6.1.2. $V_i, i = \overline{1, n}$, se numesc **camere** sau **poligoane Voronoi** în punctul p_i . Vârfurile diagramei $\text{Vor}(S)$ se numesc **vârfuri Voronoi**, iar muchile, **muchii Voronoi**.

Este clar faptul că în cazul în care cele n puncte ale mulțimii S sunt coliniare, diagrama $\text{Vor}(S)$ constă în $n - 1$ drepte paralele și n poligoane Voronoi delimitate de acestea.

În continuare vom presupune că **punctele din S sunt în poziție generală**, adică nu există trei puncte coliniare sau patru puncte conciclice.

Propoziția 6.1.1. *Orice vârf Voronoi are gradul exact 3.*

Demonstrație. Fie v un vârf Voronoi, care este intersecția muchiilor Voronoi e_1, e_2, \dots, e_k :

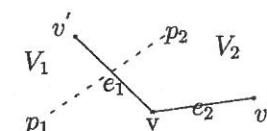


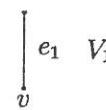
Deoarece $v \in e_1$, rezultă că $\text{dist}(v, p_1) = \text{dist}(v, p_k)$. În același mod, din $v \in e_2$, rezultă că $\text{dist}(v, p_1) = \text{dist}(v, p_2)$. Obținem egalitățile următoare:

$$\text{dist}(v, p_1) = \text{dist}(v, p_2) = \dots = \text{dist}(v, p_k).$$

De aici rezultă că punctele p_1, p_2, \dots, p_k sunt conciclice. Pe de altă parte, punctele lui S fiind în poziție generală, vom avea $k \leq 3$. Vom arăta că egalitățile $k = 1$ sau $k = 2$ nu pot avea loc.

Cazul $k = 2$ presupune că avem două muchii e_1, e_2 incidente în v și două poligoane Voronoi $V_1 = \mathcal{V}(p_1)$ și $V_2 = \mathcal{V}(p_2)$. Din faptul că $\text{dist}(v, p_1) = \text{dist}(v, p_2)$ rezultă că $v \in m_{12}$. Deoarece $e_1 e_2 \subset m_{12}$ deducem că v nu ar fi vârf, ci un punct interior al unei muchii Voronoi (caz degenerat și anume trei puncte coliniare).





Cazul $k = 1$ presupune că ambele "părți" ale lui e_1 sunt în V_1 , deci poligonul Voronoi V_1 nu este convex, contradicție. \square

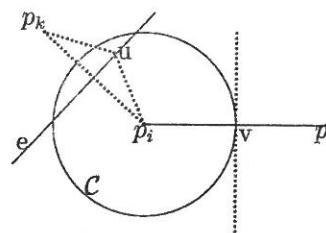
Observația 6.1.4. Pentru orice vârf Voronoi v există exact 3 poligoane Voronoi incidente în acel vârf.

Propoziția 6.1.2. Fie $p_i \in S$ fixat. Cel mai apropiat punct de p_i din S are o mulțime Voronoi pe frontieră poligonului Voronoi în p_i .

Demonstrație. Fie $p_j \in S$ astfel încât $dist(p_i, p_j) = \min_{\substack{p \in S \\ p \neq p_i}} dist(p_i, p)$. Fie v mijlocul segmentului p_ip_j și fie cercul $\mathcal{C} = \mathcal{C}(p_i, \frac{1}{2}dist(p_i, p_j))$.

Demonstrăm prin reducere la absurd că acest cerc este conținut în întregime în poligonul Voronoi V_i .

Presupunem că există o mulțime e a poligonului Voronoi V_i care conține un punct u interior cercului \mathcal{C} .



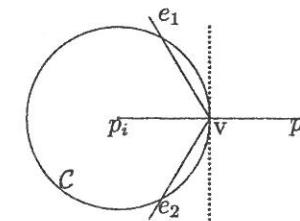
Atunci $e \subset m_{ik}$, unde $p_k \in S$. Au loc următoarele:

$$dist(p_i, p_k) \leq 2 \cdot dist(p_i, u) < 2 \cdot dist(p_i, v) = dist(p_i, p_j) = \min_{\substack{p \in S \\ p \neq p_i}} dist(p_i, p)$$

Aceasta contrazice faptul că p_j este cel mai aproape de p_i în S .

Demonstrăm acum că v este punct interior al unei mulțimi Voronoi. Pentru început observăm faptul că $\forall x \in vp_j, x \neq v$, avem $dist(x, p_j) < dist(x, p_i)$, ceea ce arată că $vp_j \setminus \{v\} \subset extV_i$. Pe de altă parte, $v \in \mathcal{C}$, deci $v \in \partial V_i$.

Presupunem că v este un vârf Voronoi. Fie e_1, e_2 mulțimile Voronoi de pe ∂V_i , incidente în vârful v .

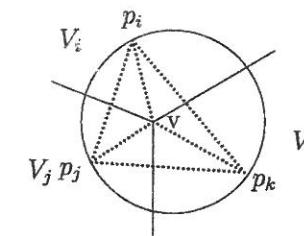


Deoarece V_i este poligon convex, rezultă că $\angle e_1 v e_2 < \pi$. Atunci e_1 sau e_2 au puncte interioare cercului \mathcal{C} , contradicție. Deducem că $\angle e_1 v e_2 = \pi$, adică $e_1 = e_2$. În concluzie, $v \in int e_1$, e_1 fiind tangentă cercului \mathcal{C} în punctul v . \square

Observația 6.1.5. Am văzut că în orice vârf Voronoi v există exact trei poligoane Voronoi incidente. Fie acestea V_i, V_j, V_k . Atunci $V_i = \mathcal{V}(p_i)$, $V_j = \mathcal{V}(p_j)$, $V_k = \mathcal{V}(p_k)$. Observăm că au loc egalitățile

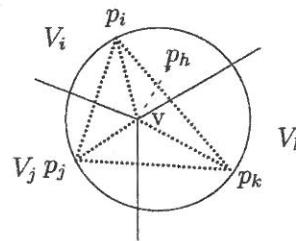
$$dist(v, p_i) = dist(v, p_j) = dist(v, p_k),$$

adică vârful v este centrul cercului circumscris $\triangle p_ip_jp_k$. Notăm acest cerc cu $\mathcal{C}(v) = \mathcal{C}(v, dist(v, p_i))$.



Propoziția 6.1.3. Pentru orice vârf Voronoi v avem $int \mathcal{C}(v) \cap S = \emptyset$, adică interiorul cercului $\mathcal{C}(v)$ nu conține puncte din S .

Demonstrație. Faptul că $p_i, p_j, p_k \notin int \mathcal{C}(v)$ este evident. Fie $p_h \in S \setminus \{p_i, p_j, p_k\}$ astfel încât $p_h \in int \mathcal{C}(v)$.



Atunci $\text{dist}(v, p_h) < r = \text{dist}(v, p_i)$, deci $v \in V_h$. Deoarece $v \in V_i$, $v \in V_j$, $v \in V_k$, va rezulta că gradul lui v este cel puțin patru; cu alte cuvinte, există patru poligoane Voronoi incidente în v , ceea ce contrazice Propoziția 6.1.1. \square

Pe baza Propozițiilor 6.1.1, 6.1.2 și 6.1.3 obținem:

Teorema 6.1.1. Fie $\text{Vor}(S)$ diagrama Voronoi a unei mulțimi finite de puncte S . Au loc următoarele proprietăți:

- (i) Un punct p este vârf pentru $\text{Vor}(S)$ dacă și numai dacă cercul de rază maximă cu centru în p care nu conține nici un punct din S în interior, notat $C_S(p)$, are trei puncte din S pe frontieră sa.
- (ii) Mediatoarea unui segment p_ip_j , $p_i, p_j \in S$, definește o muchie a diagramei $\text{Vor}(S)$ dacă și numai dacă există un punct q din plan astfel încât $\partial C_S(q) \cap S = \{p_i, p_j\}$.

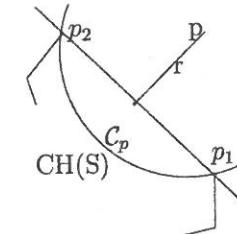
În continuare vom arăta că există o relație strânsă între diagrama Voronoi $\text{Vor}(S)$ și învelitoarea convexă $\text{CH}(S)$ a unei mulțimi S de puncte din plan.

Propoziția 6.1.4. Două puncte din S sunt vârfuri consecutive pentru $\text{CH}(S)$ dacă și numai dacă poligoanele Voronoi corespunzătoare acestor puncte au în comun o muchie Voronoi. Această muchie va fi o semidreaptă.

Demonstrație. Fie $p_1, p_2 \in S$ și $V_1 = \mathcal{V}(p_1)$, $V_2 = \mathcal{V}(p_2)$.

" \Rightarrow " Demonstrăm că dacă p_1, p_2 sunt vârfuri consecutive în $\text{CH}(S)$, atunci p_1p_2 este muchie pentru $\partial \text{CH}(S)$.

Fie $p_1p_2 \subset l$, l fiind dreapta suport a segmentului p_1p_2 . Atunci unul dintre semiplanele determinate de l nu conține nici un punct din S .



Fie r semidreapta din acest semiplan, cu originea în mijlocul lui p_1p_2 , având proprietatea $r \perp p_1p_2$. Fie $p \in r$ și fie cercul $C_p = C(p, pp_1)$.

Considerăm că p se deplasează pe r către punctul de la infinit. Atunci C_p se apropie oricăr de mult de dreapta l . Deoarece S este mulțime finită, rezultă că există un punct $p_0 \in r$ astfel încât pentru orice alt punct $p \in r$, $\text{dist}(p, l) > \text{dist}(p_0, l)$. Deci $C_p \cap (S \setminus \{p_1, p_2\}) = \emptyset$, de unde obținem

$$V_1 \cup V_2 = \{p \in r \mid \text{dist}(p, l) > \text{dist}(p_0, l)\}.$$

" \Leftarrow " Presupunem că V_1, V_2 au în comun o muchie Voronoi care este o semidreaptă. Fie aceasta r ; pentru orice punct $p \in r$, vom avea $\text{dist}(p, p_1) = \text{dist}(p, p_2)$, de unde rezultă $r \subset m_{12}$.

Fie C_p un cerc cu centru într-un punct $p \in r$ astfel încât $p_1, p_2 \in C_p$; $C_p \cup (S \setminus \{p_1, p_2\}) = \emptyset$.

Dacă p se deplasează pe r către punctul de la infinit, atunci C_p se apropie oricăr de mult de dreapta suport l a lui p_1p_2 . Deducem că unul dintre semiplanele delimitate de l nu conține puncte din S , deci p_1p_2 este muchie pentru $\partial \text{CH}(S)$. În concluzie, p_1 și p_2 sunt vârfuri consecutive. \square

Corolarul 6.1.1. Un poligon Voronoi $V_i = \mathcal{V}(p_i)$ este nemărginit dacă și numai dacă p_i este vârf pentru $\text{CH}(S)$.

Observația 6.1.6. Putem vorbi despre graful asociat unei diagrame Voronoi, pentru care nodurile sunt vârfurile Voronoi, iar muchiile sunt muchiile Voronoi. Deoarece acesta nu este un graf propriu-zis, considerăm un nou vârf v_∞ și considerăm că o semidreaptă va uni v_∞ cu un vârf al diagramei Voronoi. Este imediat faptul că în acest mod am obținut un graf conex.

Teorema 6.1.2. Diagrama Voronoi a unei mulțimi de n puncte din plan conține cel mult $2n - 5$ vârfuri, $3n - 6$ muchii și exact n regiuni.

Demonstrație. Faptul că există exact n regiuni este evident.

Vom demonstra afirmațiile de mai sus folosind teorema lui Euler, conform căreia are

loc egalitatea $v - e + f = 2$. Numărul v al vârfurilor în formula lui Euler va fi dat de cele $v - 1$ vârfuri propriu-zise, la care se adaugă v_∞ . În plus, $f = n$, $3v \leq 2e$, deoarece fiecare vârf are gradul 3, cu excepția lui v_∞ care poate avea gradul mai mare:

$$\sum_{x \in V} d(x) = 2e, \quad \sum_{x \in V} d(x) \geq 3v.$$

Folosind toate aceste informații obținem:

$$2 - n = v - e \leq \frac{2}{3}e - e = -\frac{1}{3}e$$

$$6 - 3n \leq -e$$

$$e \leq 3n - 6$$

$$v \leq \frac{2}{3}e \leq \frac{2}{3} \cdot 3(n - 2) = 2n - 4$$

deci numărul vârfurilor diagramei Voronoi este cel mult $v - 1 \leq 2n - 5$. \square

6.2 Construcția diagramei Voronoi

Prezentăm, în continuare, algoritmul de construcție a diagramei Voronoi:

Se dă: o mulțime S de n puncte din plan

Se cere: $Vor(S)$

- (1) sortează punctele din S după ordonată
- (2) împarte S în două submulțimi S_L și S_R de cardinală aproximativ egale, printr-o dreaptă verticală
- (3) construiește $Vor(S_L)$ și $Vor(S_R)$
- (4) unește $Vor(S_L)$ și $Vor(S_R)$ pentru a obține $Vor(S)$

Observăm că va fi necesară rezolvarea următoarelor probleme:

1. Vor exista muchii ale diagramei $Vor(S)$ determinate de un punct din S_L și altul din S_R . Acestea trebuie adăugate la linia (4) a secvenței de cod de mai sus.
2. Vor exista muchii ale lui $Vor(S_L)$, $Vor(S_R)$ care trebuie șterse.

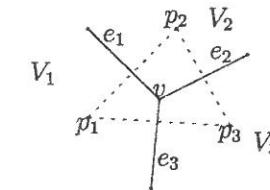
Vom rezolva, pentru început, prima problemă.

Fie σ subgraful lui $Vor(S)$ format din muchiile Voronoi determinate de perechile de puncte $p_i, p_j \in S$ astfel încât $p_i \in S_L$ și $p_j \in S_R$.

Propoziția 6.2.1. Orice vârf al lui σ are gradul exact 2.

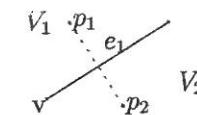
Demonstrație. σ fiind subgraf al lui $Vor(S)$, deducem că orice vârf al lui σ are gradul cel mult 3. Arătăm că gradul nu poate fi 3 sau 1.

Presupunem că pentru un vârf v al lui σ avem $d(v) = 3$; fie e_1, e_2, e_3 muchii incidente în v și V_1, V_2, V_3 poligoane incidente în v . Fie $V_1 = \mathcal{V}(p_1)$, $V_2 = \mathcal{V}(p_2)$, $V_3 = \mathcal{V}(p_3)$.



Presupunem că $p_1 \in S_L$ și $p_2 \in S_R$. Din $p_1 \in S_L$ rezultă $p_3 \in S_R$, iar din $p_2 \in S_R$ rezultă $p_3 \in S_L$, contradicție.

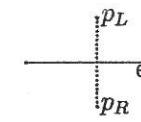
Presupunem că $d(v) = 1$. Fie e_1 muchia (unică) incidentă în v , aceasta fiind situată pe mediatotarea segmentului p_1p_2 .



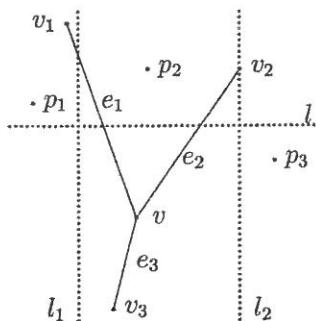
Presupunem că $p_1 \in S_L$ și $p_2 \in S_R$. Fie $p_3 \in S$. Dacă $p_3 \in S_L$, atunci $\exists e_2 \in \sigma$ (deoarece $p_2 \in S_R$ și $p_3 \in S_L$). La fel, dacă $p_3 \in S_R$, atunci $\exists e_3 \in \sigma$. Însă oricare dintre aceste situații contrazice faptul că gradul lui v este 1. \square

Propoziția 6.2.2. Orice componentă conexă a lui σ este strict monotonă. Altfel spus, orice dreaptă orizontală intersectează o componentă conexă a lui σ exact într-un punct.

Demonstrație. Arătăm, mai întâi, că nici o muchie a lui σ nu poate fi orizontală. Presupunând altfel, fie $p_L \in S_L$ și $p_R \in S_R$ astfel încât muchia determinată de cele două puncte să fie orizontală. Atunci segmentul p_Lp_R este vertical, deci S_L și S_R nu sunt separate de o dreaptă verticală.



Să presupunem că există o componentă conexă C a lui σ care nu este monotonă. Fie v_1, v_2, v , noduri ale lui σ , ca în figura următoare.



Fie v_3 vârf Voronoi astfel încât e_3 este cea de-a treia muchie incidentă în v . v_3 este sub dreapta l , deoarece poligoanele Voronoi sunt convexe. Fie $p_1, p_2, p_3 \in S$ punctele care definesc muchiile e_1, e_2, e_3 . Presupunem că $p_2 \in S_R$; atunci $p_1 \in S_L$ și $p_3 \in S_L$.

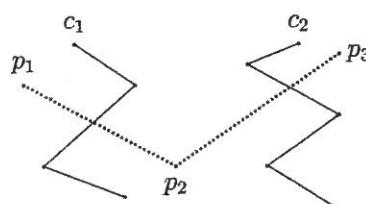
Fie l_1, l_2 drepte verticale astfel încât p_1 este în stânga lui l_1 , p_2 în dreapta lui l_1 , p_2 este în stânga lui l_2 și p_3 este în dreapta lui l_2 . Atunci p_1, p_2 sunt ambele în S_L ; contradicție. Rezultă deci că orice componentă conexă a lui σ este monotonă. \square

Pe baza unui raționament asemănător celui de mai sus, se demonstrează afirmația următoare:

Propoziția 6.2.3. σ are exact o componentă conexă.

Demonstrație. Faptul că există cel puțin o componentă conexă este evident deoarece $Vor(S)$ este conexă.

Presupunând că există C_1, C_2 componente conexe ale lui σ , se ajunge la o contradicție, la fel ca în cazul dreptelor l_1, l_2 din demonstrația anterioară.



\square

Revenim acum cu cea de-a doua problemă subliniată la începutul acestei secțiuni, și anume aceea că vor exista muchii ale lui $Vor(S_L)$, $Vor(S_R)$ care vor trebui sterse.

Rezolvarea acestei probleme se bazează pe următoarea proprietate:

Propoziția 6.2.4. Fie e o muchie Voronoi (sau o parte a ei) a lui $Vor(S_L)$ respectiv $Vor(S_R)$. Muchia e trebuie să fie în $Vor(S)$ dacă și numai dacă este în dreapta respectiv în stânga subgrafului σ .

Prezentăm algoritmul de "unire" a diagramelor $Vor(S_L)$ și $Vor(S_R)$:

Se dă: $Vor(S_L)$ și $Vor(S_R)$.

Se cere: $Vor(S)$.

(1) construiește σ

(2) șterge muchiile și muchiile parțiale ale lui $Vor(S_L)$ care sunt la dreapta lui σ

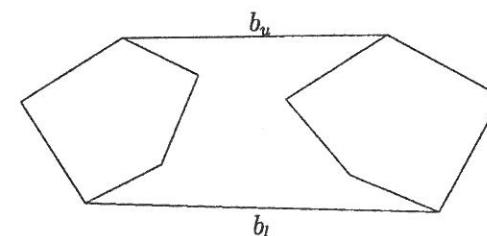
(3) șterge muchiile și muchiile parțiale ale lui $Vor(S_R)$ care sunt la stânga lui σ

În continuare vom vedea cum se construiește σ . Acesta va avea două muchii care sunt, de fapt, semidrepte. Știm că acestea corespund perechilor de vârfuri consecutive ale învelitorii convexe.

În concluzie, găsim $CH(S_L)$ și $CH(S_R)$ și construim mediatoarele pentru:

- segmentul determinat de $p_1 \in S_L$ ($y_1 = \max$) și $p_2 \in S_R$ ($y_2 = \max$);
- segmentul determinat de $p'_1 \in S_L$ ($y'_1 = \min$) și $p'_2 \in S_R$ ($y'_2 = \min$).

Notăm cu b_u respectiv b_l aceste segmente (b_u - "upper bridge", b_l - "lower bridge").



Prezentăm, în continuare, algoritmul de construcție a lui σ :

Se dă: $Vor(S_L)$, $Vor(S_R)$, $CH(S_L)$, $CH(S_R)$.

Se cere: σ și $CH(S)$.

- (1) determină b_u și b_l pentru $CH(S_L)$ și $CH(S_R)$
- (2) construiește $CH(S)$ cu b_u și b_l
- (3) construiește mediatoarele m_u și m_l ale segmentelor b_u respectiv b_l
- (4) traversează σ pornind de la capătul infinit al lui m_u , construiește fiecare latură a lui σ , până la capătul infinit al lui m_l

Observația 6.2.1. Diagramele $Vor(S_L)$ și $Vor(S_R)$ vor fi date prin LDCM.

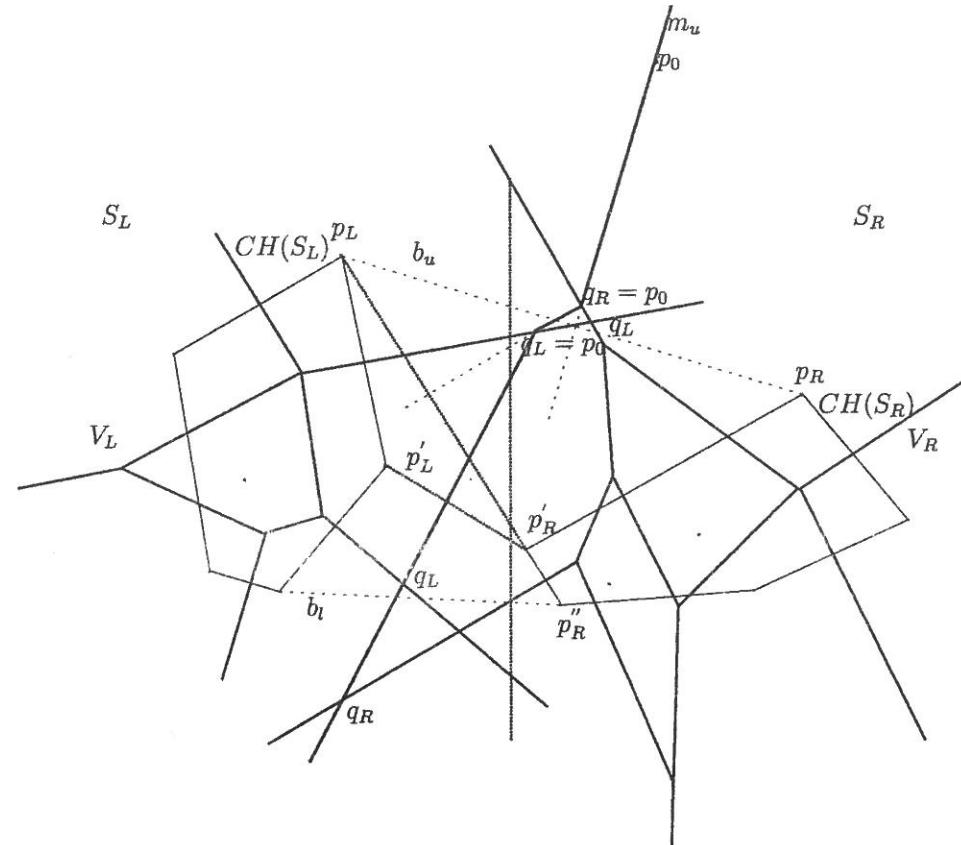
Vom preciza acum în ce constă linia (4) din algoritmul de mai sus, prin descrierea algoritmului SIGMA.

Algoritmul SIGMA:

- (1) fie p_0 pe m_u , suficient de depărtat de $b_u = p_L p_R$
- (2) fie r_0 semidreapta cu originea în p_0 , situată pe m_u , de sens opus muchiei infinite
- (3) fie V_L, V_R poligoanele Voronoi ale punctelor p_L, p_R în diagramele $Vor(S_L), Vor(S_R)$
- (4) cât timp r_0 nu coincide cu m_l
- (5) execută
- (6) determină punctele $\{q_L\} = r_0 \cap \partial V_L$, $\{q_R\} = r_0 \cap \partial V_R$
- (7) dacă p_0 este mai aproape de q_L decât de q_R
- (8) atunci
 - (9) fie q_L pe muchia e_L a lui $Vor(S_L)$, $e_L = p_L p'_L$
 - (10) $p_0 = q_L$
 - (11) $r_0 =$ semidreapta cu originea în p_0 ($= q_L$) situată pe mediatoarea segmentului $p'_L p_R$, în direcția y -descrescător
- (12) fie V_L poligonul Voronoi al lui p'_L
- (13) ultfel
- (14) reactualizează parametrii p_0, r_0 și V_R în același mod

Observația 6.2.2. Fiind dată o mulțime S de n puncte din plan, algoritmul de construire a diagramei $Vor(S)$ are complexitatea $\mathcal{O}(n \lg n)$.

Exemplul 6.2.1. Redăm în figura următoare construcția diagramei Voronoi pe baza algoritmului prezentat mai sus:

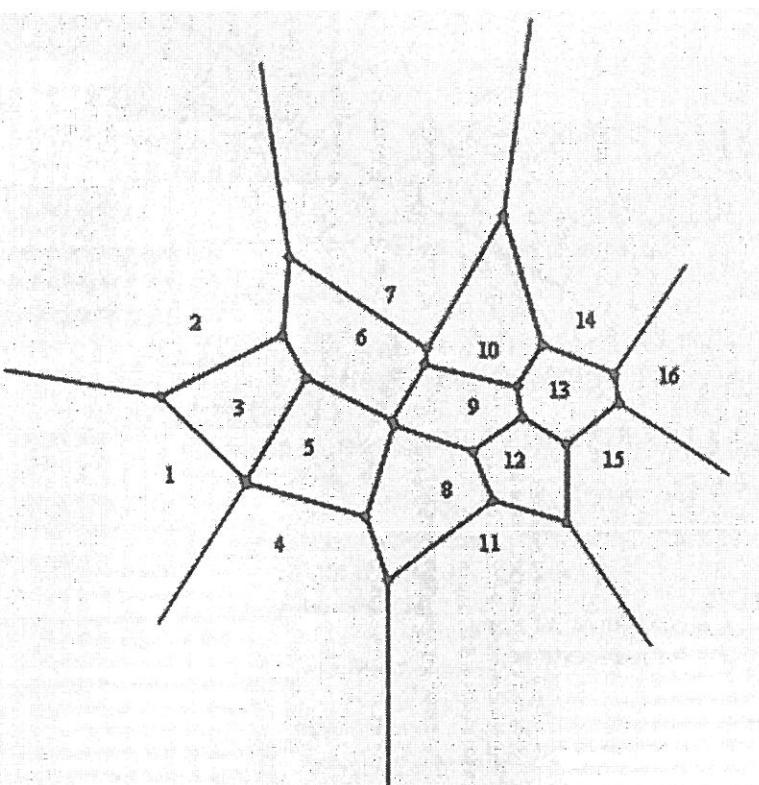


- p_0 pe mediatoarea lui $p_L p_R$;
- q_R, q_L ;
- $q_R = p_0$;
- q_R este pe mediatoarea lui $p_R p'_R$;
- mediatoarea lui $p_L p'_R$ este r_0 ;
- q_L, q_R ;
- $q_L = p_0$;
- q_L este pe mediatoarea lui $p_L p'_L$;
- mediatoarea lui $p'_R p'_L$ este r_0 .

Folosindu-ne de facilitățile grafice ale software-ului *Mathematica* (versiunea 7.0) putem reprezenta în plan graficul diagramei Voronoi astfel:

```
points = {{10.0, 35.0}, {20.0, 90.0}, {35.0, 60.0}, {50.0, 10.0},
{60.0, 45.0}, {80.0, 85.0}, {90.0, 100.0}, {110.0, 30.0},
{120.0, 60.0}, {125.0, 80.0}, {125.0, 10.0}, {134.0, 40.0},
{150.0, 65.0}, {160.0, 90.0}, {170.0, 40.0}, {190.0, 70.0}}
DiagramPlot[points]
```

Comanda *DiagramPlot[]* determină și afișează implicit diagrama Voronoi a unei multimi de puncte. Pentru multimea de puncte precizată, *Mathematica* generează următoarea diagramă Voronoi:



Capitolul 7

Triangularea Delaunay

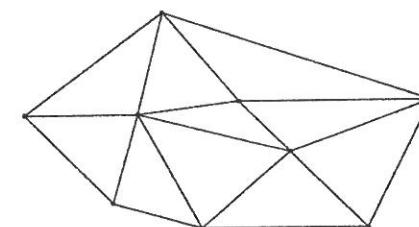
7.1 Triangularea unei multimi finite de puncte din plan

Fie $S = \{p_1, p_2, \dots, p_n\}$ o multime de n puncte din plan, nu toate coliniare. Notăm cu $Vor(S)$ diagrama Voronoi asociată, cu $CH(S)$ învelitoarea convexă a multimii S și cu $\mathcal{T}(S)$ o triangulare a lui S .

Definiția 7.1.1. O triangulare a lui S este o subdiviziune planară maximală (graf planar conex maximal) ale cărei vârfuri sunt din S .

Observația 7.1.1. Orice graf planar conex maximal are toate fețele triunghiuri, exceptând fața infinită. Aceasta este complementara învelitorii convexe $CH(S)$ în E_2 , în timp ce reuniunea fețelor mărginite este $CH(S)$.

Observația 7.1.2. Numărul triunghiurilor este același în orice triangulare a lui S ; la fel și numărul muchiilor. Numărul vârfurilor lui $CH(S)$ nu este neapărat n , ci poate fi mai mic.



Observația 7.1.3. Orice segment care unește două puncte consecutive de pe frontieră $\partial CH(S)$ a învelitorii convexe este muchie în orice triangulare a lui S .

Teorema 7.1.1. Fie S o mulțime de n puncte din plan. Fie k numărul punctelor din S situate pe $\partial CH(S)$. Orice triangulare a lui S are $2n - k - 2$ triunghiuri și $3n - k - 3$ muchii.

Demonstratie. Fie triangularea $\mathcal{T}(S)$ având m triunghiuri; atunci numărul fețelor este $m + 1$. Deoarece o față mărginită are 3 muchii, față infinită are k muchii, iar fiecare muchie este comună pentru exact două fețe, numărul muchiilor este:

$$e = \frac{3m + k}{2}.$$

Pe de altă parte, numărul vârfurilor grafului este $v = n$. Din teorema lui Euler avem:

$$\begin{aligned} v - e + f &= 2 \\ n - \frac{3m + k}{2} + m + 1 &= 2 \\ 2n - 3m - k + 2m - 2 &= 0 \\ 2n - m - k - 2 &= 0 \end{aligned}$$

Am găsit astfel numărul fețelor mărginite:

$$m = 2n - k - 2$$

Putem acum afla numărul muchiilor triangulării:

$$e = \frac{3(2n - k - 2) + k}{2} = 3n - k - 3.$$

□

Fie \mathcal{T} o triangulare a lui S având m triunghiuri. Ordonăm crescător unghiiurile celor m triunghiuri:

$$\alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_{3m}.$$

Definiția 7.1.2. Vectorul $\mathcal{A}(\mathcal{T}) = (\alpha_1, \alpha_2, \dots, \alpha_{3m})$ se numește **vector-unghi asociat triangulării \mathcal{T}** .

Fie \mathcal{T}' o altă triangulare a lui S și $\mathcal{A}(\mathcal{T}') = (\alpha'_1, \alpha'_2, \dots, \alpha'_{3m})$ vectorul-unghi asociat acesteia.

Definiția 7.1.3. Spunem că $\mathcal{A}(\mathcal{T}) = \mathcal{A}(\mathcal{T}')$ dacă $\alpha_i = \alpha'_i$, $i = \overline{1, 3m}$. Notăm

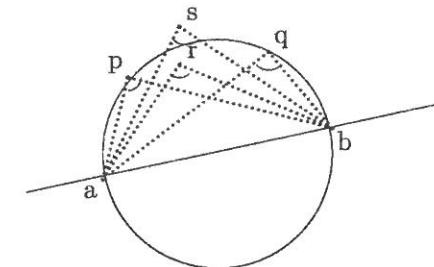
$$\mathcal{A}(\mathcal{T}') < \mathcal{A}(\mathcal{T})$$

dacă există $i \in \{1, \dots, 3m\}$ astfel încât $\alpha_j = \alpha'_j$, $\forall j < i$ și $\alpha_i > \alpha'_i$.

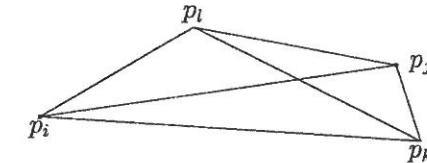
Definiția 7.1.4. O triangulare \mathcal{T} a lui S este **unghiular maximală** dacă pentru orice altă triangulare \mathcal{T}' a lui S avem $\mathcal{A}(\mathcal{T}') < \mathcal{A}(\mathcal{T})$ sau $\mathcal{A}(\mathcal{T}') = \mathcal{A}(\mathcal{T})$.

Observația 7.1.4. Fie C un cerc și fie d o secantă care îl intersectează în punctele a, b . Fie punctele p, q, r, s de aceeași parte a dreptei d astfel încât $p, q \in C$, $r \in \text{int } C$, iar $s \in \text{ext } C$. Observăm că au loc următoarele relații:

$$\angle arb > \angle apb = \angle aqb > \angle asb.$$



Fie $p_i p_j$ o muchie a unei triangulări \mathcal{T} a lui S , astfel încât ea nu este pe $\partial CH(S)$. Atunci $p_i p_j$ este comună pentru două triunghiuri $p_i p_j p_k$ și $p_i p_j p_l$. Presupunem că $p_i p_k p_j p_l$ este convex.



Dacă înlocuim $p_i p_j$ cu $p_k p_l$, obținem o nouă triangulare \mathcal{T}' .

Definiția 7.1.5. Operația descrisă mai sus se numește **schimbarea muchiei**.

Observația 7.1.5. Dacă efectuăm o astfel de operație, unghiiurile $\alpha_1, \alpha_2, \dots, \alpha_6$ în $\mathcal{A}(\mathcal{T})$ se înlocuiesc cu $\alpha'_1, \alpha'_2, \dots, \alpha'_6$ în $\mathcal{A}(\mathcal{T}')$.

Definiția 7.1.6. O muchie $p_i p_j$ se numește **ilegală** dacă

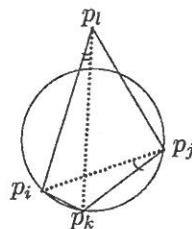
$$\min_{i=1,6} \alpha_i < \min_{i=1,6} \alpha'_i.$$

Observația 7.1.6. Dacă $p_i p_j$ este ilegală și schimbăm această muchie cu $p_k p_l$ ca mai sus, atunci $\mathcal{A}(\mathcal{T}) < \mathcal{A}(\mathcal{T}')$.

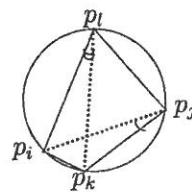
Propoziția 7.1.1. Fie $\mathcal{T}(S)$ o triangulare a mulțimii S și fie $p_i p_j$ muchia comună triunghiurilor $\Delta p_i p_j p_k$ și $\Delta p_i p_j p_l$. Fie C cercul circumscris $\Delta p_i p_j p_k$. Muchia $p_i p_j$ este ilegală dacă și numai dacă $p_l \in \text{int}C$. În plus, dacă $p_i p_k p_j p_l$ este patrilater convex neinscriptibil, exact una dintre muchiile $p_i p_j$ și $p_k p_l$ este ilegală.

Demonstrație. Deosebim următoarele trei cazuri:

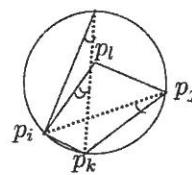
- $p_l \in \text{ext } C$



- $p_l \in C$



- $p_l \in \text{int } C$



Este evident faptul că în primele două cazuri muchia $p_i p_j$ este legală, iar în cel de-al treilea caz această muchie este ilegală, deoarece $\angle p_i p_j p_k < \angle p_i p_l p_k$.

□

Prezentăm, în continuare, **algoritmul de triangulare legală**:

Se dă: o triangulare \mathcal{T} a mulțimii S .

Se cere: o triangulare legală a lui S .

- (1) cât timp \mathcal{T} conține o muchie ilegală $p_i p_j$
- (2) execută
- (3) fie $p_i p_j p_k$ și $p_i p_j p_l$ triunghiurile adiacente muchiei $p_i p_j$
- (4) schimbă $p_i p_j$ cu $p_k p_l$

Observația 7.1.7. Vectorul unghi crește la fiecare iterație. Există un număr finit de triangulații ale lui S , deci un număr finit de pași ai algoritmului, însă algoritmul este lent.

7.2 Triangularea Delaunay

Fie $S = \{p_1, p_2, \dots, p_n\}$ o mulțime de puncte din plan. Pentru început, nu vom presupune că acestea sunt în poziție generală. Fie $Vor(S)$ diagrama Voronoi a lui S .

Definiția 7.2.1. **Graful dual diagramei $Vor(S)$** are un nod în fiecare poligon $V(p_i)$ și o muchie între două noduri dacă poligoanele corespunzătoare acestora au o latură comună.

Din definiția de mai sus deducem că fiecarei muchii Voronoi a diagramei $Vor(S)$ iată corespunde o muchie a grafului dual, iar fiecărui vârf Voronoi iata corespunde o față mărginită a acestui graf.

Definiția 7.2.2. **Graful Delaunay** asociat mulțimii S are câte un nod în fiecare punct al lui S , iar muchiile sunt de forma $p_i p_j$, unde $p_i, p_j \in S$ și poligoanele Voronoi $V(p_i), V(p_j)$ au o latură comună.

Notăm cu $\mathcal{GD} = \mathcal{GD}(S)$ graful Delaunay asociat mulțimii S .

Teorema 7.2.1. *Graful Delaunay al unei mulțimi finite de puncte din plan este un graf planar.*

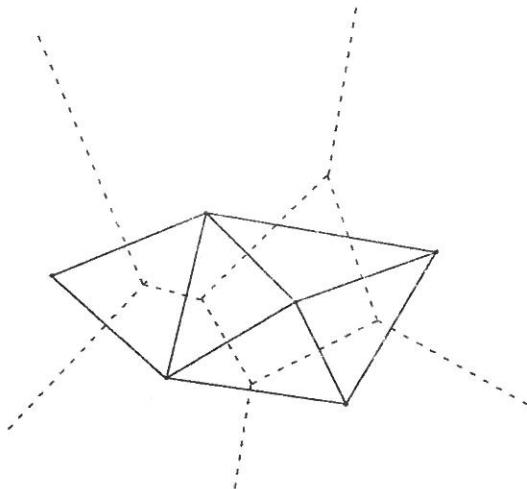
Demonstrație. Fie $S = \{p_1, \dots, p_n\}$. Din Teorema 6.1.1 rezultă faptul că un segment de formă $p_i p_j$, $p_i, p_j \in S$, este muchie a grafului $\mathcal{GD}(S)$ dacă și numai dacă există un disc închis C_{ij} astfel încât $p_i, p_j \in \partial C_{ij}$ și C_{ij} nu mai conține niciun alt punct din mulțimea S . Este clar faptul că un astfel de disc va avea centrul c_{ij} pe latura comună poligoanelor Voronoi $V(p_i)$ și $V(p_j)$.

Fie triunghiul $\Delta p_i p_j c_{ij}$ și fie $p_k p_l$ o altă muchie a grafului $\mathcal{GD}(S)$. În același mod obținem discul C_{kl} și triunghiul $\Delta p_k p_l c_{kl}$. Din modul în care au fost alese cele două discuri rezultă că $\Delta p_i p_j c_{ij} \cap \Delta p_k p_l c_{kl} = \emptyset$, de unde obținem $p_i p_j \cap p_k p_l = \emptyset$.

□

Observația 7.2.1. Graful Delaunay $\mathcal{GD}(S)$ este scufundarea planară a grafului dual diagramei Voronoi $\text{Vor}(S)$.

Observația 7.2.2. Dacă punctele mulțimii S sunt în poziție generală, atunci fiecare vârf Voronoi are gradul 3, deci fețele grafului $\mathcal{GD}(S)$ sunt triunghiuri. De aici apare denumirea de "triangulare Delaunay".



Definiția 7.2.3. O triangulare Delaunay a unei mulțimi S este o triangulare a grafului $\mathcal{GD}(S)$.

O astfel de triangulare se obține adăugând muchii grafului Delaunay. Este clar faptul că atunci când punctele mulțimii date sunt în poziție generală, nu mai trebuie adăugate muchii, în acest caz triangularea Delaunay fiind unică.

Din Teorema 6.1.1 obținem:

Teorema 7.2.2. Fie $S = \{p_1, \dots, p_n\}$ o mulțime de puncte din plan și fie $\mathcal{GD}(S)$ graful Delaunay asociat acesteia.

- (i) Punctele $p_i, p_j, p_k \in S$ sunt vârfuri ale aceleiasi fețe din $\mathcal{GD}(S)$ dacă și numai dacă cercul circumscris $\Delta p_ip_jp_k$ nu conține în interior alte puncte ale lui S .
- (ii) Punctele p_i, p_j determină o muchie a grafului $\mathcal{GD}(S)$ dacă și numai dacă există un disc închis care conține pe frontieră p_i, p_j și nu mai conține alte puncte din S .

Observația 7.2.3. $\mathcal{T} = \mathcal{T}(S)$ este o triangulare Delaunay dacă și numai dacă cercul circumscris oricărui triunghi al lui \mathcal{T} nu conține în interior puncte ale lui S .

Teorema 7.2.3. $\mathcal{T} = \mathcal{T}(S)$ este legală dacă și numai dacă \mathcal{T} este o triangulare Delaunay.

Demonstrație. Este clar faptul că orice triangulare Delaunay este o triangulare legală. Demonstrăm implicația directă prin reducere la absurd.

Fie \mathcal{T} o triangulare legală a unei mulțimi S care nu este o triangulare Delaunay. Teorema 7.2.2 implică faptul că există punctele p_i, p_j, p_k și p_l astfel încât p_l se află în interiorul cercului circumscris triunghiului $\Delta p_ip_jp_k$; altfel spus, patrulaterul $p_ip_jp_kp_l$ nu este inscriptibil. Fie p_ip_j o latură a triunghiului $\Delta p_ip_jp_l$ cu proprietatea că

$$\Delta p_ip_jp_k \cap \Delta p_ip_jp_l = p_ip_j.$$

Fie perechea $(p_ip_jp_k, p_l)$ astfel încât unghiul $\angle p_ip_jp_l$ este maxim (eventual schimbări indicii). Considerăm triunghiul $\Delta p_ip_jp_m$ situat de cealaltă parte a laturii p_ip_j față de triunghiul $\Delta p_ip_jp_k$. Din Propoziția 7.1.1 rezultă că punctul p_m nu poate fi în interiorul cercului circumscris triunghiului $\Delta p_ip_jp_k$; deducem că p_l este situat pe acest cerc. Dacă presupunem că p_jp_m are proprietatea $\Delta p_ip_jp_m \cap \Delta p_jp_mp_l = \emptyset$, atunci are loc inegalitatea $\angle p_ip_jp_l > \angle p_jp_ip_m$, ceea ce contrazice alegerea perechii $(p_ip_jp_k, p_l)$. \square

Observația 7.2.4. Din teorema de mai sus rezultă că orice triangulare unghiular optimă a lui S este o triangulare Delaunay.

7.3 Un algoritm de triangulare Delaunay

Prezentăm, în continuare, etapele obținerii unei triangulări Delaunay:

- I. Vom porni de la un triunghi suficient de "larg" astfel ca mulțimea S să fie situată în interiorul acestui triunghi. Fie $\Omega = \{p_{-1}, p_{-2}, p_{-3}\}$ mulțimea vârfurilor triunghiului. Determinăm triangularea Delaunay a mulțimii $S \cup \Omega$.
- II. Pornim cu o submulțime a lui $S \cup \Omega$ și adăugăm câte un punct p_i astfel încât să avem tot timpul o triangulare Delaunay a mulțimii cu care lucrăm. Inițial vom considera chiar submulțimea Ω , pentru care triangularea este $\mathcal{T}(\Omega) = \Delta p_{-1}p_{-2}p_{-3}$.
- III. Eliminăm din triangularea Delaunay a mulțimii $S \cup \Omega$ muchiile care unesc vârfuri din S cu vârfuri din Ω .

Este clar că trebuie precizat în ce anume constă pasul II:

- (i) Dacă p_i este situat în interiorul triunghiului, atunci unim p_i cu vârfurile triunghiului.
- (ii) Dacă p_i este situat pe o latură, atunci unim p_i cu vârfurile triunghiurilor ce au drept latură comună latura respectivă.
- (iii) Legalizăm muchiile ilegale pentru triangularea $\mathcal{T}(\Omega \cup \{p_i\})$.

Prezentăm, în continuare, algoritmul de triangulare Delaunay:

Se dă: $S = \{p_1, p_2, \dots, p_n\}$.

Se cere: o triangulare Delaunay a lui S .

```

(1) fie  $p_{-1}, p_{-2}, p_{-3}$  astfel încât  $S$  este conținută în  $\Delta p_{-1}p_{-2}p_{-3}$ 
(2)  $\mathcal{T} = \Delta p_{-1}p_{-2}p_{-3}$ 
(3) pentru  $i = 1, n$  execută
    (4)     // inserăză  $p_i$  în  $\mathcal{T}$ 
    (5)     găsește un triunghi  $p_i p_j p_k \in \mathcal{T}$  care conține  $p_i$ 
    (6)     dacă  $p_i \in \text{int } \Delta p_i p_j p_k$ 
    (7)         atunci
            adaugă muchiile din  $p_i$  la vârfurile  $p_j, p_k$ 
            //  $p_i p_j p_k$  este împărțit în trei triunghiuri
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
    (8)         altfel //  $p_i$  este pe o latură, de exemplu  $p_i p_j$ 
            adaugă muchiile  $p_i p_k$  și  $p_i p_j$ 
            // aici  $p_i$  este vârful celui de-al doilea triunghi adjacent lui  $p_i p_j$ 
            // cele două triunghiuri se înlocuiesc cu patru triunghiuri
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
    (13)     altfel //  $p_i$  este pe o latură, de exemplu  $p_i p_j$ 
            adaugă muchiile  $p_i p_k$  și  $p_i p_j$ 
            // aici  $p_i$  este vârful celui de-al doilea triunghi adjacent lui  $p_i p_j$ 
            // cele două triunghiuri se înlocuiesc cu patru triunghiuri
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
    (14)     altfel //  $p_i$  este pe o latură, de exemplu  $p_i p_j$ 
            adaugă muchiile  $p_i p_k$  și  $p_i p_j$ 
            // aici  $p_i$  este vârful celui de-al doilea triunghi adjacent lui  $p_i p_j$ 
            // cele două triunghiuri se înlocuiesc cu patru triunghiuri
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
    (15)     altfel //  $p_i$  este pe o latură, de exemplu  $p_i p_j$ 
            adaugă muchiile  $p_i p_k$  și  $p_i p_j$ 
            // aici  $p_i$  este vârful celui de-al doilea triunghi adjacent lui  $p_i p_j$ 
            // cele două triunghiuri se înlocuiesc cu patru triunghiuri
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
    (16)     altfel //  $p_i$  este pe o latură, de exemplu  $p_i p_j$ 
            adaugă muchiile  $p_i p_k$  și  $p_i p_j$ 
            // aici  $p_i$  este vârful celui de-al doilea triunghi adjacent lui  $p_i p_j$ 
            // cele două triunghiuri se înlocuiesc cu patru triunghiuri
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
    (17)     altfel //  $p_i$  este pe o latură, de exemplu  $p_i p_j$ 
            adaugă muchiile  $p_i p_k$  și  $p_i p_j$ 
            // aici  $p_i$  este vârful celui de-al doilea triunghi adjacent lui  $p_i p_j$ 
            // cele două triunghiuri se înlocuiesc cu patru triunghiuri
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
    (18)     altfel //  $p_i$  este pe o latură, de exemplu  $p_i p_j$ 
            adaugă muchiile  $p_i p_k$  și  $p_i p_j$ 
            // aici  $p_i$  este vârful celui de-al doilea triunghi adjacent lui  $p_i p_j$ 
            // cele două triunghiuri se înlocuiesc cu patru triunghiuri
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
    (19)     altfel //  $p_i$  este pe o latură, de exemplu  $p_i p_j$ 
            adaugă muchiile  $p_i p_k$  și  $p_i p_j$ 
            // aici  $p_i$  este vârful celui de-al doilea triunghi adjacent lui  $p_i p_j$ 
            // cele două triunghiuri se înlocuiesc cu patru triunghiuri
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
    (20)     altfel //  $p_i$  este pe o latură, de exemplu  $p_i p_j$ 
            adaugă muchiile  $p_i p_k$  și  $p_i p_j$ 
            // aici  $p_i$  este vârful celui de-al doilea triunghi adjacent lui  $p_i p_j$ 
            // cele două triunghiuri se înlocuiesc cu patru triunghiuri
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
             $Le(p_i, p_j, p_k, \mathcal{T})$ 
             $Le(p_i, p_k, p_j, \mathcal{T})$ 
    (21) elimină  $p_{-1}, p_{-2}, p_{-3}$  și muchiile corespunzătoare.

```

Redăm în continuare modulul $Le(p_i, p_j, p_k, \mathcal{T})$ utilizat în algoritmul de mai sus, unde p_i este punctul inserat, iar $p_i p_j$ este muchia care trebuie eliminată:

$Le(p_i, p_j, \mathcal{T})$:

```

(1) dacă  $p_i p_j$  este ilegală
(2)     atunci
(3)         fie  $\Delta p_i p_j p_k$  adjacent  $\Delta p_r p_i p_j$  de-a lungul lui  $p_i p_j$ 
(4)         înlocuiește  $p_i p_j$  cu  $p_r p_k$ 
(5)          $Le(p_r, p_i, p_k, \mathcal{T})$ 
(6)          $Le(p_r, p_k, p_j, \mathcal{T})$ .

```

Observația 7.3.1. Algoritmul de triangulare Delaunay are complexitatea $\mathcal{O}(n \lg n)$.

Bibliografie

- [1] Avis, David, Toussaint, Godfried, *An efficient algorithm for decomposing a polygon into star-shaped pieces*, Pattern Recognition 13 (1981), 295–298
- [2] de Berg, Mark; Cheong, Ottfried; van Kreveld, Marc; Overmars, Mark, *Computational Geometry. Algorithms and Applications*, Springer, 1st ed. 1997, 2nd ed. 2000, 3rd ed. 2008
- [3] Blaga, Paul, *Geometrie Computatională și Combinatorică. Introducere*, Note de curs, Cluj-Napoca, 2006
- [4] Burdescu, Dimitrie Dan, *Analiza Complexității Algoritmilor*, Ed. Albastră, 1998
- [5] Chazelle, Bernard; Guibas, Leonidas, *Visibility and intersection problems in plane geometry*, Discrete Comput. Geom. 4 (1989), 551–581
- [6] Chazelle, Bernard; Dobkin, David, *Intersection of convex objects in two and three dimensions*, Journal ACM 34 (1987), 1–27
- [7] Chazelle, Bernard; Dobkin, David, *Optimal convex decompositions*, Computational Geometry (G.T. Toussaint ed), North-Holland (1985), 63–133
- [8] Chazelle, Bernard, *Computational Geometry and Convexity*, PhD Thesis, Yale University, 1980
- [9] Chazelle, Bernard, *An optimal convex hull algorithm in any fixed dimension*, Discrete Comput. Geom. 10 (1993), 377–409
- [10] Chazelle, Bernard; Edelsbrunner, Herbert, *An optimal algorithm for intersecting line segments in the plane*, Journal ACM 39 (1992), 1–54
- [11] Chazelle, Bernard, *Triangulating a simple polygon in linear time*, Discrete Comput. Geom. 6 (1991), 485–524
- [12] Chazelle, Bernard; Devillers, Olivier; Hurtado, Ferran; Mora, Merce; Sacristan, Vera; Teillaud, Monique, *Splitting a Delaunay triangulation in linear time*, Algorithmica 34 (2002), 39–46

- [13] Chazelle, Bernard, *Computational geometry: a retrospective*, Computing in Euclidean Geometry 2nd ed. (D.-Z. Du and F. Hwang, eds), World Scientific Press (1995), 22–46
- [14] Chazelle, Bernard, *Tools from Computational Geometry*, <http://www.cs.princeton.edu/~chazelle/pubs.html>
- [15] Chazelle, Bernard, *Computational geometry for the gourmet. Old fare and new dishes*, Lect. Notes Comput. Sci. 510 (1991), 686–696
- [16] Chazelle, Bernard, ; Goodman, Jacob; Pollack, Richard, *Advances in discrete and computational geometry*, Contemporary Mathematics, 223, AMS, Providence, 1998
- [17] Chen, Jianer, *Computational Geometry: Methods and Applications*, Texas A.M. University, 1996, <http://faculty.cs.tamu.edu/chen/>
- [18] Chvatal, Vaclav, *A combinatorial theorem in plane geometry*, Journal of Combinatorial Theory B 18 (1975), 39-41
- [19] Demaine, Erik; Mitchell, Joseph; O'Rourke, Joseph, *The Open Problems Project*, <http://maven.smith.edu/~orourke/TOPP/>
- [20] Edelsbrunner, Herbert, *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987
- [21] Edelsbrunner, Herbert, *Triangulations and meshes in computational geometry*, Acta Numerica 9 (2000), 133-213
- [22] Eppstein, David, *Graph-theoretic solutions to computational geometry problems*, Lecture Notes in Computer Science 5911 (2010), 1-16
- [23] Estivill-Castro, Vladimir, *An Introduction to Computational Geometry by the Art Gallery Theorems*, (Spanish) Aportaciones Matemáticas. Notas de Investigación 10, 1994
- [24] Fisk, Steve, *A short proof of Chvatal's watchman theorem*, Journal of Combinatorial Theory B 24 (1978), 374
- [25] Gavrilova, Marina (ed.), *Generalized Voronoi Diagram. A Geometry-Based Approach to Computational Intelligence*, Studies in Computational Intelligence 158, Springer, 2008
- [26] Goodman, Jacob; Pach, Jnos; Pollack, Richard (eds), *Twentieth Anniversary Volume: Discrete and Computational Geometry*, Springer, Reprinted from Discrete & Computational Geometry 39 No. 1–3, 2008

- [27] Goodman, Jacob; O'Rourke, Joseph (eds), *Handbook of Discrete and Computational Geometry*, Series on Discrete Mathematics and its Applications, CRC Press, 1997
- [28] Goodman, Jacob; O'Rourke, Joseph (eds), *Handbook of Discrete and Computational Geometry*, 2nd ed., Discrete Mathematics and its Applications, Chapman & Hall/CRC, 2004
- [29] Graham, Ronald, *An efficient algorithm for determining the convex hull of a finite planar set*, Information Processing Letters 1 (1972), 132–133
- [30] Hales, Thomas C., *Computational discrete geometry*, Springer Lecture Notes in Computer Science 6327 (2010), 1–3
- [31] Homotcovschi, Laurențiu, *Geometrie Computatională*, Colecția Biblioteca de Matematică, Constanța, 2002
- [32] Homotcovschi, Laurențiu; Nicolescu, Bogdan, *The construction of a Voronoi polygon*, An. St. Univ. Ovidius Constanța, Ser. Mat. 7, No. 2 (1999), 23-34
- [33] Honsberger, Ross, *Mathematical gems II*, Dolciani Mathematical Expositions 2, Mathematical Association of America IX (1976), 104–110
- [34] Jarvis, Ray, *On the identification of the convex hull of a finite set of points in the plane*, Information Processing Letters 2 (1973), 18-21
- [35] Laszlo, Michael, *Computational Geometry and Computer Graphics in C++*, Prentice Hall, 1996
- [36] Mount, David, *Computational Geometry*, CMSC 754, Lecture Notes, University of Maryland, 2002
- [37] Munteanu, Marian; Nistor, Ana Irina, *Algoritmi de Triangulare*, Casa Editoriala Demiurg, Iași, 2008
- [38] Pișcoran, Laurian, *Elemente de Geometrie Computatională*, Ed. Risoprint, Cluj-Napoca, 2008
- [39] Preparata, Franco; Shamos, Michael Ian, *Computational Geometry. An Introduction*, Springer-Verlag, 1985
- [40] O'Rourke, Joseph, *Computational Geometry in C*, Cambridge Univ. Press, 1st ed. 1994, 2nd ed. 1998
- [41] O'Rourke, Joseph, *Computational geometry column 48*, Int. J. Comput. Geom. Appl. 17, No. 4 (2007), 397–399

- [42] O'Rourke, Joseph, *Art Gallery Theorems and Algorithms*, Oxford University Press, 1987, <http://maven.smith.edu/~orourke/books/> ArtGalleryTheorems/art.html
- [43] O'Rourke, Joseph; Streinu, Ileana, *The vertex-edge visibility graph of a polygon*, Comput. Geom. 10, No.2 (1998), 105–120
- [44] Shamos, Michael Ian, *The early years of computational geometry – a personal memoir*, American Mathematical Society. Contemp. Math. 223 (1999), 313–333
- [45] Toussaint, Godfried, *Computational geometry. A survey and editorial of the special issue*, Pattern Recognit. Lett. 14, No.9 (1993), 697–702
- [46] Urrutia, Jorge, *Open Problems in Computational Geometry*, Springer. Lect Notes Comput. Sci. 2286 (2002), 4–11
- [47] Vlada, Marin, *Abordarea modernă a conceptului de algoritm*, Conferința Națională de Învățământ Virtual, ediția a II-a, 2004
- [48] Wang, Ren-Hong, *What is computational geometry*, AMS/IP Stud. Adv. Math. 20 (2001), 491–495
- [49] Zaharie, Daniela, *Algoritmica*, <http://www.info.uvt.ro/~dzaharie>

Cuprins

Introducere	5
1 Preliminarii	7
1.1 Complexitatea algoritmilor	7
1.2 O problemă de intersecție	8
1.3 Preliminarii geometrice	11
1.4 Grafuri geometrice în plan	12
2 Învelitori convexe în plan	15
2.1 Învelitoarea convexă a unei mulțimi finite de puncte din plan	15
2.2 Determinarea învelitorii convexe a unei mulțimi finite din plan	17
3 Triangularea unui poligon	23
3.1 Problema galeriei de artă	23
3.2 Existența unei triangulări	28
3.3 Proprietățile triangulării	31
3.4 Aria unui poligon	32
3.5 Triangularea poligoanelor simple prin otectomie	36
4 Partiționarea unui poligon în poligoane monotone	39
4.1 Poligoane monotone	39
4.2 Triangularea unui poligon monoton	42
4.3 Partiționarea poligoanelor simple în poligoane monotone	47
4.4 Partiționarea convexă	50
5 Probleme de intersecție	53
5.1 Intersecții de semiplane	53
5.2 Lista dublu conectată a muchiilor	55
5.3 Suprapunerea a două subdiviziuni ale planului	57
5.4 Aplicații ale algoritmului MAPOVERLAY	61
6 Diagrama Voronoi	65
6.1 Definiție. Proprietăți fundamentale.	65
6.2 Construcția diagramei Voronoi	72

7 Triangularea Delaunay	79
7.1 Triangularea unei mulțimi finite de puncte din plan	79
7.2 Triangularea Delaunay	83
7.3 Un algoritm de triangulare Delaunay	85
Bibliografie	87

Table of Contents

Introduction	5
1 Preliminaries	7
1.1 Complexity of Algorithms	7
1.2 An Intersection Problem	8
1.3 Geometric Preliminaries	11
1.4 Geometric Graphs into Plane	12
2 2D Convex Hulls	15
2.1 Convex Hull of a 2D Finite Set of Points	15
2.2 Determining the Convex Hull of a 2D Finite Set of Points	17
3 Polygon Triangulation	23
3.1 Art Gallery Problem	23
3.2 The Existence of a Triangulation	28
3.3 Triangulation Properties	31
3.4 Polygon Area	32
3.5 Triangulation of Simple Polygons by Ear Clipping	36
4 Polygon Partitioning into Monotone Polygons	39
4.1 Monotone Polygons	39
4.2 Triangulation of a Monotone Polygon	42
4.3 Simple Polygon Partitioning into Monotone Polygons	47
4.4 Convex Partitioning	50
5 Intersection Problems	53
5.1 Half-plane Intersection	53
5.2 Double Connected Edge List	55
5.3 Overlap of Two Subdivisions of the Plane	57
5.4 Applications of the Algorithm MAPOVERLAY	61
6 Voronoi Diagram	65
6.1 Definitions and Properties	65
6.2 Voronoi Diagram Construction	72

7 Delaunay Triangulation	79
7.1 Triangulation of a 2D Finite Set of Points	79
7.2 Delaunay Triangulation	83
7.3 Delaunay Triangulation Algorithm	85
Bibliography	87