"Sunt măr de lângă drum și fără gard"

Înfășurătoarea CONVEXĂ în 2D

Ion Cozac

Determinarea înfășurătorii convexe a unei mulțimi de puncte este o problemă elementară în geometria computațională. Determinarea înfășurătorii convexe este un prim pas care apare în foarte mulți algoritmi geometrici și aceasta deoarece surprinde o idee aproximativă despre forma sau proporțiile unei mulțimi de date.

Problemă: Într-o livadă avem *n* meri și dorim să construim un gard, care înconjoară livada și are perimetrul minim. Care sunt merii situați lângă gard?

Vom prezenta două exemple intuitive care ne ajută să înțelegem următoarea expunere teoretică. Înfășurătoarea convexă a unei mulțimi de puncte în plan este conturul închis de o bandă de cauciuc întinsă în jurul cuielor bătute în fiecare din punctele date. Frontiera înfășurătorii convexe în spațiul tridimensional este forma pe care o ia un ambalaj plastic strâns întins în jurul punctelor.

Definiția 1: O mulțime *S* este *convexă* dacă și numai dacă $x \in S$ și $y \in S$ implică $xy \subseteq S$; prin *xy* am notat segmentul închis care unește punctele *x* și *y*.

Segmentul *xy* este mulțimea tuturor punctelor de forma $\alpha x+\beta y$, cu $\alpha,\beta \geq 0$ și $\alpha+\beta=1$; pentru cazul bidimensional $x=(x_1,x_2)$ și $y=(y_1,y_2)$; $x+y=(x_1+y_1,x_2+y_2)$.

Definiția 2: Înfășurătoarea convexă a unei mulțimi de puncte *S* este intersecția tuturor mulțimilor convexe care includ mulțimea *S*, notată *conv*(*S*).

Proprietăți:

conv(*S*) - în plan este poligonul de arie minimă care include mulțimea *S*;

conv(*S*) - în spațiu este poliedrul de volum minim care include mulțimea *S*.

Înfășurătoarea convexă a unei mulțimi de puncte din plan este descrisă prin frontiera poligonului convex care are proprietățile menționate mai sus. Acesta va fi descris prin lista vârfurilor acestuia, luate în sens trigonometric.

Determinarea înfășurătorii convexe

Fiind date *n* puncte în plan prin coordonatele (x_i, y_i), $i=1,...,n$, se cere să se determine vârfurile poligonului care descrie înfășurătoarea convexă. Algoritmul prezentat în continuare este preluat din [**Joseph O'Rourke** - *Computational Geometry in C*] cu câteva modificări menite să simplifice implementarea, și în același timp să se obțină un cod mai rapid. Acesta a fost elaborat de **R. L. Graham** în 1972.

În prima etapă se alege un punct care are ordonata *y* minimă; dacă există mai multe astfel de puncte, se alege cel care are abscisa *x* minimă. Originea sistemului de coordonate se translatează în acest punct și noua poziție se plasează (prin interschimbare) pe prima poziție în șir. Următoarele puncte din șir se ordonează după regula următoare:

$P_i(x_i,y_i)<P_j(x_j,y_j)$ dacă și numai dacă $\varphi_i<\varphi_j$ sau $\varphi_i=\varphi_j$ și $r_i<r_j$, unde (r,φ) reprezintă coordonatele polare ale punctului $P(x,y)$. Acestea se determină astfel:

$\varphi = arctan(y/x)$; $r = \sqrt{x^2+y^2}$ (figura 1).

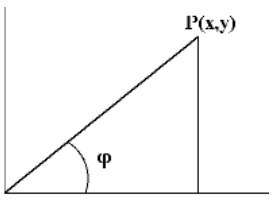

Figura 1. Coordonate polare

Pentru implementare nu ne putem permite să folosim aceste formule care sunt foarte costisitoare (ca timp) și introduc erori de rotunjire. Folosim în schimb următoarea proprietate: $\varphi_i < \varphi_j$ dacă și numai dacă $(x_i y_j - x_j y_i) > 0$.

Să justificăm această afirmație. În figura 2 am reprezentat triunghiul format de vârfurile O, P_i, P_j a cărui arie este jumătate din valoarea absolută a determinantului:

$$\begin{vmatrix} 1 & 0 & 0 \\ 1 & x_i & y_i \\ 1 & x_j & y_j \end{vmatrix}$$

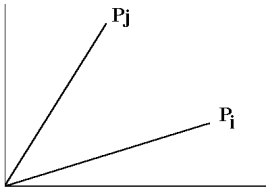

Figura 2. Cum determinăm dacă $P_i < P_i$?

Aria triunghiului este pozitivă dacă și numai dacă punctele O, P_i, P_j apar în sens trigonometric.

În caz de egalitate a ariilor, pentru economie de timp se compară direct valorile $x_i^2 + y_i^2$ și $x_j^2 + y_j^2$.

În a doua etapă se parcurg în ordine punctele și se determină care dintre ele se vor situa pe frontiera poligonului convex căutat. Observăm că originea sistemului de coordonate se va situa sigur pe frontiera căutată; acesta va fi introdus în lista de ieșire. În continuare, pentru fiecare punct P_k se verifică dacă este îndeplinită următoarea condiție: unghiul format de ultimele trei puncte din lista de ieșire (P_{k-2}, P_{k-1}, P_k) să aibă măsura mai mică decât π. Dacă această condiție este îndeplinită, P_k poate rămâne (deocamdată!) în lista de ieșire, în caz contrar un unghi cu măsura mai mare decât π introduce o concavitate și în consecință punctul P_k trebuie scos din lista de ieșire. În figura 3 este ilustrată această situație.

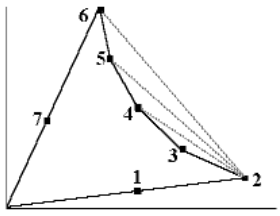

Figura 3. Construirea înfășurătorii convexe

Evoluția listei de ieșire (în cazul figurii 3) este următoarea:

L: {0}
L: {0,1}
L: {0,1,2}: {0,2}
L: {0,2,3}
L: {0,2,3,4}: {0,2,4}
L: {0,2,4,5}: {0,2,5}
L: {0,2,5,6}: {0,2,6}
L: {0,2,6,7}
L: {0,2,6,7,0}: {0,2,6,0}

În unele situații este nevoie de mai multe reveniri, atunci când depistăm un unghi care are măsura mai mare decât π, așa cum se vede în figura 4.

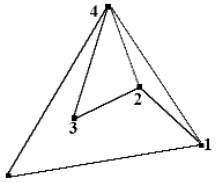

Figura 4. O evoluție sinuoasă în construirea înfășurătorii convexe

Cum determinăm dacă măsura unghiului (P_{k-2}, P_{k-1}, P_k) este mai mică decât π? Această condiție este îndeplinită dacă și numai dacă cele trei puncte apar în sens trigonometric: valoarea determinantului (aria triunghiului format cu vârfurile respective) este pozitivă:

$$\begin{vmatrix} 1 & x_{k-2} & y_{k-2} \\ 1 & x_{k-1} & y_{k-1} \\ 1 & x_k & y_k \end{vmatrix} > 0$$

În final coordonatele din lista de ieșire se corectează (prin translatarea originii sistemului de coordonate) pentru a coincide cu cele date la intrare.

Să studiem complexitatea acestui algoritm. În prima parte am parcurs șirul pentru a determina noua origine a sistemului de coordonate, apoi am ordonat punctele în raport cu coordonatele polare ale acestora. Rezultă un ordin de complexitate $O(n \log n)$, deoarece $O(n)$ este absorbit de ordinul maxim dat de algoritmul de sortare.

În partea a doua am parcurs din nou șirul sortat pentru a insera în lista de ieșire punctele care îndeplinesc condițiile impuse. Această parcurgere necesită între n și $2n$ pași, de unde rezultă că algoritmul lui Graham are ordinul de complexitate $O(n \log n)$.

O limită inferioară a complexității

Ne punem următoarea întrebare: se poate proiecta un algoritm care să determine înfășurătoarea convexă cu un ordin de complexitate mai mic (sub $O(n \log n)$)? Să privim figura 4. Fiind date n valori reale $x_1, ..., x_n$, se reprezintă punctele $P_i(x_i, x_i^2)$ aflate pe parabola

$y = x^2$. Dacă se determină înfășurătoarea convexă a mulțimii de puncte P_1, P_2, ..., P_n, acest algoritm poate fi folosit pentru a sorta un șir de numere date. Pe de altă parte, știm [**Donald Knuth** - *Algoritmi de sortare și căutare*] că un algoritm de sortare optim are, în cel mai defavorabil caz, ordinul $O(n \log n)$. În concluzie, nu se poate proiecta un algoritm care să determine înfășurătoarea convexă cu un ordin de complexitate inferior. Am obținut astfel următoarea

Teoremă
Orice algoritm care determină înfășurătoarea convexă are un ordin de complexitate $O(n \log n)$.

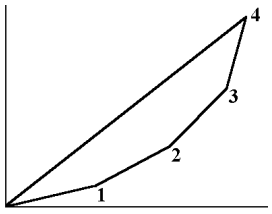

Figura 5. Cum se poate sorta un șir de numere cu ajutorul înfășurătorii convexe

Aplicație: diametrul unei mulțimi de puncte

Fiind dată o mulțime de puncte în plan, diametrul acestei mulțimi este distanța maximă dintre două puncte ale acestei mulțimi.

Pentru a elabora un algoritm performant să observăm că diametrul unei mulțimi de puncte unește întotdeauna două puncte care sunt vârfuri ale înfășurătorii convexe. Această observație poate simplifica în unele cazuri munca de căutare, de aceea într-o primă etapă trebuie să determinăm înfășurătoarea convexă. Un algoritm naiv ar compara două câte două punctele aflate pe înfășurătoare, efectuând astfel un număr de $n(n-1)/2$ comparații având complexitate pătratică.

Un algoritm performant care determină diametrul unui poligon convex este prezentat în [**Franco Preparata, Michael Shamos** - *Computational Geometry*]. Acesta se bazează pe următorul rezultat:

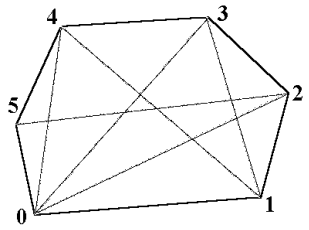

Figura 6. Determinarea diametrului prin examinarea perechilor de puncte antipodale

Diametrul unui poligon convex este distanța maximă dintre două puncte antipodale. Spunem că o pereche de puncte este *antipodală* dacă prin cele două puncte se pot duce două drepte paralele care încadrează complet poligonul convex (vezi figura 6).Prezentăm în continuare un algoritm care determină perechile de puncte antipodale; considerăm că poligonul este dat prin lista vârfurilor în sens trigonometric, punctele fiind numerotate P_0, P_1, ..., P_{n-1}.

```
subprogram perechi_antipodale;
p:=n-1;
q:=next(p);
cât timp (Aria(p,next(p),next(q))>
                        Aria(p,next(p),q))
  q:=next(q);
sfârsit cât timp;
q0:=q;
cât timp (q0)
  p:=next(p);
  scrie(p,q);
  cât timp (Aria(p,next(p),next(q))>
                        Aria(p,next(p),q))
    q:=next(q);
    dacă ((p,q)≠(q0,0)) scrie(p,q);
  sfârsit cât timp;
  dacă (Aria(p,next(p),next(q))=
                        Aria(p,next(p),next(q)))
    dacă ((p,q)≠(q0,n-1)) scrie(p,next(q));
sfârsit cât timp;
sfârsit subprogram;
```

Observații:
- într-o primă etapă se parcurg o parte din vârfurile poligonului până când se întâlnește vârful cel mai îndepărtat de segmentul (`p,next(p)`);
- am folosit funcția `next(k)` pentru a putea parcurge în ordine ciclică vârfurile poligonului; aceasta se implementează astfel: `next:=(k+1) mod n`;
- funcția `Aria(i,j,k)` calculează dublul ariei triunghiului format de vârfurile P_i, P_j, P_k; această valoare este întotdeauna pozitivă deoarece vârfurile sunt date în sens trigonometric;
- în etapa a doua se determină perechile antipodale; este tratat și cazul în care două segmente ale poligonului sunt paralele.

Observăm că acest algoritm ne furnizează perechile de puncte antipodale în timp liniar, deci complexitatea unui algoritm care determină diametrul unui poligon convex are ordinul de complexitate $O(n)$.

Ion Cozac este lector la Universitatea "Petru Maior" din Târgu-Mureș și poate fi contactat prin e-mail la adresa cozac@uttgm.ro