Geometrie Computațională

Bibliografie

(este și pe Moodle)

1. Computational Geometry - de Berg
2. Computational Geometry - A Survey

Proiectele

Vor fi prezentate în săptămâna 8; vezi lista pe Moodle

Ordonarea punctelor - Context 1D

Fie că suntem în R, fie că suntem în R^n, dacă avem 3 puncte coliniare putem determina dacă un punct P este între A și B.
Putem lucra cu distanțe sau cu vectori. Vectorii sunt mai buni, mai eficienți (nu folosim radicali).

Punctele A, P, B sunt coliniare(A != B, P != B) ⇔ exista un R astfel încât AP = rPB, unde r se numește raportul punctelor A, P, B
r (A, P, B) ⇔ AP = r PB
Mijlocul segmentului: r (A, M, B) = 1, deoarece AM = 1 MB

Noțiuni de geometrie afină

- Combinație liniară: vectori
- Combinații afine și convexe: puncte

Când lucrez cu puncte, combin puncte (fac combinații afine). mA + nB = un nou punct, trebuie ca m + n = 1.
În R^d:
Fie v1, v2, ..., vp vectori, o combinație liniară este a1 v1 + a2 v2 + ... + ap vp, cu coeficienți reali.
Fie A1, A2, ..., Ap puncte, o combinație afină (aka baricentrică) este l1 A1 + l2 A2 + ... + lp Ap unde **suma coeficienților este 1**.
O combinație convexă este un punct de forma l1 A1 + l2 A2 + ... + lp Ap unde l1, l2, ... ln **sunt între 0 și 1** și **au suma 1**.

Exemplu

Pentru două puncte o combinație afină este o dreaptă, iar combinația convexă este segmentul. Pentru trei puncte combinația afină este un plan, combinația convexă este un triunghi cu interiorul lui.

Combinație liniară: l A + m B = (1 - alpha) A + alpha B.
Interpretare: avem un balansoar, și avem greutăți puse pe A și pe B, de masă m_A și m_B.
Centrul de greutate este $m_A / (m_A + m_B) * A + m_B / (m_A + m_B) * B$.

Exercițiu

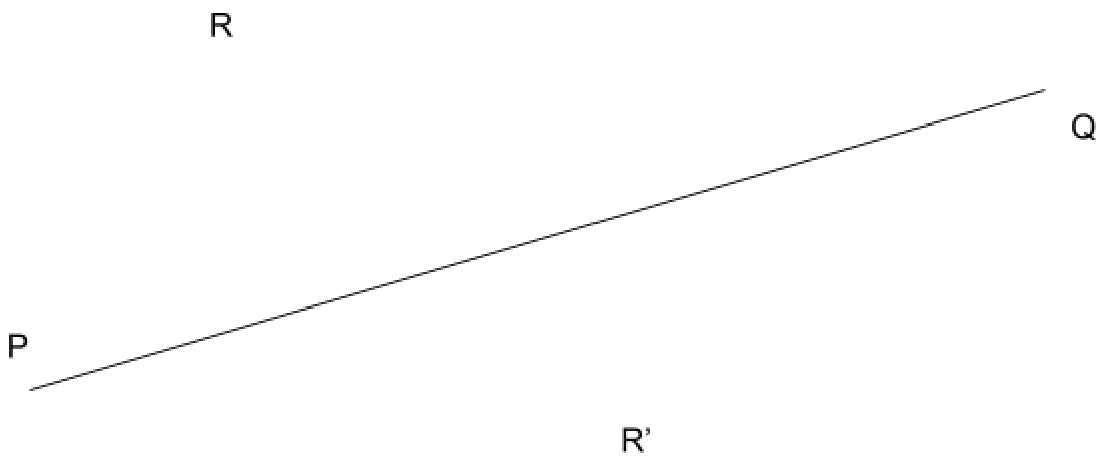
Cum aflăm combinația baricentrică plecând de la raport?
Știm r (A, C, B) = -½
AC = -½ CB
=> 1 CA - ½ CB = 0 = CC (ok, dar nu e combinație validă)
=> 2 CA - CB = 0 = CC (acum e combinație afină)
C = 2A - B

Concluzie: dacă avem punctele A, P, B coliniare și distincte, le putem caracteriza poziția lor relativă prin două căi:

- raportul r (A, P, B)
- combinații afine: P = (1 - alpha) * A + alpha * B

Context 2D

Vrem o cantitate numerică care să exprime faptul că un punct se află „la stânga” sau „la dreapta” unei drepte.



Produs vectorial (cross product)

Definiție:

- **Geometric:** un vector cu o serie de proprietăți
  - perpendicular pe v și pe w
  - care are sensul dat de regula șurubului (a burghiului) drept
  - are lungimea dată de o anumită formulă (aria paralelogramului determinat de v și w)
- **Numeric (în R^3):** știm că v = (v1, v2, v3); w = (w1, w2, w3); v, w în R^3
  v x w = ?

Se calculează în modul următor:

$$\mathbf{a} \times \mathbf{b} \equiv \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix}.$$

! Deosebire între produsul scalar (dot product) și produsul vectorial (cross product).
Produsul scalar este: <v, w> = v1 * w1 + v2 * w2 + v3 * w3

Exemplu: Fie v = (4, 1, 0); w = (0, 2, 1)
<v, w> = 2
v x w =

$$\begin{vmatrix} 4 & 0 & e1 \\ 1 & 2 & e2 \\ 0 & 1 & e3 \end{vmatrix}$$

= e1 - 4 e2 + 8 e3 = (1, -4, 8)

Observație: cum se calculează produsul vectorial și ce rezultat se obține pentru doi vectori “orizontali” (ultima componentă zero)?

v x w = (0, 0, v1 w2 - v2 w1)

Intuitiv: perpendicular pe doi “orizontali” e un “vertical”

Lemă: Fie P, Q, R, 3 puncte din planul orizontal.
PQ x PR = (q1 - p1, q2 - p2, 0) x (r1 - p1, r2 - p2, 0)

$$= \left(0, 0, \begin{vmatrix} 1 & 1 & 1 \\ p1 & q1 & r1 \\ p2 & q2 & r2 \end{vmatrix}\right)$$

= (0, 0, Det(P, Q, R))

Dacă R1 e la stânga lui PQ, R2 e la dreapta lui PQ:
PQ x PR1 = (0, 0, Det(P, Q, R1)), și Det (P, Q, R1) > 0
PQ x PR2 = (0, 0, Det(P, Q, R2)), și Det (P, Q, R2) < 0

Aplicații:
- dacă un punct e la stânga și la dreapta unei muchii orientate
- natura unui viraj în parcurgerea unei linii poligonale
- dacă un poligon e convex (viraje doar la stânga sau doar la dreapta) sau concav (viraje în ambele direcții)

Acoperiri convexe

Vezi curs

Algoritmi lenți (naivi)

Puncte extreme O(n^4)

Un punct este extrem dacă nu există niciun segment cu capetele în figură care să-l conțină. În cazul discului: toate punctele de pe cerc sunt extreme!

Un punct nu este extrem dacă este inclus într-un triunghi format de alte puncte.

Ordonarea punctelor după unghiul polar: nu este obligatoriu să calculăm unghiurile, putem să le ordonăm cu ajutorul testului de orientare.

Punct interior: calculăm centrul de greutate al punctelor de pe învelitoare. Sau doar centrul de greutate a 3 dintre puncte.

Muchii ale frontierei O(n^3)

Algoritmi care se folosesc de ordinea punctelor

Graham's Scan

Le luăm în ordine după unghiul polar, și adăugăm incremental puncte. Când în 3 puncte avem un viraj la dreapta, eliminăm punctul din mijloc.
Șmecherie: le sortăm lexicografic, după x și apoi după y

Graham Scan (varianta Andrew) - Algoritmul recursiv

Împărți figura în două părți, și determinăm frontiera inferioară și frontiera superioară.

Jarvis' March

Luăm un punct, căutăm în O(n) următorul punct de pe frontieră. O să fie complexitate O(h * n), unde h e numărul de puncte de pe frontieră.

Se poate mai eficient decât O(n log n)?

Nu, pentru că problema găsirii acoperiri convexe e echivalentă cu sortarea unor numere. Dacă avem x1, ... , xn și vrem să le sortăm, luăm parabola y = x^2, și avem punctele (x_i, x_i^2).

Alți algoritmi

QuickHull, Divide et impera, Chan's algorithm

Aplicație pentru acoperiri convexe

Diametrul unei mulțimi de puncte

Problemă: dată fiind o mulțime de n puncte din plan, notată cu M, să se determine x și y care sunt cele mai îndepărtate. Notăm distanța dintre aceste puncte diam(M).

- Abordare 1 (naivă): comparăm toate perechiile de puncte și luăm maximul O(n^2)
- Abordare 2: soluția eficientă

Proprietate: fie M o mulțime cu n elemente. Determinarea diam(M) necesită cel puțin n log n operații.

Proprietate: diametrul unei mulțimi finite de puncte este egal cu diametrul mulțimii date de vârfurile acoperirii convexe

Cu alte cuvinte (mai matematic):

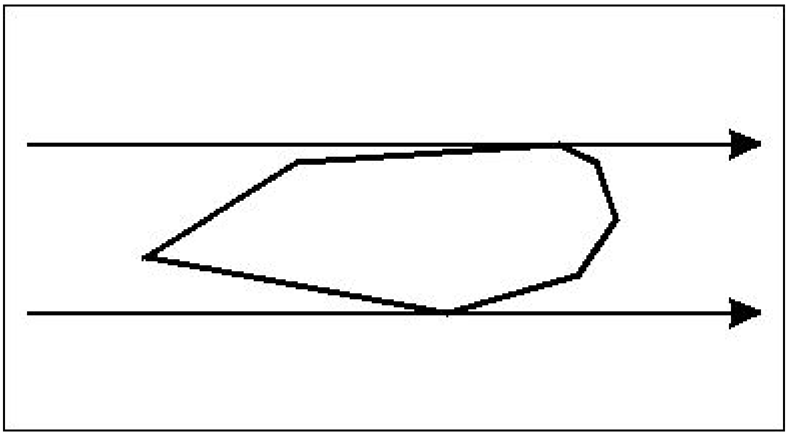
diam (M) = diam (Conv (M)) = diam(**v**),

unde **v** = mulțimea de vârfuri ale acoperirii convexe

Teoremă: diametrul unui poligon convex poate fi determinat în O(n) (și obținem inclusiv vârfurile cele mai îndepărtate)

Corolar: dacă avem o mulțime arbitrară de puncte găsim acoperirea în O(n log n) și apoi avem per total complexitatea soluției de O(n log n) + O(n) = O(n log n)

Ideea de demonstrație a teoremei



Drepte suport

Definim noțiunea de **dreaptă suport** pentru un poligon convex: trece printr-un vârf al poligonului și are toate celelalte vârfuri de aceeași parte a sa (în mod echivalent: cel mult încă un vârf e situat pe acea dreaptă).

Observație: dacă fixezi un vârf, există o infinitate de drepte suport care trec prin acest vârf.
Observație: dreapta suport este analoagă cu tangenta într-un punct al unui cerc C

Pereche de vârfuri antipodale

Două vârfuri distincte sunt **antipodale** dacă admit drepte suport paralele.

Observație: dacă p, q dau un diametru (au distanță maximă) atunci ele sunt antipodale. Reciproc nu e neapărat adevărat.

Ideea algoritmului de timp liniar pentru determinarea vârfurilor antipodale

Fixăm un vârf P al unui poligon convex.
Fie P_{i-1} și P_{i+1} predecesorul și succesorul când P este parcurs în sens trigonometric.

Notăm cu $q^{(i)}_L$= cel mai îndepărtat de dreapta $P_i P_{i+1}$ în sens orar (L de la left)
Notăm cu $q^{(i)}_R$= cel mai îndepărtat de dreapta $P_{i-1}P_i$ în sens antiorar/trigonometric (R de la right)

Toate vârfurile între $q^{(i)}_L$ și $q^{(i)}_R$ sunt antipodale

Diametrul unei mulțimi de puncte din R^2 este egal cu cea mai mare distanță dintre toate **dreptele suport**.

Triangularea poligoanelor

Descrierea problemei: **vezi slide-urile și cursul**
Observație: un poligon poate fi triangulat cu ajutorul diagonalelor

Problema galeriei de artă

Exemplu

Amplasarea camerelor: colorăm vârfurile cu 3 culori, astfel încât să nu fie două la fel adiacente.

3-colorare

Contraexemplu pentru 3-colorare: poligon care are „găuri” în interior (graful are cicluri)

Contraexemple

Uneori necesare: în unele cazuri sunt necesare multe camere
Întotdeauna suficiente: notăm cu n1, n2, n3 numărul de vârfuri colorate cu cele trei culori: n1 + n2 + n3 = n

Presupunem prin absurd că n1 > [n/3], n2 > [n/3], n3 > [n/3]. Atunci n1 > n/3, n2 > n/3, n3 > n/3. Dar atunci depășim n. Deci cel puțin una dintre n1, n2, n3

Algoritmi de triangulare

„Ear cutting” / „ear clipping” (vezi curs)

https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf
Fie P = (P1, ..., Pn) un poligon.

Terminologie

- vârfuri convexe / concave („reflexe”) ale unui poligon
  => se determină folosind viraje
  => cel mai la stânga vârf este sigur convex (cel mai mic lexicografic după x și y)
- vârf principial
  P_i este vârf principal dacă segmentul $[P_{i-1}, P_{i+1}]$ nu intersectează laturile triunghiului
  ⇔ Nu există alt vârf în interiorul sau pe laturile triunghiului format din (P_{i-1}, P_i, P_{i+1})

Descompunere în poligoane y-monotone

Conceptul de poligon y-monoton

Poligonul este y-monoton dacă (descrieri echivalente):

- poligonul poate fi parcurs de sus în jos în două moduri (pe două drumuri) fără întoarceri în sus
- orice dreaptă orizontală (perpendiculară pe Oy) intersectează poligonul și interiorul după o mulțime conexă (mulțimea vidă, un punct, sau un segment)

Triangularea unui poligon y-monoton

Algoritm bazat pe dreapta/linia de baliere (line sweep). În cazul nostru, dreapta este orizontală.

- Reținem **statutul** dreptei de baliere: adică avem o stivă a dreptelor deja întâlnite dar care mai au nevoie de diagonale / mai pot să apară în triunghiuri.
  Clarificare: când este eliminat un vârf? Când a fost trasată o diagonală situată mai jos de aceasta.
- **Evenimente**: modificarea statutului
  => vârfurile poligonului, în prealabil ordonate după y; pentru fiecare vârf știm dacă este pe lanțul din stânga sau pe cel din dreapta

Ce se întâmplă la evenimente?

- Cazul 1: vârful nou întâlnit este pe lanțul opus ultimului vârf din stivă => pentru elementele din stivă adaug noi diagonale / formez triunghiuri.
- Cazul 2: vârful nou întâlnit este pe același lanț cu celelalte vârfuri, și la dreapta jos față de ele
- Cazul 3: vârful nou întâlnit este pe același lanț și spre stânga

Input: Un poligon y-monoton $\mathcal{P}$.
Output: O triangulare a lui $\mathcal{P}$.

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur șir, ordonat descrescător, dupa y (dacă ordonata este egală, se folosește abscisa). Fie $v_1, v_2, \ldots, v_n$ șirul ordonat.
2. Inițializează o stivă vidă $\mathcal{S}$ și inserează v_1, v_2.
3. **for** $j = 3$ **to** $n - 1$
4.   **do** **if** v_j și vârful din top al lui $\mathcal{S}$ sunt în lanțuri diferite
5.   **then** extrage toate vârfurile din $\mathcal{S}$
6.   inserează diagonale de la v_j la vf. extrase, exceptând ultimul
7.   inserează v_{j-1} și v_j în $\mathcal{S}$
8.   **else** extrage un vârf din $\mathcal{S}$
9.   extrage celelalte vârfuri din $\mathcal{S}$ dacă diagonalele formate cu v_j sunt în interiorul lui $\mathcal{P}$; inserează aceste diagonale; inserează înapoi ultimul vârf extras
10.   inserează v_j în $\mathcal{S}$
11. adaugă diagonale de la v_n la vf. stivei (exceptând primul și ultimul)

Intersecții

a) cum se stabilește dacă două segmente se intersectează?
b) cum se determină punctul de intersecție dintre două segmente?

La ambele: care este complexitatea algebrică?

Input:
A = (x_A, y_A), B = (x_B, y_B);
C = (x_C, y_C), D = (x_D, y_D)

a) Testăm dacă se intersectează în felul următor: [AB] și [CD] neincluse în aceeași dreaptă se intersectează dacă și numai dacă A și B sunt de părți opuse ale lui [CD] și C și D sunt de părți opuse ale lui [AB]
Putem face asta cu testul de orientare, adică calculând un polinom de gradul II.

b) Scriem segmentele sub formă de combinații afine și rezolv sistemul de ecuații
[AB] intersectat cu [CD] = M dacă și numai dacă există lambda, mu astfel încât
M = (1 - lambda) * A + lambda * B = (1 - mu) * C + mu * D
În coordonate:
x_M = (1 - lambda) * x_A + lambda * x_B = (1 - mu) * x_C + mu * x_D
y_M = (1 - lambda) * y_A + lambda * y_B = (1 - mu) * y_C + mu * y_D
⇔ (x_B - x_A) * lambda + (x_D - x_C) * mu = x_C - x_A
(y_B - y_A) * lambda + (y_D - y_C) * mu = y_C - y_A
⇔ delta = | x_B - x_A x_D - x_C |
| y_B - y_A y_D - y_C | = polinom de gradul II

Complexitatea algebrică constă în a calcula două polinoame de gradul II. Pentru că apoi înlocuim în formula inițială, rămânem cu un polinom de gradul III pe un polinom de gradul II.

Intersecția segmentelor în R^1

Soluția are complexitatea O(n log n + k), unde n e numărul de puncte din input, și k este numărul de segmente care se intersectează.

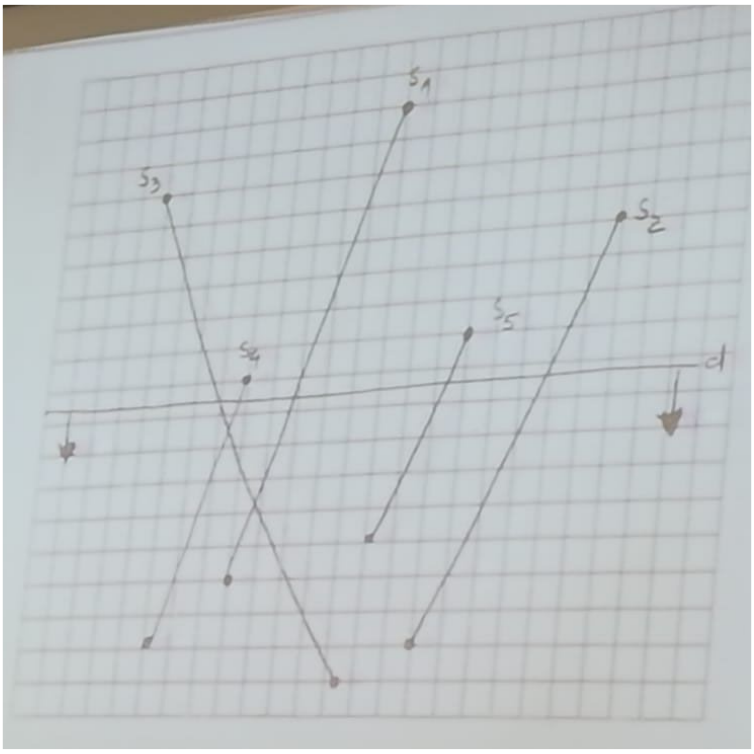
Marchezi care puncte sunt capete din stânga și care sunt capete din dreapta. Ordonezi punctele după coordonată, și le parcurgi de la stânga la dreapta. Când întâlnești un nou capăt de segment, actualizezi lista de segmente deschise și adaugi intersecții. Când se termină un segment îl scoți din listă.

Pentru mai multe explicații vezi cursul și pagina:
https://www.ti.inf.ethz.ch/ew/courses/CG/lecture/Chapter%205.pdf

Intersecția segmentelor în R^2

Folosim o dreaptă de baleiere orizontală. Vezi link-ul de mai sus pentru descrierea pe lung. Algoritmul e descris și în cartea Computational Geometry, și în slider-uri.

Exemplu cu construcția arborelui de căutare



Subdiviziune planară

Reținem liste dublu înlănțuite de vârfuri, fețe și muchii orientate

Suprapunerea starturilor tematice (overlay)

Conceptul cheie: **semi-muchie** (muchie-orientată), numită în engleză *half-edge*

Dat fiind un poligon (eventual cu goluri), acesta are:
- **frontieră exterioară**, care poate fi parcursă cu ajutorul semi-muchiilor a. î. *poligonul* să fie <u>la stânga</u> frontierei, iar *virajele convexe* să fie la stânga
- **frontieră interioară** (dacă există goluri), eventual cu mai multe goluri; poligonul este tot <u>la stânga</u>, dar *virajele convexe* sunt la dreapta

Listă dublu înlănțuită / Doubly Connected Edge List (DCEL)

Vârfuri:
v1 = (0, 4)
v2 = (4, 4)
v3 = (4, 0)
v4 = (0, -2)

Muchii: v1-v2, v2-v3, v1-v3 și v4-v3

Semi-muchii / muchii orientate:
Exterioare triunghi: (v1, v2), (v2, v3), (v3, v4), (v4, v3), (v3, v1)
Interioare triunghi: (v2, v1), (v1, v3), (v3, v2)

Față	Outer component (frontiera exterioară)	Inner component (frontiera interioară)
f1 (planul exterior)	null	(v3, v1)
f2 (triunghiul)	(v1, v3)	null

Semi-muchie	Origin	Twin	Next	Prev	Incident face
(v1, v2)	v1	(v2, v1)	(v2, v3)	(v3, v1)	f1
(v2, v3)	v2	(v3, v2)	(v3, v4)	(v1, v2)	f1
(v3, v4)	v3	(v4, v3)	(v4, v3)	(v2, v3)	f1
(v4, v3)	v4	(v3, v4)	(v3, v1)	(v3, v4)	f1

Explicați cum anume, folosind pointerii de mai sus:
- poate fi parcursă frontiera unei fețe (poligon) exterioară/interioară
- cum pot fi găsite toate semi-muchiile incidente cu un vârf

Overlay (suprapunerea straturilor tematice)

Să presupunem că avem un patrulater S_1 (4 vârfuri, 8 semi-muchii, 2 fețe) și realizez overlay-ul cu încă un triunghi S_2 (3 vârfuri, 6 semi-muchii, 2 fețe).

Overlay

Rezultatul are:
- 9 vârfuri (2 vârfuri detectate ca intersecții de segmente)
- 22 semi-muchii
- 4 fețe

Actualizarea listei de semi-muchii

Principiu: „fața delimitată de o muchie este la stânga acesteia”
Exemplu: prin vârful v al lui S_1 trece o muchie a lui S_2

O semi-muchie e se sparge în e’ și e’’

	Origin	Twin	Next	Prev
e’	v	vezi e tilda	Next(e)	Parcurgem toate muchiile incidente
e’’	Origin(e)	vezi e tilda	Parcurgem toate muchiile incidente	Prev(e)

Prev(e’) = cea mai apropriată muchie în sens trigonometric
Next(e’’) = cea mai apropriată în sens anti-trigonometric

Programare liniară

Pregătiri

Fie v, w doi vectori din R^3.
$K(v, w) = \arccos(\langle v, w \rangle / (||v|| * ||w||))$, care este în $[0, \pi]$
(Ne folosim de faptul că $\langle v, w \rangle / (||v|| * ||w||)$ este în $[-1, 1]$)

$\langle v, w \rangle = v_1 * w_1 + v_2 * w_2 + v_3 * w_3$
$||v|| = \sqrt{\langle v, v \rangle}$

Pentru a măsura unghiul dintre direcția dată și fețe este suficient să calculăm / să manevrăm unghiul dintre direcția dată și vectorii normali la fețele respective. Un vector normal la un plan este un vector perpendicular pe plan de normă 1).

Condiția ca matrița să blocheze sau să nu blocheze extragerea într-o direcție dată

Spunem că fața ^f1 a matriței, care corespunde feței f1 a piesei, blochează extragerea în direcția d
⇔ unghiul dintre normala lui f1 și d este mai mic decât 90 de grade
⇔ cos(d, normala lui f1) > 0

Această condiție trebuie verificată pentru <u>toate fețele</u>.

Detalierea (condiția scrisă în coordonate)

Fără a restrânge generalitatea, putem presupune că d = (d_x, d_y, 1) (de ce?).

Fie f o față fixată a obiectului, și avem n(f) normala la fața f.
Faptul că fața ^f nu blochează extragerea în direcția d

    ⇔ <normala lui f, d> <= 0 (*f)
    ⇔

Fixat f, și este căutată direcția d a. î. să fie verificată inegalitatea *f.

*f este o inecuație care descrie un semiplan

Exemple

1. Semiplane și intersecții
   -x + y + 1 <= 0
   -y - 3 <= 0
   2x + 3y - 5 <= 0
   Se obține un triunghi din intersecție
2. Legătura dintre normale, extragerea unui obiect și sisteme de inecuații
   (a) Obiectul nu poate fi extras în sus
       Normalele: (0, -1, 1), (0, 1, 1), (0, 1, 0), (0, 0, -1), (0, -1, 0)
       (0, -1, 1) -> 0 * x + (-1) * y + 1 <= 0 | y >= 1
       (0, 1, 1) -> 0 * x + 1 * y + 1 <= 0 | y <= -1
       (0, 1, 0) -> 0 * x + 1 * y + 0 <= 0 | y <= 0
       (0, 0, -1) -> 0 * x + 0 * y - 1 <= 0 | -1 <= 0
       (0, -1, 0) -> 0 * x - 1 * y + 0 <= 0 | y >= 0
   (b) Obiectul poate fi extras doar în sus

Intersecții de semiplane - Caracterizare „cantitativă”

Varianta 1

Alg. de tip divide et impera și overlay (vezi suport de curs)

Varianta 2

Terminologie:
- semiplan inferior (lower half plane)
- semiplan superior (upper half plane

Nu toate semiplanele sunt relevante pentru intersecție:
- Lower envelope: planurile care sunt relevante pentru semiplanele inferioare
- Upper envelope: planurile care sunt relevante pentru semiplanele superioare

Facem o separare asemănătoare cu Graham’s Scan

Transformare de dualitate

Plan primar / plan dual

Reguli

- unui punct p = (p_x, p_y) din planul primal i se asociază o dreaptă, notată p^*, în planul dual:
  p^* : (y = p_x * x - p_y)   == duala lui p
- unei drepte neverticale din planul primal d : (y = m_d x + n_d) i se asociază un punct din planul dual, notat d^*:
  d^* = (m_d, -n_d)

Invariante

- Păstrează relația de apartenență
  p aparține lui d ⇔ d^* aparține lui p^*

- Păstrează ordinea
  p este situat deasupra dreptei d (neverticală) ⇔ la fel și d^* deasupra lui p^*

Exemple

Fie p, q cu p != q. Fie r un punct situat dedesubtul dreptei d = pq. Să analizăm configurația duală.

Avem că r este dedesubtul lui d = pq
În planul dual: d^* este sub dreapta r*

=> p^* și q^* nu sunt relevante pentru intersecția planelor

Fie P o mulțime de puncte. Ce înseamnă că un segment [pq] cu p, q din P participă la frontiera superioară a lui Conv(P)?

Înseamnă că toate celelalte puncte sunt situate sub dreapta pq.

Dual: să considerăm dreptele p^* și q^* și punctul de intersecție d^* => punctul d^* este situat dedesubtul dreptelor corespunzătoare celorlalte puncte. Prin trecere la semiplane inferioare (adică ne uităm la semiplanele inferioare dreptelor duale):

dintre p^*_inf, q^*_inf, și r^*_inf „contează” doar p^*_inf și q^*_inf

Cu mai multe puncte:

dacă avem o dreaptă pq și multe puncte sub ea, dreptele p^* și q^* se vor intersecta, iar celelalte drepte vor fi deasupra semiplanului inferior determinat de intersecție (deci nu vor conta)

În primal:

Pentru partea superioară a frontierei convexe contează doar p și q

În dual:

Pentru intersecții de semiplane inferioare contează doar p^* și q^*

Concluzie

A determina frontiera superioară a acoperirii convexe pentru mulțimea de puncte P este echivalent cu a determina LE pentru semiplane inferioare determinate de dreptele duale.

(Analog: frontiera inferioară și UE)

Intersecții de semiplane - Aspecte calitative

Exemplu la caz general: problemă de programare liniară 1-dimensională (d = 1)
Coordonata: x_1 = x

cx = funcție obiectiv

maximizează (cx)

```
{
    a_1 x <= b_1
    ...
    a_n x <= b_n
}
```

Exemplu „concret”
maximizează (2x)
{
    3x <= 6 => x <= 2
    -2x <= 4 => x >= -2
    6x <= 6 => x <= 1
}

Intervale care trebuie intersectate:
    (-inf, 2], [-2, inf), (-inf, 1]

Intersecția: [-2, 1] se numește **regiune fezabilă**
Trebuie să maximizăm cx = 2x pe acest interval => maximul este x = 1, y = 2
Pentru d = 1, complexitatea este **liniară O(n)**.

Exemple de funcții obiectiv:
    c(x, y) = y
        => vrem să găsim cel mai sus punct din regiunea fezabilă

    c(x, y) = -y
        => vrem să găsim cel mai de jos punct din regiunea fezabilă

Algoritm incremental

- Adăugăm constrângerile incremental, una câte una

Se alege M mult mai mare decât 0 și se definesc noi constrângeri, astfel:
Fie c = (c_x, c_y):
    m_1 = {
            x <= M, dacă c_x > 0
            x >= -M, dacă c_x <= 0
        }
    m_2 = {
            y <= M, dacă c_y > 0
            y => -M, dacă c_y <= 0
        }

Exemplu
În continuare: c = (0, -1), așadar:
    m_1 : x >= -M
    m_2 : y >= -M

Fie (H, c) un program liniar cu constrângerile h_1, h_2, …, h_n

Se notează
    H_i = { m_1, m_2, h_1, ..., h_i }
        // ^-- mulțime de semiplane

$C_i = m_1$ intersectat cu m_2 intersectat cu h_1 … intersectat cu h_i
// ^-- regiune fezabilă

Notația merge și pentru $i = 0$:
$H_0 = \{ m_1, m_2 \}$
$C_0 = m_1$ intersectat cu m_2

Observație:

I. C_0 îl include pe C_1 care îl include pe … care îl include pe $C_n = C$
II. Pentru fiecare i, regiunea fezabilă C_i, dacă este nevidă, are un vârf care reprezintă o soluție (optimă) a problemei (H_i, c) notat cu v_i (! depinde de alegerea lui m_1 și m_2).

Dacă avem un punct de optim pe parcurs v_i, acesta nu se schimbă dacă adăugăm un nou semiplan h_i și v_i aparține lui h_i.

Observație: fie $1 <= i <= n$. Presupunem că avem deja determinate $C_{i - 1}$ și $v_{i - 1}$.
Considerăm h_i. Sunt două situații:

I. Dacă $v_{i - 1}$ aparține lui h_i, atunci $v_i = v_{i - 1}$
II. Dacă $v_{i - 1}$ nu aparține lui h_i, atunci
    A. Fie C_i = mulțimea vidă.
    B. Fie v_i aparține lui d_i, unde d_i este dreapta care mărginește h_i.
    Găsirea lui v_i revine la găsirea lui p de pe d_i care maximizează $f_C (p)$, date constrângerile deja existente (p aparține lui h, pentru h din H_i)
    ⇔ De fapt este o problemă de programare liniară 1D

Fie (X_i) $i =1,n$ variabila aleatoare.

X_i = {
0 dacă $v_{i - 1}$ aparține lui h_i
1, dacă $v_{i - 1}$ nu aparține lui h_i
}

Timpul total este sumă de $X_i * O(i)$ pentru $1 <= i <= n$

E(acea sumă) = sumă de $O(i) * m(X_i)$, vrem să arătăm că este mai mică sau egală cu sumă de $O(i) * 2/i = O(n)$

Vrem să arătăm că: $m (X_i) <= 2/i$

Altfel spus, vrem să arătăm că probabilitatea ca $v_{i - 1}$ să nu aparțină lui h_i este $<= 2/i$

Backward analysis

Care este probabilitatea ca $v_{n - 1}$ să nu aparțină lui h_n, adică la adăugarea lui h_n, vârful $v_{n - 1}$ să fie modificat în v_n?

Echivalent: Care este probabilitatea ca eliminând unul dintre semiplane să fie modificat vârful optim?

Doar două dintre drepte determină vârful optim. Șansa să eliminăm o dreaptă care afectează punctul de intersecție este $2/n$.

$\Rightarrow m(X_i) \le 2/i$

Triangulări

Proprietăți

Proprietate. Numărul de triunghiuri, muchii și vârfuri ale unei triangulări
Demonstrație. Construim graf:

- vârfurile grafului: punctele inițiale, în număr de n
- muchiile: laturile triunghiului ($n_m = ?$)
- fețele: triunghiurile ($n_t = ?$) + fața exterioară (= 1)

Relația lui Euler:

$$n - n_m + (n_t + 1) = 2$$

Incidențe (adiacențe) între muchii și fețe:

$2 * n_m$	=	$3 * n_t + k$
incidențe din „perspectiva muchiilor”		incidențe din „perspectiva fețelor”

Calitatea triangulărilor

Triangulări mai bune: au unghiurile mai mici

O muchie este ilegală dacă $\min(A(T_{AC})) < \min(A(T_{BD}))$

Excepție de la definiția muchiei ilegale (caz degenerat)

Fie ABCD un patrulater inscriptibil (adică A, B, C, D conciclice = situate pe același cerc).

Vectorii $A(T_{AC})$ și $A(T_{BD})$ nu sunt neapărat egali dar $\min(A(T_{AC})) = \min(A(T_{BD}))$
În acest caz: ambele muchii sunt **legale**.

Criteriul numeric: există o formulă (vezi curs) pentru a determina dacă punctele sunt conciclice sau nu.

Teoremă: pentru a determina triangularea optimă, există un algoritm incremental randomizat, în timp mediu O(n log n), folosind O(n) memorie medie.

Diagrame Voronoi

O **diagramă Voronoi** o împărțire a planului R^2 în n celule V(P_1), ..., V(P_n) în care fiecare celulă conține toate punctele care sunt mai apropiate de ea decât de celelalte.

Exemple

- Două puncte (situri) distincte
  Diagrama este formată din două semiplane, separate de mediatoarea segmentului [AB]
- Trei puncte necoliniare
  Mediatoarele determină trei regiuni, și se intersectează într-un punct comun (centrul cercului circumscris)

Numărul de muchii / vârfuri al diagramei

Demonstrația rezultatului referitor la numărul de muchii sau de vârfuri

n_v <= 2n - 5
n_m <= 3n - 6

Numărul de semidrepte == numărul de puncte de pe acoperirea convexă a punctelor

Presupunem că punctele nu sunt coliniare (altfel e cazul trivial).
Adăugăm un **punct la infinit** v_inf, și unim toate semidreptele cu punctul la infinit.
Astfel obținem un **graf planar conex**.

Vârfurile (nodurile) sunt vârfurile diagramei Voronoi + v_inf
Număr = n_v + 1

Muchiile sunt muchiile diagramei Voronoi
Număr = n_m

Fețele sunt celulele
Număr = n_situri

Incidențe:
- fiecare muchie este incidentă cu exact două vârfuri
- fiecare vârf (inclusiv v_inf) este incident cu cel puțin 3 muchii

Localizarea punctelor în plan

Point Location

<u>Motivație</u>: dată o subdiviziune planară și un punct **q** (*query*), să se determine poziția lui q în raport cu subdiviziunea.

Subdivizare a planului în fâșii (benzi) verticale

Se efectuează <u>căutarea după abscisă</u> pentru identificarea fâșiei verticale.
După aceea, se efectuează <u>căutarea în cadrul unei fâșii</u>, fiind indicat elementul subdiviziunii care conține punctul *q* (se realizează în raport cu segmente).

1) Identificarea fâșiilor verticale, timp de căutare: O(log n)
2) Memorie: în cel mai rău caz O(n^2)

Presupuneri

1. Putem găsi un dreptunghi mare D care să conțină toată subdiviziunea inițială
2. Presupunem că nu există două vârfuri distincte cu același x

Harta trapezoidală a unei subdiviziuni planare

Pentru fiecare vârf al subdiviziunii sunt considerate două „extensii" verticale (superioară/inferioară) care se „opresc" atunci când este întâlnit un alt segment sau D => harta trapezoidală formată din

- segmentele **S**
- dreptunghiul mare **D**
- extensiile

Cum este memorat un trapez?

Rețin segmentele între care se află, și două vârfuri (din stânga și din dreapta).

Observație: sunt posibile configurații pentru vârful lp(T) asociat unui trapez T:

1. este vârful unui triunghi
2. lp este capătul segmentului superior (latura este extensia inferioară)
3. lp este capătul segmentului inferior (latura este extensia superioară)
4. lp nu este capăt de segment (latura este formată din ambele extensii)
5. lp este vârf al dreptunghiului D

Exemplu (referitor la numărul de trapeze și de vârfuri asociate unei hărți trapezoidale)
Cazul n = 1, 4 trapeze (= 3n + 1), 10 vârfuri distincte (= 6n + 4)
Exercițiu: luați alte exemple de hărți trapezoidale în care se realizează / nu se realizează numărul maxim de trapeze / vârfuri

Proprietăți

Pentru o mulțime de n segmente, harta trapezoidală are *cel mult* **3n + 1 trapeze** și **6n + 4 vârfuri**.
Fiecare trapez este adiacent cu **cel mult 4 trapeze** adiacente.

Trapezele se numesc adiacente dacă

Căutarea într-un graf asociat unei hărți trapezoidale

Rădăcina este D, frunzele sunt trapezele individuale.
Nodurile conțin segmentele într-o formă simbolică.

Există două tipuri de noduri:

- **x-nod** (p)
  q este la stânga sau la dreapta dreptei verticale care trece prin p (comparații de abscise)

- **y-nod** (s)
  q este deasupra / dedesubtul lui s (testul de orientare)

Exemplul 1 (structura de căutare asociată)

```
        p1
    /       \
T1              q1
              /     \
            s1      T4
          /   \
        T2    T3
```

Exemplul 2 (adaug un nou segment, cu o extremitate în T3 și una în T4)

```
     __p1_
    /       \__
  T1       ___ r1___
          /            \_____
        s1                   r2
      /    \               /     \
    T2     p2            s2      T7
          /    \        /    \
        T3     s2     T5     |
              /    \         |
            T4     |---------T6
```

Timpul **mediu** de căutare este **log n**

Utilizare practică - Mișcarea unui robot

Dorim să deplasăm un robot (punct) din M_start în M_end, robotul se deplasează prin trapeze.
Pași:

1. Determinăm spațiul liber => harta trapezoidală a spațiului liber C_l, notată cu T(C_l)
2. Construiesc un graf asociat spațiului liber C_l, în care trapezele sunt noduri și muchiile reprezintă adiacența (timp liniar O(n), deoarece am cel mult 3n trapeze)
3. Fiind date M_start și M_end, se caută un drum în interiorul lui C_l
    - Dacă sunt în același trapez: OK
    - Dacă sunt în trapeze diferite: se folosesc centrele de greutate ale trapezelor în care sunt situate punctele, și mijloacele laturilor paralele ale trapezelor adiacente (muchii verticale)

Pregătire pentru examen

Data: 28.01.2020, conform programării. Repartizarea pe săli o să fie pusă pe Moodle.
Consultații (doar întrebări): 27.01.2020, o să fie anunțat pe Moodle

Structura: Lucrare scrisă (60p/100p), timp de lucru 1½h - 2h
Materiale: cu cărțile pe masă - avem voie cu materiale, **FĂRĂ resurse electronice**

Fiecare lucrare trebuie să fie <u>originală</u>

NU COPIAȚI !

<u>Subiecte</u>
Vor fi 7 subiecte:

1. Aplicație elementară din preliminarii (5p)
   Exemplu: coordonate carteziene/polare, produs vectorial, raport, test de orientare

Urmează două subiecte relativ *elementare* legate de **problemele prezentate la curs**.

+ pot apărea atât formulări *directe* (ex. aplică algoritmul) cât și *indirecte* (ex. dați exemple de puncte pentru care se întâmplă un anumit lucru).
+ vezi secțiunile de exerciții din suportul de curs pentru modele de subiecte.
+ eventual cu grad de dificultate mai ridicat

2. (10p)
3. (10p)

4. Complexitate algebrică (10p)
   <u>Exemplu</u>: în R^2 sunt date 3 drepte, prin ecuațiile lor generale. Care este complexitatea algebrică a calculelor pentru a stabili:
    a) dacă cele trei drepte au <u>exact</u> un punct *comun*
    b) coordonatele unicului punct de intersecție (presupunem că a) este verificat)

   Rezolvare:
   a)

d: ax + by + c = 0
d’: a’ x + b’ y + c’’ = 0
d’’: a’’ x + b’’ y + c’’ =0
Date de intrare: a, b, c, a’, b’, c’, a’’, b’’, c’’
Condiția de concurență (într-un singur punct):
| a b c |
| a’ b’ c’ | = 0   +   minor de ord II != 0
| a’’ b’’ c’’ |

---> polinom de gradul III

| a b | = ab’ - a’b = polinom de gradul II cu nedeterminatele a, b’, a’, b
| a’ b’ |

b) Pp. că se verifică a): punctul de intersecție este dat de soluțiile sistemului (de ex.)
{   ax + by + c = 0
{ a’x + b’y + c’ = 0
delta = ab’ - a’b = polinom de gradul II
delta_x = | -c b | = polinom de gradul II
| -c’ b’ |
x = delta_x / delta = (pol gr. II) / (pol gr. II)

5. Algoritmi discutați (10p):
   - background logic/matematic
   - complexitatea timp/spațiu

   Exemple:
   - la Graham’s scan, varianta Andrew: unde intervine în mod esențial ordonarea punctelor?
   - de ce la hărțile trapezoidale sunt preferate „extensiile verticale” (superioare și inferioare) dreptelor verticale duse prin extremitățile segmentelor (adică vârfuri)?
   - de justificat (scurt, concis, la obiect):
     - un algoritm discutat la curs (sau o parte a sa)
       Ex.: dacă la Graham’s scan punctele ar fi deja sortate, cât ar fi complexitatea?
     - un pas/anumiți pași ai unui algoritm din suportul de curs;
     - un algoritm indicat explicit în subiect
   - subiectul vizează înțelegerea cursului

6. Algoritm / problemă cu parametri (10p)
   Transferăm o problemă geometrică într-un algoritm.
   Exemplu: fie A = (0, 0), B = (2, 0), C = (0, 2), D = (alfa, alfa) cu alfa nr. real.
   Să se scrie un algoritm care să indice vârfurile acoperirii convexe a mulțimii de puncte {A, B, C, D} și centrul de greutate al poligonului asociat acestei acoperiri.
   Cazuri:
   1. alfa < 0

2. 0 < alfa < 1
3. alfa > 1

Cazuri degenerate:

4. alfa = 0 (D coincide cu A)
5. alfa = 1

Se punctează:

- figura
- înțelegerea contextului geometric
- calcule (geometrice)
- sesizarea cazurilor degenerate + interpretarea lor
- alegerea corectă a inputurilor

<u>Detalii</u>

Pentru alfa < 0, acoperirea convexă este dată de D, B, C, iar centrul de greutate este G = ((alfa + 2)/3, (alfa + 2)/3)

Pentru alfa = 0, caz degenerat, A = D, acoperirea convexă este dată de A, B, C. G = (⅔, ⅔)

Pentru 0 < alfa < 1, acoperirea convexă este dată de A, B, C, iar G = (⅔, ⅔)

Pentru alfa = 1, caz degenerat, acoperirea convexă este dată de A, B, C, G = (⅔, ⅔)

Pentru alfa > 1, acoperirea este dată de A, B, C, D, centrul este G = media coordonatelor punctelor = ((alfa + 2)/4, (alfa + 2)/4)

<u>Algoritmul</u>:

Input: alfa
Output: calculele de mai sus

7. Ceva mai greu (5p)