Intersecții

Mihai-Sorin Stupariu

Sem. I, 2019-2020

Motivație (Probleme geometrice în context 2D)

- **Problema 1.** Dată o listă (mulțime ordonată) de puncte $\mathcal{P} = (P_1, P_2, \ldots, P_m)$, să se stabilească dacă ea reprezintă un poligon (linie poligonală fără autointersecții).
- **Problema 2.** Date două poligoane $\mathcal{P}$ și $\mathcal{Q}$, să se stabilească dacă se intersectează (interioarele lor se intersectează).
- **Problema 3.** Dată o mulțime $\mathcal{S} = \{s_1, \ldots, s_n\}$ de n segmente închise din plan, să se determine toate perechile care se intersectează.
- **Problema 3'.** Dată o mulțime $\mathcal{S} = \{s_1, \ldots, s_n\}$ de n segmente închise din plan, să se determine toate punctele de intersecție dintre ele.

Determinarea complexității algebrice

- Determinarea complexității algebrice revine la a stabili natura calculelor care trebuie efectuate / a expresiilor care trebuie evaluate pentru a rezolva o problemă (polinoame de un anumit grad, rapoarte de polinoame, etc.). Nu interesează de câte ori se repetă un anumit tip de calcule, ci cât de complexe sunt acestea.
- Pentru a stabili **dacă** două segmente se intersectează: se aplică testul de orientare $\longrightarrow$ **polinoame de gradul II**
- Pentru a **determina explicit** punctul de intersecție dintre două segmente $[AB]$ și $[CD]$: se pornește de la condiția $(1-\lambda)A+\lambda B=(1-\mu)C+\mu D$, care reprezintă un sistem cu necunoscutele λ și μ; sunt calculate λ, μ (dacă există) $\longrightarrow \frac{\textbf{polinom de gradul II}}{\textbf{polinom de gradul II}}$, apoi prin înlocuire în relația inițială se găsesc coordonatele punctului de intersecție $\longrightarrow \frac{\textbf{polinom de gradul III}}{\textbf{polinom de gradul II}}$,

Algoritmul trivial

- **Idee de lucru:** Sunt considerate toate perechile de segmente și se determină cele care se intersectează / se calculează punctele de intersecție.
- **Complexitate:**
  - timp: $O(n^2)$
  - memorie: $O(n)$
  - algebric: polinoame de gradul II (Problema 3), rapoarte de polinoame (Problema 3')
- **Comentariu:** În anumite cazuri: optim (dacă toate segmentele se intersectează).
- Algoritmi mai eficienți (*output* / *intersection sensitive*)?

Rezolvarea problemei în context 1D

- Ordonarea lexicografică a extremităților segmentelor / intervalelor într-o listă $\mathcal{P}$.
- Lista $\mathcal{P}$ este parcursă (crescător); lista $\mathcal{L}$ a segmentelor care conțin punctul curent din $\mathcal{P}$ este actualizată:
  - dacă punctul curent este marginea din stânga a unui segment s, atunci s este adăugat la listă $\mathcal{L}$
  - dacă punctul curent este marginea din dreapta a unui segment s, atunci s este șters din $\mathcal{L}$ și se rapoartează intersecții între s și toate segmentele din $\mathcal{L}$
- **Teoremă.** Algoritmul are complexitate $O(n \log n + k)$ și necesită $O(n)$ memorie (k este numărul de perechi ce se intersectează).

Un prim algoritm

- Dreapta de baleire l: **orizontală**, astfel că toate intersecțiile situate deasupra dreptei de baleiere au fost detectate.
- **Statutul (sweep line status)**: mulțimea segmentelor care intersectează l.
- **Evenimente (event points)**: capetele segmentelor $\longrightarrow$ actualizarea statutului
- **Obs. 1.** Sunt testate pentru intersecție segmente care intersectează, la un moment dat, $l \longrightarrow$ (sunt testate segmentele care sunt "aproape" de-a lungul axei Oy) $\longrightarrow$ încă ineficient.
- **Obs. 2.** Dreapta de baleiere are, de fapt, o variație "discretă", nu continuă.

Un algoritm eficient [Bentley și Ottmann, 1979; Mehlhorn și Näher, 1994]

- **Idee de lucru:** ordonare (segmente, evenimente).
  - Segmentele: ordonate folosind extremitățile superioare (lexicografic, x apoi y).
  - Evenimente (puncte): (lexicografic, y apoi x).
  - Statutul: **listă** (mulțime ordonată)
- **Avantaj:** în momentul modificării statutului, sunt testate intersecțiile doar în raport cu vecinii din listă (sunt testate segmentele care sunt "aproape" de-a lungul axei Ox).
- **Fundamental:** Punctele de intersecție devin, la rândul lor, evenimente, deoarece schimbă ordinea segmenetelor care le determină $\longrightarrow$ chiar dacă nu sunt determinate explicit, trebuie inserate în lista de evenimente (compararea coordonatelor poate necesita utilizarea unor polinoame de gradul V)

3 tipuri de evenimente

- Marginea superioară a unui segment
  - apare un nou segment în statutul liniei de baleiere, ce trebuie inserat corespunzător
  - testat, în raport cu vecinii, dacă au puncte de intersecție sub linia de baleiere $\longrightarrow$ vor deveni ulterior evenimente
- Marginea inferioară a unui segment
  - eliminat un segment din statutul liniei de baleiere
  - testare pentru segmentele vecine nou apărute
- punctele de intersecție (inserate în mod corespunzător pe parcurs)
  - segmentele care le determină trebuie "inversate" în statutul liniei de baleiere
  - testare pentru segmentele vecine nou apărute

Implementare: structuri de date utilizate

- $\mathcal{Q}$ coada de evenimente (*event queue*)
  - puncte, ordonate lexicografic, y apoi x; memorată într-un arbore binar de căutare echilibrat (*balanced binary search tree*)
  - evenimentele nou detectate trebuie inserate în mod corespunzător!
  - de evaluat: complexitatea-timp (pentru o inserare, numărul de repetiții); complexitatea spațiu
- $\mathcal{T}$ – **statut**: arbore binar de căutare echilibrat (*balanced binary search tree*)
  - pe frunze: segmente, este reținută ordinea segmentelor de la stânga la dreapta
  - în nodurile interne: segmente, privite ca elemente de ghidare
  - de evaluat: complexitatea-timp (pentru o actualizare, numărul de repetiții)

Algoritm INTERSECTII

- **Input.** O mulțime de segmente din planul $\mathbb{R}^2$.
- **Output.** Mulțimea punctelor de intersecție (explicit sau doar formal); pentru fiecare punct precizează segmentele pe care se găsește.

1. $\mathcal{Q} \leftarrow \emptyset$. Inserează extremitățile segmentelor în $\mathcal{Q}$; împreună cu marginea superioară a unui segment memorează și segmentul
2. $\mathcal{T} \leftarrow \emptyset$.
3. **while** $\mathcal{Q} \neq \emptyset$
4. **do** determină evenimentul p care urmează în $\mathcal{Q}$ și îl șterge
5. ANALIZEAZA (p)

ANALIZEAZA (p)

1. Fie $U(p)$ mulțimea segmentelor a căror extremitate superioară este p (pentru cele orizontale este marginea din stânga) - stocată cu p în $\mathcal{Q}$.
2. Determină toate segmentele care conțin p: sunt adiacente în $\mathcal{T}$ (de ce?) Fie $D(p)$, respectiv $Int(p)$, mulțimea segmentelor care au p drept margine inferioară, respectiv conțin p în interior.
3. **if** $U(p) \cup D(p) \cup Int(p)$ conține mai mult de un segment
4.   **then** raportează p ca punct de intersecție, împreună cu segmentele din $U(p)$, $D(p)$, $Int(p)$

ANALIZEAZA (p)

5. șterge segmentele din $D(p) \cup Int(p)$ din $\mathcal{T}$
6. inserează segmentele din $U(p) \cup Int(p)$ în $\mathcal{T}$ (ordinea segmentelor pe frunzele lui $\mathcal{T}$ coincide cu ordinea în care sunt intersectate de o linie de baleiere situată imediat sub p)

ANALIZEAZA (p)

if $U(p) \cup Int(p) = \emptyset$
  then fie s_l și s_r vecinii din stânga/dreapta ai lui p din $\mathcal{T}$
    DETERMINAEVENIMENT (s_l, s_r, p)
  else fie s' din $U(p) \cup Int(p)$ cel mai în stânga în $\mathcal{T}$
    fie s_l vecinul din stânga al lui p
    DETERMINAEVENIMENT (s_l, s', p)
    fie s'' din $U(p) \cup Int(p)$ cel mai în dreapta în $\mathcal{T}$
    fie s_r vecinul din dreapta al lui p
    DETERMINAEVENIMENT (s'', s_r, p)

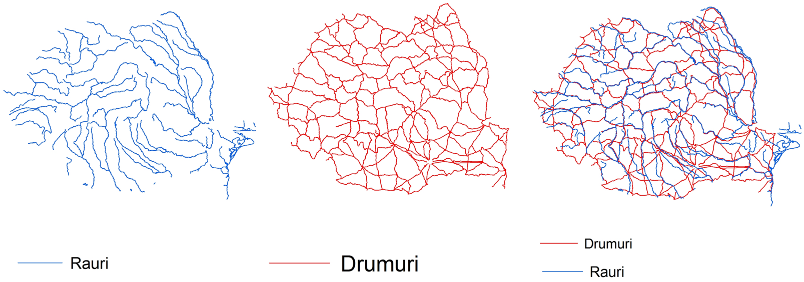
DETERMINAEVENIMENT (sgt_l, sgt_r, p)

1. **if** sgt_l și sgt_r se intersectează sub linia de baleiere sau pe linia de baleiere, dar la dreapta lui p și punctul de intersecție nu este deja în $\mathcal{Q}$
2. **then** inserează punctul de intersecție ca eveniment în $\mathcal{Q}$

Rezultatul principal (intersecții de segmente)

Teoremă. *Fie S o mulțime care conține n segmente din planul $\mathbb{R}^2$. Toate punctele de intersecție ale segmentelor din S, împreună cu segmentele corespunzătoare, pot fi determinate în $O(n \log n + I \log n)$ timp, folosind $O(n)$ spațiu (I este numărul punctelor de intersecție).*

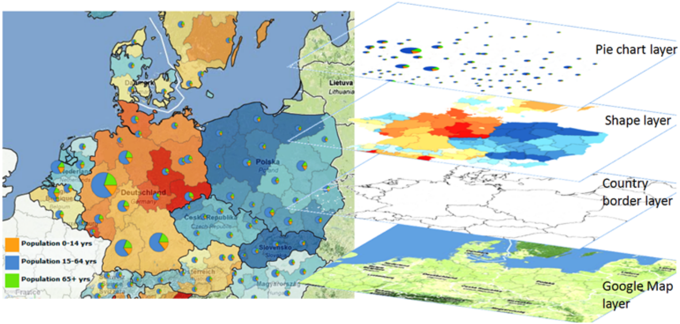
Suprapunerea unor segmente cu conținut tematic diferit



Red-blue intersections [Mairson și Stolfi, 1988; Mantler și Snoeyink, 2000]

Teoremă. Pentru două mulțimi R și B de segmente din $\mathbb{R}^2$ având interioare disjuncte, perechile de segmente ce se intersectează pot fi determinate în $O(n \log n + k)$ timp și spațiu liniar, folosind predicate (polinoame) de grad cel mult II ($n = |R| + |B|$) și k este numărul perechilor ce se intersectează).

Motivație - suprapunerea straturilor tematice



Sursa: https://s-media-cache-ak0.pinimg.com/originals/37/90/86/37908600ab7db99c424c3bc6e1ddb740.jpg

Subdiviziuni planare

- Conceptul de subdiviziune planară: vârfuri, muchii, fețe.
- **Listă dublu înlănțuită** [Müller și Preparata, 1978] (trei înregistrări: vârfuri, fețe, muchii orientate (semi-muchii)).
  - **Vârf** v: coordonatele lui v în $\mathit{Coordinates}(v)$, pointer $\mathit{IncidentEdge}(v)$ spre o muchie orientată care are v ca origine
  - **Față** f: pointer $\mathit{OuterComponent}(f)$ spre o muchie orientată corespunzătoare frontierei externe (pentru fața nemărginită este **nil**); listă $\mathit{InnerComponents}(f)$, care conține, pentru fiecare gol, un pointer către una dintre muchiile orientate de pe frontieră
  - **Muchie orientată** $\vec{e}$: pointer $\mathit{Origin}(\vec{e})$, pointer $\mathit{Twin}(\vec{e})$ pointer $\mathit{IncidentFace}(\vec{e})$, pointer $\mathit{Next}(\vec{e})$, pointer $\mathit{Prev}(\vec{e})$.
- Oricărei subdiviziuni planare $\mathcal{S}$ i se asociază o listă dublu înlănțuită $\mathcal{D}_{\mathcal{S}}$.

Algoritm OVERLAY $(\mathcal{S}_1, \mathcal{S}_2)$

- **Input.** Două subdiviziuni planare $\mathcal{S}_1$, $\mathcal{S}_2$ memorate în liste dublu înlănțuite $\mathcal{D}_{\mathcal{S}_1}, \mathcal{D}_{\mathcal{S}_2}$
- **Output.** Overlay-ul $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$ dintre $\mathcal{S}_1, \mathcal{S}_2$, memorat într-o listă dublu înlănțuită $\mathcal{D}_{\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)}$

1. Copiază listele $\mathcal{D}_1, \mathcal{D}_2$ într-o nouă listă dublu înlănțuită $\mathcal{D}_{\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)}$
2. Calculează toate intersecțiile de muchii dintre $\mathcal{S}_1$ și $\mathcal{S}_2$ cu algoritmul INTERSECTII. La fiecare eveniment, pe lângă actualizarea lui $\mathcal{Q}$ și $\mathcal{T}$, efectuează:
   - Actualizează $\mathcal{D}_{\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)}$, în cazul în care evenimentul implică atât muchii ale lui $\mathcal{S}_1$, cât și ale lui $\mathcal{S}_2$
   - Memorează noile muchii orientate adecvat

Algoritm SUPRAPUNERE $(\mathcal{S}_1, \mathcal{S}_2)$

3. Determină ciclii de frontieră din $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$, stabilește natura (exteriori/interiori)
4. Construiește graful $\mathcal{G}$ ale cărui noduri corespund ciclilor de frontieră și ale cărui arce unesc fiecare ciclu corespunzând unui gol cu ciclul de la stânga vârfului cel mai din stânga și determină componentele conexe ale lui $\mathcal{G}$
5. **for** fiecare componentă a lui $\mathcal{G}$
6. **do** Fie $\mathcal{C}$ unicul ciclu de frontieră exterioară a componentei și fie f fața mărginită a ciclului. Creează o înregistrare pentru f, setează *OuterComponent*(f) (către una din muchiile lui $\mathcal{C}$), construiește lista *InnerComponents*(f) (pentru fiecare gol, pointer către una dintre muchiile orientate). Pentru fiecare muchie orientată, *IncidentFace*() către înregistrarea lui f
7. Etichetează fiecare față a lui $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$

Rezultate principale

- **Teoremă. (Overlay-ul hărților)** *Fie* S_1 *o subdiviziune de complexitate* n_1, S_2 *o subdiviziune de complexitate* n_2, *fie* $n := n_1 + n_2$. *Overlay-ul dintre* S_1 *și* S_2 *poate fi construit în* $O(n \log n + k \log n)$, *unde* k *este complexitatea overlay-ului.*
- **Corolar. (Operații boolene)** *Fie* $\mathcal{P}_1$ *un poligon cu* n_1 *vârfuri și* $\mathcal{P}_2$ *un poligon cu* n_2 *vârfuri; fie* $n = n_1 + n_2$. *Atunci* $\mathcal{P}_1 \cup \mathcal{P}_2$, $\mathcal{P}_1 \cap \mathcal{P}_2$ *și* $\mathcal{P}_1 \setminus \mathcal{P}_2$ *pot fi determinate în timp* $O(n \log n + k \log n)$, *unde* k *este complexitatea output-ului.*