Algoritmi

Domača naloga 3

Sara Bizjak | 27202020

April 2021

Problem 1 - Konveksna ovojnica

Naj bosta S_1 in S_2 konveksni ovojnici v ravnini.

A del:

Algoritem za izračun unije teh dveh ovojnic.

B del:

Algoritem, ki preveri, če se konveksni ovojnici sekata.

Problem 2 - Delaunay triangulacija

Podan imamo naključnosti algoritem za iskanje Delaunay triangulacije (opisan v [1]) s pričakovanim časom izvajanja $\mathcal{O}(n \log n)$. Pokažemo, da je najslabši čas izvajanja $\mathcal{O}(n^2)$.

Zgeneriramo primer, pri katerem se za vsako dodano točko p_i v i-item koraki generira okoli i novih robov (z *edge flipping-om*).
Opazujmo množico n točk P, ki leži na pozitivni veji parababole $y = x^2$, torej

$$P = \{(t_1, t_1^2), (t_2, t_2^2), \ldots, (t_n, t_n^2)\},$$

kjer so $t_1, t_2, \ldots t_n$ pozitivna realna števila. BŠS predpostavimo $0 < t_1 < \cdots < t_n$.
Trdimo, da je v Delaunay triangulaciji P-ja najbolj leva točka (t_1, t_1^2) sosednja z vsako drugo točko iz P. To pomeni, da z dodajanjem vsake nove točke (t_0, t_0^2) v P, tako da velja $0 < t_0 < t_1$ (torej bo ta na novo dodana točka najbolj leva), generiramo tudi n novih robov Delaunay triangulaciji. Če torej dodajamo točke od desne proti levi, je število robov v Delaunayevi triangulaciji enako $\Omega(n^2)$.

Dokaz.
Naj bodo $0 < a < b < c$ pozitivna realna števila in naj bo $\mathcal{C}(a, b, c)$ krožnica skozi točke

$(a, a^2), (b, b^2), (c, c^2)$, ki tvorijo trikotnik v triangulaciji.
Trdimo, da $\mathcal{C}(a, b, c)$ ne vsebuje nobene take točke d, kjer

$$a < d < b \text{ ali } c < d$$

Pokažimo to s protislovjem in recimo, da točka (d, d^2) leži na krožnici $\mathcal{C}(a, b, c)$. To je natanko tedaj, ko velja

$$\begin{vmatrix} 1 & a & a^2 & a^2 + a^4 \\ 1 & b & b^2 & b^2 + b^4 \\ 1 & c & c^2 & c^2 + c^4 \\ 1 & d & d^2 & d^2 + d^4 \end{vmatrix} = 0,$$

kar je enako pogoju

$$(a - b)(a - c)(b - c)(a - d)(b - d)(c - d)(a + b + c + d) = 0.$$

Od tukaj sledi, da mora veljati ena od zvez

$$\begin{aligned} d &= a \\ d &= b \\ d &= c \\ d &= -a - b - c < 0, \end{aligned}$$

saj iz $(a - b), (a - c), (b - c)$ ne moremo dobiti ničle, ker $0 < a < b < c$. Velja tudi, da $(a - d) \neq (b - d) \neq (c - d) \neq (a + b + c + d)$, kar pomeni, da parabola dejansko seka krožnico $\mathcal{C}(a, b, c)$ v teh štirih točkah. Iz tega sledi, da točka (d, d^2) leži znotraj krožnice $\mathcal{C}(a, b, c)$, če velja

$$-a - b - c < d < a \text{ ali } b < d < c$$

Sklepi nam podajo protislovje.
Če torej dodajamo točke, kot je opisano v primeru, je časovna zahtevnost algoritma res s$\Omega(n^2)$.

Problem 3 - Iskanje točk s kd, četrtinskimi in intervalnimi drevesi

A del:

Kd-drevesa, četrtinska drevesa in intervalna drevesa so podatkovne strukture za hranjenje množice točk v ravnini. Vsaka izmed teh struktur ima dotično prednost pred ostalimi pri določenih pogojih.

- *Kd-drevo – linearna prostorska zahtevnost* $\mathcal{O}(n)$. Prostorska zahtevnost intervalnih dreves je $\mathcal{O}(n \log^{d-1}(n))$, kjer je n število shranjenih točk v drevesu, d pa njihova dimenzija, četrtinskih pa $\mathcal{O}((g + 1)n)$. Tukaj z g označimo globino drevesa, ki pa je logaritemsko odvisna od razmerja med najmanjšo razdaljo med dvema točkama in stranico začetnega kvadrata. Ker je v teoriji tako razmerje lahko poljubno, je lahko potem tudi prostorska zahtevnost poljubna (velika).

- *Četrtinsko drevo – dodajanje novih točk v že zgrajeno drevo.* Če v že zgrajeno drevo dodamo nove točke, se četrtinsko drevo ne bo spremenilo, medtem ko sta lahko po dodajanju kd-drevo in intervalno drevo precej spremenjeni in drugačni, kot če bi imela že na začetku na razpolago vse točke.

- *Intervalno drevo – hitrejše poizvedbe.* Iskanje v intervalnem drevesu je navzgor omejeno z $\mathcal{O}(\log^d n)$, iskanje v kd-drevesu poteka v $\mathcal{O}(n)$. V primerjavi s četrtinskim drevesom pa ima boljši čas za iskanje tuidi v primerih, ko so točke manj ugodno razporejene. Taka razporeditev je opisana v prvi alineji.

B del:

Naj bo P množica točk v 3-dimenzionalnem prostoru. Opišemo algoritem za izdelavo osmiškega drevesa točk iz P.
Vhod algoritma je množica točk P. Če začetne kocke nimamo podane, jo izračunamo s pomočjo ekstremov v vseh treh smereh, tj. v x, y in z smeri, saj smo v 3-dimenzionalnem prostoru.
Ko imamo začetno kocko določeno, algoritem ponavljamo rekurzivno. Če je v kocki več kot ena točka, jo razdelimo na osem enako velikih delov (oktantov), ki predstavljajo otroke kocke, ki smo jo razdelili. Enako ponavljamo na otrocih z več kot eno vsebovano točko. Algoritem zaključi in vrne osmiško drevo, ko v nobenem delu ni več kot ene točke.

Literatura

[1] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried CheongSchwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.

[2] Don Sheehy, *Computational Geometry: Lecture 7*, 2010, [ogled 15. 4. 2021], dostopno na `www.cs.cmu.edu/afs/cs/academic/class/15456-s10/ClassNotes/lecture7.pdf`