# 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.*

Algoritem sprejme dve ovojnici, točke so urejene od skrajno leve po x-koordinati in potem po vrsti v nasprotni smeri urinega kazalca.
Ideja je, da vsako ovojnico naprej razdelimo na zgornjo in spodnjo pot (spodnje in zgornje točke). Dobimo ju tako, da za vsako ovojnico posebej poiščemo skrajno levo in skrajno desno točko po x-koordinati. Nato potujemo od leve proti desni v nasprotni smeri urinega kazalca, kar nam poda spodnjo pot in od desne proti levi v nasprotni smeri urinega kazalca, kar nam poda zgornjo pot. Obe zgornji in spodnji poti vsake ovojnice združimo v eno (enako urejeno kot opisano zgoraj) pot. Iz teh dveh poti zgradimo ovojnico, zgornjo in spodnjo posebej. To naredimo tako, da točke dodajamo po vrsti, eno po eno (inkrementalno). Ko želimo točko dodati, opazujemo orientacijo oz. smer zavoja, ki mora biti t. i. zavoj v levo. Na koncu zgornjo in spodnjo ovojnico le še združimo, kar je tudi izhod algoritma.
Algoritem je prirejen po predavanjih in vajah pri predmetu *Računska geometrija, FMF*.
Predpostavke, po katerih deluje opisan algoritem:

- $S_1$ in $S_2$ sta konveksni ovojnici.
- Točke so zapisane v obratni smeri urinega kazalca, začenši z najbolj levo po x-koordinati.
- Če $S_1$ vsebovana v $S_2$, je unija konveksnih ovojnic $S_2$ in obratno.
- Algoritem deluje tudi, če so tri točke vhodne konveksne ovojnice kolinearne.

Psevdo-koda opisanega algoritma:

---

```
def unija(S1, S2):
    S1_zgornja_pot, S1_spodnja_pot = razdeli(S1)
    S2_zgornja_pot, S2_spodnja _pot= razdeli(S2)
```

```
    zgornja_pot = S1_zgornja_pot +  S2_zgornja_pot
    spodnja_pot = S1_spodnja_pot +  S2_spodnja_pot
    zgornja_ovojnica = naredi_zgornjo_ovojnico(zgornja_pot)
    spodnja_ovojnica = naredi_spodnjo_ovojnico(spodnja_pot)
    ovojnica = zdruzi(zgornja_ovojnica, spodnja_ovojnica)
    vrni zdruzeno
```

---

Podrobnosti delovanja algoritma so vidne v priloženi datoteki `dn3_code.py`.

### B del: *Algoritem, ki preveri, če se konveksni ovojnici sekata.*

Konveksni ovojnici $S_1$ in $S_2$ se sekata, če obstaja vsaj eno presečišče med stranicama iz $S_1$ in $S_2$. Algoritem generira vse pare med stranicami ene in druge konveksne ovojnice in za vsak par preveri ali se sekata. Če obstaja vsaj en tak par, ki se seka, potem vrne `True`, sicer vrne `False`. Za ugotavljanje presečišč med dvema daljicama je uporabljena funkcija iz knjižnice `shapely`. Predpostavke, po katerih deluje opisan algoritem:

- Če je ena konveksna ovojnica popolnoma vsebovana v drugi, se ne sekata.
- Če sta konveksni ovojnici enaki, se sekata.
- Če se konveksni ovojnici zgolj dotikata, se sekata.

Psevdo-koda opisanega algoritma:

---

```
def presek(S1, S2):
    seznam_presecisc = []
    pari_stranic = generiraj_pare(S1, S2)
    za vsak par iz pari_stranic:
        dodaj se_sekata(par) v seznam_presecisc
    ce True vsebovan v seznam_presecisc:
        vrni True
    sicer:
        vrni False
```

---

Podrobnosti delovanja algoritma so vidne v priloženi datoteki `dn3_code.py`.

### C del:

Koda je dostopna v priloženi datoteki `dn3_code.py`.

## 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 $\Theta(n^2)$.

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 (povezana) z vsako drugo točko iz $P$. Če torej dodajamo točke od desne proti levi, je število robov v Delaunayevi triangulaciji enako $\Omega(n^2)$.

*Dokaz.*
Pokažimo to s protislovjem in recimo, da zadnja dodana točka (najbolj leva), označimo jo z $(d, d^2)$, ni povezana z vsemi ostalimi. Tedaj obstajajo $0 < a < b < c$ pozitivna realna števila različna od $d$, tako da točke $(a, a^2), (b, b^2), (c, c^2)$ tvorijo trikotnik v triangulaciji.
Ker po definiciji Delaunay triangulacije krog skozi tri točke iz $P$ ne vsebuje nobene druge točke, želimo pokazati, da $\mathcal{C}(a, b, c)$ ne vsebuje nobene take točke $d$.
Točka $(d, d^2)$ leži na krožnici $\mathcal{C}(a, b, c)$ 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, torej zadnja dodana točka res porodi robove z Delanuay triangulaciji z vsemi ostalimi točkami v $P$.
Če torej dodajamo točke, kot je opisano v primeru, je časovna zahtevnost algoritma res $\Omega(n^2)$. Primer je povzet po [3].

## 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`.

[3] *Worst case of the randomized incremental Delaunay triangulation algorithm*, [ogled 15. 4. 2021], dostopno na `cstheory.stackexchange.com/questions/12147/what-is-the-worst-case-of-the-randomized-incremental-delaunay-triangulation-algo`.

[4] *Convex polygons*, [ogled 15. 4. 2021], dostopno na `hackmd.io/@US4ofdv7Sq2GRdxti381_A/ryFmIZrsl?type=view`.