

11 Poligon i poligonska površ

Poligoni su važni objekti u geometriji, a posebno u njenim primenama u računarstvu. Naime, da bi se sa krivima i površima efikasnije računalo i da bi se one lakše grafički predstavljale, one se aproksimiraju poligonima, odnosno poligonskim površima. U ovom poglavlju će biti reči o poligonima u ravni.

Definicija 11.1 Poligonska linija $A_1 \dots A_n A_{n+1}$ je unija duži $A_1 A_2, \dots, A_n A_{n+1}$ koje nazivamo **ivice poligonske linije**. Tačke $A_1, \dots, A_{n+1}$ nazivaju se **temena poligonske linije**. Ako je $A_{n+1} = A_1$ kažemo da je **poligonska linija zatvorena** i nazivamo je **poligonom** $A_1 \dots A_n$. Ako nikoje dve ivice poligonske linije nemaju zajedničkih tačka osim što svake dve susedne ivice imaju zajedničko teme poligonsku liniju nazivamo **prostom**.

Iako poligoni izgledaju jednostavni, geometrija i algoritmi koji su za njih vezani mogu biti veoma složeni. U ostatku priče o poligonima pretpostavljamo da su oni ravanski, tj. pripadaju nekoj ravni.

11.1 Unutrašnjost poligona

Sada nam je cilj da definišemo pojam unutrašnjosti i spoljašnjosti poligona. Za to nam treba sledeća

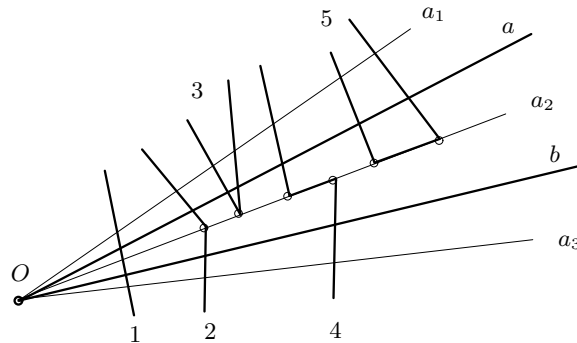
Lema 11.1 *Neka su a i b dve poluprave u ravni poligona $p = A_1 \dots A_n$ sa istim temenom $O \notin p$ koje ne sadrže ni jedno teme poligona. Tada su brojevi $k(a)$ i $k(b)$, presečnih tačaka tih dveju polupravih sa ivicama poligona p iste parnosti.*

Dokaz: Posmatrajmo prave $a_1 = OA_1, \dots, a_n = OA_n$. One razlažu ravan poligona na $m \leq n$ uglova. Razlikujemo tri slučaja.

a) Pretpostavimo da obe poluprave a i b pripadaju istom uglu i neka je to ugao $\angle a_1 a_2$. Pošto unutar tog ugla nema temena poligona, svaka ivica poligona seče pravu a ako i samo ako seče b , pa je tada $k(a) = k(b)$.

b) Pretpostavimo sada da prave a i b pripadaju susednim uglovima, recimo $a \in \angle a_1 a_2, b \in \angle a_2 a_3$. Tada je za ivice poligona mogući sledeći slučajevi:

- 1) Ivica nema temena na a_2 , a seče a_1 i a_3 . Zato ona seče i a i b jedanput.
- 2) Jedno teme poligona je na a_2 , a dve ivice kojima je to teme zajedničko seku po jednu od polupravih a_1 i a_3 . Zato jedna od te dve ivice seče a , a druga b .
- 3) Jedno teme poligona je na a_2 , a dve ivice kojima je to teme zajedničko seku obe, recimo, a_1 . Zato obe te ivice seku a , a nijedna od njih ne seče b .
- 4) Jedna ivica poligona pripada polupravoj a_2 , a dve njoj susedne ivice seku po jednu od pravih a_1 i a_3 . Zato jedna od te dve ivice seče a , a druga b .
- 5) Jedna ivica poligona pripada polupravoj a_2 , a dve njoj susedne ivice seku seku obe, recimo, a_1 . Zato obe te ivice seku a , a nijedna od njih ne seče b .



Slika 20: $k(a) - k(b)$ je paran broj

U svakom od slučajeva 1)-5) razlika $k(a) - k(b)$ je paran broj. Dakle $k(a)$ i $k(b)$ su iste parnosti ako poluprave a i b pripadaju susednim uglovima.

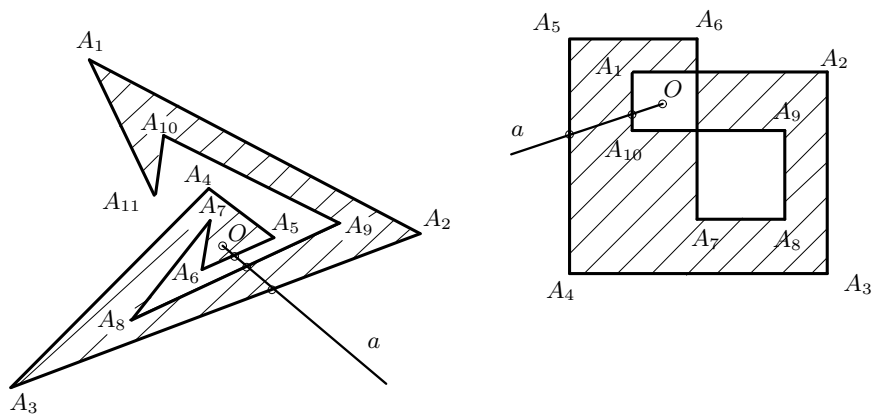
c) Konačno, ako a i b ne pripadaju ni istom, ni susednim uglovima, moguće je naći r pravih $t_1, \dots, t_r$ tako da su $a, t_1, \dots, t_r, b$, redom, u susednim uglovima. Prema dokazanom pod b) brojevi

$$k(a) - k(t_1), \quad k(t_1) - k(t_2), \quad \dots, \quad k(t_r) - k(b)$$

su parni, pa je i njihov zbir $k(a) - k(b)$ paran, tj. $k(a)$ i $k(b)$ su iste parnosti. $\square$

Definicija 11.2 Za tačku O u ravni poligona p , $O \notin p$, kažemo da pripada **unutrašnjosti poligona** p ako je $k(a)$ paran broj, gde je $k(a)$ broj presečnih tačaka poligona i proizvoljne poluprave a sa temenom O koja ne sadrži ni jedno teme poligona. Ako je taj broj neparan tačka O pripada **spoljašnjosti poligona**.

Na osnovu prethodne leme ova definicija ne zavisi od izbora poluprave a .



Slika 21: Unutrašnjost prostog i složenog poligona

Iako definicija unutrašnjosti važi za svaki poligon, ona se poklapa sa intuicijom samo za proste poligone. Za poligone koji nisu prosti intuicija ne daje dobar odgovor šta je unutrašnjost, a šta spoljašnjost. U ostatku priče o poligonima pretpostavljamo da su poligoni prosti.

Sljedeću teoremu navodimo bez dokaza.

Teorema 11.1 *Prost poligon razbija ravan na unutrašnjost i spoljašnjost. Unutrašnjost i spoljašnjost su neprazni i povezani geometrijski likovi.*

11.2 Triangulacija poligona

Pod **triangulacijom prostog poligona** podrazumevamo razlaganje unutrašnjosti poligona na trouglove, međusobno nepresecajućim unutrašnjim dijagonalama.

Teorema 11.2 *Svaki prost poligon ima triangulaciju. Svaka triangulacija prostog poligona sa n temena se sastoji od tačno $n - 2$ trougla.*

Dokaz: Dokazaćemo ovu teoremu indukcijom po n - broju temena poligona $A_1 A_2 \dots A_n$. Ako je $n = 3$ poligon je trougao i teorema važi.

Neka je $n > 3$ i pretpostavimo da je teorema tačna za sve $m < n$.

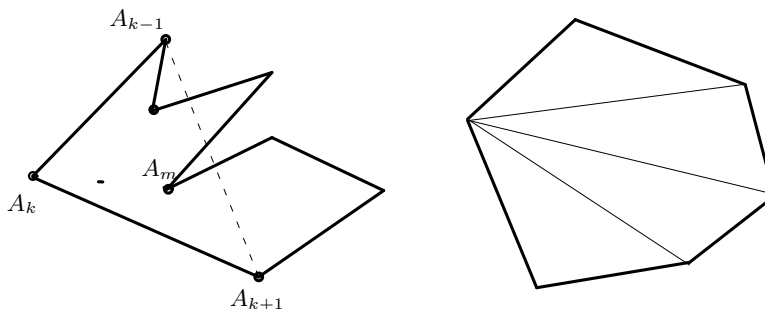
Dokažimo prvo da postoji unutrašnja dijagonala poligona. Neka je A_k "najlevlje" teme polinoma, a ako takvih ima više odaberimo, određenosti radi, ono "najniže". Tada su A_{k-1} i A_{k+1} njemu susedna temena poligona. Ako je dijagonala $A_{k-1} A_{k+1}$ unutrašnja dokazali smo. Ako nije, tada postoji jedno ili više temena poligona unutar trougla $A_{k-1} A_k A_{k+1}$ ili na dijagonali $A_{k-1} A_{k+1}$. Neka je A_m takvo teme koje je najdalje od dijagonale $A_{k-1} A_{k+1}$. Dijagonala $A_k A_m$ ne seče ni jednu ivicu poligona, jer bi takva ivica imala teme unutar trougla $A_{k-1} A_k A_{k+1}$ koje je dalje od $A_{k-1} A_{k+1}$ nego teme A_m . Da je dijagonala $A_k A_m$ unutrašnja sledi iz pretpostavke da je teme A_k najlevlje (objasniti!).

Unutrašnja dijagonala razbija poligon $A_1 A_2 \dots A_n$ na dva prosta poligona sa brojem temena manjim od n . Po induktivnoj hipotezi ta dva poligona imaju triangulacije, što nam daje i triangulaciju polaznog poligona.

Neka je sada data proizvoljna triangulacija poligona $A_1 A_2 \dots A_n$ i neka je d neka unutrašnja dijagonala te triangulacije. Ona razbija poligon na dva prosta poligona sa m_1 , odnosno m_2 temena i važi $m_1 + m_2 = n + 2$. Po pretpostavci indukcije ta dva poligona su triangulacijom razbijena na $m_1 - 2$, odnosno $m_2 - 2$ trougla, pa je poligon $A_1 A_2 \dots A_n$ razbijen na

$$(m_1 - 2) + (m_2 - 2) = (m_1 + m_2) - 4 = n - 2$$

trouglova. □



Dokaz ove teoreme nam zapravo daje jedan mogući algoritam za triangulaciju poligona. Taj algoritam je rekurzivan.

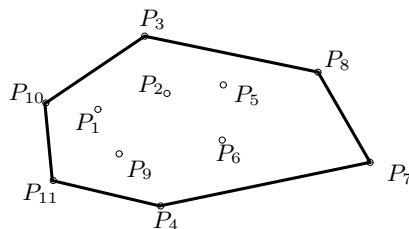
U zavisnosti od načina određivanja unutrašnje dijagonale brzina tog algoritma može da varira između $O(n^2)$ i $O(n^3)$. Postoje i (prilično složeni) algoritmi za triangulaciju čija je brzina reda $O(n \log(n))$.

11.3 Konveksni omotač skupa n tačaka ravni

Podsetimo se da je lik $\mathcal{F}$ **konveksan** ako za svake dve tačke $A, B \in \mathcal{F}$ koje mu pripadaju i cela duž AB pripada liku $\mathcal{F}$. **Konveksni omotač** $\mathcal{CH}(\mathcal{F})$ nekog lika $\mathcal{F} \subset E^n$ je najmanji konveksan podskup od E^n koji sadrži $\mathcal{F}$. U ovom poglavlju bavićemo se određivanjem konveksnog omotača skupa $\mathcal{F} = \{P_1, \dots, P_n\}$ koji se sastoji od n tačaka ravni. Može se dokazati da taj konveksni omotač uvek postoji, da je jedinstven i da je to konveksan poligon čija su temena neke od tačaka skupa $\mathcal{F}$.

Pre nego što prikazemo dva algoritma za nalaženje konveksnog omotača razjasnimo šta znači odrediti konveksni omotač. Pretpostavimo da nam je dat skup $\mathcal{F} = \{P_1, \dots, P_n\}$ tačaka ravni tako što je svaka tačka zadata svojim koordinatama. Treba odrediti ne samo koja od tih tačaka predstavljaju temena konveksnog omotača nego i vratiti ih u smeru suprotnom od kazaljke na satu. Recimo, konveksni omotač na slici 23 je uredjen skup

$$\{P_3, P_{10}, P_{11}, P_4, P_7, P_8\}. \quad (34)$$



Slika 23: Primer konveksnog omotača skupa od 11 tačaka

11.3.1 Konveksni omotač-algoritam brzine (spori algoritam)

Posmatrajmo usmerenu duž $P_k P_l$ koja pripada konveksnom omotaču skupa $\mathcal{F}$. Primetimo da je za svaku tačku $P_m \in \mathcal{F}$, različitu od P_k i P_l trougao $\triangle P_k P_l P_m$ pozitivne orijentacije. Možemo reći i da je tačka P_m "levo" od orijentisane duži $P_k P_l$. Važi i obrnuto: ako je svaka tačka P_m levo od neke usmerene duži $P_k P_l$, tada je ta usmerena duž u konveksnom omotaču. Ova karakterizacija duži koje učestvuju u konveksnom omotaču predstavlja dobru osnovu za algoritam koji dajemo u pseudokodu.

Algoritam Spori konveksni omotač

Ulaz: Niz $P[i]$ koji se sastoji od n tačaka ravni

Izlaz: Niz $CH[i]$ koji se sastoji od $k \leq n$ tačaka konveksnog omotača suprotno od smeru kazaljke na satu

```

for(i=1, i<n, i++){
    for(j=1, j<= n, j++){
        ivica = TRUE;
        for(k=1, k<= n, k++){
            if(not(levo(P[k],P[i], P[j]))){
                ivica = FALSE;
                break;
            }
        }
        if(ivica) dodaj(P[i], P[j], CH);
    }
}
preuredi(CH);

```

Primitimo da postoje tri ugnježdene petlje koje zavise od n što nam daje vreme izvršavanja reda $O(n^3)$. Poslednji korak, preuredjivanje liste CH može se izvesti u vremenu reda $O(n^2)$, pa je ukupna efikasnost algoritma reda $O(n^3)$.

Iako ovaj pseudokod deluje veoma jednostavno postoji nekoliko mesta koja zaslužuju pažnju.

Funkciju `levo` je najlakše implementirati koristeći orijentaciju trougla (vidi poglavlje 4.2.3). Funkcija `levo(P[k], P[i], P[j])` treba da vrati TRUE samo ako je tačka P[k] levo od usmerene duži P[i]P[j] ili ako joj pripada! Naime, ako P[k] pripada duži P[i]P[j] (koja je na rubu konveksnog omotača) tada obe duži P[i]P[k], P[k]P[j] pripadaju rubu konveksnog omotača, ali se mogu zameniti jednom duži, duži P[i]P[j].

Takodje:

- `levo(P[k], P[i], P[j]) = FALSE`, ako $i = j$, jer P[i]P[i] nije ivica;
- `levo(P[k], P[i], P[j]) = TRUE`, ako $k = i$ ili $k = j$.

Funkcija `dodaj(P[i], P[j], CH)` simbolično označava da je orjentisanu ivicu P[i]P[j] treba dodati u konveksni omotač. Medjutim, na kraju algoritma vidimo da je potrebno preurediti temena CH da bi ona predstavljala temena ruba konveksnog omotača.

Recimo, na primeru sa slike 23, pre preuredjivanja CH bi imao sledeći izgled:

$$CH = \{(P_3, P_{10}), (P_4, P_7), (P_7, P_8), (P_8, P_3), (P_{10}, P_{11}), (P_{11}, P_4)\}$$

dok bi posle preuredjivanja morao (do na cikličku permutaciju) da bude dat kao u (34). Drugim rečima, algoritam ne odredjuje ivice konveksnog omotača redom koji nam je potreban. Takodje u neuredjenom nizu CH, svako se teme javlja dva puta.

Zato, kada se detektuje ivica konveksnog omotača, umesto pozivanja funkcije `dodaj(P[i], P[j], CH)` ima više smisla upisati samo indekse i, j u neki privremeni niz celih brojeva. U našem primeru taj bi niz bio:

`indeksi = { 3, 10, 4, 7, 7, 8, 8, 3, 10, 11, 11, 4 }.`

Analizom tog niza nije teško rekonstruisati uredjeni konveksni omotač, što se ostavlja za vežbu čitaocu.

11.3.2 Konveksni omotač (brzi algoritam)

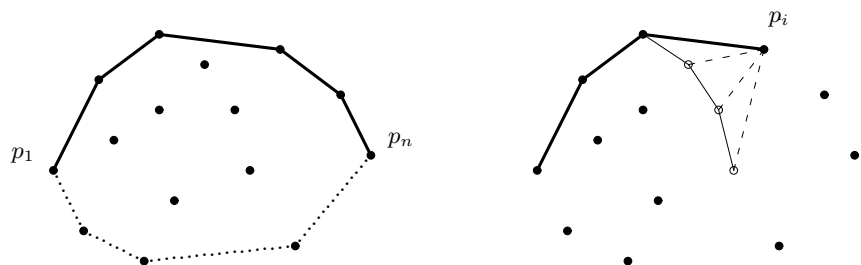
Sada ćemo da damo dosta efikasniji algoritam za nalaženje konveksnog omotača. Algoritam je *inkrementalnog tipa*, tj. niz, tačke se u konveksni omotač dodaju jedna po jedna, ali se nakon svakog dodavanja konveksni omotač ažurira. Nakon dodavanja poslednje tačke i ažuriranja konveksni omotač sadrži tačan rezultat.

Grugo govoreći, algoritam se sastoji iz četiri koraka:

1. Sortiranje datog niza tačaka po x koordinati. Označimo sortirani niz tačaka sa $\{P_1, \dots, P_n\}$;
2. Odredjivanje "gornjeg dela" konveksnog omotača;
3. Odredjivanje "donjeg dela" konveksnog omotača;
4. Spajanje gornjeg i donjeg dela u ceo konveksni omotač.

Ukoliko se prilikom sortiranja desi da nekoliko tačaka ima istu x koordinatu, onda se one sortiraju po y koordinati (tzv. *leksikografski poredak*).

Primetimo da najlevlja i najdesnija tačka (P_1 i P_n) razdvajaju gornji od donjeg omotača. Te dve tačke pripadaju i jednom i drugom delu, o čemu treba voditi računa prilikom njihovog spajanja (korak 4).



Slika 24: Odredjivanje "gornjeg i donjeg dela" konveksnog omotača

Objasnimo sada kako odrediti gornji omotač. Osnovni korak algoritma je kako ažurirati gornji omotač nakon dodavanja temena P_i , tj. kako iz gornjeg omotača tačaka $P_1, \dots, P_{i-1}$ odrediti gornji omotač tačaka $P_1, \dots, P_i$.

Primetimo da svake tri uzastopne tačke gornjeg omotača prave "desni zaokret", tj. njima određen trougao je negativne orijentacije.

Dakle, kada dodamo tačku P_i gornjem omotaču $\{Q_1 = P_1, Q_2, \dots, Q_k\}$ tačaka $P_1, \dots, P_{i-1}$ tako da dobijamo privremeni rezultat $\{Q_1, Q_2, \dots, Q_{k-1}, Q_k, P_i\}$ treba ispitati da li, poslednje tri tačke tj. Q_{k-1}, Q_k, P_i prave desni zaokret. To znači da je tačka P_i desno od usmerene duži $Q_{k-1}Q_k$. Ako je odgovor potvrđan, ništa ne treba dirati. Ako nije, treba izbaciti srednju tačku Q_k , čime se dobija rezultat $\{Q_1, Q_2, \dots, Q_{k-1}, P_i\}$. Ali u ovom slučaju još nismo završili, pošto ovaj postupak treba primenjivati na poslednje tri tačke, izbacujući srednju, sve dok one ne naprave desni zaokret ili dok ne ostanu samo dve tačke u gornjem omotaču.

Na Slici 24. tačke koje izbacujemo iz gornjeg omotača nakon dodavanja tačke P_i su označene "praznim" krugom.

Primetimo da su tačke dobijenog gornjeg omotača već složene u smeru kazaljke na satu.

Donji omotač se određuje slično. Za prvu tačku donjeg omotača uzima se najdesnije tačka P_n , tako da se donji omotač pravi sdesna na levo. Tada svake tri uzastopne tačke donjeg omotača prave takodje desni zaokret.

Algoritam Brzi konveksni omotač

Ulaz: Niz $P[i]$ koji se sastoji od n tačaka ravni

Izlaz: Niz CH[i] koji se sastoji od $k < n$ tačaka konveksnog omotača suprotno od smera kazaljke na satu

sortiraj P[i] po x koordinati

```
gornji = {P[1], P[2]};
for(i=3, i<= n, i++){
    gornji = dodaj(gornji, P[i]) = {Q[1] = P[1], ...Q[k], P[i]};
                                     // k zavisi od i
    while( k > 1 ili levo(P[i], Q[k-1], Q[k])){
        gornji = izbaci(gornji, Q[k]);
        k--;
    }
}

donji = {P[n], P[n-1]};
for(i=n-2, i>=, i--){
    donji = dodaj(donji, P[i]) = {Q[1] = P[n], ...Q[k], P[i]};
    while( k > 1 ili levo(P[i], Q[k-1], Q[k])){
        donji = izbaci(donji, Q[k]);
        k--;
    }
}

CH = spoji(gornji, donji);
```

Primedba 11.1 U tekstu ovog poglavlja, ulaz i izlaz algoritama smo obično zadavali kao nizove. Česta je praksa da se oni, a i drugi pomoćni nizovi u algoritmima zadaju kao povezane liste. Čitaoci familijarni sa listama se ohrabruju da njih koriste prilikom implementacije algoritama i rešavanja zadataka koji slede.

11.1 Napisati funkciju

```
int Prost(Tacka *poligon, int n)
```

koja vraća 1 ako je dati poligon prost, a 0 ako nije prost. Poligon je dat kao niz od n tačaka.

11.2 Napisati funkciju

```
int Unutar(Tacka A, Tacka *poligon, int n)
```

koja za datu tačku A i poligon zadat kao niz tačaka dužine n proverava da li je tačka unutar poligona. Funkcija treba da vrati 1 ako je tačka unutar poligona, 0 ako pripada poligonu, a -1 ako je tačka izvan poligona.

11.3 Napisati funkciju

```
int Triangulisi(Tacka *poligon, int n, Tacka *triangulacija)
```

koja računa triangulaciju datog poligona od n tačaka. Funkcija treba da vrati 0 ako dodje do greške (recimo, poligon nije prost), a broj trouglova triangulacije inače. Niz od $n - 2$ trougla triangulacije (tj. niz od $3(n - 2)$ tačke) treba upisati na mesto pokazano pokazivačem $*triangulacija$.

11.4 a) Napisati funkciju

```
int CrtajOmotac(Tacka *p, int n)
```

koja crta konveksni omotač niza $p[i]$ od n tačaka. Funkcija treba da vrati broj tačaka koje učestvuju u konveksnom omotaču.

b) Modifikacijom prethodne funkcije napisati funkciju

```
int Omotac(Tacka *p, int n, Tacka *poli)
```

koja tačke konveksnog omotača upisuje u niz poli. Tačke rezultata treba da budu složene suprotno od smera kazaljke na satu.

11.5 *Uraditi prethodni zadatak pod b) koristeći brzi algoritam za određivanje konveksog omotača.*