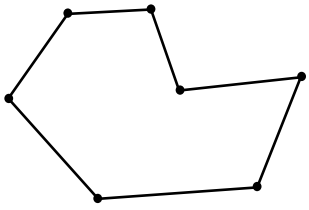
Računska geometrija z mnogokotniki

Jure Slak

→ **Računska geometrija je veda, ki se ukvarja z reševanjem geometrijskih problemov z računalnikom. Problemi, ki jih rešuje, so zelo raznovrstni (ustvarjanje modelov za animirane filme in igre, oblikovanje resničnih objektov, rekonstrukcija površin in iskanje najbližjih točk). V članku bomo ukvarjali z delom, ki razvija algoritme in podatkovne strukture za reševanje problemov z osnovnimi geometrijskimi objekti, kot so točke, daljice in geometrijski liki.**

Geometrijski problemi so zelo lahko rešljivi za človeški um, za računalnik pa so velikokrat precej težji, kot se zdi na prvi pogled. Tako že s pogledom na sliko 1 lahko hitro ugotovimo, ali je lik konveksen.



SLIKA 1.
Primer lika, za katerega človek hitro ugotovi, ali je konkaven ali konveksen.

Spomnimo se definicije konveksnega lika: *Lik je konveksen, če vsaka daljica, katere krajišči ležita znotraj lika tudi sama v celoti leži znotraj lika.*

Zgornja definicija je matematično elegantna; pove da konveksni lik ne sme imeti vbočenih delov, računsko pa ni dosti uporabna. Podobno velja za velik del Evklidske geometrije: lastnosti, definirane s pomočjo geometrijskih objektov, kot so simetrale, enaki koti, vzporednice, nam pri računalniku ne pomagajo veliko, saj z njimi ne znamo delati.

Predstavitev objektov

Prišli smo do prvega problema računske geometrije in sicer, kako predstaviti geometrijske objekte. Odgovor je dandanes relativno samoumeven, s koordinatami. Ta ideja je bila razvita precej pozno. Idejo koordinatnega sistema pripisujemo Renéju Descartesu, ki je živel v 17. stoletju. Zapis točke v ravnini s pomočjo dveh števil je omogočil sistematično povezavo med vizualno naravnano geometrijo in računsko naravnano algebro. Na enak način bomo točke predstavili tudi mi, kot par dveh decimalnih števil (x, y). To sicer ni povsem natančna predstavitev, saj točke $(\sqrt{3}, \pi)$ je bomo mogli predstaviti popolnoma natančno.

Sedaj, ko znamo predstaviti točke, se lahko lotimo predstavitve bolj zapletenih objektov: daljico npr. predstavimo kot par dveh točk, krog pa kot par središča in polmera. Mnogokotnik predstavimo kot zaporedje točk na njegovem robu. Lik na sliki 1 je v programu, v katerem je narisan, predstavljen z zaporedjem sedmih točk:

- $[(545.4688, 1628.2812), (34.1016, 900.9766), (806.8750, 37.3047), (2174.3359, 135.7812), (2549.3750, 1082.8125), (1503.8672, 969.1797), (1261.4453, 1666.1719)]$.

Takih predstavitev je več: odvisno je, pri kateri točki začnemo in v katero smer naštevamo točke. Podobno kot pri trikotnikih bomo rekli, da ima mnogokotnik pozitivno orientacijo, če jih naštevamo v nasprotni smeri urinega kazalca, sicer pa ima negativno orientacijo. Že problem skladnosti likov tako postane zanimiv. Kako lahko ugotovimo, ali predstavljata enak lik, če imamo dva seznama točk, ki predstavljata dva mnogokotnika? Ne moremo samo preveriti, ali sta seznama enaka, ampak moramo preveriti vse možne ciklične zamike prvega seznama in vse ciklične zamike obrnjenega prvega seznama, če se morda ujemajo z drugim seznamom.

Na tem mestu lahko bralec razmisli, kako bi za lik, podan samo s koordinatami preverili, ali je konveksen. Kako bi izračunal njegovo ploščino ali obseg? Algoritmi, ki jih bomo izpeljali, bodo relativno enostavni, vendar bi se brez njih marsikdo ujel v neskončno obravnavo različnih možnosti in posebnih primerov.

Obseg

Pri obsegu standardna definicija prenese tudi v računsko. Obseg mnogokotnika je vsota dolžin vseh njegovih stranic. Dolžino stranice izračunamo kot razdaljo med dvema točkama, za katero uporabimo znano formulo, ki izvira iz Pitagorovega izreka:

- $$d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}. \quad (1)$$

Izračunajmo obseg zgornjega lika s programskim jezikom Python. Točke predstavimo kar z dvema seznamoma x in y koordinat. Velikokrat se za geometrijske objekte naredi svoj razred, kar bomo za voljo enostavnosti tokrat izpustili. Lik na sliki 1 tako predstavimo s spodnjo kodo:

```
x = [545.46875, 34.101562, 806.875, 2174.335938,
     2549.375, 1503.867188, 1261.445312]
y = [1628.28125, 900.976562, 37.304688, 135.78125,
     1082.8125, 969.179688, 1666.171875]
```

Oglejmo si funkcijo, ki izračuna obseg lika:

```
from math import sqrt
def obseg(x, y):     n = len(x) # dolžina seznama x,
      predpostavimo, da je y enako dolg
   o = 0
   for i in range(n):
      j = (i+1) % n # naslednja točka
      # prištejemo dolžino trenutne stranice k
         skupnemu obsegu
      o += sqrt((x[i]-x[j])**2 + (y[i]-y[j])**2)
   return o
```

Vidimo, da je funkcija precej kratka in enostavna. Sprehodimo se po vseh točkah lika za i od 0 do $n - 1$. Za i-to točko naprej ugotovimo njeno naslednjo točko j, ki je kar $i + 1$, če pa je i slučajno zadnja točka, tako da bi $i + 1$ gledal preko konca seznama, pa nastavimo j na 0. Posebni obravnavi zadnje točke se enostavno ognemo s pomočjo operatorja modulo (%), ki vrne ostanek pri celoštevilskem deljenju. Število n je v našem primeru 7, in ko je i 0, 1, 2, 3, 4, ali 5, je $i + 1$ manjši od 7. Ostanek pri deljenju s 7 ničesar ne spremeni in je j kar enak $i + 1$. Ko pa je $i + 1$ enak 7, je ostanek pri deljenju 7 s 7 enak 0, kar je tudi pravilna vrednost za j. Nato k trenutnemu obsegu samo prištejemo razdaljo med točkama (vrstica 8 neposredno uporabi formulo (1)) in ga na koncu vrnemo. Ko funkcijo poženemo s koordinatami našega lika za parametra x in y, dobimo rezultat približno 6944.19.

Ploščina

Izračun ploščine je že zanimivejši od izračuna obsega. Za posebne like, kot so npr. krog ali pravokotnik, ploščino znamo izračunati po formuli. Kaj pa za splošen mnogokotnik?

Začnimo najprej s trikotniki. Osnovna formula za ploščino trikotnika je polovica produkta dolžin višine in osnovnice. V praksi ta formula ni tako dobra, saj je izračun višine težji kot izračun ploščine. To formulo pogosteje uporabimo v obratni smeri, tako da iz ploščine in stranice dobimo višino.

Druga možnost je Heronov obrazec:

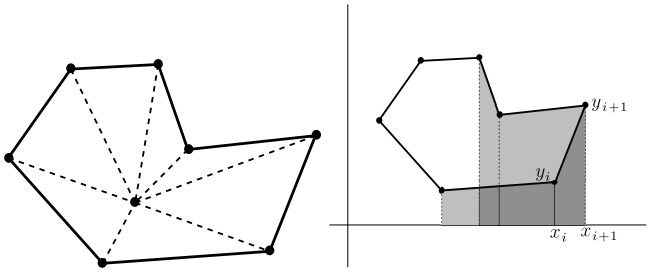
- $$p = \sqrt{s(s-a)(s-b)(s-c)}, \quad s = \frac{a+b+c}{2},$$

ki uporablja le dolžine stranic, toda ima nepotrebno korenjenje. Kot verjetno veste, obstaja tudi formula, ki izrazi ploščino neposredno iz koordinat trikotnika: Če imamo točke $(x_1, y_1), (x_2, y_2), (x_3, y_3)$, potem velja, da je ploščina enaka

- $$p = \frac{1}{2}|x_1y_2 + x_2y_3 + x_3y_1 - x_2y_1 - x_3y_2 - x_1y_3|.$$

Kasneje se bomo kar na splošnejšem primeru mnogokotnika prepričali, da formula drži. Če spustimo absolutno vrednost, dobimo iz formule še več podatkov: predznak namreč pove tudi orientacijo trikotnika.

Znamo uporabiti znanje o ploščinah trikotnikov pri računanju ploščine mnogokotnika? Prva ideja je, da bi mnogokotnik razdelili na trikotnike, podobno kot na sliki 2 levo.



SLIKA 2.
Možni izračuni ploščine mnogokotnika

Ta ideja je za izračun ploščine odlična, toda izkaže se, da je razdelitev mnogokotnika na trikotnike precej težji problem kot izračun ploščine same, saj ne obstaja vedno točka, ki bi jo bilo možno povezati z vsemi oglišči, kot v primeru zgoraj. Taki mnogokotniki imajo celo posebno ime, imenujejo se *zvezdasti*.

Druga ideja je, da ploščino mnogokotnika zapišemo kot vsoto ploščin trapezov, kot na sliki 2 desno. Oglejmo si »navpični« trapez, ki ga dve zaporedni točki (x_i, y_i) in (x_{i+1}, y_{i+1}) tvorita z x-osjo, tj. trapez na točkah $(x_i, 0), (x_i, y_i), (x_{i+1}, y_{i+1}), (x_{i+1}, 0)$. Tak trapez ima višino $x_i - x_{i+1}$ in srednjico $\frac{y_i+y_{i+1}}{2}$, torej je njegova predznačena ploščina enaka $\frac{y_i+y_{i+1}}{2}(x_i - x_{i+1})$. Če seštejemo vse trapeze, dobimo vsoto

- $p = \frac{y_1 + y_2}{2}(x_1 - x_2) + \ldots + \frac{y_n + y_1}{2}(x_n - x_1)$

Trapezi pod likom imajo višino negativno, medtem ko imajo zgornji trapezi višino pozitivno. Del ploščine pod likom se tako ravno odšteje. Enako velja za bolj zapletene like. Podoben sklep bi lahko naredili, tudi če bi risali trapeze po y osi ali pa če bi risali trikotnike do vsake stranice iz izhodišča – v vsakem primeru dobimo neko obliko zgornje formule. Zgornja formula je z vidika računske geometrije odlična, saj direktno poda rezultat iz osnovnih podatkov. Poleg tega je zanimiva tudi matematično; z njo namreč lahko dokažemo, da ima vsak mnogokotnik s celoštevilskimi koordinatami ploščino, ki je celoštevilska ali pa na polovici med dvema celima številoma.

Preverimo lahko tudi, da se za trikotnik splošna formula ujema s prej napisano do absolutnih vrednosti natančno:

- $$\begin{aligned} p &= \frac{y_1 + y_2}{2}(x_1 - x_2) + \frac{y_2 + y_3}{2}(x_2 - x_3) \\ &\quad + \frac{y_3 + y_1}{2}(x_3 - x_1) \\ &= \frac{y_1x_1}{2} + \frac{y_2x_1}{2} - \frac{y_1x_2}{2} - \frac{y_2x_2}{2} \\ &\quad + \frac{y_2x_2}{2} + \frac{y_3x_2}{2} - \frac{y_2x_3}{2} - \frac{y_3x_3}{2} \\ &\quad + \frac{y_3x_3}{2} + \frac{y_1x_3}{2} - \frac{y_3x_1}{2} - \frac{y_1x_1}{2} \\ &= \frac{1}{2}(x_1y_2 + x_2y_3 + x_3y_1 - x_2y_1 - x_3y_2 - x_1y_3) \end{aligned}$$

Ravno predznak ploščine nosi dodatno informacijo o orientaciji mnogokotnika, kar je računski odgovor na geometrijsko vprašanje.

Oglejmo si primer implementacije metode za računanje ploščine v Pythonu.

```
def ploscina(x, y):
    n = len(x) # dolžina seznama x, predpostavimo, da
        je y enako dolg
    p = 0
    for i in range(n):
        j = (i+1) % n # naslednja točka
        # prištejemo dolžino trenutne stranice k skupni
            ploščini
        p += (y[i] + y[j])*(x[i] - x[j])
    return p/2
```

Program je podoben tistemu za obseg, kjer zopet uporabimo enak trik s % za naslednika. Za naš lik program vrne približno 2508792.033, kar pove, da ima lik pozitivno orientacijo.

Na koncu velja dodati še opombo, da formula velja za enostavne mnogokotnike, to so mnogokotniki, ki nimajo samopresečišč, kar je velika večina mnogokotnikov v praksi, npr. geografske meje. Tudi za pravokotnike s samopresečišči obstajajo formule, vendar je potrebno najprej definirati, kaj je notranjost lika.

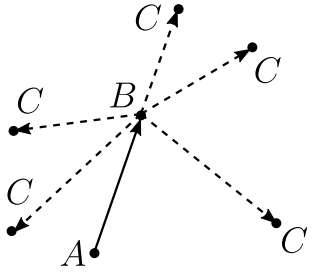
Konveksnost

Kot zadnji primer si oglejmo, kako preverimo, ali je mnogokotnik konveksen. Kot smo že razmislili, nam geometrijska definicija ne koristi kaj dosti, toda uporabimo lahko drugačen geometrijski opis, ki je bližje računski geometriji. Zamislimo si, da hodimo po stranicah mnogokotnika v pozitivni smeri. Vsak ovinek, ki ga naredimo, ko pridemo na novo stranico,

mora biti v levo, sicer imamo »vdolbino« in s tem kršimo konveksnost. Pogoj za konveksnost je torej, da morajo biti vsi ovinki pri obhodu v levo. Ostaja samo še, kako ugotoviti, v katero smer je bil ovinek.

Ovinek v levo

Obravnavajmo situacijo na sliki 3 s pomočjo koordinat in poskusimo izpeljati zanesljiv postopek za ugotavljanje smeri ovinka. S primerjavami koordinat posameznih točk in obravnavo ogromno možnosti se morda uspemo prebiti skozi. Kot vedno pa ne želimo obravnavati posebnih primerov, ampak si želimo postopek, ki bo neodvisen od smeri in bo deloval za vse konfiguracije.



SLIKA 3.
Ugotavljanje smeri ovinka

Ideja se ponuja iz prejšnjega primera. Če si ogledamo trikotnik $\triangle ABC$ in njegovo orientacijo, vidimo, da je negativna, če C leži desno od nosilke AB, pozitivna, če C leži levo od nosilke AB, trikotnik pa je izrojen, če C leži na nosilki AB. V mislih lahko preverimo vse možne lege C, da potrdimo veljavnost zgornjega razmisleka. Orientacijo trikotnika znamo izračunati s pomočjo ploščine. Če izračunamo predznačeno ploščino p, lahko preverimo predznak in vemo, da je ovinek v levo, če je $p > 0$, v desno, če je $p < 0$, in naravnost, če je $p = 0$. Ker nas zanima samo predznak, se lahko tudi izognemo deljenju z 2. Pri iskanju konveksne ovojnice bodo ovinki naravnost šteli kot levi, kar tudi upoštevamo v spodnji funkciji:

```
def v_levo(x1, y1, x2, y2, x3, y3):
    p = x1*y2+x2*y3+x3*y1-x2*y1-x3*y2-x1*y3
    return p >= 0 # vrnemo True, če je ovinek v levo
        ali naravnost
```

Za zainteresirane velja omeniti, da obstaja operacija vektorskega produkta, ki ponuja globljo izpeljavo zgornje formule z več geometrijskega razumevanja, s pomočjo katere enostavno izpeljemo tako formulo za ploščino kot tudi odgovor na vprašanje smeri ovinka.

Preverjanje konveksnosti

Ostane nam le še, da uporabimo zgoraj naučeno, da preverimo, ali je lik konveksen. Za vsakega izmed n ovinkov preverimo, ali je ovinek v levo. V primeru desnega ovinka sporočimo, da lik ni konveksen, sicer pa, da je. To počne spodnja funkcija:

```
def je_konveksen(x, y): # predpostavimo pozitivno
    orientacijo
    n = len(x)
    for i in range(n):
        j = (i+1) % n
        k = (i+2) % n
        if not v_levo(x[i], y[i], x[j], y[j], x[k], y[k]):
            return False # če ovinek ni v levo, zaključimo
    return True
```

Predpostavka o pozitivni orientaciji lahko enostavno preverimo in po potrebi obrnemo seznam točk, preden pokličemo zgornjo funkcijo. V funkciji smo zopet uporabili trik s % za naslednika in še za drugega naslednika. Na primeru našega lika funkcija vrne `False`, saj ovinek $(2549.375, 1082.812) \to (1503.867, 969.179) \to (1261.445, 1666.171)$ ni v levo ($p = -756257.856$).

Drugi problemi

Za zainteresirane bralce lahko omenimo še dva problema podobne težavnosti. Prvi je problem presečišča dveh premic, vsaka podana z dvema točkama. Razviti je potrebno algoritem, ki vrne koordinate presečišča, ali pa pove, da se premici prekrivata ali da sta vzporedni. Tudi tu je možno doseči, da se nobenih premic ne obravnava posebej (npr. navpičnih), ampak obstaja direkten izračun. Drugi problem pa je problem presečišča dveh daljic: ali se sploh sekata in kje. Oba problema je možno rešiti s spoznanimi orodji, le pravilen pristop in dobro mero potrpljenja je potrebno imeti.

× × ×