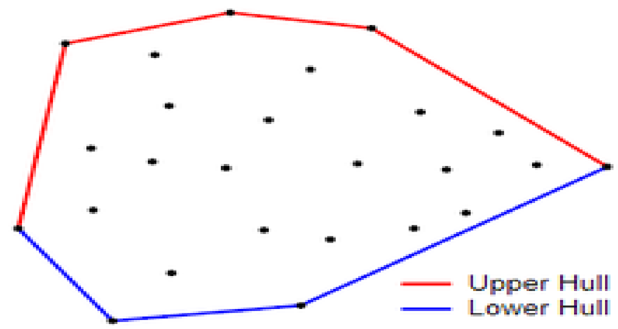
Algoritam "Monotone chain" za konstrukciju konveksnog omotača

David Ivić



Opis problema

Za proizvoljan skup tačaka potrebno je odrediti njihov konveksni omotač, odnosno najmanji konveksi poligon tako da obuhvata samo tačke na krajevima. Jedno od objašnjenja konvkesnog omotača: Na dasci se nalazi zakucano *n* eksera. Ako bismo stavili gumenu traku oko njih, ekseri koji bi zatezali traku jesu tačke konveksnog omotača.

Ulaz: *skup od n tačaka u ravni*

Izlaz: *skup tačaka koje predstavljaju konveksan omotač*

Naivno rešenje problema

Naivno rešenje algoritma se svodi na proveravanje orijentacije svake tri tačke I na kraju uklanjanje duplikata. Očekivana složenost ovog algoritma je O(n^3).

Složenost potiče iz činjenice da za svaki n(n-1)/2 par tačaka posmatramo orijentaciju sa još n-2 tačke.

Naredni pseudokod daje bolji prikaz prethodno opisanog rešenja.

```
Algoritam: KonveksniOmotac
Ulaz: Skup tačaka u ravni P = {P1, P2, ..., Pn}
Izlaz: Lista L koja sadrži temena konveksnog omotača u smeru kazaljke na
    satu

E := 0
za sve parove različitih tačaka P i Q iz skupa P
    valid := true
    za sve tačke R iz P koje su različite od P i Q
        ako je trojka PQR negativno orijentisana onda
            valid := false;
    ako je valid onda
        dodaj usmerenu duž PQ u E
Od skupa stranica E konstruiši listu temena L tako da kreiraju poligon u
negativnoj orijentaciji
```

Napredni algoritam

Ideja naprednog algoritma se svodi na veliko preskakanje tačaka za koje smo sigurni da (za sada) nisu deo konveksnog omotača.

Prvi korak jeste sortiranje tačaka po rastućim x-koordinatama (ukoliko su jednake, po y-koordinatama). Ovim se postiže mogućnost da uočimo ekstremne tačke, odnosno tačke koje se nalaze skroz levo i skroz desno za dati ulaz, kao i da dobijemo redosled po kom obilazimo tačke. Povučenom linijom između ta dva čvora smo ugrubo podeili skup na dva polu-skupa, tačke iznad povučene prave i tačke ispod nje. Ideja koja se krije iza ove podele jeste da konveksni omotač izgradimo upravo od ovih polovina, omotača gornje i donje polovine. Kao što je već napomenuto, kada gradimo polovinu obrađujemo tačke samo sa jedne strane prave. Naime, za konstruisanje gornje polovine nas ne interesuju tačke koje se nalaze ispod povučene prave jer smo za njih sigurno da neće biti deo rezultujućeg skupa za tu polovinu. Sličan princip se koristi i za konstruisanje donje polovine.

Konstruisanje polovine

U nastavku će biti opisan način na koji se konstruišu polovine konveksnog omotača.

Kao što je već napomenuto tačke date na ulazu prolaze kroz uslov da li se nalaze sa odgovarajuće strane prave povečene između dve ekstremne. Za tačke koje prolaze ovaj uslov želimo da proverimo njihovu orijentaciju. Naime, tačke postaju deo konveksnog omotača dokle god ne naruše *clockwise* orijentaciju što se proverava sledećim uslovom:

```
bool MonotoneChain::checkCondition(Sides side)
{
    return m_lastPointAdded>=2 &&
        GetPointSide(m_result[m_lastPointAdded-2], m_result[m_lastPointAdded-1], m_result[m_lastPointAdded]) ==side
}
```

Primetimo i to da ovaj uslov ne može biti deo "IF" naredbe, već naredbe "While". Ovo je bitno jer ako dođemo do tačke koja menja našu orijetanciju treba da proverimo koje sve tačke iza nje treba izbaciti iz konveksnog omotača jer one narušavaju tu orijentaciju. U nastavku je dat prikaz dela koda za konstruisanje gornje polovine konvesnog omotača.

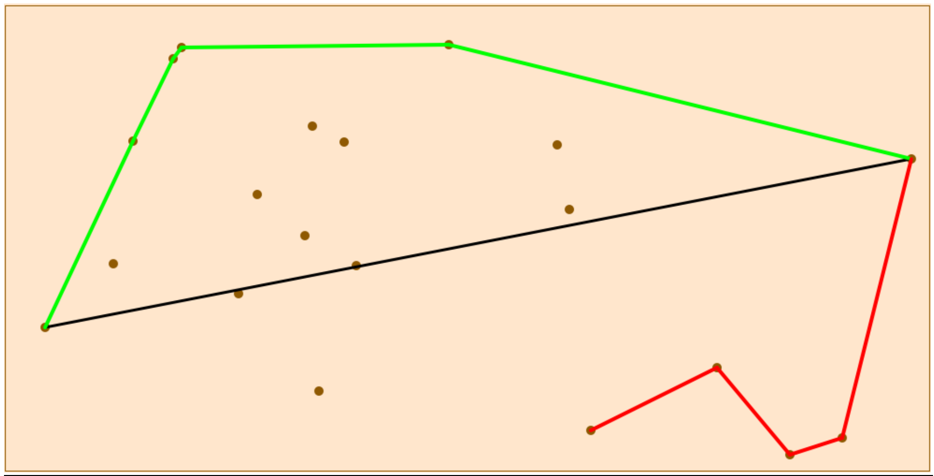
```
//upper hull
for (; _points[i]!=m_maxXmaxY; ++i)
{
if (GetPointSide(m_minXminY, m_maxXmaxY, _points[i]) == Sides::LEFT)
    continue;
m_result.push_back(_points[i]);
m_lastPointAdded++;
AlgorithmBase_updateCanvasAndBlock();
while(checkCondition(Sides::RIGHT))
{
    m_result.pop_back();
    m_result.pop_back();
    m_result.push_back(_points[i]);
    m_lastPointAdded--;
    AlgorithmBase_updateCanvasAndBlock();
}
}
m_result.push_back(m_maxXmaxY);
m_lastPointAdded++;
AlgorithmBase_updateCanvasAndBlock();
while(checkCondition(Sides::RIGHT)){
    m_result.pop_back();
    m_result.pop_back();
    m_result.push_back(m_maxXmaxY);
    m_lastPointAdded--;
}
```

Sličan princip je primenjen za konstruisanje donje polovine omotača.

Složenost prikazanog algoritma je O(nlong) zbog početnog sortiranja.

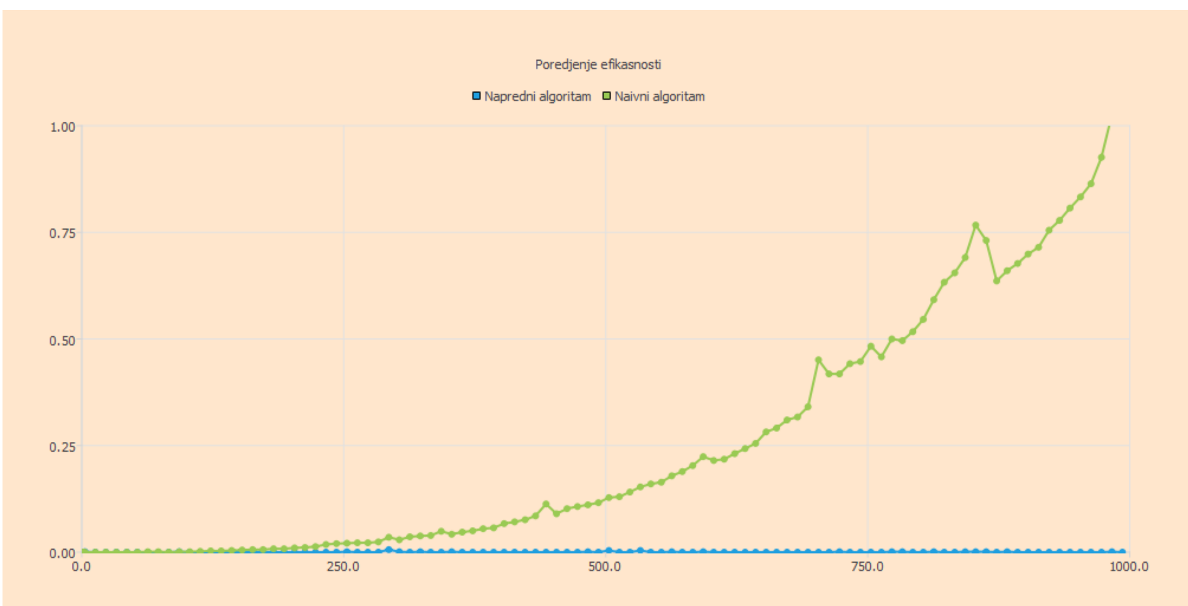
Vizuelizacija algoritma

Linija zelene boje obeležava gornju polovinu konveksnog omotača, dok se crvena linija odnosu na donju polovinu omotača. Linija crne boje povezuje dve ekstremne tačke i pravi razliku između polovina.



Poredjenje efikasnosti naivnog i naprednog algoritma

Na slici se nalazi prikaz odnosa naprednog i naivnog algoritma



Testiranje ispravnosti algoritma

Naziv testa	Opis testa	Ulaz	Očekivani izlaz
verticalTest	Vertikalan niz tačaka	100 100 100 200 100 300 100 400 100 500	Poklapanje rezultata oba algoritma
horizontalTest	Horizontalan niz tačaka	100 100 200 100 300 100 400 100 500 100	Poklapanje rezultata oba algoritma
diagonalDownTest	Kolinearne tačke na dijagonali	50 50 100 100 150 150 200 200 250 250	Poklapanje rezultata oba algoritma
diagonalUpTest	Kolinearne tačke na dijagonal	50 250 100 200 150 150 200 100 250 50	Poklapanje rezultata oba algoritma
lessThanThreePoints	Manje od tri tačke	XX	Poklapanje rezultata oba algoritma
rectangleTest	ulaz u obliku pravougaonika	50 50	Poklapanje rezultata oba algoritma

		200 50 200 200 50 200	
triangleHDownLTest	Kolinearne tačke u obliku trougla	100 100 200 200 300 200 400 200 500 200	Poklapanje rezultata oba algoritma
triangleHDownRTest	Kolinearne tačke u obliku trougla	50 100 100 100 200 100 300 100 500 50	Poklapanje rezultata oba algoritma
triangleHDownRTest	Kolinearne tačke u obliku pravouglog trougla	250 50 250 100 250 150 250 200 250 250 50 250 100 250 150 250 200 250	Poklapanje rezultata oba algoritma
triangleVUpLTest	Kolinearne tačke u obliku trougla	100 400 200 100 200 200 200 300	Poklapanje rezultata oba algoritma
triangleVUpRTest	Kolinearne tačke u obliku trougla	100 100 100 200	Poklapanje rezultata oba algoritma

		100 300 100 400 300 500	