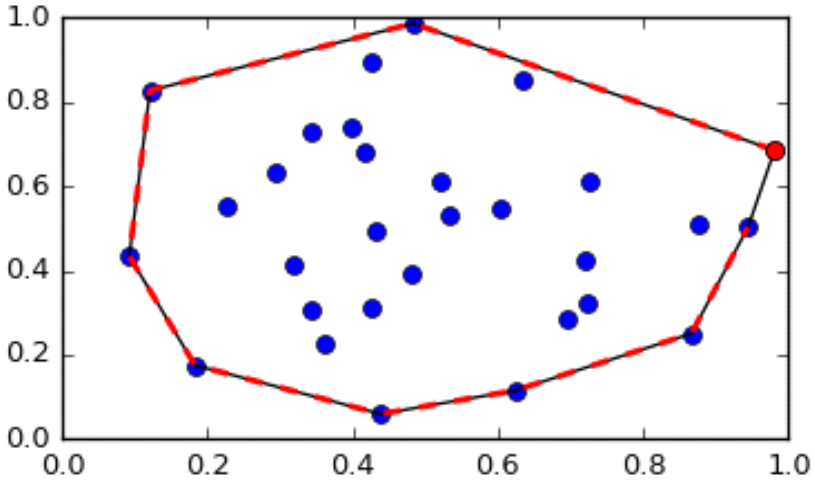
Geometrijski algoritmi @ MATF

Algoritam "Gift wraping" 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 $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$
Izlaz: Lista L koja sadrži temena konveksnog omotača u smeru kazaljke na satu

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

Napredni algoritam

Algoritam "Jarvi's March", koji je poznat i još kao i "Gift Wrapping", nudi rešenje za pronalazak konveksnog omotača na napredan način. Alternativno ime "Gift Wrapping" je proisteklo iz situacije da se algoritam završava tek kada poslednju tačku omotača povežemo sa prvom (kao pakovanje poklona).

Ideja algoritma je sledeća:

Kreće se od najlevlje tačke (od tačke sa najmanjom X koordinatom) jer se za nju zna da će biti deo konveksnog omotača.

Bira se sledeća tačka iz skupa i ona predstavlja sledeći potencijalni čvor našeg omotača. Povlači se duž između prethodno određene tačke i proverava se pozicija svake tačke iz početnog skupa u odnosu na tu duž. Ukoliko naiđemo na tačku koja se nalazi levo od te duži, ona je naš nov potencijalni čvor omotača. Tek kada uradimo obilazak celog ulaznog niza i utvrdimo da ne postoji tačka koja je *više* levo od naše potencijalne tačke, ubacujemo je u niz rešenja, a duž predstavlja ivicu konveksnog omotača.

Algoritam se završava tek kada uradimo povezivanje poslednjeg i prvog čvora omotača.

Uslov za završetak se svodi na činjenicu da je ulazni skup konačan i njegova složenost je O(n*h) u najgorem slučaju, gde je *n* veličina ulaza, a *h* veličina niza izlaznih tačaka, pa se tako ovaj algoritam svrstava u izlazno-zavisne algoritme.

Određivanje pozicije nove tačke u odnosu na duž se određuje u konstantom vremenu:

```
int GiftWrap::GetPointSide(QPoint A, QPoint B, QPoint P){

    int d = (P.x() - A.x())*(B.y() - A.y()) - (P.y() - A.y())*(B.x() - A.x());
    if (d == 0)
        return Sides::COLLINEAR;
    else if (d < 0)
        return Sides::LEFT;
    else
        return Sides::RIGHT;
}
```

Nama je interesantna situacija samo kada je *d < 0* jer sastavljamo konveksni omotač u *counterclockwise* redosledu. Za potrebe pravljenja omotača u drugom smeru, zanimala bi nas situacija kada je *d >0.*

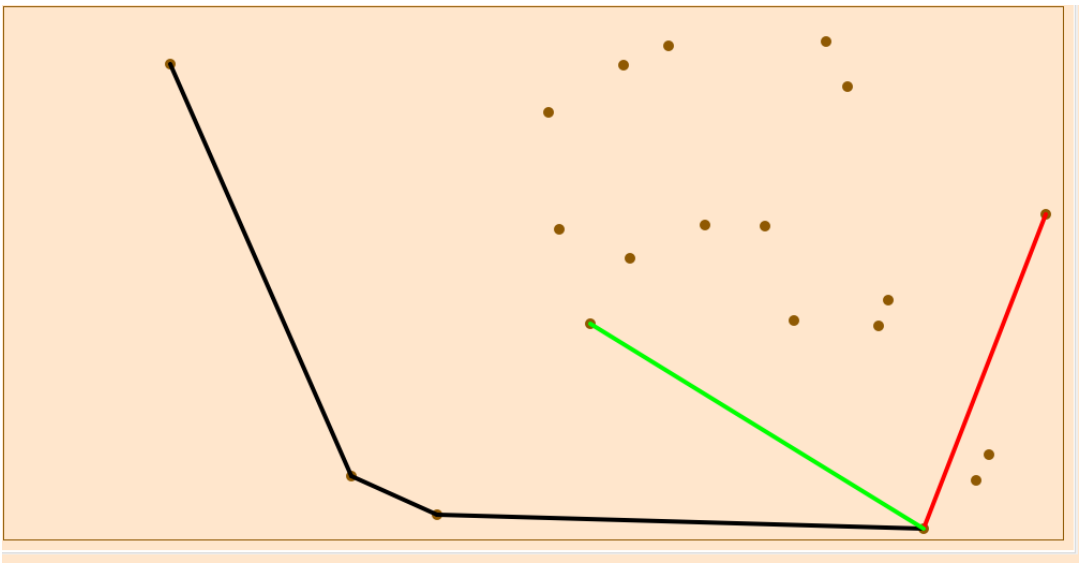
```
        for (m_workingNode = _points.begin(); m_workingNode != _points.end(); ++m_workingNode) {
            if ( (PointALreadyIn(m_result, *m_workingNode) && m_workingNode!=IteratorOfStart) || m_workingNode == m_potentialNode )
                continue;
            if (GetPointSide(m_result.back(), *m_potentialNode, *m_workingNode) == Sides::LEFT){
                m_potentialNode = m_workingNode;
            }

        }
        if (!PointALreadyIn(m_result, *m_potentialNode) || m_potentialNode == IteratorOfStart){
            m_result.push_back(*m_potentialNode);

        }
    }while(m_result.front() != m_result.back());
```

Prikaz jezgra algoritma i trenutak odlučivanja da li ćemo promeniti potencijalni (*m_potentialNode*) čvor omotača za novi (*m_workingNode*).

Rezultat se smešta u *m_result*. Na slici su prikazane i dodatne provere za preskakanje određenih tačaka (tačke za koje je već utvrđeno da su delovi konveksnog omotača).

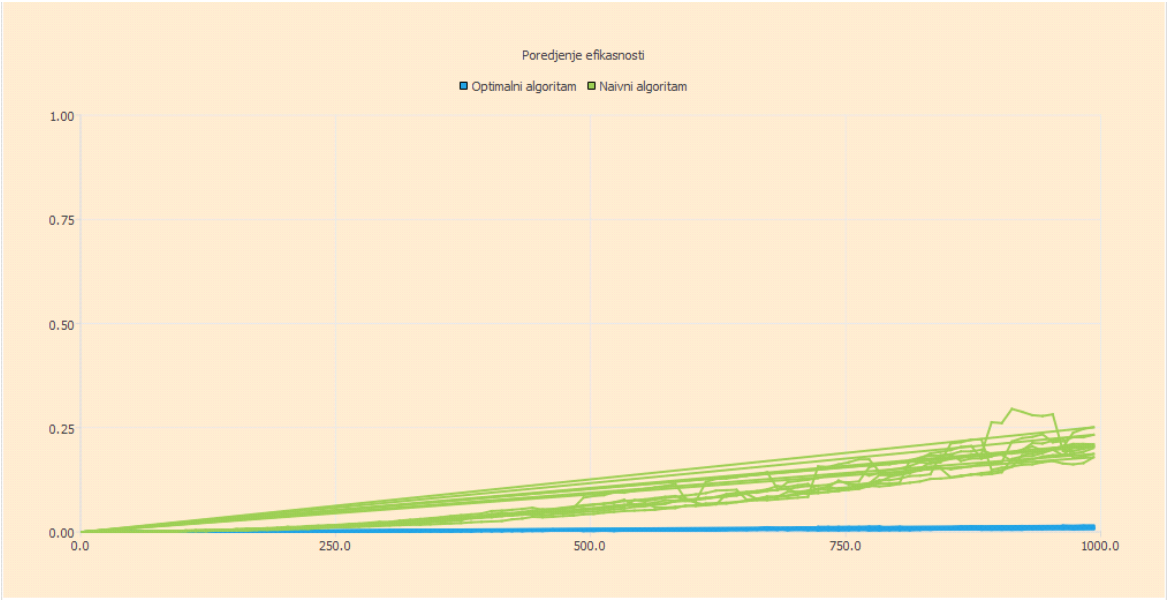
Vizuelizacija algoritma

Prikaz algoritma sa trenutnim potencijalnim novim čvorom (čvor povezan crvenom linijom sa konveksnim omotačem), dosadašnjim pronađenim (crnom) i “radnim” čvorom (zelenom).



Poredjenje efikasnosti naivnog i naprednog algoritma

Na slici se nalazi prikaz odnosa naprednog i naivnog algoritma



Primećujemo da za desetine nasumičnih ulaza koji idu i do 1000 tačaka algoritam ne gubi na svojoj efikasnosti (ili je taj gubitak zanemarljivo mali).

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 50 200 50 200 200 50 200	Poklapanje rezultata oba algoritma
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	Poklapanje rezultata oba algoritma

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