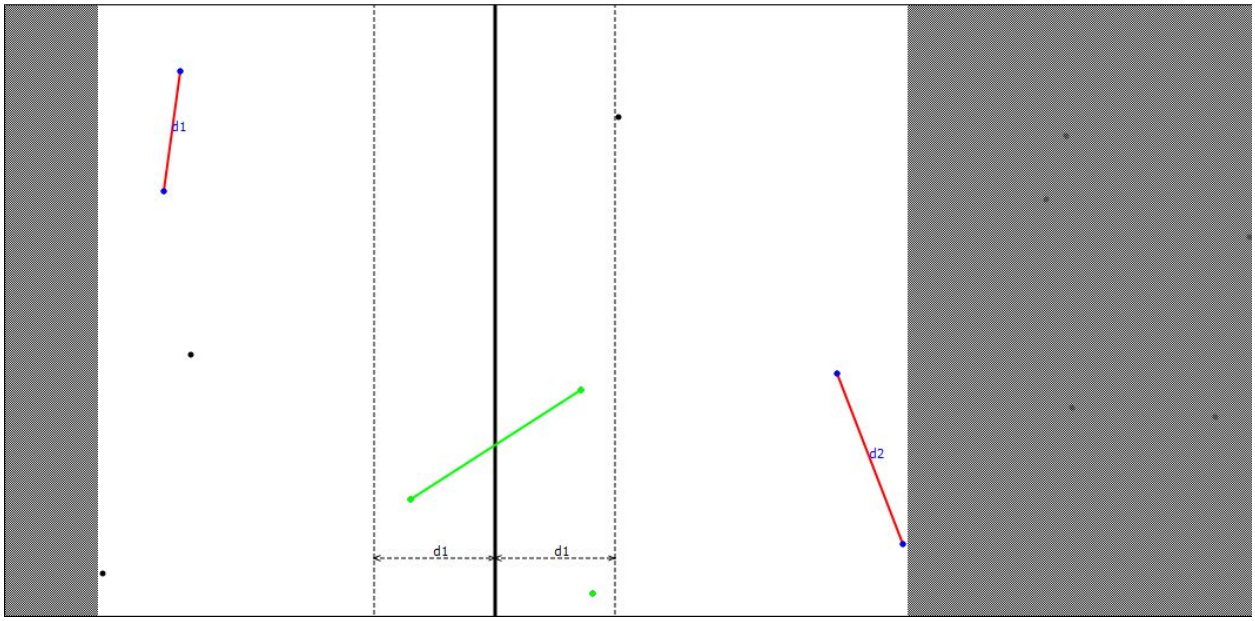
Geometrijski algoritmi @ MATF

Algoritam za određivanje dve najbliže tačke u ravni

Miloš Samardžija



Opis problema

Neka je dat skup od n tačaka u ravni. Potrebno je odrediti dve najbliže tačke iz zadatog skupa. Kao funkcija rastojanja, koristi se euklidsko rastojanje između tačaka. Ukoliko rešenje nije jedinstveno, vratiti bilo koji par tačaka koji zadovoljava kriterijum.

Ulaz: *skup od n tačaka u ravni*

Izlaz: *dve tačke sa najmanjim rastojanjem*

Naivno rešenje problema

Naivno rešenje definisanog problema je intuitivno, i podrazumeva izračunavanje rastojanja za svaki par tačaka iz skupa, i pronalaženje minimuma među njima. Složenost algoritma je $O(n^2)$ i sledi iz ukupnog broja parova tačaka koje treba ispitati ($\frac{(n-1)*n}{2}$). Pseudokod je prikazan na slici 1.

```
1:  procedure NajblizeTacke(tacke)
2:      minTacke = (tacke[0], tacke[1])
3:      minRast = dist(minTacke.p1, minTacke.p2)
4:      for i = 0 to tacke.size() − 1 do
5:          for j = i + 1 to tacke.size() do
6:              if dist(tacke[i], tacke[j]) < minRast then
7:                  minTacke = (tacke[i], tacke[j])
8:                  minRast = dist(minTacke.p1, minTacke.p2)
9:              end if
10:         end for
11:     end for
12:     return minTacke
13: end procedure
```

Slika 1: Naivni algoritam za pronalaženje dve najbliže tačke u ravni

Optimalni algoritam

Optimalni algoritam je zasnovan na strategiji podeli pa vladaj (*eng. divide and conquer*) i podrazumeva podelu problema na dva manja potproblema. Pod pretpostavkom da umemo da rešimo dva problema manjih dimenzija, vršimo spajanje rešenja dva potproblema u rešenje koje je optimalno za ceo problem.

Ukoliko ovo prevedemo u termine definisanog problema, ideja optimalnog algoritma bi bila sledeća: podeli zadati skup tačaka na dva jednaka dela (deljenje se vrši na osnovu vrednosti x koordinata tačaka), pronađi rešenja levog i desnog podskupa, a onda dobijena rešenja iskoristi za pronalaženje glavnog rešenja.

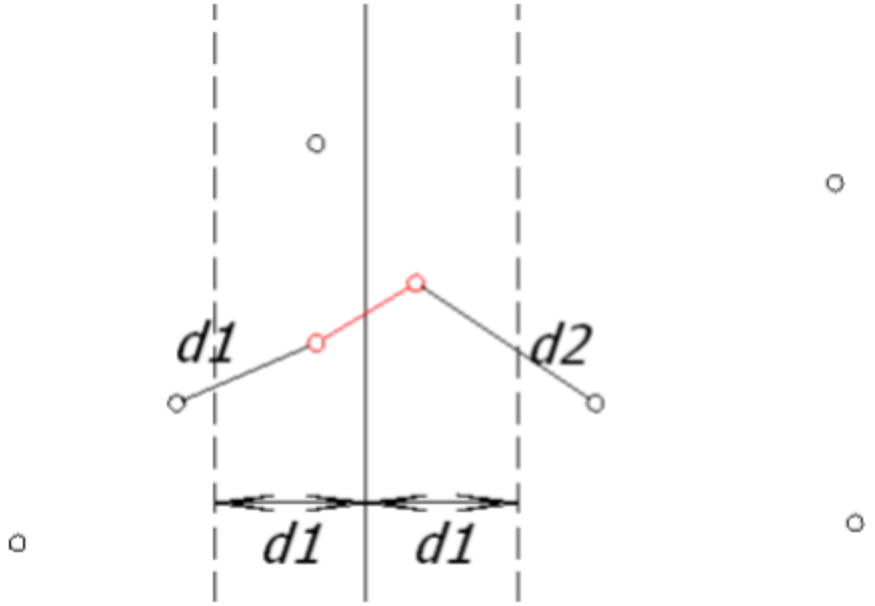
Postavlja se sledeće pitanje: da li umemo da rešimo problem manjih dimenzija? Odgovor je potvrdan. Deljenjem skupa tačaka na dva (približno[1]) jednaka dela dovoljan broj puta, dolazi se do skupova za čije dimenzije trivijalno možemo da odredimo rešenje (bazni slučajevi sa 2 ili 3 tačke). Problem koji preostaje je kako da od rešenja za dva potproblema dobijemo rešenje za problem veće dimenzije. Za trenutak pretpostavimo da je ovo moguće, i pogledajmo rekurentnu relaciju ovog algoritma, kako bismo mogli da diskutujemo o ciljanoj složenosti za optimalni algoritam.

Rekurentna relacija algoritma je: $T(n) = 2T(\frac{n}{2}) + O(f(n))$. Videli smo da je asimptotska složenost naivnog rešenja kvadratna. Pitamo se da li je moguće postići bolju složenost. Pogledajmo malo detaljnije šta je potrebno uraditi kako bismo spojili rešenja dva potproblema u jedno optimalno. Prva ideja do koje se dolazi je da kao rešenje uzmemo rešenje sa manjim rastojanjem, odnosno, ako pretpostavimo da su $d1$ i $d2$ rešenja prvog i drugog potproblema, respektivno, onda kao rešenje većeg problema uzimamo $min(d1, d2).$ Međutim, vrlo brzo dolazimo do zaključka da se u optimalnom rešenju jedna tačka može nalaziti u prvom podskupu, a druga u drugom podskupu. Dakle, prilikom spajanja rešenja, potrebno je razmatrati i parove tačaka između dva podskupa. To nas dovodi do zaključka da je za spajanje dva rešenja u jedno potrebno ispitati $\frac{n^2}{4}$ parova. Na osnovu master teoreme, zaključujemo da će složenost ovakvog rešenja i dalje biti kvadratna.

Da li imamo pametnije rešenje? Ukoliko malo bolje razmislimo, videćemo da nije potrebno proveravati sve parove tačaka iz prvog i drugog podskupa. Neka je x_{max} tačka sa maksimalnom x kordinatom u levom[2] podskupu, i neka je x_{min} tačka sa minimalnom x koordinatom u desnom podskupu. Tada vrednost mid definišemo kao $\frac{x(x_{min}) + x(x_{max})}{2}$. Ukoliko postoji rešenje koje je optimalnije od $d = min(d1, d2)$, onda jedna tačka pripada skupu $L = \{x \in X \mid x(x) \in (mid - d, mid]\}$, a druga pripada $D = \{x \in X \mid x(x) \in [mid, mid + d)\}$, gde je X početni skup. Odnosno, ovo geometrijski možemo posmatrati kao "traku" širine $2d$, koja je ilustrovana na slici 2.

[1] U slučaju neparnog broja tačaka u skupu, možemo definisati da drugi podskup dobija jednu tačku više u odnosu na prvi podskup.
[2] Tačka zadovoljava relaciju "biti u levom podskupu" ukoliko se nalazi među prvih n/2 tačaka u skupu tačaka sortiranih rastuće po vrednosti x koordinate.

Zašto je dovoljno proveravati samo parove tačaka iz skupova L i D? Pretpostavimo suprotno, odnosno, imamo tačku P levog podskupa koja se ne nalazi u L, i tačku Q desnog podskupa koja se nalazi u D (ali i ne mora), i one čine optimalno rešenje. Onda znamo da je $dist(P, Q) < d$, a tada sigurno važi $x(Q) - x(P) < d$. Međutim, ovo dovodi do kontradikcije, jer smo pretpostavili da se bar jedna od tačaka P i Q ne nalazi u traci širine $2d$, a samim tim je nemoguće da apsolutna razlika vrednosti x koordinata bude manja od d.

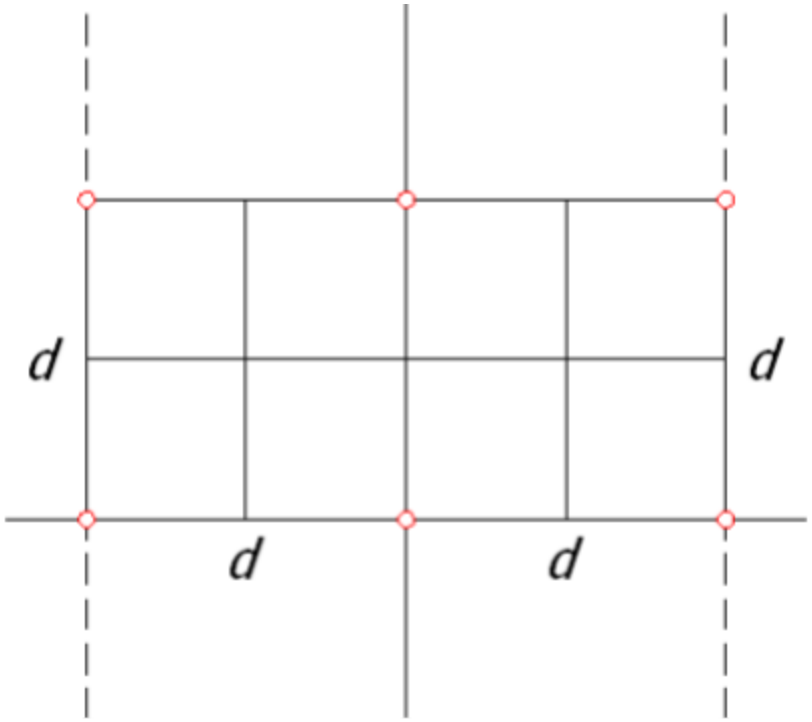


Slika 2: potrebno je proveravati samo parove iz trake dužine 2d

Ovo deluje kao mnogo efikasnije rešenje. U proseku, bolje će se ponašati od kvadratnog algoritma, ali je složenost nažalost i dalje kvadratna, jer će se u najgorem slučaju sve tačke nalaziti u $2d$ traci.

Bilo bi dobro kad bismo osmislili rešenje linearne složenosti, jer bi onda celokupna složenost algoritma bila $O(nlogn)$ čime bismo dobili mnogo efikasniji algoritam. Da li je zaista potrebno da poredimo sve parove u $2d$ traci? Pokušajmo da primenimo sličnu ideju kao sa x koordinatom. Vidimo da tu važi isto tvrđenje: $|y(P) - y(Q)| < d$, ukoliko je par (P, Q) optimalno rešenje. Koristeći ovaj uslov, može se pokazati da će za svaku tačku iz trake biti provereno najviše 7 narednih (tačke sa većom y koordinatom) tačaka, odnosno da

tačka P može biti udaljena od tačke Q najviše 7 tačaka. Formalni dokaz ovog tvrđenja je previše komplikovan, ali se korektnost tvrđenja može videti na slici 3. Formira se 8 kvadrata dimenzija $\frac{d}{2}x\frac{d}{2}$, i u svakom od njih može da se nađe najviše jedna tačka, jer su dimenzije dijagonala $\frac{d}{2}\sqrt{2} < d$, a sve tačke unutar jednog kvadrata se nalaze u tačno jednoj od dve d trake, a znamo da najmanje rastojanje u svakoj od d traka sigurno nije manje od d. Na slici 3 je dat primer gde se u svakom kvadratu nalazi po jedna tačka (tačke koje leže na središnjoj liniji su dvostruke).



Slika 3: Svaki kvadrat sadrži tačno jednu tačku

Preostaje još jedan problem koji treba rešiti. Tačke su sortirane po x koordinatama, ali nama trebaju tačke iz $2d$ trake sortirane po y koordinatama. Ovo se lako rešava proširivanjem induktivne hipoteze: umemo da odredimo najmanja rastojanja u L i D (iznad pronaći definiciju skupova L i D), i umemo da sortiramo tačke u L, i tačke u D.

I konačno, da bismo spojili rešenja dva potproblema u jedno, potrebno je izvršiti spajanje dva sortirana niza u jedan ($O(n)$), izdvojiti elemente iz $2d$ trake ($O(n)$), i za svaki element P_i iz $2d$ trake izračunati rastojanje do P_j $(j > i \wedge y(P_j) - y(P_i) < d)$. Na osnovu tvrđenja znamo da $j - i \leq 7$, a odatle sledi da je složenost ovog koraka takođe O(n).

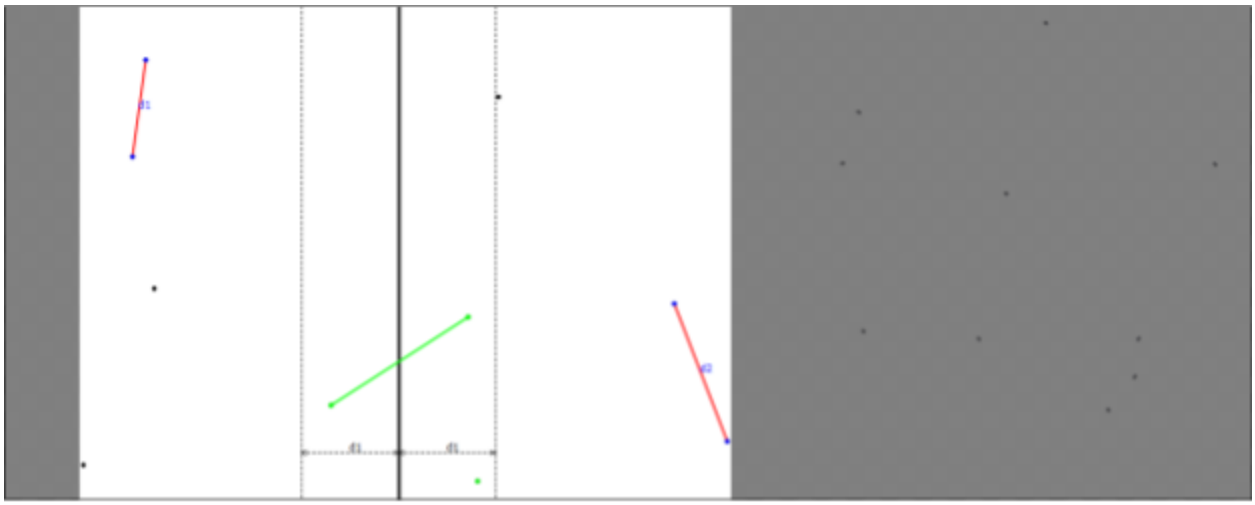
Dakle, ukupna složenost spajanja dva rešenja u jedno je $O(n)$, pa je složenost konstruisanog algoritma $O(nlogn)$. Pseudokod algoritma je prikazan na slici 4. *Sort3* je pomoćna funkcija koja sortira tri tačke, *merge* spaja dva sortirana niza u jedan, *triTacke* vraća dve od tri tačke sa najmanjim rastojanjem i *pronadjiMinUTraci* pronalazi par sa najmanjim rastojanjem u $2d$ traci.

```
1:  procedure NajblizeTacke(tacke, l, r)
2:      if r − l == 2 then
3:          merge(tacke, l, l + 1, r)
4:          return (tacke[l], tacke[l + 1])
5:      else if r − l == 3 then
6:          sort3(tacke, l)
7:          return triTacke(tacke[l], tacke[l + 1], tacke[l + 2])
8:      end if
9:      m = l + (r − l)/2
10:     midLine = tacke[m − 1].x + (tacke[m].x − tacke[m − 1].x)/2
11:     levo = NajblizeTacke(tacke, l, m)
12:     desno = NajblizeTacke(tacke, m, r)
13:     merge(tacke, l, m, r)
14:     d1 = dist(levo.p1, levo.p2)
15:     d2 = dist(desno.p1, desno.p2)
16:     d = min(d1, d2)
17:     kandidati = pronadjiKandidate(tacke, l, r, d, midLine)
18:     traka = pronadjiMinUTraci(kandidati, d)
19:     return triTacke(levo, desno, traka)
20: end procedure
```

Slika 4: Optimalni algoritam za pronalaženje dve najbliže tačke u ravni

Vizuelizacija algoritma

Na slici 5 je prikazana vizuelizacija optimalnog algoritma. Crvene duži sa plavim temenima predstavljaju rešenja potproblema. Zelena temena u $2d$ traci predstavljaju temena koja treba proveravati prilikom spajanja rešenja potproblema. Zelena duž označava trenutni par tačaka koji se proverava. Potproblemi su podeljeni podebljanom vertikalnom crnom linijom, a tačke koje ne pripadaju trenutnom problemu koji se rešava su u osenčenom delu.



Slika 5: Vizuelizacija algoritma

Poredjenje efikasnosti naivnog i optimalnog algoritma

U tabeli je prikazano vreme (u sekundama) izvršavanja naivnog i optimalnog algoritma za različite dimenzije ulaza.

alg. / dim. ulaza	100	1000	10000	20000	30000
naivni	0.003	0.125	11.112	44.693	101.022
optimalni	< 0.001	0.012	0.075	0.105	0.156

Da bismo stekli pravi osećaj o tome koliko je optimalni algoritam brži od naivnog, dovoljno je reći da za 54 sekunde naivni algoritam obradi malo više od 20000 tačaka, a za to isto vreme optimalni algoritam obradi 8 miliona tačaka.

Testiranje ispravnosti algoritma

Testiranje ispravnosti algoritma, odnosno ispravnosti same implementacije je izvedeno unit testovima (Google Test). Ispravnost i asimptotska vremenska složenost konstruisanog algoritma su dokazani u ovom dokumentu.

Naziv testa	Opis testa	Ulaz	Očekivani izlaz
invalidInput1	Zadavanje praznog niza tačaka. Algoritam neće biti izvršen.	[]	INVALID_INPUT

invalidInput2	Zadavanje samo jedne tačke. Algoritam neće biti izvršen.	[{5, 5}]	INVALID_INPUT
compareOutputs1	Zadavanje nasumičnog niza tačaka dimenzije 30.	Niz dimenzije 30	Poklapanje rezultata naivnog i optimalnog algoritma
compareOutputs2	Zadavanje nasumičnog niza tačaka dimenzije 300.	Niz dimenzije 300	Poklapanje rezultata naivnog i optimalnog algoritma
twoPoints	Zadavanje dve tačke. Testira se bazni slučaj.	[{569, 318}, {503, 312}]	{{503, 312}, {569, 318}}
threePoints	Zadavanje tri tačke. Testira se bazni slučaj.	[{715, 230}, {505, 320}, {710, 340}]	{{715, 230}, {710, 340}}
multipleSolutions	Zadavanje niza tačaka u kojem postoji više rešenja.	[{615, 330}, {405, 320}, {680, 250}, {550, 420}]	{{680, 250}, {615, 330}}
randomTest	Nasumičan niz tačaka.	[{410, 215}, {560, 450}, {700, 550}, {525, 305}, {560, 350}, {415, 250}, {102, 421}, {132, 312}]	{{410, 215}, {415, 250}}
minLeft	Minimalno rastojanje se nalazi u levom potproblemu.	[{530, 135}, {315, 401}, {475, 360}, {695, 360}, {310, 365}, {560, 370}]	{{310, 365}, {315, 401}}
minRight	Minimalno rastojanje se nalazi u desnom potproblemu.	[{470, 335}, {310, 375}, {425, 390}, {745, 105}, {210, 475}, {750, 165}]	{{745, 105}, {750, 165}}
minCenter	Minimalno rastojanje se nalazi u $2d$ traci.	[{530, 355}, {630, 335}, {110, 375}, {505, 360}, {545, 160}, {140, 265}, {550, 275}]	{{530, 355}, {505, 360}}