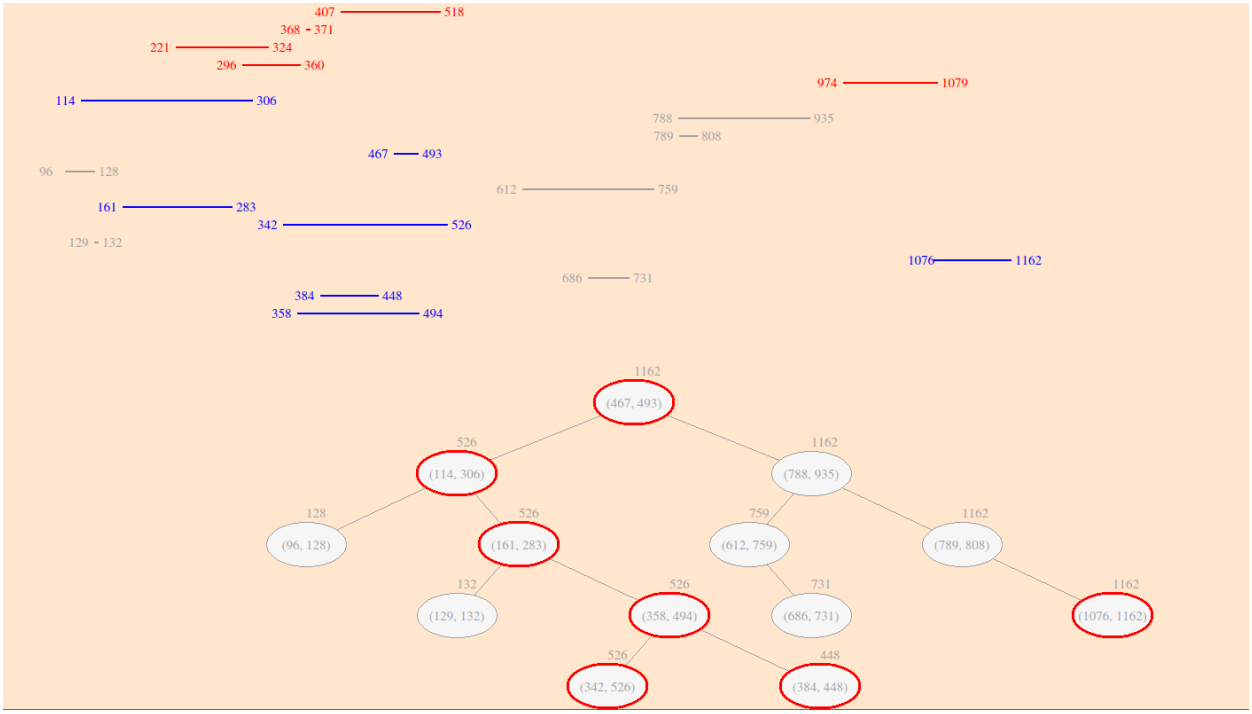
Geometrijski algoritmi @ MATF

Algoritam za određivanje preseka duži na pravoj korišćenjem interval search tree strukture

Ozren Demonja



Opis problema

Na horizontalnoj pravoj se nalazi skup duži koje su zadate levim i desnim temenom. Za proizvoljan skup duži koje pripadaju datoj pravoj, imaju istu y koordinatu kao i prethodni skup duži, potrebno je pronaći sve duži iz skupa sa kojima se one seku.

Ulaz: *skup A koji čine n duži na pravoj i skup B koji čine m duži na pravoj za koje se traži presek*

Izlaz: *skup duži iz A sa kojima duži iz skupa B imaju presek*

Naivno rešenje problema

Naivno rešenje se svodi na proveravanje preseka svake duži iz A sa svakom duži iz B. Duži l1 i l2 imaju presečnih tačaka ako važi da je levo teme l1 ≤ od desnog temena l2 i obrnuto. Složenost ovakvog pristupa je $O(nm)$.

Pseudokod:

```
Algoritam:    PronađiPresek(A, B)

1:  E := {}
2:  za svaku duž l1 iz A
3:      za svaku duž l2 iz B
4:          ako se l1 i l2 seku onda
5:              dodaj duž l1 u E
6:  return E
```

Algoritam zasnovan na stablu pretrage intervala

Algoritam zasnovan na stablu pretrage intervala koristi strukturu podataka koja se naziva stablo pretrage duži (engl. *interval search tree*). Ova struktura podataka je predstavljena binarnim stablom pretrage. U svakom čvoru stabla čuva se jedna duž zadata svojim levim i desnim krajem. Čvorovi unutar stabla su sortirani po levom kraju duži. Pored informacija o duži u čvoru se još čuva i podatak o maksimalnom desnom temenu u podstablu.

Postoji više vrste binarnih stabala pretrage. Ukoliko se koristi klasično binarno stablo pretrage tj. stablo koje nije balansirano u najgorem slučaju imamo vremensku složenost $O(n)$ za pronalaženje preseka sa jednom duži što odgovara složenosti pristupa grubom silom. Iz tog razloga se koriste samobalansirajuća stabla. To su čvor-zasnovana binarna stabla pretrage koja automatski održavaju svoju visinu malom. Postoji više vrsta samo samobalansirajućih stabala. U radu se koriste crveno-crna stabla (engl. *red-black tree*) koja će biti opisana u nastavku.

Crveno-crno stablo

Crveno-crno stablo je binarno stablo pretrage koje garantuje da broj čvorova na najdužem putu od korena do jednog lista stabla nikada nije veći od dvostrukog broja čvorova na najkraćem putu od korena do jednog lista stabla. Drugim rečima, imaćemo visinu koja je uvek $\leq 2\log_2(n+1)$ gde je n broj čvorova. Ovo nam je bitno jer na taj način imamo logaritamsku složenost dodavanja, brisanja i pretrage. Zbog ovih osobina koristimo crveno-crno stablo kao osnovu za stablo pretrage duži.

U crveno-crnom stablu, svaki čvor čuva u sebi informacije o jednoj duži, maksimalnom desnom temenu u podstablu, boji, roditelju, levom i desnom sinu. Čvorovi su međusobno sortirani po levim krajevima duži.

Da bi se u stablo dodao novi čvor, potrebno je prvo izvršiti balansiranje. Da bi crveno-crno stablo bilo balansirano narednih 5 uslova mora biti ispunjeno:

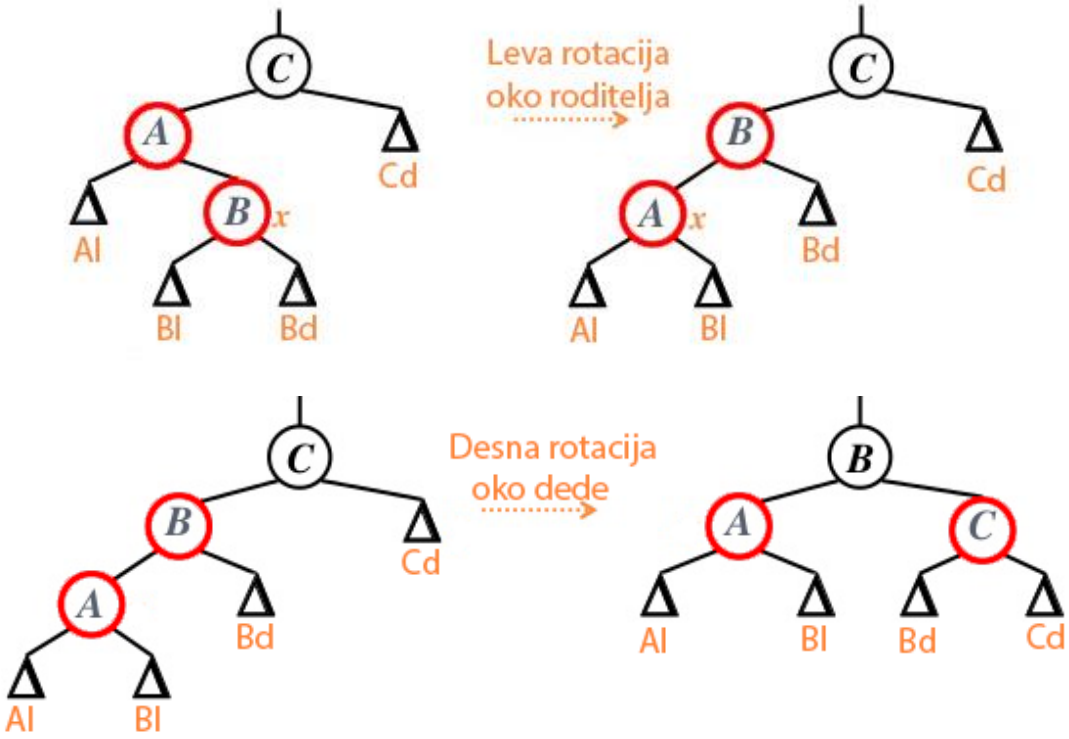
1. Svaki čvor stabla je ili crven ili crn
2. Koren stabla je crn
3. Svaki list (nul-čvor) je crn
4. Ni jedan crveni čvor nema crvene dece
5. Broj crnih čvorova na svakom putu od jednog čvora stabla do njemu pripadajućih listova je jednak

Kada smo opisali pravila, prelazimo na dodavanje čvora. Kada dodajemo novi čvor prvo proveravamo da li je to upravo koren stabla. Ukoliko jeste bojimo ga u crno i čuvamo kao koren. Ukoliko nije spuštamo se u stablu od korena ka listovima modifikujući usput ukoliko je potrebno informaciju o maksimalnom desnom temenu u podstablu. Kada smo stigli do listova dodajemo novi cvor, bojimo ga u crveno. Pošto novododati čvor nema dece postavljamo njegovo desno teme za vrednost maksimalnog temena u njegovom desnom podstablu.

Kada je novi čvor dodat postoji mogućnost da je narušeno neko od pravila crveno crnih stabala zbog toga se posle svakog dodavanja vrši provera zadovoljivosti uslova. Ukoliko je narušeno neko pravilo, čvorovi se ponovo boje ili se vrši njihova rotacija.

Ukoliko je stric (na slici D) novog dodatog čvora crven, onda imamo jednostavniju situaciju. Potrebno je samo promeniti boju strica i oca (na slici A) u crno i proveriti da li je deda (na slici C) koren ili nije. Ukoliko je deda koren onda se on ne modifikuje, a ukoliko nije onda se on boji u crveno i rekurzivno poziva funkcija za dedu.

Ukoliko stric novog dodatog čvora nije crven, onda je potrebno izvršiti određenu rotaciju oko dede. Ukoliko je roditelj levo dete dede, a novi dodati čvor desno dete roditelja potrebno je prvo rotirati ulevo oko roditelja a zatim udesno oko dede.



Nakon izvršenih rotacija potrebno je modifikovati informaciju o maksimalnom desnom temenu u podstablu sva tri aktera.

Ukoliko je novi dodati čvor levo dete roditelja potrebno je izvršiti samo desnu rotaciju oko dede. Nakon izvršenih rotacija potrebno je modifikovati informaciju o maksimalnom desnom temenu u podstablu oba aktera. Simetrično važi i za slučajeve kada je roditelj desno dete dede, a čvor levo ili desno dete roditelja.

Pseudokod leve i desne rotacije:

```
LEFT-ROTATE(T, x)
y ← x->right
    x->right← y->left
  y->left->p ← x
y->p ← x->p

  if x->p = Null
    then T->root ← y

   else if x = x->p->left
     then x->p->left ← y

   else  x->p->right ← y

y->left ← x
x->p ← y
```

```
RIGHT-ROTATE(T, x)
y ← x->left
      x->left← y->right
    y->right->p ← x
 y->p ← x->p

  if x->p = Null
       then T->root ← y

  else if x = x->p->right
       then x->p->right ← y

   else  x->p->left ← y

 y->right ← x
x->p ← y
```

Runtime : O(1)

Pošto je složenost rotacija konstantna a ubacivanja novog čvora $O(\log(n))$ složenost konstrukcije stabla je $O(\text{n } \log(n))$. Kada smo opisali ubacivanje novih duži u strukturu ostaje nam još da opišemo pronalaženje preseka.

Algoritam za traženje preseka zasnovan na stablu pretrage duži

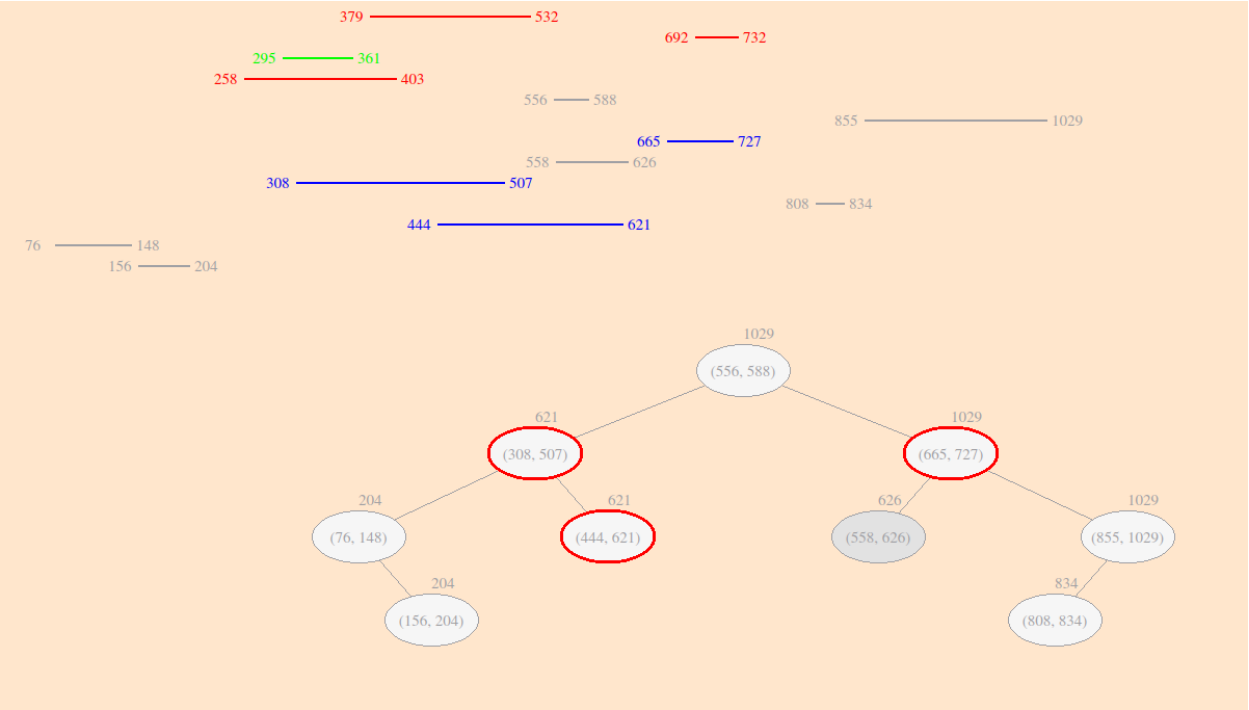
Za svaku od m duži iz skupa B se proverava da li postoji presek. Kada se za datu duž (l, d) iz skupa B proverava da li postoji presek sa skupom duži A, prvo se proverava da li duž seče duž u korenu stabla. Zatim se provera da li je l veće od maksimuma koji se čuva u levom podstablu. Ako jeste ide se u desno podstablo, ako nije ide se u levo podstablo. Kad se ode u desno podstablo, nema potrebe da se ispituje i levo podstablo naknadno. Ukoliko se u levom podstablu ne pronađe presek, takođe nema potrebe naknadno ispitivati i desno podstablo. Ovo je ispravno jer ako je l veće od maksimuma u levom podstablu znači da tamo nema preseka jer počinje posle kraja svih, a ako se ne pronađe u levom podstablu znači da se završava pre početka svih duži. Složenost ovakvog pristupa je $O(\mathrm{k} + \log(n))$ gde je k broj pronađenih preseka što znači da je ovaj algoritam izlazno-zavisan.

Pseudokod:

```
Algoritam:    PronađiPresek(koren, duž)

1:  E := 0
2:  ako je koren null
3:      vrati se
3:  ako postoji presek korena i duži
4:      dodaj koren.duž u E
5:  ako postoji levi sin korena i njegov maksimum je je veći od leve koordinate duži
6:      idi rekurzivno levo
7   ako prethodni uslov nije ispunjen ili je u levom podstablu pronadjen presek
8:      idi rekurzivno desno
```

Vizuelizacija algoritma



U gornjem delu slike vidimo sivim obojene duži. To su duži koje se nalaze na pravoj i sa kojima nije pronađen presek
Plavo obojene duži su one koje se nalaze na pravoj i sa kojima je pronađen presek
Crven obojene duži su one za koje tražimo preseke sa skupom duži.
Zeleno obojena duž je ona duž za koju se u tom trenutku traži presek sa skupom duži.
U donjem delu slike prikazano je stablo pretrage duži maksimalne dubine 4.
Čvorovi koji nisu podebljani crvenom elipsom su oni čvorovi za koje ne postoji presek.
Čvorovi koji su podebljani crvenom elipsom su oni čvorovi za koje je pronađen postoji presek.
Čvorovi koji je obojeni tamnijom bojom je onaj čvor na kome se trenutno vrši pretraga.

Poredjenje efikasnosti naivnog i optimalnog algoritma

U tabeli je prikazano vreme u sekundama izvršavanja naivnog i optimalnog algoritma pri različitim dimenzijama ulaza. Redovi koji su obojeni u sivo (parni redovi) predstavljaju mere naivnog algoritma (u tabeli Naivni alg.), a beli mere optimalnog (u tabeli Optimalni alg.). Prva dimenzija (u tabeli n/vertikala) predstavlja broj duži za koje se ispituje dok druga(u tabeli m/horizontala) predstavlja broj duži na pravoj.

n / m	algoritam	100	1.000	10.000	60.000	100.000
10	Optimalni alg.	4e-05	0.000384	0.004498	0.049147	0.09023
	Naivni alg.	4.1e-05	0.000325	0.004112	0.044451	0.092631
100	Optimalni alg.	0.000121	0.000922	0.010999	0.118369	0.223629
	Naivni alg.	0.000265	0.001563	0.016421	0.26822	0.43638
1.000	Optimalni alg.	0.000862	0.010005	0.082637	0.880428	1.60528
	Naivni alg.	0.002878	0.012969	0.229316	1.82408	3.42786
5.000	Optimalni alg.	0.004641	0.035047	0.354817	2.98162	8.30512
	Naivni alg.	0.007309	0.078188	0.847073	7.70152	18.5567

Iz tabele vidimo da je struktura stablo pretrage duži lošija kada postoji veliki broj duži na pravoj a mali broj duži za koje se ispituje. Kako se situacija menja tako struktura pokazuje svoju moć.

Testiranje ispravnosti algoritma

Testiranje ispravnosti algoritma izvedeno je jediničnim testovima korišćenjem Google Test okvira. Prvi argument ulaza su duži na pravoj, a drugi duž za koje je potrebno odrediti presek.

Naziv testa	Opis testa	Ulaz	Očekivani izlaz
emptyInput1	Zadavanje ulaza koji nije ispravan. Program ispisuje poruku o grešci, a rezultujući niz treba da bude prazan.	[], []	0
emptyInput2	Zadavanje neispravnih presečnih duži. Program ispisuje poruku o grešci, a rezultujući niz treba da bude prazan.	Niz dimenzije 20, []	0

emptyInput3	Zadavanje neispravnih duži. Program ispisuje poruku o grešci, a rezultujući niz treba da bude prazan.	[], Niz dimenzije 10	0
duplicateLines1	Zadavanje ulaza gde se neke duži javljaju više puta. Algoritam će ih tretirati kao različite duži.	[{10, 100}, {8, 18}, {10, 100}, {111, 158}, {24, 100}], [{8, 15}, {19, 128}, {157, 190}]	8
duplicateLines2	Zadavanje ulaza gde se neke duži javljaju više puta. Algoritam će ih tretirati kao različite duži.	{10, 100}, {8, 18}, {15, 100}, {111, 158}, {24, 100}], [{8, 15}, {19, 128}, {8, 15}]	10
duplicateLines3	Zadavanje ulaza gde se neke duži javljaju više puta. Algoritam će ih tretirati kao različite duži.	[{10, 100}, {8, 18}, {10, 100}, {111, 158}, {24, 100}], [{8, 15}, {19, 128}, {8, 15}]	10
noIntersection1	Zadavanje presecajuće duži koja se nalazi iza svih duži na ravni. Rezultirajući niz bi trebao biti prazan.	[{10, 100}, {8, 18}, {15, 80}, {5, 14}, {24, 28}], [{1, 4}]	0
noIntersection2	Zadavanje presecajuće duži koja se nalazi između svih duži na ravni. Rezultirajući niz bi trebao biti prazan.	{10, 15}, {8, 14}, {18, 80}, {5, 14}, {20, 28}], [{16, 17}]	0
noIntersection3	Zadavanje presecajuće duži koja se nalazi ispred svih duži na ravni. Rezultirajući niz bi trebao biti prazan.	[{10, 27}, {8, 18}, {15, 17}, {5, 14}, {24, 28}], [{29, 34}]	0
ascOrderInsert	Popunjavanje stabla u rastućem poretku. Algoritam bi trebao normalno da se izvršava	{5, 14}, {8, 18}, {10, 100}, {15, 80}, {24, 28}, {25, 29}], [{29, 34}]	3
descOrderInsert	Popunjavanje stabla u rastućem poretku. Algoritam bi trebao normalno da se izvršava	{35, 48}, {28, 38}, {25, 29}, {21, 47}, {14, 80}, {10, 19}, {1, 10}], [{29, 34}]	4

randomInput1	Zadavanje nasumičnog niza linija veličine 5 i 20	Niz dimenzije 5, Niz dimenzije 20	Poklapanje rezultata naivnog i optimalnog algoritma
randomInput2	Zadavanje nasumičnog niza linija veličine 10 i 50	Niz dimenzije 10, Niz dimenzije 50	Poklapanje rezultata naivnog i optimalnog algoritma
randomInput3	Zadavanje nasumičnog niza linija veličine 20 i 100	Niz dimenzije 20, Niz dimenzije 100	Poklapanje rezultata naivnog i optimalnog algoritma
randomInput4	Zadavanje nasumičnog niza linija veličine 20 i 200	Niz dimenzije 20, Niz dimenzije 200	Poklapanje rezultata naivnog i optimalnog algoritma