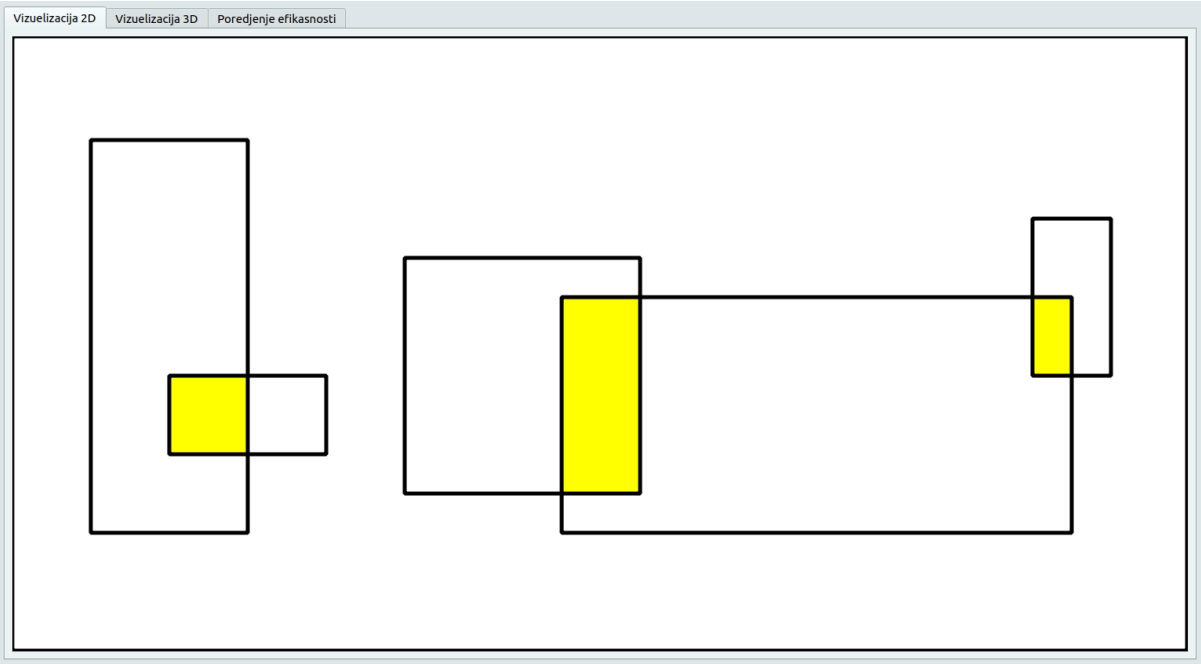
Geometrijski algoritmi @ MATF

Algoritam za određivanje prese-cajućih parova pravougaonika sa stranicama paralelnim osama

Lazar Vasović, 1011/2020



Opis problema

Pravougaonik je primer četvorougaone geometrijske figure u ravni. Naspramne stranice svakog pravougaonika su paralelne i jednake dužine, dok su susedne stranice normalne jedna na drugu, što znači da zaklapaju prav ugao. Posebno zanimljivu klasu pravougaonika čine oni čije su stranice paralelne osama. Njima se mogu predstaviti komponente mikroprocesora, a mogu poslužiti i kao apstrakcija bilo kog fizičkog objekta u ravni (obuhvatajući pravougaonik, engl. *bounding rect* ili *box*), što ima veliku primenu u računarskoj grafici, ali i bazama podataka. U svim primenama, značajno je odrediti postoji li presek (kolizija) ovih figura, npr. postoji li mogućnost da su se dva objekta sudarila ili da je mikroprocesor nekorektno dizajniran. Upravo je taj problem obrađen u ovom projektu.

Ulaz: *niz od n pravougaonika sa stranicama paralelnim osama*

Izlaz: *niz od k parova pravougaonika između kojih postoji neprazan presek*

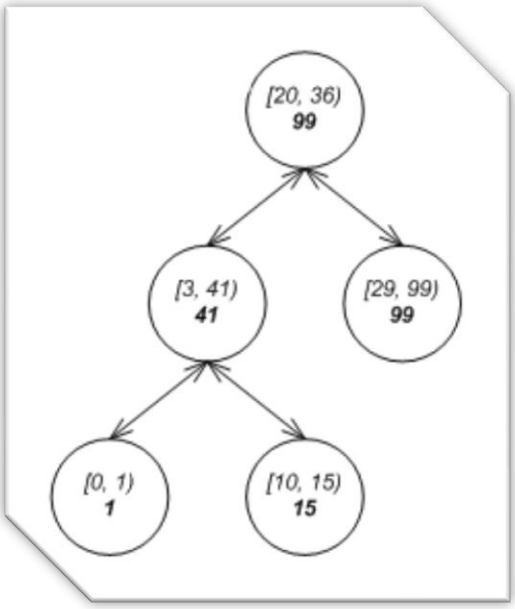
Algoritam grube sile

Naivni algoritam, koji ne zahteva previše razmišljanja, zasniva se na proveri svakog para pravougaonika metodom grube sile. Preciznije, dovoljno je proveriti svaku dvočlanu kombinaciju, pošto je „seku se“ primer refleksivne i simetrične binarne relacije. To znači da je broj provera $\binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$ za n ulaznih pravougaonika, što je u neku ruku i optimalno, u slučaju da se svi elementi seku. Ne računajući niz za skladištenje rezultata, ne postoji potreba za dodatnim prostorom.

Metod brišuće prave

Pametnije rešenje, u ovom projektu uzeto za „naivni“ algoritam, odnosno algoritam na čije se nadmašivanje po učinku cilja, zasnovano je na metodu brišuće prave (pokretne linije). Konceptualno, zamišlja se horizontalna prava koja se kreće tako što joj vremenom opada y koordinata. U svakom trenutku je poznat status te prave, koji odgovara skupu pravougaonika koje ona trenutno seče. Nakon svake promene statusa, neophodno je proveriti da li se novododati pravougaonik seče sa prethodno prisutnima (ukoliko je status proširen) ili jednostavno nastaviti dalje (ukoliko je status sužen).

Realizacija ovog agoritma zasnovana je na balansiranim stablima pretrage. Preciznije, koriste se samobalansirajuća crveno-crna drveta, i to dva – jedno kao status, a drugo kao red događaja. Status je zapravo intervalno stablo, te u čvorovima skladišti pravougaonike kao x intervale, koji kao metapodatak čuvaju maksimalnu desnu granicu podstabla. Red događaja skladišti pravougaonike kao y pozicije (svaki dvaput) u nerastućem poretku, sa glavnim ciljem soritranja. To služi kao model kretanja brišuće prave, koja zastaje pri nailasku na gornje i donje stranice pravougaonika, čime postaje poznato koji tačno pravougaonik u nekom koraku treba dodati u status ili pak izbaciti iz njega.



Slika 1. Status kao intervalno stablo. U drvetu su pravougaonici predstavljeni $\boldsymbol{x}$ intervalima, dok je desni maksimum metapodatak. Kako se nalaze u statusu, svi se seku po $\boldsymbol{y}$ dimenziji, čime je problem redukovan.

Primer statusa dat je na slici 1. Ovako odabrana struktura podatka garantuje performanse ne gore od kvadratnih – cenu pravljenja $O(n \log n)$, zapravo se cenom svakog pojedinačnog dodavanja i brisanja $O(\log n)$, i cenu upita $O(\log n + k)$, pri čemu je k broj pronađenih intervala (traže se intervali tj. njima predstavljeni pravougaonici koji se seku sa novododatim). Kako je i red događaja slična struktura, i njegovo popunjavanje i iteracija kroz njega uzimaju vreme $O(n \log n)$, čime je ukupna vremenska složenost izloženog algoritma $O(n \log n + k)$. Ovo je primer pristupa zavisnog od izlaza, tako da u složenosti figurira unapred nepoznati izlazni parametar, što je ovde broj preseka koji se traže. Već je iz analize grube sile poznat red ukupnog broja preseka $k \leq O(n^2)$, što je onda dominantna komponenta složenosti. Ipak, to podrazumeva da se svaki ulazni pravougaonik seče se sa skoro svakim drugim, što je retkost u realnim primenama, u kojima dominira komponenta $O(n \log n)$, što sveukupno ovaj algoritam čini boljim po performansama (odzivu) od algoritma grube sile.

Strategija podeli pa vladaj

Rešenje u neku ruku slično prethodnom zasnovano je na strategiji podeli pa vladaj, a 2010. su ga predložili naučnici Frank Devai (*Frank Dévai*) i Laslo Nojman (*László Neumann*) i predstavili na IEEE konferenciji (*2010 10th IEEE International Conference on Computer and Information Technology*) kroz rad „A Rectangle-Intersection Algorithm with Limited Resource Requirements“. Ideja je da se u svakom koraku algoritma prostor pretrage preseka podeli na dva približno jednaka potprostora, pri čemu su elementi podele vertikalne ivice pravougaonika sortirane po x koordinati.

Inicijalizacija algoritma sastoji se od popunjavanja niza V vertikalnih ivica i njima pridruženih pravougaonika, nakon čega se ovaj niz sortira neopadajuće, što je vremenska složenost $O(n \log n)$. Vertikalne ivice implementirane su kao par tipa ivice (leva ili desna) i pokazivača na odgovarajući pravougaonik. Dodatno se inicijalizuje niz H pravougaonika (isto pokazivača) i sortira neopadajuće po y koordinati donje stranice. Kao i prethodni, i ovaj niz je dužine $2n$ (zasad ima samo n pravougaonika, ali je dodatni kapacitet neophodan za kasnije particionisanje u linearnom vremenu), tako da popunjavanje i sortiranje isto uzimaju $O(n \log n)$ vremena, što je zasad i ukupno vreme.

Rekurzivna procedura za otkrivanje preseka deli prostor na dva podskupa V_1 i V_2, s tim što se zbog očuvane sortiranosti to prosto radi pomoću indeksa, u konstantnom vremenu, bez dodatnog prostora. Prolaskom kroz podskupove mogu se pronaći četiri skupa kandidata za brz pronalazak svih preseka:

- S_{11} – pravougaonici koji su sasvim levo od tačke podele (desna stranica im je levo),
- S_{22} – pravougaonici koji su sasvim desno od tačke podele (leva stranica im je desno),
- S_{12} – pravougaonici koji su levo od tačke podele i obuhvataju desni potprostor (leva stranica im je levo, a desna prevazilazi najdešnju ivicu desnog potprostora ili je to ta ivica),
- S_{21} – pravougaonici koji su desno od tačke podele i obuhvataju levi potprostor (desna stranica im je desno, a leva prevazilazi najlevlju ivicu levog potprostora ili je to ta ivica).

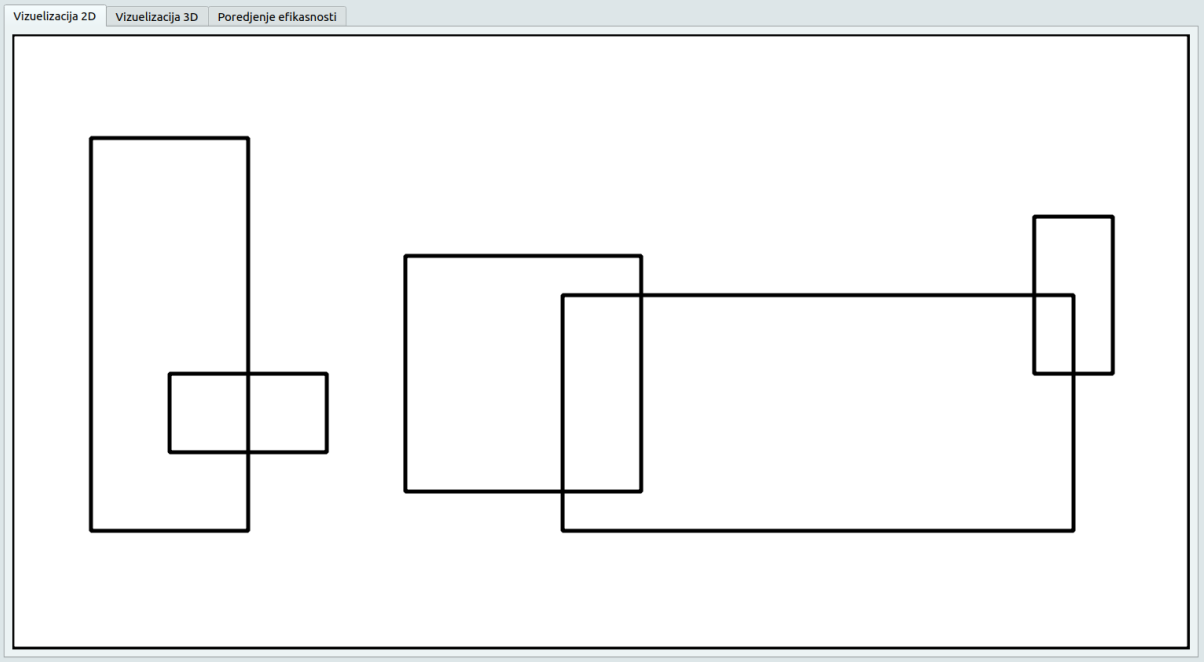
Ovi skupovi su u radu realizovani kao dodatno polje pripadnosti u klasi koja predstavlja pravougaonik, a njihovo popunjavanje dešava se u linearnom vremenu. Prednost ovoga ogleda se u znatnom olakšavanju sledećeg koraka, što je particionisanje niza H na podnizove H_1 i H_2, koji čuvaju sortiranost i preuzimaju ulogu originalnog niza u potprostorima. Kako su u pitanju pokazivači, pristup pripadnosti skupu omogućava particionisanje u linarnom vremenu. Ono se sprovodi tako što se, prolaskom kroz originalni niz H, u levi podniz smeštaju elementi skupova S_{11} i S_{12}, u desni elementi S_{22} i S_{21}, dok se pravougaonici koji nisu dodeljeni nijednom skupu smeštaju u oba podniza. Veličina podnizova ograničena je veličinom potprostora, tako da je vreme izvršavanja svakog rekurzivnog poziva $O(n)$. Jednačina procedure je stoga $T(2n) = 2T(n) + O(n)$, sa rešenjem $O(n \log n)$.

Ostaje još dopunjavanje svakog rekurzivnog poziva proverom skupova kandidata. Potencijalno se seku pravougaonici iz parova skupova S_{12} i S_{22}, S_{21} i S_{11}, S_{12} i S_{21}. Naime, prema definiciji samih skupova, elementi u njima se obavezno seku po x dimenziji, pa je neophodno samo proveriti seku li se i po y. Ovo je moguće vrlo efikasno uraditi zbog uređenja niza H, koji omogućava sveukupno linearni prolaz po broju preseka, po čemu je ovaj algoritam sličan prethodnom, zasnovanom na metodu brišuće prave. Glavna razlika je što se tamo kretalo kroz red događaja, dok je uređenje čuvano u statusu, a ovde se kreće kroz potprostore (štaviše, moguće je niz V kroz rekurzivne pozive shvatiti kao implicitno segmentno stablo), dok se uređenje čuva u nizu koji se efikasno particioniše, odnosno ažurira pred svaki rekurzivni poziv. Dakle, i ovde je ukupno vreme izvršavanja reda $O(n \log n + k)$.

Sveukupno gledano, oba algoritma su jednakog asimptotskog ponašanja. Štaviše, moguće je pokazati da je to ponašanje i asimptotski optimalno prema algebarskom modelu drveta odlučivanja (klasičan model RAM-a računara, za razliku od specijalnih modela kao što je npr. kvantni, koji omogućava konstrukciju znatno bržih algoritama), pošto odgovara donjoj granici problema jedinstvenosti elemenata (postoje li dva ista elementa u nizu), koji je linearno svodljiv na presecanje pravougaonika. Oba algoritma su asimptotski jednaka i optimalna i po prostornoj složenosti. Naime, ne računajući prostor neophodan za skladištenje skupa preseka, koji je očekivano reda $O(k)$, i jedan i drugi pristup zahtevaju samo $O(n)$ dodatnog prostora. Pritom je strategija podeli pa vladaj znatno jednostavnija utoliko što od pomoćnih struktura podataka koristi samo nizove (i par se može shvatiti kao dvočlani niz), dok je metod brišuće prave zasnovan na kompleksnim samobalansirajućim stablima. Ovo je čini bržom i pogodnom za slabije sisteme sa ograničenim resursima, kao što su mobilni telefoni. Ipak, loša strana strategije podeli pa vladaj je to što se pojedini preseci ponavljaju, dok drvo nema taj problem.

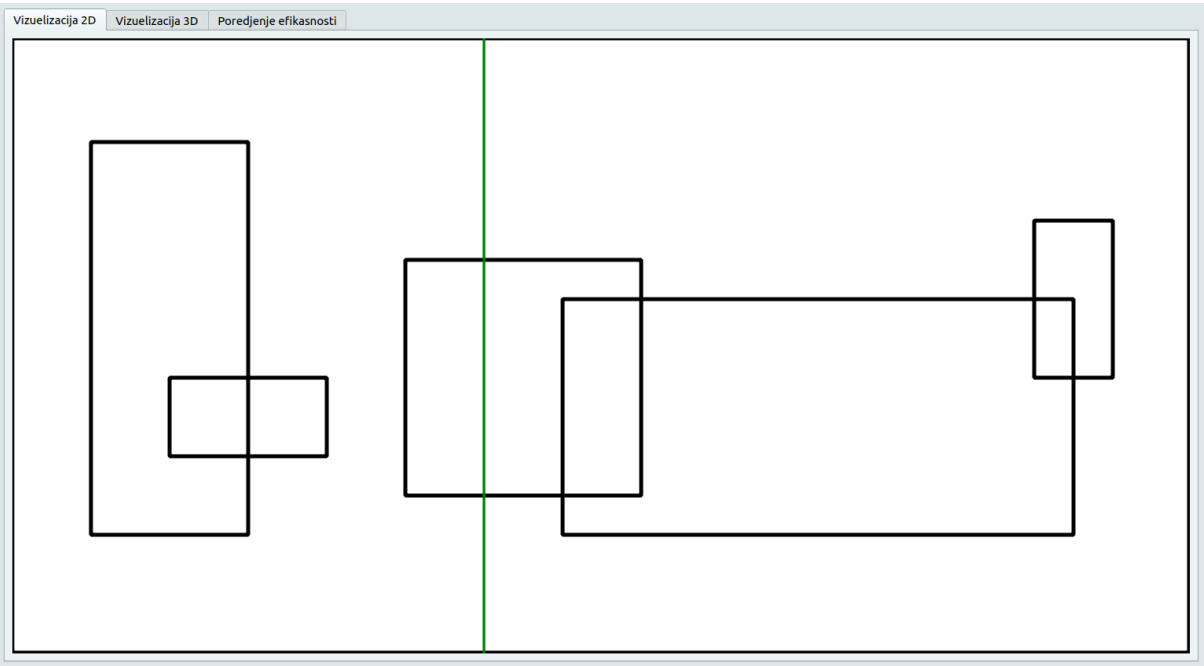
Vizuelizacija glavnog algoritma

Algoritam će biti ukratko vizuelizovan na skupu pet pravougaonika za koje važi da su u relativno opštem položaju, pošto postoje samo tri preseka (maksimum bi bio deset). Biće preskočeni koraci koji se ponavljaju, a naglašeni oni važni. Postavka je data na slici 2, kao i u prvoj testnoj ulaznoj datoteci.



Slika 2. Postavka za prikaz vizuelizacije algoritma podeli pa vladaj

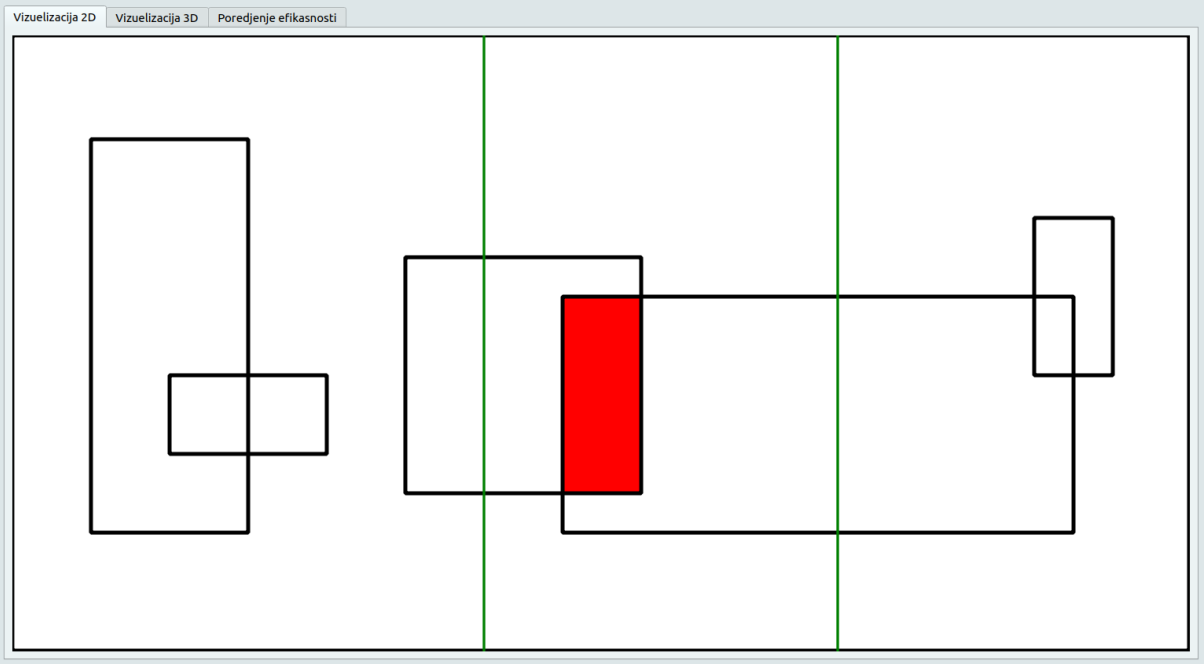
Strategija podeli pa vladaj fizički deli prostor pretrage na dva po veličini (broju vertikalnih ivica) jednaka potprostora. Pritom izračunava skupove kandidata i čuva sortiranost pomoćnog niza pravougaonika po vertikalnom položaju donje ivice. Prva tačka (linija) podele prikazana je na slici 3.



Slika 3. Prva tačka podele vizuelizacije algoritma podeli pa vladaj

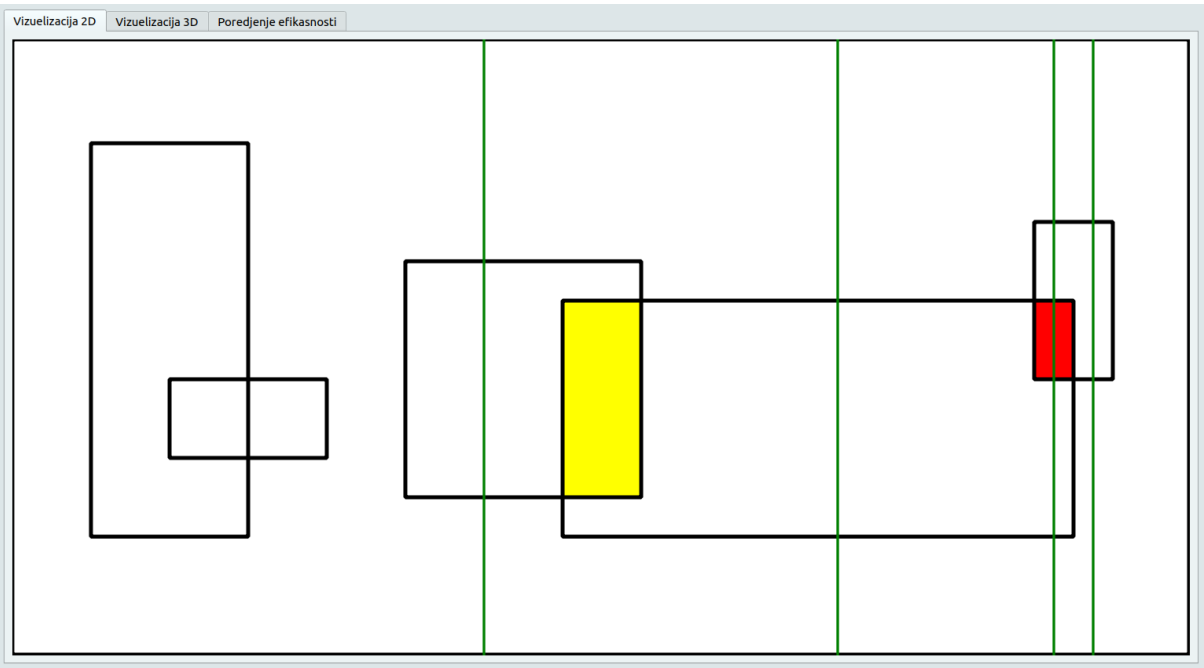
Nakon provere skupova kandidata, otkriven je prvi presek – između trećeg i četvrtog pravougaonika, gledano sleva nadesno, kako i sâm algoritam deli prostor. Treći pravougaonik pripao je skupu S_{11} (desna strana mu je u levoj polovini), dok je drugi upao u skup S_{21} (desna strana mu je u desnoj

polovini, a leva prekriva levu polovinu), što su skupovi čiji se pripadnici obavezno seku po x koordinati, te je neophodno brzo proveriti da li isto važi i za y dimenziju. Prikaz toga dat je na slici 4.



Slika 4. Prvi pronađeni presek vizuelizacije algoritma podeli pa vladaj

Isto se dogodilo sa pretposlednjim i poslednjim pravougaonikom u nizu. Prikaz toga dat je na slici 5.

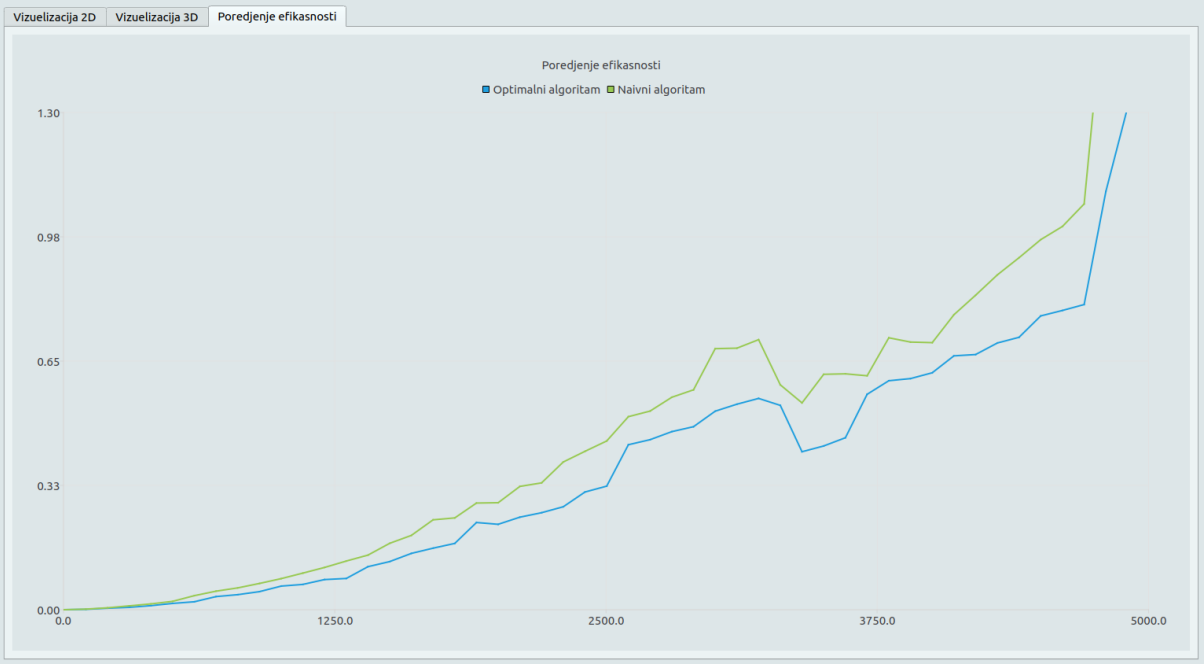


Slika 5. Drugi pronađeni presek vizuelizacije algoritma podeli pa vladaj

Apsolutno istim postupkom dolazi se do konačnog rešenja, prikazanog na naslovnoj stranici. Izlaskom iz rekurzije, nestaju sve prikazane podele (uspravne zelene linije), dok svi preseci ostaju žuti. Kako je dosad verovatno primećeno, novootkriveni preseci obojeni su karakterističnom crvenom bojom.

Poređenje efikasnosti optimalnih algoritama

Na slici 6 prikazan je grafik zavisnosti vremena izvršavanja predloženih algoritama od veličine slučajnog ulaza. Primetna je njihova istovetna asimptotska složenost, ali i prednost strategije podeli pa vladaj („optimalnog“ algoritma) u odnosu na metod brišuće prave („naivni“) u konstantnim faktorima.



Slika 6. Grafik zavisnosti vremena izvršavanja predloženih algoritama od veličine slučajnog ulaza

Testiranje ispravnosti algoritama

Cela logika algoritama sadržana je u klasi PresekPravougaonika. Ne računajući konstruktor i destruktor, kao ni virtuelni metod za crtanje algoritma, njen javni interfejs sastoji se iz tri metoda bez povratne vrednosti, kojima se pokreću odgovarajući algoritmi, kao i tri metoda koja vraćaju rezultate tih algoritama u vidu uređenih skupova preseka. Upravo su ovi metodi okosnica napisanih jediničnih testova. Testovi su realizovani pomoću GoogleTest radnog okvira (biblioteke), zasnovanog na dobro poznatoj xUnit arhitekturi, a njihov tačan i detaljan popis dat je u tabeli 1 koja sledi. Svaki od testova je napisan u skladu sa paradigmom (šablonom) AAA (engl. *arrange-act-assert*), koja podrazumeva implementaciju iz tri dela – postavljanja podataka, izračunavanja i provere ispunjenosti očekivanja.

Naziv testa	Opis testa	Ulaz	Očekivani izlaz
datotekaTest1	Zadavanje ulaza koji odgovara nekom relativno opštem slučaju, onom vizuelizovanom u okviru ove dokumentacije	Niz dimenzije 5 (pet) iz datoteke input1.txt	Poklapanje rezultata sva tri izložena algoritma i veličine izlaza 3 (tri)

datotekaTest2	Zadavanje još jednog lakog ulaza za sticanje predstave o algoritmu	Niz dimenzije 4 (četiri) iz datoteke input2.txt	Poklapanje rezultata sva tri izložena algoritma i veličine izlaza 2 (dva)
datotekaTest3	Zadavanje ulaza koji odgovara kao najgorem slučaju – svi se seku, a pritom se ponavalju mnoge koordinate ivica	Niz dimenzije 5 (pet) iz datoteke input3.txt	Poklapanje rezultata sva tri izložena algoritma i veličine izlaza 10 (deset)
randomTest1	Pravljenje 10 (deset) nasumičnih elemenata	Slučajni niz dužine 10	Poklapanje rezultata sva tri izložena algoritma
randomTest2	Pravljenje 100 (stotinu) nasumičnih elemenata	Slučajni niz dužine 100	Poklapanje rezultata sva tri izložena algoritma
randomTest3	Pravljenje 1000 (hiljadu) nasumičnih elemenata	Slučajni niz dužine 1000	Poklapanje rezultata sva tri izložena algoritma
nemaPravougaonika	Zadavanje praznog ulaza. Kada nema elemenata, sigurno nema ni preseka.	Prazan niz (dužine nula)	Prazan niz preseka
jedanPravougaonik	Zadavanje jednočlanog ulaza. Sigurno bez drugih elemenata nema preseka.	Slučajni jednočlani niz	Prazan niz preseka

Tabela 1. Prikaz napisanih jediničnih testova za klasu PresekPravougaonika

Uz testove je pokrenuto i nekoliko alata za profajliranje, koji su pomogli u strožoj proceni učinkovitosti testiranja, kao i same testirane implementacije. Prvi od njih su alati iz paketa Valgrind. Pomoću njih je pokazano da nema problema u radu sa memorijom (kako hipom, tako stekom), nitima ili kešom prilikom izvršavanja. Drugi alat jeste gcov, odnosno njegova grafička verzija LCOV, koji služi za izračunavanje pokrivenosti koda testovima. Pomoću njega je pokazano da navedeni skup od osam testova pokriva visokih 92 % linija koda, pri čemu nepokrivenih 8 % isključivo čini crtanje algoritma. Ono se ne sprovodi u toku testiranja, ali je iz upotrebe aplikacije jasno da radi u skladu sa očekivanjima. Jednaka je pokrivenost funkcija, iz istog razloga. Pokrivenost grana je nešto niža, oko dve trećine, ali to je donekle i očekivano, pošto se veliki broj grana odnosi na slučajeve koje u suštini nema potrebe simulirati, kao što je npr. slučaj da neka alokacija izbaci izuzetak. U takvim trenucima je, ipak, sasvim u redu da program padne bez ikakvog pokušaja oporavka ili izveštaja šta se dogodilo.