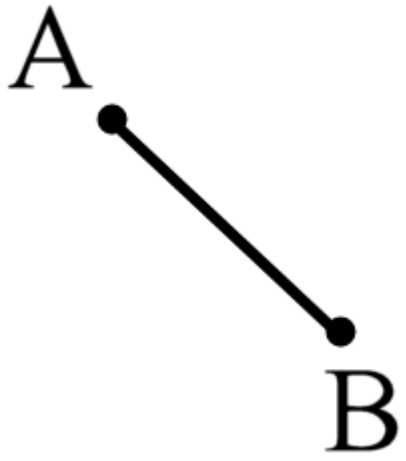
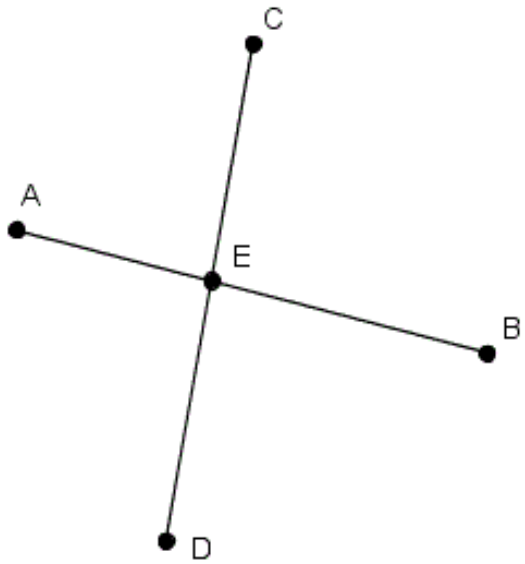
PRESJEK SEGMENATA U RAVNINI





Filip Srnec

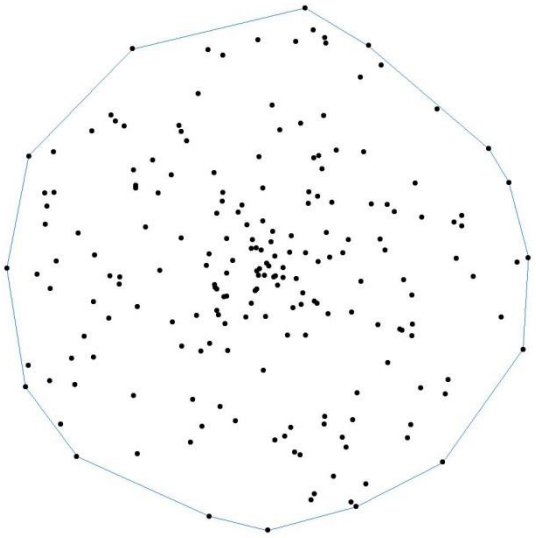
PMF – MO, 15.1.2015.

RAČUNALNA GEOMETRIJA *(Computational Geometry)*

Algoritmi za rješavanje geometrijskih problema

PRIMJERI PROBLEMA:

- Najbliži par točaka
- Konveksna ljuska (*Convex hull)*
- Triangulacija polinoma
- **Presjek segmenata u ravnini**
- **…**



Terminologija…

TOČKA $p = (x, y) \in \mathbb{R}^2$

KONVEKSNA KOMBINACIJA točaka $p_1 = (x_1, y_1)$ i $p_2 = (x_2, y_2)$ je svaka točka $p_3 = (x_3, y_3)$ takva da postoji $\alpha \in [0, 1]$ takav da vrijedi $p_3 = \alpha p_1 + (1 - \alpha) p_2$.

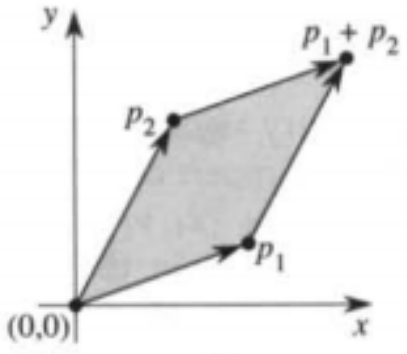
SEGMENT (line segment, dužina) $\overline{p_1 p_2}$ je skup svih konveksnih kombinacija točaka p_1 i p_2.

p_1 p_2 - rubne točke segmenta
$\overrightarrow{p_1 p_2}$ - usmjeren segment

Cross product

Cross product $p_1 \times p_2$ je determinanta $\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1$.

$|p_1 \times p_2|$ se interpretira kao površina paralelograma formiranog točkama $(0,0)$, p_1, p_2 i $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$.



Presjek dva segmenta

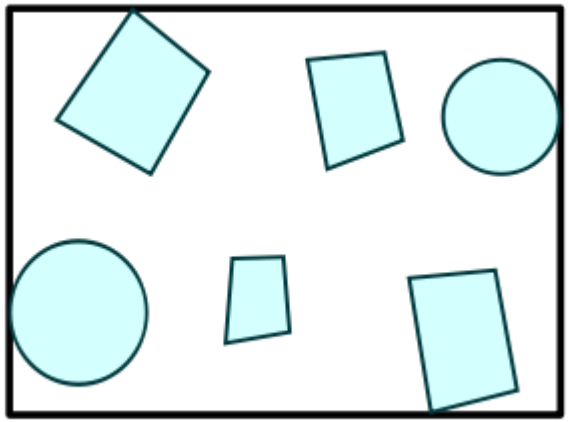
DVA KORAKA ALGORITMA

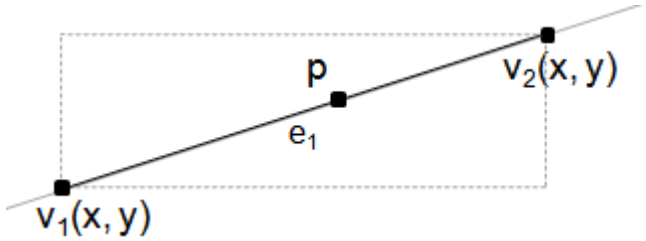
1. ***BRZO ODBACIVANJE** (quick rejection)*
2. ***UTVRĐIVANJE „OKRUŽUJU” LI OBA SEGMENTA PRAVAC NA KOJEM SE NALAZI DRUGI SEGMENT** (straddle)*

Brzo odbacivanje

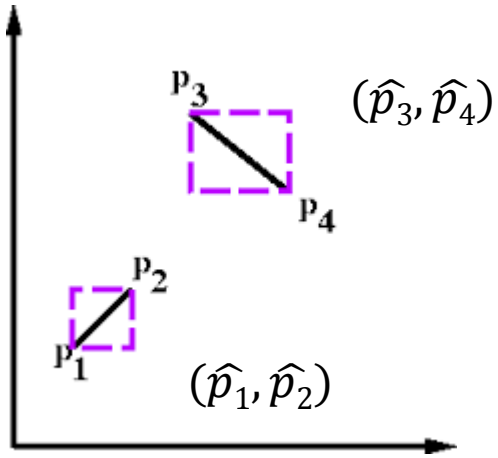
Bounding box geometrijskog objekta je najmanji pravokutnik koji sadrži taj objekt i čije stranice su paralelne sa x-osi i y-osi.

Bounding box segmenta $\overline{p_1 p_2}$ je pravokutnik $(\widehat{p_1}, \widehat{p_2})$ gdje je $\widehat{p_1} = (\widehat{x_1}, \widehat{y_1})$ „donji lijevi", a $\widehat{p_2} = (\widehat{x_2}, \widehat{y_2})$ „gornji desni" vrh pravokutnika, tj.
$\widehat{x_1} = min(x_1, x_2)$, $\widehat{y_1} = min(y_1, y_2)$,
$\widehat{x_2} = max(x_1, x_2)$, $\widehat{y_2} = max(y_1, y_2)$,



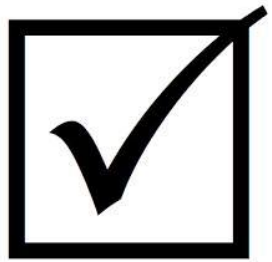


Dva segmenta nemaju presjek ako im se ne sijeku pripadni *bounding boxevi*.



$$(\hat{x}_2 \geq \hat{x}_3) \wedge (\hat{x}_4 \geq \hat{x}_1) \wedge (\hat{y}_2 \geq \hat{y}_3) \wedge (\hat{y}_4 \geq \hat{y}_1) \equiv false$$

Dakle, ako gornji uvjet nije zadovoljen, segmenti se sigurno ne sijeku.

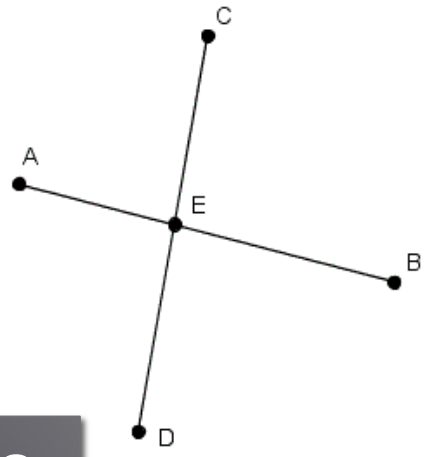


Presjek dva segmenta

DVA KORAKA ALGORITMA

1. ***BRZO ODBACIVANJE** (quick rejection)* ☑
2. ***UTVRĐIVANJE „OKRUŽUJU” LI OBA SEGMENTA PRAVAC NA KOJEM SE NALAZI DRUGI SEGMENT** (straddle)*

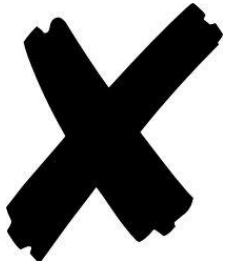
Segment $\overline{p_1p_2}$ „**okružuje**” pravac na kojem se nalazi segment $\overline{p_3p_4}$ ako se p_1 nalazi s jedne, a p_2 s druge strane tog pravca. (*NEPRECIZNO*)



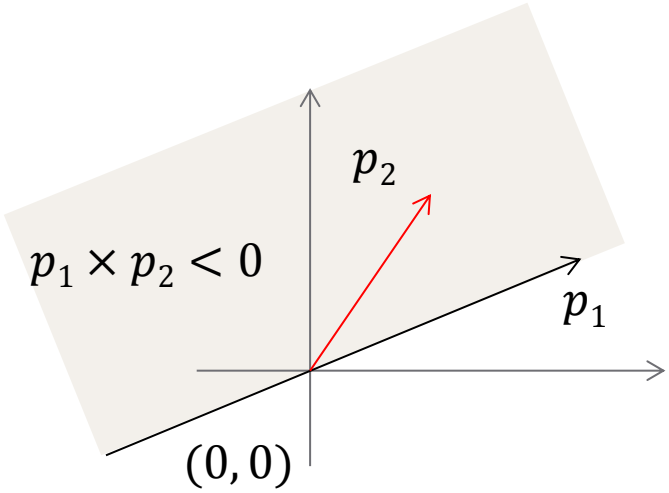
Dva segmenta se sijeku ako i samo ako su prošli *brzo odbacivanje* (prvi korak algoritma) i svaki od njih „*okružuje*” pravac na kojem se nalazi drugi segment.

KAKO?

CROSS PRODUCT

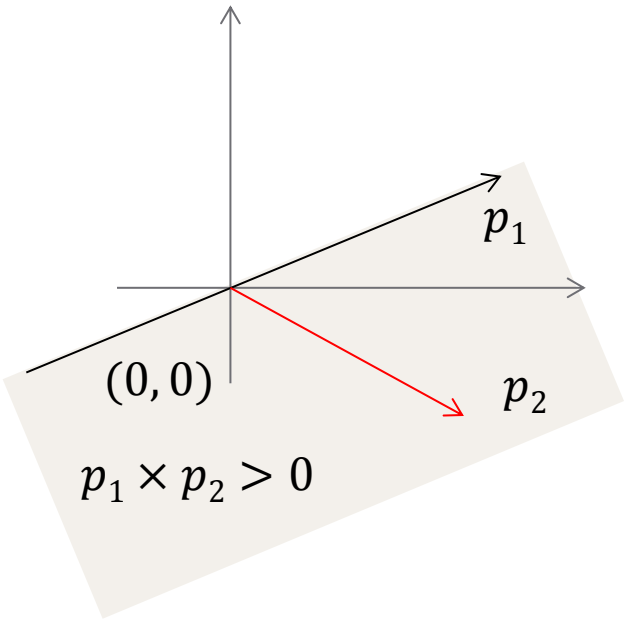


Lako se pokaže…



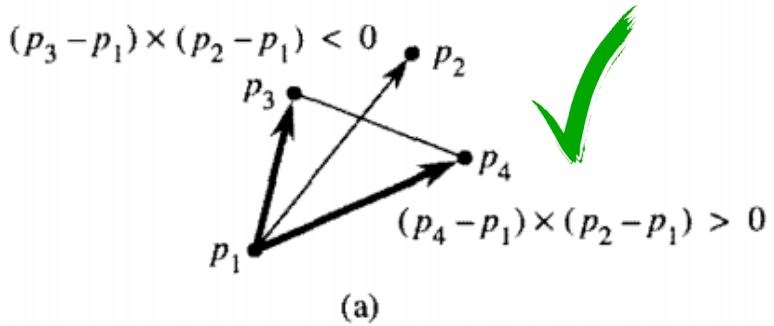
p_2 je „s lijeve strane”
pravca koji sadrži p_1

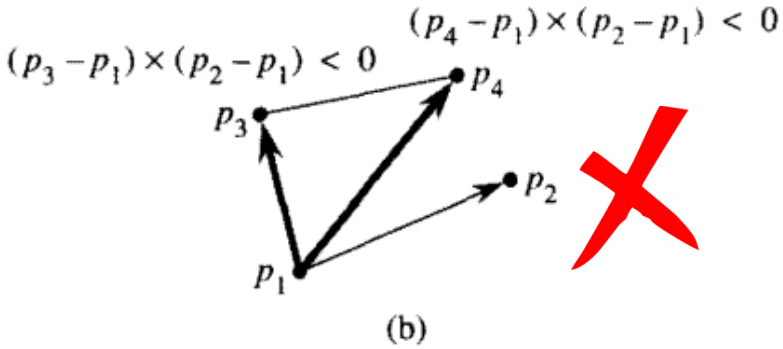
p_2 je „s desne strane”
pravca koji sadrži p_1

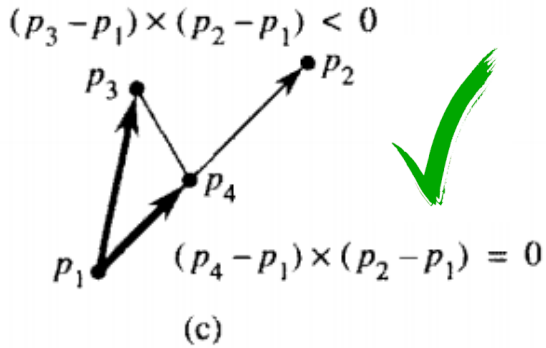


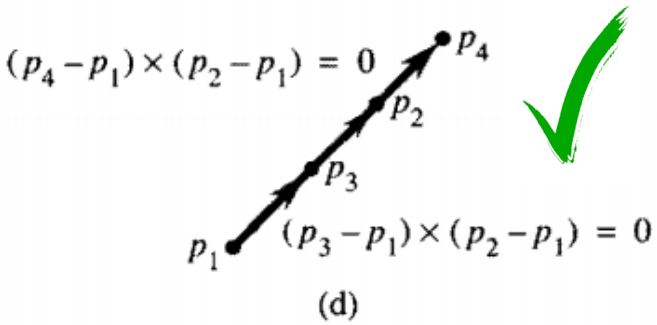
$$p_1 \times p_2 = 0 \Leftrightarrow p_1 \; i \; p_2 \; su \; kolinearne$$

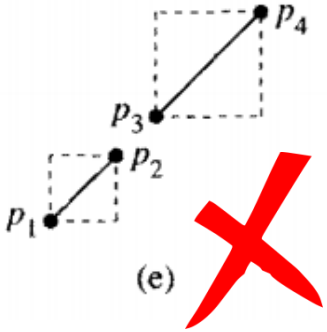
Uočimo…









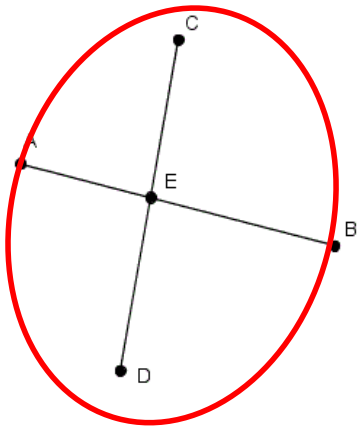


ZAŠTO?

Zaključak!

Segment $\overline{p_1p_2}$ „**okružuje**” pravac na kojem se nalazi segment $\overline{p_3p_4}$ ako:

- $(p_3 - p_1) \times (p_2 - p_1)$ i $(p_4 - p_1) \times (p_2 - p_1)$ imaju **različite** predznake
- $(p_3 - p_1) \times (p_4 - p_1) = 0$ ili $(p_4 - p_1) \times (p_2 - p_1) = 0$ i zadovoljen je uvjet da se **barem jedna** od točaka p_3 i p_4 nalazi na $\overline{p_1p_2}$



Taj uvjet je zadovoljen ako je zadovoljen ***quick rejection*** **test** (prvi korak algoritma)

Presjek dva segmenta

DVA KORAKA ALGORITMA

1. ***BRZO ODBACIVANJE** (quick rejection)* ☑
2. ***UTVRĐIVANJE „OKRUŽUJU” LI OBA SEGMENTA PRAVAC NA KOJEM SE NALAZI DRUGI SEGMENT** (straddle)* ☑

Presjek dva segmenta

```
bool quickRejection(LineSegment segment1, LineSegment segment2) {
    double x1 = min(segment1.point1.x, segment1.point2.x);
    double y1 = min(segment1.point1.y, segment1.point2.y);
    double x2 = max(segment1.point1.x, segment1.point2.x);
    double y2 = max(segment1.point1.y, segment1.point2.y);

    double x3 = min(segment2.point1.x, segment2.point2.x);
    double y3 = min(segment2.point1.y, segment2.point2.y);
    double x4 = max(segment2.point1.x, segment2.point2.x);
    double y4 = max(segment2.point1.y, segment2.point2.y);

    return (x2 >= x3) && (x4 >= x1) && (y2 >= y3) && (y4 >= y1);
}

bool straddle(LineSegment segment1, LineSegment segment2) {
    double cross1 = crossProduct(segment2.point1 - segment1.point1, segment1.point2 - segment1.point1);
    double cross2 = crossProduct(segment2.point2 - segment1.point1, segment1.point2 - segment1.point1);

    return ((cross1 >= 0 && cross2 <=0) || (cross1 >= 0 && cross2 <= 0));
}


bool intersect(LineSegment segment1, LineSegment segment2) {
    if (quickRejection(segment1, segment2)) {
        if (straddle(segment1, segment2) && straddle(segment2, segment1)) {
            return true;
        }
    }

    return false;
}
```

Konstantan broj operacija!

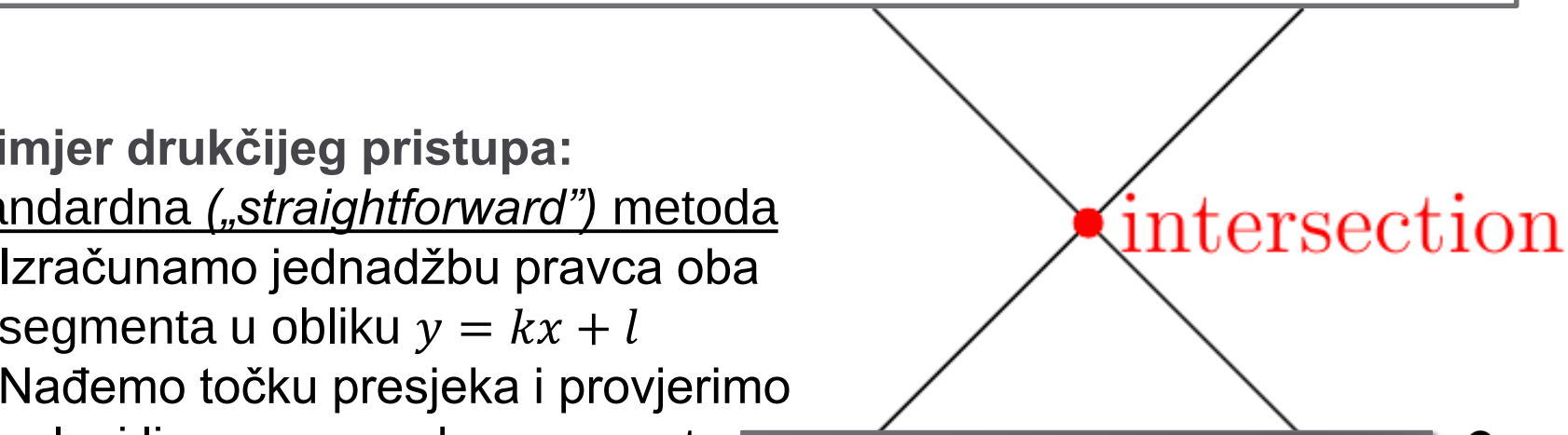
$O(1)$

ZAŠTO TAKO?

Koristimo samo zbrajanje, oduzimanje, množenje i uspoređivanje, nema trigonometrijskih funkcija ni dijeljenja (mogu biti skupe za izračunavanje, izbjegavamo veće greške zaokruživanja)

Primjer drukčijeg pristupa:
standardna *(„straightforward")* metoda

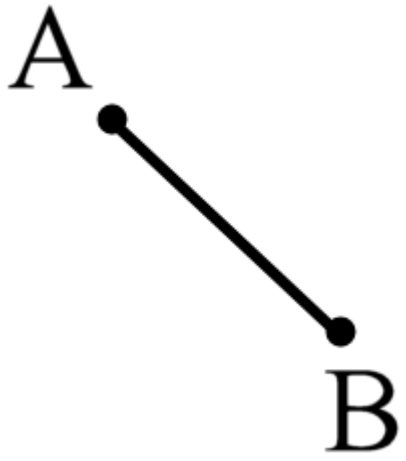
- Izračunamo jednadžbu pravca oba segmenta u obliku $y = kx + l$
- Nađemo točku presjeka i provjerimo nalazi li se ona na oba segmenta



Osjetljivost na preciznost operacije dijeljenja kad su pravci skoro paralelni!

!

PRESJEK SEGMENATA U RAVNINI



PROBLEM:

Dan je skup $L = \{l_1, l_2, \ldots, ln\}$ koji sadrži n segmenata u ravnini. Potrebno je odrediti sijeku li se bilo koja dva od tih n segmenata. Pretpostavimo da se nikoja tri segmenta ne sijeku u istoj točki te da ne postoji segment okomit na x-os.

IDEJA!

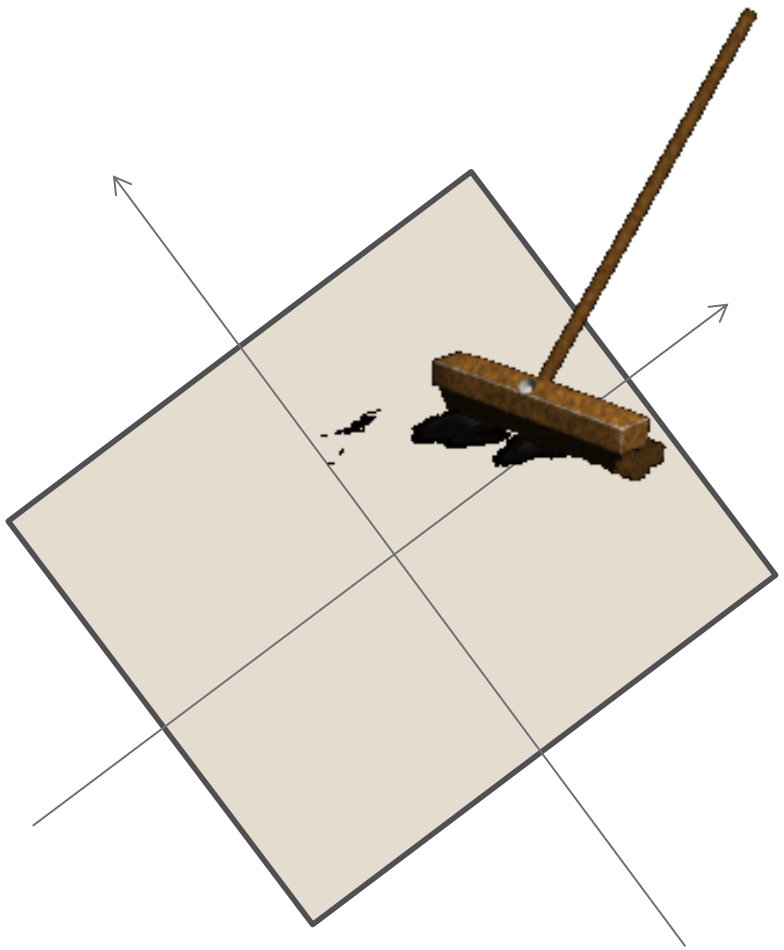
„GRUBA SILA” – promatramo sve parove segmenata u skupu i koristimo *intersecton(LineSegment l1, LineSegment l2)* za utvrđivanje presjeka

Algoritam je očito kvadratne složenosti u broju segmenata u skupu!

$O(n^2)$

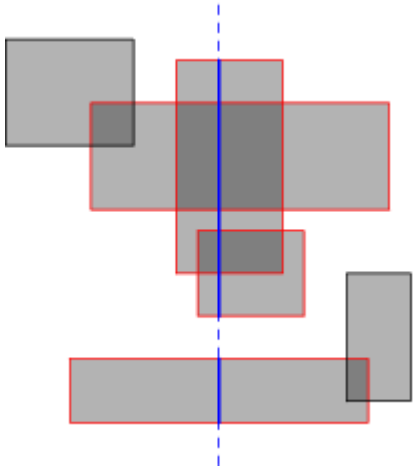
Možemo li bolje?

SWEEPING



SWEEPING

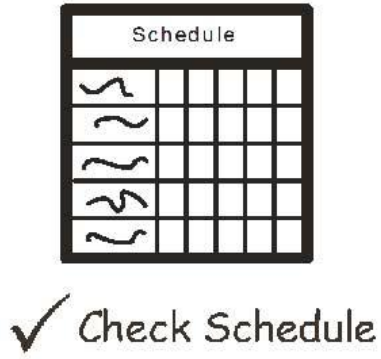
- Česta metoda u rješavanju geometrijskih problema
- Imaginarni pravac, ***SWEEP LINE,*** prolazi kroz dani skup geometrijskih objekata



- U ravnini je on najčešće okomit na x-os i prolazi „slijeva na desno”
- Dimenzija kojom se ***SWEEP LINE*** kreće **(najčešće x–os)** tretira se kao dimenzija vremena.

Sweeping algoritmi – dva skupa podataka

SWEEP LINE STATUS – daje vezu između objekata koje *SWEEP LINE* siječe u danom trenutku



EVENT – POINT SCHEDULE – niz koordinata koje definiraju trenutnu poziciju *SWEEP LINE* (*najčešće x koordinate točaka koje promatramo, sortirane s lijeva na desno*)

Svaka koordinata u nizu je jedan ***EVENT*** (događaj) kad se mijenja ***SWEEP LINE STATUS***

Kako simuliramo *SWEEPING*?

Sortiramo sve rubne točke segmenata iz L koristeći sljedeći (*leksikografski*) uređaj:

$$p_1 \prec p_2 \overset{def}{\Longleftrightarrow} \begin{cases} x_1 < x_2 \\ x_1 = x_2 \wedge y_1 < y_2 \end{cases}$$

Gdje su $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$.

Svaka točka u sortitanom nizu, (svaka rubna točka nekog segmenta iz L) je jedan ***EVENT*** (događaj)

EVENT – POINT SCHEDULE

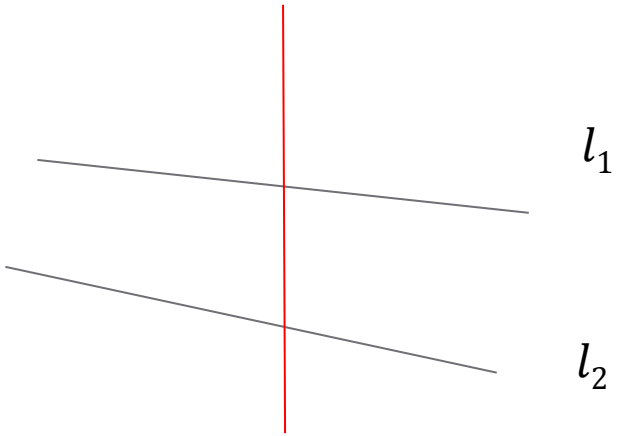
Što još trebamo?

Treba nam **uređaj** na skupu svih segmenata!

Nema vertikalnih segmenata → *SWEEP LINE* svaki segment siječe u jednoj točki

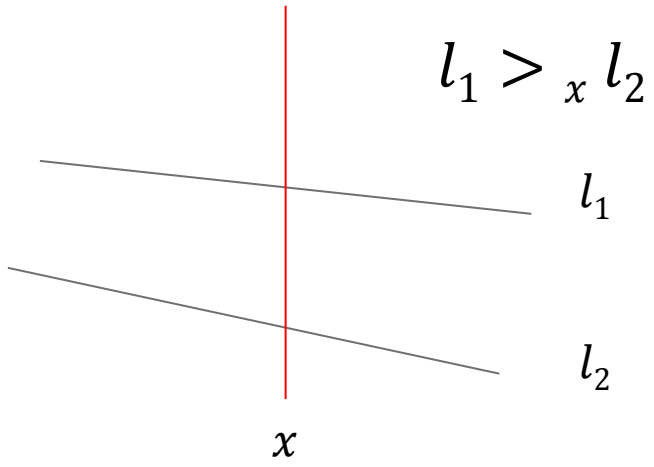
---

Kažemo da su segmenti l_1 i l_2 ***USPOREDIVI na pravcu x*** ako vertikalan pravac (*SWEEP LINE*) x siječe oba ta segmenta.

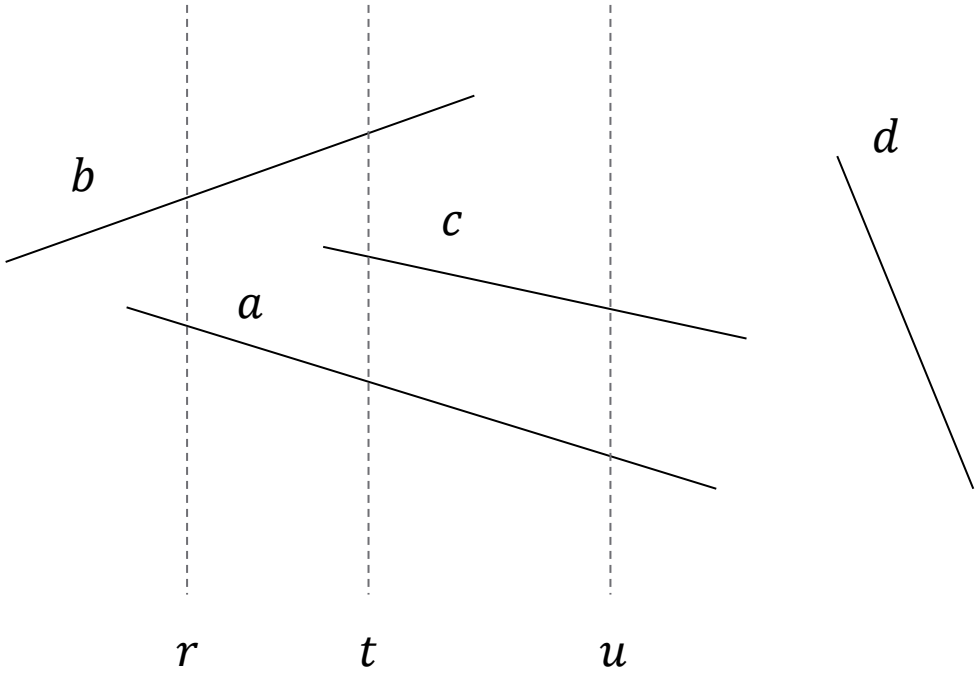


Kažemo da je l_1 ***IZNAD (ABOVE)*** l_2 i pišemo $l_1 >_x l_2$ ako su l_1 i l_2 usporedive na x i y koordinata točke presjeka l_1 i vertikalnog pravca x je veća od y – koordinate točke presjeka l_2 i istog vertikalnog pravca.

Kažemo da je l_1 ***ISPOD (BELOW)*** l_2 i pišemo $l_1 <_x l_2$ ako su l_1 i l_2 usporedive na x i y koordinata točke presjeka l_1 i vertikalnog pravca x je manja od y – koordinate točke presjeka l_2 i istog vertikalnog pravca.



Za dani x, $>_x$ je totalni uređaj.



VRIJEDE RELACIJE:

$$a <_r b$$
$$a <_t c <_t b$$
$$c <_u b$$

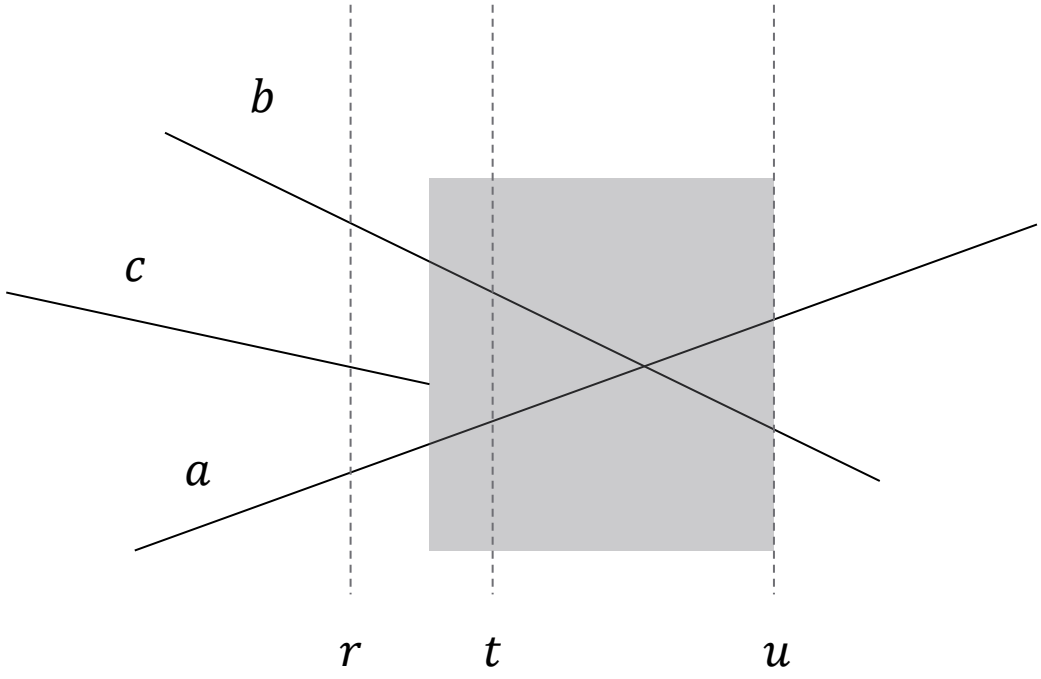
d nije usporediv s ni jednim od a, b, c

IDEJA!

Segment ulazi u „uređaj” (T) kad *SWEEP LINE* dođe do njegove početne točke, a izlazi iz „uređaja” kad dođe do njegove završne točke!

SWEEP LINE STATUS!

Što kad *SWEEP LINE* prijeđe presjek neka dva segmenta?



Pravci r i u su s lijeve i desne strane, respektivno, od točke presjeka segmenata a i b te imamo $b >_r a$ and $a >_u b$.

Nikoja tri segmenta se ne sijeku u istoj točki

Postoji vertikalni pravac t za koji su segmenti a i b **uzastopni** u uređaju $>_t$

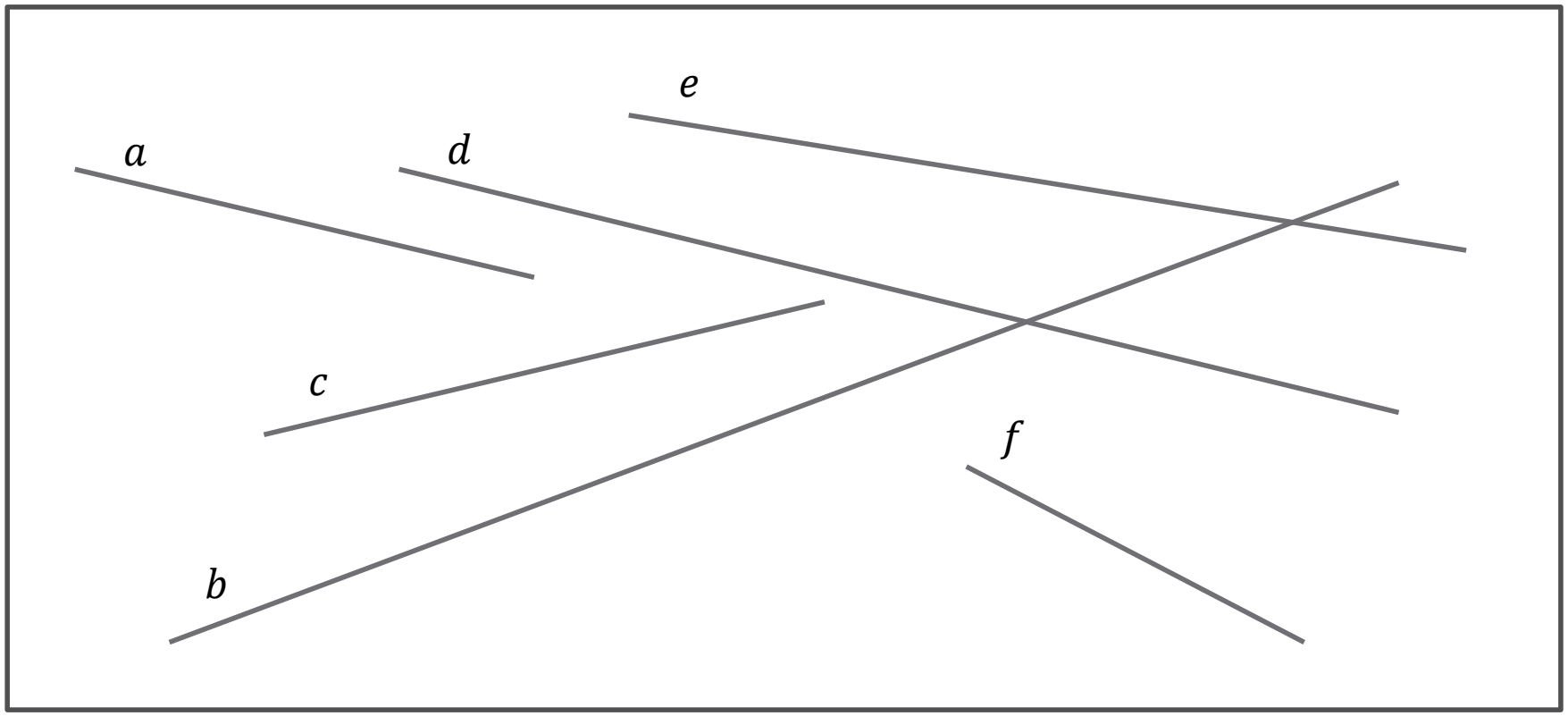
ALGORITAM (opisno)

Prolazimo po rubnim točkama p segmenata sortiranih s lijeva na desno:

1) Ako je p **lijevi rub** nekog segmenta l, stavljamo l u uređaj T,
   - Ako postoji segment l_1 u T takav da je l_1 **neposredno iznad** l
     - Provjeri sijeku li se l i l_1 (ako da, *GOTOVO*)
   - Ako postoji segment l_2 u T takav da je l_2 **neposredno ispod** l
     - Provjeri sijeku li se l i l_2 (ako da, *GOTOVO*)

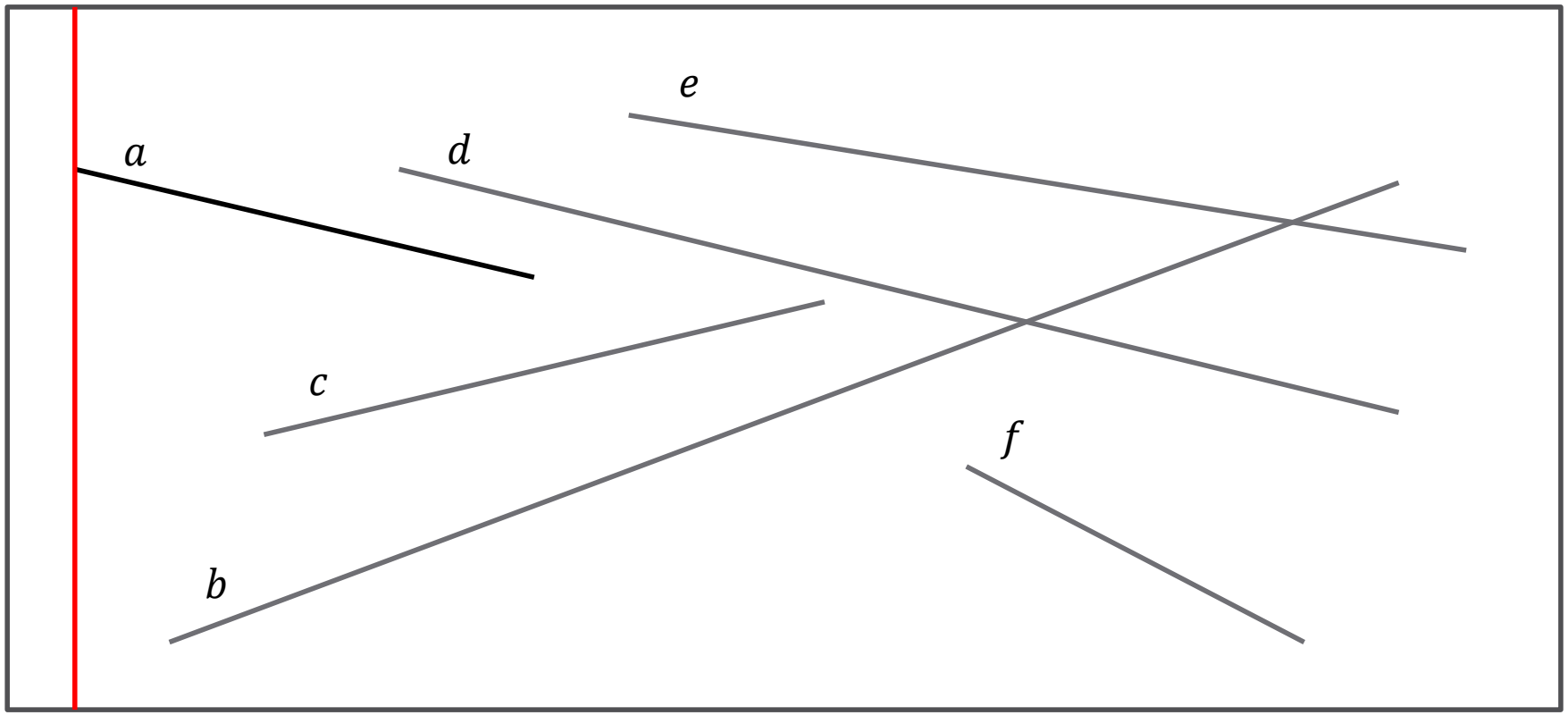
2) Ako je p **desni rub** nekog segmenta l, izbacujemo l iz uređaja T
   - Ako postoje segmenti l_1 i l_2 u T koji su **neposredno iznad** i **neposredno ispod** l
     - Provjeri sijeku li se l_1 i l_2 (ako da, *GOTOVO*)

ALGORITAM (primjer izvršavanja)



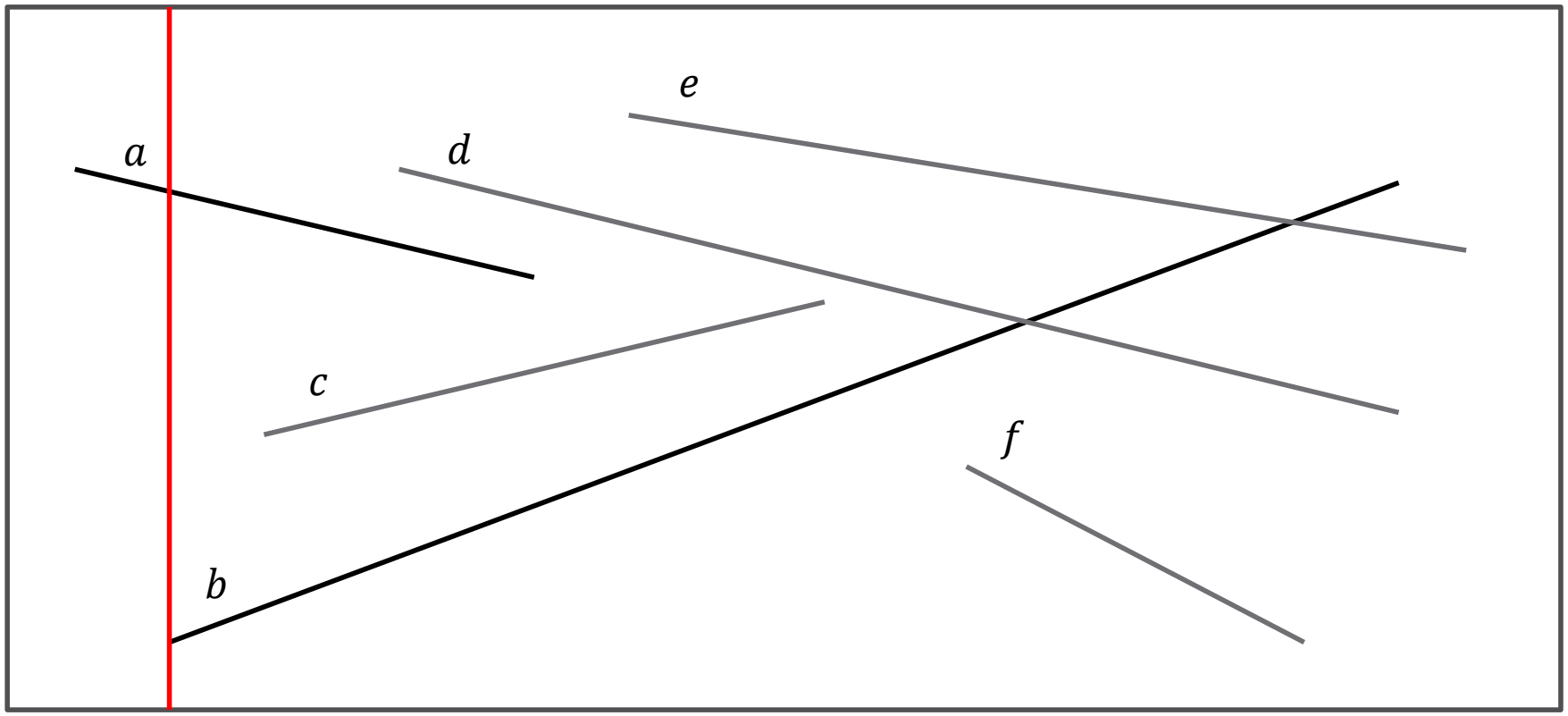
$$T = \emptyset$$

ALGORITAM (primjer izvršavanja)



$$T = \{a\}$$

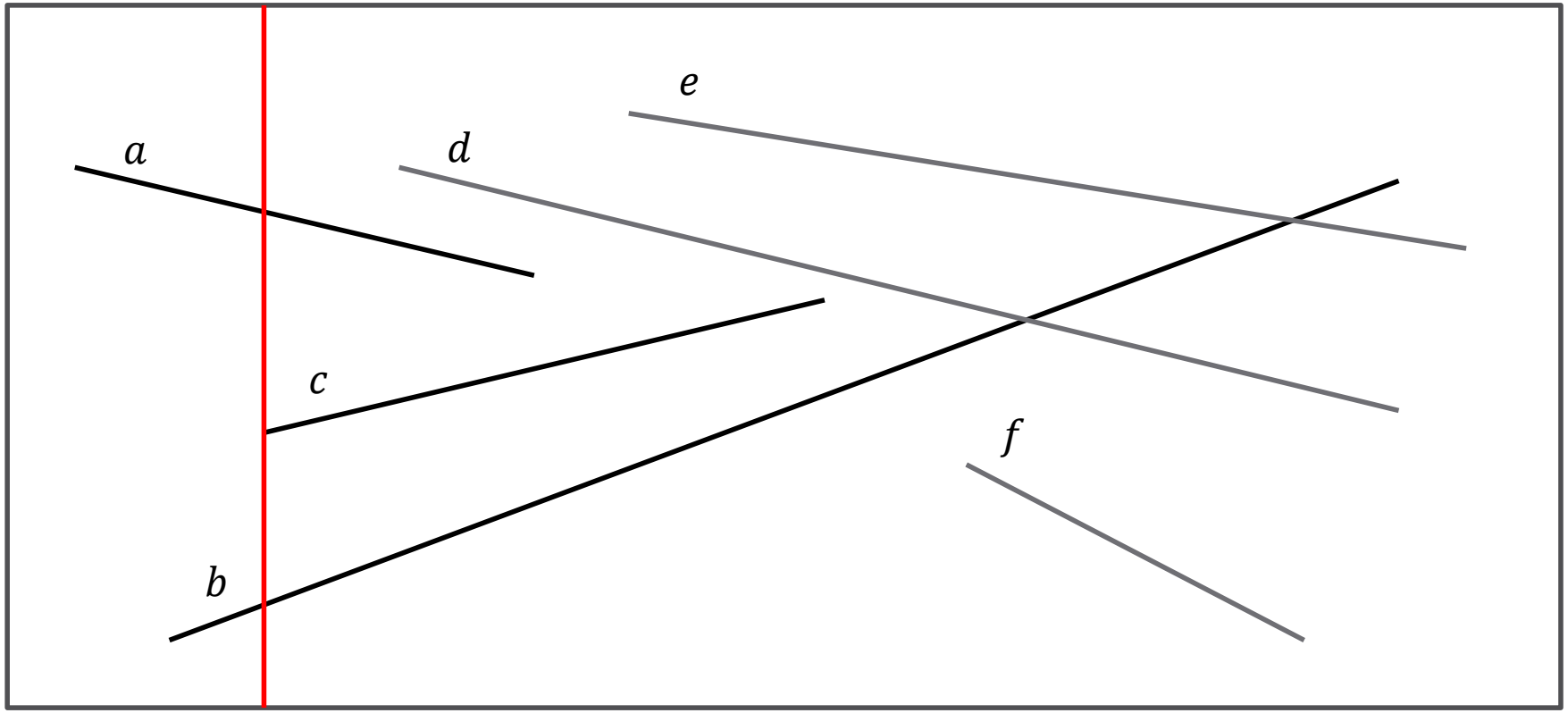
ALGORITAM (primjer izvršavanja)



$T = \{b, a\}$

a i b se ne sijeku

ALGORITAM (primjer izvršavanja)

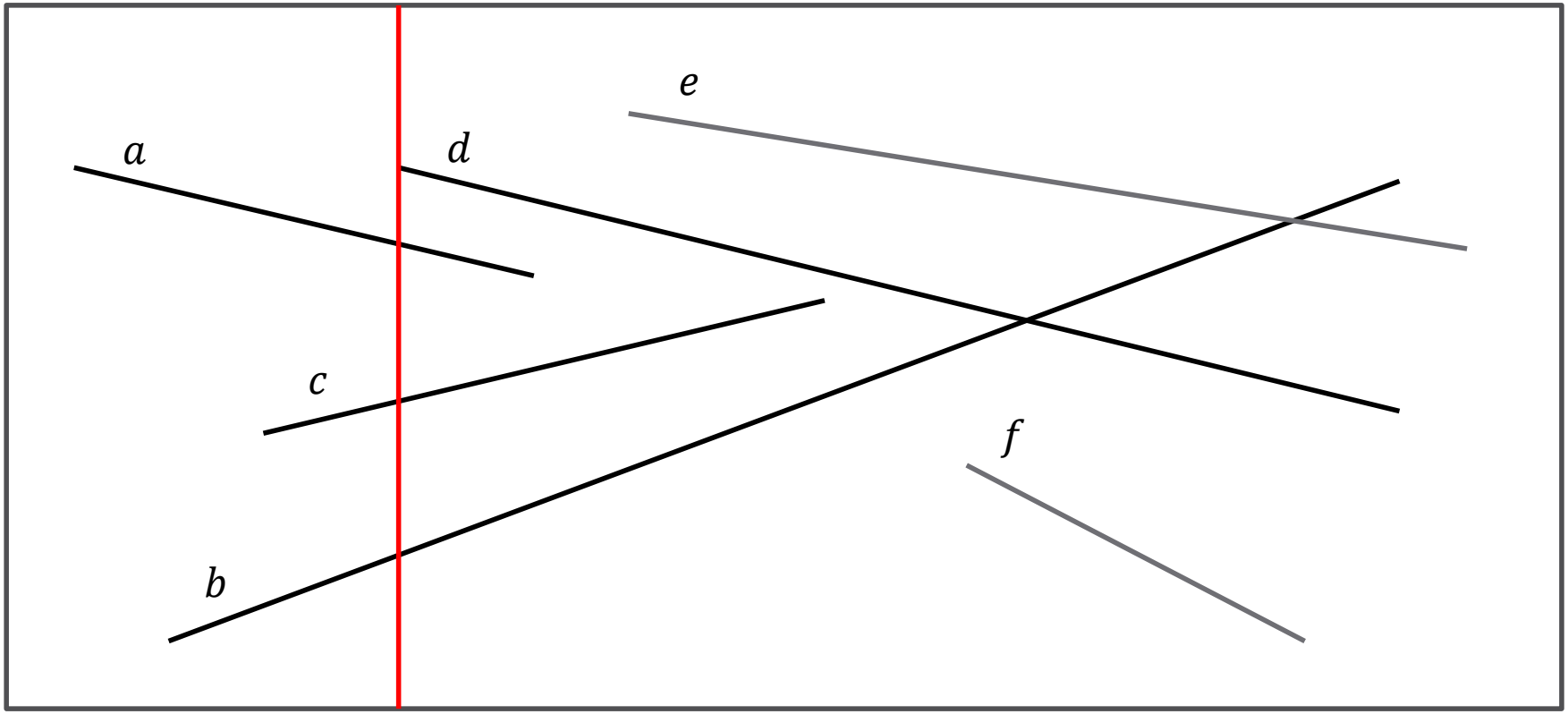


$T = \{b, c, a\}$

c i b se ne sijeku

c i a se ne sijeku

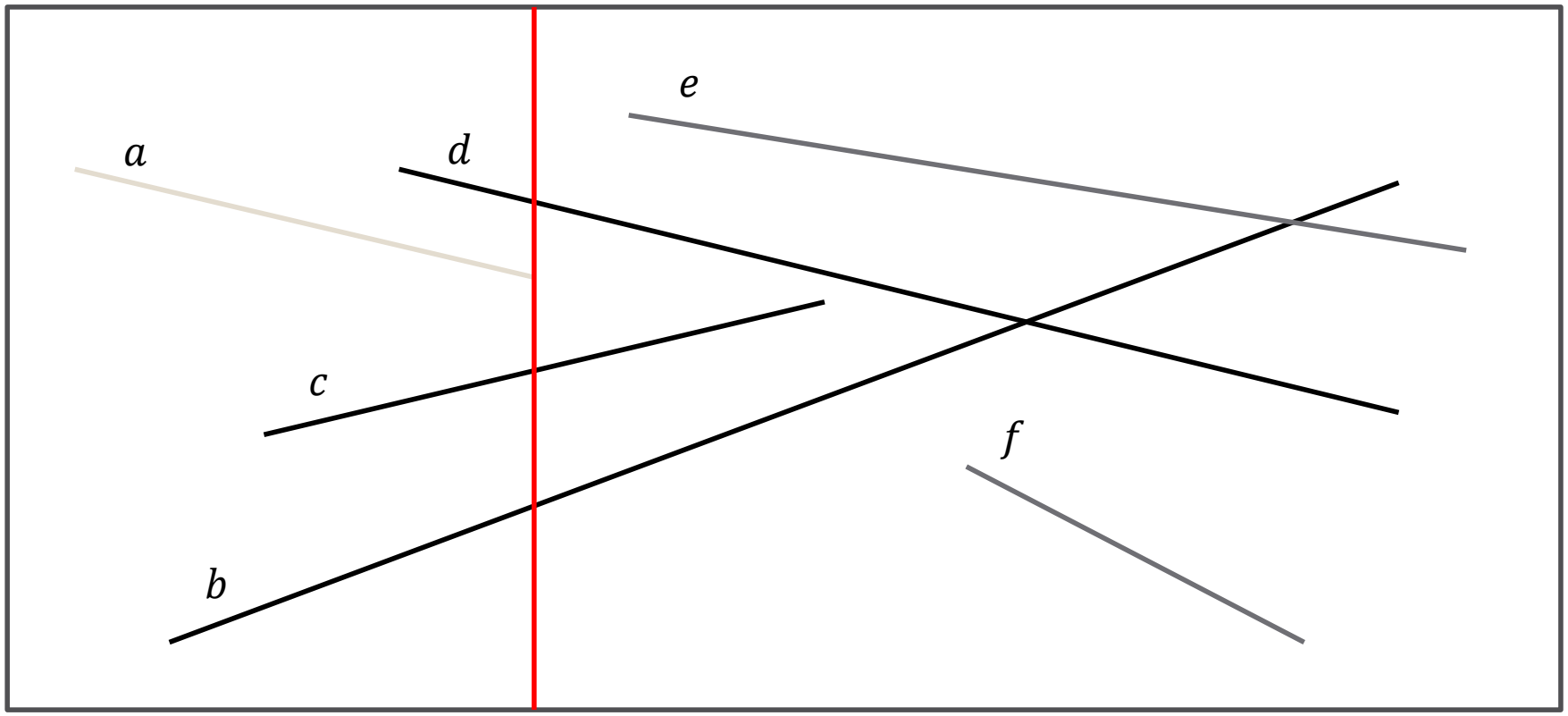
ALGORITAM (primjer izvršavanja)



$T = \{b, c, a, d\}$

d i a se ne sijeku

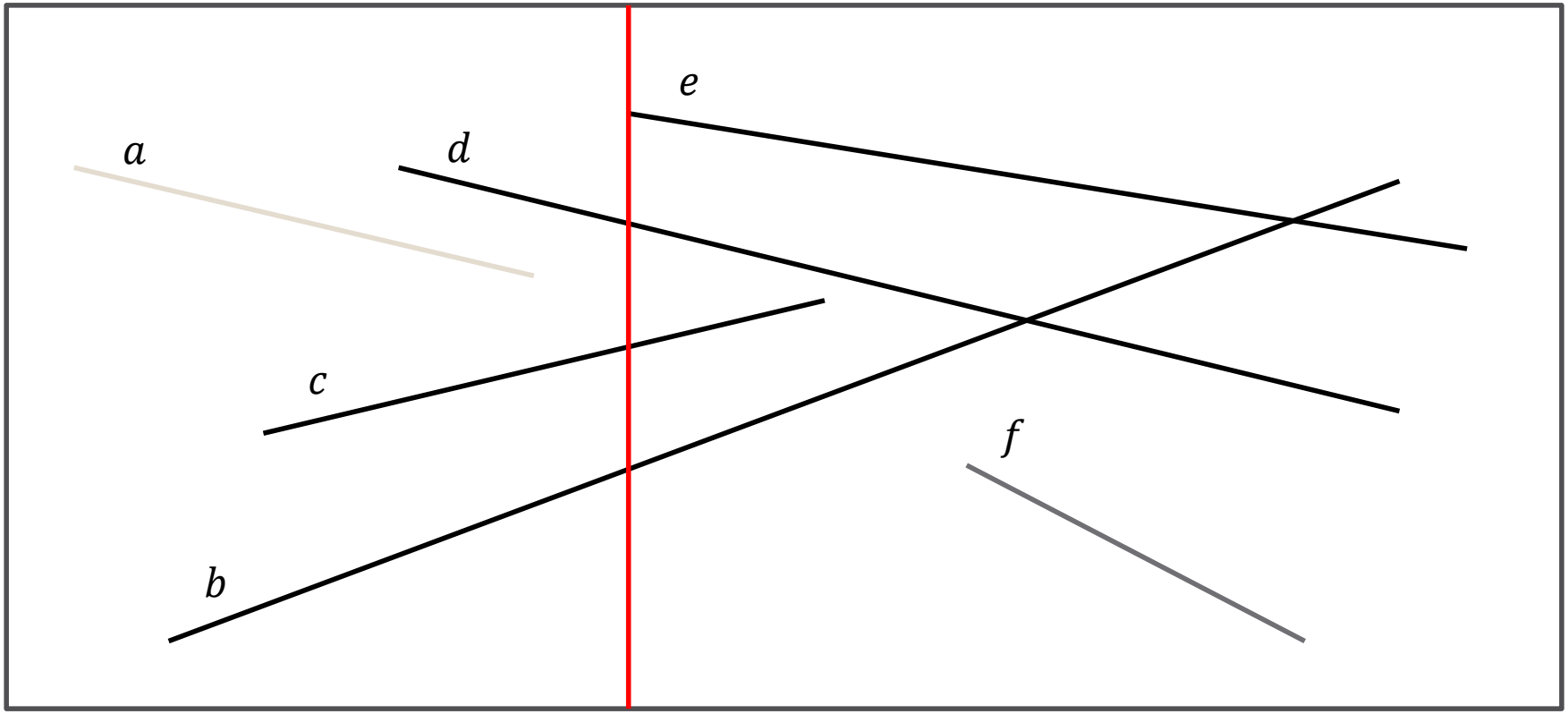
ALGORITAM (primjer izvršavanja)



$T = \{b, c, d\}$

d i c se ne sijeku

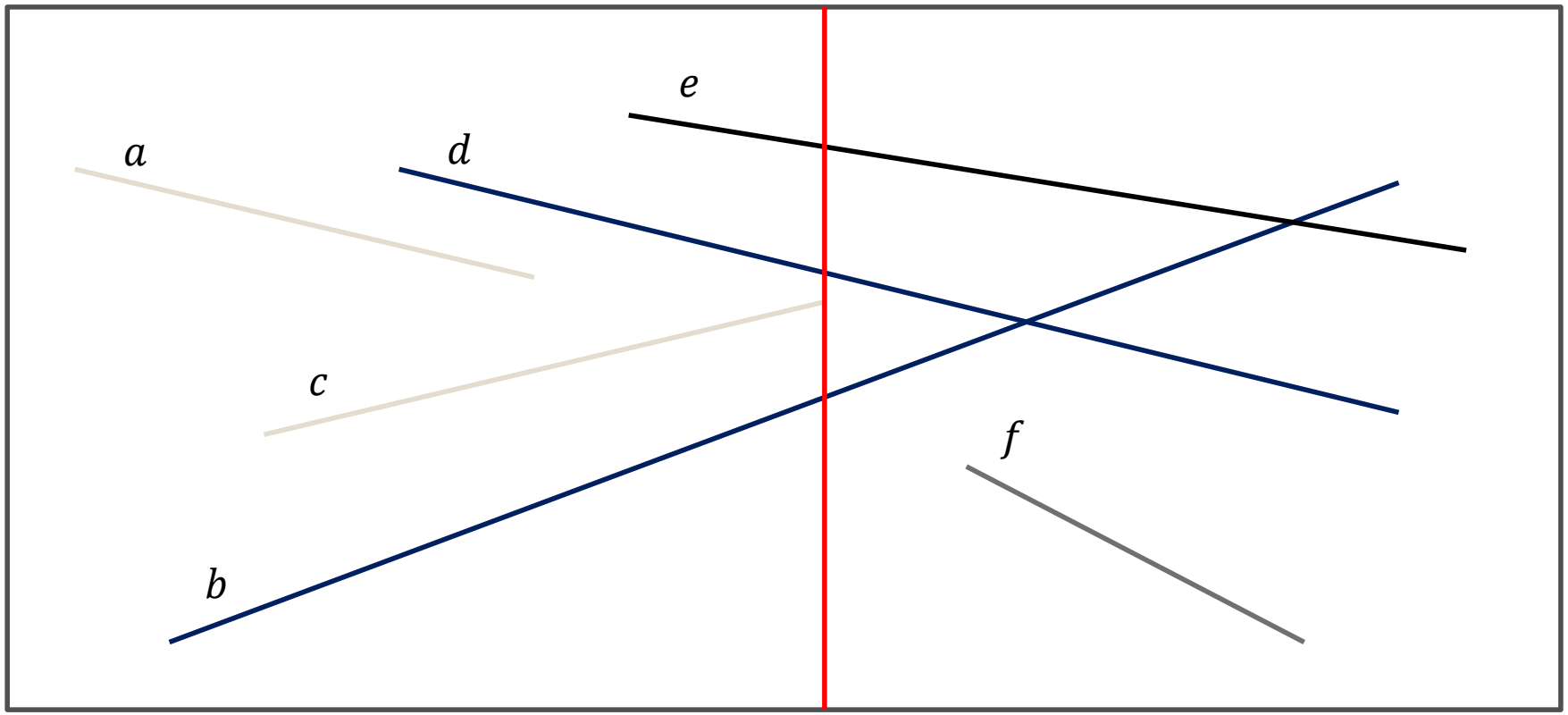
ALGORITAM (primjer izvršavanja)



$T = \{b, c, d, e\}$

e i d se ne sijeku

ALGORITAM (primjer izvršavanja)



$T = \{b, d, e\}$

b i d se sijeku

ALGORITAM (pseudokod)

anySegmentsIntersect(L)

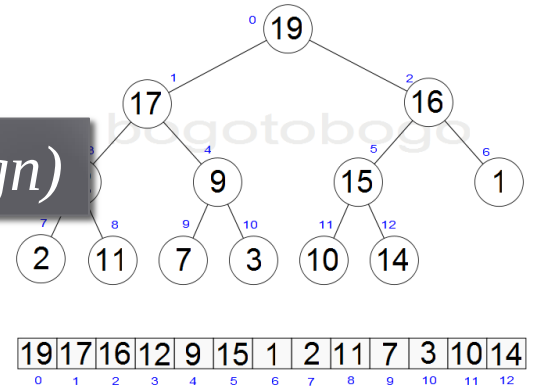
1. T = ∅
2. sort the endpoints of the segments in L using $\prec$
3. for each point p in sorted list of endpoints
4. if p is the left endpoint of a segment l
5. **insert**(T,l)
6. if s = **above**(T, l) exist
7. if **intersects**(s,l)
8. return $true$
9. if s = **below**(T, l) exist
10. if **intersects**(s, l)
11. return $true$
12. if p is the right endpoint of a segment l
13. if s = **above**(T, l) exist and t = **below**(T, l) exist
14. if **intersects**(s,t)
15. return $true$
16. **delete**(T, l)
17. return $false$

- *Sortiranje rubnih točaka*

Heap (hrpa)

Heapsort

O(nlogn)



ALGORITAM (analiza)

anySegmentsIntersect(L)

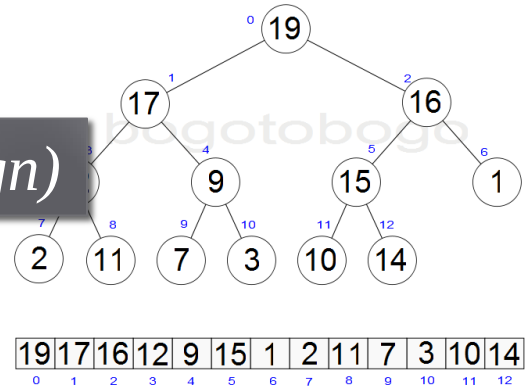
1. T = ∅
2. sort the endpoints of the segments in L using $\prec$ — *O(nlogn)*
3. for each point p in sorted list of endpoints
4. if p is the left endpoint of a segment l
5. **insert**(T,l)
6. if s = **above**(T, l) exist
7. if **intersects**(s,l)
8. return $true$
9. if s = **below**(T, l) exist
10. if **intersects**(s, l)
11. return $true$
12. if p is the right endpoint of a segment l
13. if s = **above**(T, l) exist and t = **below**(T, l) exist
14. if **intersects**(s,t)
15. return $true$
16. **delete**(T, l)
17. return $false$

- *Sortiranje rubnih točaka*

Heap (hrpa)

Heapsort

O(nlogn)



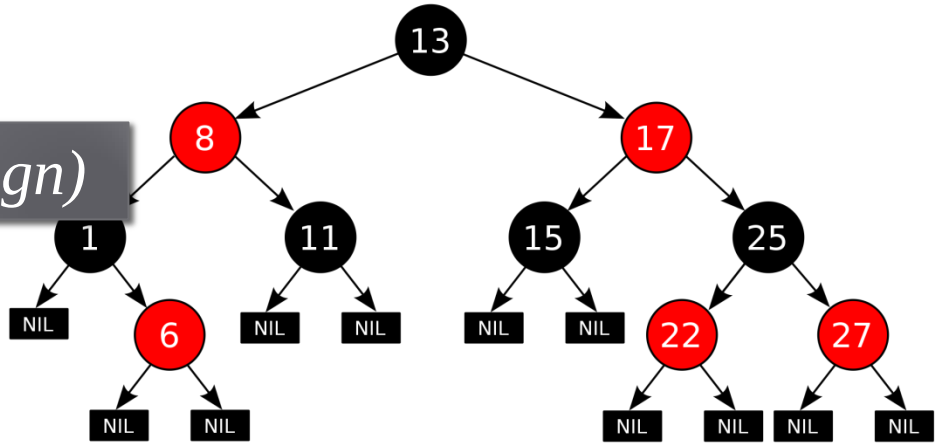
- *Uređaj T*

Red – black tree

Poseban slučaj samobalansirajućeg binarnog stabla

Above(T, l)
Bellow(T, l)
Insert(T, l)
Delete(T, l)

O(logn)



ALGORITAM (analiza)

anySegmentsIntersect(L)

1. T = ∅
2. sort the endpoints of the segments in L using $\prec$ — *O(nlogn)*
3. for each point p in sorted list of endpoints
4. if p is the left endpoint of a segment l
5. **insert**(T,l) — *O(logn)*
6. if s = **above**(T, l) exist — *O(logn)*
7. if **intersects**(s,l)
8. return $true$
9. if s = **below**(T, l) exist — *O(logn)*
10. if **intersects**(s, l)
11. return $true$
12. if p is the right endpoint of a segment l
13. *O(logn)* if s = **above**(T, l) exist and t = **below**(T, l) exist
14. if **intersects**(s,t)
15. return $true$
16. **delete**(T, l) — *O(logn)*
17. return $false$

ALGORITAM (analiza)

Znamo otprije…

anySegmentsIntersect(L)

1. T = ∅
2. sort the endpoints of the segments in L using $\prec$ — $O(nlogn)$
3. for each point p in sorted list of endpoints
4. if p is the left endpoint of a segment l
5. **insert**(T,l) — $O(logn)$
6. if s = **above**(T, l) exist — $O(logn)$
7. if **intersects**(s,l) — $O(1)$
8. return $true$
9. if s = **below**(T, l) exist — $O(logn)$
10. if **intersects**(s, l) — $O(1)$
11. return $true$
12. if p is the right endpoint of a segment l
13. if s = **above**(T, l) exist and t = **below**(T, l) exist — $O(logn)$
14. if **intersects**(s,t) — $O(1)$
15. return $true$
16. **delete**(T, l) — $O(logn)$
17. return $false$

ALGORITAM (analiza)

anySegmentsIntersect(L)

1. T = ∅
2. sort the endpoints of the segments in L using $\prec$ — $O(nlogn)$
3. for each point p in sorted list of endpoints
4. if p is the left endpoint of a segment l
5. **insert**(T,l) — $O(logn)$
6. if s = **above**(T, l) exist — $O(logn)$
7. if **intersects**(s,l) — $O(1)$
8. return $true$
9. if s = **below**(T, l) exist — $O(logn)$
10. if **intersects**(s, l) — $O(1)$
11. return $true$
12. if p is the right endpoint of a segment l
13. if s = **above**(T, l) exist and t = **below**(T, l) exist — $O(logn)$
14. if **intersects**(s,t) — $O(1)$
15. return $true$
16. **delete**(T, l) — $O(logn)$
17. return $false$

$2n$ puta

Zaključak

Vremenska složenost algoritma je $O(nlogn)$

Najgori slučaj (*u punom smislu, naime algoritam staje čim nađe **neki** presjek*)

!

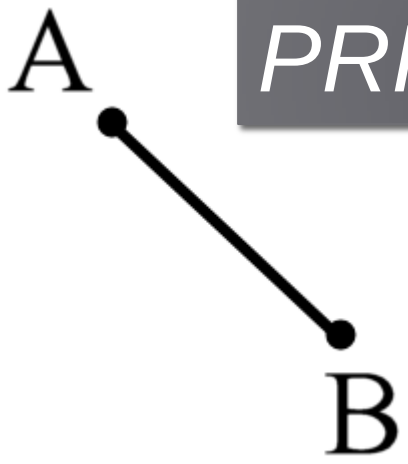
Prostorna složenost algoritma je $O(n)$

Teorem

Poziv funkcije *anySegmentsIntersect(L)* vraća $true$ ako i samo ako postoje l_1 i l_2 iz L koje se sijeku.

Q.E.D.

PRESJEK SEGMENATA U RAVNINI

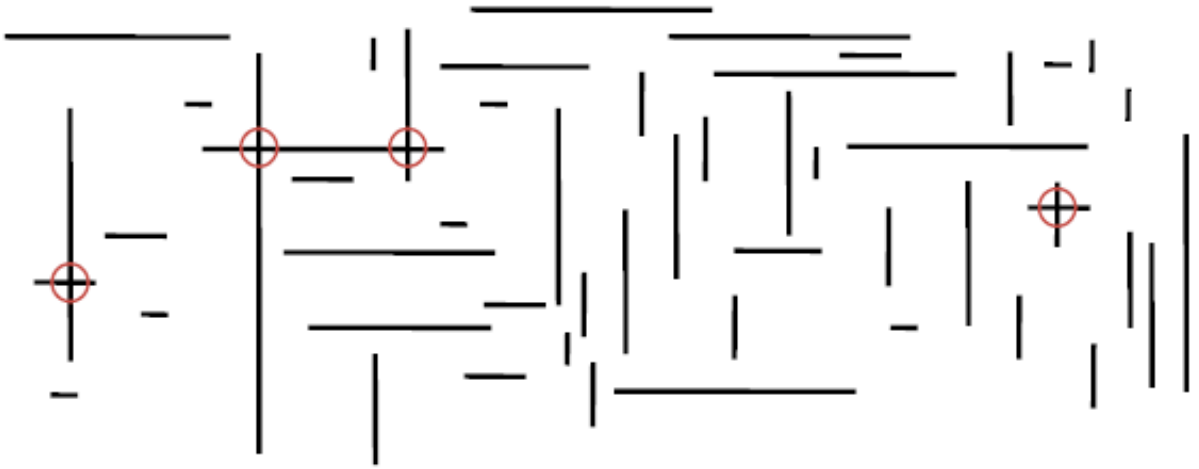


PRIMJER

PROBLEM:

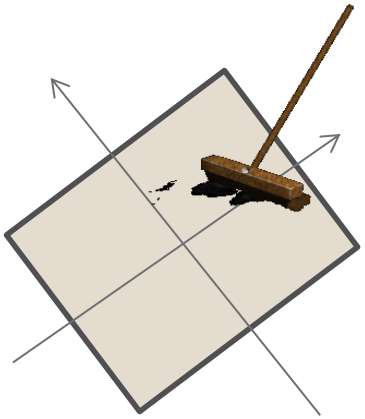
Dan je skup $L = \{l_1, l_2, \ldots, l_n\}$ koji sadrži n segmenata u ravnini koji su isključivo ili horizontalni ili vertikalni. Potrebno je pronaći sve segmente koji se sijeku. Pretpostavimo da su sve rubne točke različite.

Brute force algoritam $O(n^2)$



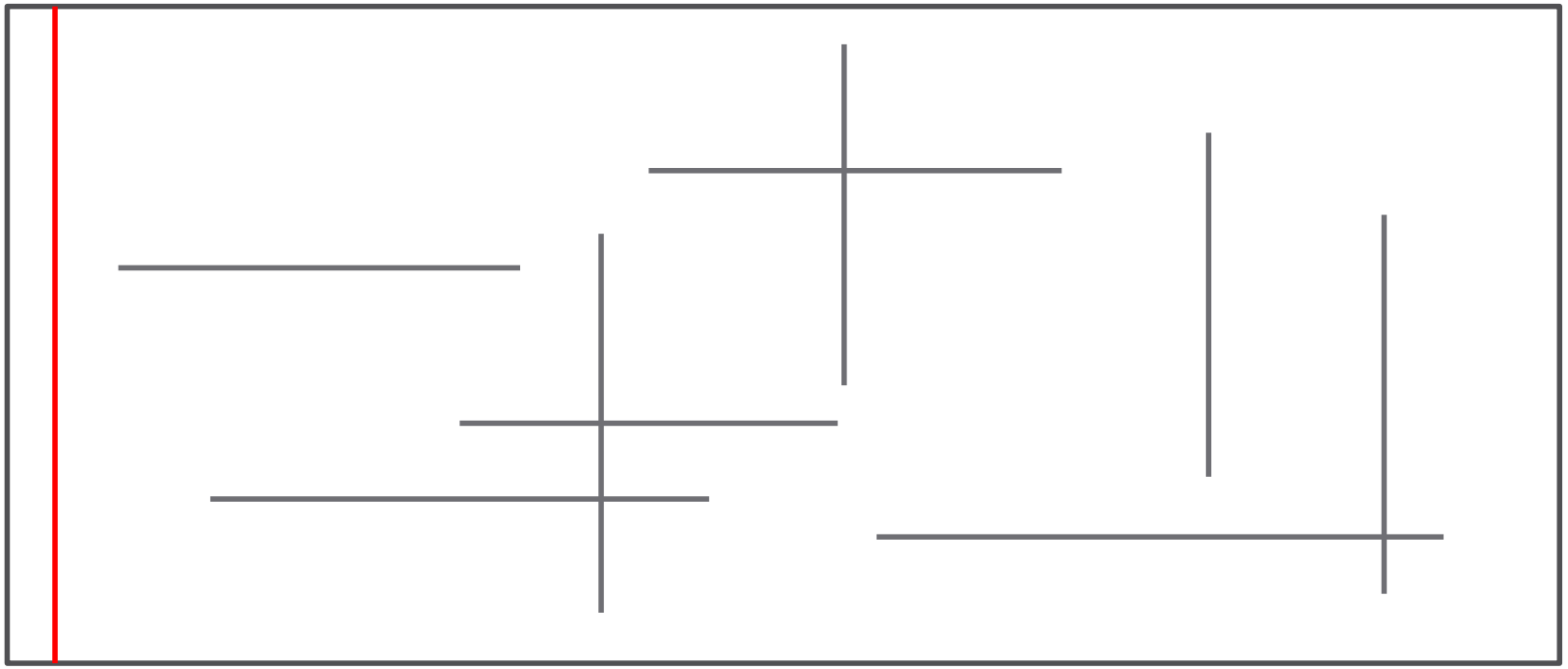
Ideja!

SWEEP LINE algoritam



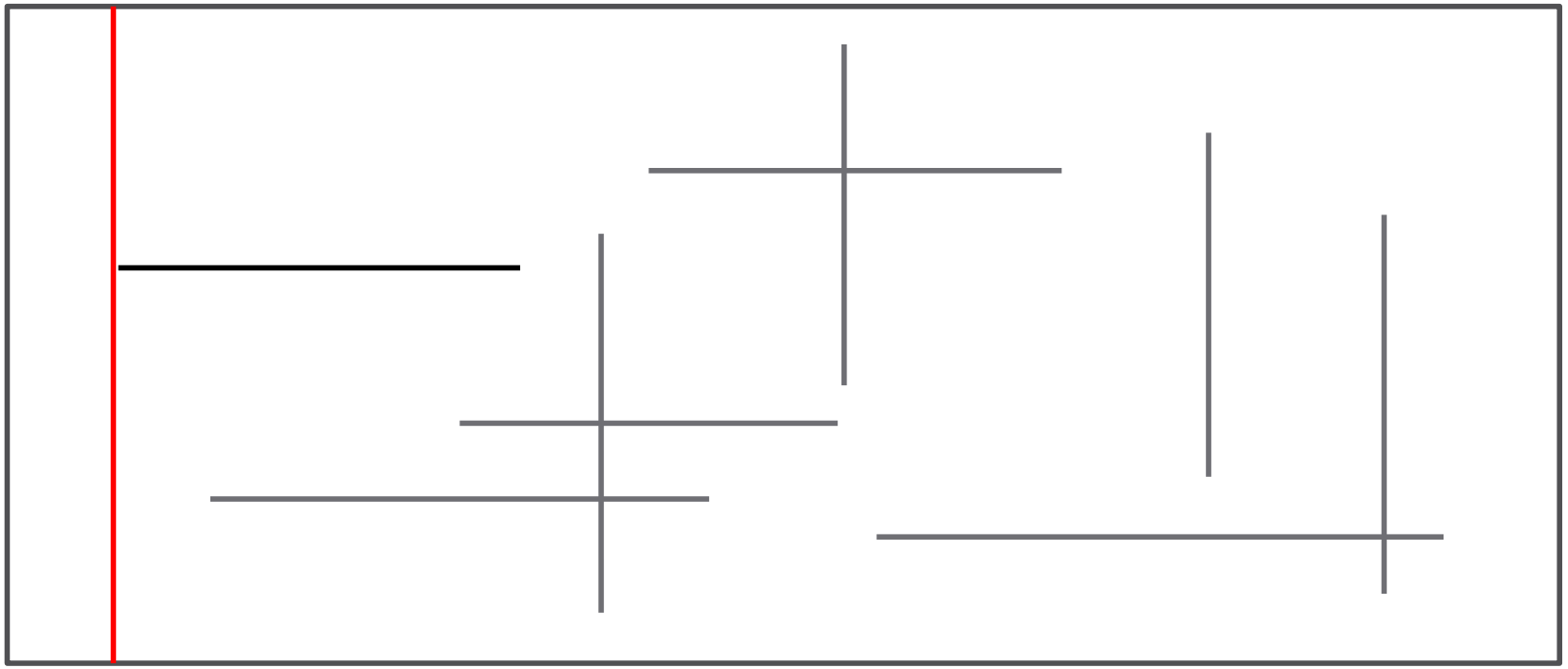
Orthogonal line segment intersection

- Pomičemo SWEEP LINE (koristimo ranije opisan uređaj)



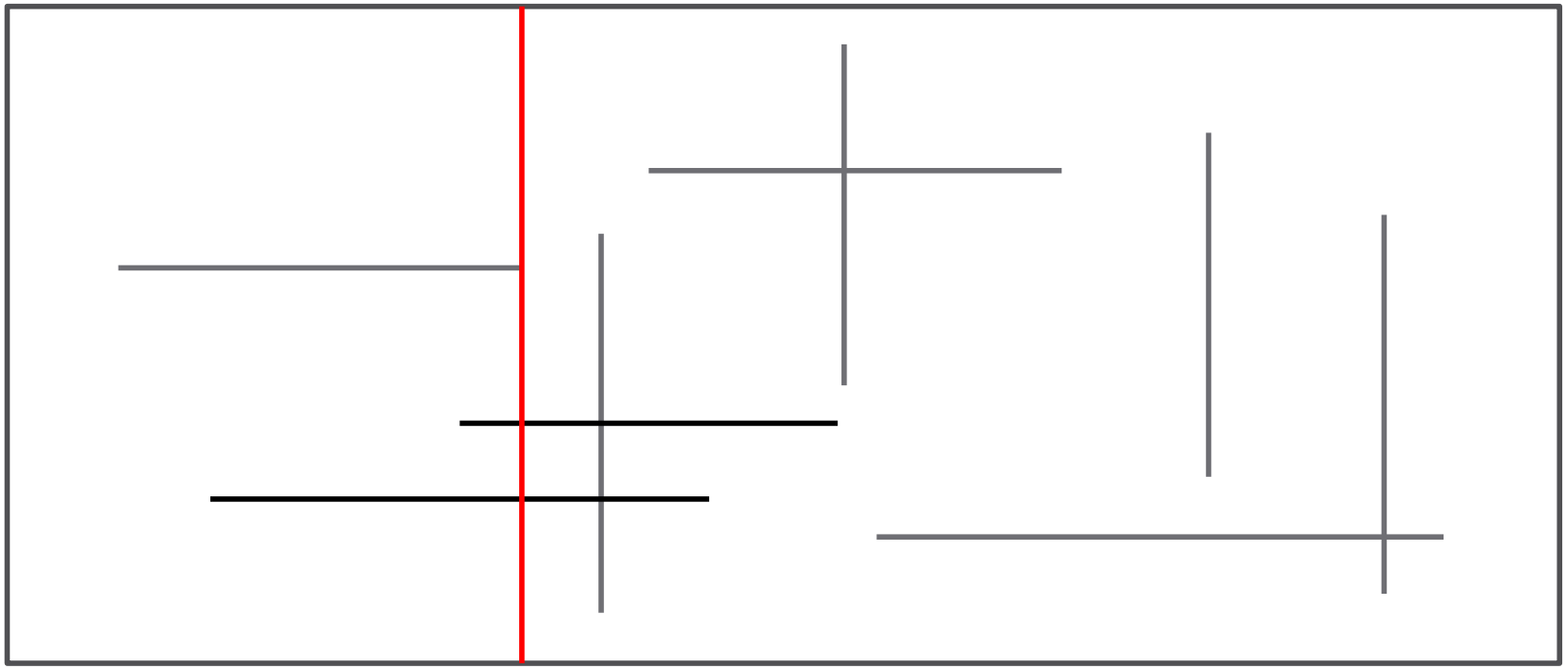
Orthogonal line segment intersection

- Pomičemo *SWEEP LINE* (koristimo ranije opisan uređaj)
- Horizontalni segment – lijevi rub: stavljamo segment u uređaj T (*horizontalne segmente uspoređujemo po pripadnoj y-koordinati*)



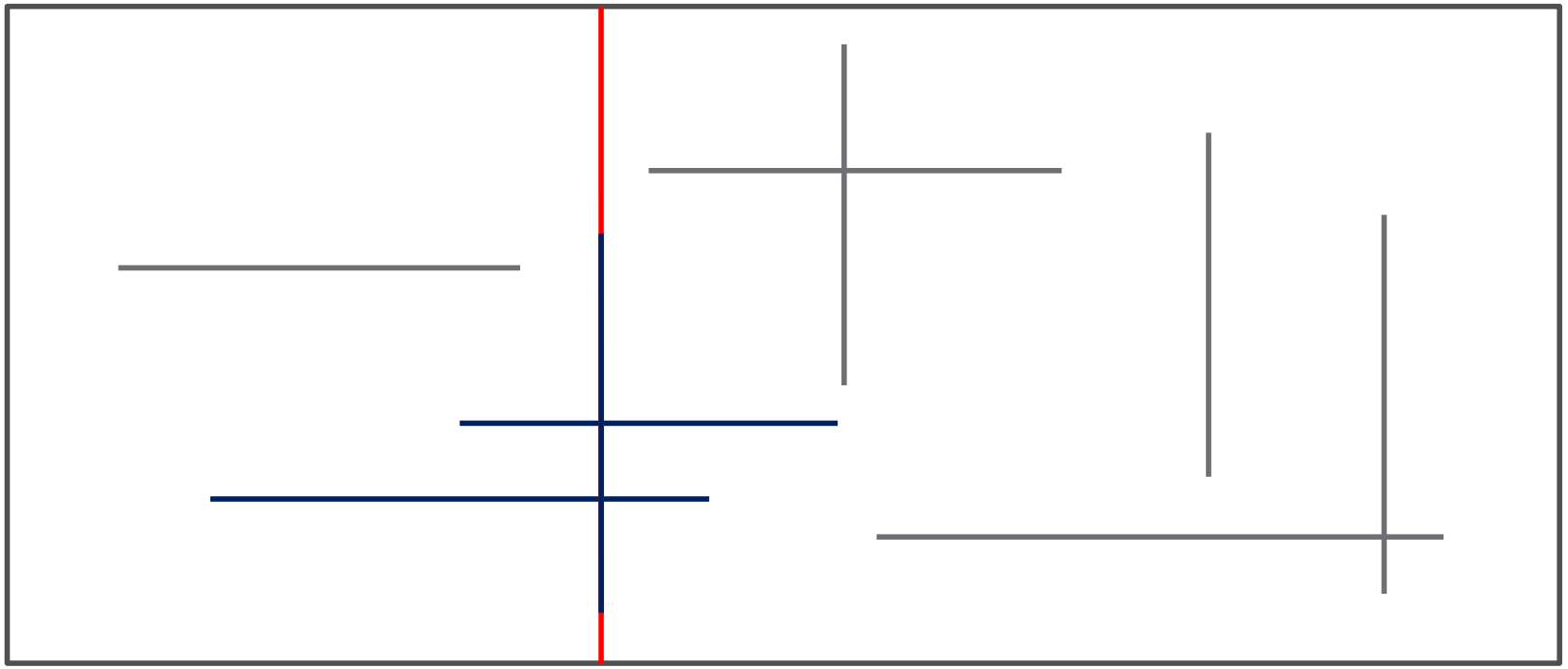
Orthogonal line segment intersection

- Pomičemo *SWEEP LINE* (koristimo ranije opisan uređaj)
- Horizontalni segment – lijevi rub: stavljamo segment u uređaj T (*horizontalne segmente uspoređujemo po pripadnoj y-koordinati*)
- Horizontalni segment – desni rub: mičemo segment iz uređaja T



Orthogonal line segment intersection

- Pomičemo *SWEEP LINE* (koristimo ranije opisan uređaj)
- Horizontalni segment – lijevi rub: stavljamo segment u uređaj T (*horizontalne segmente uspoređujemo po pripadnoj y-koordinati*)
- Horizontalni segment – desni rub: mičemo segment iz uređaja T
- Vertikalni segment – provjeravamo presjek sa segmentima u uređaju *Range search po y-koordinati*



Propozicija

Vrijeme izvršavanja *orthogonalSegmentsIntersect* algoritma je proporcionalno $nlogn + r$ gdje je n broj segmenata, a r broj točaka presjeka.

Dokaz

- Sortiranje rubnih točaka (*heapsort*)
- Stavljanje segmenata u uređaj (*Red-black tree*)
- Izbacivanje segmenata iz uređaja (*Red-black tree*)
- Provjera postoji li presjek (*Range search u RBT*)

nlogn

nlogn + r

Q.E.D.

Dakle, vremenska složenost je O(nlogn).

Propozicija

Vrijeme izvršavanja *orthogonalSegmentsIntersect* algoritma je proporcionalno $nlogn + r$ gdje je n broj segmenata, a r broj točaka presjeka.

Dokaz

- Sortiranje rubnih točaka (*heapsort*)
- Stavljanje segmenta u uređaj (*Red-black tree*)
- Izbacivanje segmenta iz uređaja (*Red-black tree*)
- Provjera postoji li presjek (*Range search u RBT*)

nlogn

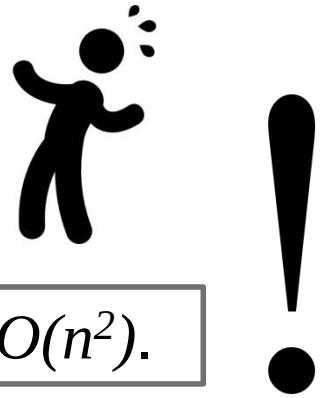
nlogn + r

Q.E.D.

Dakle, vremenska složenost je O(nlogn).

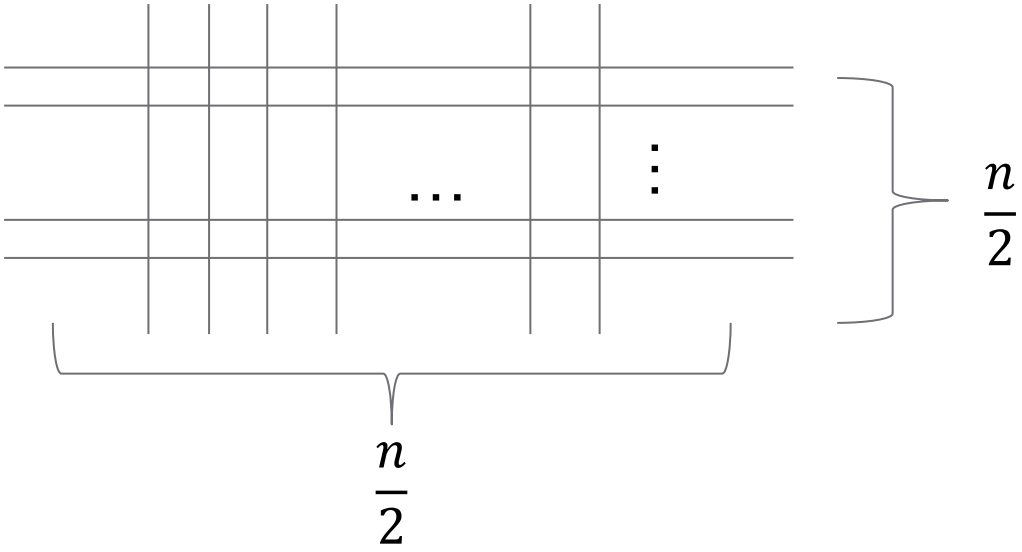
JESMO LI SIGURNI?

Problem…



Propozicija

Broj presjeka u skupu od n segmenata je $O(n^2)$.



$\frac{n^2}{4}$

Teško je predvidjeti ponašanje algoritma za *slučajno generirane* test primjere!

Odrediti vremensku složenost za geometrijske probleme nije jednostavno!

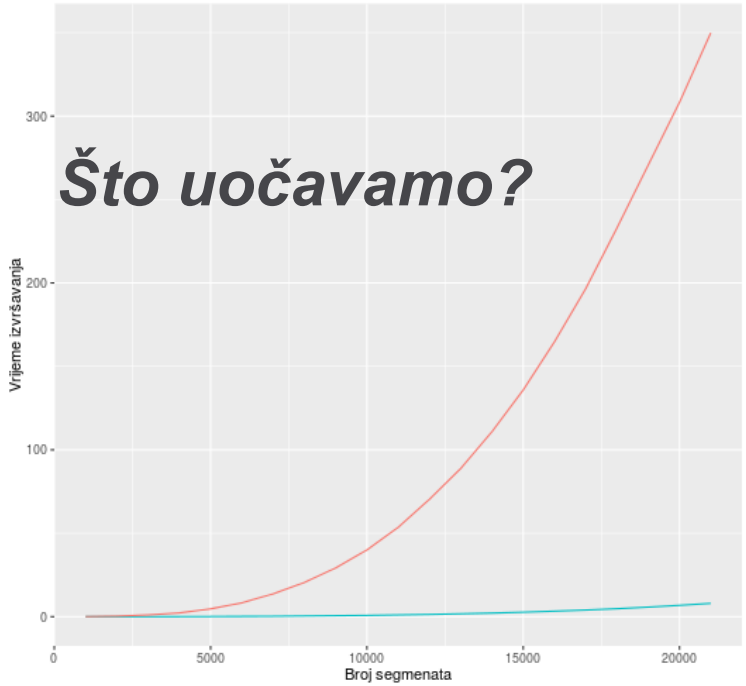
Rezultati

n	t/SL	t/BF
128	0.0076	0.0038
256	0.0084	0.0086
512	0.0097	0.0276
1024	0.0133	0.0993
2048	0.0251	0.3935
4096	0.0667	1.6299
8192	0.2076	11.0203
16384	0.8526	39.1816

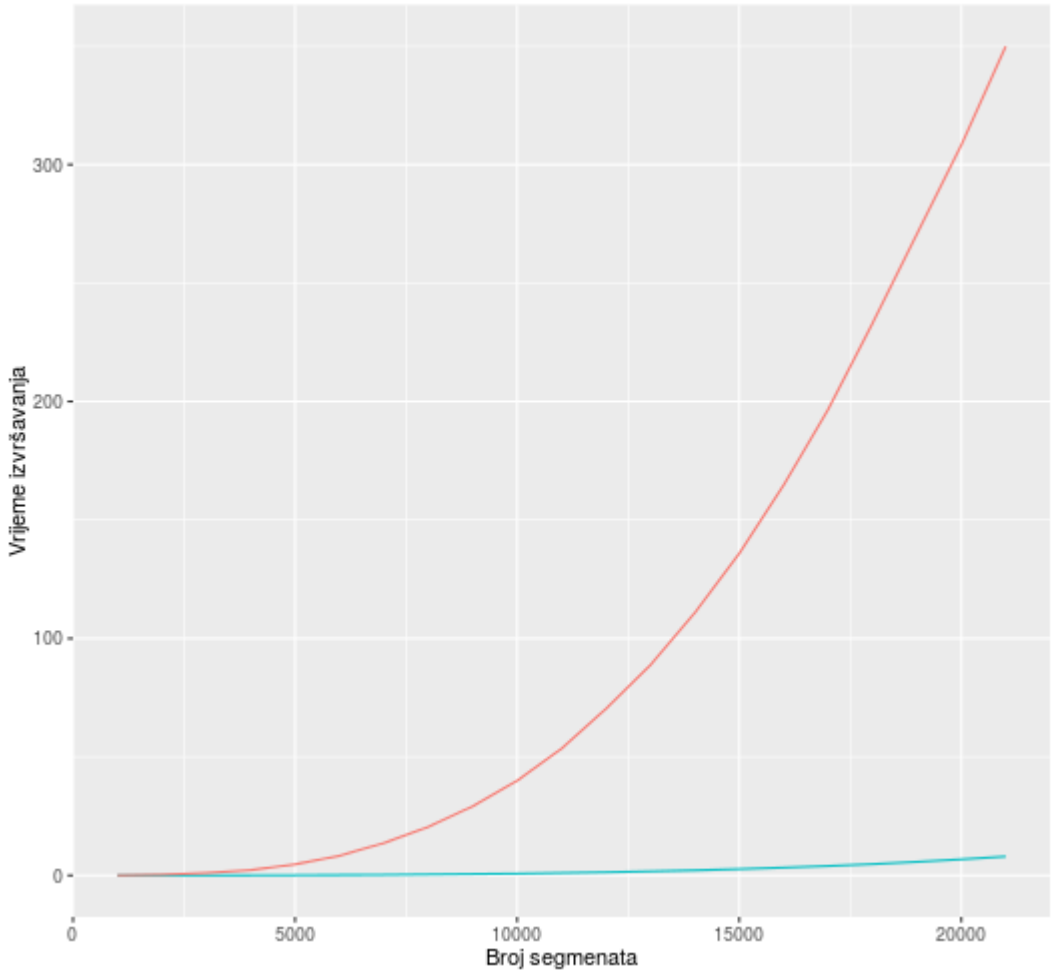
n - broj dužina
SL – *Sweep line* algoritam
BF – *Brute Force* algoritam

!

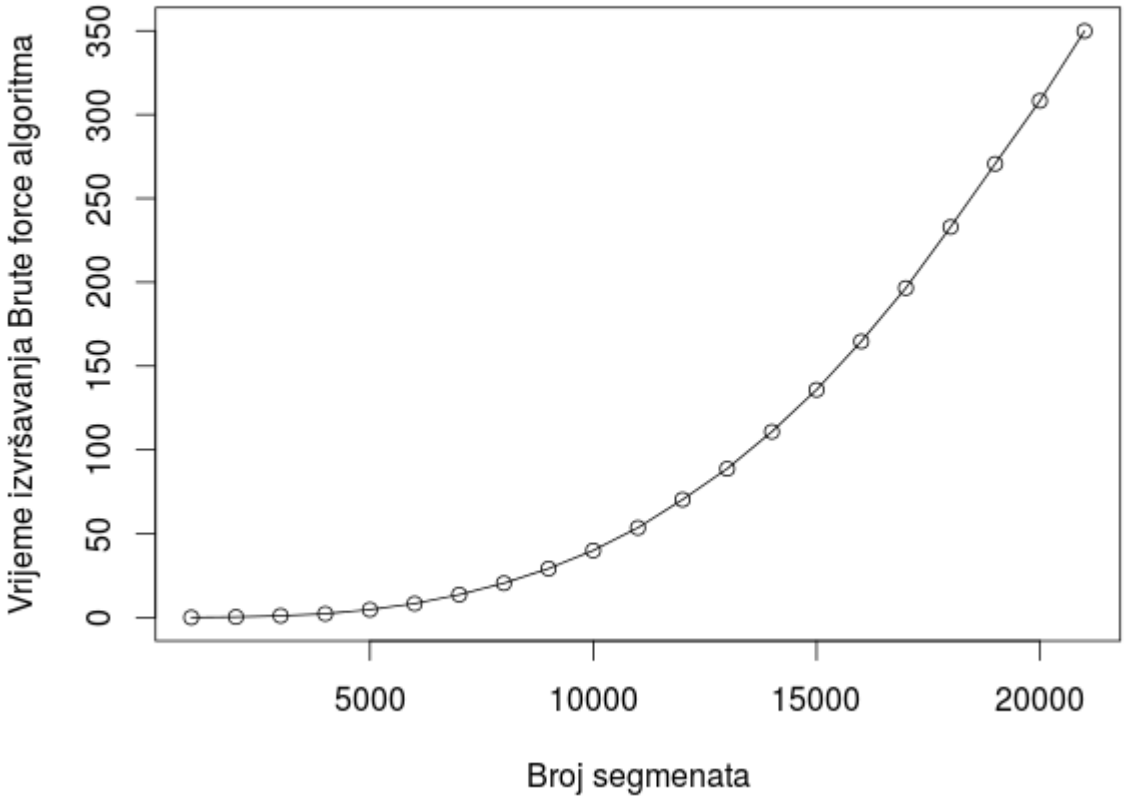
4x



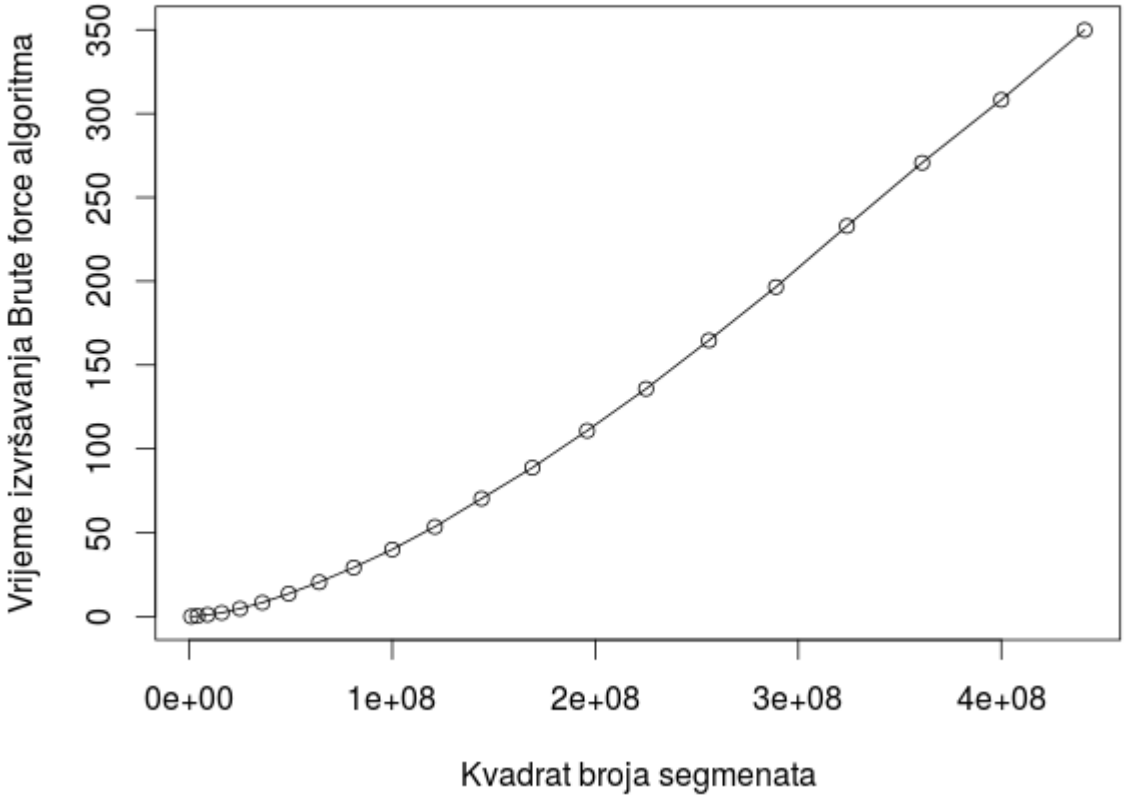
Rezultati


300
200
100
0
Vrijeme izvršavanja
0
5000
10000
15000
20000
Broj segmenata

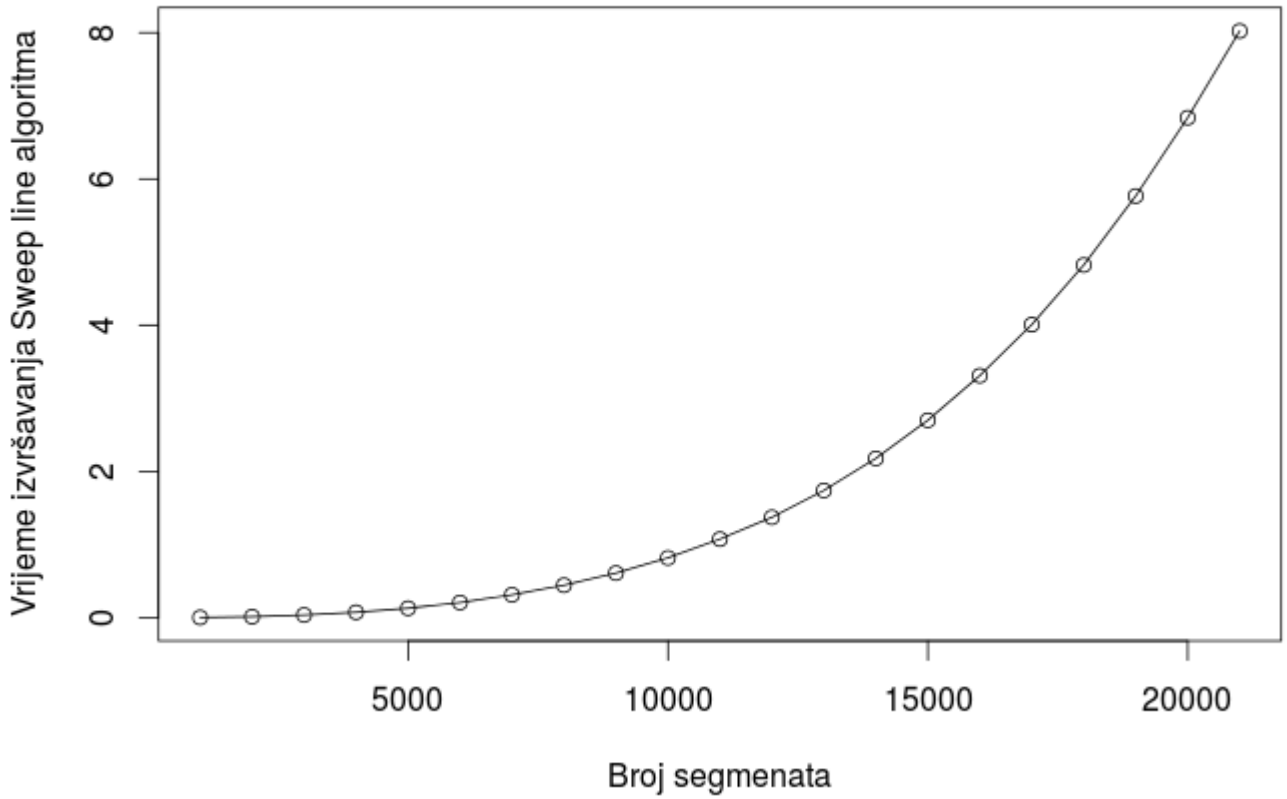
Rezultati


Vrijeme izvršavanja Brute force algoritma
0
50
100
150
200
250
300
350
5000
10000
15000
20000
Broj segmenata

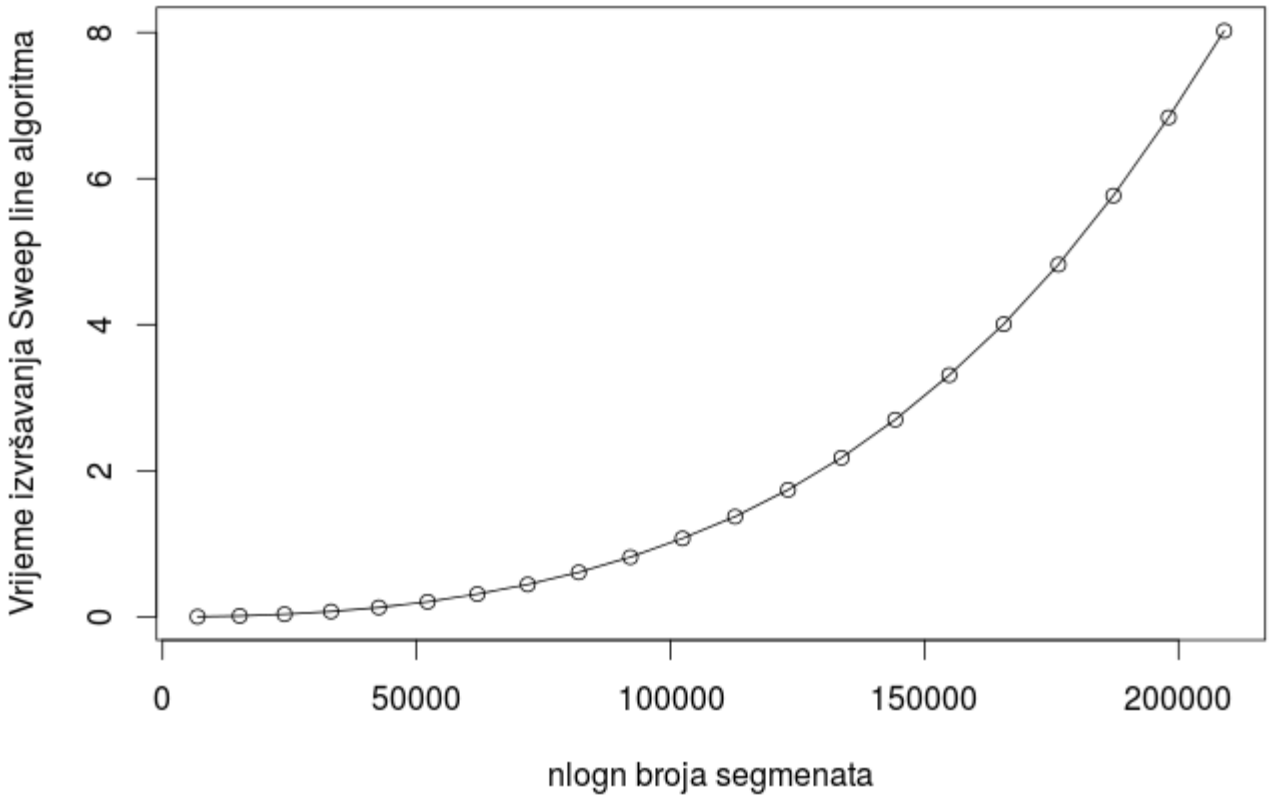
Rezultati


Vrijeme izvršavanja Brute force algoritma
0
50
100
150
200
250
300
350
0e+00
1e+08
2e+08
3e+08
4e+08
Kvadrat broja segmenata

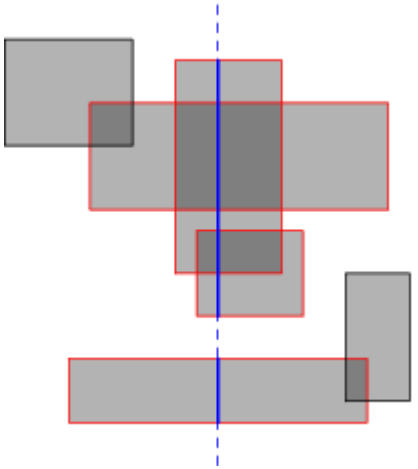
Rezultati


Vrijeme izvršavanja Sweep line algoritma
0
2
4
6
8
5000
10000
15000
20000
Broj segmenata

Rezultati


Vrijeme izvršavanja Sweep line algoritma
0
2
4
6
8
0
50000
100000
150000
200000
nlogn broja segmenata

Primjeri i komentar rezultata…



Literatura

[1] T. H. Cormen, C. E. Leisersos, R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 2000.

[2] T. H. Cormen, C. E. Leisersos, R. L. Rivest, *Introduction to Algorithms, Second Edition*, The MIT Press, Cambridge, Massachusetts, 2001.

[3] M. H. Alsuwaiyel, *Algorithms Design Techniques and Analysis,* World Scientific Publishing Co. Pte. Ltd., 1999.

[4] R. Sedgewick, K, Wayne, *Algorithms, Fourth Edition*, Addison – Wesley, 2011.