STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

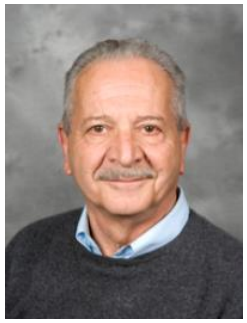
Dátové štruktúry a algoritmy

Výpočtová geometria

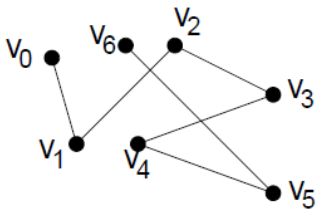
28. 11. 2017

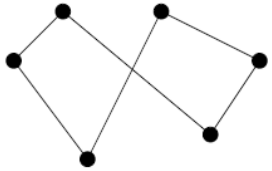
zimný semester 2017/2018

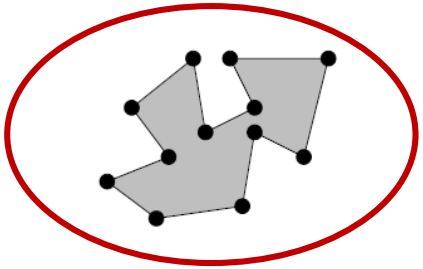
Výpočtová geometria



- Geometrické algoritmy
- Začiatky 1975 – Shamos, Preparata
- Spracovanie geometrických objektov:
  - Bod, Priamka, Úsečka,
  - Trojuholník, Mnohouholník,
  - Kružnica
  - ...
- Čo sa týka mnohouholníkov, budú nás zaujímať len jednoduché (nepretínajúce sa) mnohouholníky





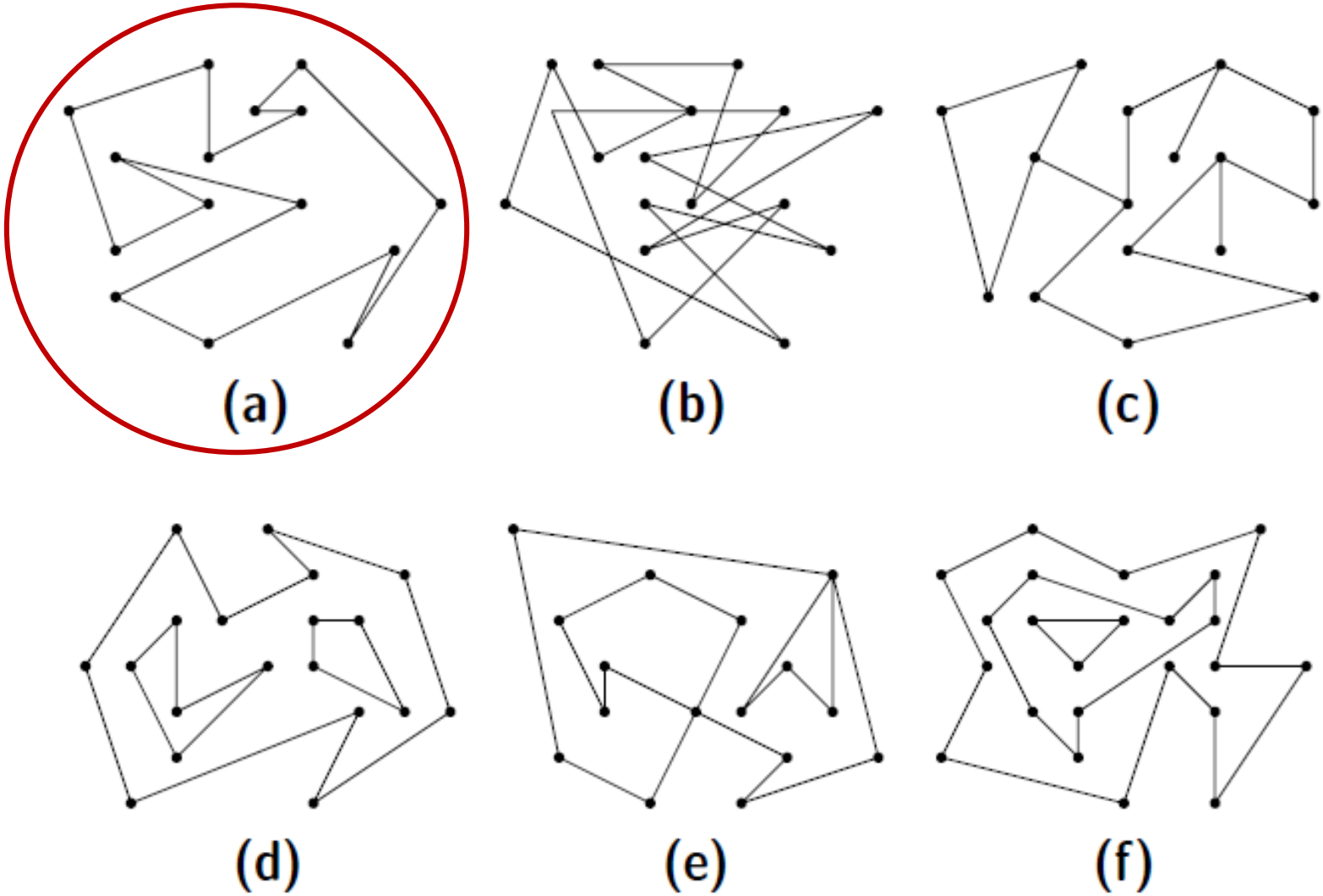


Obmedzenia algoritmov

- Človek vs počítač
  - Pre človeka ľahké (metóda pozriem vidím)
  - Pre počítač náročné (výpočet naslepo)

- Špeciálne (až degenerované) prípady
  - Kolineárnosť bodov
  - Splývajúce úsečky
  - Degenerované uhly
- Presnosť výpočtov
  - Zaokrúhľovanie

Čo z tohto je mnohouholník?

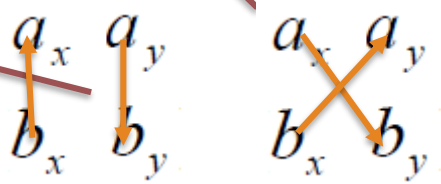


Analytická geometria – priamka

- Výpočty „naslepo“
- Bod v dvojrozmernej euklidovskej rovine E_2
  - Súradnice x, y
- Priamka medzi dvomi bodmi $A=[a_x, a_y]$, $B=[b_x,b_y]$ je rovnica pre množinu bodov $[x,y] \in E_2$

$$(a_y - b_y)x + (b_x - a_x)y + (a_x b_y - a_y b_x) = 0$$

- Mnemotechnická pomôcka:
  (kreslíme domček sprava hore)



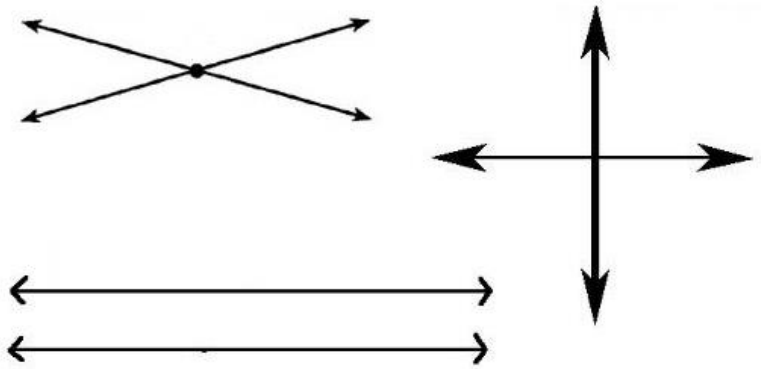
- Čo to znamená?
  - Pre niektoré x,y rovnica platí, pre iné neplatí.
  - Body pre ktoré rovnica platí sú na tejto priamke.
  - Kde sú tie ostatné?

Analytická geometria – dve priamky

- Uvažujme dve priamky:

$$p : a_1 x + b_1 y + c_1 = 0$$

$$q : a_2 x + b_2 y + c_2 = 0$$



- Zaujíma nás ich vzájomná poloha.
- Aká môže byť vzájomná poloha?
  - Rovnobežné, totožné
  - Rôznobežné – kde je priesečník

Analytická geometria – dve priamky (2)

- Uvažujme dve priamky:

$$p : a_1 x + b_1 y + c_1 = 0$$
$$q : a_2 x + b_2 y + c_2 = 0$$

- Vypočítame determinant D:

$$D = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \qquad \mathrm{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \qquad |\mathrm{A}| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

- Ak D=0, priamky sú rovnobežné.
  - Ako zistíme totožnosť?

Analytická geometria – dve priamky (3)

- Uvažujme dve priamky:

$$p : a_1 x + b_1 y + c_1 = 0$$
$$q : a_2 x + b_2 y + c_2 = 0$$

$$D = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$$

- Ak D≠0, priamky sú rôznobežné.
  - Ako určíme priesečník *X*?

$$X = [\frac{D_1}{D}, \frac{D_2}{D}]$$

$$D_1 = \begin{vmatrix} -c_1 & b_1 \\ -c_2 & b_2 \end{vmatrix}$$

$$D_2 = \begin{vmatrix} a_1 & -c_1 \\ a_2 & -c_2 \end{vmatrix}$$

- Ako cvičenie si zopakujte tento postup výpočtom s využitím smernice...

Analytická geometria – vzdialenosť od priamky

- Bod M=[m,n] a priamka p: $ax+by+c_1 = 0$
- (kolmá) vzdialenosť bodu M od priamky p

$$|M, p| = \frac{|am + bn + c_1|}{\sqrt{a^2 + b^2}}$$

- Priamka p a rovnobežná priamka q: $ax+by+c_2 = 0$
- Vzdialenosť p od q:

$$|p, q| = \frac{|c_1 - c_2|}{\sqrt{a^2 + b^2}}$$

Analytická geometria – uhol

- Uvažujme priamku $p : a_1 x + b_1 y + c_1 = 0$
- Smerový vektor (rovnobežný s jej smerom) ***s***
- Kolmý (normálový) vektor (nejaký vektor kolmý na s) ***t***

$$\boldsymbol{s} = (-b_1, a_1), \quad \boldsymbol{t} = (a_1, b_1)$$

- Norma $\|\boldsymbol{u}\|$ vektora $\boldsymbol{u} = (u_1, u_2)$ definujme $\|\boldsymbol{u}\| = \sqrt{u_1^2 + u_2^2}$
- Uvažujme aký uhol α zvierajú dva vektory ***u*** a ***v***:

$$\boldsymbol{u} = (u_1, u_2), \boldsymbol{v} = (v_1, v_2)$$

$$u_1 v_1 + u_2 v_2 = \|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\| \cos\alpha$$

- Uhol dvoch priamok?

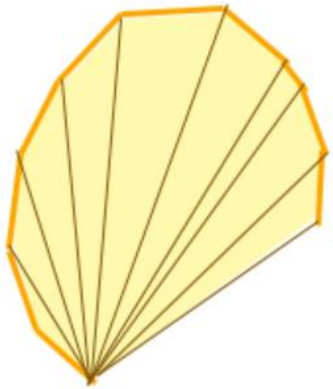
Analytická geometria – obsah

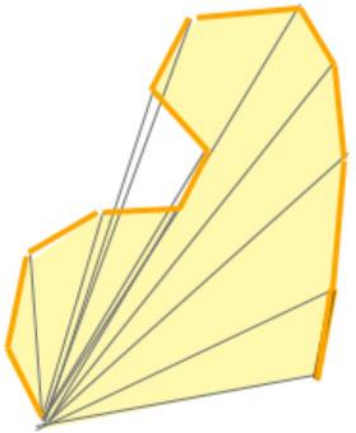
- Obsah rovnobežníka, ktorý určujú vektory u a v bude |S|

$$S = \begin{vmatrix} u_1 & u_2 \\ v_1 & v_2 \end{vmatrix}$$

- Ako vypočítam obsah trojuholníka ABC?
  - Obsah rovnobežníka určeného vektormi deleno 2

- Obsah mnohouholníka?
  - Konvexný je ľahký
    rozdelím na trojuholníky
  - Čo nekonvexný?



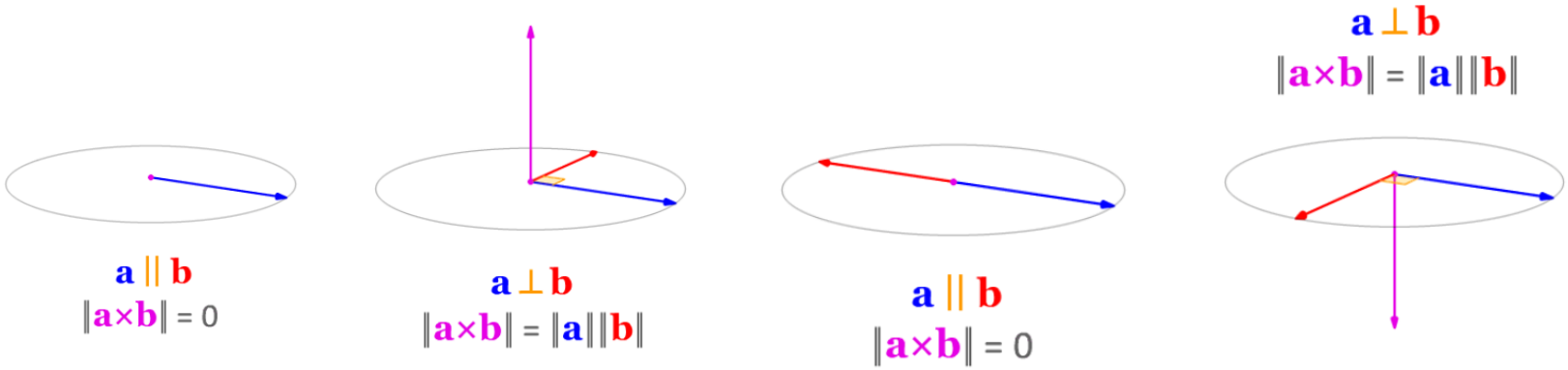


Analytická geometria – orientovaný obsah

- Zaujímavá vlastnosť: obsah S rovnobežníka je orientovaný (má znamienko + / –)

$$S = \begin{vmatrix} u_1 & u_2 \\ v_1 & v_2 \end{vmatrix}$$

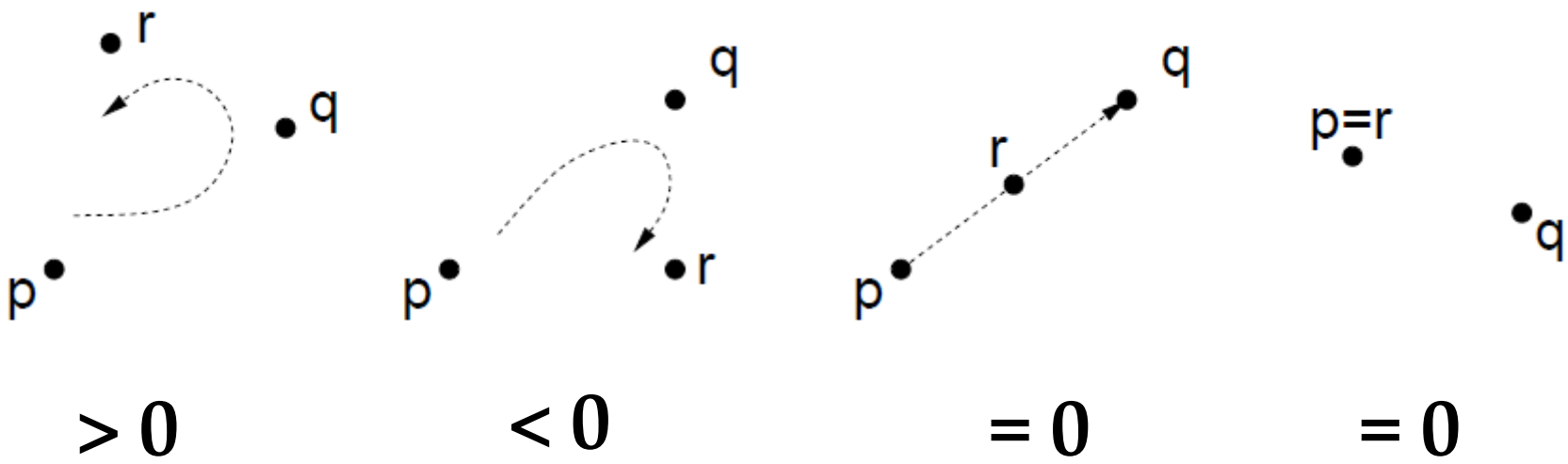
- Podľa čoho sa určí znamienko?



- S > 0 ak sa pri „cestovaní“ po bodoch A → B → C točíme doľava (proti smeru hodinových ručičiek)
- S < 0 ak sa točíme doprava (po smere hodinových ručičiek)

Precvičenie orientácia bodov

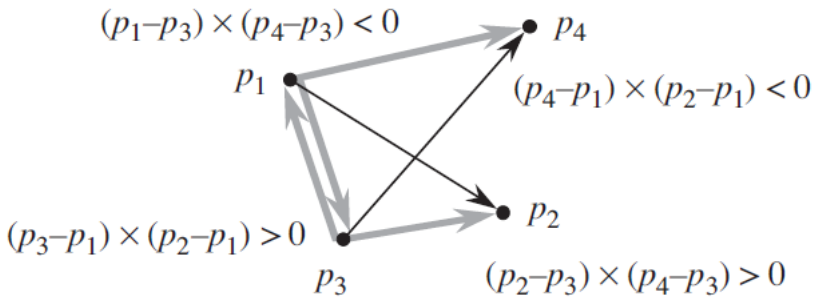
- Daná je trojica bodov (p,q,r) aká je orientácia týchto bodov?

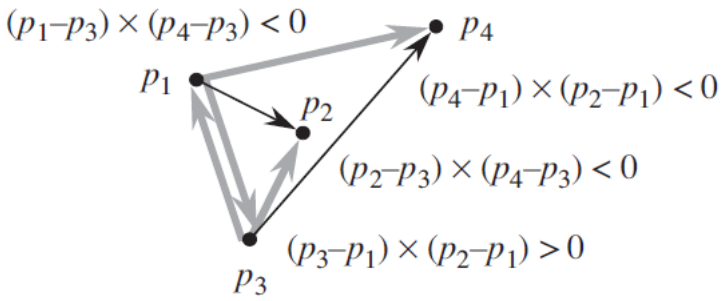


> 0 < 0 $= 0$ $= 0$

Analytická geometria – priesečník úsečiek

- Úsečky ≠ priamky

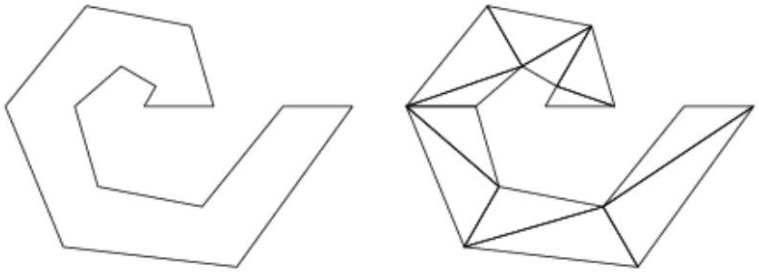




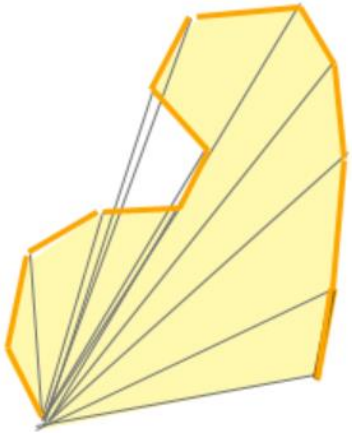
- Riešenie: Preskúmať orientáciu pri prechodoch medzi každou trojicou bodov (p_1, p_2, p_3, p_4)

Výpočtová geometria – obsah mnohouholníka

- Ako vypočítame obsah nekonvexného mnohouholníka?
- Nejako určíme rozklad na trojuholníky – triangulácia



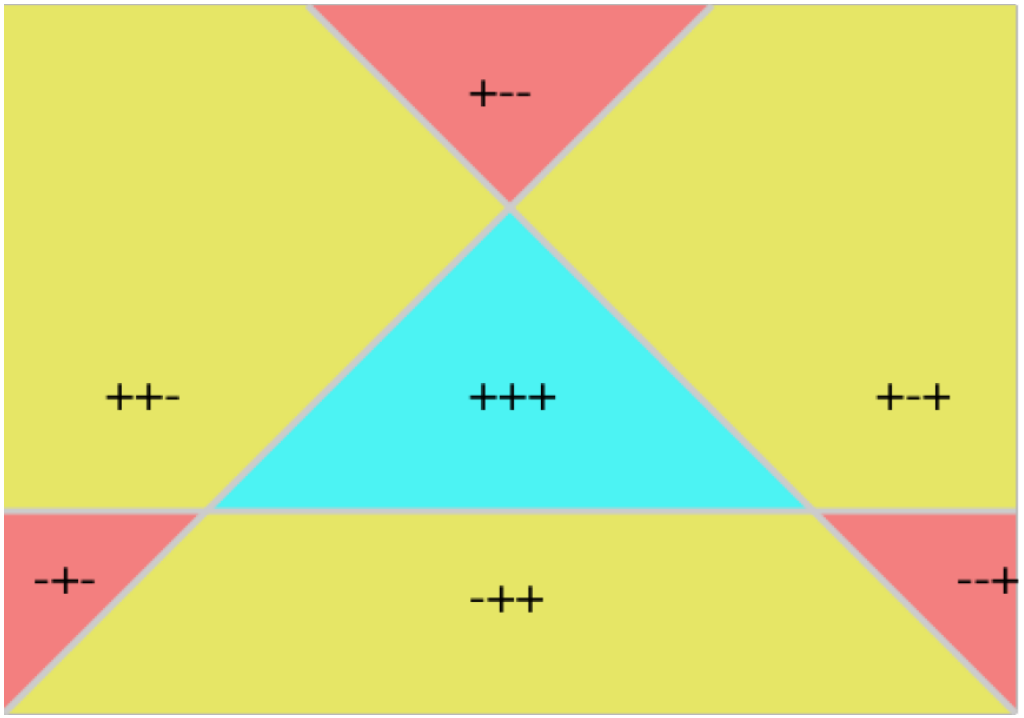
- Využijeme výpočet s orientovaným obsahom
- Funguje!
  - Nekonvexné oblasti sa odpočítajú



Bod v trojuholníku

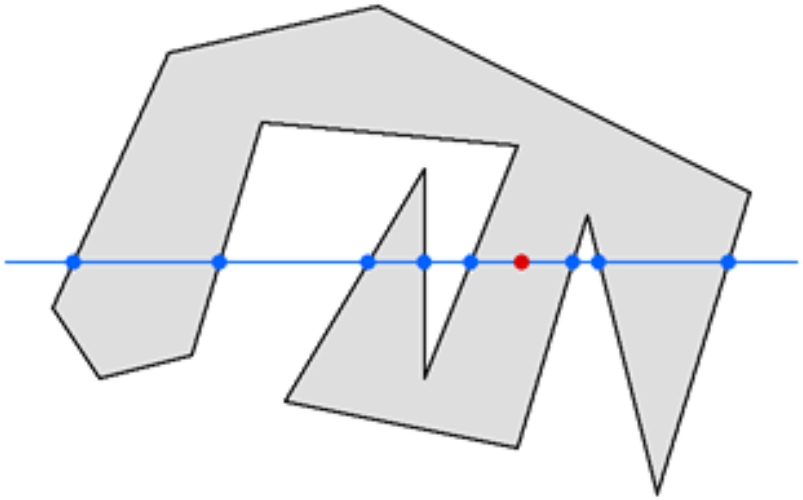
- Nachádza sa bod P v trojuholníku ABC ?

- Preskúmame orientácie k všetkým stranám trojuholníka
  - A → B → P
  - B → C → P
  - C → A → P
- Všetky musia mať rovnaké znamienko ...



Bod v mnohouholníku

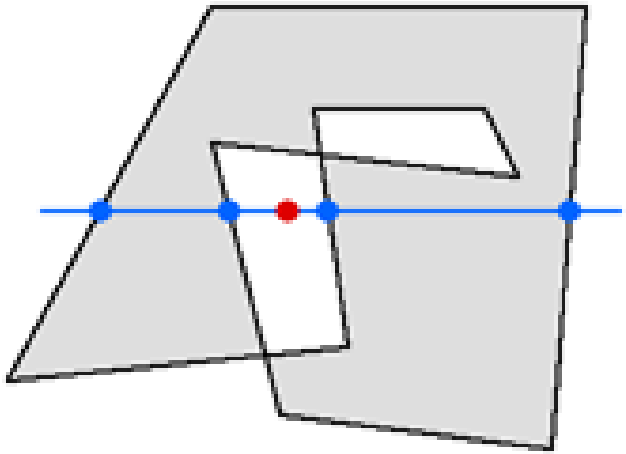
- Vystrelíme polpriamku (lúč – ray) a spočítame počet priesečníkov s obvodom mnohouholníka



- Párny počet = bod je vonku
- Nepárny počet = bod je vnútri

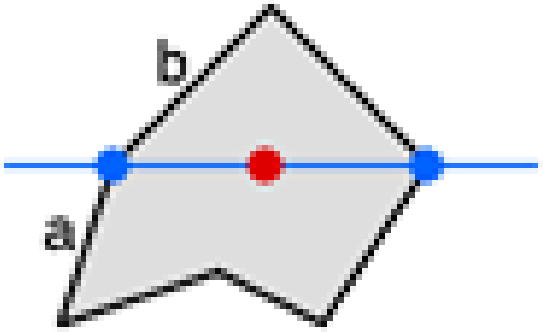
Bod v mnohouholníku (2)

- Pre mnohouholníky, ktoré nie sú jednoduché ...
  to nefunguje vždy:

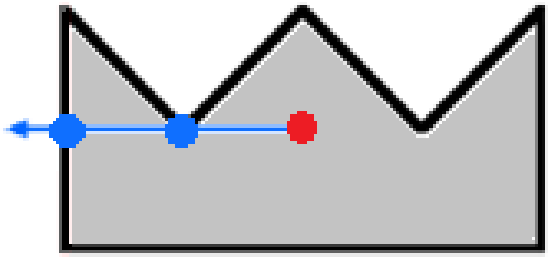


Bod v mnohouholníku (3)

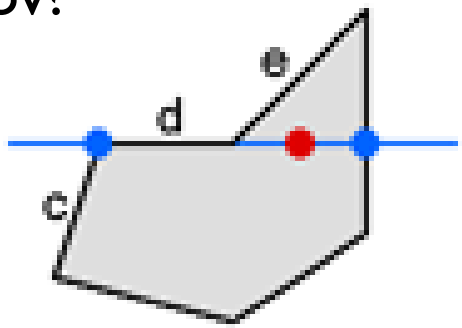
- Postup: prejdeme každú stranu a skontrolujeme, či pretína polpriamku...



- Čo keď pretína v bode?
  - Môže vyjsť von a môže zostať dnu...



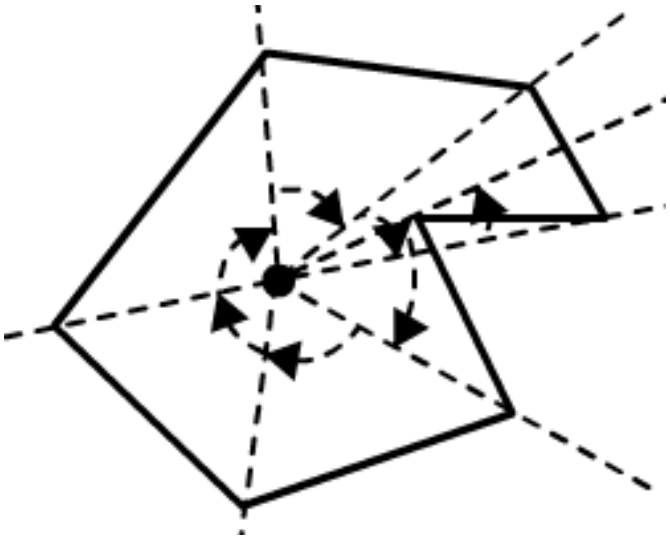
Čo keď je „nekonečne veľa“ priesečníkov?

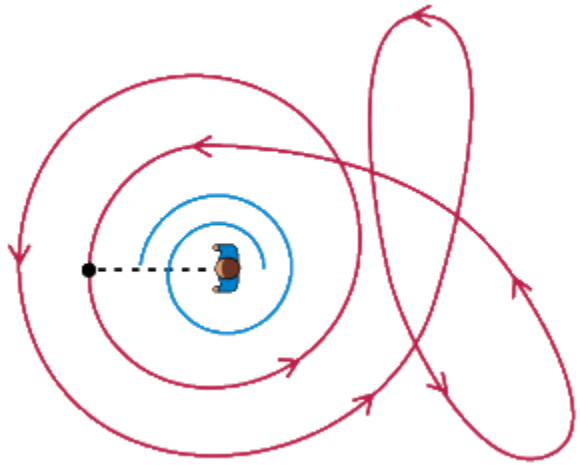


- Veľa špeciálnych prípadov ...

Bod v mnohouholníku – otáčanie

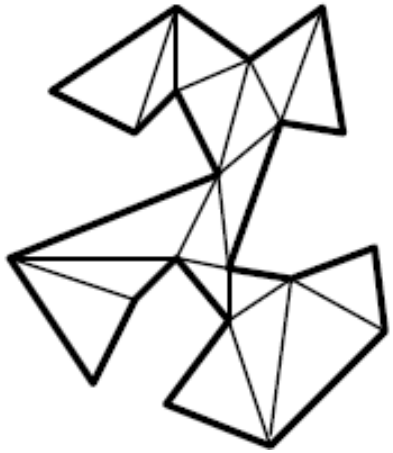
- Pozorovateľ z bodu P sa otáča po obvode mnohouholníka, a ak sa otočí celú otočku tak je vo vnútri, inak sa otočí o nič = vráti sa naspäť
- Počet otočení = ovíjacie číslo (winding number)





Bod v mnohouholníku – trojuholník

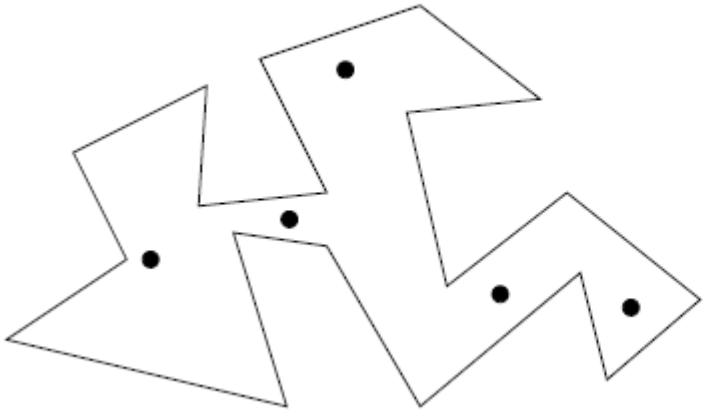
- Mnohouholník si rozdelíme na trojuholníky
- Zistíme, či sa bod nachádza v niektorom z trojuholníkov

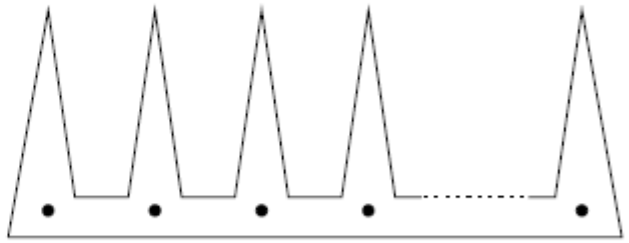


- Ako nájdeme trianguláciu?

Strážci v múzeu

- Múzeum je mnohouholník, treba umiestniť (stojacich) strážcov tak, aby sledovali každú stenu (stranu mnohouholníka) na ktorej sú exponáty.

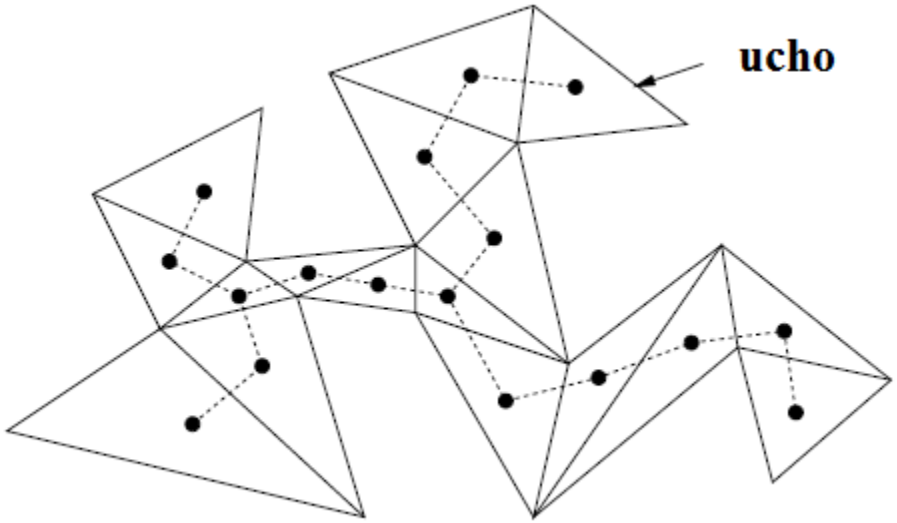




- Pre mnohouholník s N vrcholmi je potrebných N/3 strážcov.

Triangulácia

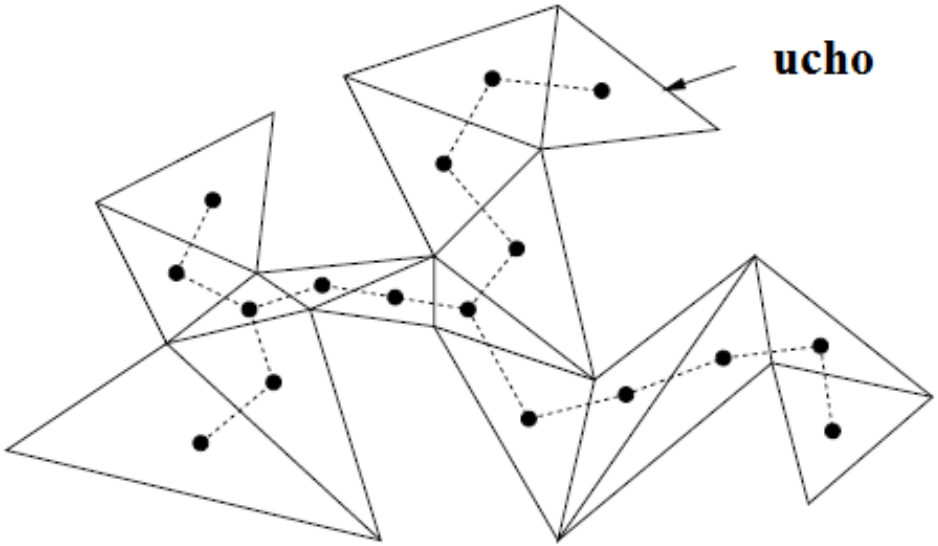
- Každý mnohouholník s N vrcholmi má trianguláciu s N-3 diagonálami a N-2 trojuholníkmi



- Spojenie stredov susedných trojuholníkov vytvára strom.

Ako nájdem trianguláciu?

- Algoritmus orezávania uší (Ear clipping)
- Vždy existuje aspoň jednu ucho .... prečo?



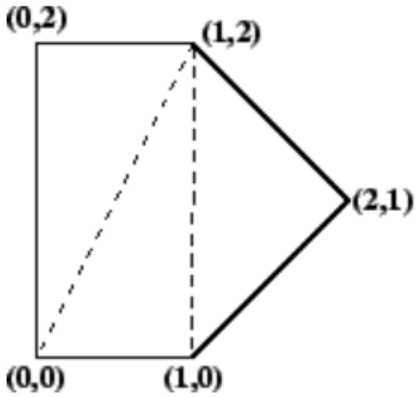
- Ako rýchlo nájdem nejaké ucho?

Algoritmus orezávania uší – zložitosť

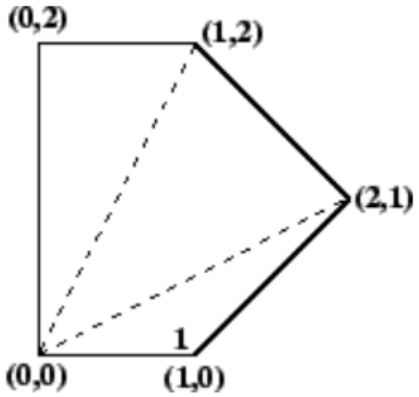
- Potrebujem nájsť N uší
- Každé nájdem v čase $O(N^2)$
  - Celkovo $O(N^3)$
- Vylepšenie?
  - Keď už odrežem jedno ucho, ktoré ďalšie uchá mohli vzniknúť?
  - Budem upravovať stav „uchovosti“ … :)
  - Celkovo $O(N^2)$

Najkratšia triangulácia konvexného mnohouholníku

- Najkratšia triangulácia konvexného mnohouholníku: Dané sú vrcholy konvexného N-uholníku po obvode, triangulácia je množina N-3 nepretínajúcich sa diagonál. Nájdite takú triangulláciu, pre ktorú súčet dĺžok diagonál je najmenší možný.



$8 + 2\sqrt{2} + 2\sqrt{5}$



$4 + 2\sqrt{2} + 4\sqrt{5}$

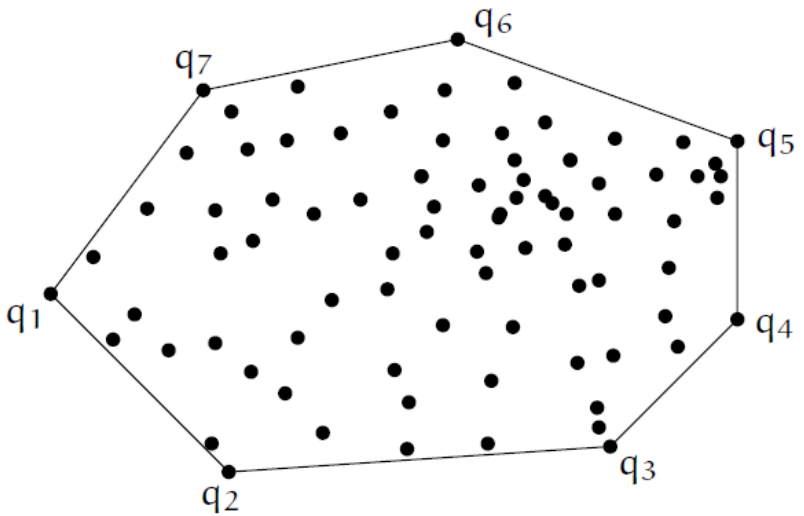
Najkratšia triangulácia konvexného N-uholníka

- Označme vrcholy po obvode 1, 2, ..., N
  Budeme počítať minimálny súčet obvodov trojuholníkov triangulácie – zodpovedá to minimálnemu súčtu dĺžok diagonál (prečo?)
- **Intervalové DP – definovať podproblémy**:
  - Označme D_{ij} minimálnu dĺžku (súčtu obvodov trojuholníkov v triangulácii) pre mnohouholník z vrcholov i, i+1, …, j
- **Rekurentný vzťah medzi podproblémami**:
  - Ak je k < i+2, mnohouholník i..k je triviálny: $D_{ik} = 0$
  - Inak, možeme vybrať (vyskúšame všetky možnosti) také j, že i < j < k a započítame obvod trojuholníka i,j,k:

$$D[i][k] = \begin{cases} 0 & k < i+2 \\ \min_{i<j<k}\{D[i][j] + D[j][k] + w(i,j,k)\} & \text{inak} \end{cases}$$

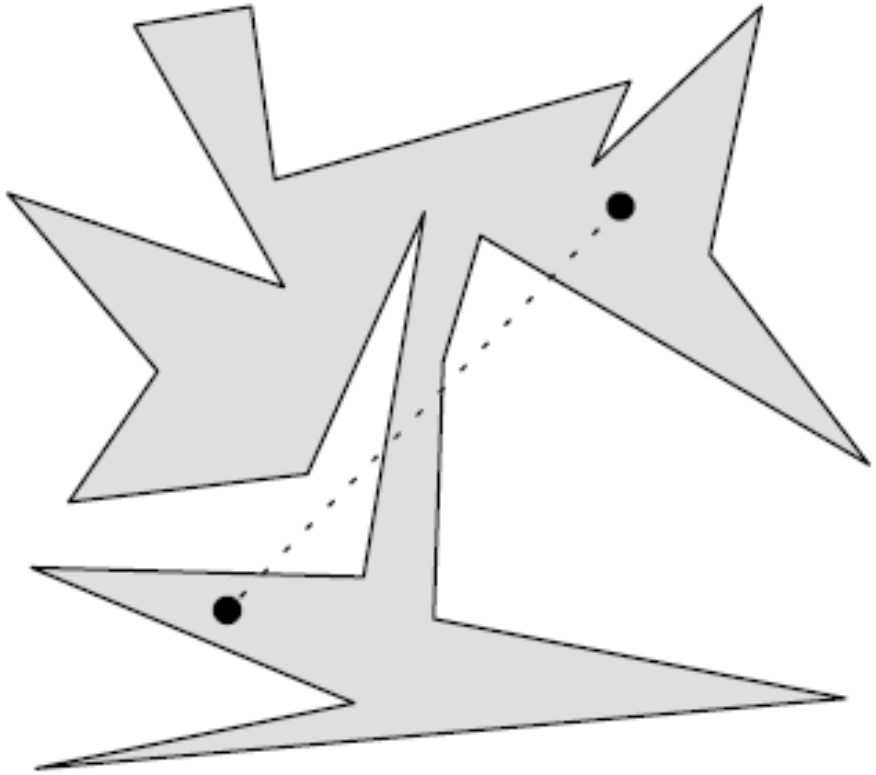
Konvexný obal

- Daných máme N bodov v rovine
- Zaujíma nás nejaká hranica týchto bodov
- Formálne: Najmenší konvexný mnohouholník, ktorý ich všetky obsahuje



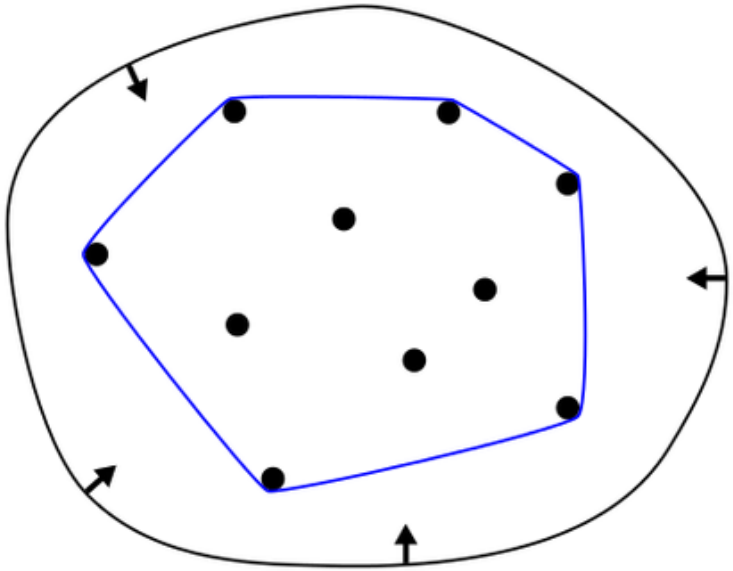
Nekonvexný mnohouholník

- Taký v ktorom sa môžeme hrať „skrývačku“



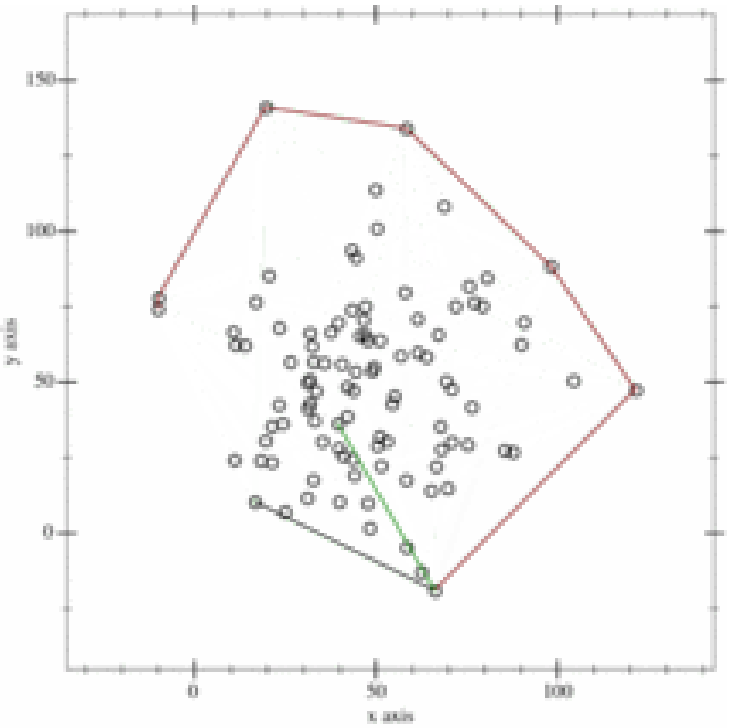
Konvexný obal (2)

- Ako ho nájdem?
- Človek: elastickou gumičkou...



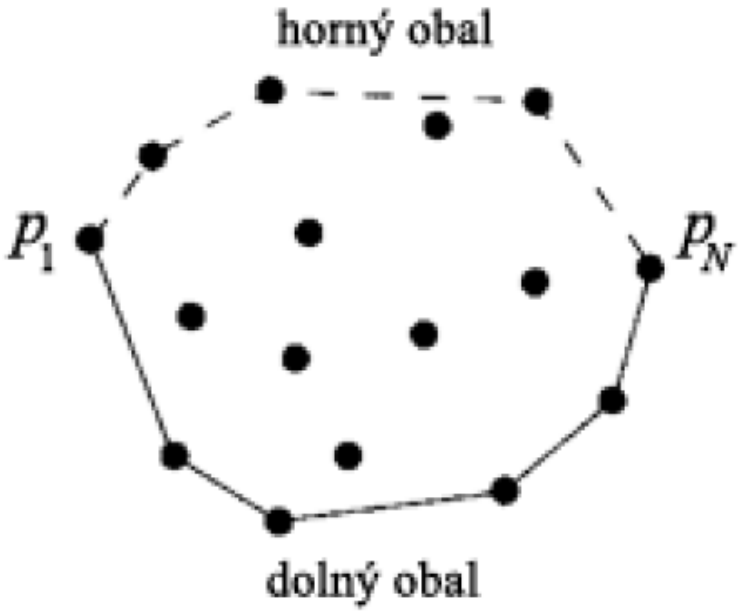
Metóda obaľovania balíčka

- Začnem v extrémnom bode, ktorý je určite na konvexnom obale:
  - najmenšia x-ová súradnica (v prípade rovnosti menšia y-ová)
- Postupne pridávam po jednom bode na konvexný obal
- Ktorý bod bude nasledujúci?
  - Taký, ktorý má najväčší uhol spomedzi zostávajúcich bodov



Metóda horného a dolného obalu

- Body usporiadam lexikograficky (podľa x-ovej súradnice, v prípade rovnosti podľa y-ovej)
- Určím konvexný obal zdola a zhora a spojím



Výpočet dolného obalu

- Postupne spracúvame body zo vstupnej množiny
- Po každom spracovanom bode si udržiavame čiastočný konvexný obal doposiaľ spracovanej časti
- Kľúčový krok je **pozmenenie doterajšieho konvexného obalu po pridaní bodu p_i**
- Inými slovami:
  Pre určený dolný konvexný obal z bodov $p_1, \ldots p_{i-1}$, chceme určiť konvexný obal rozšírenej množiny bodov $p_1, \ldots, p_i$

Výpočet dolného obalu – Implementácia

- Predpokladáme usporiadané body:

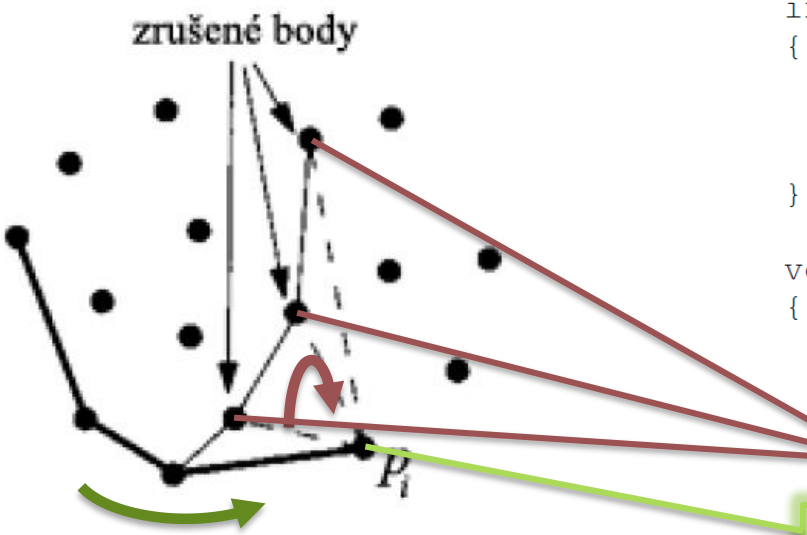
```
point p[maxn];
point o[2*maxn];  /* obal */
int np, no;

void hull(void)
{
  int i;
  /* začíname */
  o[0]=p[0]; o[1]=p[1];
  no=2;
  /* dolný obal */
  for(i=2;i<np;i++)
    insert(i);
  /* horný obal */
  for(i=np-2;i>=0;i--)
    insert(i);
  /* obal sú body o[0..no-1] */
}
```

Ako pridáme bod do konvexného obalu

- Pri obchádzaní obvodu mnohouholníka sa v každom vrchole točíme: alebo doľava, alebo doprava (alebo ideme rovno).
- **V konvexnom mnohouholníku (obale) by sme sa mali točiť vždy jedným smerom.**
- Ako spracujeme bod p_i ?
  - Bod p_i pridáme do čiastočného konvexného obalu (bod na hranici obalu, označme ho o_k)
  - Opakovane: preskúmame poslednú trojicu bodov, ak sa pri prechode $o_{k-2} \rightarrow o_{k-1} \rightarrow o_k$ točíme doprava, tak bod o_{k-1} je vnútorný a odstránime ho.

Ako pridáme bod do konvexného obalu

- Bod p_i pridáme do čiastočného konvexného obalu (bod na hranici obalu, označme ho o_k)
- Opakovane: preskúmame poslednú trojicu bodov, ak sa pri prechode $o_{k-2} \rightarrow o_{k-1} \rightarrow o_k$ točíme doprava, tak bod o_{k-1} je vnútorný (mimo konv. obalu) a **odstránime ho**.



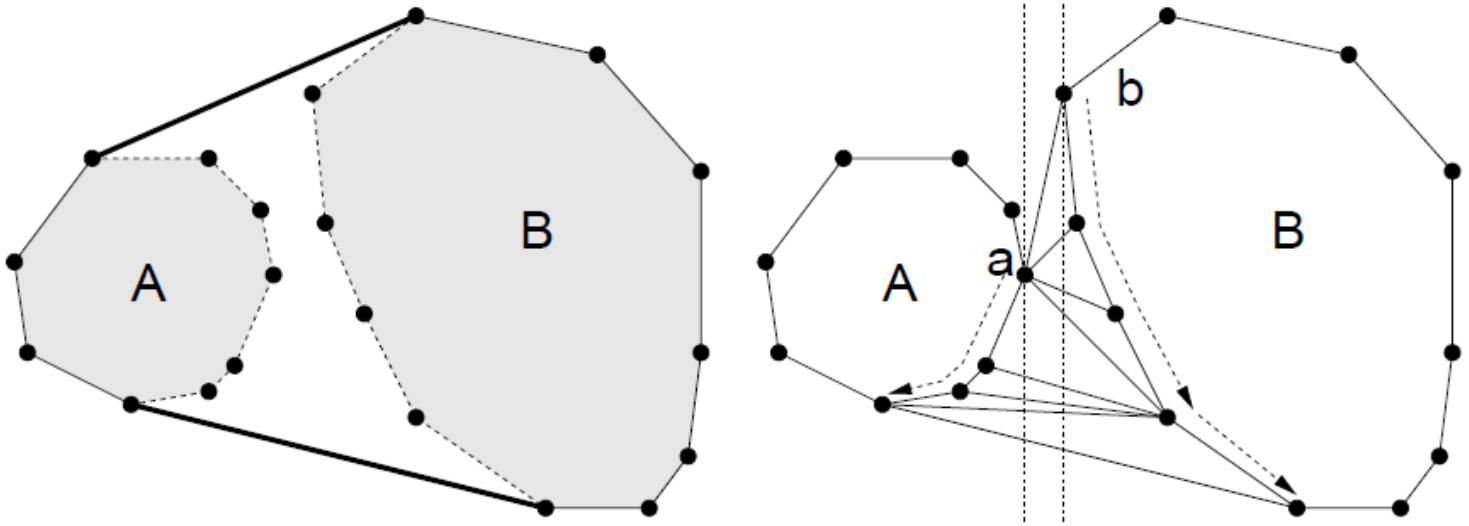

```
int ccw(point a, point b, point c)
{
  /* <0 doprava, =0 rovno, >0 doľava */
  return(c.x-a.x)*(c.y-b.y) -
         (c.x-b.x)*(c.y-a.y);
} /* vekotorový súčin */

void insert(int i)
{
  /* pokým sa točíme doprava */
  while (no>=2 &&
    ccw(o[no-2],o[no-1],p[i]) < 0)
    no--;
  /* nový bod */
  o[no++]=p[i];
}
```

Zložitosť výpočtu konvexného obalu

- **Metóda obaľovania balíčku** (Gift wrapping)
  - Pridávam N bodov, pre každý musím prezrieť zostávajúcich (najviac N), čiže **celkovo vykonám $O(N^2)$ operácií**
- **Metóda dolného a horného obalu**
  - Body usporiadam – O(N log N)
  - Pridávam N bodov do dolného obalu: pre každý možno niekoľkokrát odstránim "predposledný"
  - Najviac odstránim N bodov (keďže som naviac N pridal)
  - Vykonám teda najviac O(N) operácií
  - Horný obal analogicky
  - **Celkovo vykonám O(N log N) operácií**

Konvexný obal – ďalšie metódy

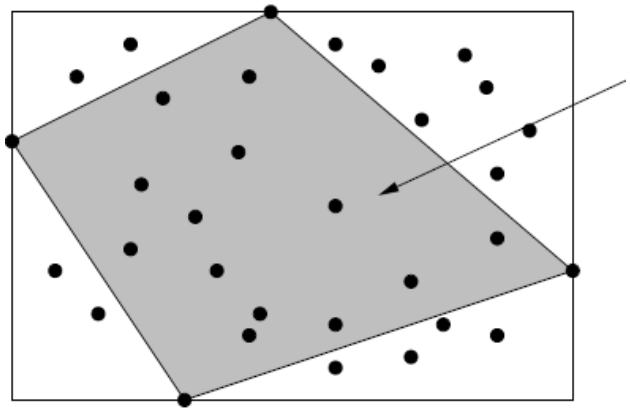
- Rozdeľuj a panuj (divide and conquer):
  - Rozdelíme množinu N bodov na polovicu (podľa x-súradnice),
  - určíme konvexné obaly partícií, ktoré následne
  - spojíme do veľkého konvexného obalu



- Pre efektívny výpočet O(N log N) je kľúčové spojenie dvoch menších obalov v čase O(N) – hľadanie dotyčníc

Konvexný obal – QuickHull

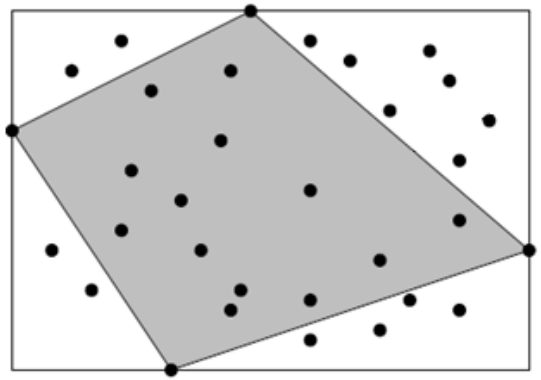
- Analógia Quicksortu
- Za predpokladu rovnomerne rozdelených bodov v ohraničujúcom obdĺžniku, očakávaný počet bodov na (hranici) konvexnom obale je O(log N)
- Čo najrýchlejšie odstrániť body vo vnútri obalu.
  - Nájsť najmenšiu a najväčšiu x-ovú súradnicu
  - Nájsť najmenšiu a najväčšiu y-ovú súradnicu



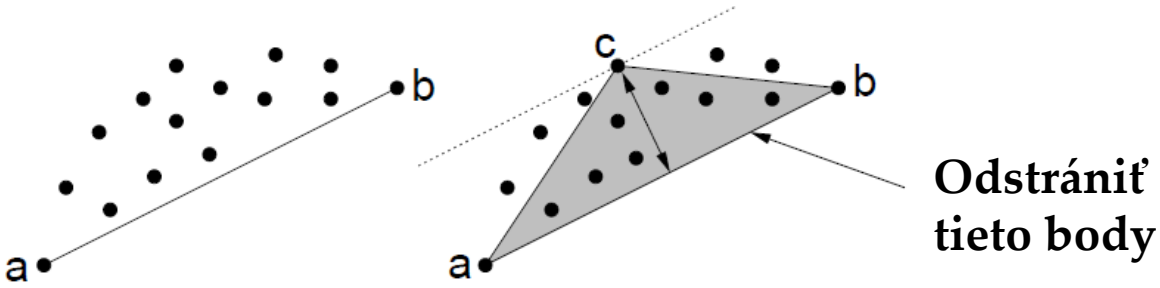
Odstrániť body vo vnútri štvoruholníka z týchto bodov

Konvexný obal – QuickHull (2)

- Rekurzívne odstráň body z okrajových trojuholníkov a dokončí konvexný obal



- Rekurzívny krok:

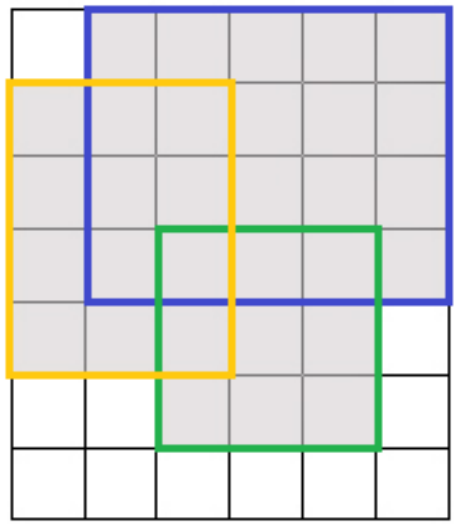


Pridaj bod c do konvexného obalu

- Pre vhodné vstupy O(N log N), pre nevhodné $O(N^2)$

Zjednotenie obdĺžnikov

- Pre daných N obdĺžnikov so stranami rovnobežnými so súradnicovými osami, nájdite obsah zjednotenia.
- Napr.

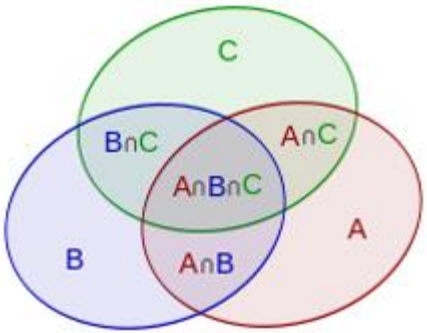


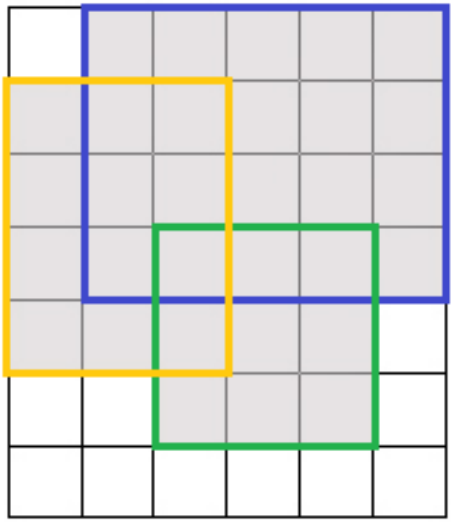
Obsah (prekrytej oblasti)?

31 políčok

Princíp zapojenia a vypojenia

- Do plochy:
  - Pripočítame plochu každého obdĺžnika
  - Odpočítame plochu prieniku každej dvojice
  - Pripočítame plochu prieniku každej trojice





Obsah modrého: 4x5=**20**
Obsah žltého: 4x3=**12**
Obsah zeleného: 3x3=**9**
Prienik modrého a žltého: 6
Prienik modrého a zeleného: 3
Prienik žltého a zeleného: 2
Prienik všetkých: **1**

- **Plocha zjednotenia: (20+12+9) – (6+3+2) + 1 = 31**

Zovšeobecnenie pre 4 obdĺžniky

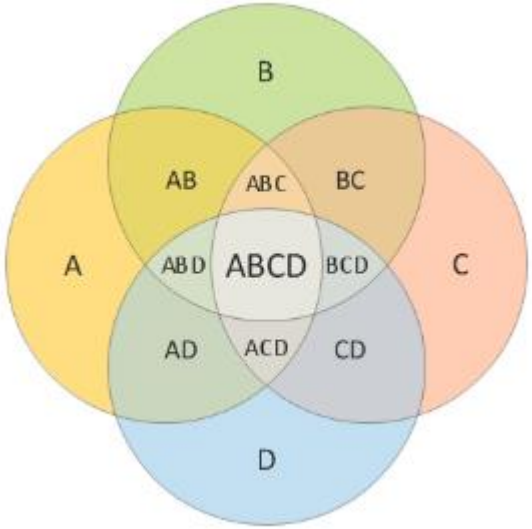
- Do plochy:
  - **Pripočítame** plochu každého obdĺžnika
  - **Odpočítame** plochu prieniku **každej dvojice**
  - **Pripočítame** plochu prieniku **každej trojice**
  - **Odpočítame** plochu prieniku **každej štvorice**

+4 (A, B, C, D)

–6 (AB,AC,AD, BC, BD, CD)

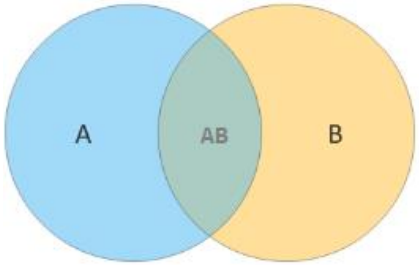
+4 (ABC, ABD, ACD, BCD)

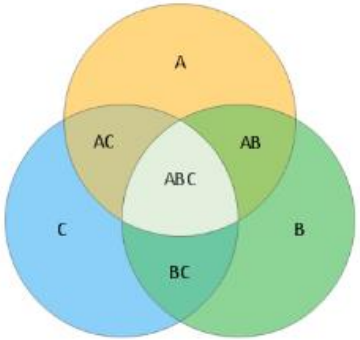
–1 (ABCD)

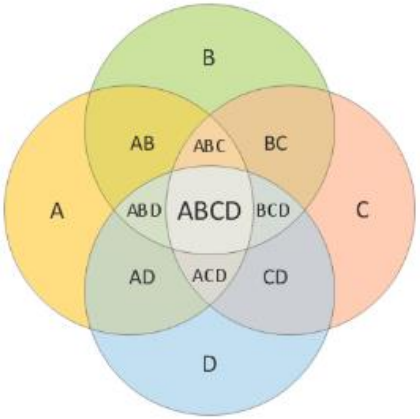


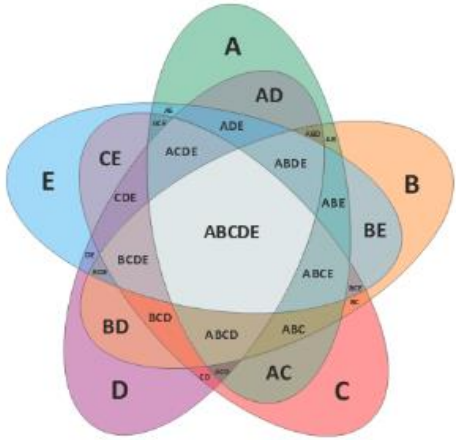
Zovšeobecnenie pre N obdĺžnikov

- Do plochy:
  - **Pripočítame** plochu **každej 2k+1-tice**: 1, 3, 5, …
  - **Odpočítame** plochu **každej 2k+2-tice**: 2, 4, 6, …









- Koľko je rôznych k-tic? $\binom{n}{k}$
- Zložitosť výpočtu? **exponenciálna**

Množinový zápis

- N(A) – počet prvkov množiny A

$$N(A_1 \cup A_2 \cup \cdots \cup A_n)$$
$$= \sum_{1 \le i \le n} N(A_i) - \sum_{1 \le i < j \le n} N(A_i \cap A_j)$$
$$+ \sum_{1 \le i < j < k \le n} N(A_i \cap A_j \cap A_k)$$
$$- \cdots + (-1)^{n+1} N(A_1 \cap A_2 \cap \cdots \cap A_n).$$

$$N(A_1 \cup A_2) = N(A_1) + N(A_2) - N(A_1 \cap A_2)$$

$$\begin{aligned} N(A_1 \cup A_2 \cup A_3) &= N(A_1) + N(A_2) + N(A_3) - N(A_1 \cap A_2) \\ &\quad - N(A_1 \cap A_3) - N(A_2 \cap A_3) + N(A_1 \cap A_2 \cap A_3) \\ &= \sum_{1 \le a \le 3} N(A_a) - \sum_{1 \le a < b \le 3} N(A_a \cap A_b) + N(A_1 \cap A_2 \cap A_3) \end{aligned}$$

$$\begin{aligned} N(A_1 \cup A_2 \cup A_3 \cup A_4) &= \sum_{1 \le a \le 4} N(A_a) - \sum_{1 \le a < b \le 4} N(A_a \cap A_b) \\ &\quad + \sum_{1 \le a < b < c \le 4} N(A_a \cap A_b \cap A_c) - N(A_1 \cap A_2 \cap A_3 \cap A_4) \end{aligned}$$

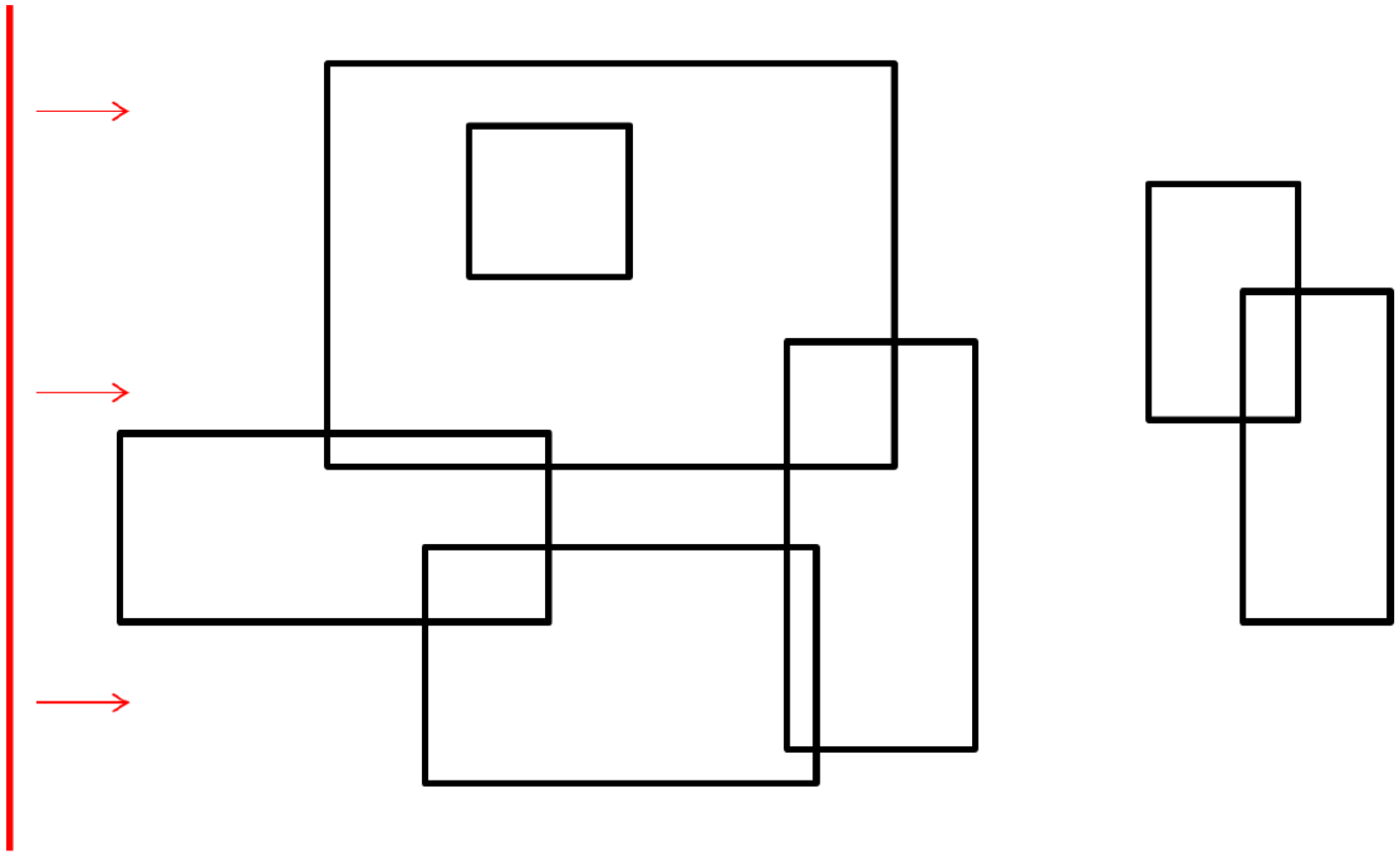
Ako navrhnúť efektívnejší algoritmus?

- Nemôžeme sa problém snažiť vyriešiť „hneď všade“ ako sa nato pozeral princíp zapojenie a vypojenia
- Pokúsime sa problém riešiť postupne a nedostatky vyriešiť lokálne: **zametanie (sweep line)**
- Podobne ako upratovanie neporiadku doma:

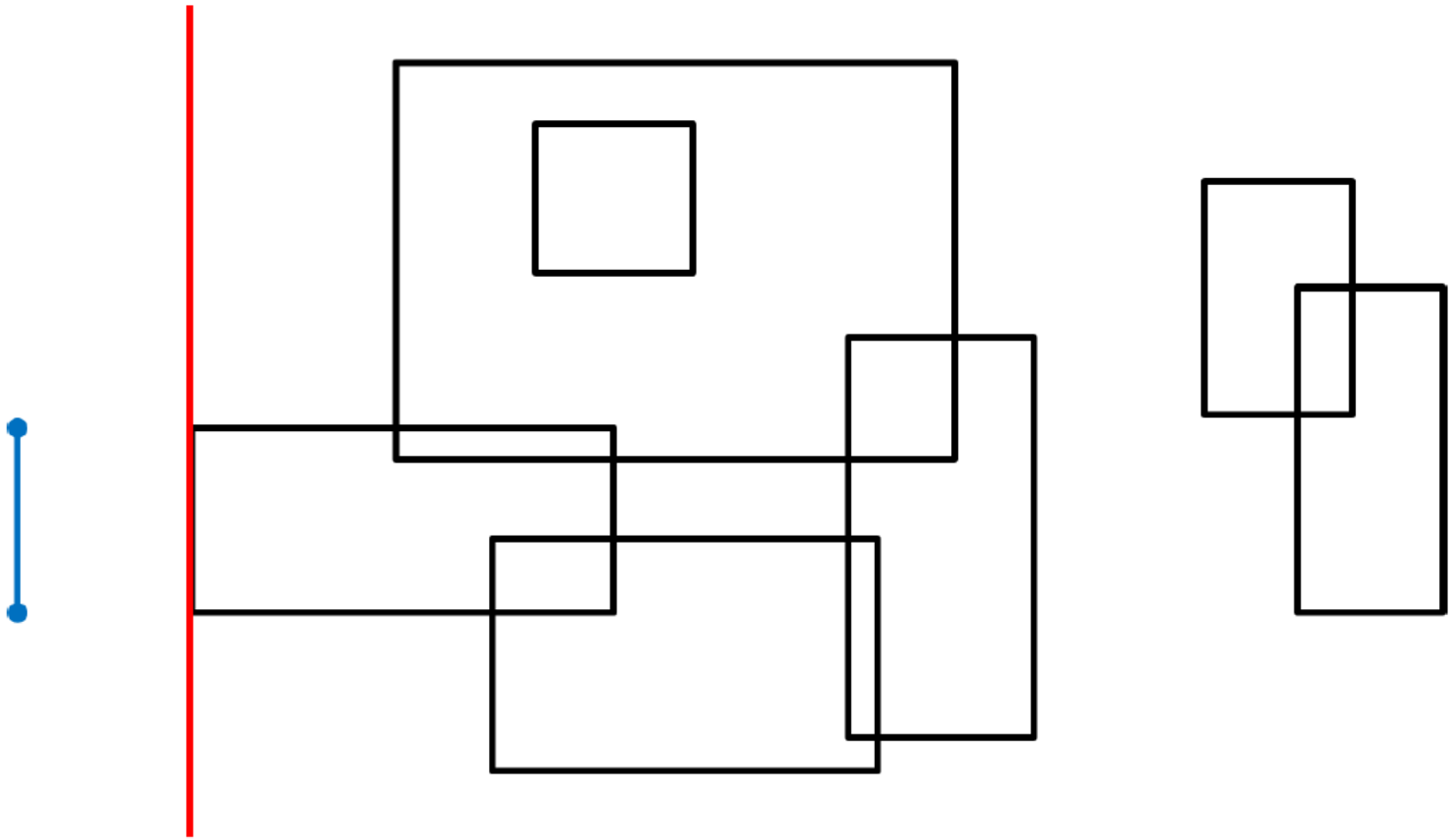
Zametanie (Plocha zjednotenia obdĺžnikov)

- Hlavná myšlienka:
  posúvame zametaciu čiaru (sweep line) z jednej strany na druhú cez celú rovinu a pri posune vyriešime problém lokálne
- Aktuálny stav problému si udržiavame vo vhodnej dátovej štruktúre
- Nesimulujeme spojitý proces posúvania, ale definujeme udalosti, ktoré (pri spojitom posúvaní) spôsobujú zmeny v použitej dátovej štruktúre
- Udržiavame si množinu obdĺžnikov pretínajúcich zametaciu čiaru
- Udalosti: ľavé a pravé strany obdĺžnikov
- Stačí si pamätať y-ové súradnice obdĺžnikov (intervaly)

Zametanie (Plocha zjednotenia obdĺžnikov)



Zametanie – udalosť (pridanie)

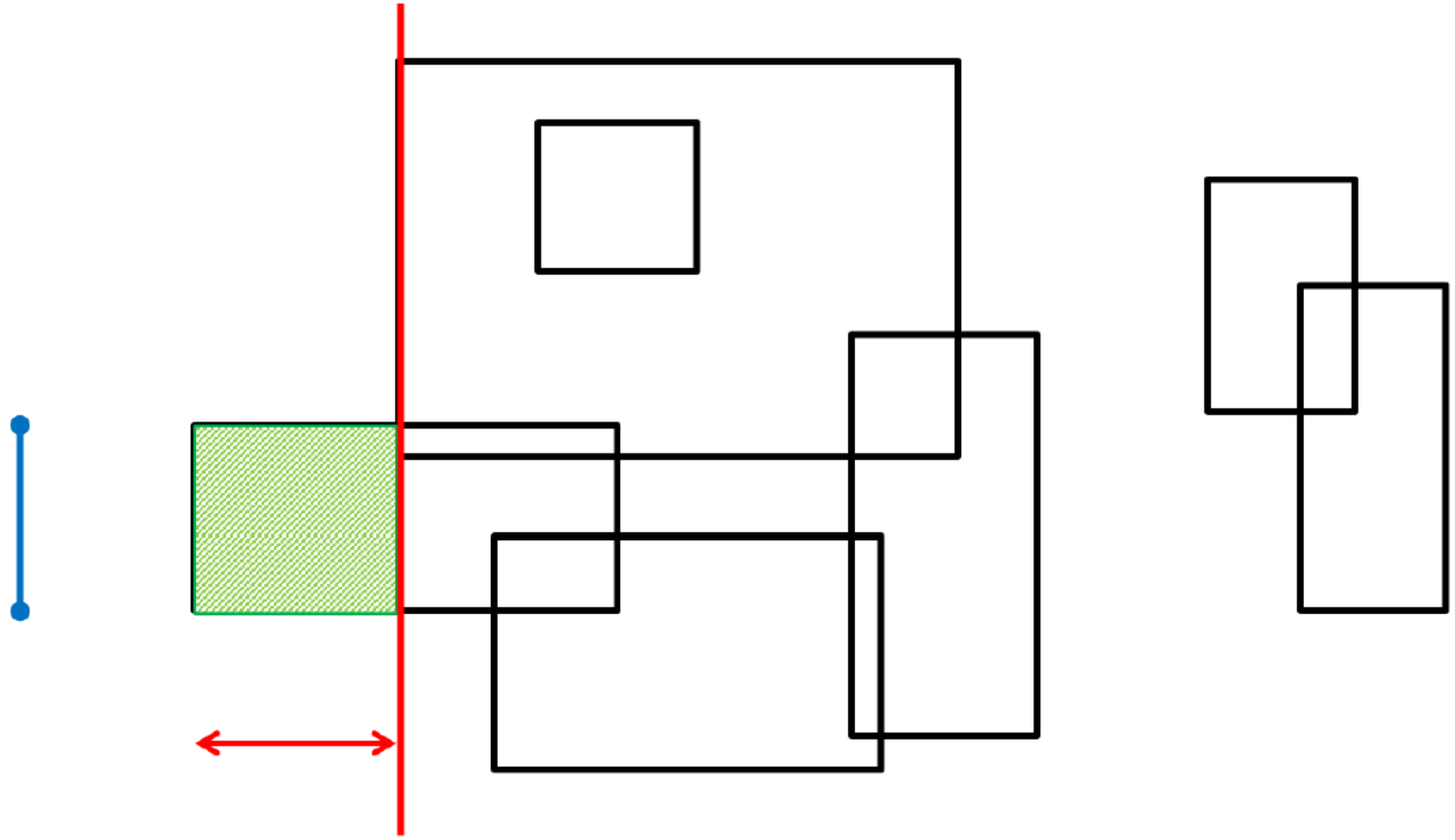


ľavá strana obdĺžnika

=

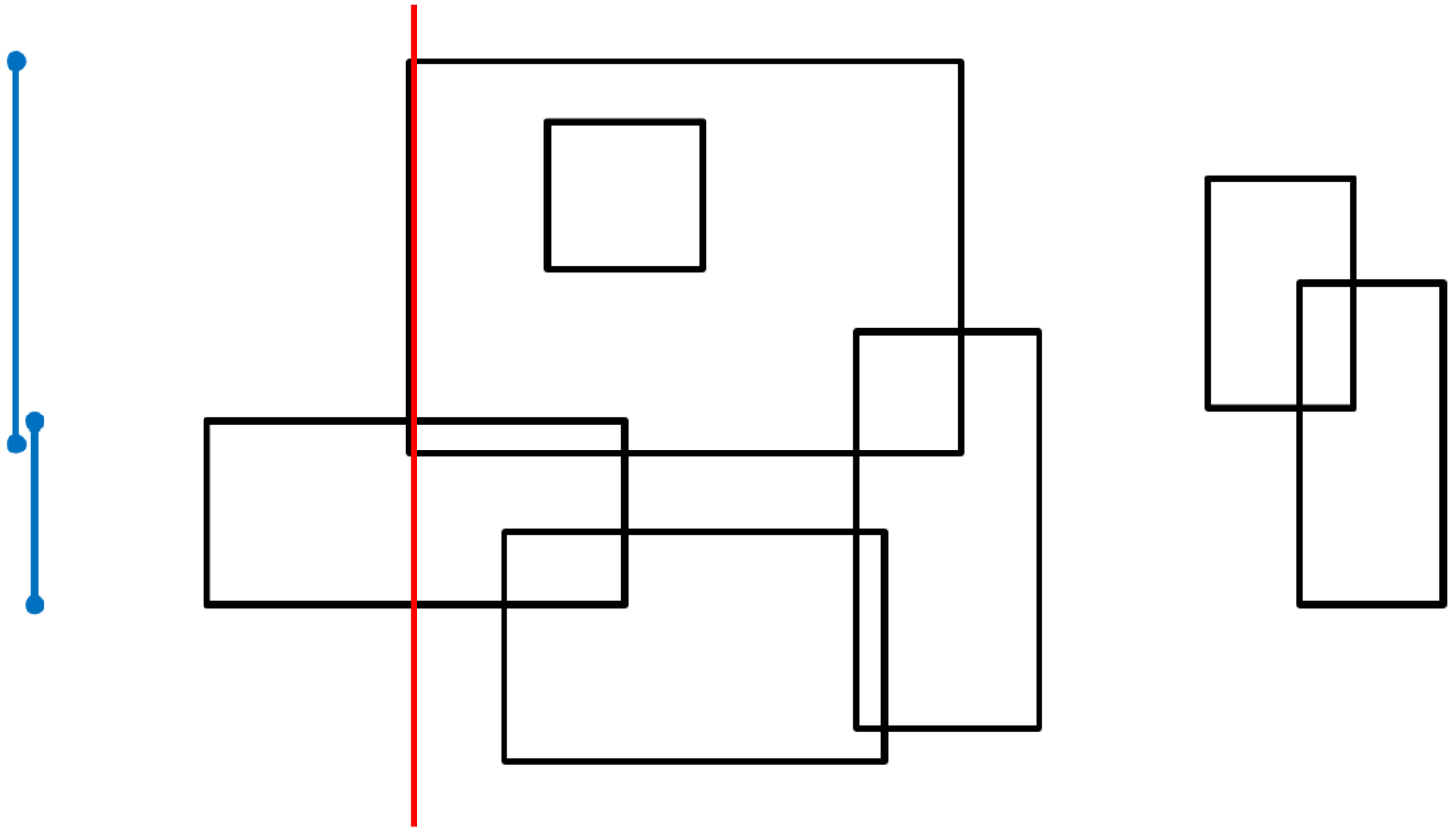
pridanie modrého intervalu

Zametanie – posun



Pri posune pripočítame plochu zjednotenia obdĺžnikov
zelená plocha = (dĺžka modrých intervalov) x (dĺžka posunu)

Zametanie – udalosť (pridanie)

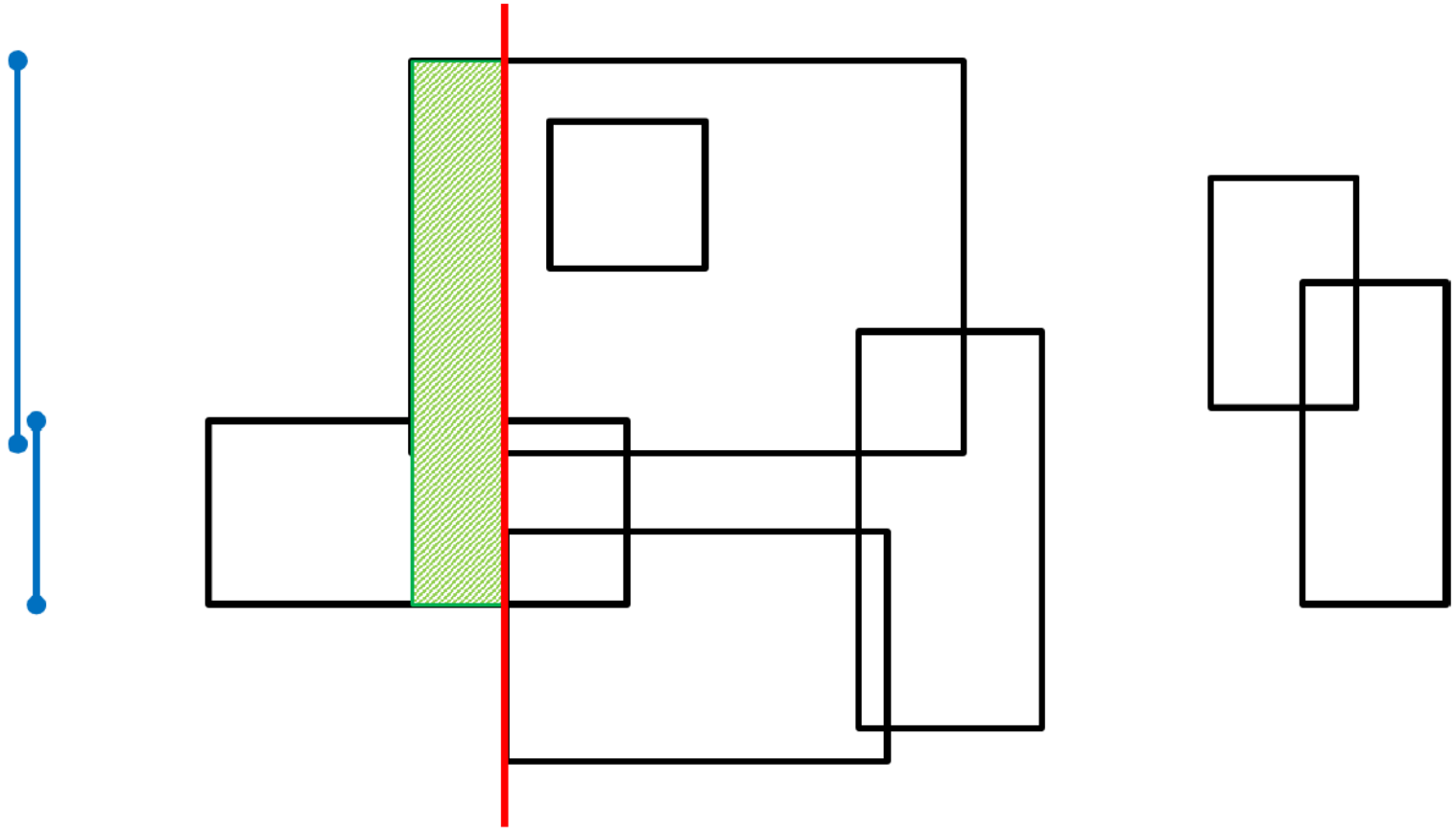


ľavá strana obdĺžnika

=

pridanie modrého intervalu

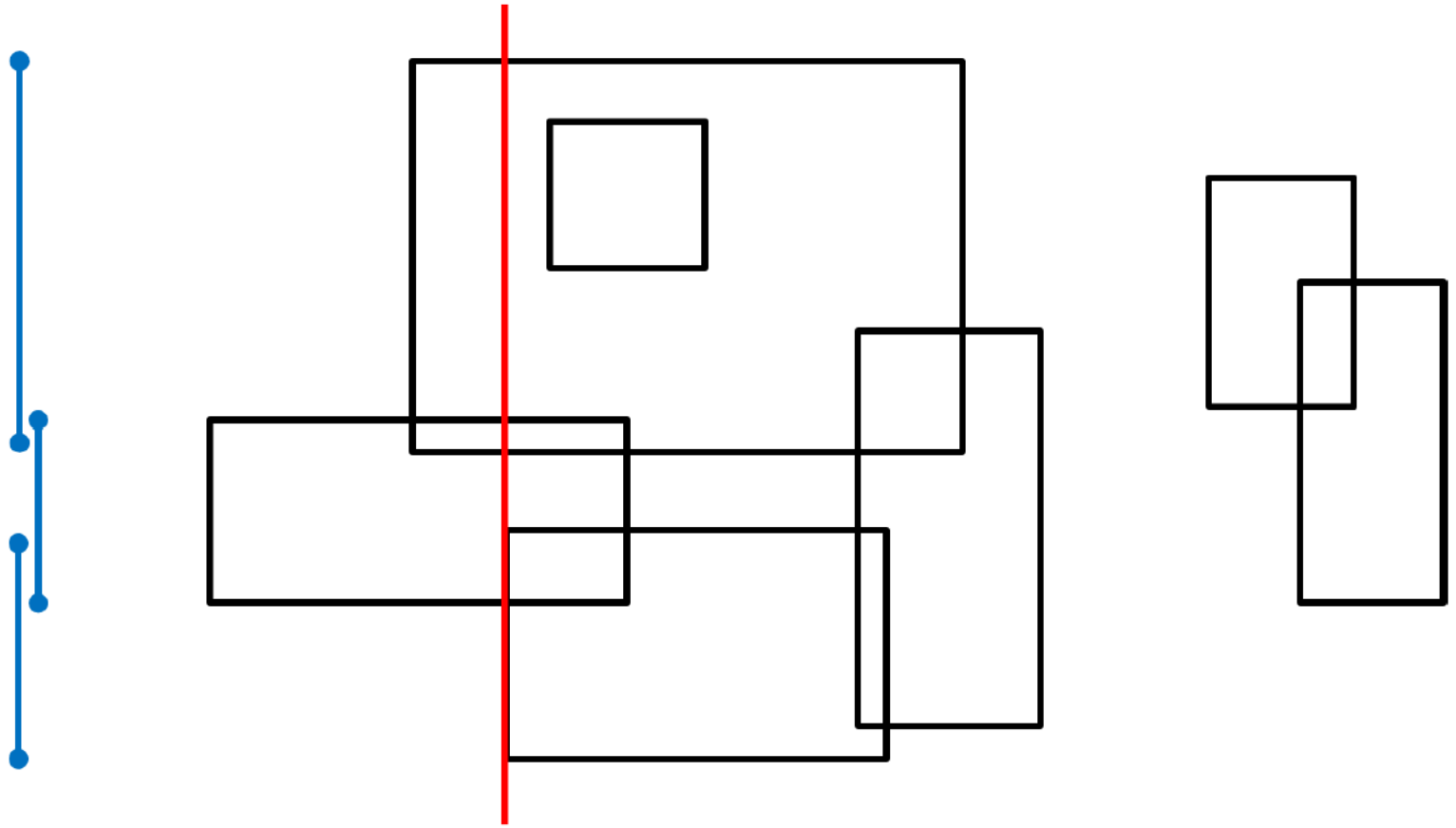
Zametanie – posun



Pri posune pripočítame plochu zjednotenia obdĺžnikov

zelená plocha = (dĺžka modrých intervalov) x (dĺžka posunu)

Zametanie – udalosť (pridanie)

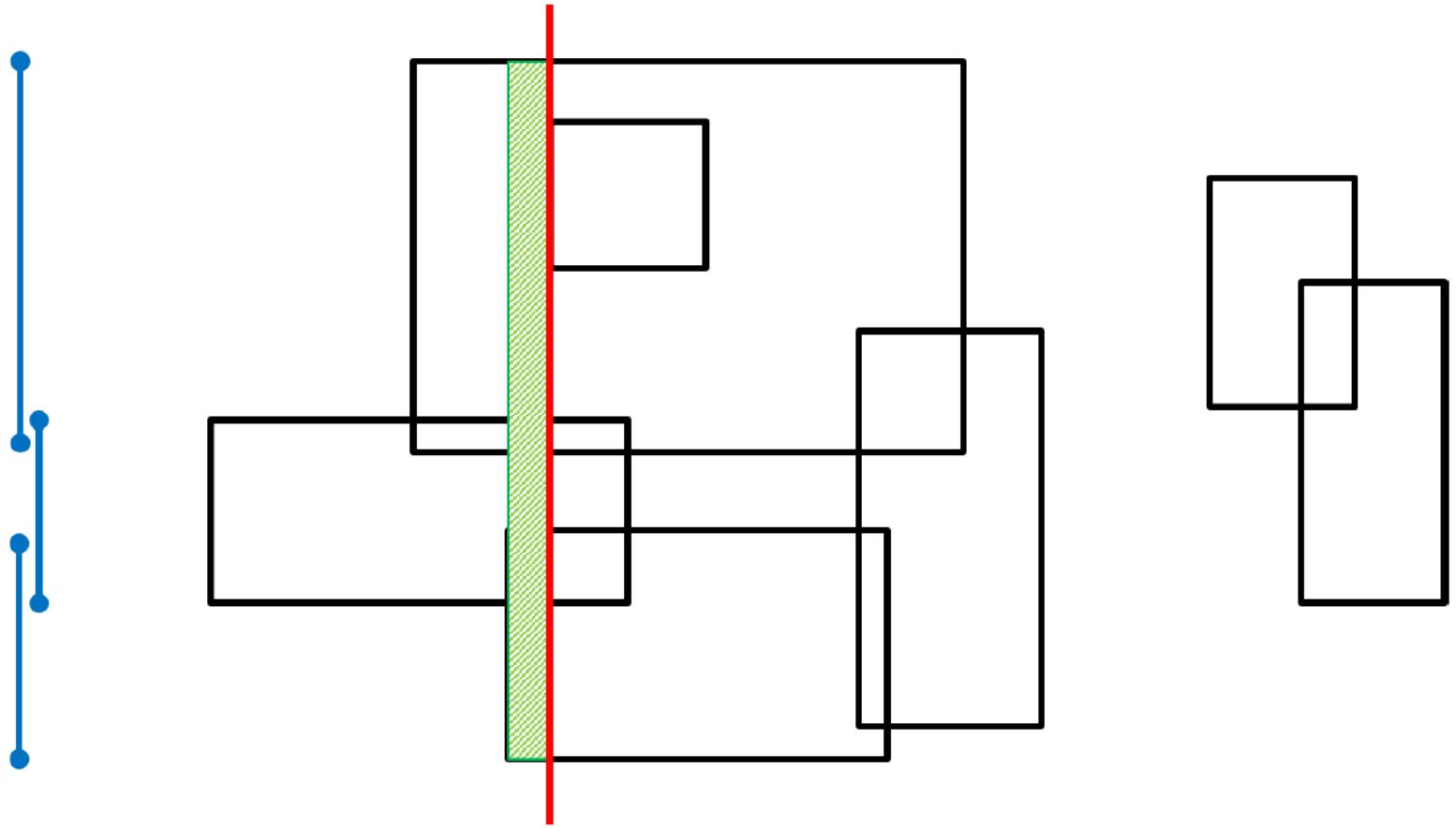


ľavá strana obdĺžnika

=

pridanie modrého intervalu

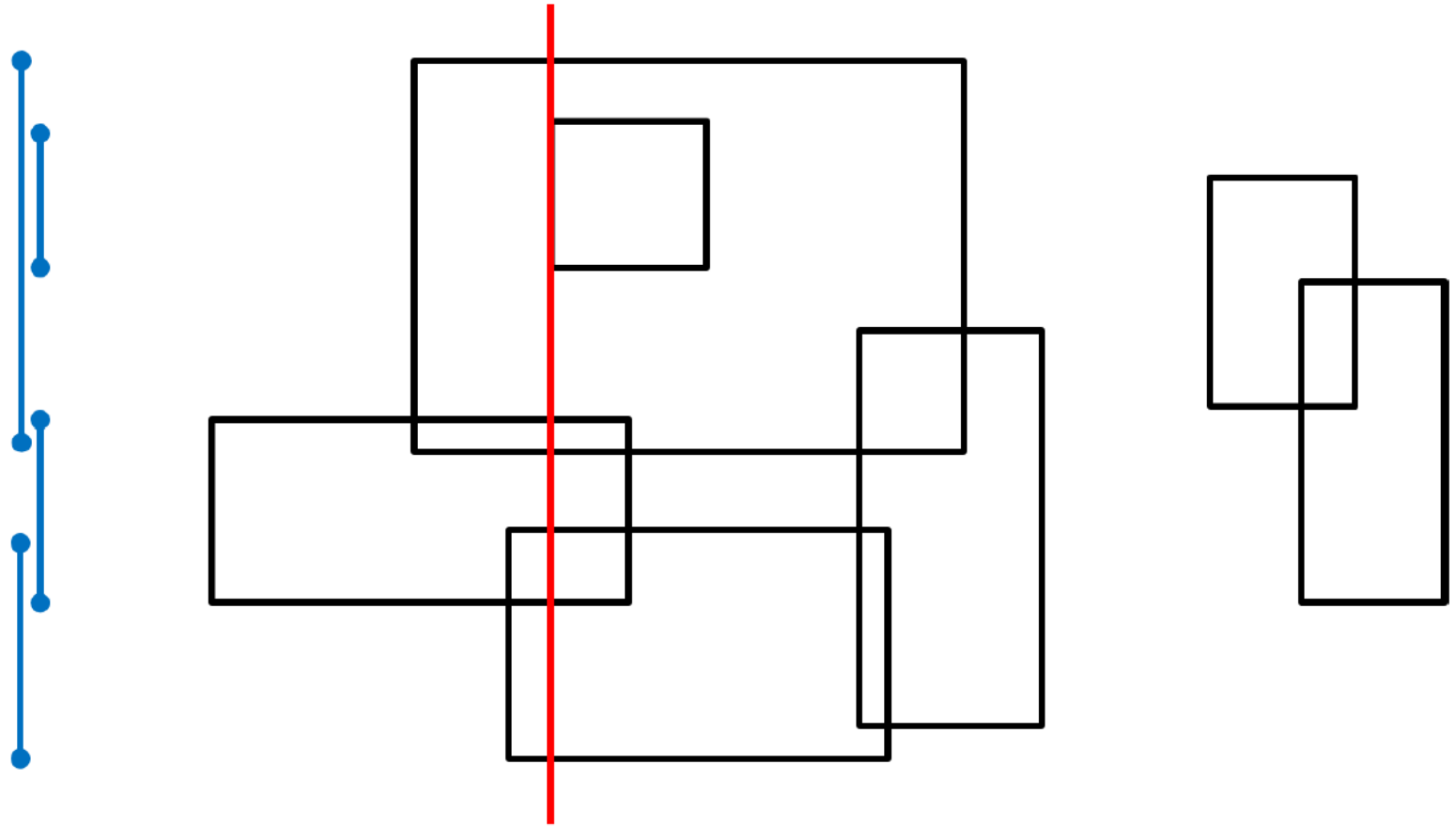
Zametanie – posun



Pri posune pripočítame plochu zjednotenia obdĺžnikov

zelená plocha = (dĺžka modrých intervalov) x (dĺžka posunu)

Zametanie – udalosť (pridanie)

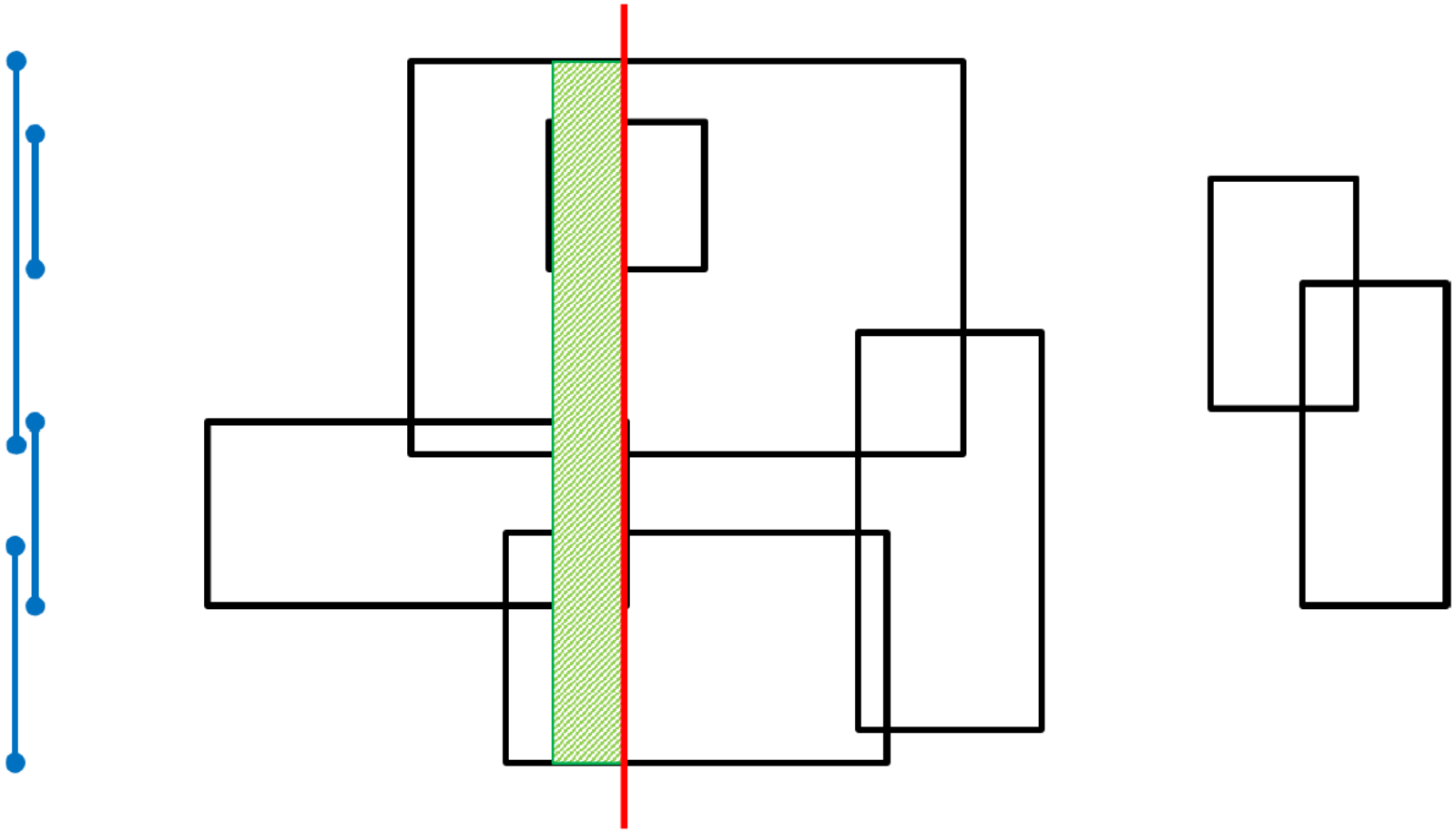


ľavá strana obdĺžnika

=

pridanie modrého intervalu

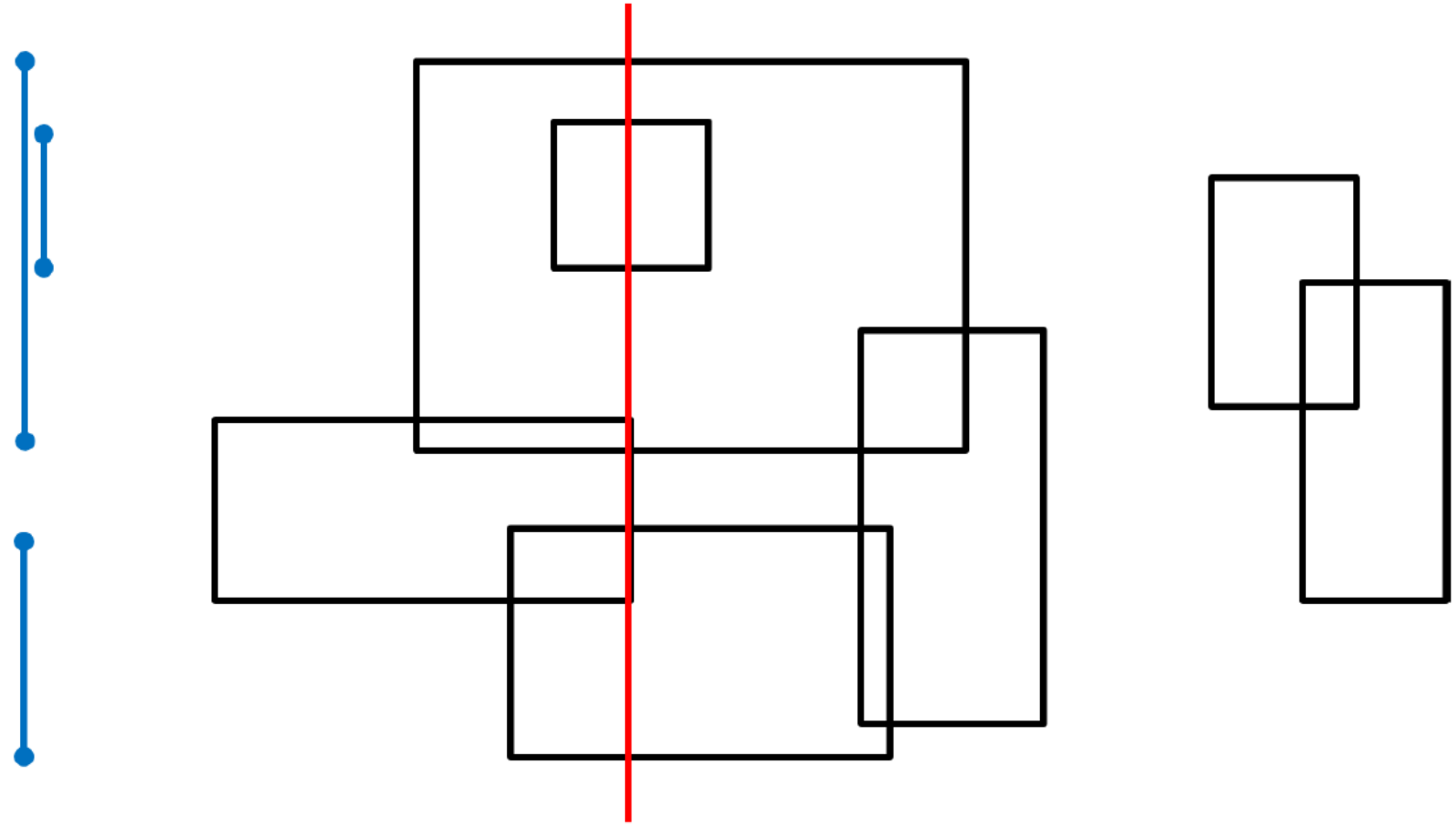
Zametanie – posun



Pri posune pripočítame plochu zjednotenia obdĺžnikov

zelená plocha = (dĺžka modrých intervalov) x (dĺžka posunu)

Zametanie – udalosť (odobratie)

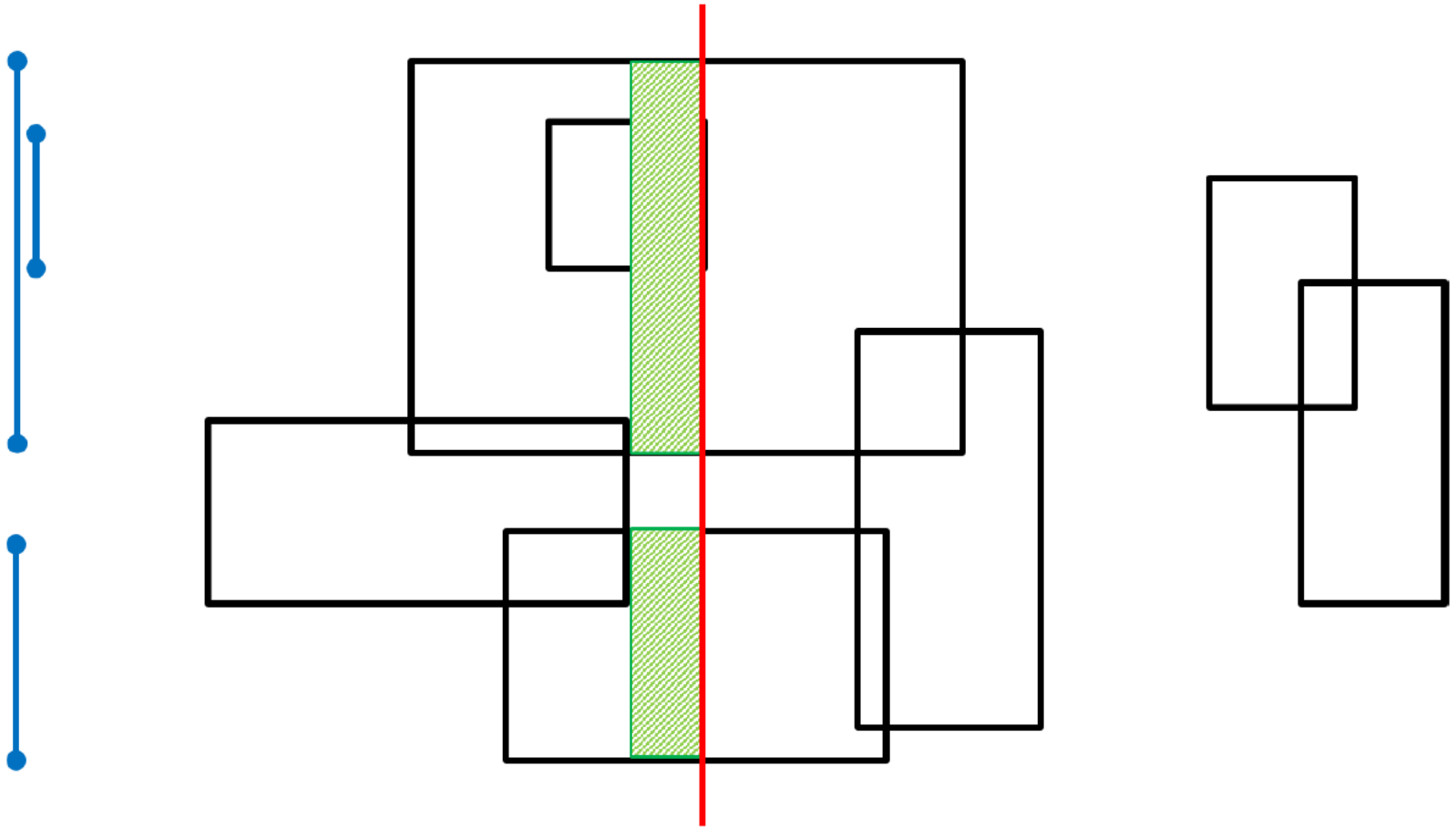


pravá strana obdĺžnika

=

odobratie modrého intervalu

Zametanie – posun



Pri posune pripočítame plochu zjednotenia obdĺžnikov

zelená plocha = (dĺžka modrých intervalov) x (dĺžka posunu)

Plocha zjednotenia obdĺžnikov zametaním

- Udalosti: ľavé a pravé strany obdĺžnikov
- Spracúvame od najmenšej x-ovej súradnice po najväčšiu:
  1. **Ak zametacia čiara narazí na ľavú stranu obdĺžnika**

     Vložiť interval y-ových súradníc obdĺžnika do vyhľadávacieho stromu
  2. **Ak zametacia čiara narazí na pravú stranu obdĺžnika**

     Odstrániť interval y-ových súradníc obdĺžnika z vyhľadávacieho stromu
  3. **Pri posune na ďalšiu udalosť:**

     Pripočítaj plochu obdĺžnikov pretínajúcich zametaciu čiaru:
     (dĺžka zjednotenia intervalov) x (dĺžka posunu)

- Dĺžka zjednotenia 1D intervalov
  - Vyriešime ako operáciu vo vyhľadávacom strome

Plocha zjednotenia obdĺžnikov – zložitosť

- Usporiadanie udalostí **O(N log N)**
- Spracovanie 2N udalostí (podľa x-ovej súradnice):
  1. **Ak zametacia čiara narazí na ľavú stranu obdĺžnika**

     O(log N) Vložiť interval y-ových súradníc obdĺžnika do vyhľadávacieho stromu
  2. **Ak zametacia čiara narazí na pravú stranu obdĺžnika**

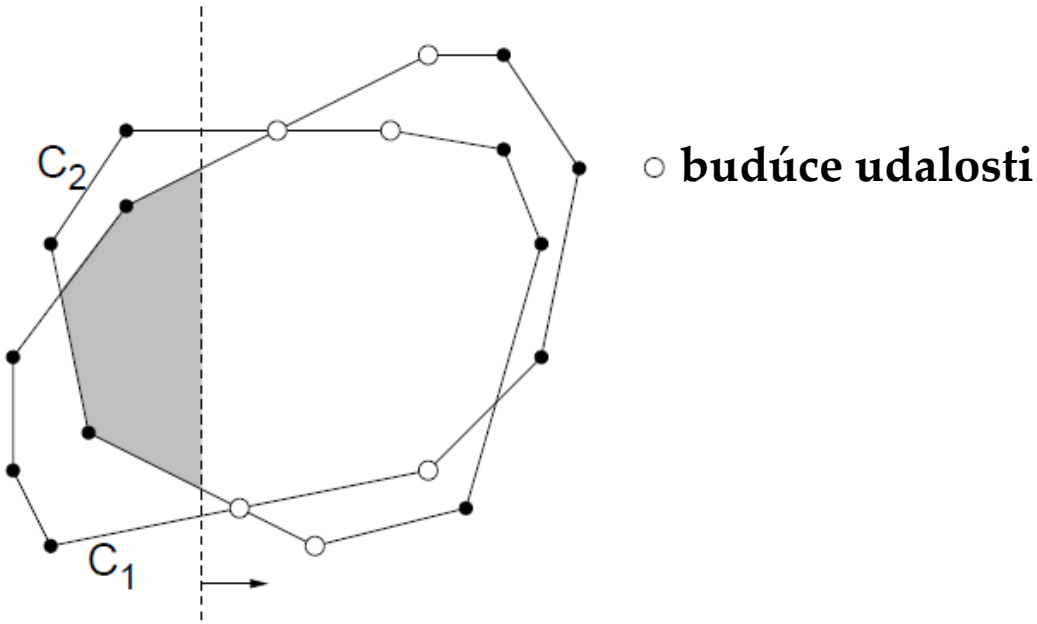
     O(log N) Odstrániť interval y-ových súradníc obdĺžnika z vyhľadávacieho stromu
  3. **Pri posune na ďalšiu udalosť:**

     O(log N) Pripočítaj plochu obdĺžnikov pretínajúcich zametaciu čiaru:
     (dĺžka zjednotenia intervalov) x (dĺžka posunu)
- Celkovo **O(N log N)**

Zametanie – ďalšie problémy

- Dĺžka obvodu zjednotenia obdĺžnikov
  - Analogicky

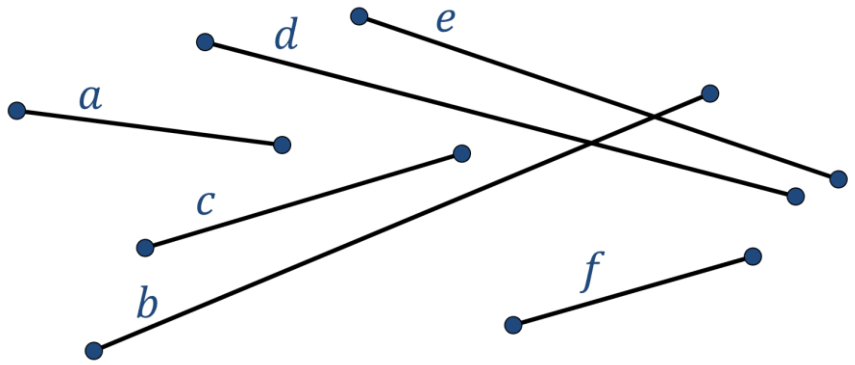
- Prienik dvoch konvexných mnohouholníkov



Zametanie – priesečníky úsečiek

- Pre daných N úsečiek:
  - zistiť či sa niektoré z nich pretínajú (Shamos–Hoey)
  - nájsť všetkých K priesečníkov (Bentley-Ottmann)

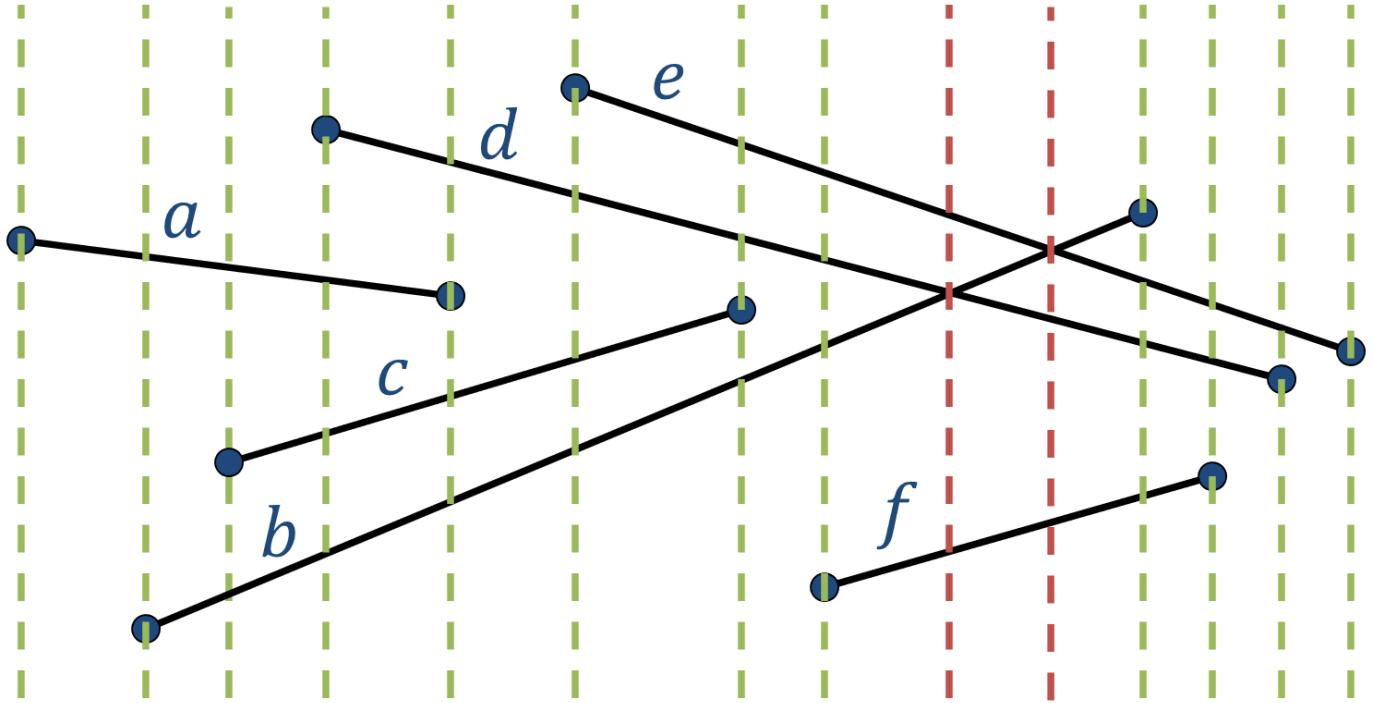
- Hrubé riešenie $O(N^2)$
  - Skontrolovať každú úsečku s každou inou



- Ako by sa to dalo lepšie?

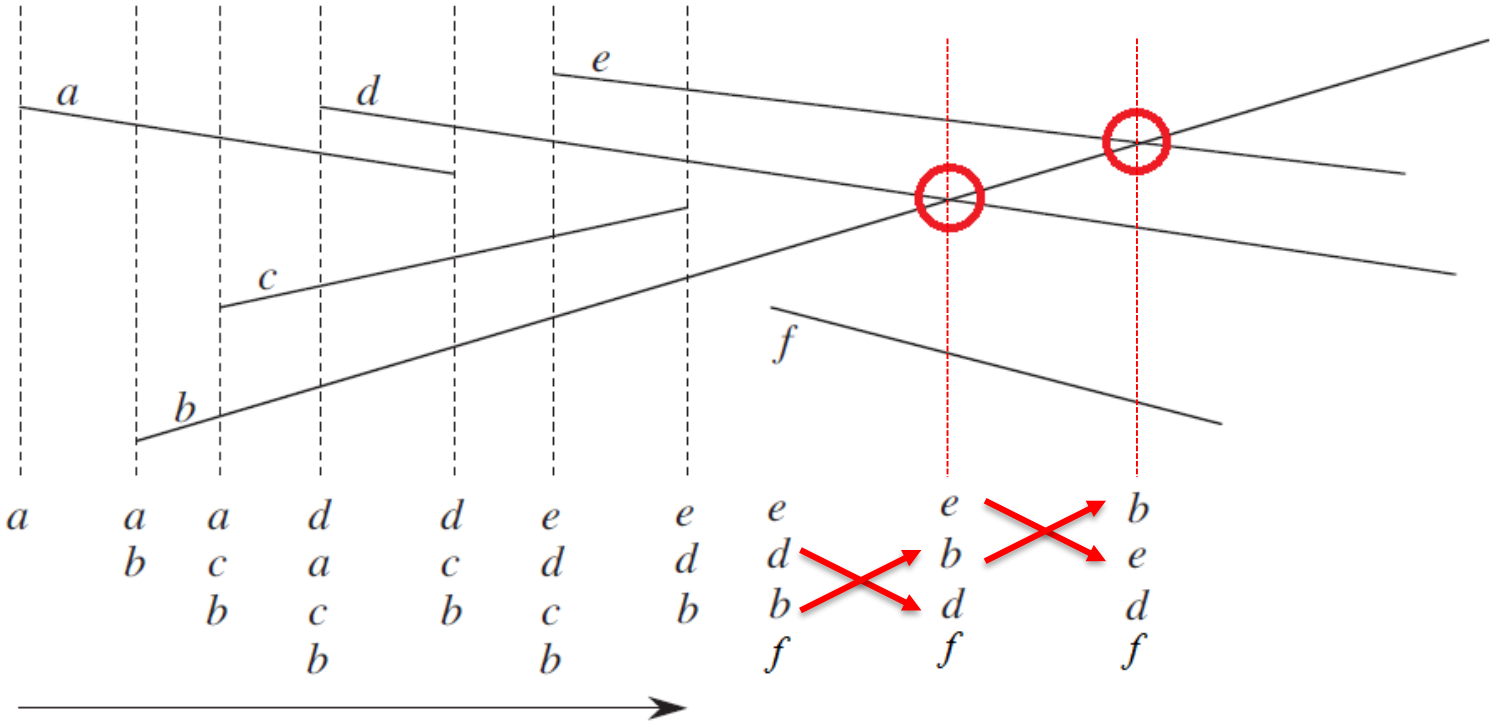
Zametanie – priesečníky úsečiek

- Udalosti:
  - **Koncové body úsečiek**
  - **Priesečníky (vopred ich nepoznáme!)**



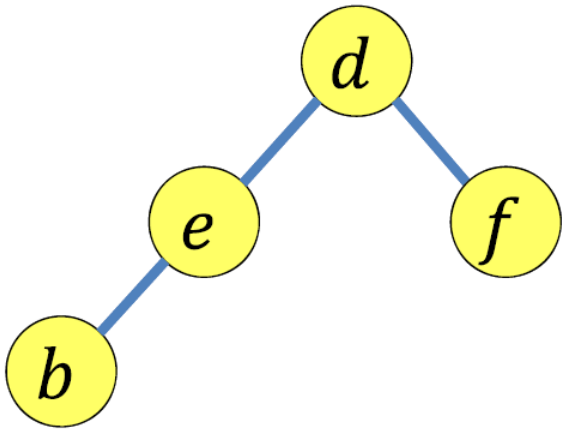
Priesečníky úsečiek – myšlienka algoritmu

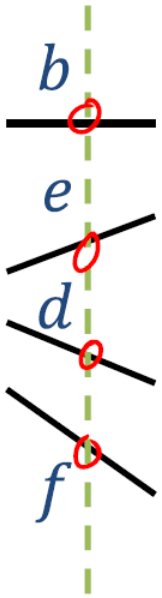
- Udržiavame si poradie (podľa y-ovej súradnice) úsečiek pretínajúcich zametaciu čiaru
- Spracúvame udalosti a vždy keď sa poradie dvoch susedných úsečiek vymení, tak sme **našli priesečník**



Priesečníky úsečiek – schéma algoritmu

- Udržiavame si poradie (podľa y-ovej súradnice) úsečiek pretínajúcich zametaciu čiaru **vo vyhľadávacom strome (napr. AVL)**:





- **Ako určíme kam úsečku vložiť?**
  - **V strome vyhľadávame podľa toho, či je bod NAD úsečkou (ideme doľava) alebo POD úsečkou (ideme doprava)**

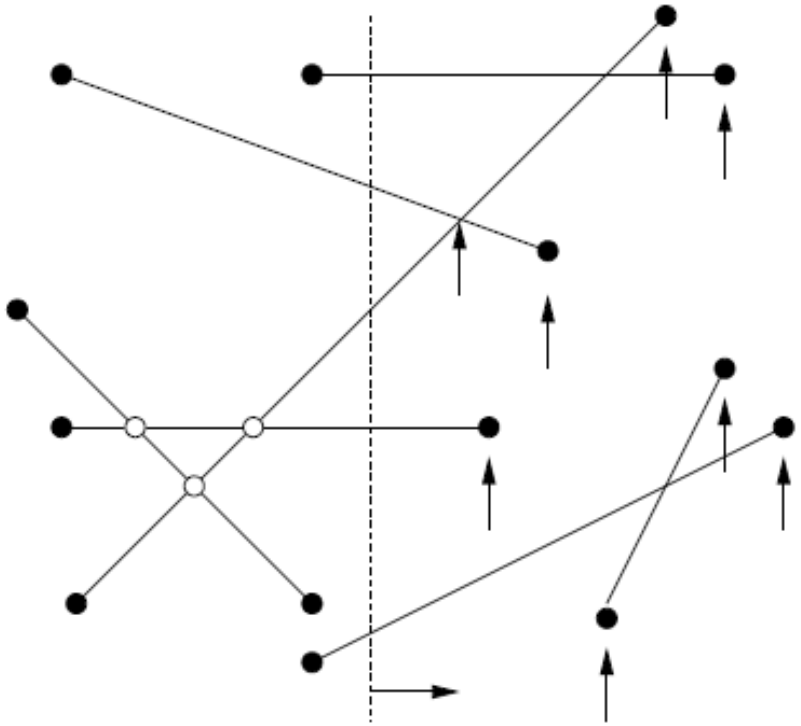
Priesečníky úsečiek – schéma algoritmu (2)

- Udalosti si udržiavame v prioritnom rade Q (podľa x-ovej súradnice, pre rovnakú hodnotu x-ovej súradnice sekundárne podľa y-ovej).
- **Ak zametacia čiara narazila na začiatočný bod úsečky u: pridáme úsečku u do stromu**
  - Pridáme udalosť: koncový bod úsečky
  - Pridáme (možné budúce) udalosti: priesečník(predchodca(u),u), priesečník(u,nasledovník(u))
- **Ak zametacia čiara narazila na koncový bod úsečky u: odstránime úsečku u zo stromu**
  - Odstránime budúce udalosti (priesečníky) týkajúce sa u
- **Ak zametacia čiara narazila na priesečník: vymeníme poradie úsečiek v strome**

Priesečníky úsečiek – pseudokód

```
FIND-INTERSECTIONS(Lines:U)
1 T ← prázdny AVL strom
2 Q ← BUILD-HEAP({u_left, u_right})
3 while Q nie je prázdny:
4   event = EXTRACT-MIN(Q)
5   if event je u_left then
6     INSERT(T, u_left)
7   if event je u_right then
8     DELETE(T, u_right)
9   if event je meet(u,v) then
10    PRINT intersection(u,v)
11    SWAP(T, u, v)
12 foreach u = upravené vrcholy a ich susedia:
13    DELETE(Q, meet(u,*), meet(*,u))
14    INSERT(Q, meet(u,PREDECESSOR(T,u)))
15    INSERT(Q, meet(u,SUCCESSOR(T,u)))
```

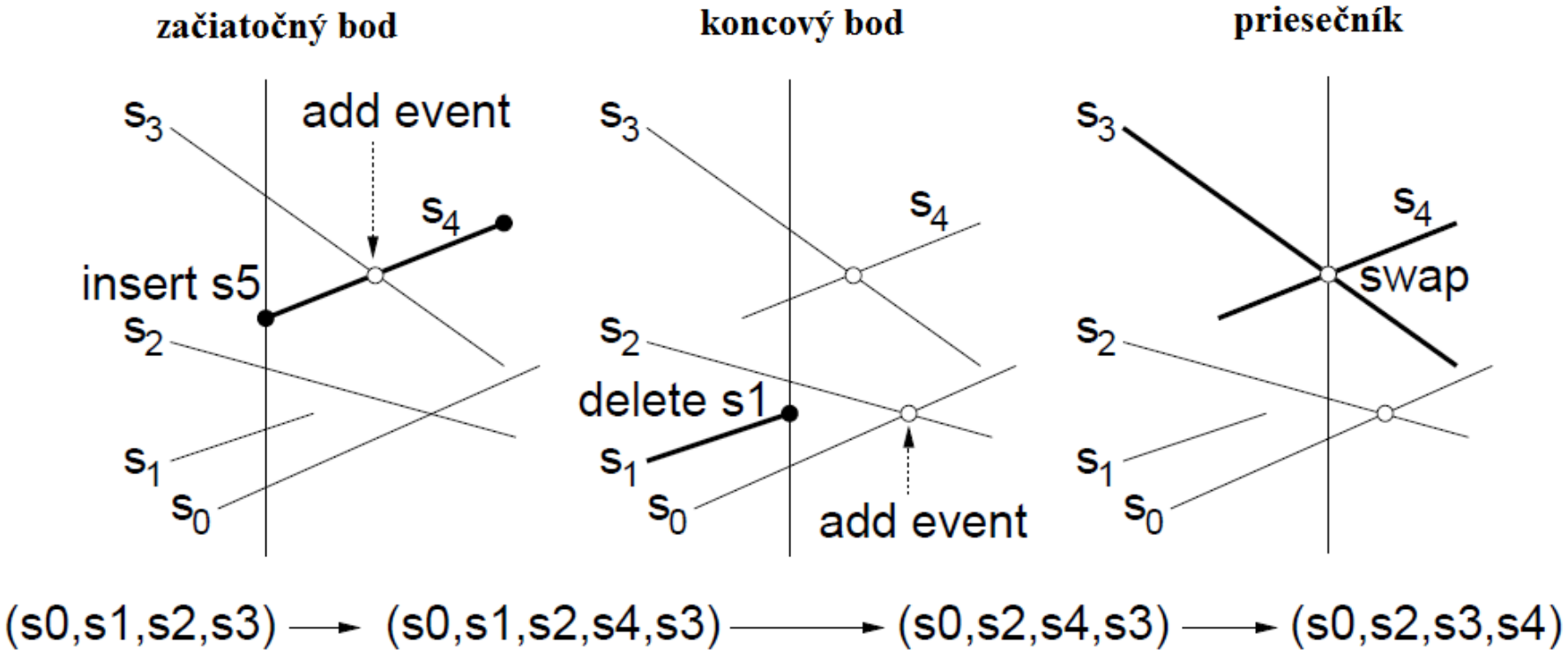
Priesečníky úsečiek – rozbor prípadov



budúce udalosti

nájdené priesečníky

Priesečníky úsečiek – rozbor prípadov



- Ako sa spracujú zvislé úsečky? (netreba ich ošetrovať špeciálne)
  - Narazíme na dolný koncový bod: vložíme horný koncový bod a tiež budúci priesečník so susednou úsečkou,
  - Opakovane spracúvame priesečník tejto úsečky (so susednou): pridáme priesečník s nasledujúcou, až kým nedorazíme do koncového bodu.

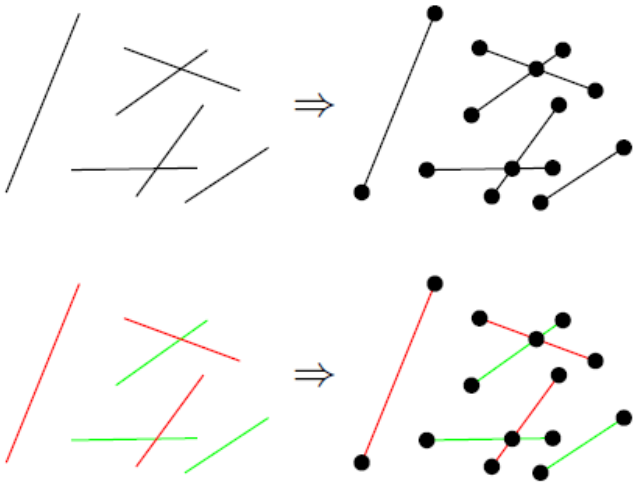
Priesečníky úsečiek – zložitosť

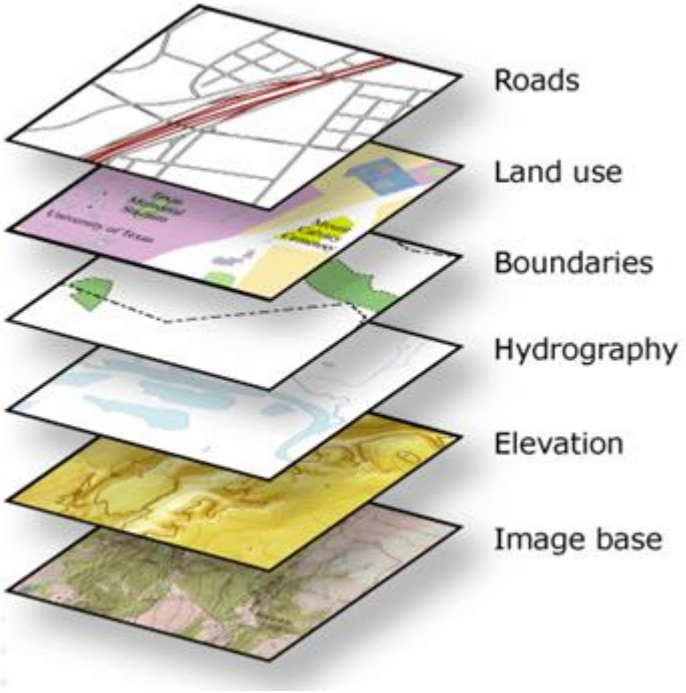
```
M=N+K        FIND-INTERSECTIONS(Lines:U)
             1 T ← prázdny AVL strom
O(N)         2 Q ← BUILD-HEAP({u_left, u_right})
N+K krát:    3 while Q nie je prázdny:
O(logM)      4   event = EXTRACT-MIN(Q)
             5   if event je u_left then
O(logM)      6     INSERT(T, u_left)
O(logM)      7   if event je u_right then
O(logM)      8     DELETE(T, u_right)
K krát:      9   if event je meet(u,v) then
             10    PRINT intersection(u,v)
O(logM)      11    SWAP(T, u, v)
O(1)krát:    12 foreach u = upravené vrcholy a ich susedia:
O(logM)      13    DELETE(Q, meet(u,*), meet(*,u))
O(logM)      14    INSERT(Q, meet(u,PREDECESSOR(T,u)))
O(logM)      15    INSERT(Q, meet(u,SUCCESSOR(T,u)))
```

Celkovo: $O((N+K)\log(N+K))$

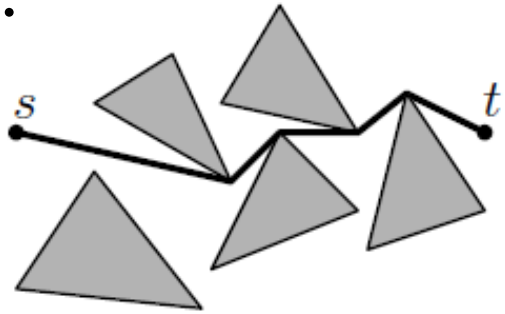
Priesečníky úsečiek ako vrstvy mapy

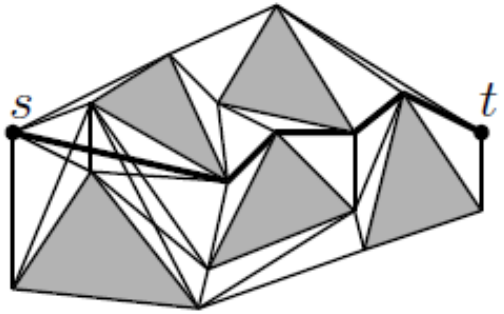
- Prekryv vrstiev na mape:





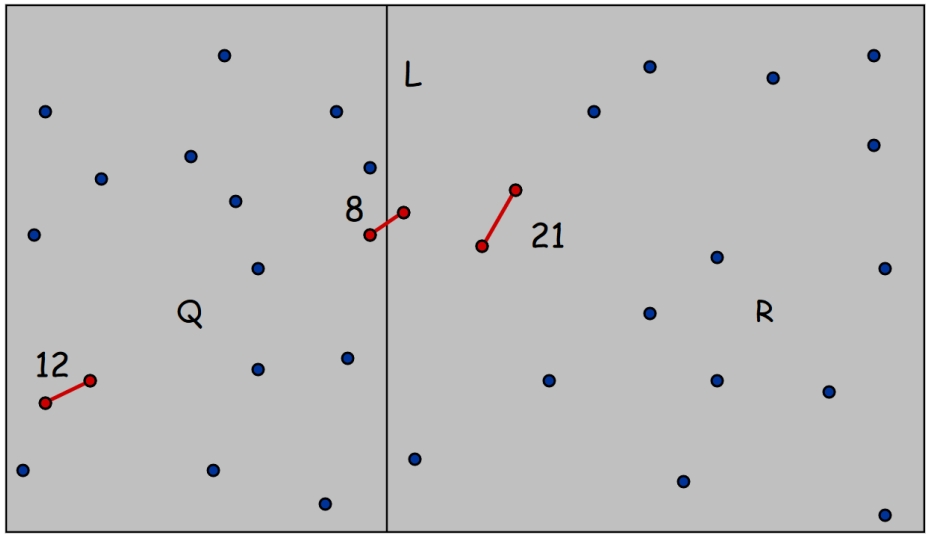
- Viditeľnosť pri pohybe robota v prostredí:





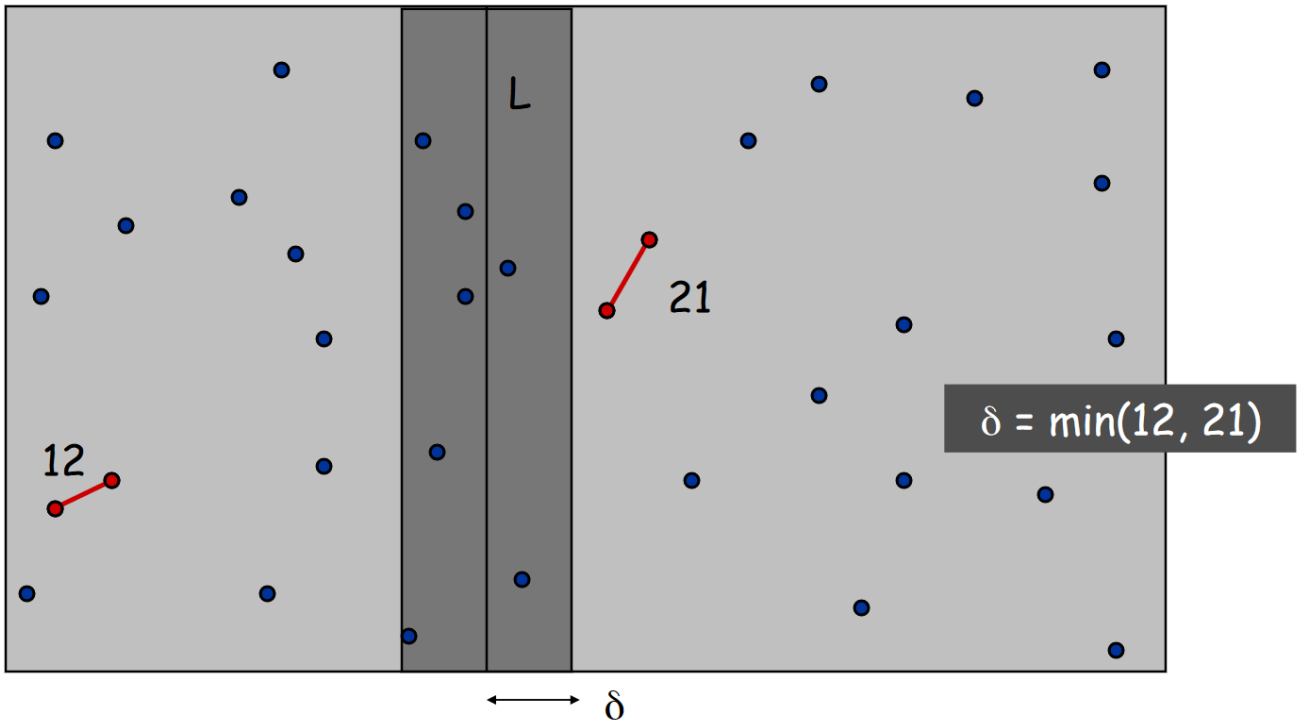
Najbližšia dvojica bodov

- Hrubou silou $O(N^2)$
- Lepšie: rozdeľuj a panuj
  - Rozdeľ podľa x-ovej súradnice na dve polovice
  - Nájdi najbližšiu dvojicu v oboch partíciach
  - Prehľadaj okraj partícií, či tam nie je „bližšia“ dvojica ako sme našli v partíciach...



Najbližšia dvojica bodov (2)

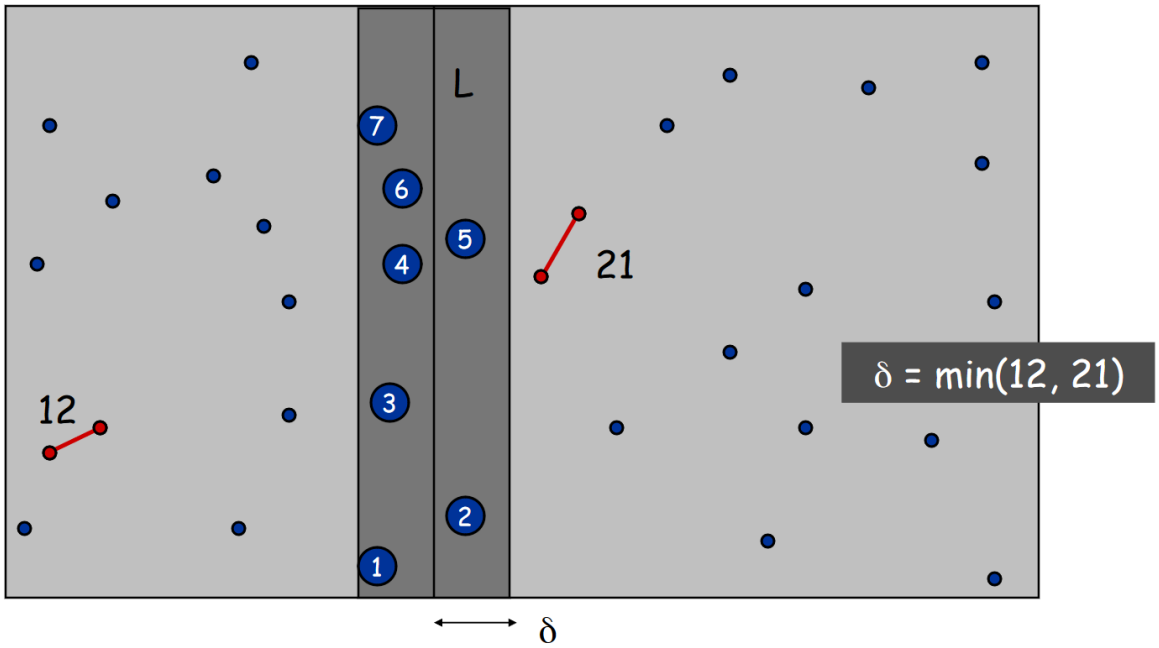
- Ako efektívne prehľadať okraj partícií?
- Zoberme si pás široký najviac ako najlepšie riešenie z partícií:



- Koľko tam môže byť bodov? N ... N^2 dvojíc :(

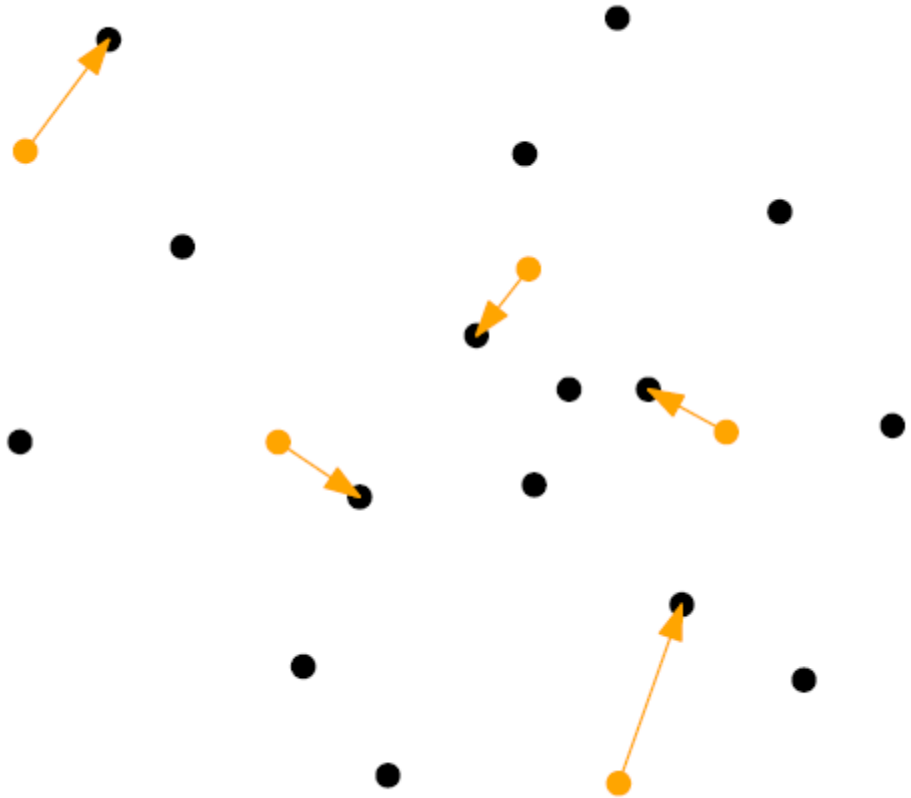
Najbližšia dvojica bodov (3)

- Musíme kontrolovať každú dvojicu bodov v pásiku?
- Stačí len body, ktoré sú blízko seba (aby vzdialenosť nepresiahla najkratšiu vzdialenosť nájdenú v partíciach)



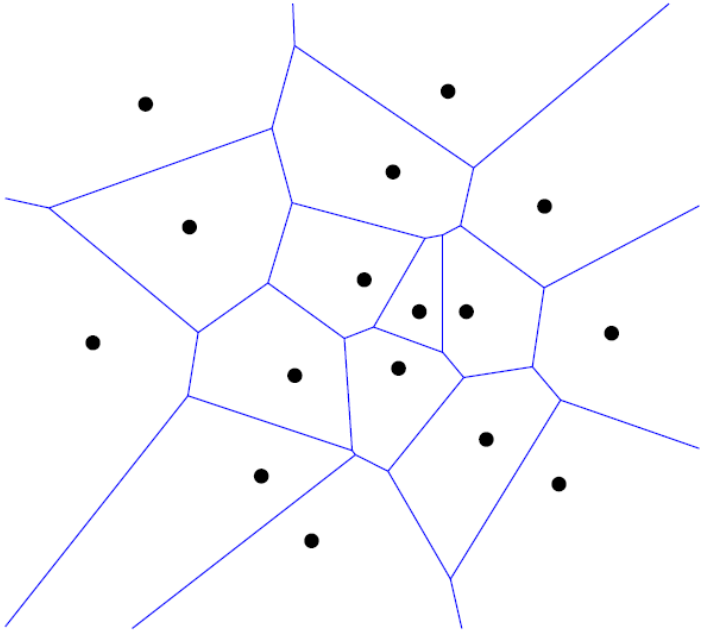
- Ako cvičenie ... stačí pozrieť do 7 susedných v usporiadanom zozname – celková zložitosť potom bude O(N log N)

Všetky najbližšie body



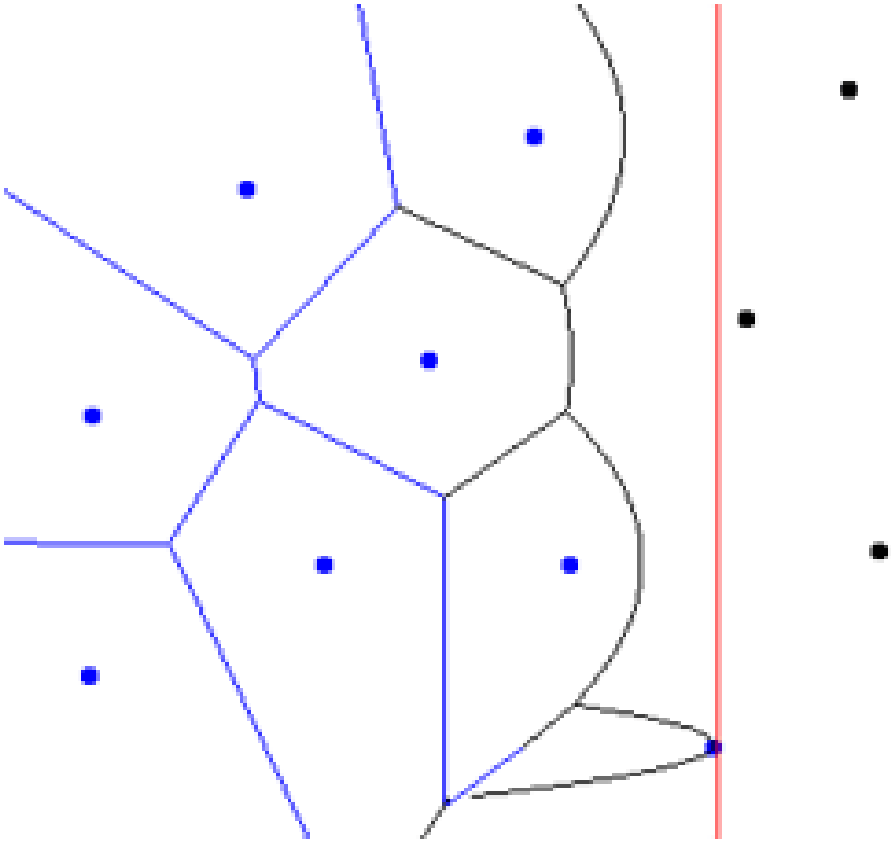
Všetky najbližšie body

- Daná je množina N význačných bodov
  - Voronoiova bunka (pre význačný bod) – množina bodov v rovine, ktoré sú najbližšie k význačnému bodu
  - Voronoiov diagram – množina Voronoiových buniek



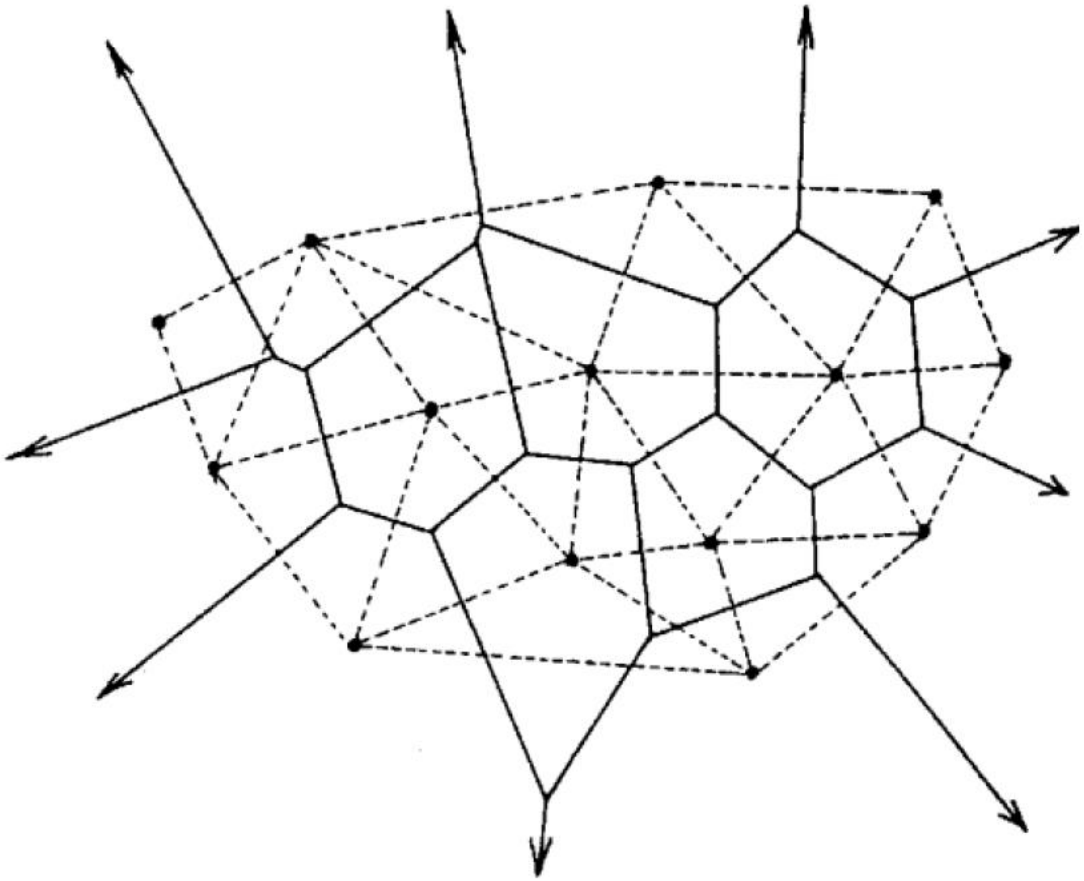
Voronoiov diagram – konštrukcia

- Fortunov algoritmus O(N log N)
  - zametaním



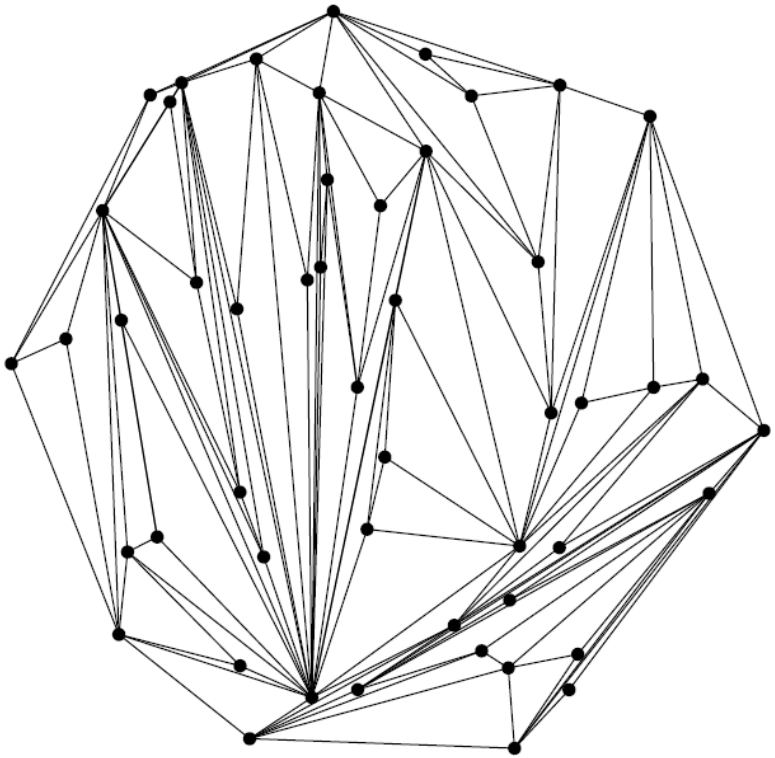
Voronoiove diagramy

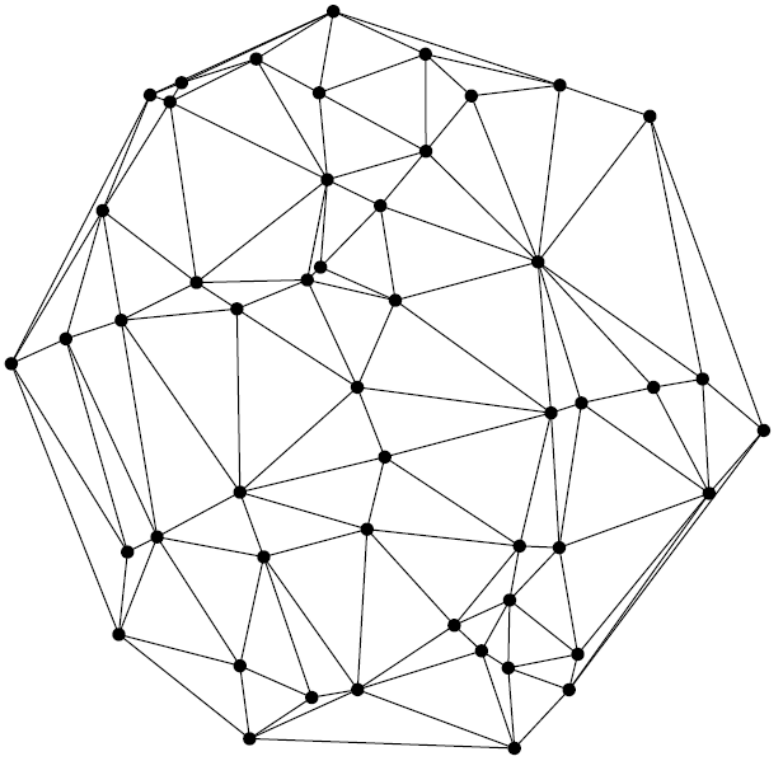
- Duálna štruktúra = Delaunayova triangulácia
  - V istom zmysle je to „najkrajšia“ triangulácia



Delaunayova triangulácia

- Vidíte rozdiel? Ktorá je krajšia? :)

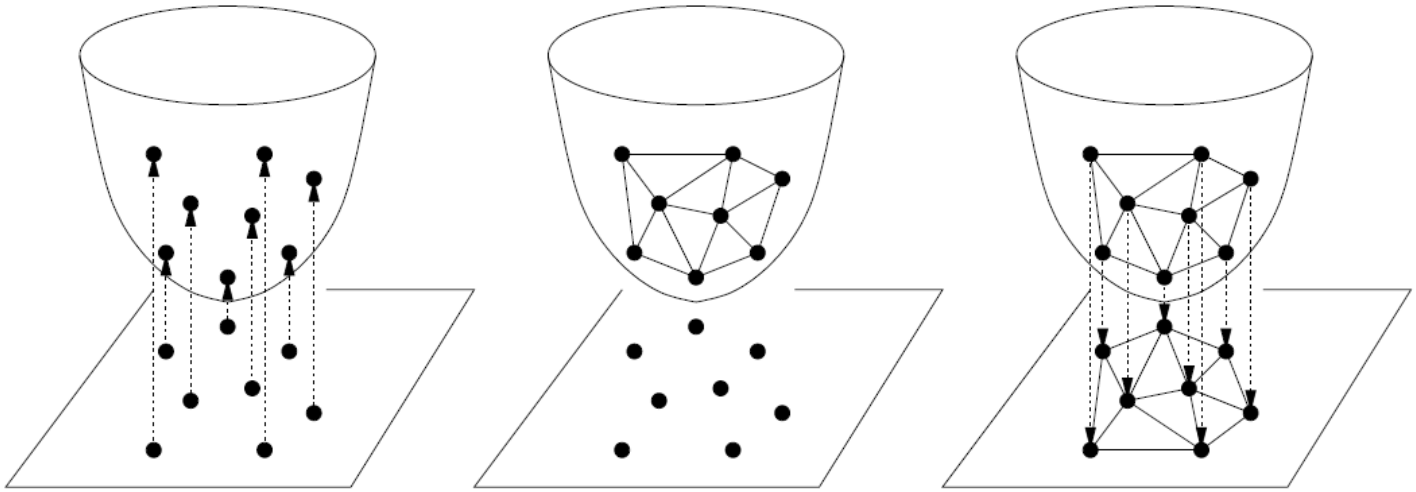




- Najmenší uhol je najväčší možný ...

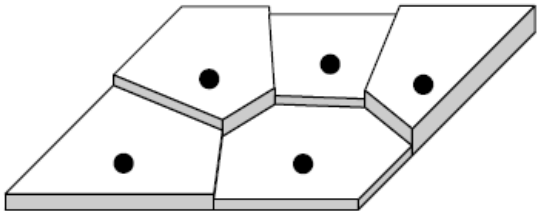
Delaunayova triangulácia a konvexný obal?

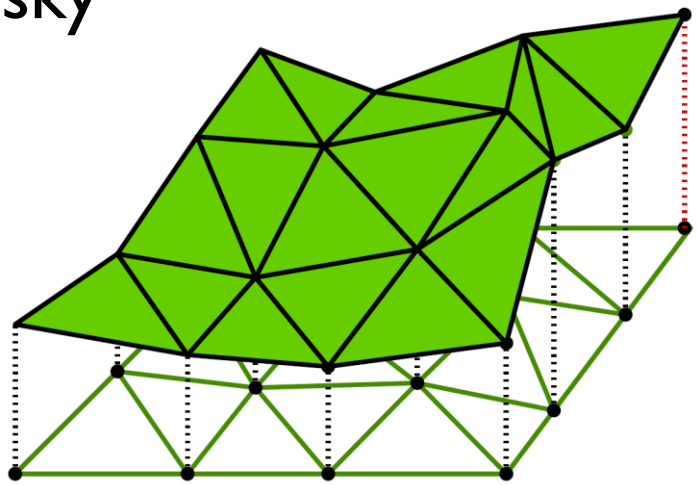
- Zobrazíme body na paraboloid:
  $z = x^2 + y^2$
- Nájdeme konvexný obal vo vyššej dimenzii
- Naspäť premietneme do nižšej dimenzie

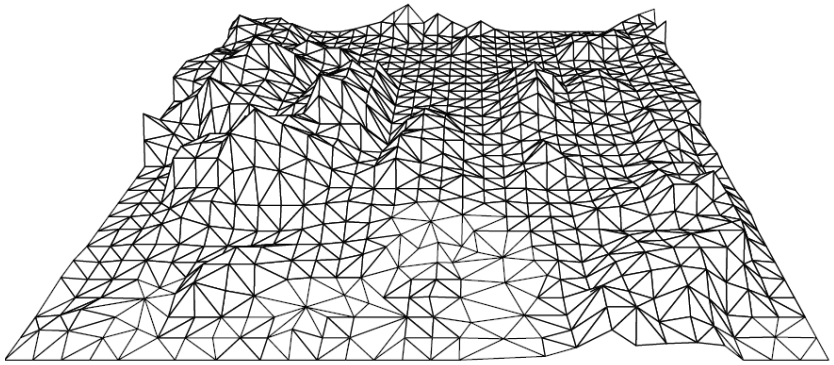


Aplikácie

- Rekonštrukcia terénu
- Dané sú vzorky elevácií v bodoch
- Použitie najbližšieho vzorkovaného bodu (nie je veľmi dobré)
- Lepšie: triangulácia a interpolácia výšky

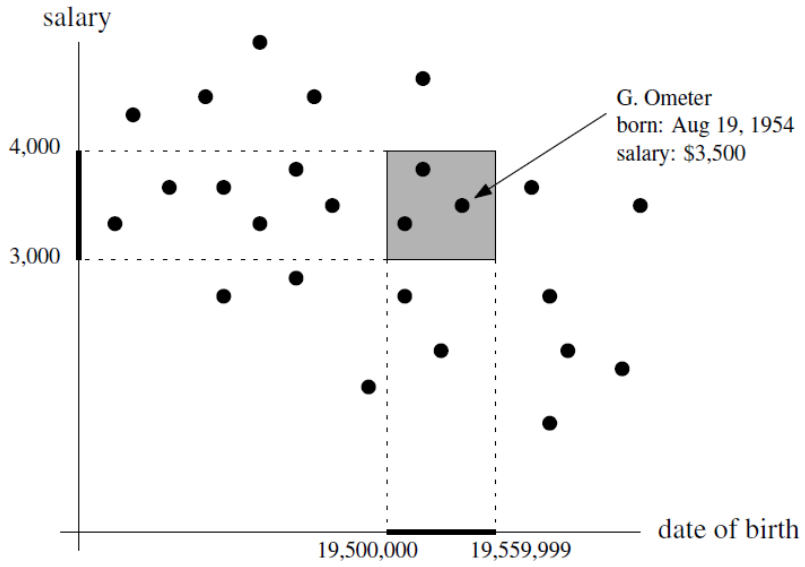


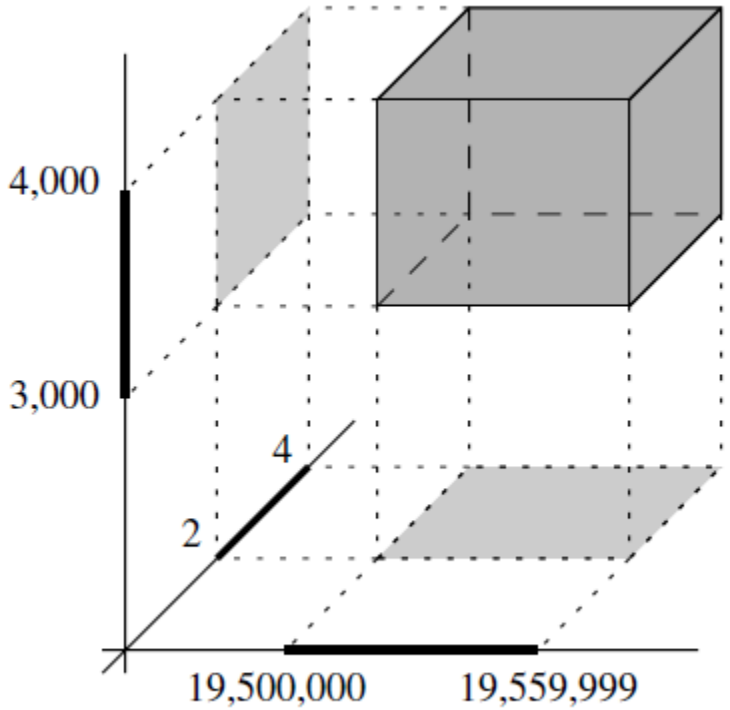




Vyhľadávanie vo viacrozmerných dátach

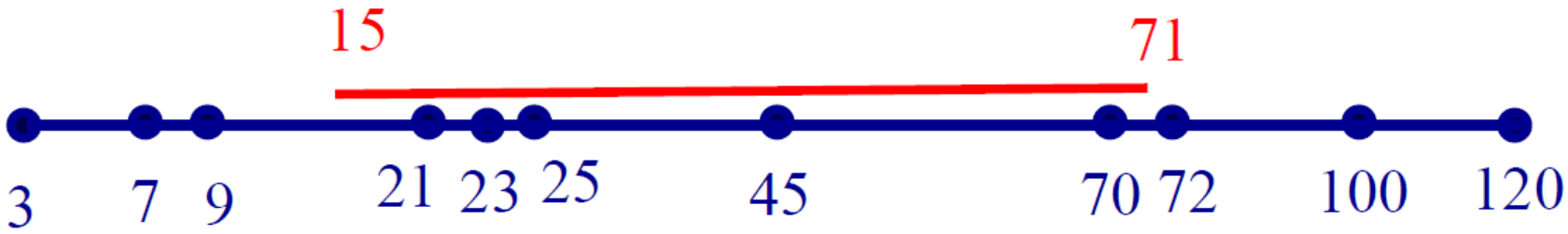
- N bodov, nájdi body v obdĺžniku, kružnici, ...
- Ortogonálne vyhľadávanie





Vyhľadávanie vo viacrozmerných dátach

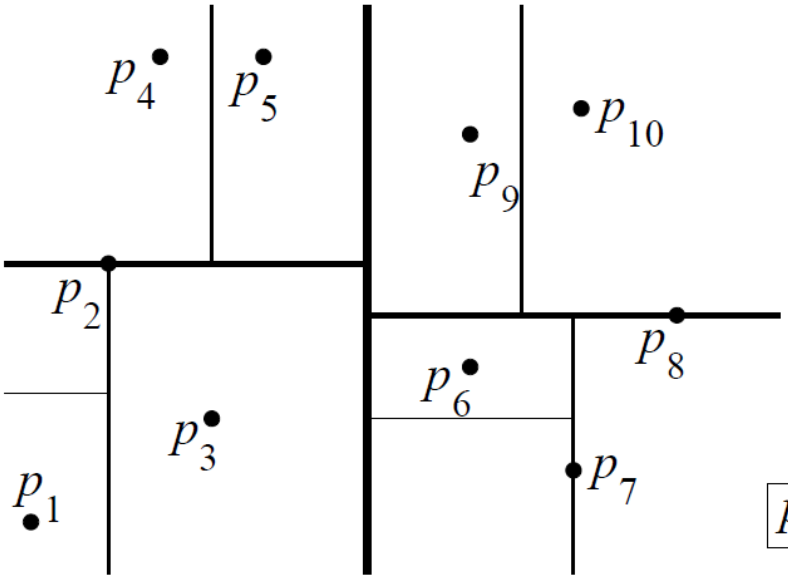
- V jednom rozmere: 1D vyhľadávanie
- Query je interval



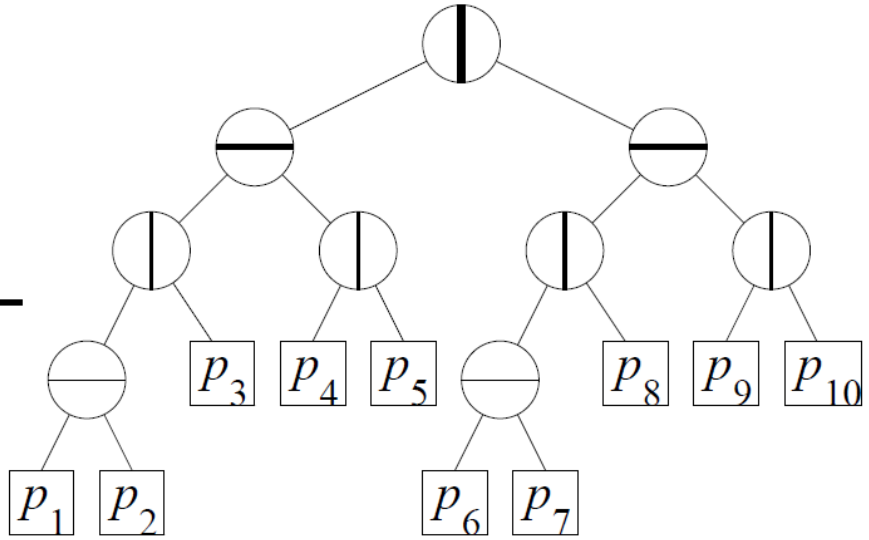
- Intervalový (binárny vyhľadávací strom)
  O(N log N + K)
  kde K je počet výsledkov

Vyhľadávanie vo viacrozmerných dátach

- Vo vyšších rozmeroch: kD stromy
- Query je obdĺžnik



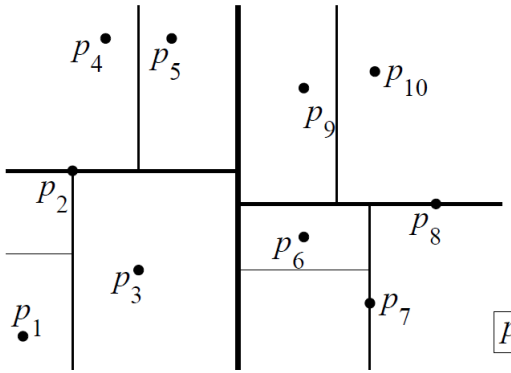
Rozdelenie roviny



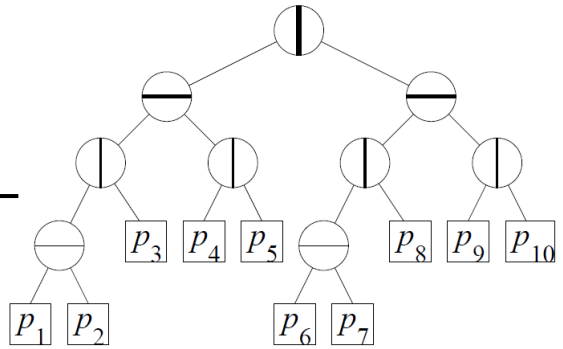
Stromová štruktúra

Vyhľadávanie vo viacrozmerných dátach

- Vo vyšších rozmeroch: kD stromy
- Query je obdĺžnik



Rozdelenie roviny

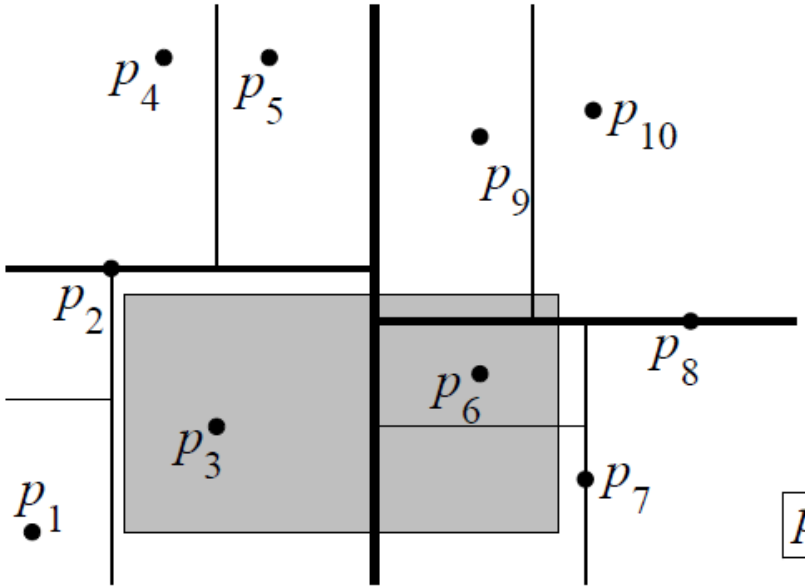


Stromová štruktúra

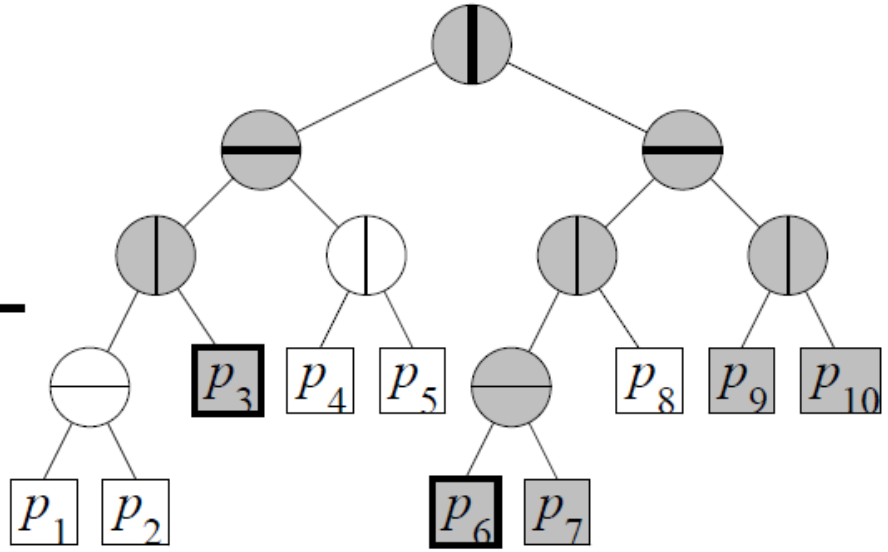
- Každý vrchol je ortogonálny rez roviny podľa x-ovej alebo y-ovej súradnice
- Dimenzie rezov sa striedajú x, y, x, y, ...
- Vytvorenie O(N log N)

Vyhľadávanie vo viacrozmerných dátach

- Vyhľadanie v kD strome
  - Vnárame sa do vrcholov, do ktorých zasahuje query



Vyhľadávaný rozsah



Navštívené vrcholy stromu

  - Vyhľadanie O(sqrt(N) + K), kde K je veľkosť výstupu

Posledný priestor na otázky ...