Datové struktury a algoritmy

Část 12

Výpočetní geometrie

Petr Felkel

9.1.2007

Výpočetní geometrie
Computational Geometry (*CG*)

Úvod

Příklady úloh

Algoritmické techniky – paradigmata

- řazení - jako předzpracování
- rozděl a panuj (*divide and conquer*)
- zametací technika (*scan-line*)
- geometrické místo (*Locus approach*)

Výpočetní geometrie?

Vznikla „v roce 1975“ – M. I. Shamos
(1850 Dirichlet, 1908 Voronoi, … maximum článků 1991)

Dvojí cíl: Zkoumá *teorii* (kombinatorickou strukturu geometrických objektů)
i *praxi*: Hledá *optimální algoritmy* pracující s *geometrickými objekty* a implementuje je

- body, přímky, úsečky, mnohoúhelníky – polygony,…

Aplikace např. v oblastech:

- databázové systémy, robotika, počítačová grafika, počítačové vidění, rozpoznávání obrazů, …
- řadu problémů lze formulovat geometricky

Výpočetní geometrie?

Charakteristická je

- velká probádanost algoritmů v rovině (2D)
- nárůst složitosti v prostoru (3-D) či v n-dimenzích
- méně znalostí o řešení v n-dimenzích

dále zůstaneme v rovině (2D)

- ukázky typických úloh
- ukázky technik při návrhu algoritmů

Výpočetní geometrie (CG)

Úvod

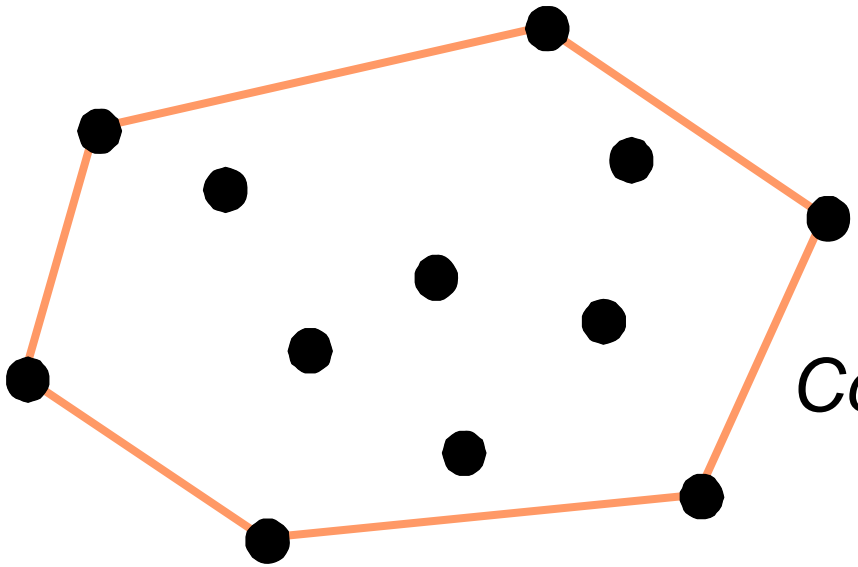
Příklady úloh

Algoritmické techniky – paradigmata

- řazení - jako předzpracování
- rozděl a panuj (*divide and conquer*)
- zametací technika (*scan-line*)
- geometrické místo (*Locus approach*)

Konvexní obálka

= nejmenší konvexní mnohoúhelník, který obsahuje všechny zadané body

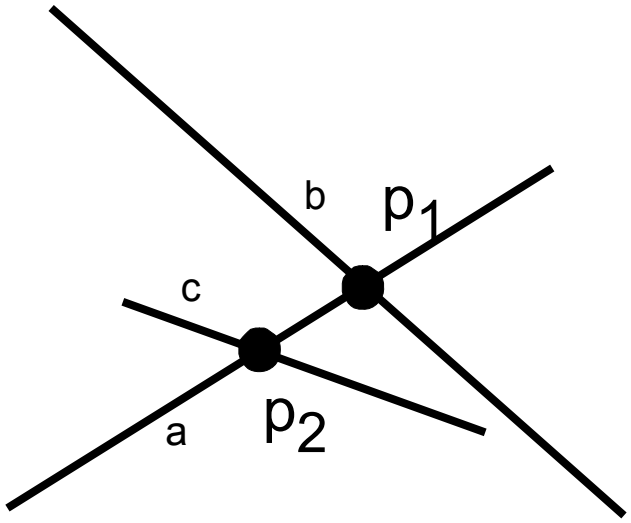


V – množina bodů

Convex Hull CH(*V*)

Průsečík úseček

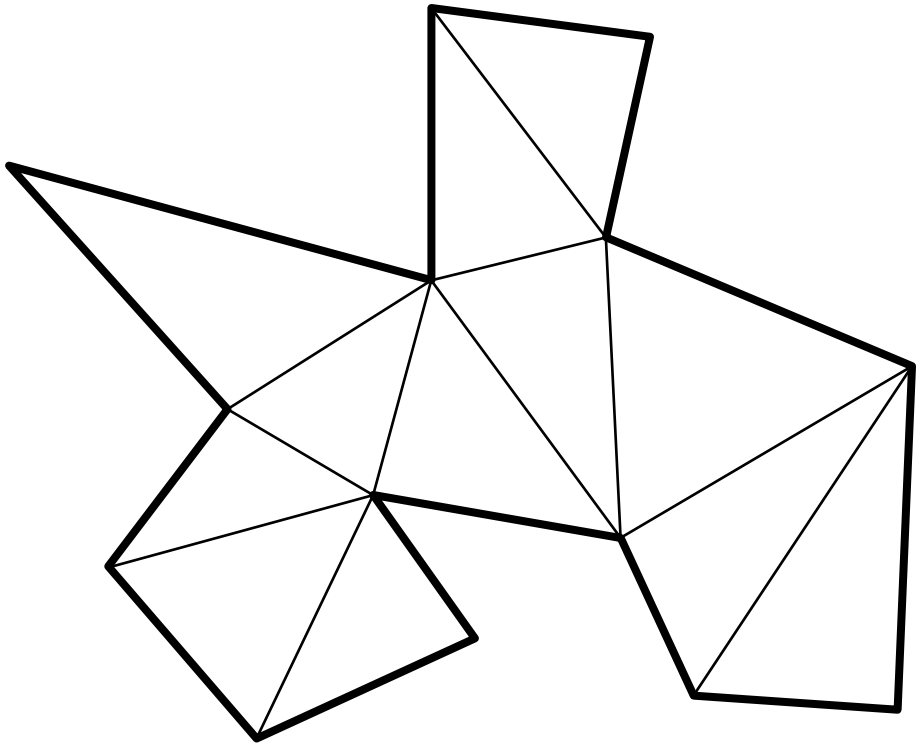
Problém: Zjistit, zda se objekty protínají



- Průsečík složitého objektu
  rozložit
  na průsečíky částí
- např. na průsečíky úseček

Triangulace mnohoúhelníka

Problém: Rozklad objektu na nepřekrývající se části

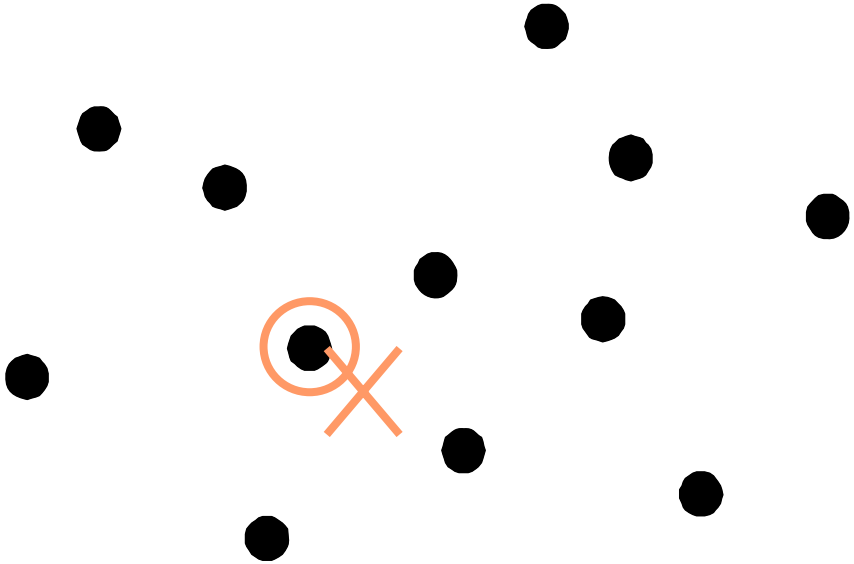


- complex -> *simplex*
  - 2D – trojúhelníky
  - 3D - čtyřstěny (tetrahedron)

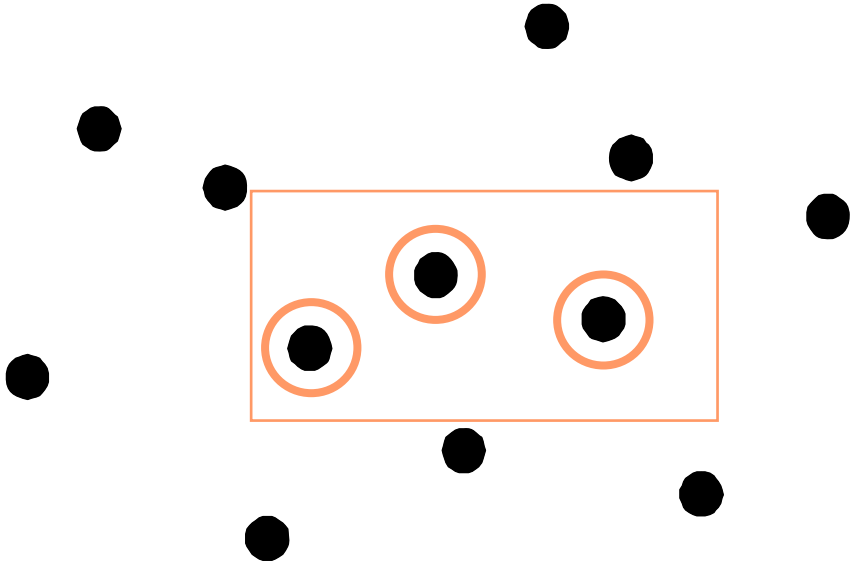
Vyhledávání

Předzpracování - Vytvoření struktury pro rychlé nalezení:

Nejbližšího souseda
(*nearest neighbor*)

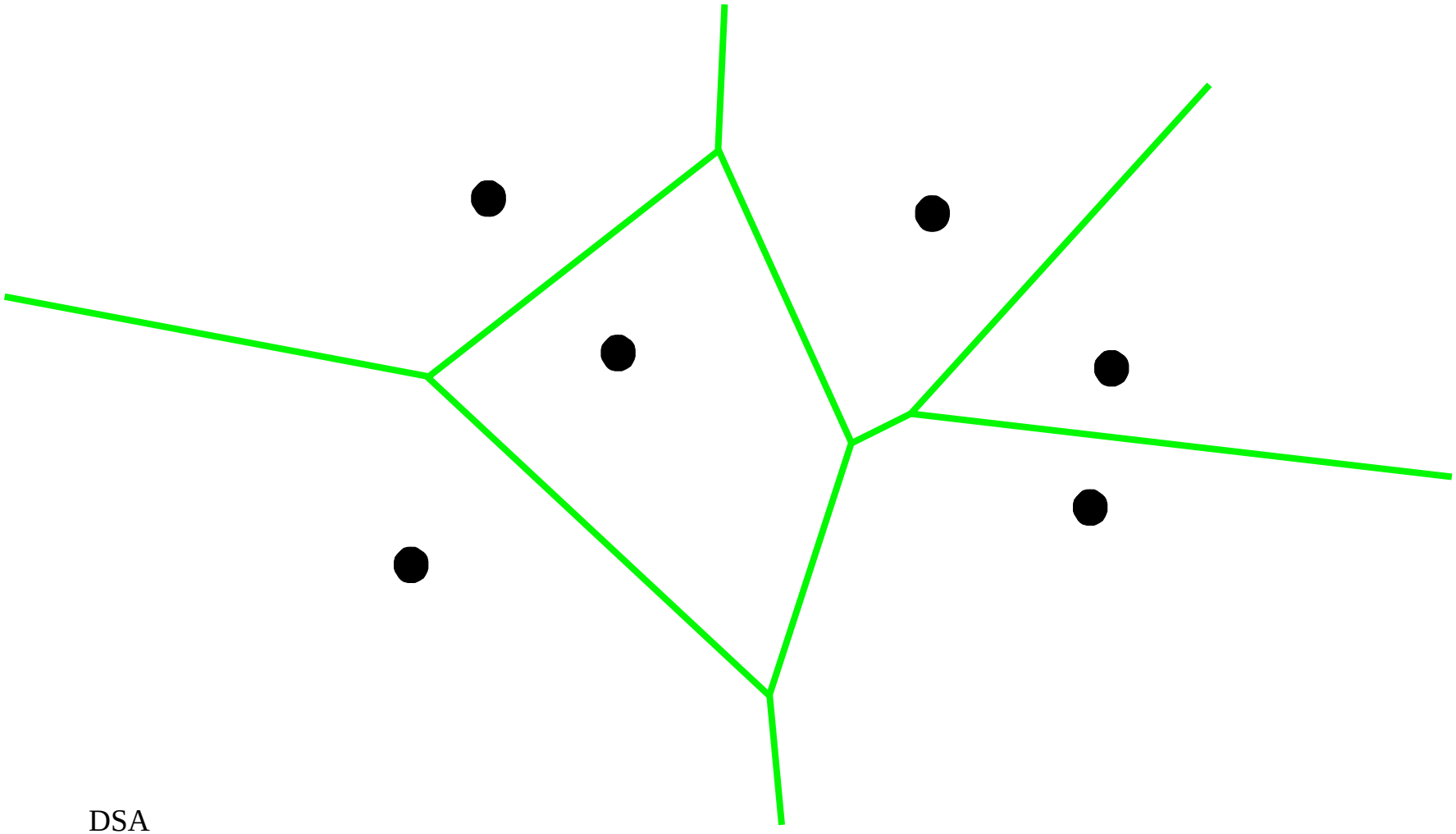


Bodů v daném rozsahu
(*range query*)



Voroného diagram

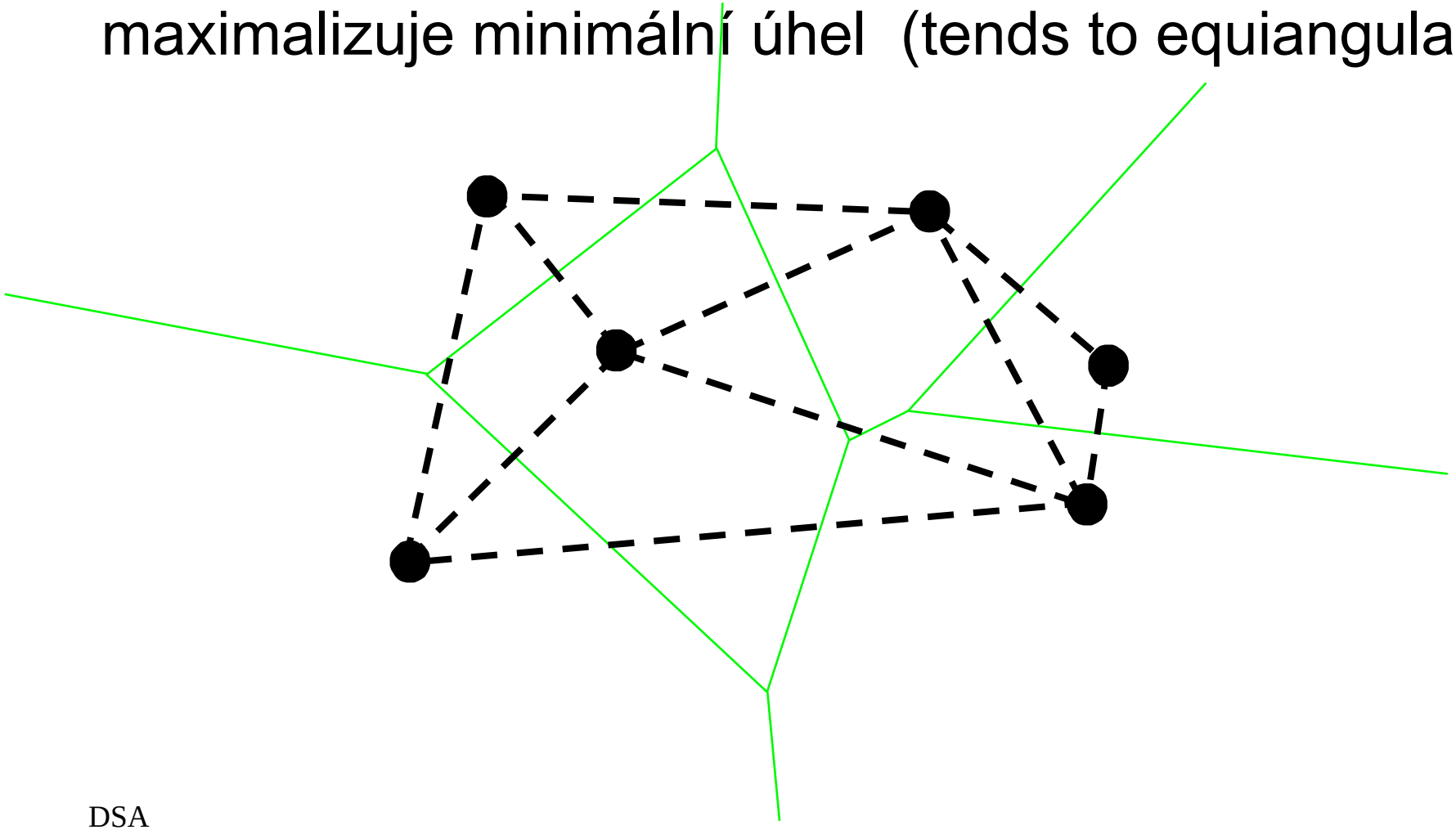
= Struktura pro vyhledání nejbližšího souseda



Delaunayova triangulace

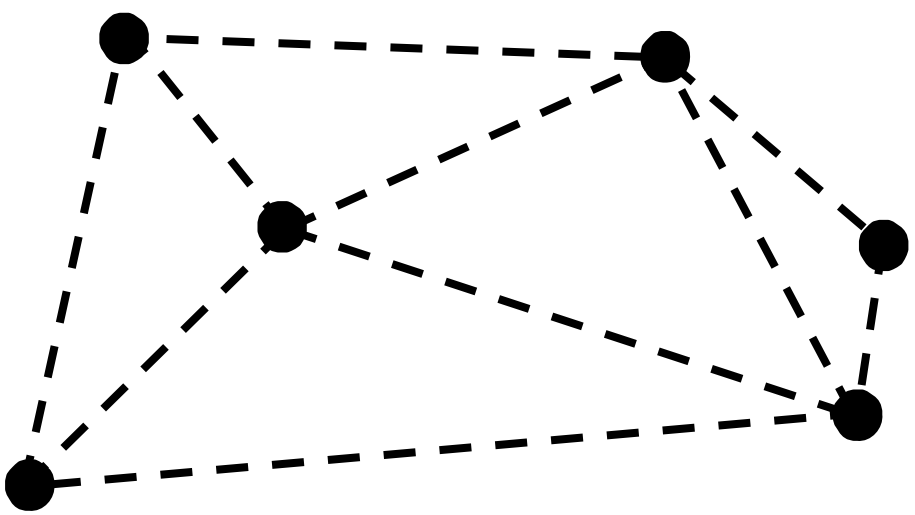
= triangulace množiny bodů (duální k VD)

maximalizuje minimální úhel (tends to equiangularity)

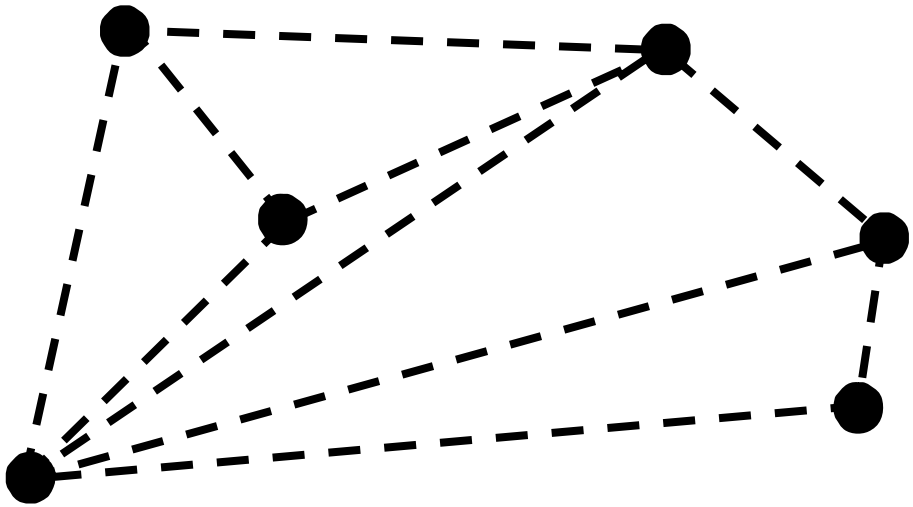


Delaunayova triangulace

Delaunayova



Není Delaunayova



Výpočetní geometrie (CG)

Úvod

Příklady úloh

Algoritmické techniky – paradigmata

- řazení - jako předzpracování
- rozděl a panuj (*divide and conquer*)
- zametací technika (*scan-line*)
- geometrické místo (*Locus approach*)

Algoritmické techniky – paradigmata

= principy návrhu efektivních algoritmů

zůstávají stejné i pro velmi odlišné aplikace

složitost

– hrubá síla – prohledá (zkusí) všechno

“O >> Ω“

– efektivní algoritmus – co nejúspornější,

ale “O > Ω”

– optimální algoritmus – dosáhl dolní meze složitosti

“O = Ω = Θ”

Výpočetní geometrie (CG)

Úvod

Příklady úloh

Algoritmické techniky – paradigmata

- řazení - jako předzpracování
- rozděl a panuj (*divide and conquer*)
- zametací technika (*scan-line*)
- geometrické místo (*Locus approach*)

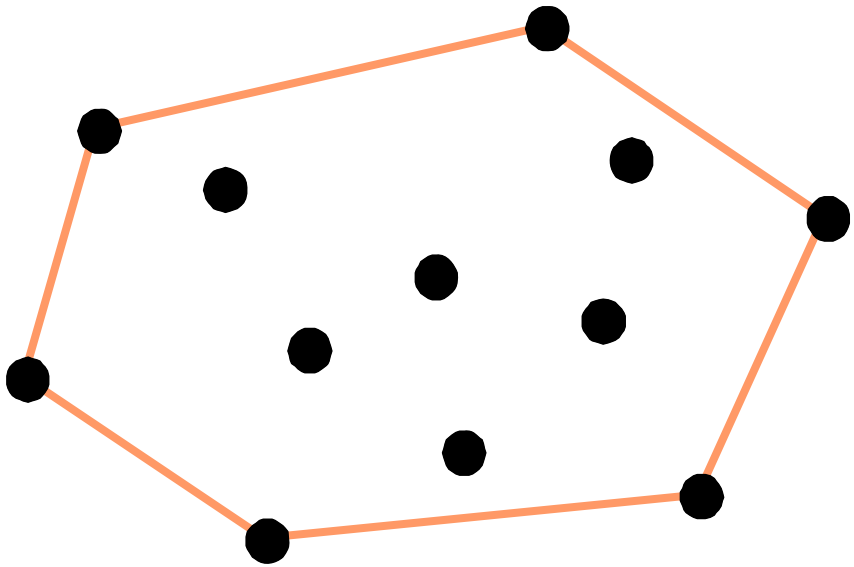
Řazení

- předzpracování dat, které
- vede k jednoduššímu zpracování
- typicky
  - podle některé ze souřadnic
    (např. dle osy *x* či *y*)
  - nebo dle úhlu kolem daného bodu/ů

Př. použití: konvexní obálka

Konvexní obálka

= nejmenší konvexní mnohoúhelník, který obsahuje všechny zadané body

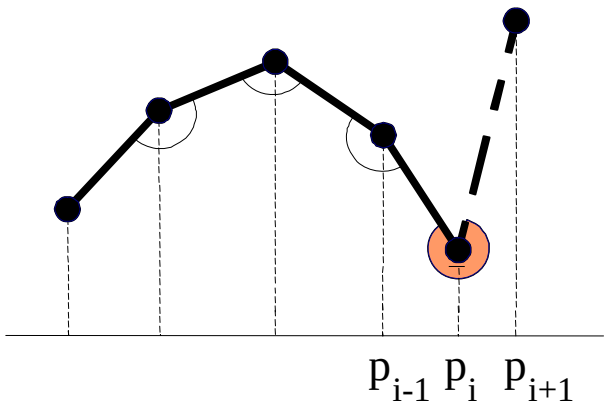


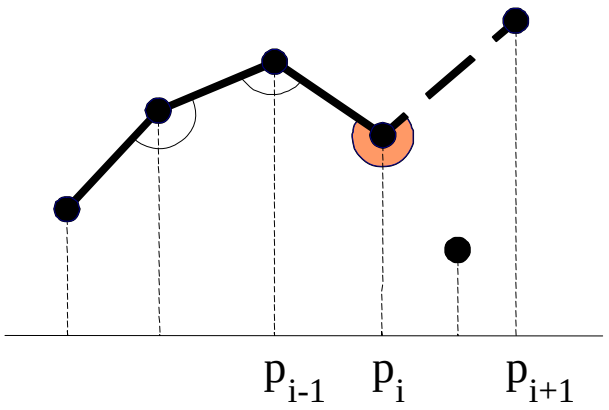
V – množina bodů

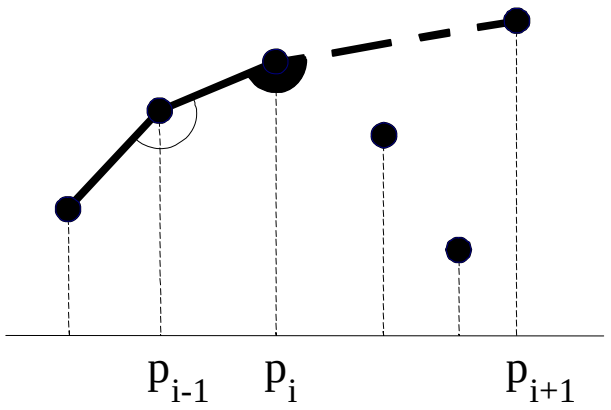
Convex Hull CH(*V*)

Grahamův algoritmus 1/3

1. Seřadíme body $p \in V$ dle x; (Orig: podle úhlu k x-)
2. Najdeme horní, pak dolní řetěz
   a) p_{min} (a p_{max}) $\in CH(V)$. Vezmi $p_1=p_{min}$, p_2=další, i=2
   b) přidáme bod p_{i+1}, i=i+1
   c) kontrola úhlu (p_{i-1}, p_i, p_{i+1})
      - **konvexní**
      - **konkávní**: vypustíme p_i, i=i-1

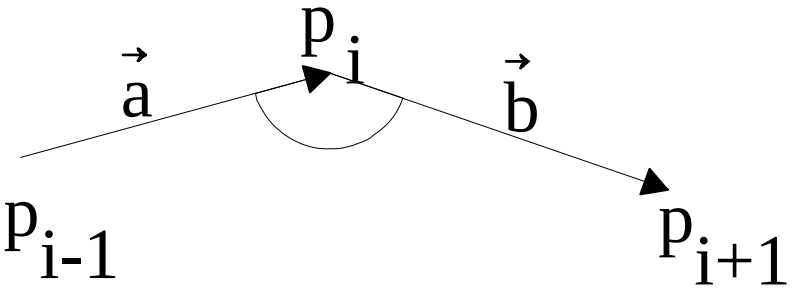






Grahamův algoritmus 2/3

Kontrola úhlu (p_{i-1}, p_i, p_{i+1})



Vektorový součin $\vec{b} \times \vec{a}$

- kolmý na rovinu => jediná z-ová složka > 0, x- a y-ová = 0

$$\begin{vmatrix} x_{i+1} - x_i & y_{i+1} - y_i \\ x_i - x_{i-1} & y_i - y_{i-1} \end{vmatrix} \begin{cases} > 0 & \Rightarrow \textbf{konvexní (R)} \\ < 0 & \Rightarrow \textbf{konkávní (L)} \end{cases}$$

$$\begin{vmatrix} b_x & b_y \\ a_x & a_y \end{vmatrix} = b_x a_y - a_x b_y$$

$$\vec{a} = p_i - p_{i-1} = (a_x, a_y)$$

$$\vec{b} = p_{i+1} - p_i = (b_x, b_y)$$

$$p_i = [x_i, y_i, z_i]$$

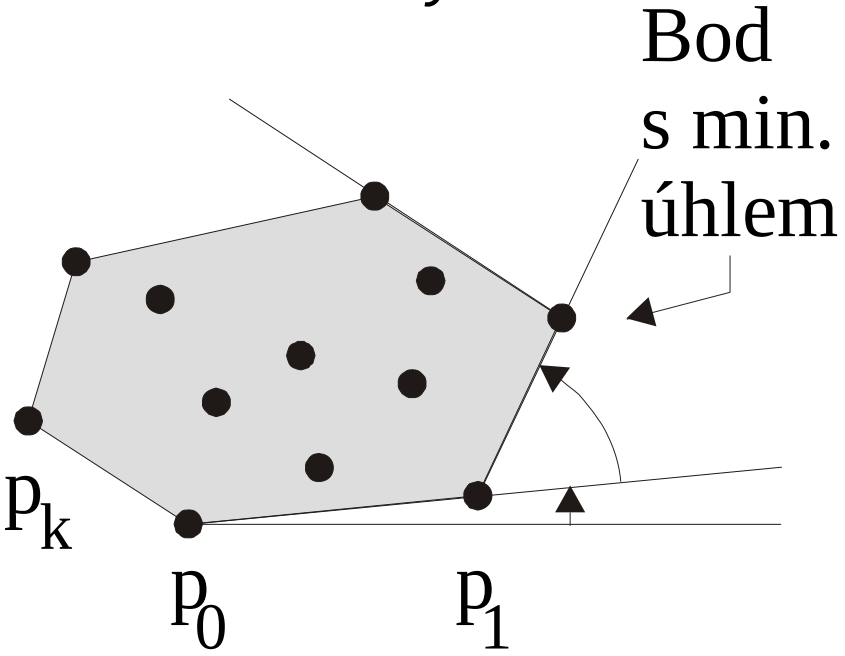
Grahamův algoritmus 3/3

Složitost

1. Seřazení bodů dle x	O(*n* log *n*)
2. Nalezení horního a dolního řetězu	O(*n*)
=> celkem	O(*n* log *n*)

Jarvisův algoritmus balení dárku (*gift wrapping*)

1. Vezmeme bod p s minimální souřadnicí *y* a vodorovnou přímku
2. Otáčíme přímku kolem p dokud nenarazí na bod q
3. p = nový nejbližší bod q
4. Dokud ($p \neq p_0$) jdi na 2



Složitost: O(*n*) + O(*n*) * *k* => celkem O(*k***n*) vhodný pro málo bodů na konvexním obalu

Výpočetní geometrie (CG)

Úvod

Příklady úloh

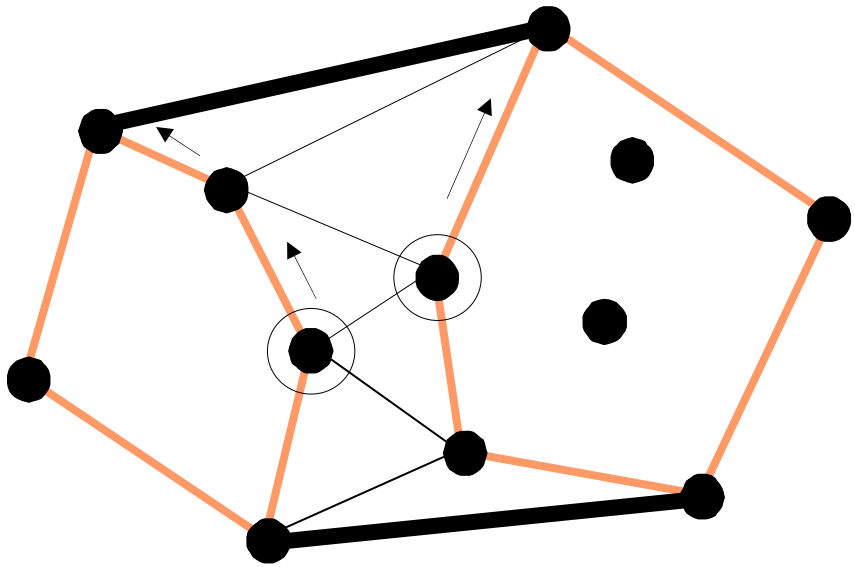
Algoritmické techniky – paradigmata

- řazení - jako předzpracování
- rozděl a panuj (*divide and conquer*)
- zametací technika (*scan-line*)
- geometrické místo (*Locus approach*)

Konvexní obálka metodou D&C

```
Seřaď body dle x

Obálka(X) // Rekurzivně
  Děl X na 2 části L a P
  Obálka(L), Obálka(P)
  Spoj obálky(L,P)
     Horní most
        (nejbližší body, L proti, P po směru
  hodin)
     pak dolní most (naopak)
```



Výpočetní geometrie (CG)

Úvod

Příklady úloh

Algoritmické techniky – paradigmata

- řazení - jako předzpracování
- rozděl a panuj (divide and conquer)
- zametací technika (Plane-sweep, *scan-line*)
- geometrické místo (*Locus approach*)

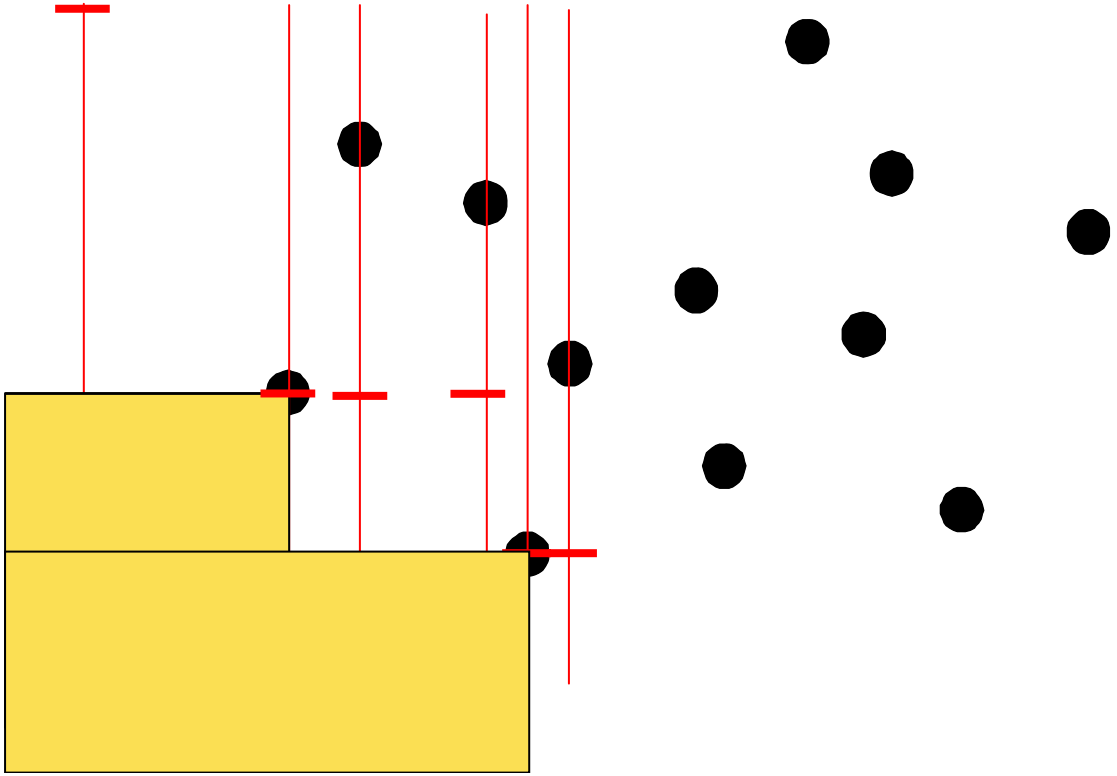
Zametací technika – princip

- Svislou přímku (*scanline, SL,* zametací přímku) suneme zleva doprava přes množinu objektů
- Nesuneme se spojitě, ale skáčeme mezi body, kde je nutno zastavit
- Pamatujeme si informace o objektech nalevo od zametací přímky (*y*-stuktura, *T*)
- Při průchodu nad objektem *y*-stukturu aktualizujeme
- Body jsou v prioritní frontě (*postupový plán, x*-struktura, *B*)

Příklad 1: Minimální body

Od nich nalevo ani dolů není žádný bod

1. Seřadíme body dle souřadnice *x* -> x-struktura
2. Inicializujeme y-strukturu (1 hodnota) na ∞
3. Scanline umísťujeme zleva doprava do bodů v x-str.
4. Do y-strukt. ukládám minimální y-souřadnici



O(n log n) - řazení

O(1) na bod

O(n) pro všechny body

O(n log n) - celkem

Příklad 2: Průsečíky úseček

Nalezení všech průsečíků zadaných úseček

– rychleji než každá s každou, tj. než $O(n^2)$

– počítám průsečíky jen mezi sousedními úsečkami v T

T, y-struktura = úsečky v pořadí jak protínají scanline

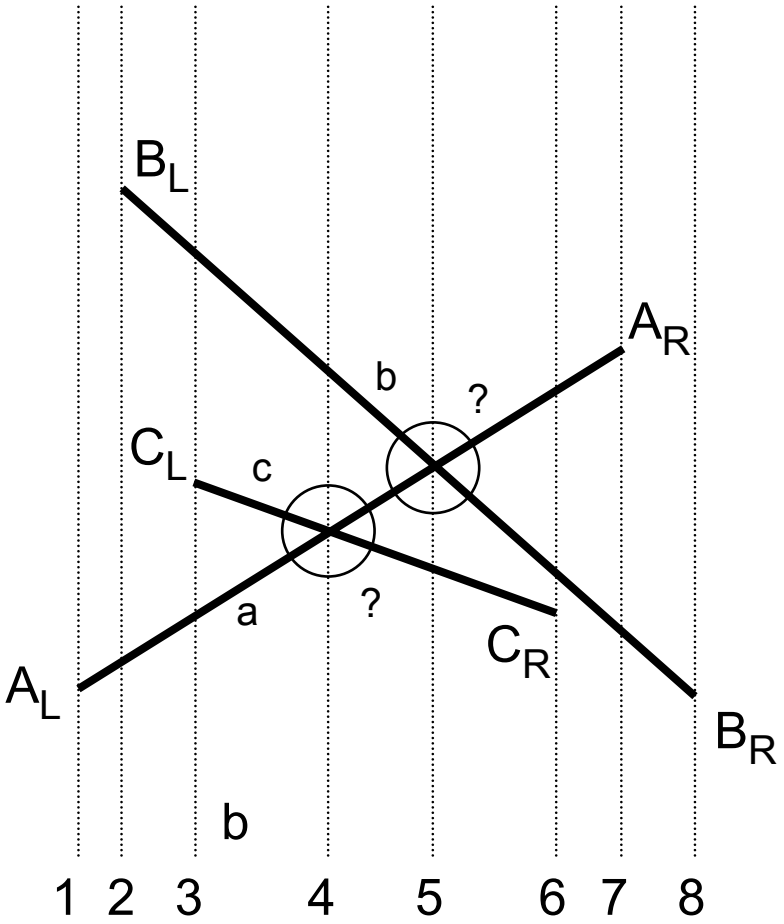
B, postupový plán, x-struktura

= body, kde se mění pořadí úseček:

– na začátku jen *koncové body* úseček

– průběžně vkládám *průsečíky* sousedních úseček v T

Průsečíky úseček



1. Inicializace
   - koncové body –> B
   - T prázdné

B: A_L, B_L, C_L, C_R, A_R, B_R

T: prázdná1

Průsečíky úseček

2. dokud není *B* prázdná

- vezmi bod *p* z *B* (a smaž ho v *B*)
- dle typu bodu *p* aktualizuj *T* strukturu:
  - L ... začíná nová úsečka *s* – oddělí bývalé sousedy
  - P ... končí úsečka s, její sousedi budou sousedy
  - průsečík ... průsečík úseček *s* a *s'*

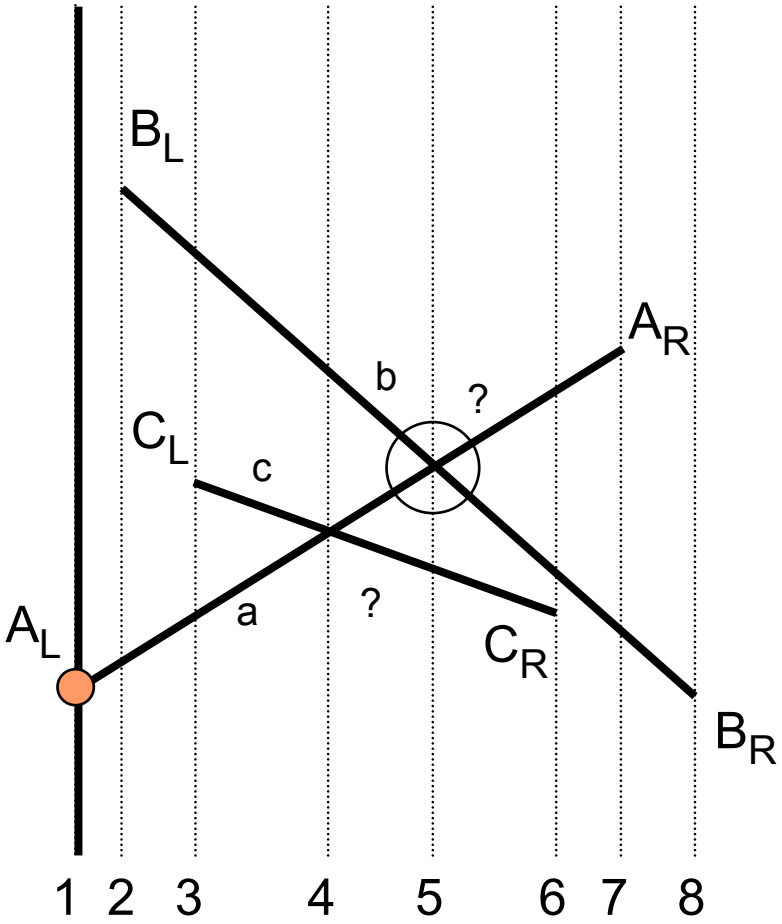
Průsečíky úseček

- L = Začíná nová úsečka s – oddělí bývalé sousedy
  - Najdi v T sousedy s (úsečky s_1 a s_2)

    (s je úsečka s počátečním bodem p)
  - if(protíná s_1 x s_2) odstraň průsečík

    (už nejsou sousedy)
  - vlož průs. s_1 x s a s x s_2 do B

Průsečíky úseček



B_0: ~~A_L~~,B_L, C_L,C_R,A_R,B_R

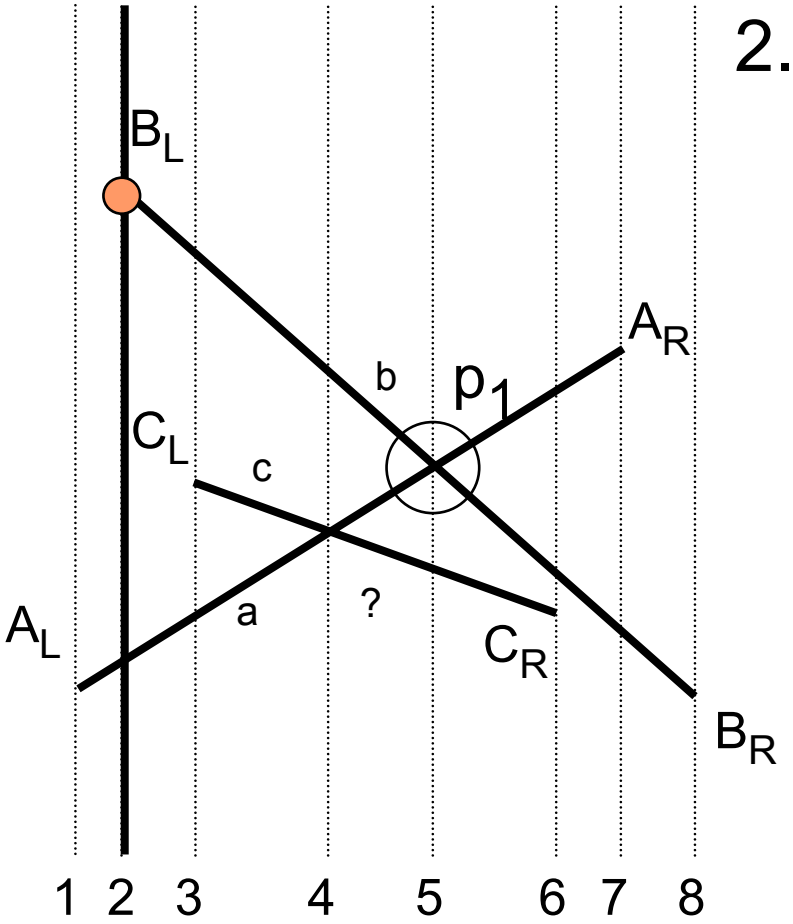
T_0: prázdná

1. A_L

B_1: ~~B_L~~, C_L,C_R,A_R,B_R

T_1: a

Průsečíky úseček



2. B_L

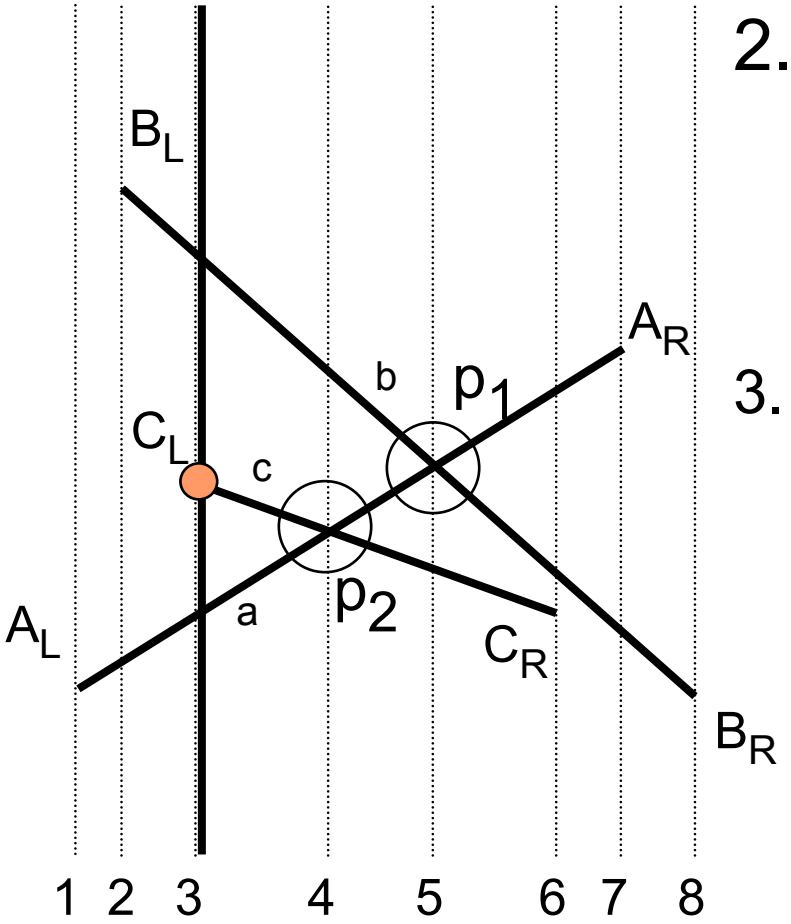
$p_1 = s_1 \times s = a \times b$ ... vložit

B_2: C_L,p_1, C_R, A_R,B_R

T_2: a, b

s_1 s

Průsečíky úseček



2. B_L

B_2: ~~C_L~~,~~p_1~~, C_R, A_R,B_R

T_2: a, b

3. C_L

$p_1 = s_1 \times s_2 = a \times b$...smazat

$p_2 = s_1 \times s = a \times c$ ... vložit

B_3: ~~p_2~~, C_R, A_R,B_R

T_3: a, c, b

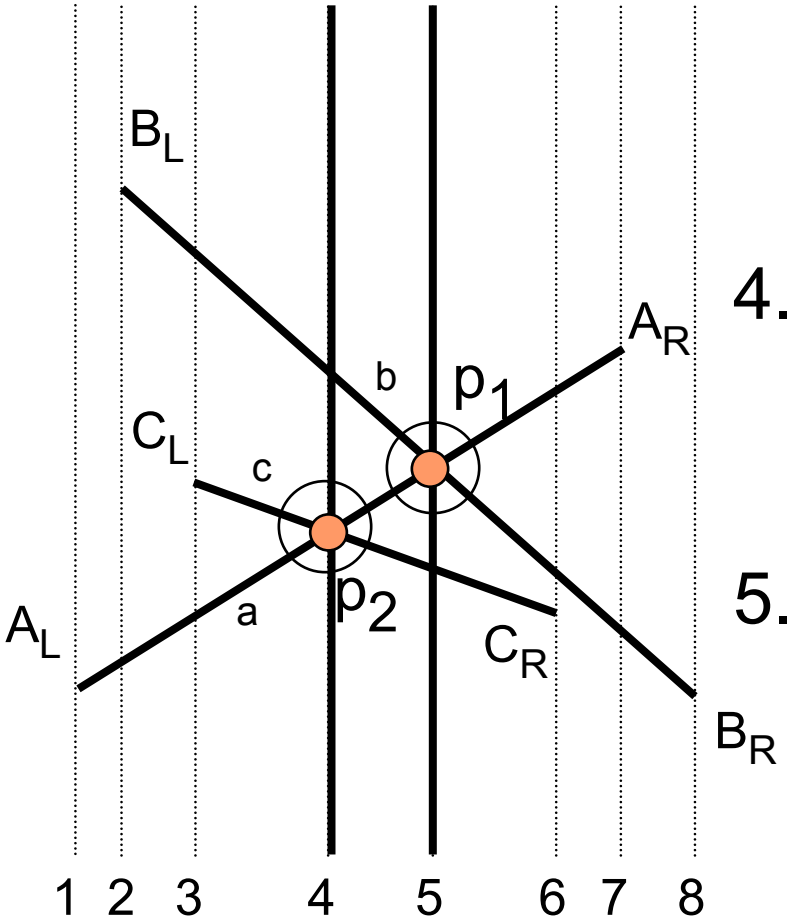
s_1 s s_2

Průsečíky úseček

- průsečík úseček *s* a *s'*
  - Najdi v *T* sousedy *s* a *s'*(úsečky s_1 a s_2)
  - prohoď *s* a *s'* v T
  - if(protíná *s* x s_1) odstraň průsečík z B
  - if(protíná *s'* x s_2) odstraň průsečík z B

    (už nejsou sousedy)
  - vlož průs. *s* x s_2 a *s'* x s_1 do *B*

Průsečíky úseček



B_3: ~~p_2~~, C_R, A_R,B_R

T_3: a, c, b

4. p_2 změna pořadí úseček a,c

B_4: ~~p_1~~, C_R, A_R,B_R

T_4: c, a, b

5. p_1 změna pořadí úseček a,b

B_5: C_R, A_R,B_R

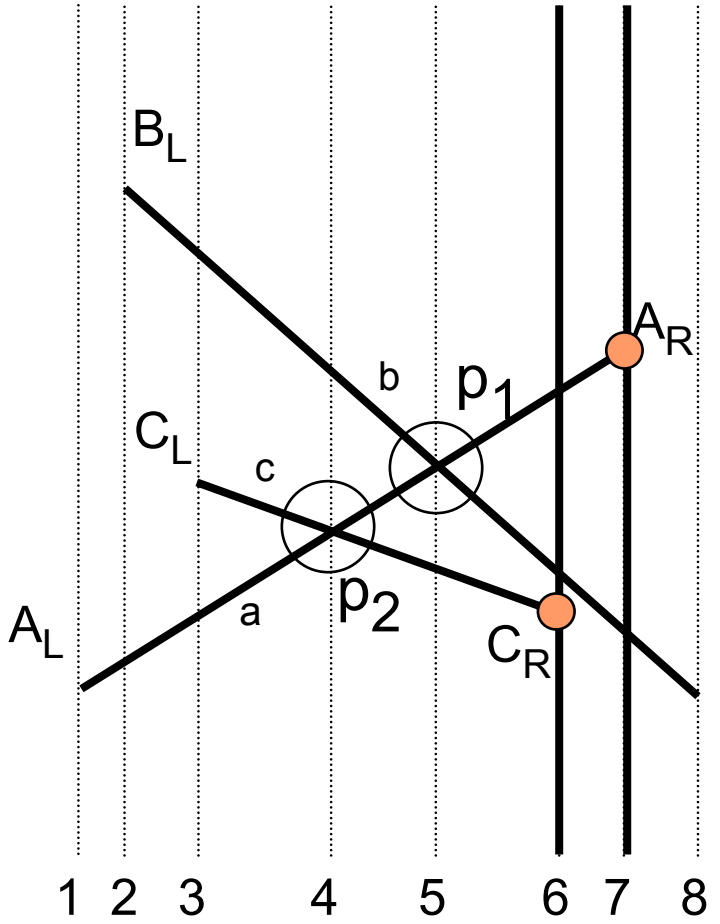
T_5: c, b, a

Průsečíky úseček

- R = končí úsečka s, její sousedi budou sousedy
  - Najdi v *T* sousedy *s* (úsečky s_1 a s_2)

    (*s* je úsečka s koncovým bodem *p*)
  - smaž *s* z *T*
  - if(protíná s_1 x s_2) vlož průsečík do *B*

Průsečíky úseček



B_5: ~~C_R~~, ~~A_R~~,~~B_R~~
T_5: ~~c~~, b, a

6. C_R

B_6: ~~A_R~~,~~B_R~~
T_6: b, ~~a~~

7. A_R

B_7: ~~B_R~~
T_7: ~~b~~

8. B_R

B_7: prázdná
T_7: prázdná

Průsečíky úseček

Paměť O(n)

Operační složitost

- n+k poloh
- každá log n

=> O(k+n) log n

Výpočetní geometrie (CG)

Úvod

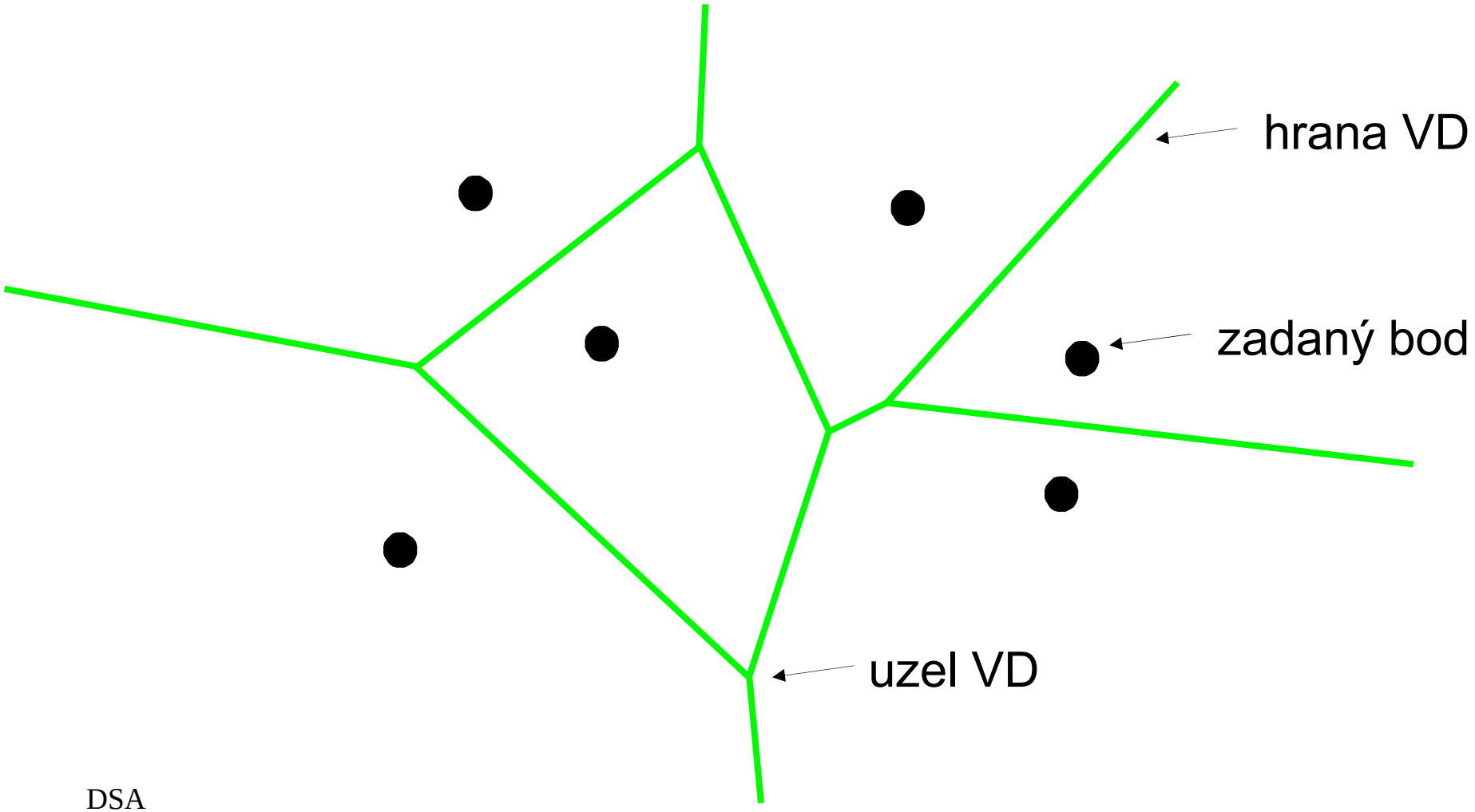
Příklady úloh

Algoritmické techniky – paradigmata

- řazení - jako předzpracování
- rozděl a panuj (*divide and conquer*)
- zametací technika (*scan-line*)
- geometrické místo (Locus approach)

Voroného diagram (VD)

Struktura pro vyhledání nejbližšího souseda

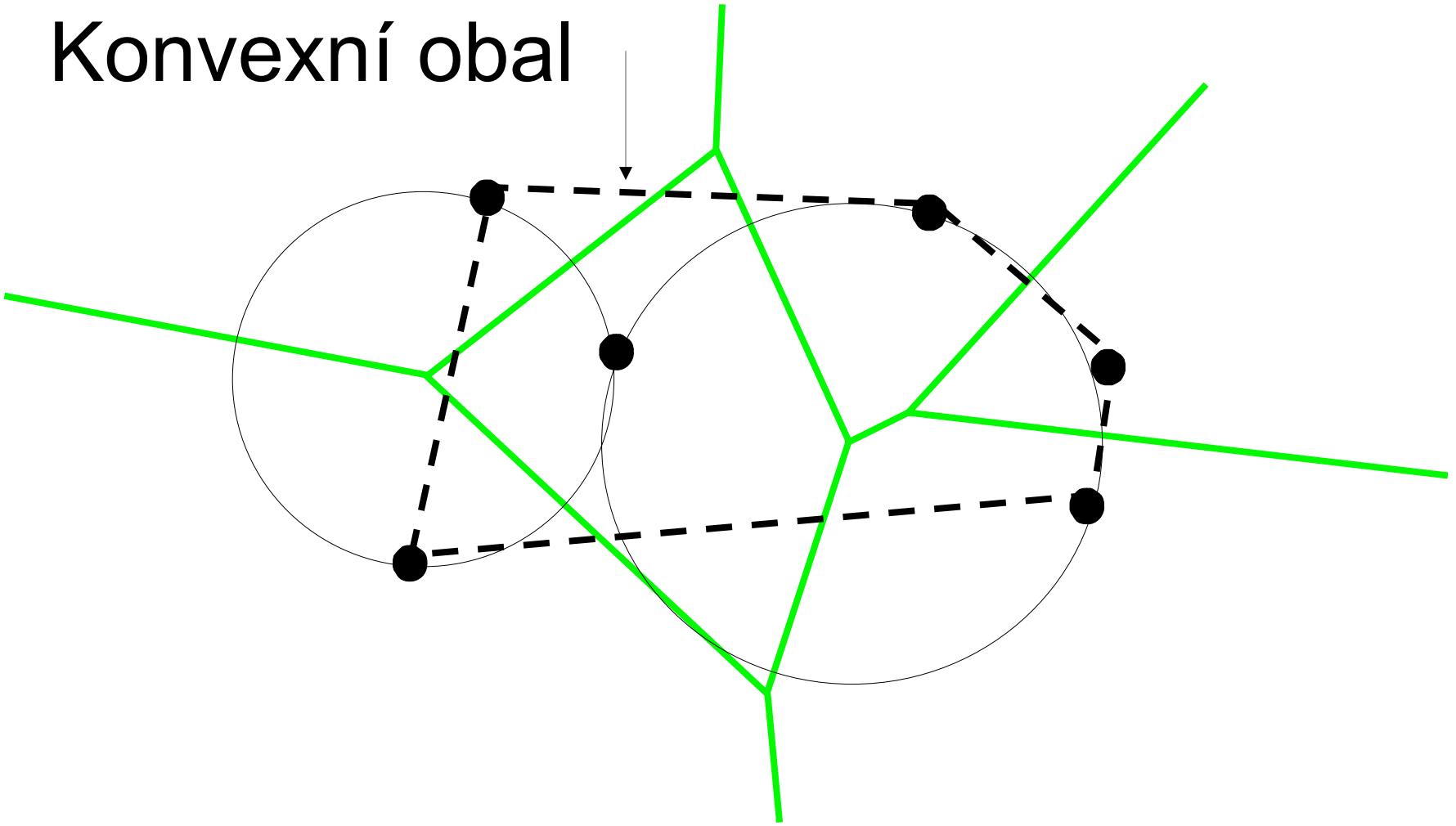


Voroného diagram

= planární graf (též Voronoiův diagram)

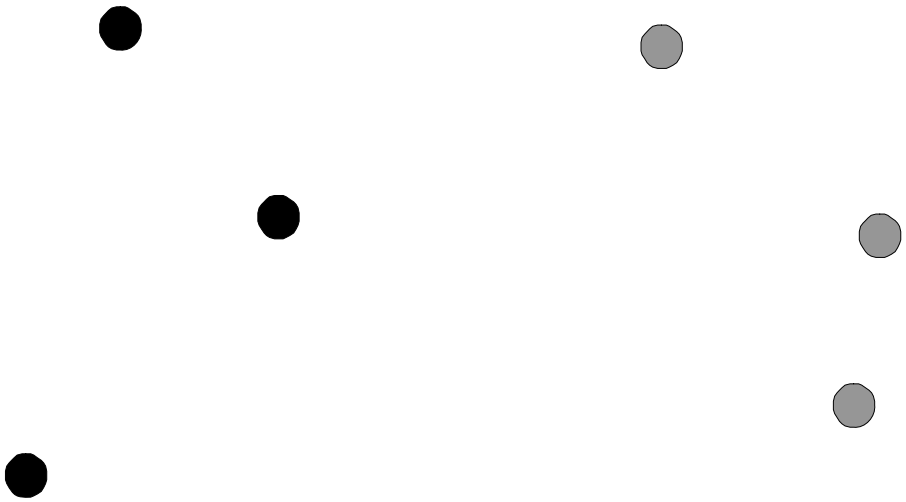
- obsahuje *n* oblastí (*n* = počet bodů - vstup)
- hrana = gmb. stejně vzdálených od dvou bodů
  = osa spojnice těchto dvou bodů
- uzel = střed kružnice opsané ≥ 3 bodům
- uzly mají stupeň ≥ 3
- počet uzlů 2*n*-4, počet hran *n*-6, tj. O(*n*)
- otevřené oblasti odpovídají bodům konvexní obálky

Voroného diagram



Voroného diagram

Konstrukce metodou rozděl a panuj

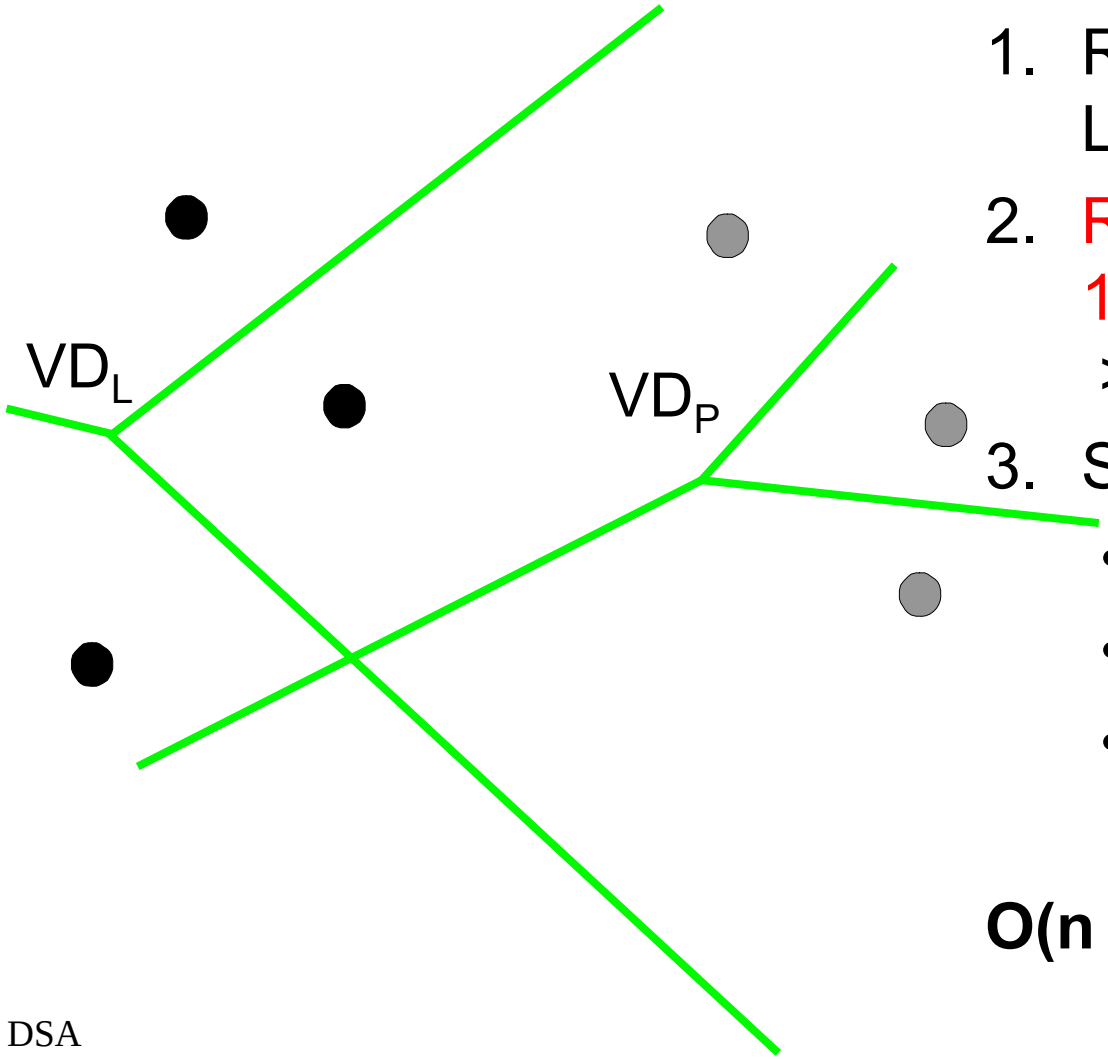


1. Rozděl body dle x-souř. na L a P
2. Rekurze na L a P
   1-3 body => návrat
   >3 body => rekurze
3. Spoj VD_L a VD_P
   - monotónní řetěz úseček
   - zkrať protnuté hrany
   - nové hrany z řetězu ús.

O(n log n)

Voroného diagram

Konstrukce metodou rozděl a panuj

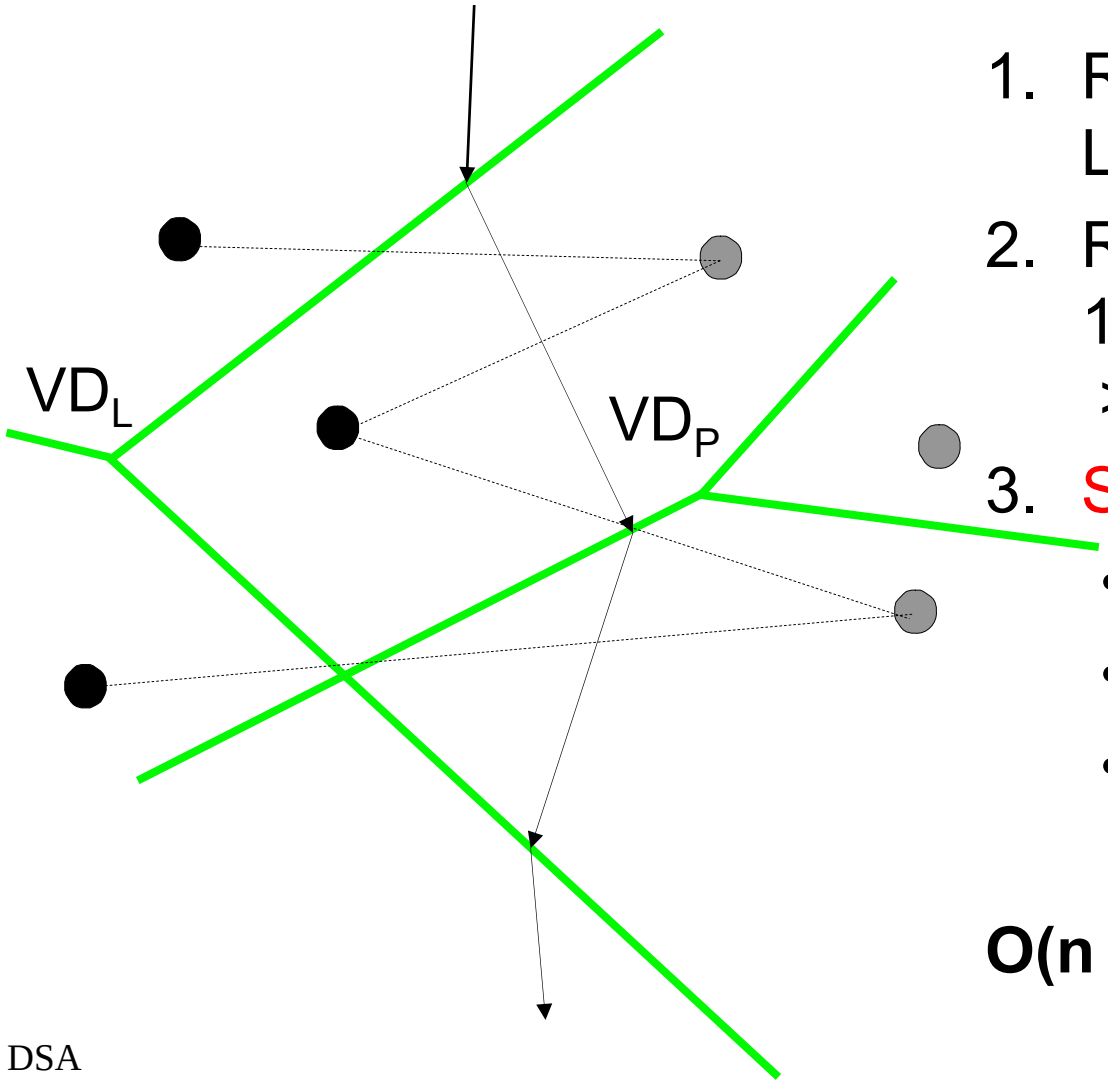


1. Rozděl body dle x-souř. na L a P
2. Rekurze na L a P
   1-3 body => návrat VD
   >3 body => rekurze
3. Spoj VD_L a VD_P
   - monotónní řetěz úseček
   - zkrať protnuté hrany
   - nové hrany z řetězu ús.

O(n log n)

Voroného diagram

Konstrukce metodou rozděl a panuj

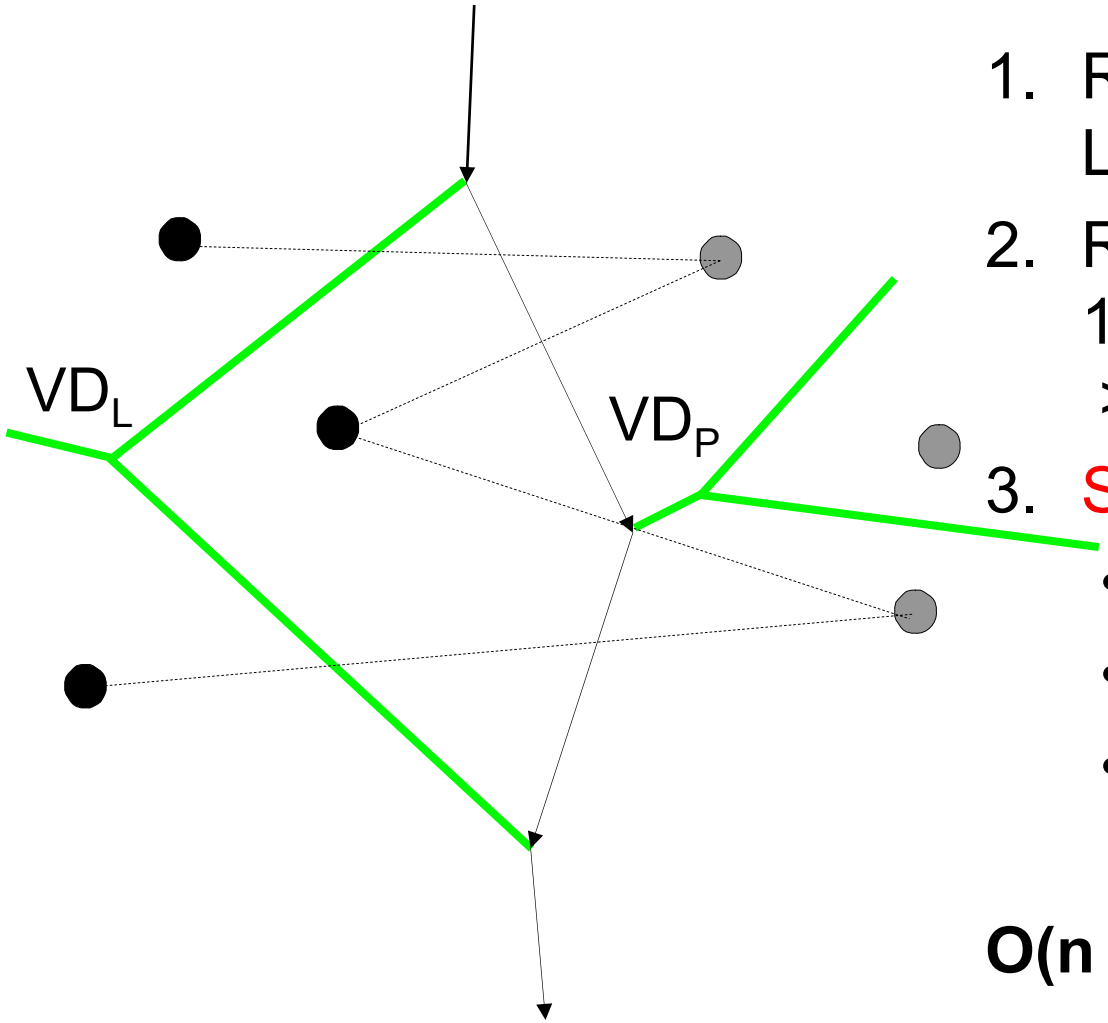


1. Rozděl body dle x-souř. na L a P
2. Rekurze na L a P
   1-3 body => návrat
   >3 body => rekurze
3. Spoj VD_L a VD_P
   - monotónní řetěz úseček
   - zkrať protnuté hrany
   - nové hrany z řetězu ús.

O(n log n)

Voroného diagram

Konstrukce metodou rozděl a panuj

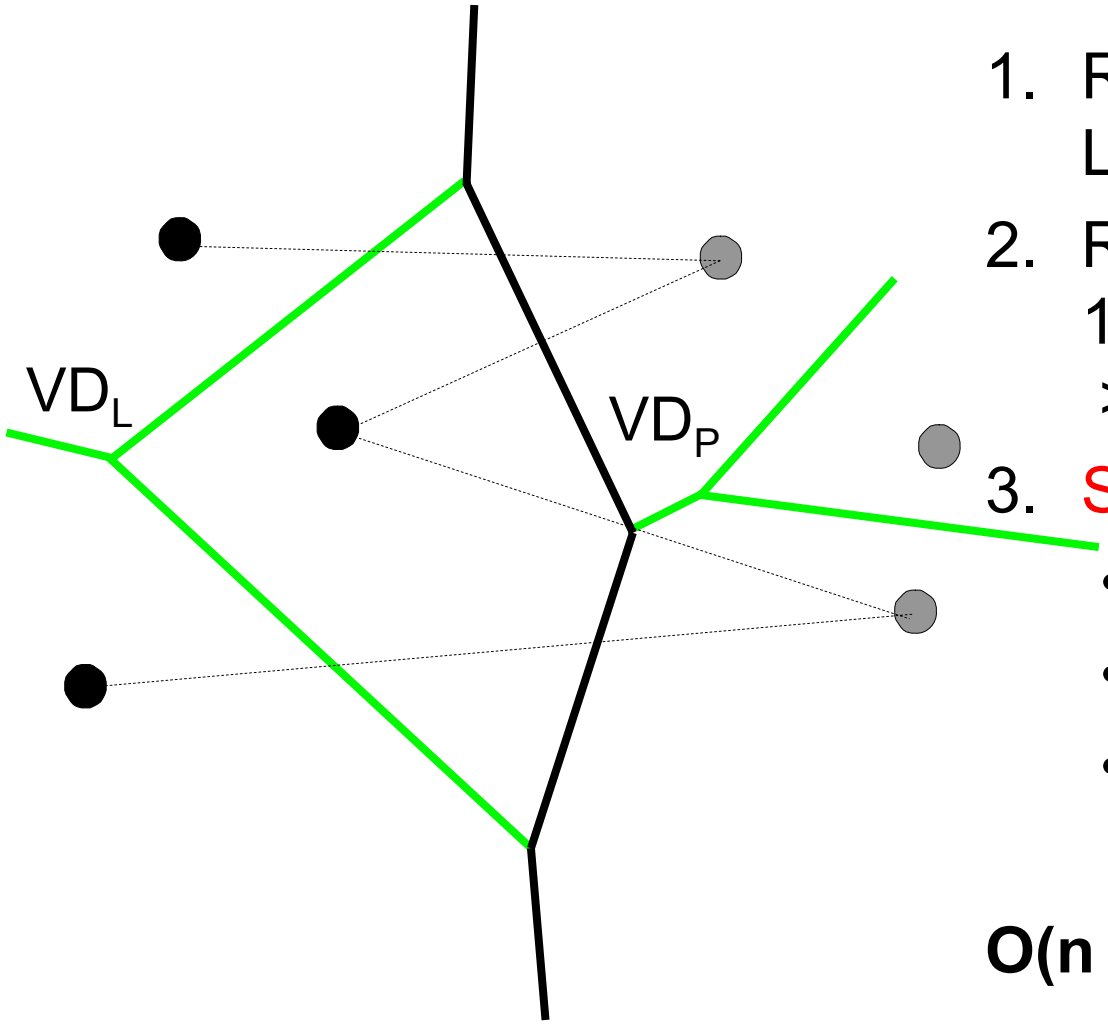


1. Rozděl body dle x-souř. na L a P
2. Rekurze na L a P
   1-3 body => návrat
   >3 body => rekurze
3. Spoj VD_L a VD_P
   - monotónní řetěz úseček
   - zkrať protnuté hrany
   - nové hrany z řetězu ús.

O(n log n)

Voroného diagram

Konstrukce metodou rozděl a panuj

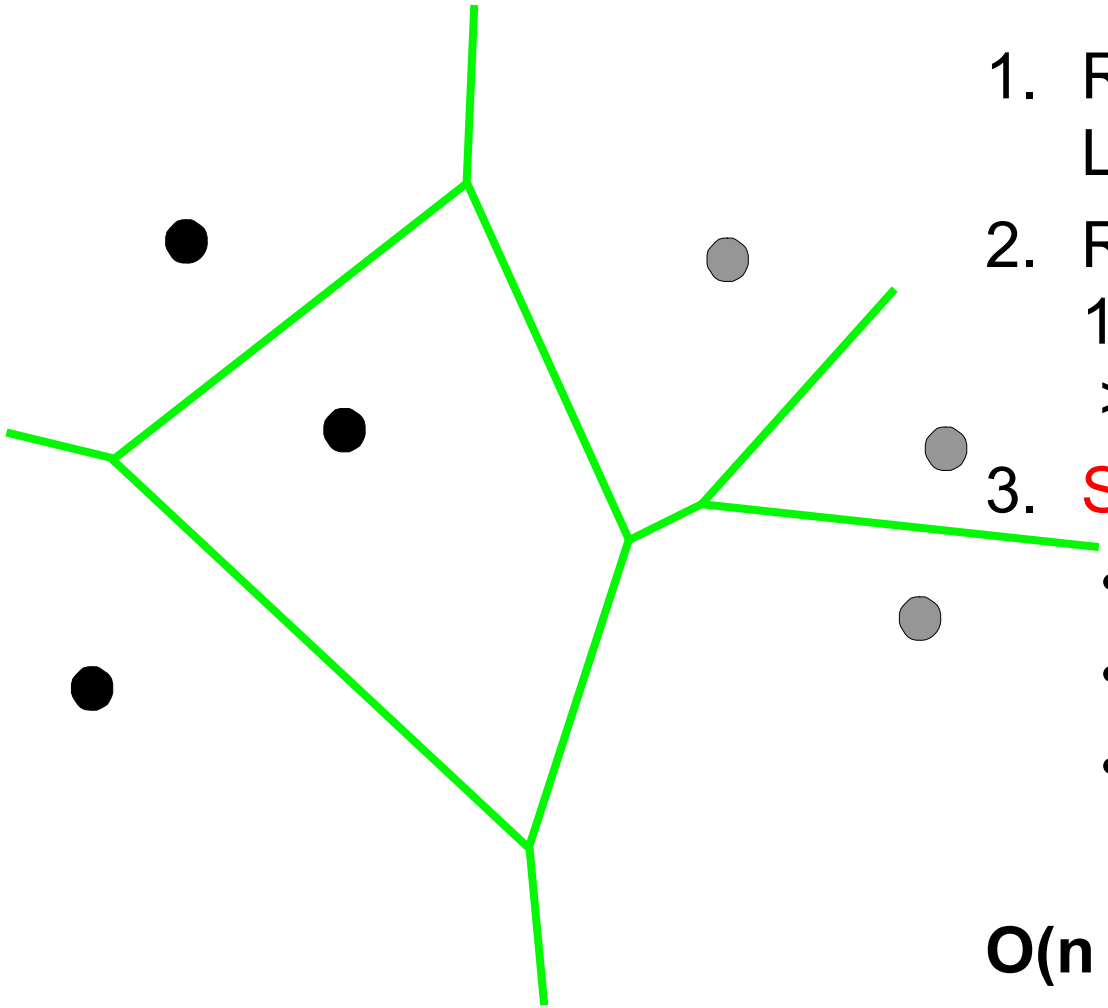


1. Rozděl body dle x-souř. na L a P
2. Rekurze na L a P
   1-3 body => návrat
   >3 body => rekurze
3. Spoj VD_L a VD_P
   - monotónní řetěz úseček
   - zkrať protnuté hrany
   - nové hrany z řetězu ús.

O(n log n)

Voroného diagram

Konstrukce metodou rozděl a panuj



1. Rozděl body dle x-souř. na L a P
2. Rekurze na L a P
   1-3 body => návrat
   >3 body => rekurze
3. Spoj VD_L a VD_P
   - monotónní řetěz úseček
   - zkrať protnuté hrany
   - nové hrany z řetězu ús.

O(n log n)

Links to web

Collections of geometry resources (Rozcestníky)

N. Amenta, Directory of Computational Geometry Software, http://www.geom.umn.edu/software/cglist/.

D. Eppstein, *Geometry in Action*, http://www.ics.uci.edu/~eppstein/geom.html.

Jeff Erickson, Computational Geometry Pages, http://compgeom.cs.uiuc.edu/~jeffe/compgeom/

References

Jan Slovák, Geometrické algoritmy I, skripta na webu, 1994, ftp://www.math.muni.cz/pub/math/people/Slovak/lectures/geometricke.algoritmy/galgi.ps

Rourke: Computational geometry in C, 2nd ed., Cambridge University Press, 1998, http://maven.smith.edu/~orourke/books/compgeom.html

… for more books, see the collections on the previous slide

KONEC

Mnoho štěstí na zkouškách !