Kapitola 1

Konvexní objekty, incidence, průniky

1.1 Konvexní objekty a VHR

Algoritmus 1.1 POINTINQ

Vstup: bod x a *VHR* mnohoúhelníka P

Výstup: zjištění, zda $x \in P$

1. $p =$ těžiště P_0; $i := 0$
2. `while` $P_i \neq P$ `do`
   2.1. zjistíme, ve které výseči z p na vrcholy P_i leží x
   2.2. $i := i + 1$ a v další iteraci rozvíjíme nalezenou výseč
3. určíme, zda je x uvnitř výsledné výseče

Algoritmus 1.2 LINEINTERSECTIONQ
Vstup: přímka p a *VHR* mnohoúhelníka P
Výstup: $p \cap P$

1. $i := 0$
2. najdeme $d(P_0)$ /* $d(P_0)$ je 'nejbližší bod' k p mezi vrcholy P_0 */
3. `if` $p \cap P_0 = \emptyset$ `then`
   - 3.1. `repeat`
     - 3.1.1. $i := i + 1$
     - 3.1.2. najdeme $d(P_i)$ s pomocí $d(P_{i-1})$
     - 3.1.3. zjistíme s pomocí $d(P_i)$, zda $p \cap P_i = \emptyset$
   - 3.2. `until` $p \cap P_i \neq \emptyset$ `or` $P_i = P$
4. `if` $p \cap P_i = \emptyset$ `then` /* $i = k$ */
   - 4.1. `return` $\emptyset$
5. `else`
   - 5.1. `while` $P_i \neq P$ `do`
     - 5.1.1. najdeme $e_1(i), e_2(i)$
     - 5.1.2. $i := i + 1$
   - 5.2. `return` $(p \cap e_1(k)) \cup (p \cap e_2(k))$

1.2 Konvexní obaly

Algoritmus 1.3 GIFT-WRAPPINGHULL2D (JARVISMARCH)

Vstup: jednoduchý mnohoúhelník P^n zadaný jako seznam bodů $p_0, \dots, p_n$ uspořádaný ve směru hodinových ručiček

Výstup: hraniční body horního konvexního obalu všech vrcholů mnohoúhelníka P^n ve směru hodinových ručiček

1. vybereme bod s nejmenší y-ovou souřadnicí za aktuální vrchol
2. `repeat`
   2.1. spojíme aktuální vrchol s ostatními body a za další vrchol BCH vybereme nejbližší z těch, které spojuje s výchozím nejmenší úhel počítaný mezi jejich spojnicí a poslední přidanou hranou (máme-li zatím pouze první bod, vezmeme místo této hrany osu x)
   2.2. aktuální vrchol := nově vybraný vrchol
3. `until` aktuální vrchol je opět ten úplně původní

Algoritmus 1.4 GRAHAMSCAN1

Vstup: jednoduchý mnohoúhelník P^n zadaný jako seznam bodů $p_0, \dots, p_n$ uspořádaný ve směru hodinových ručiček

Výstup: hraniční body horního konvexního obalu všech vrcholů mnohoúhelníka P^n ve směru hodinových ručiček

V algoritmu vystupuje zásobník $Q : q_0, \dots, q_t$, kde t je vždy vrchol tohoto zásobníku. Zásobník má dno q_0 přístupné ke čtení.

1. najdi body p_r a p_l
2. inicializuj zásobník: $q_0 := p_r$; $q_1 := p_l$
3. $s := 0$
4. `while` p_s je vrchol nalevo od $\overline{q_0 q_1}$ `do` /* přeskočíme body pod přímkou $\overline{p_r p_l}$ */
   4.1. $s := s + 1$
5. přidej p_s na vrchol zásobníku
6. `while` $s \leq n$ `do`
   6.1. `repeat` /* vyhledáme body, které jsou mimo mnohoúhelník daný body v zásobníku Q */
      6.1.1. $s := s + 1$
   6.2. `until` p_s je napravo od $\overline{q_0 q_t}$ `or` p_s je nalevo od $\overline{q_{t-1} q_t}$ `or` $s = n$
   6.3. `while` p_s je nalevo od $\overline{q_{t-1} q_t}$ `and` $s < n$ `do` /* přidáme bod p_s, ale tak, aby nová hrana neporušila konvexnost */
      6.3.1. odstraň q_t z vrcholu zásobníku
   6.4. přidej p_s na vrchol zásobníku

$q_0, \dots, q_t$ je horní konvexní obal

Algoritmus 1.5 GrahamScan2

Vstup: konečná množina S bodů v rovině
Výstup: vrcholy konvexního obalu S, setříděné ve směru hodinových ručiček

1. Vybereme libovolný vnitřní bod konvexního obalu (nemusí být prvkem S) a spojíme ho se všemi body S.

2. Setřídíme body podle velikosti úhlů jejich spojnic s vybraným bodem.

3. Posloupnost pošleme na vstup algoritmu GrahamScan1 pro horní i dolní konvexní obal a spojíme je.

Algoritmus 1.6 GRAHAMSCANHULL

Vstup: konečná množina S bodů v rovině
Výstup: vrcholy konvexního obalu S, setříděné ve směru hodinových ručiček

1. Setřídíme body lexikograficky

2. Takovou posloupnost pošleme na vstup algoritmu GRAHAMSCAN1 pro horní i dolní konvexní obal a spojíme je.

Algoritmus 1.7 MergeHull2D

Vstup: konečná množina S bodů v rovině

Výstup: vrcholy konvexního obalu S, setříděné ve směru hodinových ručiček

1. `if` $|S| \leq 3$ spočítáme konvexní obal přímo

2. Rozdělíme body na dvě části přibližně stejné mohutnosti a zavoláme rekurzivně MergeConvexHull

3. Výstupem předchozího kroku jsou dva konvexní mnohoúhelníky M_1, M_2 (které se mohou překrývat). Vybereme libovolný vnitřní bod p v M_1. (Ten bude jistě uvnitř jejich konvexního obalu.)

4. `if` p zároveň uvnitř M_2

   4.1. `then` zatřídíme vrcholy obou mnohoúhelníků M_1 a M_2 do monotónními posloupnostmi vzhledem k úhlům, které svírají jejich spojnice s bodem p.

   4.2. `else` /* p není vnitřní bod mnohoúhelníka M_2 */

      4.2.1. spustíme z bodu p tečny na M_2 a vytvoříme konvexní obal $M_2' = M_2 \cup \{p\}$.

      4.2.2. zatřídíme vrcholy obou mnohoúhelníků M_1 a M_2' do monotónními posloupnostmi vzhledem k úhlům, které svírají jejich spojnice s bodem p.

5. Sestrojenou monotónní posloupnost vrcholů pošleme na vstup algoritmu GrahamScan1.

Algoritmus 1.8 IncrementalHull2D

Vstup: konečná množina S bodů v rovině
Výstup: vrcholy konvexního obalu S, setříděné ve směru hodinových ručiček

1. $CH := p_1$
2. `for` $p \in S$ `do`
    2.1. `if` $p \notin CH$ najdi horní a dolní tečnu z p k CH
    2.2. $CH :=$ setříděné vrcholy konvexního obalu $CH \cup \{p\}$
3. `return` CH

Algoritmus 1.9 GIFT-WRAPPINGHULL3D

1. Najdeme nejprve bod s nejmenší z-ovou souřadnicí a jednu hranu konvexního obalu. Tuto hranu můžeme získat např. tak, že vstupní množinu promítneme do roviny určené osami xz a tam metodou GIFT-WRAPPINGHULL2D najdeme tuto jednu hranu. Zvolíme tuto hranu za aktuální.

2. `repeat` Kolem aktuální hrany natáčíme rovinu do všech bodů vstupní množiny a vybereme ten bod, pro nějž odpovídající rovina dělí prostor na dva poloprostory, z nichž jeden obsahuje celou vstupní množinu bodů. Takový bod se dá opět najít určením minimálního úhlu, který svírají všechny takové roviny s rovinou stěny BCH, jejíž je aktuální hrana částí (opět, máme-li zatím k dispozici pouze inicializační hranu z prvního kroku, vezmeme rovinu xy).

3. `until` Mnohostěn je uzavřen, tedy neexistují v něm „neuzavřené“ hrany (jsou to hrany, které zatím incidují pouze s jednou stěnou). Najdeme-li takovou hranu, vezmeme ji za aktuální a pokračujeme v cyklu.

Algoritmus 1.10 INCREMENTALHULL3D

Vstup: konečná množina S bodů v prostoru

Výstup: dvojitý spojitý seznam $CH(S)$ vrcholů, hran a stěn konvexního obalu množiny S

1. Najdi 4 body p_1, p_2, p_3, p_4 v S tvořící čtyřstěn
2. $C := CH(p_1, p_2, p_3, p_4)$
3. Spočti náhodné pořadí bodů $p_5, \ldots$
4. Inicializuj graf konfliktů $\mathcal{G}$ pro stěny C a body $p_5, \ldots$.
5. `for` $r := 5$ `to` $|S|$ `do`

   5.1. `if` $F_c(p_r) \neq \emptyset$ `then`

   5.1.1. zruš všechny stěny v $F_c(p_r)$ z C

   5.1.2. sestroj seznam L všech hran horizontu viditelnosti z p_r

   5.1.3. `for` $e \in L$ `do`

   5.1.3.1. vytvoř stěnu f obsahující e a p_r

   5.1.3.2. `if` f koplanární s sousední stěnou f' podél e

   5.1.3.3. `then`
   spoj f a f' do jediné stěny f
   $P_c(f) := P_c f'$

   5.1.3.4. `else`
   vytvoř uzel f v G
   $P(e) := P_c(f_1) \cup P_c(f_2)$ kde f_1 a f_2 jsou hrany sousedící s e
   `for` $p \in P(e)$ `do` je-li f viditelná z p, přidej hranu (p, f) do G

   5.1.3.5. zruš uzel p_r a všechny uzly $f \in F_c(p_r)$ z G, spolu s přilehlými hranami

6. `return` C

Algoritmus 1.11 MERGEHULL3D

Vstup: konečná množina S bodů v prostoru

Výstup: dvojitý spojitý seznam $CH(S)$ vrcholů, hran a stěn konvexního obalu množiny S

1. `if` $|S| \leq 4$ spočítáme konvexní obal přímo

2. Lexikograficky utřídíme body, rozdělíme je na dvě části přibližně stejné mohutnosti a zavoláme rekurzivně MERGEHULL3D

3. Výstupem předchozího kroku jsou dva konvexní obaly M_1, M_2 (které jsou disjunktní). "Rafinovanou" verzí Gift-wrapping obalíme M_1 a M_2 do společného konvexního obalu

2D algoritmy	nejhorší případ	očekávaný při distribuci		
		v r-úhelníku	v kruhu	v rovině
počet hran	$O(n)$	$O(r \log n)$	$O(n^{1/3})$	$O((\log n)^{1/2})$
Gift-wrap2D	$O(n^2)$	$O(n \log n)$	$O(n^{4/3})$	$O(n\sqrt{\log n})$
MergeHull	$O(n \log n)$	$O(n)$	$O(n)$	$O(n)$
ostatní 2D	$O(n \log n)$			
3D algoritmy	nejhorší případ	distribuce bodů		
			v kouli	v prostoru
počet stěn	$O(n)$		$O(n^{1/2})$	$O(\log n)$
Gift-wrap3D	$O(n^2)$		$O(n^{3/2})$	$O(n \log n)$
Incremental3D	$O(n^2)$	$O(n \log n)$		
MergeHull3D	$O(n \log n)$			

Časové složitosti algoritmů, nejhorší případ a očekávané časy (normální distribuce v $\mathbb{R}^2$, $\mathbb{R}^3$, resp. rovnoměrné rozdělení)

1.3 Odhady pro náhodnostní algoritmy

Konfigurační prostor je čtveřice (X, Π, D, K) s následující specifikací:

X: Vstupní množina objektů pro zpracování geometrickým algoritmem. (V případě algoritmu 1.10 jsou to body p_i v prostoru $\mathbb{R}^3$.)

Π: Množina konfigurací, tj. objektů, které jsou během popisovaného geometrického algoritmu konstruovány. (V 1.10 to jsou jisté modifikace "okřídlených hran" vytvářených při běhu algoritmu.)

D: Přiřazení *definiční množiny* $D(\Delta)$ každé konfiguraci $\Delta \in \Pi$. Vždy $D(\Delta) \subset X$ a existuje číslo d ohraničující shora velikost všech $D(\Delta)$. (V algoritmu 1.10 to jsou koncové vrcholy tří hran definující "křídlo", $d = 4$.)

K: Přiřazení *množiny konfliktů* $K(\Delta)$ každé konfiguraci $\Delta \in \Pi$. Vždy $K(\Delta) \subset X$ a $K(\Delta) \cap D(\Delta) = \emptyset$. (V případě 1.10 je $K(\Delta)$ množina všech bodů v X, které "vidí" křídlo Δ)

Pro každou podmnožinu $S \subset X$ definujeme

$$\mathcal{T}(S) = \{\Delta \in \Pi; D(\Delta) \subset S, K(\Delta) \cap S = \emptyset\}.$$

Cílem algoritmu je spočíst $\mathcal{T}(X)$. Pro odhad potřebného času a paměti pracujeme s množinami $X_r = \{p_1, \ldots, p_r\}$.

1.1 Lemma. *Očekávaný počet konfigurací v $\mathcal{T}(X_r) \setminus \mathcal{T}(X_{r-1})$ je nejvýše*

$$\frac{d}{r} E(|\mathcal{T}(X_r)|).$$

1.2 Lemma. *Očekávaný počet konfliktů vytvořených během celého běhu algoritmu je nejvýše*

$$\sum_{r=1}^{n} \frac{d^2}{r^2}(n-r)E(|\mathcal{T}(X_r)|).$$

1.4 Průniky konvexních útvarů

Algoritmus 1.12 ConvexPolyIntersection

Vstup: dva konvexní polygony P, Q (nemusí být uzavřené), zadané posloupnostmi sousedních bodů $p_1, \ldots, p_m$ a $q_1, \ldots, q_n$

Výstup: konvexní polygon ohraničující $P \cap Q$

1. Získáme bod p uvnitř P (např. těžiště libovolných tří nekolineárních vrcholů P).

2. `for` $i = 1 \ldots n$ vytvoříme výseče s_i ohraničené polopřímkami $\overline{pp_i}$ a $\overline{pp_{i+1}}$ $(p_{n+1} = p_1)$.

3. Najdeme výseč s_i, ve které se nachází bod q_1.

4. `for` $i = 2$ `to` n určíme průnik úsečky $L(q_{i-1}, q_i)$ s P, případně její polohu (uvnitř – vně). (V čase $O(k+1)$ diskutujeme polohu $L(q_{i-1}, q_i)$, kde k je počet polopřímek $\overline{pp_j}$ proťatých úsečkou $L(q_{i-1}, q_i)$.)

5. Výstupem předchozího kroku jsou buď průniky hran, nebo informace, zda $Q \subset P$ nebo $P \subset Q$ nebo $P \cap Q = \emptyset$.

Protože každá hraniční polopřímka oblasti s_i protíná nejvýše dvě hrany, potřebujeme nejvýše čas $O(2m)$ pro nalezení průsečíků. Zároveň však musíme projít všechny body Q^n, proto výsledný čas je $O(m+n)$

Algoritmus 1.13 HALFPLANEINTERSECTION
Vstup: poloroviny $H_1, \dots, H_n$
Výstup: jejich průnik $\bigcap_{i=1}^{n} H_i$

1. `if` $n \leq 2$ `then`
   1.1. Spočítej průnik nejvýše dvou polorovin a pošli ho na výstup
2. `else`
   2.1. Rozděl $H_1, \dots, H_n$ na dva přibližně stejně velké díly a zavolej rekurzivně sama sebe pro každý z nich
   2.2. Takto získané částečné výsledky jsou konvexními mnohoúhelníkovými oblastmi (na rozdíl od konvexních mnohoúhelníků nemusí být ohraničené). Na takové může být aplikován algoritmus CONVEXPOLYYINTERSECTION

Algoritmus 1.14 SegmentIntersection

Vstup: množina úseček $L_1, \dots, L_n$ v rovině

Výstup: množina jejich průsečíků

1. v–struktura := $\emptyset$
2. h–struktura := všechny koncové body úseček setříděné podle x-ové souřadnice
3. `while` h–struktura $\neq \emptyset$ `do`
   - 3.1. p := bod v h–struktuře s minimální x-ovou souřadnicí
   - 3.2. odstraň p z h–struktury
   - 3.3. `if` p je levý koncový bod úsečky `then` /* *nová aktivní* */
     - 3.3.1. j := číslo úsečky, jejíž je p koncový bod (L_j)
     - 3.3.2. vlož L_j do v–struktury
     - 3.3.3. vyhledej čísla sousedů (i, k) úsečky L_j ve v–struktuře
     - 3.3.4. vlož $L_i \cap L_j$ a $L_j \cap L_k$ do h–struktury, pokud existují
     - 3.3.5. odstraň $L_i \cap L_k$ z h–struktury, pokud tam je
   - 3.4. `elsif` p je pravý koncový bod úsečky `then`
     - 3.4.1. j := číslo úsečky, jejíž je p koncový bod (L_j)
     - 3.4.2. vyhledej čísla sousedů (i, k) úsečky L_j ve v–struktuře
     - 3.4.3. odstraň L_j z v–struktury
     - 3.4.4. vlož $L_i \cap L_k$ do h–struktury, je-li napravo od pročesávací přímky
   - 3.5. `else` /* *p musí být průsečík přímek* */
     - 3.5.1. (i, j) := indexy úseček, jejichž je p průsečík
     - 3.5.2. zaměň L_i a L_j ve v–struktuře
     - 3.5.3. vyhledej čísla (h, k) dalších sousedů L_i a L_j ve v–struktuře
     - 3.5.4. vlož $L_h \cap L_i$ a $L_j \cap L_k$ do h–struktury, je-li L_h nyní soused L_i (resp. L_k soused L_j), a jsou-li tyto průniky napravo od pročesávací přímky
     - 3.5.5. odstraň $L_h \cap L_j$ a $L_i \cap L_k$ z h–struktury
     - 3.5.6. pošli (p, i, j) na výstup

1.5 Překrytí rovinných rozdělení

Algoritmus 1.15 MapOverlay

Vstup: Dvě rovinná rozdělení S_1 a S_2 rerezentovaná dvojitým spojovým seznamem hran.

Výstup: Rovinné rozdělení D vzniklé překryvem S_1 a S_2 reprezentované dvojitým spojovým seznamem hran.

1. Zkopíruj oba dvojité spojové seznamy S_1 a S_2 do (zatím nekorektního) seznamu D.

2. Aplikuj SegmentIntersection na hrany v $S_1 \cup S_2$ s následujícími modifikacemi (pročesávání obrazně provádíme zhora dolů):

   - Při detekci průniku dvou úseček z téhož S_i (úsečky uvažujeme uzavřené) neprovádíme žádnou akci na D.
   - Při detekci průniku úsečky z S_1 a S_2 zavedeme nový vrchol (pokud náhodou nejde o průnik ve vrcholech), zatřídíme nově vzniklé hrany a původní hrany z nového vrcholu, upravíme seznamy dvojčat hran.
   - Přiřadíme ke každému procházenému vrcholu v při pročesávání první hranu e(v) nalevo od něj.
   - Pro každý vrchol v v S_1 určíme stěnu $f_2(v)$ v S_2 v níž leží a naopak

   /* Nyní je k dispozici korektně upravená část D: vše co se přímo týká hran a vrcholů, nejsou však korektně zadány odkazy mezi stěnami a hranami */

3. Vytvoř uzly grafu G hraničních cyklů (reprezentované hranami končícími v nejlevějším nejspodnějším vrcholu), včetně formálního nekonečného cyklu, a pro každý zjisti, zda se jedná o vnější nebo vnitřní hranici.

4. Vytvoř hrany hranu grafu G, které spojují nejlevější vrchol v každého vnitřního cyklu (je v reprezentaci uzlu G) s nejbližší půlhranou $e(v)$ vlevo od něj.

5. `for` každou souvislou komponentu G (tj. pro každý uzel w popisující vnější cyklus) `do`

5.1. Vytvoř záznam f pro stěnu s danou vnější hranicí a nastav jeho ukazatel na vnější hranici na některou z hraničních půhran (třeba tu z reprezantace zpracovávaného uzlu G).

5.2. Pro všechny ostatní uzly v zpracovávané souvislé komponentě G nastav seznam ukazatelů na vybrané půlhrany z vnitřních hraničních cyklů.

5.3. Nastav ukazatele všech hran z hraničních cyklů na incidující stěnu na f

6. Označ každou ze stěn dvojicí jmen stěn v S_1 a S_2 v nichž leží (máme k dispozici $f_1(v)$ nebo $f_2(v)$ nebo jde o nově přidaný vrchol).

Kapitola 2

Voroného diagramy, triangulace, rovinná rozdělení

2.1 Voroného diagramy

Algoritmus 2.1 VoronoiDiagram

Vstup : konečná množina bodů $S \subset \mathbb{R}^2$
Výstup : Voroného diagram množiny S reprezentovaný dvojitým spojovým seznamem hran

1. `if` $|S| \leq 3$ `then return` Voroného diagram S

2. `else`
   Lexikograficky setřiď body v S
   rozděl na půlky S_L aS_R
   $VDL :=$ VoronoiDiagram(S_L); $VDR :=$ VoronoiDiagram(S_R)

3. /* sešívání */

   3.1. $h := 1$
   $T_L \ldots$ dolní tečna $BCH(S_L), BCH(S_R)$ spojující $x \in S_L$, $y \in S_R$
   $L \ldots$ polopřímka, která je osou $L(x, y)$
   $l_1 := L$

   3.2. `while` L protíná VDL nebo VDR

3.2.1. $VD := VDL \cup VDR$ /* formálně spojíme oba DCEL a budeme dále upravovat */

3.2.2. `if` L protne nejdříve $e \in VDL$ `then`

3.2.2.1. $z :=$ bod průniku

3.2.2.2. l_h ukonči v z

3.2.2.3. $x' :=$ bod S_L tak, že $e \in VR(x), e \in VR(x')$

3.2.2.4. $x := x'; h := h + 1$

3.2.2.5. $L :=$ polopřímka začínající v z, osa $L(x, y)$

3.2.2.6. $l_h := L$

3.2.3. `else` SYMETRICKY PRO S_R

3.2.4. uprav lokálně strukturu DCEL pro provedené změny

4. `return` VD

Algoritmus 2.2 BEACHLINEVD

Vstup : konečná množina bodů $S \subset \mathbb{R}^2$

Výstup : Voroného diagram množiny S, vložený do obdélníkového boxu, reprezentovaný dvojitým spojovým seznamem hran

1. Inicializuj strukturu událostí $\mathcal{Q}$ se všemi řídícími body
   /* $\mathcal{Q}$ je prioritní fronta událostí lexikograficky sestupně setříděná podle y a x; kromě řídících bodů ještě obsahuje tzv. kružnicové události, u kterých si zavádíme i ukazatel na list v $\mathcal{T}$, který má při této události zmizet */

2. Inicializuj binární vyhledávací strom $\mathcal{T}$ reprezentující 'surfovou vlnu' pročesávající přímky pro první událost a tuto odstraň z $\mathcal{Q}$
   /* $\mathcal{T}$ má listy reprezentující zleva doprava uspořádáné řídící body, které definují příslušné části parabolických oblouků; vnitřní uzly $\mathcal{T}$ pak popisují body zlomů surfové vlny v podobě dvojic příslušných definujících řídících bodů; navíc $\mathcal{T}$ uchovává pro každý list ukazatel na (kružnicovou) událost v $\mathcal{Q}$, při které má zmizet a pro každý vnitřní uzel ukazatel na půlhranu ve vytvářeném Voroného diagramu, která tento bod zlomu obsahuje */

3. `while` $\mathcal{Q}$ neprázdné `do`

   3.1. `if` první událost v $\mathcal{Q}$ je řídící bod $p_i \in S$

   3.2. `then` HANDLESITEEVENT(p_i)

   3.3. `else` HANDLECIRCLEEVENT(p_ℓ)
   /* p_ℓ je nejnižší bod příslušné kružnice */

   3.4. odstraň událost z $\mathcal{Q}$

   /* V Q už nejsou další události, pročesávací surfová vlna ovšem ještě popisuje dosud nezapracované neohraničené hrany diagramu */

4. Vytvoř hraniční box pro výstup a dokonči popis vrcholů a cyklů hraničních hran Voroného diagramu

5. Vytvoř seznam stěn a příslušné ukazatele do a z seznamu hran.

Algoritmus 2.3 HANDLESITEEVENT(p_i)

1. Najdi v $\mathcal{T}$ oblouk α nad novým řídícím bodem p_i
2. Zruš v $\mathcal{Q}$ všechny kružnicové události obsahující α
3. Nahraď list v $\mathcal{T}$, který reprezentoval α novým podstromem se třemi listy: střední představuje nový řídící bod p_i, dva sousedé p_j pak jsou tvořeny tím bodem, který vytvářel oblouk α. Nové vnitřní uzly (p_j, p_i) a (p_i, p_j) reprezentují nové zlomy na surfové vlně
4. Vyvaž znovu $\mathcal{T}$
5. Vytvoř nové záznamy v DCEL Voroného diagramu pro hrany oddělující $VD(p_i)$ a $VD(p_j)$
6. Prověř všechny nově vzniklé trojice po sobě jdoucích oblouků a zařaď příslušné kružnicové události do $\mathcal{T}$, pokud kružnice protíná pročesávací přímku a zároveň v $\mathcal{Q}$ ještě není.

Algoritmus 2.4 HANDLECIRCLEEVENT(p_ℓ)

1. Najdi v $\mathcal{T}$ oblouk α nad p_ℓ (který má zmizet)
2. Zruš v $\mathcal{Q}$ všechny kružnicové události obsahující α
3. Zruš v $\mathcal{T}$ list reprezentující α a obnov interní uzly, tj. dvojice popisující zlomy surfové vlny
4. Vyvaž $\mathcal{T}$
5. Vytvoř záznam pro vrchol Voroného diagramu zadaný středem kružnice a záznamy pro nové půlhrany vycházející z tohoto vrcholu. Nastav také příslušné ukazatele v DCEL, které jsou známy.
6. Prověř všechny nově vzniklé trojice po sobě jdoucích oblouků a zařaď příslušné kružnicové události do $\mathcal{T}$, pokud kružnice protíná pročesávací přímku a zároveň v $\mathcal{Q}$ ještě není.

Algoritmus 2.5 AddPointtoVD

Vstup: Množina S, její Voroného diagram $VD(S)$, bod $y \notin S$, který má být přidán do diagramu, a bod $x \in S$ takový, že $y \in VR(x)$

Výstup: Vrátí $VD(S \cup \{y\})$

1. $p := x$
2. `repeat`
   - 2.1. $\{a, b\}$:= průnik osy úsečky $L(p, y)$ s hranicí Voroného oblasti bodu p (a a b mohou být identické)
   - 2.2. Najdi bod z takový, který je sousedem bodu p ve $VD(S)$ a na jehož hranici leží některý z bodů a, b, navíc z jsme ještě v tomto algoritmu nenavštívili (takový bod nemusí existovat, nebo mohou být dva a musíme jeden vybrat)
   - 2.3. $p := z$
3. `until` Vrátíme se zpátky do výchozího bodu x (tedy $p = x$), nebo nelze najít bod z v bodu 2.2
4. `if` Do bodu x jsme se v předchozím cyklu nevrátili `then`
   - 4.1. Zopakuj předchozí postup z bodu x opačným směrem

/* Aktualizace proběhne v čase $O(s)$, kde s je počet sousedů bodu y. I s vyhledáním výchozího bodu x tedy lze dosáhnout času $O(s + \log n)$. */

Algoritmus 2.6 RemovePointfromVD

Vstup: Množina S, její Voroného diagram $VD(S)$, bod $y \in S$, který má být odebrán z diagramu.

Výstup: Vrátí $VD(S \setminus \{y\})$

1. Najdeme množinu $S_y \subset S$ všech sousedů bodu s
2. Vytvoříme $VD(S_y)$
3. Nahradíme oblast bodu y diagramem $VD(S_y)$

/* Aktualizace proběhne v čase $O(s \log s)$, kde s je počet sousedů bodu y. */

Algoritmus 2.7 kTHORDERVORONOIDIAGRAM

Vstup: Množina bodů v rovině S a požadovaný řád diagramu $k < |S|$
Výstup: $VD_k(S)$

1. Setrojíme $VD_1(S)$ a všechny jeho vrcholy hranic označíme jako vrcholy typu I.

2. `for` $j = 1$ `to` $k - 1$ `do`

   2.1. Původní vrcholy typu I zůstávají a označíme je jako vrcholy typu II. Původní vrcholy typu II zmizí. Ten, který odpovídal volbě $T_1' = R \cup \{y, z\}, T_2' = R \cup \{x, z\}, T_3' = R \cup \{x, y\}$, se stane vnitřním bodem oblasti odpovídající množině $R \cup \{x, y, z\}$.

   2.2. Původní oblasti $VR_j(T)$ potřebujeme rozdělit na části příslušející oblastem $VR_{j+1}(T \cup \{x\})$, kde $x \in S \setminus T$. Toto rozdělení provedeme konstrukcí $VD_1(S \setminus T)$ provedenou pouze na „území“ oblasti $VR_j(T)$. Potom oblast $VR_1(x)$ v diagramu $VD_1(S \setminus T)$ v průniku s původní oblastí $VR_j(T)$ dává oblast $VR_{j+1}(T \cup \{x\})$ Voroného diagramu řádu $j + 1$.
   /* Při konstrukci diagramu $VD_1(S \setminus T)$ stačí přitom uvažovat body $S \setminus T$ sousedící s vrcholy typu I na hranici oblasti $VR_j(T)$. */

/* Sestrojení diagramu $VD_1(S)$ zabere $O(n \log n)$ času. Přechod od diagramu $VD_j(S)$ k diagramu $VD_{j+1}(S)$ zabere čas $O(s \log s)$, kde s je počet vrcholů typu I. */

2.2 Triangulace

Algoritmus 2.8 GREADYTRIANGULATION

Vstup: Konečná množina bodů S v rovině

Výstup: Triangulace konvexního obalu S

1. $\binom{n}{2}$ možných hran uspořádáme dle velikosti do zásobníku /* pokud jsou některé už umístěny v zadání, lze přidat test korektnosti */

2. `repeat` odeber aktuální (resp. nejkratší) hranu v ze zásobníku a testujeme kompatibilitu (např. křížení hran) v $T \cup \{v\}$

3. `until` je dosažen potřebný počet hran nebo zásobník je prázdný

/* Časová složitost: 1. krok $O(n^2 \log n)$. (Je-li $\varphi(m)$ doba potřebná pro test kompatibility grafu o m vrcholech, pak čas ve druhém kroku je $O(n^2\varphi(n))$). Testujeme-li protnutí nově přidávané hrany se všemi hranami už v triangulaci obsaženými, je $\varphi(n) = O(n)$ a tedy celkový čas $O(n^3)$ */

Algoritmus 2.9 DELAUNAYTRIANGULATIONVIAVORONOI

Vstup: Konečná množina bodů S v rovině
Výstup: Triangulace konvexního obalu S

1. Vytvoříme VORONOIDIAGRAM(S).
2. Projdeme DCEL a umístíme všechny hrany triangulace.

Algoritmus 2.10 DELAUNAYTRIANGULATIONSWEEP

Vstup: Konečná množina bodů S v rovině
Výstup: Triangulace konvexního obalu S reprezentovaná DCEL

Modifikujeme algoritmus BEACHLINEVD tak, aby výstupem byla přímo Delaunyova triangulace.

Algoritmus 2.11 DelaunayTriangulationRandomized

Vstup: Konečná množina n bodů S v rovině

Výstup: Triangulace konvexního obalu S reprezentovaná DCEL

1. Přidej k S body $p_{-3} = (-3M, -3M)$, $p_{-2} = (0, 3M)$, $p_{-1} = (3M, 0)$, kde M je maximum z absolutních hodnot souřadnic bodů v S.

2. Inicializuj triangulaci $\mathcal{T}$ obsahující jediný trojúhelník $p_{-3}p_{-2}p_{-1}$

3. Inicializuj vyhledávací strukturu $\mathcal{D}$ (orientovaný acyklický graf) obsahující jediný list odpovídající tomuto trojúhelníku

4. Spočti náhodnou permutaci bodů $p_1, \ldots, p_n$.

5. `for` $r = 1$ `to` n `do`

   5.1. Najdi trojúhelník $p_ip_jp_k$ obsahující p_r.

   5.2. `if` p_r je uvnitř trojúhelníku $p_ip_jp_k$ `then`

      5.2.1. přidej hrany (p_r, p_i), (p_r, p_j), (p_r, p_k)

      5.2.2. obnov $\mathcal{D}$

      5.2.3. LegalizeEdge($p_r, (p_i, p_j), \mathcal{T}$)

      5.2.4. LegalizeEdge($p_r, (p_j, p_k), \mathcal{T}$)

      5.2.5. LegalizeEdge($p_r, (p_k, p_i), \mathcal{T}$)

   5.3. `else` /* p_r leží na hraně, řekněme (p_i, p_j) */

      5.3.1. přidej hrany (p_r, p_k), (p_r, p_l) do třetích vrcholů trojúhelníků incidentních s hranou (p_i, p_j) a tuto rozděl na dvě

      5.3.2. obnov $\mathcal{D}$

      5.3.3. LegalizeEdge($p_r, (p_i, p_l), \mathcal{T}$)

      5.3.4. LegalizeEdge($p_r, (p_l, p_j), \mathcal{T}$)

      5.3.5. LegalizeEdge($p_r, (p_j, p_k), \mathcal{T}$)

      5.3.6. LegalizeEdge($p_r, (p_k, p_i), \mathcal{T}$)

6. Zruš vrcholy p_{-3}, p_{-2}, p_{-1} a všechny incidentní hrany

7. `return(`$\mathcal{T}$`)`

Algoritmus 2.12 LEGALIZEEDGE(p_r, (p_i, p_j), $\mathcal{T}$)
/* Vkládaný bod je p_r, zkoumaná možná nelegální hrana v $\mathcal{T}$ patřící trojúhelníku $p_r p_i p_j$ je (p_i, p_j) */

1. `if` oba indexy i, j záporné `then return`
2. najdi třetí vrchol p_k trojúhelníku sousedícího s $p_r p_i p_j$ podél hrany (p_i, p_j)
3. `if` QNOTLEGAL((p_i, p_j), p_r, p_l) `then`
   3.1. Zruš hranu (p_i, p_j)
   3.2. vytvoř hranu (p_r, p_k)
   3.3. obnov $\mathcal{D}$
   3.4. LEGALIZEEDGE(p_r, (p_i, p_k), $\mathcal{T}$)
   3.5. LEGALIZEEDGE(p_r, (p_k, p_j), $\mathcal{T}$)

Algoritmus 2.13 QNOTLEGAL((p_i, p_j), p_r, p_l)

1. `if` všechny indexy i, j, r, l kladné `then`
   `if` p_l uvnitř kružnice propsané $p_i p_j p_k$ `then return(true)`
   `else return(false)`
2. `elseif` právě jeden index záporný `then`
   je–li i nebo j záporné `return(true)` jinak `return(false)`
3. `elseif` právě dva indexy jsou záporné `then`
   /* musí být jeden z i, j a jeden z k, l */
   Je–li negativní index u i, j v absolutní hodnotě menší než ten u k, l, pak `return(false)`, jinak `return(true)`

/* tři záporné indexy nemohou nastat */

Algoritmus 2.14 PATHTRIANGULATE

Vstup: Rovinný graf G obsahující body a (případně) hrany
Výstup: Triangulace obsahující G

1. $G :=$ CONVEXHULL(G)

2. $G :=$ REGULARIZE(G)
3. $G :=$ BALANCEWEIGHTS(G)
4. $G :=$ TRIANGULATEPOLYGONS(G)

Algoritmus 2.15 REGULARIZE(G)
Vstup: Rovinný graf G
Výstup: Regulární graf obsahující G
Reprezentace struktur jako vyvážený vyhledávací strom ($\Longrightarrow$ úpravy v $O(\log n)$).

1. inicializace pročesávací přímky pro v_n (každá hrana v $IN(v_n)$ je hranicí intervalu a v_n je přiřazeným bodem všech)
2. `for` $i := n - 1$ `to` 1 `do`
   - 2.1. vyhledej interval, ve kterém leží v_i
   - 2.2. `if` $|OUT(v_i)| = 0$ `then` spoj v_i s bodem přiřazeným nalezenému intervalu
   - 2.3. obnov statut pročesávací přímky (sloučit intervaly ohraničené $OUT(v_i)$ v jeden, a ten rozdělit podle hran $IN(v_i)$)
3. SYMETRICKY JAKO BOD 2, ZDOLA NAHORU

/* Čas potřebný k jednomu průchodu cyklu 2 (analogicky bod 3) je díky vyhledávání mezi intervaly $O(\log n + h_i)$, kde h_i je počet hran vycházejících z bodu v_i (dělení na intervaly pomocí v_i, analogicky pro opačný průchod). Součet h_i přes všechna i je lineární, proto v celkové časové složitosti dominuje $O(\log n)$. Čas je tedy celkem $O(n \log n)$. Prostor je lineární ($O(n)$). */

Algoritmus 2.16 BALANCEWEIGHTS(G)
Vstup: Regulární graf G
Výstup: Váhy W hran, které reprezentují nějakou monotónní úplnou množinu řetězců

1. `for` všechny hrany e `do` $W(e) := 1$
2. `for` $i := 2$ `to` $n - 1$ `do`

2.1. $W_{IN}(v_i) := \sum_{e \in IN(v_i)} W(e)$

2.2. $d :=$ první výstupní hrana v $OUT(v_i)$

2.3. `if` $W_{IN}(v_i) > |\,OUT(v_i)|$ `then`

2.3.1. $W(d) := W_{IN}(v_i) - |\,OUT(v_i)| + 1$

3. `for` $i := n - 1$ `to` 2 `do`

3.1. $W_{OUT}(v_i) := \sum_{e \in OUT(v_i)} W(e)$

3.2. $d :=$ poslední vstupní hrana v $IN(v_i)$

3.3. `if` $W_{OUT}(v_i) > W_{IN}(v_i)$ `then`

3.3.1. $W(d) := W_{OUT}(v_i) - W_{IN}(v_i) + W(d)$

Spotřebujeme lineární množství paměti. Čas je také lineární, protože procházíme dvakrát všechny vrcholy (kromě v_1 a v_n), přičemž akce provedená v každém průchodu je konstantní.

Algoritmus 2.17 TriangulatePolygons(G)
/* Pro všechny monotónní polygony v G provádí následující triangulaci */

1. u_1, u_2 dáme do zásobníku STACK; $i := 1$

2. `for` všechny $u_i, i > 2$ `do`

2.1. `if` u_i sousedí s $v_1 := bottom$(STACK)
ale nesousedí s $v := top$(STACK) `then`

2.1.1. přidej hrany $(u_i, v_2), (u_i, v_3), \ldots, (u_i, v)$

2.1.2. smaž zásobník a vlož do něho prvky v, u_i

2.2. `elsif` u_i sousedí s $v := top$(STACK)
ale nesousedí s $v_1 := bottom$(STACK) `then`

2.2.1. `while` délka zásobníku je větší než 1 a $\angle(u_i v_{top} v_{top-1}) < 180^o$ `do`

2.2.1.1. přidej hranu (u_i, v_{top-1}) a seber vrchol zásobníku

2.2.2. přidej u_i do zásobníku

2.3. `else` /* sousedí s oběma */

2.3.1. přidej diagonály $(u_i, v_{bottom+1}), \ldots, (u_i, v_{top-1})$ /* algoritmus končí */

2.3 Vyhledávání v rovinných rozděleních

Algoritmus 2.18 VYHLEDÁVÁNÍ V RVS

Vstup: bod q a RVS pro cesty s vrcholy v množině S
Výstup: dvě sousední cesty v RVS a jejich hrany, mezi kterými bod q leží, nebo informace, že leží mimo $CH(S)$

1. Pokud q neleží v pásu mezi nejvyšším a nejnižším vrcholem, pak neleží v žádné buňce
2. $l := 0; r := k + 1$ (k je počet cest, $k = 2^{\alpha+1} - 1$)
3. $P_0 := P_1; P_{k+1} := P_k; L :=$ horizontální přímka procházející q
4. $e_l :=$ hrana v P_l protínající L
5. $e_r :=$ hrana v P_r protínající L
6. Pokud q není mezi hranami e_l, e_r, pak není v žádné buňce
7. `while` $r > l + 1$ `do`
   - 7.1. $m := \lceil (l + r)/2 \rceil$
   - 7.2. `if` $R(e_l) \geq m$ `then` $l := m$
   - 7.3. `elsif` $L(e_r) \leq m$ `then` $r := m$
   - 7.4. `else` najdi hranu v P_m protínající L a aktualizuj l, r, e_l, e_r

Algoritmus 2.19 FINDPOSITION(e)

Vstup: Hrana e a hodnoty $L(e)$, $R(e)$
Výstup: $Pos(e)$

1. `if` $L(e) = R(e)$ `then`
   - 1.1. $Pos(e) := L(e)$ a skonči
2. `else`
   - 2.1. $poc := -1; l := L(e); r := R(e); b := TRUE$

2.2. `while` $l \neq r$ `do`

2.2.1. $poc := poc + 1$

2.2.2. `if` l je liché `then` $b := FALSE$

2.2.3. $l := \lfloor l/2 \rfloor$

2.2.4. $r := \lfloor r/2 \rfloor$

2.3. `if` b `then`

2.3.1. $Pos(e) := L(e)$

2.4. `else`

2.4.1. $Pos(e) := (2l + 1).2^{poc}$

Algoritmus 2.20 REFINEMENTBASEDSEARCHSTRUCTURE

Vstup: Jednoduché rovinné rozdělení S s n vrcholy (vložené do 'velikého trojúhelníku')

Výstup: Vyhledávací struktura DAG (Directed Acyclic Graph) s logaritmickou hloubkou a lineární složitostí

1. `if` $n \leq 100$ `return` 'ANOTHERMETHOD'(S) /* pro malá rozdělení prostě řešíme nějakou přímou jednoduchou metodou (třeba kachlíky) */

2. Najdi velkou podmnožinu I nezávislých vektorů stupně nejvýše 9 /* mohutnost alespoň $(4n - 9)/70$ */

3. Odstraň I a roztrianguluj vzniklé mnohoúhelníky v S

4. `return` REFINEMENTBASEDSEARCHSTRUCTURE(S).

Algoritmus 2.21 TRAPEZOIDALMAP

Vstup: Množina neprotínajících se úseček S (tj. mohou se protnout jen v koncových bodech)

Výstup: Vyhledávací struktura lichoběžníků $\mathcal{T}(S)$, spolu s vyhledávací strukturou $\mathcal{D}$ (Directed Acyclic Graph)

1. Urči ohraničující obdélník R obsahující všechny úsečky z S (bounding box) a inicializuj pro něj $\mathcal{T}$ a $\mathcal{D}$

2. Spočti náhodnou posloupnost $s_1, \ldots, s_n$ úseček z S.

3. `for` $i := 1$ `to` n `do`
   3.1. FollowSegment($\mathcal{T}, s_i$) /* Najde množinu lichoběžníků $\Delta_0, \ldots, \Delta_k \in \mathcal{T}$ protnuté s_i. */
   3.2. Odstraň nalezené $\Delta_0, \ldots, \Delta_k$ z $\mathcal{T}$ a nahraď je nově vzniklými
   3.3. Odstraň listy v $\mathcal{D}$ odpovídající $\Delta_0, \ldots, \Delta_k$, nahraď je novými a aktualizuj $\mathcal{T}$ a $\mathcal{D}$

Algoritmus 2.22 FollowSegment($\mathcal{T}, s_i$)
Vstup: Lichoběžníkové rozdělení $\mathcal{T}$ a úsečka s_i uvnitř
Výstup: Posloupnost lichoběžníků proťatých s_i
/* p je levý konec s_i q pravý */

1. Hledej v $\mathcal{D}$ lichoběžník Δ_0, ve kterém je p
2. $j := 0$
3. `while` q e napravo od pravého konce Δ_j `do`
   3.1. `if` pravý konec Δ_j leží nad s_i
      3.1.1. `then` $\Delta_{j+1} :=$ dolní pravý soused Δ_j
      3.1.2. `else` $\Delta_{j+1} :=$ dorní pravý soused Δ_j
   3.2. $j := j + 1$
4. `return` $\Delta_0, \ldots, \Delta_j$.

2.4 Geometrické transformace

2.1 Definice. Pro $p = [x, y, z]$ značíme $|p| = \sqrt{x^2 + y^2 + z^2}$. Funkce $\varphi : \mathbf{R}^3 \longrightarrow \mathbf{R}^3$ definovaná
$$\forall p \in \mathbf{R}^3 : \varphi(p) = \frac{p}{|p|^2}$$
se nazývá *inverze vzhledem ke sféře*
$$k \equiv x^2 + y^2 + z^2 = 1$$

Sféra k je v inverzi z předchozí definice právě množinou samodružných bodů.

2.2 Lemma. *Roviny a sféry jsou v inverzi vzhledem ke sféře k zobrazeny na roviny nebo sféry.*

Uvažujme sféru se středem s a poloměrem r. Pak její rovnice je

$$|p-s|^2 = r^2$$
$$(= (x(p)-x(s))^2 + (y(p)-y(s))^2)$$

Pro $|s| \neq r$ uvažme $p' = \varphi(p)$

$$|x-y|^2 = (x-y, x-y)$$

$$|p' - \frac{s}{|s|^2 - r^2}|^2 =$$

$$= \frac{1}{|p|^2} + \frac{|s|^2}{(|s|^2-r^2)} - \frac{2ps}{|p|^2(|s|^2-r^2)} =$$

$$= \frac{r^2}{(|s|^2-r^2)^2}$$

Je tedy φ(sféry) = sféra se středem $s' = \frac{s}{|s|^2-r^2}$ a poloměrem $\frac{|r|}{|s|^2-r^2|}$.

Jestliže střed s zvolíme pevně a $r \to |s|$, pak s' „utíká" po polopřímce $\overline{0s}$ do nekonečna, proto obrazem limitní sféry procházející počátkem bude rovina kolmá na $\overline{0s}$.

Víme, že složení dvou inverzí po sobě dává identitu

$$\varphi \circ \varphi = id$$

Vezmeme-li tedy všechny sféry procházející počátkem, jsou jejich obrazy všechny možné roviny počátkem neprocházející. Po aplikaci stejné inverze zobrazíme tedy tyto roviny na sféry procházející počátkem.

Roviny počátkem procházející se zobrazí samy na sebe.

Celkem tedy se každá sféra zobrazí na sféru nebo rovinu, a každá rovina se zobrazí na rovinu nebo sféru.

Algoritmus 2.23 VoronoiViaConvexHull

Vstup: Množina bodů v $\mathbb{R}^2$
Výstup: Voroného diagramy nejnižšího a nejvyššího řádu

/* Uvažujeme rovinu $\eta \equiv z = 1$, která je tečná ke sféře k v bodě $(0, 0, 1)$. Její obraz je sféra se středem $s = (0, 0, 1/2)$ a poloměrem $1/2$. Uvažujeme body $p_i \in \eta, i = 1, \ldots, n$ (množina S). Označíme $z_i = \varphi(p_i) \in \varphi(\eta)$, $S' = \{z_1, \ldots, z_n\}$. Konvexní obal $BCH(S')$ rozdělíme na disjunktní části C_1 – body viditelné z počátku O a C_2 – neviditelné přes BCH. Pokud žádné čtyři body v S neleží na kružnici, jsou v $BCH(S')$ samé trojúhelníky. */

1. Přidáme ke všem bodům na vstupu třetí souřadnici rovnou 1.

2. Určíme obrazy z_i v inverzi vzhledem k jednotkové sféře.

3. Nechť F je neviditelná stěna v $BCH(S')$ (tj. aspoň jeden bod $z_i \in C_2$). Pak rovina zadaná vrcholy $z_i, z_j, z_k \in F$ se zobrazí na sféru k_F. Na vnitřek k_F se zobrazí ve φ poloprostor neobsahující počátek a oddělený rovinou F. Odtud plyne, že vnitřek k_F neobsahuje žádné body S. Proto střed kružnice, která je průnikem sféry k_F s rovinou θ, je vrchol $VD_1(S)$.

4. Nechť F je viditelná stěna v $BCH(S')$. Pak ze stejného důvodu bude k_F procházet body p_i, p_j, p_k a uvnitř budou všechny body S. Proto střed kružnice $k_F \cap \theta$ je vrchol $VD_{n-1}(S)$.

Tento algoritmus lze zobecnit pro všechny dimenze $d \geq 3$. Konvexní obaly v $\mathbf{R}^d$ pro $d > 2$ umíme najít v čase $O(n^{\lfloor d/2 \rfloor + 1}) + O(n^{\lfloor d/2 \rfloor} \log n)$, pro $\mathbf{R}^3$ speciálně v čase $O(n \log n)$.

Rozdělení vrcholů v $BCH(S')$ na C_1, C_2 podle „viditelnosti“ z počátku provedeme snadno výpočtem orientovaného objemu příslušných rovnoběžnostěnů.

2.3 Definice. Nechť h je (nevertikální) nadrovina v $\mathbf{R}^{d+1}$ daná rovnicí $x_0 = p_1 x_1 + \ldots + p_d x_d + p_{d+1}$. Pak definujeme *duální bod k nadrovině* h: $\delta(h) = [p_1, \ldots, p_d, p_{d+1}] = p$ a *duální nadrovinu k bodu* $p = [p_1, \ldots, p_d, p_{d+1}]$: $\delta(p) \equiv x_0 = -p_1 x_1 - \ldots - p_d x_d + p_{d+1}$. *Vertikální vzdálenost* bodu $p = [p_1, \ldots, p_d, p_{d+1}]$ od nadroviny $h \equiv x_0 = q_1 x_1 + \ldots + q_d x_d + q_{d+1}$ definujeme

$$vd(h, p) := p_{d+1} - (p_1 q_1 + \ldots p_d q_d + q_{d+1})$$

2.4 Lemma.

$$vd(h,p) = -vd(\delta(p), \delta(h))$$

$$p \in h \Longleftrightarrow \delta(h) \in \delta(p)$$

2.5 Důsledek. *Nechť $p_1, p_2 \in \mathbf{R}^3$ jsou body v prostoru, $h_1, h_2 \subset \mathbf{R}^3$ roviny. Jestliže $L(p_1, p_2) \subset h_1 \cap h_2$, pak $L(\delta(h_1), \delta(h_2)) \subset \delta(p_1) \cap \delta(p_2)$.*

Nechť h_i je množina rovin v $\mathbf{R}^3$, $h_i^{\pm}$ příslušné poloprostory nad resp. pod h_i. Chceme dostat

$$S = \bigcap_{i=1}^{m} h_i^+ \cap \bigcap_{i=m+1}^{n} h_i^- = S^+ \cap S^- \tag{2.1}$$

2.6 Lemma. *Rovina h_a neobsahuje žádnou stěnu S^+ právě tehdy, když $\delta(h_a)$ není vrcholem horního konvexního obalu množiny $\{\delta(h_i) \mid 1 \leq i \leq n\}$.*

Je-li h_a „nadbytečná“, potom existuje nejbližší vrchol v v S^+ a tři incidentní stěny h_i, h_j, h_k tak, že

$$h_i^+ \cap h_j^+ \cap h_k^+ = h_i^+ \cap h_j^+ \cap h_k^+ \cap h_a$$

Pak $\delta(h_a)$ leží na nebo pod $\delta(v)$. Navíc $\delta(v)$ je rovina určená body $\delta(h_i), \delta(h_j), \delta(h_k)$. Proto $\delta(h_a)$ není v příslušném konvexním obalu.

Algoritmus 2.24 HalfSpaceIntersection

Vstup: Množina poloprostorů $h_i^{\pm}$ v $\mathbb{R}^d$

Výstup: Průnik všech poloprostorů $h_i^{\pm}$

/* předpokládáme (pro jednoduchost), že žádná nadrovina h_i není "vertikální" */

1. Najdeme vrcholy konvexních obalů množin $\delta(h_1^+), \ldots, \delta(h_m^+)$ a $\delta(h_{m+1}^-), \ldots, \delta(h_n^-)$ a tím určíme množiny S^+ a S^- ze vztahu 2.1. Jsou to konvexní mnohostěny.

2. Spočítáme průnik $S^+ \cap S^-$.

Kapitola 3

Vyhledávání dle rozsahů a iso-ortogonální objekty

Anglický ekvivalent pro tuto třídu problémů zní *Orthogonal Range Searching*.

3.1 Multidimenzionální binární strom

Algoritmus 3.1 1D–TreeRangeSearch
Vstup: Množina n bodů na přímce a dotazovaný rozsah, tj. interval $[x', x'']$.
Výstup: body, které patří do dotazovaného rozsahu

1. Binárním vyhledáváním najdeme nejlevější bod p v intervalu $[x', x'']$.
2. Postupně zpracováváme bod za bodem, až narazíme na pravý konec x''.

/* Potřebná datová struktura je binární strom, jehož listy jsou propojeny ukazately do řetězu (*threaded binary tree*). Čas vyhledání je $\Theta(\log n + k)$ (optimální) při použité paměti $\Theta(n)$. */

3.1 Definice. *2-D strom* má s každým uzlem spojen obdélník $\mathcal{R}(v)$ a množinu bodů $S(v) \subseteq S$ obsažených v $\mathcal{R}(v)$. S v spojujeme vybraný bod $P(v) \in S(v)$. Bodem $P(v)$ vedeme přímku rovnoběžnou s jednou z os, které rozdělí $S(v)$ přibližně na poloviny. V dělení pokračujeme při obracení orientace dělicí přímky na každé úrovni stromu, až v získaných obdélnících nejsou žádné body z S. Takové uzly jsou listy 2-D stromu. Každý uzel má tvar $(P(v), t(v), M(v))$, kde

$P(v)$ je bod spojený s tímto uzlem, $t(v)$ je orientace (vertikální nebo horizontální), a $M(v)$ je buď x-ová souřadnice (pro vertikální uzly) nebo y-ová (pro horizontální uzly).

Algoritmus 3.2 2D–TreeRangeSearch

Vstup: vrchol v, ve kterém začíná vyhledávání, interval $D = [x_1, x_2] \times [y_1, y_2]$
Výstup: na výstupu se postupně objevují výsledné body dotazu
Vyvolání: je-li T 2-D strom, pak SEARCH($root(T)$)

`Procedure` SEARCH

1. `if` $t(v) =$ vertikální `then`
   1.1. $(l, r) := (x_1, x_2)$
2. `else`
   2.1. $(l, r) := (y_1, y_2)$
3. `if` $l \leq M(v) \leq r$ `then`
   3.1. `if` $P(v) \in D$ `then` pošli $P(v)$ na výstup
4. `if` v není list `then`
   4.1. `if` $l < M(v)$ `then` SEARCH($LSON(v), D$)
   4.2. `if` $M(v) < r$ `then` SEARCH($RSON(v), D$)

Prostor: 2-D strom potřebuje paměť $\Theta(n)$
Příprava: 2-D strom se sestrojí v čase $O(n \log n)$
Čas vyhledání: Není logaritmický, jak by se mohlo zdát. Algoritmus totiž navštěvuje řadu uzlů stromu „zbytečně“, tj. aniž by to vedlo k příspěvku do výstupu. Jejich počet je nejvýše $O(\sqrt{n})$, proto vyhledání proběhne v čase $O(\sqrt{n} + s)$, kde s je délka výstupu.

Pomocí obecných d-D stromů lze pro všechny dimenze $d \geq 2$ řešit rozsahový dotaz pro n bodů v $\mathbf{R}^d$ v čase $O(dn^{1-1/d}+s)$, při paměti $\Theta(dn)$ a v čase přípravy $\Theta(dn \log n)$.

3.2 Metoda přímého přístupu

Budeme pracovat v rovině s množinou S bodů o mohutnosti n. Každým bodem $p \in S$ proložíme horizontální i vertikální přímku. Tyto přímky rozdělí rovinu na $(n+1)^2$ (zobecněných) obdélníků. Zvolíme-li rozsah $D = [x_1, x_2] \times [y_1, y_2]$ a pohybujeme-li každým krajním bodem tohoto rozsahu $(x_1, y_1), (x_2, y_2)$ uvnitř jednoho z výše uvedených $(n+1)^2$ obdélníků, výsledek dotazu zůstane stále stejný. Máme tedy celkem

$$\binom{n+1}{2} \times \binom{n+1}{2} = O(n^4)$$

možných výsledků (vybereme vždy 2 z $(n+1)$ pásů na každé z os). K jejich uložení do paměti je však potřeba $O(n^5)$ prostoru, protože výsledky mají délku $O(n)$.

Zlepšení: Pro rozsah $D = [x_1, x_2] \times [y_1, y_2]$ provedeme přímý přístup pouze ve směru osy x. Tak dostaneme ukazatel na vyhledávací strom všech bodů množiny S, které leží v daném x-ovém pásu, odpovídajícím intervalu $[x', x'']$ na ose x. Jinými slovy, pro každou dvojici pásů, kterých je $n+1$, si pamatujeme vyhledávací strom všech bodů z S, které leží mezi těmito pásy.

Provedeme normování reálných souřadnic na celočíselné souřadnice $1, \dots, n$, tzn. seřadíme tyto souřadnice podle velikosti a očíslujeme podle pořadí. Dostaneme pak v čase $O(\log n)$ pro daný interval dvě čísla $i, j \in \{1, \dots, n+1\}$. Dvojice (i, j) odpovídá binárnímu vyhledávacímu stromu s $(j-i)$ listy. Celkem potřebujeme paměť

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n+1} (j-i) = \frac{(n+1)^3 - (n+1)}{6} = O(n^3)$$

Všimněme si, že při radikálním snížení potřebné paměti (o dva řády), zůstal zachován logaritmický vyhledávací čas.

Dalšího vylepšení — *kalibrace*. Budeme vyhledávat na ose x postupně ve dvou úrovních. Osa x bude rozdělena tzv. *hrubou kalibrací* na intervaly, jejichž hraničními body zůstávají x-ové souřadnice bodů množiny S, ale ne nutně všechny. *Jemná kalibrace* nám už konečně rozdělí každý interval hrubé kalibrace na konečné intervaly. Při vyhledávání podle zadaného intervalu $[x_1, x_2]$ takto dostaneme dvojici hrubých intervalů, které jsou intervalem $[x_1, x_2]$ plně pokryty, a dvě dvojice jemných intervalů po stranách krajních hrubých intervalů. Každá dvojice hrubých intervalů ukazuje na nějaký vyhledávací strom, a každá dvojice jemných intervalů uvnitř jednoho hrubého intervalu ukazuje na svůj vyhledávací strom bodů S obsažených v odpovídajících pásech.

Předpokládejme, že hrubá kalibrace obsahuje n^α intervalů ($0 < \alpha \leq 1$). Máme $O(n^{2\alpha})$ možných dvojic těchto intervalů, a každá taková dvojice ukazuje na strom o velikosti $O(n)$. Potřebná paměť je tedy $O(n^{2\alpha+1})$. Každé rozdělení hrubé jednotky má $n^{1-\alpha}$ jemných intervalů, hrubých jednotek je n^α. Pro každou dvojici jemných intervalů, kterých je v každém hrubém intervalu $O(n^{2(1-\alpha)})$, si pamatujeme vyhledávací strom o velikosti $O(n^{1-\alpha})$. Celková paměť pro jemnou kalibraci je tedy $O(n^{3-2\alpha})$. Spolu s hrubou kalibrací je potřebný prostor dohromady

$$O(n^{2\alpha+1}) + O(n^{3-2\alpha})$$

Minima dosáhneme pro $\alpha = 1/2$ (tedy hrubých intervalů je $\sqrt{n}$, stejně jako jemných uvnitř každého hrubého), a toto minimum je $O(n^2)$. Časově se nic nezmění, pouze logaritmicky procházíme tři stromy za sebou (musí být vyvážené), proto čas je $O(3 \log n) = O(\log n)$.

Zvyšujeme-li počet úrovní kalibrací na k, vyjde vždy optimum jako $\sqrt[k]{n}$ intervalů na každé úrovni kalibrace, a výsledný prostor bude $O(n^{1+2/k})$ při zachování logaritmického času.

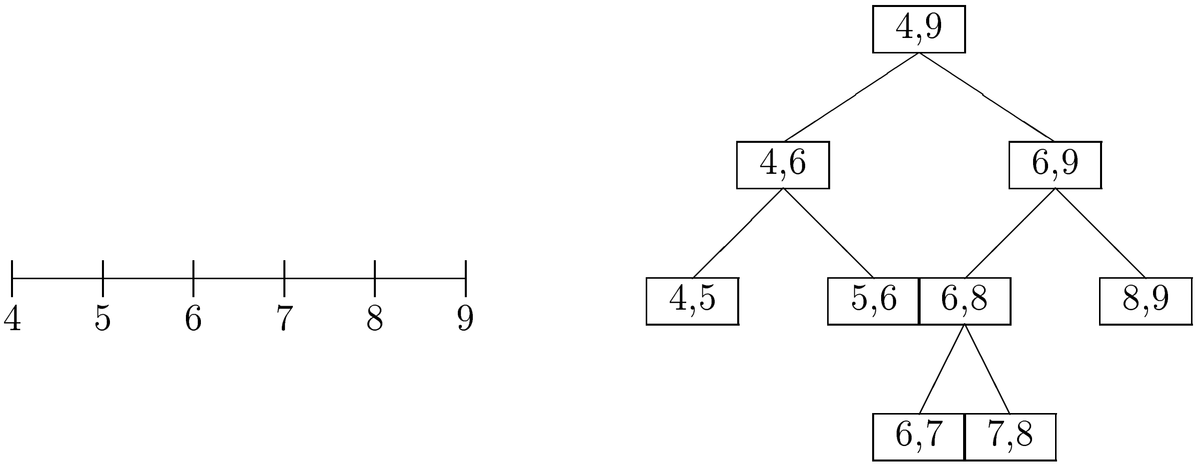
3.2 Tvrzení. *Pro každé $\varepsilon : 1 > \varepsilon > 0$ lze vyhledávání podle rozsahu v množině n bodů v rovině uskutečnit v čase $O(\log n)$ při paměťové náročnosti $O(n^{1+\varepsilon})$.*

3.3 Rozsahový strom

Strom úseček bude sloužit k uchovávání (aplikací požadovaných) informací o uvažovaných úsečkách. K použití potřebujeme implementovat operace INSERT a DELETE sloužící k vložení a zrušení příslušné informace o dané úsečce. Budeme stručně hovořit o „alokování“ úsečky v uzlech stromu.

Algoritmus 3.3 INSERT (na stromu úseček)

Vstup: Úsečka (b, e) a kořen v úsečkového stromu s uzly $u = (B(u), E(u))$
Výstup: Tentýž strom s alokovanou úsečkou (b, e)
Vyvolání: INSERT$((b, e), v)$



Obrázek 3.1: Strom úseček pro normalizovanou sadu bodů na přímce

1. `if` $b < B(v)$ `and` $e \geq E(v)$ `then`
   1.1. alokuj (b, e) k v
2. `else`
   2.1. `if` $b < \lfloor (B(v) + E(v))/2 \rfloor$ `then`
      2.1.1. INSERT$(b, e, LSON(v))$
   2.2. `if` $\lfloor (B(v) + E(v))/2 \rfloor < e$ `then`
      2.2.1. INSERT$(b, e, RSON(v))$

Pracuje v čase $O(\log n + s)$, kde s je čas potřebný k alokaci potřebné informace. Zejména je z algoritmu patrné, že každá vstupní úsečka (b, e) bude alokována v $O(\log(e - b))$ uzlech. Podrobnější rozbor ukazuje, že interval (b, e) se rozloží na nejvýše $\lceil \log(e - b) \rceil + \lfloor \log(e - b) \rfloor - 2$ úseček, které jsou uzly v našem stromu úseček.

Procedura DELETE je zcela shodná s INSERT, až na výměnu příkazu „alokuj“ v 1.1 příkazem „zruš alokaci“.

3.3 Příklad. Zpracování rozsahového dotazu na přímce: rozsah (který je vlastně úsečkou) zpracujeme procedurou INSERT pro strom úseček vytvořený pro zadané body množiny S. V tomto případě je význam příkazu „alokuj“ takový, že na výstup dáme všechny body množiny S patřící do příslušné úsečky.

Obecný postup: Pracujeme v d-dimenzionálním prostoru a zpracováváme postupně souřadnice $x_1, \ldots, x_d$. Pro dimenzi $d = 1$ definujeme rozsahový strom jako standardní vyhledávací strom.

Pro $d \geq 2$ definujeme rozsahový strom jako strom úseček T^* pro množinu bodů na přímce $\{x_1(p); p \in S\}$. Vzali jsme tedy v úvahu pouze první souřadnice bodů a normovali a vytvořili úsečkový strom z předchozí podkapitoly. Do stromu se však nebudou vkládat žádné úsečky.

Vzory intervalů $[B(v), E(v)]$ pro uzel v v projekci $\mathbf{R}^d \longrightarrow \mathbf{R}$ dané souřadnicí x_1 označíme $S_d(v)$. Jsou to ty body, jejichž první souřadnice leží v intervalu odpovídajícím uzlu v, tedy všechny body $p \in S$ takové, že $B(v) \leq x_1(p) \leq E(v)$. Pro tyto body definujeme množinu $S_{d-1}(v) := \{(x_2(p), \ldots, x_d(p)); p \in S_d(v)\}$ bodů $(d-1)$-ní dimenze, které vzniknou z $S_d(v)$ naopak odstraněním prvních souřadnic. Uzel v dále obsahuje ukazatel na rozsahový strom v dimenzi $(d-1)$ pro množinu $S_{d-1}(v)$.

Problémy: Pro realizaci rozsahového stromu bude potřeba více než lineárně mnoho paměťového prostoru. Rozsahový strom nižší dimenze se totiž konstruuje velice velkoryse. Je-li např. strom na obrázku 3.1 primární strukturou rozsahového stromu, pak bod odpovídající bodu 7 po normování se objeví jako $(d-1)$-dimenzionální bod v uzlech $\boxed{4{,}9}$, $\boxed{6{,}9}$, $\boxed{6{,}8}$, $\boxed{6{,}7}$ (je vhodné mít intervaly z jedné strany otevřené a z druhé zavřené, zde jsme použili zprava zavřené, v opačném případě by se bod 7 objevil v uzlu $\boxed{7{,}8}$). Lze tedy zpozorovat pravidlo, že pro každý bod najdeme právě jednu cestu z kořene k listu stromu, pro niž je v každém rozsahovém stromu dimenze $(d-1)$ odpovídajícím uzlu této cesty daný bod obsažen.

Vyhledávání probíhá tak, že každý z alokovaných intervalů pro souřadnici x_1 vede k dílčímu $(d-1)$-rozměrnému problému. Vezmeme tedy interval rozsahu odpovídající první souřadnici a najdeme v rozsahovém stromu ty uzly, jejichž odpovídající intervaly $[B(v), E(v)]$ jsou celé v rozsahovém intervalu obsaženy, a to zároveň uzly nejvýše ve stromu, které této podmínce vyhovují. V nich pak řešíme stejný problém o dimenzi nižší.

Nechť $Q(n, d)$ je vyhledávací čas pro n d-rozměrných bodů. Dílčích problémů je

$$Q(n, d) = O(\log n) + \sum Q(n(v), d-1),$$

kde sčítání probíhá přes alokované uzly v a $n(v) = E(v) - B(v)$. Alokovaných uzlů je méně než $2\lceil \log n \rceil - 2$ a $n(v) \le n$, proto můžeme aproximovat

$$Q(n, d) = O(\log n).Q(n, d-1)$$

$$Q(n, 1) = O(\log n) \Longrightarrow Q(n, d) = O(\log^d n)$$

Podobnou úvahou odvodíme potřebnou paměť

$$S(n, d) = O(n \log^{d-1} n)$$

3.4 Věta. *Pomocí rozsahových stromů lze rozsahový dotaz v d-dimenzích zvládnout v čase $O(\log^d n)$ a paměti $O(n \log^{d-1} n)$.*

3.5 Poznámka. S použitím podobné konstrukce jako v redukované vyhledávácí struktuře používané k vyhledávání v rovinných rozděleních lze snížit čas potřebný pro algoritmus. Této technice se říká *přemosťování*, a spočívá v přidání jistých ukazatelů mezi sekundárními stromy. Dosáhneme tak času $O(\log^{d-1} n)$ při prostoru $O(n \log^{d-1} n)$.

3.4 Iso-ortogonální objekty

Iso-ortogonální objekty jsou takové, které jsou jistým způsobem složeny z množin rovnoběžných úseček, obdélníků, kvádrů, apod. Nejjednodušší jsou opět jednorozměrné úlohy.

3.4.1 Míra sjednocení úseček

Mějme dáno n intervalů $[a_1, b_1], \dots, [a_n, b_n]$ v $\mathbf{R}$. Chceme získat míru (tedy celkovou délku) jejich sjednocení.

Uspořádáme koncové body úseček a budeme je postupně procházet zleva doprava. Budeme v každé chvíli vědět, uvnitř kolika úseček se nacházíme. V každém bodě zvýšíme nebo snížíme tento počet podle toho, zda jsme narazili na levý nebo pravý koncový bod. Zároveň upravíme průběžnou míru, tedy přičteme vzdálenost k dalšímu bodu, pokud budeme uvnitř alespoň jedné úsečky.

Algoritmus 3.4 IntervalUnionMeasure
Vstup: úsečky $[a_1, b_1], \dots, [a_n, b_n]$ na přímce
Výstup: míra jejich sjednocení

1. $(X(1), \dots, X(2n)) :=$ uspořádaná posloupnost počátečních a koncových bodů úseček
2. $X(0) := X(1)$
3. $M := 0$
4. $C := 0$
5. `for` $i := 1$ `to` $2n$ `do`
   - 5.1. `if` $C \neq 0$ `then` $M := M + X(i) - X(i-1)$
   - 5.2. `if` $X(i) =$ levý konec `then` $C := C + 1$ `else` $C := C - 1$

Času dominuje krok číslo 1, tedy $O(n \log n)$.

3.4.2 Míra sjednocení obdélníků

Stejnou úlohu nyní řešíme pro obdélníky v rovině.

Použijeme metodu *pročesávání*. Pročesávací přímka bude vertikální a budeme prostor procházet zleva doprava. Zastavovat se budeme v krajních bodech x-ových intervalů obdélníků. V každém takovém bodě spočítáme míru sjednocení úseček, které protnou pročesávací přímku, a vynásobenou vzdáleností k dalšímu vrcholu ji přičteme k průběžně počítané míře.

Při přechodu od $X(i-1)$ k $X(i)$ musíme obnovit míru sjednocení příslušných intervalů ve směru osy y. Při pročesávání lze jako statut událostí s výhodou použít úsečkový strom. Procedury na stromu úseček si však musíme zprecizovat, tzn. upřesnit způsob zachovávání informace o míře. Nejprve tedy budeme definovat operace na stromu úseček INSERT, UPDATE, DELETE.
Globální proměnné: $C(v)$ – počet úseček, které jsou alokovány; $m(v)$ – příspěvek do míry
`procedure` INSERT(b, e, v)

1. `if` $b \leq B(v)$ `and` $E(v) \leq e$ `then` $C(v) := C(v) + 1$

2. `else`
   (a) $b < \lfloor (B(v) + E(v))/2 \rfloor$ `then` INSERT$(b, e, LSON(v))$
   (b) $\lfloor (B(v) + E(v))/2 \rfloor \leq e$ `then` INSERT$(b, e, RSON(v))$
3. UPDATE(v)

Procedura DELETE je duální k INSERT, tj. místo přičítání se jednička odečítá.

`procedure` UPDATE(v)

1. `if` $C(v) \neq 0$ `then` $m(v) := E(v) - B(v)$
2. `else`
   (a) `if` v není list `then` $m(v) := m(LSON(v)) + m(RSON(v))$
   (b) `else` $m(v) := 0$

Hlavní program bude vypadat takto:

Algoritmus 3.5 RectangleUnionMeasure
Vstup: Množina n iso-ortogonálních obdélníků
Výstup: Míra jejich sjednocení

1. do $(X(1), \ldots, X(2n))$ uspořádáme posloupnost x-ových souřadnic bodů, v případě rovnosti spodní před vrchní (tj. včetně násobností)
2. $X(0) := X(1)$
3. $M := 0$
4. inicializuj úsečkový strom T pro y-ové souřadnice bodů
5. `for` $i = 1$ `to` $2n$ `do`
   5.1. $m1 := m(root(T))$
   5.2. $M := M + m1 * (X(i) - X(i-1))$
   5.3. `if` $X(i)$ je levý bod `then`
      5.3.1. INSERT$(b_i, e_i, root(T))$

5.4. `else`

5.4.1. DELETE($b_i, e_i, root(T)$)

Procházíme $O(n)$ bodů, v nichž provádíme logaritmické operace na stromě úseček, tedy algoritmus proběhne v čase $O(n \log n)$.

Pro obdélníky jsme v každém bodě měli vlastně za úkol spočítat míru sjednocení n úseček, tedy úloha se redukovala na problém o dimenzi nižší. Tento přístup lze uplatnit v libovolné dimenzi.

Pro objekty podobné ortogonálním obdélníkům obecně v $\mathbf{R}^d$ lze aplikovat pročesávání postupně ve všech dimenzích až dojde na přímku. Např. pro $d = 3$ redukujeme úlohu na řešení n problémů v $\mathbf{R}^2$, celkový čas je tedy $O(n^2 \log n)$. Díky této úvaze dostáváme následující větu.

3.6 Věta. *Míra sjednocení iso-ortogonálních objektů v $\mathbf{R}^d, d \geq 2$, se dá spočítat v čase $O(n^{d-1} \log n)$*

3.4.3 Obvod sjednocení obdélníků

Obvod sjednocení obdélníků (délka hranice) lze spočítat stejnou metodou jako míru jejich sjednocení. Budeme opět používat strom úseček jako statut pročesávací přímky při pročesávání, musíme však upravit některé části algoritmu tak, abychom dostali obvod místo míry.

Vertikální komponenty přispívají při přechodu od $X(i-1)$ k $X(i)$ právě délkou m_i, použitou už v předchozím algoritmu.

Nechť P_i je sjednocení disjunktních komponent statutu událostí. Počet disjunktních komponent v P_i vynásobený dvěma uložím do parametru α. K údržbě parametru α je třeba parametrů $LBD(v), RBD(v)$:

$LBD(v) = 1$ je-li $B(v)$ dolní bod intervalu v $[B(v), E(v)] \cap P_i$
$LBD(v) = 0$ jinak
$RBD(v) = 1$ je-li $E(v)$ horní bod intervalu v $[B(v), E(v)] \cap P_i$
$RBD(v) = 0$ jinak

Algoritmus je velmi podobný jako algoritmus na spočítání míry sjednocení obdélníků (3.5), proto uvedeme pouze změny v proceduře UPDATE a hlavním programu.

`procedure`UPDATE

1. `if` $C(v) \neq 0$ `then`
   (a) $\alpha(v) := 2; m(v) := E(v) - B(v)$
   (b) $LBD(v) := 1; RBD(v) := 1$
2. `else if` v není list tt then
   (a) $\alpha(v) := \alpha(LSON(v)) + \alpha(RSON(v)) - 2 * RBD(LSON(v)) * LBD(RSON(v))$
   (b) $LBD(v) := LBD(LSON(v))$
   (c) $RBD(v) := RBD(RSON(v))$
   (d) $m(v) := m(LSON(v)) + m(RSON(v))$
3. `else` $LBD(v) = 0$, $RBD(v) = 0$, $\alpha(v) = 0$, $m(v) = 0$

Algoritmus 3.6 RectangleUnionBoundaryMeasure
Vstup: Množina n iso-ortogonálních obdélníků
Výstup: Obvod jejich sjednocení

1. do $(X(1), \ldots, X(2n))$ uspořádáme posloupnost x-ových souřadnic bodů, v případě rovnosti spodní před vrchní
2. $X(0) := X(1)$
3. $m0 := 0; P := 0$
4. inicializuj úsečkový strom T pro y-ové souřadnice bodů
5. `for` $i = 1$ `to` $2n$ `do`
   5.1. $\alpha 1 := \alpha(root(T))$
   5.2. `if` $X(i)$ je levý bod `then`
      5.2.1. INSERT$(b_i, e_i, root(T))$
   5.3. `else`
      5.3.1. DELETE$(b_i, e_i, root(T))$
   5.4. $m1 := m(root(T))$
   5.5. $P := \alpha 1 * (X(i) - X(i-1)) + |m1 - m0| + P$

5.6. $m0 := m1$

Čas: při n zastaveních pročesávací přímky, kdy každé spotřebuje logaritmický čas, dostáváme celkový čas $O(n \log n)$.