Jan Slovák

Geometrické algoritmy I.

podle přednášek zpracoval Josef Pojsl
říjen 1993 – leden 1994

Obsah

Poznámka úvodem

Cítím povinost říci úvodem něco o smyslu a cíli přednášek, na jejichž základě tyto učební texty vznikly, a také o okolnostech jejich vzniku.

V rámci zavádění nového cyklu přednášek blízkých aplikacím, ale s matematickou náplní, pro studenty vyšších ročníků odborné informatiky a odborné matematiky, jsem přijal úkol připravit přednášky z oboru, kterému se anglicky říká „computational geometry“. Někteří kolegové to (skoro škodolibě) komentovali, že z dostupných geometrů jsem najčastěji zapínal počítač, proto prý já. Hned jsem zjistil, že se jedná o obor v bouřlivém rozvoji, se stovkami nedávných časopiseckých publikací, ale s velice málo monografickými texty. Shromáždil jsem tedy alespoň to, co bylo dostupné, vybral jsem několik témat a snažil se uvést zájemce do této oblasti. Rozhodl jsem se, že tématicky přednášku rozdělím na dvě poloviny. V prvním semestru se zabývám lineárně definovanými objekty, ve druhém pak obecnými algebraicky zadanými útvary. V obou případech mi jde o předvedení několika základních principů výstavby algoritmů a předvedení jejich možných aplikací.

Tento text pokrývá přednášky z první části. Musím říci, že v roce jeho vzniku jsem měl radost z dobré odezvy u posluchačů a zejména si moc cením práce pana Josefa Pojsla, který, prakticky bez mého dalšího přispění, celé učební texty na základě přednášek sepsal, opatřil obrázky a typograficky dovedl do stavu, který nyní máte před sebou.

Za obsahovou stránku ovšem musím ručit sám. Přednáška se jistě částečně překrývá s přednáškou z grafiky, zejména 1. kapitolu je třeba brát jako jakousi rozcvičku. V dalších čtyřech kapitolách se snažím systematicky probrat řadu základních úloh a zároveň základních principů tvorby algoritmů. Jakékoli komentáře, dotazy, výhrady apod. posílejte prosím na adresu `slovakmath.muni.cz`.

Brno 1995, Jan Slovák

O rok později nemám moc co dodat, snad jen, že v obou polovinách přednášky stále více využívám služeb programového systému MAPLE V. Doufám, že to není jen můj dojem, že se jedná o systém užitečný, a věřím, že seznámení s ním je samo o sobě přínosem. Zápočtové projekty, převážně vypracované právě na základě tohoto systému, lze najít v mém adresáři `/VYUKA/Maple/galg` na stroji hunter, připojím do něj i letošní.

V textech jsem v podstatě jen opravil některé z pravopisných chyb, překlepů, apod. Děkuji všem, které mne na spoustu takových nedostatků upozornili.

Brno 1996, Jan Slovák

1 Konvexní mnohoúhelníky

Objekty, které budeme popisovat (a nejen v této kapitole), se vždy nacházejí v Euklidovském prostoru příslušné dimenze d, který značíme $\mathbf{E}^d$. Bude-li v textu uveden prostor $\mathbf{R}^d$, budeme s ním někdy implicitně zacházet jako s jeho euklidovským rozšířením.

Bez snahy o exaktní definice zavedeme nyní pojmy, u kterých by mohl nejednoznačný výklad vést ke zmatení.

1.1 Definice.

Konvexní množina: Množina D v prostoru $\mathbf{E}^d$ je konvexní, pokud pro každé její dva elementy q_1, q_2 je celá úsečka $\overline{q_1 q_2}$ obsažena v množině D.

Konvexní obal: Konvexní obal množiny S je **hranice** nejmenší (ve smyslu množinové inkluze) konvexní množiny obsahující celou množinu S[1].

Mnohoúhelník: V $\mathbf{E}^2$ je mnohoúhelník definován jako konečná množina úseček, jejichž koncové body jsou sdíleny vždy právě dvěmi z nich, přičemž žádná vlastní podmnožina těchto úseček není mnohoúhelník (tedy nemá tu vlastnost, že by koncové body všech takto vybraných úseček byly sdíleny právě dvěma úsečkami z této podmnožiny). Taková definice nám zaručí, že budeme pracovat s cyklickými řetězci na sebe navazujících úseček.

Jednoduchý mnohoúhelník: je takový, ve kterém se žádné dvě nesousedící hrany neprotínají. Jednoduché mnohoúhelníky dělí rovinu na dvě části, z nichž jedna je ohraničená (*vnitřek*) a druhá neohraničená (*vnějšek mnohoúhelníka*).

Konvexní mnohoúhelník: je takový jednoduchý mnohoúhelník, jehož vnitřek je konvexní množina.

Jádro mnohoúhelníka: je množina bodů k z vnitřku jednoduchého mnohoúhelníka takových, že pro každý bod p mnohoúhelníka je celá úsečka $\overline{kp}$ uvnitř mnohoúhelníka.

Analogicky definujeme pojmy *mnohostěn*, *jednoduchý mnohostěn*, *konvexní mnohostěn* a *jádro mnohostěnu*.

V této kapitole se budeme zabývat konvexními útvary. Nejprve představíme konvexní mnohoúhelníky a jednu jejich reprezentaci, s níž se naučíme provádět základní operace. Přejdeme k problému nalezení konvexního obalu množiny bodů, a nakonec uvedeme několik aplikací tohoto problému.

1.1 Průniky konvexních mnohoúhelníků

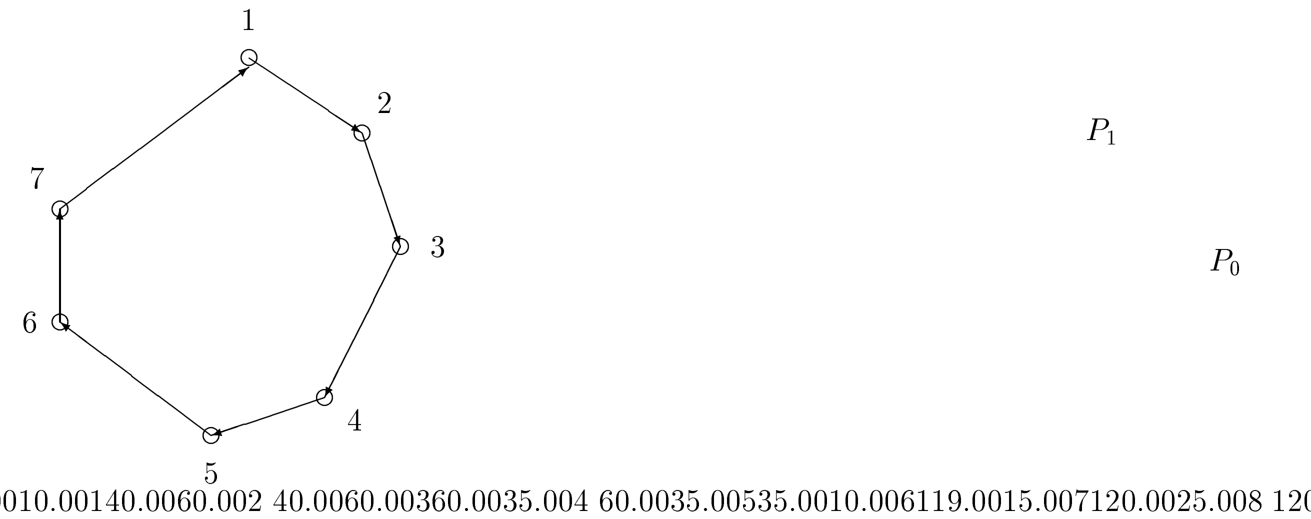
Mnohoúhelník P s n body budeme někdy značit P^n.

Základní operace a problémy nad konvexními mnohoúhelníky jsou tyto (v závorce je vždy uveden čas, kterého dosáhneme)

1. Pro konvexní mnohoúhelník P^n rozhodněte, zda daný bod leží uvnitř či vně P^n (lze v čase $O(\log n)$)

2. Pro danou přímku (úsečku) p najděte $p \cap P^n$ (lze v čase $O(\log n)$)

[1] Ne každý systém množin obsahuje nejmenší prvek. My však máme existenci takové nejmenší množiny zaručenu. Ověření ponecháváme čtenáři.



Obrázek 1: Vyvážená hierarchická reprezentace konvexního sedmiúhelníka

3. Pro P^n a Q^m zjistěte, zda $P^n \cap Q^m = \emptyset$ (lze v čase $O(\log(n+m))$)
4. Pro P^n a Q^m najděte $P^n \cap Q^m$ (lze v čase $O(n+m)$)

Časové složitosti, kterých dosáhneme, budou silně ovlivněny datovou strukturou, která nám bude sloužit pro reprezentaci mnohoúhelníků.

1.2 Definice. Posloupnost $P_0, \dots, P_k$ mnohoúhelníků se nazývá *vyvážená hierarchická reprezentace (VHR)* konvexního mnohoúhelníka P, pokud

- P_0 má nejvýše 4 vrcholy
- $P_k = P$
- P_{i-1} se obdrží z P_i vynecháním vhodných vrcholů, přitom z každých tří po sobě jdoucích je alespoň jeden vynechán a ze čtyř po sobě jdoucích alespoň jeden zůstane zachován.

VHR lze uchovávat jako vyvážený (2,4) strom (viz obrázek 1). Počítáme-li úrovně stromu od nuly, lze i-tou úroveň stromu *VHR* chápat jako posloupnost vrcholů, které dohromady dávají mnohoúhelník P_i. V kořeni stromu na obrázku 1 je mnohoúhelník $P_0 =$ 1-3-5-1. Úsečka 1-3 se v levém podstromu rozvíjí na řetězec 1-2-3, úsečka 3-5 v prostředním podstromu na 3-4-5, a úsečka 5-1 v pravém podstromu na 5-6-7-1.

Každý uzel stromu *VHR* může mít dva nebo tři následníky, kromě uzlů na poslední úrovni, které nemají žádné následníky, a kořene, který má dva, tři nebo čtyři následníky.

VHR je lineární strukturou vůči počtu vrcholů. Ověření je ponecháno na čtenáři.

Pro odvozování časových složitostí jednotlivých algoritmů nás bude zajímat, jakých časů lze dosáhnout při procházení *VHR*.

1.3 Lemma. *Všechny obvyklé operace v (2,4) stromu lze provádět v čase $O(\log n)$; výška takového stromu je $O(\log n)$. VHR mnohoúhelníka P lze získat v čase $O(n)$.*

Důkaz: Obvyklou operací zde rozumíme např. *Insert* a *Delete,* při nichž projdeme strom od kořene k nějakému listu. Taková cesta má délku rovnou výšce stromu. (2,4) strom má nejvýše $O(\log_2 n)$, nejméně $O(\log_4 n)$ úrovní, celkem tedy $O(\log n)$. Udržení vyváženého stromu vyžaduje nejvýše jeden další průchod zpět ke kořeni, což nenaruší logaritmickou složitost.

Konstrukci stromu *VHR* můžeme provádět tak, že jdeme od poslední úrovně (celý mnohoúhelník) směrem nahoru. Musíme vždy projít všechny uzly dané úrovně a vytvářet uzly na nižší úrovni. Výsledný čas bude odpovídat celkovému počtu uzlů ve stromu, a ten je lineární. □

Dospěli jsme do stadia, kdy se můžeme pustit do řešení jednotlivých problémů.

1.4 Věta. *Nechť $P_0, \ldots, P_n$ je VHR konvexního mnohoúhelníka P^n. Problém, zda bod $x \in P^n$, lze rozhodnout v čase $O(\log n)$.*

Důkaz: Analýzou následujícího algoritmu.

Algoritmus 1.1 ROZHODNUTÍ, ZDA BOD LEŽÍ UVNITŘ NEBO VNĚ MNOHOÚHELNÍKA
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

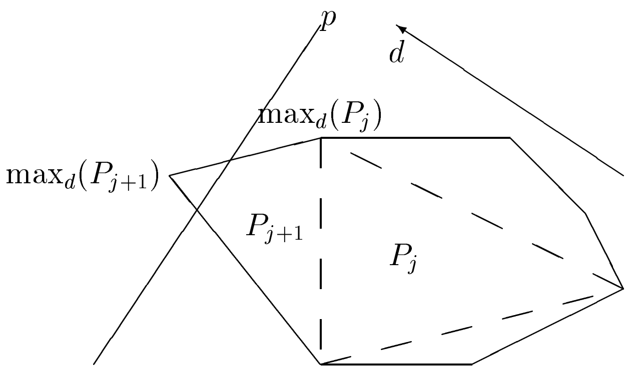
Algoritmus nejprve v cyklu nalezne výseč danou polopřímkami spuštěnými z bodu p na všechny vrcholy P, ve které leží bod x. Pak se ověří, zda je bod x vně nebo uvnitř trojúhelníka, který vznikne ohraničením této výseče hranou odpovídající této výseči.

Bod $p =$ těžiště P_0 najdeme v konstantním čase. Tento bod je vnitřním bodem všech P_i. Bodem p vedeme polopřímky do všech vrcholů P_0. V konstantním čase zjistíme, ve které výseči leží x. V uzlu následující úrovně, který je rozvinutím hrany dané touto výsečí, budeme dále hledat, ve které výseči P_1 leží bod x. Až se tímto způsobem dostaneme k P_k, známe výseč $P_k = P$, ve které leží bod x. Pak zjistíme v konstantním čase, zda bod v nalezené výseči leží uvnitř či vně mnohoúhelníka.

Všechny operace v jednom uzlu lze provést v čase $O(1)$. Časová složitost je tedy úměrná hloubce stromu, což je $O(\log n)$. □

1.5 Věta. *Nechť $P_0, \ldots, P_k$ je VHR(P^n), p je přímka. Průnik $p \cap P$ lze najít v čase $O(\log n)$.*

Důkaz: Budeme opět procházet *VHR* od kořene až k listu. Jestliže při průchodu narazíme na průnik P_i s přímkou p, zjistíme hrany $e_1(i), e_2(i)$ z P_i, které přímka p protíná. Dále přejdeme k $e_1(i+1), e_2(i+1)$ z P_{i+1}, které p protíná. To zabere konstantní čas, protože $e_1(i+1)$ je prvkem „rozvinutí“ hrany $e_1(i)$ při přechodu od P_i k P_{i+1}. Jakmile tedy v průběhu provádění



Obrázek 2: Vztah maximálních bodů a průniku přímky s mnohoúhelníkem

algoritmu zjistíme, že p má s nějakým mnohoúhelníkem P_i neprázdný průnik, najdeme takto nakonec průnik $P^k = P$ s přímkou p, což je kýžený výsledek.

Než se však k takovému P_i dostaneme, musíme projít mnohoúhelníky P_j, které přímka p neprotíná. Pokud tato situace trvá až k $P^k = P$, získáváme informaci, že $p \cap P = \emptyset$.

V případě, že $p \cap P_j = \emptyset$, rozvíjíme vrcholy „maximální“ ve směru kolmém na p. Jsou to vrcholy, jejichž průměty na přímku kolmou na p jsou nejblíže přímce p. Takové mohou být nejvýše dva, viz lemma 1.6. Jestliže p neprotíná P_j, ale protíná P_{j+1}, musí přímka p protínat P_{j+1} v jednom z řetězců, které vzniknou rozvinutím těch hran P_j, které incidují s maximálními body ve směru kolmém na p. Toto si můžeme dovolit tvrdit díky konvexnosti všech mnohoúhelníků P_i. Obrázek 2 ilustruje situaci. Bod 3.1.3 algoritmu 1.2 se odkazuje na tuto úvahu.

Následující lemma postihuje vztah mezi maximálními body v pevném směru po sobě jdoucích mnohoúhelníků ve *VHR*.

1.6 Lemma. *Nechť $d(P_i)$ je množina vrcholů maximálních ve směru d. Pak $|d(P_i)| \leq 2$ a pokud $v \in d(P_{i+1})$, pak existuje $w \in d(P_i)$ tak, že buď $v = w$ nebo mezi v a w leží nejvýše dva vrcholy v seznamu P_{i+1}.*

Důkaz: Kdyby mezi body v a w byly tři nebo více vrcholů, nemohl by se žádný z nich vyskytovat v P_i, jinak by byl maximální místo w. Potom ale tyto tři vrcholy spolu s v tvoří souvislou řadu čtyř vrcholů, které byly při přechodu od P_{i+1} k P_i vynechány, což odporuje definici *VHR*. □

Známe tedy způsob, jak v konstantním čase přejít od maximálních vrcholů v daném směru k maximálním vrcholům v tomto směru pro následující mnohoúhelník ve *VHR*. V následujícím algoritmu se body $d(P_i)$ v bodě 3.1.2 naleznou výše popsaným způsobem.

Přejdeme nyní k samotnému algoritmu.

Algoritmus 1.2 NALEZENÍ PRŮNIKU PŘÍMKY S MNOHOÚHELNÍKEM

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

1. $i := 0$

2. najdeme $d(P_0)$
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` $(p \cap e_1(k)) \cup (p \cap e_2(k))$

Algoritmus projde vždy strom *VHR* a na každé úrovni stromu (jeden průchod cyklem) provede konstantní operaci v obou cyklech algoritmu, tedy výsledný čas je $O(\log n)$. □

1.7 Důsledek. *Problém nalezení $P \cap s$, kde s je úsečka, lze řešit v čase $O(\log n)$.*

Důkaz: Aplikujeme algoritmus 1.2 na přímku, na které úsečka s leží, a pak v konstantním čase zjistíme průnik výsledku algoritmu s úsečkou s. □

1.8 Poznámka. Algoritmy 1.1 a 1.2 jsou typickými příklady metody *vyhledávání ve stromu*. Oba postupují od kořene a na základě jistého kriteria se na každé úrovni stromu rozhodují, které uzly budou „zajímavé“ na příští úrovni. Takto se dostanou až k listům, kde se dá již zjistit konečný výsledek.

Následující dvě úlohy zkoumají průnik dvou konvexních mnohoúhelníků. Uvidíme, že pro pouhé zjištění, zda tento průnik je prázdný či ne, lze nalézt při použití *VHR* rychlejší algoritmus, než pro nalezení celého průniku.

1.9 Věta. *Nechť P_0 až P_k je VHR mnohoúhelníka P^m a Q_0 až Q_l je VHR mnohoúhelníka Q^n. V čase $O(\log(m+n))$ lze rozhodnout, zda $Q \cap P = \emptyset$ či nikoli.*

Důkaz: Označme P_L a P_R tzv. levý a pravý monotónní mnohoúhelníkový řetězec (řetězce vzniknou roztržením celého mnohoúhelníkového řetězce na dva podřetězce v bodech s maximální a minimální y-ovou souřadnicí – viz obrázek 3). Je zřejmé, že $P_L \cap P_R = P$ a $Q_L \cap Q_R = Q$. Dále platí:

$$P \cap Q = \emptyset \iff P_L \cap Q_R = \emptyset \vee Q_L \cap P_R = \emptyset \tag{1}$$

3.005.001155.005.002 155.0050.0033.0050.004 80.0050.00559.0042.006 59.0042.00759.0019.008 59.0019.

Obrázek 3: Pravý a levý monotónní řetězec mnohoúhelníka

Bereme tedy v úvahu dvě dvojice monotónních řetězců. Popíšeme nyní, jak hledat průnik jedné takové dvojice, např. P_L a Q_R, druhý případ je symetrický.

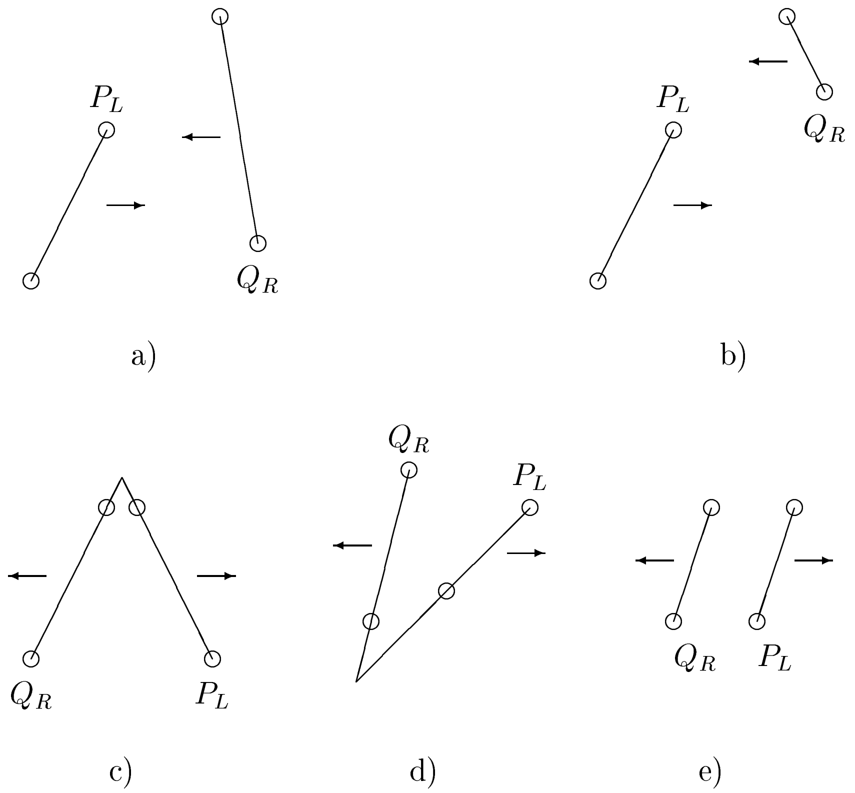
Na řetězcích P_L a Q_R budeme postupovat metodou půlení intervalu (zde dělicí „body“ intervalů budou jednotlivé hrany řetězců). Na počátku bereme v úvahu celé řetězce, v každém kroku se pak rozhodneme, zda nás zajímá oblast nad nebo pod aktuální hranou v obou řetězcích. Z takových dvou oblastí pak vybereme hrany uprostřed (půlící hrany), a dále postupujeme stejně. Jestliže dojdeme k nedělitelnému intervalu a na cestě jsme nenarazili na případ, který nám zaručuje existenci průniku P_L a Q_R – viz obr. 4a), potom průnik neexistuje. Tento postup nám zaručuje logaritmický čas.

Diskusi vzájemných poloh aktuálních hran popíšeme za pomoci obrázku 4. Nastane-li případ a), pak řetězce P_L a Q_R mají neprázdný průnik. V případě b) nás u P_L bude dále zajímat oblast nad aktuální hranou. V případě, kdy aktuální hrany jsou „zády k sobě“ – případy c), d), e) na obrázku, diskutujeme přímky, na kterých leží aktuální hrany. Protože tyto přímky ohraničují poloroviny, ve kterých leží vždy celý odpovídající mnohoúhelník (díky jejich konvexnosti), může nastat průnik pouze tam, kde se tyto přímky protínají. V případě c) nás tedy budou dále zajímat horní oblasti pro oba řetězce, v případě d) spodní oblasti, a v případě e) už průnik nemůže existovat, protože přímky jsou rovnoběžné.

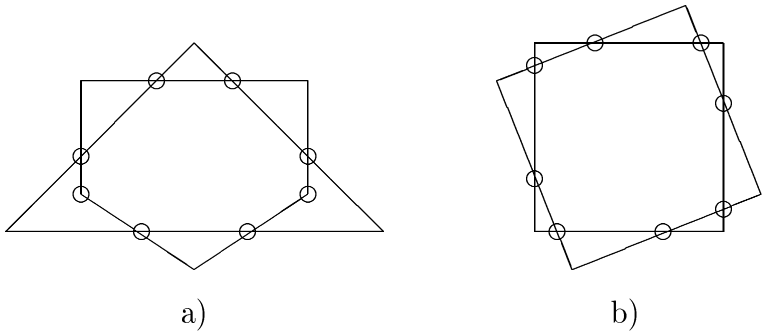
Celkem v logaritmickém čase zjistíme, zda existuje průnik P_L a Q_R, a totéž se dá zcela analogicky zjistit pro P_R a Q_L. Díky vztahu 1 tak známe v logaritmickém čase odpověď na otázku, zda existuje průnik mnohoúhelníků P a Q. $\square$

1.10 Poznámka. Algoritmus popsaný v předchozím důkaze používá metody tzv. *binární lokalizace*. Český termín půlení intervalu je v podstatě ekvivalentem. Tyto algoritmy se snaží lokalizovat daný jev tak, že rozpůlí interval, ve kterém probíhá hledání jevu, a podle nějakého kriteria mohou jeden ze dvou takto vzniklých intervalů vyloučit. Hledání pak rekurzivně probíhá na tomto menším intervalu.

Musí však existovat nějaká „zarážka“, která na jisté úrovni rekurze algoritmus zastaví. V našem případě je to nedělitelnost intervalu. Hledáme-li touto metodou iracionální kořen polynomu, musíme mít jako zarážku jistou „přesnost“, nebo-li velikost intervalu, který už nebudeme dále půlit, pouze jeho střed prohlásíme za kořen.



Obrázek 4: Diskuse vzájemných poloh aktuálních hran monotónních řetězců



Obrázek 5: Průniky délky $O(m + n)$ a) troj- a pětiúhelníka b) dvou čtyřúhelníků

1.11 Věta. *Nechť P^m, Q^n jsou konvexní mnohoúhelníky. Pak $P \cap Q$ lze zjistit v čase $O(m+n)$.*

Důkaz: Nejdříve získáme bod p uvnitř P (např. těžiště libovolných tří vrcholů P). Z bodů P i Q vytvoříme posloupnosti bodů $p_1, \dots, p_m$ a $q_1, \dots, q_n$ takové, že vždy sousední body jsou spojeny hranou. Označíme pro $i = 1 \dots n$ výseče ohraničené polopřímkami $\overline{pp_i}$ a $\overline{pp_{i+1}}$ ($p_{n+1} = p_1$) jako s_i. V lineárním čase se dá zjistit, ve které výseči s_i se nachází bod q_1. V čase $O(k + 1)$ určíme průnik úsečky $L(q_1, q_2)$ s P, případně její polohu (uvnitř – vně). Zde k je počet polopřímek $\overline{pp_i}$ proťatých úsečkou $L(q_1, q_2)$. Zřejmě je $k < m$. Takto pokračujeme pro všechny body q_j. Po průchodu získáme buď průniky hran, nebo informaci, 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 $O(m + n)$. □

Lepšího času než $O(m+n)$ nelze obecně dosáhnout, protože výsledek má v nejhorším případě právě tuto délku. Průnikem troj- a pětiúhelníka na obrázku 5a) je totiž právě osm bodů. Dva čtverce na obrázku 5b) mají rovněž průnik délky osm.

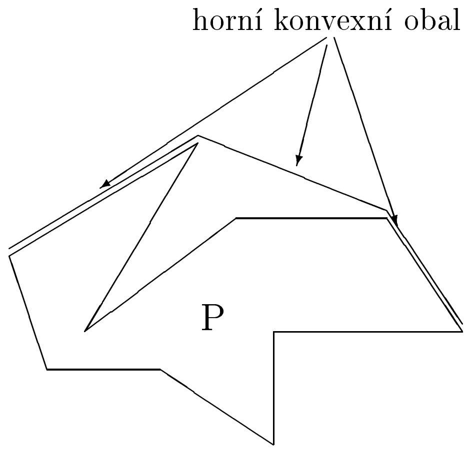
1.12 Poznámka. Postup uvedený v důkazu věty 1.11 nachází průsečíky hran mnohoúhelníků v takovém pořadí, které produkuje mnohoúhelník ohraničující průnik vnitřků mnohoúhelníků P a Q. Chceme-li tedy najít průnik vnitřků P a Q, nemusíme body dále třídit. Pro ověření si stačí uvědomit, že uvedený postup prochází hrany Q postupně podle sousednosti. Úseky hran z Q, které ohraničují průnik, jdou za sebou ve stejném pořadí, pouze se mezi ně případně vkládají úseky hran z P. K pochopení zde opět pomůže obrázek 5.

1.2 Konvexní obal bodů v rovině

Nalezení konvexního obalu konečné množiny bodů v rovině chápeme jako zadání hraničních bodů konvexního obalu seřazených v jistém směru (např. ve směru hodinových ručiček).

Uvedeme postupně několik algoritmů, u kterých určíme nejprve asymptoticky nejhorší časy. V části 1.4 tyto algoritmy porovnáme z hlediska času očekávaného.

1.13 Poznámka. Konvexní obal množiny S budeme značit $BCH(S)$ (z angl. *Boundary Convex Hull*). Vnitřek konvexního obalu budeme značit $CH(S)$ (*Convex Hull*).



Obrázek 6: Horní konvexní obal

1.14 Věta. *Nechť $S \subset \mathbf{R}^2$ je konečná množina bodů v rovině, P^n jednoduchý mnohoúhelník, jehož vrcholy jsou právě všechny body z S. Konvexní obal $BCH(S)$ lze spočítat v lineárním čase $O(|S|)$ ([Preparata-85], str. 166–171).*

Důkaz: V lineárním čase najdeme vrcholy $p_l, p_r \in S$ s minimální a maximální x-ovou souřadnicí. Úsečka $L(p_r, p_l)$ leží v konvexním obalu. Díky jednoduchosti P lze úlohu řešit zvlášť pro tzv. horní a dolní konvexní obal (viz obr. 6 a následující odstavec).

Konvexní obal je vlastně konvexní mnohoúhelník. Body p_r a p_l budou zcela jistě jeho vrcholy. Tyto dva vrcholy roztrhnou konvexní obal na dva řetězce úseček, vedoucí z p_r do p_l, každý v jiné polorovině podle přímky $\overline{p_r p_l}$. Řetězec nad touto přímkou nazveme *horní konvexní obal*, pod ní *dolní konvexní obal* bodů množiny S. Problém nalezení dolního konvexního obalu je duální k nalezení horního konvexního obalu a níže uvedený algoritmus se dá snadno upravit tak, aby počítal dolní konvexní obal.

Označení „napravo“ resp. „nalevo“ od $\overline{pq}$, která se v algoritmu hojně vyskytují, znamenají lokalizaci v pravé resp. levé polorovině dané přímkou $\overline{pq}$. Orientace je taková, jako bychom se „dívali“ od bodu p směrem k bodu q. Prakticky se tento test pro bod x provede výpočtem determinantu matice ze sloupců $\overline{pq}, \overline{px}$. Tím získáme orientovaný obsah příslušného rovnoběžníka, a podle znaménka +/- je bod nalevo/napravo.

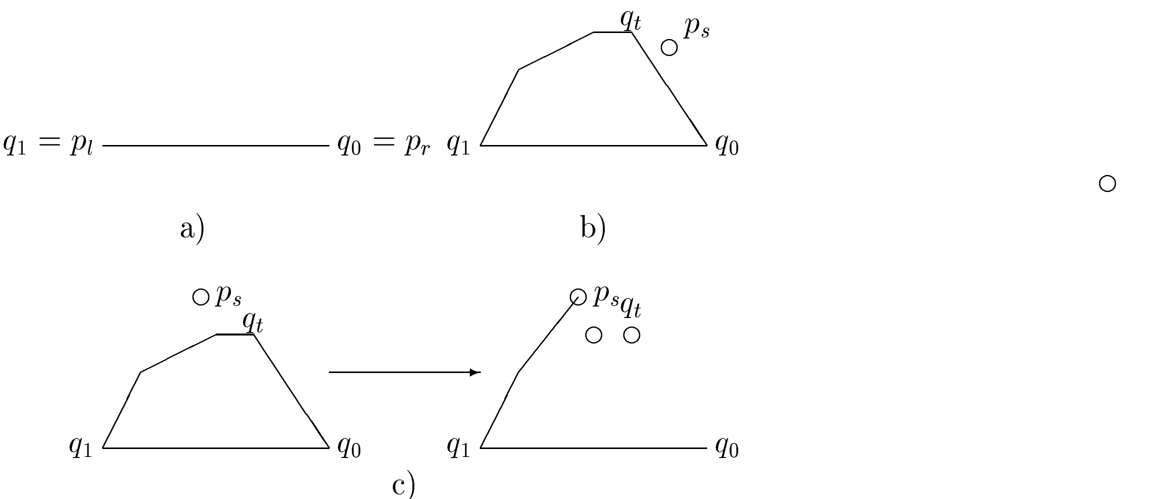
Algoritmus 1.3 NALEZENÍ HORNÍHO KONVEXNÍHO OBALU JEDNODUCHÉHO MNOHOÚHELNÍKA

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

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

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

1. najdi body p_r a p_l



10.0050.00115.0045.002 15.0045.00324.0045.004 24.0045.00529.0055.006

Obrázek 7: Konstrukce horního konvexního obalu jednoduchého mnohoúhelníka

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 $\overline{p_r p_l}$ – viz obr. 7a)
   - 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 – viz obr. 7b)
     - 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 – viz obr. 7c)
     - 6.3.1. odstraň q_t z vrcholu zásobníku
   - 6.4. přidej p_s na vrchol zásobníku

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

Diskusí poloh bodu p_s vůči bodům zásobníku Q (obrázek 7) si ověříme následující invarianty, které platí vždy na začátku cyklu 6:

- $q_0 = p_r$; $q_1 = p_l$; $t \geq 2$; $q_t = p_s$
- $s \leq n$

- $q_0, \ldots, q_t, q_0$ je konvexní mnohoúhelník
- horní konvexní obal S je posloupnost vybraná z $q_0, \ldots, q_t, p_{s+1}, \ldots, p_n$

První dva body jsou zřejmé. Zachování konvexnosti ve třetím bodě se ukáže snadno z obrázku 7, když si uvědomíme, v jakém pořadí nastává přidávání a ubírání bodů na vrcholu zásobníku. K ověření čtvrtého bodu vede poněkud složitější úvaha.

V situacích a) a b) na obrázku 7 nenastane žádná komplikace. Nastane-li situace c) – bod p_s nalevo od $\overline{q_{t-1}q_t}$, může být bod p_s nalevo (jako je tomu na obrázku) nebo napravo od $\overline{q_0q_t}$. V druhém případě je jasné, že bod q_t a ani žádný jiný bod vyloučený ze zásobníku v cyklu 6.3 nemohou být už vrcholy konvexního obalu. Pro první případ si musíme uvědomit, že z bodu p_s vede lomená čára jako součást mnohoúhelníka P^n do bodu q_0. Ta musí protínat přímku $\overline{q_{t-1}q_t}$, protože bod p_s je od ní vlevo a bod q_0 vpravo (jinak by q_t nebyl přidán do zásobníku za q_{t-1}). Protože je mnohoúhelník P^n jednoduchý a jeho vrcholy jsou zadány podle směru hodinových ručiček, bude tato lomená čára protínat polopřímku $\overline{q_{t-1}q_t}$ až za bodem q_t. Bod q_t se tak octne uvnitř konvexního obalu, a nemůže už být na jeho hranici. To stejné platí i pro ostatní body odstraněné ze zásobníku v cyklu 6.3.

K odvození času běhu algoritmu vede úvaha zkoumající zacházení s jednotlivými vrcholy mnohoúhelníka. Po nalezení bodů p_r a p_l, které je lineární záležitostí, je každý vrchol P^n konfrontován se zásobníkem Q právě jednou, nepočítáme-li cyklus 6.3. Tento cyklus bude však mít maximálně n iterací v průběhu celého algoritmu, protože je zde odstraňován vrchol zásobníku, a to se každému bodu z P^n může stát nejvíce jednou. Protože příslušnost bodu do jedné z polorovin určených danou přímkou lze testovat v konstantním čase, algoritmus 1.3 pracuje lineárně.

□

1.15 Věta. *Nechť $S \subset \mathbf{R}^2$ je konečná množina o n bodech. Pak $BCH(S)$ lze spočítat v čase $O(n \log n)$.*

Důkaz: Problém řeší následující algoritmus.

Algoritmus 1.4

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

Lexikograficky uspořádáme S, tj. $p < q$ jestliže $x(p) < x(q)$ nebo $x(p) = x(q)$ a $y(p) < y(q)$. Nechť $p_1, \ldots, p_n$ je výsledná posloupnost. Na ni aplikujeme algoritmus 1.3.

I když takto zadaná posloupnost bodů $p_1, \ldots, p_n$ není mnohoúhelníkem, lze ji použít jako vstup výše uvedeného algoritmu, protože ten vyžaduje pouze to, aby se žádné dvě nesousední hrany $p_{i-1}p_i$ pro $i = 1, \ldots, n$ neprotínaly a aby z každého bodu p_i pro $i < n$ existovala lomená čára tvořená posloupností hran vybranou z P^n do bodu q_0. V našem případě $q_0 = p_n$, a tedy obě podmínky jsou splněny.

Výsledný čas je $O(n \log n) + O(n)$. □

Vstup pro algoritmus 1.3 se dá získat z obecné množiny bodů ještě jiným způsobem. Celý algoritmus je nazván *Graham's Scan*.

Algoritmus 1.5 Graham's Scan

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

Vybereme libovolný vnitřní bod konvexního obalu (nemusí být prvkem S) a spojíme ho se všemi body S. Setřídíme body podle velikosti úhlů jejich spojnic s vybraným bodem a takovou posloupnost pošleme na vstup algoritmu 1.3.

Posloupnost bodů setříděných podle jejich úhlů už jistě dává jednoduchý mnohoúhelník a je tedy použitelná pro algoritmus 1.3. Graham's Scan pracuje díky třídění bodů opět v čase $O(n \log n)$.

1.16 Poznámka. Následující tři algoritmy pracují tak, že počítají konvexní obal nějakých podmnožin vstupní množiny, a pak s pomocí těchto částečných výsledků produkují celkový konvexní obal. První algoritmus se liší od zbylých dvou pouze v tom, jakým způsobem dělí vstupní množinu.

Tato metoda se nazývá *rozděl a panuj* a je pro ni charakteristické (i když ne nutné) rekurzivní volání procedur. Typický algoritmus *rozděl a panuj* rozdělí vstupní soubor na několik částí (mohou být buď srovnatelně velké nebo jsou záměrně některé větší a jiné menší), a pro ně zavolá rekurzivně sám sebe (část *rozděl*). Jakmile jsou známy výsledky těchto volání, aplikuje se na ně nějaká efektivní metoda, která produkuje celkový výsledek (část *panuj*).

Metoda *rozděl a panuj* se často používá v praxi při analýze problémů (může to být analýza hospodaření podniku nebo přepis algoritmu do procedurálního programovacího jazyka). Celá úloha se dělí na podproblémy, až vznikne jistá hierarchie částečných problémů. Jejich spojování do výsledků je pak jasnější a přehlednější.

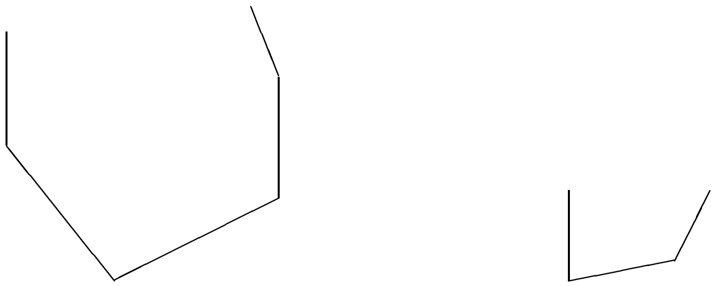
Po rozdělení vstupní množiny a spočítání konvexních obalů takto rozdělených částí budeme mít za úkol spočítat konvexní obal dvou konvexních obalů (tedy konvexních mnohoúhelníků), nebo bodu a mnohoúhelníka. Následující lemma řeší oba problémy, v případě dvou mnohoúhelníků však pouze pro případ, kdy jsou jejich vnitřky disjunktní. Tuto vlastnost v našem algoritmu zaručíme.

1.17 Lemma. *$BCH(P \cup Q)$ disjunktních konvexních mnohoúhelníků P^m a Q^n lze najít v čase $O(m+n)$.*

Důkaz: V čase $O(m)$ najdeme hranu v P nebo Q tak, že P a Q padnou do různých polorovin. Tím umožníme nalezení dvou nejbližších bodů z $p_0 \in P$ a $q_0 \in Q$.

V prvním kroku postupujeme po P proti směru hodinových ručiček, dokud se zmenšuje úhel s vybranou hranou v Q. Označme získaný bod p_1. Tak získáme tečnu z q_0 na P. V dalším kroku zaměníme P a Q, postupujeme po Q ve směru hodinových ručiček, a získáme tečnu z p_1 na Q. Po konečném počtu kroků dostaneme jednu z tečen P a Q – viz. obrázek 8. Obrácením orientací získáme druhou tečnu. Při celém průběhu algoritmu procházíme každým vrcholem nejvýše jednou. $\square$

1.18 Lemma. *$VHR(BCH(P \cup Q))$ dvou disjunktních konvexních mnohoúhelníků P^m a Q^n, známe-li jejich VHR a tečny, lze spočíst v čase $O(\log(m+n))$.*



95.0027.00153.0051.002 53.0051.00338.0056.004 38.0056.00521.0048.006 95.0027.007103.00

Obrázek 8: Tečny dvou konvexních disjunktních mnohoúhelníků

Důkaz: Musíme vyloučit řetězce mezi body dotyku. Můžeme si to představit tak, že získáme v každém ze stromů *VHR* pro oba mnohoúhelníky dva ukazatele na listy, z nichž jeden znamená začátek a druhý konec řetězce, který má být odstraněn ve výsledném mnohoúhelníku (body dotyku tečen). Odstraníme nyní tyto listy v obou stromech, a ve vyšších úrovních stromu dále ty uzly, které vedou pouze na listy, jež mají být odstraněny. Získáme tak dva „hybridní" stromy (viz obr. 9a), které nemusí splňovat podmínky *VHR*. Z nich musíme stvořit jediný *VHR* strom.

Konstrukci začneme na nejnižší úrovni. Jeden řetězec listů vložíme mezi řetězec zbylých listů ve druhém stromě, a to na místo, odkud byly předtím listy odstraněny. Tím na této úrovni končíme.

Na každé vyšší úrovni je třeba zajistit, aby platily podmínky *VHR*. Jde především o zajištění toho, aby každý uzel měl dva až čtyři potomky. Vezmeme v úvahu seřazené uzly této úrovně obou hybridních stromů. Budou nás nyní zajímat pouze krajní uzly obou těchto posloupností. Někdy dva z těchto uzlů splynou v jeden, jindy je třeba uzel rozdělit na dva (viz obr. 9b,c). Vnitřní uzly těchto posloupností však mohou být zachovány. Tato vlastnost, zjednodušeně vysvětlitelná tím, že jsme vzali vždy souvislý řetězec listů a jejich předchůdců v původním *VHR*, nám zaručí konstantní čas výpočtu na každé úrovni konstrukce nového stromu. Nově zkonstruovaný strom už má všechny vlastnosti *VHR*, a počet jeho vrcholů je maximálně $m+n$. Znamená to tedy, že konstrukce *VHR* zabere $O(\log(m+n))$ času. □

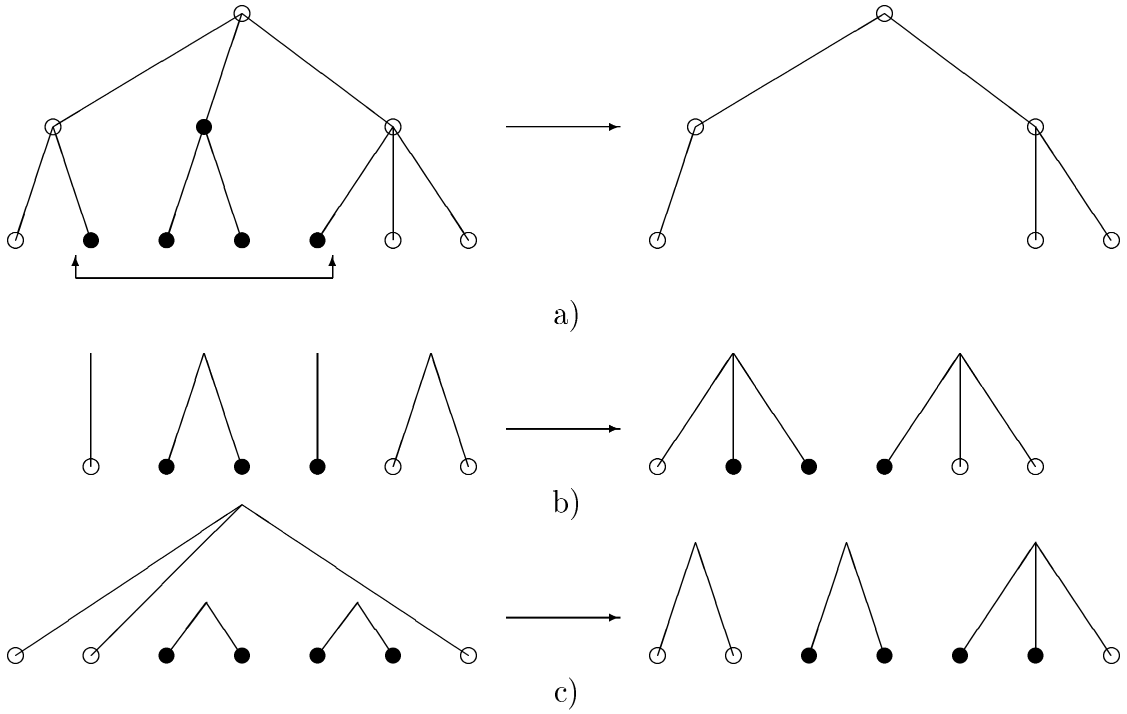
1.19 Věta. *Nechť* $S \subset \mathbf{R}^2$ *je množina o* n *bodech,* $p \in \mathbf{R}^2$ *je bod. Je-li dána* $VHR(BCH(S))$*, pak*

$$VHR(BCH(S \cup \{p\}))$$

spočteme v čase $O(\log n)$.

Důkaz: Podle věty 1.4 zjistíme v čase $O(\log n)$, zda $p \in BCH(S)$. Pokud ano, je výsledkem $BCH(S)$. Pokud ne, najdeme v čase $O(\log n)$ body dotyku tečen spuštěných z bodu p na $BCH(S)$:

Body dotyku tečen z p na P_0 lze získat v konstantním čase. Jdeme-li od P_i k P_{i+1}, musí ležet nový bod dotyku v rozvinutí některé ze dvou hran, které sousedí s původním bodem dotyku, pokud jím nezůstane on sám. Každý krok je konstantní, proto celkově máme čas $O(\log n)$.



Obrázek 9: Postup při spojování *VHR* disjunktních mnohoúhelníků; a) hybridní strom, b) spojení uzlů, c) rozdělení uzlu

Tyto body dotyku jsou vrcholy mnohoúhelníka $BCH(S)$. Aktualizace VHR pak proběhne v čase $O(\log n)$ podle lemmatu 1.18, dosadíme-li za mnohoúhelník Q^n bod p. □

Můžeme přistoupit k prvnímu algoritmu *rozděl a panuj*. Ten ze vstupní množiny vybere jeden bod, spočítá konvexní obal zbylých $n-1$ bodů, a pak tento jeden bod přidá. Rekurze se zde dá snadno nahradit iterativním postupem. Výhodou tohoto přístupu je jeho dynamičnost – po zpracování libovolné podmnožiny elementů ze vstupu známe pro ně výsledek.

Algoritmus 1.6
Vstup: konečná množina S bodů v rovině
Výstup: vrcholy konvexního obalu S, setříděné ve směru hodinových ručiček
Algoritmus pro tvorbu konvexního obalu lze získat jako důsledek věty 1.19 tak, že body množiny S bereme ze vstupu po jednom a konvexní obal konstruujeme iterativně přidáváním těchto bodů. Výsledný čas je opět $O(n \log n)$).

Druhý algoritmus rozděluje vstupní množinu na dvě stejně velké podmnožiny, pro které rekurzivně zavolá sám sebe. Výsledky pak spojí v konvexní obal pomocí lemmatu 1.17. Aby toto lemma bylo použitelné, musíme zaručit disjunktnost částečných výsledků. To provedeme tak, že obě podmnožiny budou vždy celé ležet v různých polorovinách podle nějaké přímky rovnoběžné s osou y. Za tímto účelem body na začátku algoritmu třídíme podle x-ové souřadnice.

Algoritmus 1.7 ALGORITMUS PRO ZÍSKÁNÍ KONVEXNÍHO OBALU METODOU *Rozděl a panuj*
Vstup: konečná množina S bodů v rovině
Výstup: vrcholy konvexního obalu S, setříděné ve směru hodinových ručiček

1. Uspořádáme body podle x-ových souřadnic (pro jednoduchost předpokládejme, že jsou navzájem různé).

2. Rozdělíme body na dvě stejně velké poloviny podle x-ové souřadnice, pro něž rekurzivně zavoláme tento algoritmus. Zbývá-li nyní jediný bod, nepoužijeme další rekurzivní volání, ale pošleme tento bod na výstup jako konvexní obal sama sebe („zarážka" rekurze).

3. Použijeme lineární algoritmus pro nalezení konvexního obalu dvou disjunktních konvexních mnohoúhelníků (viz lemma 1.17) aplikovaný na konvexní obaly obou polovin původní množiny, které jsme dostali jako výsledky rekurzivních volání v předchozím kroku.

Počáteční uspořádání spotřebuje $O(n \log n)$ času.

Čas rekurzivní části:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$$
$$T(n/2) = T(\lfloor n/4 \rfloor) + T(\lceil n/4 \rceil) + O(n/2)$$

atd. Celkem to je $O(n \log n)$.

1.20 Poznámka. Našli jsme doposud čtyři algoritmy, které pracují v čase $O(n \log n)$. Můžeme si tedy mezi nimi vybrat metodu odpovídající nejlépe konkrétní implementaci. Lepšího času než $O(n \log n)$ však už nelze dosáhnout, protože dostaneme-li na vstupu množinu bodů náhodně rozmístěných na kružnici, nalezení jejich konvexního obalu znamená právě jejich setřídění podle úhlů na této kružnici. Proto nenajdeme algoritmus pracující v lepším asymptotickém čase.

Tato úvaha by mohla znamenat, že jsme našli optimální algoritmy pro nalezení konvexního obalu. Uvedeme však ještě další dva algoritmy, jejichž obecná časová složitost bude v jednom případě i horší než $O(n \log n)$, ale očekávaná složitost pro jisté distribuce bodů v rovině bude lepší. Tento rozbor provedeme v části 1.4.

Následující algoritmus kombinuje metodu Graham's Scan a přístup *rozděl a panuj*. Budeme nyní rozdělovat vstupní množinu na podmnožiny bez ohledu na rozdělení bodů podle souřadnic. Konvexní obal těchto podmnožin se bude počítat rekurzivně. Pro výsledné mnohoúhelníky s potenciálně neprázdným průnikem spočítáme jejich konvexní obal metodou Graham's Scan. Využijeme k tomu následující lemma.

1.21 Lemma. *Máme-li k dispozici dvě setříděné posloupnosti bodů, setřídíme je všechny v lineárním čase.*

Důkaz: Udržujeme dva ukazatele dovnitř posloupností, do každé jeden. Na počátku ukazují oba na nejmenší prvky obou posloupností. Postupujeme tak, že vybereme vždy menší ze dvou prvků, na které ukazujeme, a odpovídající ukazatel posuneme na další prvek v jeho posloupnosti. □

A nyní již samotný algoritmus.

Algoritmus 1.8 MERGEHULL
Vstup: konečná množina S bodů v rovině
Výstup: vrcholy konvexního obalu S, setříděné ve směru hodinových ručiček

1. Pro jistý malý počet bodů na vstupu spočítáme konvexní obal přímo (např. jediný bod je sám svým konvexním obalem)

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

3. Máme nyní dva konvexní mnohoúhelníky, které se mohou překrývat. Vybereme libovolný vnitřní bod p jednoho z nich. Ten bude jistě uvnitř jejich konvexního obalu.

   Je-li p zároveň uvnitř druhého mnohoúhelníka, jsou vrcholy obou mnohoúhelníků jak spolu sousedí monotónními posloupnostmi vzhledem k úhlům, které svírají jejich spojnice s bodem p.

   Není-li p vnitřní bod druhého mnohoúhelníka, spustíme naň z bodu p tečny. Body dotyku roztrhnou mnohoúhelník na dva řetězce. Ten bližší bodu p bude jistě celý uvnitř výsledného konvexního obalu a nebudeme jej proto brát v úvahu. Vnější řetězec je opět posloupností monotónní vzhledem k úhlům spojnic jednotlivých bodů s bodem p.

   Dostáváme tak dvě posloupnosti bodů setříděné podle jejich úhlů. V lineárním čase z nich umíme sestrojit jedinou monotónní posloupnost, kterou pak pošleme na vstup algoritmu 1.3.

MergeHull pracuje opět v čase $O(n \log n)$, což se odvodí ekvivalentně odvození času algoritmu 1.7. Na rozdíl od něho však MergeHull netřídí na začátku body, a tak očekávaný čas bude jistě lepší. Linearita jednoho volání MergeHull (bez rekurzivní části) je nejhorší případ, kdy obě podmnožiny budou mít lineárně veliké konvexní obaly. Diskuse očekávaných časů bude provedena v části 1.4.

Algoritmus *Jarvis's March* postupuje po vrcholech konvexního obalu. Najde nejprve jeden vrchol, který má některou souřadnici extrémní a bude tedy jistě vrcholem konvexního obalu. Tento vrchol označíme za aktuální a hledáme jeho souseda v BCH.

Dále postupujeme tak, že „natáčíme" polopřímku vycházející z aktuálního vrcholu do ostatních bodů, až najdeme přímku, v jejíž jedné polorovině leží všechny ostatní body vstupní množiny. Znamená to vlastně najít bod, pro nějž přímka spuštěná z aktuálního vrcholu na něho má extrémní úhel. Tento bod je rovněž vrcholem BCH. Prohlásíme jej za aktuální vrchol. V hledání vrcholů takto pokračujeme, dokud se nevrátíme do úplně prvního vrcholu.

Algoritmus 1.9 JARVIS'S MARCH – ALGORITMUS PRO ZÍSKÁNÍ KONVEXNÍHO OBALU MNOŽINY BODŮ V ROVINĚ

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

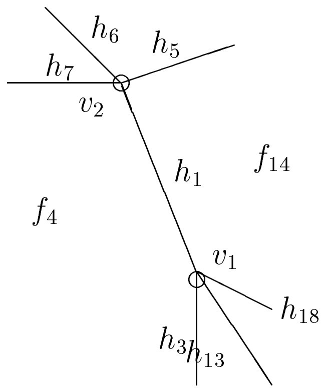
1. vybereme bod s nejmenší y-ovou souřadnicí za aktuální vrchol
2. `repeat`
   - 2.1. spojíme aktuální vrchol s ostatními body a za další vrchol BCH vybereme nejbližší z těch, které spojuje s výchozím nejmenší úhel počítaný mezi jejich spojnicí a poslední přidanou hranou (máme-li zatím pouze první bod,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

První krok algoritmu je lineární. V každém kroku cyklu je identifikována právě jedna hrana konvexního obalu. Tento krok zjišťuje extrém mezi nejvýše n vrcholy, proto zabere $O(n)$ času. Jarvis's March pracuje tedy v čase $O(n) + O(nH) = O(nH)$, kde H je počet hran výsledného BCH.

Počet hran výsledného konvexního obalu může být až $O(n)$ – opět příklad se všemi body na kružnici. Proto Jarvis's March pracuje v kvadratickém čase $O(n^2)$. Tento algoritmus však, na rozdíl od ostatních uvedených v této části, bude většinou pracovat v lepším čase, než který jsme odhadli pro nejhorší případ. Více k tomu podkapitola 1.4.

1.3 Konvexní obal množiny bodů ve vyšších dimenzích

1.22 Poznámka. Předpokládáme-li, že žádné tři body z množiny bodů v rovině neleží na přímce, je konvexním obalem těchto bodů konvexní mnohoúhelník. Neleží-li žádné čtyři body v rovině ze vstupní množiny bodů v prostoru, je jejich konvexním obalem konvexní simpliciální mnohostěn. Obecně, konvexním obalem množiny bodů v $\mathbf{R}^d$ je *d-dimenzionální simpliciální mnohostěn*, jehož stěny jsou v prostoru $\mathbf{R}^{d-1}$ simplexy. Můžeme je tedy nazvat *simpliciální nadstěny*. To ovšem platí za předpokladu, že žádných $d+1$ bodů vstupní množiny neleží v $(d-1)$-dimenzionální nadrovině.



hrana	*V1*	*V2*	*F1*	*F2*	*P1*	*P2*
1:	1	2	4	14	3	5
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Obrázek 10: Jedna položka dvojitého spojového seznamu

Metoda *Graham's Scan* z algoritmu 1.5 není ve vícerozměrném prostoru účinná.

Popíšeme si nyní zobecnění principu *Jarvis's March* ve třech dimenzích (algoritmus 1.9). Budeme se snažit pohybovat po povrchu výsledného simpliciálního mnohostěnu a postupně zařazovat jeho vrcholy, až zjistíme, že jsme se dostali na začátek našeho hledání. Metoda se však dá použít obecně v d-dimenzionálním prostoru.

Algoritmus 1.10 Gift-wrapping method

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 *Jarvis's March* 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.

Celkový čas algoritmu je opět $O(nH)$, kde H je zde počet stěn výsledného BCH. Všechny operace v prvním kroku jsme schopni vykonat v lineárním čase, cyklus proběhne H-krát, a bude hledat extrém mezi nejvýše n vrcholy. Tento časový odhad je platný pro všechny dimenze, jen H zde vystupuje jako počet simpliciálních nadstěnů, které tvoří výsledný konvexní obal.

Odvodíme nyní časovou složitost pro tři dimenze.

1.23 Věta. *Algoritmus Gift-wrapping method pracuje ve třech dimenzích v čase* $O(n^2)$.

Důkaz: Potřebujeme dokázat, že nejvyšší možný počet stěn je lineární. Musíme počítat s nejhorším případem, tedy že všechny body zůstanou vrcholy konvexního obalu (např. když všechny leží na povrchu koule). Víme, že hran je v mnohostěnu právě $\frac{3}{2}$-krát tolik, co stěn. Bereme-li totiž jednotlivé stěny a dáváme do posloupnosti jejich hrany (vždy tři), dostaneme posloupnost hran, v níž bude každá hrana právě dvakrát. Dosadíme-li do Eulerova vzorce

$$h + 2 = s + v$$

kde h je počet hran, s je počet stěn a v počet vrcholů, dostaneme rovnost

$$s = 2v - 4$$

Není-li výsledný mnohostěn simpliciální, lze každou stěnu, která není trojúhelníkem, na trojúhelníky rozdělit. Pro takto zvětšený počet stěn platí vztah z předchozího odstavce, je tedy pro nás horním odhadem.

Celkem dostáváme složitost $O(n^2)$. □

Paměťové struktury použitelné pro algoritmus Gift-wrapping method stojí za poznámku.

1.24 Poznámka. Pro mnohostěny a planární grafy se dá použít struktura *dvojitého spojového seznamu* (*double connected edge list, DCEL*, viz také [Preparata-85], str. 15–17). Pro každou hranu máme šest atributů *V1, V2, F1, F2, P1, P2*. *V1 a V2* jsou ukazatele na vrcholy, které jsou krajními body hrany. *F1 a F2* ukazují na stěny, které incidují s hranou, nejlépe v dohodnutém pořadí (např. *F1* je ta stěna, která se jeví nalevo od hrany, dívám-li se ve směru od *V1* k *V2*). *P1* je pak odkaz na tu hranu, která má s danou hranou společnou stěnu *F1* a vrchol *V1*, resp. stěnu *F2* a vrchol *V2* pro ukazatel *P2*. Budou to vlastně hrany „sousedící“ v krajních bodech dané hrany s touto hranou ve směru proti otáčení hodinových ručiček (viz obr. 10).

Zároveň s výše popsaným seznamem hran udržujeme seznam vrcholů a stěn, pro něž udržujeme jediný atribut, a to některou incidentní hranu. Uvnitř seznamu hran již můžeme velmi efektivně vyhledat všechny hrany na okraji jisté stěny, všechny hrany vycházející z jistého vrcholu, všechny vrcholy na okraji jisté stěny atd. Všechna tato vyhledávání lze provádět přímo, a výsledný čas je úměrný velikosti výsledku.

V případě algoritmu Gift-wrapping method je užitečné mít zvláštní seznam neúplných hran. Vyhledání takové hrany pak bude probíhat v konstantním čase, stejně jako ověření, zda takové hrany ještě existují, což odpovídá testu na konec algoritmu.

1.25 Poznámka. Algoritmus Gift-wrapping se dá snadno zobecnit do vyšších dimenzí. Konvexní obaly v $\mathbf{R}^d$ pro $d > 2$ lze takto najít v čase $O(n^{\lfloor d/2 \rfloor + 1}) + O(n^{\lfloor d/2 \rfloor} \log n)$, viz [Preparata-85].

1.4 Porovnání algoritmů pro nalezení konvexních obalů bodů

V rovině jsme nalezli několik algoritmů pracujících v čase $O(n \log n)$. V praxi od nich nemůžeme očekávat lepší čas, protože používají v nějaké podobě buď třídění nebo rekurzi.

Jak uvidíme, lze od algoritmů Jarvis's March a Gift-wrapping method očekávat ve skutečnosti lepší časy, než ty, které jsme odvodili pro nejhorší případy. Dalším algoritmem, který je z hlediska očekávaného času výrazně lepší než v nejhorším případě, je MergeHull (alg. 1.8).

Tento algoritmus pracoval metodou *rozděl a panuj* a volal rekurzivně sám sebe. Jeho časová složitost $T(n)$ se odvodí pomocí vztahu

$$T(n) = 2T(n/2) + C(n), \tag{2}$$

kde $C(n)$ je čas potřebný na „panování“.

1.26 Věta. *Platí následující závislost hodnoty $T(n)$ na $C(n)$ ze vztahu 2:*

$T(n)$	$C(n)$
$O(n)$	$O(n \log n)$
$O(n/\log n)$	$O(n \log\log n)$
$O(n)$	$O(n^p)\ p < 1$

Důkaz: uvedeno v [Preparata-85], str. 153 bez důkazu □

Tabulka 1 shrnuje nejhorší i očekávané časy algoritmů pro nalezení konvexního obalu.

Uvedeme, jaký počet simpliciálních nadstěn konvexního obalu množiny bodů budeme očekávat u některých distribucí bodů vstupní množiny.

1.27 Věta. *Je-li n bodů vybráno náhodně uniformní distribucí v rovinném konvexním r-úhelníku, pak pro $n \to \infty$ se počet hran konvexního obalu těchto n bodů blíží*

$$E(H) = (\frac{2r}{3})(\gamma + \log_e n) + O(1), \tag{3}$$

kde γ je Eulerova konstanta [2].

Důkaz: Neuveden, [Preparata-85] odkazuje na text [Rényi-63]. □

1.28 Věta. *Je-li n bodů vybráno náhodně uniformní distribucí uvnitř d-dimenzionální hyperkoule, pak pro $n \to \infty$ se počet simpliciálních nadstěn konvexního obalu těchto n bodů blíží*

$$E(H) = O(n^{(d-1)/(d+1)}), \tag{4}$$

1.29 Věta. *Je-li n bodů vybráno náhodně normální distribucí v celém prostoru $\mathbf{R}^d$ pak pro $n \to \infty$ se počet simpliciálních nadstěn konvexního obalu těchto n bodů blíží*

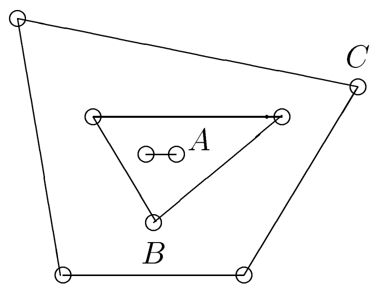
$$E(H) = O((\log n)^{(d-1)/2}), \tag{5}$$

Důkaz: Ani důkaz posledních dvou tvrzení neuvádíme, [Preparata-85] odkazuje na [Raynaud-70]. □

[2] $\gamma = \lim_{n\to\infty} (\sum_{i=1}^{n} 1/i - \log_e n) \doteq 0.5772156649015328606065120900082\ldots$

algoritmus	nejhorší případ	distribuce bodů v r-úhelníku	v kruhu	v rovině
počet hran	$O(n)$	$O(r \log n)$	$O(n^{1/3})$	$O((\log n)^{1/2})$
Jarvis's March	$O(n^2)$	$O(n \log n)$	$O(n^{4/3})$	$O(n\sqrt{\log n})$
MergeHull	$O(n \log n)$	$O(n)$	$O(n)$	$O(n)$
ostatní	$O(n \log n)$			
algoritmus	nejhorší případ	distribuce bodů	v kouli	v prostoru
počet stěn	$O(n)$		$O(n^{1/2})$	$O(\log n)$
Gift-wrapping	$O(n^2)$		$O(n^{3/2})$	$O(n \log n)$

Tabulka 1: Shrnutí časových složitostí algoritmů pro nalezení konvexního obalu



Obrázek 11: Konvexní vrstvy

1.5 Aplikace

Uvedeme nyní dva problémy, které jsou příbuzné konvexním obalům.

Konvexní vrstvy (Convex Layers) Zadání spočívá v nalezení do sebe vnořených konvexních obalů, mezi nimiž neleží žádné další body vstupní množiny. Jde vlastně o opakované hledání konvexního obalu pro body uvnitř předchozího konvexního obalu – viz obr. 11.

S tím souvisí i problém zjištění hloubky bodu v dané množině, což je pořadí konvexní vrstvy, ve které bod leží, počítáno od vnějšku. Např. na obrázku 11 má bod A hloubku 3 a bod B hloubku 2.

Je jistě možné opakovaně aplikovat některý algoritmus pro hledání konvexního obalu, to by však vedlo ke složitosti nejhoršího případu $O(n^2 \log n)$. Vhodnější by byl Jarvis's March, který bude pracovat v čase $\Theta(n^2)$. [Preparata-85] uvádí na straně 173 odkaz na [Chazelle-83], kterému se podařilo docílit pro oba problémy času $\Theta(n \log n)$.

Shluky Jde o nalezení shluků (clusters) bodů s co nejmenším průměrem.

Při hledání shluků narazíme na problém průměru množiny bodů, což je největší vzdálenost mezi dvojicí bodů v množině. Tento průměr odpovídá průměru konvexního obalu dané množiny. Je-li již znám BCH této množiny, lze průměr najít v čase lineárním (jednoduché cvičení). Konvexní obal dané množiny najdeme v čase $O(n \log n)$.

2 Voroného diagramy

Voroného diagramy[3] mají několik modifikací. Budeme se zabývat nejprve nejjednodušší z nich, a pak uvedeme jistá zobecnění tohoto případu. V některých podkapitolách se budeme zabývat problémy, které lze s využitím Voroného diagramů efektivně řešit.

2.1 Konstrukce Voroného diagramu

V této části uvedeme definici nejjednoduššího Voroného diagramu v rovině. Ukážeme několik základních vlastností, a přejdeme ke konstrukci tohoto diagramu v optimálním čase.

2.1 Definice. Nechť $S = \{x_1, \ldots, x_n\} \subset \mathbf{R}^2$. *Voroného oblast* bodu $x_i \in S$ je

$$VR_S(x_i) := \{y \in \mathbf{R}^2 \mid \forall 1 \leq j \leq n : dist(x_i, y) \leq dist(x_j, y)\}$$

Voroného diagram příslušný množině S je rozdělení roviny na n Voroného oblastí, značíme $VD(S)$.

Na obrázku 12 je ilustrován případ Voroného diagramu pro množinu pěti bodů v rovině.

2.2 Lemma. *$VR(x_i)$ je konvexní mnohoúhelníková oblast.*

Důkaz: Označme

$$H_{i,j} := \{y \in \mathbf{R}^2 \mid dist(x_i, y) \leq dist(x_j, y)\}$$

To je zřejmě polorovina a zároveň konvexní útvar. Navíc $VR_S(x_i)$ se dá zapsat jako $\bigcap_{i \neq j} H_{i,j}$. Pak $VR_S(x_i)$ musí být konvexní a je průnikem polorovin, tedy je to mnohoúhelníková oblast.

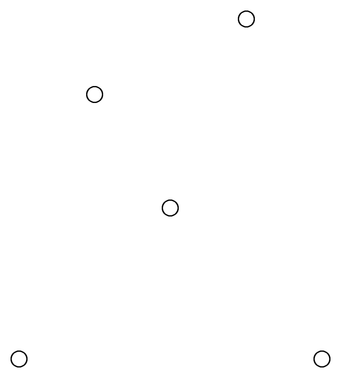
$\square$

2.3 Lemma. *$VR_S(x_i)$ je neohraničená oblast právě tehdy, když x_i leží na okraji konvexního obalu množiny S ($x_i \in BCH(S)$).*

Důkaz: $\Longrightarrow$: Nechť $VR_S(x_i)$ je neohraničená. Pak v ní jistě leží nějaká polopřímka vycházející z x_i. Tato musí protínat $BCH(S)$, např. v hraně $L(x_j, x_k)$. Za účelem sporu dále předpokládejme, že x_i neleží na hranici konvexního obalu. Tedy $x_i \notin L(x_j, x_k)$, a ve $VR(x_i)$ jsou body, které jsou blíže x_j nebo x_k než x_i, což je spor.

$\Longleftarrow$: Nechť $x_i \in BCH(S)$, x_j a x_k jsou jeho sousedi. Pak body na polopřímce kolmé k $L(x_j, x_k)$ procházející bodem x_i jsou blíže k x_i než ke všem ostatním bodům, tedy oblast $VR(x_i)$ není ohraničená. $\square$

[3]V anglickém literatuře je uveden termín *Voronoi diagrams.* Diagramy uvádíme jako Voroného, protože Voronoiho je jazykolam, a kromě toho je pravděpodobné, že jméno je to ruské [*voronoj*] nebo francouzské [*voronoa*]. V obou případech je přivlastňovací tvar [*voroného*].



65.0025.00165.005.002 65.0025.00391.0051.004 91.0051.00568.0058.006 68.0058.00747.0043

Obrázek 12: Voroného diagram pěti bodů

Seznámíme se nyní s pojmem *Delaunayova triangulace*. Souvisí úzce s Voroného diagramy, je vlastně první jejich využitelnou aplikací. S pomocí této triangulace lze také snadno ukázat některé vlastnosti Voroného diagramů.

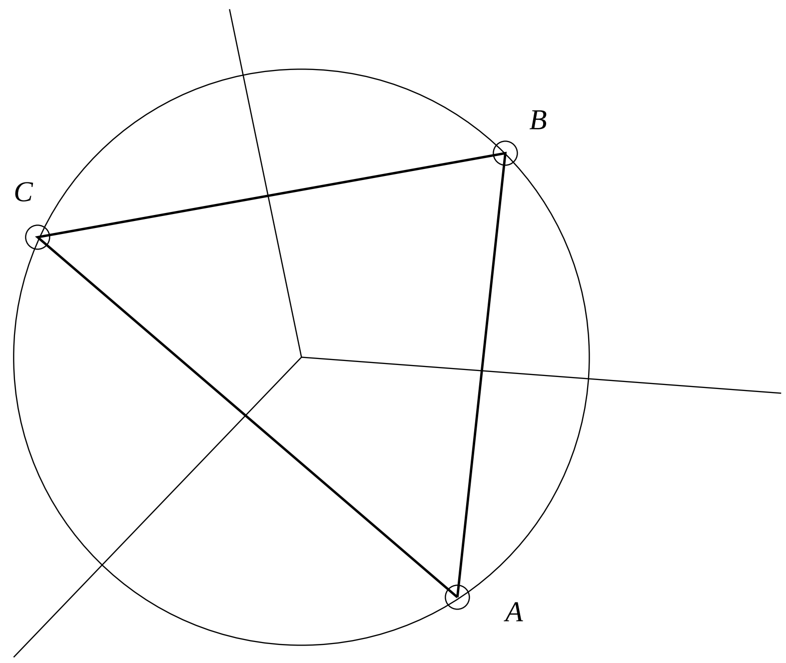
2.4 Definice. Nechť $S \subset \mathbf{R}^2$, $|S| = n$. Předpokládejme dále, že žádné čtyři body z S neleží na kružnici. Pak graf D s vrcholy z S a hranami spojujícími právě ty body z S, jejichž Voroného oblasti sdílí hranu, je triangulací $CH(S)$. Nazýváme ji *Delaunayova triangulace*.

2.5 Věta. *Delaunayova triangulace má tu vlastnost, že kružnice opsaná libovolnému jejímu trojúhelníku neobsahuje již žádné další body vstupní množiny.*

Důkaz: Voroného oblasti jsou mnohoúhelníkové oblasti, ohraničené osami úseček, které spojují jisté dvojice bodů z množiny S. K tomuto výsledku vede úvaha z důkazu lemmatu 2.2. Vezmeme nyní v úvahu libovolný trojúhelník ABC Delaunayovy triangulace, jak ukazuje také obrázek 13. Osy úseček $\overline{AB}$, $\overline{BC}$, $\overline{AC}$, značené postupně c, a, b, tvoří hranice jednotlivých Voroného oblastí odpovídajících bodům A, B a C. Jejich průsečík K je tedy bodem, který patří zároveň do všech tří oblastí. Jeho vzdálenost je tedy od všech tří bodů stejná, a zároveň vzdálenost od libovolného jiného bodu množiny je větší. Tento bod je tedy středem kružnice, na níž leží body A, B i C, a uvnitř níž neleží již žádný další bod množiny S. □

2.6 Lemma. *Voroného diagram množiny S s n body má nejvýše $2n-4$ vrcholů a $3n-6$ hran.*

Důkaz: Delaunayova triangulace D_S je rovinný graf, má tedy nejvýše $3n-6$ hran (pro $n > 2$). Zdůvodnění:



Obrázek 13: Jeden trojúhelník Delaunayovy triangulace a jeho souvislost s Voroného diagramem

Uspořádáme do posloupnosti všechny hrany všech stěn v rovinném grafu. Počet hran budeme značit h, počet stěn s a vrcholů n. Každá hrana se v této posloupnosti vyskytuje právě dvakrát (za každou stěnu, kterou ohraničuje, jednou), posloupnost má proto délku $2h$. Za každou stěnu jsou v posloupnosti pro $n > 2$ alespoň tři hrany, proto délka posloupnosti je alespoň $3s$. Tedy $2h \geq 3s$. Využijeme Eulerova vztahu $h + 2 = n + s$:

$$\begin{array}{rcccl} 3h+6 & = & 3n+3s & & \\ 3h+6-3n & = & 3s & \leq & 2h \\ h & & & \leq & 3n-6 \end{array}$$

V nedegenerovaném případě (žádné čtyři body neleží na kružnici) jsou hrany v bijekci s hranami $VD(S)$ a počet hran je roven $3n - 6$, v degenerovaném případě méně. Všechny vrcholy $VD(S)$ mají stupeň alespoň tři, proto počet vrcholů je nejvýše $\frac{2}{3}(3n - 6) = 2n - 4$. $\square$

Nyní se vrhneme na konstrukci Voroného diagramu. Nejprve odhadneme nejlepší časovou složitost, které můžeme dosáhnout.

2.7 Věta. *Každý algoritmus pro nalezení Voroného diagramu množiny n bodů bude pro nejhorší případ potřebovat alespoň $\Omega(n \log n)$ času.*

Důkaz: Uvážíme-li nejhorší případ množiny n bodů ležících na kružnici, budou Voroného oblasti ohraničené vždy dvěma polopřímkami vycházejícími ze středu této kružnice. Tyto polopřímky jsou osami úseček spojujících sousední body. Pro nalezení Voroného diagramu je tedy potřeba odhalit dvojice sousedních bodů na kružnici, což odpovídá setřídění bodů podle úhlů. $\square$

Následující věta se skládá z několika tvrzení, na kterých je postaven celý náš algoritmus konstrukce Voroného diagramu. Věta operuje se dvěma stejně velkými podmnožinami množiny S, jejichž Voroného diagramy jsou známy. To vlastně už napovídá, jak bude vypadat algoritmus ke kterému směřujeme: metoda *rozděl a panuj* s rekurzivním voláním sebe sama.

2.8 Věta. *Nechť $S = \{x_1, \dots, x_n\} \subset \mathbf{R}^2$, nechť S_L a S_R je levá a pravá polovina S vzhledem k lexikálnímu setřídění. Předpokládejme, že známe $VD(S_L)$ i $VD(S_R)$. Definujme*

$$P := \{y \in \mathbf{R}^2 \mid dist(y, S_L) = dist(y, S_R)\}$$

P je tedy množina všech bodů, které mají stejnou vzdálenost od S_L a S_R. Označme dále $L(P)$ část roviny nalevo od P, $R(P)$ napravo od P.

Potom platí:

1. *$P = \{y \mid y$ leží na hraně $VD(S)$ kolmé na spojnici $L(x_i, x_j)$ pro nějaké $x_i \in S_L$, $x_j \in S_R\}$*
2. *P je monotónní (při vhodné orientaci žádná z hran P nesměřuje dolů)*
3. *$VD(S) = (VD(S_L) \cap L(P)) \cup P \cup (VD(S_R) \cap R(P))$*
4. *„Nejspodnější“ hrana P je polopřímka a zároveň osa dolní tečny $BCH(S_L)$ a $BCH(S_R)$.*

Důkaz:

1. Označme pravou stranu výrazu jako Q. Máme nyní dokázat dvě množinové inkluze.
   - $P \subseteq Q$: Nechť bod $y \in P$. Tedy vzdálenost bodu y od množin S_L a S_R je stejná. Vybereme nyní ten bod z množiny S_L, který má od y nejmenší vzdálenost, a označíme x_L. Stejně vybereme bod x_R v množině S_R. Víme, že vzdálenost y od x_L je stejná jako jeho vzdálenost od x_R. Kromě toho je libovolný jiný bod S_L i S_R (a tedy i S) dále od bodu y. Proto y ve $VD(S)$ odděluje $VR(x_L)$ od $VR(x_R)$, leží tedy na hraně kolmé na spojnici bodů x_L a x_R.
   - $Q \subseteq P$: Nechť $y \in Q$. Pak

     $$\forall x \in S \; dist(y, S) = dist(y, x_i) = dist(y, x_j) \leq dist(y, x)$$

     Z toho už plyne, že

     $$dist(y, S_L) = dist(y, x_i) = dist(y, x_j) = dist(y, S_R)$$

     a bod y je tedy prvkem P.
2. Díky lexikálnímu rozdělení množiny S na S_L a S_R můžeme úsečky v P orientovat tak, aby body z S_L byly vždy vlevo při pohledu ve směru orientace úseček.
3. Označme pravou stranu výrazu jako D. Máme nyní dokázat dvě množinové inkluze.
   - $VD(S) \subseteq D$: $y \in VD(S)$, tj. y leží na některé z hran, proto existují i, j tak, že $dist(y, x_i) = dist(y, x_j) \leq dist(y, x)$ $\forall x \in S$. Pokud $x_i, x_j \in S_L$, pak zřejmě $y \in VD(S_L) \cap L(P)$. Je-li $x_i \in S_L, x_j \in S_R$, pak $y \in P$. Analogicky pokud $x_i, x_j \in S_R$, pak $y \in VD(S_R) \cap R(P)$.

- $VD(S) \supseteq D$: $y \in D$. Je-li $y \in P$, pak $y \in VD(S)$.
  $y \in VD(S_L) \cap L(P)$: $y \in L(P) \Rightarrow dist(y, S_L) \leq dist(y, S_R)$ a protože $y \in VD(S_L)$, existují $x_i, x_j \in S_L$ tak, že $dist(y, S_L) = dist(y, x_i) = dist(y, x_j) \leq dist(y, x)\ \forall x \in S \Rightarrow y \in VD(S)$.
  Pro $y \in VD(S_R) \cap L(P)$ je důkaz analogický.

4. Víme, že nejspodnější hrana P je také hranou $VD(S)$. Je zřejmé, že je to polopřímka. Zbytek tvrzení je intuitivně jasný, formálně plyne z lemmatu 2.3.

□

V následujícím algoritmu budeme předpokládat, že už známe $VD(S_L)$ i $VD(S_R)$, a že množiny S_L a S_R jsou rozděleny podle lexikografického uspořádání jako ve větě 2.8. Podle bodu 3 předchozí věty stačí najít dělicí lomenou čáru P, a ohraničit pomocí ní původní Voroného diagramy. Sjednocením dostaneme Voroného diagram celé množiny S. Jde vlastně o poslední krok algoritmu *rozděl a panuj*, který spojuje dohromady mezivýsledky.

Označíme L nejspodnější hranu lomené čáry P (bod 4 věty 2.8).

Předpokládejme, že postupujeme při hledání P odspodu, a máme už části $L, e_1, e_2, \dots, e_n$ lomené čáry P, přičemž poslední úsečka či polopřímka zatím není omezena. Pak buď neprotíná žádnou z hran $VD(S_L)$, $VD(S_R)$ – v tom případě jsme právě získali poslední polopřímku („nehornější“ hranu) L' čáry P, nebo protíná např. nejdříve $VD(S_L)$, a to v bodě z na hraně e. V S_L jsou body x_i, x'_i takové, že $e \in VR(x_i), e \in VR(x'_i), dist(x_i, z) = dist(x'_i, z)$. e_n však musí být osou úsečky spojující jeden z bodů x_i, x'_i (nechť je to např. x_i) a nějaký bod z S_R, např. x_j. Jelikož bod z leží na e_n, je stejně vzdálen od bodů x_i, x'_i i x_j. Zároveň však žádný bod množiny S není bodu z blíže, proto z je vrchol $VD(s)$. Zde také je končí hrana e_n lomené čáry P, a začíná další hrana e_{n+1}, která je osou úsečky $L(x'_i, x_j)$.

Algoritmus 2.1 ALGORITMUS TZV. „SEŠÍVÁNÍ“ $VD(S_L)$ A $VD(S_R)$

Vstup : $VD(S_L)$, $VD(S_R)$
Výstup : $VD(S_L \cup S_R)$

1. $h := 1$
   - $T_L \dots$ dolní tečna $BCH(S_L), BCH(S_R)$ spojující $x \in S_L, y \in S_R$
   - $L \dots$ polopřímka, která je osou $L(x, y)$
2. $l_1 := L$
3. `while` L protíná $VD(S_L)$ nebo $VD(S_R)$

   3.1. `if` L protne nejdříve $e \in VD(S_L)$ `then`

   3.1.1. $z :=$ bod průniku

   3.1.2. l_h ukonči v z

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

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

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

3.1.6. $l_h := L$

3.2. `else` SYMETRICKY PRO S_R

Časová složitost tohoto algoritmu při reprezentaci Voroného diagramu dvojitým spojovým seznamem (*double-connected-edge-list*, viz poznámka 1.24 a algoritmus 1.10 *Gift-Wrapping*) je $O(n)$, kde n je počet bodů sjednocení množin S_R a S_L. Počet všech hran obou diagramů $VD(S_L)$ i $VD(S_R)$ totiž nepřevýší dohromady $3n-6$ podle lemmatu 2.6, a ani čára P nemůže mít více než n hran, tedy čas zůstává lineární.

Nyní si můžeme dovolit tvrdit, že

2.9 Věta. *Voroného diagram lze sestavit v optimálním čase $O(n \log n)$.*

Důkaz: Optimálnost času ukazuje věta 2.7. Algoritmus pracující v tomto čase lze zkonstruovat metodou *rozděl a panuj*. Musíme nejprve lexikálně uspořádat body množiny S. Potom budeme dělit S vždy na dvě stejně velké poloviny podle uspořádání, a rekurzivně volat tento algoritmus pro obě poloviny zvlášť. Spojení výsledků zařídí předchozí algoritmus (2.1). Jako zarážka rekurze může sloužit test na velikost vstupní množiny. Je-li totiž vstupem jediný bod, jeho Voroného diagram je celá rovina.

Počáteční fáze setřídění bodů bude trvat $O(n \log n)$ času. Označíme-li časovou složitost zbytku algoritmu pro n bodů jako $T(n)$, dostáváme: $T(n) = 2T(n/2) + O(n)$ a $T(1) = 1$, tedy $T(n) = O(n \log n)$. □

2.2 Voroného diagramy v řece

Prvním možným zobecněním Voroného diagramů je uvedení roviny do rovnoměrného pohybu. Příměr k tekoucí řece je zde velice výstižný. Obyčejný Voroného diagram je speciálním případem tohoto pro rychlost plovoucí roviny rovnu nule.

Z daných bodů roviny se šíří konstantní rychlostí v *vzruch* všemi směry. Máme vytvořit rozdělení roviny, která „pluje“ konstantní rychlostí w. Označme $\alpha = \frac{w}{v}$ – tzv. *relativní rychlost*. Z bodu $(0,0)$ se v čase t rychlostí v pod úhlem θ dostaneme do bodu

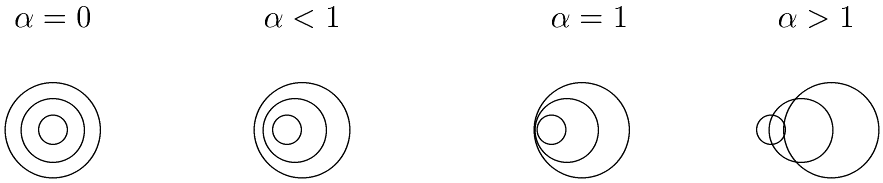
$$(x, y) = (tw + tv\cos\theta, tv\sin\theta)$$

Jde tedy o kružnici se středem $(tw, 0)$ a poloměrem tv.

2.10 Definice. *Voroného diagram* $VD^\alpha(S)$ definujeme jako rozdělení celé roviny na oblasti $VR^\alpha(x_i), x_i \in S$, do nichž se z bodu x_i dostaneme nejrychleji za výše uvedených podmínek.

2.11 Poznámka. Předchozí definice je korektní, protože pro příslušnost k Voroného oblastem v řece nejsou rozhodující obě hodnoty w i v, ale stačí jejich poměr α. K ověření vede následující úvaha.

Nechť hodnoty rychlostí jsou v a w a do bodu y se dostaneme z bodu x_i v čase t_i. Změníme-li obě hodnoty rychlostí na jejich k-násobek, lze snadno ověřit, že se z bodu x_i dostaneme v čase t_i/k do bodu y. Minimum mezi těmito hodnotami zůstane tedy zachováno, a s ním i příslušnost bodu y k Voroného oblasti příslušného diagramu.



116.0025.001135.0014.002 116.0025.003135.0036.004

Obrázek 14: Kružnice dosahu pro některé hodnoty α

10.0040.00120.0060.002 20.0060.00340.0060.004 40.0060.00530.0040.006 30.0040.00710.0040.008

Obrázek 15: Kužele pod body množiny S a úhel průmětu zpět do roviny

15.0030.00132.0013.002 15.0030.00328.0052.004 15.0030.0053.0026.00690.0052.00790.0013.008150.0035

Obrázek 16: Výsledné Voroného diagramy v řece pro některé hodnoty α

2.12 Poznámka. Pro $\alpha \geq 1$ ve $VR^\alpha(x_i)$ není žádný bod s menší x-ovou souřadnicí než má x_i.

Voroného diagram $VD^\alpha(S)$ lze sestrojit pomocí kuželů pod body množiny S. Vložíme naši rovinu do třírozměrného prostoru a nechť třetí rozměr odpovídá času t, přičemž naše původní rovina budiž položena v časové rovině $t = 0$. Z každého bodu vyšleme po jeho časové ose kružnice, do kterých se právě dá z tohoto bodu dostat v daném čase t. Tyto kružnice budou mít rovnoměrně rostoucí poloměr a dostaneme tak vždy plášť kužele. Tento postup je ilustrován na obrázku 15. Promítneme-li průniky těchto kuželů kolmo zpět do roviny, dostaneme $VD^0(S) = VD(S)$.

Výše uvedený postup lze zobecnit pro plovoucí rovinu. Kružnice pod body množiny S budou v čase nejen rovnoměrně zvětšovat svůj poloměr a pohybovat se zároveň po ose kolmé na původní rovinu (rychlostí v), ale budou se zároveň pohybovat ve směru osy x rovnoměrným pohybem rychlostí w. Kužely budou v kolmém průmětu přesně kopírovat dosah vzruchů vypuštěných z bodů množiny S. Tím docílíme toho, že kolmým průmětem průniků takových kuželů budou Voroného oblasti VR^α.

Kružnice pod body se vlastně pohybují rychlostí danou vektorovým součtem rychlostí $\vec{v}$ a $\vec{w}$. Její výslednice svírá s rovinou xy úhel γ z obrázku 15. Promítneme-li tedy průniky původních kuželů, nepohybujících se ve směru osy x, zpět do roviny xy pod úhlem znázorněným na obrázku 15, dostáváme opět správný výsledek.

Po promítnutí se paraboly (průniky plášťů kuželů) promítnou na hyperboly. V případech, kdy $\alpha \geq 1$, přichází do úvahy pouze ty oblasti, kam se vůbec dá z jednotlivých bodů dostat. Ty jsou ohraničeny polopřímkami vedenými z těchto bodů pod úhlem γ; viz také obrázek 14, případ $\alpha > 1$. Tyto polopřímky jsou průmětem površek na příslušném kuželu vymezujících viditelnost při promítání. Odtud plyne, že hranice Voroného oblastí přechází z těchto polopřímek na paraboly právě v průmětech míst, kde se protínají zmíněné površky s průniky kuželů. Obrázek 16 naznačuje výsledné Voroného diagramy $VD^\alpha(S)$, rozdělené na charakteristické případy podle hodnoty α.

2.13 Poznámka. Pro $\alpha \geq 1$ je každý bod množiny S nejlevějším bodem své oblasti. Díky tomu můžeme zvolit alternativní postup pro konstrukci $VD^\alpha(S)$. Jedná se o metodu tzv. *pročesávání* (plovoucí horizont, sweep). Tento algoritmus nebudeme podrobně rozebírat, zmíníme se ale stručně o principu metody, která bude později užitečná v řadě dalších problémů. Tato metoda je založena na (lexikografickém) setřídění bodů v S a zavedení tzv. *horizontální a vertikální struktury*.

- Horizontální struktura, tedy rovnoběžná s osou x, obsahuje polohy tzv. *pročesávací přímky,* které je třeba diskutovat. Jde tedy o jakýsi setříděný seznam bodů na ose x, o kterých víme, že se v nich může měnit pro nás významná veličina. Zpočátku algoritmu bude tato struktura obsahovat právě x-ové souřadnice bodů z S, později se k nim budou přidávat průsečíky hranic jednotlivých oblastí a body, ve kterých přechází hranice oblastí z přímek na hyperboly.

- Vertikální struktura (*pročesávací přímka*) je rovnoběžná s osou y. Necháme ji probíhat postupně všemi body z horizontální struktury. Na ní budou vždy ležet významné hranice oblastí (buď body z původní množiny, nebo jiné průsečíky oblastí).

Při jednotlivých diskusích vertikální struktury (pročesávací přímky) v čase $O(\log n)$ získáme algoritmus o celkovém čase $O(n \log n)$, protože všech bodů ze vstupní množiny i průsečíků oblastí dohromady (a tedy prvků horizontální struktury) je $O(n)$.

V některých dalších pročesávacích algoritmech může být pročesávací přímka naopak horizontální. V těchto případech bude zaměněn faktický význam horizontální a vertikální struktury. Obecně se lze setkat i s jiným pročesávacím objektem než je přímka, např. ve vyšších dimenzích, často také hovoříme o *struktuře událostí,* resp. o *statutu událostí.*

Podstatou pročesávání je redukce dimenze za cenu přechodu k dynamickým strukturám. Později se k této třídě algoritmů vrátíme podrobněji.

V dalších dvou podkapitolách zmíníme aplikace Voroného diagramů při řešení dvou častých úloh: nalezení nejbližších sousedů v množině bodů a nalezení minimálního pokrývajícího stromu množiny bodů v Euklidovské rovině.

2.3 Problémy nejbližších sousedů

Budeme se zabývat třemi problémy nejbližších sousedů:

1. Z konečné množiny bodů $S \subset \mathbf{R}^2, |S| = n$ vyber takové dva body, které mají nejmenší vzdálenost (problém nalezení nejbližšího páru)

2. Pro každý bod $x \in S$ najdi jeho nejbližšího souseda (problém nalezení nejbližších sousedů)

3. Máme-li dánu pevně množinu S a libovolný bod v rovině x, najdi v množině S bod nejbližší bodu x

Problém 3 znamená vlastně vyhledání oblasti ve Voroného diagramu množiny S, ve které leží bod x. Máme-li tedy předspočítán Voroného diagram, stačí použít některý z algoritmů na vyhledávání v rovinných rozděleních, které uvedeme v kapitole 3.2.

Na první pohled se může zdát podivné, že první dva problémy budeme řešit ve stejném čase. Nalezení nejbližšího páru budeme totiž řešit tak, že najdeme nejbližší sousedy, a pak z nich vybereme tu nejbližší dvojici.

Pokusíme se naznačit důkaz, že výše uvedený postup nám skutečně zaručí optimální čas i pro nalezení nejbližšího páru. Využijeme obecný princip redukce problémů:

Podaří-li se transformovat v čase $O(f(n))$ problém A na problém B, a má-li problém B časovou složitost $O(g(n))$ nám předem známou, a je-li funkce g asymptoticky větší než f, víme, že problém A můžeme řešit v čase $O(g(n))$ nebo horším. Kdyby se nám totiž podařilo vyřešit problém A v čase lepším, mohli bychom pomocí transformace v čase $O(f(n))$ dosáhnout celkem času lepšího než $O(g(n))$ k vyřešení problému B, což je spor.

Problém 1 je transformovatelný na

zjisti, zda mezi n reálnými čísly jsou alespoň dvě stejná

takto:

Najdeme nejbližší dvojici z $(x_i, 0), (x_j, 0)$. Pokud je jejich vzdálenost větší než nula, jsou všechna daná čísla různá. Problém, zda mezi n reálnými čísly jsou 2 stejná, je řešitelný v čase $\Omega(n \log n)$. Důkaz je uveden v [Preparata-85], strana 192. Transformace zabere čas $O(n)$. Kdyby se tedy podařilo najít nejbližší pár v čase lepším než $\Omega(n \log n)$, po transformaci bychom znali lepší čas než $\Omega(n \log n)$ i pro problém zjištění rovnosti čísel.

Problém nalezení nejbližších sousedů se dá v čase $O(n)$ transformovat na nalezení nejbližšího páru. Ten totiž musí být mezi n nejbližšími sousedy, a prostým vyhledáním minima ho lze vyhledat. Nejbližší sousedy tedy také nepůjde nalézt v čase lepším než $\Omega(n \log n)$. Následující úvahy směřují ke zjištění, že to právě v uvedeném čase jde.

Následující lemma charakterizuje chování nejbližších sousedů ve Voroného diagramu.

2.14 Lemma. *Nechť $S \subset \mathbf{R}^2; x, y \in S$. Je-li $VR(x) \cap VR(y) = \emptyset$ nebo $|\, VR(x) \cap VR(y)| = 1$, pak existuje $z \in S$ tak, že $dist(x,z) \leq dist(x,y)$ a $VR(x)$ a $VR(z)$ sdílí hranu.*

Důkaz: Pro případ $VR(x) \cap VR(y) = \emptyset$ je to zřejmé. Nechť $\{p\} = L(x,y) \cap VR(x)$. Bod p patří do $VR(z)$ pro jisté z takové, že $VR(x)$ a $VR(z)$ sdílí hranu. Zároveň $dist(x,p) = dist(z,p)$, tedy

$$dist(x,z) \leq 2\, dist(x,p) \leq dist(x,y)$$

Rovnost může nastat pouze pro $dist(p,x) = dist(p,y)$ a $p \in L(x,z)$. □

Z lemmatu bezprostředně plyne

2.15 Důsledek. *Pro každý bod x z množiny S platí, že jeden z jeho nejbližších sousedů mezi body z S s ním sdílí hranu ve $VD(S)$.*

Nejbližší sousedy lze tedy hledat mezi sousedy ve Voroného diagramu.

2.16 Věta. *Nechť $S \subset \mathbf{R}^2, |S| = n$. Je-li dán $VD(S)$, pak problém nalezení všech nejbližších sousedů je řešitelný v čase $O(n)$. Proto i problém nalezení nejbližší dvojice lze řešit v čase $O(n)$.*

Důkaz: Podle lemmatu 2.14 má každý prvek $x \in S$ nejbližší sousedy $y \in S$ takové, že $VR(x)$ a $VR(y)$ sdílí hranu. Při hledání budeme pro každý bod hledat pouze mezi jeho sousedy ve $VD(S)$. Projdeme tedy každou hranu nejvýše dvakrát. Počet hran $VD(S)$ je podle lemmatu 2.6 lineární. $VD(S)$ reprezentovaný dvojitým spojovým seznamem můžeme projít v lineárním čase. Mezi lineárním počtem výsledků najdeme minimum, tedy nejbližší pár, také v lineárním čase. □

2.17 Důsledek. *Problém nejbližších sousedů i nejbližšího páru lze řešit v optimálním čase* $O(n \log n)$.

Důkaz: Tvrzení plyne z úvah o optimalitě problémů dříve v této podkapitole, z věty 2.9 o konstrukci Voroného diagramu v čase $O(n \log n)$, a z předchozí věty. □

2.4 Minimální pokrývající strom

Obecný problém nalezení minimálního pokrývajícího stromu (kostry) je pokryt přednáškou doc. Poláka *Grafové algoritmy*. Běžně se uvádí dva algoritmy (např. v [MIT][4]):

- *Kruskalův algoritmus* pracuje v čase $O(E \log E)$, kde E je počet hran grafu
- *Primův algoritmus* dává výsledek v čase $O(E \log V)$ při použití haldy, resp. $O(E+V \log V)$ při použití *Fibonacciho haldy*, kde E je opět počet hran a V počet vrcholů grafu

Stojí za zmínku, že pravděpodobně první algoritmus řešící tento problém nabídl brněnský matematik prof. Otakar Borůvka, když dostal ve 20-tých letech za úkol navrhnout elektrifikaci Moravy.

My se budeme zabývat Euklidovskou modifikací problému. Uzly našeho grafu budou vnořeny do roviny s Euklidovskou metrikou, a graf budeme považovat za úplný.

Problém Nalezněte pro $S \subset \mathbf{R}^2, |S| = n$ strom T takový, že uzly jsou všechny body z S, hrany jsou ohodnoceny vzdáleností mezi odpovídajícími body, a součet délek hran stromu T je minimální mezi všemi takovými stromy.

Aplikujeme-li Kruskalovy a Primovy výsledky, dostáváme čas $O(n^2 \log n)$, resp. $O(n^2)$ v případě Primova algoritmu s použitím Fibonacciho haldy.

Dále uvidíme, že speciální případ Euklidovské varianty nám za pomoci Delaunayovy triangulace dá výsledek lepší.

2.18 Lemma. *Minimální pokrývající strom T množiny n bodů existuje tak, že všechny jeho hrany leží v Delaunayově triangulaci D dané množiny.*

Důkaz: Nechť T je strom minimální pokrývající a ze všech takových ten, který má maximální počet hran v D. Předpokládejme, že v T je hrana (x, y), která neleží v D. Potom $VR(x)$a $VR(y)$ nemají společnou hranu. Pak (podle lemmatu 2.14) existuje $z \in S$ tak, že $VR(x)$ a $VR(z)$ mají společnou hranu a $dist(x, z) \leq dist(x, y)$, navíc $dist(y, z) < dist(x, y)$. Obě hrany (y, z) a (x, z) nepatří zároveň do T, a buď

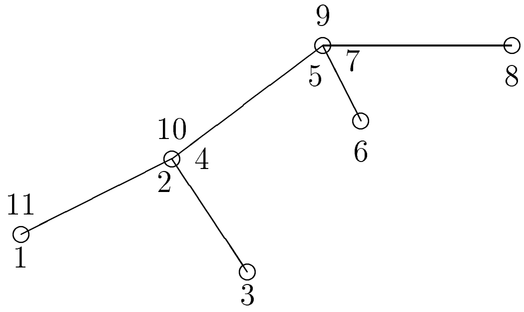
$$T_1 = (T - \{(x, y)\}) \cup \{(x, z)\}$$

nebo

$$T_2 = (T - \{(x, y)\}) \cup \{(y, z)\}$$

bude opět pokrývající strom. T_2 má menší ocenění hran a T_1 má sice menší nebo rovné ocenění hran jako T, ale má zato více hran z D. V obou případech tedy nastane spor. □

[4] Jde o brožurku, kterou používá při výuce doc. Polák. Bohužel neznám přesnější údaje, jako rok a místo vydání, proto se obraťte přímo na pana docenta.



Obrázek 17: Průchod kostrou při navštívení každého uzlu alespoň jednou a každé hrany právě dvakrát

2.19 Věta. *Minimální Euklidovský pokrývající strom konečné množiny $S \subset \mathbf{R}^2$ lze najít v čase $O(n)$, je-li dána Delaunayova triangulace D.*

Důkaz: Podle předchozího lemmatu stačí diskutovat hrany D, což je rovinný graf, a ten má lineární počet hran. □

Známou aplikací minimální kostry je nalezení jistého přibližného řešení pro problém obchodního cestujícího v grafu.

Problém obchodního cestujícího spočívá v nalezení Hamiltonovské kružnice (kružnice procházející každý vrchol právě jednou), která má minimální součet délek hran.

Pohybujeme-li se pouze po minimální kostře a každou hranu můžeme navštívit právě dvakrát, lze projít všemi vrcholy alespoň jednou a dostat se zpátky do výchozího vrcholu. Na obr. 17 je naznačen takový průchod očíslováním vrcholů, jak jsou postupně navštíveny. Dostáváme tak cestu délky dvojnásobné oproti minimální kostře. Každá Hamiltonovská kružnice, i ta nejkratší, která je kýženým řešením, má o jednu hranu víc než nějaká kostra. Tato kostra je však delší nebo rovna než minimální kostra. Celkem je dvojnásobek minimální kostry menší než dvojnásobek řešení problému obchodního cestujícího, a přitom jsme v této délce mohli navštívit všechny uzly.

V Euklidovském prostoru můžeme při procházení grafem přeskočit ty uzly, které už byly navštíveny. Víme totiž, že taková „zkratka“ bude skutečně kratší. Na obr. 17 tak dostaneme kružnici procházející postupně uzly $1 \to 2 \to 3 \to 5 \to 6 \to 8 \to 11 = 1$.

2.20 Důsledek. *Pro Euklidovský problém obchodního cestujícího lze najít v čase $O(n \log n)$ (nebo v čase $O(n)$, je-li už dána Delaunayova triangulace) přiblížení, jehož délka dosahuje maximálně dvojnásobku optima.*

2.5 Dynamická aktualizace Voroného diagramu

Mnoho skutečných aplikací má dynamickou povahu, data se mění v čase. Abychom nemuseli datové struktury vždy celé znovu počítat i při malé změně dat, navrhujeme tzv. *dynamické* *datové struktury.* Jde vlastně o návrh algoritmů, které umožňují efektivně do struktury vložit prvek a vybrat z ní prvek.

Uvedeme nyní algoritmus pro přidání prvku do Voroného diagramu a popíšeme algoritmus, který by prvek z Voroného diagramu naopak vybral.

Algoritmus přidání prvku y do Voroného diagramu předpokládá znalost bodu $x \in S$, v jehož Voroného oblasti $VR(x) \in VD(S)$ leží bod y. Problémem vyhledání takové oblasti se budeme zabývat v kapitole 3.2. Najdeme sice algoritmus, který vyhledává v čase $O(\log n)$, nebudeme však schopni v takovém čase aktualizovat příslušnou vyhledávací strukturu. V každém případě se nabízí lineární prohledávání všech oblastí.

Následující popis algoritmu je možné konfrontovat s obrázkem 18.

Budeme postupovat po hranici Voroného oblasti nově přidávaného bodu y. Nejprve najdeme hranu, která leží uvnitř $VR(x)$. Tou bude osa úsečky $L(x, y)$ ohraničená oblastí $VR(x)$. Průnik této osy s Voroného oblastí bodu x budou dva body $\{a, b\}$, které se jistě stanou vrcholy v novém Voroného diagramu. Nechť např. bod b leží na hranici mezi $VR(x)$ a $VR(z)$. Pak bod b je stejně vzdálen od bodů x, y, z, a uvnitř kružnice těmto třem bodům opsané už neleží žádný další bod z $S \cup \{y\}$, proto b bude vrcholem výsledného diagramu.

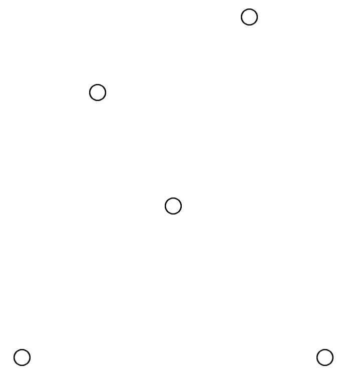
Bodem z nyní nahradíme bod x. Budeme tedy hledat průsečíky $L(z, y)$ s Voroného oblastí bodu z. Jedním z nich bude jistě bod b, protože je středem kružnice opsané trojúhelníku $\triangle xyz$. Druhý takový bod však již nemusí existovat, jako v části b) obrázku 18. V tomto případě je nutné se vrátit k výchozímu bodu x a postupovat opačným směrem, tedy pomocí vrcholu a k bodu r atd. V případě, že konečná Voroného oblast bodu y bude ohraničená jako v části a) obrázku 18, nebude nutné se vracet, protože se po hranici této oblasti dostaneme zpátky do výchozího bodu x.

Algoritmus 2.2 PŘIDÁNÍ PRVKU DO VORONÉHO DIAGRAMU

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



75.00100.00175.0080.002 75.00100.003101.00126.004 57.00118.00575.00100.006 57.00118.00737.001

Obrázek 18: Dynamická aktualizace Voroného diagramu – přidání bodu

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 pro nalezení $VD(S \setminus \{y\})$, tedy vypuštění prvku z množiny S, spotřebuje čas $O(s \log s)$, kde s je počet sousedů bodu y ve Voroného diagramu. Stačí totiž vyhledat tyto sousedy a přepočítat Voroného diagram pouze pro ně. Hranice jejich Voroného oblastí s ostatními body zůstane nezměněna.

2.6 Voroného diagramy vyšších řádů

Vraťme se k definici Voroného diagramu z podkapitoly 2.1. Pro každý bod p roviny jsme hledali ten bod x z dané konečné množiny bodů S, pro který je vzdálenost $dist(p, x)$ minimální. Vybíráme tedy tu *jednoprvkovou podmnožinu* S, ke které má bod p nejblíže. Tento Voroného diagram označíme jako *diagram prvního řádu*.

Voroného diagram druhého řádu bude vycházet z vyhledání *dvouprvkové podmmnožiny* S, ke které má bod p nejblíže ze všech možných dvouprvkových podmnožin S. Pro každou dvouprvkovou podmnožinu $T \subseteq S$ pak označíme všechny body, které byly takto přiřazeny této množině, jako *Voroného oblast druhého řádu množiny* T.

Následující výklad pracuje s obecným řádem Voroného oblastí a diagramů.

2.21 Definice. Nechť $S \subset \mathbf{R}^2$ je konečná množina, T její podmnožina. Pak Voroného oblast podmnožiny T množiny S definujeme takto:

$$VR(T) := \{p \in \mathbf{R}^2 : \forall x \in (S - T)\ \forall y \in T : dist(p, y) \leq dist(p, x)\}$$

S takto definovanou množinou bodů se nepracuje příliš pěkně. Raději si vyjádříme každou Voroného oblast pomocí jednodušších útvarů.

2.22 Věta.

$$VR(T) = \bigcap_{\substack{x_i \in T \\ x_j \in S - T}} H_{i,j},$$

kde $H_{i,j} = \{y; dist(x_i, y) \leq dist(x_j, y)\}$.

2.23 Poznámka. Množina $H_{i,j}$ z předcházející věty je polorovina, jejíž hraniční přímkou je osa úsečky $L(x_i, x_j)$ a která obsahuje bod x_i.

2.24 Důsledek.

1. *$VR(T)$ je vždy konvexní mnohoúhelníková oblast.*

2. *$VR(T)$ může být i prázdná množina.*

2.25 Definice. *Voroného diagram k-tého řádu* $VD_k(S)$ je rozdělení roviny na Voroného oblasti $VR(T)$ pro všechny podmnožiny T množiny S, které obsahují právě k prvků.

2.26 Poznámka. $VD_0(S)$, stejně jako $VD_n(S)$ pro $|S| = n$ jsou triviální diagramy s jedinou oblastí, a to celou rovinou. $VD_1(S)$ je nejjednodušší případ Voroného diagramu ze začátku této kapitoly.

Vrchol ve $VD_k(S)$ se najde např. takto: Zvolíme libovolné tři body $x, y, z \in S$. Pak střed kružnice proložené těmito body je vrcholem hranic ve $VD_k(S)$, kde $k-1$ je počet bodů uvnitř této kružnice (nepočítaje body x, y, z).

Podrobný popis všech vrcholů ve $VD_k(S)$ je obsažen v následující větě, která zároveň ukazuje zajímavou souvislost mezi vrcholy a hranami diagramů sousedních řádů. Tím získáme iterativní konstrukci Voroného diagramů všech řádů.

2.27 Věta. *Vrcholy hranic oblastí ve $VD_k(S)$ se objeví právě ve dvou Voroného diagramech $VD_l(S)$, $VD_{l+1}(S)$, kromě těch, které jsou pouze ve $VD_{n-1}(S)$; $|S| = n$. Každý vrchol, který se objeví poprvé ve $VD_l(S)$, odpovídá volbě*

$$T_1 = R \cup \{x\}, T_2 = R \cup \{y\}, T_3 = R \cup \{z\}$$

pro jistou podmnožinu $R \subset S$, která má $l-1$ prvků, a body $x, y, z \in S$ nepatřící do R (tento vrchol je středem kružnice procházející body x, y, z). Tentýž vrchol se pak objeví ve $VD_{l+1}(S)$ jako vrchol odpovídající

$$T_1' = R \cup \{y, z\}, T_2' = R \cup \{x, z\}, T_3' = R \cup \{x, y\},$$

přičemž původní hrany ve $VD_l(S)$ vycházející z tohoto vrcholu jsou v bijekci s hranami vycházejícími ze stejného vrcholu ve $VD_{l+1}(S)$, jedná se o opačné polopřímky (viz obr. 20). Navíc je ve $\bigcup_{k=1}^{n-1} VD_k(S)$ $O(n^3)$ vrcholů.

Důkaz: používá složitou konstrukci. Nejprve vložíme rovinu xy do třírozměrného prostoru (je dána osami, tedy prochází bodem $[0, 0, 0]$). V tomto prostoru sestrojíme rotační paraboloid P daný rovnicí

$$P \equiv z(x, y) = x^2 + y^2 \tag{6}$$

Paraboloid P se dotýká roviny xy právě v bodě $[0, 0, 0]$.

K prvkům $s_i \in S$ sestrojíme body $z_i \in P$ jako jejich kolmé projekce na paraboloid P. V každém z_i uvažujme tečnou rovinu π_i k P. Parametrizujeme-li π_i body kolmé projekce do roviny xy, dostaneme:

$$p \in \mathbf{R}^2(xy) : \pi_i(p) = z_i + z'(s_i)(p - s_i) \tag{7}$$

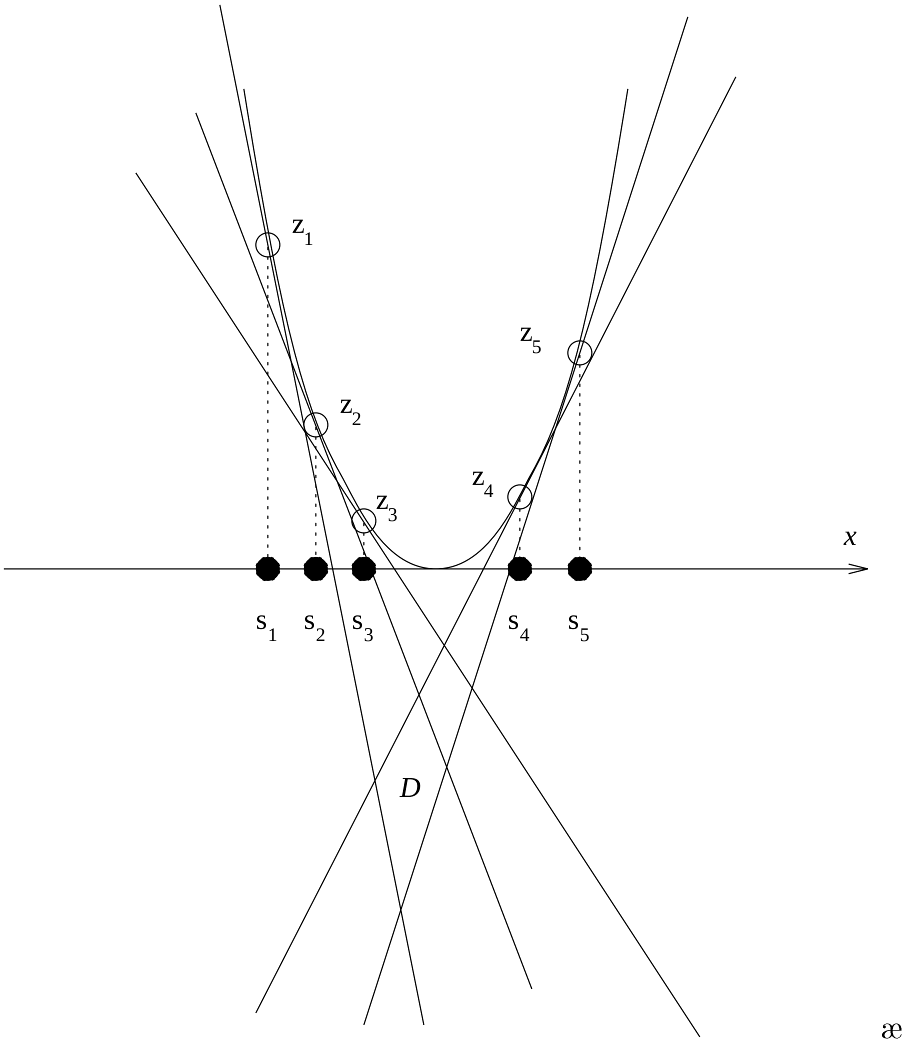
Nyní si dokážeme následující lemma:

2.28 Lemma. *Body stejně vzdálené od s_i a s_j jsou právě průmětem množiny $\pi_i \cap \pi_j$*

Důkaz: Hledáme takové body, pro které $\pi_i(p) = \pi_j(p)$. Nechť $p = (x, y)$.

$$\begin{array}{rccl}
 & x_i^2 + y_i^2 + 2x_i(x - x_i) + 2y_i(y - y_i) & = & \\
= & x_j^2 + y_j^2 + 2x_j(x - x_j) + 2y_j(y - y_j) & \Rightarrow & \\
\Rightarrow & x_i^2 + y_i^2 - 2x_i x - 2y_i y & = & / + x^2 + y^2 \\
= & x_j^2 + y_j^2 - 2x_j x - 2y_j y & \Rightarrow & / + x^2 + y^2 \\
\Rightarrow & (x - x_i)^2 + (y - y_i)^2 = (x - x_j)^2 + (y - y_j)^2 & &
\end{array}$$

tedy vzdálenost bodu p od bodů s_i, s_j je shodná. □



Obrázek 19: Konstrukce Voroného diagramů vyšších řádů na přímce

Pokračování důkazu věty 2.27:

Roviny π_i dělí $\mathbf{R}^3$ na oblasti $D_1, \dots, D_m$. Pro každý bod $s_i \in S$ definujme H_i jako ten poloprostor vymezený π_i, který neobsahuje paraboloid P. Pro každou rovinu π_i mohou nastat ve vztahu ke konkrétní oblasti D dva případy:

1. Oblast D leží celá v H_i (tedy leží v jiném poloprostoru ohraničeném pi_i než paraboloid P)

2. Oblast D neleží v H_i (a leží ve stejném poloprostoru jako P).

Obrázek 19 ilustruje situaci v rovině. Místo paraboloidu zde máme parabolu a k ní tečné přímky. H_i zde bude vždy ta polorovina, která neobsahuje parabolu a jejíž dělicí přímka odpovídá té tečně k parabole, jejíž bod dotyku je kolmou projekcí bodu x_i z osy x (která koresponduje s prostorem, ve kterém hledáme Voroného diagramy) na parabolu.

Ke každé oblasti D můžeme definovat množinu $T(D) \subseteq S$ těch s_i, pro něž je $D \subseteq H_i$. Nyní dokážeme, že průmět oblasti D zpět na rovinu xy nám dá právě Voroného oblast množiny $T(D)$.

Nechť $q' \in D$ a q je projekce q' do roviny xy. Vezmeme nyní v úvahu každou dvojici bodů $s_i, s_j \in S$ takovou, že $s_i \in T(D)$ a $s_j \notin T(D)$. Pak $q' \in H_i, q' \notin H_j$. Podle lemmatu 2.28 musí tedy být průmět q' blíže bodu s_i než s_j. To ale znamená, že $q \in H_{i,j}$. Odtud plyne podle věty 2.22, že $q \in VR(T(D))$.

Nyní musíme ještě dokázat, že není-li bod q průmětem žádného prvku oblasti D, pak není prvkem $VR(T(D))$. K tomu budeme potřebovat ještě jedno lemma:

2.29 Lemma. *Pro libovolné číslo $0 < k < n$ a pro každý bod q v rovině xy lze najít bod q' takový, že q' je prvkem nějaké oblasti D, jejíž množina $T(D)$ má právě k prvků, a q je kolmým průmětem q' na rovinu xy.*

Důkaz: V bodě q vztyčíme kolmici l na rovinu xy. Její průsečík s paraboloidem P označíme q_0. Tento bod leží v oblasti D_0, pro niž platí, že $T(D_0) = \emptyset$. Tato oblast (obsahující celý paraboloid P) je totiž vůči všem rovinám π_i v poloprostoru obsahujícím P, a tedy není v žádném H_i. Žádná rovina π_i není kolmá na rovinu xy, proto přímka l protne postupně všechny tyto roviny. Budeme-li přímku l trasovat od bodu q_0 dolů, při každém protnutí s rovinou π_i se dostaneme do poloprostoru H_i. Tím se dostaneme do oblasti D, která má o jeden více bodů ve své množině $T(D)$ než předcházející. Po protnutí všech n rovin tak získáme pro každé číslo k odpovídající vzor průmětu q'. □

Díky tomuto lemmatu můžeme dokončit důkaz věty 2.27.

Není-li tedy bod q v rovině xy průmětem žádného bodu oblasti D, je jistě průmětem jiného bodu q' oblasti D' takové, že $T(D)$ a $T(D')$ mají stejný počet prvků. Bod q je tedy prvkem $VR(T(D'))$, a nemůže být zároveň prvkem $VR(T(D))$.

$s_i \in T(D)$ jsou právě body určující $VR(T(D))$ a patří do $VD_{|T(D)|}(S)$.

Vezměme libovolný vrchol v Voroného diagramu libovolného řádu. Víme, že musí být průmětem vrcholu některé oblasti D podle předchozích úvah. Tedy pouze vrcholy těchto oblastí se mohou stát vrcholy Voroného diagramu.

Nechť v' je vrchol v rozdělení prostoru rovinami π_i. Pak v' je průnikem tří rovin (zde zanedbáme případ, kdy se čtyři takové roviny protínají v jednom bodě – čtyři body množiny S leží na kružnici). Dostáváme tedy trs tří rovin, které rozdělují prostor na osm oblastí. Navíc žádná z rovin není kolmá na rovinu xy. Dvě z těchto osmi oblastí budou obsahovat po promítnutí do roviny xy bod v' uvnitř sebe. Nazveme je „horní“, resp. „dolní“ oblast a označíme D^h, resp. D^d. Množinu $T(D^h)$ označíme jako R, počet jejích prvků nechť je $l-1$. Body odpovídající jednotlivým rovinám označíme x, y, z. Je zřejmé, že $T(D^d) = T(D^h) \cup \{x, y, z\}$.

Zbývajících šest oblastí tvoří dvě sdružené trojice. Do oblastí „vyšších“ se dostaneme z horní oblasti tak, že překročíme právě jednu ze tří rovin, které ji ohraničují. Po promítnutí těchto oblastí (případně ohraničených dalšími rovinami) dostaneme Voroného oblasti množin $R \cup \{x\}$, $R \cup \{y\}$ a $R \cup \{z\}$, podle toho, kterou rovinu jsme překročili. Obraz v bodu v' po promítnutí bude vrcholem takových Voroného oblastí, tedy vrcholem řádu l.

Analogicky získáme z dolní oblasti tři „nižší“ oblasti, které po promítnutí odpovídají Voroného oblastem množin $R \cup \{x, y\}$, $R \cup \{y, z\}$ a $R \cup \{x, z\}$. Tyto tři oblast mají také jeden z vrcholů bod v. Bod v se tedy opakuje jako vrchol řádu $l + 1$.

Celkem dostáváme, že každý vrchol bude v průmětu jednou ve $VD_l(S)$ a jednou ve $VD_{l+1}(S)$ s výjimkou těch, které se objeví poprvé ve $VD_{n-1}(S)$.

Každý vrchol Voroného diagramu je dán trojicí bodů vybraných z množiny S, jejichž odpovídající roviny se protínají ve vzoru pro tento vrchol. Takových průsečíků rovin najdeme $\binom{n}{3}$, což je $O(n^3)$. $\square$

S pomocí právě dokázané věty se lze pustit do konstrukce Voroného diagramů vyšších řádů.

Algoritmus 2.3 Nalezení Voroného diagramu libovolného řádu

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`

   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 $R \cup \{x, y, z\}$.

   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 $VR_j(T)$.

   Zjednodušeně se dá tato konstrukce popsat také takto: Vezmeme v úvahu každý vrchol v typu I. Ten je průsečíkem tří úseček (nebo polopřímek), které jsou osami úseček, spojující vždy nějakou dvojici bodů množiny S. Na obrázku 20 jsou tyto hrany souvislé čáry. My tyto čáry ve vrcholu v „obrátíme“, čímž vzniknou hrany Voroného diagramu vyššího řádu, na obrázku vyznačené přerušovaně. Vrchol v zůstane nadále vrcholem Voroného diagramu, ale stane se vrcholem typu II.

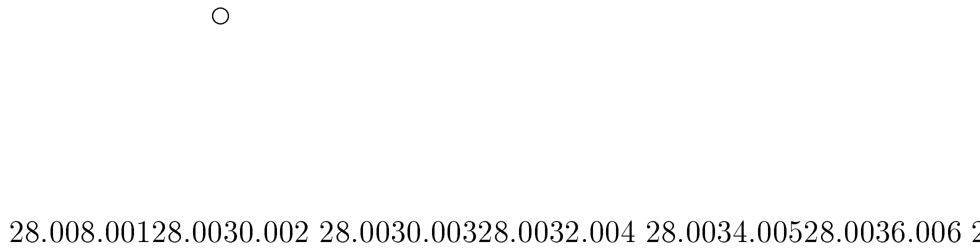
   Vzniklé obrácené hrany se mohou protínat. Najdeme-li takový průsečík, označíme ho jako vrchol typu I a obě hrany do něho směřující ukončíme. Tyto hrany budou osami úseček $L(x, y)$ a $L(x, z)$ pro nějaké tři body $x, y, z \in S$. Nechť si čtenář rozmyslí, proč tomu tak je. Z uvažovaného vrcholu nyní vypustíme novou hranu Voroného diagramu, která je osou úsečky $L(y, z)$, orientovanou tak, aby ležela ve větším úhlu svíraném původními hranami. Na obrázku 21 jsou původní hrany nakresleny plnými čárami a nová hrana přerušovaně.

Časová složitost: 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.

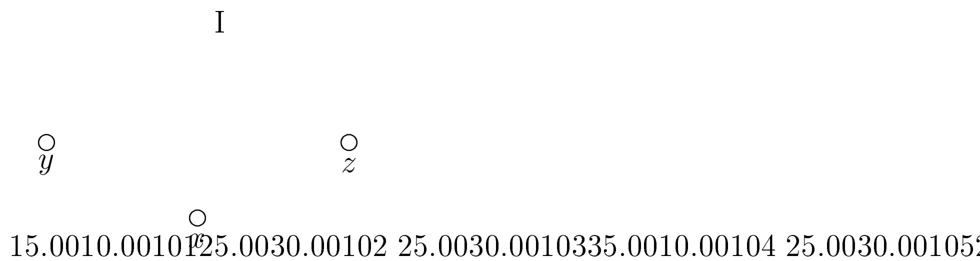
Lze ukázat, že s pro $VD_k(S)$ je nejvýše

$$2k(N-1) - k(N-1) - \textstyle\sum_{i=1}^{k} \text{ neomezených oblastí } VD_i(S)$$

2.30 Tvrzení. *Voroného diagram řádu k, $VD_k(S), |S| = n$ lze získat v čase $O(k^2 n \log n)$. Všechny $VD_k(S)$ lze takto získat v čase $O(n^3 \log n)$.*



Obrázek 20: Obrácení polopřímek ve vrcholu typu I při konstrukci Voroného diagramu vyššího řádu



Obrázek 21: Nalezení nového vrcholu typu I jako průsečíku hran

2.31 Poznámka. Optimální čas pro nalezení Voroného diagramů všech řádů je $O(n^3)$. Lze ho dosáhnout přímým výpočtem vrcholů pomocí konstrukce s paraboloidem, která byla použita v důkazu věty 2.27. Všech těchto bodů je $O(n^3)$ a algoritmus je může počítat přímo analyticky pro všechny možné trojice rovin.

Problém nejbližších sousedů lze zobecnit pro libovolný řád, stejně jako Voroného diagramy. Ve vyšších řádech však budeme uvažovat pouze dotazovou variantu problému k nejbližších sousedů z podkapitoly 2.3.

Problém Máme-li dánu množinu bodů S, číslo k a libovolný bod v rovině x, najdi těch k bodů množiny S, které jsou bodu x nejblíže.

Problém lze řešit předspočítáním Voroného diagramu daného řádu nebo všech řádů, a následným vyhledáním v takovém rovinném rozdělení. Jak uvidíme v kapitole 3.2, lze v rovinných rozděleních vyhledávat v logaritmickém čase.

2.32 Důsledek. *Problém k nejbližších sousedů lze řešit v čase $O(\log n + k)$ s předpřípravou $O(k^2 n \log n)$.*

Na závěr povídání o Voroného diagramech vyšších řádů uvedeme několik poznámek k dualitě. Voroného oblast množiny T, jak jsme ji definovali, je množina všech bodů, které jsou blíže všem bodům množiny T než libovolnému bodu množiny $S \backslash T$. To ale znamená, že je to oblast takových bodů, které jsou dále všem bodům množiny $S \setminus T$ než libovolnému bodu množiny T. Problém nejvzdálenějších sousedů je tedy duální k problému nejbližších sousedů a dá se na něho převést. Hledáme-li totiž k nejvzdálenějších sousedů bodu x mezi body množiny S, vyhledáme oblast $VR_{n-k}(T)$ v diagramu $VD_{n-k}(S)$ a výsledkem je množina $S \setminus T$. Zejména diagram $VD_{n-1}(S)$ lze použít při hledání nejvzdálenějšího souseda.

2.33 Důsledek. *Problém k nejvzdálenějších sousedů lze řešit v čase $O(\log n + (n-k))$ s předpřípravou $O((n-k)^2 n \log n)$.*

Je tedy Voroného diagram $VD_{n-1}(S)$ duální k $VD_1(S)$. Vezmeme-li poloroviny $H_{i,j}$ z lemmatu 2.2 definované

$$H_{i,j} := \{y \in \mathbf{R}^2 \mid dist(x_i, y) \leq dist(x_j, y)\}$$

pro každou dvojici bodů $x_i, x_j \in S$ a z nich dostaneme Voroného oblast prvního řádu jako $VR(x_i) = \bigcap_{i \neq j} H_{i,j}$, lze duálně definovat opačné poloroviny

$$\overline{H_{i,j}} := \{y \in \mathbf{R}^2 \mid dist(x_i, y) \geq dist(x_j, y)\}$$

a potom analogicky platí $VR_{n-1}(x_i) = \bigcap_{i \neq j} \overline{H_{i,j}}$[5]. Ve výsledném Voroného diagramu řádu $(n-1)$ budou však existovat oblasti pouze pro body na konvexním obalu celé množiny S, což si může čtenář dokázat jako cvičení.

[5]Zřejmě zde platí vztah $H_{i,j} = \overline{H_{j,i}}$. Pro lepší ilustraci duality však uvádíme tuto notaci.

2.7 Voroného diagramy ve vyšších dimenzích

Poslední možností rozšíření Voroného diagramů, kterou se budeme zabývat, je zvětšení dimenze našeho prostoru. Jak dále uvidíme, s dimenzí roste rychle složitost Voroného diagramu.

Už v dimenzi $d = 3$ může mít Voroného diagram $VD_1(S), |S| = n$ až $O(n^2)$ hraničních stěn. Např. [Preparata-85] uvádí následující příklad:

2.34 Příklad. Mějme množinu n bodů v prostoru se sudým počtem prvků n. Nechť polovina z nich je rozmístěna na ose z a druhá polovina na kružnici ležící v rovině xy se středem v bodě $[0, 0, 0]$. Voroného oblasti patřičné k bodům na kružnici mají za sousedy všechny oblasti bodů na přímce a naopak. Celkem je tedy $O(n^2)$ hran výsledného diagramu, přestože má $O(n)$ oblastí.

Časy dosažené v rovině tedy v prostoru nutně selhávají.

Naznačíme nyní postup konstrukce Voroného diagramu v prostoru. Uděláme to tak, že popíšeme konstrukci v rovině s tím, že její rozšiřitelnost o jednu dimenzi bude intuitivně jasná. Takový postup volíme proto, že je při konstrukci nutno pracovat v prostoru o dimenzi vyšším než je prostor výsledného diagramu. Všechny následující úvahy tedy povedou k diagramu v rovině.

Konstrukce bude probíhat tak, že rovinu vložíme do prostoru jako rovinu rovnoběžnou s rovinou xy na úrovni $z = 1$. Pomocí zobrazení *inverze vzhledem ke sféře* promítneme všechny množiny S z této roviny na sféru. Potom dokážeme úzkou souvislost mezi Voroného diagramem původní množiny a konvexním obalem obrazů bodů této množiny.

Body v prostoru bude potřeba vložit do čtyřrozměrného prostoru a zde najít konvexní obal jejich obrazů v inverzi. Všechny tyto operace umíme analyticky provést, přestože si je neumíme představit.

2.35 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*

$$k \equiv x^2 + y^2 + z^2 = 1$$

2.36 Poznámka. Koule k je v inverzi z předchozí definice právě množinou samodružných bodů.

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

Důkaz: 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. □

Uvažme 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 $1/2$. Uvažme body $p_i \in \eta, i = 1, \ldots, n$ (množina S). Označme $z_i = \varphi(p_i) \in \varphi(\eta)$, $S' = \{z_1, \ldots, z_n\}$. Konvexní obal $BCH(S')$ rozdělíme na disjunktní části C_1 – body viditelné z počátku O a C_2 – neviditelné přes BCH. Protože žádné čtyři body v S neleží na kružnici, jsou v $BCH(S')$ samé trojúhelníky.

1. 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 $VD_1(S)$.

2. 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 $VD_{n-1}(S)$.

Výše uvedený 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 $O(n \log n)$. Viz poznámka 1.25 a algoritmus typu rozděl a panuj v $\mathbf{R}^3$.

Navíc musíme rozdělit vrcholy v $BCH(S')$ na C_1, C_2 podle „viditelnosti“ z počátku. Prakticky test provedeme snadno výpočtem orientovaného objemu příslušného rovnoběžnostěnu.

2.38 Poznámka. Z konstrukce $VD_{n-1}(S)$ plyne, že neprázdné oblasti přísluší (v interpretaci nejvzdálenějších bodů) právě bodům na hranici konvexního obalu ($BCH(S)$).

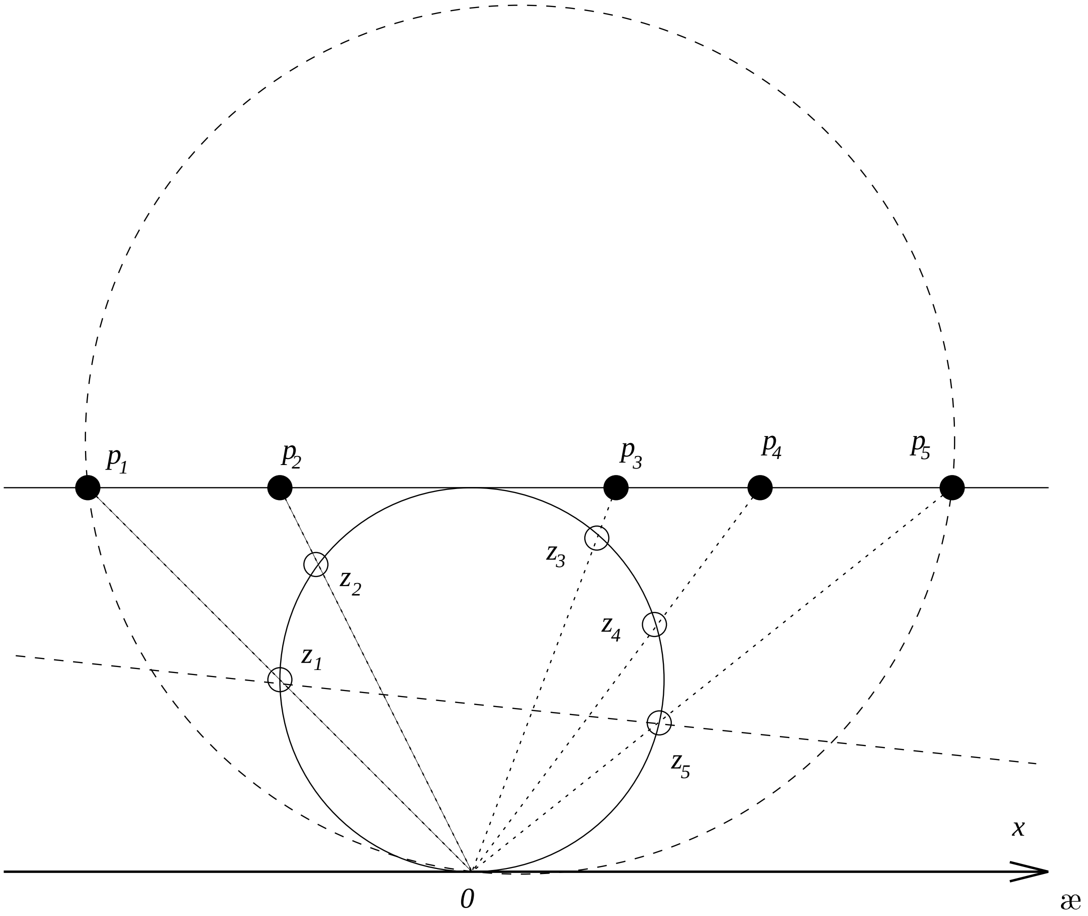
2.8 Díry a pokrytí

Krátce se zmíníme o sadě problémů souvisejících s Voroného diagramy, které se dají souhrnně označit jako *díry a pokrytí*.

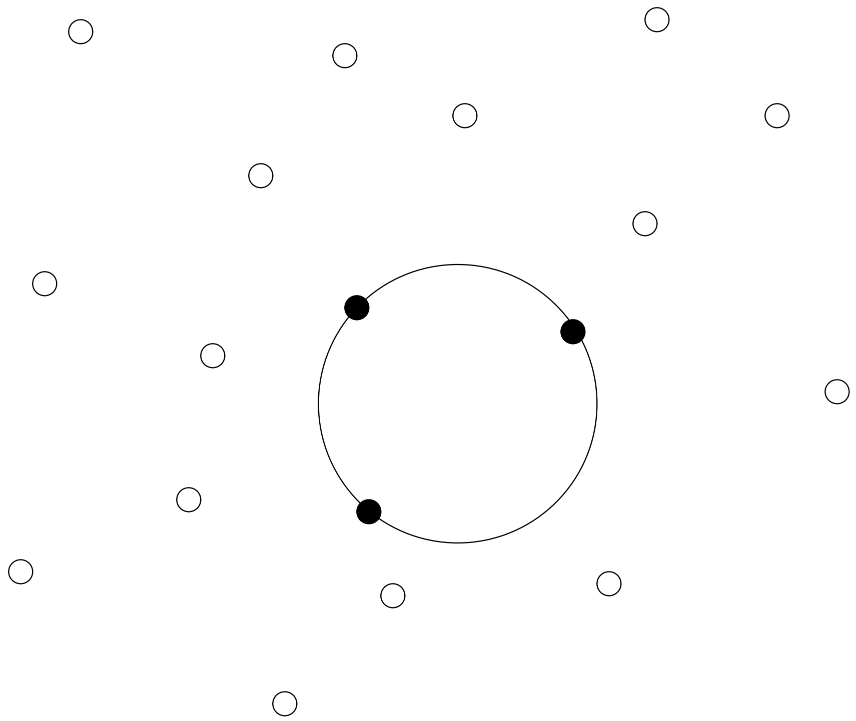
2.39 Definice. *Dírou* pro $S \subset \mathbf{R}^2$ rozumíme kruh neobsahující žádný bod množiny S.

Pokrytí je množina kruhů pokrývajících společně celou množinu S.

Budeme se zabývat pouze dvěma problémy. Možností je však více, zájemce odkazujeme na [Preparata-85].



Obrázek 22: Ilustrace kruhové inverze v dimenzi 2



Obrázek 23: Největší prázdný kruh

Problémy

1. Najdi největší díru pro množinu S, jejíž střed leží uvnitř konvexního obalu množiny S (největší prázdný kruh – viz obrázek 23)

2. Najdi nejmenší pokrytí množiny S skládající se z jediného kruhu (nejmenší pokrývající kruh – viz obrázek 24)

Na obou ilustracích jsou body, které určují výsledný kruh, vyznačeny plným kroužkem, ostatní body množiny prázdným kroužkem.

2.40 Poznámka. Nejmenší pokrývající kruh je jednoznačně určen a buď prochází třemi body z S nebo jeho průměr je průměrem S. Obě možnosti ukazuje obrázek 24.

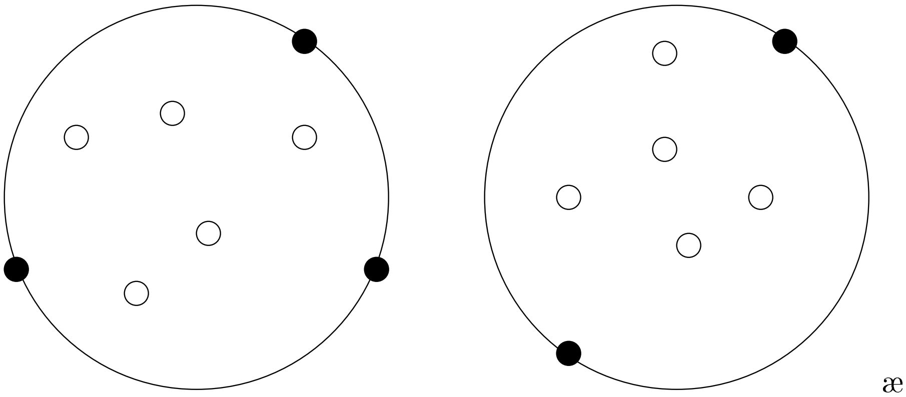
Problém největšího prázdného kruhu je znám v operačním výzkumu jako *Minimum Facilities Location Problem*.

Problém 2 je duální k 1. Oba byly studovány od roku 1860 a ještě v 70tých letech tohoto století s výsledky

1. Největší prázdný kruh: $O(n^2)$

2. Nejmenší pokrývající kruh: $O(n^3)$

S pomocí Voroného diagramů dosáhneme pro oba problémy následujících časů.



Obrázek 24: Dvě možnosti pro nejmenší pokrývající kruh

2.41 Věta. *V obou případech obsahuje vstupní množina S n bodů.*

1. *Nejmenší kruh pokrývající S lze sestrojit v čase $O(n \log n)$*

2. *Největší prázdný kruh se středem uvnitř $CH(S)$ lze získat v čase $O(n \log n)$*

Důkaz: Nejmenší pokrývající kruh: Pokud kruh prochází třemi body z S, pak jeho střed je vrcholem $VD_{n-1}(S)$. Takových vrcholů je lineárně mnoho. Nechť kruh „realizuje“ průměr S. Pak jeho střed je na ose úsečky $L(x_i, x_j)$. Část této osy musí být hranou ve $VD_{n-1}(S)$. Tedy stačí projít hrany $VD_{n-1}(S)$ a hledat nejvzdálenější dvojici bodů z S odpovídající těmto hranám.

Největší prázdný kruh: V tomto případě je střed kruhu nutně vrcholem $VD_1(S)$ nebo některým z nově vzniklých vrcholů $VD_1(S) \cap BCH(S)$. My však víme, že

- každá hrana $VD_1(S)$ protíná nejvýše dvě hrany $BCH(S)$

- každá hrana $BCH(S)$ protne alespoň jednu hranu $VD_1(S)$

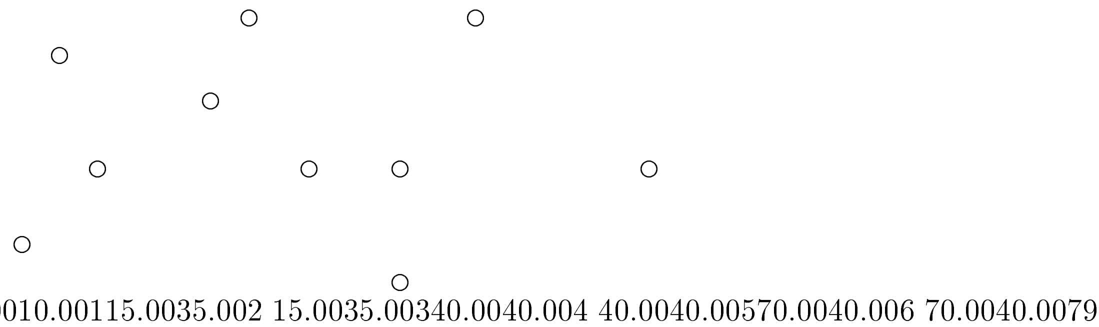
Každou hranu $VD_1(S)$ projdeme nejvýše dvakrát a najdeme všechny průniky v čase $O(N)$. V lineárním čase také najdeme všechny vrcholy $VD_1(S)$. Nejvíce času tedy zabere sestrojení $VD_1(S)$ a $BCH(S)$, což lze oboje v čase $O(n \log n)$. □

2.42 Poznámka. Pan Megiddo ([Megiddo-83]) vylepšil hledání největšího prázdného kruhu tak, že prohlíží a konstruuje jen část $VD_{n-1}(S)$ a jeho algoritmus pracuje v optimálním čase $\Theta(n)$.

3 Triangulace a vyhledávání v rovinných rozděleních

V této kapitole se budeme zabývat nejprve triangulací množiny bodů v rovině. Získáme tak rozdělení roviny na trojúhelníkové oblasti.

Budeme dále požadovat, abychom v takto vytvořené struktuře mohli efektivně vyhledávat, tedy pro daný bod zjistit, v které části triangulace leží. Tuto úlohu, zvanou *vyhledávání v rovinných rozděleních*, zobecníme na dělení roviny na mnohoúhelníkové oblasti, a tak získáme zároveň algoritmy na vyhledávání ve Voroného diagramech uvedených v předchozí kapitole.



Obrázek 25: Triangulace bodů v rovině

3.1 Triangulace

Běžnou úlohou v mnoha geometrických aplikacích je nalezení *triangulace* množiny bodů v rovině. Jde o vytvoření sítě trojúhelníků, které se nepřekrývají a přitom pokrývají celý vnitřek konvexního obalu dané množiny. Přitom vrcholy trojúhelníků mají být identické s body vstupní množiny. Jinými slovy, hledáme rovinný graf s vrcholy identickými s body dané množiny, přičemž stěny tohoto grafu mají být trojúhelníky. Obrázek 25 ukazuje příklad takové triangulace.

V předchozí kapitole jsme pomocí Voroného diagramu nalezli tzv. *Delaunayovu triangulaci* (viz definice 2.4, následující věta a obrázek 13). Naučili jsme se ji také konstruovat v čase $O(n \log n)$.

Pro danou množinu bodů však jistě existuje mnoho triangulací. Proto se často na triangulační algoritmy aplikují různé požadavky.

Běžné požadavky na triangulaci:

- minimalizovat váhy hran (např. délky)

  Není známo, zda existuje polynomiální algoritmus

- minimalizovat úhly trojúhelníků

  Není známo, zda existuje polynomiální algoritmus

- máme předem zadánu množinu „povinných" hran v triangulaci

  Uvedeme dva algoritmy

Je dokázáno, že Delaunayova triangulace minimalizuje maxima úhlů v trojúhelnících. Kromě toho má tato triangulace další příjemnou vlastnost: uvnitř kružnice opsané každému jejímu trojúhelníku neleží již žádný další vrchol triangulace.

Naivní implementace triangulace (tzv. *hladová*):

Algoritmus 3.1 Hladová triangulace (Gready Triangulation)

1. $\binom{n}{2}$ možných hran uspořádáme dle velikosti do zásobníku

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. $O(n^2 \log n)$

2. Je-li $\varphi(m)$ doba potřebná pro test kompatibility grafu o m vrcholech, pak čas 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 $O(n^3)$

Lepší test kompatibility, který pracuje v čase $O(\log n)$, byl navržen v [Gilbert-79], a je uveden také v [Preparata-85]. Přitom nedojde ke zvýšení paměťových nároků. My zde uvedeme pouze výsledek.

3.1 Tvrzení GILBERTOVO. *Hladová triangulace pro n bodů v rovině se dá provést v čase $O(n^2 \log n)$ a prostoru $O(n^2)$.*

3.2 Poznámka. Hladová triangulace neřeší optimálnost hran. Může však zahrnout předem zadanou množinu hran povinných.

Hladová triangulace nás co do časové složitosti nemůže uspokojit. Uvedeme dále algoritmus lepší, který se dá nazvat *triangulace pomocí monotónních řetězců.*

Princip spočívá v rozdělení úlohy na postupné triangulování jednodušších oblastí. Pro $S \subset \mathbf{R}^2$, $|S| = n$, chceme triangulaci konvexního obalu množiny S. Rozdělíme si jej tedy na tzv. *monotónní polygony* a provedeme jejich triangulaci.

3.3 Definice. Polygon P se nazývá *monotónní vzhledem k přímce l*, je-li jednoduchý a je sjednocením dvou monotónních řetězců vzhledem k l (tzn. průměty vrcholů na l tvoří monotónní posloupnost).

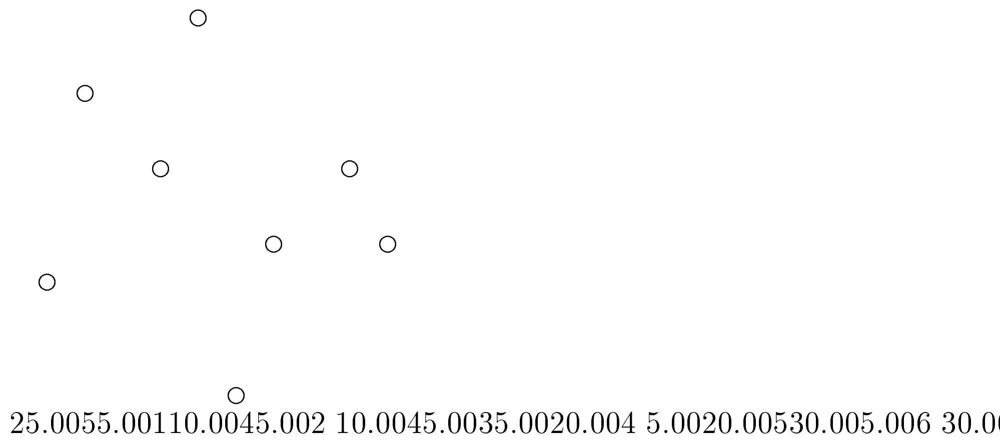
Nejprve sestrojíme algoritmus na rozdělení $CH(S)$ na monotónní polygony s vrcholy v S. Konstruujeme posloupnost monotónních řetězců s vlastnostmi

1. řetězce obsahují všechny body S a všechny zadané hrany

2. pro každé řetězce P_i, P_j platí, že všechny vrcholy P_j, které nepatří do P_i, leží na stejné straně od P_i

3.4 Definice. Systém řetězců výše popsaný nazýváme *monotónní úplná množina řetězců* (pro rovinný graf G).

Příklad monotónní úplné množiny řetězců je na obrázku 26.



Obrázek 26: Monotónní úplná množina řetězců

3.5 Definice. Nechť G je rovinný přímočarý graf s vrcholy $V_G = \{v_1, \dots, v_n\}$ uspořádanými lexikograficky podle $[y, x]$. *Vrchol* v_j v G nazveme *regulární*, pokud existuje $i < j < k$ tak, že hrany (v_i, v_j) a (v_j, v_k) patří do G. *Graf* G nazveme *regulární*, je-li každý vrchol $v_j, 1 < j < n$ regulární. Hrany $(v_i, v_j), i < j$, označujeme jako *vstupní pro* v_j ($\in IN(v_j)$), resp. *výstupní pro* v_i ($\in OUT(v_i)$).

Předpokládejme, že v G máme vybrány monotónní řetězce spojující v_n s v_1 a označme $w(e)$ jako počet řetězců, ve kterých hrana e leží (váha e). Klademe

$$W_{IN}(v) := \sum_{e \in IN(v)} w(e)$$

$$W_{OUT}(v) := \sum_{e \in OUT(v)} w(e)$$

3.6 Poznámka. Pokud máme naopak zadány váhy hran vzniklé z množiny řetězců, umíme sestrojit nějakou množinu řetězců kompatibilní s vahami. Pokud systém byl úplný, opět získáme úplný systém. Postupně z v_n tvoříme cesty do v_1 tak, že z uspořádaných množin $IN(v_i)$ (podle hodinových ručiček) zvolíme poslední jako následující hranu cesty a snížíme její váhu o 1.

Díky regulárnosti G umíme pro každý vrchol $v \in G$ najít monotónní řetězec procházející v. Tomuto algoritmu budeme říkat *balancování vah*, protože najdeme váhy pro nějakou úplnou množinu řetězců, z nichž se dají podle předchozího bodu tyto řetězce zkonstruovat v lineárním čase.

Potřebujeme „přeložit“ vlastnosti monotónní úplné množiny řetězců do „jazyka vah“. Následující vlastnosti vah nám zaručí, že váhy budou popisovat nějakou monotónní úplnou množinu řetězců.

1. $w(e) \geq 1$

   ...říká, že každá hrana i každý vrchol se objeví v množině řetězců (úplnost)

2. $\forall v_j \in G, 1 < j < n : W_{IN}(v_j) = W_{OUT}(v_j)$

…zajišťuje, že půjde o monotónní řetězce, tedy kolik jich do vrcholu vstoupí „zespodu“, tolik jich bude pokračovat „výše“

Následuje algoritmus, který v regulárním grafu spočítá takové váhy, které budou reprezentací nějaké monotónní úplné množiny řetězců.

Algoritmus 3.2 Balancování vah

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.

Uvedli jsme algoritmus na nalezení množiny řetězců regulárního grafu. Budeme však na celkovém algoritmu požadovat, aby pracoval s libovolně zadanou množinou povinných hran. Tento neúplný graf je třeba nejprve regularizovat, abychom mohli výše uvedený algoritmus aplikovat.

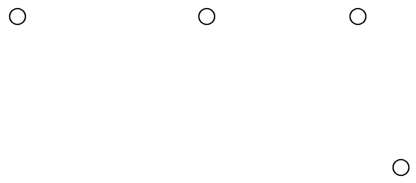
Regularizaci grafu provedeme metodou *pročesávání*, která již byla zmíněna v kapitole 2.2. Metoda pročesávání (sweep) potřebuje následující paměťové struktury:

- (vertikální) struktura pro zachycení událostí pro nás významných

  …uspořádané vrcholy od shora dolů (za předpokladu, že žádné dva z nich nemají shodnou y-ovou souřadnici)

- (horizontální) struktura pro zachycení statutu *pročesávací přímky*

  …seznam průsečíků přímky uspořádaný zleva doprava a každému intervalu na přímce přiřadíme vrchol s nejnižší y-ovou souřadnicí „nad ním“



70.0055.00180.005.00215.006.00326.0025.004 26.0025.00535.006.006 26.0025.0075.0050.0

Obrázek 27: Regularizace grafu – pročesávací algoritmus

Při průchodu shora dolů zajistíme, aby z každého vrcholu vedla alespoň jedna hrana „nahoru“, nebo-li $OUT(v_i) \neq \emptyset$. Analogicky po průchodu zdola nahoru povede z každého vrcholu alespoň jedna hrana „dolů“. Obrázek 27 ilustruje průchod shora dolů a tři pozice pročesávací přímky, která je vždy horizontální. Na této přímce budou vždy intervaly s tzv. přiřazeným vrcholem (s nejnižší y-ovou souřadnicí nad tímto intervalem). Pročesávací přímka v nejvyšší pozici je rozdělena na pět intervalů. U tří prostředních je naznačen vrchol, který je jim přiřazený. V prostřední pozici prochází pročesávací přímka vrcholem, mění se zde její stav. Intervaly pod body v_j, v_k jsou nejprve vyloučeny dle množiny hran $OUT(v_i)$. Poté je vložen nový interval pod vrcholem v_i. Takový je také stav pročesávací přímky ve třetí pozici.

Algoritmus 3.3 Regularizace grafu
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 ($O(n)$).

3.7 Věta. *Pro danou množinu bodů v rovině $S = \{v_1, \dots, v_n\}$ lze najít rozdělení konvexního obalu $CH(S)$ monotónní úplnou množinou řetězců zahrnující všechny předem dané hrany M v čase $O(n \log n)$ a prostoru $O(n)$.*

Důkaz: V čase $O(n \log n)$ seřadíme vrcholy a ve stejném čase provedeme regularizaci grafu. V čase $O(n)$ pak provedeme balancování vah a zároveň sestrojíme požadované řetězce. □

3.8 Věta. *Monotónní polygon s n hranami lze triangulovat v čase $O(n)$.*

Důkaz: V lineárním čase sestrojíme setřídění dvou hraničních monotónních řetězců, nechť $u_1, \dots, u_n$ je výsledná posloupnost. Pak $\forall u_i, 1 \le i \le n-1 \exists j > i$ tak, že (u_i, u_j) je hrana v P.

Idea algoritmu: v průchodu shora dolů lze přidávat „diagonály“. Užijeme zásobník s přístupným dnem (k nahlédnutí). V zásobníku bude vždy část hranice P tak, že u_j, u_{j+1}, u_{j+2} tam patří, jestliže

$$\angle(\overline{u_j u_{j+1}}, \overline{u_{j+2} u_{j+1}}) \ge 180^o$$

V zásobníku jsou uloženy vrcholy (uspořádané podle y-ové souřadnice), které už byly navštíveny, ale stále ještě se pro ně nenašla vhodná diagonála.

Algoritmus 3.4 TRIANGULACE MONOTÓNNÍHO POLYGONU

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 $v_1 := \mathit{bottom}(\text{STACK})$ ale nesousedí s $v := \mathit{top}(\text{STACK})$ `then`
     - 2.1.1. přidej hrany $(u_i, v_2), (u_i, v_3), \dots, (u_i, v)$
     - 2.1.2. smaž zásobník a vlož do něho prvky v, u_i
   - 2.2. `elsif` u_i sousedí s $v := \mathit{top}(\text{STACK})$ ale nesousedí s $v_1 := \mathit{bottom}(\text{STACK})$ `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 $(u_i, v_{bottom+1}), \dots, (u_i, v_{top-1})$ /* algoritmus končí

Na obrázku 28 jsou v částech a), b) a c) ilustrovány polohy vrcholů diskutované postupně v bodech 2.1, 2.2 a 2.3 algoritmu. Plné čáry jsou vždy hrany už dříve známé a přerušované čáry značí hrany právě přidávané. Podrobnou diskusi správnosti algoritmu přenecháváme čtenáři. □

55.0010.00190.0015.002 90.0015.00385.0025.004 55.0010.00559.0012.006 61.0013.00765.0015.008 6

Obrázek 28: Možné pozice vrcholů monotónního polygonu diskutované při jeho triangulaci

3.9 Věta. *Triangulaci konvexního obalu množiny $S \subset \mathbf{R}^2$, $|S| = n$ s předem zadanou množinou hran $T \subset S^2$ lze provést v optimálním čase $O(n \log n)$ a prostoru $O(n)$.*

Důkaz: Podle věty 3.7 úplná množina monotónních řetězců dělící celý $CH(S)$ na lineárně mnoho monotónních polygonů s hraničními řetězci bez splývajících hran se najde v čase $O(n \log n)$. Přitom lze zároveň odebírat vznikající monotónní polygony. Pak každá hrana je nejvýše ve dvou polygonech, tj. celkem potřebujeme čas

$$\begin{array}{ccccccc} BCH & & \text{regularizace} & & \text{řetězce} & & \text{triangulace} \\ O(n \log n) & + & O(n \log n) & + & O(n) & + & O(n) \end{array}$$

□

3.2 Vyhledávání v rovinných rozděleních

Anglický termín je *Geometric Searching*. Jde v podstatě o problém typu dotazu na příslušnost bodu v rovině (daného dotazem) k buňce rovinného rozdělení (daného předem).

Jinými slovy, máme dáno pevné (ve smyslu okamžiku dotazu) rovinné rozdělení. Na vstup přicházejí dotazy ve formě bodu a výstupem je buňka rozdělení, ve které tento bod leží.

Rozdělení se však může čas od času měnit, mluvíme pak o dynamické struktuře.

Vyhledání lze řešit procházením celé struktury v lineárním čase. Dotazy jsou však častou operací, a budeme od nich asi očekávat čas lepší.

Budeme používat tato kriteria:

- čas potřebný pro odpověď (průměrný, nejhorší)
- spotřebovaná paměť
- čas potřebný na předpřípravu (konstrukce paměťových struktur)
- čas potřebný na úpravu struktur při změně dat (dynamická struktura)

3.10 Definice. *Rovinné rozdělení* je přímočaré vnoření rovinného grafu do $\mathbf{R}^2$. *Zobecněné rovinné rozdělení* může navíc obsahovat neohraničené mnohoúhelníkové oblasti.

Takové definici odpovídají všechny triangulace a Voroného diagramy. Naučíme se tedy vyhledávat mimo jiné v nich.

Uvedeme nyní tři možné přístupy k řešení problému.

3.2.1 Metoda pásů

Idea Setřídíme vrcholy podle y-ové souřadnice. Tím se $\mathbf{R}^2$ rozpadne na horizontální pásy. Průnik grafu G s jedním pásem se sestává z úseček, které jsou částmi hran v G. Tyto se navíc neprotínají (mohou mít jeden společný bod na hranici pásu). Proto můžeme setřídit úsečky (např. odleva doprava). Tím získáme zjemnění G na „kachličky“ lexikograficky uspořádané. V této struktuře již můžeme efektivně vyhledávat.

Počet kachliček však může být až kvadratický, jak ukazuje příklad na obrázku 29. Tak obrovské nároky na paměť jsou pro aplikace s velkým množstvím dat nepřijatelné.

5.005.00110.0055.002 10.0055.00315.0010.004 15.0010.00520.0050.006 20.

Obrázek 29: Pro tento vzor rozdělení roviny na dvě části roste počet kachliček kvadraticky oproti počtu vrcholů

Přesto umíme postavit vyvážený strom pro $O(n^2)$ kachliček, ve kterém se dá vyhledávat v čase $O(\log(n^2)) = O(\log n)$. Čas přípravy je ohraničen počtem kachliček, a bude tedy také kvadratický.

Při konstrukci kachliček lze využít pročesávání, kde struktura událostí je uspořádaný seznam vrcholů a statut pročesávací přímky jsou hrany uchovávané ve $(2,3)$ stromu.

3.11 Tvrzení. *Metodou pásů lze vyhledávat v rovinném rozdělení s n vrcholy v čase $O(\log n)$ s pamětí $O(n^2)$ a časem přípravy $O(n^2)$.*

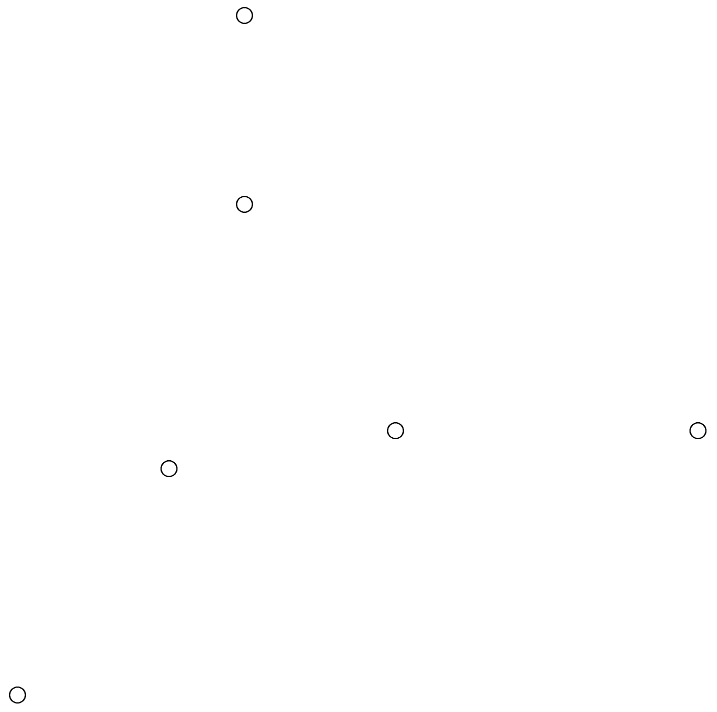
3.2.2 Metoda cest

Jestliže použijeme úplnou monotónní množinu řetězců v grafu G (tzv. cesty), dostaneme algoritmus, který bude vyhledávat v čase $O(\log^2 n)$, potřebná paměť bude $O(n)$ a čas přípravy $O(n \log n)$.

- Algoritmus regularizace 3.3 nepřidává vrcholy, výsledných hran je $O(n)$, potřebný čas je $O(n \log n)$.
- Algoritmus pro nalezení cest pracuje v čase $O(n)$. Jestliže prohledáme cesty zleva doprava, pak v každé cestě přibereme alespoň jednu novou hranu, tedy cest je $O(n)$.
- Lineárním hledáním najdeme dvě následující cesty, mezi kterými se hledaný bod nachází, proto čas vyhledání je $O(\log^2 n)$.

Kdybychom si pamatovali všechny cesty jako posloupnosti hran, dostali bychom kvadratickou paměťovou složitost. Dá se však využít faktu, že hran i cest je dohromady lineárně mnoho, a při malém snížení časového výkonu také dosáhneme paměti $O(n)$.

Budeme si pamatovat cesty v binárním stromu, a každou hranu v tomto stromu uložíme pouze pro jednu cestu, a to pro tu, která je ve stromu nejvýše z těch cest, ve kterých se daná hrana nachází. Tak zajistíme lineární spotřebu paměti. Při vyhledávání pak ale budeme muset



48.00100.00118.0010.002 19.0010.00349.0075.004 49.0075.00549.00100.006 50.00100.00750

Obrázek 30: Graf, na němž ilustrujeme implicitní reprezentaci cest

možná prohledávat nejen hrany alokované dané cestě, ale i všem cestách výše ve stromu. Těchto cest bude $O(\log n)$, a v každé bychom prováděli vyhledávání v čase $O(\log n)$, proto můžeme očekávat celkový čas $O(\log^2 n)$. Skutečná realizace bude trochu odlišná, bude ale pracovat v uvedeném čase a prostoru.

3.12 Definice. *Implicitní reprezentace cest* je tabulka uspořádaných dvojic $(L(e), R(e))$ pro hrany $e \in G$ s vlastností, že e patří právě do cest P_j takových, že $L(e) \leq j \leq R(e)$.

3.13 Poznámka. Tabulku implicitní reprezentace cest lze zkonstruovat v lineárním čase přímo v algoritmu balancování cest (3.2).

Obrázek 30 a tabulka 2 ilustrují tabulku implicitní reprezentace cest pro šest bodů v rovině.

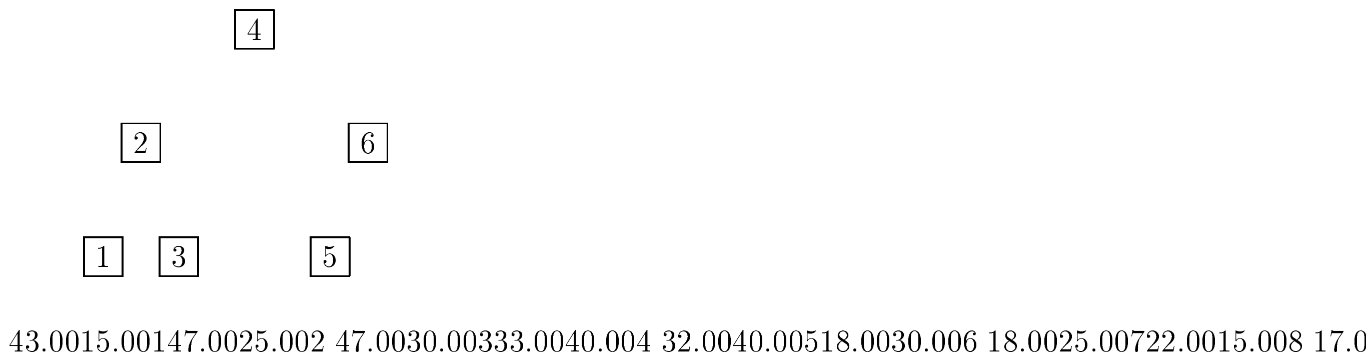
3.14 Definice. *Redukovaná vyhledávací struktura* (RVS) je vyvážený strom obsahující v každém uzlu i vyhledávací strom T_i pro y-ové souřadnice koncových bodů všech hran e, pro které je i-tá cesta nejvýše položenou cestou ve stromu, ve které se nachází. Pro jednotlivé hrany označíme jejich výskyt ve stromu jako $Pos(e)$ (viz obrázek 30 a 31).

3.15 Věta. *Vyhledávání v redukované vyhledávací struktuře probíhá v čase $O(\log^2 n)$, prostoru $O(n)$ a přípravném čase $O(n \log n)$.*

cesta 1 : 0 e_1 5
cesta 2 : 0 e_2 1 e_4 5
cesta 3 : 0 e_2 1 e_5 4 e_{12} 5
cesta 4 : 0 e_2 1 e_6 3 e_{10} 4 e_{12} 5
cesta 5 : 0 e_2 1 e_7 2 e_8 3 e_{11} 5
cesta 6 : 0 e_3 2 e_9 5

hrana e	$L(e)$	$R(e)$	$Pos(e)$
e_1	1	1	1
e_2	2	5	4
e_3	6	6	6
e_4	2	2	2
e_5	3	3	3
e_6	4	4	4
e_7	5	5	5
e_8	5	5	5
e_9	6	6	6
e_{10}	4	4	4
e_{11}	5	5	5
e_{12}	3	4	4

Tabulka 2: Popis průběhu cest v grafu a tabulka implicitní reprezentace cest



Obrázek 31: Redukovaná vyhledávací struktura

Důkaz: Pokud umíme získat $Pos(e)$ v čase alespoň $O(n \log n)$, umíme sestrojit RVS v tomto čase též. Následující algoritmus založený na vyjádření ve dvojkové soustavě to zvládne v čase logaritmickém.

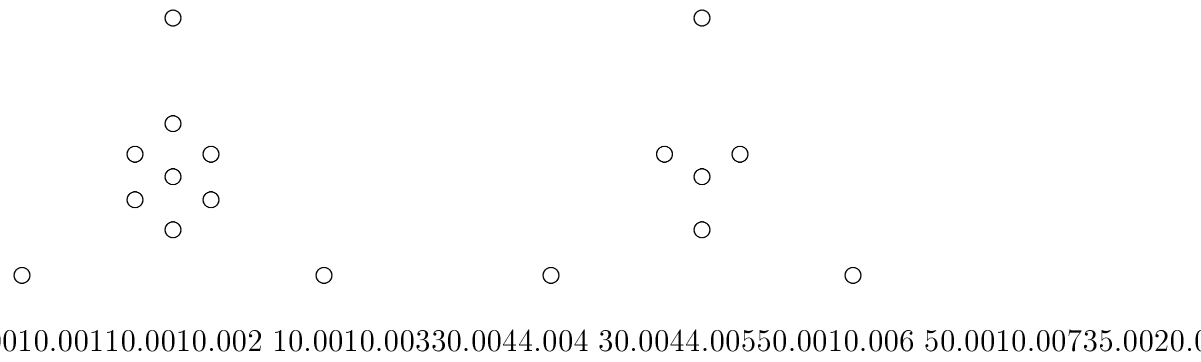
Algoritmus 3.5 ZJIŠTĚNÍ $Pos(e)$ PRO HRANU e
Vstup: hrana e a hodnoty $L(e), R(e)$
Výstup: hodnota $Pos(e)$

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

Časovou složitost vyhledávání a prostor jsme již odvodili dříve. Nyní nabízíme algoritmus takového vyhledávání.

Algoritmus 3.6 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`



Obrázek 32: Vypuštění množiny nezávislých vrcholů v grafu

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

Invariant cyklu 7: q leží mezi P_l, P_r a L protíná e_l, e_r. Protože začneme s $l = 0$ a $r = k + 1$, je podmínka v 7 poprvé splněna a m ukazuje na kořen RVS. V dalších průchodech cyklem 7 se pak posunujeme od kořene k listům. Tím je zajištěno, že průnik L s P_m můžeme vždy hledat v příslušném vyhledávacím stromu T_m. □

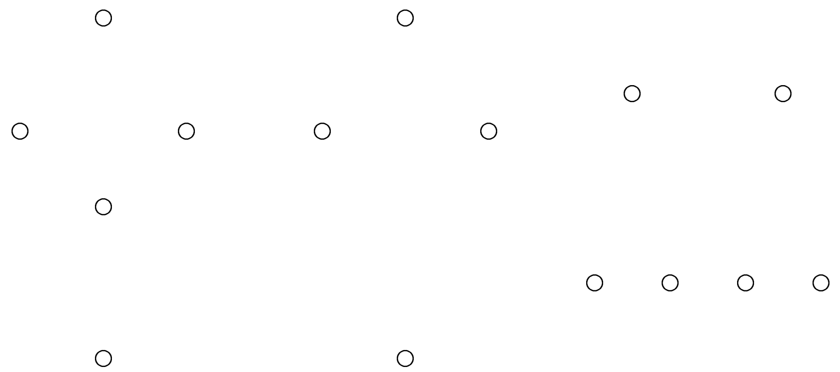
3.16 Poznámka. [Preparata-85] uvádí způsob, jak dosáhnout v RVS vyhledávacího času $O(\log n)$ při zachování paměti na $O(n)$. Ve vyhledávací struktuře se přidají tzv. *mosty*, což jsou ukazatele mezi jednotlivými hranami alokovanými v různých uzlech nadřazeného stromu (odtud název *přemosťování*). Pomocí takových mostů se lze z uzlu odpovídajícímu dané cestě dostat ke všem hranám a nemusí se provádět hledání ve všech uzlech výše ve stromu.

3.2.3 Metoda postupného zjemňování

Každé rovinné rozdělení umíme triangulovat tak, že nepřidáváme nové vrcholy a zachováváme dané hrany, a to v čase $O(n \log n)$ a prostoru $O(n)$. Můžeme se tedy v dalším omezit na tzv. *jednoduchá rozdělení*, tj. taková, že všechny buňky rozdělení jsou trojúhelníky včetně neohraničených.

Postupně procházíme velké množiny nezávislých vrcholů v grafu (tedy těch, které nesdílí hranu), tyto vypustíme a výsledek znovu triangulujeme (viz obrázek 32). Po jistém počtu kroků dostaneme jediný trojúhelník (abychom to zajistili, musíme přidat tři nové body, jejichž trojúhelník bude obsahovat všechny ostatní body).

Při odstraňování vrcholů budeme mezi trojúhelníky ve staré a nové triangulaci, které spolu incidují, udržovat ukazatele. Můžeme tak vytvořit strom, jehož kořenem bude už zmíněný trojúhelník obsahující všechny ostatní body. Každá jednotlivá úroveň stromu bude reprezentovat jednu triangulaci, uzly budou trojúhelníky. S hloubkou stromu budou přibývat vrcholy, až na



Obrázek 33: Část vyhledávací struktury pro zjemňování

poslední úrovni bude prvotní triangulace. Hrany ve stromu budou odpovídat relaci incidence, a povedou vždy mezi uzly sousední úrovně. Nepůjde však o typický strom, protože do uzlu může vést více hran z uzlů předchozí úrovně. Část takového stromu ukazuje obrázek 33. Vyhledávání bude začínat v kořeni a pokračovat pouze po hranách.

Nyní odvodíme složitost tohoto algoritmu.

3.17 Lemma. *Nechť $G = (V, E)$ je rovinný graf s $n = |V|$ vrcholy s minimálním stupněm 3. Nechť $V' \subset V$. Pak existuje nezávislá množina $I \subseteq V \setminus V'$ vrcholů stupně nejvýše 9 o mohutnosti alespoň*

$$(4n + 12 - 7|V'|)/70$$

Navíc I lze nalézt v čase $O(n)$.

Důkaz: Nechť V'' je množina vrcholů v G se stupni nejvýše 9, označme $x = |V''|$. G je rovinný graf, proto má nejvýše $3n - 6$ hran. Dále máme $n - x$ vrcholů stupně alespoň 10 a každý jiný vrchol má stupeň alespoň 3, tedy $[3x + 10(n - x)]/2 \leq 3n - 6$. Odtud $x \geq (4n + 12)/7$. Uvažme nyní podgraf indukovaný uzly $V'' - V'$. Ten má alespoň $x - |V'|$ vrcholů a každý má stupeň nejvýše 9. Tzn. můžeme obarvit vrcholy deseti barvami tak, aby každé dva sousední byly obarveny různě. To lze provést v čase $O(n)$. (Barvit barvami $1, \ldots, 10$ a vrcholy bereme v libovolném pořadí, použijeme vždy nejnižší barvu dosud nepoužitou u sousedů.) Pak tam musí podle Dirichletova principu existovat množina vrcholů se stejnou barvou o mohutnosti alespoň $(x - |V'|)/10$, a ty spolu po dvou nesousedí. Zároveň $(x - |V'|)/10 \geq (4n + 12 - 7|V'|)/70$. □

3.18 Věta. *Nechť G je jednoduché rovinné rozdělení s n vrcholy. Pak vyhledání v hranách G lze řešit v čase $O(\log n)$, prostoru $O(n)$ a v přípravném čase $O(n \log n)$.*

Důkaz: Zvolíme si vhodný konstantní počet vrcholů, pro které úlohu řešíme přímou diskusí, např. $n = 100$. Nechť $n > 100$. Nechť I je nezávislá množina vrcholů se stupněm nepřevyšujícím 9, které neleží na hraničním trojúhelníku. Použijeme lemma 3.17 s $|V'| = 3$ (hraniční trojúhelníky). To znamená, že v čase $O(n)$ najdeme takovou množinu, pro niž $|I| \geq (4n-9)/70$.

Vynecháním I získáme G_1 s nejvýše $n - |I| \leq 66n/70 + 1$. Přitom buňky v G_1 jsou mnohoúhelníky s $3 \leq m \leq 9$ hranami, tj. každou z nich umíme triangulovat v konstantním čase $O(1)$.

Rekurzivní aplikací tohoto postupu získáme požadovanou vyhledávací strukturu. □

4 Průniky

V této části uvedeme řešení několika úloh, které jsou založeny na vyhledání průniku (nebo průniků) jistých geometrických objektů.

V prvním případě půjde o nalezení průniků všech dvojic úseček. Budeme tedy mít za úkol zjistit, které dvojice z nich vybrané se protínají a ve kterém bodě.

Ostatní úlohy budou mít poněkud jiný charakter. Půjde o nalezení společného průniku všech objektů na vstupu.

4.1 Protínání úseček

Mějme n úseček $L_1, \ldots, L_n$ v rovině. Úkolem je najít všechny jejich vzájemné průsečíky. Je-li počet průsečíků $s = O(n^2)$, nelze čekat lepší čas běhu algoritmu.

Najdeme algoritmus pracující v čase $O((n+s)\log n)$ metodou *pročesávání*, kde s je délka výstupu, tedy počet průsečíků. Budeme postupovat zleva doprava, tedy pročesávací přímka bude vertikální.

- *Statut pročesávací přímky* (vertikální struktura) bude tvořen aktivními úsečkami, tedy těmi, které pročesávací přímka právě protíná, setříděnými podle výšky průniku s pročesávací přímkou L do vyhledávacího stromu.

- *Struktura událostí* (horizontální struktura) bude setříděná posloupnost koncových bodů úseček a průniků aktivních úseček, ležících napravo od L (ve vyhledávacím stromu).

Požadujeme, aby každý průchod novou událostí byl zvládnut v čase $O(\log n)$. Potřebujeme tedy vhodnou datovou strukturu, poskytující následující funkce:

Find(p) — pro daný bod najde interval jej obsahující

Input(q) — vloží další úsečku do L nebo událost

Delete(q) — zruší úsečku nebo událost

Pred,Succ — předchůdce a následník

Interchange(p,q) — výměna dvou úseček protínajících L

Struktura pročesávací přímky je jako seznam aktivních úseček inicializována na prázdnou. Na začátku algoritmu bude však potřeba setřídit koncové body úseček podle jejich x-ové souřadnice a inicializovat podle nich h–strukturu.

Algoritmus 4.1 Nalezení všech průsečíků množiny úseček

Vstup: množina úseček $L_1, \ldots, L_n$ 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 úsečka*

   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` /* *stará aktivní úsečka je odstraněna*

   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

Invariant algoritmu: je-li p průnik aktivních úseček (tedy zařazených v současné době ve v–struktuře) L_i, L_j a ve vertikálním pásu není žádný koncový bod nebo další průnik, pak p se objeví v horizontální struktuře pro další průchod. Z toho také plyne, že všechny průniky nalevo od pročesávací přímky už byly nalezeny.

Čas: $O(n + s)$ průchodů událostmi, každý v čase $O(\log n)$, proto výsledný čas je $O((n + s) \log n)$.

Vertikální struktura je realizovatelná v prostoru $O(n)$. Při naivním přístupu k horizontální struktuře (bez odstraňování v bodech 3.3.5 a 3.5.5) dostaneme prostor $O(n+s)$, v opačném případě dosáhneme $O(n)$.

4.1 Poznámka. Modifikací algoritmu 4.1 lze získat algoritmus, který v čase $O(n \log n)$ zjistí, zda existuje průsečík alespoň dvou úseček. Úprava spočívá ve vypuštění všech kroků ošetřujících nalezené průniky. Pokud bychom tento modifikovaný algoritmus nezastavili po nalezení prvního průsečíku, určitě nalezne průnik s nejmenší x-ovou souřadnicí, ne však nutně jako první nalezený průnik.

Mnoho problémů vede na úlohu nalezení průsečíků sady úseček. Uvedeme některé z nich spolu s časy, kterých lze pomocí našeho algoritmu dosáhnout.

4.2 Důsledek.

1. *Protnutí dvou mnohoúhelníků lze zjistit v čase $O(n \log n)$ a prostoru $O(n)$.*
2. *Jednoduchost mnohoúhelníka lze zjistit v čase $O(n \log n)$ a prostoru $O(n)$.*
3. *Protnutí hran grafu vnořeného do roviny lze zjistit v čase $O(n \log n)$ a prostoru $O(n)$.*

4.2 Průnik polorovin

Nechť $H_1, \dots, H_n$ jsou poloroviny v rovině. Úkolem je nalézt jejich průnik $\bigcap_{i=1}^n H_i$.

Použijeme metodu *rozděl a panuj*. Pro tuto metodu je charakteristické využití rekurze.

Algoritmus 4.2 NALEZENÍ PRŮNIKU POLOROVIN
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 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 aplikována věta 1.11 o nalezení průniku dvou konvexních mnohoúhelníků v lineárním čase, protože v jejím důkazu není ohraničenost mnohoúhelníků použita.

Čas: Je-li $T(n)$ čas, který algoritmus spotřebuje, platí $T(n) = 2T(n/2) + O(n)$ a $T(1) = 1$. Odtud plyne, že $T(n) = O(n \log n)$.

Dosáhli jsme tedy času $O(n \log n)$. Jistě nás bude zajímat, zda lze dosáhnout výsledku v čase lepším. Na příkladě si ukážeme, že ne.

4.3 Příklad. Poloroviny H_i nechť mají hranice tečné k parabole $y = x^2$ v bodech (x_i, x_i^2). Průnik má hranici dánu hraničními přímkami daných polorovin uspořádanými podle směrnice. Nalezení jejich průniku tedy znamená právě uspořádání směrnic takových hraničních přímek. Proto řešení tohoto problému bude časově alespoň tak složité jako třídění, tedy $O(n \log n)$.

Na základě tohoto výsledku již můžeme formulovat větu.

4.4 Věta. *Průnik n polorovin lze získat v optimálním čase $O(n \log n)$.*

Důkaz: plyne z předchozích úvah. □

4.3 Průnik konvexních mnohoúhelníků

Budeme nyní zkoumat nalezení průniku n konvexních mnohoúhelníků.

Každý konvexní k-úhelník je průnikem polorovin ohraničených hranami k-úhelníka (tzv. *opěrné nadroviny*). Lze použít předchozí výsledek pro průnik polorovin s využitím znalosti asociativity průniku. Proto průnik n konvexních mnohoúhelníků, kde i-tý mnohoúhelník má k_i vrcholů (hran), lze spočítat v čase

$$O(\sum_{i=1}^{n} k_i \log(\sum_{i=1}^{n} k_i)).$$

Jestliže pro jednoduchost ohraničíme počet vrcholů pro těchto n mnohoúhelníků číslem k, pak dostaneme časovou složitost $O(kn \log(kn))$.

Ve výše popsaném algoritmu jsme postupovali tak, že jsme rozdělili všechny mnohoúhelníky podle jejich opěrných přímek na poloroviny a pak jsme spočítali průnik všech těchto polorovin. Můžeme však použít jiného postupu: Nebudeme mnohoúhelníky rozbíjet, ale raději s nimi co nejdéle pracovat jako s objekty vcelku.

Naše procedura bude množinu n k-úhelníků dělit metodou *rozděl a panuj* na stejně dvě stejně velké podmnožiny $n/2$ k-úhelníků, a pro ně odděleně zavolá rekurzivně sama sebe. Po zjištění výsledků spočítá jejich průnik metodou uvedenou v důkazu věty 1.11. Pokud je na vstupu jediný mnohúhelník, je průnikem sama se sebou a tedy výstupem algoritmu. Tento případ slouží jako zarážka rekurze.

Čas $T(k, n)$ tohoto algoritmu odvodíme, pokud si uvědomíme, že platí

$$T(k, n) = 2T(k, n/2) + kn,\ T(k, 1) = O(k)$$

Celkově dostáváme čas $O(kn \log n)$.

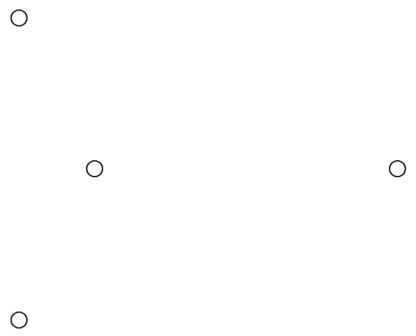
4.4 Jádro jednoduchého mnohoúhelníka

Budeme se zabývat problémem nalezení jádra jednoduchého mnohoúhelníka.

V úvodní kapitole jsme uvedli definici jádra jednoduchého mnohoúhelníka (tedy takového, jehož hrany se neprotínají). Toto jádro je vlastně průnikem všech polorovin, které ohraničují vnitřek jednoduchého mnohoúhelníka.

4.5 Poznámka.

- Jádro je vždy konvexní mnohoúhelník (viz obr. 34).
- Jednoduchý mnohoúhelník je konvexní právě, když je roven svému jádru.
- Pro mnohoúhelník, který není jednoduchý (jeho některé hrany se protínají jinde než ve vrcholech), jádro vůbec nedefinujeme.



Obrázek 34: Jádro nekonvexního jednoduchého mnohoúhelníka

Při testování příslušnosti daného bodu $x \in \mathbf{R}^2$ do vnitřku daného konvexního mnohoúhelníka jsme využívali binárního vyhledávání. Tento princip lze použít vždy, když jádro mnohoúhelníka P je neprázdné.

Přímá aplikace průniku příslušných polorovin dává algoritmus nalezení jádra o složitosti $O(n \log n)$, kde n je počet hran mnohoúhelníka. Lze to však zlepšit na $O(n)$ následujícím způsobem.

Mnohoúhelník budeme reprezentovat jako dvojitý cyklický seznam vrcholů v_i a hran

$$e_j\colon\ v_0 e_1 v_1 e_2 \dots e_{14} v_{14} e_0$$

Za v_0 zvolíme vrchol reflexní (s úhlem větším než 180^o), pokud takový neexistuje, je uvažovaný mnohoúhelník konvexní, tj. roven svému jádru. Budeme postupovat po vrcholech $v_1, \dots$ mnohoúhelníku a v každém kroku získáme mnohoúhelník K_i, který nemusí být ohraničený. Vznikající mnohoúhelníky K_i budou vždy konvexní aproximací výsledného jádra. Poslední z nich, označíme jej K, bude jádrem mnohoúhelníka.

Budeme potřebovat dva body F_i, L_i definované jako nejvzdálenější body dotyku tečen spuštěných z vrcholu v_i na mnohoúhelník K_i. Body F_i, L_i ležící na tečnách f_i, l_i spuštěných z bodu v_i na mnohoúhelník K_i jsou ilustrovány na obrázku 35.

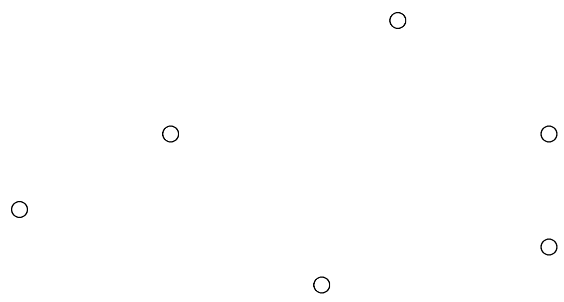
Algoritmus 4.3 Nalezení jádra jednoduchého mnohoúhelníka

Vstup: Mnohoúhelník $e_j\colon\ v_0 e_1 v_1 e_2 \dots e_n v_n e_0$
Výstup: Jeho jádro K (může být i prázdné)

Inicializace: Najdeme v_0, pokud neexistuje, je původní mnohoúhelník svým jádrem. Dále potřebujeme inicializovat K_1, F_1, L_1:

$$\begin{aligned} K_1 &:= \infty e_1 v_0 e_0 \infty \\ F_1 &:= \text{bod v nekonečnu na polopřímce } \infty e_1 v_0 \\ L_1 &:= \text{bod v nekonečnu na polopřímce } v_0 e_0 \infty \end{aligned}$$

Přechod od K_i ke K_{i+1}: F_i, L_i jsou body na hranicích opěrných polorovin pro K_i procházejících vrcholem v_i, a to ty nejvzdálenější v K_i.



Obrázek 35: Tečny spuštěné z bodu na mnohoúhelník a jejich nejvzdálenější body dotyku

1. v_i byl reflexní vrchol, polopřímka $\overline{v_{i+1}v_i}$ určuje další možné zmenšení K_i
   1.1. F_i je nalevo od $\infty e_{i+1}v_{i+1}$ (tedy tečna f), potom zatím nic neměníme
   1.2. F_i je napravo a L_i nalevo, pak musíme najít nové průniky a dostaneme novou část K_{i+1}. Navíc musíme aktualizovat F_i, L_i
2. Podobně pro v_i konvexní.

Algoritmus se buď ukončí již v některém z kroků z vrcholu v_i do vrcholu v_{i+1} zjištěním, že jádro je prázdné, nebo po projití všech vrcholů. V každé konstrukci následujícího K_{i+1} provádíme: vypouštění hran (každou nejvýše jednou), nalezení nových bodů F_{i+1}, L_{i+1} (posunují se vždy proti směru hod. ručiček). Pokud $K \neq \emptyset$, nemohou F_i a L_i oběhnout kolem celého mnohoúhelníka více než jednou, najdeme tedy tímto algoritmem jádro K v celkem lineárním čase $O(n)$. Bohužel, jestliže $K = \emptyset$, mohou tyto body obíhat až $O(n)$ krát kolem dokola, průběh algoritmu si tedy v tomto případě vyžádá čas $\Omega(n^2)$, viz obr. 36. Jednoduchou modifikací algoritmu spočívající v přidání testu kontrolujícího obíhání bodů F_i, L_i dosáhneme lineárního času i v případě $K = \emptyset$.

4.5 Průnik poloprostorů

V podkapitole 4.2 jsme se zabývali nalezením průniku polorovin. Nyní bude vstupem pro stejnou úlohu množina 3-dimenzionálních poloprostorů.

Pomocí konstrukce jisté relace mezi body a nadrovinami v prostoru libovolné dimenze převedeme problém nalezení průniku poloprostorů na duální problém nalezení konvexního obalu množiny bodů.

4.6 Definice. Nechť h je (nevertikální) nadrovina v $\mathbf{R}^{d+1}$ daná rovnicí $x_0 = p_1x_1 + \ldots + p_dx_d + p_{d+1}$. Pak definujeme *duální bod k nadrovině* h: $\delta(h) = [p_1, \ldots, p_d, p_{d+1}] = p$ a *duální nadrovinu k bodu* $p = [p_1, \ldots, p_d, p_{d+1}]$: $\delta(p) \equiv x_0 = -p_1x_1 - \ldots - p_dx_d + p_{d+1}$

4.7 Definice. *Vertikální vzdálenost* bodu $p = [p_1, \ldots, p_d, p_{d+1}]$ od nadroviny $h \equiv x_0 = q_1x_1 + \ldots + q_dx_d + q_{d+1}$ definujeme

$$vd(h,p) := p_{d+1} - (p_1q_1 + \ldots p_dq_d + q_{d+1})$$

52.0045.00153.0033.002 53.0033.00376.0055.004 76.0055.00528.0055.006 28.0055.00750.0023.008

Obrázek 36: Škaredý mnohoúhelník s prázdným jádrem

4.8 Lemma.

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

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

Důkaz: Přímým dosazením. □

4.9 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

$$S = \bigcap_{i=1}^{m} h_i^+ \cap \bigcap_{i=m+1}^{n} h_i^- = S^+ \cap S^- \tag{8}$$

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

Důkaz: 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. □

Najdeme-li tedy vrcholy konvexních obalů množin

$$\delta(h_1^+), \ldots, \delta(h_m^+)$$

a

$$\delta(h_{m+1}^-), \ldots, \delta(h_n^-),$$

známe množiny S^+ a S^- ze vztahu 8. Jsou to konvexní mnohostěny. Spočítáním jejich průniku dostaneme průnik všech poloprostorů. Tuto fázi algoritmu rozebírá podrobněji [Preparata-85].

5 Vyhledávání dle rozsahu

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

Budeme mít zadánu množinu S o n záznamech (mohou to být buď body nebo celé objekty = množiny bodů). Dotaz je zadání rozsahu, tedy ve směru všech os maxima a minima, který nás zajímá. Odpovědí na dotaz je podmnožina $S' \subseteq S$, která inciduje se zadaným rozsahem.

Na čas přípravy se nebudeme zaměřovat, bude nás zajímat hlavně paměť a čas potřebné k realizaci a zpracování dotazu. Podstatnou roli hraje volba vhodné datové struktury.

5.1 Příklad. Uvažujme množinu n bodů na přímce a dotazovaný rozsah je interval $[x', x'']$. Vyhledání provedeme takto:

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 $\Theta(n)$.

V dalším se soustředíme na vyhledávání bodů ve vícerozměrných prostorech, které už není tak elementární jako na přímce. Navíc nenajdeme algoritmus pracující v optimálním logaritmickém vyhledávacím čase a přitom s lineárními nároky na paměť. Uvedeme tři algoritmy, které budou představovat jisté kompromisy mezi požadavky na paměť a čas.

5.1 Multidimenzionální binární strom

V této podkapitole nadefinujeme strukturu zvanou multidimenzionální nebo také k-D strom. Uvedeme algoritmus na vyhledání dle rozsahu v tomto stromě. V k-D stromech se dají provádět také operace vkládání a odstraňování bodů, jde tedy o dynamickou strukturu. Tyto algoritmy jsou uvedeny v [Bentley-75].

5.2 Definice. *Zobecněný obdélník* v $\mathbf{R}^2$ je kartézský součin intervalů $[x_1, x_2] \times [y_1, y_2]$, kde $x_1, x_2, y_1, y_2 \in \mathbf{R} \cup \{\infty, -\infty\}$. Připouštíme tedy také neohraničené případy obdélníků.

5.3 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.

5.4 Poznámka. 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).

Na obrázku 37 je znázorněn 2-D strom pro množinu devíti bodů v rovině.

A nyní již slíbený algoritmus vyhledávání. Jde o vyhledávání ve stromu, proto uvedeme algoritmus, kde vrchol stromu, ve kterém vyhledávání začíná, bude dán parametricky. Vyhledání v celé struktuře se provede voláním této procedury se skutečným parametrem, který je kořenem stromu.

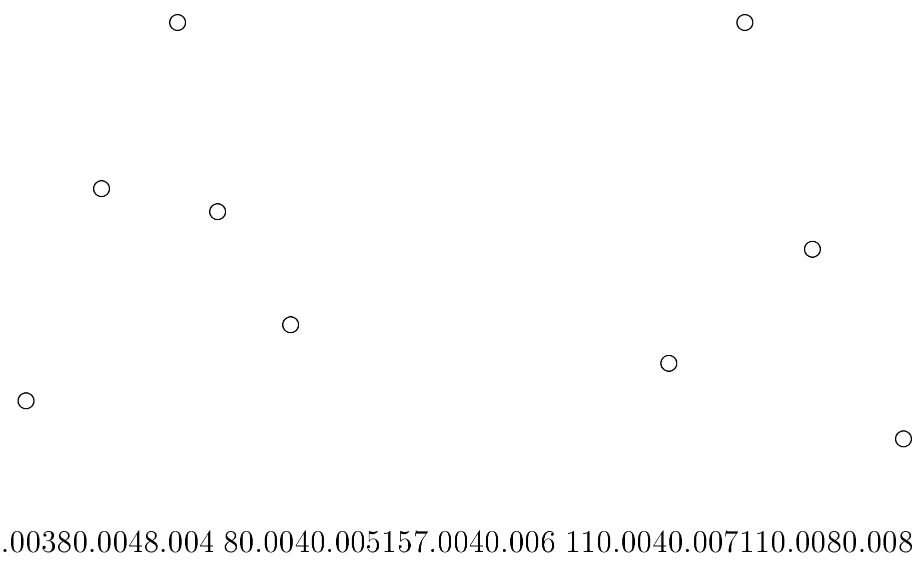
Algoritmus 5.1 RANGE SEARCHING METODOU 2-D STROMU
Vstup: vrchol v, ve kterém začíná vyhledávání, interval $D = [x_1, x_2] \times [y_1, y_2]$
Výstup: na výstupu se postupně objevují výsledné body dotazu
Vyvolání: je-li T 2-D strom, pak SEARCH($root(T)$)
`Procedure` SEARCH

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($RSON(v), D$)

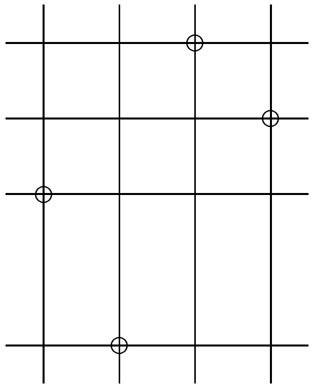
Prostor: 2-D strom potřebuje paměť $\Theta(n)$
Příprava: 2-D strom se sestrojí v čase $O(n \log n)$
Čas vyhledání: Není logaritmický, jak by se mohlo zdát. Algoritmus totiž navštěvuje řadu uzlů stromu „zbytečně“, tj. aniž by to vedlo k příspěvku do výstupu. Podrobnější (nepříjemnou) diskusí se ověří,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

Odvození časových i paměťových složitostí pro rozsahový dotaz v d-D stromu (tedy libovolné dimenze) je uvedeno v [Preparata-85] s výsledky:

5.5 Tvrzení. *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 $\Theta(dn \log n)$.*



Obrázek 37: 2-D strom pro devět bodů v rovině



Obrázek 38: Rozdělení roviny na $(n+1)^2$ zobecněných obdélníků horizontálními a vertikálními přímkami procházejícími zadanými body

5.2 Metoda přímého přístupu

Budeme nyní pracovat pouze 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 (viz obrázek 38). 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 $O(n)$.

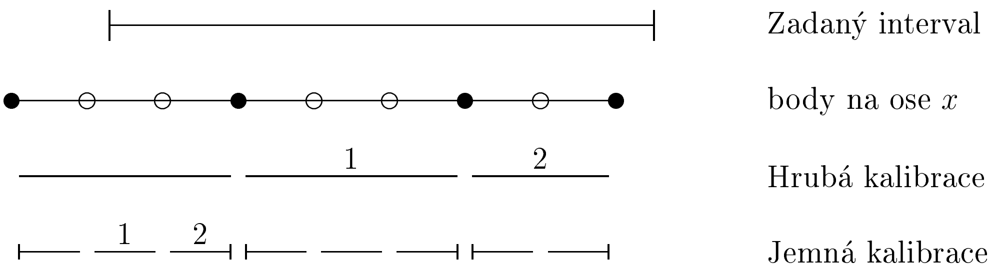
Velikost paměti potřebné pro provedení dotazu se však dá zmenšit. Pro rozsah $D = [x_1, x_2] \times [y_1, y_2]$ provedeme přímý přístup pouze ve směru osy x. Tak dostaneme ukazatel na vyhledávací strom všech bodů množiny S, které leží v daném x-ovém pásu, odpovídajícím intervalu $[x', x'']$ na ose x.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

Provedeme nejdříve normování reálných souřadnic na celočíselné souřadnice $1, \ldots, n$, tzn. seřadíme tyto souřadnice podle velikosti a očíslujeme podle pořadí. Dostaneme pak v čase $O(\log n)$ pro daný interval dvě čísla $i, j \in \{1, \ldots, n+1\}$. 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 tohoto přístupu se dá dosáhnout tzv. *kalibrací*. Budeme vyhledávat na ose x postupně ve dvou úrovních. Osa x bude rozdělena tzv. *hrubou kalibrací* na intervaly, jejichž



Obrázek 39: Kalibrace osmi bodů na ose a zasažené intervaly

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.

Na obrázku 39 je znázorněna kalibrace osmi bodů, rozdělení na hrubé a jemné intervaly, a dále hrubé i jemné intervaly, zasažené daným rozsahem (označeny 1 a 2). Obrázek ilustruje speciální případ, kdy dvojice určených intervalů v obou kalibracích spolu sousedí. Obecně dostaneme v hrubé kalibraci nějakou dvojici intervalů a v jemné jednu nebo dvě dvojice, vždy uvnitř jednoho hrubého intervalu.

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

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

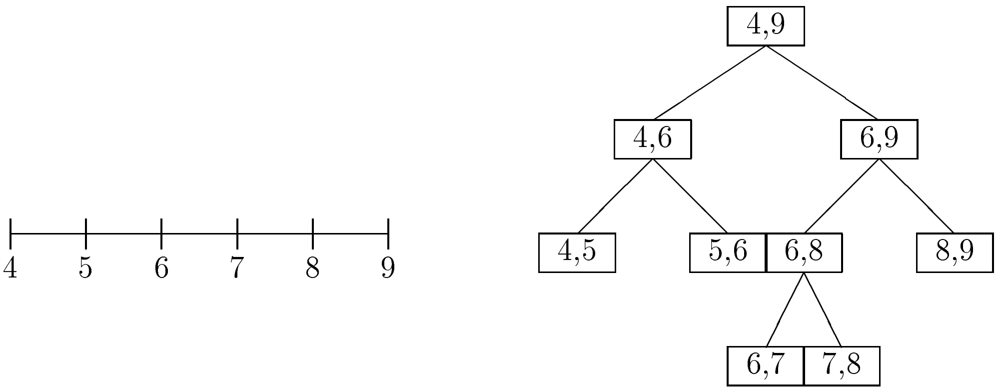
Minima dosáhneme pro $\alpha = 1/2$ (tedy hrubých intervalů je $\sqrt{n}$, stejně jako jemných uvnitř každého hrubého), a toto minimum je $O(n^2)$. Časově se nic nezmění, pouze logaritmicky procházíme tři stromy za sebou (musí být vyvážené), proto čas je $O(3 \log n) = O(\log n)$.

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.

5.6 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 $O(n^{1+\varepsilon})$.*

5.3 Rozsahový strom

Pro další metodu budeme potřebovat datovou strukturu vhodnou pro dynamické zpracovávání úseček s koncovými body v předem známé (statické) množině hodnot. Tato datová struktura



Obrázek 40: Strom úseček pro normalizovanou sadu bodů na přímce

nachází velice rozmanité uplatnění, my se s ní setkáme ještě v příští kapitole. Proto ji zavedeme obecněji, než budeme bezprostředně potřebovat.

5.7 Definice. Známe-li předem množinu koncových bodů uvažovaných úseček, zkonstruujeme *strom úseček*, který je ilustrován na obrázku 40. Máme-li dáno $2n$ možných koncových bodů, dostaneme právě $2n-1$ úseček, které už neobsahují žádné další koncové body. Tyto úsečky jsou právě listy konstruovaného stromu úseček. Konstrukce stromu je zřejmá z obrázku.

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 5.2 Operace 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$((b, e), v)$

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$(b, e, RSON(v))$

Tato procedura 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".

5.8 Příklad. Zpracování rozsahového dotazu na přímce pak lze provést tak, že 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.

Nyní můžeme navrhnout další metodu zpracování rozsahového dotazu, a to nad množinou bodů v prostoru libovolné dimenze. Pracujeme obecně v d-dimenzionálním prostoru a zpracováváme postupně souřadnice $x_1, \dots, x_d$. Pro dimenzi $d = 1$ definujeme rozsahový strom jako standardní vyhledávací strom z příkladu 5.1.

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 $S_{d-1}(v) := \{(x_2(p), \dots, x_d(p)); p \in S_d(v)\}$ 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 $S_{d-1}(v)$.

Všimněme si, že 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 40 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. V dimenzi 1 jde o problém z příkladu 5.1.

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 $n(v) \leq n$, 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

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

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

5.10 Poznámka. S použitím podobné konstrukce jako v redukované vyhledávácí struktuře používané k vyhledávání v rovinných rozděleních (rozebírá podkapitola 3.2.2) lze snížit čas potřebný pro algoritmus. Této technice se říká *přemosťování*, a spočívá v přidání jistých ukazatelů mezi sekundárními stromy. Viz poznámka 3.16 a [Preparata-85]. Dosáhneme tak času $O(\log^{d-1} n)$ při prostoru $O(n \log^{d-1} n)$.

6 Úlohy o obdélnících

V aplikacích se často objevují útvary, jejichž stěny jsou kolmé na souřadné osy. Hovoříme o *iso-ortogonálních* objektech. V této kapitole se budeme zabývat úlohami v rovině, začneme ale na přímce, tedy s úsečkami.

6.1 Míra sjednocení úseček

Máme dáno n intervalů $[a_1, b_1], \ldots, [a_n, b_n]$ 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 6.1 MÍRA SJEDNOCENÍ INTERVALŮ
Vstup: úsečky $[a_1, b_1], \ldots, [a_n, b_n]$ na přímce
Výstup: míra jejich sjednocení

1. $(X(1), \ldots, X(2n))$:= 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 $O(n \log n)$.

6.2 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 z definice 5.7. 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 6.2 Míra sjednocení obdélníků

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 $O(n \log n)$.

Stojí za povšimnutí, že pro obdélníky jsme v každém bodě měli vlastně za úkol spočítat míru sjednocení n úseček, tedy úloha se nám 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.

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

6.2 Poznámka. Pro $d \geq 3$ to však jde zvládnout v čase $O(n^{d-1})$ pomocí datové struktury zvané *quad-tree* – viz [Finkel-74].

6.3 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 (6.2), 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 6.3 Obvod sjednocení obdélníků

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`

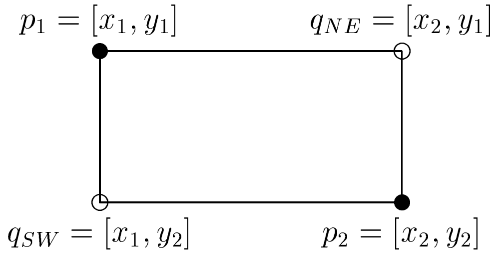
5.3.1. DELETE$(b_i, e_i, root(T))$

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 $O(n \log n)$.



Obrázek 41: SW a NE body k nesrovnatelné dvojici bodů

6.4 Uzávěry sjednocení obdélníků

6.3 Definice. O bodech $p_1 = (x_1, y_1), p_2 = (x_2, y_2)$ řekneme, že jsou *neporovnatelné*, jestliže $(x_1 - x_2)(y_1 - y_2) < 0$.

V takovém případě nazveme body $q_{SW} = (x_1, y_2)$ a $q_{NE} = (x_2, y_1)$, kde $x_1 < x_2$ a $y_2 < y_1$, SW (South-West) a NE (North-East) body k bodům p_1, p_2 (viz obrázek 41).

6.4 Definice. Oblast $\mathcal{R} \subset \mathbf{R}^2$ je NE- (resp. SW-) uzavřená, jestliže pro každé dva neporovnatelné body $p_1, p_2 \in \mathcal{R}$ leží i jejich q_{NE} (resp. q_{SW}) v $\mathcal{R}$.

6.5 Věta. *NE-, SW- i NE-SW-uzávěr sjednocení ortogonálních obdélníků lze spočítat v čase* $O(n \log n)$.

Důkaz: Dá se najít algoritmus používající opět metodu *pročesávání* pracující v tomto čase. Popisuje jej např. [Preparata-85]. □

SW-uzávěr sjednocení obdélníků má praktické uplatnění v databázích.

7 Plánování pohybu robota

7.1 Motivace

Jedním z hlavních cílů robotiky je konstrukce samostatných robotů, tj. robotů, kterým stačí říct co mají dělat, ale už ne jak to mají dělat. Mimo jiné to znamená, že robot musí být schopen plánovat si vlastní cestu. K tomu ale potřebuje znát prostředí ve kterém se pohybuje. To spočívá např. ve znalosti rozmístění překážek v prostoru. Pomocí těchto znalostí se může robot pohybovat aniž by se s některou překážkou srazil.

V této kapitole se pokusíme nastínit základní techniky používané při plánování pohybu. Obecný problém je však příliš složitý a proto použijeme několik zjednodušení. Budeme se pohybovat pouze ve 2-rozměrném prostoru, všechny překážky budeme uvažovat statické a budeme také předpokládat, že robot se může pohybovat všemi směry.

7.2 Pracovní prostor a konfigurační prostor

7.1 Definice. Předpokládejme, že robot $\mathcal{R}$ je jednoduchý mnohoúhelník se souřadnicemi $[x_1, y_1], \ldots, [x_n, y_n]$. *Konfigurací* $\mathcal{R}$ rozumíme vektor (x, y), značíme $\mathcal{R}(x, y)$. V konfiguraci $\mathcal{R}(x, y)$ má $\mathcal{R}$ souřadnice $[x_1 + x, y_1 + y], \ldots, [x_n + x, y_n + y]$. $\mathcal{R}(0, 0)$ tedy můžeme ztotožnit s vlastním robotem. *Pracovním prostorem* robota $\mathcal{R}$ rozumíme množinu $S = \{P_1, \ldots, P_t\}$ překážek.

Může-li se robot i otáčet, je jeho konfigurace $\mathcal{R}(x, y, \phi)$.

7.2 Definice. Prostor parametrů robota $\mathcal{R}$ nazýváme *konfiguračním prostorem* a značíme $\mathcal{C}(\mathcal{R})$.

7.3 Poznámka. Pro planárního robota který se může jen pohybovat je $\mathcal{C}(\mathcal{R})$ rovno $\mathbf{R}^2$ a pro robota, který se může i otáčet je to $\mathbf{R}^2 \times \langle 0, 360)$.

Nyní tedy máme vzájemě jednoznačnou korespondenci mezi body konfiguračního prostoru a umístěním robota v pracovním prostoru. Ne všechny body konfiguračního prostoru jsou však přípustné. Přípustné jsou jen ty body v nichž se robot v odpovídajícím bodě pracovního prostoru nedotýká žádné překážky.

7.4 Definice. *Zakázaným prostorem*, značíme $\mathcal{C}_z(\mathcal{R}, S)$, rozumíme ty body konfiguračního prostoru $\mathcal{R}$, které odpovídají bodům v nichž se $\mathcal{R}$ protíná s nějakou překážkou ze S. Ostatní body $\mathcal{C}(\mathcal{R})$ nazýváme *volným prostorem* a značíme $\mathcal{C}_p(\mathcal{R}, S)$.

Každá cesta robota pracovním prostorem odpovídá křivce v konfiguračním prostoru. Dokonce každá bezkolizní cesta odpovídá křivce v volným prostoru. Nyní umíme přenášet jak robota, tak i cestu z pracovního do konfigueračního prostoru. Teď to ještě naučíme překážky.

7.5 Definice. Množinu bodů $p \in \mathcal{C}(\mathcal{R})$ takových, že $\mathcal{R}(p)$ se protíná s překážkou P nazýváme *konfiguračním prostorem překážky* P, zkráceně $\mathcal{C}$*-překážkou* P.

7.3 Bodový robot

Dříve než se budeme zabývat obecným mnohoúhelníkovým robotem, podívejme se jak vypadá situace je-li robot jediný bod. Možná se to zdá být příliš velkým zjednodušením, ale ukážeme, že to není tak docela pravda. Mějme tedy robota $\mathcal{R}$ a množinu mnohoúhelníkových disjunktních překážek $P_1, \ldots, P_t$ jejichž celkový počet cholů e n. Pro našeho bodového robota konfigurační i pracovní prostory splývají. Nebudeme se zabývat hledáním jediné cesty robota, ale pokusíme se reprezentovat volný prostor vhodnou datovou strukturou pomocí které budeme moci co možná nejrychleji nalézt libovolnou cestu robota.

Pro zjednodušení situace si přidáme ještě jednu obdélníkovou překážku B takovou, že všechny překážky leží uvnitř tohoto obdélníku a překážka samotná je vnějšek tohoto obdélníku. Máme tedy

$$\mathcal{C}_p = B \setminus \bigcup_{i=1}^{t} P_i.$$

K reprezentaci $\mathcal{C}_p$ použijeme lichoběžníkovou mapu. Lichoběžníková mapa množiny ohraničených úseček dostaneme tak, že z každého koncového bodu úsečky spustíme dvě úsečky. Jednu nahoru a jednu dolů až narazí buď na další usečku nebo na hranici. Šestá kapitola knihy [Berg-97] uvádí pravděpodobnostní algoritmus, který dokáže najít lichoběžníkovou mapu v očekávaném čase $O(n \log n)$ [6]. Nyní uvedeme algoritmus počítající $\mathcal{C}_p$, který tento algoritmus používá.

Algoritmus 7.1 SPOČTI VOLNÝ PROSTOR
Vstup: Množina disjunktních mnohoúhelníků S
Výstup: Lichoběžníková mapa $\mathcal{C}_p(\mathcal{R}, S)$ pro robota $\mathcal{R}$

1. Buď E množina hran mnohoúhelníků z S
2. Vypočti lichoběžníkovou mapu $\mathcal{T}(E)$ algoritmem LICHBĚŽNÍKOVÁ MAPA
3. Vymaž z $\mathcal{T}(E)$ lichoběžníky, které leží uvnitř nějakého mnohoúhelníku z S

Zbývá jen promyslet, jak rozhodnout, leží-li lichoběžník uvnitř překážky nebo ne. To je ale jednoduché, protože víme ke které překážce patří horní i dolní hrana každéhu lichoběžníka. O tom zda máme lichoběžník vymazat nebo ne umíme vzhledem k uspořádání hran v $\mathcal{T}(E)$ rozhodnout v konstantním čase. Dostáváme tedy následující lemma.

7.6 Lemma. *Lichoběžníkovou mapu volného prostoru (značíme $\mathcal{T}(\mathcal{C}_p)$) bodového robota pohybujícího se v prostoru disjunktních mnohoúhelníkových překážek s celkem n hranami můžeme spočítat pravděpodobnostním algoritmem v očekávaném čase $O(n \log n)$.*

Jak použít $\mathcal{T}(\mathcal{C}_p)$ k nalezení cesty robota z pozice p_s do p_c? Jsou-li p_s i p_c ve stejném lichoběžníku, je to snadné, neboť cesta je právě úsečka $p_s p_c$. V opačném případě zkonstruhujeme atlas v $\mathcal{T}(\mathcal{C}_p)$. Tento atlas je graf vložený do $\mathcal{C}_p$. Kromě první a poslední části bude cesta vést po tomtu grafu. Zkonstruhujme nejprve vrcholy tohoto grafu. Vrcholem budou středy všech lichoběžníků a také středy všech vertikálních hran dvou sousedních lichoběžníků. Hrany bodou právě ty spojnice dvou vrcholů z nichž jeden leží v centru lichoběžníka a druhý na jeho hranici. Použití atlasu k nalezení cesty cesty z p_s do p_c demostruje následující algoritmus.

Algoritmus 7.2 SPOČÍTEJ CESTU
Vstup: Lichoběžníková mapa $\mathcal{T}(\mathcal{C}_p)$, atlas $\mathcal{G}$ a startovní a cílová pozice p_s a p_c
Výstup: Cesta z p_s do p_c pokud existuje, oznámení že neexistuje jinak

1. Najdi lichoběžníky $\triangle_s$ a $\triangle_c$ obsahující p_s a p_c
2. `if` $\triangle_s$ nebo $\triangle_c$ neexistují `then`
   2.1. `return` Cesta neexistuje. p_s nebo p_c leží v zakázaném prostoru
3. `else`
   3.1. Buďte v_s resp. v_c vrcholy $\mathcal{G}$ ve středu $\triangle_s$ resp. $\triangle_c$

[6] Zatím nebude uveden. Možná časem budu-li na to mít čas.

3.2. Najdi v $\mathcal{G}$ cestu z v_s do v_c pomocí prohledávání do šířky

3.3. `if` taková cesta neexistuje `then`

3.3.1. `return` Cesta z p_s do p_c neexistuje

3.4. `else`

3.4.1. `return` Vrať cestu složenou z úsečky z p_s do v_s, cesty z v_s do v_c v $\mathcal{G}$ a úsečky z v_c do p_c

Prodiskutujme nejdříve korektnost algoritmu. Každá vrácená cesta je bezkolizní, protože všechny úseky každé cesty leží uvnitř lichoběžníků, které celé leží v volném prostoru. Naopak existuje-li nějaká cesta z p_s do p_c, musí p_s i p_c ležet v nějakém lichoběžníku v $\mathcal{C}_p$. Cesta z p_s do p_c musí procházet posloupností lichběžníků $\triangle_1, \triangle_2, \ldots, \triangle_k$. Z definice je $\triangle_s = \triangle_1$ a $\triangle_c = \triangle_k$. Buď v_i střed $\triangle_i$ a nechť cesta vede z $\triangle_i$ do $\triangle_{i+1}$, pak $\triangle_i$ a $\triangle_{i+1}$ musí být sousední a mít společnou svislou hranu. Z konstrukce atlasu $\mathcal{G}$ plyne, že v $\mathcal{G}$ existuje cesta z v_i do v_{i+1} skládající se ze dvou hran, existuje tedy cesta z v_s do v_c a algoritmus prohledávání grafu do šířky nějakou takovou cestu najde.

Podívejme se nzní na časovou složitost tohoto algoritmu. Konstrukce lichoběžníkové mapy zabere čas $O(n \log n)$. Nalezení $\triangle_s$ a $\triangle_c$ zabere $O(n)$. Prohledávání do šířky je lineární vzhledem k velikosti grafu. Počet vrcholů $\mathcal{G}$ je lineární vzhledem k počtu vrcholů překážek. Počet hran je také lineární, protože se jedná o planární graf. Prohledávání tedy zabere čas $O(n)$. Čas k vypsání cesty je zhora omezen počtem hran v $\mathcal{G}$, který je $O(n)$. Můžeme tedy formulovat následující větu.

7.7 Věta. *Buď $\mathcal{R}$ bodový robot a S množina mnohoúhelníkových překážek s celkem n hranami. Pak si můžeme předpočítat S v očekávaném čase $O(n \log n)$ tak, že každou cestu robota $\mathcal{R}$ můžeme najít v čase $O(n)$ pokud existuje, případně ve stejném čase rozhodnout, že neexistuje.*

Předchozí věta nám zaručuje nalezení cesty pokud existuje, neříká však nic o tom nakolik je takto nalezená cesta optimální.

7.4 Minkowského sumy

7.8 Definice. *Minkovského soumou* množin $S_1, S_2 \subseteq \mathbf{R}^2$, značíme $S_1 \oplus S_2$, rozumíme množinu

$$S_1 \oplus S_2 := \{(p_x + q_x, p_y + q_y); (p_x, p_y) \in S_1, (q_x, q_y) \in S_2\}.$$

Dále pro $p \in S$ a $S \subseteq \mathbf{R}^2$ definujme $-p := (-p_x, -p_y)$ a $-S := \{-p; p \in S\}$.

Jednoduchým pozorováním vidíme, že maximální bod Minkowského sumy $P \oplus R$, kde P a R jsou roviné objekty, ve směru $\vec{d}$ je součtem maximálních bodů P a R ve směru $\vec{d}$.

7.9 Věta. *Minkowského suma $P \oplus R$ dvou konvexních mnohoúhelníků s n a m hranami je také konvexní mnohoúhelník s nejvíce $n + m$ hranami.*

Důkaz: Konvexnost plyne přímo z definice. Vezměme hranu e z $P \oplus R$. Tato hrana je maximální ve směru své normály. Musí být tedy generována body z P a R které jsou maximální v témže směru. Navíc buď v P nebo R (nebo i v obou) musí být hrana, která je v tomto směru také maximální. Poznamenejme si tuto hranu. Tímto způsobem je každá hrana označena nejvíce jednou. Celkový počet hran $P \oplus R$ je tedy maximálně $n + m$. □

7.10 Definice. Dvojici roviných objektů o_1 a o_2 nazýváme párem *pseudodisků*, jestliže množiny $o_1 \backslash o_2$ a $o_2 \backslash o_2$ jsou souvislé. Množinu objektů nazveme množinou pseudodisků, jestliže každá dvojice objektů je pár pseudodiků.

7.11 Věta. *Buďte P_1 a P_2 disjunktní konvexní mnohoúhelníky a R konvexní mnohoúhelník. Pak Minkowského sumy $P_1 \oplus R$ a $P_2 \oplus R$ tvoří pár pseudodisků.*

Důkaz: Důkaz se provede tak, že předpokládáme existenci alespoň dvou souvislých komponent $(P_1 \oplus R) \backslash (P_2 \oplus R)$ a jednoduchými hrátkami s maximálními směry dojdeme ke sporu. □

7.12 Věta. *Složitost sjednocení množiny mnohoúhelníkových pseudodisků s celkem n hranami je $O(n)$.*

Důkaz: Důkaz se provede tak, že každý vrchol sjednocení přiřadíme nějakému vrcholu nějakého pseudodisku tak, že každý vrchol je přiřazen nejvíce dvakrát. To vede k hranici $2n$ pro složitost sjednocení. □

Algoritmus 7.3 MINKOWSKÉHO SUMA

Vstup: Konvexní mnohoúhelníky P a R s vrcholy $v_1, \dots, v_n$ a $w_1, \dots, w_m$ setříděnými proti směru pohybu hodinových ručiček tak, že v_1 a w_1 mají nejmenší y-ové souřadnice (případně i x-ové)

Výstup: Minkowského suma $P \oplus R$

1. $i \leftarrow 1, j \leftarrow 1$
2. $v_{n+1} \leftarrow v_1, w_{m+1} \leftarrow w_1$
3. `repeat`
   - 3.1. Přidej $v_i + w_j$ do $P \oplus R$
   - 3.2. `if` $\angle(v_i v_{i+1}) < \angle(w_j w_{j+1})$ `then`
     - 3.2.1. $i \leftarrow i + 1$
   - 3.3. `else if` $\angle(v_i v_{i+1}) > \angle(w_j w_{j+1})$ `then`
     - 3.3.1. $j \leftarrow j + 1$
   - 3.4. `else`
     - 3.4.1. $i \leftarrow i + 1$
     - 3.4.2. $j \leftarrow j + 1$
4. `until` $i = n + 1$ `and` $j = m + 1$

Výrazem $\angle(uv)$ značíme úhel, který svírá vektor $\vec{uv}$ s kladným směrem osy x.

Algoritmus pracuje v lineárním čase, protože při každém průchodu cyklu se inkrementuje buď i nebo j a po dosažení hodnoty $n + 1$ resp. $m + 1$ tato již neroste. K důkazu korektnosti algortimu lze využít stejné argumenty jako v důkaze věty 7.9. Můžeme tedy formulovat větu.

7.13 Věta. *Minkowského suma dvou konvexních mnohoúhelníků s n resp. m vrcholy je možné spočítat v čase $O(n + m)$.*

7.5 Mnohúhelníkový robot

7.14 Definice. $\mathcal{C}$-překážkou P, značíme $\mathcal{CP}$, rozumíme množinu

$$\mathcal{CP} := \{(x,y) \in \mathcal{C}(\mathcal{R}); \mathcal{R}(x,y) \cap P \neq \emptyset\}.$$

7.15 Věta. *Buď $\mathcal{R}$ robot a P překážka. Pak $\mathcal{CP} = P \oplus (-\mathcal{R}(0,0))$.*

Důkaz: Naším cílem je ukázat, že množiny $\mathcal{R}(x,y)$ a P mají neprázdný průnik, právě tehdy, když $(x,y) \in P \oplus (-\mathcal{R}(x,y))$.

Předpokládejme nejprve, že $q = (q_x, q_y)$ leží v $\mathcal{R}(x,y) \cap P$. Z toho, že $q \in \mathcal{R}(x,y)$ máme $(q_x - x, q_y - y) \in \mathcal{R}(0,0)$ neboli $(-q_x + x, -q_y + y) \in -\mathcal{R}(0,0)$. Protože $q \in P$ dostáváme $(x,y) \in P \oplus (-\mathcal{R}(0,0))$.

Naopak, nechť $(x,y) \in P \oplus (-\mathcal{R}(0,0))$. Pak musí existovat $(r_x, r_y) \in \mathcal{R}(0,0)$ a $(p_x, p_y) \in P$ takové, že $(x,y) = (p_x - r_x, p_y - r_y)$, čili $(p_x, p_y) = (r_x + x, r_y + y)$ což implikuje $\mathcal{R}(x,y) \cap P \neq \emptyset$.
□

7.16 Věta. *Buď $\mathcal{R}$ konvexní mnohoúhelníkový robot a S množina mnohoúhelníkových překážek s celkem n hranami. Pak složitost volného prostoru $\mathcal{C}_p(\mathcal{R}, S)$ je $O(n)$.*

Důkaz: Nejdříve natriangulujeme každou překážku. Tím dostaneme množinu $O(n)$ trojúhelníků. Povolený prostor je doplněk sjednocení $\mathcal{C}$-překážek těchto trojúhelníků. Podle věty 7.11 tvoří tedy množinu pseudodisků a podle věty 7.12 má jejch sjednocení lineární složitost.
□

Zbývá najít algoritmus hledající volný prostor. Místo toho uvedeme algoritmus, který počítá zakázený prostor a ten volný dostaneme jako jeho doplněk. Připomeňme, že máme-li množinu $\{P_1, \ldots, P_n\}$ trujúhelníků získaných triangulací překážek, je

$$\mathcal{C}_z = \bigcup_{i=1}^{n} \mathcal{CP}_i = \bigcup_{i=1}^{n} P_i \oplus (-\mathcal{R}(0,0)).$$

K výpočtu jednotlivých Minkowského sum můžeme použít algoritmus 7.3.

Algoritmus 7.4 Zakázaný prostor
Vstup: Množina $\mathcal{C}$-překážek $\mathcal{CP}_1, \ldots, \mathcal{CP}_n$
Výstup: Zakázaný prostor $\mathcal{C}_z$

1. `if` n=1 `then`
   1.1. `return` $\mathcal{CP}_1$
2. `else`
   2.1. $\mathcal{C}_z^1 \leftarrow$ *Zakázaný prostor*($P_1, \ldots, P_{\lceil n/2 \rceil}$)
   2.2. $\mathcal{C}_z^2 \leftarrow$ *Zakázaný prostor*($P_{\lfloor n/2 \rfloor + 1}, \ldots, P_n$)
   2.3. `return` $\mathcal{C}_z^1 \cup \mathcal{C}_z^2$

7.17 Lemma. *$\mathcal{C}_p$ konvexního robota a množiny mnohoúhelníkových překážek s celkem n hranami lze spočítat v čase $O(n \log^2 n)$.*

Důkaz: Mnohoúhelník s m hranami umíme triangulovat v čase $O(m \log m)$. Je-li tedy m_i složitost překážky P_i, pak triangulace všech t překážek trvá právě $O(n \log n)$, protože

$$\sum_{i=1}^{t} m_i \log m_i \leq \sum_{i=1}^{t} m_i \log n = n \log n.$$

Spočítat všechny $\mathcal{C}$-překážky travá celkem $O(n)$. Sjednocení z řádku 2.3 lze podle druhé kapitoly knihy [Berg-97] provést v čase $O((n_1 + n_2 + k) \log(n_1 + n_2))$, kde n_1, n_2 a k je složitost $\mathcal{C}_z^1$, $\mathcal{C}_z^2$ a $\mathcal{C}_z^1 \cup \mathcal{C}_z^2$. Podle vety 7.16 je složitost volného a tedy i zakázaného prostoru lineární vzhledem k celkové složitosti všech překážek. n_1, n_2 i k jsou tedy všechny $O(n)$ a složitost řádku 2.3 je tedy $O(n \log n)$. Pro celkovou složitost algoritmu tedy dostáváme rekurentní formuli $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n \log n)$. Jejím řešením je $O(n \log^2 n)$.

□

Nyní máme spočítán volný prostor a můžeme pokračovat jako v kapitole 7.3 o bodovém robotovi. Spočítáme lichoběžníkovou mapu volného prostoru a atlas. Pro startovní a cílovou pozici v pracovním prostoru najdeme odpovídající body v konfiguračním prostoru, najdeme cestu pomocí lichoběžníkové mapy a atlasu a výsledek přeneseme zpět do pracovního prostoru. Následující věta nám shrnuje předchozí výsledky.

7.18 Věta. *Buď $\mathcal{R}$ konvexní mnohoúhelníkový robot a S množina disjunktních mnohoúhelníkových překážek s celkem n hranami. Pak si můžeme předpočítat S v očekávaném čase $O(n \log^2 n)$ tak, že každou cestu robota $\mathcal{R}$ můžeme najít v čase $O(n)$ pokud existuje, případně ve stejném čase rozhodnout, že neexistuje.*

7.6 Rotační robot

Mějme konvexního mnohoúhelníkového robota $\mathcal{R}$, který se může pohybovat a otáčet v prostoru tvořeném množinou $P_1, \ldots, P_t$ disjunktních mnohoúhelníkových překážek. Konfigurační prostor robota je tedy $\mathbf{R}^2 \times \langle 0, 360)$. V tomto případě je $\mathcal{CP}$ definováno jako

$$\mathcal{CP} = \{(x, y, \phi) \in \mathbf{R}^2 \times \langle 0, 360); \mathcal{R}(x, y, \phi) \cap P \neq \emptyset\}.$$

Volný prostor je opět komplement sjednocení $\mathcal{C}$-překážek. Už samotná $\mathcal{C}$-překážka je poměrně komplikovaná a tím více volný prostor. Zjednodušíme si tedy celý problém v tom smyslu, že nebudeme uvažovat libovolné ϕ, ale dovolíme robotovi pootočit se jen o nějaký pevně stanovený úhel $360/k$, čímž nám vznikne $k$jakýchsi „řezů“. Pro každé $\phi_i = i \cdot (360/k)$ $i = 0, \ldots, k = 1$ spočítáme řez volného prostoru. Nyní můžeme pro každé $\mathcal{R}(0, 0, \phi_i)$ spočítat $\mathcal{T}_i$ a $\mathcal{G}_i$ s využitím teorie z předešlých kapitol. Pohyb robota se teď skládá ze dvou typů pohybu. Buď to je klasický pohyb v jednom řezu, nebo pootočení což reprezentuje přeskok z řezu příslušného ϕ_i do ϕ_{i+1}. Teď potřebujme spojit $\mathcal{G}_i$ do $\mathcal{G}$. Uděláme to tak, že pro každé dvě sousední lichoběžníkové mapy $\mathcal{T}_i$ a $\mathcal{T}_{i+1}$ (index 0 ztotožňujeme s k) sestrojíme jejich překryv (kapitola 2 knihy [Berg-97]). Tím najdeme všechny dvojice $\triangle_1 \in \mathcal{T}_i$ a $\triangle_2 \in \mathcal{T}_{i+1}$ které se protínají. Nechť bod $(x, y, 0)$ je z tohoto průniku. Do $\mathcal{G}$ přidáme vrvholy (x, y, ϕ_i) a $(x, y, \phi_{i+1}$ a spojíme je hranou. Cesta vedoucí přes tuto hranu bude repreztnovat ono pootočení. Vrchol (x, y, ϕ_i) spojíme hranou se středem $\triangle_1$ a $(x, y, \phi_{i+1}$ se středem $\triangle_2$. Tyto hrany leží v příslušných řezech a odpovídají tedy obyčejnému pohybu. Teď když máme sestrojen graf $\mathcal{G}$, můžeme hledat cestu z $\mathcal{R}(x_s, y_s, \phi_s)$ do $\mathcal{R}(x_c, y_c, \phi_c)$. Nejdříve najdeme nejbližší řezy k ϕ_s a ϕ_c. V těchto řezech nejdeme odpovídající lichoběžníky $\triangle_s$ a $\triangle_c$ obsahující startovní a cílovou pozici. Pokud alespoň jeden z nich neexistuje, leží některý z bodů v zakázaném prostoru a cesta neexistuje. V opačném případě určíme vrholy v_s a v_c grafu $\mathcal{G}$ ležící v středech $\triangle_s$ a $\triangle_c$. Nyní můžeme v $\mathcal{G}$ najít cestu z v_s do v_c (pokud existuje). Celá cesta se tedy skládá z pěti částí. Nejdříve otočení do řezu ϕ_i, dále z posunutí do v_s, cesty z v_s do v_c v $\mathcal{G}$, posunutí do (x_c, y_c, ϕ_j) a otočení do (x_c, y_c, ϕ_c).

Tato metoda má jednu drobnou vadu: není vždy korektní. V některých případech může algortimus chybně ohlásit, že cesta neexistuje. Nastává to např. když startovní bod leží ve volném prostoru, ale tento bod v nejbližším řezu už leží v zakázaném prostoru. Další chybou může být, že nalezená cesta nemusí být bezkolizní. Posuny jsou v pořádku. Problémy může způsobit otáčení. Tyto problémy může způsobit „napůl“ pootočený robot. Oba dva problémy můžeme částečně řešit zvýšením počtu řezů, ale korektnost výsledku nám to nezaručí. K vyřešení druhého problému můžeme použít malý trik. Místo robota $\mathcal{R}$ budeme uvažovat robota vzniklého z $\mathcal{R}$ tak, že $\mathcal{R}$ pootočíme ve směru i proti směru pohybu hodinových ručiček o $360/k$ a budeme uvažovat konvexní obal takto vzniklé plochy $\mathcal{R}'$. Lichběžníkové mapy a graf budeme počítat s použitím $\mathcal{R}'$ místo $\mathcal{R}$. Platí že cesta nalezená pro robota $\mathcal{R}'$ je bezkolizní pro $\mathcal{R}$. Problém s nenalezením existující cesty tím však vyřešen není.

Literatura

[Berg-97] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry - Algorithms and Applications, Springer Verlag, 1997

[Mehlhorn-84] Kurt Mehlhorn, Data Structures and Algorithms, Springer Verlag, 1984

[Preparata-85] Franco Preparata, Michael Shamos, Computational Geometry, Springer Verlag, 1985

[Rényi-63] A. Rényi, R. Sulanke, Ű̈ber die konvexe Hulle von n zufällig gewahlten Punkten, I, *Z. Wahrschein.* 2, 75–84, 1963

[Raynaud-70] H. Raynaud, Sur l'enveloppe convexe des nuages des points aléatoires dans $\mathbf{R}^n$, I, *J. Appl. Prob.* 7, 35–48, 1970

[Chazelle-83] B. M. Chazelle, Optimal algorithms for computing depths and layers, *Proc. 21st Allerton Conference on Comm., Control and Comput.*,427–436, Oct. 1983

[MIT] Introduction to Algorithms, Chapter 24 Minimum Spanning Trees

[Megiddo-83] N. Megiddo, Linear time algorithm for linear programming in $\mathbf{R}^3$ and related problems, *SIAM J. Comput.* 12(4), 759–776, Nov. 1983

[Gilbert-79] P. N. Gilbert, New results on planar triangulations, Tech. Rep. ACT-15, Coord. Sci. Lab., University of Illinois at Urbana, July 1979

[Bentley-75] J. L. Bentley, Multidimensional binary search trees used for associative searching, *Communications of the ACM* 18, 509–517 (1975)

[Finkel-74] R. A. Finkel, J. L. Bentley, Quad-trees; a data structure for retrieval on composite keys, *Acta Informatica* 4, 1–9 (1974)