Úvod

Úloha lokalizace bodu spočívá v nalezení oblasti, v níž se nachází daný bod. Mějme například mapu Evropy a místo na ní zadané pomocí zeměpisné šířky a délky. V takové situaci je naším úkolem odpovědět na otázku, ve kterém státě ono místo leží. S řešením polohy bodu se tedy setkáme u geografických informačních systémů (GIS). Dalšími praktickými aplikacemi mohou být určování polohy místa kriminálního činu na mapě policejních okrsků nebo neustálé zjišťování polohy lodi při mořeplavbě vzhledem k mělčinám a jiným nástrahám, o nichž máme informace v mapě. Důležité je rovněž využití lokalizace bodu v počítačové grafice, v počítačem podporovaném projektování (CAD) a v robotice. Potřeba znát polohu bodu v různých odvětvích roste, ať už jako samostaný problém nebo jako dílčí problém nějaké komplexnější úlohy, seznámíme se tedy v této práci s efektivním algoritmem.

Podobné problémy se vyskytovaly i v minulosti, proto již k úloze lokalizace bodu existuje několik přístupů a různě efektivních algoritmů. K vyhledávacímu času $\mathcal{O}(\log n)$ a paměťové složitosti $\mathcal{O}(n)$ vedou čtyři zcela rozdílné postupy. Jedná se o řetězcovou metodu, metodu zjemnění pomocí triangulace, metodu využívající persistentní vyhledávací stromy a náhodnostní přírůstkovou metodu. Tato práce vychází z posledně jmenované, především z algoritmu popsaného v [1], tedy z Mulmuleyova algoritmu [3] a Seidelovy prezentace [4].

Hlavními zdroji při zpracovávání dané problematiky byla jednak kniha [1], jednak přednáška [2]. Některé informace pro čtvrtou kapitolu týkající se výpočetní složitosti byly čerpány z [5]. Všechny příklady a obrázky byly pro účely této práce vytvořeny.

První kapitola se věnuje základním strukturám, s nimiž budeme pracovat, a základnímu principu vyhledávání. Nejdříve popíšeme dvojitě souvislý seznam hran, což je systém tabulek používaný k reprezentaci rovinných podrozdělění. Dále se zabýváme vhodnou úpravou původně zadané mapy na lichoběžníkovou mapu, v níž se efektivněji vyhledává. Následně popíšeme vyhledávací strukturu a obecný postup při vyhledávání.

V další kapitole se čtenář dočte o samotnému algoritmu, který je rozdělen na několik dílčích částí, což dobře znázorňuje první pseudokód na straně 15. Kroky tohoto algoritmu pak blíže rozebereme a jejich kostry opět zapíšeme v pseudokódech. Tímto čtenář získá dobrou představu o tom, jak uvedený algoritmus naprogramovat.

V předchozím textu předpokládáme, že mapa, v níž hledáme, neobsahuje žádnou svislou úsečku. Jinými slovy pracujeme pouze s případy, kdy každé dva body mají různé x-ové souřadnice. Toto a další zjednodušení jsou učiněna kvůli snazšímu objasnění fungování algoritmu. Třetí kapitola speciální případy rozebírá a uvádí postup, jak se s nimi vypořádat.

V poslední kapitole určíme, jaká je odhadovaná složitost uvedeného algoritmu. Od-

hadneme časovou složitost vytvoření struktur pro vyhledávání, i velikost těchto struktur a časovou složitost vyhledávání zadaného bodu, která je logaritmická.

Hlavní přínos této práce spočívá ve zjednodušení dvojitě souvislého seznamu pro potřeby lokalizace bodu, v objasnění obecného postupu při vyhledávání v kapitole 1, v detailnějším rozpisu jednotlivých kroků v kapitole 2, a to především kroků pátého a šestého, které jsou v [1] uvedeny poměrně stručně a bez pseudokódu. Text je psán způsobem, který má čtenáři přiblížit danou problematiku natolik, aby byl schopen podle uvedených postupů a pseudokódů algoritmus řešící úlohu lokalizace bodu naprogramovat.

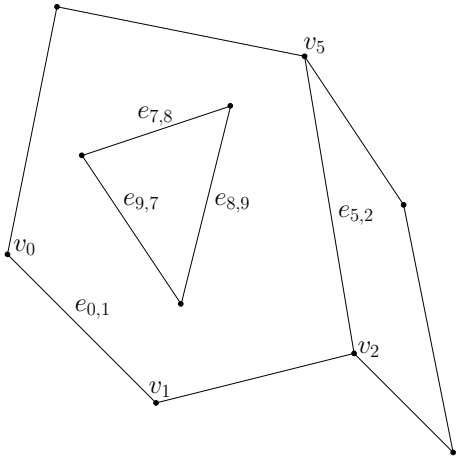
Kapitola 1

Základní konstrukce

V této kapitole se blíže seznámíme s *dvojitě souvislým seznamem hran*, *lichoběžníkovou mapou* a vyhledávací strukturou. První z nich reprezentuje rovinné podrozdělení, které je na vstupu celé lokalizace. Potřebujeme tedy pochopit vlastnosti dvojitě souvislého seznamu hran a naučit se s ním manipulovat. Druhá struktura vznikne úpravou původní mapy do podoby, jež lépe vyhovuje požadavkům algoritmu. Odůvodníme zde formu této struktury a na příkladech ukážeme, jak funguje. Poté si ukážeme, jak vypadá vyhledávací struktura. Jedná se acyklický orientovaný graf, v němž procházíme od kořene k listům, které mají vazbu na oblast příslušnou hledanému bodu. Nakonec si objasníme obecný postup uplatňovaný při vyhledávání zadaného bodu.

1.1 Dvojitě souvislý seznam hran

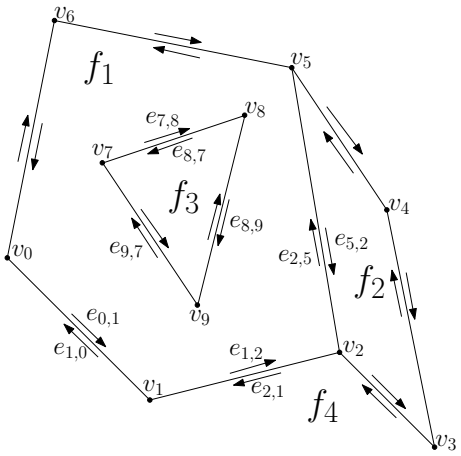
Lokalizaci bodu provádíme v rovinném podrozdělení, tj. v určitém rozčlenění roviny na menší oblasti, jejichž hranice tvoří libovolné mnohoúhelníky. Jedná se o orientovaný graf, který nemusí být nutně souvislý.



Obrázek 1: Neorientovaná forma podrozdělení

Na obrázku 1 jsou uzly označeny jako v_0, v_1, v_2, v_5, hranami jsou $e_{0,1} = (v_0, v_1)$ nebo $e_{5,2} = (v_5, v_2)$. Oblasti jsou dány hranami, jež tvoří jejich hranici, čili malý trojúhelník uvnitř pětiúhelníku je určen množinou hran $\{e_{7,8}, e_{8,9}, e_{9,7}\}$. Při práci s podrozdělením považujeme hrany i oblasti za otevřené. Znamená to, že uzly nejsou prvkem žádné hrany, stejně jako nepatří do žádné oblasti, a ani hrany nepatří do oblastí, které ohraničují. Nějaký vztah však mezi uzly určujícími hranu a samotnou hranou je, říkáme tedy, že jsou *sousední*, analogicky toto platí o dvojicích uzel-oblast a hrana-oblast.

Při uchovávání informací o mapě chceme šetřit místo a umět rychle vyhledat potřebná data, ale zároveň chceme, abychom měli dostatek informací pro určité úkony. Můžeme například potřebovat procházet hranici oblasti hranu po hraně nebo zjistit všechny hrany, které sousedí s daným uzlem. Jako užitečné se ukazuje rozdělit každou hranu na dvě opačně orientované hrany, označíme je slovem *půlhrany*. Odtud již lze opravdu reprezentovat podrozdělení jako orientovaný graf. Původní obrázek se nám mírně změní:



Obrázek 2: Orientované rovinné podrozdělení

U hrany platilo, že měla dvě sousední oblasti, proto půlhrana má vždy právě jednu. Půlhrany veďme tak, že jejich sousední oblasti se budou vždy nacházet po levé straně od půlhrany, jdeme-li od jejího počátečního uzlu ve směru šipky. Tuto úmluvu již respektuje i obrázek 2, viz výše. Dále vidíme, že každá půlhrana má jednoznačně určeného následníka i předchůdce, takže budeme-li sledovat následníky určité půlhrany, projdeme po vnitřní nebo vnější hranici její sousední oblasti. Tímto se dostáváme také k situaci oblastí, které mohou mít dvě oddělené hranice (vnitřní a vnější) jako je tomu u oblasti f_1 z obrázku 2. Nicméně i zde platí úmluva o poloze oblasti nalevo vůči svým hraničním půlhranám, čili půlhrany vnější hranice míří proti směru hodinových ručiček, zatímco půlhrany vnitřní hranice míří po směru hodinových ručiček.

Kompletní záznamy o uzlech, půlhranách a oblastech obsažené v tabulkách pak nazýváme *dvojitě souvislý seznam hran*, viz tabulky:

Uzel	Souřadnice	Vycházející půlhrana
v_0	$[0;4]$	$e_{0,1}$
v_2	$[7;2]$	$e_{2,5}$
v_3	$[9;0]$	$e_{3,2}$
v_7	$[2;6]$	$e_{7,8}$
$\vdots$	$\vdots$	$\vdots$

Oblast	Vnější hranice	Vnitřní hranice
f_1	$e_{1,2}$	$e_{8,9}$
f_2	$e_{3,4}$	nil
f_3	$e_{9,8}$	nil
f_4	nil	$e_{6,5}$
$\vdots$	$\vdots$	$\vdots$

Půlhrana	Výchozí uzel	Dvojče	Sousední oblast	Následník	Předchůdce
$e_{1,0}$	v_1	$e_{0,1}$	f_4	$e_{0,6}$	$e_{2,1}$
$e_{2,5}$	v_2	$e_{5,2}$	f_1	$e_{5,6}$	$e_{1,2}$
$e_{8,7}$	v_8	$e_{7,8}$	f_3	$e_{9,8}$	$e_{7,9}$
$e_{9,7}$	v_9	$e_{7,9}$	f_1	$e_{7,8}$	$e_{8,9}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabulky nám ukazují obecný dvojitě souvislý seznam hran. Záleží však na konkrétním použití, zda všechny údaje tak, jak jsou zapsány, opravdu potřebujeme. Dokonce i některé potřebné údaje lze uchovat úsporněji.

Podíváme-li se na tabulku s uzly, zjistíme, že dvojice uzel a z něj vedoucí půlhrana se vyskytuje i v tabulce pro půlhrany, po řadě v druhém a prvním sloupci. Jelikož tabulka uzlů nemá pro úlohu lokalizace bodu další uplatnění v podobě informací navíc, které by obsahovala, můžeme ji zrušit. Zbývá pouze někde uložit souřadnice uzlů, což můžeme udělat v tabulce půlhran ve sloupci *Výchozí uzel*.

V tabulce oblastí nelze zřejmě vypustit nic, navíc musíme počítat ještě s jedním sloupcem, který bude obsahovat název oblasti. Tento název bude totiž konečným výstupem lokalizace ve chvíli, kdy nalezneme oblast, v níž se daný bod nachází.

Poslední možnosti ušetření hledejme u půlhran, vhodné pojmenování nám velmi pomůže. Podívejme se zpátky na obrázek 2 a všimněme si, že tam je každá půlhrana označena vždy dvěma čísly, z nichž to první označuje výchozí uzel a to druhé uzel, kam půlhrana míří. Ze sloupce *Výchozí uzel* tedy lze s klidným svědomím odejmout název uzlu - víme přeci, že $e_{i,j}$ vychází z uzlu v_i - a ponechat v něm jen souřadnice onoho uzlu. Sloupec *Dvojče* snadno oželíme, víme-li, že dvojčetem k půlhraně $e_{i,j}$ bude vždy $e_{j,i}$. Sloupce *Sousední oblast* a *Následník* ponecháme v původní podobě. Konečně poslední sloupec také vynecháme, jelikož přechůdce půlhrany lze zjistit pomocí sloupce *Následník*. Hledanou půlhranu najdeme ve sloupci *Následník* a ve sloupci *Půlhrana* v příslušném řádku je onen hledaný předchůdce. Zjednodušené tabulky pak vypadají takto:

Oblast	Vnější hranice	Vnitřní hranice
f_1	$e_{1,2}$	$e_{8,9}$
f_2	$e_{3,4}$	nil
f_3	$e_{9,8}$	nil
f_4	nil	$e_{6,5}$

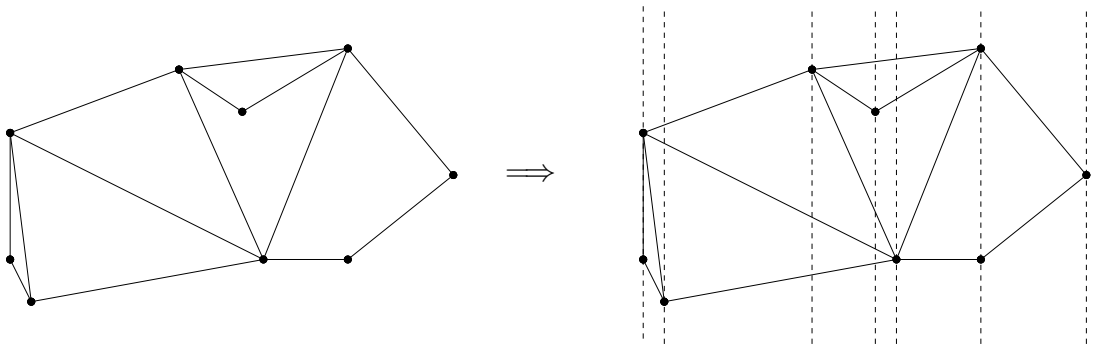
Půlhrana	Výchozí uzel	Sousední oblast	Následník
$e_{1,0}$	$[3;1]$	f_4	$e_{0,6}$
$e_{2,5}$	$[7;2]$	f_1	$e_{5,6}$
$e_{8,7}$	$[5;7]$	f_3	$e_{9,8}$
$e_{9,7}$	$[4;3]$	f_1	$e_{7,8}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Pro implementaci tabulek bych zde rád zdůraznil dvě věci, které nejsou z výše uvedených příkladů patrné. Zaprvé namísto názvů půlhran a oblastí ve sloupcích *Vnější hranice, Vnitřní hranice, Sousední oblast* a *Následník* používáme ukazatele, nikoliv proměnné. Zadruhé každá oblast může mít několik podoblastí, čili sledováním následníků jedné půlhrany nemusíme být schopni projít po celé vnitřní hranici její sousední oblasti. Ve sloupci *Vnitřní hranice* proto musí být uvedena půlhrana pro každou podoblast dané oblasti.

1.2 Lichoběžníková mapa $\mathcal{T}(S)$

Abychom se přiblížili vhodnému vyjádření zadané mapy začněme s jednoduchou strukturou. Mějme nějaké rovinné podrozdělení $\mathcal{S}$ dané n hranami. Předpokládáme, že se tyto hrany nekříží. Přesněji dvě různé hrany mají právě jeden nebo žádný společný sousední uzel. Jinými slovy hrany se zde nikdy neprotínají, jsou vždy disjunktní.

Představme si podrozdělení $\mathcal{S}$, abychom v něm mohli vyhledávat veďme každým uzlem vertikálně přímku, viz obrázek 3. Máme-li zadán bod q, jehož polohu máme zjistit, postupujeme následovně. Nejdříve se podíváme na jeho x-ovou souřadnici a porovnáme ji se souřadnicemi jednotlivých uzlů, tím určíme, mezi kterými dvěma uzly se nachází. Dále již prohledáváme jen pás určený těmito dvěma uzly. Tento pás obsahuje jen nekřížící se hrany, proto je můžeme seřadit od nejnižší po nejvyšší. Bereme postupně jednotlivé hrany a vždy se ptáme, zda je hledaný bod nad nebo pod ní. Takto se dostaneme do části podrozdělení, která již přísluší jedné původní oblasti a její název je výsledkem hledání.



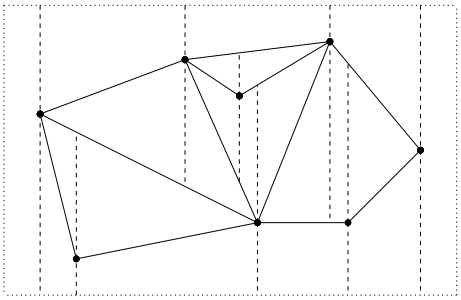
Obrázek 3

Vidíme, že takový systém vyhledávání funguje, problém však spočívá v přílišné velikosti potřebné datové struktury. Vždyť podrozdělení $\mathcal{S}$ má až $2n$ uzlů, museli bychom tedy

uchovávat informace až o $2n + 1$ pásech, přičemž každý pás může obsahovat až n úseček. Dohromady bychom získali kvadratickou složitost, což téměř znemožňuje použití takového systému v praxi.

Myšlenka podobného hledání však není špatná. Potřebujeme ji jen upravit tak, aby počet úseček nebyl o mnoho větší než v podrozdělení $\mathcal{S}$. Pokoušíme se zde vlastně o zjemnění, které usnadní vyhledávání, ale nezpůsobí příliš velký nárůst složitosti. Zkusme proto každým uzlem vést vertikální přímku, která povede na obě strany jen tak daleko, dokud se nedotkne již existující hrany, viz obrázek 4. O zjemnění se určitě jedná a v sekci 1.4. si dokážeme, že skutečně nemá o mnoho větší složitost než původní podrozdělení $\mathcal{S}$.

Učiňme nyní dvě zjednodušení, která nám usnadní práci. Prvním z nich bude vypořádání se s neohraničenou oblastí, v níž se nachází všech n hran. Tato oblast může být poměrně veliká, ale nám záleží pouze na tom, zda v ní bod leží, či nikoliv. Proto zavedeme obdélník $\mathcal{R}$ obsahující všechny uzly. Cokoliv bude ležet vně tohoto obdélníku, patří do neohraničené oblasti. Vnitřek obdélníka $\mathcal{R}$ podrobíme hlubšímu zkoumání. Druhé zjednodušení již není tak snadno pochopitelné. Budeme předpokládat, že žádné dva uzly nemají stejnou x-ovou souřadnici. V praxi se může stát, že toto často platit nebude. Nicméně pro usnadnění pochopení následujících pasáží to nyní předpokládejme. V kapitole 3 ukážeme, jak se problémové polohy bodů řeší.



Obrázek 4: Lichoběžníková mapa ohraničená obdélníkem $\mathcal{R}$

Nyní již tedy máme *lichoběžníkovou mapu*, viz obrázek 4. Snadno lze vidět, že všechny podoblasti v $\mathcal{R}$ jsou lichoběžníky (odtud ostatně název mapy), případně trojúhelníky. Na trojúhelníky však lze pohlížet jako na degenerované lichoběžníky s jednou hranou délky nula. Precizujme nyní samotné vytvoření lichoběžníkové mapy. Nechť S označuje množinu všech hran obsažených v podrozdělení $\mathcal{S}$. Pro odlišení S a $\mathcal{S}$ budeme prvkům množiny S říkat úsečky a jejich koncovým uzlům body. Jak již víme, žádné dvě úsečky z S se nekříží a všechny se nacházejí uvnitř obdélníka $\mathcal{R}$. Navíc předpokládáme, že žádné dva různé koncové body úseček z S nemají stejnou x-ovou souřadnici. Takové množině S pak říkáme *množina úseček v obecné poloze*. Lichoběžníkovou mapu potom získáme vedením svislých prodloužení oběma koncovými body každé úsečky v S. *Horní svislé prodloužení* i *dolní svislé prodloužení* každého bodu vedou jen tak daleko, dokud se nestřetnou s jinou úsečkou z S, popřípadě se stranou obdélníka $\mathcal{R}$. Jelikož lichoběžníková mapa vzniká v závislosti na množině S, značíme ji $\mathcal{T}(S)$.

1.3 Jednotlivé lichoběžníky

Podívejme se nyní obecně na každý možný typ lichoběžníka, který se může v lichoběžníkové mapě objevit. Každý lichoběžník Δ má jednu nebo dvě svislé strany a právě dvě strany, které nejsou svislé. Označme $top(\Delta)$ a $bottom(\Delta)$ úsečky obsahující nesvislé strany Δ. Snadno odvodíme, že se vždy jedná o úsečky z S nebo o vodorovné strany obdélníka $\mathcal{R}$.

Vertikální strany mohou být horním nebo dolním svislým prodloužením nebo svislou stranou obdélníka $\mathcal{R}$. Přesněji můžeme rozlišit pět různých situací pro levou stranu i pro pravou stranu. Podívejme se tedy, jak může obecně vypadat levá svislá strana libovolného lichoběžníka Δ:

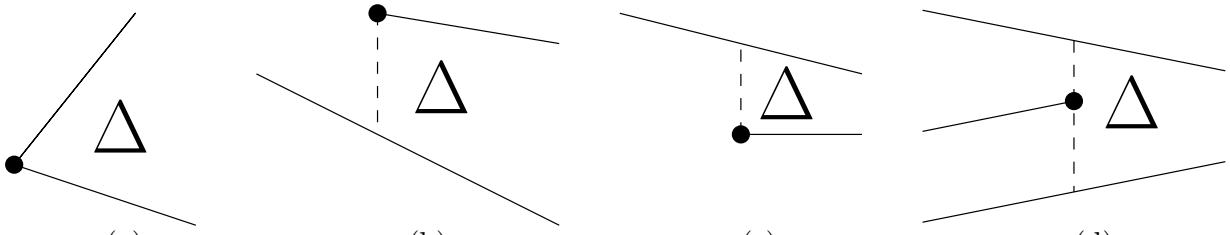
(a) Degeneruje do jednoho bodu, ten je zároveň levým koncovým bodem $top(\Delta)$ i $bottom(\Delta)$.

(b) Tvoří ji dolní svislé prodloužení levého koncového bodu $top(\Delta)$.

(c) Tvoří ji horní svislé prodloužení levého koncového bodu $bottom(\Delta)$.

(d) Skládá se z horního i dolního prodloužení bodu, který je pravým koncovým bodem nějaké třetí úsečky.

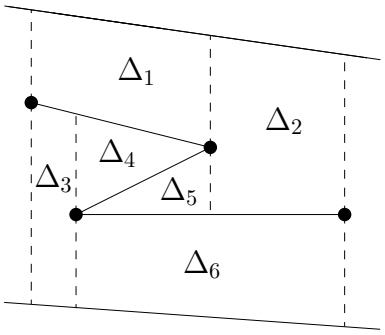
(e) Rovná se levé svislé straně obdélníka $\mathcal{R}$. Toto platí pro právě jeden lichoběžník v lichoběžníkové mapě $\mathcal{T}(S)$ a to ten, který se nachází nejvíce vlevo.



Výčet možných situací pro pravou svislou stranu je analogický. Ověřme si, že tyto výčty jsou již úplné. Svislá strana nenulové délky původně v S žádná nebyla, protože jsme předpokládali, že každé dva body měly různé x-ové souřadnice. Degenerovaná strana je řešena v případě (a). Nenulová strana musela tedy vzniknout buď jako nějaké svislé prodloužení (případy (b),(c) a (d)) nebo jako strana obdélníka $\mathcal{R}$, který jsme přidali do původní mapy, což postihuje přípac (e). Nic dalšího jsme k podrozdělení nepřidávali, proto jinak svislá strana vzniknout nemohla a náš výčet je kompletní.

Lze si všimnout, že pro každý lichoběžník Δ, vyjma toho nejvíce vlevo, je jeho levá vertikální strana do určité míry určena jedním koncovým bodem p. Tento bod je buďto levým koncovým bodem $top(\Delta)$ či $bottom(\Delta)$, případně obou, nebo pravým koncovým bodem třetí úsečky. Označíme jej $leftp(\Delta)$, přičemž pro případ (e) definujeme jako $leftp(\Delta)$ levý dolní vrchol obdélníka $\mathcal{R}$. Obdobným způsobem definujeme $rightp(\Delta)$. Pro každý lichoběžník Δ tedy máme čtyři údaje, které jej jednoznačně určují: $top(\Delta)$, $bottom(\Delta)$, $leftp(\Delta)$ a $rightp(\Delta)$.

Zaveďme označení, že lichoběžník Δ_i je *sousedem* pro lichoběžník Δ_j a obráceně (tj. vztah býti sousedem je symetrický) pokud Δ_i a Δ_j mají společnou svislou stranu a platí, že $top(\Delta_i) = top(\Delta_j)$ nebo $bottom(\Delta_i) = bottom(\Delta_j)$. Bavíme-li se o množině úseček v obecné poloze, pak má každý lichoběžník nejvýše čtyři takovéto sousedy. Konkrétně jim říkejme *levý horní soused*, *levý dolní soused*, *pravý horní soused* a *pravý dolní soused*.



Obrázek 5

Na obrázku 5 vidíme, že Δ_1 má pouze jednoho pravého souseda, a to pravého dolního, kterým je Δ_2. Lichoběžník Δ_2 má naproti tomu dva levé sousedy, přičemž Δ_1 je tím horním a Δ_5 tím dolním. Dobře si všimněte, že Δ_4 nemá žádného pravého souseda a Δ_5 nemá žádného levého souseda, což je způsobeno tím, že mají své vertikální strany v těchto směrech degenerované.

1.4 Složitost $\mathcal{T}(S)$

Když jsme si nyní předvedli, jak mohou lichoběžníky v $\mathcal{T}(S)$ obecně vypadat, můžeme již ukázat, že složitost $\mathcal{T}(S)$ není o mnoho větší než složitost původního podrozdělení $\mathcal{S}$. Jinými slovy počet uzlů a lichoběžníků v $\mathcal{T}(S)$ závisí jen lineárně na počtu hran n.

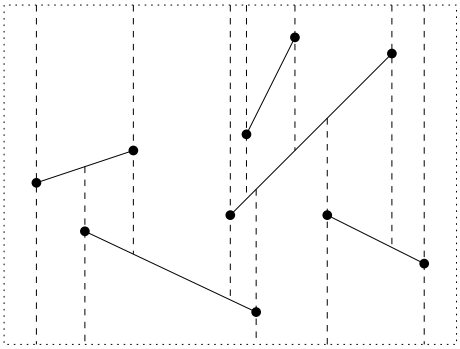
Lemma 1. *Lichoběžníková mapa $\mathcal{T}(S)$ množiny S, kterou tvoří n úseček v obecné poloze, obsahuje nejvýše $6n + 4$ uzlů a nejvýše $3n + 1$ lichoběžníků.*

Důkaz. Uzly v $\mathcal{T}(S)$ mohou být vrcholy obdélníka $\mathcal{R}$, ty jsou 4. Dále se může jednat o koncové body úseček z S, kterých je nejvýše $2n$. A konečně to mohou být uzly, které vznikly v místech, kde se horní a dolní svislá prodloužení koncových bodů dotýkají úseček z S nebo stran obdélníka $\mathcal{R}$. Jelikož koncových bodů v S je nejvýše $2n$ a z každého vedou dvě prodloužení, je těchto nově vzniklých uzlů maximálně $2(2n)$. Celkem máme tedy nejvýše $4 + 2n + 2(2n) = 6n + 4$ uzlů.

Počet možných lichoběžníků závisí na počtu uzlů. S ohledem na výčet všech možností, jak může vypadat levá strana libovolného lichoběžníka, který jsme učinili v minulé sekci, spočítejme lichoběžníky podle jejich *leftp*(Δ). Každému lichoběžníku Δ zřejmě přísluší právě jeden *leftp*(Δ). Levý koncový bod úsečky může být *leftp*(Δ) nejvýše pro dva různé

lichoběžníky, viz případy (b) a (c). Pravý koncový bod je *leftp*(Δ) pro nejvýše jeden lichoběžník, viz (d). Není ovšem příliš jasné, jestli to platí i u trojúhelníků jako v případě (a). Na první pohled to vypadá, že jeden *leftp*(Δ) patří třem lichoběžníkům, ale uvědomme si, že na místě levého vrcholu trojúhelníka se nachází dva body - levý koncový bod *top*(Δ) a levý koncový bod *bottom*(Δ), necháme proto bod z *top*(Δ) jako *leftp*(Δ) pro horní lichoběžník a bod z *bottom*(Δ) jako *leftp*(Δ) pro dolní lichoběžník a trojúhelník. Trojúhelníky tedy uvedené pravidlo také splňují. Jelikož máme n úseček, dostáváme tímto způsobem odhad $2n + n$. Poslední možná situace, totiž (e) nastává pro právě jeden lichoběžník, a to ten nejvíce vlevo, pro nějž je *leftp*(Δ) levý dolní vrchol obdélníka $\mathcal{R}$. Dohromady proto $\mathcal{T}(S)$ obsahuje maximálně $2n + n + 1 = 3n + 1$ lichoběžníků. □

Hraniční případ pro oba horní odhady, nastává tehdy, pokud se žádné dva uzly nepřekrývají, viz obrázek 6. Pro $n = 5$ zde máme v lichoběžníkové mapě $\mathcal{T}(S)$ opravdu $6n + 4 = 34$ uzlů a $3n + 1 = 16$ lichoběžníků.



Obrázek 6

Taková situace sice v praxi těžko nastane, neboť pak bychom vyhledávali v mapě s pouze jednou oblastí. Jelikož však používáme vyhledávací algoritmus ještě před samotným hledáním zadaného bodu q, musíme i tuto situaci uvažovat. Při konstrukci vyhledávací struktury (viz sekce 2.1) totiž do prázdné mapy postupně přidáváme jednotlivé úsečky.

1.5 Vyhledávací struktura $\mathcal{D}$

Připomeňme, že na začátku dostaneme podrozdělení $\mathcal{S}$. Pro lokalizaci bodu potřebujeme vytvořit lichoběžníkovou mapu $\mathcal{T}(S)$ popsanou v sekci 1.2. a zároveň vyhledávací strukturu, která by nám umožnila v $\mathcal{T}(S)$ efektivně vyhledávat. Nyní se tedy zaměřme na podobu vyhledávací struktury $\mathcal{D}$.

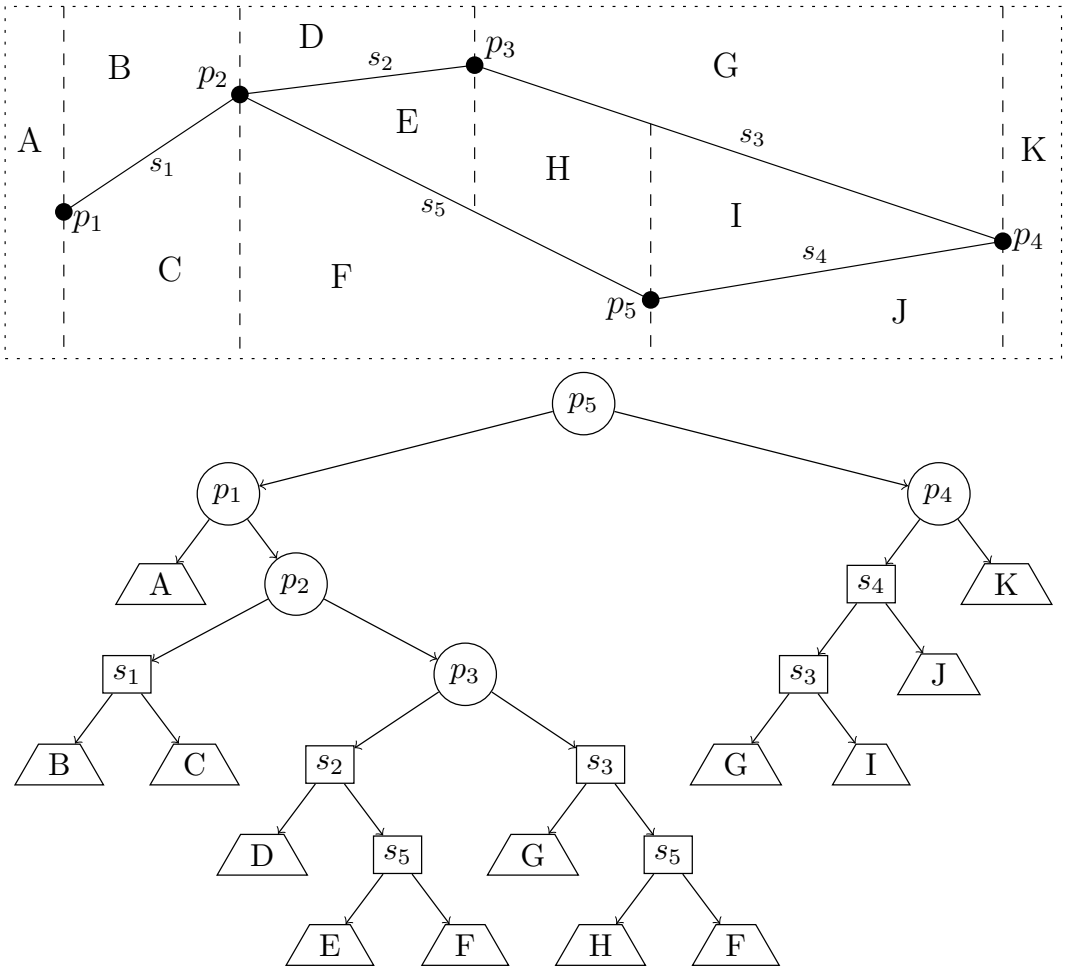
$\mathcal{D}$ je acyklický orientovaný graf s jedním kořenem, kde z každého vnitřního uzlu vedou právě dvě hrany. Z koncových uzlů nevede žádná hrana, budeme je zde označovat slovem *listy*. Vnitřní uzly v tomto grafu jsou dvojího typu a to x-ové uzly, které nesou názvy koncového bodu nějaké úsečky z S, a y-ové uzly pojmenované úsečkou z S, kterou reprezentují. V listech tohoto grafu se pak nalézají jednotlivé lichoběžníky z $\mathcal{T}(S)$. Pokud bude

z kontextu jasné, že hovoříme o $\mathcal{D}$, budeme někdy, kvůli větší srozumitelnosti, místo o uzlu příslušejícímu bodu p (resp. úsečce s) mluvit o uzlu p (resp. s).

Při vyhledávání oblasti, v níž leží zadaný bod q, procházíme graf $\mathcal{D}$ od kořene k listům a v každém x-ovém uzlu posuzujeme, zda hledaný bod leží napravo či nalevo od bodu v x-ovém uzlu. V y-ových uzlech se ptáme, jestli q leží nad nebo pod danou úsečkou. Když jsme v y-ovém uzlu grafu $\mathcal{D}$, musíme se rovněž ujistit o tom, že pomyslná svislá přímka procházející bodem q protíná úsečku, jejíž pojmenování tento uzel nese, jinak by totiž otázka nad či pod postrádala smysl.

Jak jste si zajisté všimli, nevěnujeme se případům, kdy q leží na svislém prodloužení nějakého bodu, ani případům, kdy se q nalézá přímo na některé z n úseček. Tyto výjimky vyřešíme v následující sekci. Prozatím budeme předpokládat, že se q nalézá vždy uvnitř nějakého lichoběžníka v $\mathcal{T}(S)$, proto popsané vyhledávání skončí vždy v některém z listů.

Datová struktuta $\mathcal{D}$ a lichoběžníková mapa $\mathcal{T}(S)$ jsou vzájemně propojeny. To znamená, že každý list v $\mathcal{D}$ obsahuje ukazatel na lichoběžník v $\mathcal{T}(S)$, který reprezentuje a naopak. Uveďme si nyní na příkladu vztahy mezi $\mathcal{D}$ a $\mathcal{T}(S)$:



Obrázek 7: Příklad lichoběžníkové mapy a jí příslušné vyhledávací struktury

Struktura $\mathcal{D}$ na obrázku 7 byla konstruována pro úsečky v pořadí s_4, s_1, s_3, s_2, s_5. Graf je orientován tak, že umístění nalevo či napravo od x-ových uzlů odpovídá jejich skutečné poloze vůči bodům v těchto uzlech. Umístění nalevo od y-ového uzlu znamená polohu nad danou úsečkou a umístění vpravo polohu pod úsečkou.

Každý x-ový uzel v $\mathcal{D}$ je na obrázku právě jednou. Toto platí obecně a odpovídá to vyhledávání v pásech, jak bylo popsáno v sekci 1.2. Naproti tomu y-ové uzly se v $\mathcal{D}$ mohou vyskytovat několikrát. Každý y-ový uzel se zřejmě v $\mathcal{D}$ vyskytuje alespoň jednou. Další výskyt úsečky s_i je vždy způsoben tím, že na ní končí svislé prodloužení některého z koncových bodů jiné úsečky, která se v pořadí úseček nachází před s_i. Například uzel s_3 se v $\mathcal{D}$ vyskytuje dvakrát. Na s_3 totiž v lichoběžníkové mapě $\mathcal{T}(S)$ končí svislé prodloužení uzlu p_5, který je koncovým bodem úsečky s_4, a ta se v pořadí úseček vyskytuje před s_3. Bod p_5 je navíc i koncovým bodem s_5, ale ta se nachází v pořadí až za s_3, což již další výskyt s_3 nezpůsobí.

Listy odpovídající lichoběžníkům jsou na obrázku 7 z důvodu lepší přehlednosti grafu uvedeny víckrát, viz lichoběžníky F a G. Znamená to, že v dané vyhledávací struktuře $\mathcal{D}$ vede do takového listu více možných cest. V praxi je ale každý lichoběžník v $\mathcal{D}$ právě jednou, pouze do něj mohou vést hrany z více vnitřních uzlů zároveň. Odtud také plyne, proč jsme při definici $\mathcal{D}$ nepoužili pojmu strom, neboť se v těchto případech o strom nejedná.

Všimněme si rovněž toho, že úsečka s_i se v y-ovém uzlu v $\mathcal{D}$ objeví vždy až tehdy, když na otázku, zda hledaný bod leží nad či pod s_i, lze jednoznačně odpovědět. Buďto se v grafu nad uzlem s úsečkou s_i již vyskytují oba uzly odpovídající jejím koncovým bodům (viz např. uzel s_1), nebo se tam místo nich vyskytují uzly koncových bodů, které jsou nad či pod úsečkou s_i a zároveň jsou ještě více vpravo (respektive více vlevo) než její levý (respektive pravý) koncový bod. Tento druhý případ nastal u obou uzlů pro úsečku s_3. Ten, který se v grafu nachází více vpravo, má nad sebou uzel odpovídající jeho koncovému bodu p_4. Místo uzlu p_3 zde však supluje uzel p_5. Bod p_5 je skutečně pod úsečkou s_3 a zároveň více vpravo než p_3. Nalézáme-li se tedy v uzlu pro úsečku s_i pro jakékoliv i, nacházíme se buď právě v pásu tvořeném jejími koncovými body, nebo v pásu ještě užším.

1.6 Postup vyhledávání bodu q

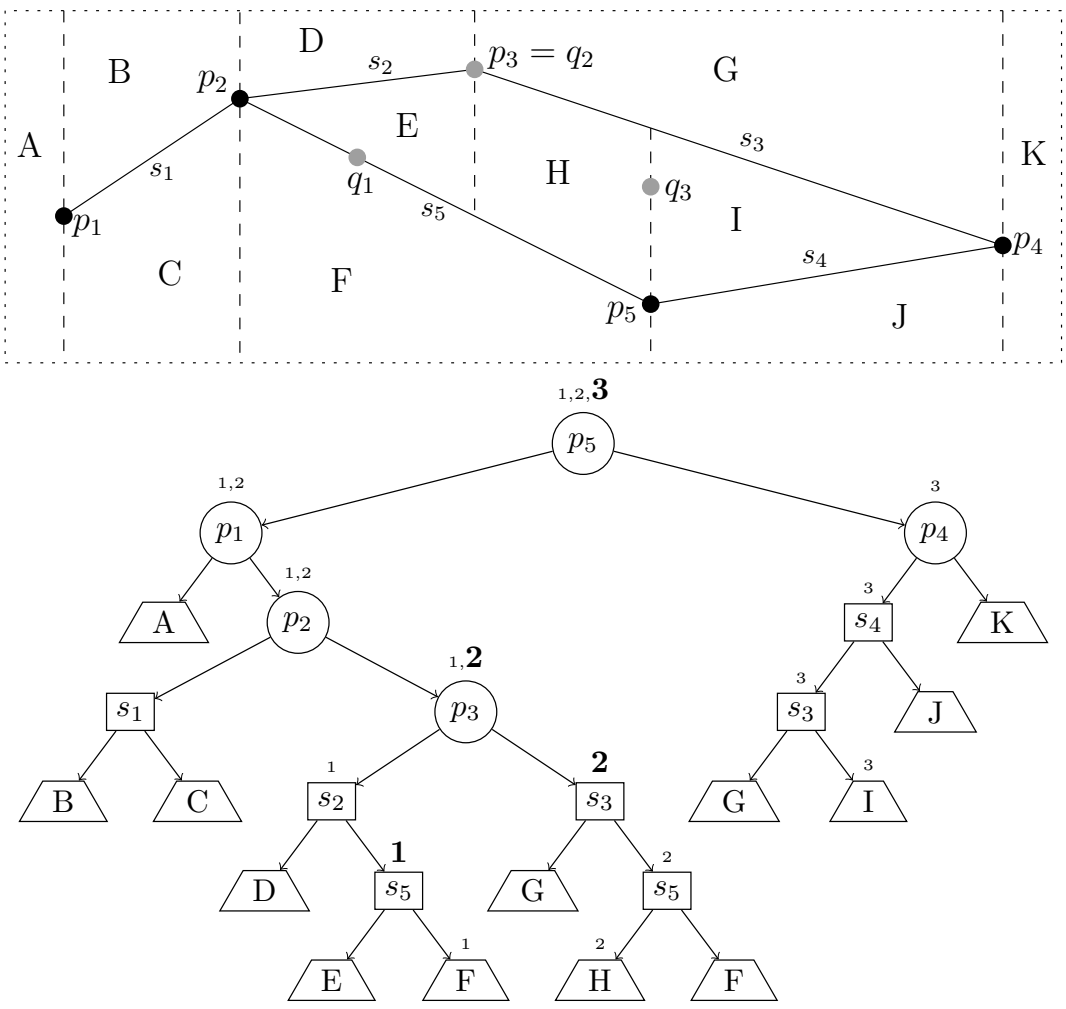
Vraťme se nyní k původní úloze lokalizace bodu. Na vstupu dostaneme rovinné podrozdělení $\mathcal{S}$ v podobě dvojitě souvislého seznamu n hran a hledaný bod q. Dále dojde k vytvoření lichoběžníkové mapy $\mathcal{T}(S)$ a datové struktury $\mathcal{D}$ pomocí algoritmu, který popíšeme v kapitole 2. Již na jeho začátku ověříme, zda hledaný bod náleží do obdélníka $\mathcal{R}$. V případě, že nikoliv, vyhledávání skončí s výsledkem, že bod q leží v neohraničené oblasti. Jestliže však $q \in \mathcal{R}$, budeme procházet grafem $\mathcal{D}$ a posuzovat polohu bodu q vůči koncovým bodům a úsečkám z S.

Jestliže bod q leží uvnitř nějakého lichoběžníka, vyhledávání dospěje do odpovídajícího listu v $\mathcal{D}$. Ve dvojitě souvislém seznamu hran máme pro každou půlhranu $e_{i,j}$ uveden ukazatel na oblast, která se nachází nalevo od ní. Využijeme toho a všechny úsečky budeme reprezentovat jako hrany orientované zleva doprava. Pro nalezený lichoběžník Δ v $\mathcal{T}(S)$

se tedy podíváme na oblast přilehlou k $bottom(\Delta)$. Název této oblasti je potom finálním výstupem lokalizace bodu q.

Shoduje-li se hledaný bod s některým z bodů lichoběžníkové mapy nebo leží uvnitř některé ze zadaných úseček, výsledkem hledání může být skutečnost, že bod leží přímo na původní mapě. V řadě aplikací ovšem takový výsledek nebude vyhovující, proto lze v takovém případě bod q symbolicky posunout. Rovněž situace, kdy se bod q nachází na některém ze svislých prodloužení, brání dokončení vyhledávání požadované oblasti. Vyřešme nyní tyto tři komplikace.

Při procházení vyhledávací strukturou, se můžeme v x-ových uzlech ptát, zda q leží vlevo od daného bodu, jestliže ne, pak pokračujeme automaticky doprava. Podobně v y-ových uzlech zjistíme, jestli se q nachází nad úsečkou, když ne, postupujeme dolů. Symbolicky tedy posouváme bod q doprava nebo dolů, případně oběma směry zároveň. Tímto zaručíme, že vyhledávání pokaždé skončí v některém z lichoběžníků.



Obrázek 8: Speciální polohy hledaného bodu q

Obrázek 8 ukazuje řešené tři případy, kdy q_1 leží na úsečce, q_2 se shoduje s jedním z bodů a q_3 se nachází uvnitř svislého prodloužení. Každému z hledaných bodů přísluší v grafu $\mathcal{D}$ cesta, kterou při prohledávání procházíme. Pro každý bod q_i, kde $i \in \{1, 2, 3\}$ tedy projdeme postupně uzly označené i, uzel s velkým tučným číslem i znamená, že v něm dojde k symbolickému posunutí podle výše definovaných pravidel.

Ověřme ještě, že takový postup zachová oblast, v níž se q nalézal. Leží-li q na svislém prodloužení nějakého uzlu p, právě jednou (v uzlu p) posuneme q doprava. Víckrát se to nemůže stát, neboť předpokládáme body s různými x-ovými souřadnicemi. Skončíme v lichoběžníku, který má q na své levé straně. Jelikož q patřil svislému prodloužení, oblast příslušející nalezenému lichoběžníku se rovná oblasti, do níž q patří. Nacházel-li se bod q uvnitř úsečky s, při průchodu grafem $\mathcal{D}$ se jen jednou (v uzlu s) použije symbolické posunutí dolů. Uzel s se sice ve vyhledávací struktuře může objevit víckrát, ale ne na cestě k danému bodu q, odtud pouze jedno posunutí. Poloha q na úsečce znamená, že leží na hranici dvou oblastí, nalezený lichoběžník bude patřit do jedné z nich, což je opět korektní. Poslední případ kombinuje oba předchozí, když se q shoduje s koncovým bodem p úsečky s. Zároveň tedy leží na svislém prodloužení i na úsečce, proto se obě uvedená posunutí provedou, každé opět jen jednou. Výsledný lichoběžník bude jeden z těch, které mají bod q na své hranici a který leží vpravo dole od bodu q. Jelikož se hledaný bod nacházel na hranici několika oblastí, najdeme jednu z nich.

Uvedli jsme tedy celý postup od zadání ve formě podrozdělelní $\mathbb{S}$ až po nalezení oblasti, který při lokalizaci bodu používáme. Výsledná oblast buď opravdu daný bod q obsahuje, nebo q leží na její hranici.

Kapitola 2

Náhodnostní přírůstkový algoritmus

Uveďme pro úplnost, co značí jednotlivé charakteristiky algoritmu v názvu kapitoly. Označení *náhodnostní* znamená, že uvnitř algoritmu figuruje princip náhody, v našem případě náhodné uspořádání. Některá uspořádání nemusí vést k vytvoření dobré datové struktury. S určitou pravděpodobností lze však předpokládat, že průměr přes všechny potenciální výsledky uspořádání dobrý bude. Proto se také někdy používá označení *pravděpodobnostní*.

Algoritmus nazýváme *přírůstkový*, neboť se v jeho průběhu postupně přidávají jednotlivé úsečky z množiny S. Vždy se tak vytvoří dočasná lichoběžníková mapa $\mathcal{T}$ a jí příslušná vyhledávací struktura $\mathcal{D}$. Teprve na závěr získáme konečnou podobu jak $\mathcal{T}(S)$ tak $\mathcal{D}$.

2.1 Konstrukce $\mathcal{D}$ a $\mathcal{T}(S)$

Uveďme nyní algoritmus, který současně vytvoří obě požadované struktury.

Algoritmus MAPA(S)
Vstup: Množina S navzájem se nekřížících n úseček
Výstup: Lichoběžníková mapa $\mathcal{T}(S)$ a k ní příslušející vyhledávací struktura $\mathcal{D}$

1. Urči obdélník $\mathcal{R}$ tak, aby obsahoval všechny úsečky z S. Inicializuj lichoběžníkovou mapu $\mathcal{T}$ a vyhledávací strukturu $\mathcal{D}$.

2. Urči náhodné pořadí úseček $s_1, s_2, \ldots, s_n$ z S.

3. **for** $i \leftarrow 1$ **to** n

4. **do** Najdi množinu lichoběžníků $\Delta_0, \Delta_1, \ldots, \Delta_k$ v $\mathcal{T}$, které úsečka s_i protíná ve vnitřním bodě.

5. Z $\mathcal{T}$ odstraň $\Delta_0, \Delta_1, \ldots, \Delta_k$ a nahraď je novými lichoběžníky, které vzniknou přidáním úsečky s_i do $\mathcal{T}$.

6. Z $\mathcal{D}$ odstraň listy příslušející lichoběžníkům $\Delta_0, \Delta_1, \ldots, \Delta_k$ a vytvoř nové listy pro nově vzniklé lichoběžníky v $\mathcal{T}$. Propoj nové listy s již existujícími uzly přidáním nových uzlů.

Tento algoritmus vytvoří obě požadované struktury. Podívejme se nyní podrobněji na jeho jednotlivé kroky. Protože používáme *přírůstkový* algoritmus, zavedeme označení pro množinu již přidaných úseček $S_i := \{s_1, s_2, \ldots, s_i\}$, přičemž klademe $S_0 := \emptyset$.

2.2 Obdélník $\mathcal{R}$ a inicializace $\mathcal{T}$ a $\mathcal{D}$

Obdélník $\mathcal{R}$ určíme pomocí souřadnic jeho čtyř vrcholů $R_{ld}, R_{pd}, R_{lh}, R_{ph}$, kde R_{ld} znamená *levý dolní* vrchol, ostatní analogicky. V následujících přiřazeních $[x_i, y_i]$ značí souřadnice počátečního bodu i-té půlhrany. Funkcemi *ceiling* (resp. *floor*) máme na mysli *horní celou část* daného čísla (resp. *dolní celou část*). Snadno se ověří, že tento výpočet zaručí, že všechny body množiny S budou ležet uvnitř $\mathcal{R}$.

$$
\begin{array}{lll}
x_{min} := \min\limits_{i=1,\ldots,2n} x_i & x_1 := ceiling(x_{min}) - 1 & R_{ld} := [x_1, y_1] \\
x_{max} := \max\limits_{i=1,\ldots,2n} x_i & x_2 := floor(x_{max}) + 1 & R_{pd} := [x_2, y_1] \\
y_{min} := \min\limits_{i=1,\ldots,2n} y_i & y_1 := ceiling(y_{min}) - 1 & R_{lh} := [x_1, y_2] \\
y_{max} := \max\limits_{i=1,\ldots,2n} y_i & y_2 := floor(y_{max}) + 1 & R_{ph} := [x_2, y_2]
\end{array}
$$

V tomto momentě se zeptáme, zda hledaný bod q vůbec leží v obdélníku $\mathcal{R}$, jestliže se nachází na jeho hranici nebo vně, víme, že q náleží neohraničené oblasti a celý algoritmus skončí. Pokud je uvnitř, pokračujeme dál. Množina úseček, se kterou nyní na počátku algoritmu pracujeme je prázdná množina S_0. Proto lichoběžníkovou mapu $\mathcal{T}(S_0)$ tvoří pouze obdélník $\mathcal{R}$. Obdobně datová struktura $\mathcal{D}$ příslušná $\mathcal{T}(S_0)$ obsahuje pouze jeden list, reprezentující obdélník $\mathcal{R}$.

2.3 Náhodné pořadí

Co se týče náhodného pořadí, předpokládáme existenci nějakého náhodného generátoru čísel, nazveme jej RANDOM(k), který pro zadané celé číslo k určí náhodné číslo v rozmezí od 1 do k. Potom použijeme následujícího algoritmu:

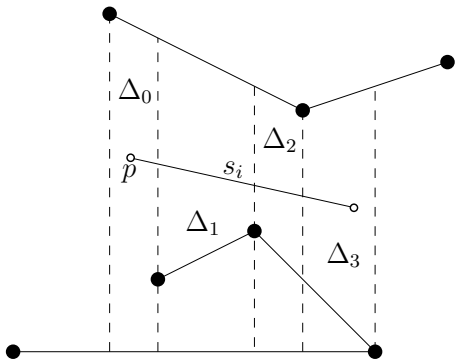
Algoritmus PERMUTACE(S)
Vstup: Uspořádaná množina n úseček $\{s_1, s_2, \ldots, s_n\}$
Výstup: Množina n úseček $\{s_1, s_2, \ldots, s_n\}$ s náhodným pořadím

1. **for** $k \leftarrow n$ **downto** 2
2. **do** $r \leftarrow$ RANDOM(k)
3. Zaměň s_k a s_r.

2.4 Lichoběžníky protínané úsečkou s_i

Ve čtvrtém, pátém a šestém kroku algoritmu MAPA se přidává i-tá úsečka s_i a obě struktury se poté aktualizují. Abychom věděli, kde všude se $\mathcal{T}$ a $\mathcal{D}$ změní, musíme nejprve najít ty lichoběžníky, které přidávaná úsečka s_i protíná. Z předpokladu, že se úsečky nekříží a z toho, že žádný koncový bod úsečky neleží uvnitř jiné úsečky, plyne, že úsečka s_i může protínat pouze svislá prodloužení některých koncových bodů již přidaných úseček. Seřadíme-li tedy protínané lichoběžníky $\Delta_0, \Delta_1, \ldots, \Delta_k$, stačí nalézt ten nejvíce vpravo (Δ_k) či nejvíce vlevo (Δ_0) a skrze znalost levých (resp. pravých) sousedů určíme všechny ostatní. My půjdeme cestou od levého koncového bodu s_i, proto hledejme Δ_0. Jakmile určíme Δ_j, tak pokud $\mathit{rightp}(\Delta_j)$ leží nad s_i, potom Δ_{j+1} bude pravý dolní soused Δ_j, v opačném případě se bude jednat o pravého horního souseda.

Pro nalezení lichoběžníka Δ_0, nejprve potřebujeme zjistit polohu levého koncového bodu úsečky s_i, nazvěme jej p. Pokud bod p dosud není přítomen v lichoběžníkové mapě $\mathcal{T}(S_{i-1})$ (viz obrázek 9), použijeme pro jeho nalezení vyhledávací algoritmus. Lichoběžníková mapa $\mathcal{T}(S_{i-1})$ a jí příslušná vyhledávací struktura $\mathcal{D}$ jsou již v této dílčí situaci použity. Vyhledávání skončí v listu $\mathcal{D}$, který přísluší hledanému lichoběžníku Δ_0.

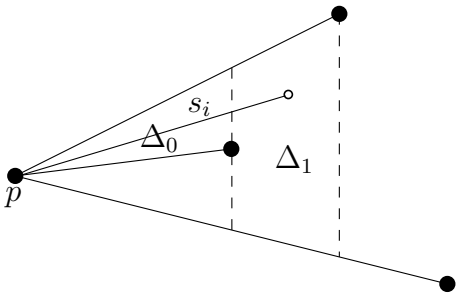


Obrázek 9: Levý koncový bod přidávané úsečky dosud neležel v mapě $\mathcal{T}$

Pokud je však bod p již součástí lichoběžníkové mapy $\mathcal{T}(S_{i-1})$ (viz obrázek 10), obdobné vyhledávání neuspěje, neboť v nějakém x-ovém uzlu struktury $\mathcal{D}$ bod p nebude ležet ani vpravo ani vlevo. Jelikož p je levým koncovým bodem, vyhledávání necháme pokračovat doprava. Uvědomme si, že toto nastane pouze jednou, neboť předpokládáme body s různými x-ovými souřadnicemi. Bod p je zároveň koncovým bodem nějaké úsečky s, která již v $\mathcal{T}(S_{i-1})$ leží. V příslušném y-ovém uzlu, tedy bod p nebude ležet ani nad ani pod. Porovnáme proto směrnice obou úseček. Označme pravý koncový bod úsečky s_i jako $q = [q_x, q_y]$, pravý koncový bod úsečky s jako $r = [r_x, r_y]$ a souřadnice bodu p jako $p = [p_x, p_y]$. Pak

$$\text{lichoběžník } \Delta_0 \text{ leží } \begin{cases} \text{pod } s & \text{je-li } \frac{r_y - p_y}{r_x - p_x} > \frac{q_y - p_y}{q_x - p_x} \text{ (tj. úsečka } s \text{ je strmější než } s_i \text{).} \\ \text{nad } s & \text{je-li } \frac{r_y - p_y}{r_x - p_x} < \frac{q_y - p_y}{q_x - p_x} \text{ (tj. úsečka } s \text{ je mírnější než } s_i \text{).} \end{cases}$$

Rovnost zřejmě nenastane, neboť $s_i \notin S_{i-1}$, úsečky jsou navzájem různé a maximální možný počet společných bodů - jedna - je již vyčerpán bodem p.



Obrázek 10: Levý koncový bod přidávané úsečky již je součástí mapy $\mathcal{T}$

Zbývá se pouze přesvědčit, že nerozhodná situace nemůže nastat nejprve v y-ovém uzlu. Kdyby tomu tak bylo, v tu chvíli se pro nějakou úsečku s musíme nacházet v pásu určeném jejími dvěma koncovými body nebo v nějakém ještě užším pásu. Jelikož hledáme bod p, levá strana tohoto pásu musí být určena jím samotným. Kdyby byla určena bodem více vpravo, vyhledávání by zřejmě do námi uvažovaného y-ového uzlu nedospělo. Protože se nacházíme v pásu určeném p, tak se již uzel příslušný bodu p nutně nalézá v grafu na cestě, po níž jsme do y-ového uzlu došli. Nerozhodná situace tedy nastala nejprve v x-ovém uzlu.

Jak vidíme, vyhledávání nakonec vždy dospěje do lichoběžníka Δ_0, který se ze všech lichoběžníků $\Delta_0, \Delta_1, \ldots, \Delta_k$ nachází nejvíce vlevo. Z jeho znalosti již výše uvedeným postupem získáme ostatní. Celkem lze čtvrtý krok algoritmu MAPA vyjádřit následovně:

Algoritmus SLEDUJ ÚSEČKU($\mathcal{T}(S_{i-1})$,s_i)
Vstup: Lichoběžníková mapa $\mathcal{T}(S_{i-1})$ a přidávaná úsečka s_i
Výstup: Lichoběžníky $\Delta_0, \Delta_1, \ldots, \Delta_k$ protínané úsečkou s_i

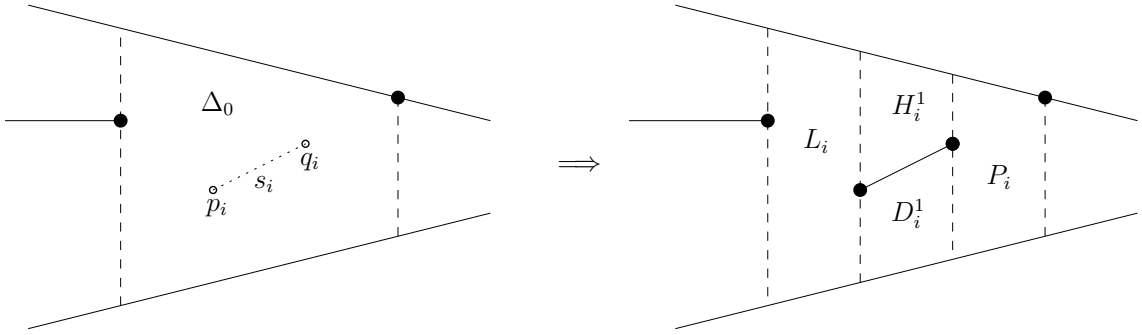
1. Označme levý a pravý koncový bod úsečky s_i jako p_i a q_i.
2. Proveďme vyhledávání ve struktuře $\mathcal{D}$ s bodem p_i, abychom nalezli lichoběžník Δ_0.
3. $j \leftarrow 0$;
4. **while** bod q_i leží napravo od *rightp*(Δ_j)
5. **do if** *rightp*(Δ_j) leží nad s_i
6. **then** Δ_{j+1} je pravý dolní soused Δ_j
7. **else** Δ_{j+1} je pravý horní soused Δ_j
8. $j \leftarrow j + 1$;
9. **return** $\Delta_0, \Delta_1, \ldots, \Delta_k$

2.5 Aktualizace $\mathcal{T}$

Nyní si objasníme aktualizaci lichoběžníkové mapy po přidání úsečky s_i, tedy jak z $\mathcal{T}(S_{i-1})$ vznikne $\mathcal{T}(S_i)$. Levý a pravý koncový bod úsečky si pojmenujme po řadě p_i a q_i. Označme si také nově přidávané lichoběžníky podle jejich polohy vzhledem k úsečce s_i. Lichoběžník nalevo (resp. napravo) od úsečky, pokud takový existuje budeme značit L_i (resp. P_i).

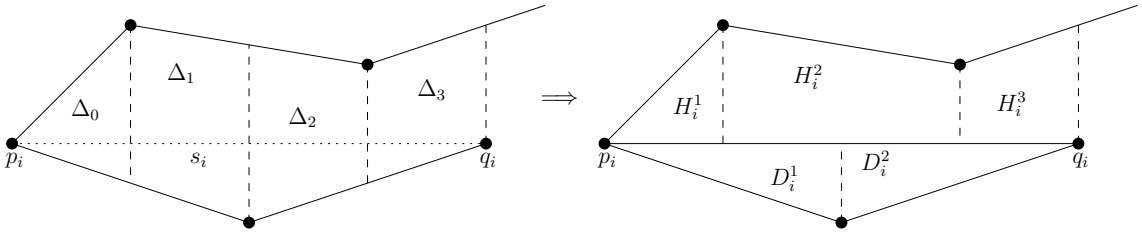
Lichoběžníky nad (resp. pod) nazvěme H_i^u (resp. D_i^v), přičemž indexy u a v udávají pořadí horních a dolních lichoběžníků.

Nejprve vyřešíme případ, kdy celá přidávaná úsečka leží uvnitř nějakého lichoběžníka, viz obrázek 11. Tehdy je protínaným lichoběžníkem pouze Δ_0. Levý lichoběžník L_i má stejný *top*, *bottom* a *leftp* jako Δ_0, ale $\mathit{rightp}(L_i) = p_i$, podobně P_i se od Δ_0 liší pouze v tom, že $\mathit{leftp}(P_i) = q_i$. Horní a dolní lichoběžník jsou také jednoznačně určeny pomocí s_i, p_i, q_i a informací o Δ_0.



Obrázek 11: Přidávaná úsečka leží celá uvnitř jednoho lichoběžníka

Dále si ukažme, jak algoritmus postupuje, v případě, kdy počet protínaných lichoběžníků je více než jedna, tedy kdy $k \neq 0$, a zároveň oba koncové body úsečky již leží v lichoběžníkové mapě, viz obrázek 12. Jelikož p_i už nepřidáváme, žádný lichoběžník L_i nevznikne. Víme, ale že na místě Δ_0 vzniknou H_i^1 a D_i^1, snadno vypočteme jejich *top*, *bottom* i *leftp*, tyto informace o nich zapíšeme a necháme pro tuto chvíli oba lichoběžníky „nedokončené“. Pro zjištění *rightp* potřebujeme vědět, jestli $\mathit{rightp}(\Delta_0)$ leží nad či pod úsečkou s_i, podle toho $\mathit{rightp}(H_i^1) := \mathit{rightp}(\Delta_0)$ nebo $\mathit{rightp}(D_i^1) := \mathit{rightp}(\Delta_0)$. Stejný postup aplikujeme i na další lichoběžníky $\Delta_1, \ldots, \Delta_{k-1}$. Poté všechny započaté lichoběžníky dokončíme přiřazeními $\mathit{rightp}(H_i^u) := q_i$ a $\mathit{rightp}(D_i^v) := q_i$. Nakonec podle přítomnosti či absence q_i v $\mathcal{T}(S_{i-1})$ určíme P_i, v našem případě tedy P_i nemáme.



Obrázek 12: Přidávaná úsečka protíná více lichoběžníků

Pro aktualizaci lichoběžníkové mapy musíme uvažovat všechny možnosti polohy úsečky s_i. Záleží na přítomnosti p_i a q_i v $\mathcal{T}(S_{i-1})$ a také na tom zda $k = 0$. V podobě pseudokódu vypadá celý postup takto:

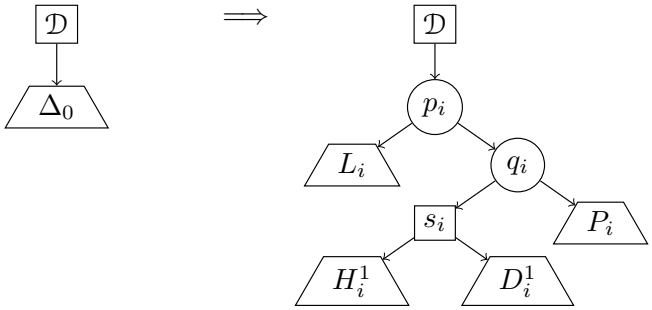
Algoritmus AKTUALIZUJ MAPU($\mathcal{T}(S_{i-1})$, s_i, $\Delta_0, \ldots, \Delta_k$)
Vstup: Lichoběžníková mapa $\mathcal{T}(S_{i-1})$, přidávaná úsečka s_i a množina lichoběžníků $\Delta_0, \Delta_1, \ldots, \Delta_k$ protínaných úsečkou s_i
Výstup: Aktualizovaná lichoběžníková mapa $\mathcal{T}(S_i)$

1. $u \leftarrow 1$; $v \leftarrow 1$;
2. $\mathit{leftp}(H_i^u) \leftarrow p_i$; $\mathit{top}(H_i^u) \leftarrow \mathit{top}(\Delta_0)$; $\mathit{bottom}(H_i^u) \leftarrow s_i$;
3. $\mathit{leftp}(D_i^v) \leftarrow p_i$; $\mathit{top}(D_i^v) \leftarrow s_i$; $\mathit{bottom}(D_i^v) \leftarrow \mathit{bottom}(\Delta_0)$;
4. **if** $p_i \notin \mathcal{T}(S_{i-1})$
5. **then** $\mathit{leftp}(L_i) \leftarrow \mathit{leftp}(\Delta_0)$; $\mathit{top}(L_i) \leftarrow \mathit{top}(\Delta_0)$;
6. $\mathit{bottom}(L_i) \leftarrow \mathit{bottom}(\Delta_0)$; $\mathit{rightp}(L_i) \leftarrow p_i$;
7. **for** $j \leftarrow 0$ **to** $k-1$
8. **do if** $\mathit{rightp}(\Delta_j)$ leží nad s_i
9. **then** $\mathit{rightp}(H_i^u) \leftarrow \mathit{rightp}(\Delta_j)$; $u \leftarrow u+1$;
10. $\mathit{leftp}(H_i^u) \leftarrow \mathit{leftp}(\Delta_{j+1})$; $\mathit{top}(H_i^u) \leftarrow \mathit{top}(\Delta_{j+1})$;
11. $\mathit{bottom}(H_i^u) \leftarrow s_i$;
12. **else** $\mathit{rightp}(D_i^v) \leftarrow \mathit{rightp}(\Delta_j)$; $v \leftarrow v+1$;
13. $\mathit{leftp}(D_i^v) \leftarrow \mathit{leftp}(\Delta_{j+1})$; $\mathit{top}(D_i^v) \leftarrow s_i$;
14. $\mathit{bottom}(D_i^v) \leftarrow \mathit{bottom}(\Delta_{j+1})$;
15. $\mathit{rightp}(H_i^u) \leftarrow q_i$; $\mathit{rightp}(D_i^v) \leftarrow q_i$;
16. **if** $q_i \notin \mathcal{T}(S_{i-1})$
17. **then** $\mathit{leftp}(P_i) \leftarrow q_i$; $\mathit{top}(P_i) \leftarrow \mathit{top}(\Delta_k)$;
18. $\mathit{bottom}(P_i) \leftarrow \mathit{bottom}(\Delta_k)$; $\mathit{rightp}(P_i) \leftarrow \mathit{rightp}(\Delta_k)$;

2.6 Aktualizace $\mathcal{D}$

Současně s lichoběžníkovou mapou je potřeba změnit i jí odpovídající datovou strukturu $\mathcal{D}(S_{i-1})$. Vraťme se k lichoběžníkové mapě na obrázku 11. List Δ_0 zde musíme nahradit malým stromem se dvěma x-ovými uzly pro p_i a q_i, jedním y-ovým uzlem pro s_i a čtyřmi listy pro lichoběžníky, viz obrázek 13. Zde bychom čtenáře rádi upozornili na jedno pracovní označení. Budeme zde používat sousloví *nahradit stromem*, což není úplně přesné. Ve skutečnosti vždy přidáme všechny vnitřní uzly daného stromu, ale lichoběžníky přidáme pouze tehdy, pokud se v $\mathcal{D}$ ještě nenacházejí. Jestliže už některý z lichoběžníků v $\mathcal{D}$ leží, povedeme do něj pouze hranu, ale nevytvoříme jej znovu.

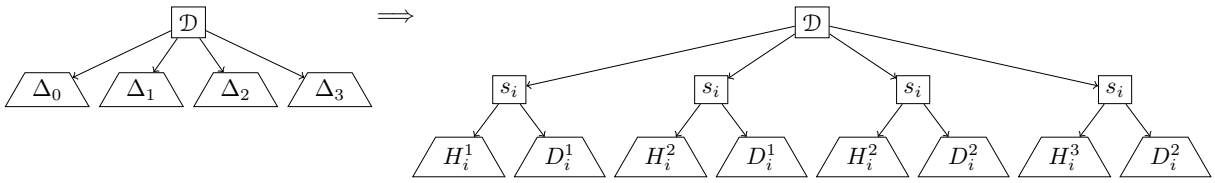
Naproti tomu v situaci znázorněné na obrázku 12 nemusíme vůbec přidávat x-ové uzly, protože se již oba v $\mathcal{D}(S_{i-1})$ nacházejí. Stačí tedy každý protínaný lichoběžník nahradit jedním y-ovým uzlem pro s_i a dvěma listy pro lichoběžníky. Abychom zjistili, které dva lichoběžníky leží pro dané Δ_j nad a pod s_i, musíme pro $j = 1, \ldots, k-1$ určovat, jestli $\mathit{rightp}(\Delta_j)$ leží nad nebo pod s_i. Pracujeme vždy s H_i^u a D_i^v, proto musíme vždy po nahrazení lichoběžníka stromem zvýšit jeden z indexů u a v. Pokud se $\mathit{rightp}(\Delta_j)$ nachází nad s_i, znamená to, že z $\mathit{rightp}(\Delta_j)$ vede svislé prodloužení, které „ukončuje“ lichoběžník H_i^u a zároveň „otevírá“ H_i^{u+1}, proto zvýšíme u o jedna, v opačném případě postupujeme



Obrázek 13: Aktualizace vyhledávací struktury po změně mapy, viz obrázek 11

analogicky. První a poslední protínané lichoběžníky opět řešíme zvlášť, podle toho, jestli p_i a q_i jsou již v $\mathcal{T}(S_{i-1})$.

Změnu datové struktury přidáním úsečky s_i z obrázku 12, znázorňuje obrázek 14. Ačkoliv jsou však listy $\Delta_0, \Delta_1, \Delta_2, \Delta_3$ na obrázku nakresleny na stejné úrovni, v grafu $\mathcal{D}(S_{i-1})$ mohou být hierarchicky různě vysoko. Při aktualizaci $\mathcal{D}$ tedy může docházet ke změnám v celém grafu.



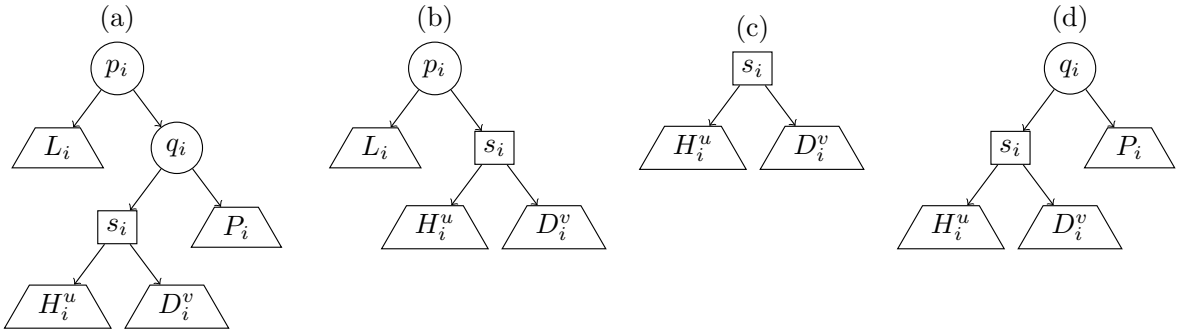
Obrázek 14: Aktualizace vyhledávací struktury po změně mapy, viz obrázek 12

V průběhu aktualizace $\mathcal{D}(S_{i-1})$ se ptáme na stejné otázky a používáme stejné čítače jako v algoritmu AKTUALIZUJ MAPU, proto se nabízí spojení obou dílčích algoritmů do jednoho, pro lepší přehlednost však uvádíme oba pseudokódy zvlášť.

Algoritmus AKTUALIZUJ GRAF($\mathcal{D}(S_{i-1})$, s_i, $\Delta_0, \ldots, \Delta_k$)
Vstup: Datová struktura $\mathcal{D}(S_{i-1})$, přidávaná úsečka s_i a množina lichoběžníků $\Delta_0, \Delta_1, \ldots, \Delta_k$ protínaných úsečkou s_i
Výstup: Aktualizovaná datová struktura $\mathcal{D}(S_i)$

1. $u \leftarrow 1$; $v \leftarrow 1$;
2. **if** $p_i \notin \mathcal{T}(S_{i-1})$
3. **then if** $q_i \notin \mathcal{T}(S_{i-1})$ **and** $k = 0$
4. **then** Nahraď list Δ_0 stromem (a), viz obrázek 15.
5. **else** Nahraď list Δ_0 stromem (b).
6. **else if** $q_i \notin \mathcal{T}(S_{i-1})$ **and** $k = 0$
7. **then** Nahraď list Δ_0 stromem (d).
8. **else** Nahraď list Δ_0 stromem (c).

9. **for** $j \leftarrow 0$ **to** $k-2$
10. **do if** *rightp*(Δ_j) leží nad s_i
11. **then** $u \leftarrow u+1$;
12. **else** $v \leftarrow v+1$;
13. Nahraď list Δ_{j+1} stromem (c).
14. **if** $k \neq 0$
15. **then if** *rightp*(Δ_{k-1}) leží nad s_i
16. **then** $u \leftarrow u+1$;
17. **else** $v \leftarrow v+1$;
18. **if** $q_i \notin \mathcal{T}(S_{i-1})$
19. **then** Nahraď list Δ_k stromem (d).
20. **else** Nahraď list Δ_k stromem (c).



Obrázek 15: Stromy, jimiž nahrazujeme změněné lichoběžníky

Probrali jsme tedy podrobněji jednotlivé kroky algoritmu MAPA, který pro zadanou množinu úseček S zkonstruuje jednak lichoběžníkovou mapu $\mathcal{T}(S)$, jednak datovou strukturu $\mathcal{D}$ potřebnou pro vyhledávání v ní.

Kapitola 3

Speciální případy

V předchozím textu jsme učinili zjednodušující předpoklad. Řekli jsme si, že budeme uvažovat pouze množiny S, kde každé dva různé body mají různé x-ové souřadnice. Nyní odbouráme nutnost tohoto předpokladu a zaručíme funkčnost algoritmu pro obecně libovolnou množinu nekřížících se úseček S.

3.1 Různé x-ové souřadnice

Uvědomme si, že pro danou množinu S jsme volili vložení do systému souřadnic zcela libovolně. Lze proto uvažovat o otočení souřadných os o dostatečně malý úhel tak, aby se případné rovnosti souřadnic změnily v ostré nerovnosti. Bohužel takové otočení může vést ke komplikacím. Jednak pokud máme na začátku za souřadnice čísla celá, případně racionální s malým počtem desetinných míst, otočením výhodu snadnějšího a rychlejšího počítání ztratíme, jednak se mohou objevit zaokrouhlovací chyby, tedy dva body s různými x-ovými souřadnicemi, se mohou otočením v x-ových souřadnicích přiblížit natolik, že jejich zaokrouhlení si již budou rovna.

Výhodnější je proto provádět odlišnou transformaci, a to jen symbolickou. Jelikož řešíme pouze rovnost x-ových souřadnic, stačí použít afinní zobrazení podél x-ové osy. Říkejme tomuto zobrazení *šikmá transformace* (anglicky *shear transformation*) a označme jej φ. Pro malé $\varepsilon > 0$ potom

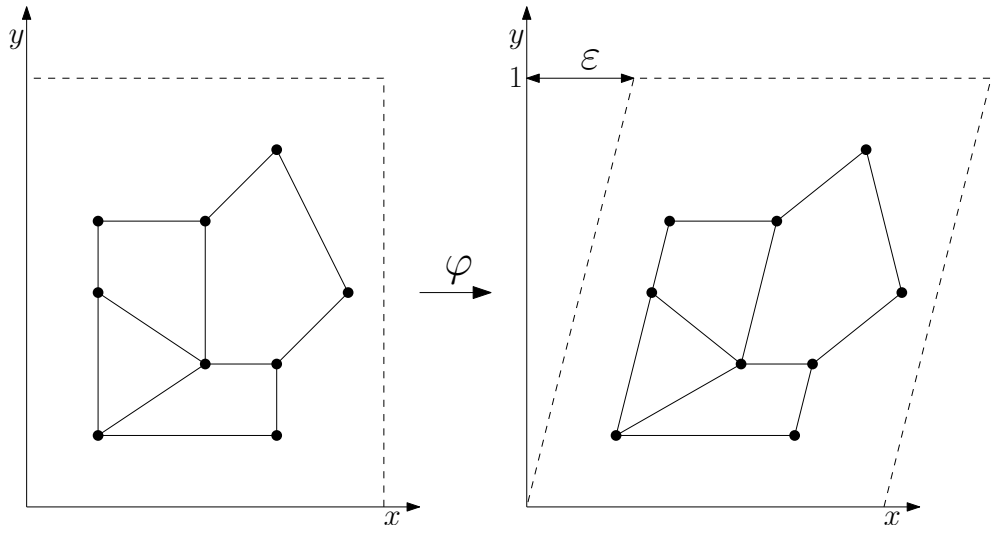
$$\varphi : [x, y] \mapsto [x + \varepsilon y, y] \,.$$

Z nezápornosti ε plyne, že různé koncové body úseček z $\varphi S := \{\varphi(s) \,|\, s \in S\}$ budou mít různé x-ové souřadnice. Navíc pro dostatečně malé ε bude pro libovolné koncové body $p = [p_x, p_y]$ a $r = [r_x, r_y]$ z S platit: $p_x < r_x \implies \varphi(p)_x < \varphi(r)_x$. Požadované ε lze zvolit takto:

$$0 < \varepsilon < \min\left\{1, \frac{x_j - x_i}{y_i - y_j} \;\middle|\; 1 \le i \le 2n,\ 1 \le j \le 2n,\ x_i < x_j,\ y_j < y_i\right\},$$

kde $[x_i, y_i]$ probíhá množinu koncových bodů úseček z S. Vidíme, že platí:

$$\varphi(p)_x < \varphi(r)_x \iff (p_x < r_x) \vee (p_x = r_x \wedge p_y < r_y) \,.$$



Obrázek 16: Šikmá transformace φ

Tedy uspořádání podle x-ové souřadnice po transformaci φ je stejné jako lexikografické uspořádání před transformací. Proto není potřeba počítat hodnotu ε, stačí uvažovat lexikografické uspořádání bodů z S.

Zbývá se pouze přesvědčit, že ani ve druhém kroku algoritmu SLEDUJ ÚSEČKU (viz popis na straně 17), kde porovnáváme směrnice dvou úseček, konkrétní ε znát nepotřebujeme. Mějme body $p = [p_x, p_y]$, $q = [q_x, q_y]$ a $r = [r_x, r_y]$ takové, že úsečky pq a pr nemají společný vnitřní bod a platí $\varphi(p)_x < \varphi(q)_x$ a $\varphi(p)_x < \varphi(r)_x$. Potom skutečnost, že směrnice $\varphi(pq) <$ směrnice $\varphi(pr)$ znamená:

1. pro $p_x < q_x$ a $p_x < r_x$:

$$\frac{q_y - p_y}{q_x - p_x + \varepsilon(q_y - p_y)} < \frac{r_y - p_y}{r_x - p_x + \varepsilon(r_y - p_y)},$$

   což je pro dostatečně malé ε ekvivalentní s:

$$\frac{q_y - p_y}{q_x - p_x} < \frac{r_y - p_y}{r_x - p_x},$$

   tedy směrnice$(pq) <$ směrnice(pr)

2. pro $p_x < q_x$, $p_x = r_x$ a $p_y < r_y$:

$$\frac{q_y - p_y}{q_x - p_x + \varepsilon(q_y - p_y)} < \frac{r_y - p_y}{\varepsilon(r_y - p_y)} = \frac{1}{\varepsilon},$$

   tedy jistě směrnice$(pq) < \infty$.

Proto pro vhodné malé ε je porovnání směrnic úseček $\varphi(pq)$ a $\varphi(pr)$ stejné jako porovnání směrnic úseček pq a pr. Přitom pokud $p_x = r_x$ a $p_y < r_y$, pak pokládáme

směrnice$(pr) = \infty$. V případě, kdy směrnice $\varphi(pq) >$ směrnice $\varphi(pr)$ postupujeme naprosto analogicky.

Zjistili jsme, že jediná změna v běhu algoritmu pro φS spočívá v lexikografickém porovnání bodů, když určujeme jejich pořadí podle x-ových souřadnic. Algoritmus sice konstruuje lichoběžníkovou mapu pro φS a datovou strukturu příslušnou $\mathcal{T}(\varphi S)$, ale samotnou hodnotu ε jsme nikde nepotřebovali znát, proto ani není nutné ε na začátku počítat. Stačí nám jen, že takové ε splňující námi požadované podmínky, existuje.

3.2 Obecná poloha bodu q

Případy, kdy hledaný bod q leží na některém svislém prodloužení nebo na úsečce z S, jsme již rozebrali v sekci 1.6., podívejme se však, co se změní použitím transformace φ.

Různé body jsou vždy transfomovány na body s různými x-ovými souřadnicemi. Bod $\varphi(q)$ proto bude ležet na transformovaném svislém prodloužení bodu $\varphi(p)$ právě tehdy, když platí $q = p$, což jsme již vyřešili. Vztah bodu a úsečky se zobrazením φ také nezmění, proto pro všechny úsečky $s \in S$ platí $q \in s \iff \varphi(q) \in \varphi(s)$.

Všechny postupy, které jsme uváděli dříve lze tedy použít i na transformované úsečky $\varphi(s_i)$ a bod $\varphi(q)$. Navíc zobrazení φ má pouze symbolický charakter, a proto nemusíme provádět další výpočty pro jeho realizaci.

Kapitola 4

Odhady složitosti

V následující kapitole dokážeme, že algoritmus byl konstruován chytře, tzn. je poměrně efektivní. Učiníme proto jisté odhady na časovou složitost konstrukce lichoběžníkové mapy a vyhledávací struktury. Zjistíme, jakou přibližnou velikost bude mít vyhledávací struktura $\mathcal{D}$, a odhadneme časovou náročnost i samotného vyhledávání zadaného bodu q. Budeme se tedy snažit odůvodnit, že naše volba náhodnostního algoritmu opravdu má deklarované výhody.

Jak již bylo řečeno dříve náhodnost celého algoritmu MAPA spočívá v jeho druhém kroku, kde generujeme náhodné pořadí úseček. Podíváme-li se na obrázek 7, dokážeme něco říci o pořadí, podle něhož bylo $\mathcal{D}$ konstruováno. Předně první úsečka v tomto pořadí musela být s_4, neboť na začátku vždy vkládáme úsečku dovnitř do obdélníka $\mathcal{R}$, čili prvním přidávaným bodem je levý koncový bod první úsečky. Druhou úsečkou byla jistě s_1. Nejsme ovšem schopni určit, pořadí na třetím a čtvrtém místě, kde se nachází dvojice s_2 a s_3. Vidíme tedy, že různá uspořádání mohou vést na stejnou datovou strukturu. Stejně dobře ovšem pouhá záměna pořadí dvou úseček může zcela změnit $\mathcal{D}$ a tudíž potenciálně i délku cesty, kterou musíme v grafu projít, abychom se dostali do lichoběžníka, v němž bod q leží.

4.1 Výpočetní složitost algoritmu MAPA

Obecně pracujeme s $n!$ různými uspořádáními a také $n!$ různými strukturami $\mathcal{D}$. Přitom nás zajímají uspořádání vedoucí na vyhledávací strukturu, která má malou složitost a krátký vyhledávací čas. Uspořádání ovšem volíme náhodně, proto musíme počítat i s uspořádáními, která nebudou vyhovovat těmto podmínkám. V této souvislosti se používá pojmů *očekávaná složitost* nebo *očekávaný vyhledávací čas*, což znamená průměrnou složitost nebo čas, kde se průměr počítá přes všech $n!$ možných permutací prvků z S.

V praxi tedy mohou nastat případy, kdy vyhledávací struktura nebude pro dané pořadí úseček dobrá, ale jejich podíl na celkovém počtu případů je vždy dostatečně malý. Ukažme si nejprve, že očekávaný vyhledávací čas je $\mathcal{O}(\log n)$.

Věta 2. *Mějme datovou strukturu $\mathcal{D}$ zkonstruovanou algoritmem MAPA. Potom pro libovolný bod q bude očekávaný vyhledávací čas $\mathcal{O}(\log n)$.*

Důkaz. Jelikož strukturu $\mathcal{D}$ již máme hotovou, rychlost vyhledávání záleží pouze na tom, jak dlouhou cestu musíme v $\mathcal{D}$ projít. Z algoritmu AKTUALIZUJ MAPU na straně 21, respektive z obrázku 15, je dobře patrné, že cesta se může v každém kroku algoritmu prodloužit nejvýše o tři uzly. Odtud pro libovolný zadaný bod q platí, že nejdelší možná cesta do tohoto bodu má délku $3n$. Krokem algoritmu zde rozumíme provedení všech příkazů pro dané i v cyklu FOR ve třetím kroku algoritmu MAPA.

Chceme-li ovšem zjistit průměrnou délku cesty přes všech $n!$ možností, využijeme pravděpodobnost. Mějme daný bod q a k němu cestu v $\mathcal{D}$. Zřejmě každý uzel na této cestě byl vytvořen v některém z n kroků algoritmu. Nechť pro $i = 1, 2, \ldots, n$ značí X_i náhodnou veličinu, která udává počet uzlů na cestě k bodu q, jež byly vytvořeny v i-tém kroku algoritmu. Jelikož X_i je opravdu náhodná veličina, neboť závisí pouze na pořadí úseček, platí pro střední hodnoty následující:

$$\mathrm{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathrm{E}[X_i].$$

Víme, že $0 \leq X_i \leq 3$. Označme P_i pravděpodobnost, že v i-tém kroku byl vytvořen uzel na cestě k bodu q. Potom pro $i = 1, 2, \ldots, n$:

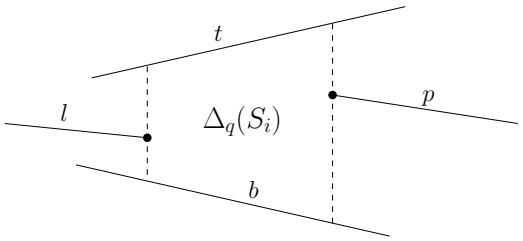
$$\mathrm{E}[X_i] \leq 3P_i.$$

Uvědomme si nyní, za jakých podmínek v i-tém kroku vznikne nový uzel na cestě ke q. Stane se tak právě tehdy, když se lichoběžník obsahující bod q liší od lichoběžníku, který q obsahoval v kroku předchozím. Jinými slovy nový uzel vznikne, pokud přidávaná i-tá úsečka protne lichoběžník, v němž q dosud ležel. Označme lichoběžník obsahující bod q v i-tém kroku symbolem $\Delta_q(S_i)$. Uvedené pravděpodobnosti pak splňují

$$P_i = \Pr[\Delta_q(S_i) \neq \Delta_q(S_{i-1})] = \Pr[\Delta_q(S_i) \notin \mathcal{T}(S_{i-1})].$$

Platí-li nerovnost mezi lichoběžníky, pak byl $\Delta_q(S_i)$ vytvořen při přidávání úsečky s_i. Vzniklé lichoběžníky Δ budou mít na své hranici buď část úsečky s_i nebo alespoň její koncový bod. Tedy *top*(Δ) nebo *bottom*(Δ) se bude rovnat s_i, případně *leftp*(Δ) či *rightp*(Δ) bude jeden z koncových bodů s_i.

Nechť S_i je libovolná podmnožina S. Pravděpodobnost, že se minimálně jeden z údajů *top*$(\Delta_q(S_i))$, *bottom*$(\Delta_q(S_i))$, *leftp*$(\Delta_q(S_i))$ a *rightp*$(\Delta_q(S_i))$ bude lišit od příslušného údaje pro $\Delta_q(S_{i-1})$ odpovídá pravděpodobnosti, že se v i-tém kroku prodlouží cesta k bodu q. Lichoběžník $\Delta_q(S_i)$ zůstane pro libovolnou permutaci úseček v S_i vždy stále stejný, může však být určen až čtyřmi různými úsečkami, přičemž extrémní případ zobrazuje obrázek 17. Pořadí úseček v S_i je náhodné, proto se všechny úsečky mohou vyskytnout na i-tém místě se stejnou pravděpodobností. Úsečka l z obrázku 17 bude tedy poslední v pořadí s pravděpodobností $1/i$, stejnou pravděpodobnost mají i úsečky t, p a b. Tyto



Obrázek 17: Lichoběžník určený čtyřmi úsečkami

pravděpodobnosti sice mohou být závislé, neboť na určení některých lichoběžníků stačí pouze dvě úsečky, ale my potřebujeme horní odhad. Je-li na i-tém místě jedna z úseček určujících $\Delta_q(S_i)$, dojde k prodloužení cesty k bodu q. Dohromady proto dostáváme:

$$P_i \leq 4/i.$$

Poskládáme-li všechny uvedené rovnosti a nerovnosti, obdržíme horní odhad očekávaného vyhledávacího času:

$$\mathrm{E}\left[\sum_{i=1}^{n} X_i\right] \leq \sum_{i=1}^{n} 3P_i \leq \sum_{i=1}^{n} \frac{12}{i} = 12\sum_{i=1}^{n} \frac{1}{i} = 12H_n.$$

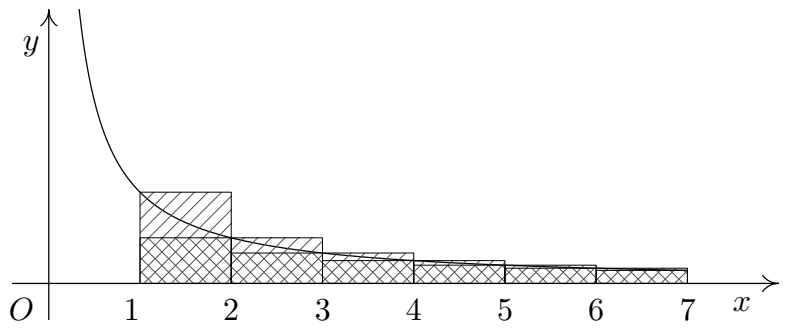
V tomto případě H_n znamená n-té *harmonické číslo*:

$$H_n := \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}.$$

Pokud máme $n > 1$, pak pro harmonická čísla H_n také platí následující nerovnost:

$$\ln n < H_n < \ln n + 1. \tag{4.1}$$

Z druhé části nerovnosti 4.1 vidíme, že očekávaný vyhledávací čas je skutečně $\mathcal{O}(\log n)$, zatímco první nerovnost říká, že tento odhad není zbytečně nadhodnocený. Nerovnost plyne ze vztahu mezi $\int_1^n \frac{1}{x}\mathrm{d}x = \ln x$ a $\sum_{i=1}^{n} \frac{1}{x} = H_n$. Platí totiž $H_n - 1 < \int_1^n \frac{1}{x}\mathrm{d}x < H_{n-1}$, kde $H_n - 1$ a H_{n-1} jsou po řadě dolní a horní integrální součet, viz obrázek 18. □



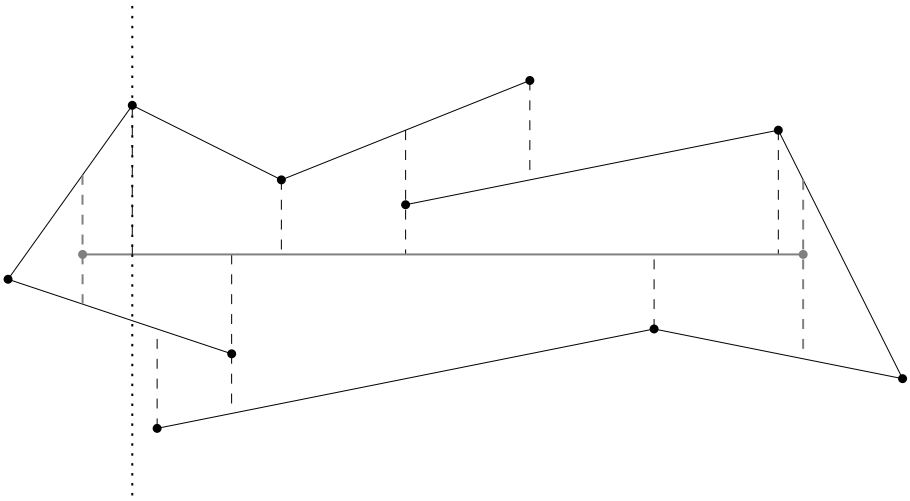
Obrázek 18: Integrální součty funkce $1/x$ na intervalu od 1 do 7

Věta 3. *Očekávaná velikost datové struktury $\mathcal{D}$ je $\mathcal{O}(n)$.*

Důkaz. Podobně jako v předchozím důkazu, závisí odhad na nějaké veličině. Zde se jedná o počet uzlů, který stačí odhadnout, abychom zjistili přibližnou velikost $\mathcal{D}$.

Jak jsme již uvedli dříve $\mathcal{D}$ obsahuje dva typy uzlů: listy, které přímo korespondují s lichoběžníky v $\mathcal{T}(S)$ (tj. každý lichoběžník se v $\mathcal{D}$ vyskytuje právě jednou), a vnitřní uzly, jimž odpovídají úsečky z S a jejich koncové body. Počet listů je dle Lemmatu 1 nejvýše $3n + 1$, tedy $\mathcal{O}(n)$, zbývá odhadnout kolik vnitřních uzlů bude v $\mathcal{D}$ obsaženo.

Nechť u_i označuje počet lichoběžníků vytvořených v i-té iteraci algoritmu, respektive počet nově vzniklých listů v $\mathcal{D}$. Potom nových vnitřních uzlů bude v i-té iteraci právě $u_i - 1$. Dokažme to indukcí vůči počtu lichoběžníků, které jsou přidáním úsečky s_i protnuty a k jejichž nahrazení v i-té iteraci dochází. Je-li protnut jeden lichoběžník, pak na obrázku 15 (viz strana 22) vidíme, že ve všech případech je počet vnitřních uzlů právě o jedna menší než počet listů v $\mathcal{D}$, tedy tvrzení platí. Předpokládejme, že jsme již uvedené dokázali pro n protínaných lichoběžníků a mějme nyní úsečku protínající $n + 1$ lichoběžníků. Veďme pomocnou svislou přímku některým z vrcholů protínaných lichoběžníků tak, abychom protnuli úsečku s_i a rozdělili lichoběžníky na dvě skupiny (viz obrázek 19).



Obrázek 19: Rozdělení protínaných lichoběžníků na dvě skupiny

V levé skupině mějme nyní u lichoběžníků a v pravé v lichoběžníků. Všimněme si, že jeden lichoběžník počítáme dvakrát, neboť pomocná přímka jej rozděluje. (Rozmyslete si, že takový lichoběžník vždy existuje a je právě jeden.) Jelikož $u < n$ a $v < n$, tak podle předpokladu vznikne $u - 1$ a $v - 1$ nových vnitřních uzlů, celkem tedy $(u - 1) + (v - 1) = u + v - 2$. Když si uvědomíme dvakrát počítaný lichoběžník, tak v i-té iteraci vznikne $u + v - 1$ lichoběžníků a nových vnitřních uzlů tudíž máme skutečně o jedna méně. Tímto je vztah mezi počtem nových vnitřních uzlů a nových listů v $\mathcal{D}$ dokázán.

Očekávaná velikost $\mathcal{D}$ se proto, s využitím nezávislosti hodnot $u_i - 1$, rovná

$$\mathcal{O}(n) + \mathrm{E}\left[\sum_{i=1}^{n}(u_i - 1)\right] = \mathcal{O}(n) + \sum_{i=1}^{n}\mathrm{E}[u_i].$$

Dále již potřebujeme jen shora ohraničit číslo u_i pro $i = 1, 2, \ldots, n$. Zafixujme si tedy znovu libovolnou $S_i \subseteq S$ a definujme novou veličinu λ pro každý lichoběžník $\Delta \in \mathcal{T}(S_i)$ a úsečku $s \in S_i$:

$$\lambda(\Delta, s) := \begin{cases} 1 & \text{jestliže lichoběžník } \Delta \text{ vznikl přidáním úsečky } s \text{ do } \mathcal{T}(S_{i-1}). \\ 0 & \text{jinak.} \end{cases}$$

Už v předchozím důkazu jsme zjistili, že každý lichoběžník je určen nejvýše čtyřmi úsečkami. Pro pevně daný lichoběžník Δ proto existují maximálně čtyři možné úsečky s, pro něž by se $\lambda(\Delta, s) = 1$. Počet nově vzniklých lichoběžníků v $\mathcal{T}(S_i)$ zároveň bude nejvýše počet všech lichoběžníků v $\mathcal{T}(S_i)$, který je z Lemmatu 1 $\mathcal{O}(i)$. Dostáváme tedy nerovnost:

$$\sum_{s \in S_i} \sum_{\Delta \in \mathcal{T}(S_i)} \lambda(\Delta, s) \leq 4|\mathcal{T}(S_i)| = \mathcal{O}(i),$$

tzn. sečteme-li počet všech lichoběžníků, které mohly vzniknout v i-tém kroku, bude jich nejvýše $\mathcal{O}(i)$.

Počet lichoběžníků vzniklých přidáním úsečky $s_i \in S_i$ je u_i. Protože pořadí úseček v S_i generujeme náhodně, očekávanou hodnotu u_i určíme pomocí průměru přes všechny $s \in S_i$:

$$\mathrm{E}[u_i] = \frac{1}{i} \sum_{s \in S_i} \sum_{\Delta \in \mathcal{T}(S_i)} \lambda(\Delta, s) \leq \frac{\mathcal{O}(i)}{i} = \mathcal{O}(1).$$

Odvodili jsme, že v každém kroku algoritmu vznikne očekávaný počet $\mathcal{O}(1)$ nových lichoběžníků, tedy i $\mathcal{O}(1)$ nových vnitřních uzlů v $\mathcal{D}$. Spolu s již určeným počtem listů v $\mathcal{D}$ máme proto očekávanou velikost $\mathcal{O}(n)$, neboť

$$\mathcal{O}(n) + \sum_{i=1}^{n} \mathrm{E}[u_i] \leq \mathcal{O}(n) + \sum_{i=1}^{n} \mathcal{O}(1) = \mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n). \qquad \square$$

Věta 4. *Nechť S je množina n úseček v obecné poloze. Pak algoritmus MAPA vytvoří lichoběžníkovou mapu $\mathcal{T}(S)$ a její příslušnou datovou strukturu $\mathcal{D}$ v očekávaném čase $\mathcal{O}(n \log n)$.*

Důkaz. Díky dokázaným větám, máme nyní situaci již poměrně snadnou. První dva kroky algoritmu MAPA mají složitost pouze $\mathcal{O}(n)$, neboť výpočet minim či maxim stejně jako změnu pořadí lze reprezentovat jako nezávislé FOR cykly délky n. Třetí krok znázorňuje suma pro $i = 1, \ldots, n$. Ve čtvrtém kroku je nejnáročnější lokalizace levého koncového bodu úsečky s_i v $\mathcal{T}(S_{i-1})$, která z Věty 2 proběhne v očekávaném čase $\mathcal{O}(\log i)$. Pro odhadnutí složitosti pátého a šestého kroku využijeme očekávaný počet nových lichoběžníků a vnitřních uzlů $\mathrm{E}[u_i]$. Z toho plyne celková složitost algoritmu MAPA:

$$\mathcal{O}(n) + \sum_{i=1}^{n} \big\{ \mathcal{O}(\log i) + \mathcal{O}(\mathrm{E}[u_i]) \big\} = \sum_{i=1}^{n} \mathcal{O}(\log i) \leq \mathcal{O}(n \log n). \qquad \square$$

Všimněme si, že očekávání v uvedených větách znamenalo průměrování přes všechna možná uspořádání úseček, nezáleželo ovšem vůbec na konkrétních úsečkách nebo na hledaném bodě q. Celý algoritmus tedy provede lokalizaci libovolného daného bodu v jakémkoliv podrozdělení $\mathcal{S}$ v očekávaném čase $\mathcal{O}(n \log n)$.

Literatura

[1] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Third Edition, 2008.

[2] M. Čadek, *Přednáška Geometrické algoritmy*, PřF MU, podzim 2008.

[3] K. Mulmuley, *A fast planar partition algorithm, I.*, Journal of Symbolic Computation, 10:253-280, 1990.

[4] R. Seidel, *A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons*, Comput. Geom. Theory Appl., 1:51-64, 1991.

[5] H. S. Wilf, *Algorithms and Complexity*, 1994, `http://www.math.upenn.edu/~wilf/AlgoComp.pdf`, květen 2009.