Korespondenční Seminář z Programování

29. ročník **KSP** **Únor 2017**

Milí řešitelé a řešitelky!

„Už sníh se ztrácí ze strání a zem začíná žít, jenom blejsklo slunce do louží, vtom parťák na nás vlít. . .“ zpívá se v jedné písničce. Před naším Matfyzem už po sněhu nezbyly ani ty louže, tak zase hurá do práce. Přinášíme vám nové úlohy i další díl seriálu o stromech a těšíme se na vaše řešení :-)

Zejména pro začínající řešitele chystáme i letos **jarní soustředění**, tentokrát od 29. 4. do 6. 5. ve Štědrákově Lhotě (kousek od Šumperka). Budeme rádi, pokud o soustředění dáte vědět svým méně zkušeným kamarádům, kteří by do tajů programování a algoritmizace chtěli proniknout hlouběji.

Za řešení KSP je možné být přijat na MFF UK bez přijímacích zkoušek. Na získání osvědčení úspěšného řešitele je letos třeba získat v hlavní kategorii alespoň 150 bodů. Osvědčení je třeba dodat do 30. 4., maturanti tak mají poslední šanci si ho vysloužit včas. Kvůli ubývajícímu času jim ale nabízíme přednostní opravení a dodání osvědčení interní cestou. Ozvěte se nám, pokud se vás prominutí přijímaček za letošní KSP týká.

Termín série: **Pondělí 3. dubna 2017 v 8:00**

Odevzdávání: Přes web na adrese *https://ksp.mff.cuni.cz/submit/*

Odměna série: **Sladkou odměnu** pošleme každému, kdo získá alespoň **polovinu bodů z pěti odevzdaných úloh**.

Čtvrtá série dvacátého devátého ročníku KSP

Chodbou se rozléhal hovor, smích i šustění papírů, jak se někteří studenti ještě snažili honem rychle něco doučit. Vilém pobaveně sledoval svého kamaráda, který rozebíral, jaké všechny otázky by v testu nechtěl potkat.

„A hlavně doufám, že se nebudou ptát na Novákovy spisky o štěstí a náhodě,“ máchl Bernard rukama.

„Tak u nich snad stačí vědět, že je postupně někdo krade z Archivu, ne?“

„Další!“ přerušil kamarády mocný hlas. Oba chlapci se usmáli a Vilém se vydal na klasickou kontrolu před testem.

Kontrolující muž mu položil několik obvyklých otázek a začal kontrolovat Vilémovo oblečení. „Kapesníčky?“ ujišťoval se, když pohmatem našel předmět v kapse kalhot. Vilémovi se rozbušilo srdce, zatajil dech a jen němě přikývl. Snad proto ho muž chvíli nejistě pozoroval, pak rázně sáhl do kapsy. . . a vytáhl malou měkkou podkovu. Chvíli se tvářil zmateně, pak mu zazářily oči, když si všiml několika čtyřlístků umně maskovaných ve vzoru Vilémovy košile.

„Čtyřlístky na zkoušku a podkova na kontrolu? Dobrej nápad, chlape,“ poplácal Viléma po zádech a čtyřlístky jeden po druhém zabavil. „Máte štěstí, že studujete zrovna na Institutu náhodných a nápadně nenáhodných jevů,“ dodal ještě a s úsměvem ho poslal do zkouškové místnosti.

Vilém rozhodně na čtyřlístky nespoléhal, a tak navzdory jejich zabavení za hodinu a půl odevzdával hustě popsanou písemku. Dva zkoušející na sebe mrkli. „Oni mají body ze semestru, že? Můžeme při odevzdání přijímat jen písemky od lidí, kteří mají víc bodů než jejich předchůdce?“

29-4-1 Odevzdávání písemek — 10 bodů

Studenti utvořili frontu na odevzdání písemek, každý ji odevzdá jednomu ze dvou zkoušejících. Každý student má také nějaké body ze semestru. Aby ale mohl student písemku odevzdat, musí mít více bodů než poslední student, který odevzdal písemku stejnému opravujícímu; předbíhat se nesmí. Komu má kdo odevzdávat, aby mohli odevzdat všichni?

Formálněji: je dána posloupnost celých čísel. Vaším úkolem je rozdělit ji na dvě rostoucí podposloupnosti (případně zjistit, že to nejde).

Můžeme si představit, že každý prvek posloupnosti obarvíme jednou ze dvou barev (třeba modrou nebo červenou). A chceme to udělat tak, aby modré prvky tvořily rostoucí podposloupnost – tedy pokud budeme číst zleva doprava jen modré prvky (červené přeskakujeme), budou přečtená čísla v rostoucím pořadí.

To samé musí platit pro červené prvky. Žádný prvek nemůže být obarven dvěma barvami ani zůstat neobarvený.

Formát vstupu: Posloupnost celých čísel.

Formát výstupu: Na prvním řádku modrá podposloupnost (všechny modré prvky čtené zleva doprava v původním pořadí), na druhém červená.

Ukázkový vstup:

```
9 4 14 5 17 8 26
```

Ukázkový výstup:

```
9 14 17 26
4 5 8
```

Ukázkový vstup:

```
8 1 11 12 0 6
```

Ukázkový výstup:

```
NELZE
```

Nezapomeňte důkladně zdůvodnit, proč vaše řešení funguje.

Kdo ví, nakolik zkoušející jen provokovali, rozhodně se všem studentům povedlo písemku odevzdat. Vilém si ještě zašel s Bernardem na chvíli sednout ven a pak už hurá domů.

Následující ráno ho při pohledu do zrcadla přivítala černá plocha. „Zase? Tyhle krámy nic nevydrží. . . “ povzdechl si a vyměnil baterky. Ale pořád je to asi lepší než mít doma skleněné zrcadlo. Příběhů o strašidelných koncích už slyšel víc než dost.

Tenhle trend začal před pár lety, když se do aut místo zpětných zrcátek začaly instalovat zadní kamery. Tím sice neubylo dopravních nehod, ale výrazně se zmírnily jejich následky.

Smartphony po svém příchodu zase rychle vytlačily malá osobní zrcátka, se kterými vás dnes nepustí ani do letadla.

Jakmile dostatečně klesly ceny a spotřeba velkých LCD panelů, začala být i velká domácí zrcadla nahrazována elektronickými. Zpočátku to byly jednoduchoučké přístroje tvořené jen kamerou a displejem. Ale ve světě, kde i rychlovarné konvice mají Wi-Fi, nemohla zrcadla zůstat dlouho pozadu.

Dnes jsou z nich velmi chytrá (někteří stále trvají na tom, že příliš chytrá) a flexibilní zařízení. Nejenže si na nich ráno přečtete noviny či prohlédnete kalendář, ale například mezi náctiletými děvčaty je velmi oblíbená aplikace Zrcadlo, zrcadlo. . .

Jako každému složitějšímu zařízení, ani zrcadlům se nevyhnuly počítačové viry a útoky. Nejčastěji v podobě šmírování kamerou, ale rozmohly se i různé vtípky od nevinných (obraz vzhůru nohama) po celkem necitlivé (vidíte za zády strašidelný stín či svou podobiznu digitálně zestárlou o deset let). Ale to je malá daň za bezpečnost. V posledních měsících se dokonce mluví o plošném zákazu skleněných zrcadel v EU. . .

Dost filozofování o zrcadlech, za chvíli začíná ústní zkouška! Vilém se rychle sbalil a vyrazil.

* * *

Čekání na zkoušku si krátil prohlížením různých hracích automatů, které byly rozmístěny po chodbě. Žádné výhry nevydávaly, ale studenti si na nich mohli vyzkoušet své štěstí a různé způsoby, jak ho ovlivnit.

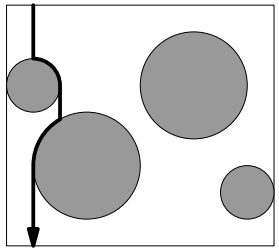
Jeden z automatů ho zvláště zaujal. Vypadal pro účely výzkumu štěstí až podezřele deterministicky. . .

29-4-2 Hrací automat — 12 bodů

Na zdi visí hrací automat. Je tvořen úzkým prostorem mezi dvěma skly, do kterého je možné z libovolného místa podél horní hrany vhodit kuličku. Ta pak padá dolů a cestou naráží na kruhové překážky, po kterých se vždy skutálí a dál padá opět kolmo k zemi. Nakonec dopadne na dolní hranu automatu, podél které je stupnice udávající skóre.

Celou situaci si můžeme představit dvojrozměrně. Plocha automatu je obdélník, po kterém jsou nepravidelně rozmístěné překážky ve tvaru kruhu. Překážky se nikdy nepřekrývají. Vhozená kulička je ve srovnání s těmito kruhy velmi malá, takže si ji můžeme představit jako bod. Volným prostorem padá vždy kolmo dolů. Pokud dopadne na kruhovou překážku, skutálí se po jejím horním okraji tak, jak byste čekali – doleva, pokud dopadla nalevo od středu překážky, doprava, pokud napravo.

Pro jednoduchost předpokládejte, že když kulička dopadne přesně na střed překážky, padá vždy vpravo.



Na vstupu dostanete nejdřív popis automatu: výšku H, šířku W a počet překážek N. Následují pro každou překážku tři čísla udávající střed a poloměr. Poté následuje Q dotazů. Každý dotaz je popsán jedním reálným číslem udávajícím x-ovou souřadnici místa, kde na horním okraji automatu vhodíme kuličku. Pro každý dotaz chceme určit x-ovou souřadnici místa na dolním okraji automatu, kam kulička dopadne. Předpokládejte, že dotazů bude řádově alespoň tolik jako překážek.

Střih. Mezitím na nedaleké policejní stanici. . .

„Hlášení přibývá. Podivné nehody, zmizení. . . Včera jednoho člověka dvakrát zasáhl blesk! Oba zásahy přežil, ale bylo to na železničním přejezdu, a než se stačil probrat, přejel ho vlak. Kolemjdoucí prý tou dobou v okolí zahlédli černou kočku. Ale na takovouhle smůlu černá kočka nestačí. . . “

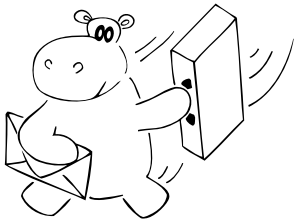
„Může to samozřejmě být nějaký přirozený zdroj smůly, ale kolegové z kriminálky se domnívají, že za tímhle případem, a spoustou podobných, stojí nějaká organizovaná zločinecká banda.“

„Dokonce dvě,“ přihodil nově příchozí policista. „Zachytili nějaké výhružné dopisy, které si posílají mezi sebou. Máme spoustu podezřelých, ale zatím netušíme, kdo kam patří. . . “

29-4-3 Výhružné dopisy — 11 bodů

Dva znepřátelené gangy si vyměňují výhružné dopisy. Oba gangy dohromady čítají N lidí. Každý člověk patří do právě jednoho z gangů, ale nevíme kterého. O každém člověku víme, kolik odeslal a kolik přijal výhružných dopisů.

Rádi bychom zjistili, kdo komu posílal dopisy. To nejspíš nepůjde jednoznačně, ale stačí nám najít jedno libovolné řešení. Jedna dvojice odesílatel-adresát si mohla poslat více dopisů, v takovém případě nás zajímá kolik. Výhružné dopisy se posílají pouze mezi gangy, nikdy uvnitř gangu.



Jinými slovy, hledáme orientovaný bipartitní multigraf (tedy graf, kde mezi jednou dvojicí vrcholů může vést více hran) – vrcholy představují lidi, partity gangy a každá hrana jeden poslaný dopis (orientovaná od odesílatele k příjemci). Pro každý vrchol máme předepsán jeho vstupní a výstupní stupeň (kolik hran v něm má začínat a kolik končit). Rozdělení vrcholů na partity není určeno, to je třeba najít. Parity mohou být různě velké.

Formát vstupu: Na prvním řádku bude přirozené číslo N udávající počet lidí (vrcholů). Dále pro každého člověka řádek se dvěma nezápornými celými čísly udávajícími, kolik dopisů daný člověk přijal a kolik odeslal.

Formát výstupu: Pro každou dvojici odesílatel-adresát, která si poslala alespoň jeden dopis, vypište trojici čísel: pořadová čísla odesílatele a adresáta (0 až $N-1$) a počet dopisů, které odesílatel poslal adresátovi. Pokud existuje více řešení, vypište libovolné. Vstupy budou zadány tak, aby vždy alespoň jedno řešení existovalo.

	řešitel	*škola*	*ročník*	*sérií*	*H3-1*	*H3-2*	*H3-3*	*H3-4*	*H3-5*	*H3-6*	*H3-7*	*série*	*celkem*
25.	Filip Masár	PiarGNitra	3	2								0,0	27,4
26.	Petr Gebauer	GMělník	3	2								0,0	26,8
27.	Michal Kodad	SPŠ_Smíchov	1	6								0,0	26,5
28.	Jiří Löffelmann	GLitoměřPH	3	6	8	10						18,0	25,9
29.	Václav Pavlíček	SPSE_Pard	1	6								0,0	25,5
30.	Ondřej Gonzor	G Brandýs	0	2	2	1						4,8	23,6
31.	Anna Řechtáčková	GJarošeBO	4	2								0,0	22,7
32.	Kristián Jacik	GSRandyJN	4	1								0,0	22,6
33.	Ondřej Krsička	GJarošeBO	1	2								0,0	22,0
34.	Stanislav Lukeš	GPísnickáPH	4	13		10						10,0	21,9
35.	Anna Hollmannová	GSRandyJN	0	3			1	1				2,8	21,5
36.	Daniel Skýpala	GTomkovaOL	−1	2			6					7,1	19,6
37.	Radek Olšák	MensaG	2	1								0,0	18,4
38.	Jindřich Dítě	VOSPŠŽďár	1	2						1		1,6	17,2
39.	Přemysl Šťastný	GŽamberk	4	15	8							8,0	15,6
40.	Ondřej Cach	SPSE_Pard	1	1								0,0	15,4
41.–42.	Vojtěch Hudec	G_ČTřebová	3	3								0,0	12,1
	Josef Polášek	GKepleraPH	1	1	8	1	1					12,1	12,1
43.	Vojtěch Lengál	GZborovPH	3	1								0,0	11,0
44.	Dalibor Kramář	G BO-Řeč	2	1	0				8			8,7	8,7
45.–46.	Adam Dřínek	GNAlejíPH	3	1								0,0	8,0
	Jakub Suchánek	GOpatovPHA	3	3	8							8,0	8,0
47.	Jan Neumann	GNAlejíPH	3	2								0,0	7,7
48.–49.	Jakub Dobrý	GMikuláš PL	3	4								0,0	7,6
	Anna Šebestíková	GČeskáČB	2	2								0,0	7,6
50.	Michael Kozel	GZborovPH	3	1								0,0	7,5
51.	Jan Jeníček	GNAlejíPH	1	1								0,0	7,4
52.	Jakub Jirkal	GJungmanLT	2	1								0,0	7,2
53.	Jakub Spišák	G VBN Prie	4	1								0,0	7,0
54.–55.	Erik Kučák	GHorMichal	4	1								0,0	6,7
	Martin Miller	GVoděraPH	3	1								0,0	6,7
56.	Michal Töpfer	G DrJPekMB	4	11								0,0	6,6
57.	Eliška Vlčinská	GHladnov	2	2								0,0	6,3
58.	Václav Šraier	GČeskoliPH	4	10			6					5,7	5,7
59.	Jan Bíl	GDašickáPA	4	1	2							4,0	4,0
60.	Jonáš Havelka	GJírovcČB	1	2								0,0	2,2

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.

Webové stránky: *https://ksp.mff.cuni.cz/*
E-mail: *ksp@mff.cuni.cz*
Diskusní fórum: *https://ksp.mff.cuni.cz/forum/*

Chcete-li s námi komunikovat bezpečně, můžete si ověřit náš HTTPS certifikát – jeho SHA1 fingerprint je: `E9:DB:EE:C6:62:BC:14:DE:09:E4:E8:97:DC:36:0E:87:B3:50:B0:01`.

Nejprve dokážeme, že součet všech čísel mezi dvěma výskyty téhož vrcholu v v ET-posloupnosti je nulový. Určitě to stačí dokázat pro „sousední“ výskyty, tedy takové, mezi nimiž jsme v nenavštívili. Nechme DFS běžet tak dlouho, než dospěje do prvního z našich dvou výskytů vrcholu v. Pak bude pokračovat do některého ze synů vrcholu v, načež proleze celý podstrom pod tímto synem, a nakonec se vrátí zpět do v, což bude druhý z výskytů. Kdykoliv při tomto průchodu prošel po nějaké hraně dolů, vrátil se po ní pak nahoru, takže k celkovému součtu tato hrana přispěje nulou.

Nyní dokážeme, že součet všech čísel mezi jakýmkoli výskytem vrcholu p a jakýmkoli výskytem vrcholu x je roven součtu cesty mezi x a p. Vzhledem k předchozímu odstavci si můžeme vybrat konkrétní výskyty: pro p si vybereme ten, z nějž odejdeme hranou vedoucí směrem k x; pro x zvolíme první výskyt.

Uvažujme, co DFS provede mezi těmito dvěma výskyty. Určitě prošlo po cestě z p do x. Všechny podstromy odpojující se od této cesty doleva, kompletně prošlo, takže celkem přispěly nulou. Podstromy odpojující se vpravo vůbec nenavštívilo. Nenulou tedy přispěly pouze hrany na cestě.

K odpovídání na daný typ dotazů tedy stačí předpočítat prefixové součty pro ohodnocenou ET-posloupnost, a pamatovat si pro každý vrchol jeho libovolný výskyt. To zvládneme v lineárním čase, na dotazy pak odpovídáme odečtením dvou prefixových součtů, tedy v konstantním čase.

Úkol 6: Syn v zadaném směru

Dostaneme vrchol x a jeho předchůdce p. Chceme najít toho ze synů vrcholu p, který leží na cestě z p do x. Použijeme podobný trik jako pro výpočet LCA. ET-posloupnost ohodnotíme hloubkami a budeme hledat vrchol s minimální hloubkou ležící mezi libovolným výskytem vrcholu x a posledním výskytem vrcholu p.

Z toho přímo nic nezjistíme: minimum se evidentně nabývá pro vrchol p. Ale pokud v zadaném intervalu nalezneme *nejlevější* minimum, je to ten z výskytů vrcholu p, do nějž jsme se z x vrátili. Těsně před ním v posloupnosti leží hledaný syn.

Stačí nám tedy vylepšit strukturu pro intervalová minima, aby vždy našla nejlevější výskyt minima v intervalu. To se dá zařídit třeba tak, že do ET-posloupnosti pro i-tý výskyt vrcholu v místo hloubky $d(v)$ zapíšeme uspořádanou dvojici $(d(v), i)$ a dvojice budeme porovnávat lexikograficky. Nebo můžeme dvojici zakódovat do přirozeného čísla $d(v) \cdot n + i$.

Takto upravená struktura bude stejně rychlá jako ta původní, takže s předvýpočtem v $\mathcal{O}(n \log n)$ dokážeme odpovídat v konstantním čase.

Martin „Medvěd“ Mareš

Výsledková listina třetí série dvacátého devátého ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*H3-1*	*H3-2*	*H3-3*	*H3-4*	*H3-5*	*H3-6*	*H3-7*	*série*	*celkem*
0.					8	10	11	11	9	13	15	60,0	180,0
1.	Lukáš Rozsypal	GÚstavníPH	4	6	8	10	5,5		9		11	47,0	140,2
2.	Richard Hladík	GOAMarLaz	4	23	8							8,0	121,6
3.	Tomáš Domes	MendelG_OP	4	4		10		7	8		11	39,1	112,3
4.	Jakub Pelc	G UherBrod	3	8		10	11		9		15	45,0	100,6
5.	Pavel Turek	GTomkovaOL	4	7	8	10	6		9		12	47,6	99,8
6.	Roman Bujdák	G JM Galanta	3	3	8	10	4		7			31,2	99,5
7.	Peter Grajcar	GMetodovaBA	3	3	2	10	3		6			27,1	92,7
8.	Rajmund Hruška	GPošKošice	4	2	8	10			9			27,0	70,0
9.	Pavel Turinský	G Brandýs	4	12	8	10			9		2	28,4	67,4
10.	Filip Geib	G MMH LM	3	5		6			5			14,5	66,6
11.	Jonáš Fiala	GJungmanLT	4	7								0,0	56,0
12.	Martin Picek	GJirsíkaČB	2	2	8							8,0	55,0
13.	Martin Kurečka	GJarošeBO	3	1	8	10			9	8	15	53,3	53,3
14.	Jakub Pintera	SPŠ Prosek	4	2	8							8,0	51,4
15.	Miroslav Hrabal	GTomkovaOL	3	4	8	10						18,0	43,6
16.	Matouš Bílek	GJŠkodyPŘ	2	1		10	6	6			10	41,0	41,0
17.	Matouš Mařík	G_Krumlov	4	1								0,0	40,7
18.	Lukáš Caha	GZborovPH	3	2								0,0	38,7
19.	Kateřina Čížková	G_Rokycany	3	2	8	8			8			25,8	34,6
20.	Jan Kaifer	GKepleraPH	1	5	8	10	6					24,0	34,5
21.	Tomáš Raunig	GHlu	2	2								0,0	34,3
22.	Františ ek Kmječ	G Brandýs	1	4								0,0	33,3
23.	Kryštof Mitka	ZŠUniverzum	0	3	2	10						13,8	31,2
24.	František Deckert	GOpatovPHA	4	1	8	10	11					29,0	29,0

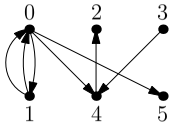
Ukázkový vstup:

```
6
2 3
1 2
1 0
0 1
2 1
1 0
```

Ukázkový výstup:

```
0 1 1
0 4 1
0 5 1
1 0 2
3 4 1
4 2 1
```

Odpovídající bipartitní multigraf vypadá následovně:



I v tomto případě jde jen o jedno z mnoha možných řešení.

Toto je praktická open-data úloha. V odevzdávacím systému si necháte vygenerovat vstupy a odevzdáte příslušné výstupy. Záleží jen na vás, jak výstupy vyrobíte.

Do místnosti vešel starší strážník, patrně náčelník zdejší stanice. „Tak,“ odmlčel se před nepříjemnou otázkou, „kdo tady bude v pátek?“

„Já ne!“ ozval se jeden z policistů. „Viděli jste můj horoskop?“

Ostatní si vytáhli každý po jedné sirce.

Náčelník přešel do vedlejší místnosti, kde mezitím pokračovala přestavba místní počítačové sítě. Byla tam spousta počítačů, switchů, routerů a mezi nimi čím dál více kabelů. „Pozor kabel!“ zavolal jeden z instalujících.

„Na co tu máme víc počítačů než celý útvar informační bezpečnosti? O ty kabely se brzo někdo přerazí. . . “

„Evropská unie,“ odvětil montér, nepřestávaje tahat kabely. „Pokud neutratíme všechno, musíme dotaci vrátit. Ale oficiálně – jako zálohu pro případ výpadku.“

29-4-4 Policejní síť — 8 bodů

Policejní síť tvoří N počítačů, každý může být propojen s několika dalšími. Síť tvoří strom.

Navíc máme určeno K důležitých počítačů (vrcholů). Ty chceme spárovat do dvojic, které se budou navzájem zálohovat: pokud jeden počítač z dané dvojice přestane fungovat, zastoupí jeho činnost ten druhý.

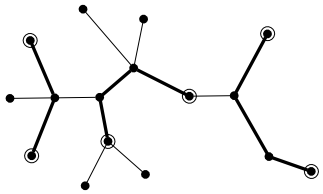
Aby to bylo možné, musí si počítače ve dvojici průběžně navzájem předávat všechna svá data – ta tečou po cestě, která dané dva vrcholy ve stromu spojuje (říkejme jí *spojení*).

Není určeno, který počítač máme spárovat s kterým, můžeme si vybrat libovolně. Ale protože nakoupené počítače přes svou cenu nejsou příliš výkonné, nechceme, aby jakýmkoli počítačem procházelo víc než jedno spojení.

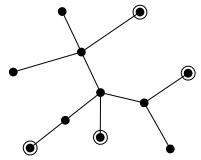
Tedy hledáme takové rozdělení důležitých vrcholů do dvojic, aby cesty spojující vrcholy v každé dvojici byly navzájem disjunktní – neměly žádné společné vrcholy ani hrany.

Pokud existuje více možných řešení, vypište všechna.

Na následujícím obrázku je příklad takového stromu. Důležité vrcholy (dostanete na vstupu) jsou označeny kroužkem. Nalezená spojení (výstup algoritmu) jsou zvýrazněná tučně. V tomto případě existuje jediné řešení.



Ale pro následující zadání žádné řešení neexistuje:



Ráno se Vilémovi povedlo při vypínání budíku spadnout z postele a v koupelně si málem zlomil nohu. Když mu jeho chytré zrcadlo s kalendářem prozradilo, že je čtvrtek 12., raději nepřemýšlel o tom, jak zlý bude příští den, a rozhodl se doplnit zásoby.

Jelikož nemohl najít oblíbenou tašku, vzal si prostě svůj běžný batoh a přihodil raději pár talismanů. Jim navzdory ho málem praštily vchodové dveře. Ještě že na ulici nebylo moc lidí, takže si netrhl takovou ostudu.

Neprošel ani celou ulicí, když se mu rozvázala tkanička, on o ni zakopl a při pádu hodil klíče do kanálu. Tehdy ho dostihla děsivá myšlenka. Jeho středeční zkouška byla předevčírem, přesto jeho zrcadlo včera tvrdilo, že je středa, a to znamená. . . že je pátek třináctého!

Na chvíli ho popadl vztek. Dávno bylo schváleno, že ten den budou vždy od časného rána do večera znít sirény, a zatím ticho. Holt byly sirény asi rozbité, jako obvykle. Vztek ale vystřídala zvědavost. Kdo je tedy těch pár lidí, kteří si troufli vyjít ven?

V té chvíli se ovšem jeden z těch lidí vydal právě k Vilémovi. „Co tu sakra vyvádíš? Vy nováčci 'ste pořád větší amatéři!“ nadával, popadl Viléma a vlekl ho ulicí. Vilémovi se hlavou honily myšlenky a moc nestíhal vnímat cestu. Najednou byli před jakousi budovou, najednou se v ní proplétali různými neznačenými schodišti a východy. . . a najednou byli na místě. Mohli být vážně ve třináctém patře?

Vilém se opatrně rozhlédl kolem sebe. Jeho pozornost upoutala zejména knihovna u zdi. Byly v ní vyskládané různé knihy a sešítky a. . . to snad nebylo možné! Novákovy spisky o štěstí a náhodě. Mohly všechny ukradené kopie z Archivu dokumentů o mezipřirozenosti zmizet sem?

29-4-5 Chybějící spisek — 12 bodů

Z Archivu dokumentů o mezipřirozenosti bylo postupně ukradeno mnoho očíslovaných spisků. Vilém nyní M těchto spisků vidí v knihovně v tajné skrýši a zajímalo by ho, jaké nejmenší číslo spisku tu chybí.

Protože ale v místnosti není sám, nemůže spisky jen tak přerovnávat, stejně tak si nemůže prostě vytáhnout papír a tužku a psát si poznámky. Musí si vystačit s omezeným množstvím paměti poskytnutým vlastní hlavou. . .

Formálněji: v paměti máte k dispozici nesetříděnou posloupnost N čísel. Tato část paměti je jen pro čtení, to zejména znamená, že si posloupnost nemůžete setřídit. Dále máte k dispozici konstantní množství pomocné paměti. Určete nejmenší přirozené číslo, které se v posloupnosti nevyskytuje.

Ukázkový vstup:	*Ukázkový výstup:*
3 0 9 8 6 1	2

Něco Viléma vrátilo do přítomnosti, upoutalo jeho pozornost zpátky do místnosti. Mezitím se tu nahromadilo dost lidí, a ti všichni teď umlkli a pozorovali přicházejícího muže.

Vilém si nebyl jistý, jestli ho víc znepokojuje černá kočka v mužově náručí, nebo fakt, že nikoho jiného tato maličkost zjevně netrápí.

„Jsem rád, že vás tu všechny zase vidím," ujal se muž slova. „Jak jistě víte, černá kočka je ideálním prostředkem k neutralizaci nežádoucích osob. Výsledek nejenže vypadá jako nehoda, ona to doopravdy je nehoda.

Musíte správně odhadnout počet. Jedna obvykle nestačí, ale pokud jich použijete příliš," pohlédl přísně na jednoho z přítomných, „jsou druhý den noviny plné zpráv o zásazích blesku na železničním přejezdu. Jenže i když se trefíte, občas někdo kočku zahlédne...

Proto jsem vážně rád, že se nám tehdy povedlo zfalšovat utracení téhle krásky," zdvihl kočku spočívající mu v náručí. „Určitě si vzpomínáte, jak byla vláda nesvá z černé kočky, která nenosí smůlu, ale nejspíš ani oni nevěděli, jak silnou mají v ruce zbraň."

„Naši výzkumníci se ovšem pustili do práce a zjistili...," muž se zamyšleně zamračil, pak mávl rukou, „nějakou genovou odlišnost. A protože tu od nás mají skvělé vybavení a jsou šikovní, dovolte mi představit vám," dramaticky se otočil a ukázal k přicházející hnědé kočce.

Obezřetně k ní došel, opatrně ji vzal do náruče a pak si z přihlížejících vyvolal jednoho neochotného dobrovolníka. O pár spadlých předmětů, vylitých skleniček, zakopnutí a dalších pozoruhodných náhod později začínalo být jasné, jak mocnou zbraň má skupina k dispozici.

„Hele, a proč vlastně nenabarvíme nějakou černou kočku?" zeptal se tiše někdo poblíž Viléma. „Nepamatuješ si snad, co se stalo Dlouhánovi, když se o to pokusil?" odpověděl hned jiný a pak byl klid, dokud se slova opět neujal šéf.

„Věřím, že s naším novým miláčkem se nám bude dobře dařit a pěkně se rozrosteme. Proto si taky už brzy pořídíme nové sídlo, hned jak vymyslíme, jak velký pozemek vlastně chceme koupit."

29-4-6 Nové sídlo — 11 bodů

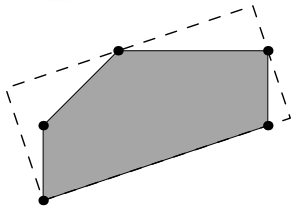
Zločinecká organizace si chce postavit nové sídlo. To má půdorys ve tvaru konvexního mnohoúhelníku. Ráda by koupila nejmenší možný obdélníkový pozemek, na který se budova vejde. Ve městě jsou pozemky drahé a jiný než obdélníkový vám neprodají. Zároveň by ale chtěla, aby budova alespoň jednou stranou přiléhala k ulici (tedy k okraji pozemku).

Formálně: je dán konvexní mnohoúhelník. Najděte (obsahem) nejmenší obdélník, do kterého se mnohoúhelník celý vejde. Obdélník může být v obecné poloze (libovolně natočený), ale chceme, aby alespoň jedna strana mnohoúhelníku přiléhala ke straně opsaného obdélníku.

Formát vstupu: Počet vrcholů mnohoúhelníka N a souřadnice jeho vrcholů v pořadí na obvodu. Můžete předpokládat, že souřadnice jsou celočíselné.

Formát výstupu: Jedno reálné číslo udávající obsah nejmenšího opsaného obdélníku splňujícího popsaná kritéria.

Ukázkový vstup:	*Ukázkový výstup:*
5 0 0 6 2 6 4 2 4 0 2	22



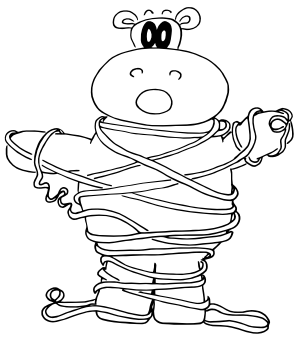
Tohle je zlé...

Vilém tušil, že by měl něco udělat, ale neměl ponětí co. Zavolat policii zřejmě nepomůže. Dnes si nikdo zasáhnout netroufne. I kdyby troufl, nejspíš by to skončilo fiaskem.

V zoufalství prohrábl kapsy. Zalaminovaný čtyřlístek měl jeden list ulomený. Pochopitelně.

Krom jiného vytáhl i malé klubíčko vlněné příze, kterým občas bavíval své domácí kočky. Teprve při pohledu na něj si pořádně uvědomil, že nebezpečná kočka nosící smůlu je pořád kočka. A kočky si rády hrají.

Pár metrů odmotal, zbytek zajistil uzlem a hodil směrem k hnědé kočce. Plán byl jednoduchý: upoutat její pozornost, přitáhnout klubíčko zpět a s trochou štěstí (ehm) ji tím přilákat.



Ale přece byste nečekali, že v pátek třináctého nějaký plán vyjde. Zhruba ve stejnou chvíli se staly dvě věci: šéf se ohlédl Vilémovým směrem a provázek se přetrhl. Kočka viděla klubíčko dopadnout asi metr před sebe. Najednou pocítila zvlášní nutkání líně odhlédnout a odejít, které přicházelo zdánlivě odnikud.

Ale sama měla jiný názor. Tady se objevil zajímavý předmět, který chce prozkoumat, a pokud si vesmír myslí, že by měla znuděně odejít, tak si vesmír může s prominutím trhnout.

Neslyšně se připlížila ke klubíčku a párkrát do něj šťouchla packou. Vždy se o kus pohnulo a skutálelo zpátky.

„Co tam blbneš? Nehraj si a dávej pozor!" adresoval nevrle šéf Vilémovi.

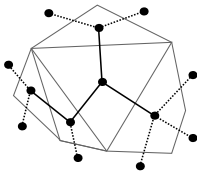
Ono se to nehýbe! Asi spí... Kočka se napřáhla a prudce vymrštila směrem ke klubíčku, odrážejíc ho o několik metrů kupředu. Wihíí!

Šéf, nezpozorovav toto dění, se pomalu rozešel směrem k Vilémovi právě ve chvíli, kdy kočka vyběhla za klubíčkem — a zkřížila tím šéfovi cestu.

* * *

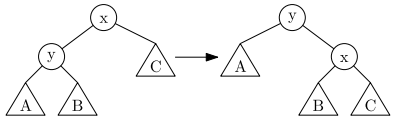
29-3-6 Obrazec pro draka

Pro jednodušší řešení úlohy je dobré převést si mnohoúhelník na něco, s čím se pracuje lépe. Vytvoříme si graf, jehož vrcholy představují trojúhelníky a hrany reprezentují tyče. Vrcholy sousedních trojúhelníků jsou tedy spojené hranou. Ještě se nám bude hodit, když i strany mnohoúhelníku budou hrany a za každou stranou budeme mít také vrchol. Můžeme si všimnout, že tento graf je stromem.

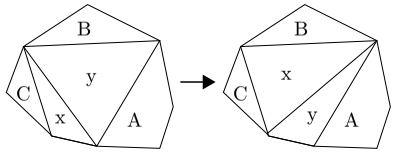


Nyní si můžeme vybrat jeden z vrcholů mimo mnohoúhelník a prohlásit ho za kořen našeho, nyní binárního, stromu. Teď by nás zajímalo, co udělá s naším stromem jedno překlopení tyče. Ukážeme si, že odpovídá operaci stromové rotace.

Rotace je „otočení" hrany mezi dvěma vrcholy, kde zachováváme pořadí vrcholů a podstromy převěsíme, viz obrázek.



Přesně tohle udělá zvednutí tyče a její umístění napříč:



Počáteční i cílový obrazec převedeme na binární strom, kde za kořen zvolíme ten stejný vrchol (neboli vrchol za stejnou stranou mnohoúhelníku). Teď hledáme, jak převést pomocí rotací jeden na druhý. Ještě je dobré si očíslovat listy – tedy vrcholy za hranami mnohoúhelníku. Aby dva stromy reprezentovaly stejnou triangulaci, tak musí sedět i očíslování listů.

Stromové rotace mají jednu důležitou vlastnost, totiž zachovávají pořadí listů. Nestane se nám tak, že by se pořadí listů nějakým způsobem pomíchalo (což by znamenalo, že by se nám i mnohoúhelník musel nějak překlápět). Zachování této vlastnosti je důležité, protože bychom jinak mohli sice vymyslet způsob, jak přejít od jednoho stromu k druhému, ale neseděly by nám listy, tedy by vlastně vůbec nemuselo jít o tu samou triangulaci.

Nyní už jsme velmi blízko cíle. Připomínáme, že hledáme libovolnou posloupnost rotací, nemusí být nutně nejkratší. Pokud budeme opakovat dostatečně dlouho rotace do jednoho směru, třeba doleva, získáme lineární strom.

Stačí začít u kořene a opakovat rotace, dokud není vpravo jenom list. Poté půjdeme k jeho pravému synovi a budeme to opakovat. V každé rotaci jeden vrchol zařadíme do lineárního stromu a tedy nám to bude trvat $\mathcal{O}(n)$ kroků.

To samé můžeme provést s cílovým stromem. Využijeme toho, že k rotaci vpravo je rotace vlevo inverzní operací. Abychom získali hledanou posloupnost překlopení, můžeme provést rotace stromu počátečního obrazce na lineární strom a pak pozpátku ty, co jsme provedli s cílovým stromem.

Strom sestrojíme v lineárním čase, obě převedení na lineární strom zvládneme $\mathcal{O}(n)$ rotacemi, a protože každá trvá jen konstantní čas, tak celkový čas bude $\mathcal{O}(n)$. Počet rotací bude také $\mathcal{O}(n)$.

Najít nejkratší posloupnost rotací, která převádí jeden binární strom na druhý, v polynomiálním čase zatím bohužel neumíme. Jestli to vůbec jde, je stále otevřený problém. Umožnilo by nám to efektivně spočítat rotační vzdálenost dvou stromů, totiž kolik nejméně rotací je potřeba pro převedení jednoho stromu na jiný. To je hezká metrika – způsob, jak měřit „vzdálenost" (rozdílnost) dvou stromů.

Jirka Sejkora

29-3-7 Stromoví předci

Úkol 1: Chytřejší značkování

Ke kořeni chceme stoupat z obou vrcholů současně. Jelikož nám asi nedali paralelní počítač, budeme to co nejvěrněji simulovat. Vždycky jeden krok na cestě z prvního vrcholu, pak jeden z druhého, a tak dále. Vrcholy na obou cestách značkujeme a jakmile první vrchol dostane obě značky, je to hledaný společný předchůdce (LCA).

Jak dlouho to trvá? Označme d_1 a d_2 vzdálenosti k LCA. Tento LCA dostane první značku po d_1 krocích, druhou po d_2. Algoritmus se tedy zastaví po $\mathcal{O}(\max(d_1, d_2))$ krocích, což je totéž jako požadovaných $\mathcal{O}(d_1 + d_2)$.

Úkol 2: Minimum svislé cesty

Chceme počítat minimum ohodnocení hran na „svislé" cestě mezi vrcholem x a jeho předkem p. Předpočítáme si hloubky vrcholů $d(v)$, takže dotaz umíme přeložit na minimum na cestě mezi x a $Pra(x, d(x) - d(p))$.

V zadání jsme ukázali, jak si předpočítat skočky, tedy hrany z v do $Pra(v, 2^i)$, a pak počítat $Pra(w, k)$ složením $\mathcal{O}(\log n)$ skoček. Nyní si pro každou skočku předpočítáme ještě minimum z ohodnocení přeskakovaných hran. To pro každou zvládneme v konstantním čase složením dvou už spočítaných minim. A při skládání $Pra(v, 2^k)$ ze skoček rovnou složíme i příslušná minima.

Předvýpočet trvá $\mathcal{O}(n \log n)$, pak odpovídáme v $\mathcal{O}(\log n)$.

Úkol 3: Součet svislé cesty

Součet je mnohem jednodušší. Spočítáme si analogii prefixových součtů, tedy součty $S(v)$ všech hran z kořene do v. Součet na cestě mezi x a jeho předkem p pak je prostě $S(x) - S(p)$. Předvýpočet trvá $\mathcal{O}(n)$, na dotazy odpovídáme v $\mathcal{O}(1)$.

Úkol 4: Minimum obecné cesty

Minimum nebo součet na obecné cestě mezi x a y spočteme tak, že nejdříve nalezneme $\ell = lca(x, y)$. Pokud je $x = \ell$ nebo $y = \ell$, cesta je svislá a jsme hotovi. V opačném případě cestu rozložíme na dvě svislé cesty: z x do ℓ a z ℓ do y, pro které už umíme odpovědět.

Úkol 5: Součet pomocí ET-posloupnosti

Dostaneme ET-posloupnost, do níž jsme při průchodu hranou směrem dolů napsali její ohodnocení, a při návratu nahoru minus ohodnocení. Chceme počítat součty na svislých cestách, opět označíme x nižší vrchol cesty a p ten vyšší.

do šířky stále dostávat políčka ve správném pořadí – takové prohledávání do šířky totiž přiřadí políčkům stejná ohodnocení, jako kdybychom ho spustili na původní pláni. Nad tímto prohledáváním do šířky také můžeme přemýšlet jako nad Dijkstrovým algoritmem, který namísto haldy používá frontu.

Protože hlídek bylo k, tak takto upravená pláň má rozměry $O(k^2)$ (mezi sousedními hlídkami je maximálně jeden zkontrahovaný sloupec nebo řádek) a předpočítat si všechny nejkratší cesty tedy bude trvat $O(k^4)$.

Potřebujeme ještě umět zjistit, zda je v obdélníku definovaném startem a cílem hlídka. To můžeme udělat jednoduše pomocí dvoudimenzionálních prefixových součtů (o nich si můžete přečíst třeba v naší kuchařce základních algoritmů).[5] Hlídky budeme považovat za jedničky a prázdná políčka za nuly. V daném obdélníku pak bude hlídka právě tehdy, když je v něm nenulový součet.

Dotaz pak bude vypadat následovně: Pokud mezi políčky není žádná hlídka, délka nejkratší cesty je $|x_s - x_c| + |y_s - y_c|$. Pokud mezi nimi hlídka je, využijeme naší kontrahované pláně, kde máme pro každou dvojici políček předpočítanou nejkratší cestu

Drobnou nesnází je, že start (nebo cíl) mohou ležet uvnitř nějakého kontrahovaného obdélníku. Můžeme si rozmyslet, že část cesty, která je v tomto obdélníku, může jít libovolnou nejkratší cestou do rohu nejbližšího k cíli (respektive startu), tuto část cesty spočítáme jako v případě výše.

A dál už pak vyhledáváme jen ve zkontrahované pláni, respektive v předpočítané datové struktuře délek cest mezi dvojicí zkontrahovaných políček.

Předpočítání kontrahované pláně bude trvat $O(MN + k^4)$ a stejně prostoru bude zabírat výsledná datová struktura. Na dotazy pak budeme schopni odpovídat v konstantním čase.

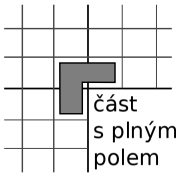
Kuba Tětek

29-3-5 Dračí zámek

K zadání této úlohy jste všichni dostali nápovědu, totiž kuchařku o metodě Rozděl a panuj. Toho jste všichni správně využili, a třebaže došlá řešení využívala různé přístupy, vždy byly založeny na této metodě.

Takže jak se k úloze správně postavit? Připomeňme, že čtvercová mřížka má rozměry $N \times N$, kde N je nějaká mocnina dvojky. Je snadné si všimnout, že pro $N = 1$ má úloha triviální řešení: máme jediné plné pole.

Pro větší N chceme celé zadání rozdělit na menší úlohy. Mřížku uprostřed rozsekneme na čtyři podmřížky, každou s rozměry $\left(\frac{N}{2}\right) \times \left(\frac{N}{2}\right)$. Na podmřížku, kde se vyskytuje plné pole, můžeme rekurzivně zavolat stejný algoritmus. Pro ostatní podmřížky si pomůžeme tak, že do rohu, který vzájemně tvoří, vložíme navíc dílek:



V každé z nich se teď nachází plné pole, tudíž i na ně se můžeme zavolat rekurzivně.

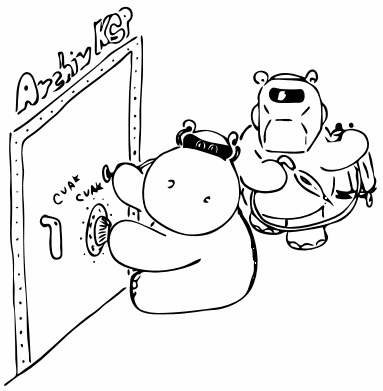
Celý algoritmus slouží zároveň i jako důkaz, že kteroukoliv mřížku velikosti $N = 2^k$ lze pokrýt dílky. Počáteční pozorování pro $k = 0$ je indukčním předpokladem, rozdělení na podmřížky indukčním krokem.

Pokud bychom chtěli algoritmus implementovat (což jsme od vás nepožadovali), je zajímavé se podívat na časovou složitost. Budeme následovat výše uvedený postup a vytvoříme funkci, která vždy položí nový dílek a navíc pro $N > 1$ zavolá rekurzivně čtyřikrát sama sebe. Samotný průběh funkce má konstantní časovou složitost (pouze vypočítáme polohu plného pole), takže zbývá vyřešit, kolikrát se zavolá.

Zkusíme pro změnu přemýšlet odspodu: voláme funkci na každou podmřížku velikosti 1×1, a těch se v mřížce nachází N^2; dále na každou podmřížku velikosti 2×2, těch je celkem $\frac{N^2}{4}$. Dostáváme se tak k součtu řady: $N^2 + \frac{N^2}{2} + \frac{N^2}{4} + \ldots + 1$. K jejímu řešení můžeme využít například kuchařkovou větu (Master Theorem), která nám odpoví, že celková časová složitost je $O(N^2)$.

Program (Python):
`http://ksp.mff.cuni.cz/viz/29-3-5.py`

Kuba Maroušek



[5] `http://ksp.mff.cuni.cz/viz/kucharky/zakladni-algoritmy`

Na policejní stanici byl toho dne klid. Seděl tu jediný strážník, který si vytáhl kratší sirku, a pozorně si prohlížel protější zeď. Raději si netroufl ani číst noviny.

Když tu z ničeho nic do služebny zmateně vběhl vyděšený muž. Vypadal, jako by spatřil nějaký přízrak, a pravděpodobně netušil, kde se nachází. Proběhl kanceláří bez ohlédnutí otevřenými dveřmi do sousední místnosti… kde zakopl o jeden z nově natažených síťových kabelů.

Při pádu mu vypadla spousta schémat a náčrtků popisujících plány jeho zločinecké organizace, poznámky o genetice černých koček, seznamy likvidovaných osob. Strážník k němu opatrně přišel a papíry prolistoval.

„Kdopak se to chytil – do naší sítě?“ nemohl si odpustit úsklebnou poznámku. „Z toho si nic nedělejte, tohle se tu stává pravidelně. V pátek třináctého zločince nechytáme. Nejenže to není bezpečné, ale hlavně to není potřeba. Přichází sami…“

Nepodceňujte kočky.

Příběh pro vás přichystali

Karry Burešová & Filip Štědronský

29-4-7 Rozebíráme stromy — 15 bodů

Vítejte u dalšího dílu stromového seriálu. Tentokrát se zaměříme na *cestové operace*. Tím myslíme takové, které dostanou dva vrcholy stromu a mají něco provést s cestou mezi nimi. Třeba zjistit, jak je tato cesta dlouhá, nebo najít na ní hranu s největším ohodnocením.

Pro mělké stromy je to snadné: hledáme-li cestu mezi vrcholy x a y, stačí z obou vrcholů stoupat směrem ke kořeni, až se obě cesty poprvé protnou v nejbližším společném předchůdci p (to jsme prozkoumali v minulém dílu). Hledanou cestu pak můžeme poskládat ze dvou částí: cesty mezi x a p a cesty mezi y a p. Vše zvládneme v čase lineárním s hloubkou stromu.

Snadné to je i pro stromy, které se vůbec nevětví, tedy samy mají tvar cesty. V druhém dílu jsme se naučili přimět intervalové stromy, aby v logaritmickém čase vyhodnocovaly dotazy na libovolné intervaly posloupnosti. Můžeme si tedy pořídit intervalový strom pro posloupnost hran na cestě a intervaly pak budou odpovídat jejím podcestám. Dokonce pomocí líného vyhodnocování zvládneme rychle měnit ohodnocení hran na podcestě.

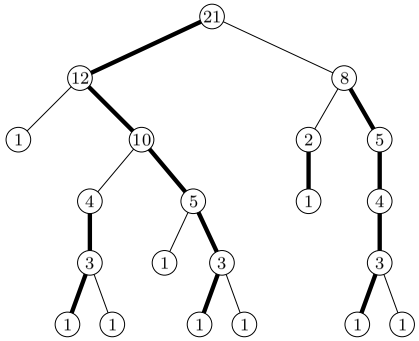
V tomto dílu si ukážeme, jak tyto dvě techniky spojit. Zavedeme takzvanou *dekompozici stromu na lehké a těžké hrany* neboli *heavy-light dekompozici*. Ta nám pomůže k rychlému vyhodnocování cestových dotazů v libovolně hlubokém a libovolně košatém stromu. Jen se nám strom nebude smět pod rukama měnit, tedy až na ohodnocení vrcholů a hran.

Heavy-light dekompozice (HLD)

Nejprve pár definic. Mějme nějaký zakořeněný strom. Označíme $T(v)$ *podstrom* složený z vrcholu v a všech jeho (i nepřímých) potomků. *Velikostí podstromu* míníme počet jeho vrcholů a budeme ji značit $size(v)$.

Hrany z každého vrcholu do jeho synů rozdělíme na *lehké* a *těžké* následovně: hranu do největšího podstromu prohlásíme za těžkou, všechny ostatní za lehké. Existuje-li více největších podstromů, vybereme si libovolný jeden z nich.

Jak to dopadne pro jeden konkrétní strom, vidíme na následujícím obrázku. Čísla ve vrcholech jsou jejich *size*.



Těžké hrany jsou na obrázku vyznačeny tučně a zjevně tvoří cesty. To není náhoda, platí to v každém stromu. Stačí si uvědomit, že z každého vrcholu může vést dolů nejvýše jedna těžká hrana. (Dokonce víme, že není-li vrchol list, vede z něj právě jedna taková.)

Navíc platí, že lehkých hran není nikdy mnoho za sebou. Přesněji řečeno, na cestě z kořene do libovolného vrcholu v leží nejvýše $\log_2 n$ lehkých hran (n jako obvykle značí velikost celého stromu).

Pojďme to dokázat. Vydejme se z kořene do v a sledujme, jak se mění *size* aktuálního vrcholu. Za chvíli uvidíme, že kdykoliv projdeme po lehké hraně, klesne *size* aspoň dvakrát. Po k lehkých hranách tedy klesne aspoň 2^k-krát, takže kdyby bylo $k > \log_2 n$, nezbyly by v podstromu žádné vrcholy.

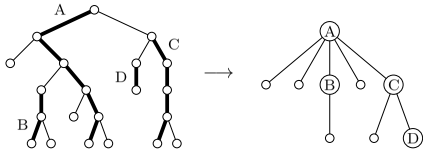
Dobrá, proč tedy na lehké hraně *size* tolik klesá? Uvažme lehkou hranu z nějakého vrcholu u do jeho syna ℓ. Z definice musí existovat i těžká hrana do jiného syna t a platí $size(t) \geq size(\ell)$. Jenže podstromy $T(\ell)$ a $T(t)$ jsou součástí $T(u)$, takže $size(u) > size(t) + size(\ell)$ a z toho ihned $size(u) > 2\, size(\ell)$.

Pojďme zopakovat, co jsme zjistili:

- Heavy-light dekompozice rozkládá strom na *těžké cesty*, které jsou propojené *lehkými hranami*.
- Každý vrchol leží na právě jedné těžké cestě. (Tedy připustíme-li i cesty složené z jediného vrcholu.)
- Každá lehká hrana vede z vrcholu nějaké těžké cesty do nejvyššího vrcholu jiné těžké cesty.
- „Lehká hloubka“ je logaritmická. Přesněji řečeno, mezi každými dvěma těžkými cestami leží $\mathcal{O}(\log n)$ lehkých hran.

Úkol 1 [2b]: Někdy se používá jiná definice těžkých hran: hrana z vrcholu v do syna s je těžká, pokud $size(s) > size(v)/2$. Rozmyslete, jak se takto definovaná dekompozice bude lišit od té naší. Překreslete podle toho předchozí obrázek.

Dekompozici si můžeme představit i tak, že každou těžkou cestu zkontrahujeme do jediného vrcholu. Tím dostaneme jiný strom, v němž zbudou jen původní lehké hrany a bude logaritmicky hluboký. Jak vypadá, je vidět na následujícím obrázku. Vrcholy bez písmenek odpovídají triviálním (jednovrcholovým) těžkým cestám.



Výpočet dekompozice

Nyní se podíváme, jak HLD reprezentovat v paměti a zejména jak ji rychle sestrojit.

V každém vrcholu v našeho stromu si budeme pamatovat:

- $size(v)$ – velikost podstromu pod v
- $hson(v)$ – do kterého syna vede těžká hrana (nebo $\emptyset$, pokud žádný není, což je možné jen tehdy, je-li v list)
- $path(v)$ – odkaz na těžkou cestu, na níž vrchol leží
- $index(v)$ – kolikátý v pořadí na těžké cestě je (číslujeme od nuly od spodního vrcholu cesty)

Těžké cesty si pamatujeme bokem. Pro cestu p uložíme:

- $lparent(p)$ – otec *kořene cesty* (tak budeme říkat jejímu nejvyššímu vrcholu, tedy vrcholu s nejvyšším indexem); tedy vrchol, z nějž vede do kořene lehká hrana. Může být $\emptyset$, pokud je kořen cesty také kořenem celého stromu.
- $plen(p)$ – délka cesty (počet hran na ní)
- $pvertex(p)$ – pole vrcholů cesty v pořadí jejich indexů

Dekompozici můžeme snadno spočítat dvojím prohledáním do hloubky. Při tom prvním stanovíme velikosti podstromů, rozhodneme, které hrany jsou lehké a těžké, a vypočteme indexy. Druhé sestrojí popisy jednotlivých těžkých cest.

První prohledání vypadá následovně. Spouštíme ho v kořeni stromu a při návratu z rekurze počítá jednotlivé vlastnosti vrcholů podle definice.

```
HLD1(v):
 1. size(v) ← 1
 2. h ← ∅                                  ◁ kam povede těžká hrana?
 3. Pro všechny syny s vrcholu v:
 4.     HLD1(s)
 5.     size(v) ← size(v) + size(s)
 6.     Pokud h = ∅ nebo size(s) > size(h):
 7.         h ← s
 8. hson(v) ← h
 9. Pokud h = ∅:                           ◁ jsme v listu
10.     path(v) ← nová těžká cesta
11.     index(v) ← 0
12. Jinak:                                 ◁ pokračování těžké cesty
13.     path(v) ← path(h)
14.     index(v) ← index(h) + 1
```

Druhé prohledávání spouštíme opět z kořene. Jako druhý parametr předáváme otce aktuálního vrcholu, pro kořen je to $\emptyset$. Vždy posbíráme vrcholy na jedné těžké cestě a pak se zavoláme na všechny jejich syny.

```
HLD2(v, o):
 1. p ← path(v)          ◁ vrchol v je kořenem těžké cesty p
 2. lparent(p) ← o
 3. plen(p) ← index(v)
 4. pvertex(p) ← nové pole délky plen(p) + 1
 5. Dokud v ≠ ∅:                  ◁ procházíme těžkou cestu
 6.     pvertex(p)[index(v)] ← v
 7.     Pro všechny syny s vrcholu v:
 8.         HLD2(s, v)           ◁ při tom rekurze na syny
 9.     v ← hson(v)              ◁ pokračujeme po těžké cestě
```

Obě prohledání provedou konstantní množství práce pro každý vrchol a hranu stromu, celkem tedy $\mathcal{O}(n)$. Dodejme, že by se všechno dalo zvládnout během jediného DFS, ale ve dvou nám to přijde přehlednější.

Stromoví předchůdci

Abychom si osahali, jak se s HLD zachází, zkusíme ji použít na úlohu hledání nejbližšího společného předchůdce z minulého dílu.

Vzpomeňme si na primitivní algoritmus, který z každého ze zadaných vrcholů prošel po cestě do kořene, značkoval vrcholy a číhal, kde se obě cesty poprvé potkají. Teď na to půjdeme podobně, ale místo jednotlivých vrcholů budeme značkovat najednou celé těžké cesty.

Pojmenujme zadané vrcholy u a v. Nejprve budeme stoupat z u ke kořeni. Kdykoliv jsme v nějakém vrcholu x, z $path(x)$ se dozvíme, na které těžké cestě p se nacházíme. Pak hned vyskočíme z kořene těžké cesty po lehké hraně do vrcholu $lparent(p)$. Ještě si k těžké cestě poznačíme novou položku $enter(p)$ říkající, kudy jsme na cestu vstoupili. Nenavštívené cesty budou mít $enter(p) = \emptyset$.

Poté budeme stoupat z v. Stejným způsobem, ale místo značení těžkých cest budeme naopak testovat, zda už nejsou označené. Dříve či později narazíme na těžkou cestu p, na níž jsme už byli při stoupání z u. Přitom víme, kudy jsme se na tuto cestu v obou případech napojili – v jednom místě napojení teď stojíme, druhé máme uložené v $enter(p)$. Hledaným společným předchůdcem je vyšší z těchto dvou míst, což poznáme podle indexů vrcholů.

V pseudokódu to vypadá takto:

```
lca(u, v):
 1. x ← u                                  ◁ z u do kořene
 2. Dokud x ≠ ∅, opakujeme:
 3.     p ← path(x)
 4.     enter(p) ← x
 5.     x ← lparent(p)
 6. y ← v                                  ◁ z v do kořene
 7. Dokud enter(path(y)) = ∅, opakujeme:
 8.     y ← lparent(path(y))
 9. r ← enter(path(y))                     ◁ místa napojení: r a y
10. Dokud u ≠ ∅, opakujeme:                ◁ smažeme značky
11.     enter(path(u)) ← ∅
12.     u ← lparent(path(u))
13. Pokud index(y) > index(r):             ◁ vrátíme vyšší z r a y
14.     Vrátíme y.
15. Jinak:
16.     Vrátíme r.
```


poslední znak řetězce a dát jej na první místo. Všimněte si, že pokud zrotujeme do nějakého stavu

$$r_k\delta_{k+1}\ldots\delta_{N-1}r_{N-1}\delta_0 r_0\ldots\delta_{k-1}$$

a na konec zkopírujeme první znak (r_k), výsledek je palindromem právě tehdy, když osa prochází bodem r_k. Analogicky pokud rotací dostaneme řetězec

$$\delta_k r_k\ldots\delta_{N-1}r_{N-1}\delta_0 r_0\ldots\delta_{k-1}r_{k-1}$$

a opět zkopírujeme δ_k, výsledek je palindromem právě tehdy, když osa prochází středem úsečky mezi body P_{k-1} a P_k.

Uvědomme si, že to jsou jediné možnosti, kde osa souměrnosti může ležet. Máme-li totiž těžiště T (nebo jiný libovolný bod na ose souměrnosti), bod A a jeho obraz A', tak úhel mezi přímkou TA a osou souměrnosti musí být stejný jako mezi osou a přímkou TA'. Představme si, že by tedy osa dělila nějaký úhel δ_k na úhly δ'_k a δ''_k. Nechť je δ'_k ten menší z nich a bod při tomto úhlu (P_k nebo P_{k-1}) označíme K. Bod K se musí zobrazit na jiný bod, který dává s osou úhel δ'_k (resp. $360° - \delta'_k$, podle toho, jak se na to díváte), ale všechny ostatní body dávají úhel větší (resp. menší), K tedy nemá obraz a zkoumaná přímka není osa.

Stačí vyzkoušet všechny tyto rotace a podívat se zda nejsou *skoropalindromem*. Pokud si budeme uchovávat řetězec ve spojovém seznamu s ukazatelem na začátek i konec, další rotaci vytvoříme v konstantním čase, stačí odebrat prvek z konce seznamu a dát jej na začátek. Samotná kontrola, jestli je řetězec *skoropalindromem*, bude v lineárním čase. Stačí zkontrolovat jestli je první prvek shodný s předposledním, druhý s předpředposledním atd. To zvládneme pomocí dvou ukazatelů, které vždy posuneme o jednu pozici.

Nicméně to opět vypadá, že jsme si vůbec nepomohli. Jednu rotaci zkontrolujeme v čase $\mathcal{O}(N)$, ale rotací je také $\mathcal{O}(N)$, dohromady dostaneme opět $\mathcal{O}(N^2)$. Nicméně častým trikem, když hledáme vhodnou rotaci, je negenerovat nové a nové rotace, nýbrž řetězec zkopírovat dvakrát za sebe. Zkusme to také.

Původně jsme hledali rotaci s palindromem délky $2N-1$ (nezapomínejme, že N je počet bodů, délka s je tedy $2N$), stejně tak můžeme hledat palindrom délky $2N-1$ ve zdvojeném řetězci. To můžeme udělat tak, že najdeme nejdelší palindrom liché délky, pokud je delší než $2N-1$, můžeme jej jednoduše zkrátit (odebíráním vždy dvojicí znaků z kraje) na tuto délku. A jak najít nejdelší palindrom? Na to se podíváme spolu v páté sérii. Zatím jen prozradíme, že to zvládneme v lineárním čase.

Už jsme téměř na konci, ale nesmíme zapomenout ještě na jednu věc. Na začátku tohoto řešení jsme předpokládali, že žádné dva body nebudou mít přiřazený stejný úhel φ_i.

S tím se už dá celkem snadno vypořádat. Trochu upravíme konstrukci řetězce s. Když budeme mít několik bodů stejný úhel, napíšeme jejich vzdálenosti do tohoto řetězce hned za sebou v setříděném pořadí. Protože ale potřebujeme, aby se sekvence bodů se stejným úhlem četla stejně popředu i pozpátku (abychom mohli problém převést na hledání palindromu), tak ji hned za ni zapíšeme znova, v opačném pořadí. Náš řetězec může vypadat například takto: $\delta_0 r_0 r_1 r_2 r_2 r_1 r_0 \delta_3 r_3 r_3 \delta_4\ldots$

Odpovídajícím způsobem upravíme i délku hledaného palindromu. Ta je $2N+A$, kde N je počet bodů a A je počet nenulových δ_i.

Pojďme si to shrnout a podívat se na výslednou složitost. Posunout body podle těžiště, stejně tak spočítat polární souřadnice zvládneme v lineárním čase. Setříděním bodů podle úhlů strávíme $\mathcal{O}(N \log N)$. Zkonstruovat a zdvojit řetězec opět zvládneme lineárně a konečně jsme slíbili, že nalezení samotného palindromu jde také rychle. Celkově jsme se tedy konečně dostali na časovou složitost $\mathcal{O}(N \log N)$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/29-3-3.py`

Dominik Smrž & Martin „Medvěd“ Mareš

29-3-4 Mezi hlídkami

Ze všeho nejdříve úlohu převedeme na variantu, kde je zakázáno vstupovat pouze na políčka s hlídkou, ale na sousední šlápnout můžete. Uděláme to tak, že přidáme „virtuální“ hlídku na všechna pole sousedící s hlídkami ze zadání. Odpověď pro takto upravenou úlohu a vstup bude stejná jako pro původní zadání.

Jednou z možností, jak se úloha dala řešit, bylo si na začátku najít pomocí prohledávání do šířky nejkratší cesty mezi všemi dvojicemi políček. Při odpovídání na dotaz už budeme mít délku nejkratší cesty předpočítanou a můžeme ji jen vrátit.

Protože počítáme cesty na poli velikosti $N \times M$ políček, zabere nám vyhledání nejkratších cest z jednoho políčka do všech ostatních čas $\mathcal{O}(NM)$ (jedno prohledávání do šířky). Jelikož potřebujeme počítat vzdálenost mezi všemi dvojicemi políček, musíme prohledávání spustit pro každé políčko samostatně, což zabere $\mathcal{O}((NM)^2)$ času a $\mathcal{O}((NM)^2)$ paměti. To není příliš rychlé, zkusíme to vylepšit.

Rychlejší postup

Můžeme si všimnout, že vzhledem k malému počtu hlídek nejkratší cesta půjde většinu času po části pláně, kde žádné hlídky nejsou. Toho jde využít a dosáhnout tak lepší časové i paměťové složitosti. Dál v řešení budeme pracovat se souřadnicemi startu x_s, y_s a souřadnicemi cíle x_c, y_c.

Řešení si rozdělíme na dva případy – buď neexistuje hlídka, která je v obdélníku definovaném startem a cílem, a délka nejkratší cesty je tedy $|x_s - x_c| + |y_s - y_c|$, nebo nám po cestě nějaká hlídka bude překážet a budeme to muset vyřešit. Tyto dva případy také musíme být schopni odlišit, což vyřešíme v poslední části řešení.

Chtěli bychom si předpočítat nejkratší cestu mezi každými dvěma vrcholy, jenže to by trvalo příliš dlouho. Všimneme si ale, že ve sloupcích a řádcích, kde není hlídka, se „nic neděje“. Pokud je takových řádků nebo sloupců více vedle sebe, tak vždy všechny sloučíme (zkontrahujeme) do jednoho a poznamenáme si ke každému políčku, kolika políčkům odpovídá ve vertikálním a horizontálním směru. Jednotlivá zkontrahovaná políčka tak mohou odpovídat i docela velkých obdélníkům v původní pláni. Nejkratší cesty si pak předpočítáme až na této upravené pláni.

Při slučování si u každého políčka z původní pláně navíc zaznamenáme, kde je jeho „sloučená verze“ v kontrahované pláni. To se nám bude později hodit a takto se k této informaci dostaneme v konstantním čase.

Na této kontrahované pláni budeme chtít hledat nejkratší cesty. Může se zdát, že po úpravě, kdy některá políčka reprezentují větší vzdálenosti, nebude běžné prohledávání do šířky stačit, ale můžeme si rozmyslet, že díky kontrahování vždy celých řádků a sloupců bude i obyčejné prohledávání

nějaká tato dvojice není navzájem obrazem, některý bod z dvojice nemá obraz a zkoumaná přímka není osou.

Původní setřídění bodů zvládneme v čase $\mathcal{O}(N \log N)$, třídění menších seznamů stihneme ještě rychleji. Kontrolu po setřídění stihneme v lineárním čase. Jelikož zkoumaných přímek je stále lineárně, činí celková složitost $\mathcal{O}(N^2 \log N)$.

Celý tento algoritmus můžeme ještě zrychlit použitím hešovací tabulky.[4] Jednoduše si souřadnice všech bodů do jedné takové tabulky uložíme. Pak pro každý bod spočítáme souřadnice jeho obrazu podle dané osy a podíváme se do tabulky, jestli se tam bod s takovými souřadnicemi nachází. Jelikož zjištění existence v hešovací tabulce proběhne v průměrně konstantním čase, získáme ověření osy v čase průměrně lineárním. Celkově tedy v průměrném čase $\mathcal{O}(N^2)$. Toto řešení už stačilo na získání plného počtu bodů.

Těžiště na pomoc

Pojďme se ale ještě podívat na řešení jiného typu, třeba povede k ještě lepším výsledkům. Někteří z vás chytře využili toho, že těžiště bodů se jistě nachází na ose souměrnosti. Proč tomu tak vlastně je? Těžiště můžeme počítat postupně, tedy tak, že si body rozdělíme do skupinek, spočítáme těžiště skupinek a pak spočítáme těžiště těchto těžišť (každé z těchto těžišť ještě vážíme počtem bodů z příslušné skupinky).

Můžeme tedy vzít body po dvojicích – vždy si vezmeme bod a jeho obraz (body, co leží přímo na ose, necháme samostatně). Těžiště každé této dvojice (tj. střed příslušné úsečky) se nachází přesně na ose. Těžiště samostatného bodu je přímo tento bod. Každé takto spočítané těžiště se nachází na ose, tedy i těžiště těchto těžišť – jakkoli vážené – se bude opět nacházet na ose. Tím pádem tam bude ležet i těžiště původních bodů.

Poznamenáme ještě, že těžiště dokážeme spočítat v lineárním čase. Stačí spočítat průměr x-ových a průměr y-ových souřadnic jednotlivých bodů. Výsledkem jsou souřadnice těžiště.

Takto získáme jeden bod osy, musíme ještě přijít na druhý. Jeden způsob je opět zkoušet středy úseček P_0P_k pro ostatní k. Tento způsob je zdánlivě lepší oproti předchozímu v tom, že můžeme poměrně rychle odmítat přímky, které nejsou osami. Protože aby se P_0 zobrazilo na P_k, musí být přímka P_0P_k se středem S kolmá na osu TS (T je těžiště) a navíc musí TS protínat P_0P_k ve středu úsečky P_0P_k. Všechno toto dokážeme zkontrolovat v konstantním čase a rychle tak odmítnout spoustu potenciálních os.

Když však všechny tyto rychlé kontroly uspějí, musíme opět ověřit, jestli je daná přímka skutečně osou. To můžeme provést jedním ze způsobů popsaným v předchozí části.

Nicméně v obecném případě jsme si moc nepomohli. Jako účinný protipříklad se ukáže množina vrcholů pravidelného n-úhelníku sjednocená s množinou bodů rovnostranného trojúhelníku se stejným těžištěm, která způsobí, že celá množina osově symetrická není.

Sami si můžete vyzkoušet, že pokud budou vrcholy z trojúhelníku brány až jako poslední v pořadí, skutečně jsme si, co se rychlosti týká, oproti předchozímu postupu vůbec nepomohli – stále budeme muset zkoušet $\mathcal{O}(N)$ os a žádnou se nám nepodaří odmítnout rychlým způsobem. Každou osu budeme muset ověřit pomalým způsobem, celkově jsme tedy stále na složitosti $\mathcal{O}(N^2)$. Pro jiné přístupy je ještě horší případ, kdy všechny body z trojúhelníku i z n-úhelníku mají od těžiště stejnou vzdálenost.

Zdá se, že najít těžiště nám vlastně vůbec nepomohlo. Kdepak, opak je pravdou. Na následujících řádcích si ukážeme ještě rychlejší řešení, které se právě o těžiště opírá.

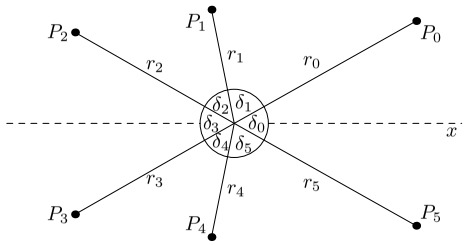
Jde to ještě rychleji

Odrazíme se od souřadnic těžiště. Všechny body posuneme tak, aby těžiště bylo na počátku souřadnicového systému a nadále budeme počítat s těmito upravenými souřadnicemi. Pak víme, že osa souměrnosti, pokud existuje, bude procházet počátkem (tj. těžištěm).

Dále budeme pracovat s takzvanými polárními souřadnicemi, tj. místo x-ové a y-ové souřadnice budeme mít u každého bodu úhel, který svírá x-ová osa s přímkou spojující daný bod s počátkem (těžištěm), a vzdálenost od počátku.

Potom si seřadíme body podle úhlu (prozatím budeme předpokládat, že žádné dva body nemají stejný úhel). Máme tedy u každého bodu P_i úhel φ_i. V tomto setříděném pořadí budeme nadále vrcholy zpracovávat, ale ukáže se, že důležité pro nás bude pamatovat si rozdíl úhlu oproti předchozímu vrcholu. Tedy pro každý vrchol spočítáme $\delta_i = \varphi_i - \varphi_{i-1}$ a pro nultý vrchol $\delta_0 = \varphi_0 + 360° - \varphi_{N-1}$. Všimněte si, že součet přes všechna δ_i nám dá 360°.

Pokud vzdálenost jednotlivých bodů budeme značit pomocí r_i, můžeme teď úhly a vzdálenosti zapsat do řetězce $s = \delta_0 r_0 \delta_1 r_1 \ldots \delta_{N-1} r_{N-1}$. Tedy jednotlivé r_i a δ_i budeme chápat jako jednotlivé znaky řetězce. Všimněte si, že z tohoto řetězce lze zpětně zrekonstruovat původní rozložení bodů (až na rotaci okolo středu, která neovlivňuje osovou souměrnost). Prostě se vždy otočíme o daný úhel δ_i a nakreslíme bod ve vzdálenosti r_i od počátku.



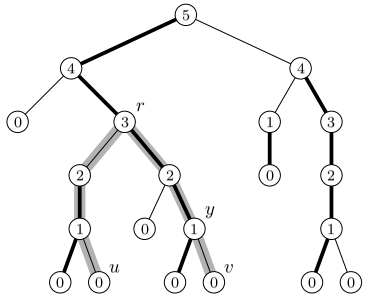
Představme si na chvíli, že osa x je hledanou osou souměrnosti. Uvažujme případ, kdy žádný bod neleží na pravé straně této osy (tedy neexistuje bod s $\varphi_i = 0$). Pak není těžké si představit, že některé úhly a vzdálenosti si musí odpovídat. Konkrétně $r_0 = r_{N-1}$, $\delta_1 = \delta_{N-1}$, $r_1 = r_{N-2}$, $\delta_2 = \delta_{N-2}$ atd. Pro případ, kdy bod leží na pravé straně osy x dostaneme podobné rovnosti, jen trochu posunuté, $\delta_0 = \delta_{N-1}$, $r_1 = r_{N-1}$ atd.

Každopádně si všimněte, že oba případy znamenají, že řetězec s je *skoropalindrom* (palindrom je řetězec, který se stejně čte zepředu i zezadu, tedy že první znak je stejný jako poslední, druhý je stejný jako předposlední atd.). Přesněji řečeno v prvním případě dostaneme palindrom, když za s připíšeme první znak s (tj. δ_0), ve druhém, když před s připíšeme jeho poslední znak (tedy r_{N-1}).

Bohužel nemáme zaručeno, že osou souměrnosti bude osa x. Takže musíme řetězec s vhodně *zrotovat*, tj. opakovaně brát

[4] http://ksp.mff.cuni.cz/viz/kucharky/hesovani

Průběh algoritmu můžeme sledovat na následujícím obrázku. Čísla ve vrcholech jsou jejich indexy, tučně jsou zvýrazněny těžké cesty, šedivě je podbarvena cesta mezi u a v.



Časová složitost funkce *lca* je $\mathcal{O}(\log n)$, neboť při každém stoupání navštíví nejvýše logaritmicky těžkých cest a každou z nich zpracuje v konstantním čase. Získali jsme tedy datovou strukturu pro společné předchůdce, které stačí předvýpočet v čase $\mathcal{O}(n)$ a odpovídá na dotazy v $\mathcal{O}(\log n)$.

Úkol 2 [3b]: Navrhněte algoritmus, který bude pomocí HLD odpovídat na dotazy na *vzdálenost* zadaných dvou vrcholů (tedy počet hran na cestě mezi nimi).

Kudy, kudy cestička?

Nyní se podíváme na to, jak pomocí HLD odpovídat na cestové dotazy. Předvedeme si to na příkladu cestových minim. Dostaneme zadaný strom, jehož hrany budou ohodnoceny celočíselnými *cenami*. Pak nám budou přicházet dotazy na nejlevnější hranu na cestě mezi zadanými vrcholy.

Pro strom si opět spočítáme HLD a každou těžkou cestu navíc opatříme intervalovým stromem, který bude umět odpovídat na intervalová minima cen na této cestě. Ceny těžkých hran si tedy budeme pamatovat v intervalových stromech, zatímco ceny lehkých hran uložíme samostatně. Jeden intervalový strom vybudujeme v čase lineárním v délce jeho těžké cesty a jelikož každý vrchol leží na právě jedné těžké cestě, sestrojíme celou datovou strukturu v čase $\mathcal{O}(n)$.

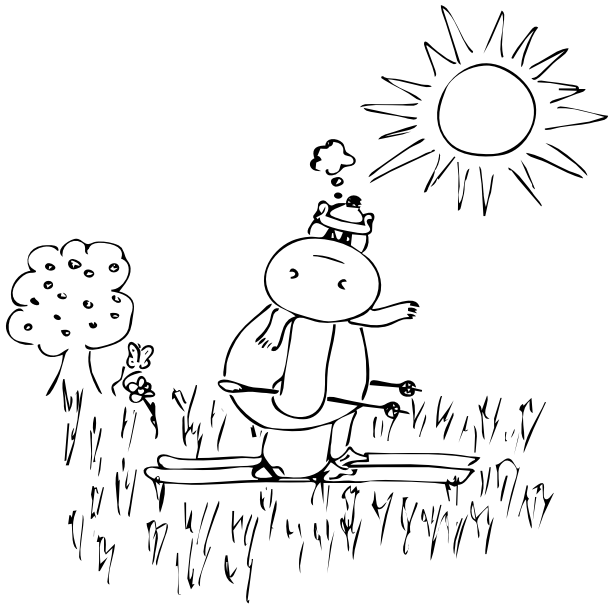
Nyní přijde dotaz na cestu mezi nějakými vrcholy u a v. Spustíme algoritmus pro $lca(u, v)$ a během výpočtu si zapamatujeme, kterými lehkými hranami a kterými částmi těžkých cest jsme prošli. Pak stačí najít nejlevnější z těchto lehkých hran a minim částí těžkých cest. Lehkých hran je $\mathcal{O}(\log n)$ a každou zpracujeme v konstantním čase, těžkých cest je také $\mathcal{O}(\log n)$ a na každé provádíme intervalový dotaz v čase $\mathcal{O}(\log n)$. Dohromady tedy strávíme čas $\mathcal{O}(\log^2 n)$.

Podobně můžeme provádět cestový update. Pořídíme si intervalové stromy s líným vyhodnocováním, které dovedou intervalový update v logaritmickém čase. Kdykoliv třeba budeme chtít zdražit všechny hrany na nějaké cestě o δ, rozložíme update na zdražení $\mathcal{O}(\log n)$ lehkých hran a $\mathcal{O}(\log n)$ intervalových updatů částí těžkých cest v jejich intervalových stromech. Vše opět stihneme v čase $\mathcal{O}(\log^2 n)$.

Úkol 3 [5b]: Zrychlete cestový dotaz na $\mathcal{O}(\log n)$ za předpokladu, že ceny hran se od inicializace struktury již nezmění. Inicializace by měla stále pracovat v lineárním čase.

Úkol 4 [5b]: Mějme graf s ohodnocenými hranami a jeho minimální kostru.[1] Pro každou hranu, která neleží v kostře, spočítejte, o kolik nejvýše můžeme její ohodnocení snížit, aby kostra stále zůstala minimální.

Martin „Medvěd“ Mareš



[1] http://ksp.mff.cuni.cz/viz/kucharky/minimalni-kostry

Recepty z programátorské kuchařky: Geometrie

Geometrické algoritmy

V dnešním díle našeho kuchařkového speciálu se budeme učit vařit geometrické problémy. A co že si představujeme pod pojmem geometrický problém? Trochu analytické geometrie, například zjištění, na které straně orientované přímky bod leží, trocha plotů, neboli konvexních obalů, a obecně mnoho zametání.

V celé kuchařce se omezíme pouze na dvourozměrné problémy, tedy na algoritmy v rovině. Některé postupy se dají zobecnit pro trojrozměrné, a většinou i pro n-rozměrné problémy, ale to je již nad rámec této kuchařky.

Geometrické základy

Nejdříve trocha středoškolské analytické geometrie pro ty, kdo ji ještě neměli. Ostatní mohou tuto sekci přeskočit.

Každý bod v rovině můžeme určit jeho souřadnicemi vůči osám. Nejběžněji se používá takzvaný *kartézský souřadný systém,* tedy dvě na sebe kolmé osy označované jako x-ová osa (vodorovná) a y-ová osa (svislá). Obvykle se uvažuje, že hodnoty na osách rostou směrem doprava (osa x) a směrem nahoru (osa y), my se toho budeme v naší kuchařce držet.

Místo, kde se obě osy protínají, se označuje jako *počátek* soustavy souřadnic. Samotné *souřadnice* bodu zapisujeme jako dvojici čísel, která udávají, o kolik jednotek se musíme posunout ve směru které z os, abychom z počátku dorazili do bodu, kterému souřadnice patří. Počátek má souřadnice $[0, 0]$. Bod se souřadnicemi $[a, b]$ leží na pozici, kterou získáme tak, že se od počátku posuneme o a jednotek ve směru první osy (x-ové) a o b jednotek ve směru druhé osy (y-ové).

Vše ostatní funguje tak, jak jsme se učili při geometrii na základní škole, tedy úsečka je určena dvěma krajními body, obdélník čtyřmi a podobně. Ještě si ale řekneme, co je to vektor, a zavedeme některé další pojmy.

Často potřebujeme popsat vzájemnou polohu dvou bodů. Můžeme například udat jejich vzdálenost a směr (třeba jako úhel vzhledem k ose x). Praktičtější ale bývá říci, o kolik se liší jejich x-ové a y-ové souřadnice. To nám dá dvojici čísel, které říkáme *vektor*.

Pokud například k bodu $[1, 1]$ přičteme vektor $a = (2, -1)$, dostaneme se do bodu $[3, 0]$. Stejně tak, pokud odečteme například bod $[4, 2]$ od bodu $[1, 3]$, tak dostaneme vektor $b = (-3, 1)$ udávající jejich vzájemnou polohu.

Pomocí vektoru a bodu tedy lze určit přímku. Bod nám určí, kam umístit vektor, a vektor nám určí směr přímky z daného bodu. Tomuto vektoru se říká *směrový vektor,* nebo také někdy *směrnice,* dané přímky nebo úsečky.

Samotné vyjádření přímky nebo úsečky poté může být ve dvou tvarech. Prvním z nich je *parametrický tvar*. Základem je nějaký bod $A = [a_x, a_y]$. Od toho se ve směru směrového vektoru $u = (u_x, u_y)$ můžeme pohybovat libovolně a stále budeme na přímce. To nám vede na následující tvar, kde t je libovolný reálný parametr, neboli proměnná, za kterou si můžeme dosadit jakékoliv reálné číslo a vždy nám vyjde bod na přímce. Parametrický tvar vypadá takto:

$$x = a_x + tu_x$$
$$y = a_y + tu_y$$

To samé můžeme vyjádřit i vektorově, tedy $X = A + tu$.

Pro ilustrování funkce parametru, když bude $t = 0$, tak dostaneme výchozí bod přímky. Pokud poté budeme s parametrem hýbat od $-\infty$ do $+\infty$, dostaneme postupně všechny body na přímce.

Druhým způsobem zápisu je *obecný tvar přímky*. K jeho vyjádření budeme potřebovat kolmý vektor ke směrovému vektoru, tomu se také říká *normálový vektor*. V rovině ho získáme jednoduše. Pokud je $v = (v_x, v_y)$ směrnice přímky, tak vektor na něj kolmý má tvar $n = (v_y, -v_x)$. Jako poznámku pro zvídavé můžeme uvést, že *skalární součin* těchto vektorů, tedy součin po složkách ($v \cdot n = ab + b(-a)$), je roven 0, což je také jedna z definic kolmosti.

A jak tedy vypadá slibovaný obecný tvar přímky? Pokud je $n = (a, b)$ normálový vektor přímky, tak obecný tvar přímky je rovnice $ax + by + c = 0$. Dobře, a a b máme, jak ale zjistit c? Normálový vektor určuje směr, kterým přímka povede, ale stále ji můžeme libovolně posouvat. Potřebujeme ještě znát jeden bod, který na naší přímce leží, aby byla určená jednoznačně.

Když dosadíme souřadnice takového bodu do rovnice přímky s neznámou c, získáme tak rovnici pro c, kterou vyřešíme. A máme hotovo, známe hodnoty všech koeficientů v rovnici. Ještě si můžeme všimnout, že pro $c = 0$ prochází přímka počátkem.

Takovéto tvary se hodí nejen pro nějaké zapsání přímek, ale také pro *zjištění jejich průsečíku*. Když hledáme průsečík, hledáme vlastně místo, kde mají obě přímky navzájem stejné x-ové a y-ové souřadnice. A to vede na jednoduché soustavy lineárních rovnic, které jistě již vyřešit umíte.

Ještě si ale zdůrazněme rozdíl úseček oproti přímkám. V případě parametrického tvaru omezujeme velikost parametru t (například $t \in \langle 0, 1 \rangle$) a v případě obecného tvaru omezujeme rozsah jedné ze souřadnic (například $x \in \langle -2, 2 \rangle$). V případě, že bychom chtěli vyjádřit polopřímku, si parametr nebo souřadnici omezíme pouze z jedné strany.

Nakonec si ukážeme jednu základní aplikaci parametru a parametrického vyjádření úsečky. Jak snadno spočítat střed nějaké úsečky AB? V takovém případě není nic jednodušší, než si vzít vektor $B - A$, přenásobit ho parametrem $1/2$ (střed úsečky je v polovině její délky) a přičíst k bodu A. Triviální úpravou pak zjistíme, že střed úsečky můžeme spočítat jako aritmetický průměr jejích krajních bodů:

$$A + \frac{1}{2} \cdot (B - A) = \frac{A + B}{2}$$

Jako příklad na rozkoukání si ukážeme, jak zjistit, na které straně přímky leží bod.

Zjištění polohy bodu vůči přímce

Nejdříve si zavedeme pojem orientovaná přímka. Když budeme mít přímku určenou dvojicí bodů A a B, budeme se na ni dívat, jako kdybychom stáli v prvním bodě (bod A) a dívali se směrem ke druhému (bod B). Pak již máme jasně definovanou pravou a levou stranu a můžeme říci, kde vůči přímce bod leží.

Vezměme si tedy přímku určenou body A a B a bod X. Určíme si vektory $u = X - A$ a $v = B - A$ (s prvky u_x, u_y, respektive v_x, v_y) a porovnáme úhel mezi nimi.

- $(1, -1) \to (0, 1)$
- $(0, -1) \to (-1, 1)$

V obou situacích děláme to, že přičteme dvojici $(-1, +2)$, což je v součtu nula, a tím vlastně posíláme mínus jedničku dál doleva.

Můžeme tedy zahájit převod od nejmenšího řádu zprava a takto si mínus jedničky průběžně eliminovat, nebo si je posílat dál doleva. Máme ale jednu situaci, kterou jsme si nepopsali – co když se nám vedle sebe objeví dvě mínus jedničky?

Na chvíli si povolíme použít i hodnotu -2 a podívejme se, co se nám při přičtení dvojice $(-1, +2)$ může stát:

- $(-1, -1) \to (-2, 1)$
- $(-1, -2) \to (-2, 0)$
- $(0, -2) \to (-1, 0)$
- $(1, -2) \to (0, 0)$

Zkusme si to na čísle 5 zapsaném jako $1, 0, -1, -1$. Při převodu zprava dostaneme postupně $1, 0, -2, 1$, pak $1, -1, 0, 1$ a nakonec $0, 1, 0, 1$, což odpovídá číslu 5.

Převod tedy umíme udělat lineárním průchodem číslem od nejmenšího řádu k největšímu a eliminováním mínus jedniček pomocí přičítání vzoru $(-1, +2)$. Pokud takto převedeme obě čísla, už je snadno porovnáme binárně (průchodem od největšího řádu a hledáním první pozice, kde se liší).

Porovnání odečtením

Pokud vám převod do normální dvojkové soustavy přijde jako nehezký trik, dá se porovnání udělat i odečtením jednoho čísla od druhého. Podle toho, jestli nám výsledek vyjde kladný, nebo záporný (což poznáme podle znaménka nejvyšší jedničky), určíme snadno, které číslo je větší.

Odečítání můžeme dělat klasickým školním postupem od nejmenšího řádu. Pokud nám výsledek odečtení vyjde 1, 0 nebo -1, je vše v pořádku. Pokud nám vyjde menší, než -1, tak musíme udělat převod -1 do vyššího řádu (a k tomu současnému přičteme $+2$, vlastně opět aplikujeme vzor $(-1, +2)$). Pokud nám vyjde naopak větší než 1, přičteme -2 a pošleme převod 1 (tedy použijeme vzor $(+1, -2)$).

Pojďme se podívat na průběh výpočtu třeba čísla $1, -1, -1$, od kterého odečteme číslo $1, 1$ (neboli $1 - 3 = -2$). V prvním kroku nám na poslední pozici výsledku vznikne -2, což převedeme na 0 a do vyššího řádu pošleme -1. Na druhé pozici dostaneme -3 (i s převodem), což převedeme na -1 a do vyššího řádu pošleme -1. A nakonec na nejvyšší pozici dostaneme $1 - 1 = 0$, čímž získáme správný výsledek $0, -1, 0$.

Tento postup zabere také lineární čas vzhledem k velikosti vstupních čísel. Na oba postupy se můžete podívat v přiloženém programu.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/29-3-2.py`

Jirka Setnička

29-3-3 Skřetí věže

Mnoho z vás přišlo s nápadem zkoušet různé přímky a ověřit, jestli se náhodou nejedná o hledanou osu. Všechny možné přímky však určitě vyzkoušet nemůžeme, těch je nekonečně mnoho. Které přímky tedy připadají v úvahu?

Využijeme toho, že každý bod musí mít při osové souměrnosti svůj obraz. Očíslujeme si tedy body postupně P_0, P_1, …, P_{N-1}. Budeme nejprve předpokládat, že bod P_0 se zobrazí na nějaký jiný bod a ne sám na sebe.

Všimněte si, že pokud bychom věděli, na který bod se P_0 zobrazí, je osa souměrnosti jednoznačně určená: musí to být osa úsečky spojující P_0 s jeho obrazem. My samozřejmě nevíme, na který bod se P_0 zobrazí, ale můžeme vyzkoušet všechny možnosti. Tím získáme $N - 1$ přímek, mezi nimiž se určitě hledaná osa nachází (za předpokladu, že nějaká osa existuje a bod P_0 není obrazem sebe sama).

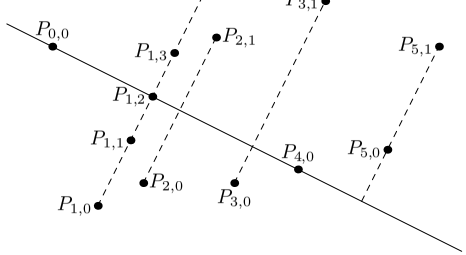
Rozmyslíme si ještě případ, kdy P_0 je sám sobě obrazem a nachází se tedy přímo na ose. Nejjednodušší je vzít místo P_0 bod P_1 a k němu stejným postupem zkoušet body P_2 až P_{N-1}. Takto vyřešíme případy, kdy alespoň jeden z bodů P_0 a P_1 neleží na ose. Pokud by oba ležely na ose, je osou přímka P_0P_1 – tu také přidáme do seznamu kandidátů.

Tímto postupem jsme tedy získali $2N - 2$ přímek, mezi nimiž se určitě osa nachází (existuje-li). Stačí pro každou z přímek ověřit, jestli osou skutečně je, tj. jestli má každý bod svůj obraz.

Nejprve si pro každý bod spočítáme, kam by se při dané ose zobrazil. Pokud bod leží přímo na ose, je sám sobě obrazem. Pokud na ose není, chce to trochu počítání, ale nic náročného. Vezmeme přímku procházející daným bodem, která je kolmá na osu, a spočítáme její průsečík s osou. Tento průsečík se musí nacházet ve středu úsečky spojující daný bod a obraz, takže souřadnice obrazu se už jednoduše dopočítají.

Pro každý bod tedy víme, kam se zobrazí. Teď už stačí zkontrolovat, že v místě obrazu leží nějaký jiný bod – v opačném případě zkoumaná přímka osou není. Pokud bychom při kontrolování obrazu pokaždé procházeli všechny body, bude nám kontrola jednoho obrazu trvat $\mathcal{O}(N)$, kontrola všech obrazů $\mathcal{O}(N^2)$ a prozkoumání všech $2N - 2$ přímek $\mathcal{O}(N^3)$.

Tento postup lze zrychlit vhodným setříděním bodů. Pro každou potenciální osu body setřídíme podle polohy jejich průmětu na osu (to je pata kolmice k ose, na níž leží daný bod). Všimněte si, že tento průmět jsme už spočítali při hledání souřadnic obrazu. Můžeme využít toho, že pokud má být bod P_k obrazem P_ℓ, budou souřadnice jejich průmětů na osu stejné.



Pokud máme tedy takto setříděné body, můžeme je brát postupně. Vždy vezmeme všechny body se stejnými souřadnicemi průmětu a uložíme je do dalšího pole. Toto další pole opět setřídíme, tentokrát podle pozice na přímce, na které se všechny nacházejí (to je nějaká přímka kolmá k ose). Je zřejmé, že první bod v tomto menším seznamu musí být obrazem posledního, druhý předposledního atd. Pokud si

Vzorová řešení třetí série dvacátého devátého ročníku KSP

29-3-1 Verbování

Představme si, že jsme u města Leyfast a procházíme očíslované domy. V každém z domů máme několik možností, jak se rozhodnout. Abychom nalezli nejlepší řešení, můžeme zkusit každou možnou kombinaci rozhodnutí a vybrat tu nejvýhodnější, ale to by trvalo příliš dlouho.

Můžeme také zkusit vybírat další krok *hladově*, neboli vybírat možnost neporušující pravidla, která nám lokálně (pro tento dům) dá nejlepší výsledek. To bude sice rychlé, ale nedostaneme takhle správnou odpověď. Zkuste si to na nějakých vstupech, k rozbití tohoto postupu stačí již ukázkový vstup ze zadání.

Problémem hladového řešení je, že nijak nerespektuje to, že volba v i-tém domě ovlivňuje možné volby v okolních domech. Připomeňme si, co můžeme v domě udělat. Pokud skrz domy půjdeme odzadu, tak můžeme:

- Naverbovat vojáka, pokud jsme tak neučinili v předchozím a neučiníme-li tak v ani následujícím domě.
- Vzít zbraně, pak musíme nutně naverbovat vojáka v následujícím domě.
- Nevzít zbraně ani nenaverbovat vojáka.

Volba, která ovlivňuje výběr v okolních domech, je verbování. Pokud se zkusíme podívat na problém omezený jen na prvních i domů, tak by nás pro další rozhodování mohlo zajímat, jaké nejvyšší bojeschopnosti umíme dosáhnout, pokud si v i-tém domě dovolíme naverbovat vojáka a pokud si zde vojáka nepovolíme naverbovat.

Postavíme si pro to rekurzivní funkci $B(i, (true|false))$, která bude počítat přesně toto. Pokud se nám povede ji spočítat, tak celkovou maximální bojeschopnost získáme zavoláním $B(N, true)$.

Teď si budeme muset sestavit funkci (pro připomenutí, v Z_i je zisk bojeschopnosti při braní zbraní z i-tého domu, v V_i to samé, ale pro verbování z i-tého domu).

- Pro $i \leq 0$ bude mít funkce vždy hodnotu 0.
- $B(i, false)$ bude maximum z:
  - $Z_i + V_{i-1} + B(i-2, false)$ (budeme brát zbraně, což vynutí verbování v $i-1$; lze použít jen pro $i > 1$)
  - $B(i-1, true)$ (nebudeme dělat nic)
- $B(i, true)$ bude maximum z $B(i, false)$ a navíc:
  - $V_i + B(i-1, false)$ (budeme verbovat)

Takovouto funkci lze jednoduše naprogramovat. Horší je exponenciální časová složitost způsobená větvením výpočtu v každém domu. Naštěstí funkce, kterou jsme právě definovali, závisí pouze na dvou parametrech: i a *verbovat*.

Výsledky volání si tak můžeme ukládat do tabulky velikosti $2N$. Při opakovaném zavolání pak stačí vrátit už dříve spočítaný výsledek. Spočítáme tedy nejvýše $2N$ hodnot funkce B, proto dostáváme lineární složitost vzhledem k počtu hodnot na vstupu.

Zbývá domyslet, jak navíc zjistit jeden z plánů verbování nabývající hodnoty $B(N, true)$. Například můžeme spustit znovu trochu modifikovanou funkci počítající B, která bude nyní vracet odkaz na i-tý prvek spojového seznamu, který reprezentuje jeden ze znaků (z,v,-). Všimněme si, že takto použijeme jen N položek seznamu, protože už máme spočítány hodnoty B a v každém kroku tak voláme pouze jednou naši modifikovanou funkci.

Na trochu (ale jen konstantně) elegantnější řešení s nahrazením spojového seznamu řetězcem se můžete podívat do vzorového programu v C++.

Program (C++):
`http://ksp.mff.cuni.cz/viz/29-3-1.cpp`

Marek Černý

29-3-2 Trpasličí závaží

K porovnávání váhy sad závaží a protizávaží se dalo použít několik různých postupů. Jedním z nich bylo převést zápis na takový, který bude obsahovat jen jedničky a nuly, a tento pak porovnat jako se porovnávají binární čísla. Druhou možností je porovnat zápisy v této „rozšířené dvojkové soustavě“ přímo.

Za chvíli si ukážeme obě možnosti, nejdříve ale proveďme pozorování. Podobně jako v klasické dvojkové soustavě má každá pozice dvojnásobnou hodnotu než pozice předchozí. Když si budeme postupně sčítat hodnoty na dalších pozicích (za nějakou pozicí s hodnotou x), tak nejdříve dostaneme polovinu x, z další pozice čtvrtinu (neboli polovinu té zbývající poloviny do x) a tak dále. Každá další pozice nám součet více přiblíží k hodnotě x, ale nikdy ho nepřesáhne. Protože pozic v binárním čísle je konečně mnoho, přiblíží se na jedničku a víc už ne – matematici by řekli:

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

Takže pokud bude nejlevější nenulová pozice čísla zapsaného v této soustavě záporná, tak i kdyby všechny ostatní pozici již byly kladné, dostaneme nejvýše -1 (a tedy celé číslo bude záporné). A naopak, pokud bude kladná, tak číslo bude kladné.

Podle tohoto můžeme udělat první porovnání a pokud mají nejvyšší pozice obou porovnávaných čísel rozdílná znaménka, můžeme rovnou oznámit výsledek a končíme. Dál tedy budeme zabývat jen případy, kdy jsou znaménka na nejvyšších pozicích stejná.

Další triviální pozorování je, že překlopením všech znamének na opačná vlastně jen změníme znaménko celého čísla.

Převod do dvojkové soustavy

Pro jednoduchost budeme popisovat převod pro čísla, jejichž nejvyšší nenulová pozice je kladná (když tak si je podle předchozího pozorování překlopíme a zapamatujeme si, že je číslo vlastně záporné).

Budeme se chtít zbavit všech výskytů -1 v zápisu čísla. Všimněme si, že následující zápisy můžeme bez změny hodnoty převádět:

Pokud jste už měli analytickou geometrii, určitě znáte vzoreček na výpočet úhlu mezi dvěma vektory. Vzoreček má tvar:

$$\cos\alpha = \frac{u \cdot v}{|u||v|}$$

Jeho nevýhodou je, že výpočet inverzní funkce $\cos^{-1}$ trvá dlouho. Je proto lepší použít jiný způsob výpočtu, kde si vystačíme pouze s násobením.

Tím jiným způsobem je výpočet determinantu matice určené těmito vektory. *Matice* je pouze tabulka, kde jsou vektory poskládány pod sebe (ta naše tedy bude velká 2 na 2 políčka).

Determinant matice této velikosti nám udává obsah rovnoběžníku určeného zadanými vektory. A navíc znaménko determinantu nám říká, jestli je úhel mezi vektory (měřený v kladném směru, tedy proti směru hodinových ručiček) menší než π, nebo větší než π.

Kdo se ještě s determinanty nesetkal, může brát následující vzorec pro výpočet determinantu matice dva krát dva jako kouzelnou formuli. Kdo přesto chce zdůvodnění, může si zkusit udělat rozbor všech vzájemných poloh dvou přímek (a jejich směrových vektorů), které mohou nastat. Po chvíli dojdete ke vztahu přesně odpovídajícímu následujícímu vzorečku:

$$d = u_x v_y - u_y v_x.$$

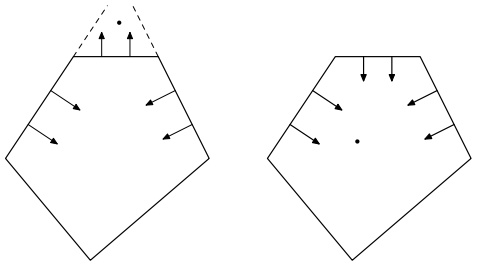
Pokud vyjde d kladné, je bod napravo od přímky, pokud vyjde d záporné, je bod nalevo od přímky, a konečně, pokud vyjde $d = 0$, tak bod leží na přímce.

Bod a konvexní mnohoúhelník

Konvexní mnohoúhelník je takový, který nemá žádný vnitřní úhel větší než 180°. Jinou definicí je, že pokud si zvolíme libovolné dva body v mnohoúhelníku a natáhneme úsečku mezi nimi, nikdy nám nevyleze z mnohoúhelníku ven.

Když už víme, co konvexní mnohoúhelník je, jak zjistíme, jestli nějaký bod leží v něm, nebo ne? Využijeme vlastnosti konvexnosti. Stačí nám jít po hranách na obvodu a zjišťovat, jestli hledaný bod leží na stejné straně všech hran (tedy přímek určených koncovými body hran), nebo neleží.

Pokud bod leží na stejné straně všech hran, nachází se uvnitř mnohoúhelníku. Pokud se ale vůči jen jediné hraně nachází na jiné straně než vůči ostatním, leží bod vně mnohoúhelníku. Nejlépe to vysvětlí obrázek:



Tomuto postupu se také někdy říká test polorovinami. Každá kontrola nám zabere konstantně mnoho času. Časová složitost tohoto postupu je tedy lineární vzhledem k počtu hran, neboli $\mathcal{O}(N)$.

Bod a nekonvexní mnohoúhelník

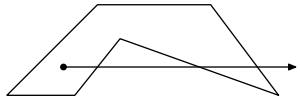
Pro nekonvexní útvary je již postup o něco těžší, protože jak si můžeme všimnout, postup s kontrolováním polohy bodu vůči všem hranám fungovat nebude.

Můžeme si ale na chvíli zahrát na Robina Hooda a ze zkoumaného bodu vystřelit šíp, respektive vést polopřímku. Pěkně se nám bude počítat, pokud polopřímku povedeme rovnoběžně s nějakou z os (třeba ve směru $(1, 0)$). Celé řešení pak spočívá v počítání, kolikrát polopřímka protne hranici mnohoúhelníku.

Můžeme si totiž všimnout, že finálně polopřímka skončí venku a nikdy více již do mnohoúhelníku nevstoupí. A pokaždé, když do mnohoúhelníku vstoupí, musí z něj zase někdy vystoupit. Pokud tedy bod leží venku, začali jsme polopřímku vést zvenku, a tedy bude počet protnutí sudý, pokud bod leží uvnitř, tak bude počet protnutí hranice lichý.

Jediné, na co je potřeba dát pozor, je situace, kdy polopřímka povede přesně skrz nějaký vrchol. V takovém případě se musíme podívat na opačné krajní body hran, které se v tomto vrcholu stýkají. Pokud se obě nachází ve stejné polorovině určené polopřímkou, jen jsme se vrcholu dotkli, ale neprošli jsme skrz (a tedy nepočítáme žádný průsečík). Pokud se ale krajní body hran nachází v opačných polorovinách, znamená to, že jsme ve vrcholu hranici profali a musíme započítat jeden průsečík.

Jako cvičení na rozmyšlenou necháme situaci, kdy se druhý krajní bod jedné z hran nachází na polopřímce.

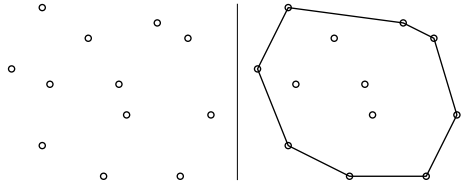


Opět musíme zkontrolovat polopřímku vůči všem hranám, takže časová složitost je znovu $\mathcal{O}(N)$ (i když s o něco vyšší konstantou, protože spočítání průsečíku je více početních operací než jeden test polorovinou).

Konvexní obal a zametání roviny

Podíváme se na jeden z nejznámějších geometrických problémů, totiž hledání konvexního obalu množiny bodů v rovině. *Konvexní obal* je nejmenší konvexní mnohoúhelník, který obsahuje všechny zadané body. Můžeme si všimnout, že všechny vrcholy výsledného mnohoúhelníka musí být nějaké body ze zadané množiny, jinak bychom mohli mnohoúhelník ještě zmenšit (a nebyl by to konvexní obal).

Jako motivaci si představte třeba situaci, že máte sad ovocných stromů a chcete je oplotit co nejkratším plotem. Jak takový plot, nebo obecně obal, nalézt?



Vlevo neobalené body, vpravo obalené.

Ukážeme si postup, kterému se říká *zametání roviny*. Je to trik, který najde uplatnění u mnoha různých geometrických problémů a vyplatí se ho umět.

Základní myšlenka spočívá v tom, že nějakou přímkou, říkejme jí *zametací přímka,* přejedeme přes celou rovinu (od minus nekonečna do plus nekonečna, zleva doprava nebo shora dolů) a vždy, když zametací přímka protne nějaký pro nás zajímavý bod, zpracujeme příslušnou událost. *Událost* je něco významného, co souvisí s příslušným bodem (průsečík přímek, vrchol mnohoúhelníka apod.)

Ale jak jet přímkou postupně od minus nekonečna do plus nekonečna? To není vůbec nutné. Pohyb přímky můžeme začít v nějakém startovním bodě (většinou první událost v setříděné posloupnosti událostí) a ukončit ho po zpracování všech událostí. Navíc nebudeme přímkou pohybovat plynule, ale budeme jí vždy skákat z události na událost (protože mezi událostmi se nic zajímavého neděje).

Vraťme se k našemu problému s konvexním obalem. Jako události budeme brát všechny body, které dostaneme na vstupu. V tomto případě nám žádné nové události v průběhu výpočtu vznikat nebudou, takže frontu událostí můžeme implementovat jako lineární spojový seznam.

Na začátku si body setřídíme podle jejich x-ové souřadnice (zatím budeme pro jednoduchost předpokládat, že žádné dva body nemají stejnou x-ovou souřadnici), začneme je zametací přímkou postupně procházet zleva doprava a budeme si udržovat konvexní obal bodů, které jsme už zpracovali.

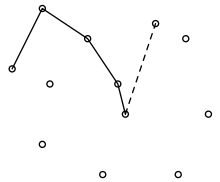
V průběhu výpočtu si budeme konstruovat horní a dolní *obálku*. Obě obálky budou určitě začínat v nejlevějším a končit v nejpravějším bodě (jednoduchým pozorováním lze nahlédnout, že tyto body do obalu určitě patří). A jak už název napovídá, horní obálka půjde vrchem a bude se zatáčet stále doprava, a dolní obálka naopak půjde spodem a bude se stále zatáčet doleva.

Můžeme se pro zjednodušení dohodnout, že nejlevější i nejpravější bod patří do obou obálek. Když pak horní a dolní obálku spojíme, dostaneme konvexní obal.

Horní (respektive dolní) obálku si budeme udržovat jako lineární seznam vrcholů.

Teď si ukážeme, jak bude probíhat jeden krok zpracování. Výpočet se bude provádět samostatně pro horní a dolní obálku, my si ho ukážeme jen pro horní (pro dolní je až na zrcadlení stejný).

Uvažujme, že už máme nějakou část horní obálky, skočili jsme zametací přímkou na další bod a ten teď chceme přidat. Podíváme se na poslední bod v horní obálce a zkontrolujeme úhel poslední hrany v obálce a úsečky mezi posledním bodem obalu a novým bodem.



K tomu můžeme využít například test polorovinami z úvodu kuchařky (pokud nový bod leží vůči poslední hraně horní obálky napravo, je vnitřní úhel konvexní, pokud nalevo, je úhel konkávní). Jestliže se horní obálka zatáčí doprava, máme vyhráno, přidáme nový bod do obálky a můžeme se posunout na další bod. Zajímavější je ale situace, kdy se nám obálka stočí doleva a vznikne konkávní úhel.

Pokud se podíváme na obrázek výše, jasně vidíme, že je potřeba dosavadní poslední bod obálky odebrat a zkusit spojit nově přidávaný bod s předposledním. Odstraníme tedy poslední bod obálky a budeme test opakovat s předposledním bodem.

Takto budeme pokračovat (a případně vyhazovat další body), buď než bude úhel hran konvexní, nebo dokud nám v obálce nezůstane pouze jeden bod (počáteční). Pak nový bod přidáme do obálky a pokračujeme s dalším.

Výše popsaný postup je nejvýhodnější provádět najednou pro obě dvě obálky. Tedy každý bod se pokusíme připojit k horní i dolní obálce a podle toho obě obálky příslušně upravíme.

Proč tento postup funguje? Postupně projdeme všechny body a každý z nich se alespoň na chvíli stane posledním bodem obálky. Při změně obálky se obsažená plocha v konvexním obalu vždy pouze zvětší a žádný bod nám tedy nemůže zůstat mimo konvexní obal.

Ještě jsme zapomněli na případ, kdy úhel není ani konvexní, ani konkávní. V takovém případě se rozhodneme, jestli budeme vrchol tohoto úhlu započítávat mezi vrcholy konvexního obalu. Obvykle se takový vrchol z konvexního obalu vyhazuje, ale nakonec vždycky záleží, k čemu ten konvexní obal vlastně potřebujeme.

Skončíme, až zametací přímkou skočíme na poslední bod a zpracujeme ho. V tomto bodě se nám obálky spojí a dostaneme celý konvexní obal. Teď ale přichází otázka, kolik času nám tento postup zabere?

Může se zdát, že hodně, protože při vyhazování bodů z obálky můžeme postupně vyhodit skoro všechny body. Označme si velikost zadané množiny (počet bodů na vstupu programu) N. Musíme si uvědomit, že každý bod do obálky přidáme pouze jednou a vyhodíme ho také maximálně jednou, tedy časová složitost je lineární k velikosti množiny, čili $\mathcal{O}(N)$, v případě, že již máme setříděný vstup. Pokud ne, musíme ještě přičíst čas potřebný k setřídění bodů, tedy $\mathcal{O}(N \log N)$ při použití nějakého rychlého třídicího algoritmu.[2]

Nakonec ještě zbývá dořešit více bodů se stejnou x-ovou souřadnicí. Pokud to nejsou krajní body, tak nám to v postupu nevadí. Menším problémem je, když to jsou počáteční, nebo koncové body. Problém ale snadno vyřešíme tím, když body seřadíme lexikograficky, tedy nejdříve podle x, a pokud je stejné, pak podle y. To nám jednoznačně určí pořadí bodů a počáteční i koncový bod.

Také si to můžeme představit tak, že rovinu nepatrně natočíme. Tím se určitě konvexní obal (až na natočení) nezmění, nikde nebudou dva body nad sebou a z pohledu algoritmu je to vlastně totéž, jako bychom prošli body v lexikografickém pořadí.

Hledání průsečíků úseček

Nakonec si ukážeme ještě jeden typický zametací problém, který principu zametání využívá o trochu více než konvexní obal. Představte si, že máte v rovině N úseček a chcete najít všechny jejich průsečíky.

Hledáme samozřejmě co nejrychlejší algoritmus vzhledem k N a počtu průsečíků P.

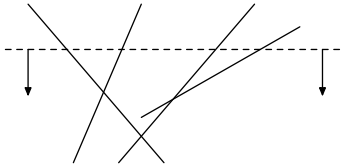
[2] `http://ksp.mff.cuni.cz/viz/kucharky/trideni`

Bystří si již jistě spočítali, že průsečíků může být v extrémním případě až N^2, a tedy nic rychlejšího než zkontrolovat každou úsečku se všemi dalšími v tomto případě není.

Ale takové případy se moc často nestávají, spíše naopak. Uvažujme tedy, že průsečíků je řádově tolik, kolik je úseček a v tom případě je výše popsaný algoritmus již pomalý.

Předpokládejme pro zjednodušení, že v žádném bodě se neprotínají tři a více úseček, žádné dvě úsečky nemají více než jeden společný bod (neleží přes sebe) a žádná úsečka není ani přesně svislá, ani přesně vodorovná. Vyřešení takovýchto případů spočívá v snadných úpravách uvedeného řešení.

Použijeme opět zametací přímku (pro lepší představu teď jdoucí shora dolů, obecně ale nemá směr zametání význam), kterou budeme skákat přes události, a na ní si budeme udržovat aktuální stav. Nazvěme ji třeba *průřezem*. Jak už název napovídá, bude udržovat pořadí úseček, které aktuálně protínají zametací přímku. Jelikož se průřez bude po každé události měnit, budeme pro něj potřebovat šikovnou datovou strukturu. Ale na to se podíváme až potom, co si rozebereme události, ať víme, co od průřezu budeme chtít.



Stejně jako v minulém případě budou mezi událostmi všechny body na vstupu (tedy počáteční i koncové body úseček), vyskytnou se tam ale i další. Pojďme si tedy trochu lépe rozebrat události a akce, které se při nich mají stát:

- *Začátek úsečky:* Přidáme úsečku na správné místo do průřezu, spočítáme případné průsečíky s okolními úsečkami a přidáme je do seznamu událostí.
- *Konec úsečky:* Smažeme úsečku z průřezu, a jelikož se nám dvě okolní úsečky dostanou smazáním této k sobě, musíme ještě spočítat jejich případný průsečík a přidat ho do seznamu událostí.
- *Průsečík:* Započítáme a zapíšeme si průsečík úseček, prohodíme pořadí těchto dvou úseček na průřezu, a jelikož se nám k sobě na průřezu dostaly nové úsečky, musíme spočítat, jestli se někde protínají, a případně průsečíky přidat do seznamu událostí.

Spočítání průsečíků úseček je jednoduchá analytická geometrie. Nejdříve porovnáme jejich směrnice. Pokud jdou od sebe, nemusíme se o nic starat, pokud jdou k sobě, spočítáme, ve kterém bodě se protnou. A když máme tento bod, jenom ověříme, jestli leží na obou úsečkách (neboli že úsečky nekončí ještě před spočítaným průsečíkem).

Když se podíváme na požadavky, hodilo by se nám umět v průřezu rychle vyhledávat, přidávat a mazat, k čemuž nám nejlépe poslouží vyhledávací strom. Ale co za informace si budeme o úsečkách ve vrcholech stromu pamatovat? Jejich aktuální x-ovou pozici (tedy přesněji x-ovou souřadnici bodu této úsečky na úrovni zametací přímky)? Tu bychom museli po každé události u všech úseček přepočítat, budeme na to tedy muset jít chytřeji.

Ve vrcholech stromu si budeme ukládat pouze nějaký rovnicový tvar úsečky (například její obecnou rovnici, nebo směrnici a bod) a vždy, když budeme vyhledávat ve stromu, tak si na základě aktuální y-ové pozice zametací přímky spočítáme v konstantním čase aktuální x-ovou pozici úsečky (jednoduchým doplněním do obecné rovnice) a podle toho se budeme ve vyhledávacím stromu pohybovat.

Máme tedy datovou strukturu pro průřez, ale jak dlouho budou trvat operace s ní? Jelikož v každou chvíli bude ve vyhledávacím stromu maximálně N vrcholů (tedy maximálně tolik, kolik je úseček), budou všechny operace se stromem trvat $\mathcal{O}(\log N)$.

Do seznamu událostí budeme potřebovat také přidávat prvky, takže tentokrát se nám mnohem více hodí použití nějaké haldy. Opět si můžeme uvědomit, že v haldě bude najednou pouze $\mathcal{O}(N)$ prvků (za každou úsečku její začátek a konec a průsečíky úseček vedle sebe na průřezu, tedy maximálně $N-1$ průsečíků), takže operace v ní bude trvat $\mathcal{O}(\log N)$.

Když už máme vybudované datové struktury, podívejme se na to, jak algoritmus poběží. Na začátku přidáme do průřezu první úsečku a do seznamu událostí všechny začátky i konce úseček. Pak již jen postupujeme po událostech, každou událost zpracujeme podle postupu výše a skončíme ve chvíli, kdy nám dojdou všechny události.

Algoritmus funguje správně, jelikož postupně projde přes všechny průsečíky (když jedna úsečka protíná více dalších, tak postupným prohazováním v průřezu se dostanou všechny tyto dvojice vedle sebe a všechny průsečíky přidáme do událostí) a žádný průsečík neprojdeme vícekrát.

Zpracování jakékoliv události nás stojí konstantní množství operací s datovými strukturami, a protože každá z těchto operací stojí maximálně $\mathcal{O}(\log N)$, tak nás zpracování jedné události stojí $\mathcal{O}(\log N)$. Počet událostí je $2N + P$ kde N je počet úseček a P počet průsečíků na výstupu, tedy celková časová složitost je $\mathcal{O}((N+P)\log N)$. Pro pořádek ještě uveďme paměťovou složitost, které je díky použitým datovým strukturám $\mathcal{O}(N)$.

Můžeme si všimnout, že pokud by průsečíků bylo řádově N^2, tak jsme si vlastně pohoršili. Předpokládali jsme ale situaci, kdy je průsečíků řádově stejně jako úseček. V tomto případě je náš algoritmus výrazně rychlejší.

Závěr

Prošli jsme si základní geometrické algoritmy pro rovinné problémy a ukázali jejich základní myšlenky. Různou aplikací a kombinací těchto postupů můžeme řešit většinu lehčích geometrických problémů v rovině, které potkáme.

Jen jako ochutnávku si ještě uvedeme například *Voroného diagramy,* což je rozklad roviny na oblasti, které jsou vždy nejblíž danému bodu (motivací může být například přiřazení obcí na mapě k nejbližšímu krajskému městu). Při jejich konstrukci se také uplatní zametání roviny, ovšem tentokrát již ne přímkou, ale pomocí zametacích parabol.

A jak jsme si uvedli na začátku, mnohé z uvedených postupů lze zobecnit z roviny i do prostoru, ale o tom někdy jindy. Pokud máte zájem o další informace o geometrických algoritmech, tak vás můžeme odkázat na studijní text k přednášce ADS[3] na stránkách Martina Mareše.

Pokud stále nemáte geometrie dost, můžete si ještě zkusit vyhledat pojmy *kombinatorická a výpočetní geometrie.* Dostanete se tak ke spoustě dalších zajímavých materiálů.

Jirka Setnička

[3] `http://mj.ucw.cz/vyuka/ads/43-geom.pdf`