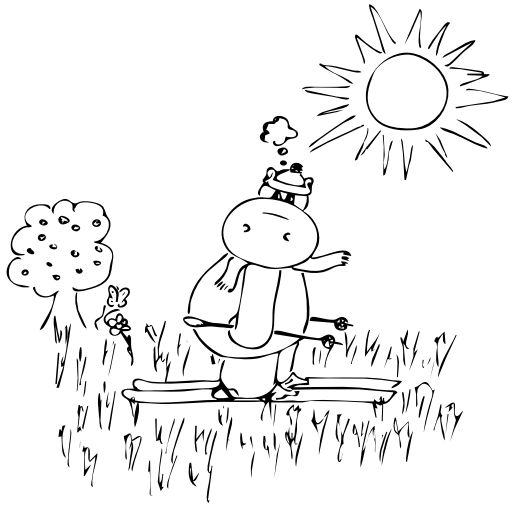
	řešitel	*škola*	*ročník*	*sérií*	*3-1*	*3-2*	*3-3*	*3-4*	*3-5*	*3-6*	*3-7*	*3-8*	*série*	*celkem*
56.	Ondřej Borýsek	GJarošeBO	3	2									0,0	3,3
57.	Adam Husník	GArabskáPH	2	1									0,0	3,0
58.	Jiří Muller	G_Roudnice	3	1									0,0	2,5
59.	David Nápravník	GLitoměřPH	3	1									0,0	2,2
60.–61.	Michael Bausano	GTěš	4	1									0,0	2,0
	Martin Zoula	GNadKavaPH	4	4									0,0	2,0
62.–63.	Peter Matta	G KošiceS	4	1									0,0	1,0
	Michal Rickwood	G_ČTřebová	2	1					1				1,0	1,0



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

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: `0E:D9:B6:E5:6F:B0:51:D9:66:EB:E9:29:E4:58:AB:5F:99:D6:FD:A3`.

Korespondenční Seminář z Programování

28. ročník **KSP** **Březen 2016**

Milí řešitelé a řešitelky!

Zíííív! Sluníčko se směje na obloze a říká cosi o jaru. Organizátoři KSPčka tkví pod peřinou a říkají cosi o zimním spánku. Stromy opatrně vystrkují listy z pupenů. Organizátoři opatrně vytahují zpod polštáře papír s nápady na úlohy. Po nebi se prohánějí mráčky a mávají ptáčkům. Po papíře se prohánějí tužky a sepisují zadání. Vstávat a řešit! A proč? No protože je jaro!

Při příležitosti jarní rovnodennosti (včerejší) vám posíláme zadání čtvrté série, jak stvořené ke čtení pod rozkvetlým stromem.

Připomínáme, že každému řešiteli, který v tomto ročníku z každé série dostane alespoň 5 bodů, darujeme propisku, blok, tužku, a možná i něco navíc.

Termín série: **Pondělí 2. května 2016 v 8:00 SELČ** (CodEx má termín stejný)
Odevzdávání: Přes web na adrese *https://ksp.mff.cuni.cz/submit/*.
Odměna série: **Čokoládu** pošleme každému, kdo **z úloh této série získá alespoň 42 bodů**.

Čtvrtá série dvacátého osmého ročníku KSP

Příběh pěti domů

Utahuji poslední šroub, ještě pro jistotu naposled změřím napětí, balím si nářadí a vyrážím zpátky domů. Už se tu nechci zdržovat ani minutu. Je pátek po deváté večer a mě ještě doma čekají přípravy na víkend a taky jsem dětem slíbil pohádku.

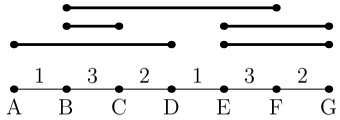
Pracuji jako technik pro jednu telefonní společnost. Práce to není špatná, docela mě i baví a hlavně mě v běžném životě příliš nezatěžuje. Akorát když v mém okruhu nastane náhlý výpadek, jako třeba teď, tak to musím hned jít opravit. Ale co nadělám, aspoň že se to tentokrát stalo přímo v našem bloku, tak jen stačí projít ulicí a jsem doma.

28-4-1 Sledování telefonů — 9 bodů

V ulici stojí v řadě N domů. Každý z nich má jeden telefon a je propojen telefonní linkou s dvěma sousedními domy (jedním v případě krajních). Grafově řečeno tvoří telefonní síť cestu: vrcholy představují domy a hrany spoje. Pokud zavoláme z domu a do b, musí hovor projít přes všechny spoje ležící na cestě mezi a a b.

U každého spoje víme, kolik přes něj za poslední týden prošlo hovorů. Na základě této informace bychom chtěli zjistit, kdo komu volal. To samozřejmě nelze určit jednoznačně, stejný provoz na spojích může vytvořit mnoho různých řešení. Vyberte to s nejmenším celkovým počtem hovorů (pokud je více takových, libovolné z nich).

Například pro $N = 7$ a počty hovorů na spojích postupně 1,3,2,1,3,2 muselo proběhnout nejméně pět hovorů a mohly vypadat například takto:



Tedy proběhly hovory A → D, B → C, B → F a E → G a znovu E → G.

S menším počtem hovorů nelze vytvořit zadané vytížení linek.

Lehčí varianta (za 5 bodů): Řešte za předpokladu, že se počty zachycených hovorů na libovolných dvou sousedních spojích liší maximálně o 10.

V mé ulici stojí pět domů a při takové večerní procházce si aspoň zas jednou udělám obrázek, jak se daří sousedům. Všichni zde žijeme už téměř deset let a k našemu nastěhování se váže společný příběh.

⋆⋆⋆

Bylo mi tehdy 25 let a o tomto roce můžu jednoznačně mluvit jako o roce smůly. Firma, ve které jsem byl zaměstnán, prodělávala, a tak musela čistit. Jako nezkušený mladík jsem to odnesl já a další práci dlouho nemohl sehnat. A aby toho nebylo málo, tak mi navíc o pár týdnů později vykradli byt a stopy zakryli tím, že ho prostě zapálili. Pachatele se dopadnout nepodařilo, vchodové kamery nikoho nezaznamenaly a ani nebyly patrné jakékoliv známky vniknutí. Prostě záhada a pojištěný jsem nebyl. Takže jsem najednou neměl ani práci, ani kde bydlet, a nápad, jak z toho ven, už vůbec ne.

Večer, když jsem svůj žal zapíjel v místním lokále, neměl jsem totiž kam jinam jít, si ke mně přisedl uhlazený muž v obleku a začal se vyptávat, co mě trápí. Stručně jsem mu popsal svou situaci, svěsil hlavu a zhrzele seděl dál. Opravdu jsem neměl náladu se s někým vykecávat.

Muž si na papír načmáral pár poznámek a pak povídal: „Pracuji pro společnost jménem Druhá šance a mám pro vás nabídku. S kolegy zrovna zakládáme nový projekt a vy byste pro nás byl ideální kandidát. Vaší účastí byste vyřešil všechny problémy s prací a bydlením, které vás momentálně sužují. Rozhodnutí je teď na vás. Pokud byste měl zájem se dozvědět více, zavolejte na toto číslo a domluvíme si schůzku. Moc ale neváhejte, naše prostředky jsou omezené a mohli bychom místo vás sehnat někoho jiného.“

Předal mi svou vizitku, rozloučil se a s úsměvem odešel. O nabídce jsem dlouze nepřemýšlel a hned ráno jsem volal, že přijímám. Schůzka byla další pondělí.

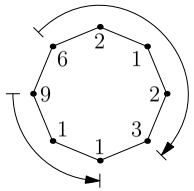
Přišel jsem na zadanou adresu a byl usazen do čekárny. Bylo nás tam celkem pět, tři muži a dvě ženy. Na pohled jsme všichni byli ve věku kolem pětadvaceti let. Z kanceláře vyšel muž, kterého jsem již znal, v ruce držel nějaké papíry a povídal: „Výborně! Tak jste tu všichni. Vybrali jsme

do našeho projektu právě vás pět, protože si myslíme, že jste poslední dobou v životě neměli štěstí a potřebujete dostat svou »druhou šanci«. Tu vám můžeme nabídnout. Já si teď ještě rychle musím něco dopřipravit. Zatím si projděte a podepište tyto dokumenty a trochu se seznamte.“

28-4-2 Podepisování dokumentů — 8 bodů

V kruhu sedí N lidí, kterým potřebujeme předat dokumenty k podepsání. O každém víme přesně, jak dlouho mu bude trvat, než dokumenty zvládne projít. K dispozici máme dvě sady dokumentů, které dáme dvěma lidem, kteří v kruhu sedí vedle sebe. Ti pak dokumenty pošlou dále po kruhu. Navrhněte algoritmus, který zjistí, kterým dvěma sousedům dokumenty podat, aby celkový čas procházení dokumentů všech lidí byl co nejmenší.

Například pro následující skupinku (čísla udávají dobu prohlížení v minutách):



je jedno z možných řešení naznačené šipkami (dokumenty dáme na začátku osobám s časy 6 a 9). Celkový čas bude 14 minut. Rozmyslete si, že lépe to nejde.

První žena byla bruneta oblečená v červených šatech a botách na vysokých podpatcích. Svou image měla doplněnou o černý módní klobouk, zlaté náušnice a náhrdélník. Nebyla vyloženě krásná, ale jak se takto vyšvihla, tak se rozhodně bylo na co koukat. Manžel se s ní prý rozvedl, protože dle jeho názoru zbytečně utrácela za nesmysly a sama toho moc nevydělala. Nakonec zmínila, že její manžel stejně byl jen takový blbeček, který ji nedokázal docenit, že si ji ani nezasloužil a že si určitě brzy najde lepšího.

Druhým byl pohublý muž, který pořád něco ťukal do tabletu a moc se s námi nechtěl bavit. Pracovával jako programátor, pak se ale nepohodl s nadřízenými, když chtěl všechny pracovní dny využívat home office a komunikovat pouze po chatu. S rodiči si také nerozumněl. Neměli totiž pochopení pro hraní online počítačových her a stravování pomocí objednávek přes internet. Prostě takový ajťák.

Třetí osobou byla rozcuchaná potetovaná slečna, oblečená v hipsterském stylu a vlasy obarvenými na tmavě zelenou. Dle svých slov žije pohodový život, ve kterém jí toho moc nechybí. Stálou práci nemá, ale když potřebuje, dopomáhá si brigádami, většinou roznášením letáků. Od rodičů odešla v sedmnácti, protože prý měli příliš oldschoolový pohled na svět a zbytečně ji omezovali. Sedí tu s námi, protože dostala nabídku a nové příležitosti neodmítá.

Čtvrtým přítomným byl muž, takový trochu podivín. Měl vystudovaná práva, sociologii a nyní začal chodit na ekonomku. V životě se dostal do slepé uličky, když přesáhl věk dvaceti šesti let a za svá studia musel začít platit. Neměl totiž z čeho. Navíc to byl vitarián, což také nevycházelo nejlevněji. Ve svém životě dodržoval přísná pravidla ohledně stravování, denního režimu, spánkového režimu a tak vůbec.

Pátý jsem byl já, muž s vyhořelým bytem, bez práce a s nulovou představou, jak tuto zoufalou situaci řešit. Jinak ale obyčejný chlap s ne příliš náročnou představou o životě. Chtěl jsem hlavně založit rodinu, o tu se postarat, najít si pár přátel a s těmi občas něco podniknout či v zimě vyjet na hory. Hlavně aby bylo pořád co dělat a za čím jít.

Po vyslechnutí příběhů všech přítomných bylo naprosto zjevné, že jsme všichni navzájem diametrálně odlišní a máme naprosto jinak seřazené své životní hodnoty.

28-4-3 Řazení životních hodnot — 10 bodů

Máme zadanou množinu X s N navzájem různými celými čísly a posloupnost $R_1, \ldots, R_{N-1}$ operátorů menší než (<) a větší než (>). Najděte uspořádání $x_1, \ldots, x_N$ čísel z množiny X takové, aby platilo

$$x_1 R_1 x_2 R_2 \cdots R_{N-1} x_N.$$

Formát vstupu: Na prvním řádku dostanete číslo N. Na druhém řádku najdete N čísel tvořících množinu X (v neurčeném pořadí) a na třetím $N-1$ znaků < nebo > bez mezer.

Formát výstupu: Jeden řádek obsahující čísla z X v takovém pořadí, že splňují zadané relace. Správných řešení může být víc, vypište libovolné z nich.

Ukázkový vstup:	*Ukázkový výstup:*

`6`
`42 2 3 8 1 5`
`>><>< `
	`8 5 2 3 1 42`

Výstup je správný, protože platí: $8 > 5 > 2 < 3 > 1 < 42$.

Tato úloha je praktická a řeší se ve vyhodnocovacím systému CodEx.[1] Přesný formát vstupu a výstupu, povolené jazyky a další technické informace jsou uvedeny v CodExu přímo u úlohy.

Po chvíli z kanceláře znova vyšel ulízlý muž v obleku. Vybral od nás podepsané dokumenty a povídal: „Jmenuji se Dalimír a mým úkolem je uvést vás do celého projektu a postarat se o všechny formality s ním spojené. Všem pěti z vás nějakým způsobem pomůžeme vyřešit váš aktuální problém s bydlením a financemi…“

„Heeej! Já nemám žádnej problém a určitě nepotřebuju ničí pomoc!“ ozvala se zelenovlasá hipsterka.

Dalimír ale pokračoval dál, jako by ji vůbec neslyšel: „Na kraji města v ulici Nádvorní jsme postavili pět domů a každý z nich se chystáme darovat jednomu z vás spolu ještě s dalšími výhodami. Jste připraveni se na ně jít podívat?“

Nezmohl jsem se ani na slovo. Tak úžasnou zprávu jsem vůbec nečekal. Ostatní na tom byli podobně. Vyhublý ajťák dokonce na chvíli přestal koukat do tabletu a dámě v klobouku v obličeji na okamžik zableskl výraz pokory a věku.

Po chvíli ticha se ozval věčný student: „A nebudeme mít pak problémy se zdaněním? Prošly ty domy oficiální kolaudací? A je legální v nich už bydlet?“

„Nebojte, o to vše jsme se postarali,“ uklidňoval jej Dalimír.

„Tak vyrazíme!“ pokračoval.

Dojeli jsme na místo a začali procházet ulicí. Dalimír nám vše pečlivě popisoval. Kde najdeme zastávky, kterým směrem je nejbližší obchod, a tak podobně. Až jsme došli k prvnímu, obrovskému domu.

„Nyní mi dovolte, abych vám představil první dům. Jedná se o tuto vilu se zahradou, bazénem a dvěma garážemi! Garáž samozřejmě není prázdná, ale najdete v ní nejnovější model auta Subaru XV Crosstrek. Samotný dům se pyšní

[1] `http://ksp.mff.cuni.cz/viz/codex`

Výsledková listina třetí série dvacátého osmého ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*3-1*	*3-2*	*3-3*	*3-4*	*3-5*	*3-6*	*3-7*	*3-8*	*série*	*celkem*
0.					7	9	10	11	8	12	13	15	61,0	178,0
1.	Václav Volhejn	GKepleraPH	3	18	8	8,5	10	11	8	12	13	15	61,0	178,0
2.	Jakub Pelc	G UherBrod	2	3	7				8		13	9	37,0	140,7
3.	Jan Bouček	GKepleraPH	3	7			10		8	12		12	43,2	140,6
4.	Pavel Turek	GTomkovaOL	3	3	10	7		7	6	10	7		49,0	136,5
5.	Richard Hladík	GOAMarLaz	3	18		9	10	11	8	12	7		50,0	134,0
6.	Jiří Sejkora	GVoděraPH	4	6		9		10	6		11	12	50,4	130,8
7.	Stanislav Lukeš	GPísnickáPH	3	9	1	5							6,4	81,2
8.	Leonard Mentzl	GŘíč	3	2									0,0	76,9
9.	Jonáš Fiala	GJungmanLT	3	3					6		13		19,0	74,5
10.	Michal Töpfer	G DrJPekMB	3	8				3		11		10	24,0	72,5
11.	Josef Gajdůšek	SŠKKamPard	3	3					2				2,0	67,9
12.	Vojtěch Lukeš	GPikaPL	4	2									0,0	62,0
13.	Václav Pavlíček	ZŠ Ždírec nD	0	3					1				1,0	47,7
14.	Petr Chmel	G_Kralupy	3	3					3				3,0	46,3
15.	Lukáš Vlček	GMikulášPL	2	3					5				5,0	44,1
16.	Jakub Tětek	Dollar Ac	2	8			10			12			22,0	44,0
17.	Daniel Herman	GŠKo	3	2	4	9	7	1	3	1	0		29,2	42,3
18.	Miroslav Hrabal	GTomkovaOL	2	2	2				6				9,0	38,7
19.	Jakub Dobrý	GMikulášPL	2	3					0				0,0	38,5
20.	Ján Chudý	GŽilina	4	1									0,0	37,7
21.	Pavel Turinský	G Brandýs	3	7									0,0	35,9
22.	Ondrej Pudiš	GŽilina	4	1									0,0	34,2
23.	Václav Šraier	GČeskoliPH	3	7									0,0	32,9
24.	Přemysl Šťastný	GŽamberk	3	10					8				8,0	30,0
25.	Jakub Lukeš	GNAlejíPH	3	4					0				0,0	29,2
26.	Lukáš Rozsypal	GÚstavníPH	3	2					0		5		8,7	28,8
27.	Vanda Hendrychová	GHeyrovPH	4	1									0,0	28,7
28.	Vladimír Bartovic	G AM Trnava	4	2									0,0	28,0
29.	Roman Beňo	GJHroncaBA	3	2									0,0	25,6
30.	Jiří Vozár	G UherBrod	4	7									0,0	25,2
31.	Michal Kodad	ZŠJílovsPH	0	3		2							3,6	24,3
32.	Zdenko Čepan	GPartizans	3	2									0,0	22,6
33.	Michal Jireš	GRNK	1	1									0,0	22,5
34.	David Ucháč	eduSOŠ PA	3	2					1				1,0	21,1
35.	Marco Souza de Joode	GNadŠtolPH	−1	2									0,0	20,8
36.	Sam Friedlaender	GKepleraPH	−1	2									0,0	20,3
37.	Jiří Löffelmann	GLitoměřPH	2	1									0,0	20,0
38.	Jan Pokorný	G Bučovice	4	8									0,0	17,7
39.	Alexej Popovič	SlovanGOL	4	2									0,0	16,1
40.	Filip Geib	G MMH LM	2	2		1			1				3,3	15,8
41.	Alena Tesařová	GVídeňskBO	4	2		1			0				2,3	15,5
42.	Jan Kaifer	GČesBrod	0	2	6		4			2,5			15,4	15,4
43.	Petr Gebauer	GMělník	2	1									0,0	10,6
44.–47.	Jan Gocník	GJŠkodyPŘ	4	4									0,0	10,0
	Jan Priessnitz	GJarošeBO	3	1									0,0	10,0
	Filip Šohajek	GUHradiště	−1	1									0,0	10,0
	David Žáček	GZborovPH	3	2									0,0	10,0
48.	Jan Neumann	GNAlejíPH	2	1									0,0	9,0
49.–53.	David Blažek	SPŠÚžlabPH	3	1									0,0	8,0
	František Kmječ	G Brandýs	0	1									0,0	8,0
	Lucie Kubíčková	GFXŠaldyLI	2	1									0,0	8,0
	Antonín Prantl	G Strakon	3	1									0,0	8,0
	Zuzana Svobodová	G FrýdlNOs	4	2									0,0	8,0
54.	Jakub Matěna	GČeskoliPH	4	4									0,0	4,3
55.	Lukáš Mičan	GČeskáČB	2	1									0,0	4,0

laci. Výpočet je ale stále pomalý a je jen malá šance, že tak hladové řešení překonáme. Hlavně bychom si museli trochu déle počkat.

Co uděláme my, je, že znovu použijeme mravenčí algoritmus, ale omezíme množství měst, ze kterých budeme vybírat následníka. Jelikož se pohybujeme v rovině, tak se nám takřka nikdy nevyplatí jít moc daleko. Další město můžeme tedy vybírat například z nejbližších K, kde K bude rozumně malé. Třeba 10.

Tím se algoritmus dost zrychlí. Už si můžeme dovolit použít více mravenců. Použijeme jich aspoň 100 až 1 000, aby v jedné iteraci zvládli označit rozumné množství hran feromony a další iterace tak měly větší variabilitu.

Pokud takové řešení s parametry $\alpha = 2$, $\beta = 2$, $\rho = 0.5$ a pro 1 000 mravenců necháme běžet 500 iterací (několik hodin), dostaneme se na řešení o hodnotě 20 550 678, což není o moc lepší než hladový algoritmus, ale aspoň něco. Tento algoritmus můžete vidět ve zdrojovém kódu.

Já pak ještě experimentoval s tím, že feromony zanechává jen W nejlepších mravenců z jedné iterace, ale dostal jsem se tak jen na víceméně stejně dobrý výsledek.

Program (C++):
`http://ksp.mff.cuni.cz/viz/28-3-8.cpp`

Další možné postupy

Evoluční a jim podobné algoritmy jsou většinou navržené obecně, aby se jimi dala řešit širší škála problémů. Proto nevyužívají charakteristiky žádného konkrétního z nich. Na jednu stranu je výhoda, že se dají prakticky vždy nějakým způsobem použít. Na druhou stranu ale mají nevýhodu v tom, že jsou málokdy nejlepší. Když se totiž zaměříme na konkrétní problém a navrhneme algoritmus přímo vhodný pro něj, tak obvykle dosáhneme vyššího úspěchu.

Kromě toho existují ještě heuristiky, které se snaží vylepšovat nějaké již existující řešení. Pro takové heuristiky se hodí jako počáteční řešení zvolit buď výsledek nějakého hladového algoritmu nebo právě výsledek některého z evolučních algoritmů.

Pro náš problém se dají použít například následující heuristiky:

- Prohodíme 2 hrany. Pokud se řešení zlepší, necháme je tak, jinak je vrátíme zpátky. (Díky této heuristice se postupně zbavujeme i křížení hran.)
- Prohodíme 3 hrany. Pakliže už nefunguje prohazování dvou hran, můžeme začít prohazovat 3, což nám dává další možnosti.
- Přeházení K po sobě jdoucích měst. Také může pomoci.

U všech těchto heuristik nové řešení přijmeme pouze pokud nám heuristika pomohla. Tyto heuristiky uvádíme jen pro příklad. Ty nijak nesouvisí s evolučními algoritmy. Pomocí nich se řešení dalo vylepšit až na 18 000 000.

Karel Tesař

dvěma kuchyněmi, třemi koupelnami a ložnice pro hosty je samozřejmostí. Interiér je navíc zkrášlen řadou obrazů historického i moderního umění. . . “ představoval Dalimír.

28-4-4 Podivuhodný obraz — 12 bodů

V domě visí podivuhodný obraz. Je na něm nakreslených N bodů, které jsou spojeny dohromady M čarami. Celý obraz je černo-červený a má zajímavou vlastnost. Pokud z bodu vedou alespoň 2 čáry, některá z nich je černá a některá červená.

Co kdyby ale obraz vypadal jinak? Šel by pořád takto nakreslit? Vymyslete algoritmus, který na vstupu dostane neorientovaný graf a obarví jeho hrany dvěma barvami tak, že každý vrchol se dotýká hran obou barev (případně zjistěte, že to nejde). Pozor, vstupní graf nemusí být souvislý.

Lehčí varianta (za 8 bodů): Řešte za předpokladu, že všechny vrcholy mají sudé stupně.

Formát vstupu: Na prvním řádku dostanete dvě celá čísla N a M udávající počet vrcholů a hran. Každý z následujících řádků popisuje jednu hranu pomocí dvojice čísel vrcholů (vrcholy číslujeme od nuly).

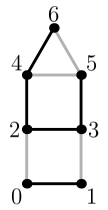
Formát výstupu: Vypište celkem M řádků popisujících barvy hran ve stejném pořadí, jako byly na vstupu. Barva každé hrany je buď 0 (černá), nebo 1 (červená, na obrázku šedá). Pokud graf nejde správně obarvit, vypište jediný řádek obsahující číslo -1.

Ukázkový vstup:

```
7 9
0 1
0 2
1 3
2 3
2 4
3 5
4 5
4 6
5 6
```

Ukázkový výstup:

```
0
1
1
0
0
0
1
0
1
```



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

„To zní úplně jako sen!“ vykřikla nadšením dáma v klobouku.

„A to ještě není všechno,“ pokračoval Dalimír, „starat se o takový dům není sranda. Proto také od nás máte k dispozici komorníka, uklízečku, zahradníka, osobního šoféra a hlídače k vašim službám. Součástí je také finanční dar, který by měl stačit na veškerou údržbu na alespoň dalších pět až deset let.“

„A tento dům se všemi jeho výhodami jsme připravili. . . Chvíle napětí. . . Pro vás, slečno!“ ukázal na dámu v klobouku.

„To. . . to je naprosto úžasné,“ rozpačitě děkovala dáma, „určitě se budu snažit vás nezklamat a budu domu dělat dobré jméno. Určitě vám dokážu, že si jej zasloužím.“

„Ano, ano. Uvnitř na vás čeká komorník, který Vám vše ukáže,“ ukazuje na vchod Dalimír, „tak se běžte seznámit a my ostatní budeme pokračovat dál.“

Tak to je teda hustý, říkám si pro sebe. Jestli všechny domy budou takový, tak jsem totálně za vodou.

„Nyní přicházíme ke druhému domu,“ znova mluví Dalimír, „tento dům se pyšní těmi nejmodernějšími technologiemi, které doposud lidstvo vyvinulo. Nejen že budete mít k dispozici ty nejlepší počítače a další hračky, ale ještě k tomu vám každý rok budeme dodávat nové. Společnost vám budou dělat domácí roboti, kteří za vás vysají, automaticky ovládaní droni, kteří doletí ke dveřím pro poštovní zásilky, vyzvednou krabici s jídlem, a čeká na vás ještě fůra dalších technologických vychytávek. Navíc je všechno centrálně ovladatelné a synchronizované. Když na to přijde, tak celý den nebudete muset vylézt z postele, a přitom se o všechno zvládnete postarat. Vysokorychlostní internet a pokrytí Wi-Fi v celém areálu je samozřejmostí.“

„Tak to už se nemůžu dočkat, až se nastěhuju, to je přesně pro mě,“ ušklíbla se ironicky hipsterka. Všichni už jsme tušili, kdo bude novým vlastníkem domu. Vyhublému ajťákovi začaly svítit oči a pozorně poslouchal každé slovo jako do té doby nikdy.

Vtom se zablesklo a před domem se objevil urostlý muž s mečem v ruce. Na jeho rukou a nohou probleskovaly pruhy modrého světla. Pár vteřin tam stál, rozhlížel se, ale pak se zase zablesklo a byl fuč.

„A teď jste mohli vidět. . . To byla asi. . . ukázka automatického zastrašování pomocí holografické stráže,“ pokračoval Dalimír, „není to ale jediný bezpečnostní prvek, který zde je. Celý areál je pokrytý kamerami, které zevnitř můžete sledovat z libovolného přístroje a vstoupit můžete jen po identifikaci svou oční duhovkou.“

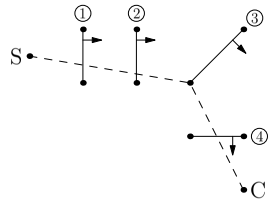
„Jak už asi tušíte, tak tento dům je přímo dělaný pro vás, pane,“ ukázal na ajťáka a ten se zaradoval: „Já. . . nemůžu se dočkat až si všechny tyhle super věci vyzkouším a pořádně nakonfiguruji! Tenhle dům je to nejlepší, co jsem zatím ve svém životě viděl!“

„Běžte ke vchodu. Tam vám kolega naskenuje duhovku a provede další inicializaci. My budeme pokračovat dál,“ povídá Dalimír.

„Nyní přicházíme ke třetímu domu. V tom se určitě nikdy nudit nebudete! Uvnitř najdete bowlingové dráhy, kulečník, saunu, minikino s vířivkou, venkovní párty bazén, minigolfové hřiště a kroketové hřiště. Ať budete chtít podniknout cokoliv, nikdy nebudete muset chodit moc daleko. . . “

28-4-5 Hra kroket — 11 bodů

Ve hře kroket přesouváme míček po hřišti pomocí několika úderů holí. Míček považujme za bod nulové velikosti, který se po úderu pohybuje po úsečce. Celá trasa míčku tedy tvoří lomenou čáru. Na hřišti jsou také rozmístěné branky, které si budeme představovat jako úsečky. Cílem hry je, aby míček projel všechny branky, a to v předepsaném pořadí. Navíc každá branka má určený směr, kterým je třeba ji projet.



Na vstupu dostanete startovní a cílovou pozici míčku a seznam branek (každá je určena svými krajními body). Najděte nejkratší trasu míčku, která začně na startu, skončí v cílovém bodě a projede všechny branky správným směrem a ve správném pořadí.

Jednoduchý příklad vstupu naleznete na obrázku. Plnou čarou jsou značeny branky, šipky a čísla u nich udávají očekávané pořadí a směr průjezdu. Čárkovaná čára ukazuje správnou nejkratší trasu míčku.

Ještě dodáme, že je povoleno projet stejnou branku vícekrát, stačí, aby jeden z průjezdů dodržel správné pořadí a směr. Například je korektní projet branky v pořadí 1, 2, 4, 3, 4, protože když nepočítáme první průjezd čtvrtou brankou (který tím pádem může být i špatným směrem), dostaneme správné pořadí.

Lehčí varianta (za 7 bodů): Všechny branky jsou svislé, projíždějí se zleva doprava a jejich x-ové souřadnice tvoří rostoucí posloupnost (nejdřív je třeba projet nejlevější branku, potom druhou nejlevější, ..., až nakonec nejpravější). Start je nalevo od všech branek a cíl napravo. Jinými slovy, stačí hrát jen zleva doprava.

„... Součástí je také bar a taneční klub, který můžete použít jak pro soukromé účely, tak pro pořádání veřejných akcí. Personál včetně DJe od nás samozřejmě máte k dispozici. Pak vzadu můžete provozovat vlastní tetovací salón, vedle něhož najdete skleník, kde roste takové zelené překvapení," dokončuje vyčerpávající popis Dalimír.

„Tak počkat! Co myslíte tím zeleným překvapením? A vůbec mám takový pocit, že pořádání párty v obytné čtvrti je zakázané. Ty určitě budou narušovat noční klid..." začal jej okamžitě napomínat student.

„Tak tím zeleným překvapením jsem samozřejmě myslel bio okurky a domácí zelí," vysvětluje Dalimír a při tom nepatrně mrkne na hipsterku, „co jste myslel vy? A o rušení nočního klidu nemějte obavy. Celý areál je vybaven moderním odhlučňovacím systémem, takže s tím by neměl být absolutně žádný problém."

Dalimír pokračuje: „A pro koho máme přichystaný tento dům? Ano, jste to vy, zelenovlasá dívko!"

Hipsterka radostně naběhla dovnitř a my ostatní pokračovali dál, k domu číslo čtyři.

„Nebudeme to dál napínat. Další dům je určen zde pro našeho věčného studenta a můžeme jej klidně nazývat Domem vědomostí," představuje čtvrtý dům Dalimír.

„Součástí tohoto domu je obrovská knihovna, ve které jsou k dispozici všechny studijní materiály, odborné články a literatura, které v posledních 50 letech lidstvo vyprodukovalo. K tomu je dodána přehledová brožura k bezproblémové orientaci v celém archivu. Navíc jsme Vám zařídili licenci pro sledování přednášek z pěti největších univerzit na světě. Kdyby Vám přeci jen něco chybělo, nebo vyšel nějaký nový článek, který byste chtěl, neváhejte nám říct a my jej pro vás obstaráme. Dále kolem domu máte dost prostoru pro pěstování ovoce, zeleniny, bylinek a obilovin, které byste k životu mohl potřebovat. Dodáme Vám ještě několik profesionálních ekologických zahradníků, kteří s Vámi na pěstování budou spolupracovat. Společně toho pravděpodobně vypěstujete více, než budete potřebovat, a jelikož pozemek patří Vám, tak vás přebytek nejspíš i uživí. A málem bych zapomněl, to auto, které tady stojí, je Váš nový elektromobil, šetrný k životnímu prostředí," dokončuje Dalimír.

„Tak to je naprosto boží! Tohle je doslova a do písmene můj sen! Živý, tady, přede mnou a ještě k tomu patří mně! Vůbec nevím, jak Vám mám poděkovat," povídá student, ale není na něm vidět žádná výraznější emoce.

„Jen mi neděkujte a raději se běžte zabydlet," posílá jej Dalimír dovnitř.

„Tak teď ještě vyřešit vás," říká směrem ke mně, „dům pro vás je ještě kousek dál po cestě."

Přicházíme ke znatelně menšímu domu, než byly všechny předchozí, který na první pohled vypadá naprosto obyčejně.

Dalimír začíná představovat: „Pro vás máme postavený klasický rodinný dům čtyři plus jedna. Oproti běžným tři plus jedna má navíc druhou ložnici, která se na takto odlehlém místě určitě bude hodit a případně se dá přestavět na dětský pokoj. K domu patří menší zahrada vhodná pro letní posezení venku. Jinak je to klasický byt. Kuchyň je vybavena sporákem a myčkou, obývák televizí a DVD. Také pro vás máme Volkswagena. Je sice trochu ojetý, ale zase málo žere ... A kdybyste měl zájem, tak vás doporučíme do jedné telefonní společnosti, kde zrovna shání zdatného technika. To jste vystudoval, je to tak?"

„Ano, je to tak," odpovídám.

„Tak to je asi všechno. Jak se Vám to líbí?" dokončuje a ptá se Dalimír. Já na to rozpačitě odpovídám: „No... ano, líbí... Samozřejmě, že líbí. Ale... je tu jedna věc, která mi vrtá hlavou... Ostatní dostali výrazně větší a luxusnější domy a v podstatě se jim dneska splnily všechny sny, zatímco pro mě máte »jen« takovýhle »obyčejný dům«?"

Dalimír hned reaguje: „Rozumím. Tento dar od nás nemusíte přijmout. Pořád máte možnost jej odmítnout. Jsem si celkem jistý, že získáním tohoto domu budete ve značně lepší situaci, než jste byl před týdnem..."

„Ano, ano. Já si určitě nechci stěžovat. Jsem samozřejmě nadšen a vaši nabídku přijímám všemi deseti! Jen mě trochu zaskočil ten nepoměr vůči ostatním. Ale už raději mlčím," vymlouvám se.

„Tak to bude nejlepší," říká Dalimír, „já bych si dokonce vsadil, že se svým novým obydlím budete spokojený. A kdo ví, jestli ne dokonce víc než ostatní? Tak se běžte dovnitř trochu porozhlédnout, já teď ještě musím dořešit pár věcí a trochu si to všechno utřídit."

28-4-6 Mediánové třídění — 9 bodů

Máme dohromady N čísel k setřídění. Také máme takovou speciální krabičku. Té na začátku zadáme fixní liché K a pak do ní začneme postupně vkládat nějaké prvky. Po každém vloženém prvku nám krabička řekne medián z posledních K vložených prvků, tedy takový prvek, který by se po jejich setřídění nacházel na prostředním místě. Krabička začne vracet mediány teprve poté, co do ní vložíme alespoň K prvků.

Například pokud pro $K = 3$ do krabičky vložíme postupně prvky 4, 7, 1, 2, 3, 5, řekne nám hodnoty $\emptyset$, $\emptyset$, 4, 2, 2, 3 (kde $\emptyset$ značí prázdný výsledek, když krabička obsahuje méně než K prvků). Například první dvojka je mediánem posloupnosti 7, 1, 2.

Vymyslete, jak s pomocí takovéto krabičky setřídit čísla v lineárním čase. K si můžete zvolit libovolné, klidně pro každý vstup jiné. Předpokládejte, že přidání prvku do krabičky trvá konstantní čas.

Po tom dni a podepsání všech smluv jsem Dalimíra už nikdy neviděl a o Druhé šanci nikdy neslyšel.

* * *

Od té doby se u nás v ulici ledacos změnilo. Teď když procházím kolem prvního domu, tak z něj věčně slyším křik a hádky. Dáma už se za tu dobu stihá potřetí rozvádět. Nikdo pro ní není dost dobrý. Co na tom, že vlastní úspěšný salón krásy a večeří kaviár? Co na tom, že vyhrává všechny soudní spory o majetek a děti, když žádnému z nich ne-

rovin. Oblasti tedy mají tvar konvexních mnohoúhelníků, případně na jedné straně otevřených do nekonečna.

Vyzkoušíme si to na tomto lese:

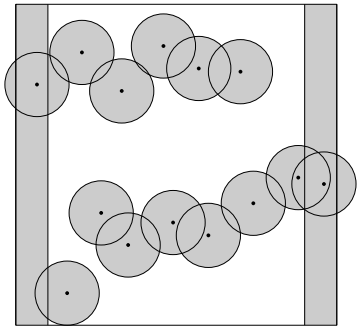
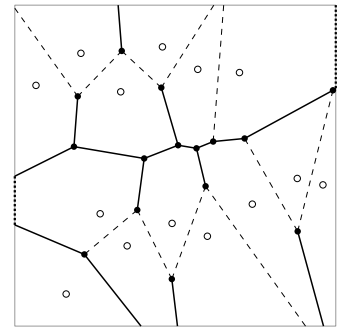


Diagram pro něj vyjde následovně. (Plné i čárkované čáry jsou hranice oblastí, brzy uvidíme, v čem se liší. Obrázek jsme ořízli okrajem lesa.)



Rozkladu roviny na tyto oblasti se říká *Voroného diagram* (zkoumal ho začátkem 20. století ruský matematik Georgij Voronoj). Je dobré vědět, že je to rovinný graf (až na neomezené stěny, ale to je detail), tudíž pro S stromů má $\mathcal{O}(S)$ vrcholů a $\mathcal{O}(S)$ hran. Také se hodí znát algoritmus pro jeho konstrukci v čase $\mathcal{O}(S \log S)$, jako obvykle mazaným zametením roviny – detaily viz medvědí knížka.

Na chvíli zapomeňme, že les má nějaké okraje. Pak platí, že může-li lesem projít kruhová legie, pak jím může projít po hranách Voroného diagramu. Není divu: každá hrana diagramu je osa úsečky mezi dvěma stromy (nebo její část), takže jdeme-li po hraně, dodržujeme maximální možnou vzdálenost od této dvojice stromů.

Stačilo by tedy sestrojit diagram, pro každou hranu spočítat vzdálenost stromů, které tato hrana odděluje, a pokud je to méně než D, hranu smazat (na obrázku jsou tyto hrany čárkované). Tím by vznikl nějaký lineárně velký graf, který bychom vzápětí prohledali do šířky. To dává hezké řešení v čase $\mathcal{O}(S \log S)$.

Jenže teď si na okraje zase musíme vzpomenout. Co to způsobí? Část diagramu skončí za okrajem, takže ji odřízneme. Naopak máme povoleno chodit podél okraje (v uctivé vzdálenosti $D/2$). Proto se podíváme na konce hran, které se zarazily o okraj, a zkusíme je propojit. Každá po sobě jdoucí dvojice takových hran ohraničuje společnou stěnu diagramu, uvnitř níž leží nějaký strom. Pokud je tento strom vzdálený od okraje aspoň D, můžeme mezi hranami projít podél okraje. To znázorníme dalším typem hran (na obrázku tečkované). Bude jich nejvýše $\mathcal{O}(S)$ a spočítáme je také v čase $\mathcal{O}(S)$.

Shrneme-li tyto úvahy (a doplníme spoustu detailů, o které jsme vás ošidili), získáme algoritmus řešící úlohu v čase $\mathcal{O}(S \log S)$ bez jakýchkoliv požadavků na rozsah souřadnic a celočíselnost.

Martin „Medvěd" Mareš

28-3-8 Inteligence hejna

Měli jsme za úkol hledat co nejkratší kružnici, která prochází všemi městy České republiky, přičemž text seriálu napovídal, že by se mohl použít mravenčí algoritmus. Tím to také zkusíme, ale nebude to úplně jednoduché. Úlohu budeme řešit jen pro všechna města. Pro jejich menší část by řešení bylo obdobné.

Část z vás se spletla a namísto kružnice hledala cestu. To ale nebyl velký problém. Myšlenka v následujících řešeních je v zásadě stejná, jen se na konci musíme vrátit do počátečního vrcholu.

Základní mravenčí algoritmus

První problém, na který můžeme narazit, je, že výpočet trvá hrozně dlouho. Průchod jednoho mravence přes všechna města může zabrat až vteřinu. To nás motivuje použít málo mravenců, protože chceme algoritmus nechat běžet více iterací a dozvědět se aspoň nějaký výsledek.

Abychom ušetřili aspoň trochu času, předpočítáme si matici vzdáleností měst, abychom je nemuseli počítat stále dokola.

Další problém může nastat s hodnotami vhodnosti hran b_{ij} a intenzitou feromonů f_{ij}. V textu seriálu se doporučuje dát $b_{ij} = 1/d_{ij}$, kde d_{ij} je délka hrany, a k f_{ij} přičítat hodnoty $1/L$, kde L je délka nalezené cesty.

V praxi je dobré se snažit obě tyto hodnoty držet řádově stejné a při tom respektovat používané datové typy. Při spuštění algoritmu můžeme zjistit, že vzdálenosti měst se pohybují v rozmezí [501; 485 824], zatímco délky okružních jízd dosahují řádu 10 000 000. Pokud tedy použijeme $b_{ij} = 1\,000/d_{ij}$ a k f_{ij} budeme přičítat $10^7/L$ dostaneme se vždy zhruba do řádu jednotek.

Takto upraveným základním algoritmem spolu s parametry $\alpha = 2$, $\beta = 2$ a vaporizací $\rho = 0{,}5$ jsem za několik desítek iterací a zhruba s jednotkami mravenců dosáhl řešení 23 495 526.

Vícekrát jsem to nespouštěl a ladit parametry se mi nechtělo, protože to běželo dost pomalu.

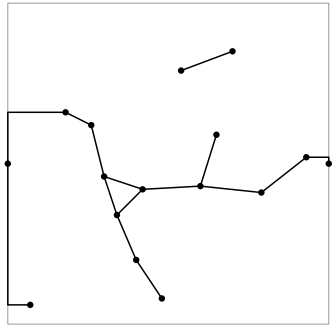
Hladový algoritmus

Teď chvíli zapomeneme na mravence a zkusíme úlohu vyřešit hladově. Začneme v náhodném městě a vždy se přesuneme do nejbližšího nenavštíveného města. Takto pokračujeme až dokud všechny neprojdeme. Kvalita řešení se může lišit v závislosti na výběru počátečního města. Tak se také vyplatí algoritmus spustit vícekrát.

Vašek Volhejn vyzkoušel pro hladový algoritmus všechny možné začátky a jako nejlepší řešení mu vyšlo 20 763 908.

Hladoví mravenci

A co takhle oba postupy zkombinovat? Nabízí se například vzít několik hladových řešení a vzít je jako počáteční popu-



Stačí tedy zjistit, zda ℓ a p leží v téže komponentě souvislosti. To můžeme otestovat třeba prohledáváním do šířky. Jistou nevýhodou ovšem je, že náš graf může mít až $\mathcal{O}(S^2)$ hran (rozmyslete si, jak by takový les vypadal). Bude tedy lepší nedržet si celý graf v paměti a vytvářet ho podle potřeby za běhu.

Budeme si udržovat frontu stromů, které máme zpracovat. Na počátku do ní vložíme stromy dosažitelné z ℓ, čili ty, jejichž x-ová souřadnice je menší než D. Pak postupně odebíráme stromy z fronty. Pro každý strom najdeme všechny jeho sousedy, tedy stromy ve vzdálenosti menší než D, a pokud jsme je ještě neviděli, přidáme je do fronty. Skončíme, pokud se fronta vyprázdní, nebo pokud objevíme strom s x-ovou souřadnicí větší než $L - D$.

To je všechno. Snadné, že? Ale pomalé… Navštívíme až S vrcholů a pro každý z nich procházíme S potenciálních sousedů. Jednoho souseda naštěstí stihneme zpracovat v konstantním čase (místo vzdáleností porovnáváme jejich druhé mocniny, abychom nemuseli odmocňovat), takže celý výpočet trvá $\mathcal{O}(S^2)$.

Program (C):
`http://ksp.mff.cuni.cz/viz/28-3-7-quad.c`

Pomohou políčka?

Existuje řada technik, kterými se dá hledání sousedních stromů urychlit. Můžeme například rozdělit les na políčka velikosti $D \times D$ a pro každé políčko si předpočítat, jaké stromy v něm leží. Sousedé daného stromu se pak nacházejí buď ve stejném políčku, nebo v některém z osmi okolních.

Tento trik může pro některé kombinace L a D výrazně pomoci. Limity v zadání úlohy bohužel připouštěly i případy, kdy je D řádově stejné jako L. Tehdy je políček konstantní počet, takže okolní políčka obsahují podstatnou část všech stromů. Škoda, tudy cesta nevede.

Znovu zametání

Není náhoda, že jsme k této sérii přibalili geometrickou kuchařku, v níž se ukazuje princip zametání roviny. Hodí se i pro tuto úlohu.

Les budeme zametat zleva doprava svislou přímkou. Budeme si při tom pamatovat, které stromy jsme už potkali. Pokaždé, když zametací přímka narazí na nový strom, najdeme jeho sousedy mezi předchozími stromy a natáhneme do nich hrany. Hledání si urychlíme dvěma triky:

- Nemá smysl zkoušet stromy, které se v y-ové souřadnici liší od nového stromu o více než D.
- Pokud se dva stromy vyskytly v témže řádku, stačí uvažovat pravý z nich. Ten levý je buďto moc daleko, nebo už leží v téže komponentě jako pravý, takže stačí hranu přivést do pravého. (Nezapomeňte, že souřadnice stromů jsou celočíselné, a proto je možných řádků málo.)

Pořídíme si tedy pole indexované číslem řádku $(0 \dots L-1)$. V něm si budeme pro každý řádek pamatovat, jaký nejpravější strom jsme tam viděli. Objeví-li se nový strom v y-tém řádku, stačí se podívat na nejpravější stromy v řádcích $y-D$ až $y+D$ a pro každý z nich zkontrolovat, jak je daleko.

Sousedy jednoho vrcholu tedy projdeme v čase $\mathcal{O}(D)$. Celý algoritmus nejprve stráví čas $\mathcal{O}(S \log S)$ setříděním všech stromů a $\mathcal{O}(L)$ inicializací pomocného pole. Poté prochází S stromů a nad každým z nich stráví čas $\mathcal{O}(D)$. Tím vytvoří graf o nejvýše $\mathcal{O}(SD)$ hranách, který pak prohledá do šířky v čase $\mathcal{O}(SD)$.

Celková časová složitost je tedy $\mathcal{O}(S \log S + L + SD)$.

Implementační (in)triky

Náš ukázkový program se drží předchozího nápadu, ale pro jednoduchost se v některých detailech odchyluje.

Především si místo třídění stromů podle souřadnic vytvoří matici $L \times L$, která o jednotlivých bodech lesa říká, zda tam leží strom. Zametání lesa pak prostě prochází tuto matici po sloupcích a hledá jedničky. Místo $\mathcal{O}(S \log S)$ tedy potřebuje čas $\mathcal{O}(L^2)$. To pro limity ze zadání je spíše lepší.

Také místo toho, abychom graf udržovali v paměti a pak ho celý prohledali, udržujeme komponenty souvislosti průběžně v datové struktuře Union-Find. Pro každou z $\mathcal{O}(SD)$ hran tedy voláme Union, který trvá $\mathcal{O}(\log S)$. Kdybychom byli pilnější, použili jsme rychlejší Union-Find se složitostí $\mathcal{O}(\log^* S)$[12] nebo $\mathcal{O}(\alpha(S))$, který je popsaný v kuchařce o minimálních kostrách.[13]

I s jednoduchým Union-Findem má náš program složitost $\mathcal{O}(L^2 + SD \log S)$, což je pro slíbené velikosti vstupu naprosto postačující.

Program (C):
`http://ksp.mff.cuni.cz/viz/28-3-7-grid.c`

Divotvorný diagram

Zbývá hrstka pochybností, zda by úlohu šlo řešit efektivněji, zejména v případě, kdy by nám nikdo neslíbil celočíselné souřadnice. Pochybnosti samozřejmě nejsou na místě, známe řešení v čase $\mathcal{O}(S \log S)$ s použitím jednoho milého králíka z kouzelnického klobouku výpočetních geometrů. Jen je to trochu pracnější, takže vše pouze načrtneme. Detaily můžete najít v geometrické kapitole medvědí knížky.[14]

Nechť $\mathcal{S}$ je množina stromů v rovině. Pro každý strom $s \in S$ definujeme jeho *oblast* R_s jako množinu všech bodů roviny, pro něž je s nejbližší strom (respektive jeden z nejbližších stromů, pokud jich je víc).

Podívejme se na nějaké dva stromy x a y. Všechny body roviny, které mají k x blíže než k y, tvoří polorovinu (hraniční přímkou této poloroviny je osa úsečky xy). Proto musí každá oblast R_s být průnikem konečně mnoha polo-

[12] Funkce $\log^* x$ říká, kolikrát musíme číslo x opakovaně zlogaritmovat, než se poprvé dostaneme pod 1. Roste tedy velice pomalu. Funkci $\alpha(x)$ zde definovat nebudeme, ale prozradíme, že roste ještě mnohem pomaleji.

[13] `http://ksp.mff.cuni.cz/viz/kucharky/kostry`

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

může dopřát pocit fungující rodiny a namísto mateřského objetí jim pronajímá chůvu, aby se mohla naplno věnovat svému účesu a nehtům. Tato dáma dostala před deseti lety neuvěřitelné možnosti. Bohužel se ale snažila urvat více, než dokázala sama unést. Dělala spoustu věcí, které nebyly zrovna správné, ale dělala je prostě jen proto, že si je mohla dovolit. Podle mě teď určitě nevede život, jaký si předtím vysnila.

Blížím se ke druhému domu. U něj je zaparkovaná blikající sanitka. Už zase… Vyhublý ajťák se totiž všemi technologiemi a vymožeností nechal natolik unést, že jen zřídkakdy vycházel ven. Dnešní svět to bohužel umožňuje. Když se nad tím zamyslíte, tak už vlastně neexistuje mnoho věcí, které by se nedaly zařídit online. Jeho život byl naprosto oddán virtuální realitě a moderním technologiím. Bohužel virtuálnímu světu dával výraznou přednost před tím fyzickým a ignoroval potřeby svého vlastního těla. A tak začal trpět ochabnutím svalů, záněty zápěstí a nakonec se mu začala bortit páteř.

Před necelým rokem ochrnul na dolní polovinu těla, když upadl na zem po cestě na záchod. Zachránili jej až kamarádi z online hry, když se do ní dva dny po sobě nepřipojil a ani o sobě nedal vědět. Dostat se pak k němu a poskytnout mu lékařskou pomoc byl také problém. Projít přes všechna technologická zabezpečení, která v domě měl, by byl úkol minimálně pro CIA. Ještě že měl jen obyčejná skleněná okna.

Po tomto incidentu mu byl přidělen osobní pečovatel. Ten se následně stal jeho nejbližším přítelem ve fyzickém světě za všechny ty roky. Vlastně byl zároveň jediným člověkem, který trávil v jeho přítomnosti více jak deset minut denně.

Když jsem míjel jeho dům, zrovna se sanitka rozhoukala a začala odjíždět.

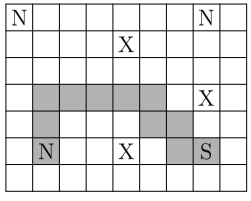
28-4-7 Jízda sanitkou — 11 bodů

Sanitka má naloženého stabilizovaného pacienta a potřebuje jej bezpečně odvézt z jeho domu do některé z nemocnic. Ve městě ale probíhají na různých místech opravy, v jejichž okolí to nebezpečně drncá a navíc poblíž nich hrozí uvíznutí v zácpě. Proto by se řidič rád držel od míst oprav co nejdál.

Máte zadanou mapu města jako čtvercovou mřížku s vyznačenou počáteční pozicí, pozicemi nemocnic a pozicemi všech míst, kde probíhají opravy. Najděte největší K takové, že existuje cesta ze startu do nějaké nemocnice, která se po celou dobu drží ve vzdálenosti alespoň K od všech míst oprav. Zároveň také najděte libovolnou cestu splňující toto omezení.

Po mapě se pohybujeme pouze vodorovně a svisle. Vzdálenost měříme v manhattanské metrice, tedy jako součet rozdílů x-ových a y-ových souřadnic. Například políčka (1,2) a (5,3) mají vzdálenost 4+1 = 5.

Pro následující mapu (S značí start, N nemocnici a X místo oprav) je největší $K = 2$ a jedna z možných optimálních cest je vyznačena šedě. Snadno nahlédnete, že pro $K = 3$ žádná cesta neexistuje.



Ve třetím domě byl klid. Nedá se to vůbec srovnávat s tím, co se tam dělo před pěti až deseti lety. Bývala tam divoká párty minimálně každý druhý den, která se bez výjimky protahovala minimálně do pozdního rána dalšího dne. Já sám jsem se tam také dvakrát vydal a můžu říct, že to bylo super! Pak jsem ale další dva dny téměř jen ležel, spal a střízlivěl. Takovouhle akci bych dokázal přežít maximálně dvakrát měsíčně, tak je pro mě naprosto nepochopitelné, že to ta zelenovlasá, dnes už černovlasá, hipsterka zvládla táhnout přes čtyři roky v kuse zhruba čtyřikrát týdně.

Celou dobu to vypadalo, že si vše maximálně užívá. Pak ale jednou z ničeho nic, když se párty zase dobře rozjela, proskočila zavřeným oknem a dopadla přímo do bazénu. Nic vážného se jí nestalo a nikdo přesně neví, proč to vlastně udělala. Nikdo jí v té době nebyl dost blízký, aby dokázal něco tušit, natož pak předpovídat. Z tohoto incidentu se ale nedokázala vzpamatovat a další skoro dva roky strávila na psychiatrickém oddělení. Asi to holka s tou volností, spontánností a nevázaností přehnala. Žila naplno každý moment a jen pro ten moment. Už si ale nenašla chvilku, aby se sama ohlédla odkud, kam, za čím, pro koho a proč jde. A tak se jí tyhle nevyřešené a odkládané pocity hromadily tak dlouho, až to najednou vybouchlo.

Dneska ale žije docela jiný život. Našla si stálého přítele a společně přízemí jejího domu přestavěli z nočního klubu na moderní školku. Jsme spolu přátelé a občas se navštěvujeme. Myslím si, že navzdory svému divokému mládí je to celkem fajn a rozumná holka.

Teď už se blížím ke čtvrtému domu. Tam se toho za těch deset let moc nezměnilo. Předtím tam žil podivný věčný student, jehož život byl řízen přísnými pravidly. Dnes tam žije podivný věčný student, jehož život je řízen ještě přísnějšími pravidly. Akorát nyní má svou sbírku titulů obohacenou o mnohé další obory. Každý den vstává a chodí spát ve stejnou dobu, má pečlivě navržený jídelníček, třikrát týdně cvičí a vůbec v jeho životě není žádná nepravidelnost. Namísto kamarádů má pouze spolupracovníky a spolubadatele.

Ze začátku jsem jej párkrát zkusil někam pozvat, ale v jeho rozvrhu se těžko dá hledat minuta, která by nebyla zabraná v rámci jeho složitého týdenního režimu. Teď o něm absolutně nedokáži říct, zda je v životě šťastný a zda má to, co hledal. Nerozumím mu, nemáme si spolu co říct, emoce na sobě nedává znát, ale třeba je tak šťastný.

Teď už se konečně blížím ke svému domu. Tam na mě čeká manželka a mé dvě krásné děti, pro které jsem na zahradě v posledním roce postavil malé dětské hřiště. Myslím, že teď žiju spokojený život, ve kterém ještě stále jdu za svým cílem. Posledních deset let pro mě nebylo vždy jednoduchých. Musel jsem se hodně snažit v práci a dělat přesčasy, abych získal vyšší kvalifikaci a zvedli mi plat. Doma nám zas nedávno prasklo potrubí a děti jsme museli složitě dovážet k babičkám a vůbec celkem často musíme řešit takové náhlé problémy. Ale zatím jsme to vždy všechno nějak zvládli.

Jsem opravdu rád, že mi kdysi byl přidělen právě tento dům a ne žádný jiný. Jsem opravdu spokojený. V tom se tehdy Dalimír trefil.

Všem mým sousedům se v ten den splnil jejich sen a začali si hlavně užívat a po nějaké době se toho přesytili. Nikdy v životě se nenaučili, jak o věci v životě bojovat a jak se o své cíle snažit. Já oproti nim dostal jen podmínky, které mi umožňovaly si za svými cíli jít a snažit se je plnit. Samotný fakt, že se mi to daří a postupně za mnou jsou vidět výsledky, mě naplňuje opravdovým štěstím. Takovým

tím pocitem zadostiučinění, který moji sousedé dlouho nebo dokonce nikdy neměli šanci poznat.

Oni do náruče dostali svůj cíl a v tom okamžiku se přestali zajímat o jakoukoliv cestu, která by je někam mohla dovést. Já jsem dostal možnost stavět si svou vlastní cestu a jít po ní. A to je přesně ono, protože právě ta cesta je můj cíl! Ano, je to tak: „Cesta je cíl!“

Příběh pro vás napsal

Karel Tesař

28-4-8 Strojové učení — 20 bodů

Co je to strojové učení? Lidské učení je schopnost adaptace na nové situace. Když budeme poprvé hrát šachy, asi nám to nepůjde moc dobře, ale podruhé to půjde trochu lépe, a když budeme hrát dost dlouho, můžeme se stát experty. Můžeme se naučit hrát dobře šachy, i když na začátku dostaneme jenom jejich pravidla, která nám umožňují dělat špatné i dobré tahy. Nikdo nám nedá přesný postup, jak být expert – v naší hlavě hraním šachů vznikne schopnost hrát je dobře.

Strojové učení se snaží replikovat tuhle schopnost v počítačích. Je velmi užitečné umět řešit problémy, na které neznáme žádné jasné algoritmy. To platí obzvlášť, když se snažíme automatizovat nějaký aspekt lidské inteligence. Jedna praktická aplikace je třeba *počítačové vidění*. Dnes počítače umí číst psaný text, rozeznávat lidské tváře, nebo třeba hledat ve fotkách dopravní značky, a to všechno s nadlidskou přesností. Algoritmus strojového učení například dostane 5 000 příkladů 10 různých dopravních značek a jeho výstupem bude postup, jak je od sebe rozpoznat.

Většina materiálů o strojovém učení je buď v angličtině nebo používá anglickou terminologii. Aby pro vás bylo snadnější si dle zájmu dohledat víc informací, budeme české termíny zavádět i s jejich anglickými ekvivalenty. Jestli byste chtěli strojovému učení věnovat víc samostudia, můžete si třeba najít materiály ke kurzu Machine Learning na Courseře.[2]

Druhy strojového učení

Strojové učení se dělí na tři široké kategorie: *učení s učitelem* (*supervised learning*), *bez učitele* (*unsupervised learning*) a *zpětnovazební učení* (*reinforcement learning*).

Zmíněný problém klasifikace dopravních značek patří pod *učení s učitelem*. Jakýsi *učitel* nám ukázal příklady a řekl nám „tohle je stopka“, „tohle je zákaz vjezdu“, a tak dále. Naším úkolem je podle těchto *trénovacích dat* vyrobit algoritmus, který bude fungovat dobře nejen na příkladech, které jsme dostali od učitele, ale i na těch, které jsme ještě nikdy neviděli.

Učení bez učitele se snaží najít nějaké pravidelnosti, ale nemá zvenku zadáno, čím se takové pravidelnosti budou vyznačovat. Když máme nějakou sadu dat, o které nic nevíme, učení bez učitele nám třeba může pomoct najít nějaké jejich „významné vlastnosti“, na které se pak můžeme zaměřit zvlášť. Mezi učení bez učitele patří třeba *detekce anomálií*, která hledá ve vstupních datech vzorky, které významně vybočují z „pravidelné struktury“. Když třeba máme datacentrum plné serverů, můžeme se snažit detekcí anomálií najít servery, se kterými je něco nějakým způsobem špatně, i když předem nevíme, jakým způsobem se mohou porouchat a jak se to projeví v parametrech, které měříme.

Zpětnovazební učení je o něco speciálnější. Používá se na učení chování v prostředí, ve kterém nemusí být podle výstupu naučeného systému hned jasné, jestli se chová chytře. Představte si třeba, že se snažíme naučit hrát Pacmana. Algoritmu říkáme, jak vypadá labyrint, kde se Pacman nachází, kde jsou duchové a kde je jídlo a on nám říká, kam chce, abychom šli. Chceme, aby se naučil, jak se má hýbat, aby sežral co nejvíc jídla a pokud možno neumřel. Když algoritmus řekne „jdi vlevo,“ tak nevíme hned, jestli to byl dlouhodobě dobrý krok, nebo špatný krok – to záleží na tom, jak se algoritmus zachová v budoucnosti (například jestli ho vlivem tohoto kroku za 5 tahů zabije duch). Pacman provádí posloupnost akcí v nějakém prostředí a snaží se, aby se nakonec naučil co nejlepší algoritmus pro hraní.

Učení bez učitele na to použít zřejmě nejde (protože máme konkrétní cíl – snažíme se maximalizovat snědené jidlo). Učení s učitelem se taky nehodí. Intuitivně bychom se totiž neučili hrát dobře – učili bychom se jenom napodobovat učitele. Pokud by tedy učitel byl třeba mistr světa v Pacmanovi, tak by ho náš naučený algoritmus neuměl porazit, protože se jenom naučí to, co umí učitel. Náš cíl je najít co nejlepší algoritmus, který je v rámci pravidel Pacmana možný.

Dnes si ukážeme pár základních algoritmů učení s učitelem.

Učení s učitelem

Zkusme se třeba naučit podle výšky a obvodu pasu předpovídat váhu lidí. Nejdříve najdeme nějaké dobrovolníky (nechť je jich N) a posbíráme jejich výšky, obvody a váhy. Vstupním informacím, podle kterých se snažíme předpovídat hmotnost, říkáme *příznaky* (*features*). Počet příznaků, tedy u nás 2, označíme jako p; první příznak je výška a druhý je obvod pasu. Uložíme si je do vektorů $x^{(1)}$, $x^{(2)}$, …, $x^{(N)}$. To, co se snažíme naučit předpovídat – tedy váhu – si uložíme do $y^{(1)}, \dots y^{(N)}$. Všem vstupním i výstupním datům, se kterými pracujeme, se říká souhrnně *dataset*. Označíme jej jako $\mathcal{D}$.

Když třeba $x^{(8)} = (1.87, 93)$ a $y^{(8)} = 85$, tak osmý dobrovolník měřil 1.87 m, měl obvod pasu 93 cm a vážil 85 kg. Informacím, které máme o jednom dobrovolníkovi, tedy dvojici $(x^{(8)}, y^{(8)})$, říkáme společně *vzorek* (*sample*). Používáme standardní vektorové značení, takže $x_1^{(8)} = 1.87$ a $x_2^{(8)} = 93$.

Většina aplikací učení s učitelem je *klasifikace* nebo *regrese*. Klasifikace je přiřazení vstupního vzorku do jedné z *kategorií*. Regrese je předpověď hodnoty nějaké funkce, která vede do reálných čísel. Předpovídání hmotnosti je regrese. Určování, jaký typ dopravní značky jsme vyfotili, je klasifikace.

Náš cíl je najít nějakou funkci f takovou, že pokud možno pro každý vzorek (x, y) je $y = f(x)$, nebo aspoň mezi nimi nebude velký rozdíl. Kromě toho ale chceme ještě jednu důležitou vlastnost: f by měla *dobře generalizovat*. Učení je hledání vhodné funkce f.

První požadavek, tedy aby f na známých vzorcích odpovídala správně, se dá splnit spoustou způsobů. Jeden z nich je třeba ten, že by si f při učení zapamatovala všechna trénovací data, a když by dostala vstup x, tak by se podívala, jestli tenhle vstup byl v trénovacích datech, a jestli byl, vrátila by jeho příslušný výstup, a jinak by vrátila třeba 9 999. Máte-li nepříjemný pocit, že na tomhle nápadu je něčo špatně, máte ho zcela správně.

[2] `http://coursera.org/learn/machine-learning`, nejbližší termín kurzu začíná 21. března.

Máme tedy algoritmus, který nalezne plán pro daný strom začínající v kořeni a končící v prvním patře: Rekurzivním zavoláním téhož algoritmu nalezneme plán pro první podstrom kořene a obrátíme v něm pořadí vrcholů (jako první helmici obdrží nějaký vrchol v prvním patře tohoto podstromu, jako poslední jeho kořen). Za tento převrácený plán připojíme rovněž převrácené plány pro druhý, … až poslední podstrom. Helmice tedy skončí v kořeni posledního podstromu, tedy v prvním patře celého stromu, což přesně potřebujeme.

Pro zjednodušení implementace nemusíme plány ani dodatečně obracet. Stačí rekurzivnímu volání navíc předat jeden parametr, zda má generovat normální (z kořene do prvního patra), nebo obrácený (z prvního patra do kořene) plán. Díky tomu si nemusíme plány ukládat, nýbrž můžeme vrcholy rovnou vypisovat.

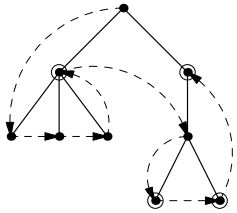
Snadno si rozmyslíte, že sousední vrcholy plánu jsou vzdálené maximálně tři hrany. Zbývá ještě vyřešit okrajové podmínky rekurze. Zastavíme se na jednovrcholovém stromě (tvořeném listem původního stromu), pro který je plánem prostě jednoprvková posloupnost obsahující tento vrchol.

Celý algoritmus v pseudokódu:

NajdiPlán(u, *obrať*):

1. Pokud *obrať* = 0: vypiš u.
2. Pro všechny v syny u:
3. NajdiPlán(v, $1 - obrat$)
4. Pokud *obrať* = 1: vypiš u.

Rozmyslete si, že opravdu odpovídá výše popsanému. Výsledný plán může vypadat třeba takto:



Zakroužkované jsou ty vrcholy, ve kterých hledáme obrácený plán. To jsou právě vrcholy v lichých patrech.

Obecné grafy

Zobecnit řešení na všechny grafy už je jednoduché. Z libovolného grafu můžeme udělat strom tak, že najdeme jeho kostru (např. prohledáním do hloubky) a na ni spustíme výše popsaný algoritmus. Protože pro každý strom existuje řešení, hrany mimo kostru jsou přebytečné a můžeme je ignorovat.

Navíc hledání kostry a konstrukci plánu nemusíme provádět odděleně, nýbrž je můžeme spojit do jednoho DFS průchodu grafem. Podrobněji ve vzorovém programu.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/28-3-6-dfs.py`

Filip Štědronský

28-3-7 Legie v lese

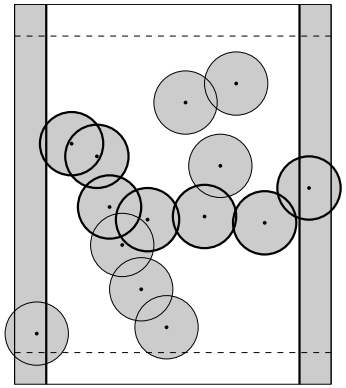
Les je obdélník o hraně L, v němž leží S bodových stromů. Horní stranou do obdélníku vstoupí kruhová legie o průměru D a chce vylézt spodní stranou. Nesmí přitom narazit na žádný strom, ani na levou či pravou stranu obdélníku. Střed legie tedy musí stále udržovat vzdálenost alespoň $D/2$ od všech stromů i od bočních stran lesa.

Překážky a ploty

Navigační problémy tohoto druhu se řeší snáz, pokud se lesem místo kruhové formace pohybuje jediný bod. Úlohu proto trochu přeformulujeme: každý strom „nafoukneme“ na kruh o poloměru $D/2$ a boční strany lesa roztáhneme směrem dovnitř na obdélníky široké rovněž $D/2$. Aby se všechny překážky vešly do lesa, roztáhneme les nahoře i dole o $D/2$.

Legii naopak smrskneme do jediného bodu. Ten může stát právě na těch místech, která neleží v žádném kruhu ani obdélníku.

Dostaneme tedy nějaké překážky ve tvaru kruhů a obdélníků a ptáme se, zda bod může projít shora dolů a vyhnout se všem překážkám.



Na předchozím obrázku to možné není, protože v něm existuje *plot* – posloupnost na sebe navazujících překážek, která spojuje levý okraj s pravým. Každá trasa shora dolů musí plot alespoň na jednom místě protnout, takže trasa narazí na alespoň jednu překážku.

Dokonce platí, že kdykoliv nejde projít shora dolů, existuje nějaký plot, který tomu brání. Vskutku: množina bodů, do kterých se dá shora dojít, tvoří nějakou uzavřenou oblast. Hranice této oblasti se skládá z částí horní, levé a pravé strany lesa a nějakých částí překážek. Představme si, že hranici této oblasti obcházíme od levého horního rohu lesa proti směru hodinových ručiček. Mezi posledním odchodem z levé strany a následujícím příchodem na tu pravou jsme museli projít přes na sebe navazující části překážek. To ovšem znamená, že tyto překážky tvoří plot.

Gigantický graf

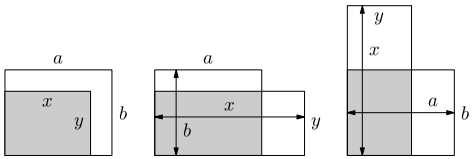
Potřebujeme tedy vymyslet, jak hledat plot. Pomůžeme si grafem: jeho vrcholy budou reprezentovat stromy, navíc přidáme vrchol ℓ pro levou stranu lesa a p pro pravou. Hranou spojíme každé dva vrcholy, jejichž vzdálenost v rovině je menší než D. To nastane právě tehdy, když se příslušné překážky protínají. Plot pak odpovídá cestě z vrcholu ℓ do vrcholu p v tomto grafu. Pro příklad z předchozího obrázku vypadá graf následovně:

Jak se zbavit otáčení

Úloha má nepříjemně mnoho stupňů volnosti: nejen, že si vybíráme, které desky použijeme, ale pro každou ještě určujeme natočení. Pojďme se otáčení zbavit. Dokážeme, že pokud každou desku předem natočíme „naležato“, tedy tak, aby její šířka byla větší nebo rovná výšce, o optimální řešení nepřijdeme.

Nejprve uvažujme situaci se dvěma deskami rozměrů $a \times b$ a $x \times y$ (první rozměr je šířka, druhý výška). Jistě můžeme předpokládat, že $a \geq b$ a $x \geq y$ (jinak desku otočíme) a navíc $b \geq y$ (první deska je ta vyšší). Aby vznikl kvádr maximálního objemu, chceme desky posunout a otočit tak, aby se překrývaly v co největší ploše. Překryv jistě nezmenšíme, přiložíme-li na sebe levé dolní rohy obou desek.

Nyní rozlišíme dva případy. V prvním je $x \leq a$, čili nižší deska je i užší (levý obrázek). Tehdy se desky překrývají celou plochou té menší z nich, takže se jistě nevyplatí kteroukoliv desku otáčet.



Zbývá nám případ $x > a$. Pokud obě desky ponecháme orientované stejně (prostřední obrázek), jejich společná část bude mít obsah $a \cdot y$. Pakliže je vůči sobě otočíme (pravý obrázek), vyjde obsah $b \cdot y$. První varianta je ovšem stejná nebo lepší, neboť $a \geq b$.

V obou případech platí, že překrývající se část má opět šířku větší nebo rovnou výšce, takže postup můžeme opakovat a o všech použitých deskách dokázat, že jsme je mohli nechat naležato. Ve zbytku řešení tedy budeme bezelstně předpokládat, že deskami není možné otáčet.

Elementární řešení

Náš první pokus o algoritmus bude založený na jednoduchém pozorování: výsledný kvádr zdědí jak svou šířku, tak svou výšku od některé desky, každou možná od jiné. Stačí tedy vyzkoušet n možných šířek, k nim n možných výšek a pokaždé probrat všechny desky a použít ty, které jsou dostatečně široké a vysoké. To zvládneme v čase $\mathcal{O}(n^3)$ a získáme za to pár bodů.

Mimochodem, řešení tohoto druhu by fungovalo, i kdybychom brali v úvahu otáčení. Výšku i šířku kvádru bychom vybírali ze všech výšek i šířek (celkem $4n^2$ možností) a při testování, zda se deska vejde, bychom zkoušeli obě možné orientace.

Chytřejší řešení

V minulém řešení počítáme stále dokola podobné věci, takže zkusíme výpočet přeuspořádat, abychom se tomu vyhnuli.

Nadále budeme zkoušet všech n možných šířek. Pro každou šířku w vybereme všechny dostatečně široké desky. Budeme je procházet od nejvyšší k nejnižší a průběžně přepočítávat aktuální kvádr. Pro nejvyšší desku samotnou má kvádr šířku w, výšku této desky a tloušťku 1. Když přidáme další, nižší desku, šířka zůstane, výška klesne a tloušťka vzroste o 1. Tak pokračujeme a pokaždé spočítáme objem aktuálního kvádru a započítáme ho do průběžného maxima.

Toto vše se dá stihnout v čase $\mathcal{O}(n^2)$: na počátku výpočtu setřídíme všechny desky podle výšky. Pak zkoušíme (v libovolném pořadí) všech n možných šířek, pro každou z nich probíráme desky v setříděném pořadí, přeskakujeme ty příliš úzké a pro ostatní v konstantním čase přepočítáváme objem kvádru.

Autor přiznává, že stále věří v existenci rychlejšího řešení, ale postupně si všechna taková vyvrátil. Kdybyste o nějakém věděli, dejte nám prosím vědět.

Program (C):
`http://ksp.mff.cuni.cz/viz/28-3-5.c`

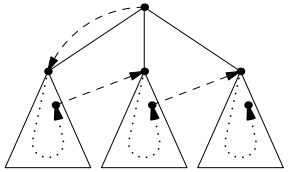
Martin „Medvěd“ Mareš

28-3-6 Počítání přeživších

Aby řešení mohlo existovat, graf musí být souvislý. To budeme nadále předpokládat. Nejprve vyřešíme jednodušší variantu, kdy graf přátelství je stromem. Hledanému předpisu, kdo komu má helmici předat, budeme říkat *plán* pro daný strom. Ten si lze představit prostě jako posloupnost vrcholů, kde dva sousední jsou vždy vzdálené ve stromě nejvýše tři hrany.

Strom si zakořeníme a použijeme obvyklý trik na stromové úlohy. Nejprve rekurzivně najdeme plán pro každý podstrom kořene a pak tyto plány nějak napojíme, abychom z nich poskládali plán pro celý strom. Předpokládejme pro začátek (později se ukáže, že je to trochu složitější), že každým podstromem začne helmice putovat od kořene.

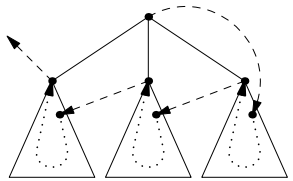
To znamená, že když helmice opouští první podstrom, musíme ji předat rovnou kořeni druhého podstromu (přes kořen celého stromu to nejde, tam už byla):



Tečkované čáry značí rekurzivně spočtené plány pro jednotlivé podstromy, čárkované jejich napojení. Aby předání mezi podstromy bylo korektní, musí plán pro každý podstrom končit v prvním patře tohoto podstromu (hned pod jeho kořenem). Kdyby končil hlouběji, předání bude přes víc než tři hrany.

Abychom mohli naše řešení použít rekurzivně, musí tedy i výsledný plán pro celý strom končit v prvním patře. Ale plán nastíněný v obrázku výše zřejmě končí v patře druhém...

Leč není třeba propadat panice. Všimneme si, že kdybychom z kořene poslali helmici do vrcholu, který je aktuálně poslední, a celý průchod průchod podstromy obrátili (díky neorientovanosti hran můžeme), skončíme v kýženém prvním patře:



Taková funkce f by fungovala perfektně na trénovacích datech, ale jakmile bychom chtěli předpovědět hmotnost někoho, koho jsme ještě neviděli, byla by zcela k ničemu. *Generalizace* je právě tato schopnost fungovat dobře i na lidi, které jsme při učení ještě neviděli.

Příznaky

Když chceme mít dobré předpovědi, samozřejmě velmi záleží na tom, abychom zvolili dobré příznaky. Musí obsahovat nějakou užitečnou informaci, na které závisí ta veličina, kterou předpovídáme. Když se snažíme naučit předpovídat hmotnost lidí, nejspíš nám pomůže vědět fyzikální vlastnosti, které s hmotností souvisí: třeba výšku, obvod pasu, věk nebo procento tuku v těle. Naopak nám asi nepomůže vědět barvu očí nebo oblíbenou kapelu.

Než spustíme algoritmus strojového učení, vyplatí se nejdřív zamyslet nad reálnou strukturou problému a zkusit pro něj vymyslet co nejužitečnější příznaky. Zkusme třeba předpovídat podle výšky a hmotnosti pravděpodobnost srdeční příhody v následujícím roce. Samotná hmotnost a výška sice jsou užitečné informace (když vážím 250 kg, jsem rizikovější, než kdybych vážil 70 kg), ale lepší nejspíš bude si přidat jako příznak BMI ((hmotnost v kg)/(výška v m)^2). BMI zohledňuje, že různí lidé mají různou postavu – je asi zdravější vážit 100 kg a být vysoký 2 metry, než vážit 80 kg a být vysoký 1.5 metru.

Příznaky jde široce dělit na kategorické a numerické. Numerické příznaky se dají přirozeně vyjádřit jako číslo, třeba výška v centimetrech nebo barva pixelu v obrázku. Kategorické příznaky můžou být třeba krevní skupina nebo státní občanství. Existuje pro ně nějaký poměrně malý počet možných hodnot (třeba krevní skupiny jsou {0$^+$, 0$^-$, A$^+$, A$^-$, B$^+$, B$^-$, AB$^+$, AB$^-$}) a na těchle hodnotách nemusí dávat smysl obvyklé číselné operace. Mohli bychom sice třeba označit krevní skupiny místo názvů pořadovými čísly ($0^+ = 1, \ldots, \mathrm{AB}^- = 8$), ale operace nad takovým označením nejsou užitečné – sice nad naším označením můžeme tvrdit, že $(0^-) + (\mathrm{B}^-) = (\mathrm{AB}^-)$, ale to neodpovídá žádnému vztahu v reálném světě. Stejně tak není B$^-$ v žádném smyslu „větší než“ 0$^+$. Oproti tomu třeba může dávat smysl porovnávat výšky dvou lidí nebo počítat jejich rozdíly.

Hodně algoritmů potřebuje, aby jejich vstupy byla čísla. Tehdy potřebujeme kategorické příznaky „zakódovat“ do číselných. Nejobvyklejší takové kódování z příznaku, který obsahuje jednu z K kategorií, vyrobí K číselných příznaků, jeden pro každou kategorii. Když vzorek patří do i-té kategorie, nastavíme i-tý příznak na 1 a ostatní na 0.

Předpokládejme, že existuje nějaký *skutečný* vztah $\hat{f}$, který z x dělá y (tedy z výšek a obvodů pasu dává hmotnost). Příznaky, které měříme, ale nemusí stačit na zcela přesnou odpověď: i když mají Jana a Katka stejnou výšku a obvod pasu, můžou mít jinou hmotnost, protože se Katka ráno nenasnídala. Existuje nějaký vliv vnějších příznaků, které neměříme (nebo možná ani z principu měřit nejdou – třeba máme nepřesnou váhu). Dá se to neformálně napsat jako $y = \hat{f}(x) + \varepsilon$: předpovídaná hodnota y se skládá ze složky, která závisí na x, a nějakého náhodného (a snad malého) ε, ve kterém jsou schované vlivy, které neumíme měřit.

Funkci f, kterou se snažíme naučit, se říká *model* – snaží se co nejpřesněji *modelovat,* co by dělala $\hat{f}$, kdybychom se jí mohli zeptat.

Měření chyby modelu

Po modelech chceme, aby byly co nejpřesnější a aby dobře generalizovaly. Chceme tedy, aby na *neznámém vzorku,* který *nebyl k dispozici učícímu algoritmu,* daly dobrou předpověď.

Existují různé metriky pro to, jak *dobrá* předpověď je. Většina z nich jsou nějaká míra *chyby.*

Pro naše předpovídání hmotnosti se třeba hodí velmi obvykle používaná *kvadratická odchylka.* Kvadratická odchylka modelu f na vzorku (x, y) je rovna $(y - f(x))^2$. Intuitivně jí velké odchylky vadí mnohem víc než malé.

Pro klasifikační úlohy se jako metrika hodí *accuracy*.[3] Accuracy na jednom vzorku je 1 tehdy, pokud jej f předpoví správně, a 0, když na něm udělá chybu. Když třeba předpovídáme, jakou dopravní značku obsahuje obrázek, zajímá nás jenom, jestli najdeme tu správnou. Když si spleteme stopku se zákazem vjezdu, je to pro accuracy stejně špatné, jako bychom si ji spletli s přikázaným směrem jízdy.

Zatím jsme si ukázali definice dvou různých chyb modelu na jednom vzorku. Po modelu chceme, aby měl co nejmenší *střední hodnotu chyby,* neboli aby na náhodně vybraném neznámém vzorku byl co nejpřesnější.

Když máme nějaký model f, jak zjistíme střední hodnotu chyby na neznámém vzorku? Neznámé vzorky jsou pro nás nedostupné – nemůžeme jít změřit 7 miliard lidí a spočítat, jakou chybu průměrně děláme. Máme k dispozici jenom data od dobrovolníků. Dělá se to tak, že učícímu algoritmu nedáme všechna data, která máme. Rozdělíme dataset $\mathcal{D}$ na *trénovací množinu* $\mathcal{S}$ a *testovací množinu* $\mathcal{T}$, třeba v poměru 90 %/10 %. Učícím algoritmům dáme k dispozici jenom trénovací data. Testovací vzorky před ním skryjeme. Až nám učící algoritmus dá model f, spočítáme jeho průměrnou chybu na testovací množině. S trochou statistiky se dá ukázat, že průměrná chyba na testovací množině rozumně odhaduje střední chybu na *všech neznámých vzorcích.*

Úkol 1 [1b]: Je potřeba, aby rozdělení na testovací a trénovací množinu bylo *náhodné.* Ať dataset $\mathcal{D}$ obsahuje nejdřív 100 značek „stop“, pak 100 značek „dej přednost v jízdě“ a nakonec 100 značek „slepá ulice“. Vymyslete, co a jak by se mohlo rozbít, kdybychom testovací množinu vybrali nenáhodně.

Když třeba po modelu chceme malé kvadratické odchylky, chceme malou střední kvadratickou odchylku na neznámých datech. Tu odhadneme podle střední kvadratické odchylky na testovací množině $\mathcal{T}$:

$$E \approx E_{\mathcal{T}} = \mathrm{MSE}_{\mathcal{T}} = \frac{1}{|\mathcal{T}|} \sum_{(x,y)\in\mathcal{T}} (y - f(x))^2.$$

Střední kvadratické odchylce se říká anglicky *mean square error* (*MSE*). Když nás zajímá accuracy na neznámých datech, odhadneme ji podobně: pomocí průměrné accuracy na $\mathcal{T}$.

Čím více dat dáme k dispozici algoritmu strojového učení, tím více se jim bude moct přizpůsobit a tím bude naučený model dávat přesnější předpovědi. Na druhou stranu, čím větší máme testovací množinu, tím přesněji chyba na testovací množině $E_{\mathcal{T}}$ odhadne skutečnou chybu na neznámých vzorcích E.

[3] Kromě accuracy se pro klasifikátory měří i metriky *precision* a *recall*. České překlady tady bohužel nemají tak dobře zavedený význam, jako anglické termíny.

Vzpomeňte si na f, která si uložila všechna trénovací data do tabulky a na všechno kromě nich vrátila 9 999. Takový model má na trénovacích datech nulovou chybu ($E_{\mathcal{S}} = 0$). Na testovacích datech $\mathcal{T}$, která nemá v tabulce, ale odpoví strašně špatně, takže průměrná chyba na testovacích datech $E_{\mathcal{T}}$ nám správně řekne, jak strašně moc špatný tenhle model je.

Přeučení a porovnávání modelů

Pokud se naučíme model, který je hodně dobrý na trénovacích datech, ale podstatně horší na testovacích datech, může to být kvůli *přeučení* (neboli *overfittingu*). K přeučení dochází, pokud učícímu algoritmu umožníme, aby se příliš silně adaptoval na nějaké zvláštnosti trénovacích dat, které ale obecně neplatí.

Hodně algoritmů strojového učení funguje tak, že postupně po *epochách* víc a víc adaptuje model na trénovací data. Vyrobí tedy posloupnost modelů $f_1, f_2, \ldots$, ve které jsou modely postupně čím dál tím adaptovanější na trénovací data, ale po nějaké době se začnou přeučovat, a tedy začnou být míň užitečné pro obecné použití. Podobná situace nastane, když zkoušíme na jedněch datech různé algoritmy strojového učení a chceme z naučených modelů vybrat ten nejvhodnější.

Očividný přístup, jak vybrat z modelů $f_1, f_2, \ldots$ ten nejlepší, je změřit chybu všech modelů na testovacích množině a vrátit ten, který ji má nejmenší, ale tenhle přístup je rozbitý.

Proč je rozbitý? Vzpomeňte si, že testovací množina se používá k *odhadování skutečné chyby*. Když si z modelů vybereme ten, který na nejmenší testovací chybu, bude jeho testovací chyba *příliš optimistický* odhad skutečné chyby.

Ilustrujeme si tenhle problém malým myšlenkovým experimentem. Představte si, že naše modely jsou tři férové mince. Pokud padne panna, model dá správný výsledek, a když padne orel, dá špatný výsledek. Každý model tedy ve skutečnosti dá správný výsledek v 50 % případů.

Testovací množinu vyrobíme tak, že každou mincí desetkrát hodíme. Na první vyjdou 3 orli, na druhé 7 orlů a na třetí 5 orlů. Vybereme si tedy druhý model a budeme si o něm myslet, že je přesný v 70 % případů. To je víc než jeho skutečných 50 % – druhý model jenom měl to štěstí, že při našem testu naházel nejvíc orlů. Jsme moc optimističtí o jeho výkonu, a to o 20 %.

Co kdybychom neházeli třemi mincemi, ale 10 000 mincemi? Skoro určitě (s pravděpodobností asi 0.99994) se stane, že náhodou některá z nich nahází 10 orlů. Byli bychom *extrémně optimističtí* – mysleli bychom si, že jsme našli minci, na které vždycky padají orli, i když je férová. Přidání dalších modelů způsobilo, že náš odhad je *horší* – teď už se mýlíme o 50 % místo 20 %.

Správné řešení je udělat „výběr nejlepší mince“ a „odhad pravděpodobnosti“ jako nezávislé experimenty: nejdřív 10× hodit a vybrat nejperspektivnější minci, a pak jí znova 10× hodit a podle druhých hodů odhadnout její „cinklost“. Když jsme v první fázi vybrali minci, co naházela 10 orlů, ale ve skutečnosti je férová a jenom měla štěstí, tak druhá fáze pořád bude 10 hodů férovou mincí a nejspíš nám řekne, že skutečná pravděpodobnost padnutí orla na vybrané minci je 0.5.

Když vybíráme z více modelů ten nejlepší a pak chceme vědět, jaký výkon od něho můžeme očekávat na nových datech, jedno ze správných řešení je rozdělit data na tři množiny: *trénovací, validační* a *testovací* (třeba v poměru 80 %/10 %/10 %). Trénovací množina se dá k dispozici učícím algoritmům (nebo nad ní iteruje jeden algoritmus a leze z něj posloupnost modelů). Jako nejlepší model vybereme ten, co má nejmenší chybu na *validační* množině. Tím pádem „neznečistíme testovací množinu“ a budeme ji moci dál používat k dobrému odhadování skutečné chyby.

Chyby na validační a testovací množině odpovídají naměřeným „cinklostem“ v první a druhé fázi myšlenkového experimentu s mincemi.

Teď už se trochu vyznáme v základních termínech a souvislostích, tak se konečně vrhneme na něco programovacího.

Lineární regrese

Algoritmy obecného učení jsou si obecně hodně podobné:

- Předepíší, jaký obecný tvar budou mít modely f, které z nich budou padat. Tenhle předpis bude obsahovat vstupní vektor x a nějaký vektor *parametrů*, který se označuje β. Konkrétní model dostaneme, když do předpisu dosadíme parametry.
- Říkají, jakou chybu se snaží minimalizovat.
- Popisují, jak efektivně najít takové β, že předpis f s dosazenými parametry β bude mít co nejmenší chybu.

Lineární regrese konkrétně:

- Hledá *lineární model*. Lineární model je funkce f, která na vstupu x dá jako výstup $f(x) = \vartheta + \sum_{i=1}^{p} \alpha_i x_i$. Konstanty ϑ a α_i jsou parametry určující konkrétní lineární model. Dohromady je těchto parametrů $(p+1)$, tedy vektor parametrů β je $(p+1)$-rozměrný: nejdříve obsahuje p složek $\beta_1 = \alpha_1, \beta_2 = \alpha_2, \ldots \beta_p = \alpha_p$ a pak jednu složku $\beta_{(p+1)} = \vartheta$. Graf lineárního modelu je přímka v p-rozměrném prostoru .[4]
- Chyba, kterou minimalizuje, je *střední kvadratická odchylka* na trénovací množině, proto se se jí taky říká *metoda nejmenších čtverců*:

$$E = \mathrm{MSE}_{\mathcal{S}}(\beta) = \frac{1}{|\mathcal{S}|} \sum_{(x,y)\in\mathcal{S}} (y - f_\beta(x))^2.$$

- Efektivně najde dobrou hodnotu vektoru β pomocí *gradientové metody*.[5]

Kdyby nám nezáleželo na času na naučení, stačily by nám první dva „moduly“. Mohli bychom si jenom vygenerovat biliardu modelů (třeba pro všechny hodnoty všech složek β od -100 do 100 s krokem 0.1), pro každý spočítat chybu a vybrat ten, co ji má nejmenší. To ale rychle přestane být efektivní pro hodně parametrů.

Úkol 2 [1b]: Jak závisí časová složitost tohohle hloupého přístupu „vygenerujeme si tabulku se spoustou modelů a vybereme ten nejlepší“ na velikosti trénovací množiny a počtu příznaků p?

Efektivnější minimalizace chyby vyžaduje trochu matematické analýzy. O kvadratické chybě se dají dokázat užitečné

[4] Pokud znáte skalární součin, všimněte si, že $f(\vec{x}) = (\vec{x}, 1) \cdot (\vec{\alpha}, \vartheta) = (\vec{x}, 1) \cdot \beta$.

[5] Pro lineární regresi existuje i vzorec, ze kterého jde nejlepší model přímo spočítat, ale gradientová metoda je jednodušší na pochopení a obecnější – používá se ve spoustě dalších učících algoritmů.

4. $H[x] \leftarrow \mathbf{R}$
5. Označ z_1 až z_k všechny závislosti x.
6. Pro $i = 1$ až k:
7. $\quad h \leftarrow$ Ohodnoť(z_i)
8. $\quad$ Pokud $h = \mathbf{X}$:
9. $\quad\quad H[x] \leftarrow \mathbf{X}$, vrať $\mathbf{X}$.
10. $H[x] \leftarrow f(H[z_1], \ldots, H[z_k])$, přičemž f je logická funkce předepsaná pro formulář x (AND, OR nebo XOR).
11. Vrať $H[x]$.

Protože chceme zjistit ohodnocení všech formulářů, prostě funkci Ohodnoť zavoláme postupně na všechny vrcholy. Celkem budeme potřebovat lineární čas, přesněji $\mathcal{O}(N + M)$, kde N je počet formulářů a M je celkový počet závislostí. To nahlédneme následovně. Vnitřní část funkce Ohodnoť provedeme pro každý vrchol nejvýše jednou. Tím pádem nejvýše jednou provedeme vnitřní cyklus přes všechny hrany vycházející z daného vrcholu. Tudíž na každou hranu se podíváme maximálně jednou.

Ještě musíme uvážit čas strávený na prvních třech řádcích funkce Ohodnoť, které mohou být provedeny víckrát. Ale funkce Ohodnoť je volána jen ze dvou míst:

- Z hlavní smyčky, kde je volána jednou pro každý vrchol (celkem $N\times$).
- Rekurzivně z jiného volání Ohodnoť, ale to se děje právě v místě, kde procházíme výstupní hrany. A už víme, že každou hranu navštívíme maximálně jednou, tedy těchto volání funkce Ohodnoť je dohromady maximálně M.

Z tohoto odhadu je taky vidět, že teď už se náš algoritmus nemůže zacyklit.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/28-3-3-dfs.py`

Filip Štědronský

28-3-4 Katapulty

Podívejme se prvně na lehčí variantu. Nabízí se umisťovat katapulty hladově, tedy dát každý katapult co nejvěži to půjde.

Řekněme, že projdeme bitevní lajnu zleva a při procházení si udržujeme množinu dosud neumístěných katapultů, které můžeme umístit na aktuální políčko.

Na každém políčku pak přidáme do množiny katapulty, které můžeme nově umístit, a poté z téhle množiny vezmeme katapult, jehož úsek končí nejdříve. Zkontrolujeme, že pravá hranice povoleného úseku je maximálně rovna aktuální pozici, a pokud ano, katapult umístíme na toto políčko. Pokud náhodou ne, zahlásíme, že řešení neexistuje. Budeme-li mít zrovna prázdnou množinu, jednoduše se posuneme na další políčko.

Snadno nahlédneme, že pokud náš algoritmus vydá řešení, bude toto řešení korektní – každý katapult bude ve svém úseku a zároveň bude na každém políčku jen jeden.

Musíme ale také ukázat, že pokud naopak ohlásíme neexistenci řešení, skutečně žádné řešení neexistuje. Když jsme vyhlásili neúspěch, máme nějaký katapult P, který jsme nedokázali umístit. Na každém políčku v úseku, kam smíme P umístit, už se ale vyskytuje jiný katapult, jehož úsek končí nejpozději stejně, tedy žádný z nich nemůžeme přesunout doprava.

Ještě musíme uvážit, jestli jsme některý z těch katapultů nemohli umístit více vlevo. Podobným argumentem ale ukážeme, že ne – když vezmeme největší levou hranici těchto katapultů a podíváme se na úsek mezi ní a začátkem úseku pro P, opět máme na každém políčku katapult, který nemůžeme přesunout doprava.

Takto dojdeme až na začátek lajny. Vlastně to znamená, že se snažíme umístit $L + 1$ katapultů na L políček, a to evidentně jít nemůže. Náš algoritmus tedy ohlásí neúspěch, pouze když řešení neexistuje.

Připomeňme teď, že počet katapultů označujeme K a délku lajny N.

Nejprve si všechny katapulty seřadíme podle levé hranice vzestupně. Tím pádem se můžeme postupně posouvat v poli katapultů a nalezení katapultů, které máme přidat do množiny, trvá lineárně v jejich počtu, za celý algoritmus tedy $\mathcal{O}(K)$. Samotné řazení nám zabere $O(K \log K)$.

K udržování katapultů, resp. k jejich přidávání a nalezení toho s minimální pravou hranicí, nám dobře poslouží halda. Tak bude každá operace trvat $\mathcal{O}(\log K)$.

Nejprve tedy seřadíme katapulty a pak projdeme celou bojovou lajnu. Tento průchod nám trvá $\mathcal{O}(N)$, na každém políčku se ptáme na katapult s minimální souřadnicí a možná přidáváme katapulty do množiny. Takových přidání je ale dohromady $\mathcal{O}(K)$, takže celková časová složitost algoritmu je $\mathcal{O}(N \log K)$. Přesněji tedy $(K \log K + N \log K)$, ale klidně předpokládejme $K \leq N$. Paměťová složitost je lineární v počtu katapultů, tedy $O(K)$.

Pěkné je si všimnout, že políčka, na kterých se nic neděje, jsou nezajímavá a vlastně nás jen zdržují. Líbilo by se nám umět skákat jen po těch, kde se něco děje. To ale umíme zařídit – o každém katapultu víme, kde jeho úsek začíná a končí, takže se můžeme buď posunout o jedno políčko (máme-li po umístění katapultu neprázdnou množinu), nebo rovnou skočit na příští událost.

Události si můžeme udržovat v haldě, na každém políčku tedy ještě přidáme případné zjištění příští události. Jelikož pro umístění K katapultů použijeme nejvýš K políček, bude celková složitost $\mathcal{O}(K \log K)$.

Teď těžší varianta, a pozor, přijde trik :) Všimneme si, že souřadnice katapultů jsou nezávislé – bez ohledu na to, do kterého řádku katapult umístíme, můžeme ho umístit do stejných sloupců.

Řešením úlohy tak není nic jiného než dvojí spuštění lehčí varianty, jednou pro řádky, jednou pro sloupce. A jelikož dvojka je konstanta, časová složitost zůstává $\mathcal{O}(K \log K)$.

Program (C):
`http://ksp.mff.cuni.cz/viz/28-3-4.c`

Program (C++):
`http://ksp.mff.cuni.cz/viz/28-3-4.cpp`

Karry Burešová

28-3-5 Závaží z fošen

Nejprve fošny ze zadání ohoblujeme až na strohou geometrickou realitu: Dostali jsme n desek, každá má danou šířku a výšku, všechny mají jednotkovou tloušťku. Chceme vytvořit kvádr. Uděláme to tak, že si vybereme podmnožinu desek, položíme je na sebe a případně některé z nich o 90 ° otočíme. Nakonec je ořízneme na šířku nejužší desky a výšku té nejnižší. Chceme přitom, aby vznikl kvádr o největším možném objemu.

že jsou položené na mřížce o velikosti $|A|$, $|B|$, ve které souřadnice vrcholů určují indexy jednotlivých řetězců s tím, že vlevo nahoře máme vrchol $S = [0, 0]$ a vpravo dole je vrchol $C = [a_{n-1}, b_{n-1}]$.

Pro každý vrchol dále bude platit, že z něj vede hrana doprava a dolů vždy, pokud není poslední ve své souřadnici, a dále si zavedeme zkratkové hrany, které nám budou vést přes diagonálu dolů doprava. Tyto hrany budou vést z těch vrcholů, které nám označují znak společný oběma řetězcům.

Nyní, když máme takový graf postavený, nalezneme nejkratší cestu z vrcholu $[0, 0]$ do vrcholu $[A, B]$. Jakmile jsme jednu takovou cestu našli, vydáme se po ní a zařídíme se podle toho, kterým směrem se rozhodneme vydat z vrcholu $[a_i, b_j]$:

- Pokud je následující vrchol cesty napravo od aktuálního, vypíšeme na výstup znak a_i.
- Pokud je následující vrchol cesty směrem dolů od aktuálního, vypíšeme znak b_i.
- Pokud se následující vrchol cesty nachází směrem po diagonále, je celkem jedno, který znak necháme vypsat, protože aktuální vrchol označuje společný znak.

Tímto jsme vypsali na výstup hledanou nejkratší společnou nadposloupnost. Samotné sestavení grafu nám potrvá $\mathcal{O}(|A| \cdot |B|)$, jelikož tento graf má počet vrcholů stejný, jako je součin délky obou řetězců $A \cdot B$. Nejkratší cestu potom umíme nalézt v lineárním čase jednoduchým prohledáním, takže nám to časovou složitost neporuší. Výsledný algoritmus má tímto stejnou asymptotickou složitost jako algoritmus popsaný výše.

Program (C):
`http://ksp.mff.cuni.cz/viz/28-3-2.c`

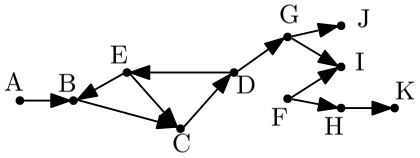
Program (C++):
`http://ksp.mff.cuni.cz/viz/28-3-2.cpp`

Václav Končický a Karry Burešová

28-3-3 Formulář na zbroj

Nejprve si úlohu převedeme na grafovou. Pro každý formulář vytvoříme jeden vrchol. Říkejme, že formulář x (přímo) *závisí* na formuláři y, pokud k získání x musíme předložit vyplněné y. Pro každou takovouto dvojici vytvoříme v grafu orientovanou hranu $x \to y$. Některé formuláře jsou vydávány „zadarmo“ (nemají závislosti). Odpovídající vrcholy v grafu nemají žádné výstupní hrany – takovéto vrcholy obvykle nazýváme *stoky*.

Graf závislostí může vypadat třeba takto:



Stoky jsou v tomto případě vrcholy I, J, K. Dále řekneme, že nějaký formulář x *nepřímo závisí* na y, pokud z x vede orientovaná cesta do y. Takový x nemůžeme získat, aniž bychom někdy předtím získali y. Například F nepřímo závisí na K, a vskutku: bez K nedostaneme H, a tedy ani F.

[10] `http://ksp.mff.cuni.cz/viz/kucharky/grafy`
[11] `http://ksp.mff.cuni.cz/viz/kucharky/dynamika`

Pokud nějaký formulář leží na orientovaném cyklu, nepřímo závisí sám na sobě. Takový určitě nemůžeme získat: například abychom získali E, potřebujeme C, který vyžaduje D, a ten zase E. Jinými slovy abychom dostali E, museli bychom nejdřív mít E. To určitě nejde, protože úředníci nevydávají formuláře na dluh.

Dalším formulářem, který určitě nemůžeme získat, je A, protože závisí na B, který leží na cyklu.

Řešení prohledáváním do hloubky

Pro každý formulář budeme chtít určit jeho *ohodnocení,* které může nabývat jedné ze tří následujících hodnot:

- 0, pokud formulář lze získat vyplněný záporně
- 1, pokud formulář lze získat vyplněný kladně
- **X**, pokud formulář není možné získat

Ohodnocení formuláře x najdeme jednoduchou rekurzivní funkcí Ohodnoť(x). Ta nejprve rekurzivním zavoláním získá ohodnocení všech přímých závislostí x, na jejich základě určí ohodnocení x, a to vrátí. To vlastně není nic jiného než prohledávání do hloubky.[10]

Protože funkce Ohodnoť může být na tentýž formulář zavolána mnohokrát a nechceme plýtvat časem opakováním toho samého výpočtu, použijeme dynamické programování.[11] Již spočítaná ohodnocení si budeme ukládat do pole H indexovaného čísly formulářů. Pokud je při zavolání Ohodnoť(x) hodnota $H[x]$ již známa, prostě ji rovnou vrátíme a nepočítáme znovu.

Kostra ohodnocovací logiky je jednoduchá: pokud některou ze závislostí x nejde získat (má ohodnocení **X**), ani x nemůžeme získat, tedy vrátíme rovněž **X**. Pokud naopak jde získat všechny, pak jde získat i x a jeho ohodnocení vytvoříme tak, že ohodnocení závislostí zkombinujeme předepsanou logickou funkcí.

Ale to nestačí. Pokud graf obsahuje cykly, takováto implementace by skončila nekonečnou rekurzí. Potřebujeme umět nějak cykly detekovat a vrcholy na nich označit jako nezískatelné.

To můžeme udělat například tak, že na začátku funkce Ohodnoť(x) si vrchol x označíme jako *rozpracovaný* (a na konci toto označení zase zrušíme). Pokud potom někdy při procházení závislostí narazíme na rozpracovaný vrchol, víme, že leží na cyklu. Proč? Pokud narazíme při prohledávání na rozpracovaný vrchol x, znamená to, že funkce Ohodnoť(x) aktuálně běží (v nějakém vyšším patře rekurze). Všechny vrcholy, které navštívíme, zatímco běží Ohodnoť(x), patří mezi (nepřímé) závislosti x. Takže pokud narazíme na x, znamená to, že x závisí na x.

Zavedeme si dvě pomocné hodnoty, které se mohou vyskytovat v poli H, pokud ohodnocení daného vrcholu ještě neznáme:

- **?**, pokud jsme dané pole ještě nenavštívili
- **R**, pokud je vrchol právě rozpracovaný

Upravený algoritmus bude vypadat následovně:

Ohodnoť(x):

1. Pokud $H[x] = \mathbf{R}$:
2. $\quad H[x] \leftarrow \mathbf{X}$, vrať **X**.
3. Pokud $H[x] \neq$ **?**: vrať $H[x]$.

vlastnosti. Zaprvé když si vyjádříme E jako funkci parametrů $\beta = (\alpha_1, \ldots \alpha_p, \vartheta)$, zjistíme, že je *spojitá.* Zadruhé má E jediné lokální minimum, které je i její jediné globální minimum. Zatřetí nemá žádné inflexní body.

Gradientová metoda

Gradientová metoda (*gradient descent*) umí takové funkce minimalizovat. Obecně tato metoda funguje tak, že začneme v nějakém libovolném, třeba nulovém počátečním bodu β_0. Potom se podíváme, kterým směrem bychom mohli trošku posunout β_0 tak, abychom tím co nejvíc snížili E.

Je vám ten nápad povědomý? Je vám povědomý zcela správně. Ve druhém díle seriálu jsme si ukazovali, jak hledat „horolezením“ extrémy reálných funkcí. Gradientová metoda je horolezení velmi podobná. Liší se tím, že „má kompas“ – umí najít zlepšující bod v okolí efektivněji než náhodným zkoušením.

Kompasu se říká *gradient.* Gradient E v bodu β_0 se značí $\nabla E(\beta_0)$ a je to vektor, který říká, kterým směrem máme jít z β_0, abychom šli co nejstrměji směrem zvyšující se E. V každém bodu může obecně gradient vést jiným směrem.

Když gradient otočíme, dostaneme směr největšího *poklesu* E. Když je délka gradientu v bodě β velká, znamená to, že E v β rychle roste/klesá. Naopak malý gradient znamená, že se nacházíme v placaté oblasti a nulový gradient znamená, že β je lokální minimum, maximum, nebo inflexní bod. Protože E inflexní body nemá, značí v ní nulový gradient lokální minimum nebo maximum.

Krok gradientové metody číslo t začne v souřadnicích β_{t-1}. Spočítá si v tomhle bodě gradient $\nabla E(\beta_{t-1})$, nějaký jeho γ-násobek odečte od souřadnic a tím spočítá β_t.

Jak velké má být γ? Čím menší, tím déle bude gradientová metoda běžet, ale tím přesněji se zase trefí do lokálního optima. Když bude γ moc velká, budou kroky gradientové metody lokální minimum „přeskakovat“, což se projeví tak, že se po pár iteracích dostaneme k nekonečně velkým hodnotám parametrů a nic se nenaučíme. Pokud se vám taková věc stane, zkuste γ zmenšit o pár řádů. Hodně učících algoritmů má podobný parametr ovlivňující velikost jednoho učícího kroku. Většinou se mu říká *learning rate,* rychlost učení.

Po odečtení γ-násobku gradientu přejdeme na krok $(t+1)$: spočítáme gradient v β_t, odečteme jeho γ-násobek, a tak dále. Skončíme, když je splněné nějaké kritérium, například když už jsme počítali dost dlouho nebo když je gradient hodně malý. Pokud gradientová metoda skončí v místě, kde je gradient skoro nulový, našla lokální minimum.

Gradient je rovný nule i v lokálním maximu. Když bychom měli tu neuvěřitelnou smůlu, že bychom začali gradientovou metodu dokonale přesně v lokálním maximu, tak bychom hned skončili a vrátili lokální maximum. To se ale skoro určitě nestane. Stane se možná, že začneme gradientovou metodu blízko lokálního maxima. Potom nás od něho ale první krok trochu oddálí (protože jde směrem klesající E), druhý krok nás oddálí ještě víc a tak dále, takže v lokálním maximu neskončíme.

Gradientová metoda tedy skončí poblíž lokálního minima, které je kvůli vlastnostem E i globální minimum, a tedy bod, ve kterém skončíme, bude určovat parametry lineárního modelu s malou chybou.

[6] `http://ksp.mff.cuni.cz/viz/evoluce`

Zbývá už poslední kousek skládačky: jak gradient E spočítáme?

Pokud bychom o E nevěděli nic, mohli bychom místo počítání gradientu zkusit jenom o trošku pohnout každou složkou β a vybrat to drobné pohnutí, které E nejvíc zmenší. To bude poměrně pomalé, ale jednoduché a bude to fungovat na všechna E, co jsou spojitá, mají jediné lokální minimum, které je zároveň i globální minimum a nemají inflexní body.

Gradient se dá pro naši definici chyby spočítat explicitně. Je to vektor o $(p+1)$ složkách a jeho i-tá složka se spočítá tímto průměrem přes všechny trénovací vzorky:

$$\nabla E(\beta)_i = 2 \cdot \frac{1}{|\mathcal{S}|} \sum_{(x,y)\in\mathcal{S}} x_i (f_\beta(x) - y).$$

Poslední složka (odpovídající členu ϑ) se spočítá jako:

$$\nabla E(\beta)_{(p+1)} = 2 \cdot \frac{1}{|\mathcal{S}|} \sum_{(x,y)\in\mathcal{S}} (f_\beta(x) - y).$$

V obou rovnicích je f_β lineární model určený parametry β.

Bitevní plán: Začneme v libovolném (třeba nulovém) bodě $\beta_0 = (\vec{\alpha}_0, \vartheta_0)$. Spočítáme gradient, odečteme jeho γ-násobek a opakujeme, dokud gradient není malý vektor nebo nás to nepřestane bavit. Uložíme výsledný model.

Je zaručeno, že gradientová metoda minimalizuje spojité funkce, které mají jediné globální a lokální minimum a nemají inflexní body. Často se ale používá i na spojité funkce, které tuhle vlastnost nemají. Potom může skončit v lokálním, ale ne nutně globálním optimu, nebo v inflexním bodě. V praxi nám to ale často nevadí – i tak dává použitelně dobré výsledky. Gradientové metodě také můžeme pomoct *náhodnými restarty* – pustíme ji několikrát za sebou z různých náhodně zvolených β_0. Dá se pak doufat, že projdeme větší kus definičního oboru E, takže v nejlepším nalezeném bodě budeme blíž globálnímu minimu.

Úkol 3 [6b]: Stáhněte si ze stránky seriálu[6] dataset o spotřebě benzínu v USA. Soubor má 5 sloupců dat oddělených čárkami. Každý řádek jsou data z jednoho státu. První sloupec je vybíraná daň z benzínu v centech na galon, druhý je průměrná mzda v dolarech, třetí délka dálnic ve státu v mílích, čtvrtý je poměr obyvatel s řidičským průkazem a pátý je roční spotřeba benzínu ve státě v milionech galonů.

Naprogramujte lineární regresi a pomocí ní odvoďte vzorec na předpovídání spotřeby benzínu podle ostatních hodnot. Data nemusíte dělit na testovací a trénovací množinu. Pro nás fungovalo dobře začít v nulových parametrech a pak provést 100 000 iterací gradientové metody s $\gamma = 10^{-8}$. Pošlete svůj zdrojový kód a nalezenou střední kvadratickou odchylku na celém datasetu. Zkuste ji mít menší než 21 800.

Algoritmus K nejbližších sousedů

Algoritmus K nejbližších sousedů (anglicky *K-nearest neighbors* nebo *KNN*) je založený na jednoduché myšlence: když chceme podle mojí výšky a obvodu pasu předpovědět mou hmotnost, odhadneme ji podle lidí, kteří jsou mi podobní výškou a obvodem pasu.

Model si bude pamatovat všechna trénovací data a jeho jediný parametr je přirozené číslo K. Když nám přijde nový vstup x, najdeme ve trénovacích datech K vzorků, které jsou mu nejbližší. Podíváme se na známé výstupy pro nejbližší sousedy, nějak je zagregujeme a výsledek vrátíme.

Konkrétní použití se liší podle toho, jak nadefinujeme, že je vstup x „blízký“ k nějakému trénovacímu vzorku, a podle toho, jak z výstupů na nejbližších sousedech vyrobíme předpověď pro neznámý vstup.

Blízkost se může definovat třeba podle malé euklidovské vzdálenosti

$$d(x, x^{(i)}) = \sum_{j=1}^{p} (x_j - x_j^{(i)})^2.$$

Euklidovská vzdálenost se hodí na numerické příznaky. Pokud ji ale použijeme, je potřeba dát si pozor na to, aby rozdíly v příznacích byly stejně významné.

Co tím myslím? Vzpomeňme si na příklad s předpovídáním hmotnosti a podívejme se na tři vzorky s těmito vstupními příznaky:

příznak	$i=1$	$i=2$	$i=3$
$x_1^{(i)}$: výška v m	1.93	1.92	1.4
$x_2^{(i)}$: obvod pasu v cm	83	86	82

Dobrovolník 1 se liší od dobrovolníka 2 jedním centimetrem výšky a třemi centimetry obvodu pasu. Dobrovolník 1 se od dobrovolníka 3 liší 53 centimetry výšky a jedním centimetrem obvodu pasu. Nemusíme být experti na antropologii, abychom očekávali, že hmotnost dobrovolníka 1 bude podobnější hmotnosti dobrovolníka 2 než dobrovolníka 3.

Rozhodli jsme se, že první složka (výška) bude číslo v metrech a druhá složka (obvod pasu) v centimetrech. Když si spočítáme euklidovskou vzdálenost tak, jak jsme si ji nadefinovali, dostaneme:

$$d(x^{(1)}, x^{(2)}) = (1.93 - 1.92)^2 + (83 - 86)^2 = 9.0001$$

$$d(x^{(1)}, x^{(3)}) = (1.93 - 1.4)^2 + (83 - 82)^2 = 1.2809$$

Dobrovolník 3 je tedy navzdory naší intuici dobrovolníkovi 1 mnohem „euklidovsky blíž“ než dobrovolník 2. Je to tím, že jsme zvolili pro příznaky taková měřítka, že změna ve výšce je mnohem podstatnější než numericky stejně velká změna v obvodu pasu. Kdybychom tedy hledali blízké vzorky podle euklidovské metriky, neodpovídalo by „blízké okolí“ našim představám.

Tohle se dá částečně řešit tím, že si vstupní data *normalizujeme*. Jeden ze způsobů normalizace je spočítat pro každý příznak jeho minimum a maximum na trénovací množině, pak všechny jeho hodnoty přeškálovat do intervalu $[0; 1]$ a počítat euklidovské vzdálenosti na přeškálovaných příznacích. Stejné přeškálování provedeme, když máme udělat předpověď pro nový vstup.

Když jsou všechny naše příznaky diskrétní, můžeme definovat *blízkost* podle takzvané Hammingovy vzdálenosti. Hammingova vzdálenost dvou vektorů je počet jejich složek, které se liší.

Zbývá agregace předpovědí z okolí. Pro regresi je nejjednodušší vzít výstupy z nejbližších sousedů a vrátit jejich průměr. Pro klasifikaci můžeme třeba vrátit tu kategorii, která má mezi K sousedy nejvíce hlasů (a když jich nejvíce hlasů dostalo několik, vybereme z takových třeba tu první). Počítání průměru na kategoriích totiž nemusí být dobře definovaná operace.

Úkol 4 [6b]: Stáhněte si ze stránky seriálu dataset o kvalitě bílého vína. Obsahuje 12 sloupečků oddělených čárkami. První řádek popisuje význam sloupců. Prvních 11 sloupců obsahuje různé chemické vlastnosti vína a poslední jeho kvalitu mezi 1 a 10, kterou se naučíme předpovídat.

Naprogramujte algoritmus K nejbližších sousedů s normalizací příznaků a euklidovskou vzdáleností. Z kvality od K sousedů vytvářejte předpověď prostým aritmetickým průměrem.

Rozdělte dataset na trénovací, validační a testovací množinu v poměru 60 %/20 %/20 %.

Měřte střední kvadratickou chybu na trénovací a validační množině v závislosti na K. Zkuste všechna K od 1 do 40.

Jak na K záleží chyba na trénovací množině? Jak na K záleží chyba na validační množině? Proč?

Vyberte nejslibněji vypadající K a pomocí testovací množiny odhadněte, jaká bude skutečná accuracy vašeho modelu.

Tahle úloha nejspíš poběží celkem dlouho. Jestli jí chcete pomoct, můžete zkusit využít více jader procesoru. K řešení přibalte svůj program a výsledky.

Opisování od evoluce

Nejúspěšnější nám známý učenlivý systém byl vytvořen přibližně čtyřmi miliardami let evoluce. Inspirováni jeho architekturou jsme vymysleli mnoho forem *umělých neuronových sítí*. Obzvlášť v posledních několika letech jsme díky několika novým trikům a rychlejším paralelním počítačům dosáhli úžasných výsledků, mimo jiné ve zpracování obrazu a zvuku, ale třeba i přirozeného jazyka.

Informace v lidském lidským mozku posílají elektrické a chemické signály. Domníváme se, že nejdůležitější „výpočetní jednotkou“ je neuron. Většina neuronů má nějaké „vstupní kanály“, kterým se říká *dendrity,* a jeden „výstup“ – *axon*. Dendrity tvoří jakousi „stromovitou strukturu“, která sahá řádově stovky mikrometrů od těla neuronu. Zatímco dendritický strom některých neuronů má až tisíc větví, některé neurony jich mají pouhé jednotky. Délka některých lidských axonů dosahuje až 1 metru. Každý neuron má sice nejvýše jeden axon, ale ten se může mnohonásobně větvit a posílat jeho signály až stovkám jiných neuronů.

Elektrické signály v nervovém systému jsou kódované pulzně. Neuron „sbírá“ signály od svých vstupů a pokud se v neuronu nahromadí za určitou dobu dostatečné množství potenciálu, „vystřelí“ signál na výstupu. Spojením mezi neurony se říká *synapse*. Některé synapse jsou *excitační* a některé jsou *inhibiční*. Signály poslané po excitačních synapsích „zvyšuje vnitřní potenciál“ cílového neuronu, zatímco inhibice mu „brání vystřelit“. Možná překvapivě se informace v mozku šíří relativně pomalu. V závislosti na typu signálu jsou to řádově jednotky až stovky metrů za sekundu.[7]

Systému neuronových spojení v mozku se souhrnně říká *konektom*. Můžeme si jej představit velmi zjednodušeně jako poněkud gigantický orientovaný graf, jehož vrcholy tvoří jednotlivé neurony. Hrany reprezentují spojení a vedou směrem, jakým se přenáší signály: od neuronu, který je vypálí, do neuronu, který je má na vstupu. Kromě toho, že hrany mohou reprezentovat excitační nebo inhibiční spojení,

[7] Například signály od receptorů bolesti nebo teploty se šíří rychlostí asi 0.5 až 2 m/s. Oproti tomu signály od proprioreceptorů – detektorů relativní pozice jednotlivých částí těla – se šíří rychlostí mezi 80 a 120 m/s. Lidský mozek obecně funguje mnohem pomaleji než sériové počítače. Procesy v něm jsou však vysoce paralelizované.

Máme před sebou dvě symetrické pyramidy (které už možná jako pyramidy vůbec nevypadají) a chceme získat maximálně stejně, to by svádělo právě k nějaké kopírovací strategii. Jenže my začínáme a nemáme ani tu nejmenší jistotu, že soupeř bude naše tahy kopírovat. Co víc, nemáme ani jistotu, že soupeř bude hrát poslední.

Budeme tedy muset na důkaz toho, že naše strategie funguje, jít jinak. Držte si klobouky, pojedeme z kopce.

Nejprve si zavedeme další označení, tentokrát pro helmice, na kterých neleží žádná další (takže jejich shozením neshodíme nic dalšího). Těm budeme říkat *volné*.

Všimněme si, že mezi všemi helmicemi s momentálně minimální hodnotou je alespoň jedna volná. To plyne z toho, že směrem dolů hodnoty helmic neklesají. Také si snadno rozmyslíme, že my bereme vždy volnou helmici – alespoň nějaká z nich má nejmenší hodnotu, takže se nám určitě nevyplatí shazovat helmic víc.

Přiřaďme teď jednotlivým helmicím v jedné ze symetrických pyramid označení x_i taková, že $x_i \leq x_{i+1}$. Každé x_i se ve hře (resp. této hladové části hry) vyskytuje dvakrát – na začátku je jednou v první pyramidě, jednou ve druhé.

Helmice můžeme rozdělit na naše a na soupeřovy podle toho, kdo je shodil. K tomu si zavedeme další označení, a to *hlavní helmice*. To bude pro každý tah nejmenší z helmic, které hráč v daném tahu shodil. Pokud helmic shodil více, označíme ty ostatní jako *vedlejší*.

Snadno si všimneme, že všechny naše helmice jsou hlavní (jelikož nikdy neshodíme víc než jednu helmici naráz).

Nyní bychom chtěli spárovat své helmice se soupeřovými, a to tak, aby vždy naše helmice měla maximálně takovou hodnotu, jakou má spárovaná soupeřova helmice. Některé soupeřovy helmice možná zůstanou nespárované, ale to nám nijak nevadí. Kdyby se nám takové párování podařilo najít, musí být náš počet kamínků maximálně stejný jako soupeřův.

Představme si, že vždy spárujeme naši hlavní helmici s hlavní helmicí, kterou soupeř vezme v příštím tahu. Pro takové párování by nerovnost určitě platila. Jenže se může stát, že v posledním tahu hrajeme my, tedy zbývá jedna nespárovaná helmice, kterou už není s čím spárovat. Ukážeme ale, že v takovém případě můžeme helmice přepárovat.

Označme nespárovanou helmici jako X a její hodnotu jako x_i. Nyní se podíváme na suffix $x_i, \ldots, x_N$ všech hodnot, které byly ve hře. Existují dvě možnosti.

Zaprvé, existuje vedlejší soupeřova helmice s hodnotou z tohoto suffixu. Jelikož jsme zatím párovali vždy s hlavními helmicemi, je tato vedlejší helmice nespárovaná, a navíc má určitě aspoň stejnou hodnotu jako X. Tedy můžeme X spárovat s touto vedlejší helmicí. Tím máme vše spárováno a nerovnosti platí.

Zadruhé, žádná taková vedlejší helmice neexistuje. To znamená, že suffixu odpovídají pouze hodnoty hlavních helmic. Zároveň ovšem suffixu odpovídají hodnoty sudého počtu helmic, a jedna z těchto helmic je X. To znamená, že alespoň jedna hlavní helmice je spárovaná s nějakou, jejíž hodnota je menší než x_i.

Označme helmice v tomto páru jako A a B, nechť $A \leq B$ (tedy hodnota A v suffixu neleží, hodnota B ano). V dosavadním párování platí, že hodnota naší helmice je menší než soupeřovy, tedy A je naše. Přepárujme nyní X s B; A se stane nespárovanou.

Jelikož x_i bylo v suffixu nejlevěji, je určitě hodnota X neostře menší než hodnota B, takže nerovnosti nám stále platí. Může se zdát, že jsme si nepomohli, opět máme jednu nespárovanou helmici. Její hodnota ale určitě leží v řadě hodnot o jedna víc než vlevo, tedy opakováním přepárování spárujeme všechny helmice v konečném čase.

Tady ještě zdůrazněme, že pokud se dostaneme až do situace, kdy má nespárovaná helmice hodnotu x_1, musí mít soupeř nutně alespoň jednu vedlejší helmici. V opačném případě by muselo existovat párování vedoucí před suffix, ale před x_1 už nic není.

Vždy tedy umíme vytvořit párování takové, že naše helmice má maximálně stejnou hodnotu, jakou soupeřova, tedy i z hladové části hry budeme mít maximálně tolik, kolik získá soupeř.

Tím jsme dokázali, že popsaná strategie je skutečně vyhrávající. Ufff!

Karry Burešová

28-3-2 Líný písař

Označme si věty jako řetězce A a B, bez újmy na obecnosti předpokládejme, že $|A| \geq |B|$. Podívejme se nejprve, jakým způsobem sestrojíme řetězec S takový, aby obsahoval po odstranění některých znaků každou větu a také byl nejkratší, neboli aby tento řetězec byl nejkratší společnou nadposloupností A, B. Jak ale tento řetězec bude vypadat?

Určitě se nám vůbec nevyplatí mít délku $|S| > |A| + |B|$, jelikož bychom museli přidat další zbytečné znaky. Řetězec dále můžeme zkrátit tím, že znaky společné oběma větám zapíšeme do našeho řetězce S jen jednou.

Tyto společné znaky však vůbec nemusí být na stejných pozicích u obou řetězců. Příkladem jsou řetězce

$$A = \texttt{xAxBxCxDxE}, \quad B = \texttt{klmABCDkElm},$$

kde společné znaky `ABCDE` tvoří nejdelší společnou podposloupnost (zkráceně NSP). Znaky této NSP se ale v obou řetězcích vyskytují na různých indexech: v A na indexech $[1, 3, 5, 7, 9]$ a v B na indexech $[3, 4, 5, 6, 8]$. My potřebujeme zjistit jak samotnou NSP, tak indexy jejích znaků v obou řetězcích.

K vyřešení problému nalezení NSP lze využít dynamického programování, algoritmus je přímo popsán v kuchařce. Tento algoritmus nás stojí $\mathcal{O}(|A| \cdot |B|)$ času.

Jakmile máme indexy společných znaků z obou vět, můžeme sestrojit náš požadovaný řetězec S. Mějme i-tý společný znak, a_i a b_i indexy tohoto společného znaku v A a B. Potom postupně budeme zvyšovat i od 1 po délku NSP a při každé iteraci vypíšeme všechny znaky z A, B mezi $(i-1)$-ním a i-tým společným znakem a nakonec samotný i-tý znak. Tento algoritmus nám zabere $\mathcal{O}(|A| + |B|)$ času.

Celkově algoritmus zabere $O(|A| \cdot |B|)$ času. Důvod je takový, že nám právě nejvíce času potrvá vyhledání NSP. Samotné vypsání řetězce trvá jen lineárně, což nám asymptotickou složitost nezmění.

Jiný pohled na úlohu

Na tuto úlohu existuje i řešení založené na odlišné myšlence, které nakonec bude stejně efektivní. Představme si orientovaný graf, jehož vrcholy jsou reprezentovány uspořádanou dvojicí $[a_i, b_i]$. Tyto vrcholy si můžeme graficky znázornit,

Vzorová řešení třetí série dvacátého osmého ročníku KSP

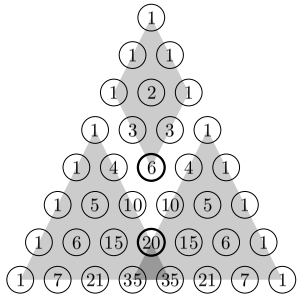
28-3-1 Pyramida z helmic

Úvodem dnešních řešení se vám všem musíme omluvit za naprosto neodpovídající ohodnocení pyramidové úlohy (a to ještě vypadalo krásně, že nejlehčí úloha vyšla jako první!). Pokud jste si tedy nad úlohou lámali hlavu a nemohli na nic kloudného přijít, možná jste jen projevili víc rozumu než orgové, kteří kdysi uvěřili tomu, že úloha je přece jednoduchoučká.

Doufáme, že příště bude bodové ohodnocení lépe vypovídat o obtížnosti úlohy, a hlavně že jsme nikoho od řešení neodradili. Ale teď už honem na řešení.

Snadno si můžeme rozmyslet, že pro $N \leq 2$ vždy vyhraje druhý hráč. Naopak dost komplikovaně můžeme dojít k tomu, že pro $N \geq 3$ existuje vyhrávající strategie pro prvního hráče.

Nejprve tuto vyhrávající strategii popíšeme, a pak teprve dokážeme, že opravdu funguje a soupeř nemá šanci. Zaveďme si teď pár označení, ať se nám strategie popisuje snáz. Předně, *lichým řádkem* budeme myslet řádek obsahující lichý počet helmic. *Drahý střed* pak bude prostřední helmice na nejspodnějším lichém řádku a *levný střed* prostřední helmice na druhém lichém řádku odspodu. Také si dovolme používat pojem pyramida, přestože tvar, který budou helmice v průběhu hry vytvářet, nebude vždy pyramidu připomínat.



1. Hned v prvním tahu shodíme levný střed. Tím jakoby vznikly dvě shodné pyramidy, které se částečně překrývají.
2. Potom budeme „kopírovat“ soupeřovy tahy, dokud soupeř nestrčí do drahého středu. Jinými slovy, soupeř strčí do nějaké helmice v jedné z pyramid, my strčíme do stejné helmice ve druhé pyramidě.
3. Když soupeř strčí do drahého středu, začneme hrát hladově. To znamená, že vždy shodíme tu helmici, která právě obsahuje nejméně kamínků.

Dokazování vezměme trochu na přeskáčku. Začněme tím, že opravdu můžeme hrát podle druhého bodu. Pyramidy vzniklé po našem prvním tahu totiž mají průnik (překryv) a nemusí být jasné, že pokud soupeř strčí do něčeho z tohoto průniku, můžeme kopírovat.

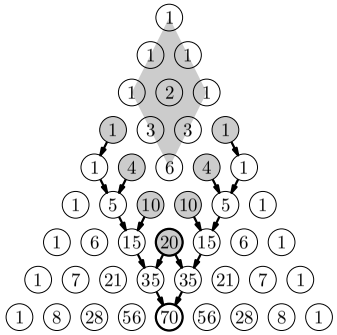
Ten průnik ale není velký: obsahuje drahý střed a případně (je-li N sudé) dvě helmice pod ním. Drahý střed máme ošetřený zvlášť. Helmice nikdy neshodí helmici stojící vedle ní, takže shodí-li soupeř jednu, my shodíme druhou. Kopírovat tedy můžeme vždy.

To, že každý tah kromě strčení do drahého středu umíme zkopírovat, je důležité. Právě proto to musí být soupeř, kdo strčí do drahého středu, případně ho shodí spolu s helmicí pod ním.

Jak se kopírovací část strategie projeví na počtu sebraných kamínků? Určitě platí, že když okopírujeme tah, sebereme maximálně tolik, kolik sebral soupeř. Ve skutečnosti sebereme ostře méně právě tehdy, když soupeř shodí něco z průniku pyramid. V takovém případě ale určitě shodil drahý střed. To dohromady znamená, že po konci kopírovací strategie se rozdíl našich a soupeřových kamínků zvýší alespoň o hodnotu drahého středu.

Nesmíme ale zapomínat, že před kopírovací částí jsme sami sebrali levný střed. My ovšem ukážeme, že součet kamínků v helmicích, které shodíme shozením levného středu, je o jedna menší než hodnota drahého středu. Když tohle platí, máme po kopírovací strategii alespoň o jeden kamínek méně než soupeř.

Indukcí podle r budeme dokazovat obecnější tvrzení: Počet kamínků získaných shozením střední helmice v r-tém lichém řádku shora je o jedna menší než počet kamínků ve střední helmici na $(r+1)$-ním lichém řádku.



Pro $r = 1$ naše tvrzení evidentně platí ($1 = 2 - 1$). Indukční krok sledujme na obrázku. Pokud shodíme střed 3. lichého řádku, což je číslo 6, spadne celý „diamant“ a z indukce už víme, že jeho součet je 19.

Shodíme-li střed o řádek níž, tedy číslo 20, spadnou navíc dvě diagonály vedoucí od okraje ke středu 20. Šipky na obrázku ukazují, že každá z těchto diagonál se sečte na 35, takže obě diagonály dohromady na 70, což je přesně střed následujícího lichého řádku.

Ovšem číslo 20 leží v obou diagonálách, takže jsme ho započítali dvakrát. To je ale správně: jednou reprezentuje samo sebe, podruhé součet diamantu nad diagonálami zvětšený o jedničku. Analogická úvaha funguje pro libovolné r.

Hurá! Tím jsme ukázali, že po kopírovací části bude mít soupeř alespoň o jeden kamínek víc. Pokud ale soupeř shodí drahý střed přímo, pokračujeme třetí částí (hladovou). Teď musíme ukázat, že i z téhle části získáme nejvýše tolik, kolik soupeř.

existuje samozřejmě velmi mnoho dalších parametrů, které mají vliv na zpracování signálů. Učení probíhá změnou vlastností konektomu, například přidáváním nových synapsí nebo změnami parametrů těch synapsí, co už existují. Když synapticky spojené neurony pálí společně, jejich spojení se posiluje.

Umělé neuronové sítě modelují skutečnost s různou úrovní detailu a mají mnoho různých aplikací, podle kterých se odvíjí jejich architektura. Začneme od nejjednodušího modelu, který dělá něco užitečného.

Umělý neuron

Namísto posílání pulzních signálů budeme reprezentovat informaci pomocí čísel. Umělý neuron je pro nás jednotka, která má n číselných vstupů $x_1, \ldots x_n$ a jeden výstup y.

Neuron bude ze svých vstupů počítat *potenciál* ξ. Některé vstupy budou k potenciálu přispívat pozitivně, některé negativně. Výstup y bude záviset na potenciálu podle takzvané *aktivační funkce*, značené a: $y = a(\xi)$. Směr příspěvku, tedy míra excitace nebo inhibice, bude pro každý vstup jiná a bude určená takzvanou *vahou* (*weight*). Váha i-tého vstupu se značí w_i.

Standardně se potenciál počítá jako součet vážených příspěvků všech vstupů. K tomuto součtu se také přidá takzvaný *bias* (český překlad by mohl být „předsudek“ nebo „sklon“) ϑ:

$$\xi = \vartheta + \sum_i w_i x_i.$$

Aktivační funkce se používají různé. Vždy jsou definované na celém $\mathbb{R}$, omezené (většinou na $[-1;1]$ nebo $[0;1]$) a rostoucí (respektive neklesající). První model, který si ukážeme, bude vracet 1, pokud je $\xi \geq 0$, a jindy -1 (tedy jeho aktivační funkce je funkce signum).

Perceptron

Takovému umělému neuronu se říká *perceptron*. Jeho výstup je 1, pokud $\vartheta + \sum_i w_i x_i \geq 0$, a jindy -1. Jsou-li vstupy perceptronu dva, je $\vartheta + \sum_i w_i x_i = 0$ rovnice přímky ve dvou rozměrech. Perceptron touhle přímkou rozděluje dvojrozměrný prostor vstupů na dvě poloroviny. V jedné vrací 1 a ve druhé vrací -1. Přidáme-li další vstupy, analogicky perceptron dělí n-rozměrný vstupní prostor na dva poloprostory.

Perceptron lze použít jako jednoduchý klasifikátor dvou kategorií, pro něž bude jeho očekávaný výstup 1 a -1. Rozdělíme si vstupní příznaky trénovacích dat do *pozitivní kategorie* $\mathcal{P}$ a do *negativní kategorie* $\mathcal{N}$.

Ukážeme si *algoritmus perceptronového učení*, o kterém víme, že pokud jdou kategorie $\mathcal{P}$ a $\mathcal{N}$ od sebe bez chyb rozdělit perceptronem (neboli nějakou nadrovinou), pak náš algoritmus nakonec najde váhy nějakého takového perceptronu.

Nejdříve ke všem vzorkům v $\mathcal{P}$ a $\mathcal{N}$ přidáme jako novou složku na začátek jedničku. Tím pádem můžeme zapomenout na bias – stane se z něho nová první váha. Vzorky „vylepšené“ o jedničky označíme jako $\mathcal{P}'$ a $\mathcal{N}'$.

Potom sjednotíme vzorky do jedné množiny. $\mathcal{N}'$ mají být vzorky, pro které $\sum_i w_i x_i < 0$, a to platí právě tehdy, když $\sum_i w_i \cdot (-x_i) > 0$. Vyrobíme si množinu $\mathcal{R}$, která se bude skládat z $\mathcal{P}'$ a minus jedničkou vynásobených vektorů z $\mathcal{N}'$. Perceptron, ve kterém jsou všechny vektory v $\mathcal{R}$ v pozitivní polorovině, bude správně klasifikovat $\mathcal{P}'$ i $\mathcal{N}'$.

Zinicializujeme perceptron libovolnými vahami, třeba nulovými. Učení probíhá tak, že iterujeme přes množinu $\mathcal{R}$ a postupně perceptron učíme jednotlivé vzorky x. Chceme po něm, aby na každém vzorku bylo $\sum_i w_i x_i$ větší než 0.

Pokud na zkoumaném vzorku je $\sum_i w_i x_i > 0$, je vzorek perceptronem rozpoznaný správně, neděláme nic a jdeme na další vzorek. Pokud je suma menší než 0, pak ke každé váze w_i přičteme $x_i \cdot \gamma$ a jdeme se učit další vzorek. γ je tady opět rychlost učení, konstanta mezi 0 a 1. Pokud najdeme perceptron, který správně klasifikuje všechny vstupy, vrátíme ho. Pokud jenom chceme dobrý perceptron, který ne nutně klasifikuje všechno dobře, můžeme místo toho jenom běžet, dokud nám nedojde čas, a pak vrátit ten perceptron, který dokázal klasifikovat nejvíc vzorků po sobě správně.

Úkol 5 [6b]: Stáhněte si ze stránky seriálu dataset o kosatcích. Tento slavný „Iris dataset“ poprvé použil v 30. letech významný biolog a statistik Ronald Fischer. Každý řádek reprezentuje vlastnosti jednoho kosatce. První řádek popisuje význam sloupečků: první dva jsou délky a šířky kališních lístků, další dva jsou délky a šířky okvětních lístků. Máme za úkol předpovědět poslední sloupec – konkrétní druh kosatce: `setosa`, `versicolor`, nebo `virginica`.

Zkuste naprogramovat rozpoznávání kosatců pomocí perceptronů. Jeden perceptron dokáže od sebe rozpoznat jenom 2 třídy, budete muset být kreativní. Není jediné správné řešení – překvapte nás! Jakou zvládnete accuracy na testovacích datech? Naše autorské řešení umí z 60 trénovacích vzorků mít na zbytku datasetu 93.42 %.

Perceptrony jsou užitečné, ale pořád docela slabé – pokud mají kategorie „složitou hranici“ a nejdou lineárně oddělit, potřebujeme na jejich naučení našim modelům povolit složitější strukturu.

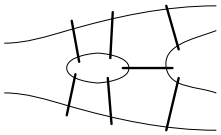
Zábavné věci se začnou dít třeba, když začneme neurony *vrstvit* do takzvaných *dopředných vrstevnatých neuronových sítí* (*feedforward neural networks*). První vrstva takové sítě jsou vstupní data. Druhá vrsta konzumuje výstup první vrstvy jako svůj vstup. Sedí v ní kupříkladu 10 neuronů a každý z nich má vlastní váhy a transformuje vstupy jiným způsobem. Výstupy druhé vrstvy jsou vstupy pro třetí vrstvu, a tak dále až do výstupní vrstvy. Informace se šíří jenom z nižších vrstev do vyšších. Vyšší vrstvy jsou schopné počítat složitější a složitější transformace vstupních dat.

Michal Pokorný

Recepty z programátorské kuchařky: Eulerovské tahy

Historický problém

V roce 1735 se švýcarskému matematikovi Leonhardu Eulerovi na stůl dostal na první pohled jednoduchý problém, který mu předložil starosta města Královec (dnešní Kaliningrad). Královcem teče řeka Pregola, na ní je několik ostrovů a ostrovy jsou spojeny se zbytkem města mosty. Dobová ilustrace situaci vystihla takto (schematická kresba):



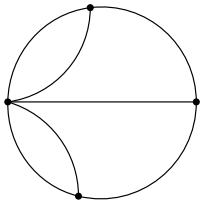
Pan starosta se pana matematika v dopise tázal, jestli je možné začít na některém z břehů (nebo ostrovů) a udělat si vycházku po městě tak, že se každým mostem projde právě jednou. Navíc chtěl procházku skončit na kusu suché země, ze kterého vyšel.

Profesor Euler jej nejprve chtěl poslat k šípku – problém jde snadno vyřešit rozborem případů, což by zvládli i tehdejší studenti střední školy (natož pak ti dnešní). Zachoval se ovšem jako pravý matematik – přišel na to, jak problém zobecnit, a mistrně vyřešil hádanku i pro všechna možná města, která kdy budou chtít pořádat podobné procházky.

Eulerovský tah

Pojďme si nyní problém popsat abstraktně a tím si připomenout grafovou terminologii. Vrcholy našeho grafu jsou kusy pevniny, ať už to budou části města nebo ostrovy. Mezi dvěma vrcholy povede hrana, pokud jsou spojeny mostem, a onen most odpovídá hraně.

V tomto zadání má smysl uvážit, že mezi dvěma kusy pevniny povede mostů více – například v Praze jich vede tolik, že se na to ptají v leckteré zeměpisné olympiádě. Graf, kde mezi vrcholy vede více hran, nazýváme *multigraf,* a pokud dvě hrany vedou mezi stejnými vrcholy, mluvíme o nich jako o *paralelních* hranách.



Obecná procházka v grafu z vrcholu A do vrcholu B (posloupnost hran taková, že cílový vrchol předchozí hrany je počáteční vrchol hrany následující) se nazývá *sled* z A do B. Ve sledu se mohou opakovat jak hrany, tak vrcholy; sled tedy není řešením našeho problému (ve sledu je možné se vrátit po hraně, ze které jsme právě přišli).

Pro naši úlohu se hodí posloupnost hran taková, že vrcholy se opakovat mohou, ale hrany nikoli. Této posloupnosti se říká *tah* z A do B. Kdyby se neopakovaly ani vrcholy, pak posloupnost označujeme jako *cestu.* Tah (respektive sled) je *uzavřený,* pokud začíná v A a končí také v A.

Podíváme-li se tedy na mapu Královce jako na multigraf, ptáme se, zdali existuje uzavřený tah takový, že každou hranu navštíví právě jednou. Takovému tahu pak říkáme *uzavřený eulerovský.*

Mimochodem, tahu se „tah“ neříká jen tak náhodou. Děti se často ve školce překonávají v umění nakreslit obrázek jedním tahem, aby se tužkou nemuselo vracet po už nakreslené čáře. Pokud si obrázek představíme jako graf (čáry jsou hrany, místa jejich setkání vrcholy), pak eulerovský tah nalezneme jen v tom obrázku, který lze nakreslit jedním tahem. V uzavřeném eulerovském tahu se pak vrátíme i do místa, kde jsme začali.

Podmínky tahu

Je na čase poodhalit řešení našeho problému s eulerovským tahem. Půjdeme na to jako matematici – nejprve ukážeme *nutnou* a hned nato *postačující* podmínku. Nutná vlastnost grafu je taková, že bez ní eulerovský tah není možné najít; postačující vlastnost je ta, se kterou vždy eulerovský tah najít umíme. Jsou-li obě podmínky stejné, pak se jedná o ekvivalenci, a tak tomu bude i nyní.

Představme si, že jsme kouzlem nějaký uzavřený eulerovský tah našli, ať už je jakýkoli. Vždy, když se dostaneme do jednoho vrcholu (a není důležité, jestli už jsme v něm byli, nebo ne), tak z něj musíme hned také odejít, abychom tah uzavřeli. A protože tah je eulerovský, každou hranou projdeme jen jednou, takže tyto dvě hrany (tu příchozí a odchozí) už nepoužijeme. U každého vrcholu mimo výchozí tedy platí, že hrany tvoří dvojice – jedna, co vedla dovnitř, a jedna, která z něj vedla ven.

Podobná věc platí i pro startovní vrchol. Sice do něj nevstoupíme poprvé pomocí hrany, takže počet navštívených hran u něj bude stále lichý – ale jen do chvíle, než se do něj naposledy vrátíme a skončíme, protože skončením jsme použili poslední hranu, která bude tvořit dvojici s hranou první.

Jakou vlastnost grafu jsme odhalili? Neplatí, že graf má sudý počet hran (protože trojúhelník jedním tahem nakreslíme a přesto má 3 hrany), ale platí, že do každého vrcholu vede sudý počet hran, tedy že graf má *všechny stupně sudé.* Nezapomeňme také na to, že graf musí být souvislý – dva oddělené obrázky jedním tahem bez zvednutí tužky nenakreslíme. Máme nutné podmínky!

Nalezení tahu

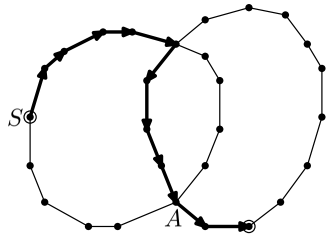
Zbývá tedy ověřit, že podmínky jsou i postačující. Mějme souvislý graf, který má všechny stupně sudé. Umíme v něm vždy najít uzavřený eulerovský tah? Ověřme to, jak se na informatiky patří – algoritmem.

Předložený algoritmus je založený na vylepšeném prohledávání do hloubky, tedy DFS, o kterém jste si mohli přečíst v první grafové kuchařce.[8]

Vyberme si vrchol, v něm začneme. Náš algoritmus musí umět označovat hrany jako „probrané“, jako to dělá DFS. Vyberme si tedy jednu hranu a pokračujme dále, zatím bez vypisování.

Po nějakém tom procházení se jistě stane, že jsme se zastavili – vrchol už nemá žádné nepoužité hrany. Nutně to znamená, že to je ten vrchol, ve kterém jsme začínali. V procházení do hloubky se vracíme zpět, ale my k tomu přidáme vypisování cesty – postupně pozpátku vypisujeme hrany, kterými se vracíme zpět v prohledávání.

[8] http://ksp.mff.cuni.cz/viz/kucharky/grafy



Na obrázku výše je příklad právě probíhajícího algoritmu. Začal ve zvýrazněném vrcholu vlevo, procházel po šipkách až do bodu A, kde volil hrany tak, že hned skončil na začátku. Dále pokračoval vypisováním hran pozpátku, až došel zase do bodu A. Zde si vybral jednu ještě nepoužitou hranu a po ní prošel celou druhou kružnici – zbytek hran – zpět do bodu A. Nyní vypisuje hrany pozpátku od bodu A.

Buď tímto výpisem dojdeme až na začátek, nebo se dostaneme do vrcholu, který má ještě nějaké nepoužité hrany (situace může vypadat třeba jako na obrázku). Potom vypisování zastavíme a pokračujeme v prohledávání DFS přes nepoužitou hranu. I tam se to může zastavit (a zastaví), i tam začneme vypisovat pozpátku. Nakonec dojdeme do původního místa rozbočení, a budeme opět pozpátku vypisovat hrany, které nás nakonec dostanou až na počátek, kde skončíme.

Najde tento algoritmus opravdu korektní uzavřený eulerovský tah? Graf byl souvislý a o algoritmu DFS se ví, že v takovém případě navštíví každou hranu právě jednou. Algoritmus opravdu vypisuje cyklus – jen je u něj trochu zvláštní způsob, jak ho vypisuje. Když dojde na křižovatku s ještě nepoužitými hranami, tak výpis zastaví, tiše po nich kráčí, označuje si je a vypisuje, až když se po nich vrací. Ověřme si, že hrany opravdu navazují.

V duchu argumentů z předcházející části víme, že jediný vrchol grafu s lichým počtem nepoužitých hran je právě ona křižovatka – a algoritmus DFS prochází graf podobně, jako jsme ho procházeli v minulé sekci, takže právě do tohoto vrcholu algoritmus dojde, až se průchod touto částí grafu zastaví.

Jakmile sem program dojde (a nezbudou mu volné hrany), začne cestovat zpět a hrany vypisovat – a opravdu, pokračuje se tedy z místa, kde naposledy přestal, a program vskutku vypíše tah přes všechny hrany v grafu – uzavřený eulerovský tah.

Věta o eulerovském tahu v celé své kráse tedy zní: *(Multi)graf obsahuje uzavřený eulerovský tah právě tehdy, když má všechny stupně sudé a je souvislý.*

Je třeba podotknout, že složitost našeho algoritmu na bázi DFS je lineární vůči velikosti grafu (počtu vrcholů a hran). Existují i jiné algoritmy pro hledání eulerovského tahu, jedna varianta například prochází grafem a vybírá si na křižovatkách takové hrany, které souvislost grafu pokud možno nepoškodí. Tyto algoritmy už nemusí mít nutně lineární časovou složitost.

Jiné druhy procházek

Nejen kreslením obrázků ze stejného bodu živ je člověk. Co kdybychom mohli začít a skončit v jiném místě, tedy ptali se po neuzavřených eulerovských tazích, změnilo by se něco? Není tomu tak, pouze nutné a postačující podmínky si vyžádají, aby všechny vrcholy měly sudý stupeň až na právě dva vrcholy, které mají lichý stupeň. Pokud nám to nevěříte, zkuste si to rozmyslet sami, opravdu to není těžké.

Smysl také dává zkusit najít ne uzavřený tah, ale uzavřenou cestu – uzavřenou cestu přes všechny vrcholy, která navštíví každý vrchol právě jednou (říká se jí „Hamiltonovská cesta“). Bohužel, ačkoli jsou problémy příbuzné, musíme vás zklamat – není znám žádný efektivní (polynomiální) algoritmus na tento problém, a kdyby jej někdo z vás nalezl, vyřešil by otázku „P vs. NP“, o níž se více dočtete v kuchařce o těžkých problémech.[9]

V matematice se také někdy zmiňují „náhodné procházky“ po grafech – můžete si je představit tak, že se po mostech města Královce motá opilec, který si hází (opilou nebo spravedlivou) mincí a podle toho se rozhoduje, přes který most jít dál. Použití mají tyto modely hlavně v matematické teorii grafů a teorii pravděpodobnosti. O tom si můžeme povědět zase někdy jindy.

Martin Böhm

[9] http://ksp.mff.cuni.cz/viz/kucharky/tezke-problemy