Korespondenční Seminář z Programování

35. ročník **KSP** **Únor 2023**

Milí řešitelé, řešitelky a řešitelčata!

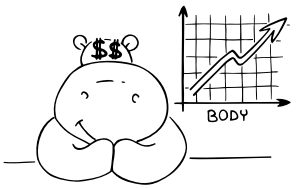
Dostává se k vám čtvrté číslo hlavní kategorie 35. ročníku KSP.

Letos se můžete těšit v každé z pěti sérií hlavní kategorie na **4 normální úlohy**, z toho alespoň jednu praktickou opendatovou. Také na **kuchařky** obsahující nějaká zajímavá informatická témata, hodící se k úlohám dané série. Občas se nám také objeví **bonusová X-ková úloha**, za kterou lze získat X-kové body. Kromě toho bude součástí sérií **seriál** na lineární algebru.

Autorská řešení úloh budeme vystavovat hned po skončení série. Pokud nás pak při opravování napadnou nějaké komentáře k řešením od vás, zveřejníme je dodatečně.

Odměny & na Matfyz bez přijímaček

Za úspěšné řešení KSP můžete být **přijati na MFF UK bez přijímacích zkoušek.** Úspěšným řešitelem se stává ten, kdo získá za celý ročník (hlavní kategorie) alespoň 50 % bodů. Za letošní rok půjde získat maximálně 300 bodů, takže hranice pro úspěšné řešitele je 150. Maturanti pozor, pokud chcete prominutí využít letos, musíte to stihnout do konce čtvrté série, pátá už bude moc pozdě. Také každému řešiteli, který v tomto ročníku z každé série dostane alespoň 5 bodů, darujeme KSP propisku, blok, nálepku na notebook a možná i další překvapení.



Termín série: **9. dubna 2023 ve 32:00 (tedy další ráno v 8:00)**

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

Značky úloh: Lehčí úloha (či její část) vhodná pro začátečníky; Praktická open-data úloha; Úloha, u které doporučujeme začíst se do kuchařky; Seriálová úloha

Odměna série: **Sladkou odměnu** si vysloužíte, když nám **napíšete řešení 1 teoretické úlohy nebo celého seriálu v angličtině.** Pro inspiraci nabízíme anglické řešení části 3. série.[1] Kdybyste si nebyli jistí terminologií, napište nám na *english@ksp.mff.cuni.cz,* rádi poradíme.

Čtvrtá série třicátého pátého ročníku KSP

35-4-1 Golfový turnaj — 14 bodů

Bjørn sleduje golfový turnaj a obdivuje hráče, kteří pár údery dopraví míček do vzdálené jamky. Hned by si to také zkusil, ale přestavět školní hřiště na golfové mu loni zakázali a do jeho pokojíku se nevejde ani jedna jamka s greenem.

Napadlo ho zahrát si s kamarády programátorský golf. Vymyslí si jednoduchou úlohu a budou se snažit napsat co nejkratší program, který ji vyřeší. Pojďte si také zahrát.

Programovací jazyk

Náš programovací jazyk počítá s celými čísly. Jediná paměť, kterou má k dispozici, je *zásobník.* Ten si můžete představit jako posloupnost čísel uspořádanou shora dolů. Nová čísla přidáváme na *vrchol* zásobníku, odebíráme opět z vrcholu.

Například chceme-li sečíst $1 + 2$, napíšeme `12a`. To je program o třech instrukcích: Instrukce `1` uloží na vrchol zásobníku číslo 1. Podobně instrukce `2` uloží 2. Poslední instrukce `a` (add) odebere dvě čísla z vrcholu zásobníku, sečte je a výsledek opět uloží na zásobník. Na konci výpočtu tedy bude zásobník obsahovat jediné číslo 3.

Každá instrukce je tvořena jedním znakem. Velká a malá písmena nerozlišujeme. Mezery, tabulátory a konce řádků mezi instrukcemi jsou ignorovány. Některé instrukce používají kulaté závorky pro vymezení bloku programu.

Omezení: Program smí obsahovat nejvýše 1000 instrukcí. Po spuštění musí doběhnout do 1 milionu kroků (jeden krok odpovídá vyhodnocení jedné instrukce nebo kulaté závorky). Na zásobníku může být současně maximálně 1000 čísel. Čísla jsou 32-bitová se znaménkem, přetečení způsobí běhovou chybu.

K dispozici jsou následující instrukce. Vždy uvádíme název instrukce, jméno pro snazší zapamatování, stav zásobníku před provedením instrukce a po něm (∅ značí „nic“) a nakonec chování instrukce.

`0` (const) $\quad \emptyset \to 0$
uloží na zásobník konstantu 0 (podobně 1–9)

`a` (add) $\quad x\ y \to x + y$
sečte dvě čísla

`c` (copy) $\quad n \to x$
zkopíruje n-tou hodnotu od vrcholu zásobníku, přičemž nultá je ta těsně pod n (`9870c` zkopíruje 7, `9872c` zkopíruje 9)

`d` (dup) $\quad x \to x\ x$
duplikuje hodnotu na vrcholu zásobníku

`e` (equal) $\quad x\ y \to x = y$
uloží 1, pokud $x = y$, jinak 0

`g` (greater) $\quad x\ y \to x > y$
uloží 1, pokud $x > y$, jinak 0

[1] `https://ksp.mff.cuni.cz/viz/35-3/reseni-en`

`i` (if) *podmínka* $\to \emptyset$
podmínka: (*instrukce*)`i` přečte hodnotu ze zásobníku a pokud je nenulová, provede *instrukce*

`k` (count) $\emptyset \to n$
spočítá, kolik čísel bylo na zásobníku před provedením

`l` (less) $x\ y \to x < y$
uloží 1, pokud $x < y$, jinak 0

`m` (mul) $x\ y \to x \cdot y$
vynásobí dvě čísla

`o` (over) $n\ x \to \emptyset$
přepíše n-tou hodnotu od vrcholu zásobníku číslem x (např. `98723o` změní 9 na 3, takže na zásobníku zbude 3 8 7)

`p` (pop) $x \to \emptyset$
smaže číslo z vrcholu zásobníku

`q` (quotient) $x\ y \to x/y$
spočítá podíl čísel x a y

`r` (remainder) $x\ y \to x \bmod y$
spočítá zbytek po dělení čísla x číslem y

`s` (sub) $x\ y \to x - y$
odečte dvě čísla

`t` (trace) $\emptyset \to \emptyset$
vypíše aktuální stav programu (k dispozici jen v simulátoru)

`w` (while) $\emptyset \to \emptyset$
cyklus: (*podmínka*)(*tělo*)`w` provede posloupnost instrukcí *podmínka*, pak odebere jedno číslo ze zásobníku a pokud je nenulové, provede posloupnost instrukcí *tělo* a znovu vyhodnocuje podmínku

`x` (xchg) $x\ y \to y\ x$
prohodí dvě čísla na vrcholu zásobníku

Jazyk si můžete vyzkoušet ve webovém simulátoru.[2]

Příklady

- `123aa` spočítá $1 + (2 + 3)$
- `12a3a` spočítá $(1 + 2) + 3$
- `12a34am` spočítá $(1 + 2) \cdot (3 + 4)$
- `15l(3)i` otestuje, jestli je $1 < 5$, a pokud ano, uloží na zásobník 3
- `0(1ad6l)(d)w` uloží na zásobník čísla 1 2 3 4 5 6

Úkoly

1. Vytvořte na zásobníku číslo 2023. *(2 body)*
2. Na zásobníku dostanete nějaká čísla (alespoň jedno). Nahraďte je jejich součtem. *(2 body)*
3. Na vrcholu zásobníku je číslo $y \geq 0$, pod ním x. Nahraďte je mocninou x^y. *(2 body)*
4. Na zásobníku dostanete dvě kladná čísla. Nahraďte je jejich největším společným dělitelem. *(2 body)*
5. Na zásobníku dostanete číslo $1 \leq n \leq 100$. Nahraďte jej n-tým nejmenším prvočíslem. Například pro vstup 3 je správný výstup 5. *(3 body)*
6. Na zásobníku dostanete posloupnost $1 \leq n \leq 20$ čísel. Seřaďte ji od nejmenšího (na dně) po největší (na vrcholu). *(3 body)*

Odevzdávání

Toto je praktická úloha. Každý úkol se chová jako jeden test. Vstupní soubor obsahuje jen číslo úkolu. Jako výstup odevzdejte svůj program. Vyzkoušíme ho na našich testovacích datech a pokud odpoví správně, přidělíme mu body. Počet bodů záleží na délce programu měřené ve znacích (mezery nepočítáme). Program stejně dlouhý jako naše referenční řešení dostane plný počet bodů. Delší programy dostanou méně, ale aspoň 30 % maxima. Pokud se vám podaří najít kratší program, můžete dostat až 30 % bodů navíc.

[2] `https://ksp.mff.cuni.cz/viz/golf`

35-4-2 Kevinovo velkolepé rozloučení — 9 bodů

Kevina přiletěla navštívit jeho kamarádka Zuzka, která bydlí v Americe. Užili si spolu spoustu zábavy, ale zítra Zuzka odlétá domů. Kevin se chce se Zuzkou rozloučit nějakým pompézním způsobem. Po Zuzčině odjezdu na letiště napsal na střechu svého domu zprávu na rozloučenou, kterou si bude moci Zuzka přečíst z letadla.

Když zprávu dopsal, všiml si, že slova ve zprávě jsou v opačném pořadí. Barva už ale zaschla a nejde smýt. Kevin se tedy rozhodl, že zprávu opraví zpřeházením střešních tašek. Na jedné tašce je jedno písmeno, případně je prázdná jako mezera. Tašky jsou ale těžké a Kevin unese jenom jednu najednou. Navíc nelze tašky odkládat na střechu jen tak bokem, takže má Kevin jedinou možnost: vždy si vybrat dvě tašky a ty prohodit.

Pomozte Kevinovi vymyslet algoritmus, který zjistí, jak přehodit pořadí slov ve zprávě. Na střeše se však nenachází žádný počítač, takže Kevin jej bude vyhodnocovat z hlavy. Zvládne si zapamatovat jen konstantní množství čísel, velkých řádově jako délka zprávy.

Váš algoritmus tedy smí používat jen konstantní množství paměti. Může se zeptat na písmeno na dané pozici a vypsat indexy dvou tašek, které má Kevin prohodit. Hledáme algoritmus s co nejkratším časem výpočtu.

Toto je teoretická úloha. Není nutné ji programovat, odevzdává se pouze slovní popis algoritmu. Více informací zde: `http://ksp.mff.cuni.cz/viz/tinfo`

35-4-3 Házení — 11 bodů

Robert se rozhodl zapsat si jako tělocvik basketbal a první hodinu semestru začali házením na koš. Jenže to není tak jednoduché. Z Vánoc ještě zbyly v tělocvičně zavěšené různé ozdoby a nikdo se nenamáhal je sundat. Robert tak musí nejen hodit míč tak, aby skončil v koši, ale i tak, aby v letu nesrazil žádnou z ozdob. Jelikož bude na koš házet ještě další hodinu, tak by se u toho rád co nejméně nadřel, tedy aby počáteční rychlost, kterou míč hodí, byla co nejmenší.

Jenže to už je docela oříšek a Robertovi se po rozcvičce sestávající z běhání a klikování nechce přemýšlet, takže je to na vás.

Situaci si můžeme představit jako 2D mřížku, kde x-ová osa je horizontální spojnice mezi Robertem a košem a y-ová osa jde do výšky. Vzdálenost mezi sousedními body mřížky odpovídá 1 m.

Robert hází z bodu $(0, 0)$ a koš se nachází v bodě $(M, 0)$, tedy předpokládáme, že Robert bude házet ze stejné výšky, jako je koš. Zároveň se v místnosti nachází N ozdob, které si můžeme představit jako obdélníky s rohovými políčky v bodech (x_1, y_1), (x_1, y_2), (x_2, y_1) a (x_2, y_2) – tedy hrany obdélníka jsou zarovnané s osami.

Míč si můžeme představit jako hmotný bod, který je vržen pod úhlem α s počáteční rychlostí V_0 (formálně V_0 je absolutní hodnota vektoru počáteční rychlosti). Míč se dále

pohybuje v souladu s fyzikálními zákony, kdy vliv odporu vzduchu můžeme zanedbat. Tíhové zrychlení uvažujeme $9.81\,\mathrm{ms}^{-2}$.

Hod je považován za platný, pokud míč netrefí žádnou z ozdob a skončí v koši. Trajektorie míče tedy musí procházet bodem $(M, 0)$ a nesmí projít vnitřkem žádného obdélníku určujícího ozdobu. Dotknout se hrany může.

Najděte α a V_0 takové, že hod s těmito počátečními podmínkami je platný a zároveň V_0 je minimální možné.

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

Formát vstupu: Na prvním řádku dostanete číslo T udávající počet hodů, které máte vyřešit.

Pro každý hod pak na prvním řádku dostanete mezerou oddělená celá čísla N a M – počet ozdob a x-ovou souřadnici koše. Na každém z následujících N řádků dostanete čtyři celá čísla x_{1i}, y_{1i}, x_{2i}, y_{2i} oddělená mezerou, která udávají souřadnice rohů ozdoby.

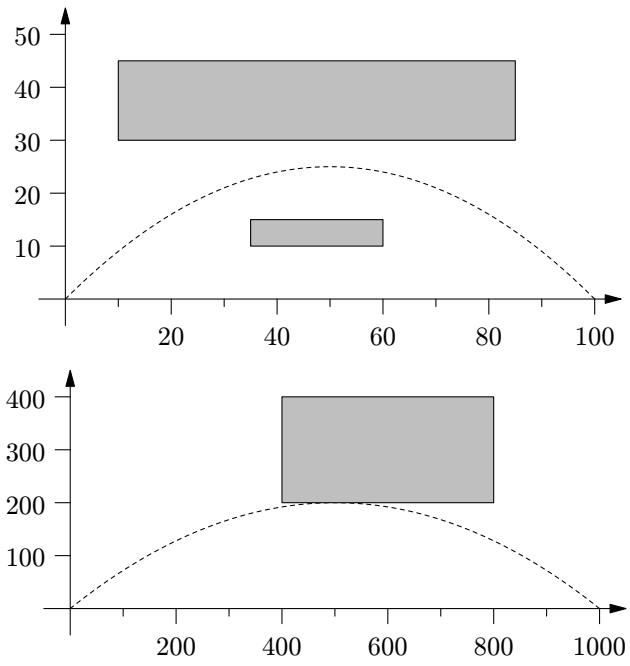
Formát výstupu: Na výstup vypište dvě mezerou oddělená čísla α (ve stupních) a V_0 (v ms^{-1}). Pozor, tato čísla téměř určitě nebudou celá, dbejte tedy na jejich přesnost. Vaše řešení bude uznané za správné, pokud relativní odchylka od vzorového řešení bude nejvýše 0.01 %.

Ukázkový vstup:

```
2
2 100
35 10 60 15
10 30 85 45
1 1000
400 200 800 400
```

Ukázkový výstup:

```
45 31.321
38.660 100.276
```



35-4-4 Hroší růže — 11 bodů

Filip má za domem N políček posetých růží hroší (*Rosa chinensis* 'Hippo'). Čaj z jejích květů (též nazývaný hrůžový) navrací vzpomínky na dávné květnové večery.

Hroší růže je poměrně náročná rostlina, takže je potřeba ji zavlažovat téměř každý den. Při té příležitosti se Filip rozhodl ze zalévání políček udělat zábavnou činnost.

Postavil systém M skluzavek, kde i-tá skluzavka vede z v_i-tého políčka na w_i-té políčko. Ačkoliv to byla jistě výhodná investice, skluzavky jsou poměrně drahá komodita, a proto *z každého políčka vede nejvýše jedna skluzavka.*

Pokud vede skluzavka z políčka a na políčko b, Filip se po ní může sklouznout. Jelikož ale skluzavky bývají nakloněné, v opačném směru by musel Filip po skluzavce šplhat, čímž by ji ale zašpinil hlínou z políček, a tak to nedělá. Přesto může být systém skluzavek navržený tak, že skluzavka povede také z políčka b na políčko a.

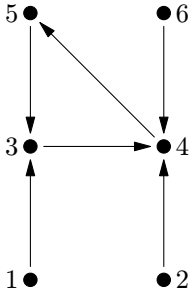
Filipův způsob zalévání probíhá každý den následovně: vybere si počáteční a koncové políčko a následně se mezi nimi klouže po skluzavkách, které vybírá vždy z aktuálního políčka. Po cestě nezapomíná zavlažovat políčka. Takto se dopravuje tak dlouho, dokud neskončí na koncovém políčku.

Filip zjistil, že se v některých případech nezvládne z jednoho políčka doklouzat na druhé. Rád by proto přidal co nejméně skluzavek, aby se mu to podařilo. Pomůžete mu s tím?

Toto je teoretická úloha. Není nutné ji programovat, odevzdává se pouze slovní popis algoritmu. Více informací zde: `http://ksp.mff.cuni.cz/viz/tinfo`

Na vstupu dostanete hodnoty N, M. Můžete předpokládat, že políčka jsou očíslovaná 1 až N. Dále dostanete specifikaci systému skluzavek, což je M dvojic čísel ve formátu v_i w_i.

Na výstup vypište *jediné číslo* – nejmenší počet skluzavek, který je potřeba k tomu, aby se z libovolného políčka šlo doklouzat na libovolné jiné.



Na obrázku nahoře je potřeba doplnit alespoň tři skluzavky. Jedním z řešení je například trojice `5 6`, `1 2` a `6 1`.

35-4-X1 Mnoho jamek — 10 bodů

Toto je bonusová úloha pro zkušenější řešitele, těžší než ostatní úlohy v této sérii. Nezískáte za ni klasické body, nýbrž dobrý pocit, že jste zdolali něco výjimečného. Kromě toho za správné řešení dostanete speciální odměnu a body se vám započítají do samostatných výsledků KSP-X.

Tato úloha vychází z první úlohy v této sérii *Golfový turnaj*. Odevzdávání a hodnocení funguje stejným způsobem.

Na zásobníku dostanete posloupnost $1 \leq n \leq 800$ čísel. Seřaďte ji od nejmenšího (na dně) po největší (na vrcholu).

Připomínáme, že v jedné chvíli se na zásobníku smí nacházet maximálně 1000 čísel a program musí doběhnout do 1 milionu kroků.

35-4-S Skalární součin — 15 bodů

Právě čtete čtvrtý díl seriálu. Pokud jste předchozí díly neřešili, pro pochopení následujících odstavců je vhodné si je přinejmenším přečíst. Navíc je stále možné odevzdávat úlohy z nich za polovinu bodů.

Transpozice

Transpozici jsme již nakousli v prvním dílu. Nyní ji budeme používat více, proto si ji nadefinujeme pořádně.

Máme-li matici $\mathbf{A}$ o rozměrech $m \times n$, její transpozice $\mathbf{A}^\top$ je matice $n \times m$. Vznikne překlopením matice $\mathbf{A}$ podél *hlavní diagonály*, tedy diagonály vedoucí z levého horního rohu. Prvek na pozici (i, j) se v transponované matici bude nacházet na pozici (j, i).

$$\begin{pmatrix} \mathbf{1} & 2 & 3 & 4 \\ 5 & \mathbf{6} & 7 & 8 \\ 9 & 10 & \mathbf{11} & 12 \end{pmatrix}^\top = \begin{pmatrix} \mathbf{1} & 5 & 9 \\ 2 & \mathbf{6} & 10 \\ 3 & 7 & \mathbf{11} \\ 4 & 8 & 12 \end{pmatrix}$$

Skalární součin

Dosud neumíme zjistit, jakou polohu vůči sobě mají dva vektory, například jaký svírají úhel. S pomocí skalárního součinu to zvládneme, cesta však bude dlouhá a klikatá. Objevíme na ní však i další, možná ještě hezčí geometrické vlastnosti.

Skalární součin dvou vektorů budeme značit $\langle \mathbf{x}, \mathbf{y} \rangle$. Spočítáme jej tak, že vektory vynásobíme po složkách a výsledky sečteme. Pochopitelně musí mít oba vektory stejné rozměry. Dostaneme skalár, odtud také pochází název. Matematicky:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} (\mathbf{x})_i (\mathbf{y})_i = \mathbf{x}^\top \mathbf{y}$$

Poslední podoba využívá maticové násobení. Vektory jsou standardně sloupcové, transpozicí vektoru $\mathbf{x}$ s n složkami vznikne matice $1 \times n$.

Snadno nahlédneme následující rovnosti:

1. $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$
2. $\langle \mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle$
3. $\langle a\mathbf{x}, \mathbf{y} \rangle = a \langle \mathbf{x}, \mathbf{y} \rangle$

Díky symetrii (1) platí linearita (2, 3) i pro druhý vektor v součinu.

Přidáme ještě čtvrtou vlastnost: $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$. Proč platí? Při násobení složek vznikají druhé mocniny, které jsou nezáporné. Navíc to znamená, že nulový výsledek nastane jen pro $\mathbf{x} = \mathbf{0}$.

Skalární součin se většinou definuje axiomaticky. Nepředepisujeme tedy konkrétní vzorec, ovšem povolíme jakýkoli vztah splňující uvedené vlastnosti. My se však omezíme na ten výše uvedený, tedy *standardní* skalární součin.

Norma

Nezápornost skalárního součinu vektoru se sebou samým nám umožní definovat *normu* neboli velikost vektoru:

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$$

Pro standardní skalární součin norma odpovídá běžné délce, kterou bychom změřili pravítkem.

Občas vektor používáme jen pro určení směru a na jeho velikosti nám nezáleží. Pak se může hodit vektor *normalizovat* tak, aby měl jednotkovou délku. Toho dosáhneme jednoduše, vektor vydělíme jeho normou.

Také můžeme změřit vzdálenost vektorů jako normu jejich rozdílu $\|\mathbf{x} - \mathbf{y}\|$.

Kolmost

K měření úhlů nám stále kus chybí, ovšem nyní se naučíme poznat alespoň ten pravý. Tvrdíme, že vektory $\mathbf{x}$ a $\mathbf{y}$ jsou na sebe kolmé právě tehdy, když je jejich skalární součin nulový. Pokud jste se již s tímto tvrzením setkali ve škole, nejspíše vám bylo předloženo bez důkazu. Zde se o něj pokusíme.

Pomůže nám Pythagorova věta. Její znění nejspíše znáte: Pro pravoúhlý trojúhelník s odvěsnami délek a, b a přeponou délky c platí vztah:

$$a^2 + b^2 = c^2$$

Trochu méně známá je věta opačná. Platí-li vztah výše pro délky stran trojúhelníka, svírají odvěsny pravý úhel.

Nyní sestavíme vektorovou podobu této rovnosti. Odvěsny označíme pomocí vektorů $\mathbf{x}, \mathbf{y}$. Přepona pak bude $\mathbf{x} - \mathbf{y}$. Odvoďme její délku:

$$\begin{aligned} \|\mathbf{x} - \mathbf{y}\|^2 &= \langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle \\ &= \langle \mathbf{x}, \mathbf{x} - \mathbf{y} \rangle - \langle \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle \\ &= \langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{x}, \mathbf{y} \rangle - \langle \mathbf{y}, \mathbf{x} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle \\ &= \|\mathbf{x}\|^2 - 2 \langle \mathbf{x}, \mathbf{y} \rangle + \|\mathbf{y}\|^2 \end{aligned}$$

Předpokládejme $\langle \mathbf{x}, \mathbf{y} \rangle = 0$. Pak dle výpočtu výše platí $\|\mathbf{x} - \mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2$ a z opačné Pythagorovy věty vyplývá, že vektory $\mathbf{x}, \mathbf{y}$ jsou sebe kolmé.

Nyní naopak předpokládejme kolmost. Dle Pythagorovy věty $\|\mathbf{x} - \mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2$, což nutně znamená $2 \langle \mathbf{x}, \mathbf{y} \rangle = 0$.

Dokázali jsme oba směry, ekvivalence tedy platí.

Ortogonální báze

Kolmosti ihned využijeme, nejdříve ovšem potřebujeme pár definic. Máme-li systém vektorů takových, že každé dva jsou na sebe kolmé, nazveme ho *ortogonální*. Pokud navíc každý vektor znormalizujeme, dostaneme *ortonormální* systém.

Tento systém samozřejmě může tvořit i bázi. Podmínku lineární nezávislosti dokonce ortonormální systémy splňují automaticky. Důkaz vynecháme, složitý však není.

Ortonormální báze nám zjednoduší hledání souřadnic. Dosud jsme je počítali pomocí Gaussovy eliminace, tedy v čase $\mathcal{O}(n^3)$. Jde to však rychleji.

Uvažme kanonickou bázi, která ortonormalitu splňuje. Souřadnice $[\mathbf{x}]_{\text{kan}}$ najdeme jednoduše, první složka souřadnic odpovídá první složce $\mathbf{x}$ atd. Na totéž lze pohlížet i jinak: První složku souřadnic dostaneme jako skalární součin prvního bazického vektoru s $\mathbf{x}$. I kdybychom bazické vektory přeházeli, stále bychom dostali správný výsledek.

Totéž funguje obecně. Mějme prostor U a jeho ortonormální bázi $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$. Souřadnice $[\mathbf{x}]_B$ budou mít v i-té složce hodnotu $\langle \mathbf{x}, \mathbf{b}_i \rangle$. Zapsáno jako lineární kombinace:

$$\mathbf{x} = \sum_{i=1}^{n} \langle \mathbf{x}, \mathbf{b}_i \rangle \, \mathbf{b}_i$$

Pojďme to dokázat. Vektor $\mathbf{x}$ určitě lze zapsat jako lineární kombinaci bazických vektorů:

$$\mathbf{x} = \sum_{i=1}^{n} a_i \mathbf{b}_i$$

Naším cílem je ukázat, že $a_i = \langle \mathbf{x}, \mathbf{b}_i \rangle$. Rozepišme tedy skalární součin:

$$\begin{aligned}
\langle \mathbf{x}, \mathbf{b}_i \rangle &= \left\langle \sum_{j=1}^{n} a_j \mathbf{b}_j, \mathbf{b}_i \right\rangle \\
&= \sum_{j=1}^{n} a_j \langle \mathbf{b}_j, \mathbf{b}_i \rangle \\
&= \sum_{j \neq i} a_j \langle \mathbf{b}_j, \mathbf{b}_i \rangle + a_i \langle \mathbf{b}_i, \mathbf{b}_i \rangle \\
&= \sum_{j \neq i} a_j \cdot 0 + a_i \cdot 1 = a_i
\end{aligned}$$

Sumu jsme rozdělili na dva případy. Pokud $j \neq i$, jsou vektory $\mathbf{b}_i$ a $\mathbf{b}_j$ kolmé z definice ortonormální báze. Naopak pro $j = i$ dostáváme skalární součin vektoru se sebou samým.

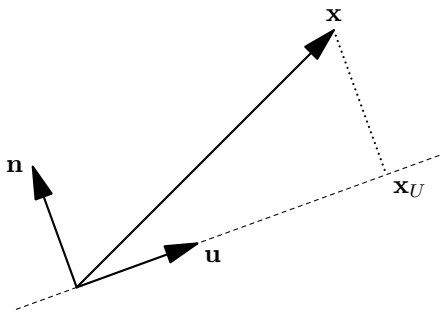
Koeficienty $\langle \mathbf{x}, \mathbf{b}_i \rangle$ se nazývají *Fourierovy koeficienty* a díky nim umíme souřadnice najít v čase $\mathcal{O}(n^2)$.

Projekce

Fourierovy koeficienty nám také odkryjí geometrický význam skalárního součinu. Uvažme přímku procházející počátkem, popíšeme ji normalizovaným vektorem $\mathbf{u}$. Dále mějme bod $\mathbf{x}$. Hledáme bod $\mathbf{x}_U$ na přímce $\mathbf{u}$, který je nejblíže bodu $\mathbf{x}$, takzvanou *projekci* bodu $\mathbf{x}$ na přímku $\mathbf{u}$.

Z geometrie víme, že projekci lze najít tak, že z bodu $\mathbf{x}$ vedeme kolmici na přímku $\mathbf{u}$. Použijeme podobný princip. Přidáme normalizovaný vektor $\mathbf{n}$ kolmý na $\mathbf{u}$ a rozložíme $\mathbf{x}$ jako lineární kombinaci těchto dvou vektorů:

$$\mathbf{x} = \langle \mathbf{x}, \mathbf{u} \rangle \, \mathbf{u} + \langle \mathbf{x}, \mathbf{n} \rangle \, \mathbf{n}$$



Hledaná projekce je $\mathbf{x}_U = \langle \mathbf{x}, \mathbf{u} \rangle \, \mathbf{u}$. Také umíme spočítat vzdálenost bodu $\mathbf{x}$ od přímky $\mathbf{u}$ jako $\|\mathbf{x} - \mathbf{x}_U\|$.

Skalární součin $\langle \mathbf{x}, \mathbf{u} \rangle$ tedy vyjadřuje velikost projekce $\mathbf{x}_U$. Takto to platí jen pro normalizovaný vektor $\mathbf{u}$, v plné obecnosti je velikost projekce $\langle \mathbf{x}, \mathbf{u} \rangle \,/\, \|\mathbf{u}\|$.

S trochou goniometrie zvládneme konečně spočítat úhel α mezi vektory $\mathbf{x}$ a $\mathbf{u}$:

$$\cos \alpha = \frac{\|\mathbf{x}_U\|}{\|\mathbf{x}\|} = \frac{\langle \mathbf{x}, \mathbf{u} \rangle \,/\, \|\mathbf{u}\|}{\|\mathbf{x}\|} = \frac{\langle \mathbf{x}, \mathbf{u} \rangle}{\|\mathbf{x}\| \cdot \|\mathbf{u}\|}$$

Projekci jsme odvozovali ve dvou dimenzích, tentýž vzorec ale platí i v prostorech s více dimenzemi. Vektory $\mathbf{x}$ a $\mathbf{u}$ totiž vždy leží v jedné rovině, takže všechny naše úvahy fungují stejně.

Podobně jako na přímku můžeme projektovat na rovinu či jiný prostor. Stačí tento prostor popsat ortonormální bází, spočítat projekci do směru každého bazického vektoru a tyto projekce sečíst.

Úkol 1 – Mnohoúhelník [4b]:

Dostanete zadané vrcholy $\mathbf{p}_1, \ldots, \mathbf{p}_n$ konvexního mnohoúhelníku v rovině a bod $\mathbf{x}$.

Popište, jak rozhodnout, zda leží bod $\mathbf{x}$ uvnitř mnohoúhelníku. Určete časovou složitost svého řešení.

Ortogonalizace

Již známe výhody ortonormální báze. Ovšem jak ji získat? K tomu nám poslouží *Gramova-Schmidtova ortogonalizace* (ve skutečnosti jde o ortonormalizaci, ovšem tento název je zažitý).

Mějme lineárně nezávislé vektory $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Chceme získat ortonormální systém vektorů $\mathbf{y}_1, \ldots, \mathbf{y}_n$ se stejným lineárním obalem.

Vektor $\mathbf{y}_1$ získáme normalizací vektoru $\mathbf{x}_1$. Vektor $\mathbf{y}_2$ musí být kolmý na $\mathbf{y}_1$. Od vektoru $\mathbf{x}_2$ tedy odečteme jeho projekci na $\mathbf{y}_1$, výsledek normalizujeme a uložíme jako $\mathbf{y}_2$. Obdobně postupujeme dále, vždy odečteme projekci na všechny předchozí vektory.

Zapsáno matematicky, $\mathbf{y}_k$ spočítáme následovně:

$$\mathbf{n}_k = \mathbf{x}_k - \sum_{i=1}^{k-1} \langle \mathbf{x}_k, \mathbf{y}_i \rangle \, \mathbf{y}_i$$

$$\mathbf{y}_k = \frac{\mathbf{n}_k}{\|\mathbf{n}_k\|}$$

Úkol 2 – Vzdálenost od roviny [6b]:

Určete vzdálenost bodu $\mathbf{x} = (6, 3, 3)^\top$ od roviny ρ, jejíž body lze vyjádřit ve tvaru $\mathbf{c} + a_1\mathbf{u}_1 + a_2\mathbf{u}_2$, kde:

$$\begin{aligned}
\mathbf{c} &= (1, 2, 3)^\top \\
\mathbf{u}_1 &= (2, -3, 6)^\top \\
\mathbf{u}_2 &= (4, 4, 2)^\top
\end{aligned}$$

Pokud použijete vzorec neuvedený v tomto textu, musíte jej odvodit.

Ortogonální doplněk

Na začátku jsme nahlédli, že standardní skalární součin lze vnímat také jako maticové násobení. Výraz $\mathbf{Ax}$ počítá najednou několik skalárních součinů, jedna složka výsledku odpovídá skalárnímu součinu jednoho řádku matice $\mathbf{A}$ s vektorem $\mathbf{x}$. Pokud vyjdou všechny skalární součiny nulové, tedy pokud $\mathbf{Ax} = \mathbf{0}$, je vektor $\mathbf{x}$ kolmý na všechny řádky matice $\mathbf{A}$. Zároveň si vzpomeňte, že jsme v posledním dílu

definovali jádro matice $\mathbf{A}$ přesně jako množinu vektorů $\mathbf{x}$, které tuto rovnici splňují.

Dostáváme tak nový pohled na jádro jako na množinu vektorů kolmých na řádky matice. Nejen to, vektory jádra jsou kolmé dokonce na všechny vektory z řádkového prostoru, protože lineární kombinace vektorů kolmých na $\mathbf{x}$ bude opět kolmá na $\mathbf{x}$.

Totéž platí v opačném směru, každý vektor řádkového prostoru je kolmý na všechny vektory jádra. Báze obou prostorů se navzájem doplňují, dohromady tvoří bázi celého prostoru $\mathbb{R}^n$, kde n je dimenze $\mathbf{x}$. Říkáme, že řádkový prostor je *ortogonální doplněk* jádra a naopak.

Tímto dostáváme návod, jak najít množinu vektorů kolmých k nějakému podprostoru U. Stačí dát bázi U do řádků matice a vyřešením soustavy rovnic najít její jádro.

Metoda nejmenších čtverců

Kromě výpočtu vzdálenosti geometrických objektů má projekce ještě jedno využití, totiž hledání přibližných řešení soustav rovnic. Uvažme soustavu $\mathbf{Ax} = \mathbf{b}$, která nemá řešení. Můžeme však najít takové $\mathbf{b}'$, pro které řešení existuje a které je co nejblíže původnímu $\mathbf{b}$.

Z minulého dílu víme, že řešení existuje, pokud pravá strana leží ve sloupcovém prostoru, což nám dává podmínku $\mathbf{b}' \in \mathcal{S}(\mathbf{A})$. Hledáme projekci, takže $\mathbf{b}' - \mathbf{b}$ musí být kolmé na $\mathcal{S}(\mathbf{A})$, jinými slovy musí ležet v jeho ortogonálním doplňku. Víme, že jádro je doplňkem řádkového prostoru, zde však máme sloupcový prostor. Není problém, transpozicí zaměníme sloupce za řádky. Hledaným doplňkem je proto $\ker(\mathbf{A}^\top)$. Dostáváme tedy rovnici $\mathbf{b}' - \mathbf{b} \in \ker(\mathbf{A}^\top)$, kterou stačí rozepsat:

$$\begin{aligned} \mathbf{A}^\top(\mathbf{b}' - \mathbf{b}) &= \mathbf{0} \\ \mathbf{A}^\top(\mathbf{Ax} - \mathbf{b}) &= \mathbf{0} \\ \mathbf{A}^\top\mathbf{Ax} &= \mathbf{A}^\top\mathbf{b} \end{aligned}$$

Matici $\mathbf{A}^\top\mathbf{A}$ můžeme vyčíslit, stejně tak vektor $\mathbf{A}^\top\mathbf{b}$. Dostaneme novou, takzvanou *normální* soustavu rovnic, která již řešení mít bude.

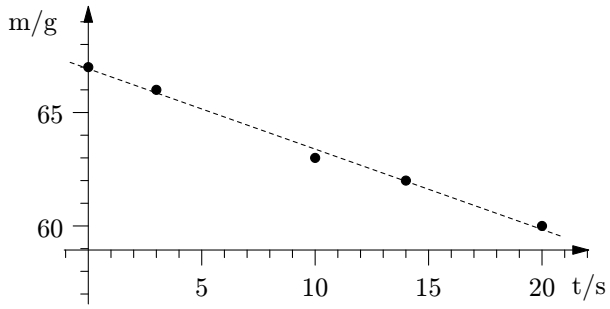
Tento přístup nese název *metoda nejmenších čtverců*, neboť minimalizujeme vzdálenost $\|\mathbf{b}' - \mathbf{b}\|$. Minimum se nabude pro stejné $\mathbf{b}'$ jako minimum druhé mocniny tohoto výrazu, a to není nic jiného než součet druhých mocnin rozdílů složek $\mathbf{b}'$ a $\mathbf{b}$.

Úkol 3 – Fyzikální měření [5b]:

David měřil, jak rychle se vypařuje tekutý dusík z láhve, ve které jej skladuje. Láhev měl položenou na váze a zaznamenával hmotnost v průběhu času.

m/g	67	66	63	62	60
t/s	0	3	10	14	20

Nyní by rád zjistil, za jak dlouho se všechen dusík vypaří. Předpokládejte, že závislost hmotnosti na čase lze odhadnout lineární funkcí, viz čárkovaná přímka na obrázku. Spočítejte rychlost vypařování metodou nejmenších čtverců a odpovězte mu.



Mimochodem, metoda minimalizuje druhé mocniny vertikálních vzdáleností mezi přímkou a body.

Vizualizace

Tento díl při důkazech využíval převážně algebraické operace, které se pro psaný text hodí více. Mnohé však lze nahlédnout i vizuálně. Pokud neznáte, velmi doporučujeme sérii videí od 3Blue1Brown,[3] zejména pak video na skalární součin, které na něj nabízí úplně jiný pohled.

David Klement

Recepty z programátorské kuchařky: Geometrie

Geometrické algoritmy

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

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

Geometrické základy

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

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

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

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

Často potřebujeme popsat vzájemnou polohu dvou bodů.

[3] `https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab`

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

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

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

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

$$x = a_x + tu_x$$

$$y = a_y + tu_y$$

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

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

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

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

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

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

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

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

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

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

Zjištění polohy bodu vůči přímce

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

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

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

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

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

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

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

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

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

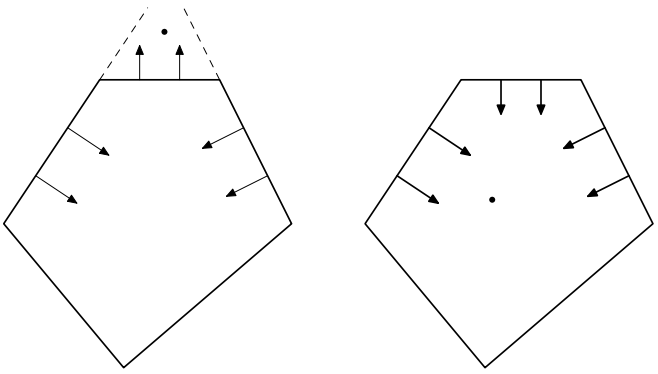
Pokud vyjde d kladné, je bod napravo od přímky, pokud vyjde d záporné, je bod nalevo od přímky, a konečně, pokud vyjde $d = 0$, tak bod leží na přímce.

Bod a konvexní mnohoúhelník

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

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

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



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

Bod a nekonvexní mnohoúhelník

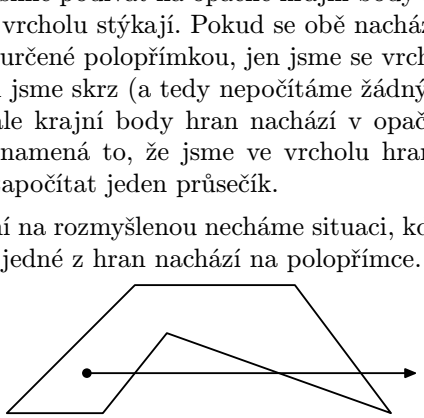
Pro nekonvexní útvary je již postup o něco těžší, protože jak si můžeme všimnout, postup s kontrolováním polohy bodu vůči všem hranám fungovat nebude.

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

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

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

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

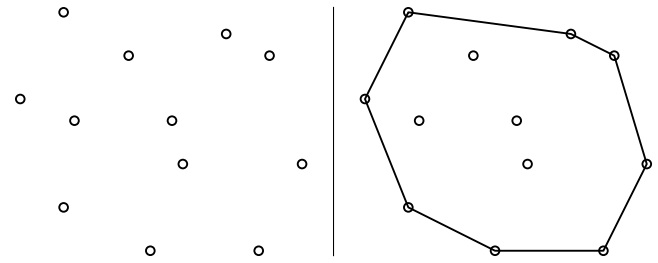


Opět musíme zkontrolovat polopřímku vůči všem hranám, takže časová složitost je znovu $\mathcal{O}(N)$ (i když s o něco vyšší konstantou, protože spočítání průsečíku je více početních operací než jeden test polorovinou).

Konvexní obal a zametání roviny

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

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



Vlevo neobalené body, vpravo obalené.

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

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

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

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

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

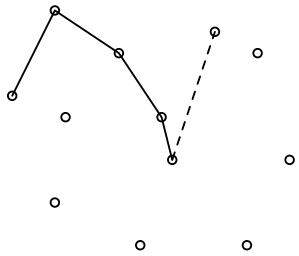
V průběhu výpočtu si budeme konstruovat horní a dolní *obálku.* Obě obálky budou určitě začínat v nejlevějším a končit v nejpravějším bodě (jednoduchým pozorováním lze nahlédnout, že tyto body do obalu určitě patří). A jak už název napovídá, horní obálka půjde vrchem a bude se zatáčet stále doprava,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

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

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

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

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



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

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

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

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

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

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

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

Může se zdát, že hodně, protože při vyhazování bodů z obálky můžeme postupně vyhodit skoro všechny body. Označme si velikost zadané množiny (počet bodů na vstupu programu) N. Musíme si uvědomit, že každý bod do obálky přidáme pouze jednou a vyhodíme ho také maximálně jednou, tedy časová složitost je lineární k velikosti množiny, čili $\mathcal{O}(N)$, v případě, že již máme setříděný vstup. Pokud ne, musíme ještě přičíst čas potřebný k setřídění bodů, tedy $\mathcal{O}(N \log N)$ při použití nějakého rychlého třídicího algoritmu.[4]

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

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

Hledání průsečíků úseček

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

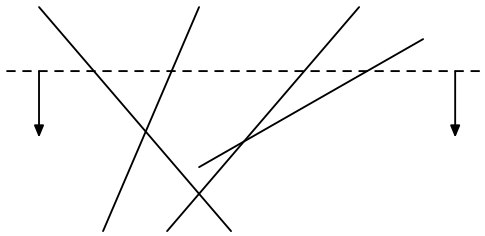
Hledáme samozřejmě co nejrychlejší algoritmus vzhledem k N a počtu průsečíků P.

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

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

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

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



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

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

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

spočítat, jestli se někde protínají, a případně průsečíky přidat do seznamu událostí.

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

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

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

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

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

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

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

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

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

Závěr

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

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

A jak jsme si uvedli na začátku, mnohé z uvedených postupů lze zobecnit z roviny i do prostoru, ale o tom někdy jindy. Pokud máte zájem o další informace o geometrických algoritmech, tak vás můžeme odkázat na *Průvodce labyrintem algoritmů* stáhnutelný ze stránek Martina Mareše.[5]

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

Jirka Setnička

[5] `http://pruvodce.ucw.cz/`

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.
Realizace projektu byla podpořena Ministerstvem školství, mládeže a tělovýchovy.

Webové stránky: *https://ksp.mff.cuni.cz/*
E-mail: *ksp@mff.cuni.cz*
Organizátoři a kontakty: *https://ksp.mff.cuni.cz/kontakty/*