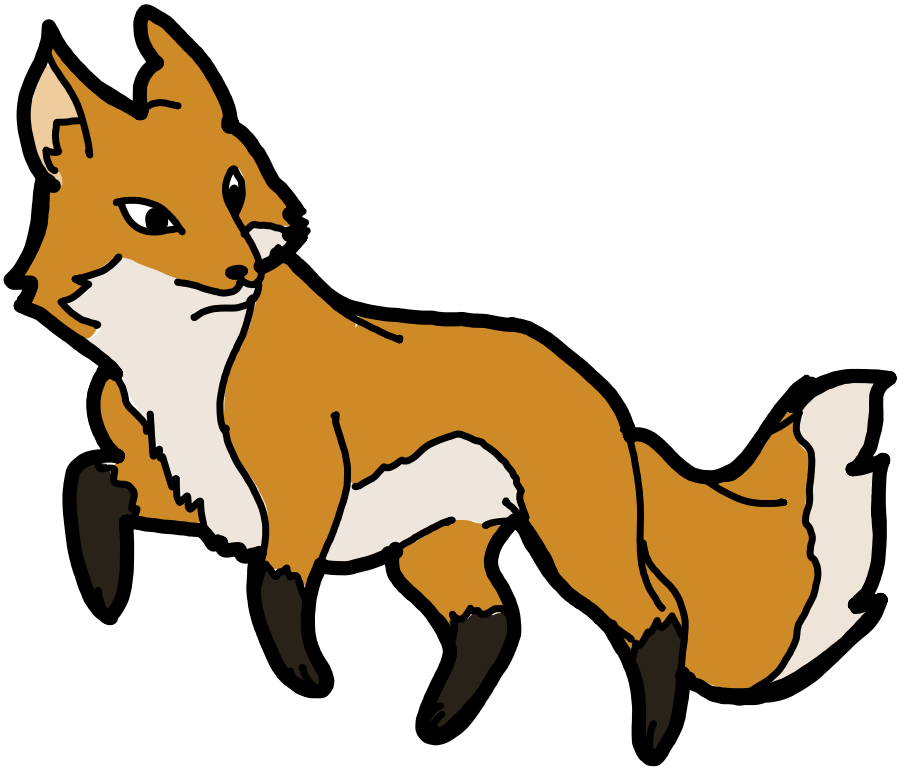
STUDENTSKÝ ČASOPIS A KORESPONDENČNÍ SEMINÁŘ

Ročník XXXII
Číslo 5

M&M

MATEMATIKA FYZIKA INFORMATIKA



Uvnitř najdete několik témat a s nimi souvisejících úloh. Zamyslete se nad nimi a pošlete nám svá řešení. My vám je opravíme a ta nejzajímavější z nich otiskneme. Nejlepší řešitele zveme na podzim a na jaře na soustředění.

Milí řešitelé,

v tomto, již pátém, čísle jsme si pro vás opět připravili několik zajímavých témat. Než se ale pustíte do jejich řešení, určitě budete chtít vědět, co vlastně budete řešit.

I tentokrát si budeme povídat o grupách, konkrétně o jejich přínosu světu počítačů. Řekneme si něco o volných grupách a konečných automatech.

Po grupách nás čeká další matematické tématko, tentokrát z kombinatoriky. Tam si trochu formálněji povíme například něco o situacích, kde nám záleží na pořadí prvků, ale vlastně ne tak úplně.

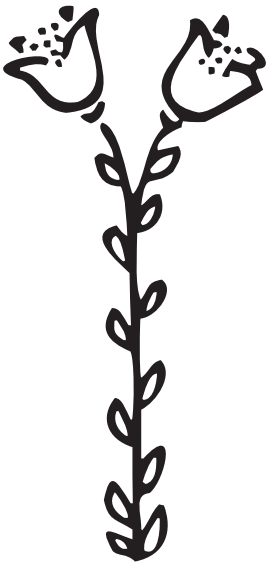
Určitě ale nezapomínáme na četné programátory mezi námi. Ti si budou moct opět přečíst něco o LISPu a jeho komunikaci s okolním světem a podívat se na řešení úloh z minulého čísla.

Bohužel se zatím rozloučíme s tématy šifrování a vajíčko parašutista, ale nezoufejte! Vášniví luštitelé si budou moct prohlédnout vzorová řešení minulých úloh.

A teď už konec zdržování, jdeme na to!

Vaši orgové M&M

Obsah



Zadání a řešení témat

1. deadline: 21. dubna 2026 | 2. deadline: 26. května 2026

Téma 1 – Grupy aneb Kterak matika k souměrnosti přišla

Díl 4: Konečné automaty

Ano, je tomu tak. Po pouti stezkami lemovanými stavebními kameny časoprostoru se navracíme do snad skromnějšího světa jedniček a nul. V tomto díle tématka totiž prohlédneme očima teoretického informatika, bychom zjistili, kde tato mohou spatřiti grup. Ježto jsou grupy strukturou vskutku universální, jistě souvisejí s nemálem informatických oblastí; my jsme zvolili jednu: *konečné automaty*.

Před jejich adventem v našem textu však musíme – neurazte se, vážení čtenáři – rozšířit vašich grupových znalostí. Přesněji, potřebujeme definovat a vysvětlit pojmy *volná grupa* a *relace*.

Volné grupy, generátory a relace

Ve zcela prvním díle tématka jsme představili *generátory*. Připomeneme se. O množině prvků $X = x_1, \ldots, x_n$ řekneme, že *generuje* grupu $(G, *, ^{-}, e)$, když každý prvek $g \in G$ můžeme napsat jako součin mocnin prvků z X, tedy jako

$$g = x_{i_1}^{k_1} * x_{i_2}^{k_2} * \cdots * x_{i_m}^{k_m} \qquad (\heartsuit)$$

pro (ne nutně různé) indexy $i_1, \ldots, i_m \in \{1, \ldots, n\}$ a celá čísla $k_1, \ldots, k_m \in \mathbb{Z}$. Rovněž připomínáme, že výraz x^k je zkratkou za x vynásobené samo se sebou k-krát, je-li $k > 0$, nebo $\overline{x}$ vynásobené samo se sebou k-krát, je-li $k < 0$. Pro $k = 0$ dodefinujeme, jak bývá zvykem, $x^0 = e$. Zde je důležité vnímat, že $*$ **není nezbytně komutativní**, takže součin $x_1 * x_2$ může být různý od $x_2 * x_1$. Dvěma zcela náhodnými příklady takto vyrobených prvků G buďtež $x_3^5 * x_2^{-2} * x_4^3$ a $x_1^{-1} * x_2^{-3} * x_1^2$.

Množina X generuje leckteré grupy, ale mezi nimi je jedna význačná (tzv. *volná grupa* nad X), v níž je téměř každý výraz tvaru ($\heartsuit$) *různým* prvkem. Jinak řečeno, téměř každé uspořádání prvků $x_1, \ldots, x_n$ (a jejich inversů) je unikátní prvek volné grupy nad X. Skládání prvků množiny X do prvků volné grupy nad X je intuitivně podobné skládání *písmen* do *slov*. Z toho důvodu se pro množinu X ujalo přízvisko *abeceda*, a pro volnou grupu nad X přízvisko *slov* v abecedě X. Vysvětlíme se pořádně.

Množinu $X = \{x_1, \ldots, x_n\}$ nazveme *abecedou* a o prvcích x_i se budeme vyjadřovat jako o *proměnných* či *písmenech*. Každému písmeni x_i přidáme jeho (formální) invers: písmeno $\overline{x_i}$. Jakoukoli posloupnost písmen $x_1, \ldots, x_n$ a jejich inversů nazveme *slovem* (v abecedě X). Připouštíme i tzv. *prázdné slovo* (prázdnou posloupnost písmen), které označíme jako e. Z daného slova vyrobíme tzv. *redukované slovo* tak, že každý pár písmene a jeho inversu (čili pár $x_i\overline{x_i}$ nebo $\overline{x_i}x_i$)

ve slově odstraníme. Například

$$x_1\overline{x_1}x_3x_3\overline{x_2}x_2\overline{x_4} \xrightarrow{\text{redukce}} x_3x_3\overline{x_4}$$

Volnou grupu nad X (značíme třeba $F(X)$ z angl. *free*) definujeme jako grupu redukovaných slov v abecedě X, kde

- binární operací je operace *skládání* slov $(*)$, která funguje zkrátka tak, že dvě redukovaná slova nalepí za sebe a výsledné slovo opět redukuje; například

  $$x_2\overline{x_3x_1} * x_1x_3x_4 = x_2x_4$$

  (proběhlo odstranění $\overline{x_1}x_1$ a $\overline{x_3}x_3$);
- inversem $(^-)$ k redukovanému slovu je (opět redukované) otočené slovo s každým písmenem nahrazeným jeho inversem, čili

  $$\overline{x_1\overline{x_2}x_1x_3} = \overline{x_3x_1}x_2\overline{x_1};$$

- neutrálním prvkem (či identitou) je prázdné slovo e, protože při složení slova s jeho inversem redukcí vlastně „odebereme všechna písmena“.

Při psaní slov ušetříme trochu práce i místa a posloupnosti stejných písmen (a jejich inversů) uvnitř redukovaných slov budeme zkracovat mocninným zápisem. Čili, slovo

$$\overline{x_2x_2}x_1x_1x_1x_4x_3x_3$$

zapíšeme jako $x_2^{-2}x_1^3x_4x_3^2$.

Při definici volné grupy záleží pouze na tom, kolik prvků má její abeceda. Volné grupy nad dvěma různými abecedami $X = \{x_1, \ldots, x_n\}$ a $Y = \{y_1, \ldots, y_n\}$ jsou v podstatě stejné: slova jedné grupy přetvoříme na slova druhé grupy jenom přepsáním písmen.

Volnou grupou nad abecedou $X = \{x\}$ o jednom písmeně je zkrátka grupa všech mocnin proměnné x s obvyklým násobením. Totiž, jediná slova, která lze vyrobit z jednoho písmene jsou slova x^k pro $k \in \mathbb{Z}$ a skládání funguje takto:

$$x^k * x^l = x^{k+l}.$$

Ti nebývale pozorní z čtenářů jste si mohli povšimnout, že tuto grupu už jsme viděli – je tou přesně grupa $(\mathbb{Z}, +, -, 0)$ celých čísel se sčítáním. Totiž, přepíšeme-li slovo x^k na celé číslo k, operaci $*$ na $+$, invers $^-$ na $-$ a prázdné slovo e na 0, pak je struktura obou grup dokonale stejná. Jednodušeji řečeno, skládání slov v $F(\{x\})$ je „to samé“, jako sčítání čísel v $\mathbb{Z}$. Vskutku, porovnejte například

$$\begin{aligned} x^k * x^l = x^m \quad &\text{s} \quad k + l = m, \\ x^k * e = x^k \quad &\text{s} \quad k + 0 = k, \end{aligned}$$

nebo

$$x^k * x^{-k} = e \quad \text{s} \quad k + (-k) = 0.$$

Skutečnost, že lze grupu $(\mathbb{Z}, +, -, 0)$ vyjádřit jako volnou, není náhoda. Jedním ze zásadních výsledků teorie grup je fakt, že v jistém smyslu, který záhy odkryjeme, jsou **všechny** grupy volné. Tento „jistý“ smysl spočívá v pojmu *relace*.

Totiž, ačkolivěk volné grupy samy o sobě nesou až překvapivě zajímavou strukturu, mají jeden zásadní kaz: jsou vždy nekonečné (s výjimkou volné grupy nad prázdnou abecedou). Abychom uměli popsat i konečné grupy jako volné, musíme nějak omezit množinu redukovaných slov. Velmi ohebným způsobem, jak takové omezení vyrobit, je považovat některá slova za stejná. Vraťme se k příkladu $F(\{x\})$. Rozhodněme se, že x^n bude prázdné slovo (pro nějaké $n \in \mathbb{N}$), tj. zaveďme „omezení“ $x^n = e$. Z této rovnosti ihned plyne též $x^{-n} = e$, protože

$$e = x^n * x^{-n} = e * x^{-n} = x^{-n}.$$

Zároveň taky $x^m = x^{m-n}$ pro každé $m \in \mathbb{Z}$. Doložíme podobně jednoduchým výpočtem:

$$x^{m-n} = x^m * x^{-n} = x^m * e = x^m.$$

To ale znamená, že naše grupa je konečná! Přeci, žádné slovo nemůže být „delší“ než x^n nebo x^{-n}, takže zde existují pouze slova $e, x, x^2, \ldots, x^{n-1}$ a jejich inversy. A jejich inversy... Opravdu? Co ale znamená rovnost

$$x^k * x^{n-k} = x^n = e$$

pro nějaké $0 \leq k < n$? Samozřejmě to, že x^{n-k} je inversem k prvku x^k, neboli $x^{-k} = x^{n-k}$. V naší grupě, kterou můžeme zkráceně značit třeba $F(\{x\} \mid x^n = e)$ tedy přežívá opravdu jen n slov: $e, x, x^2, \ldots, x^{n-1}$, protože mezi nimi najdeme i jejich inversy. Kteroupak jsme to tady dostali jen grupu? Prosíme, odpovězte nám.

Úloha 5.1 [2b]: *V prvním díle tématka jsme viděli grupu „totožnou“ s grupou $F(\{x\} \mid x^n = e)$ v tom smyslu, že má stejně prvků a jednu grupu z druhé získáme pouhým přeznačením prvků x^k, operace $*$, inversu $^-$ a prázdného slova e. Která to je a hlavně proč?*

Onomu omezení „$x^n = e$“, jež jsme uvalili na volnou grupu $F(\{x\})$, se přezdívá *relace*; nejspíše pro to, že určuje „vztah“ rovnosti mezi dříve různými prvky grupy.

Volnou grupu nad dvěma písmeny jsme ještě neviděli. Když volná grupa nad jedním písmenem byla v podstatě grupa $\mathbb{Z}$ celých čísel, mohlo by nám přijít na mysl, že volná grupa nad písmeny dvěma budou zkrátka dvojice celých čísel, tedy grupa $\mathbb{Z} \times \mathbb{Z}$, kde sčítání probíhá v každé složce zvlášť. Není tomu tak. Potíž dlí v nekomutativitě skládání slov. Totiž, ne všechna slova v grupě $F(\{x, y\})$ můžeme napsat jako $x^k y^l$ pro nějaká $k, l \in \mathbb{Z}$. Zabydlilo se zde třeba i slovo $x^3 y^2 x^{-2}$, které do tohoto tvaru převést nelze. Jak známo, sčítání je *komutativní* operace: může tedy být ekvivalentní operaci $*$ skládání slov jedině v případě, kdy i tato je komutativní. To se naopak stane jen tehdy, když máme v abecedě nejvýše jedno písmeno.

Snad však překvapí zjištění, že o volné grupě $F(\{x, y\})$ *s jistými relacemi* jsme v tématku hovořili vcelku obšírně. Nebudeme vás napínat: je jí grupa D_{2n} symetrií pravidelného n-úhelníku. Jejími generátory přeci byly jedna (vhodná) rotace a jedna reflexe. Označme si je po řadě písmeny x a y. Rotace generující D_{2n} byla speciální tím, že až její n-té opakování byla rotace o 360°, čili neutrální prvek grupy D_{2n}. Do prostředí grupy $F(\{x, y\})$ se taková vlastnost přenáší v podobě relace $x^n = e$. Dále, reflexe byly význačné tím, že byly samy sobě inversní. Tuto skutečnost můžeme vyjádřit relací $y = \overline{y}$ nebo – snad elegantněji – $y^2 = e$. Vlastně jsme tedy nahradili generující rotaci písmenem x, nějakou zvolenou reflexi písmenem y, operaci skládání funkcí ($\circ$) operací skládání slov ($*$) a identickou funkci id prázdným slovem e. Stačí to? Je grupa $(D_{2n}, \circ, {}^{-1}, \mathrm{id})$ ta samá jako $F(\{x, y\} \mid x^n = e, y^2 = e)$? Prozradíme, že nikoliv. Je potřeba ještě jedné další relace, jež reflektuje vzájemné působení reflexí na rotace a rotací na reflexe. Totiž, i s relacemi $x^n = e$ a $y^2 = e$ jsou písmena x a y zcela nezávislá. Potřebujeme nějak vyjádřit fakt, že složení rotace s reflexí je opět reflexe. Věříme, že správnou relaci vymyslíte.

Úloha 5.2 [3b + 3b]:

(1) Najděte zmíněnou relaci vyjadřující vztah rotací a reflexí v grupě D_{2n}. Opravdu stačí jen jedna.

(2) Dokažte, že grupa $F(\{x, y\})$ s relacemi $x^n = e$, $y^2 = e$ a relací z části (1) je opravdu totožná grupě $(D_{2n}, \circ, {}^{-1}, \mathrm{id})$. To znamená, že má stejně prvků, písmena x a y lze ztotožnit s generující rotací a reflexí, operace $\circ$ se „chová jako“ $*$, invers ${}^{-1}$ jako ${}^{-}$ a id *jako* e.

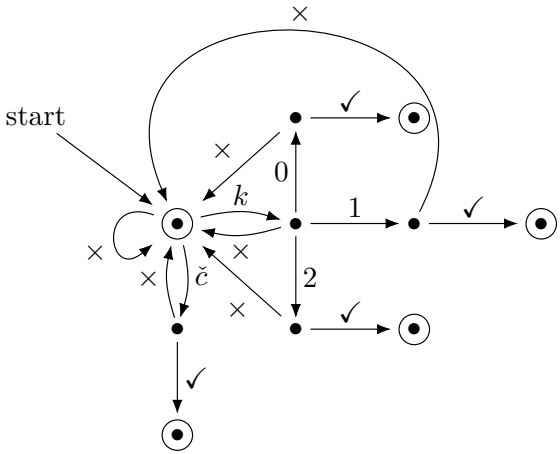
Konečné automaty

Volné grupy s relacemi mají mnoho souvislostí s oblastmi (nejen) informatiky či lingvistiky. Jedním takovým poutem jsou *konečné automaty*. My definujeme konečný automat nad abecedou X jako stroj, jenž se může nacházet vždy v jednom z konečného množství stavů a mezi těmito stavy přechází na základě vstupů v podobě písmen z abecedy X. Některé z těchto stavů jsou tzv. *přípustné* a jeden stav je vždy označen za *počáteční*.

Jako příklad uvažme prodejní automat na kávu a čokoládu. Uživatel volí nejprve nápoj a, zvolil-li kávu, volí dále množství cukru přes jednu z číselných předvoleb 0, 1 nebo 2. Nakonec musí volbu potvrdit. V kterékoli fázi výběru nápoje smí uživatel celý prodej stornovat. Abecedou bude pročež množina $\{k, č, 0, 1, 2, \times, ✓\}$, kde k znamená volbu kávy, $č$ volbu čokolády a $\times$, resp. ✓, storno, resp. potvrdit. Ne všechna slova v této abecedě přivedou automat do přípustného stavu: například po volbě čokolády nelze volit množství cukru, takže přečtení slova $č2$ vyústí v nepřípustný stav. Podobně, potvrdit nákup po volbě kávy bez udání množství cukru též není možné, takže nepřípustně dopadne i vstup $k✓$. Naopak, po přečtení $k1\times$ skončí automat ve stavu přípustném.

Zdůrazňujeme též, že pojímáme svůj prodejní automat takovým způsobem, aby nemohl skončit „uprostřed prodeje“, čili jeho jediné přípustné stavy jsou počáteční (před zahájením prodeje či pro provedení storna) a kterýkoli z koncových (představující prodej navoleného nápoje). To znamená, že ani slovo $k1$ či samotné $\check{c}$ neuvede automat do přípustného stavu. Obě tato slova musejí být rozšířena buď o písmeno ✓ – jež povede k prodeji nápoje – anebo o $\times$, zvěstující storno a návrat do počátečního stavu.

Jako mnoho struktur v matematice, i konečné automaty je velmi užitečné umět kreslit. Ukážeme jeden způsob. Jednotlivé stavy budeme representovat jako puntíky a přechody mezi stavy jako šipky (nebo smyčky, když automat na základě vstupu nemění stav).[1] Každé šipce tudíž bude náležet jedno písmeno abecedy X, které tento přechod mezi stavy způsobí. Přípustné stavy zakroužkujeme. Náš prodejní automat hledejte na obrázku 1. Rozhodli jsme se úspěšný prodej jednoho ze čtyř (tři „různé“ kávy a jedna čokoláda) nápojů representovat čtyřmi různými stavy. Bylo pochopitelně možné, třebaže snad méně přirozené, sjednotit je v jeden stav zvaný „prodej“.



Obrázek 1: Konečný automat na prodej kávy nebo čokolády.

Z obrázku 1 je zřejmé, že nekonečná slova končí v přípustném stavu jedině v případě, že obsahují $\times$; kupříkladu $\times^n$ je jedno takové slovo pro každé $n \in \mathbb{N}$. Dále, náš automat má několik „konečných“ stavů, čili stavů, z nichž již nemůže přejít do jiných. Konečné automaty spjaté s volnými grupami žádné takové mít nebudou (nelze „zakázat“ skládání slov), přesto jsme shledali užitečným podotknout, že existenci konečných stavů nic nebrání.

[1] Formálně bychom řekli, že konečný automat representujeme *toulcem*: `https://en.wikipedia.org/wiki/Quiver_(mathematics)`.

Souvislost konečných automatů s volnými grupami je poněkud přímočará: každou **konečnou** volnou grupu s relacemi[2] lze považovat za konečný automat. Každý jeden z jeho stavů odpovídá jednomu slovu a přechod ze stavu do stavu odpovídá připojení písmene na konec slova. Ještě si rozmyslíme, jak do jazyka konečných automatů přetlumočit relace. Mějme nějaká dvě slova s a t v abecedě X. Relaci $s = t$ můžeme triviálně přepsat do relace $s\overline{t} = e$ (složíme obě strany s písmenem $\overline{t}$ zprava). To znamená, že obecně každá volná grupa s relacemi nad X je tvaru

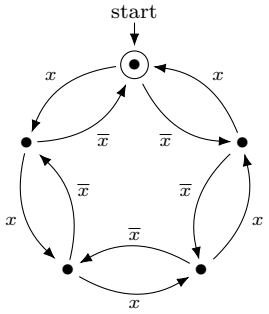
$$F(X \mid s_1 = e, s_2 = e, \ldots, s_n = e)$$

pro nějaká slova $s_1, \ldots, s_n \in F(X)$. Abychom takovou grupu representovali konečným automatem, sestavíme ho tak, že jediným přípustným (a zároveň počátečním) stavem bude prázdné slovo e a posloupnost šipek odpovídající slovu $s \in F(X)$ povede do přípustného stavu právě tehdy, když $s = e$. Ukážeme několik příkladů.

Grupu $F(\{x\} \mid x^5 = e)$ můžeme nakreslit jako pětiúhelník s jedním vyznačeným vrcholem pro prázdné slovo e a dvěma protijdoucími šipkami mezi každými dvěma vrcholy: jedné pro x a jedné pro $\overline{x}$. Vizte obrázek 2. Vskutku jedinými dvěma *redukovanými* slovy, která automat přivedou do přípustného stavu, jsou x^5 a x^{-5}. Ta jsou ale stejná slova, protože

$$e = x^5 * x^{-5} = e * x^{-5},$$

tudíž $x^{-5} = e$. Výpočet výše obecně ukazuje, že platí-li $s^n = e$ pro nějaké slovo s a $n \in \mathbb{N}$, pak též $s^{-n} = e$.



Obrázek 2: Konečný automat representující grupu $F(\{x\} \mid x^5 = e)$.

[2]Nekonečnost grupy by nutně znamenala existenci nekonečného množství stavů.

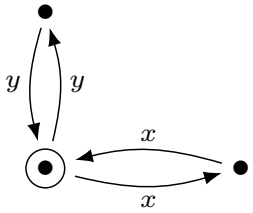
O něco složitější grupou je tzv. Kleinova čtyřgrupa[3], jež jest grupou symetrií pravidelného obdélníka či kosočtverce, které však nejsou čtverci. Lze ji vyjádřit jako volnou grupu

$$F(\{x,y\} \mid x^2 = e, y^2 = e, (xy)^2 = e).$$

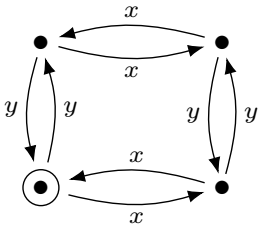
Nejprve zjistíme, kolik má tato grupa vlastně prvků (název čtyřgrupa již něco napovídá). Jistě zde leží prvky e, x, y a xy. Zjistíme, že žádné další nemá zkrátka tak, že je spolu všechny složíme. Skládání ostatních slov s e můžeme přeskočit a z definice Kleinovy grupy je každé slovo složené samo se sebou rovno prázdnému slovu. Zbývá si všimnout, že $x{*}xy = x^2y = y, xy{*}y = \overline{xy^2} = x$ a $xy = yx$. Poslední rovnost plyne z faktů, že $\overline{x} = x$ (bo $x^2 = e$), $\overline{y} = y$, $\overline{(xy)} = xy$ a výpočtu

$$xy = \overline{(xy)} = \overline{yx} = yx.$$

Kreslení příslušného konečného automatu provedeme postupně. Je jasné, že se z počátečního stavu lze hýbat buď po šipce s písmenem x, nebo po šipce s písmenem y. Zpátky z těchto stavů se do počátečního dostaneme po týchž písmenech, protože $x^2 = e$ a $y^2 = e$. Můžeme pročež začít kreslit automat třeba takto:



Ze stavu x se můžeme po šipce y dostat do xy a ze stavu y po šipce x do stavu yx. Ovšem, jak jsme výše spočetli, tyto dva stavy jsou stejné, takže můžeme automat směle dokončit. Jaká to ironie, že jej lze nakreslit jako čtverec!



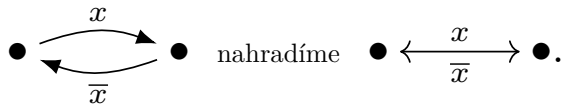
Obrázek 3: Konečný automat representující Kleinovu čtyřgrupu.

[3]Příslušný článek na Wikipedii je zde: `https://en.wikipedia.org/wiki/Klein_four-group`.

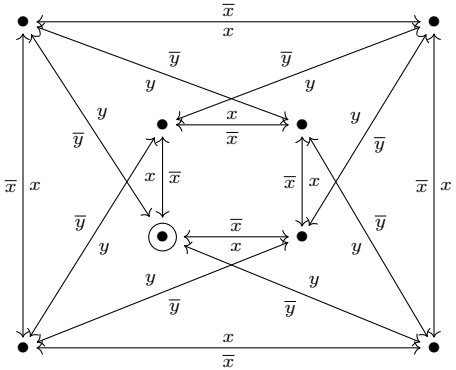
Posledním příkladem bude zprvu záhadná grupa

$$\mathbf{Q}_8 = F(\{x, y\} \mid x^4 = e, x^2 = y^2, \overline{y}xy = \overline{x}),$$

mající 8 prvků. Mohli jste o ní již slyšet jako o grupě *kvaternionů*[4], jakýchsi „zobecněných“ komplexních čísel. Nakreslit ji jako konečný automat přehledně není snadné a nebudeme zde šířeji popisovat postup. Soudíme, že bude v tomto případě vhodné protijdoucí šipky nahradit jedinou obousměrnou šipkou. Tu budeme chápat jako v jednom směru skládání s přisouzeným písmenem a v druhém jako skládaní s jeho inversem. Ještě výmluvněji, každý obrázek



Konečný automat popisující grupu $\mathbf{Q}_8$ vizte na obrázku 4.



Obrázek 4: Konečný automat pro grupu kvaternionů $\mathbf{Q}_8$.

Úloha 5.3 [3b]: *Nakreslete konečný automat pro grupu D_8 symetrií čtverce. Znalost relace z úlohy 5.2 (1) vám může přijít vhod, ale není pro nakreslení automatu nezbytná. Inspirujte se obrázkem 4. Grupy $\mathbf{Q}_8$ a D_8 stejné sice nejsou, ale jisté strukturální podobnosti mají.*

Adam, Jáchym; `grupytematko@gmail.com`
odevzdávejte do odevzdávátka

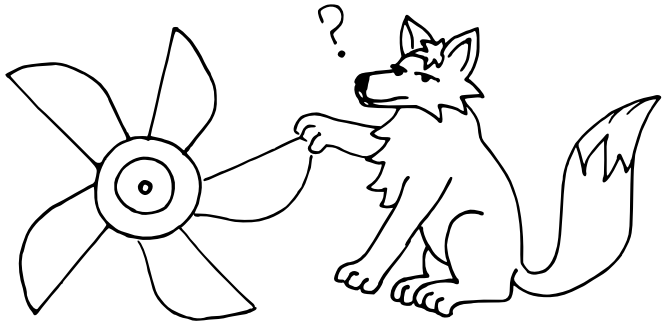
[4]Jako obvykle pro hlubší studium doporučíme článek z Wikipedie: `https://en.wikipedia.org/wiki/Quaternion_group`.

Téma 5 – Kombinatorika

Díl 3: Rotace

K pochopení tohoto dílu budete nejspíš potřebovat znát pojmy zobrazení/funkce (jedná se o synonyma) a jejich skládání. Pokud nevíte, co tyto pojmy znamenají, tak doporučujeme například text k 3. podzimní sérii PraSete z roku 2013 [5].

Zároveň je tento díl výrazně formálnější než předchozí dva, ale toho se nelekejte, všechny pojmy si intuitivně vysvětlíme. Už v prvním díle jsme se naučili řešit problémy, u nichž nám záleží, nebo nezáleží na pořadí vybíraných prvků. Co když nám však záleží na pořadí, až na rotaci? To je úloha, kterou zatím neumíme řešit. Abychom si ujasnili, o čem mluvíme, uvedeme si příklad. Chceme vytvořit náramky a máme 4 barvy korálků: modrou, zelenou, oranžovou a růžovou. Kolik různých náramků obsahujících právě 4 korálky můžeme udělat? Náramky jsou uzavřené, pootočením náramku nezískáme jiný náramek – například $MZZZ$ a $ZMZZ$ jsou shodné náramky složené z posloupnosti jednoho modrého a tří zelených korálků.



Důležité pojmy

Nyní si budeme muset definovat několik pojmů. Množinu všech permutací na množině M značíme $S(M)$. Zde ovšem budeme potřebovat přesnější definici pojmu permutace, než se kterou jsme doteď pracovali. V prvním čísle jsme si řekli, že permutace je „řazení prvků“. Přesněji se jedná o zobrazení, které nám každý prvek z M zobrazí jednoznačně na prvek M (tedy bijekci z M do M). Když jsme v prvním čísle počítali permutace, tak jsme počítali kolik různých těchto zobrazení na dané množině existuje. V našem původním příkladě jsme měli 4 kamarády a počítali, kolika různými způsoby je můžeme seřadit do fronty. To si můžeme představit tak, že máme nějakou počáteční frontu. Poté všechny fronty můžeme dostat, že nějak „proházíme“ tuhle původní frontu - neboli každému z kamarádů jednoznačně určíme, kde bude stát. Takže počet permutací jakožto jednoznačných zobrazení na nějaké množině skutečně bude odpovídat tomu, co jsme dříve počítali.

[5] https://prase.cz/archive/33/komplet3p.pdf

Teď si ovšem zvolíme jinou množinu M, než s jakou jsme pracovali doteď. Místo množiny korálků zvolíme množinu všech jejich čtyřčlenných posloupností (například $MZZZ$ a $ZMZZ$ jsou dva různé prvky naší nové množiny). To nám však dává možnost brát například posunutí korálků o jeden doprava (s tím, že poslední se posune na první pozici), jako permutaci na této množině, která například $MZZZ$ zobrazí na $ZMZZ$.

Permutace a jejich skládání mají následující vlastnosti:

(i) Pokud složíme dvě permutace, tak dostaneme opět permutaci.

(ii) Pokud máme tři zobrazení a chceme je složit, tak je jedno, jestli nejprve složíme první a druhé, nebo druhé a třetí.

(iii) Existuje permutace, která každý prvek M zobrazí sám na sebe (identita).

(iv) Ke každé permutaci existuje inverzní permutace (jejím složením s původní permutací dostaneme identitu).

Rozmyslete si, že tyto vlastnosti platí.

Díky těmto vlastnostem množina všech permutací spolu s operací skládání tvoří grupu (o nichž máme letos i celé tématko, pokud by vás zajímaly více), kterou si označíme $S(M)$ a říkáme jí grupa permutací na M (pro nás jsou relevantní pouze výše zmíněné vlastnosti, pojem grupa si zavádíme pouze kvůli jeho praktičnosti). Rozdíl mezi množinou $S(M)$ a grupou $S(M)$ je ten, že grupa v sobě kromě množiny $S(M)$ obsahuje i operaci skládání permutací. Z kontextu nám bude vždy jasné s čím pracujeme.

Podmnožinu G množiny $S(M)$ spolu s operací skládání nazveme podgrupou grupy $S(M)$, jestliže pro libovolnou permutaci z G obsahuje tato podmnožina i její inverz, pro každé dvě permutace z G obsahuje i jejich složení a obsahuje identickou permutaci. Příkladem podgrupy (která nás bude v našem příkladu zajímat) je podgrupa obsahující všechny rotace (tedy posunutí o 0 až 3 korálky doprava s tím, že korálek, který by se posunul na 5. místo, půjde na 1. místo atd.). *Rozmyslete si, že se skutečně jedná o podgrupu.*

Nyní budeme pracovat s nějakou pevnou podgrupou G (v našem případě podgrupou všech rotací). Pro libovolnou permutaci $\pi \in S(M)$ označíme

$$P_\pi = \{x \in M \mid \pi(x) = x\}$$

množinu tzv. pevných bodů, tedy bodů, které se v permutaci π zobrazí samy na sebe (například, pokud π je posunutí o 2 korálky doprava, tak se jedná o všechny posloupnosti korálků, které po tomto posunutí budou vypadat stejně jako na začátku – např. $MZMZ$).

Pro $x \in M$ označme

$$St_x = \{\pi \in G \mid \pi(x) = x\}$$

množinu těch permutací z G, které nehnou s x (například pro posloupnost korálků $MZMZ$ se bude jednat o identitu - tedy posunutí o 0 korálků a posunutí o 2 korálky).

Pro libovolný prvek x z M zavedeme orbitu O_x prvku x jako množinu

$$\{\pi(x) \mid \pi \in G\},$$

tedy množinu všech prvků, které můžeme dostat tak, že na náš původní prvek použijeme nějakou permutaci. V našem případě se jedná o všechny posloupnosti korálků, na které se jsme schopni dostat z nějaké původní posloupnosti x pouze posouváním korálků doprava – pro $x = MZMZ$ platí, že $ZMZM \in O_x$, ale $MZZM \notin O_x$. Pokud $y \in O_x$, tak $O_y = O_x$, neboli pokud jsme schopni se z y dostat na x nějakou operací z G, tak pak jsme se schopni z x i z y dostat na přesně stejnou množinu prvků. To má ale zajímavý důsledek. Všechny různé orbity v M nám množinu M rozkládají, každé $x \in M$ leží v právě jedné orbitě. Když se zamyslíme nad tím, co orbity představují v našem případě, zjistíme, že se jedná o množiny všech posloupností náramků, které na sebe můžeme rotovat. Tedy počet různých orbit je přesně to, co nás zajímá. Množinu všech (různých) orbit budeme značit $\mathcal{O}$.

Burnsidovo lemma

Nyní už se konečně dostáváme k tomu, jak řešit náš problém. Řešení nám poskytne takzvané Burnsidovo lemma, které formálně zapsané vypadá takto:

$$|\mathcal{O}| = \frac{1}{|G|} \sum_{\pi \in G} |P_\pi|. \tag{1}$$

Řečeno slovy, počet orbit je rovný průměrnému počtu pevných bodů všech permutací z G. To nám dává přesný recept, jak řešit náš problém. V našem G jsou 4 permutace (protože na 4 korálcích jsou 4 různé rotace). Pro rotaci o **0** korálků jsou všechny body pevné. Čtyřčlenných poslouností z korálků čtyř barev je 4^4, což je tedy i počet pevných bodů. Pro rotace o **2** je posloupnost korálků pevná, pokud se barva na 1. místě shoduje s barvou na 3. a barva na 2. se shoduje s tou na 4. (protože právě tyto pozice se na sebe zobrazí). Stačí nám určit počet různých posloupností o dvou prvcích (ve výsledné posloupnosti se tato dvojice zopakuje dvakrát za sebou), pevných bodů je tedy 4^2. Nakonec vezměme rotaci o **1** a o **3** korálky. V obou těchto případech jsou pevné body pouze posloupnosti, které mají všechny korálky stejné barvy. Použitím Burnsidova lemmatu dostaneme odpověď na naši otázku: $\frac{4^4+4^2+2\cdot 4}{4} = 70$.

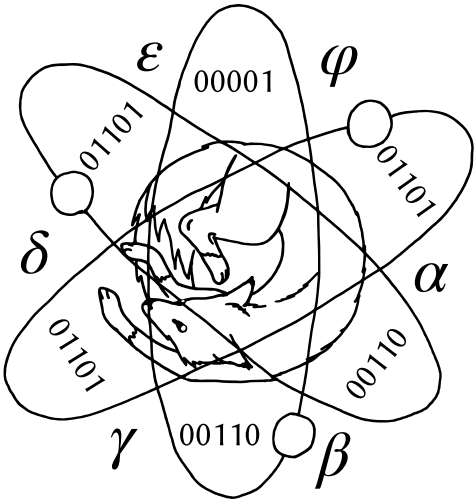
Důkaz lemmatu

Nejprve si dokážeme následující lemma:

(1) Je-li $x \in M$, tak $|O_x| \cdot |St_x| = |G|$.

(2) Jsou-li $x, y \in M$ a $y \in O_x$, tak $|St_x| = |St_y|$.

(1) Pro $y \in O_x$ označme π_y nějakou permutaci z G, která převádí x na y, tj. $\pi_y(x) = y$ (nějaká taková určitě existuje, jinak by y nebylo v O_x). Chceme ukázat, že dvojic $(y, \varrho), y \in O_x, \varrho \in St_x$ je stejný počet jako prvků G. Nejlépe to uděláme tak, že najdeme předpis, jak takové dvojici přiřadit prvek G tak, aby různým dvojicím odpovídaly různé prvky G a každý prvek v G byl použit. Dvojici (y, ϱ) přiřaďme prvek $F(y, \varrho) = \pi_y \circ \varrho$ (využíváme konvenci skládání zobrazení, kdy $f \circ g = f(g)$ - tedy nejprve aplikujeme zobrazení g a až poté zobrazení f) . Protože π_y i ϱ byly prvky G, je i jejich složení prvek G. Vezměme $(y_1, \varrho_1) \neq (y_2, \varrho_2)$. Nechť nejprve $y_1 \neq y_2$. Potom $F(y_1, \varrho_1)(x) = y_1$ a $F(y_2, \varrho_2)(x) = y_2$, protože ϱ_1 i ϱ_2 zobrazí x na x. Tedy zobrazení $F(y_1, \varrho_1)$ a $F(y_2, \varrho_2)$ se liší alespoň hodnotou v prvku x. Nechť je tedy $y_1 = y_2 = y$, a tudíž $\varrho_1 \neq \varrho_2$. Existuje proto nějaké $z \in M$, pro něž $\varrho_1(z) \neq \varrho_2(z)$. Ovšem pak i $F(y, \varrho_1)(z) \neq F(y, \varrho_2)(z)$. Z toho máme, že každá dvojice (y, ϱ) se zobrazí na jiný prvek G (neboli F je prosté). Nyní si zvolme $\sigma \in G$ a hledejme $y \in M$, $\varrho \in St_x$, že $F(y, \varrho) = \sigma$. Pokud ovšem vezmeme $y = \sigma(x)$ a $\varrho = \pi_y^{-1} \circ \sigma$ (ϱ je součástí St_x, protože $\sigma(x) = y$ a $\pi_y^{-1}(y) = x$), pak $F(\sigma(x) = y, \pi_y^{-1} \circ \sigma) = \pi_y \circ \pi_y^{-1} \circ \sigma = \sigma$ (tedy F je na). Tedy F jednoznačně zobrazí dvojici (y, ϱ) na prvky G, a proto je jejich počet stejný.



(2) Abychom ukázali, že $|St_x| = |St_y|$, tak opět najdeme nějaké zobrazení G, které každému členu St_x jednoznačně přiřadí člen St_y. Položme $G(\varrho) = \pi_y \circ \varrho \circ \pi_y^{-1}$ (π_y zvolíme stejně jako v důkazu (1)). Pro $\varrho \in St_x$ je $G(\varrho) \in St_y$, protože pokud zobrazujeme y, tak se nám nejprve v π_y^{-1} zobrazí na x, to se nám poté v ϱ nezmění a nakonec se v π_y zobrazí zpátky na y. Víme, že π_y^{-1}, π_y nám všechny věci zobrazují jednoznačně, tedy pokud máme dvě různé $\varrho_1, \varrho_2 \in St_x$, které se liší v tom, kam nám zobrazí nějaký prvek z, tak pak pokud pomocí $G(\varrho_1)$ a $G(\varrho_2)$ zobrazíme $\pi_y(z)$, tak dostaneme $\pi_y \circ \varrho_1(z)$ a $\pi_y \circ \varrho_2(z)$. Protože $\varrho_1(z) \neq \varrho_2(z)$ jedná se kvůli jednoznačnosti π_y o dva různé prvky, tedy ϱ_1, ϱ_2 se v G zobrazí na dvě různé permutace (protože se liší minimálně v tom, kam zobrazí $\pi_y(z)$). Zároveň pokud chceme najít permutaci, která se nám zobrazí v G na nějakou permutaci $\sigma \in St_y$, tak položíme $\varrho = \pi_y^{-1} \circ \sigma \circ \pi_y$, které je v St_x (to dokážeme stejně, jakože $G(\varrho) \in St_y$) a zároveň $G(\pi_y^{-1} \circ \sigma \circ \pi_y) = \pi_y \circ \pi_y^{-1} \circ \sigma \circ \pi_y \circ \pi_y^{-1} = \sigma$. Pokud si to dáme dohromady, zjistíme, že G každému členu St_x jednoznačně přiřadí člen St_y.

Nyní už si můžeme dokázat Burnsidovo lemma. Dokazovanou rovnost upravme na tvar $|\mathcal{O}| \cdot |G| = \sum_{\pi \in G} |P_\pi|$. Uvažme nyní množinu $A = \{(\pi, x) \mid \pi \in G, x \in P_\pi\}$, tedy všechny dvojice permutace a jejího pevného bodu. Nejprve berme permutace π z G jednu po druhé a pro každou zjistěme, kolik $x \in M$ je možno k π přidat, aby vznikla dvojice z A. Z toho víme, že $|A| = \sum_{\pi \in G} |P_\pi|$, protože pro každou permutaci π vezmeme právě její stacionární body. Tedy $|A|$ je rovna pravé straně lemmatu.

Teď naopak probírejme prvky M a pro každý určeme, kolik k němu můžeme přidat permutací π. Takto zjistíme, že $|A| = \sum_{x \in M} |St_x|$, protože pro každé x vezmeme právě ty permutace, které ho nezmění. Nyní pro každou orbitu $O \in \mathcal{O}$ vezmeme jeho zástupce x_O a protože orbity tvoří rozklad M a díky předchozímu lemmatu, části (2), víme, že pro každé $y \in O$ platí $St_y = St_{x_O}$, a tedy $|A| = \sum_{x \in M} |St_x| = \sum_{O \in \mathcal{O}} |O_{x_O}| \cdot |St_{x_O}|$ tím jsme si sumu po prvcích nahradili sumou po orbitách, protože pro každý $y \in O$ pro nějaké O je St_y vždy stejné. To si poté díky části (1) lemmatu můžeme dále upravit na $\sum_{O \in \mathcal{O}} |G| = |\mathcal{O}| \cdot |G|$. Dohromady tedy dostáváme $|A| = |\mathcal{O}| \cdot |G| = \sum_{\pi \in G} |P_\pi|$, což jsme chtěli dokázat.

Úlohy

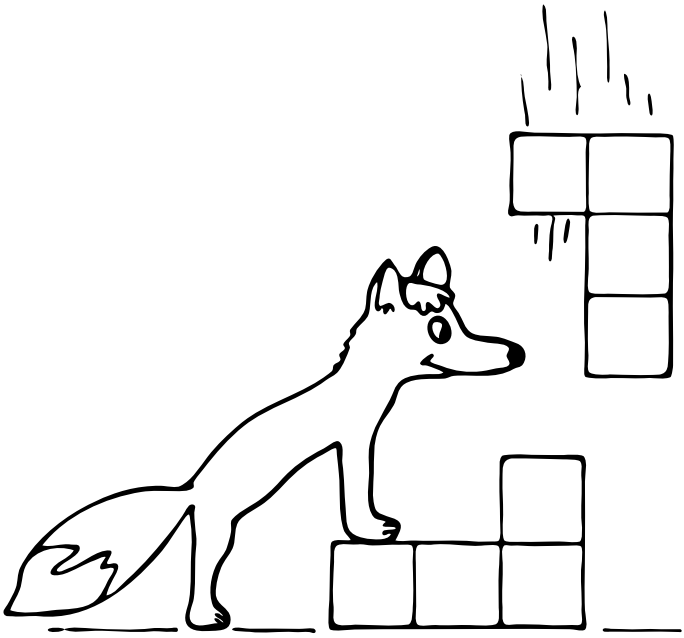
Úloha 5.1 [2b]: *Kolika způsoby můžeme obarvit tabulku 4×4 n barvami, jestliže považujeme za totožné tabulky, mezi nimiž můžeme přecházet otočením?*

Úloha 5.2 [4b]: *Máme tabulku tvaru $n \times 4$. Pepíček každé její políčko obarví jednou ze 3 barev. Určete nejmenší n takové, že nezávisle na Pepíčkově volbě barev můžeme vždy vytvořit jednobarevný čtverec 2×2 pouze prohazováním řádků a sloupců.*

Úloha 5.3 [4b]: *Jak se změní výsledek úlohy, kterou jsme počítali v tématku, pokud budeme za stejné považovat nejenom náramky, které na sebe otočíme, ale i náramky, které na sebe převrátíme - tedy efektivně zobrazíme zrcadlem (neboli nějakou rovinovou souměrností) (tedy například posloupnosti $ZMOR$ a $ZROM$ považujeme za stejné náramky)? Co když budeme moct pouze převracet a ne otáčet?*

Poznámka: Všechny úlohy nemusíte nutně řešit pomocí Burnsidova lemmatu.

Lukáš, Terka; troj.lukas@gmail.com
odevzdávejte do odevzdávátka



Téma 6 – LISP

Díl 3: Interakce s vnějším světem

Zatím jsme žili v bezpečné bublině REPLu, kde naše programy počítaly faktoriály, zpracovávaly seznamy a občas něco vypsaly na obrazovku. Skutečný software však musí komunikovat s okolním světem – číst soubory, posílat data po síti a reagovat na uživatele.

V tomto díle si ušpiníme ruce s IO (zkratka pro Input/Output) operacemi. Napíšeme si zjednodušený `wget`. To je program, který stáhne soubor přes HTTP. Typické použití z příkazové řádky je `wget http://example.com/`.

Proudy (Streams)

Všechna komunikace v Lispu (podobně jako v mnoha jiných jazycích) probíhá přes abstrakci zvanou **stream** (proud). Stream se chová jako trubka, kterou tečou data buď dovnitř (input stream), nebo ven (output stream), nebo oběma směry (io stream). V případě io streamu si můžete představit dvě souběžné trubky, jedna dovnitř a jedna ven.

Když zavoláme `(format T "Ahoj")`, Lisp ve skutečnosti posílá řetězec do speciálního streamu reprezentující standardní výstup (který standardně směřuje přímo do vašeho terminálu). Když čtete vstup pomocí `(read)`, čtete ze standardního vstupu. Oba tyto streamy jsou uloženy v globálních proměnných, které se jmenují `*standard-output*` a `*standard-input*`. `(format T "Ahoj")` je tedy ekvivalentní `(format *standard-output* "Ahoj")`.

Kouzlo je v tom, že funkce jako `print`, `format` nebo `read-line` nezajímá, kam píšou. Pokud jim podstrčíme stream, který nevede na terminál, ale do souboru, budou fungovat úplně stejně.

Práce se soubory

Poznatek o tom, že streamy se chovají podobně jako psaní/čtení do/z terminálu, uplatníme při práci se soubory. Zkusme si něco napsat do souboru. Nejprve soubor otevřeme pomocí funkce `open`, dostaneme stream a pak do něj zapíšeme pomocí `format`. Funkce `open` má spoustu klíčových parametrů, o kterých se dočtete v dokumentaci. První argument je cesta k souboru (řetězec nebo pathname). Další důležité parametry jsou:

- `:direction` má jednu ze tří hodnot `:input` (čtení, výchozí), `:output` (zápis), nebo `:io` (obousměrný),
- `:if-exists` neboli co dělat, když soubor už existuje při zápisu: `:error` (chyba, výchozí pokud `:direction` je `:output`), `:supersede`/`:overwrite` (přepsat), `:append` (začít psát na konec),
- `:if-does-not-exist` říká co dělat, když soubor neexistuje: `:error` (chyba, výchozí pokud `:direction` je `:input`), `:create` (vytvořit) nebo `nil` (vrátit `nil` místo streamu).

```
(defparameter *stream* (open "tajne.txt" :direction :output))
(format *stream* "Heslo je 1234")
(sleep 10) ; Nikam nespechame
(close *stream*)
```

Pokud program spadne před zavoláním `close`, soubor zůstane otevřený. To může vést k tomu, že se data nezapíšou na disk. Můžeme se bránit pomocí makra `with-open-file`, které se postará o to, že se soubor vždy zavře, ať už blok skončí normálně, nebo chybou.

```
(with-open-file (stream "tajne.txt" :direction :output
                                    :if-exists :supersede)
  (format stream "Heslo je 1234")
  (sleep 10) ; Porad nikam nespechame
)
```

Zkuste si program spustit v REPLu a pomocí Ctrl+C ho zabít, když spí. Uvidíte, že do souboru se data bezpečně zapíšou.

Čtení ze souboru Při čtení můžeme používat `read` (načte jednu S-expression) nebo `read-line` (načte řádek textu).

```
(with-open-file (vstup "data.txt" :if-does-not-exist NIL)
  (when vstup
    (loop for line = (read-line vstup NIL)
          while line do
      (print line)
    )))
```

Použili jsme složitější `loop` konstrukci s podmínkou, o které se můžete dočíst víc v *Lisp Cookbook: Iteration: Terminate the loop with a test (until/while)*. Fráze `(read-line vstup NIL)` znamená, že načteme řádek textu ze streamu `vstup`, a pokud narazíme na konec souboru (EOF), vrátíme `NIL`. Cyklus pak kvůli podmínce `while line` skončí, protože `line` bude nepravivá hodnota `NIL`.

Celý výraz je obalen `when`, které se spustí, pokud je jeho argument pravdivý, a nemá žádnou „else“ větev.

Síťová komunikace

Socket (doslova zásuvka, důlek) funguje podobně jako soubor a v abstraktním smyslu reprezentuje „místo“ v síťovém rozhraní počítače. Když si dva počítače chtějí povídat po síti, propojí dva sockety a posílají si mezi nimi data.

Abychom se připojili k nějakému počítači, potřebujeme znát jeho adresu, a ke kterému z jeho socketů se připojit. Potřebujeme tedy dvojici (adresa, port)[6], která jednoznačně (v rámci celého internetu) určuje socket, ke kterému se připojíme.

[6] Co přesně znamená adresa je nad rámec témátka, ale dobrým příkladem jsou domény, např. `example.com`, nebo adresy typu IPv4, např. `8.8.8.8`.

K našemu neštěstí Common Lisp ve své základní podobě sockety používat neumí. Když se v 80. a 90. letech Common Lisp standardizoval, každý operační systém a všechny implementace Lispu si udělaly vlastní rozhraní pro síťovou komunikaci. Pro přenositelný kód existuje knihovna `usocket` (*Universal Socket*), kterou načteme přes Quicklisp:

```
(ql:quickload :usocket)
```

Quicklisp je de-facto standardní balíčkovací systém pro Common Lisp (obdoba `pip` nebo `npm`). V některých distribucích SBCL může být Quicklisp už částečně nachystaný, ale nebudeme se na to spoléhat a raději si ukážeme, jak ho jednou provždy nainstalovat ručně.

Instalace Quicklispu krok za krokem

1. **Stažení instalačního souboru `quicklisp.lisp`**
   Quicklisp se instaluje jedním Lisp skriptem. Nejčastější varianta je stáhnout si ho z internetu do domovského adresáře:

   ```
   cd
   curl -O https://beta.quicklisp.org/quicklisp.lisp
   ```

   Na Windows můžeme Quicklisp nainstalovat pomocí PowerShellu (backtick na konci spojuje řádky):

   ```
   cd $HOME
   Invoke-WebRequest https://beta.quicklisp.org/quicklisp.lisp `
                     -OutFile quicklisp.lisp
   ```

2. **Spuštění Lispu a zavolání instalačního skriptu**
   Otevřeme si terminál, přejdeme do adresáře se souborem `quicklisp.lisp` a spustíme SBCL:

   ```
   cd cesta/kde/je/quicklisp.lisp
   sbcl
   ```

   V Lispovém REPLu pak napíšeme:

   ```
   (load "quicklisp.lisp")
   (quicklisp-quickstart:install)
   ```

   První příkaz načte instalační skript, druhý spustí samotnou instalaci. Quicklisp se standardně nainstaluje do adresáře `~/quicklisp` a chvíli stahuje metadata o balíčcích z internetu (podle rychlosti připojení to může trvat pár desítek sekund).

3. **Automatické zapnutí Quicklispu při startu Lispu**
   Zatím máme Quicklisp jen v aktuální relaci. Aby se nám vždy po startu SBCL automaticky načetl, necháme si upravit inicializační soubor (na Linuxu typicky `~/.sbclrc`). V REPLu zavoláme:

   ```
   (ql:add-to-init-file)
   ```

   Tento příkaz do vašeho `.sbclrc` přidá něco jako:

   ```
   (load (merge-pathnames "quicklisp/setup.lisp"
                          (user-homedir-pathname)))
   ```

   Odteď se Quicklisp při každém spuštění SBCL sám připraví, a dá nám k dispozici symbol `ql:quickload`.

4. **Ověření instalace a první balíček**
   Ukončíme aktuální SBCL (třeba `(quit)`), znovu ho spustíme a v REPLu zkusíme načíst nějakou knihovnu, například `usocket`:

   ```
   (ql:quickload :usocket)
   ```

   Pokud instalace proběhla správně, Quicklisp balík poprvé stáhne z internetu, nainstaluje ho a na konci bychom měli vidět hlášku, že systém `usocket` byl úspěšně načten.

5. **Poznámka pro Windows**
   Dál je postup stejný: spustíme `sbcl`, v REPLu zavoláme nejprve příkazy `(load "quicklisp.lisp")`, `(quicklisp-quickstart:install)`, a nakonec ještě `(ql:add-to-init-file)`. Inicializační soubor může mít na Windows jinou cestu (např. v „Uživatelském“ profilu), ale Quicklisp se o nalezení správného místa postará sám.

Stáhneme si soubor z internetu

Jak jsme předesílali, stáhneme si soubor z internetu. Už umíme uložit data do souboru, takže nám chybí jen schopnost data stáhnout z internetu.

HTTP je textový protokol, tedy způsob, jakým se počítače mezi sebou domlouvají. Socket sice umožňuje komunikaci, ale HTTP specifikuje, jak tato komunikace vypadá. Textový protokol znamená, že počítače mezi sebou přenášejí hlavně lidmi čitelný text.

HTTP je velmi rozšířený protokol, a má více verzí, které jsou postupně složitější a výkonnější, přičemž nejnovější je `HTTP3`, které už není ani textové, a používá velmi moderní způsob komunikace přes sockety (QUIC). Implementace takového protokolu je zcela nad rámec našich možností, a spokojíme se s nejjednodušší verzí, což je `HTTP1.0`.

Typická „konverzace“ mezi klientem (to jsme zatím my) a serverem vypadá tak, že klient pošle požadavek a server odpoví. Nejjednodušší typ požadvku je `GET` (anglicky dostat, získat), který server žádá o nějaký soubor:

```
GET /index.html HTTP/1.0
Host: httpforever.com
    <- prazdny radek
```

První řádek říká „chci soubor `/index.html` a mluvit protokolem HTTP/1.0“. Hlavička `Host` identifikuje server (na jedné adrese může běžet víc webů). Prázdný řádek ukončuje hlavičky.

Odpověď serveru:

```
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 5124
                                        <- prazdny radek
<!doctype html>
<html>...
```

Stavový kód 200 znamená „OK“. Po hlavičkách (opět oddělených prázdným řádkem) následuje tělo – obsah souboru.

První pokus

Zkusme se připojit k `example.com` a vypsat, co nám server pošle:

```
(ql:quickload :usocket)
(defparameter *socket* (usocket:socket-connect "example.com" 80))
(defparameter *stream* (usocket:socket-stream *socket*))
(defun http-line (stream radek)
  (format stream "~a~c~c" radek #\Return #\Linefeed)
) ;; Zapis radek zakonceny CRLF do stream
(http-line *stream* "GET / HTTP/1.0")
(http-line *stream* "Host: example.com")
(http-line *stream* "")
(force-output *stream*)
(loop for line = (read-line *stream* NIL)
      while line
      do (format t "~a~%" line))
(usocket:socket-close *socket*)
```

Pár věcí stojí za zmínku:

- HTTP vyžaduje, aby řádky končily sekvencí CR+LF (Carriage Return + Line Feed), ne jen LF. Proto nemůžeme použít `~%`, ale musíme oba znaky zapsat explicitně pomocí `~c`.
- `force-output` vynutí odeslání dat. Operační systém obvykle předává data do sítě v blocích (bufferování, typické jsou bloky po 1 KB). Bez `force-output` by požadavek mohl zůstat ve vyrovnávací paměti a zbytečně čekal, až bude blok dokončen.
- `(read-line stream NIL)` vrátí `NIL` místo chyby, když narazí na konec streamu.

Po spuštění uvidíte nezpracovanou HTTP odpověď v podobně hlaviček a těla oddělených prázdným řádkem.

Kompletní wget

Teď z toho uděláme funkci. Dostane adresu, cestu k požadovanému souboru, přeskočí hlavičky odpovědi a tělo uloží do souboru na zadané adrese na počítači:

```
(defun http-line-fmt (stream &rest zbytek)
  (http-line stream (apply #'format (cons NIL zbytek)))
)
;; Jako read-line ale dokaze z konce odstranit CRLF
(defun read-crlf-line (&rest argumenty)
  (let ((radek (apply #'read-line argumenty)))
    (if radek
      (string-right-trim '(#\Return) radek)
      NIL ;; (string-right-trim '(cokoli) NIL) vrati "NIL"
    )))
(defun empty-or-nil (retezec)
  (if retezec (string= retezec "") T)
)
(defun wget-unguarded (stream host cesta vystupni-stream)
  (progn
    (http-line-fmt stream "GET ~a HTTP/1.0" cesta)
    (http-line-fmt stream "Host: ~a" host)
    (http-line-fmt stream "")
    (force-output stream)
    ;; Cti hlavicky dokud nenarazis na prazdny radek
    (loop for hlavicka = (read-crlf-line stream NIL)
          until (empty-or-nil hlavicka) do
          (format T "~a~%" hlavicka))
    ;; Ukladej dalsi radky do souboru dokud nenarazis na konec
    (loop for radek = (read-crlf-line stream NIL)
          while radek do
          (format vystupni-stream "~a~%" radek))
  ))
(defun wget (host cesta vystupni-soubor)
  (with-open-file (vystupni-stream vystupni-soubor :direction :output
                                                   :if-exists :supersede)
    (let* (
      (socket (usocket:socket-connect host 80))
      (stream (usocket:socket-stream socket)))
      (unwind-protect
        (wget-unguarded stream host cesta vystupni-stream)
        (usocket:socket-close socket)
    )))
)
```

`unwind-protect` zajistí, že se socket zavře, i když kód uprostřed vyhodí chybu. Je to nízkoúrovňový mechanismus, na kterém stojí i `with-open-file`.

Vyzkoušíme:

```
(wget "httpforever.com" "/index.html" "stahnuto.html")
```

Úlohy

K testování můžete použít libovolnou webovou stránku. Odevzdávejte do odevzdávátka.

Úloha 5.1 [1b]: *Upravte funkci* `wget` *tak, aby na standardní výstup vypsala pouze stavový kód odpovědi (např.* `200`*) a hodnotu hlavičky* `Content-Type`*.*

Úloha 5.2 [1b]: *Přidejte podporu pro používání standardních URL adres stylu* `http://httpforever.com/index.html`*.*

Úloha 5.3 [2b]: *Přidejte podporu pro přesměrování. Pokud server vrátí stavový kód 301 nebo 302, najděte v hlavičkách hodnotu* `Location` *a stáhněte soubor z nové adresy. Omezte maximální počet přesměrování na 100, aby program necyklil.*

Problém 5.4: *Kromě* `GET` *existují i jiná HTTP „slovesa“. Přečtěte si o nich a některá z nich implementujte s vhodným rozhraním.*

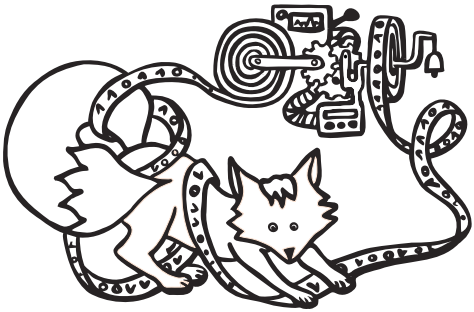
Problém 5.5: *Implementujte HTTP server, který bude schopný odpovědět naší* `wget` *implementaci na dotazy. Obsah odpovědi je na vás.*

Jan Koška; `jan.koska@email.cz`
odevzdávejte do odevzdávátka

Řešení 3. dílu

Úlohy 3.1, 3.2, 3.3

Zadání:
***Úloha 3.1**: Přečtěte ze vstupu dvě čísla na dvou řádcích a sečtěte je.*
***Úloha 3.2**: Přečtěte ze vstupu jedno číslo n a vypište čísla 1 až n.*
***Úloha 3.3**: Přečtěte číslo k na prvním řádku. Následně přečtěte k čísel na oddělených řádcích. Určete a vypište jejich průměr, minimum a maximum.*



Řešení od Mgr.^{MM} Františka Nouzy: Mgr.^{MM} František Nouza nám poskytl velmi jednoduché řešení, které snad trochu drze používá (`read`) bez uložení do proměnných. Níže najdete mírně upravenou variantu — originál používal `let` a několik `setf` místo jednoho `let*`.

```
(defun sum-two-numbers ()
  (+ (read) (read)))

(defun range-to-n ()
  (loop for i from 1 to (read) do
    (print i)
  )
)

(defun stats ()
  (let* ((k (read)) (temp (read)) (sum temp) (mn temp) (mx temp))
    (loop for i from 2 to k do
      (setf temp (read))
      (if (< temp mn) (setf mn temp))
      (if (> temp mx) (setf mx temp))
      (setf sum (+ temp sum))
    )
    (print (list (/ (float sum) k) mn mx)))
  )
```

Úloha 3.4

Zadání:

Zkontrolujte, že zadaný řetězec má správně spárované kulaté závorky.

Řešení od Martina Vagnera: Řešení poskytnutné Martinem Vagnerem si nejprve předpočítá, které znaky odpovídají zvýšení/snížení stupně vnoření.

```
(defun balanced-parentheses-p (text)
  (let (
    (arr (map 'list #'(lambda (x)
      (cond
        ((eq x *leva-zavorka*) 1)
        ((eq x *prava-zavorka*) -1)
        (T 0)
      )) text))
    (s 0))
    (dolist (a arr)
      (setf s (+ s a))
      (if (< s 0)
        (return-from balanced-parentheses-p NIL)))
    (if (= s 0) T NIL)))
```

Úloha 3.5

Zadání:

Implementujte funkci `(split-string <string> <delimiter>)`*, která rozdělí řetězec na podřetězce oddělené daným oddělovačem.*

Řešení od Doc.MM Julie Klementové:

```
(defun split-string (string delimiter)
  (let ((start 0)
      (vysledek '())
    (rozdel (char delimiter 0)))
    (loop for pos = (position rozdel string :start start)
      while pos do
        (push (subseq string start pos) vysledek)
        (setf start (1+ pos))
      finally
        (push (subseq string start) vysledek)
    )
    (setf vysledek (reverse vysledek))
    (print vysledek)))
```

Úloha 3.6

Zadání:
Implementujte váš oblíbený algoritmus pro třídění.

Řešení od Mgr.^{MM} Lukáše Komy:

Quicksort nejen rychle třídí, ale taky se snadno implementuje.

```
(defun my-sort (sequence compare-fn)
  (if (<= (length sequence) 1)
    sequence
    (let ((greater-lower (split
                          (cdr sequence)
                          (car sequence)
                          compare-fn)))
      (append
        (my-sort (car greater-lower) compare-fn)
        (list (car sequence))
        (my-sort (nth 1 greater-lower) compare-fn)
      )
    )
  )
)
```

Úloha 3.8

Zadání:
Zkontrolujte, že řetězec má správně spárované kulaté, hranaté i složené závorky (např. "[(])" není správně).

Řešení od Doc.^{MM} Julie Klementové:

```
(defun matching-bracket-p (left right)
  ;; Vrati T, pokud RIGHT odpovida LEFT.
  (or (and (string= left *leva-kulata*)
           (string= right *prava-kulata*))
      (and (string= left *leva-hranata*)
           (string= right *prava-hranata*))
      (and (string= left *leva-chlupata*)
           (string= right *prava-chlupata*))))

(defun balanced-multi-brackets-p (text)
  (let ((stack '()))
    (loop for ch across text do
      (cond
        ;; leva zavorka -> prida se na zasobnik
        ((member ch (list *leva-kulata*
```

```
                        *leva-hranata*
                        *leva-chlupata*)
                :test #'string=)
         (push ch stack))
        ;; prava zavorka -> zkontroluje shodu s posledni levou
        ((member ch (list *prava-kulata*
                        *prava-hranata*
                        *prava-chlupata*)
                :test #'string=)
         ;; prazdny zasobnik nebo nesedi -> spatne
         (if (or (null stack)
                 (not (matching-bracket-p
                        (pop stack) ch)))
             (return-from
               balanced-multi-brackets-p nil)))))
    (null stack)))
```

Problém 3.9

Zadání:
Vytvořte interpreter. Volba toho, co bude interpretovat, je na vás. Předkládáme vám možnost interpretovat aritmetické výrazy, možnosti interpretovat brainfuck a samozřejmě i možnost interpretovat cokoli uznáte za vhodné.

Pozn. red.: všichni řešitelé se rozhodli interpretovat Brainfuck.

Řešení od Doc.^{MM} Michaela Jarvise: Řešení od Doc.^{MM} Michaela Jarvise používá k reprezentaci paměti trojici `(negative-mem positive-mem pointer)`, kde dva seznamy pokrývají zápornou a kladnou stranu pásky. Díky tomu se ukazatel může posouvat oběma směry. Funkce `bf-mem-set` podle znaménka ukazatele vybere správný seznam, v případě potřeby ho prodlouží nulami a zapíše hodnotu.

Před samotným vykonáváním programu se v jednom průchodu zdrojovým kódem spárují závorky `[` a `]` do seznamu `portals`: pro každou pozici závorky si zapamatujeme index té protější. Při běhu pak skok na odpovídající závorku stojí $O(1)$, protože se stačí podívat do `portals`.

```
(defun bf-mem-set (mem val)
  (let ((p (nth 2 mem)) (m nil) (mi nil))
    (if (< p 0)
      (progn
        (setf mi 0)
        (setf p (- -1 p)))
      (setf mi 1))
    (setf m (nth mi mem))
    (setf m (extend-list m (+ p 1)))
    (setf (nth p m) val)
```

```
    (setf (nth mi mem) m)))

(defun run-brainfuck (source getchar putchar)
  (let ((portals nil))
    ;; Pair [] brackets together
    (let ((pstack (make-stack)))
      (loop for i from 0 to (- (length source) 1) do
        (setf portals (append portals (list nil)))
        (if (eq (char source i) #\[)
          (stack-push i pstack)
          (when (eq (char source i) #\])
            (let ((v (nth 0 (stack-pop pstack))))
              (setf (nth v portals) i)
              (setf (nth i portals) v)))))))
    ;; run
    (let ((mem (make-bf-mem)) (pc 0))
      (loop
        (cond
          ((eq (char source pc) #\>)
           (bf-mem-move-ptr mem 1))
          ((eq (char source pc) #\<)
           (bf-mem-move-ptr mem -1))
          ((eq (char source pc) #\,)
           (bf-mem-set mem (char-int (funcall getchar))))
          ((eq (char source pc) #\.)
           (funcall putchar (code-char (bf-mem-get mem))))
          ((eq (char source pc) #\+)
           (bf-mem-set mem (+ (bf-mem-get mem) 1)))
          ((eq (char source pc) #\-)
           (bf-mem-set mem (- (bf-mem-get mem) 1)))
          ((eq (char source pc) #\[)
           (when (eq (bf-mem-get mem) 0)
             (setf pc (nth pc portals))))
          ((eq (char source pc) #\])
           (unless (eq (bf-mem-get mem) 0)
             (setf pc (nth pc portals)))))
        (incf pc)
        (when (eq pc (length source))
          (return-from run-brainfuck mem))))))
```

Téma 26 – Šifrování

Řešení 4. dílu

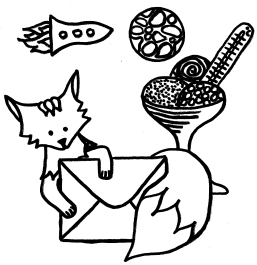
Šifrovací tématko se s vámi v tomto díle loučí a přináší řešení šifer z minulých dílů. Pokud se chcete šifrování dále věnovat, doporučujeme navštívit některou šifrovačku[7] nebo soustředění M&M, kde šifrovačka bývá často součástí programu.

Úloha 4.1

Zadání:
Vyřešte následující šifry. Zajímá nás nejen heslo, ale i princip řešení a jak jste na něj přišli.

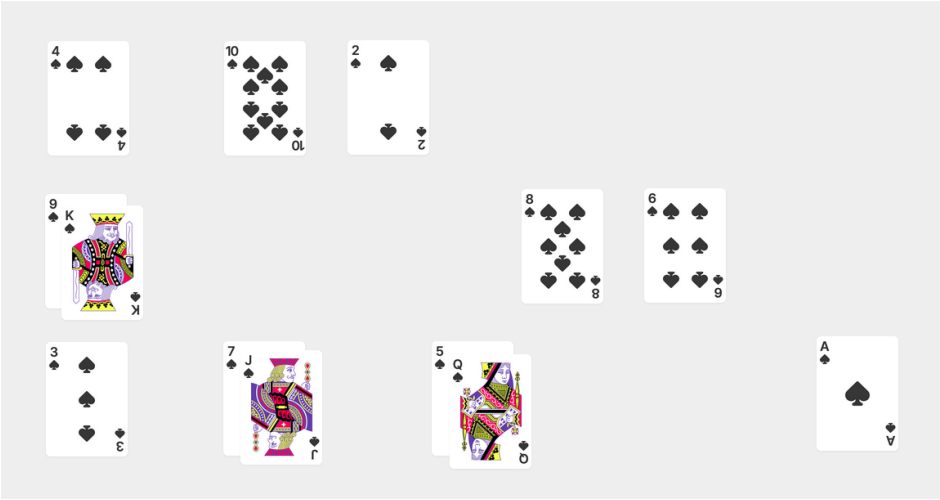
Šifra (a)
Daniela odbouraly blbosti.
Ryj!
Emil spí.
I. sepsaná práce
Obr cupuje inauguraci.
Tolerance elipsy je sedmiprocentní.
abstraktní mat
O hloupém Lobotomovi
Adiabaticky stlačil kost yetti.

Řešení:
Nejprve si přečteme popořadě první písmena všech slov. Dostaneme instrukci DOBŘE SI SPOČÍTEJ SAMOHLÁSKY. Poté, co v každém slově spočítáme samohlásky, dostaneme pro každý řádek jedno až čtyři čísla od 1 do 6 v nějakém pořadí. No a když je někde nejvýše 6, obvykle to značí Braillovo písmo. Ani tentokrát to nebude jinak. Stačí vybarvit ta políčka, která nám očíslovaly jednotlivé řádky. Jeden řádek je jedno písmeno. Čísluje se nejdříve levý sloupec shora od 1, pak pravý sloupec shora od 4 (což je standardní číslování uvedené v mnoha šifrovacích pomůckách). Po dosazení vyjde tajenka JABLUNKOV.

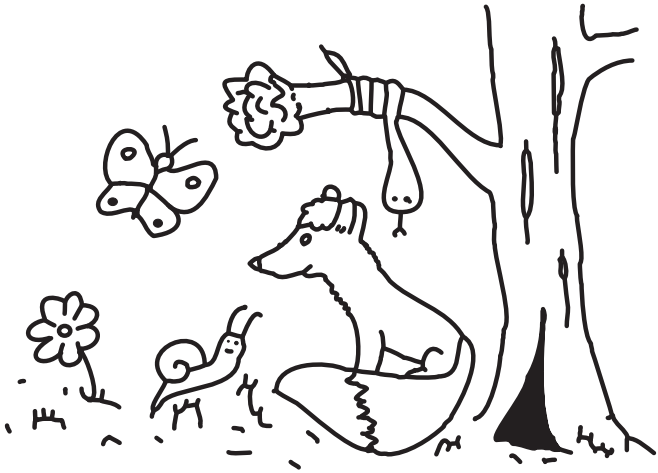


[7] Přehled nadcházejících šifrovaček najdete třeba zde: https://sifrovacky.cz/kalendar/.

Šifra (b)



Řešení:
Karty tvoří neúplnou tabulku 3 krát 9, to nás navádí na rozložení polského kříže[8]. Každé karta má své číslo (nebo symbol), které určuje pořadí písmen. Tajenkou je: ZDRAVOTNICTVÍ.



[8] https://1divci.skauting.cz/oddil/sifra-velky-polsky-kriz/

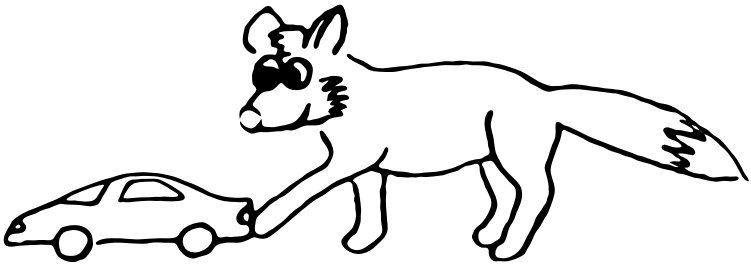
Šifra (c)
PeS Péťa se nějakým způsobem vyskytl v Indii. Moc se mu tam nelíbilo, a tak se vydal jinam. Nejprve šel dva dny na sever a pak tři dny na východ – a heleme se, skončil v Japonsku. Tam se mu moooc líbilo a udělal si tu fotečku s horou Fudži.

V Japonsku se mu líbilo tak moc, že se vydal ještě na ostrov Hokkaidó, který je jeden den cesty na sever od jeho hotelu. Jakmile tam dorazil, ochutnal místní dýňovou polévku a nezapomněl se tímto kulinářským zážitkem pochlubit na instagramu.

Když pak večer seděl na pláži, dumal nad tím, co se asi skrývá na druhé straně téhle velké louže. A tak se tam druhý den vydal na výlet. Během tříhodinové cesty lodí na západ si užíval mořského vzdoušku kolem. Když dorazil, zjistil, že je v Rusku. To ho velmi vylekalo, a pro jistotu poslal kamarádům svoji fotku, kdyby ho už náhodou nikdy neuviděli.

Z Ruska rychle odjel zpátky lodí tam, odkud ráno přijel – na ostrov Hokkaidó. Tam si mohl oddychnout a poslat kamarádům fotku jako důkaz, že se ve zdraví vrátil do civilizace. Ze všeho toho cestování začal být už Péťa unavený, a tak se vydal zpátky domů. Cesta na letiště na jihu mu zabrala čtyři hodiny. Pak už se jen proletěl nějakých jedenáct hodin na západ a byl zpátky doma. Jako první si zašel do svojí oblíbené hospody na jedno správně vychlazený. Na jeho instagramu se večer objevila fotečka s popiskem #jedineplzen.

Řešení:
PeS Péťa – odkazuje na Periodickou Soustavu Prvků. Péťa začíná v Indii, takže začneme na Indiu. Po tabulce se pak pohybujeme vždy o tolik políček (prvků), kolik udává číslo u časového údaje uvedeného u pohybu Péti. Například „šel dva dny na sever a tři dny na východ“ znamená, že se v PSP posuneme o dvě políčka nahoru a o tři doleva – skončíme na síře. Vždy, když si Péťa udělá fotku, zapíšeme si zkratku prvku, na kterém se zrovna nacházíme. Takhle postupujeme dál, až nám vyjde tajenka: SOBOTa.



Šifra (d)

Stalo se vám někdy, že jste si od někoho půjčili auto, on vám předal klíčky a vy jste nastoupili v domnění, že hned odjedete, ale vzápětí jste zjistili, že nevíte, jak zařadit zpátečku? Vyzkoušeli jste všechny možnosti, dopředu, dolů, dozadu… ale ani jedna z nich nechtěla fungovat? Po několika minutách tedy zvedáte telefon a voláte kamarádovi: „Promiň, že tě ruším, ale jak že se řadí zpátečka?“ Ten vám konečně poradí správný způsob: doleva a dozadu, zařadíte, a konečně jedete do vysněné destinace.

Přesně toto se mi nedávno stalo. Auto jsem si sice nepůjčovala, ale nevěděla jsem si rady, jak začít couvat. Způsob řazení mě lehce překvapil, ale což, ne všechny manuály to mají stejně. Po úspěšném zařazení zpátečky vyjíždím na silnici do vysněné destinace. Nejrychlejší cesta vede přes dálnici, na tu je to kousek, ale snad nepřejedu nájezd. V připojovacím pruhu se rozjíždím už na čtyřku a zrychluju pomalu až na nejvyšší rychlostní stupeň, pětku. Stále se na to řazení musím hrozně soustředit.

Jedu si to hezky 130 po dálnici, na předjíždění se občas hodí podřadit třeba na čtyřku, ale jinak pokračuju stálou rychlostí. Jak se kochám výhledy na hrádek v okolí, tak se pomalu blížím k malé koloně aut. Docela nečekaně zpomaluji, takže se spojkou řadím rovnou trojku, protože nic vyššího již nedává smysl. Kolona jede pomaličku vpřed a zanedlouho zjišťuje, že je způsobena silniční prací, opět. V jednu chvíli dokonce zastavíme úplně. Rozjíždět se je potřeba samozřejmě na jedničku, ale vypadá to, že dál už to pojede, takže můžu i dvojku.

Tak, to bylo příliš naivní. Kolona zase zastavila a já se musím zase rozjíždět, zase jednička. Pomalu se ploužím společně s ostatními, ale aspoň mi hrají fajn písničky. Vypadá to nadějně a myslím, že teď už to pojede. Už mám dvojku. Super. Vypadá to, že se vše rozjíždí a už zase jedou všichni stovkou.

No nic, dostávám trochu hlad, a tak asi zastavím na nejbližší benzínové pumpě. Sjíždím, zastavuji na parkovišti a jdu si něco koupit. Nastupuju zpět a zase to rozjíždění, ach jo, doufám, že se rozjedu bez problému, protože v tomhle autě motor občas trochu zlobí. Zpátečka, tak, a teď jednička a jedem!

Ale ne, tak ještě nejedem, musím si ještě trochu couvnout, protože jinak dojedu maximálně do auta před sebou. Takže, zase zpátečka a povedlo se. Zpátky na dálnici, hezky se rozjet se čtyřkou stejně rychle jako ostatní a můžeme frčet.

Už se blížím ke svému sjezdu a vypadá to na prudkou zatáčku. Raději podřídíme až na trojku, abych neskončila ve svodidlech. Tak, pohoda, dále pokračuju po větší rovné silnici, kde se dá jet rychle, takže zase čtyřka. I z této velké silnice zanedlouho sjíždím a blížím se k chatě. Tak, jsem tu, už jen zacouvat na místo, zpátečka, a je to. Nakonec to jelo hezky.

Řešení:
Všimneme si množství řazení v textu. V každém odstavci se obvykle nachází několikrát, a pokud pomocí toho „vybarvíme“ políčka v Braillově písmu, tak dostaneme řešení AUTÍČKO. Klíčem k přiřazení jednotlivých stupňů řazení a políček v Braillově písmu je netypické rozložení řadící páky, na které napovídá první odstavec, otočené o 90° vpravo. Každý stupeň tak odpovídá jedné z šesti pozic znaku, jak je to znázorněno na obrázku.

R	1
2	3
4	5

Šifra (e)
Mám nápad na šifru.
On mě napadl celkem nedávno.
Rým určitě používat nebudu.
Soud šifer by z toho nebyl nadšený.
Ér, kde byl rým populární, už mají řešitelé plné zuby.
Ód bych na rým mnoho nedostala.
Vím, že důležitá budou naopak první slova.
Au, jasně že si zrovna nakopnu palec.
Já mám fakt zatracenou smůlu.
Ta šifra mi ale přijde docela dobrá.
Jen doufám, že se bude líbit i řešitelům.
Jsou fakt moc šikovní.
Ti tu šifru určitě vyluští.
Kéž by se jim i líbila.
Jo, taky doufám, že v tom nebudou hledat nic složitého.
Vždyť většina toho textu vůbec k ničemu není.
Čtou ho vůbec?
I když bych jim tam vlastně mohla dát nějakou nápovědu.
No, ale nebude to pak až moc zřejmé?
Tou nápovědou bych to taky mohla celé zkazit.
Tak udělám to?
No, asi jo.
Co když si ale té nápovědy vůbec nevšimnou?
Dvou nápověd by si všimnout mohli.
Tří určitě.
A já bych tu šifru měla co nejdříve dodělat.
Jé, ono už je tolik hodin!
Lžou ty hodiny?
Ne, jasně že nelžou.
Áá, to je textu!
Hou, mám dopsáno!

Řešení:
Text dokumentuje myšlenkový pochod osoby, která má nápad na šifru a snaží se domyslet detaily. Popisovaná šifra je ovšem přímo ta, kterou se zrovna snažíme vyluštit. Z textu se dozvídáme, že šifra obsahuje tři nápovědy: Většina toho textu vůbec k ničemu není. Důležitá budou naopak první slova (namísto posledních jako tomu je u rýmů). A když se podíváme na první písemena řádků/vět, dostaneme na začátku slovo MORSEOVA (resp. MORSÉÓVA).

Co s tím? Zaměříme se na první slovo každého verše, na zbytek můžeme zapomenout. Všechna tato slova jsou jednoslabičná, obsahují buď jednu krátkou nebo dlouhou samohlásku, nebo dvouhlásku. Použijeme morseovku – krátká samohláska je tečka, dlouhá je čárka, dvouhláska je mezera mezi písmeny. Vyjde nám KODLISKA, odpověď je tedy LIŠKA.

Šifra (f)
71, 13, 39, 46„ 17, 39, 46„ 17, 79, 93, 31„ 13, 39, 97, 74„ ,

Řešení:
V na první pohled náhodné řadě dvojic čísel nás mohou zaujmout některé často se vyskytující dvojice, navíc druhé a první číslo ze dvou po sobě jdoucích dvojic se často shodují, jako by na sebe dvojčíslí navazovala. To nás po troše zkoušení navádí na to, že dvojčíslí označují spojnice v tabulce 3 krát 3 popsané po řádcích čísly od 1 do 9. Každý úsek oddělený dvojitou čárkou tak v tabulce vykreslí jedno písmeno, řešení je AHOJ.

Úloha 3.2 a 4.2

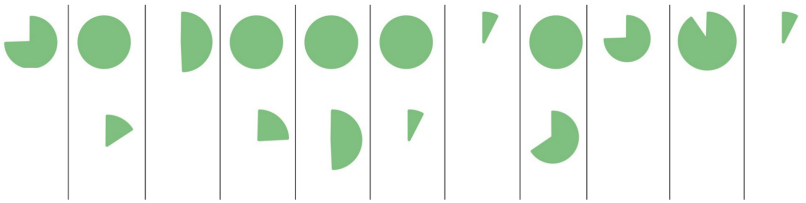
Šifra 1
Jednoho dne seděl Tomáš u svého počítače, když náhle dostal skvěou slevu na knihu o magii.
Na pdě svého domu se naučil levitovat nad zemí pomocí starého kouzla.
Koupil si levný taliman na trhu s kuriozitami.
Najednou přišla zpráva od jeho starého přítele Vinenta, Tomáš rychle vyťukal na klávesnici odpověď, že přijede.
Rozhodl se cestovat městskou hromadnou dopravou až do daleké Ameriky, do města ashingtonu.
Tm se zúčastnil rozpravy o tajemných silách v dějinách lidstva.
Během ní rozumně řekl pravdu o svých izarních schopnostech.
Později se učil pravopis v nové škole magie, aby nedělal tak často hrbé chyby, a dokonce se naučil i psát všemi deseti.
V jeho zahradě rostla kouzelná levandule, která svou ůní uměla léčit.
Nakonec tak dodržel nejdůležitější pravidlo z ikipedie, tedy že má pomáhat ostatním.

Řešení:
Po bližším ohledání si lze všimnout, že v každé z deseti vět chybí jedno písmeno a zároveň je v jednom ze slov schovaná strana (levá ve slovech sleva, levandule, levitovat, levný; pravá ve slovech zpráva, doprava, rozprava, pravda, pravopis, pravidlo). Co s tím dál, nám poradí časté zmíňky o počítači a psaní na klávesnici. Z každé věty získáme jedno písmeno tak, že na klávesnici vyjdeme od chybějícího písmena a podíváme se na klávesu nalevo nebo napravo od něj (např. v první větě se podíváme od písmene L doleva a získáme písmeno K). Jako řešení tedy vyjde KLÁVESNICE.

Šifra 2
HAMBURGER _ JABLKO _ HRUŠKA _ MRKEV
BERLÍN _ BROSKEV _ BRUSEL _ BUDAPEŠŤ
VEVERKA _ KOLOTOČ _ MRAVENEC _ KOZA
BRATROUCHOV _ SKLENÉ _ VRCOVICE _ HODOŇOVICE
PROSÍM _ AHOJ _ DĚKUJI _ RÁDO SE STALO

Řešení:
I pro tuto šifru platí, že z každé řádky získáme jedno písmeno. Vždy tři ze čtyř slov na řádku mají spolu něco společného a my hledáme slovo, které do řady nepatří. Index tohoto slova v řádce je index písmene v tomto slově, které potřebujeme do řešení šifry. Dostáváme hamburger (není to ovoce ani zelenina) → H, broskev (není město) → R, kolotoč (není zvíře) → O, Vrcovice (nebylo tam soustředění M&M) → C, ahoj (je to pozdrav) → H, dohromady HROCH.

Šifra 3



Řešení:
Z každého sloupce získáme jedno písmeno tak, že si na zeleném obrazci představíme ciferník hodin a odečteme, kolik hodin už uběhlo (pokud tam jsou dva ciferníky nad sebou, přičítáme k počtu hodin na spodním ciferníku 12). Podle toho, kolik je hodin, tolikáté písmeno abecedy hledáme. Počty hodin jsou 9, 14, 6, 15, 18, 13, 1, 20, 9, 11, 1, po převedení na abecedu vyjde INFORMATIKA.

Šifra **4**

V malém městě stál mladý detektiv Jakub u dvojice semaforů, ale ten první byl nějak rozbitý; svítila na něm všechna světla. Naštěstí alespoň ten druhý fungoval tak, jak měl, a svítilo na něm jen červené světlo. Jakub i přesto přešel na druhou stranu ulice, přičemž se ohlédl přes rameno, jako by cítil, že ho někdo sleduje. Ulice byla prázdná, jen vítr šelestil listy na stromech, a on si pomyslel, že tohle dobrodružství bude možná větší, než čekal.

Z batohu vylovil bonboniéru, ale když ji otevřel, pocítil hořké zklamání. Mladší bratr zase vyjedl všechny jeho oblíbené pralinky s nugátem a na něj zbyly jen první bonbony z první a třetí řádky a druhý bonbon z druhé řádky, tedy jen ty hnusné lékořicové. Jakub si povzdechl. Pokračoval dál a dorazil k starému domu na konci ulice. Dostal informace, že by měl být dům opuštěný, ale rozsvícená světla v celém levém křídle a svítící světlo v přízemí pravého křídla svědčila o opaku. „No nevadí," pomyslel, stejně to tu musím omrknout. Srdce mu bušilo rychleji, když se blížil k bráně. Dům vypadal jako z hororového filmu – vysoké zdi, zarostlé břečťanem. Jakub se nadechl a připravil se na to, co přijde.

Paklíčem si odemkl zadní vchod a potichu vlezl dovnitř a ocitl se v dlouhé chodbě. Rád by si dal nyní něco sladkého na nervy. Znovu tiše proklínal svého bratra, který mu nechal jen lékořicové pralinky. Chodba byla temná a vlhká, s prasklými dlaždicemi pod nohama, které vrzaly při každém kroku. Jakub se snažil jít tiše, ale ozvěna jeho dechu se nesla jako šepot duchů. Vzduch byl těžký, plný prachu a tajemství, a on cítil, jak se mu ježí chlupy na zátylku.

Vešel do velkého pokoje, kde stál masivní stůl, na němž ležela spousta starých papírů. Co bylo ale zajímavé, všechny byly nahrnuté na levou stranu a pravá polovina byla úplně prázdná. Rozhodl se, že se mu nechce se tou hromadou dokumentů prohrabovat, tak raději vyšel z pokoje zpátky na chodbu.

Před ním se do útrob domu táhla dlouhá chodba. Po každé straně chodby byla trojice dveří. Jakub postupně zkoušel zmáčknout kliku všech dveří, ale všude bylo zamčeno. Až u úplně posledních dveří vlevo se na něj usmálo štěstí; po stisknutí kliky se dveře otevřely a on tak mohl vejít do dalšího pokoje. Srdce mu tlouklo jako zvon, když se dveře s tichým zaskřípěním otevřely.

Tam na nástěnce visela podivná tabulka. Chvilku ji zkoumal a poté si všiml zajímavého faktu. Kromě toho, že někdo zapomněl vyplnit první položku v druhé řádce a třetí položku ve druhém sloupci, tam bylo ještě něco. No jasně, vždyť je to šifra! Jakub se usmál, jeho oči se rozsvítily vzrušením.

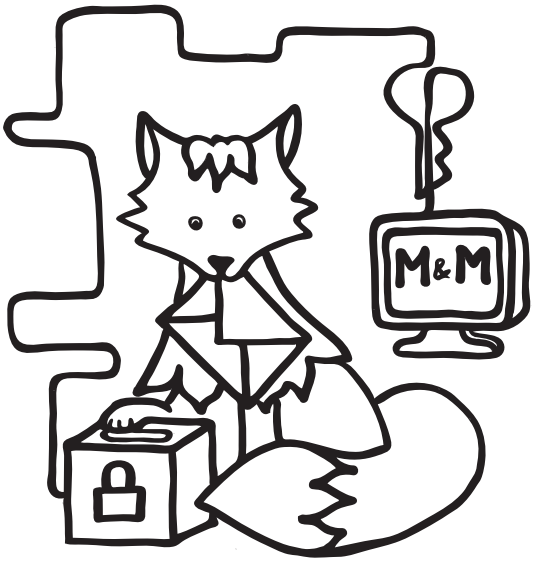
Bystrý Jakub šifru prolomil během chvilinky a zamířil do městské galerie. Přesně věděl, které výstavní síně ukrývají ono kýžené tajemství; jistě to byla ta v nejvyšším patře v pravém křídle budovy a ta ve druhém patře v levém křídle. Díky důkladnému prozkoumání těchto místností našel Jakub důležitou stopu, která mu zajistila bleskový kariérní postup. To všechno ho katapultovalo na vrchol jeho kariéry, stal se legendárním detektivem, který rozluštil největší záhadu města.

Řešení:
V této šifře je v každém odstavci zakódováno jedno písmeno pomocí Braillova písma. Vždy lze nějakou část odstavce zakreslit pomocí tabulky, která má dva sloupce a tři řádky a některá její políčka jsou podle příběhu vybarvená. Jako řešení vyjde POVOLÁNÍ.

Závěrečná metašifra
Šifra 1 + Šifra 2 + Šifra 3 + Šifra 4

Řešení:
Řešení předchozích šifer jsou: KLÁVESNICE + HROCH + INFORMATIKA + POVOLÁNÍ. Slova svým významem navádějí na tajenku PROGRAMÁTOR (hroch je maskotem Korespondenčního semináře z programování).

Olga a Pája; simova.pavla@gmail.com



Výsledky 2. deadlinu 3. čísla a 1. deadlinu 4. čísla

Poř.	Jméno	R.	$\sum_{-1}$	Témata 1	26	5	6	7	$\sum_0$	$\sum_1$
1.	Doc.[MM] J. Klementová	4	491,3		12,0	13,0	10,0	5,5	40,5	130,2
2.	Mgr.[MM] J. Thomitzek	1	99,2		6,0	8,0	10,0	6,5	30,5	99,2
3.	Doc.[MM] M. Jarvis	4	472,3							97,0
4.	Mgr.[MM] L. Koma	2	79,0			8,0	15,0	13,0	36,0	74,0
5.	Dr.[MM] S. Šimečková	4	101,2			8,0	10,0		18,0	58,7
6.	Mgr.[MM] M. Hrubá	2	52,3					3,0	3,0	52,3
7.	Dr.[MM] P. Barták	2	110,7		10,0	3,5		3,1	16,6	48,8
8.	Doc.[MM] M. Ambros	3	267,1		28,5	8,0			36,5	46,5
9.	Dr.[MM] B. Salajová	4	165,8		8,0	5,8	3,0	6,0	22,8	45,6
10.	Dr.[MM] P. Starý	4	155,6							41,3
11.	Mgr.[MM] F. Dvořák	3	91,1			7,0			7,0	36,0
12.	Doc.[MM] O. Nevěřil	4	299,7		8,0	8,0			16,0	35,3
13.	Mgr.[MM] J. Fišerová	3	58,7							32,0
14.	Bc.[MM] S. Bažantová	3	31,1		2,0			4,5	6,5	31,1
15.	Bc.[MM] E. Ježek	4	28,8							28,8
16.	Bc.[MM] T. Holásek	3	28,2							28,2
17.	Bc.[MM] R. Krzystek	3	28,1			8,0			8,0	28,1
18.	Mgr.[MM] Š. Swaczyna	1	65,0			6,3	2,0		8,3	27,8
19.	Bc.[MM] P. Fiala	4	27,5			8,0		3,0	11,0	27,5
20.	Bc.[MM] V. Kupilík	1	26,5		16,0	7,4		3,1	26,5	26,5
21.	Mgr.[MM] F. Nouza	4	91,6			4,0	8,5		12,5	23,5
22.	Dr.[MM] A. Gauchet	4	101,3		6,0	2,0			8,0	20,8
23.–25.	M. Stroff	4	19,8							19,8
	A. Mouchová	3	19,8							19,8
	V. Kubrycht	4	19,8							19,8
26.	M. Hořenek	3	19,7			4,4	4,8	10,5	19,7	19,7
27.	L. Mihola	1	19,5							19,5
28.	Q. Liao	3	19,0							19,0
29.	M. Vagner	3	15,5				5,0		5,0	15,5
30.	F. Gašparín	3	15,4			6,0	4,8	4,6	15,4	15,4
31.	Mgr.[MM] N. Jochová	3	62,2							15,1
32.–33.	Mgr.[MM] V. Kučera	4	93,0							15,0
	M. Pavlas	2	15,0		8,0			3,0	11,0	15,0
34.	M. Dvořák	3	14,9		9,0	5,9			14,9	14,9
35.	J. Štěchová	4	14,3		2,5	8,0			10,5	14,3
36.	J. Kaplický	4	18,6			7,9			7,9	13,9

Poř.	Jméno	R.	$\sum_{-1}$	Témata 1	26	5	6	7	$\sum_0$	$\sum_1$
37.	V. Holuša	3	13,8			4,4	4,8	4,6	13,8	13,8
38.	A. Ježková	2	13,0							13,0
39.	Mgr.[MM] K. Bouchalová	1	61,6							12,0
40.	O. Kočur	3	11,6			4,4	4,8	2,4	11,6	11,6
41.	Dr.[MM] J. Jedlička	4	131,7							11,0
42.	T. Zatloukal	3	10,1			6,9		3,2	10,1	10,1
43.	Doc.[MM] D. Kaňka	4	211,9							10,0
44.	Dr.[MM] K. Kučerová	1	136,5							8,5
45.–46.	M. Hošek	4	7,0							7,0
	Š. Hrdý	4	7,0							7,0
47.	Mgr.[MM] S. Ožanová	4	56,3		6,0				6,0	6,0
48.	R. Michálková	4	10,6			5,5			5,5	5,5
49.	K. Kučerová	1	3,0							3,0
50.–52.	L. Šemberová	Z9	1,2							1,2
	O. Plíšek	1	1,2							1,2
	M. Vojtěch	Z8	1,2							1,2
53.	J. Dingová	4	1,0							1,0
54.	J. Vospálek	3	0,6							0,6

Sloupeček $\sum_{-1}$ je součet všech bodů získaných v našem semináři, $\sum_0$ je součet bodů v těchto deadlinech a $\sum_1$ součet všech bodů v tomto ročníku. Tituly uvedené v předchozím textu slouží pouze pro účely M&M.

Kontakty:

M&M, OPMK, MFF UK
Ke Karlovu 3
121 16 Praha 2

E-mail: mam@matfyz.cz
Web: mam.matfyz.cz
FB: casopis.MaM

matfyz
M&M