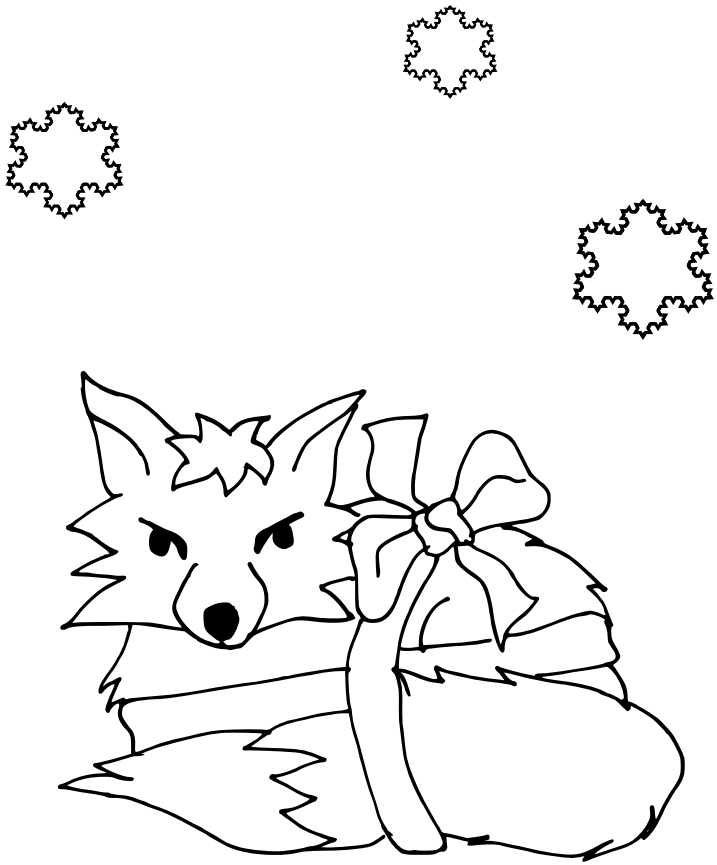
Studentský matematicko-fyzikální časopis

M&M

Ročník XVII / Číslo 3

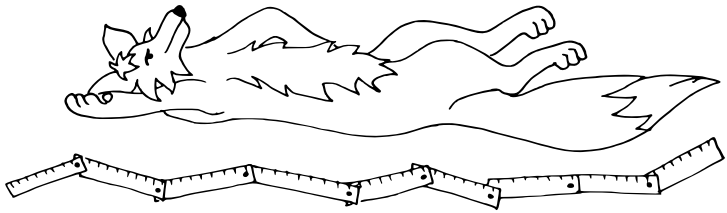


Milí kamarádi,

dostává se vám do rukou číslo, které uzavírá první polovinu tohoto ročníku. Naleznete zde novou sérii příkladů, jenž jsou tentokrát netradičně praktické. Ve fyzikální úloze od vás očekáváme naměření rychlosti padajícího listu a informatická úloha vyžaduje, abyste sedli k počítači a napsali program. Dále zde naleznete také příspěvky k tématům z prvního čísla. Chtěli bychom vás upozornit na **II. mapování**, které proběhne **15. prosince**. Více se o něm dozvíte na straně 6. Samozřejmostí je i další díl populárního seriálu o jazyku Python.

Přejeme vám, abyste si v pohodě užili Vánoční svátky. Na shledanou v novém roce 2011, do kterého vše nejlepší přejí

Organizátoři M&M



Zadání úloh

Termín odeslání třetí série: 17. 1. 2011

Úloha 3.1 – Tajuplný ostrov (4b)

Vlna, která se mi přelila přes obličej, mě probudila. V poledním slunci jsem se pomalu rozkoukával. Zjistil jsem, že jsem na pláži, kde kromě mě nic není. „Naše loď musela ztroskotat," blesklo mi hlavou. „Asi bych nejdřív měl prohledat pobřeží, abych zjistil, zda moře nevyplavilo i něco ze zásob, které byly na lodi," napadlo mě při vzpomínce na knihu o Robinsonovi.

A opravdu, po necelé půlhodině hledání jsem zahlédl nějaký předmět, který se leskl v září slunce. Ukázalo se, že se jedná o mou sbírku úloh z matematiky, která byla velice promočená. Knihu jsem opatrně otevřel. Na první pohled mě zaujal následující příklad.

Číslo 2^{2010^2} má 1216192 cifer a jeho první cifra je 1. Uměli byste určit, kolik z čísel 2^0, 2^1, …, 2^{2010^2-1} začíná cifrou 1? A co kolik jich začíná cifrou 4? Všechna čísla uvažujeme v desítkové soustavě. [1]

„Jídlo počká," řekl jsem si. Usadil jsem se na nedaleký kámen a zamyslel se nad tímto příkladem…

[1] Exponenty bývá zvykem závorkovat odshora, tedy 2^{2010^2} znamená $2^{\left(2010^2\right)}$.

Úloha 3.2 – List (4b)

Po příjemně strávené hodince nad příkladem jsem usoudil, že by bylo vhodné se konečně pustit do prohledávání okolí. Vešel jsem do nedalekého lesa, kde jsem narazil na starou studnu. „Jak může být hluboká?“ říkal jsem si. „To by se určitě dalo lehce změřit, kdybych měl po ruce nějaký kamínek.“ Ten jsem však po ruce neměl. Ovšem kolem bylo plno listí. „To by mohlo pomoct,“ řekl jsem si. Jen bych potřeboval znát, jakou rychlostí padá list...

Změřte dobu, za jakou spadne list z různých výšek na zem. Ne hned od místa „upuštění“, uvažujte jen ustálený pohyb. Ono se to ustálí rychle, uvidíte...

List aproximujte kruhovým lístkem papíru a zkuste tento čas spočítat. Zamyslete se nad tím, proč se experimentální výsledek od teoretického liší. Návod, jak zpracovávat fyzikální měření, naleznete v M&M ve 2. čísle XVI. ročníku.

Úloha 3.3 – Palisáda (4b)

„Když je tu cestička a studna, tak by tu mohli být i domorodci,“ napadlo mě a svitla mi jiskřička naděje, že nejsem na opuštěném ostrově. A opravdu to trvalo asi jen hodinu, než jsem zahlédl dvě osoby, které se o něčem dohadovaly a mířily přímo ke mně.
Když mě míjeli, změřil si mě menší z nich přísným pohledem a prohlásil: „Vypadáš celkem inteligentně, tak to bys nám mohl pomoct.“ A začal vysvětlovat...

Tento ostrov každoročně sužují nájezdy pirátů. Proto se místní domorodci rozhodli postavit na pobřeží palisádu, aby piráti nemohli přistát. Před stavbou však potřebují zjistit, jak je dlouhé pobřeží ostrova, aby si mohli připravit dostatek dřeva.

Mapování na ostrově nedosáhlo moc dobré úrovně, tak je k dispozici pouze čtverečkovaná mapa, kde 0 znamená vodu a 1 označuje suchou zem. Na okraji celé mapy je proužek široký minimálně jeden čtvereček, kde je voda. Úkol je spočítat počet hran, kde moře sousedí s ostrovem. Na ostrově se nachází několik jezer, z nichž samozřejmě piráti neútočí, a proto se do délky palisády nepočítají.

Vzhledem k velikosti ostrova napište program, který spočte délky palisády pro libovolnou mapu, kterou načte ze vstupního `mapa.in`[2] a uloží ji do výstupního souboru `mapa.out`. Toto načítání a ukládání je důležité pro testování správnosti vašeho řešení, neodbývejte ho!

[2] Příklad mapy, na které můžete testovat svůj program, najdete na adrese `http://mam.mff.cuni.cz/vstupy/mapa.in`

Úloha 3.4 – Nápis (2b)

Když jsem domorodcům řekl výsledné číslo, viděl jsem na jejich tvářích neskrývané nadšení. „Mohl bys nám ještě pomoct?“ začal jeden z nich nesměle. Přisvědčil jsem, a už mě někam vedli. Po chvíli se před námi vynořil chrám.

„Na jedné ze stěn je něco napsáno a my nevíme co. Navíc se toho část umazala. Mohl bys to pro nás rozluštit a říct nám, co tam dřív bylo napsané?“

Na stěně bylo napsáno toto:

$$(10\bullet\bullet\bullet 10\bullet\bullet)_2 \,\mathbf{mod}\, (18)_{10} = 0\,.$$

Poznáte, co bylo původně napsáno tam, kde je nyní napsáno •? Nalezněte všechna řešení.

„Karléé, večeřéé…“ ozvala se maminka. A já musel opustit svůj tajuplný ostrov a jít si dát nudle s mákem…

Řešení témat

Téma 1 – Rekurentní posloupnosti

K tématu o rekurentních posloupnostech přišly celkem čtyři příspěvky. Přispěli Doc.ᴹᴹ Alena Bušáková, Bc.ᴹᴹ Eva Gocníková, Bc.ᴹᴹ Jakub Kubečka a Bc.ᴹᴹ Vladimír Sedláček. Všichni autoři se zabývají téměř shodnými problémy, každý ale nabízí jiná řešení. Proto jsme se rozhodli tentokrát neotiskovat celý žádný z článků, ale nabízíme vám krátké shrnutí všech poznatků, které do redakce přišly. Jednalo se především o nacházení posloupností majících určité vlastnosti.

Posloupnost s členy mezi nulou a jedničkou

Velmi jednoduchý byl problém nalézt posloupnost, jejíž všechny členy se nachází mezi nulou a jedničkou. Pěkné řešení nabízí Bc.ᴹᴹ Eva Gocníková. Definuje posloupnost vztahem

$$A_{n+2} = \frac{A_{n+1} - A_n}{2}$$

s počátečními podmínkami $A_0 = 0$ a $A_1 = 1$, tedy jako aritmetický průměr předchozích dvou členů. Z vlastností aritmetického průměru (či mocninného průměru obecně) snadno dostaneme, že tato posloupnost našim požadavkům vyhovuje.

Ještě jednoduší řešení vymyslela Doc.ᴹᴹ Alena Bušáková. První takovou je posloupnost s počáteční podmínkou $A_0 = 0$ a předpisem $A_n = 2A_{n-1}$. Druhá má počáteční podmínku $A_0 = 1$ a vzorec $A_n = A_{n-1}/2$. Téměř identickou posloupností přispěl i Bc.ᴹᴹ Vladimír Sedláček. Obě posloupnosti zřejmě naše požadavky splňují. Bc.ᴹᴹ Jakub Kubečka nabízí řešení v podobě posloupnosti

$$T_{n+2} = \frac{1}{2T_{n+1}^{-1} + 2T_n^{-1}}$$

s počátečními podmínkami $T_0 = 1/2$ a $T_1 = 1/3$. I zde si můžeme snadno rozmyslet, že posloupnost vyhovuje.

Posloupnost s kladnými členy a konečným součtem

Takových posloupností existuje opravdu hodně. Nikdo zatím nepřinesl přesnou specifikaci, co musí taková posloupnost splňovat. Můžete se nad ní zkusit zamyslet. Vyhovující posloupnost (jak autoři správně píší) je například $A_0 = 1$ a $A_n = A_{n-1}/2$ z předchozího odstavce.

Posloupnost s každým třetím členem záporným a konečným součtem

I zde přišlo více řešení. Bc.ᴹᴹ Eva Gocníková vymyslela posloupnost $A_0 = 0$, $A_1 = 1$ a $A_{n+2} = -A_n - A_{n+1}$. Její členy jsou čísla $0, 1, -1, 0, 1, -1, 0, \ldots$. Periodičnost posloupnosti není těžké zdůvodnit. Z té už vyplývá konečný součet všech členů až do libovolného n. Zajímavou otázkou ale je, jaký součet dostaneme, pokud budeme sčítat až do nekonečna. Jaký součet byste přiřadili této posloupnosti? A proč?

Doc.ᴹᴹ Alena Bušáková nabízí posloupnost definovanou pomocí počátečních podmínek $A_0 = 1, A_1 = 1, A_2 = -1, A_3 = 1$ a vztahem

$$A_n = \frac{A_{n-1} + A_{n-3} + A_{n-4}}{2}$$

Konečnost jejího součtu ale přesvědčivě zdůvodnit nedokázala.

Nakonec Bc.ᴹᴹ Vladimír Sedláček vytvořil posloupnost $A_1 = 1, A_2 = 0$, $A_3 = -1$ a $A_{n+3} = A_n \cdot A_{n+1} \cdot A_{n+2} + A_n$. Samotná posloupnost je velmi podobná té, kterou nabízí jako řešení Bc.ᴹᴹ Eva Gocníková. Tedy opět můžeme snadno dokázat periodicitu i konečný součet do n–tého členu. A i zde se nabízí otázka, jaký bude součet v nekonečnu.

Odvození explicitního vzorce z rekurentního

Bc.ᴹᴹ Vladimír Sedláček se zabýval hlouběji posloupností $A_0 = 0, A_1 = 1$ a $A_{n+2} = 2(A_{n+1} + A_n)$ a odvodil explicitní vzorec pro její n–tý člen. Bohužel odvození příliš podrobně nepopsal a čtenáři, který nezná metodu, kterou využívá, by asi moc neřeklo. Proto se zde spokojíme s výsledkem, že

$$A_n = \frac{\left(1+\sqrt{3}\right)^n - \left(1-\sqrt{3}\right)^n}{2\sqrt{3}} = \sum_{k=0}^{n} \binom{n}{2k+1} \cdot 3^k \,,$$

odtud odvodil například, že každý třetí člen posloupnosti je dělitelný třemi.

Co zkoumat dále

Stále čekáme na nějaké výsledky ohledně posloupností $\{A_n\}_{n=0}^{\infty}$ a $\{B_n\}_{n=0}^{\infty}$ zadaných v prvním čísle. Zvládnete udělat nějaký odhad (nebo nejlépe přesný vzorec) pro jejich n–tý člen? Zkuste své odhady co nejlépe zdůvodnit. Dobré je

třeba určit maximální možnou chybu vašich odhadů. Otevřela se nám otázka, kdy má posloupnost konečný součet. Můžete se také pokusit navrhnout, jak byste definovali nekonečný součet řady obsahující kladné i záporné členy.

A můžeme přidat i nějaké nové otázky. Mějme periodickou posloupnost. Dokážete pro ni najít rekurentní předpis? Nebo zkuste najít nějakou nekonečnou posloupnost s nenulovými členy, která bude mít součet všech svých členů roven 42.

A nebojte se přijít s čímkoli dalším. Těšíme se na vaše příspěvky.

Kuba

Téma 2 – Mapování

Po zadání tohoto tématu došly výsledky měření od 12 účastníků, za které děkujeme. Zároveň bychom chtěli vyhlásit další kolo mapování. Tentokrát bychom rádi **15. prosince 2010** změřili srážky. Měření by mělo vypadat tak, že mezi sedmou a osmou hodinou ranní (pokud vycházíte do školy dřív, tak dřív) dáte na nějaké nezakryté místo nádobu. V osm hodin večer pak nádobu vezmete a určíte, kolik milimetrů napršelo, neboli vydělíte objem vody plochou otvoru, kterým mohla voda napršet. Pokud bude již sněžit, tak zkuste sníh roztopit a změřit, kolik milimetrů vody nasněžilo. Výsledky opět zadávejte na `http://mam.mff.cuni.cz/mapovani`. Tentokrát se můžete těšit, že vaše měření bude odměněno bodem.

Nyní k výsledkům prvního mapování. Dostali jsme od vás pouze naměřené hodnoty, které jsou uvedeny v tabulce t2.2.1. Nikdo se však nepokusil o nějaké další zpracování dat. Proto v druhé části tohoto článku naleznete návodný postup, jak je možné data dál zpracovávat.

Co s daty?

Fyzikální měření obvykle provádíme proto, abychom určili nějakou konstantu, či ověřili nějakou závislost dvou veličin. Občas jsme ale v situaci, kdy přesně neznáme teoretickou závislost, například u materiálové fyziky, ale také u meteorologie a jiných oborů fyziky. Co v takových případech? Můžeme hádat. Zvolíme si nějaký tvar funkce, třeba polynom druhého řádu

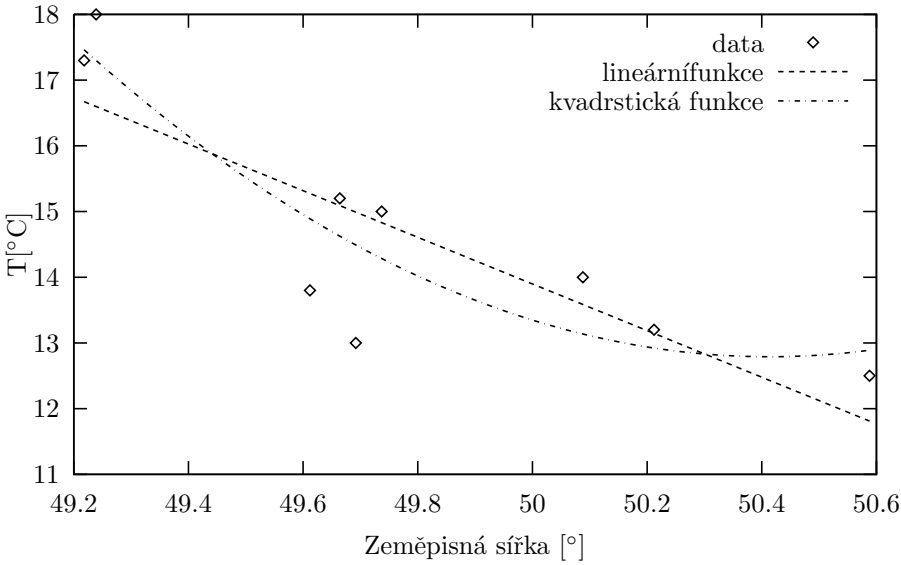
$$y = ax^2 + bx + c\,,$$

a ten zkusíme co nejlépe napasovat na naměřená data. Napasováním chápeme volbu parametrů a, b a c. Tento proces nazýváme fitováním. Nafitovat koeficienty je velice náročné, proto se obvykle používá počítač, který řeší tuto úlohu numericky.

Jak asi tušíte, tak se při řešení tohoto tématka fitování nevyhneme. Ale co fitovat? Nejdříve musíme odhadnout nějaké veličiny, od kterým očekáváme, že by na sobě mohly záviset. Řekněme, že budeme zkoumat závislost teploty na zeměpisné šířce. Víme, že na rovníku je mnohem větší horko, než na pólech. Dá se tedy očekávat, že bychom mohli nějakou závislost zjistit.

Autor	Místo měření	GPS souřadnice místa měření	Teplota [°C]
Vojtěch Kletečka	Havlíčkův Brod	49°36,7 N, 15°34,5 E	13,8
Jakub Kubečka	Dvory	50°12,7 N, 14°59,7 E	13,2
Jan Bok	Litoměřice	50°32,0 N, 14°8,0 E	16
Martina Bekrová	Trutnov	50°35,3 N, 15°53,1 E	12,5
Filip Štědronský	Plzeň, Lhota	49°41,5 N, 13°19,3 E	13
Matěj Kocián	Zlín, Letná	49°13,1 N, 17°38,6 E	17,3
Míša Kochmannová	Štáhlavy	49°39,8 N, 13°30,1 E	15,2
Jakub Štoček	Flekke, Norsko	61°20,0 N, 5°20,2 E	8,3
Petr Pecha	Valašské Klobouky	49°8,7 N, 18°0,9 E	11
Jan Škoda	Plzeň, Slovany	49°44,2 N, 13°23,5 E	15
Jan Česal	Zlín, Jižní svahy	49°14,4 N, 17°40,4 E	18
Kristýna Kohoutová	Žamberk	50°5,3 N, 16°27,8 E	14

Tabulka t2.2.1: Data naměřená při měření teploty 22. září 2010.



Obr. t2.2.1 – Naměřená data proložená lineární a kvadratickou funkcí.

Nejdříve se však musíme zamyslet nad daty. Velice si vážíme dat, které jsme dostali až z dalekého Norska, ale pro náš případ je nebudeme moci použít. Norsko je příliš vzdálené. Máme tedy mráček bodů kolem 50° s. š. a pak jeden

vzdálený. Tento bod by při fitování měl u některých funkcí příliš velikou váhu. Získali bychom sice funkci, kterou bychom mohli odhadnout celé Polsko, ale chyba by byla obrovská. Proto se nyní omezíme pouze na Českou Republiku.

Další data, která budeme muset k naší lítosti vypustit je měření z Valašských Klobouk, které jsou jedním z nejjižnějších míst, kde měření probíhalo, ale zároveň nejstudenějším. To neodpovídá teorii. Buď tedy zavrhneme naši teorii, nebo se spokojíme s tím, že naše teorie není stoprocentní. Případně můžeme začít polemizovat nad přesností měření.

Posledním měřením, kterého se vzdáme, jsou data ze Šťáhlav. Místo, které patří k nejsevernějším má velmi vysokou teplotu. Důvody, proč měření nepoužít, jsou stejné jako v předchozím případě. Data, která jsme tentokrát nepoužili se nám však můžou hodit. Pokud bychom zkoumali závislost teploty na nadmořské výšce, tak se nám pravděpodobně budou hodit.

Moje první fitování

K fitování naměřených dat můžete použít program gnuplot, jehož podrobný popis ovládání je v našem časopise ve 4. čísle XVI. ročníku. Pokud byste měli s programem problémy, můžete si napsat o radu na e–mail `radim@matfyz.cz`.

Nejdříve nainstalujeme program a naměřená data uložíme do souboru. Data jsou uložena tak, že v prvním sloupci je hodnota, podle které jsme měřili, což je v našem případě zeměpisná šířka. A v druhém sloupci je uvedena naměřená hodnota, v našem případě teplota. Sloupce jsou od sebe odděleny tabulátorem, místo desetinné čárky se používá tečka.

Pak pomocí příkazu

```
plot "soubor"
```

zobrazíme naměřená data a pokusíme se uhodnout tvar funkce. V našem případě by se mělo jednat o klesající funkci, čemuž odpovídají data. Nejdříve zkusíme to nejjednodušší, což je lineární funkce. Nadefinujeme si ji pomocí příkazu

```
f(x)=a*x+b
```

a pak už nám nic nebrání přejít k fitování. To se programu řekne příkazem[3]

```
fit f(x) "soubor" via a,b
```

Před očima by vám pak měla proběhnout hromada čísel. To program počítá koeficienty. Nakonec se zastaví a ukáže vám výsledek v podobě

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = -3.5487          +/- 0.9073       (25.57%)
b               = 191.333          +/- 45.17        (23.61%)
```

[3] Pokud má funkce více parametrů, tak je všechny musíme napsat za `via`.

Dozvídáme se tak nejen hodnotu, ale i chybu, která je pro nás velice důležitá, neboť pomocí ní můžeme porovnávat, která funkce je lepší a které parametry přesnější. Nyní nám už jen zbývá vykreslit výsledky do grafu pomocí příkazu

```
plot "soubor", f(x)
```

Takto můžeme zkoušet různé funkce a různé parametry. Občas se ale stane, že si program nemůže s naměřenými daty poradit. V takovém případě je potřeba jej trochu popostrčit tím, že mu poradíte, od jakých hodnot daných parametrů by měl zkusit fitovat. Odhadování počátečních hodnot je však celkem složité.

Závislostí teploty na zeměpisné šířce jsme zkusili fitovat lineární a kvadratickou funkcí. Výsledky pro funkci[4]

```
f(x) = a*x**2+b*x+c
```

nám vyšly překvapivě hůře

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 3.27096          +/- 1.725        (52.74%)
b               = -329.799         +/- 172.1        (52.17%)
c               = 8325.89          +/- 4290         (51.53%)
```

Co dál?

Hlavně nezapomenout na měření srážek na začátku prosince.

Při zpracování můžete navázat na výsledky, které byly popsány v předchozím textu. Můžete vyzkoušet další funkce a zkusit zhodnotit, zda výsledky odpovídají očekávaní. Jako funkce pro fitování můžete zkusit použít polynomy vyšších řádů, exponenciálu, goniometrické funkce a cokoliv dalšího vás napadne.

Můžete také zkusit zkoumat vlastní závislost. Pomocí zeměpisných souřadnic můžete v atlasu či v mapě zjistit v jaké výšce nad mořem měření probíhalo a pak zkusit zkoumat tuto závislost. A nebo můžete přijít s něčím novým, co se k tomuto tématku hodí a zatím nebylo zmíněno.

(R)adim

[4] V gnuplotu se mocnina značí dvěma hvězdičkami, tedy x^2 se napíše jako `x**2`.

Řešení úloh

Úloha 1.1 – Pokažená hra (4b)

Zadání:

Rád vzpomínám na své klukovské roky, kdy jsme si u nás za domem hrávali na malém čtvercovém plácku. Nejvíce mi v paměti utkvělo jedno pozdní sobotní odpoledne. Sešli jsme se tehdy v plném počtu šestnácti kluků. Není také divu, když jsme se odplatou měli utkat s Podkováky za to, že nám zničili jednu z našich her.

Jako první se k něčemu odvážil nejstarší Mirek, který stál spolu s Jiříkem uprostřed. Udělal dva kroky dopředu a z plna hrdla se rozkřičel na Podkováky, ať táhnou domů. Podkováci si to samozřejmě nenechali líbit a z řad Podkováků vyrazil Lojza, jenž taktéž provedl dva kroky vpřed. Postavil se tak, že stál šikmo od Mirka a hned mu to začal oplácet. A tak se do klukovské šarvátky začali postupně přidávat další a další - náš Jiřík, který stál naproti Lojzovi, popošel o dva kroky, takže měl Mirka po své pravé ruce. Ondra od Podokováků se posunul do takové pozice, aby stál v jedné přímce s Lojzou a Mirkem ...

A o co tehdy vlastně šlo? Jak se dá u klukovských svárů očekávat, byl důvod zcela malicherný.

Na náš plácek jsme tehdy rozmístili papírky s čísly $1, 2, 3, \ldots 64$ *do mřížky* 8×8 *tím způsobem, že v prvním řádku byla postupně čísla* 1 *až* 8*, v druhém* 9 *až* 16 *atd. Papírky na plácku zůstaly přes noc a Podkováci nám před některá čísla napsali mínus, a to tak zvláštně, že v každém řádku a v každém sloupečku byla právě čtyři záporná čísla. A proč to udělali? Prý chtěli vědět, jaký potom bude celkový součet všech takto upravených čísel naší hry. Kolik jim mohlo vyjít?*

Řešení:

Všimneme si, že na políčku v r–tém řádku shora a s–tém sloupci zleva bylo původně napsané číslo $8 \cdot (r-1) + s$ (řádky a sloupečky číslujeme od jedné do osmi). Každé takové číslo si můžeme představit jako součet „řádkové“ části (to je $8 \cdot (r-1)$) a „sloupcové“ části (to je samotné s).

Podívejme se nejdřív na řádkové části čísel v tabulce. Jelikož v každém řádku jsou čtyři čísla s plusem a čtyři s minusem, jejich shodnou řádkovou část budeme při sčítání řádku čtyřikrát přičítat a čtyřikrát odečítat. Není asi překvapením, že nám vyjde nula (součet celého řádku ovšem nula nejspíš nevyjde – to by se musely odečíst i sloupcové části, na což se nelze spolehnout). Součet všech řádkových částí v tabulce je roven osmkrát sečtené nule, tedy nule.

Když se teď podobně zaměříme na součty sloupcových částí v rámci jednoho sloupečku, zjistíme, že jelikož v každém sloupci jsou ze zadání čtyři čísla s minusem, je součet sloupcových částí v každém sloupci roven nule. Sloupcové části v celé tabulce se tedy také sečtou na nulu.

Hledaný součet čísel v tabulce je roven součtu řádkových a sloupcových částí. Bryskně sečteme $0 + 0 = 0$ a jsme hotovi.

Pepa

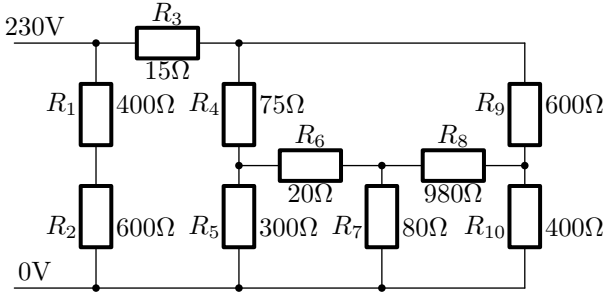
Úloha 1.2 – Osvětlení (4b)

Zadání:

Situace na plácku se postupně vyostřovala. Do souboje, který zatím probíhal jen formou nadávek, se přidal i náš Karel. Nejdříve nesměle přešlapoval na místě, ale pak udělal úkrok doprava a dvěma mohutnými skoky se objevil nedaleko Jiříka. Když nadávky přestávaly stačit, Lojza nevydržel a vrhnul se na Mirka. Mezi rvoucí se kluky skočil i Karel, ale bylo již příliš pozdě - Mirek odcházel s brekem a rozbitým nosem, což Karel Lojzovi velmi rychle a tvrdě oplatil. Nato vyrazil Tomáš od Podkováků a podobně jako prve Karel, ukročil stranou a pak učinil dva skoky. Tomáš i Karel se nyní dívali takřka z očí do očí připraveni jeden na druhého ve vteřině skočit.

Večer mírně pokročil, na plácek se snášela tma a nad hlavami kluků se začaly rozsvěcet žárovky.

Nad pláckem byla natažena síť nejrůznějších světýlek, která měla různý elektrický odpor. Jednou jsme je pro zábavu proměřili a zakreslili do následujícího schématu[5]*. Když jsem se po letech na toto schéma opět díval, všiml jsem si jedné velice zvláštně zapojené žárovky. Pokud by se totiž vymontovala, nic by se v okolním zapojení nezměnilo – tedy nezměnil by se proud protékající ostatními žárovkami. Poznáte, o jakou žárovku šlo?*



Řešení:

Nejprve si uvědomíme, že pokud se po odebrání žárovky nemá změnit proud procházející ostatními žárovkami, už v původním zapojení jí nesměl protékat žádný proud. Kdyby totiž ano, po odebrání by se změnilo napětí na větvi se žárovkou, a to by nutně vedlo ke změně proudu v ostatních žárovkách. Jak mnoho z vás správně usoudilo, na žárovky R_1, R_2, R_3 můžeme zapomenout, protože první dvě z nich jsou v paralelním zapojení se zbytkem obvodu a pokud zde změníme odpor v jedné větvi, nutně změníme i proud tekoucí ve druhé. Po odpojení R_3 by zase zbytkem netekl proud vůbec. Žárovky R_4 nebo R_9 to také být nemohou, protože oba jejich dolní konce mají určitě nižší potenciál než jejich společný horní konec. To proto, že po kterékoliv z cest z horního konce do některého z dolních musíme přes odpory, které vždy potenciál sníží – po všech možných cestách k dolní hraně jsou tyto uzly blíž k dolní hraně. Stejný argument platí pro žárovky R_5, R_7 , R_{10}.

Nyní můžeme k úloze přistoupit tak, že každou zbylou žárovku zkusíme postupně odebrat a spočítáme, jaké je napětí mezi jejími dvěma původními

[5] Ve schématu jsou žárovky znázorněny značkami pro rezistory a označeny jako $R_1, \ldots, R_{10}$.

konci – pokud je nulové, pak jsme vyhráli. Ať už odebereme R_6 nebo R_8, výsledný obvod už bude, na rozdíl od původního, jenom sadou paralelních a sériových zapojení.

Začněme případem, kdy odebereme R_8. Označme si napětí mezi pravým krajem R_3 a spodní hranou jako U a zaveďme potenciál φ tak, že spodní hrana má potenciál nulový a pravý kraj R_3 má potenciál roven U. Pak z Ohmova zákona a ze vzorců pro sériové a paralelní zapojení snadno zjistíme, že

$$\varphi_{pravý\,kraj} = \frac{U R_{10}}{R_9 + R_{10}} = \frac{3}{5} U\,,$$
$$\varphi_{levý\,kraj} = \frac{U\,\dfrac{(R_7+R_6)R_5}{R_7+R_6+R_5}}{R_4 + \dfrac{(R_7+R_6)R_5}{R_7+R_6+R_5}} \cdot \frac{R_7}{R_7+R_6} = \frac{3}{5}\,,$$

a tedy že potenciál na obou koncích R_8 je při odebrání R_8 stejný. Jde tedy o námi hledanou žárovku. Zkoumaná část obvodu se zjevně (až na hodnoty odporů) nezmění, pokud přeznačíme zároveň $R_9 \leftrightarrow R_4$, $R_5 \leftrightarrow R_{10}$ a $R_6 \leftrightarrow R_8$, takže při odebrání R_6 dostaneme

$$\varphi_{levý\,kraj} = \frac{U R_5}{R_4 + R_5} = \frac{1}{5} U\,,$$
$$\varphi_{pravý\,kraj} = \frac{U\,\dfrac{(R_7+R_8)R_{10}}{R_7+R_8+R_{10}}}{R_4 + \dfrac{(R_7+R_8)R_{10}}{R_7+R_8+R_{10}}} \cdot \frac{R_7}{R_7+R_8} = \frac{317}{325}\,.$$

Při odebrání R_6 se tedy protékající proudy změní. Úlohu šlo samozřejmě řešit i pomocí Kirchhoffových zákonů, což je myšlenkově trochu jednodušší postup, ale řešení vzniklých soustav rovnic může dát v ruce dost práce.

Irigi

Úloha 1.3 – Podkovácká čtvrť (3b)

Zadání:

Tehdy jsem se poprvé do hry zapojil i já. Udělal jsem dva kroky a postavil se za Jiříka. Natáhl jsem svůj prak a mířil přes hlavu Karla. Útok od Podkováků však přišel odjinud. Na Jiříka se vyřítila Julča, jediná slečna mezi Podkováky. Na holku se uměla rvát překvapivě dobře. A tak Jiřík s roztrženou košilí a boulí na nose utíkal domů. To mi připomnělo, jak vlastně rivalita mezi námi a Podkováky začala.

Způsobil to Jiřík, který se jednou zatoulal do jejich městské čtvrti. Podkovácká čtvrť byla tvořena čtvercovou sítí náměstíček $n \times n$, která byla spojena pravoúhlou sítí ulic (vnitřní náměstíčko bylo ulicemi propojeno právě se čtyřmi vedlejšími, krajní s okolními třemi a rohové se dvěma náměstíčky). Kolem čtvrti byly vysoké ploty továren a hlavní silnice, kudy se nedalo rychle utéct.

Vidím to jako dnes. Jednotliví Podkováci byli tehdy velmi dobře rozmístěni. Jiřík ještě stihl doběhnout na náměstíčko, které mu přišlo bezpečné. Pozorně se zaposlouchal aby uhodl, kde se jeho protivníci nacházejí. Pak hon začal. Jiřík se rozeběhl na jedno z vedlejších náměstíček.

Jen co tam doběhl, všimlo si ho nějaké dítě a běželo všem větším klukům říci, kam se Jiřík přesunul. Každý z Podkováků buď přeběhl na některé ze sousedních náměstíček, nebo vyčkával na místě. Když dusot Podkováků ustal, Jiřík se opět zaposlouchal a pak běžel zas na jedno ze sousedních náměstíček nebo vyčkával. A tak to probíhalo stále dokola.

Jiříka tehdy pořádně prohnali. Pokud by se Jiřík ve kteroukoliv chvíli ocitl na jednom z náměstíček spolu s některým z Podkováků, dostal by výprask. Podkováci ale naštěstí nejsou zas tak vytrvalí běžci a pokud by Jiřík Podkovákům unikal dostatečně dlouho[6]*, přestalo by je to bavit.*

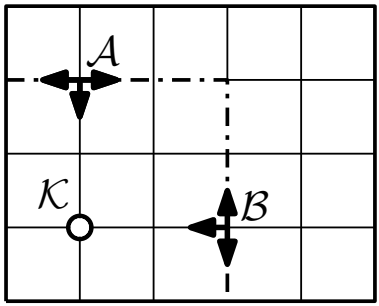
Zjistěte, kolik Podkováků je potřeba k polapení Jiříka a kolik ještě nestačí. Oba počty podpořte nějakým důkazem, například strategií k Podkováků, která Jiříka vždy chytí, nebo Jiříkovou strategií, se kterou bude l Podkovákům unikat libovolně dlouho. Strategie musí zahrnovat počáteční rozestavení i vhodný tah pro každou situaci.

Část bodů dostanete i za neoptimální počty, ale věžte, že Podkováků stačí mnohem méně než n. Je lépe dokázat slabší výsledek, než uhodnout optimum.

Řešení:

Jak spousta z vás správně napsala, chudák Karlík nemá šanci. Jednomu Podkovákovi sice uteče už ve městě 2×2, ale dva Podkováci chytí Karlíka v libovolně velkém městě. Jednomu Podkovákovi Karlík uteče tak, že vždy, když je Podkovák na vedlejším náměstí, může se Karlík přesunout k protějšímu rohu jednoho bloku domů, než co stojí Podkovák.

Dva podkováci, Alfonz a Bedřich, začnou uprostřed města a rozdělí si role – Podkovák Alfonz se bude vždy snažit být ve stejném sloupci jako Karlík a Podkovák Bedřich ve stejném řádku jako Karlík. Na začátku se budou oba jen chvíli poflakovat – Alfonz vodorovně ke Karlíkovu sloupci, Bedřich horizontálně směrem ke Karlíkovu řádku. Jakmile se Podkováci dostanou každý na Karlíkův řádek či sloupec, už nepustí Karlíka z dohledu – vždy, když se pohne vodorovně, pohne se tak i Alfonz, když horizontálně, tak Bedřich. Takto udrží Karlíka v obdélníku ohraničeném dvěma kraji čtvrti, Alfonzovým řádkem a Bedřichovým sloupcem.



Obr. u1.3.1

Pokud se ale Karlík pohne horizontálně, Bedřich se může přiblížit k protější straně obdélníku a tak ho zmenšit. Totéž může Alfonz, pokud se Karlík pohne vertikálně. Takto postupně zmenšují obdélník až Karlíka chytí. Pokud je délka

[6] Podkováci určitě neproběhnou více než 2^n ulic.

strany čtvrti n, bude jim zhruba $n/2$ kol trvat, než dostanou Karlíka pod dohled. To bude v situaci, kdy jsou oba Podkováci pořád na osách čtverce, takže obdélník s Karlíkem má strany $n/2$ a $n/2$, Podkováci ho tedy chytí nejpozději za dalších n kol, celkem za $1{,}5n$ kol.

Tomáš

Úloha 1.4 – Šachová úloha (2b)

Zadání:

Čím dál více se šeřilo a souboj s Podkováky začínal být dramatičtější. Potom, co Julča poslala domů Jiříka, se do středu dění vyřítil Pepa. Podobně jako Karel, nejdříve ukročil stranou, a pak dvěma kroky přeskočil před sebou stojícího Frantu. Postavil se tak, že mu stačil jeden skok a Julča by se dostala do jeho spárů. Při pohledu na vysokého Pepu se Julča raději vrátila na své místo, čekajíc, co se bude dít. To ocenila naše Lucka, jež udělala krok k Pepovi a poplácala jej po zádech.

Z Pepy měli Podkováci respekt. Dokonce takový respekt, že Tomáš raději vyklidil pole a šel se postavit před Julču, aby ji mohl chránit.

Čím si Pepa zasloužil takový respekt u Podkováků? Jednou se mu totiž podařilo Podkováky přechytračit.

Pepa se s Podkováky vsadil, že jim zadá šachovou úlohu, kterou nevyřeší. Zeptal se jich, kolik nejvíce dam můžou postavit na šachovnici tak, aby právě dvě políčka zůstala dámami neohrožena. Dámy se navzájem ohrožovat můžou. Podkováci na to tehdy vůbec nepřišli. A co vy, víte jak na to?

Řešení:

Jak si většina z vás správně povšimla, můžeme náš problém převést na duální úlohu. Jak umístit dvě dámy na šachovnici tak, aby ohrožovaly co nejméně polí?

Pro jednu dámu je řešení snadné. Umístíme ji do rohu, čímž způsobíme, že vůbec nebude ohrožovat jednu z úhlopříček – jeden celý sloupec a jeden celý řádek bude ohrožovat vždy, ať ji postavíme kamkoliv.

Pro dvě dámy je situace o trochu složitější. Musíme se snažit, aby sjednocení polí, které ohrožují, bylo minimální. Proto se jeví jako dobrý nápad položit je do společného sloupce, nebo řádku. Ten, jak jsme si řekli dříve, musí ohrožovat celý, donutíme je tedy alespoň, ať ho ohrožují obě zároveň.

Tím jsme se dostali do situace, kdy v řádcích a sloupcích ohrožujeme přesně 22 políček (do stejného řádku i sloupce bohužel dámy umístit nemůžeme). Zbývají nám ještě úhlopříčky. Budeme chtít, aby počet políček ve sjednocení úhlopříček byla co nejmenší.

Pokud umístíme dámu na okraj šachovnice, ohrožuje na úhlopříčkách vždy sedm políček (nepočítaje v to políčko, na němž dáma stojí). Pokud ji umístíme dále od okraje, ohrožuje jich vždy více (jedno políčko od okraje jich ohrožuje devět, dvě políčka od okraje jedenáct, tři políčka od okraje třináct).

Ať umístíme dámy jakkoliv, jejich úhlopříčky se vzájemně protnou nejvýše ve dvou bodech. Můžeme tedy „ušetřit“ dvě pole od ohrožování. Abychom vytvořili dva průsečíky, musela by být jedna z dam alespoň dvě pole od okraje (mezi ní a druhou dámou musí být lichý počet polí), což ale přidává čtyři

ohrožená políčka oproti situaci, kdy stojí obě dámy na okraji. Z toho vyplývá, že nejlepší bude postavit obě dámy na okraj. A to na stejný okraj, protože je chceme ve stejném řádku, nebo sloupci a dát je naproti sobě na opačný okraj šachovnice nám nijak nepomůže (dvou průsečíků úhlopříček tím určitě nedocílíme). Proto umístíme dámy na jedno políčko od sebe na nějakou hranu (např. na políčka A1 a A3). Tím docílíme toho, že se jejich úhlopříčky protnou v jednom bodě a dohromady budou obě dámy ohrožovat na úhlopříčkách 13 polí (mimo jich samých).

Kromě toho, úhlopříčky se vždy protnou s ohroženým řádkem nebo sloupcem od druhé dámy a to v jednom políčku od každé, pokud jsou dámy na kraji, nebo ve dvou od každé, pokud alespoň jedna z nich na kraji není.

Celkem budou obě dámy ohrožovat $22+13-2=33$ políček. Protože zadání bylo kolik můžeme umístit dam na šachovnici, aby právě dvě políčka zůstala neohrožena, musíme úlohu převrátit zase zpátky a na neohrožená pole umístit dámy. Těch bude celkem $64-33=31$.

Honza

Úloha 1.5 – A jak to bylo dál? (2b)

Zadání:
Jak si myslíte, že příběh skončil? Zkuste nám poslat své dokončení příběhu.

Řešení:
V celém příběhu je schována jedna šachová partie. Čtvercový plácek představuje šachovnici, kluci a holky potom jednotlivé figurky. Za nás hraje jako pěšec Mirek (`e2`)[7], Jiřík (`d2`) a Franta (`f2`), jako jezdec Karel (`b1`) a Pepa (`g1`). Na mě připadla úloha střelce (`f1`). Dámu nám dělala Lucka (`d1`). Za pěšce Podkováků hrál Lojza (`d7`) a Ondra (`c7`), jezdce představoval Tomáš (`g8`) a dámu Julča (`d8`).

Zápis popsané partie vypadá následovně:

```
1. e4 d5
2. d4 c6
3. Je3 dxe4
4. Jxe4 Jf6
5. Sd3 Dxd4
6. Jf3 Dd8
7. De2 Jfd7
```

Nyní jsme na tahu my, bílí. Cílem hry je jistě porazit Podkováky, tedy dát černému mat. Nejjednodušší způsob, jak toho dosáhnout je tah:

8. Jd6# 1-0

Jak mohlo například takové dokončení příběhu vypadat si můžete přečíst v řešení Štěpána Šimsy.

[7] Čísla uvedená v závorkách za jednotlivými jmény značí výchozí políčko figurky.

Karel se nenechal Tomášem odradit a dlouhým skokem přiskočil přímo před něj! V tu chvíli začal celou situaci organizovat nejdůležitější z Podkováků – Radim. Měl Julču po pravém boku a po Karlově skoku se začal bát o svůj nos. Byl mezi Podkováky velice vážený, nikdo nechtěl dovolit, aby se mu něco stalo. Radim usilovně přemýšlel, co by mohli podniknout. Kdyby dal Honza, který stál před ním, Karlovi pořádnou ránu, Karel by byl odrovnaný ale zase by po Radimovi nejspíš vyskočila Lucka. Tahle myšlenka se mu vůbec nelíbila. Navíc ať se rozhlížel kam chtěl, neměl kam utéct. Vypadalo to, že je v pasti. Začalo se mu dělat mdlo a omdlel. Tím skončila hra i všechna legrace. Všichni se šli kouknout, jestli je Radim v pořádku. Ten se za chvíli probral a potřásl rukou mě i všem mým kamarádům. Potom jsme se odebrali domů a já jsem ještě celý den nemohl potlačit pocit vítězství.

Terka

Úloha 1.6 – Seriál o pythonu I. díl (6b)

Zadání:

I (1b): *Popište, co dělá program a jakou hodnotu bude mít* `v`*. Nepoužívejte při tom Python (vyzkoušet, co dělají použité funkce, samozřejmě můžete).*

```
s="(10!)=3 628 800"; a=s[s.index(' ')-1]; b=s[s.index(a)+2:]
    a=int(a); b=b[b.index(" ")+1:]; b=int(b); v=b/2**a
```

II (1b): *Najděte nejmenší nezáporné hodnoty* `a` *a* `b` *typu* `int`*, aby platilo:*

```
max(str(a), str(b)) != str(max(a,b))
```

III (2b): *Napište funkci* `roztret(x)`*, která seznam či řetězec rozdělí co nejlépe na třetiny a vrátí trojici (seznam) těchto třetin. Třetiny se nesmí překrývat a dohromady musí dát celý vstup. Jejich délky musí být nerostoucí.*

IV(2b): *Napište funkci* `cifra(x,k)`*, která vrátí* `k`*-tou cifru zleva desítkového zápisu* `x`*. Předpokládejte celé* $\texttt{k} \geq 1$*,* `x` *může být celé i reálné (i záporné). Vyhněte se převodu na řetězec.*

Řešení:

I (1b): Tato úloha měla jen otestovat vaši schopnost indexovat řetězce od 0. Pár řešitelů se (poctivě) chytilo u posledního příkazu `v=800/2**3`, kde má mocnění přednost před dělením, a proto je výsledek 100.

III (2b): Pro všechna jednociferná čísla je porovnávání číselné hodnoty a znaků totéž, jedna hodnota proto musí být alespoň dvouciferná. Už pro `a=10` a `b=2` dopadne ale porovnání různě, protože `"10"<"2"` (porovnává se po znacích zleva).

III (2b): Je potřeba dát si pozor na to, že pouhé `l=len(x); return (x[:l/3], x[l/3:2*l/3], x[2*l/3:])` způsobí, že při l nedělitelném třemi bude nejdelší poslední část. Řešení je například:

```
def roztret(x):
  l = len(x)
  l1 = l - 2*l/3; l2 = l - l/3
  return (x[:l1], x[l1:l2], x[l2:])
```

Ještě chytřejší je indexovat rovnou od konce:

```
(x[:-(2*l/3)], x[-(2*l/3):-(l/3)], x[-(l/3):]).
```

Závorky po všech `-` jsou potřeba, protože dělení záporných celých čísel se chová konzistentně vůči posunu (`5/3==(5-6)/3 + 6/3`), ale ne negaci.

IV(2b): Důvod, proč jsme zakázali převod na řetězec (či pole cifer), je ten, že zjistit 2. cifru $1{,}234 \cdot 10^{300}$ by bylo dost neefektivní. Zjistíme raději počet cifer před desetinou čárkou a pak číslo posuneme tak, aby byla požadovaná cifra hned před desetinnou čárkou. Pak stačí spočíst tuto cifru operací `int(x%10)`. Je podstatné, že `int()` zaokrouhluje vždy směrem k nule. I proto je před začátkem výpočtu dobré zbavit se případného záporného znaménka.

```
def cifra(x, k):
  x = abs(x)
  cifer = int(math.floor(math.log10(x)))
  x = x / 10**(cifer+1-k)
  return int(x%10)
```

Při počítání cifer čísla `x` nestačí použít jen `int(math.log10(x))` právě proto, že `int()` zaokrouhluje směrem k nule. Potom by funkce pro $x < 1$ byla o jednu cifru mimo. Funkce bohužel funguje jen do přesnosti reálných čísel v Pythonu, což je asi 15 platných míst, ale to je omezení dané platformou.

Ještě poznamenejme, že funkce `cifra` funguje i pro celá čísla. Pro ta velká není sice efektivnější, než převod na řetězec, ale to už z principu velkých celých čísel, která jsou v podstatě pole cifer. Dělení při posouvání `x` je potřeba právě u celých čísel, aby funkce korektně spočetla i `cifra(10**4200 + 42, 4200)`. Pokud bychom násobili, `10**(4200-1-4200)` by vyšlo 0,1, `x` by po násobení bylo reálné a jeho 4200-tá cifra by se ztratila v nepřesnosti reálných čísel.

Tomáš

Seriál o Pythonu (III. díl)

Python má ještě hodně zajímavých vlastností, o kterých by stálo za to napsat, ale to až snad někdy jindy (či na soustředění). Po tom, co povíme něco málo o ošetření chyb v Pythonu a tvorbě vlastních objektů, vás naučíme načítat data z internetu a základy regulárních výrazů (které se k tomu velmi hodí).

Pokud byste měli se zprovozněním Pythonu stále problém, rádi vám zkusíme poradit, než se dostaneme k hraní si s konkrétními knihovnami. Děkujeme za spoustu (často originálních) řešení a komentářů!

Chyby a výjimky

V každém větším programu může dojít k chybě – otevíraný soubor neexistuje, operace není povolena, uživatelova data neumí funkce zpracovat, ... V programovacích jazycích jako C a Pascal je potřeba zkontrolovat výsledek každé volané knihovní funkce a ošetřit případnou chybu. To je ale pracné, zvlášť, když po sobě vaše funkce musí uklidit (zavřít soubory, ...) a chce o chybě informovat funkci, která ji zavolala (propagovat chybu výše v programu).

V Pythonu (a dalších jazycích jako C++, C# nebo Java) je při chybě vyvolána tzv. *výjimka* (anglicky *exception*). Výjimka vznikne buď během nekorektní operace (dělení nulou, indexování mimo pole, ...) nebo příkazem `raise`. Výjimka způsobí, že se aktuální blok okamžitě ukončí a vyskočí se do nadřazeného bloku (či volající funkce), kde výjimka pokračuje a takto postupně skáče do čím dál tím vyšších bloků, dokud buď není *chycena*, nebo nevyskočí z celého programu. Ve druhém případě Python vypíše o výjimce zprávu (včetně informace o tom, kde vznikla) a skončí.

Výjimku lze chytit pomocí bloku `try: ... except: ...`

```
try:
  x=y/z
except:
  print("Někdo tu asi dělil nulou!")
```

Všechny výjimky které se objeví v bloku `try:` způsobí, že se tento blok ukončí, provede se blok `except:` a výjimka se dále nešíří. Toto řešení ale nerozlišuje chyby, například pokud by bylo `y="Tralala"`, rovněž by došlo k výjimce. Proto existuje hierarchie výjimek a při `except:` je možné určit, které výjimky chceme chytit:

```
try:
  x=y/z
except ZeroDivisionError:
  print("Někdo tu dělil nulou!")
except TypeError, e:
  print("Vadné parametry: "+str(e))
except Exception, e:
  print("Jiná chyba "+str(e)+", předávám dál.")
```

```
  raise e
```

Tento příklad ukazuje, jak chytit různé výjimky. Druhý `except` přidává parametr `e`, do kterého se umístí objekt výjimky. Všechny nechycené výjimky propadnou dál. Třetí `except` chytí libovolnou nechycenou výjimku, ale po `print` ji hned zase vyvolá a tím pošle dál.

Vlastní výjimku vyvoláte příkazem `raise exc`, například `raise TypeError ("Vadné parametry")`. Výjimky jsou (skoro obyčejné) objekty odvozené od `Exception`, při vytvoření jim jako parametr můžete navíc předat vysvětlující řetězec.

Ke zpracování výjimky je možné ještě přidat blok `finally`, který se hodí pro úklid. Provede se těsně před tím, než program vyjde z bloku (ať už proto, že blok skončil, kvůli `return` či neošetřené (nechycené) výjimce). Pokud je výjimka ošetřena v `except` ještě před `finally`, provede se `finally` až po zpracování výjimky.

Typické použití:

```
def precti_cislo(jmeno_souboru):
  f = open(jmeno_souboru)
  try:
    return int(f.readline())
  finally:
    close f
```

Soubor bude uzavřen ať už se načtení čísla z prvního řádku souboru povede nebo způsobí výjimku (v tom případě už `return` samozřejmě neproběhne).

Nemá smysl vždy za každou cenu chytit všechny výjimky – kritické chyby (např. nešlo otevřít vstup) mohou klidně „probublat“ až nahoru a ukončit program. Chytejte ty výjimky, které umíte ošetřit.

Jak už jsme psali, jsou výjimky v Pythonu uspořádány do objektové hierarchie. Takto můžete chytat celé skupiny výjimek nebo jen konkrétní. Seznam všech výjimek a jejich hierarchii najdete v manuálu Pythonu.

Třídy a objekty

Jak už jsme psali, vše v Pythonu je objekt a může mít metody. Zkusíme v krátkosti popsat, co to přesně znamená a jak se objekty vytváří.

Každý objekt má nějaký typ, neboli je *instancí* nějaké *třídy*. Představte si to tak, že třída je typ a zároveň šablona pro tvorbu nových objektů a popisuje jejich společné vlastnosti, třeba metody. Podle této šablony se pak tvoří konkrétní instance objektů v paměti s daty konkrétních objektů a informací o jejich typu.

Například `dict` je typ pro slovníky, `d=dict()` vyrobí nový prázdný slovník stejně jako {}. Platí `type(d) == dict`, `d` je konkrétní objekt a je instancí třídy `dict`. Nové objekty se typicky vytváří voláním třídy jako funkce, často s parametry pro inicializaci objektu.

Objekty jsou zajímavé hned z několika důvodů. Ten první je to, že sdružují dohromady data (atributy) a metody do snadno uchopitelných „balíčků“

– jakmile dostanete do ruky (proměnné) objekt, automaticky k němu dostanete i většinu operací, které s ním lze provádět. Používání obyčejných funkcí navíc často vede ke jmenným konfliktům či jménům jako `pridej_do_seznamu` a `pridej_do_slovniku`.

Této vlastnosti se říká *zapouzdření (encapsulation)*[8]

Vlastní třídy

Novou třídu vytvoříte následujícím způsobem:

```
class Promenna(object):
  "Zbytečný obal na proměnnou"
  def __init__(self, hodnota):
    self.h = hodnota
  def cti(self):
    return self.h
  def zapis(self, hodnota):
    self.h = hodnota
  def __str__(self):
    return "Promenna(hodnota="+str(self.h)+")"
```

První řádek definuje jméno nové třídy (`Promenna`) a jméno rodičovské třídy (o tom později). Nepovinný řetězec je pak dokumentací dostupnou v `Promenna.__doc__` a `objekt.__doc__`. Další jsou definice metod, které se definují stejně jako funkce s tím, že jako první parametr (který se často jmenuje `self`, ale může se jmenovat jakkoliv) bude předán samotný objekt, pomocí něj se přistupuje k atributům a metodám objektu.

Metody se mohou jmenovat jakkoliv, některá jména mají ale speciální význam. `__init__` je tzv. konstruktor, tedy funkce, která se zavolá při vytváření objektu pomocí např. `p=Promenna(42.0)`. Tato funkce je jediná, která může vytvářet nové atributy (tím, že do nich přiřadí). Další funkce vytvářet nové atributy nemůžou.

Speciální funkce `__str__(self)` a `__int__(self)` se používají, když je potřeba převést objekt na řetězec (např. `print(p)`) nebo číslo. Další funkce jsou třeba `__abs__(self)` (použije se při `abs(p)`, `__mod__(self, x)` (při `p%x`) či `__neg__(self)` (při `-p`) a `__add__(self, x)` (při `p+x`). Takto a dalšími můžete vytvořit objekty, které se chovají jako čísla, seznamy, atd. Dokonce existuje metoda `__call__(self, parametry, ... )`, která pak umožňuje volat objekt jako funkci (např. `p(1)`).

Hierarchie tříd a dědičnost

Zatím jsme popsali objekty jako takové „lepší“ struktury či záznamy se zapouzdřením. Navíc ale můžeme vytvářet celé hierarchie podobných objektů.

[8] Často v kombinaci s tím, že k atributům objektu se nepřistupuje přímo, ale přes k tomu určené metody. Vnější svět pak nevidí skutečnou reprezentaci dat, která může být složitá nebo se časem vyvíjet.

Představme si například, že bychom krom `Promenna` chtěli pracovat s konstantními proměnnými, které nejdou měnit, či proměnnými omezenými na nezáporné hodnoty. Takové je možné vytvořit jako *potomky* třídy `Promenna`:

```
class Konstanta(Promenna):
  def zapis(self, hodnota):
    raise TypeError('Konstantu nelze měnit')
  def __str__(self):
    return "Konstanta(hodnota="+str(self.h)+")"
class Nezaporna(Promenna):
  def __init__(self, hodnota):
    self.h = None
    self.zapis(hodnota)[9]
  def zapis(self, hodnota):
    if hodnota<0: raise ValueError('nepovolená záporná
      hodnota')
    self.h = hodnota
  def __str__(self):[10]
    return "KladneCislo(hodnota="+str(self.h)+")"
```

Obě třídy *zdědí* všechny metody předka s tím, že mohou kterékoliv změnit. Kdykoliv pak voláte metodu objektu, volá se ta poslední definovaná[11]. Této vlastnosti se říká *dědičnost (inheritance)*.

Nyní můžeme vytvářet nejen proměnné, ale i `Konstanta` a `Nezaporna`, a používat je všechny dohromady (například jich mít pole a to sečíst pomocí `sum([i.cti() for i in pole])`). Této schopnosti potomků vystupovat (takřka) všude, kde dávají smysl předkové, se jmenuje polymorfizmus.

Nabízí se samozřejmě zajímavější použití jako například tzv. komponenty (widget) v aplikacích. Obecný `Widget` má polohu, velikost a metody jako `show` a `click` (které v samotném `Widget` nedělají nic konkrétního), a jeho potomci jsou pak například tlačítka, přepínače, menu, nebo okno. Tito předefinovávají `show` i `click` a občas přidávají další vlastnosti (např. okno může obsahovat další komponenty).

Pokud potřebujete zjistit, zda je nějaký objekt instancí nějaké třídy, stačí ověřit např. `type(p) == Promenna`. Častěji ale chcete vědět, zda je objekt instancí třídy, *nebo jejího potomka*. K tomu slouží operátor `isinstance`, např. `isinstance(p, Promenna)`. Toto jednak umožňuje rozlišovat celé podstromy tříd, jednak nikdy nevíte, kdo vyrobí dalšího potomka vaší třídy kvůli malé modifikaci.

---

[9] Nechci kontrolu psát dvakrát, ale `self.h` je třeba vytvořit.

[10] Nejrozumnější by samozřejmě bylo definovat už u `Promenna __str__` jako `return type(self).__name__ + "(hodnota=" + str(self.h) + ")"` a ušetřit si tím předefinovávání v potomcích.

[11] Jako u *virtuálních* metod v Pascalu a C++

Regulární výrazy

Regulární výraz (též regex, regexp či RE, podle anglického regular expression) je vzor, kterému odpovídá nějaká množina řetězců. Abychom mohli používat regulární výrazy v Pythonu, musíme naimportovat příslušnou knihovnu. To uděláme pomocí `import re`.

Abychom si mohli otestovat, jak regulární výrazy pracují, spustíme si interaktivní prostředí pythonu a napíšeme:

```
import re
vyskyt = re.search("ab","aabbcc")
```

Vyhledáváme v řetězci `aabbcc` vzor `ab`. Python nám vrátí objekt obsahující nalezený výraz a uloží ho do proměnné vyskyt. Budeme-li chtít zjistit na jaké pozici byl nalezen výskyt, stačí se zeptat

```
vyskyt.start()
```

Vrátí pozici 1. znaku nalezeného řetězce. Stejně tak `vyskyt.end()` vrátí poslední pozici, `vyskyt.span()` vrátí dvojici `(začátek, konec)`. Pozice v řetězci se, podle očekávání, indexuje od 0. Příkaz `vyskyt.group()` vrátí nalezený řetězec – v našem případě `ab`.

Co ale, pokud se zeptám takto?

```
vyskyt = re.search("ab","aaxxcc")
```

Python nenajde žádný výskyt podřetězce `ab` a do proměnné `vyskyt` přiřadí `None`.

Skupiny znaků

Na hledání podřetězců ovšem nepotřebujeme regulární výrazy, vystačili bychom si s `retezec.find(podretezec)`. Jedna z věcí, které přinášejí regexy navíc, jsou skupiny znaků. Uzavírají se mezi hranaté závorky `[` a `]`. Budeme-li chtít vyhledat libovolný podřetězec délky 3, který obsahuje písmenka a, b nebo c, zapíšeme ho takhle.

```
vyskyt = re.search("[abc][abc][abc]","aabbcxc")
vyskyt.group()
'aab'
```

Dostali jsme odpověď `'aab'`, protože to byl první první podřetězec `"aabbcc"`, který vyhověl regexu. Pokud bychom chtěli dostat všechny (nepřekrývající se) podřetězce, které mu vyhovují, můžeme použít:

```
re.findall("[abc][abc][abc]","aabbcc")
['aab', 'bcc']
```

Můžeme taky používat rozsahy, pomocí pomlčky. Skupinu všech malých písmen vytvoříme jako `[a-z]`, skupinu písmen nebo číslic jako `[A-Za-z0-9]`.

Dále taky občas chceme, aby na daném místě byl znak, který do skupiny *nepatří*. Toho se dosáhne pomocí znaku `^` na prvním místě ve skupině. Libovolný znak, který není číslicí tedy zapíšeme `[^0-9]`.

Některé skupiny znaků se používají tak často, že pro ně autoři Pythonu vytvořili zkratky. `\d` znamená libovolnou číslici, `\w` znak slova (tj. velká a malá

písmena, číslice a podrtržítko), `\b` hranici slova (místo mezi znaky, kde z jedné strany je slovo a z druhé ne) a `\s` prázdný znak (mezeru, tabulátor, nový řádek, ...). Všehny tyto skupiny mají i svou negovanou variantu, zapsanou velkým písmenem. Výčet není rozhodně úplný, další předdefinované skupiny najdete v dokumentaci modulu `re`. Ještě jedna veledůležitá skupina – tečka (`.`) zastupuje libovolný znak kromě nového řádku (`\n`).

Teď máme hlavu zamotanou spoustou zpětných lomítek. Aby toho nebylo málo, jak Python samotný, tak regulární výraz, považuje zpětné lomítko za speciální znak a mění význam znaku, co po něm následuje. Pokud chceme v normálním pythonovském řetězci napsat \, napíšeme `"\\"`. Pokud bychom chtěli v pythonovském regexu napsat `\d`, museli bychom napsat `"\\d"`. Abychom se z lomítkování nezbláznili, poskytuje nám Python takzvané surové (raw) řetězce. V řetězci uvozeném znakem `r` nemá zpětné lomítko žádný speciální význam. Regex, kterému vyhoví číslo o délce 3, proto můžeme zapsat takto:

```
vyskyt = re.search("\\d\\d\\d", retezec)
vyskyt = re.search(r"\d\d\d", retezec)
```

Všechny znaky, které jsou v regulárních výrazech speciální a je potřeba jim předřazovat zpětné lomítko, pokud je chcete použít bez speciálního významu, jsou tyto: `. ^ $ * + ? { } [ ] \ | ( )`

Kvantifikátory

Co když budeme chtít hledat desetiznakové slovo? Musíme to všechno desetkrát otrocky opsat? Nemusíme! Od toho máme kvantifikátory. Předchozí příklad zapíšeme lépe takto:

```
vyskyt = re.search(r"\d{3}", retezec)
```

`{3}` říká „předchozí znak (nebo skupina znaků) se opakuje třikrát“. Zápis `{3,}` říká „předchozí znak se opakuje alespoň třikrát“. Kromě toho máme speciální kvantifikátory `*` (opakuje se libovolněkrát (i nulakrát)), `+` (opakuje se alespoň jednou) a `?` (opakuje se nulakrát, nebo jednou).

Teď už toho umíme docela dost – pojďmě z textu vybrat všechna telefonní čísla:

```
re.findall(r"\d{9}", text)
```

Takhle najdeme všechny skupiny devíti číslic za sebou. Telefonní čísla ale taky někdy bývají zapsána ve skupinách po třech, budeme tedy hledat obojí:

```
re.findall(r"\d{3} ?\d{3} ?\d{3}", text)
```

Všimněte si volitelných mezer, které jsou kvantifikovány otazníkem – jsou přítomny jednou, nebo vůbec.

Skupiny

Často nepotřebujeme získat celý podřetězec, který vyhovuje regulárnímu výrazu, ale stačí nám nějaká jeho část. Někdy taky potřebujeme říct, že se má opakovat nějaký podřetězec. A taky by se nám hodilo, kdybychom mohli říct: „Na tomto místě se nachází tento řetězec, nebo tento řetězec“. Tohle všechno

umí v regulárních výrazech zařídit skupiny. Uzavírají se do kulatých závorek `(`, `)`.

```
vyskyt = re.search(r"(Jan|Zikmund|Jošt) Lucemburský",
"Byl to Jan Lucemburský, z Boží vůle král")
```

Tento regex bude odpovídat, pokud se v textu bude vyskytovat Jan, Zikumnd, nebo Jošt Lucemburský. Zároveň, zatímco ve `vyskyt.group(0)` bude uloženo `Jan Lucemburský`, ve `vyskyt.group(1)` bude uloženo `Jan`, tedy obsah první skupiny. Znak `|` je tzv. svislítko a používá se na oddělení alternativ ve skupině.

Číslo skupiny se určí podle pořadí její otvírací (levé) závorky. Stejně tak jednoduché je i opakování podřetězců:

```
vyskyt = re.search(r"(ram)+", "amramramramramra")
```

Dělení

Zatím jsme pomocí regexů pouze zjišťovali, zda odpovídají nějakému řetězci. Můžeme toho ale dělat i víc! Například dělit řetězce na části podle regulárního výrazu, kterému odpovídá oddělovač.

```
re.split(r"\W+", "Tahle veta obsahuje slova.")
['Tahle', 'veta', 'obsahuje', 'slova', '']
```

Předchozí regex rozsekal větu na slova. Lépe řečeno – použil jako oddělovač co nejdelší sekvence znaků, které nemohou být součástí slova.

Nahrazování

Poslední věcí, kterou si ukážeme, je nahrazování částí výrazu.

```
text = 'Náš Riki: jmeno="Riki" druh="Lišák" role="Maskot MaM"'
re.sub(r'(\w+)="([^"]*)"', r'jeho \1 je \2,', text)
Náš Riki: jeho jmeno je Riki, jeho druh je Lišák, jeho role je
Maskot MaM,
```

Tento aplikoval regulární výraz obsahující skupiny na `text` a nahradil všechny odpovídající podřetězce druhým parametrem funkce, ve kterém se navíc `\1` nahradí za první nalezenou skupinu, `\2` za druhou, atd. Část `textu`, která výrazu neodpovídala, byla ponechána tak.

Knihovna urllib

Knihovna `urllib` nám umožňuje pracovat s webem, konkrétně stahovat webové stránky. Ukážeme si jeden kompletní příklad:

```
import urllib
u = urllib.urlopen("http://mam.mff.cuni.cz/")
text = u.read()
u.close()
```

V proměnné `read` máme nyní uložen obsah naší titulní stránky, přesněji její HTML kód[12].

[12] více na `http://cs.wikipedia.org/wiki/HTML`

Urllib umí stáhnout stránky přes protokoly http, ftp a file (lokální soubory). V případě, že se stránku nepodaří otevřít, vyhodí výjimku, kterou je třeba ošetřit.

Urllib také umí posílat stránce POST parametry, jako při odeslání formuláře. Dělá se to následovně:

```
import urllib
params = urllib.urlencode({'jmeno': 'Jan', 'prijmeni': 'Novak',
heslo': 'ententyky'})
u = urllib.urlopen("http://www.example.com/login", params)
text = u.read()
u.close()
```

Podobným způsobem se jde přihlašovat, dají se vyplňovat kontaktní formuláře a podobně.

Tomáš & Honza

Úloha 3.5 – Seriál o pythonu III. díl (5b)

Všechny úlohy jsou řešitelné s tím, co jsme v seriálu zatím popsali, ale můžete používat i další standardní prostředky. Řešení nám prosím posílejte elektronicky včetně zdrojových souborů. Nějaké body samozřejmě získáte i za nástřel řešení, ale pro plný počet bodů je potřeba, aby program fungoval. Za nadstandardní řešení můžete dostat bonus.

I (1b): Napište regulární výraz, který bude odpovídat zápisu teploty (tj. číslo a jednotka). Nezapomeňte, že pro měření teploty máme různé jednotky.

II (4b): Napište program, který prolézá webové stránky a sbírá z nich emailové adresy. Využijte `urllib` k přístupu na internet a regulární výrazy pro hledání emailových adres a odkazů na další stránky k prozkoumání. Nezapomeňte ošetřit případné chyby a vadné odkazy, prohledávat jen do určité hloubky či počtu stránek a omezit opakované návštěvy téže stránky. Taky nám k programu pošlete stručný návod k použití.

Pokud budete zkoušet svůj program na obsah cizích stránek, zařaďte mezi přístupy nějaké zpoždění pomocí `time.sleep`. Pokud nehlídané prolézací skripty „zdivočí“, může to někomu znepříjemnit život. Mechanické zpracování dat je velmi užitečné, ale nezapomínejme na ohleduplnost (na piráty si zatím jen hrajeme).

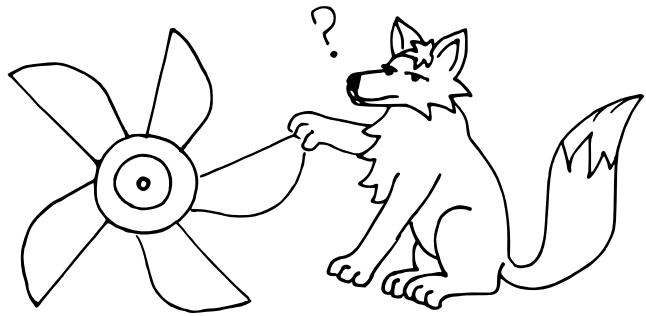
Za další luxusní vlastnosti programu nabízíme samozřejmě body navíc.

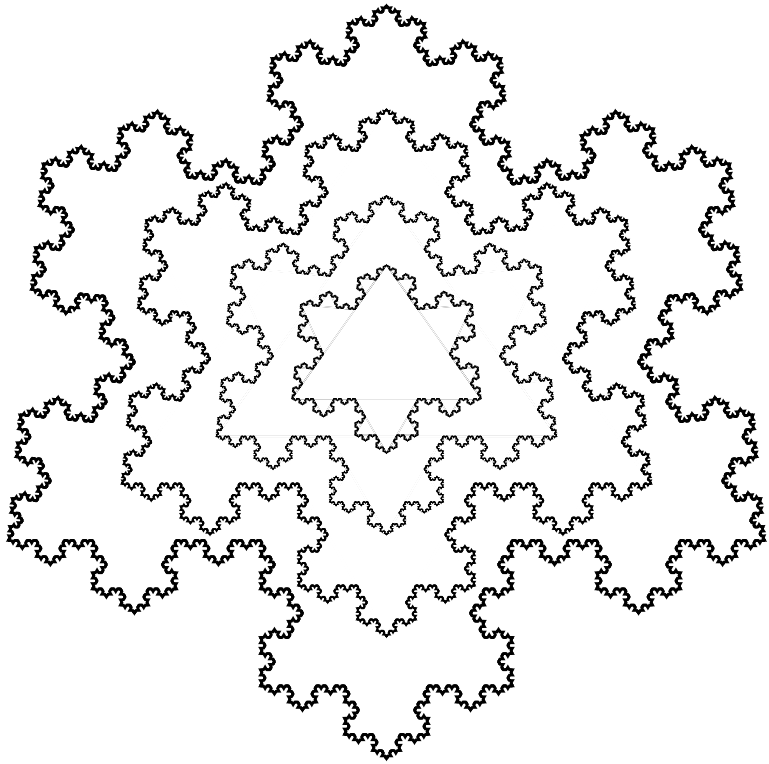
Výsledková listina

Poř.	Jméno	R.	$\sum_{-1}$	Číslo u1	u2	u3	u4	u5	t1	s1	+	$\sum_{1}$
1.	Prof.MM Š. Šimsa	2.	225	4		3	2	2		5	1	17
2–6.	Doc.MM A. Bušáková	4.	116	3			2		5	6	0	16
	Doc.MM P. Pecha	4.	199	4	4	1	2	1		4	0	16
	Dr.MM T. Pokorný	4.	68	4	4		2			6	0	16
	Dr.MM F. Štědronský	4.	94	4	4		2	1		5	0	16
	Bc.MM M. Töpfer	3.	16	4		3	2	2		5	0	16
7–8.	Bc.MM Le Anh Dung	1.	15	4		3	2			4	2	15
	Bc.MM V. Sedláček	3.	15	4	0		2	1	8		0	15
9–12.	Bc.MM L. Grund	1.	14	4	4	2	2				2	14
	Bc.MM J. Kubečka	3.	14	4	1	1	1	2	3	2	0	14
	Bc.MM T. Kubelka	3.	14	4	4	2	2	1			1	14
	Bc.MM J. Sopoušek	4.	14	2	4	3	1	1		3	0	14
13–15.	Mgr.MM M. Kocián	4.	43	2		3	2			5	0	12
	Bc.MM J. Novotná	2.	12	4		1	2	1		4	0	12
	Bc.MM J. Setnička	4.	12	4			2			6	0	12
16–18.	Bc.MM E. Gocníková	3.	11	2			1	1	7		0	11
	Bc.MM B. Mólová	3.	11	2	4		1	1		3	0	11
	Bc.MM L. Zavřel	4.	11	4		3	2	2			0	11
19–22.	Bc.MM M. Bílý	4.	10	4		2	2	2			0	10
	Bc.MM J. Bok	3.	14	4			2			4	0	10
	Dr.MM M. Kochmanová	4.	79	2	4		2			2	0	10
	Bc.MM F. Lux	4.	10	4	4		2				0	10
23.	Mgr.MM J. Škoda	4.	37	4	4		0	1			0	9
24–28.	Dr.MM M. Bekrová	4.	61	4			2			2	0	8
	Mgr.MM O. Cífka	2.	21					2		6	0	8
	Mgr.MM A. Harlenderová	3.	34	4	1		2	1			0	8
	R. Kubíček	2.	8	2	1	2	2	1			0	8
	D. Tělupil	3.	8	4			2	2			0	8
29–34.	Mgr.MM B. Böhmová	3.	37	1	4		2				0	7
	S. Havadej	4.	7	4		1	1	1			0	7
	Mgr.MM G. Kubíčková	4.	21	4			2	1			0	7
	L. Langerová	1.	7	1			2			4	0	7
	J. Svoboda	2.	7	4		2		1			0	7
	Mgr.MM K. Zemková	3.	21				1	2		4	0	7
35–39.	O. Fiedler	4.	7				1			5	0	6
	K. Kohoutová	2.	6	4			1	1			0	6
	M. Kopf	3.	6	4		1	1				0	6
	M. Landa	1.	6	4			1				1	6
	M. Poppr	1.	6	4			1				1	6

Poř.	Jméno	R.	$\sum_{-1}$	Číslo u1	u2	u3	u4	u5	t1	s1	+	$\sum_{1}$
40–41.	Dr.M F. Hlásek	4.	56	4			1				0	5
	V. Kletečka	3.	7	2	1	1	1				0	5
42–44.	Bc.M O. Mička	2.	10							4	0	4
	P. Vincena	1.	4				2	2			0	4
	D. Vít	1.	4		4						0	4
45–49.	E. Bušáková	2.	3	1			1	1			0	3
	J. Erhart	1.	3	1			1			1	0	3
	J. Kadlec	1.	3		0		2	1			0	3
	S. Ondrčková	2.	3	2		0	1				0	3
	B. Said	3.	3	1	2		0				0	3
50–51.	Bc.M P. Kratochvíl	3.	10			2	0				0	2
	O. Krčmář	2.	2		0		2				0	2
52–53.	O. Darmovzal	1.	1	1	0		0				0	1
	J. Křivonožka	3.	1				1				0	1
54.	V. Václavík	1.	0		0						0	0

Sloupeček $\sum_{-1}$ je součet všech bodů získaných v našem semináři, $\sum_{1}$ součet všech bodů v tomto ročníku. Sloupeček „+“ značí bonusové body udělované podle ročníku a součtu bodů za úlohy. Tituly uvedené v předchozím textu slouží pouze pro účely M&M.





Adresa redakce:

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

Telefon: +420 221 911 235
E-mail: MaM@atrey.karlin.mff.cuni.cz
WWW: http://mam.mff.cuni.cz

Časopis M&M je zastřešen Oddělením pro vnější vztahy a propagaci Univerzity Karlovy, Matematicko-fyzikální fakulty a vydáván za podpory středočeské pobočky Jednoty českých matematiků a fyziků.