Studentský matematicko-fyzikální časopis

M&M

Ročník XVII / Číslo 4

Milí kamarádi,

máme za sebou první půlku letošního ročníku, den otevřených dveří, Vánoce, II. mapování... Zajímavější ale je, co nás ještě čeká. Tím nejdůležitějším je **jarní soustředění**, které proběhne **od 2. do 10. dubna v Jizerských horách**. Pozveme na něj asi dvacet nejlepších řešitelů z tohoto ročníku.

Letos jsme se rozhodli vyhlásit **speciální soutěž**. Jedním ze základních pilířů našeho semináře jsou tématka, a tak se letos autor nejlepšího příspěvku k tématku může těšit na dort, který mu bude osobně doručen a předán organizátory. Uzávěrka soutěže je 31. května. Dort bude vítězi předán během června.

Vydařenou zimu přejí

Organizátoři

Zadání úloh

Termín odeslání čtvrté série: 28. 2. 2011

Úloha 4.1 – Sada čísel (3b)

Pepa měl letos opravdu zvláštní Vánoce. Představte si, že byste byli v jeho kůži. Hned první dárek, který našel pod stromečkem, byla veliká krabice. Když ji rozbalil, tak zjistil, že dostal sadu ne nutně různých 2011 reálných čísel.

Málem se lekl, že je stejná, jako dostal minulý rok. Ale pak si všiml, že tato sada je něčím zvláštní. Kdykoliv nějakých 2010 čísel ze sady vynásobí a to zbylé k výslednému součinu přičte, vyjde vždy ten samý výsledek.

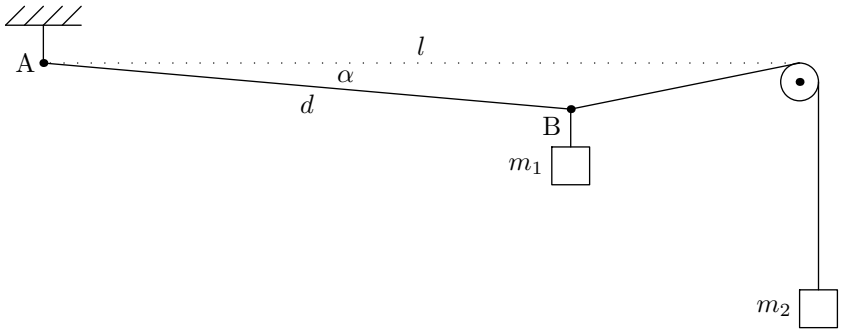
Zjistěte, kolik nejvíce mohlo být v Pepově sadě různých čísel.

Úloha 4.2 – Netradiční kyvadlo (5b)

Také další dárek, který Pepa rozbalil, nepatřil mezi něco, co obvykle naleznete pod vánočním stromečkem. Po rozbalení postupoval Pepa podle návodu. K pevnému bodu A (který byl součástí stavebnice) připevnil ideální provázek, na který ve vzdálenosti d přivázal závaží o hmotnosti m_1. Zbývající provázek přehodil přes ideální kladku, která byla ve stejné výšce jako bod A a byla ve vzdálenosti l. Na konec provázku přivázal závaží o hmotnosti m_2. Sestavená stavebnice je znázorněna na obrázku u4.2.1.

Pak Pepa celý mechanismus spustil. Na počátku byl provázek mezi bodem A a kladkou vodorovně napjatý ($\alpha = 0$). Jakou rychlost mělo závaží v závislosti na úhlu α? Pro jaký úhel α se závaží zastaví?

Nebojte se poslat jen náznak řešení. K výpočtu složitějších rovnic můžete použít počítač.



Obr. u4.2.1 – Sestavená stavebnice

Úloha 4.3 – Posloupnosti (4b)

Při rozbalování dárků si Pepa vzpomněl na peripetie, které zažil, když sháněl dárek pro svého bratra[1]. Chtěl mu dát určitou posloupnost nul a jedniček délky n. Požadovanou posloupnost však ne a ne sehnat.

Musel tak vymyslet algoritmus, kterým by převedl jinou posloupnost nul a jedniček $b_1, b_2, \ldots b_n$ na požadovanou posloupnost $a_1, a_2, \ldots a_n$. Jak by měl takový algoritmus vypadat, aby provedl co nejméně kroků?

Jedním krokem rozumíme přepsání všech číslic $b_j, b_{j+1}, \ldots b_k$ v nějakém intervalu $[j, k]$, $1 \leq j \leq k \leq n$ na jejich opak (jedničky přepíšeme na nuly a nuly na jedničky).

Úloha 4.4 – Kostičky (2b)

Poslední dárek byl pytlíček s kostičkami. Na každé kostičce byla jedna číslice od nuly do pěti. Když si Pepa tento dárek prohlížel, začal bezmyšlenkovitě sestavovat čísla v šestkové soustavě[2].

Jaký je součet všech čísel šestkové soustavy, v nichž je každá číslice obsažena právě jednou (nula může být na počátku čísla)? Výsledek uveďte v desítkové soustavě.

[1] Někdo tvrdí, že to byl dárek pro Pepovu sestru, ba dokonce, že Pepa žádného bratra nemá, ale to naštěstí v tuto chvíli není podstatné.

[2] Číslice v této soustavě jsou 0, 1,... 5. Číslo $(123)_6$ odpovídá hodnotě 51 v desítkové soustavě.

Řešení témat

Téma 3 – Ztracen v lese

Příspěvek k tomuto tématku zaslal zatím jen Mgr.ᴹᴹ Jakub Kubečka, ale ten se do toho rovnou řádně opřel. Navrhl Rikimu hned dvě strategie pro obdélníkový les v závislosti na jeho „podlouhlosti“ a „viditelnosti“ v něm a zabýval se také hvězdicemi a kruhy. Co si myslíš Ty o konceptu viditelnosti? Stojí za to ho rozvinout?

Konkrétní případy obdélníkových či kruhových lesů jsou samozřejmě dobrým začátkem, ale ještě hodnotnější by bylo nalézt strategii, při které Riki z lesa o ploše $S\,[\mathrm{km}^2]$ po několika kilometrech vybloudí nezávisle na jeho tvaru (pokud se ti to takto zdá těžké, přidej si klidně třeba předpoklad o neděravosti/konvexitě lesa). Také si můžeš zahrát na nepřítele a snažit se pro konkrétní Rikiho strategie hledat zákeřné lesy, ve kterých bude Riki bloudit dlouho předlouho.

Konečně můžeš zkoušet vylepšit konstantu $1 + 2 \cdot \pi$ určující, po kolika kilometrech nejpozději nalezne Riki (při vhodné taktice) přímou cestu vzdálenou přesně kilometr od místa svého startu. Věř, že vylepšovat je co `:-)`. Každých pár desítek metrů je tu cenných, neváhej se proto podělit o byť jen drobné vylepšení.

Pepa

Téma 4 – Sjezdovka

K tomuto tématu přišel jediný příspěvek, a to od Mgr.ᴹᴹ Jakuba Kubečky. Jakub sepsal pár základních vzorečků a uvedl několik jednoduchých postřehů, ale stále je ještě hodně co řešit.

Jakub vycházel ze zákona setrvačnosti. Tedy aby lyžař jel stále stejnou rychlostí, musí být výslednice sil na něho působících nulová. Síly, které na něho působí, jsou tíhová ($F_g = mg$, kde m je hmotnost lyžaře a g tíhové zrychlení, a ne gravitační, prosím vás, v našem g máme započteno i odstředivé zrychlení vlivem rotace Země) a třecí síla ($F_t = f \cdot F_n$, kde f je koeficient smykového tření a F_n je síla kolmá k povrchu, v tomto případě složka síly tíhové).

Svah má sklon α, který chceme zjistit. Abychom mohli dál počítat, musíme si tíhovou sílu rozložit na složku komou k povrchu F_n a rovnoběžnou s povrchem F_p

$$F_n = mg\cos\alpha \quad F_p = mg\sin\alpha\,.$$

Pak už známe i třecí sílu

$$F_t = fmg\cos\alpha\,.$$

Pak Jakub sečetl složku tíhové síly ve směru rovnoběžném se sjezdovkou a třecí sílu, a požadoval, aby tento součet byl roven nule

$$fmg\cos\alpha - mg\sin\alpha = 0\,.$$

A našel podmínku pro sklon sjezdovky

$$\operatorname{tg}\alpha = f\,.$$

Nezdá se vám, že na něco zapomněl? Na začátku uvedl zákon setrvačnosti, ze kterého plyne, že součet všech sil musí být nula, a zatím byl zkoumán jen součet složek sil v jednom směru ...

Složka tíhové síly kolmá ke sjezdovce se samozřejmě vyrovná s reakcí povrchu sjezdovky dle zákona akce a reakce. To je tak jednoduché, že bych vás urazila, kdybych se vás na to ptala. (Takže nejspíš nezapomněl.) Ale když už děláme rozbor sil, tak je vhodné to uvést.

Ale ještě zapomněl na něco. Kontakt dvou povrchů se chová jinak, když se pohybují, a jinak, když jsou v klidu. Říká se tomu statické a dynamické tření. V právě vyřešeném problému se správně použije dynamický třecí koeficient, ale kdybychom hledali podmínku, za které lyžař ze sjezdovky ještě nesjede, museli bychom použít statický, který bývá obvykle větší. Bylo by zajímavé zkusit změřit, jaký je pro sníh mezi těmito koeficienty rozdíl.

Také nebyl diskutován odpor vzduchu, který je další silou brzdící pohyb lyžaře. V zadání úloh často bývá formulka „odpor vzduchu zanedbejte", tak už si člověk tak nějak zvykl ho zanedbávat automaticky, což není úplně správně. Tentokrát v zadání taková formulka nebyla. Zkusíte se někdo zamyslet nad tím, jak moc odporová síla lyžaře ovlivní?

Dál Jakub jednoduchou úvahou o velikosti tíhové a třecí síly dospívá k podmínkám pro rozjezd

$$\operatorname{tg}\alpha' > f$$

z nenulové velmi malé rychlosti, vzhledem k minulému odstavci (řekněme, že se odpíchne) a zabrzdění

$$\operatorname{tg}\alpha'' < f\,.$$

Dále uvádí, že by bylo vhodné, aby bylo $\alpha < 90^o$.

Bylo by ale vhodné zamyslet se, v jakém bodě této sjezdovky se už rychlost lyžaře přestane měnit, a jak zařídit, aby to bylo právě v bodě změny sklonu z α na α'. Takže nás nebudou zajímat jenom sklony, ale i délky jednotlivých úseků.

Ještě bychom chtěli, aby lyžařovi rozjíždění a brzdění zabralo nějaký rozumně krátký čas, což omezí rozpětí možných délek a sklonů úseků.

Dál se Jakub zamýšlel nad sjezdovkou s úseky střídavě o sklonech α', α, α'', α, α', α, ... Tato posloupnost zřejmě vede k periodickému průběhu rychlosti lyžaře. Lze ale nějak konkrétně zvolit hodnoty sklonů a délek úseků, aby to byla sinusoida?

Takže, jak je vidět, problémů k řešení stále zůstává spousta. A sněhu venku je taky hromada. Tak doufám, že toho náležitě využijete. Zimě zdar!

Zuzka

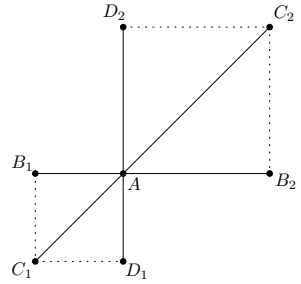
Řešení úloh

Úloha 2.1 – Ubrousky (4b)

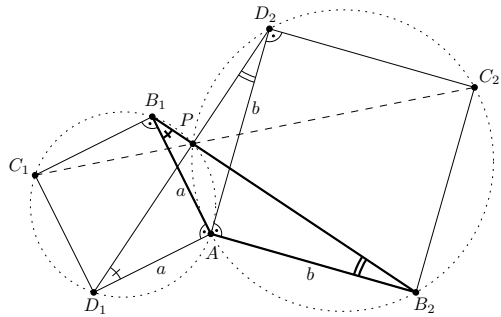
Zadání:
Během snídaně Jarka zaujaly dva čtvercové ubrousky ležící na stole, a tak si s nimi začal hrát. Dal je k sobě tak, aby se dotýkaly pouze jedním rohem, ten nazval A. Zbylé rohy jednoho ubrousku pak označil B_1, C_1 a D_1, druhého B_2, C_2 a D_2. A představoval si v duchu přímky B_1B_2, C_1C_2 a D_1D_2. Dokažte, že se mu tyto tři přímky vždy protnou v jednom bodě. A to bez ohledu na velikost a natočení ubrousků.

Řešení:
Označme P průsečík přímek B_1B_2 a D_1D_2. Ukážeme, že P leží i na přímce C_1C_2, čímž bude úloha dokázána.

Prvně vyřešme možnost, ve které body B_1, A, B_2 (a zároveň body D_1, A, D_2) leží v přímce. V takovém případě musí ubrousky vypadat jako na prvním obrázku a tvrzení úlohy je zřejmé. Dále tedy předpokládejme, že tyto trojice bodů v přímce neleží (viz druhý obrázek).



Obr. 1



Obr. 2

Obr. u2.1.1

Všimněme si, že trojúhelníky D_1AD_2 a B_1AB_2 jsou shodné. Skutečně jelikož $AB_1C_1D_1$ a $AB_2C_2D_2$ jsou čtverce, platí $|D_1A| = |B_1A|$ a $|AD_2| = |AB_2|$. Navíc

$$|\angle D_1AD_2| = 90^\circ + |\angle B_1AD_2| = |\angle B_1AB_2|\,,$$

takže trojúhelníky D_1AD_2 a B_1AB_2 jsou shodné podle věty *sus*. To mimo jiné znamená, že se shodují i v ostatních vnitřních úhlech, tedy

$$|\angle AB_2B_1| = |\angle AD_2D_1| \qquad \text{a} \qquad |\angle B_2B_1A| = |\angle D_2D_1A|\,.$$

Z první rovnosti bezprostředně plyne, že úsečka AP je z bodů B_2 a D_2 vidět pod stejným úhlem. Proto bod P leží na kružnici opsané trojúhelníku AB_2D_2, což je i kružnice opsaná čtverci $AB_2C_2D_2$. Obdobně z druhé rovnosti vyvodíme, že P leží na kružnici opsané čtverci $AB_1C_1D_1$.

Nyní už jen dvojím použitím věty o obvodových úhlech dostaneme

$$|\angle C_1PA| + |\angle APC_2| = |\angle C_1B_1A| + |\angle AD_2C_2| = 90^\circ + 90^\circ = 180^\circ$$

a jsme hotovi.

Úlohu šlo také poměrně přímočaře a elegantně vyřešit analyticky, jak pěkně předvedl Mgr.ᴹᴹ Jan Sopoušek.

Pepa

Úloha 2.2 – Lasery (4b)

Zadání:

Jarka napadla při čtení následující otázka: Jak široký musí alespoň být laserový svazek (o dané vlnové délce), pokud urazil vzdálenost ze Země na Měsíc? Uměli byste mu na ni odpovědět?

Řešení:

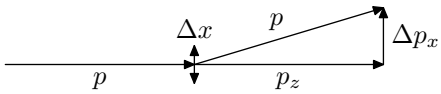
Zopakujme ešte, že zadanie obsahuje prakticky všetky potrebné vzťahy na riadne vyriešenie úlohy, takže sa toto redukuje na krátku úvahu a dosadenie do vzorčeka.

Heisenbergov princíp neurčitosti uvedený v zadaní prevedením do prostého jazyka hovorí to, že pokiaľ sa svetlo šíri priamočiaro cez úzku štrbinu, rozptyľuje sa. Keďže hľadáme minimálny priemer laserového zväzku, položíme v Heisenbergovom princípe neurčitosti namiesto neostrej nerovnosti rovnosť a dostaneme

$$\Delta p_x = \frac{\hbar}{2\Delta x}\,, \tag{u2.2.1}$$

kde predpokladáme (ako sa to často robí), že svetlo sa šíri v smere osi z. Celková veľkosť hybnosti $p = h\nu/c$ svetla (závislá na frekvencii ν) by sa ale mala zachovať a tak dostávame trojuholník hybností ako na obrázku u2.2.1. Veľkosť p_z nepoznáme, ale uhol odchylky od pôvodného smeru šírenia môžeme ľahko spočítať

$$\sin\alpha = \frac{\Delta p_x}{p}\,. \tag{u2.2.2}$$



Obr. u2.2.1

Čo táto odchylka spôsobí po ceste do vzdialenosti l? No predsa zväčší priemer zväzku o

$$h = l\tan\alpha\,, \tag{u2.2.3}$$

na všetky strany. Ach hrôza! Poznáme iba sínus uhlu a potrebujeme jeho tangens. Ale nebojme sa, uhol bude určite malý (predsa len, vychádzame z praktických skúseností s lasermi a z toho, že hľadáme minimálny priemer) a tak

platí $\tan\alpha \approx \sin\alpha \approx \alpha$[3]. Celkový priemer zväzku teda bude pôvodný priemer zväčšený o dvojnásobok h.

$$d = \Delta x + 2h = \Delta x + 2l\tan\left[\arcsin\left(\frac{\Delta p_x}{p}\right)\right] \approx \Delta x + \frac{lc}{2\pi\nu\Delta x}\,, \qquad \text{(u2.2.4)}$$

kde sme postupne dosadili za h, nahradili tangens uhlu jeho sínusom a dosadili za Δp_x z (u2.2.1) a za p. Vidíme, že šírka zväzku sa skladá z dvoch členov. Jeden závisí lineárne na Δx, teda na pôvodnom priemere lúča, druhý závisí na Δx nepriamo úmerne a prejaví sa teda hlavne pri malých priemeroch zväzku. Takýto výraz by mohol mať minimum. Mohli by sme ho hľadať nejakou numerickou metódou, ale ako na prakticky každom sústredení učíme v kurze matematickej analýzy, na hľadanie minima existuje šikovný matematický nástroj zvaný derivácia. Tak teda spočítajme prvú deriváciu vzťahu (u2.2.4) podľa Δx

$$\frac{\mathrm{d}}{\mathrm{d}\Delta x}\left(\Delta x + \frac{lc}{2\pi\nu\Delta x}\right) = 1 - \frac{1}{\Delta x^2}\frac{lc}{2\pi\nu}\,. \qquad \text{(u2.2.5)}$$

Keďže derivácia udáva smernicu dotyčnice ku grafu funkcie, či inak, ako rýchlo funkcia rastie a v minime funkcia nerastie ani neklesá, prvá derivácia v minime funkcie musí byť nulová. Spočítajme, kedy je nulová derivácia (u2.2.5)

$$\begin{aligned} 1 - \frac{1}{\Delta x^2}\frac{lc}{2\pi\nu} &= 0\,, \\ \Delta x^2 &= \frac{lc}{2\pi\nu}\,, \\ \Delta x &= \sqrt{\frac{lc}{2\pi\nu}}\,. \end{aligned} \qquad \text{(u2.2.6)}$$

V tento moment zostáva dosadiť konkrétne čísla. Najprv ale vymeňme frekvenciu žiarenia ν za jeho vlnovú dĺžku λ (platí $\nu = c/\lambda$), čím dostaneme $\Delta x = \sqrt{l\lambda/2\pi}$. Vzdialenosť Zem-Mesiac je asi pol milióna kilometrov (väčšia presnosť je zbytočná), tam a späť to teda dá $l \approx 10^9$ m. Vlnová dĺžka svetla pre červený laser je približne $\lambda \approx 650$ nm $\approx 2\pi\cdot 10^{-7}$ m. Po dosadení sa väčšina čísel krásne vykráti a dostáneme $\Delta x = 10$ m. To je trochu veľa, ale dopočítajme i celkovú šírku lúča

$$d = \Delta x + 2h = \Delta x + \frac{l\lambda}{2\pi\Delta x} \quad \rightarrow \quad d \approx 20\ \mathrm{m}\,,$$

takže odpoveď na otázku zo zadania znie: „laserový zväzok má po ceste zo Zeme na Mesiac a späť teoretický najmenší možný primer rádovo desiatky metrov“.

Čo by to bolo za počítanie príkladu, keby sme nemali nejaké porovnanie so skutočnosťou? Stránky Lunar and Planetary Institute[4] udávajú priemer laserového zväzku pri skutočnom meraní vzdialenosti Mesiaca v hodnotách okolo

[3] Uhol α dosadzujeme v radiánoch.

[4] Naleznete ich na `http://www.lpi.usra.edu/lunar/missions/apollo/apollo_11/experiments/lrr/`.

7 km na povrchu Mesiaca a až 20 km po návrate k Zemi. Vyrobiť ale laser s priemerom 10 m je nemožné a pokiaľ zmenšíme priemer zväzku na centimeter, aj nám vyjdu desiatky kilometrov, takže môžeme povedať, že náš výsledok je v dobrej zhode s praxou.

Jeffer

Úloha 2.3 – Hledání v mlze (4b)

Zadání:

Ve snu Jarek hledal v mlze ve čtverečkované síti o rozměrech $m \times n$ ztraceného kamaráda. Ten naopak hledal jeho. Mlha byla přitom tak velká, že bylo vždy vidět pouze na políčko, kde zrovna stál, a na sousední políčka. Hledali se tak, že vždy alespoň jeden z nich udělal krok do náhodně zvoleného směru (všechny možné pohyby měly stejnou pravděpodobnost) a rozhlédl se, jestli toho druhého nevidí.

Bylo by lepší, kdyby oba hledali zároveň, nebo kdyby jeden stál a druhý hledal? Jarek měl dost času, a tak na odpověď po chvíli přišel. Zkuste to i vy. Odhadněte nejdříve výsledek bez počítače. Pak napište program, který danou situaci nasimuluje, a udělejte pomocí něj statistiku, kterou nakonec nějak interpretujte. Závisí nějak výsledky na rozměrech čtverečkované sítě?

Řešení:

Zkusme se nejprve zamyslet bez pomoci počítače. Představme si na chvíli, že rozměry šachovnice jsou $1 \times \infty$ a náhodný výběr umístil naše kamarády do vzdálenosti a políček od sebe. Snadno nahlédneme, že v případě, kdy se hýbou oba kamarádi, je tato situace stejná, jako by se hýbal pouze jeden z nich a to o 2 políčka nebo o 0 políček. Přitom se pohne o 2 políčka blíž ke svému kamarádovi s pravděpodobností 1/4, o 2 políčka dál od svého kamaráda také s pravděpodobností 1/4 a stát na místě zůstane s pravděpodobností 1/2. V tabulce u2.3.1 jsou uvedeny pravděpodobnosti, že se naši přátelé najdou právě po n krocích (jsou-li mezi nimi na začátku 4 políčka).

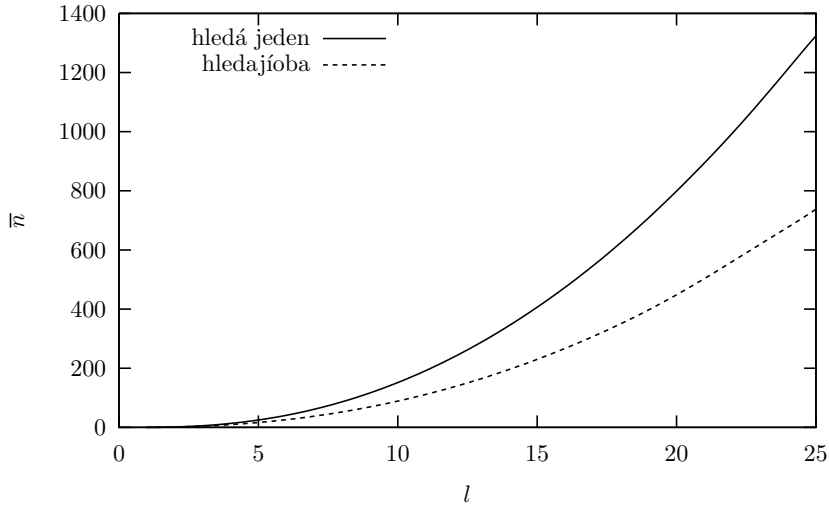
n	P (hledá jeden)	P (hledají oba)
1	0	0
2	0	$1/2^4$
3	0	$4/2^6$
4	$1/2^4$	$7/2^7$

Tabulka u2.3.1: Pravděpodobnost nalezení právě po n krocích.

Kdybychom s tabulkou pokračovali a spočítali pravděpodobnost, že se najdou nejpozději (ne právě) v n–tém kroku, poznali bychom, že je větší pravděpodobnost, že se najdou dříve, pokud budou hledat oba dva (exaktní důkaz nepředkládáme, pouze dáváme náhled).

K této úloze přišlo několik pěkných řešení a skoro všechny se shodly na výsledku počítačové simulace: „Když hledá pouze jeden kamarád (a druhý stojí),

trvá hledání zhruba dvakrát delší dobu, než když se hledají oba“. Pokud vás zajímá simulační program organizátorů, naleznete jej na našem webu v sekci Z řešení[5]. Je napsán v jazyce Perl. Graf u2.3.1 znázorňuje závislost průměrné doby, za kterou se kamarádi setkají v závislosti na velikosti (čtvercové) šachovnice.



Obr. u2.3.1 – Výsledky simulace průměrného počtu kroků $\overline{n}$ na velikosti šachovnice l.

Pro každou velikost šachovnice bylo provedeno 10000 pokusů a z nich spočten průměr. Program počítá taktéž směrodatnou odchylku, jejíž hodnota je však velmi vysoká, což svědčí o nestálosti počtu kroků. V jednotlivém pokusu se můžete najít hned, ale taky se můžete hledat opravdu hodně dlouho. V průměrném případě (který jsme zkoumali) se však vyplatí, pokud se budou hledat oba kamarádi navzájem.

Honza

Úloha 2.4 – Karty (2b)

Zadání:
Bedřich má balíček 23 *karet. Na každé z nich je na jedné straně kolečko a na druhé křížek. Přitom* 14 *karet je otočených nahoru křížkem, zbývajících* 9 *kolečkem. Bedřich tvrdí, že když mu zavážou oči a karty zamíchají (během míchání se nesmí žádná karta obrátit), dokáže je rozdělit na dvě hromádky tak, aby na každé z nich bylo stejně karet otočených křížkem.*[6] *Dokázali byste to i vy?*

Řešení:
Nejdříve si uvědomíme, že pokud vytvoříme hromádku, na které bude určitý počet koleček, tak umíme všechny karty na hromádce otočit a budeme mít tento počet křížků.

[5] `http://mam.mff.cuni.cz/zr`

[6] Během rozdělování už Bedřich může karty libovolně otáčet.

Karty Bedřich rozdělí na dvě hromádky, do první dá libovolných čtrnáct karet, do druhé zbývajících devět. Označme počet karet otočených nahoru křížkem v první hromádce k. V druhé jich pak bude určitě $14-k$. Nyní stačí otočit všechny karty v první hromádce. V ní bylo $14-k$ koleček, takže teď tam bude tolik křížků. A jsme hotovi.

Kuba

Úloha 2.5 – Seriál o Pythonu (II. díl) (7b)

Zadání:
I.a (1b): *Popište, co dělá následující program:*

```
def fix(f,*p):return f(f,*p)
print(fix(lambda f,x:(x<1 and 1) or x*f(f, x-1), 10))
```

Řešení:
Ze samotné lambda funkce není snadné vytvořit rekurzivní funkci, protože lambda (anonymní) funkce nemůže volat sebe sama. Lambda funkce „obalená“ funkcí `fix` dostane krom parametrů `p*` i sebe jako `f` a může tedy být rekurzivní.

Vnitřek lambda funkce je už jen faktoriál realizovaný s pomocí zkráceného vyhodnocovaní `and` a `or` (což je v některých jazycích běžná finta). Pokud je `x<1`, vrátí se druhý argument `and`, jinak je první argument `or` nepravda a vrátí se hodnota funkce pro `x-1`. Pozor na to, že i rekurzivnímu volání funkce `f` je potřeba předat `f`, aby mohla v rekurzi pokračovat.

Zadání:
I.b (2b): *Pomocí* `fix` *a lambda-funkcí (bez definic vlastních funkcí) spočtěte efektivně n-tý prvek Fibonacciho posloupnosti. Vyhněte se exponenciálnímu počtu volání.*

Řešení:
Je velmi jednoduché upravit předchozí příklad:

```
fix(lambda f,x:(x<=2 and 1) or f(f, x-1) + f(f, x-2), n))
```

Toto ale trvá exponenciálně dlouho – pokud si nakreslíte jednotlivá volání do stromu, zjistíte, že F_n se nasčítalo z F_n jedniček v jeho listech.

Lepší řešení je předávat si vždy dvojice (F_n, F_{n-1}). Následující dvojice pak je $(F_n + F_{n-1}, F_n)$. Dvojice si budeme předávat jako dva parametry:

```
fix(lambda f,x,a,b:(x<=2 and b) or f(f, x-1, b, a+b), n, 1, 1))
```

Funkce tak začne s parametry (n, F_1, F_2) a pokračuje přes $(n-1, F_2, F_3)$ až k $(2, F_{n-1}, F_n)$ a pak vrátí (zpět ze všech zanoření) F_n.

Zadání:
II (2b):*Seznamy v Pythonu mohou obsahovat cokoliv, včetně sebe sama. Např.:*

```
a=['A','a']; b=[a,'B',a]; c=[b,'C',a,'c',b]; a.insert(1,c)
```

Vytvořte funkci, která takové zacyklené seznamy vypisuje. Nesmíte při tom použít žádnou Pythoní funkci, která by to udělala za vás, jako `str` *či* `repr`*. Výstup by měl být co nejčitelnější,*

můžete tedy například vyzdvihnout hloubku zanoření odsazením. Bylo by navíc dobré si seznamy např. očíslovat a nějak naznačit, který už vypsaný seznam je ten vnořený.

Řešení:

U této úlohy došlo k chybě zadavatele *(Tomáše)*, který v seriálu zapomněl zmínit operátor `is` nebo funkci `id`, bez kterých je úloha takřka neřešitelná. Za chybu se velmi omlouváme a doufáme, že jste nad jejím řešením nestrávili moc času.

Operátor `is` porovnává objekty podle adresy, nikoliv podle obsahu, takže pozná, když jde o ten *samý* objekt, aniž by jej zkoumal uvnitř. Funkce

`id(cokoli)`

vrátí unikátní identifikátor objektu. Ve skutečnosti je to ukazatel převedený na číslo. Takže `a is b` je ekvivalentní `id(a) == id(b)`.

Teď tedy k samotnému řešení: Seznamy budeme vypisovat rekurzivně a hlavní myšlenka je pamatovat si, které objekty jsme během procházení už viděli a u těch místo rekurze jen vypsat jméno. Snad nejdůležitější je ale všimnout si, že pro porovnávání nesmíme používat `==`, protože to by porovnávalo cyklické seznamy rekurzivně a mohlo se zacyklit.

Použijeme tedy funkci `id` a budeme si v množině `videno` pamatovat identifikátory už vypsaných objektů. Abychom u seznamů, které jsme viděli dvakrát, mohli u prvního výskytu napsat i identifikátor (a u ostatních ne), budeme si je pamatovat v množině `dvakrat`.

```
def reclist_p(seznam, videno, dvakrat):
  videno.add(id(seznam))    # Zapamatuj si seznam
  vratit = "["              # Vraceny retezec
  for i in seznam:
    if type(i) == list:     # Prvek je seznam?
      if id(i) in videno:
        vratit += "<objekt s id=%d>, " % id(i)
        dvakrat.add(id(i))
      else:                 # i jsme jeste nevideli
        vratit += reclist_p(i, set(videno), set(dvakrat))+", "
    else:                   # Prvek je neco jineho
      vratit += "%r, " % i  # Primo vypis hodnotu
  if len(vratit)>1:         # Urizni ", " na konci
    vratit = vratit[:-2]
  if id(seznam) in dvakrat: # Na id se odkazuje, vypis ho
    vratit = ("<%d>:" % id(seznam)) + vratit
  return vratit + "]"

# Pomocna obalka
def reclist(seznam): return reclist_p(seznam, set(), set())
```

Zadání:

III (2b):*Napište program, který transponuje textový soubor jako matici, tedy prohodí znaky mezi pozicemi (řádek x, sloupec y) a (sloupec x, řádek y). Kde je to potřeba, vyplňte prázdná místa mezerami. Všechny mezery na koncích řádků odstraňte (i kdyby se jednalo o ty z původního souboru).*

Jména vstupního a výstupního souboru ideálně přečtěte z příkazové řádky (ze `sys.argv`*). Pokud jste machři, můžete ošetřit i kódování souboru jako UTF-8 a jiné. Pokud nevíte, o co jde, nemusíte se o to starat (předpokládejte 1 bajt = 1 znak), jen to možná nebude fungovat na soubory s češtinou.*

Řešení:

Předkládáme zhuštěnou verzi, která využívá kdejakou fintu, abychom ilustrovali jejich použití v praktickém programu.

Pro jednoduchost neuvažujeme různá kódování a vynechali jsme kontrolu chyb při otevírání souborů (program skončí s výjimkou bez hezké české hlášky).

```
import sys
if len(sys.argv) != 3:
  print("Pouziti: %s vstup vystup" % sys.argv[0])
  sys.exit(1) # Chceme prave 2 parametry

vstup = open(sys.argv[1], 'rt')
radky = [ r.rsrip() for r in vstup.readlines() ]
vstup.close()
```

Během načítání řádek rovnou odsekáváme konce řádků a mezery. Dále zjistíme délku nejdelšího řádku a jeden po druhém vypočteme `sloupce` (jako řetězce).

```
delka = max([len(r) for r in radky])
sloupce = []
for s in range(delka):
  sloupce.append("".join( # Seznam znaku spojit do retezce
    [ ((s<len(r) and r[s]) or ' ') for r in radky ]
    )) # Pokud je r[s] za koncem r, vezmu misto nej mezeru

vystup = open(sys.argv[2], 'wt')
vystup.write( "\n".join([ s.rstrip() for s in sloupce ]) )
vystup.close()
```

Výstupu už jen odřežeme mezery na konci a spojíme ho před vypsáním konci řádků.

Tomáš

Seriál o Pythonu (IV. díl)

V tomto díle se už úplně zaměříme na hraní si s knihovnami, tentokrát s knihovnou určenou pro psaní her a multimediálních programů.

Knihovna *PyGame* je Pythoní rozhraní k C knihovně *SDL*. SDL (Simple Directmedia Layer) je multimediální knihovna umožňující jednoduše pracovat s grafikou, zvukem, klávesnicí, myší, časem a třeba i 3D grafikou (skrze OpenGL). Programy napsané v SDL i PyGame fungují stejně dobře pod Windows, Linuxem i MacOS X.

V Linuxu najdete PyGame v balíčku `python-pygame` či podobném, ve Windows doporučuji Portable Python 1.1, jehož je PyGame součástí. O Portable Pythonu píšeme na stránkách seriálu.

Na stránkách seriálu[7] jsme pro vás krom užitečných odkazů připravili i kostry interaktivních aplikací v PyGame a pár jednoduchých funkčních příkladů.

Knihovna PyGame

Knihovna PyGame (stejně jako SDL) je rozdělena na několik částí. V základním modulu `base` jsou funkce pro zpracování událostí (jako je pohyb myši, stisk klávesy či nastavený časovač) a základní objekty jako `Rect` (obdélník) či `Color` (barva). Ostatní moduly jsou například `display`, `font`, `mixer`, `scrap`, `midi`, `joystick` a `cdrom`. Pozor na to, že ne všechny jsou vždy dostupné.

Po importu modulu `pygame` inicializujete všechny dostupné moduly příkazem `pygame.init()`, který vrátí počty `(modulu_OK, modulu_KO)`. PyGame nehlásí nekritické chyby (jako selhání inicializace zvuku či jiného modulu) výjimkami, ale musíte si o připadnou chybovou zprávu říci `pygame.get_error()`.

Pro snazší práci se může hodit naimportovat si modul pomocí `from pygame import *`, pak můžete používat knihovnu bez prefixu `pygame.`, my ho v textu často vynecháváme. Stejně tak často vynecháváme jméno podmodulu, pokud je z kontextu jasné, o který se jedná.

Poloha bodu či údaj o velikosti je v PyGame vždy dvojice `(x, y)`, resp. `(sirka, vyska)`. Všechny souřadnice jsou v Pygame (až na pár výjimek) celá čísla. Souřadný systém má, jak už je na počítačích běžné, počátek vlevo nahoře a osa y směřuje dolů.

Jak je běžné třeba v C, používají SDL a PyGame pro různé volby (tzv. *flags*) jednotlivé bity nějakého čísla. Pro čajovou knihovnu byste měli nadefinováno např. `CUKR=1, MLEKO=2, SUL=4, MASLO=8` a tibetský čaj by pak byl `uvar_caj("Cerny", MASLO | SUL)`. Jednotlivé volby jsou nejčastěji čísla 2^n a kombinují se *bitovou disjunkcí* (`|`).

Nesčítat! Pomocná definice `ANGLICKY=CUKR|MLEKO` má velmi dobrý smysl, ale výraz `uvar_caj("Earl", ANGLICKY + CUKR)` pak ilustruje možné fatální následky.

[7] `http://mam.mff.cuni.cz/python`

PyGame definuje mnoho konstant, které odpovídají různým volbám, kódům kláves nebo typům událostí. Všechny jsou přímo v modulu `pygame` a většinou jsou velkými písmeny.

Nyní k jednotlivým částem. Budeme se je snažit spíš uvést, podrobnosti a možnosti příkazů si můžete zjistit v manuálu.

Grafika

Typ `Surface` reprezentuje bitmapu, na kterou lze kreslit. To může být buď přímo okno na obrazovce, nebo bitmapa v paměti. Grafiku inicializujete pomocí `screen = display.set_mode( rozliseni_okna )`, `screen` je pak speciální bitmapa pro kreslení na obrazovku. Jako druhý parametr můžete přidat flags jako třeba `FULLSCREEN`. Pokud potřebujete pomocnou bitmapu v paměti, vytvoříte ji `b = Surface(velikost)`.

Bitmapa reprezentující obrazovku není ve skutečnosti přímo obrazovka. Ve skutečnosti je též v paměti a průběh kreslení se na obrazovce neukazuje (hlavně kvůli výkonu, ale i proto, aby nebyl vidět průběh kreslení). Vždy když chcete, aby se stav `screen` ukázal uživateli, zavolejte `display.update()`. Je to též někdy potřeba udělat, když např. přepnete mezi okny, k tomu více níže.

Barva je reprezentována objektem `Color`, vnitřně pak jako kombinace červené (R), zelené (G), modré (B) a případně průhlednosti (A) vždy v rozsahu 0-255. K inicializaci `Color` můžete použít `(r,g,b)`, `(r,g,b,a)` nebo HTML kód barvy `'#rrggbb'`. Místo `Color` můžete ve většině funkcí použít i přímo `(r,g,b)`.

Barvu můžete i převádět do modelů CMYK či HSV či ji takto nastavit, např. `c.hsva=(120, 100, 100, 255)` nastaví neprůhlednou sytou jasnou zelenou a `Color(255, 0, 0).cmy == (0, 1.0, 1.0)`. Podrobnosti viz manuál.

Interně jsou barvy reprezentovány podle nastavení systému (podle *barevné hloubky*, běžné jsou 8, 16, 24 nebo 32 bitů, každá v několika variantách). Všechny kreslící funkce barvy automaticky převádějí. Při vytvoření bitmapy si též můžete explicitně říci, jakou barevnou hloubku má mít, ale to už je spíše pokročilé použití a pro hříčky by mělo stačit s nastavení vašeho systému.

PyGame podporuje tři různé druhy průhlednosti bitmap, každá bitmapa zvlášť buď průhlednost nemá, nebo používá jeden z těchto tří druhů. Popíšeme jen nejuniverzálnější (a nejpomalejší) mód, kdy má každý bod bitmapy svou průhlednost jako součást barvy (A výše). Pokud vytvoříte novou bitmapu nebo načtete obrázek bez průhlednosti, můžete ji do bitmapy přidat zavoláním `s.convert_alpha()`.

Obdélníková oblast je reprezentována objektem `Rect`,

```
r = Rect(levy_horni_roh, velikost)
```

Informace pak získáte `r.x`, `r.y`, `r.w` a `r.h`, ale `Rect` toho umí i posun, ořezání, střed, zjištění kolize a mnoho dalšího.

Základní kreslící funkce jsou v modulu `pygame.draw`. Většina jich má jako první parametr Surface, na který se má kreslit, druhý bývá často barva. `line(s,`

`c, zbodu, dobodu, sirka=1)` kreslí čáru, `rect(s, c, Rect, sirka=0)` obdélník, `polygon(s, c, seznam_bodu, sirka)` mnohoúhelník, `circle(s, c, stred, polomer, sirka=0)` kruh a `ellipse(s, c, Rect, sirka=0)` elipsu.

Elipsa a obdélník mají vždy osy rovnoběžné s x a y. Pokud je `sirka==0`, jsou tvary vyplněné, jinak jsou nakresleny čarou dané šířky. Modul `draw` podporuje ještě vyhlazené tenké čáry pomocí `aaline`.

Rozšířené kreslící funkce jsou od verze 1.9 k dispozici v modulu

`pygame.gfxdraw`

který je potřeba zvlášť importovat. Navíc nabízí další vyhlazené kreslení (`aa...`) A také kreslení vyplněných tvarů (`filled_...`) a kreslení pomocí poloprůhledných barev (viz RGBA výše). Mají trochu jiné parametry než jejich příbuzní ve `draw`.

Mezi bitmapami je možné kopírovat obdélníkové oblasti. To je potřeba např. pokud si do pomocné bitmapy nahrajete obrázek, který chcete opakovaně zobrazovat, nebo při zobrazování textu. Příkaz `cil.blit(zdroj, kam, odkud=None, jak=0)` zkopíruje obdélník `odkud` (nebo celou bitmapu) z bitmapy `zdroj` do bitmapy `cil` na pozici `kam`. Parametr `jak` určuje speciální chování, např. `BLEND_ADD` přičte barevné hodnoty `BLEND_MIN`. `blit` respektuje průhlednost zdroje.

Bitmapa se vyplní (maže) pomocí `bitmapa.fill(barva)`), `bitmapa.copy()` vyrobí kopii.

K jednotlivým bodům (pixelům) bitmapy můžete přistupovat pomocí `s.get_at(pos)` a `s.set_at(pos, c)`.
Tyto dvě funkce jsou ale velmi pomalé, pokud byste je chtěli volat na každý pixel velké bitmapy[8].

Přímý přístup k datům bitmapy jako ke dvourozměrnému poli umožňují `PixelArray` a též modul `pygame.surfarray`. Práci s nimi ale komplikuje to, že přistupují ke skutečné reprezentaci barev, která se může lehce lišit počítač od počítače. Navíc je bitmapa, na které operují, zamčená po celou dobu jejich existence. Pokud potřebujete „rychlý“ přístup (do míry, jak je to v Pythonu možné), přečtěte si dokumentaci.

Načítání obrázků ze souborů formátů jpg, png, gif, bmp, pcx a dalších je možné pomocí `image.load(jmeno_souboru)`, který vrátí Surface. Zobrazit na screen pak musíte např. pomocí `blit`. Ukládat je možné ve formátech bmp, jpg a png pomocí `image.save(s, jmeno_souboru)`, formát se zvolí podle přípony.

Pro vykreslování textu je nejdříve potřeba vytvořit objekt typu `Font`, to je nejjednodušší pomocí `f=font.SysFont(jmeno, velikost, tucne=False, italika=False)`. Místo jména můžete použít `None`, seznam dostupných fontů vrátí `get_fonts()`.

[8] Hlavně kvůli opakovanému zamykání paměti a též na `screen`. Trochu pomůže bitmapu před přístupy zamknout `s.lock()` a pak odemknout `s.unlock()`.

Funkce fontu `f.render(text, vyhladit, barva, pozadi=None)` vrátí pomocný surface s vykresleným textem, pozadí `None` znamená průhledné (pak bude výsledná bitmapa částečně průhledná). Funkce `f.size(text)` je rychlý způsob jak zjistit, jak bude vykreslený text velký.

Transformace bitmap jsou v modulu `pygame.transform`. Většinou vytvoří a vrátí novou bitmapu s výsledkem, ale některé mají jako volitelný parametr cílovou bitmapu, která ale musí mít přesně správnou velikost.

Funkce `scale(s, velikost, cil=None)` rychle a hrubě zmenší či zvětší bitmapu, `smoothscale` je jemná varianta. Funkce `rotate(s, uhel)` hrubě rotuje, `rotozoom(s, uhel, velikost)` je jemná kombinace rotace a škálování. Funkce `flip(s, horiz, vert)` překlápí obraz kolem daných os. Zajímavá je ještě funkce `scale2x(s, cil=None)`, která velmi hladce zvětší obraz na dvojnásobek.

Události a ovládání

Interaktivní aplikace musí reagovat na události, jako stisky kláves či pohyby myší. K tomuto existují dva přístupy:

Jedna možnost je čekat na *události* a zpracovávat je jednu po druhé. Toto se velmi hodí hlavně když program reaguje na jednotlivé události, jako jsou klikání myší, stisky kláves či tiky časovače, a nezajímá vás, jak dlouho jsou klávesy stisknuty. Výhoda je, že vždy reagujete na jednu událost a program během čekání nespotřebovává výkon procesoru.

Druhá možnost je periodicky (několikrát za sekundu) testovat stav kláves a myši, pozměnit stav a třeba i vždy překreslit obrazovku. Takový přístup trvale zatěžuje procesor, ale výborně se hodí např. na hry, kde periodicky testujete, zda hráč stále tiskne šipku nebo pohybovat kočkou za kurzorem myši. Tento přístup má blízko k *aktivnímu čekání*. Nevýhodou je, že je potřeba pokaždé testovat všechny potenciálně zajímavé okolnosti, např. „už uběhlo dost času od posledního výstřelu?“

Samozřejmě je možné oba přístupy kombinovat. Dál popisujeme dvě sady mechanizmů pro zpracování událostí a aktivní čekání. Podobné mechanizmy se používají i v jiných prostředích, především události.

Události jsou v PyGame reprezentovány objekty `Event` z modulu `pygame.event`.
Tyto reprezentují události klávesnice, myši, okna (změna velikosti, zavření), joysticku, časovače a uživatelské události. PyGame má svou frontu událostí, ve které se „samy“ objevují události.

Objekt typu `Event` má vždy číselný atribut `type`, který určuje typ události. Některé typy spolu s dalšími atributy, které události tohoto typu mají:

`QUIT`		někdo chce zavřít okno aplikace
`KEYDOWN`	`unicode, key, mod`	stisk klávesy
`KEYUP`	`key, mod`	uvolnění klávesy
`MOUSEMOTION`	`pos, rel, buttons`	pohyb myši
`MOUSEBUTTONDOWN`	`pos, button`	stisk tlačítka myši

`MOUSEBUTTONUP`	`pos, button`	uvolnění tlačítka myši

Pozor na to, že `key` není znak, ale číselný kód klávesy odpovídající nějaké z konstant `K_...` , a `mod` je (bitová) kombinace stištěných modifikátorů z `KMOD_...` . `rel` je relativní pohyb kurzoru od poslední události a `buttons` je bitová kombinace stištěných tlačítek.

Krom výše uvedených jsou k dispozici ještě kódy mezi `USEREVENT` a `NUMEVENTS-1` (většinou 24 a 32).
Tyto události si můžete posílat sami nebo si můžete nastavit časovač, aby vám je posílal periodicky sám.

Další událost si z fronty vyzvednete pomocí `e=poll()` která vrací `None` pokud je fronta prázdná, `e=wait()` vrací událost a čeká na další, pokud je fronta prázdná, a `get()` vrátí celou frontu jako seznam. `peek(typy=None)` vrátí zda čekají nějaké události (jakékoliv nebo z daných typů).

Pomocí `set_blocked(typy)` a `set_allowed(typy)` je možné si vybrat, které události chcete dostávat. Na začátku jsou povoleny všechny typy.

Události je běžné zachytávat ve smyčce a zpracovávat jednu po druhé:

```
while True:
  e = event.wait()
  if e.type == QUIT or (e.type == KEYDOWN and e.key == K_ESCAPE):
    break
  if e.type == MOUSEMOTION:
    print("Mys se pohla o %s na %s !"%(e.rel, e.pos))
  if e.type == USEREVENT: # nas casovac 20x za sekundu
    pohni_figurkami_a_prekresli()
```

(Trochu jsme to zjednodušili – mohlo by se stát, že by překreslování trvalo více než 0,05 s a pak by se nám události hromadily. Pokud by to hrozilo, pomůže třeba kombinace `peek()` a `get()`. Viz též dále.)

Pokud chcete dostávat události periodicky, můžete si to objednat pomocí `time.set_timer(ms, udalost)`. Pak dostanete každých `ms` milisekund událost s typem číslo `udalost`, což musí být jedna z uživatelských událostí. Časovač zrušíte pomocí `time.set_timer(0, udalost)`.

Aktivní čekání opakovaně testuje stav vstupů. Mezi jednotlivými kroky se většinou čeká, aby k testování, změně stavu programu a překreslování docházelo zhruba stejně často. Je možné čekat vždy stejnou dobu (např. 0,05 s), ale to pak rychlost programu záleží na náročnosti vykreslování a zatížení počítače, což se může měnit. Lepší je zapamatovat si čas `t`, kdy se naposledy začalo vykreslovat, a s dalším krokem počkat do času `t`+0,05 s (nebo nečekat vůbec, pokud to trvalo déle).

Funkce `time.get_ticks()` vrací počet milisekund od zavolání `pygame.init()`.
Čekání zajišťuje funkce `time.wait(milisekund)`. Funkce `time.delay(milisekund)`

je přesnější, ale čeká aktivně a celou dobu vytěžuje procesor.

Pokud sami nechcete zpracovávat události, je přesto potřeba jednou za čas (vždy před čtením stavu vstupů) zavolat některou z funkcí pro čtení fronty událostí (např. `event.get()`), a to i když vás události samotné nezajímají. Právě během těchto funkcí totiž PyGame obnoví informace o vstupních zařízeních. `event.get()` též vybere frontu a tím zabrání jejímu přeplňování[9].

Aktuální stav kláves vrací `key.get_pressed()` jako pole `0` a `1`, které se indexuje konstantami `K_...` , např. `key.get_pressed()[K_SPACE]`. Obdobně, funkce `pygame.key.get_mods()` vrací informace o stisku modifikátorů (např. *alt* a *shift*) jako pole indexované `KMOD_...` .

Při tomto přístupu se může občas stát, že program mine krátký stisk klávesy nebo tlačítka myši. Toto lze řešit událostmi, jak popisujeme níže. Taky pozor na to, že většina počítačů zvládá zaznamenat současný stisk jen pár kláves.

Funkce `key.get_focused()` vám ještě prozradí, zda je okno vašeho programu vybráno (zaměřeno), nebo zda uživatel přepnul na jiné okno.

Aktuální stav myši vracejí funkce `mouse.get_pressed()`, `mouse.get_pos()` a `mouse.get_rel()`, ta první jako bitovou kombinaci stištěných tlačítek, druhá a třetí pak pozici a změnu pozice (vůči poslednímu volání `get_rel`).

Funkce `mouse.set_cursor` umožňuje měnit kurzor myši v okně vašeho programu, detaily ale necháme na dokumentaci. Pro začátek se může hodit, že voláním `mouse.set_cursor((8,8),(0,0),[0]*8,[0]*8)` nastavíte průhledný kurzor myši.

Kombinace obou přístupů je samozřejmě možná – můžete např. 20-krát za sekundu obnovovat stav světa a vykreslovat změny, během toho zjišťovat, zda je šipka vpravo stále stisknuta a pokaždé vybrat všechny události a kontrolovat, zda nebyl stištěn *escape* či zda uživatel neklikl. Takto je napsána jedna z koster programu na našem webu.

Zvuk

Zvukové efekty, hudbu a hlasitost má na starost modul `pygame.mixer`.

Zvuk je v počítači uložen jako hodnoty v čase nasnímané s určitou vzorkovací frekvencí. Aby nedocházelo ke zkreslení, je potřeba mít zvukové soubory nahrané se stejnou frekvencí, s jakou bude inicializován `pygame.mixer`. Běžná hodnota, kterou zvládne snad každá zvuková karta je 44100 Hz, též používaná na audio CD. PyGame ale implicitně inicializuje přehrávání s frekvencí 22050 Hz. Před (!) voláním `pygame.init()` je ale možné zavolat

```
mixer.pre_init(44100)
```

a vybrat tak vyšší frekvenci.

PyGame má k dispozici několik kanálů typu `Channel`, kde každý kanál může najednou přehrávat jen jeden zvuk. Na začátku je to 8, ale jejich počet můžete zvýšit `set_num_channels(pocet)`.

[9] Další řešení je všechny typy událostí zakázat

Objekt `Sound` reprezentuje v paměti jeden zvuk. Nahrajete ho ze souboru pomocí zvuk=Sound(jmeno_souboru). PyGame podporuje především formát WAV (ideálně nekomprimovaný, 16-bitový se znaménkem, jak je běžné na audio CD), experimentálně pak OGG.

Zvuk přehrajete pomocí `kanal=zvuk.play(opakovat=0)`. Zvuk bude přehráván `opakovat+1`-krát, nebo opakovaně pro `-1`. Funkce `play` nečeká na přehrání celého zvuku a ihned vrátí kanál, na kterém je zvuk přehráván. Zvuk je možné pustit i několikrát najednou, každá hrající kopie pak bude mít vlastní kanál.

Pokud není žádný kanál volný, je některý už hrající zvuk zastaven a použit jeho kanál. Pokud potřebujete nad kanály lepší kontrolu, můžete přímo určovat, co se na kterém kanálu děje a používat `kanal.play(zvuk)`. Více viz manuál.

Zvuky je možné ovládat buď skrze objekt `Sound`, pak ovládáte všechna jejich současná i budoucí přehrávání, nebo skrz `Channel`, kdy ovládáte jednotlivé kanály a přehrávání „kopií“ zvuků.

Funkce `zvuk.stop()` zastaví přehrávání všech kopií daného zvuku. Funkce `zvuk.set_volume(hlasitost)` nastaví hlasitost zvuku mezi 0,0 a 1,0 pro běžící i budoucí přehrávání.

Jednotlivé kanály je možné ovládat `kanal.stop()`, `kanal.pause()`, `kanal.unpause()` a `kanal.set_volume(hlasitost)`.

Z dalších funkcí se může hodit `kanal.fade_out(delka_ms)` pro postupné utlumení a obdobný volitelný parametr `fade_in` funkce `play`. Zajímavá je ještě možnost objednat si u kanálu událost (`Event`) ve chvíli, kdy dohraje, pomocí `kanal.set_endevent(typ_udalosti)` (typ události je některý z uživatelských).

Modul `pygame.music` je specializované rozhraní `pygame.mixer`, které usnadňuje přehrávaní hudby. Pokud byste ho chtěli použít, obraťte se prosím na dokumentaci PyGame.

Tomáš

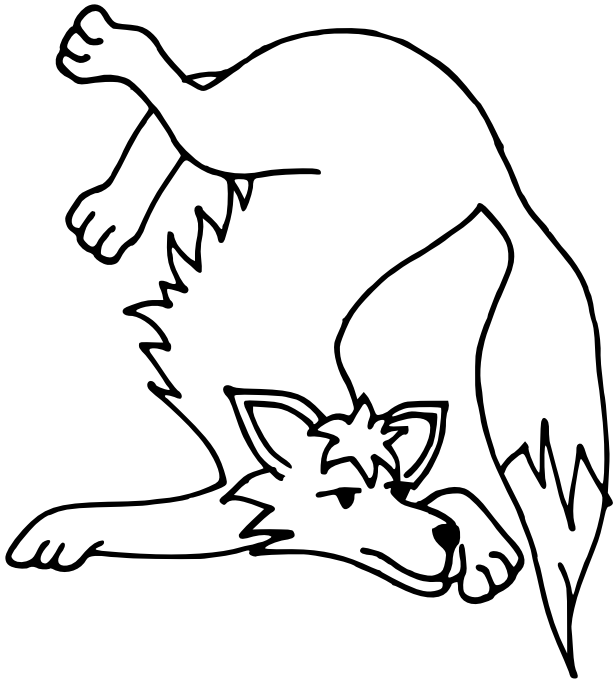
Úloha 4.5 – Seriál o Pythonu (IV. díl) (3+b)

Dnešní jedinou úlohou je napsat s pomocí PyGame jednoduchou hru nebo jiný grafický interaktivní program. Fantazii a ztřeštěnosti se meze nekladou.

Zadání odpovídá i bodování, které bude spíše jako bodování příspěvku k tématku, body ale dostanete i za jednoduché experimenty. (Dalo by se tedy říci, že úlohou je demonstrovat pochopení a použití PyGame.) Hodnotí se jak samotný program, tak nápad a provedení. Rozhodně si nelamte hlavu se složitou grafikou a hezkou hudbou, úplně postačí třeba jen mnohoúhelníky. Dobře poslouží i fotokoláž nebo dobře umístěný text.

Vzhledem k tomu, že jde o bodovanou úlohu, se snažte používat hlavně vlastní kód. Použití kusů cizího kódu není zakázáno, ale pak nám napište, které části programu jsou vaše, abychom je mohli zhodnotit. Samozřejmě můžete použít kostru programu z našeho webu.

Pokud nám to dovolíte, moc rádi vaše výtvory zveřejníme.



Konference Dolní Bečva 2010

Genetické programování

Prof.MM Petr Pecha

Úvod do problému

Genetické programování je způsob programování, který využívá metody podobné biologické evoluci. Nejprve se náhodně vygeneruje výsledek (0-tá generace) a pak se postupnými úpravami mění – vyvíjí. Tento vývoj je řízen přirozeným výběrem (přežijí jen nejsilnější, v našem případě ta řešení, která jsou lepší). Při řešení problému genetickým algoritmem nedostáváme přesný výsledek, ale pouze se k němu blížíme. Jak moc se přiblížíme, závisí na tom, jak dlouho program poběží a také, jak nastavíme přirozený výběr.

Používá se tam, kde nelze matematicky odvodit závislosti (např. složitější grafika), nebo u problémů s velkou časovou složitostí, kde nepotřebujeme přesný výsledek.

Řešení problému obecně

Nejprve musíme definovat problém a poté vědět, v jakém tvaru hledáme výsledek. Většinou se jedná o pole (i vícerozměrné) hodnot. Takovému poli budeme říkat **genetická informace**.

Abychom měli různé genetické informace jak porovnávat, musíme mít **hodnotící funkci**. Tato funkce vrací hodnotu, která udává, jak je která genetická informace „dobrá“.

Popis algoritmu:

1. Nejprve si vyrobíme několik genetických informací. To provedeme náhodně.
2. Nyní si ohodnotíme každou informaci (předáme ji hodnotící funkci a to, co vrátí, si zaznamenáme).
3. Genetické informace seřadíme podle jejich kvality.
4. Nějakou část informací **zabijeme** (typicky několik posledních, ale není to vždy nutné. Lze brát třeba i jednu náhodnou).
5. Náhodné informace **křížíme** a jiné **mutujeme**.
6. Pokračujeme krokem č. 2. Každým průchodem cyklu se nám výsledek nějak deformuje. Pokud nezabijeme nejlepšího, tak nový výsledek je vždy lepší nebo stejný.
7. Pokud další provádění výsledek nezlepšuje, je dobré spustit program znovu (bod č. 1).

V postupu jsme se zmínili o dvou neznámých pojmech: křížení a mutace.

Křížení je postup, kdy vzniká nová genetická informace. Jak už název napovídá, vyrábíme ji z několika již používaných genetických informací. Jednotlivé buňky nového pole skládáme z náhodné genetické informace, kterou již máme. Při tomto procesu zachováváme pořadí buněk.

Mutace je děj, při kterém genetická informace nevzniká ani nezaniká. Pouze se stávající změní na náhodných místech o náhodnou hodnotu.

Pokud bychom prováděli jen křížení, tak se nám geny tzv. „vyčerpají“. Mutace přispívá k lepšímu výsledku a oživuje genetické informace.

Pro úplnost zde uvádím i **zabíjení**. Je to proces, kdy program rozhodne, že daná genetická informace je špatná, vyřadí ji ze svého seznamu a již dále s ní nepracuje.

Problém zbloudilého velbloud

Pro naši práci jsme si zvolili problém, který má najít univerzální posloupnost příkazů pro velbloud.

Na vstupu máme N map, reprezentovaných polem velikosti $R \times S$. Počáteční pozice velbloud je $\left(\frac{R}{2}, \frac{S}{2}\right)$. Každé pole mapy obsahuje buď 1 (kaktus (pole, na které velbloud nemůže vstoupit)) nebo 0 (poušť (volné pole pro pohyb)).

Pohyb velbloud. Velbloud má v sobě zabudovaný zvláštní orientační smysl, proto pozná, kterým směrem jsou světové strany. Mapa je orientována rovnoběžně se směry světových stran, horní hrana je severní.

Genetická informace velbloud. Velbloud má v sobě posloupnost instrukcí popisujících jednotlivé kroky. Každou danou instrukci se snaží provést (vydá se daným směrem), ale pokud mu stojí něco v cestě (většinou kaktus), tak se vrátí na výchozí pozici (nepohne se). Jakmile dojdou velbloudovi instrukce, zastaví se a zhodnotí, kolik toho ušel. O tom, jak je dobrý, rozhodne hodnotící funkce.

Hodnotící funkce. Po velbloudovi chceme, aby došel co nejdál od počátečního bodu. Nepotřebujeme vědět kolik ušel, protože když s námi udělá okružní jízdu, a pak se vrátí zpět, tak nám údaj není k ničemu (nikam nás nedovezl).

Vzdálenost počítáme v New-Yorské metrice. To znamená, že pokud je počáteční poloha $[x_1, y_1]$ a koncová $[x_2, y_2]$, pak vzdálenost, kterou ušel, je rovna $[|x_1 - x_2| + |y_1 - y_2|]$.

Řešení problému

Program jsem psal v jazyce C. Používal jsem čtyři mapy o velikosti 60 x 60. Jednu mapu jsem měl uloženou v souboru a zbylé tři jsem dostal rotací.

Počet velbloudů jsem nastavil na 100 a délku jejich instrukcí na 150.

Vždy jsem velbloudy ohodnotil hodnotící funkcí. Tato funkce vrátí určitou hodnotu pro každou ze čtyř map. Výsledné hodnocení je součet všech čtyř výsledků. Nakonec velbloudy setřídím podle toho, jak jsou dobří (od nejlepšího po nejhorší).

Zvolil jsem si dvě hranice: **ELITA** je prvních n velbloudů a **ODPAD** je posledních m velbloudů.

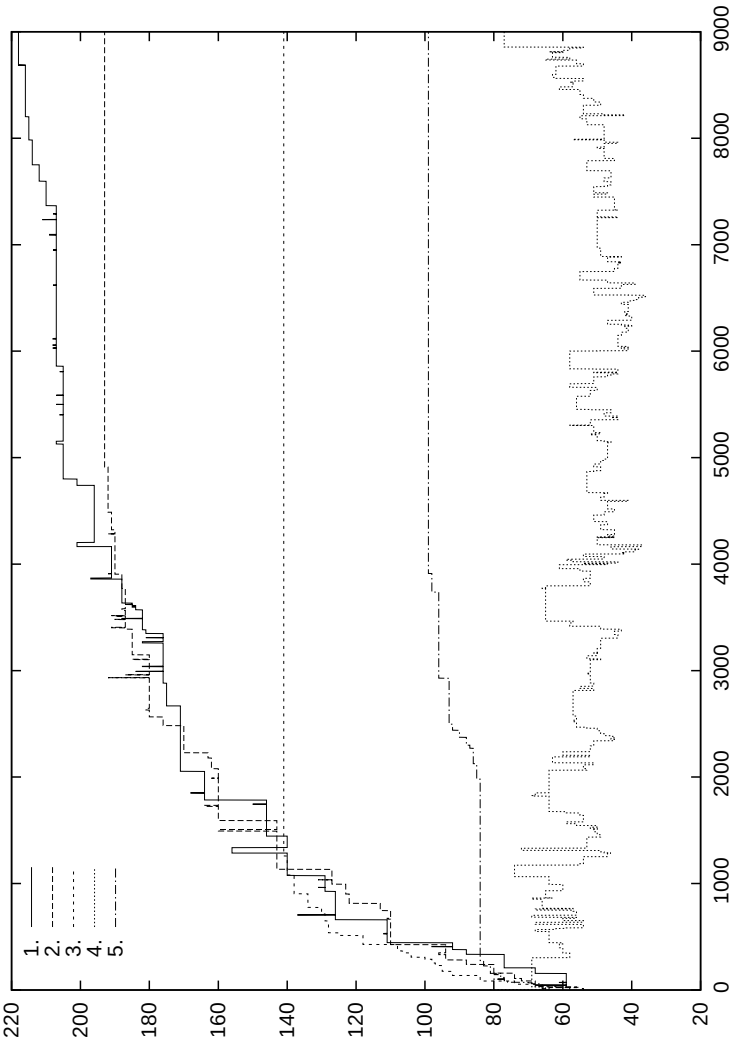
V každém cyklu vždy dva velbloudy zabiji a to ze skupiny ODPAD, jednoho vyrobím mutováním a druhého křížením. K mutování a křížení používám skupinu velbloudů ELITA. Počet cyklů jsem stanovil na 9000 a zkoušel jsem různá n a m. Průběh jsem si zaznamenával jako hodnotu nejlepšího velbloud a data jsem zakreslil do grafu.

V grafu k1.1 ukazuje křivka číslo pět situaci, kdy program pro srovnání zabíjí náhodně ze všech velbloudů, bez ohledu na hodnotící funkci.

Legenda:

Č. dat	1	2	3	4	5
ELITA	70	75	10	100	2
ODPAD	20	25	20	100	2

Pozn. red.: Zdrojový kód k programu najdete na našich stránkách v sekci „Z řešení“.



Obr. k1.1

Konference Krásno 2009

Měření Planckovy konstanty

Doc.MM Alena Bušáková, Dr.MM Filip Hlásek,
Bc.MM Martin Holeček

Úvod

Planckova konstanta je jednou ze základních fyzikálních konstant. Mimo jiné se uplatní ve vztazích částicově-vlnového dualismu nebo ve vyzařovacím zákoně černého tělesa. Naším úkolem je změřit její číselnou hodnotu pomocí běžných měřících přístrojů.

Trocha historie

Na přelomu 19. a 20. století se vědci zabývali zářením absolutně černého tělesa (tj. ideálního tělesa, které neodráží žádné světlo, které na něj dopadá) a experimentálně[10] zjistili, že celková intenzita jeho záření odpovídá čtvrté mocnině teploty tohoto tělesa.

Zjistilo se také, že světlo je vyzařováno po kvantech, tzn. $E = f \cdot \text{konstanta}$. Vědci už také věděli, že světlo jsou elektromagnetické vlny, resp. se tak domnívali na základě experimentů. Max Planck konstantu pod označením b zavedl roku 1899. Až později jí byl přisouzen nejen matematický, ale i fyzikální význam. Roku 1905 pak Albert Einstein dokázal, že energie záření opravdu (byť se nám to logicky může zdát divné) neodpovídá jeho intenzitě, ale že $E = hf$, kde h je Planckova konstanta (toto označení se pro ni používá dodnes) a f je frekvence světelné vlny.

Pojmy

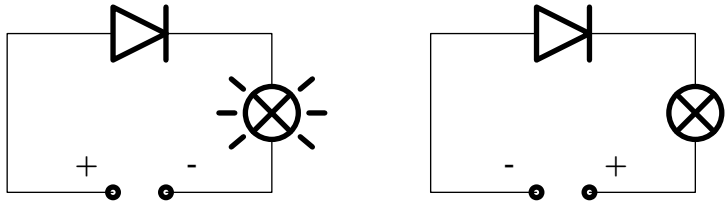
Nyní objasníme několik pojmů, které jsou potřeba pochopit, neboť jsme faktů s nimi souvisejících používali při měření.

Polovodič je materiál, jehož elektrická vodivost závisí na vnějších[11] podmínkách, tudíž se jeho vodivost změnou těchto podmínek (tzn. dodáním tepelné, elektrické nebo světelné energie) dá snadno ovlivnit. Mezi polovodiče patří například prvky `Si`, `Ge` nebo `Se` a sloučeniny jako `PbS`. Polovodiče se využívají u elektronických součástek (dioda, fotorezistor). Existují polovodiče typu `N` a `P`. Typ `N` obsahuje volné elektrony, `P` má více děr („prázdných míst po elektronech“).

[10] Samozřejmě žádné absolutně černé těleso neměli, takže pro účel experimentu použili duté těleso s velmi malým otvorem, které se pak jako absolutně černé těleso docela dobře chová.

[11] Vodivost polovodiče závisí také na vnitřních podmínkách, jako jsou různé příměsi v materiálu.

U *fotoelektrického jevu* dochází při dopadu elektromagnetického záření na materiál k uvolňují elektronu, a to buď vně látky (vnější fotoelektrický jev), nebo elektron zůstává uvnitř materiálu jako vodivostní elektrony (vnitřní fotoelektrický jev, při dodání světelné energie polovodiči). Existuje i opačný proces, např. u diody, kdy se při průchodu elektronů materiálem uvolňují fotony.
Dioda je elektrická součástka, které obsahuje `PN` přechod. Při zapojení do obvodu jí teče proud jen jedním směrem, a to při zapojení polovodiče typu `P` ke kladnému pólu zdroje. Tomuto směru zapojení se říká propustný směr a dochází zde ke kumulaci děr a elektronů v místě přechodu, což znamená, že proud může protékat. Opačný směr zapojení je závěrný, elektrony se oddálí od děr a proud neprotéká.



Obr. k2.1 – Zapojení diody v propustném a závěrném směru.

Měření

Naše měření probíhalo s využitím voltampérové charakteristiky, což je závislost proudu na napětí. Měřili jsme proud diodou a napětí na diodě, neboť tato součástka má tu vlastnost, že když na ní zvyšujeme napětí, proud jí nějakou dobu neprotéká (resp. jen nějaká konstantní minimální hodnota, která protéká i v závěrném směru), až při určité mezní hodnotě napětí $U_{\min}$ začne proud obvodem protékat, přičemž jeho růst je zpočátku exponenciální. A tato mezní hodnota byla právě to, co jsme měřili. Měření jsme prováděli na červené a žluté LED diodě. V tabulce naměřených hodnot jsou zvýrazněny ty, které v grafu odpovídají místu skoku, kde začal obvodem protékat proud.

Výpočet

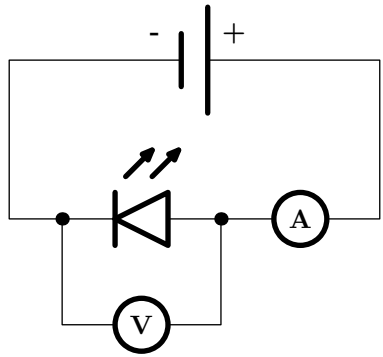
Použili jsme následující vzorce

$$E = U \cdot e\,,$$

$$f \cdot \lambda = c\,,$$

$$E = h \cdot f\ (\text{Planck} - \text{Einsteinova rovnice}),$$

kde E je energie, U je velikost elektrického napětí, e je elementární náboj, c je rychlost světla, h je Planckova konstanta, f je frekvence vyzařovaného světla a λ je jeho vlnová délka.



Obr. k2.2 – Schéma zapojení měřeného obvodu.

U [mV]	I [μA]	U [mV]	I [μA]
1253	1,5	1444	2,6
1295	1,5	1502	2,8
1300	1,5	1525	3,6
1307	1,55	1548	5,1
1315	1,6	1587	11,1
1373	1,7	1640	40,7
1414	1,8		

Tabulka k2.1: V–A charakteristika žluté diody.

Z nich jsme vyjádřili

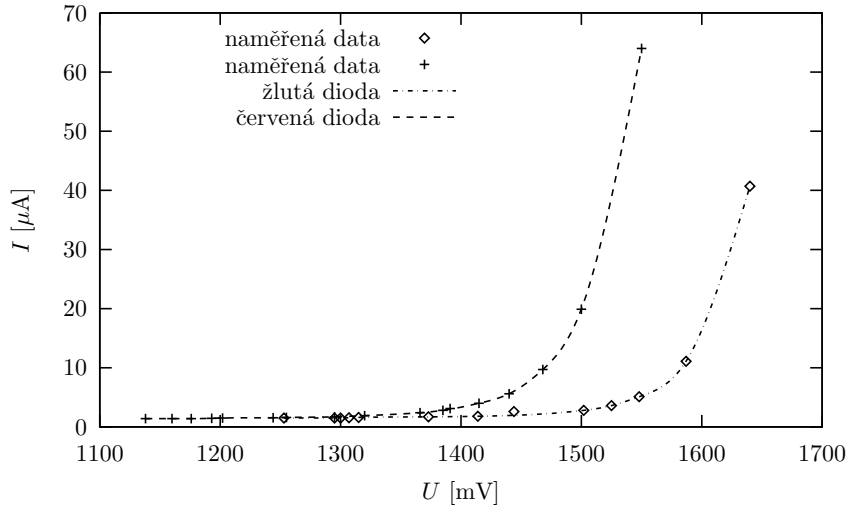
$$h = \frac{U \cdot e \cdot \lambda}{c}\,.$$

Po dosazení konstant, délek vlnění námi použitých diod a naměřených hodnot vyšlo pro žlutou diodu $h = 4,13 \cdot 10^{-34}$ Js a pro červenou diodu $h = 3,94 \cdot 10^{-34}$ Js. Vzhledem k hrubé nepřesnosti našich měřících přístrojů[12] chybu neodhadujeme, ale vychází to řádově dobře. Skutečná hodnota Planckovy konstanty je $h = 6{,}6261 \cdot 10^{-34}$ Js.

[12] Potřebovali jsme pomaličku zvyšovat napětí zdroje, abychom byli schopni poznat prahové napětí, ale zdroj nebyl schopen zvyšovat napětí po dostatečně malých krocích, navíc měření napětí i proudu bylo o řád méně přesné, než by bylo potřeba.

U [mV]	I [μA]	U [mV]	I [μA]
1138	1,4	1366	2,4
1160	1,4	1385	2,8
1176	1,4	1391	3,1
1193	1,45	1415	4
1202	1,5	1440	5,6
1244	1,55	1468	9,7
1255	1,6	1500	19,9
1295	1,7	1550	64
1320	1,9	1600	250

Tabulka k2.2: V–A charakteristika červené diody.



Obr. k2.3 – Naměřené V–A charakteristiky diod.

Závěrem

Tato konfera nebyla příliš náročná ani co do teoretických úvah, ani co do provedení pro zručného fyzika. Experimentálně jsme však ověřili platnost některých základních rovnic kvantové fyziky. Konfera byla také hodně o spolupráci mezi těmi zdatnějšími z nás a těmi, kteří voltmetr zapojovali se značným respektem, aby ručička nevylétla z přístroje, a kteří před puštěním proudu do obvodu s diodou dlouze přemýšleli, jestli náhodou není zapojena naopak. Většinu

práce zabralo hraní si s nepříliš citlivým knoflíkem zdroje, aby se jeho napětí zvyšovalo dostatečně jemně. Seznámili jsme se také s principem voltampérové charakteristiky a zjistili jsme, že i s velmi jednoduchým vybavením můžeme Planckovu konstantu změřit alespoň řádově přesně.

Malé zamyšlení nad jednotkami

V návaznosti na předchozí fyzikální článek mi dovolte malé, trochu nevážné, zamyšlení nad fyzikálními jednotkami. V některých oborech fyziky se stále můžeme setkat s jednotkami, které nespadají do soustavy SI. Příkladem může být vakuová fyzika, kde se občas stane, že při jednom experimentu máte jedno měřidlo cejchované v pascalech, druhé v torrech[13] a třetí třeba v psi[14].

Délkové jednotky	
čárka	$2{,}195 \cdot 10^{-3}\,\mathrm{m}$
ječné zrno	$\approx 3{,}4 \cdot 10^{-3}\,\mathrm{m}$
krok	$0{,}5914\,\mathrm{m}$
loket	$59{,}140 \cdot 10^{-2}\,\mathrm{m}$
pěst	$98{,}56 \cdot 10^{-3}\,\mathrm{m}$
Hmotnostní jednotky	
hřivna	$0{,}257\,\mathrm{kg}$
kámen	$10{,}275\,\mathrm{kg}$
Plošné jednotky	
dědina	asi (4 až 18)$\cdot 10^4\,\mathrm{m}^2$
role	asi $6100\,\mathrm{m}^2$
záhon	asi $400\,\mathrm{m}^2$
země	asi (2 až 9)$\cdot 10^4\,\mathrm{m}^2$
Objemové jednotky	
čarka (kalíšek)	$0{,}123 \cdot 10^{-3}\,\mathrm{m}^3$
lahvice, flaška	$5{,}8 \cdot 10^{-3}\,\mathrm{m}^3$
žejdlík	$0{,}484 \cdot 10^{-3}\,\mathrm{m}^3$

Tabulka j.1: Příklady staročeských jednotek, zdroj [1].

Jak by naše jednotky mohly vypadat, kdybychom podobně jako v USA či Velké Británii nepřešli na soustavu SI a zachovali bychom si původní jednotky? Hustotu bychom mohli měřit v kamenech na žejdlík, rychlost bychom měřili třeba v krocích za vteřinu a pokud bychom si vypůjčili jednotku „saco de azúcar“, což je Kubánská jednotka, která v překladu znamená pytel cukru a odpovídá 115,023 kg, tak bychom mohli tlak měřit v pytlích cukru na dědinu. (Upozornění: řešení zatím stále přijímáme pouze v jednotkách SI.) `:-)`

Použitá literatura

[1] V. Šindelář, L. Smrž: Nová soustava jednotek, SPN, Praha 1968.

(R)adim

[13] Torr je definován jako milimetr rtuťového sloupce 1 torr $\doteq$ 133,3 Pa

[14] Psi je definováno jako libra na čtvereční palec 1 psi $\doteq$ 6894,8 Pa

Výsledková listina

Poř.	Jméno	R.	$\sum_{-1}$	Číslo u1	u2	u3	u4	t3	t4	k	s2	+	$\sum_{1}$
1.	Prof.MM P. Pecha	4.	226	0	2	4	0			8	3	0	33
2–4.	Doc.MM A. Bušáková	4.	128	1		4	2			5		0	28
	Mgr.MM J. Kubečka	3.	28	1			2	6	3		2	0	28
	Prof.MM Š. Šimsa	2.	236	4		1	2				3	1	28
5.	Mgr.MM J. Sopoušek	4.	27	4	3	2	2				2	0	27
6.	Doc.MM F. Štědronský	4.	103				2				7	0	25
7.	Dr.MM T. Pokorný	4.	76	1		1					6	0	24
8–9.	Bc.MM V. Sedláček	3.	19	2			2					0	19
	Bc.MM J. Setnička	4.	19			4					3	0	19
10.	Bc.MM M. Töpfer	3.	16									0	16
11.	Mgr.MM A. Harlenderová	3.	40	4			2					0	14
12–13.	Bc.MM L. Dung	1.	13									0	13
	Dr.MM T. Kubelka	3.	91									0	13
14–17.	Bc.MM L. Grund	1.	12									0	12
	Mgr.MM M. Kocián	4.	43									0	12
	Bc.MM R. Kubíček	2.	12	4								0	12
	Bc.MM J. Novotná	2.	12									0	12
18–22.	Bc.MM E. Gocníková	3.	11									0	11
	Bc.MM B. Mólová	3.	11									0	11
	Bc.MM J. Svoboda	2.	11	4								0	11
	Dr.MM L. Zavřel	4.	59									0	11
	Mgr.MM K. Zemková	3.	25	4								0	11
23–27.	Bc.MM M. Bílý	4.	10									0	10
	Bc.MM J. Bok	3.	14									0	10
	Dr.MM F. Hlásek	4.	61							5		0	10
	Dr.MM M. Kochmanová	4.	79									0	10
	Bc.MM F. Lux	4.	10									0	10
28.	Mgr.MM J. Škoda	4.	37									0	9
29–31.	Dr.MM M. Bekrová	4.	61									0	8
	Mgr.MM O. Cífka	2.	21									0	8
	D. Tělupil	3.	8									0	8
32–36.	Mgr.MM B. Böhmová	3.	37									0	7
	S. Havadej	4.	7									0	7
	Mgr.MM G. Kubíčková	4.	21									0	7
	L. Langerová	1.	7									0	7
	B. Said	3.	7	4								0	7
37–39.	O. Fiedler	4.	7									0	6
	K. Kohoutová	2.	6									0	6
	M. Kopf	3.	6									0	6

Poř.	Jméno	R.	$\sum_{-1}$	Číslo u1	u2	u3	u4	t3	t4	k	s2	+	$\sum_{1}$
40–43.	V. Kletečka	3.	7									0	5
	P. Kubincová	4.	5			3	2					0	5
	M. Landa	1.	5									0	5
	M. Poppr	1.	5									0	5
44–46.	Bc.MM O. Mička	2.	10									0	4
	P. Vincena	1.	4									0	4
	D. Vít	1.	4									0	4
47–50.	E. Bušáková	2.	3									0	3
	J. Erhart	1.	3									0	3
	J. Kadlec	1.	3									0	3
	S. Ondrčková	2.	3									0	3
51–53.	L. Jančařík	4.	2	2								0	2
	Bc.MM P. Kratochvíl	3.	10									0	2
	O. Krčmář	2.	2									0	2
54–55.	O. Darmovzal	1.	1									0	1
	J. Křivonožka	3.	1									0	1
56.	V. Václavík	1.	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, sloupeček k jsou body za konferenční příspěvky a s jsou body za řešení úloh k seriálu. 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.