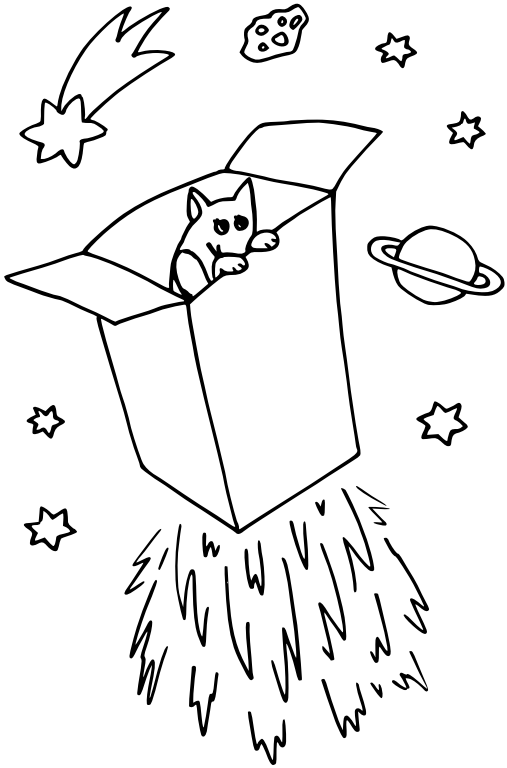
Studentský matematicko-fyzikální časopis

M&M

Ročník XXIII / Číslo 6



Zadání úloh 6. série – str. 3 • Řešení úloh 4. série – str. 5
Téma 1: Sluneční hodiny – str. 15 • Téma 2: Časosběr – str. 16
Téma 3: Filmoví poradci – str. 17
Mgr.MM Zuzana Urbanová: Tom, Jerry a fyzika – str. 17
Dr.MM Ondřej Knopp: Gravitace ve Flatlandu – str. 19
Seriál: Malý úvod do assembleru – str. 25

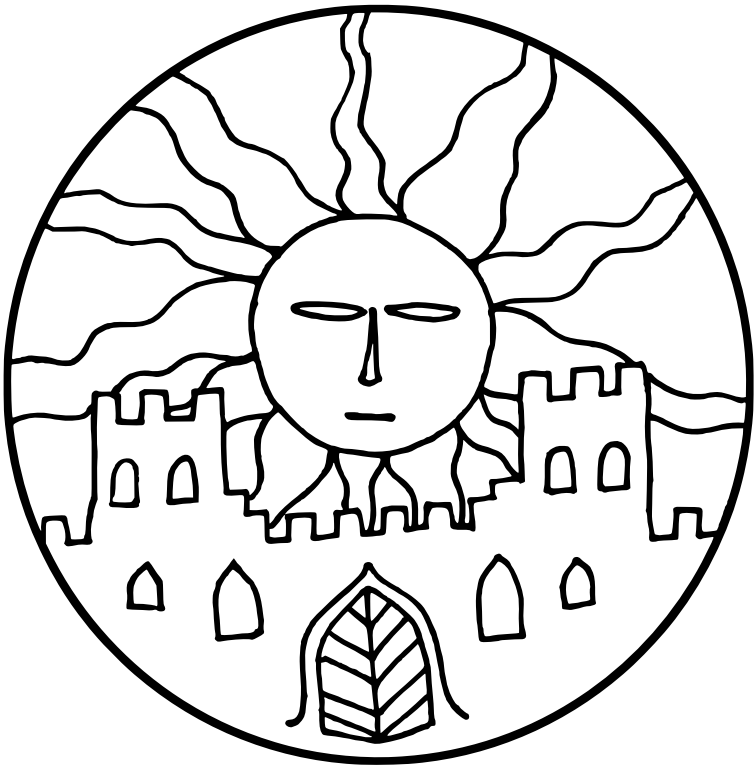
Časopis M&M a stejnojmenný korespondenční seminář je určen pro studenty středních škol, kteří se zajímají o matematiku, fyziku či informatiku. Během školního roku dostávají řešitelé zdarma čísla se zadáním úloh a témat k přemýšlení. Svá řešení odesílají k nám do redakce. My jejich příspěvky opravíme, obodujeme a pošleme zpět. Nejzajímavější řešení otiskujeme.

Milý čtenáři,

jaro nám už ťuká na okno, stejně jako nové číslo časopisu M&M! V něm je další článek o assembleru, zabývající se pamětí, spolu s dvěma novými úlohami, které ti mohou pomoci k cenným bodům. Dále číslo obsahuje pěkný příspěvek k tématku Filmoví kritici a sepsaný článek o konfeře Gravitace ve Flatlandu od Dr.^{MM} Ondřeje Knoppa. Těšíme se na další vaše články, kterým položily základ konfery na posledním soustředění.

My o soustředění a soustředění o vlkodlaku. Týdenní setkání organizátorů a řešitelů na Šumavě v městě Ankh-Morpork se vydařilo, organizátoři si to velmi užili a z výsledků anket se zdá, že účastníci také. Život pojišťováka ve středověkém městě skýtá mnohé zajímavosti, jak jsme se všichni přesvědčili. Okomentované fotografie se objeví v následujících dnech na stránkách semináře.

Vaši organizátoři



Zadání úloh

Termín odeslání 6. série: 9. 5. 2017

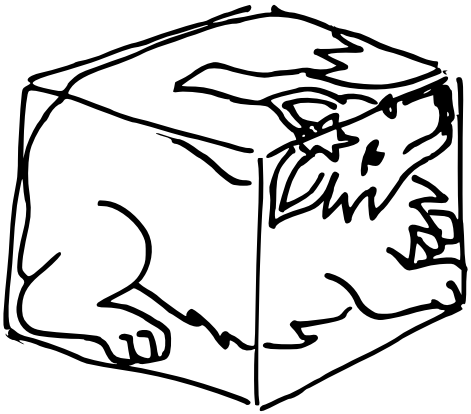
Bylo to ráno jako každé jiné. Fonda vstal z postele a šel si vyčistit zuby. Z koupelny však odcházel poměrně rozmrzelý, protože mu přestal fungovat generátor tíhového zrychlení, a tak měl vousy plné pasty. Bydlet na asteroidu se mu už dávno zajídalo, ale zdejší nemovitosti byly natolik cenově dostupné, že si zde mohl dovolit koupit vlastní pozemek i začínající mezigalaktický logistik. Navíc mu zcela vyhovovalo, že měl na celém asteroidu jediného souseda.

Otřel si fousy kapesníkem, zadržel ranní moč a nasedl do své mezigalaktické krabice, aby se dostavil do práce. Možná i díky silné ranní kávě byl natolik roztržitý, že si zapomněl zapnout bezpečnostní pás. Chvíli po startu si uvědomil, že se vzdaluje od řídicího panelu.

Úloha 6.1 – Vesmírná krabice (4b)

Fonda sa nachází na souřadnicích x_0 ve své vesmírné lodi o tvaru kvádru a má počáteční rychlost $|v_0| = 1\,\mathrm{m}\cdot\mathrm{s}^{-1}$ vůči lodi. Fonda umí ovládat motory lodě, které poskytují maximální tah do každého směru. Jiným způsobem se však uvnitř lodi pohybovat neumí.

Jaká je minimální hmotnost paliva, kterou spotřebuje na pohyb lodi, aby se dostal na původní místo x_0 a zůstal tam v klidu stát? Od stěn se umí Fonda odrážet s dokonalou pružností, stěny jsou dokonale hladké a loď je prázdná. Počítejte pro raketový pohon se specifickým impulzem $I_{sp} = 450\,\mathrm{s}$ a hmotnostním tokem $\dot{m}_R = 200\,\mathrm{kg}\cdot\mathrm{s}^{-1}$ a iontový motor s $I_{sp} = 3000\,\mathrm{s}$ a $\dot{m}_I = 3\cdot 10^{-5}\,\mathrm{kg}\cdot\mathrm{s}^{-1}$. Loď má hmotnost $M = 100\,\mathrm{t}$ a můžeme zanedbat změnu její hmotnosti. Jaký je maximální tah jednotlivých motorů ?



Fonda si oddechl, chopil se řídicího panelu a chystal se pokračovat v letu. Velmi se však vyděsil při pohledu na obrazovku mezigalaktické krabice. V průběhu jeho cesty zpět k panelu se krabice vychýlila ze svého původního směru a nyní se šinula rychlostí 0,8*c k odbočující krabici. Jelikož srážka byla zjevně nevyhnutelná, Fonda zaujal obrannou pozici doufaje v přežití. Před očima mu proběhl jeho krátký a osamělý život. Zanedlouho poté se krabicí rozezněla dunivá rána. Fonda otevřel oči, zkontroloval si svých pět končetin a byl dlouho v šoku. Ze šoku se probral až ve chvíli, kdy pilot protiletící krabice zazvonil na přetlakovou komoru. Když ho Fonda pustil dovnitř, pilot po něm chtěl, aby vypracoval vesmírný srážkový protokol a uznal vinu za nehodu v plném rozsahu.*

Úloha 6.2 – Nerovná podlaha (4b)

V mezigalaktické krabici je po srážce nerovná podlaha. Pro vyplnění formuláře na ni potřebují Fonda s pilotem postavit čtvercový stůl. Ale když chtějí stůl na podlahu položit, tak mu většinou jedna noha visí ve vzduchu a stůl se kýve, což značně znesnadňuje vyplnění formuláře. Dokážete na zvlněné podlaze vždy najít místo, ve kterém se všechny čtyři nohy stolu dotýkají podlahy?

Předpokládáme, že nám nevadí, že stůl není vodorovně, že podlaha je dost velká na to, aby se po ní dalo stolem pohodlně otáčet, a navíc v ní nejsou díry. Formálně chceme najít čtyři body tvořící daný čtverec, které leží v jedné rovině.

Po dlouhém dohadování si zkopírovali formuláře, rozloučili se, popřáli si úspěšný let a oba se vydali vlastním směrem. „V tomhle měsíci už třetí srážka, to budu muset brzy pořídit novou krabici," pomyslel si Fonda při cestě do práce. Zbytek cesty se naštěstí obešel bez větších problémů a tak Fonda bezpečně dorazil na planetu C628G4A. Při pohledu na atomové hodiny se však velmi polekal. „Ale ne, schůze začala už před hodinou, to mi určitě v mé nepřítomnosti zase přidělili nejtěžší práci. Nu což, díky tomu, že se za ni platí víc, si třeba dříve vydělám na novou krabici nebo alespoň nový generátor tíhového zrychlení," řekl si Fonda.

„Matissi, už počtvrté tento měsíc jdete pozdě. Nejdříve rozbitý humanoid, potom vyteklá pračka, pak bouračka, co to bude teď?" pravil pan Hokozuto. Po krátké omluvě Fondovi šéf přidělil následující úkol:

Úloha 6.3 – Doplnění grafu (5b)

Na začátku máme n osídlených planet tvořících planetární uskupení, mezi kterými vedou přímá meziplanetární spojení. Osídlení v galaxii se však zahušťuje a s růstem populace musí držet krok i dopravní infrastruktura. Operací asimilace nazveme začlenění nějaké osídlené planety do našeho planetárního uskupení a následné vybudování libovolně mnoha přímých meziplanetárních tahů mezi touto planetou a libovolnými jinými již osídlenými planetami z našeho uskupení.

Pomocí operace asimilace získejte planetární uskupení, kde z každé planety vede přesně n přímých meziplanetárních tahů. Navíc

a) Operaci asimilace můžete použít maximálně n-krát.

b) Operaci asimilace můžete použít kolikrát chcete, ale velikost největší planetární unie ve výsledném planetárním uskupení musí být stejná jako velikost největší planetární unie v původním planetárním uskupení. Planetární unií rozumíme skupinu osídlených planet, u kterých o každých dvou planetách platí, že mezi nimi existuje přímý meziplanetární tah.

Po náročném dni v práci si Fonda oddychl a vyrazil vstříc svému koníčku. Jako každý čtvrtek jde trénovat kolmio. V tomto sportu hrají dvě družstva proti sobě s deseti tyčemi různých délek. Vyhrává to družstvo, které dokáže z těchto tyčí rychleji vybrat tři tyče a postavit z nich trojúhelník.

Úloha 6.4 – Pečlivě schovaný trojúhelník (2b)

Délky tyčí (v metrech) ve hře kolmio nabývají hodnot z reálného intervalu $(1; 55)$. Dokažte, že hra může skončit vítězstvím některého z týmů pro libovolných deset tyčí s délkami z daného intervalu, tedy že z daných deseti tyčí je možné vybrat tři tak, že tvoří strany nějakého trojúhelníka.

Po náročném tréninku Fonda Matiss nasedl do své mezigalaktické krabice, zapnul autopilota a za tichého chrochtání krabice usnul, připravený čelit dalšímu dni.

Řešení úloh 4. série

Úloha 4.1 – Termoska (3b)

Zadání:

Mějme termosku tvořenou uzavřenou vnitřní a vnější plechovou stěnou, mezi kterými je vakuum. Jediný významný mechanismus přenosu tepla zevnitř ven je tepelným zářením přes evakuovanou mezeru.

Termoska je naplněna horkou kapalinou, která postupně chladne. Chladla by pomaleji, pokud bychom za jinak stejných podmínek vložili doprostřed vakuové mezery další uzavřenou „slupku“ z tenkého plechu ze stejného materiálu jako stěny nádoby, která nebude mít tepelný kontakt ani s vnější, ani s vnitřní stěnou? Pokud ano, jak moc pomaleji? Co když takových navzájem se nedotýkajících slupek přidáme více?

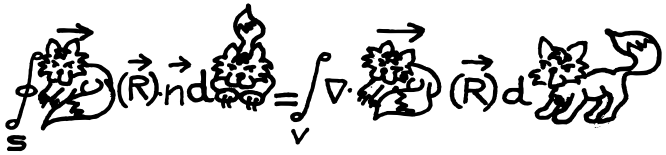
Můžete předpokládat, že teplota vnější stěny termosky je prakticky stejná jako teplota okolí, a nebude tedy záviset na kvalitě izolace. Přidávané vrstvy jsou natolik tenké a jejich vlastní tepelná kapacita tak nízká, že během zanedbatelně krátké doby dosáhnou rovnovážné teploty, a stačí tedy uvažovat jen tento ustálený stav.

Řešení:
Ačkoliv spočíst konkrétní množství tepla přenášené ven skrz stěnu termosky by bylo značně složité až nemožné, odpovědět na to, jakou relativní změnu způsobí přidání další stěny jde jednoduše jen na základě vcelku elementárních úvah a několika předpokladů.

Předně, každý povrch vyzařuje tepelné záření. Výkon vyzářený jednotkou plochy je závislý jen na teplotě a materiálu povrchu (dalo by se říci „zobecněné barvě“, která zahrnuje nejen viditelnou část, ale i zbytek spektra). Protože každá slupka termosky má vnitřní i vnější povrch ze stejného materiálu a oba povrchy jedné konkrétní slupky mají stejnou teplotu, bude také tepelný výkon vyzářený z každé jedné slupky směrem ven stejný jako výkon vyzářený směrem dovnitř.

Tepelné záření se přes vakuovou mezeru šíří bez jakékoliv interakce a dopadne na sousední stěnu. V tu chvíli se může buďto odrazit nebo pohltit.[1] Pokud se odrazí, vrací se zpět ke stěně, ze které vyšlo. Na ní se opět buďto odrazí, nebo pohltí. Řada odrazů může pokračovat, ale nakonec musí každý foton skončit pohlcený buďto ve stěně ze které vyšel, nebo v sousední stěně. Část výkonu, která byla nakonec pohlcena původní stěnou, se nepodílí na přenosu tepla přes vakuovou mezeru, takže můžeme místo celkového vyzářeného výkonu z povrchu každé stěny uvažovat jen efektivní přenesený výkon, což bude vyzářený výkon vynásobený procentem, které se pohltilo na sousední stěně.

Poměr odraženého a pohlceného záření závisí na vlnové délce dopadajícího záření, úhlu dopadu a materiálu odrážejícího povrchu. Jde opět o komplikovanou neznámou závislost, ale nám bude stačit, že pro každou mezislupku je efektivní výkon přenesený z ní na sousední slupku směrem dovnitř stejný jako efektivní výkon přenesený směrem ven. V obou směrech je totiž vyzářeno záření stejného výkonu se stejným rozdělením vlnových délek (oba povrchy jedné slupky mají stejnou teplotu a materiál), z kterého se následně stejné procento pohltí na příslušné sousední stěně (protože oba povrchy jsou opět ze stejného materiálu).[2] Můžeme tedy konstatovat, že pro každou slupku je efektivní přenesený výkon z ní na sousední slupku směrem ven i směrem dovnitř stejný.



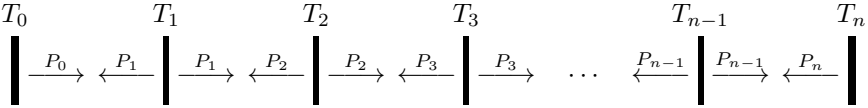
[1] Můžeme oprávněně očekávat, že kovová slupka je neprůhledná a žádná část dopadajícího záření neprojde skrz.

[2] To není úplně pravda, při detailním zkoumání můžeme narazit přinejmenším na dva vlivy rozbíjející zmíněnou symetrii. Za prvé termoska s uzavřeným vnitřním objemem musí mít nutně zakřivené stěny a vnější vydutá stěna bude hůře odrážet, protože malá část odraženého záření dopadne kvůli zakřivení zpět na tu samou stěnu. Za druhé je odrazivost kovových povrchů mírně závislá na teplotě. Vnitřní stěna, která je o něco teplejší, bude hůře odrážet dopadající záření. Oba efekty jsou ale velmi slabé a jejich zanedbání výsledek neovlivní o nic víc než jiná použitá zjednodušení.

Poslední skutečnost, kterou si musíme uvědomit, je, že každá stěna termosky je v tepelné rovnováze. Tepelný výkon, který ji opouští, musí být shodný jako výkon touto stěnou absorbovaný. Ve schematickém znázornění na Obrázku 1 to znamená rovnost

$$2P_i = P_{i-1} + P_{i+1} \tag{1}$$

pro každé i od 1 do $n-1$.



Obrázek 1: Přenos tepla mezi vrstvami tvořícími termosku. Levá stěna o teplotě T_0 tvoří vnitřek termosky, vnější stěna je vpravo a má teplotu T_n. Termoska obsahuje $n-1$ vnitřních slupek.

Je dobré si všimnout, že jsme nikde nepotřebovali vědět, jaká je závislost mezi efektivním přenášeným výkonem a teplotou stěny. Tato závislost totiž není nijak jednoduchá a závisí na mnoha parametrech, které v zadání nejsou zmíněny. Někteří z vás použili v řešení Stefan-Boltzmannův zákon v podobě

$$P = \varepsilon\sigma S T^4\,,$$

kde ε je emisivita povrchu (pro $\varepsilon = 1$ popisuje rovnice dokonale černé těleso). Tato rovnice má jedno velké „ale“. Emisivita jakéhokoliv reálného materiálu totiž není prostá konstanta, ale veličina obecně závislá na vlnové délce záření. Protože spektrum tepelného záření je závislé na teplotě povrchu (Planckův zákon, výše uvedený Stefan-Boltzmannův zákon je jen zintegrováním Planckova vztahu podle vlnové délky), bude i ε v rovnici výše obecnou a složitou funkcí teploty, navíc specifickou pro každý materiál. Protože je řešení této úlohy nezávislé na tom, jak konkrétně vypadá pro stěny termosky závislost vyzařování na teplotě, vyjde výsledek i tak „správně“. Myšlená termoska s dokonale šedými stěnami s konstantní emisivitou se bude při přidávání mezivrstev chovat stejně jako termoska z jakéhokoliv jiného materiálu. Nicméně je dobré být si vědom tohoto významného omezení platnosti Stefan-Boltzmannova zákona pro reálné povrchy.

Vraťme se ale k rovnici (1). Převedeme P_i a P_{i+1} na druhou stranu rovnítka, což ukáže, že celkový tepelný výkon přenesený přes každou mezeru mezi dvojicí slupek je stejný:

$$P_0 - P_1 = P_{i-1} - P_i = P_i - P_{i+1} = P_{n-1} - P_n = Q\,. \tag{2}$$

Tento výkon, Q, jsou tepelné ztráty termosky, tedy teplo, které opouští obsah a utíká ven. Jednoduchou úpravou rovnice (2) získáme

$$P_i = P_{i+1} + Q\,. \tag{3}$$

Protože Q je rovno rozdílu libovolné dvojice sousedních P_i, platí také $P_{i+1} = P_{i+2} + Q$ a dosazením do (3) získáme

$$P_i = P_{i+2} + 2Q\,. \tag{4}$$

Je zjevné, že tuto úpravu můžeme libovolněkrát analogicky opakovat a získaný výsledek zapsat pro obecné k jako

$$P_i = P_{i+k} + k \cdot Q\,. \tag{5}$$

Efektivní tepelný výkon P_0 opouštějící vnitřní stěnu a výkon P_n odcházející z vnější stěny dovnitř budou vždy stejné nezávisle na počtu mezistěn, protože jsou určeny jen (neměnnou) teplotou vnitřní, respektive vnější stěny. Pokud rovnici (5) vyjádříme právě pro tyto dva tepelné výkony, dostaneme:

$$P_0 = P_n + n \cdot Q \quad \Rightarrow \quad Q = \frac{P_0 - P_n}{n}\,. \tag{6}$$

Pro základní termosku bez mezislupek ($n = 1$) jsou tepelné ztráty, označme je Q_0, rovny rozdílu $P_0 - P_1$. Pokud přidáme s slupek ($n = s + 1$), budou ztráty Q_s dány vztahem

$$Q_s = \frac{Q_0}{s+1}\,. \tag{7}$$

Doba chladnutí obsahu je nepřímo úměrná tepelným ztrátám, takže při s slupkách bude pro dobu t_s v porovnání s časem t_0, který by stejné chladnutí trvalo u základní termosky, platit

$$t_s = (s+1) \cdot t_0\,. \tag{8}$$

Přidání jedné mezivrstvy prodlouží dobu, po kterou zůstane káva v termosce teplá na dvojnásobek, dvě mezivrstvy nám poskytnou oproti základní termosce trojnásobný čas na vychutnání si teplého obsahu atd.

A funguje to opravdu...?

Řešení úlohy naznačuje, že tu máme zajímavý způsob jak, alespoň pro prvních několik mezislupek, podstatně vylepšit izolační schopnosti termosky. Takže plán je jasný, vyrobíme supertermosku, začneme ji prodávat a zbohatneme... nebo ne? Funguje to tedy?

Řešení výše skutečně popisuje, jak se sníží tepelné ztráty *zářením*. Jenže aby to stačilo ke zpomalení chladnutí obsahu, musíme předpokládat (jak bylo postulováno v zadání), že záření je hlavní mechanismus přenosu tepla. To ale často není pravda.

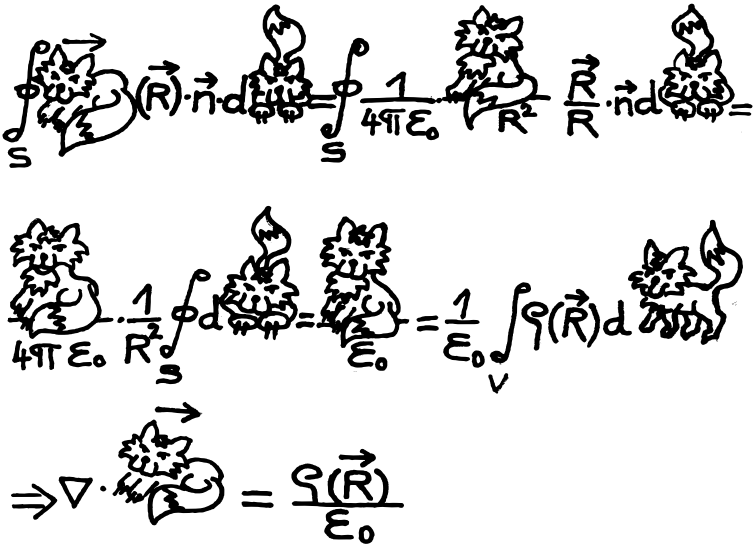
Běžné kovové materiály samy o sobě hodně dobře odráží a málo vyzařují infračervené záření, které je v našem případě hlavním nositelem tepla. Takže i základní dvouvrstvá termoska je z hlediska ztrát zářením „poměrně dobrá“. Pokud budeme tyto ztráty dále snižovat, tak dříve či později (spíše dříve) převáží jiné

mechanismy[3], které přidáváním stěn nijak neomezujeme (naopak je v praktické situaci, kdy musíme mezislupky nějak upevnit, pravděpodobně ještě zvýšíme), a v součtu už si nijak nepolepšíme. Nemluvě o tom, že vyrobit vícevrstvou termosku bude výrazně složitější a dražší.

Na druhou stranu, pokud jsou ambice tepelné izolace trochu větší, než hrníček teplé kávy, může se situace změnit. Například pro uchovávání a vedení kapalného hélia nebo jiných podobných extrémně chladných kapalin se vrstvená izolace skutečně používá.[4] Má i svůj anglický název a zkratku: MLI – multilayer insulation, a můžeme ji v různých podobách najít jak v pozemních, tak kosmických aplikacích.

Do jisté míry lze za související příklad považovat třeba i sluneční clonu Webbova teleskopu[5], ačkoliv tam hodně pomáhá skutečnost, že dalekohled nemusí být izolací „obalen“ a odražené tepelné záření může mimo pohlcení jednou ze dvou stěn také uniknout z okraje clony pryč do volného prostoru, což pochopitelně dále vylepšuje tepelnou bilanci.

Marble



[3] Vedení kolem hrdla termosky, přenos prouděním (zbytkového) vzduchu mezi stěnami, výměna vzduchu uvnitř při nalévání části obsahu do hrníčku atd.

[4] Ilustrace jednoho takového využití je k vidění třeba v článku Wrapped multilayer insulation design and testing, S. A. Dye a kol. (2014), *Cryogenics*, 64 (`http://www.questthermal.com/sites/default/files/files/Wrapped%20MLI%20Cryogenics%20Final%20article.pdf`)

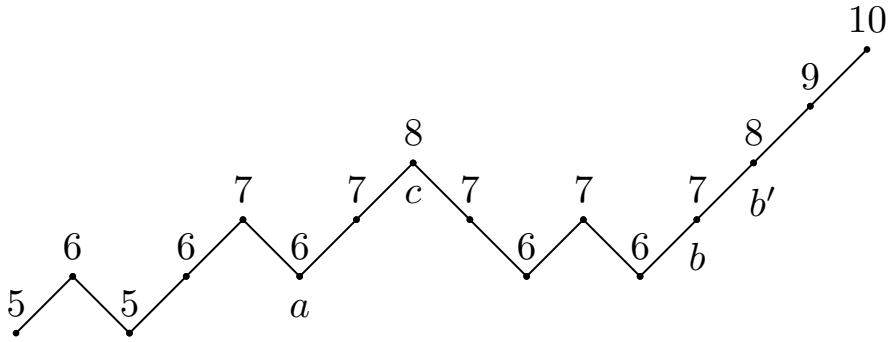
[5] `https://www.jwst.nasa.gov/sunshield.html`

Úloha 4.2 – Hřebenovka (3b)

Zadání:

Na celé cestě je N bodů tak, že mezi každými dvěma sousedními body cesta buď o jeden metr klesne, nebo o jeden metr stoupne. Cestovatele vždy zajímá, jaký je nejvyšší bod na nějakém úseku cesty, neboť z něj bude nejhezčí výhled. Na dotaz „Jaký je nejvyšší bod na cestě mezi body a a b?“ tedy jako odpověď očekává bod c, který leží mezi body a a b (včetně) a zároveň má největší možnou nadmořskou výšku (viz Obrázek 2).

Vaším cílem je navrhnout co nejefektivnější algoritmus, který pro danou cestu (zadanou jako posloupnost nadmořských výšek bodů na ní tak, jak jdou za sebou) bude odpovídat na dotazy. Dotazy samozřejmě neznáte předem, ale můžete předpokládat, že počet dotazů K je řádově větší než N – počet bodů na cestě.



Obrázek 2: Příklad hřebenovky s 16 body, nad každým bodem je jeho nadmořská výška. Pro dotaz na kus cesty mezi body a a b by správnou odpovědí byl bod c s výškou 8. Pro dotaz na cestu mezi a a b' můžeme odpovědět buď opět bodem c, nebo bodem b', jelikož oba mají stejnou výšku.

Řešení:

Nejdříve nějaké značení. Body na cestě si označíme čísly od 1 do N tak, jak jdou za sebou, a výšku bodu i si označíme v_i. Úsek cesty mezi body i a j (včetně) budeme značit $[i, j]$.

Začněme s tím nejjednodušším řešením – pro každý dotaz si prostě spočítáme maximum výšek. Spočítat maximum z L bodů snadno zvládneme v $\mathcal{O}(L)$, jeden dotaz nám tedy zabere nejhůře $\mathcal{O}(N)$. Zodpovědět všechny dotazy nám tudíž bude trvat $\mathcal{O}(KN)$. Ač toto řešení na první pohled možná nevypadá tak špatně, dokážeme ho ještě zlepšit.

Nabízí se dva přístupy ke zlepšení naivního algoritmu. První je si nad cestou vybudovat nějakou datovou strukturu, která nám hledání maxima zrychlí. Druhý spočívá v tom, že když už při zodpovídání dotazu něco spočítáme, tak si to uložíme, abychom si usnadnili práci při zodpovídání dalších dotazů. Zajímavé je, že

v obou případech dojdeme v podstatě k totožnému algoritmu jen s tím rozdílem, že v prvním případě všechnu práci oddřeme v předvýpočtu, zatímco v druhém případě se tatáž práce rozprostře do dotazů. Proto se vydáme jen prvním směrem, neboť ten je snazší na analýzu.

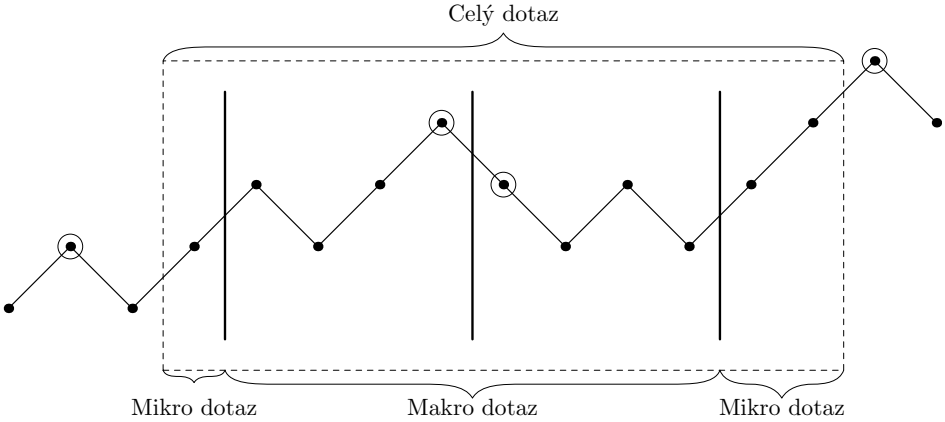
Pojďme tedy nad naší cestou vybudovat nějakou datovou strukturu, která nám zrychlí dotazy. Mnoho řešení navrhovalo použít *intervalový strom*. Intervalové stromy jsou mocnou datovou strukturou, která je určena pro zodpovídání intervalových dotazů (jako třeba „Jaký je součet čísel v daném intervalu?“)[6], na první pohled tedy vypadá jako ideální kandidát. Intervalový strom pro hledání maxima lze zkonstruovat v čase $\mathcal{O}(N)$ a potřebuje čas $\mathcal{O}(\log N)$ na dotaz, tedy celkem bychom dosáhli složitosti $\mathcal{O}(N + K \log N)$.

Bohužel, pro naše účely je intervalový strom až příliš silná zbraň (například proto, že podporuje změny posloupnosti, nad níž je vybudován, což je pro naše potřeby zbytečné) a vystačíme si s mnohem jednodušší a efektivnější strukturou. Stačí si všimnout, že různých dotazů je jen $N(N-1)/2 \leq N^2$. Můžeme si tedy předpočítat tabulku $N \times N$, která bude obsahovat odpověď pro každý možný dotaz. Na dotaz tak dokážeme odpovědět v čase $\mathcal{O}(1)$ – což je optimální! Tabulku dokážeme spočítat v čase $\mathcal{O}(N^2)$ – maximum z $[i, i]$ je v_i a maximum z $[i, j]$ je buď v_j nebo maximum z $[i, j-1]$. Celkem máme tedy složitost $\mathcal{O}(N^2 + K)$. To je čas, za který jste mohli získat plný počet bodů.

Můžeme algoritmus ještě zlepšit? Čas na dotaz už máme minimální, zkusme tedy nějak zlepšit předvýpočet. Všimněme si, že vlastně nepotřebujeme mít předpočítanou odpověď pro *každý* úsek – stačily by jen takové, ze kterých dokážeme v konstantním čase poskládat odpověď pro ostatní úseky. Předpočítáme si proto jen úseky, jejichž délky jsou mocninou dvojky – tedy pro každé $i \in \{1, 2, \ldots, N\}$ a $k \in \{0, 1, \ldots, \log N\}$ si předpočítáme maximum z úseku $[i, i + 2^k]$. Jelikož maximum z $[i, i + 2^k]$ dokážeme v $\mathcal{O}(1)$ spočítat z maxima pro $[i, i + 2^{k-1}]$ a pro $[i + 2^{k-1} + 1, i + 2^k]$, předvýpočet nám zabere čas $\mathcal{O}(N \log N)$. Dotaz na $[i, j]$ zodpovíme tak, že najdeme největší k takové, že $j - i \geq k$ (buď pomocí bitových triků, nebo si prostě pro všech N možných hodnot $j - i$ předpočítáme správné k), a vrátíme větší z maxim pro úseky $[i, i + 2^k]$ a $[j - 2^k, j]$.

Tím jsme se dostali na čas $\mathcal{O}(N \log N)$ na předvýpočet a $\mathcal{O}(1)$ na dotaz. Nicméně, předvýpočet lze stihnout v lineárním čase! K tomu použijeme následující trik: Cestu si rozdělíme na N/B bloků o B prvcích (pro vhodně zvolené B). Nad každým blokem si postavíme mikrostrukturu, která bude umět odpovídat na dotazy ohledně zadaného bloku. Následně si celou cestu „zahustíme“ – každý blok nahradíme jeho maximem, čímž dostaneme cestu délky N/B. Nad takto zahuštěnou cestou si postavíme makrostrukturu, která bude umět odpovídat na dotazy ohledně zahuštěné cesty. Každý dotaz na původní cestu pak můžeme rozdělit na jeden dotaz na makrostrukturu a nejvýše dva dotazy na mikrostruktury, viz Obrázku 3.

[6] Více se o intervalových stromech můžete dozvědět např. zde: `http://ksp.mff.cuni.cz/kucharky/intervalove-stromy/`



Obrázek 3: Ukázka dekompozice na mikro- a makrostrukturu. Cesta je rozdělena na čtyři bloky a nad každým vybudujeme mikrostrukturu. Nad maximy bloků (zakroužkované body) vybudujeme makrostrukturu. Každý dotaz na celou cestu pak můžeme rozložit na nejvýše jeden dotaz do makrostruktury a nejvýše dva dotazy do mikrostruktur.

Pro mikrostruktury použijeme naši předchozí konstrukci s kvadratickým předvýpočtem, pro makrostrukturu naopak použijeme konstrukci s $\mathcal{O}(N \log N)$. Velikost bloku B si zvolíme jako $1/2 \log N$. Dotazy tedy jistě zvládneme zodpovědět v konstantním čase, ale co předvýpočet? Konstrukce makrostruktury nám zabere čas $\mathcal{O}\left(\frac{N}{B} \log\left(\frac{N}{B}\right)\right)$. Nyní

$$\frac{N}{B} \log\left(\frac{N}{B}\right) = \left(\frac{N}{\frac{1}{2}\log N}\right) \log\left(\frac{N}{\frac{1}{2}\log N}\right) =$$

$$= \left(\frac{2N}{\log N}\right) \log\left(\frac{2N}{\log N}\right) = \frac{2N}{\log N} \cdot (\log N + \log 2 - \log\log N)$$

$$\mathcal{O}\left(\frac{2N}{\log N} \cdot (\log N + \log 2 - \log\log N)\right) = \mathcal{O}\left(\frac{N}{\log N} \log N\right) = \mathcal{O}(N),$$

ale konstrukce všech mikrostruktur nám zabere čas

$$\mathcal{O}\left(\frac{N}{B} \cdot B^2\right) = \mathcal{O}(N \log N).$$

Vypadá to tedy, že jsme si moc nepomohli. Teď ale přijde druhý trik – konečně využijeme toho, že naše cesta vždy buď o metr klesne nebo stoupne. Takových cest je totiž málo. Nejdříve si všimněme, že pokud se dvě cesty liší jen tím, v jaké výšce začínají, a posloupnost klesání a stoupání mají stejnou, pak mají maximum ve stejném bodě. Můžeme tedy předpokládat, že všechny cesty odpovídající blokům začínají ve výšce nula – z polohy maxima na cestě už snadno vyčteme jeho

hodnotu. A kolik je tedy různých cest délky B začínáních v nule? Inu, v každém bodě cesty (krom posledního) máme jen dvě možnosti – buď cesta o metr stoupne, nebo o metr klesne. Takových cest je tedy jen $2^{B-1} \leq 2^B$.

Pro námi zvolené $B = 1/2 \log N$ je různých cestiček délky B nejvýše $2^{1/2 \log N} = \sqrt{N}$. Vůbec tedy nemusíme počítat pro každý blok zvlášť jeho mikrostrukturu. Místo toho si spočítáme $\sqrt{N}$ mikrostruktur pro každou možnou cestičku délky B a každý blok si jen bude pamatovat ukazatel na mikrostrukturu, která odpovídá cestičce, kterou blok reprezentuje. Na předvýpočet mikrostruktur nám bude tedy stačit jen čas $\mathcal{O}(\log^2 N\sqrt{N}) \subseteq \mathcal{O}(N)$. Máme tedy lineární čas na předvýpočet a konstantní čas na dotaz.

Mimochodem, kdybychom na cestě povolili libovolně velká stoupání a klesání, tak dokážeme také odpovídat na dotazy v konstantním čase a mít předvýpočet v lineárním čase. To je ale o poznání komplikovanější konstrukce – hledání maxima na obecné cestě se nejdříve převede na problém hledání stromových předchůdců, který se pak převede zpět na hledání maxima na cestě, ale tentokrát už s klesáním nebo stoupáním jeden metr. Podrobněji se o tom můžete dočíst ve skriptíčkách *Krajinou grafových algoritmů*[7].

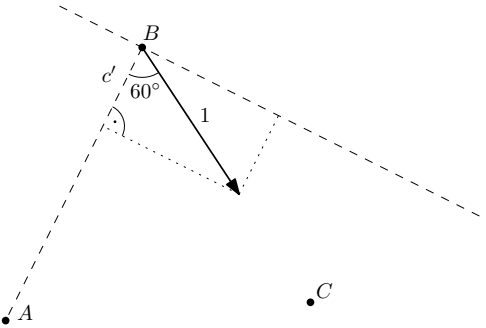
$\mathcal{O}(N)$*dra*

Úloha 4.3 – Lezoucí brouci (3b)

Zadání:

Mějme rovnostranný trojúhelník s délkou strany 1. Na každém z jeho vrcholů stojí brouk, na vrcholu A stojí brouk B_A, na vrcholu B brouk B_B a na vrcholu C brouk B_C. Brouk B_A leze tak, že neustále míří za broukem B_B, stejným způsobem leze brouk B_B za broukem B_C a brouk B_C za broukem B_A.

Všichni brouci lezou jednotkovou rychlostí. Jak dlouho bude trvat, než se všichni brouci potkají?



Obrázek 4: Rozložení směrového vektoru brouka B_B na složky.

[7] `http://mj.ucw.cz/vyuka/ga/`, kapitola 9

Řešení:
Úlohu budeme řešit tak, že si spočítáme okamžitou rychlost, kterou se dva brouci přibližují. Brouk B_A leze za broukem B_B, což do okamžité rychlosti, kterou se přibližují, přispěje 1. Současně se ale brouk B_B pohybuje rychlostí 1 za broukem B_C. Zde je potřeba si uvědomit, že celá situace byla na začátku symetrická a symetrickou tedy i zůstane – v každé chvíli tedy budou brouci tvořit rovnostranný trojúhelník. Nyní rozložíme pohyb brouka B_B do dvou složek. Rychlost brouka B_B si rozložíme na pohyb směrem k broukovi B_A a pohyb k tomuto pohybu kolmý. Můžeme si uvědomit, že tento kolmý pohyb nijak neovlivní okamžitou rychlost přibližování se brouků B_A a B_B. To můžeme nahlédnout následovně – uvažme dva body, z nichž jeden stojí na místě a ten druhý se hýbe tak, že je nenulová pouze složka kolmá k přímce definované těmito dvěma body. V takovém případě není těžké nahlédnout, že to odpovídá pohybu po kružnici, kdy se rozhodně vzdálenost těchto dvou bodů nemění, a okamžitá rychlost přibližování se těchto dvou bodů je tedy nulová. Nás bude zajímat pouze složka vedoucí za broukem B_A.

K rozložení pohybu na složky nám bude stačit použít sinus a cosinus. My chceme znát délku hrany c' (viz Obrázek 4). Ta se rovná $\cos 60° = 1/2$.

Tento argument můžeme opakovat pro všechny tři dvojice brouků. Tím zjistíme, že se všichni brouci navzájem přibližují rychlostí $1 + 1/2 = 3/2$ a celkově jim tedy bude trvat se potkat $2/3$ časové jednotky.

Kuba

Úloha 4.4 – Trezor (3b)

Zadání:
Jedenáctičlenné sdružení místních myslivců má v trezoru uložené tajné materiály. Chtějí trezor opatřit zámky tak, aby jej libovolná šestice dokázala otevřít a zároveň žádná pětice otevřít nedokázala. Kolika nejméně zámky je potřeba trezor opatřit a kolika klíči vybavit každého člena sdružení tak, aby se jim to podařilo? (Každý myslivec může mít více klíčů a zároveň od jednoho zámku může mít klíč více myslivců. Zároveň každý klíč otvírá právě jeden zámek.)

Řešení:
Ze zadání platí, že pro každou pětici musí existovat alespoň jeden zámek, k němuž nemá klíč. Zároveň k tomuto zámku musí mít klíč všichni zbývající, protože když k této pětici jednoho z nich přidáme, tak už musí trezor otevřít. Navíc každým dvěma různým pěticím musí chybět klíč k jinému zámku, jinak by ani obě dohromady (a protože jsou různé, tak je to alespoň šest lidí) nedokázaly trezor otevřít. Proto je tedy minimální počet zámků roven počtu různých pětic, což je $\binom{11}{5} = 462$.

Nyní ukážeme, proč to stačí a kolik klíčů bude mít každá osoba. Víme, že každá pětice má alespoň jeden zámek, který neotevře, a který otevře každý zbývající člověk. To je to samé jako tvrzení, že každá šestice má zámek, který nikdo ze zbývajících neotevře. To znamená, že ke každému zámku má klíč alespoň šest

lidí, takže v každé šestici bude pro každý zámek existovat někdo, kdo od něj má klíč. Takže je vidět, že 462 zámků nám stačí a každý člověk bude mít tolik klíčů, v kolika je šesticích, což spočítáme tak, že vybereme jeho (to lze jedním způsobem), a pak vybereme zbylých pět lidí (to lze $\binom{10}{5}$ způsoby), což je 252.

V této úloze jste téměř všichni, kteří jste si přečetli celé zadání, uspěli. Pouze se u několika z vás stalo, že se vám v řešení pletly pojmy „zámek“ a „klíč“, což ho občas znepřehlednilo a občas dokonce spletlo i vás. Vyplatí se proto dávat si na takové věci pozor.

Petr

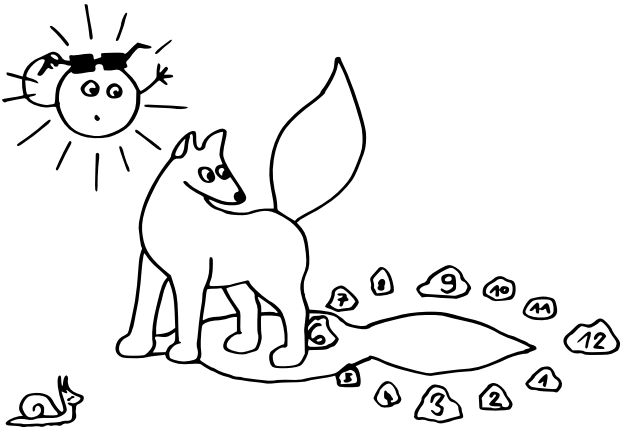
Řešení témat

Téma 1 – Sluneční hodiny

Náměty na řešení tématu najdete více rozepsané v minulých číslech, tento komentář stručně shrnuje načaté problémy, které vás můžou k řešení inspirovat:

- Předmětem tohoto tématu je sestrojení vlastního zařízení – konstrukce slunečních hodin není nijak složitá, k návrhu ciferníku můžete použít web `www.astrohk.cz/slunecni_hodiny.html`.
- Stín vrhaný předměty za slunečného dne můžeme využít k orientaci v čase i prostoru, co lze určit např. ze stínu kůlu zabodnutého v zemi (jakou křivku vykreslí v průběhu dne stín konce kůlu)?
- Doc.[MM] Petr Šimůnek ve svém řešení upozornil na několik nepřesností slunečních hodin – mimo jiné skutečnost, že Slunce není bodový zdroj, tedy hranice stínu ukazatele nebudou ostré – otázka je, jak si s touto nepřesností poradit? Jaká další omezení má přesnost slunečních hodin?
- Téma slunečních hodin si můžeme i zobecnit – šly by sestrojit měsíční hodiny? Jaké jiné objekty denní i noční oblohy mohou (nebo mohly v minulosti) být dostatečně jasné na to, abychom stín vržený ukazatelem mohli na Zemi využít k měření času?
- Analema je křivka podobná číslici 8, kterou vykreslí Slunce na obloze, pokud jeho polohu budeme zaznamenávat vždy ve stejný čas na stejném místě na Zemi každý den v průběhu jednoho roku – orientace a tvar analemy jsou dané zeměpisnou šířkou pozorovatele. Některé sluneční hodiny jsou touto „osmičkou“ vybavené z důvodu možnosti dopočtu přesnějšího časového údaje z ciferníku v případě, že pozorovatel ví, ve které části analemy by se Slunce v danou část roku mělo nacházet (časové ocejchování analemy na slunečních hodinách docela často chybí). Zkuste si nasimulovat tvar a polohu analemy z pohledu pozorovatelů na jiných planetách (můžete použít např. web `www.analemma.com`).

Dominigga



Téma 2 – Časosběr

V tomto čísle nám dorazila celkem tři videa od dvou řešitelů. Mgr.[MM]Matěj Holubička se musel poprat s rozdílnou viditelností sledovaných objektů pouhým okem a kamerou. Ve svém videu se snažil zachytit sedimentaci pevné složky moštu. Na videu toho bohužel není příliš mnoho vidět. Zajímavější by podle mého názoru bylo zamíchat nějaká zrníčka nebo prášek do čiré vody a následně sledovat sedimentaci od zakalené po čirou kapalinu. Druhé video, které ukazuje vznik bublin v perlivé vodě, je podle mě zrychlené až příliš. Problém s viditelností bublin je vyřešen perfektně, ale moc zřetelných informací jsem ve videu nenašel.

Další autorkou byla Mgr.[MM]Zuzana Urbanová, která obětovala svůj mobil a celý týden zaznamenávala klíčení řeřichy. Sledovala rozdíl mezi klíčením v půdě a ve vatě. Výsledek je ve videu jasně znatelný. Zaujalo mě i podrobné zachycení vzedmutí hlíny během klíčení. Po funkční stránce mě toto video velmi zaujalo a autorce za tento příspěvek moc děkuji. Do budoucna bych doporučil lépe uspořádat kompozici záběru, neboť druhá miska se semínky ve vatě na videu celkem zaniká. Naopak jsem rád, že autorka do videa přidala i informaci o čase, ta mi dokreslila představu o rychlosti růstu řeřichy. Titulky, které tvoří předěly mezi dny jsou podle mě příliš dlouhé. Tady bych se nebál předěly úplně vypustit a informaci o konkrétním dni přidat podobně jako informaci o čase.

Inspiraci pro vaše budoucí řešení můžete hledat třeba na YouTube, kde jsou na našem kanále[8] nahraná videa od všech řešitelů tohoto témátka.

Ročník se nám blíží ke konci, takže pokud máte ještě nějaké zajímavé nápady na zpracování, tak se blíží poslední příležitost na vytvoření a odeslání vašich řešení. K tomuto témátku zatím dorazilo 11 videí, za příspěvky děkuji všem autorům.

Béďa

[8] `https://www.youtube.com/user/KSMaM`

Téma 3 – Filmoví poradci

Do redakce nám přišel příspěvek od Mgr.MM Zuzany Urbanové, která se zabývala fyzikou v animovaném seriálu Tom a Jerry. I když se na první pohled zdá, že nemá cenu zkoumat fyzikální zákony v kreslené grotesce, článek se autorce povedl, a proto jej otiskujeme.[9] Doufáme, že příspěvek motivuje ostatní čtenáře, aby na něj reagovali nebo se věnovali otázkám z minulých čísel.

Viktor

Tom, Jerry a fyzika (11,5b)

Mgr.MM Zuzana Urbanová

Lze vnést fyziku do animovaného seriálu?

Jistě každý znáte Toma a Jerryho. Nepřátelé na život a na smrt, kteří se přesto mají svým způsobem rádi, nás provázeli naším dětstvím. V následujícím článku bych se ráda zaměřila na fungování fyzikálních zákonů v díle „Profesor Tom“, který je ke zhlédnutí například na webu [1].

Nejdříve však odpovím na jednu otázku, která vás jistě také napadla. Má vůbec smysl zabývat se fyzikálními zákony v kresleném filmu pro děti? Jednotlivé scény jsou přeci koncipovány především tak, aby pobavily diváky, než aby demonstrovaly platnost fyzikálních zákonů. Proč tedy stavět animace proti realitě?

Odpovím srovnáním Toma a Jerryho se seriálem Byl jednou jeden život. Zde jsou zábavnou formou představeny platné fakty o fungování lidského těla natolik věrně, že mnozí učitelé doporučují tento kreslený seriál jako doplněk k výuce biologie. Byl jednou jeden život je důkazem toho, že lze spojit vědu se zábavou, a proto bych ráda i do Toma a Jerryho vnesla kousek vědy a zjistila, nakolik vyobrazené scény odpovídají realitě.

Neuvěřitelná rychlost

První oblastí, na kterou se podíváme, bude rychlost běhu obou hlavních představitelů. V čase 3:50–3:55 honí Tom Jerryho okolo stěny dělící několik pokojů. Jedno kolo oba oběhnou za 1,4 sekundy. Vzhledem k tomu, že dům je uzpůsobený pro normálně velkého člověka, poloměr jednoho kola odhaduji na 2 metry. Poté by Tom i Jerry museli běžet rychlostí téměř $9\,\mathrm{m}\cdot\mathrm{s}^{-1}$. Podle [2] umí kočka domácí, což pravděpodobně Tom je, vyvinout rychlost až $13\,\mathrm{m}\cdot\mathrm{s}^{-1}$. To odpovídá dané scéně. Myš domácí však podle [3] umí běžet pouze rychlostí $3{,}6\,\mathrm{m}\cdot\mathrm{s}^{-1}$. Jerry by tedy Tomovi ve skutečnosti nemohl tak rychle utíkat a Tom by ho dostihl.

Za zmínku stojí také odstředivá síla, která při tomto pohybu vzniká. Budeme-li předpokládat, že Tom váží 6 kg, je velikost odstředivé síly rovna 243 N. Protože velikost tíhové síly je pouze 60 N, musel by být Tom vychýlený 76° od svislého směru ke středu. Ve scéně ale vidíme, že je Tom nakloněný pouze dopředu.

[9] Článek byl redakčně zkrácen.

Při takovém rozdílu odstředivé a tíhové síly by také Tom na jakékoliv vodorovné podložce uklouzl. Mohl by se však zachytávat drápy, což by vysvětlilo, že v této scéně neuklouzl.

Nadlidská síla

V čase 5:45 odhodí Jerry Toma i s kobercem do jezírka. Na to, abychom zjistili, že by se to ve skutečnosti stát nemohlo, žádné výpočty nepotřebujeme. Pojďme se ale podívat, jakou sílu by Jerry musel vyvinout, aby se mu to mohlo podařit.

Podle druhého Newtonova zákona je tato síla přímo úměrná hmotnosti Toma i s kobercem (odhadněme tuto hmotnost na 7 kg) a jejich zrychlení. Toto zrychlení je závislé na rychlosti, které Tom a koberec hned po vyhození dosáhnou, a na čase, po který se Jerry rozmachoval.

Jaká byla tedy rychlost na počátku vrhu? Nejprve odhadneme úhel vrhu, odhaduji 45°. Vrh trvá celkem 1,2 sekundy. Rychlost na počátku vrhu tedy musí být

$$v_0 = \frac{tg}{2\sin\alpha} = 8{,}5\,\mathrm{m}\cdot\mathrm{s}^{-1}.$$

Pro kontrolu vypočítáme vzdálenost, na kterou je Tom za těchto okolností vržen:

$$s = \frac{v_0^2 \sin 2\alpha}{g} = 7{,}2\,\mathrm{m}\,.$$

Tato vzdálenost by opravdu mohla odpovídat skutečnosti zobrazené ve filmu, při odhadování vzdálenosti nám pomohla koláž z jednotlivých snímků filmu.

Jerry se rozmachuje 0,21 s. Zrychlení tedy musí být rovno $40\,\mathrm{m}\cdot\mathrm{s}^{-2}$. Síla vyvinutá Jerrym by se tedy musela rovnat 280 N, což je síla odpovídající 28 kg. Jak těžké předměty umějí myši zvednout ve skutečnosti? Podle [4] je to v průměru pouhých 70 gramů. Jerry tedy musí být supermyš, obdoba našeho Supermana!

Závěrem

Toto tedy byly některé scény s fyzikálními nepřesnostmi. Zjistili jsme, že na Toma nepůsobí některé fyzikální zákony a že Jerry je supermyš. Pokud by se seriál měl řídit skutečností a věrně zobrazovat fyzikální zákony a biologické vlastnosti koček a myší, bylo by potřeba většinu scén upravit. Ale vzhledem k tomu, že účelem tohoto seriálu je pouze pobavit diváky (mimo jiné právě záměrným porušováním fyzikálních zákonů), myslím, že je přesto hodný zhlédnutí a svůj účel plní.

Zdroje

[1] `https://vimeo.com/12403873`

[2] `http://www.speedofanimals.com/animals/domestic_cat`

[3] `http://www.speedofanimals.com/animals/house_mouse`

[4] `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3725666/`

Všechny internetové zdroje platné k 10. 2. 2017.

Konference Hříběcí 2016

Gravitace ve Flatlandu (15 b)

Dr.^{MM} Ondřej Knopp

Teoreticky

Často se vysvětluje gravitace v moderním smyslu jako zakřivování časoprostoru hmotným tělesem za pomoci napnuté blány. V takové demonstraci se na blánu vloží nějaký hmotný předmět, kupříkladu kulečníková koule, která takovou blánu prohne. Často je blána napnutá už na nějaké železné obruče a prohnutí je tak vytvořeno napevno. Následovně se vezme nějaký jiný menší sféricky symetrický objekt a ten se vhodí na takovou zakřivenou blánu. Dle způsobu, jakým takový menší objekt vhodíme na blánu, nám ukáže různé trajektorie. Nejčastěji však menší objekt spadne k těžšímu objektu nebo do nejnižšího bodu na prohnuté bláně. Někdy se také podaří menší objekt vhodit takovým způsobem, že provede pár orbit okolo středu zakřivené blány, a tak se dá ukazovat existence orbit planet právě v zakřiveném časoprostoru. Důležitá otázka však je, jak i se zanedbáním tření je tato demonstrace gravitace blízká skutečnosti.

Newton a pole

Prvně je důležité zavést problematiku a pojmy spojené s popisem gravitace, tak jak je klasicky známý. Tím mám na mysli Newtonův gravitační zákon a užitečné veličiny, které nám jeho obecný popis zjednoduší.

Newtonův gravitační zákon říká, že okamžitá síla působící vzájemně mezi dvěma hmotnými objekty směřuje vzájemně přitažlivě mezi nimi a její velikost je

$$F = G\frac{Mm}{r^2}.$$

Zde m a M jsou hmotnosti daných objektů, r^2 značí čtverec jejich vzdáleností a G je Newtonova gravitační konstanta. Ta nám vztahuje naše jednotky hmotnosti a vzdálenosti vůči síle mezi nimy. Říká nám tedy jak moc silná gravitace je.

Často nám však silový popis nestačí. Další způsoby popisů jsou často v našich případech užitečné, pokud jedno z těles, kupříkladu to s hmotností M, je oproti tomu druhému tak masivní, že jím gravitace od toho druhého nepohne. Nyní se budeme bavit o dvou veličinách vlastních menšímu tělesu, na které se budeme dívat. První z nich je potenciální energie. Tato energie závisí na vzdálenosti od centrálního tělesa. Budu jí značit $E_p(r)$, kde r je právě tato vzdálenost. Pokud se těleso nachází ve vzdálenosti r_0 od centrálního tělesa a pak se přesune do vzdálenosti r, tak se $E_p(r_0) - E_p(r)$ přemění na energii kinetickou.

Obecně si tuto potenciální energii mohu vyjádřit jako

$$E_p(r) = -G\frac{Mm}{r}.$$

Obecněji užitečnější veličina je gravitační potenciál. Jde o potenciální enegii na jednotkovou hmotnost tělesa, které je tímto potenciálem ovlivněno. Potenciál se značí $\phi(r)$ a je opět funkcí vzdálenosti od centrálního tělesa. Obdobně jako potenciální energie se dá vyjádřit jako

$$\phi(r) = -G\frac{M}{r}.$$

Dále vidímě, že platí $E_p(r) = m\phi(r)$, kde m je hmotnost tělesa, které má tuto potenciální energii. Můžeme říct, že danému tělesu gravitační potenciál dává potenciální energii. Potenciál nám pomáhá oprostit se od gravitační síly dle Newtona a zaměřit se na gravitaci jako na pole. Každý hmotný objekt okolo sebe vytváří gravitační pole popsatelné potenciálem $\phi(r)$, kde každému bodu v prostoru je nějaký přiřazen. Pokud je v prostoru objektů více, jejich potenciály se sčítají. Nyní každému bodu v prostoru jsme schopni přiřadit gravitační potenciál. Pokud do prostoru nyní přidáme objekt o hmotnosti m, tak jsme schopni se dopočítat jeho potenciální energie v tomto bodě a z potenciální energie i síly, která na něj působí. Je nutné také vědět to, že pro velikost síly od jednoho centrálního objektu platí

$$F = -\frac{\mathrm{d}E_p(r)}{\mathrm{d}r} = -m\frac{\mathrm{d}\phi(r)}{\mathrm{d}r}\,, \tag{9}$$

pokud říkáme, že síla je kladná ve směru od centrálního tělesa. Je nutné si zapamatovat, že rovnice (9) platí obecně a i pro naši sílu na membráně, která má představovat gravitaci. Tuto sílu budeme schopni pro lehká tělesa okolo nějakého těžkého určit z tvaru membrány. Tento tvar zjistíme potom, co si projdeme hlavním peklem této úlohy. Až potom se vrátíme k využití těchto pojmů.

O tvaru membrány

Nyní je nutné tedy porovnat potenciál okolo jednoho hmotného objektu, který se při demonstracích nejčastěji vyskytuje. Blána se může napnout na nějakou kruhovou obruč a ze zřejmých důvodů těžká koule spadne vždy do jejího středu. Můžu předpokládat, že prohnutí blány je kruhově symetrické se středem v hmotném objektu. Za toho předpokladu můžu takové prohnutí vyjádřit jako funkci $f(r)$ výšky blány ve vzdálenosti r od osy souměrnosti, jenž prochází středem hmotné koule. Definiční obor takové funkce je však značně omezen na rozměry aparatury. Jistě platí, že $r < R$, kde R je poloměr zmiňované železné obruče. Dále je nutné, aby $r > \rho$, kde ρ je poloměr kružnice doteku těžké koule s membránou nebo poloměr menší železné obruče, která drží takto blánu dole. Definiční obor je tedy nutně $D_\mathrm{f} = (\rho,R)$. Při vytváření modelu prohnutí je nutné popsat i fyzikální chování takové blány. Blánou může být kupříkladu nějaká pružná látka. Budu předpokládat dokonalou elasticitu blány. Blána se prohne tak, aby minimalizovala svou potenciální energii. Hmotnost blány, či její plošná hustota, by měla být zanedbatelná. Potenciální energie blány bude tedy schovaná právě v jejím napnutí. Pro dokonale elastický materiál potenciální energie jeho napnutí E je přímo úměrná ploše S, na kterou je natažená, kterou zabírá napnutá membrána. Do této plochy se v případě s hmotnou koulí počítá i plocha doteku koule a membrány. Taková plocha pro obecný případ je různá pro různé prohnutí blány. Prozatím budu předpokládat, že je konstantní. Pokud je energie uložená v napnutí přímo úměrná ploše, pak se musím pokusit najít takové prohnutí blány, při kterém bude blána zabírat nejmenší plochu.

Obecně se řešením takovýchto minimalizačních úhol zabývá variační počet. Obecně jde o velice zdlouhavý postup, který nám může prozradit jaký tvar zabírá hledaná funkce $f(r)$. Konkrétně z postupu, který se nachází v plné verzi textu[10] jsme schopni zjistit, že funkce $f(r)$ má tvar

$$f(r) = a \cosh^{-1}\left(\frac{r}{a}\right) + c\,,$$

kde $\cosh^{-1}$ je inverzní cosinus hyperbolický a a, c jsou konstanty, které nám jsou schopny popsat konkrétní situace různě napnutých blán. Pojďme si nyní promyslet význam těchto konstant. Není těžké odhadnout, co bude znamenat konstanta c. Tato konstanta nám bude pouze posouvat blánou nahoru a dolů dokud nebude ve správné výšce. S konstantou a to bude složitější. Je nutné podotknout, že a nám neříká něco jenom o tvaru křivky, ale i definičním oboru funkce $f(r)$. To je

[10] https://mam.mff.cuni.cz/clanky/resitel/

způsobeno tím, že $\cosh^{-1}(x)$ je definován pouze pro $x \geq 1$. Pokud například platí, že $\rho \leq a$, kde ρ je dolní hranice definičního oboru funkce $f(r)$, pak víme, že něco nedává smysl. To znamená, že jsme a či ρ buď špatně změřili, vypočítali nebo dokonce se nacházíme v nějakém krajním případě, který z fyzikálních důvodů způsobuje to, že náš model není přesný. Dále je vidět to, že se zvyšujícím se a sice $f(r)$ roste, ale je pak méně strmá. Obecně a je právě ta důležitá konstanta, která nám popisuje falešnou gravitaci na bláně, tedy její zakřivení.

Nyní máme snad dobrý matematický popis zakřivené blány. Nyní v našem vesmíru můžeme zavést přitažlivou sílu, gravitaci, jejíž potenciál je

$$\phi(r) = ga \cosh^{-1}\left(\frac{r}{a}\right) - \phi_0,$$

kde $g = 9{,}81\,\mathrm{ms}^{-2}$ je gravitační zrychlení v prostředí, ve kterém demonstraci provádíme, a ϕ_0 je pouze nějaká konstanta, která nám určuje, v jaké vzdálenosti je potenciál nulový. Dále je zde síla, kterou je menší kulička o hmotnosti m táhnuta dolů. Pro sílu platí

$$\boldsymbol{F} = -m\frac{\partial \phi}{\partial r}\hat{\boldsymbol{r}} = -gm \cdot \sqrt{\frac{a^2}{r^2 - a^2}}\hat{\boldsymbol{r}}.$$

Zde je vyjádřeno to, že síla je radiální a míří směrem k těžké kouli, neboť $\hat{\boldsymbol{r}}$ značí jednotkový vektor mířící od těžké koule k malé kuličce.

Nyní je dobré porovnat tento model se skutečností, alespoň se skutečností z pohledu newtonowské gravitace. Pro sílu, jíž velice těžké těleso působí na nějaké menší, jako je to v tomto případě, platí

$$\boldsymbol{F}_R = -\frac{mC}{r^2}\hat{\boldsymbol{r}},$$

kde C je nějaká konstanta, která závisí i na hmotnosti velkého tělesa. Tomu odpovídá po opětovném zintegrování potenciál

$$\phi_R(r) = -\frac{C}{r} + \phi_e.$$

Už od pohledu je vidět, že v obou případech jde o velice jiné výrazy. Zda-li se však jedná o diametrálně různé výrazy, můžeme zjistit provedením následující aproximace. Nechť $a \ll r$. V tom případě mohu aproximovat potenciál kuličky na bláně jako

$$\phi(r) \approx ga \ln(r) - \phi_a.$$

Šlo o aproximaci inverzního cosinu hyperbolického na logaritmus. Při této aproximaci se ve výrazu objevilo mnoho dalších konstant, které mě však nezajímají, a tak jsem je všechny schoval do nové referenční potenciální energie. Nyní síla za použití stejné aproximace nabude tvaru

$$\boldsymbol{F} \approx -gm\frac{a}{r}\hat{\boldsymbol{r}}. \tag{10}$$

Zde je vidět, že opravdu obě pozorované síly na malý objekt jsou diametrálně jiné. Zatímco newtonovská gravitace ukazuje, že síla klesá se druhou mocninou vzdálenosti, na bláně klesá s první mocninou vzdálenosti od osy svislé symetrie.

Porovnáním potenciálů dostaneme ještě zajímavější výsledek. V našem světě existuje koncept únikové rychlosti. Protože reálný potenciál klesá s první mocninou, tak se jako každá správná hyperbola v nekonečnu blíží nějaké hodnotě. Pokud dodáme tělesu takovou energii, že je schopno překonat tuto hodnotu, pak můžu prostě odletět pryč od tělesa, které gravitační pole generuje a už se nikdy nevrátím. Na bláně má však potenciál pro velké vzdálenosti, které nás zajímají, tvar logaritmické funkce a pro ni tato podmínka neplatí. To znamená, že pro každou konečnou hodnotu existuje taková vzdálenost od středu blány, že potenciál zde už bude větší než tato konečná hodnota. To znamená, že ať dám kuličce na nekonečné prohnuté bláně jakoukoliv energii, vždy se dostane do bodu, v němž potenciál tuto energii převýší a kulička bude nucena se k centrálnímu tělesu vrátit. Takže na bláně žádná úniková rychlost pro kuličku neexistuje.

Gaussova věta

Nyní se můžeme přesunout k úvahám nad nějakými obecnějšími zákonitostmi gravitace a zamyslet se nad tím, zda existuje nějaký fyzikální model či zákon, který by se dal obecněji užít na newtonovskou gravitaci obecně. Protože pohyb nedává pro ilustraci ve směru svislém žádný smysl, celou simulaci můžeme považovat pouze za rovinnou. Můžeme se nyní zeptat konkrétněji. Existuje nějaký zákon popisující gravitaci, který se mění v závislosti na dimenzi? Odpověď zní ano! Jsme schopni takový zákon vytvořit. Jde o analogii Gaussovy věty, se kterou se můžeme setkat v souvislosti s elektrickým polem, pro pole gravitační.

Prvně jde o to, jak obecně o takových polích, resp vektorových polích přemýšlet abstraktněji. Vektorové pole se dá běžně chápat jako zobrazení, které každému bodu prostoru přidělí nějaký vektor. V tomto případě každému bodu v prostoru přiřadíme vektor intenzity gravitačního pole κ. Intenzita gravitačního pole je definovatelná jako síla na jednotkovou hmotnost. Jednotka intenzity je tudíž $\mathrm{N/kg}$ a to je stejné jako $\mathrm{m \cdot s^{-2}}$. A proto κ můžeme také považovat za gravitační zrychlení v bodě.

Pokud si představíme takové gravitační pole okolo nějaké naší hmotné koule v prostoru, tak si lze představit, jak do něj směřují siločáry tohoto pole. Můžeme si představit to, že čím silnější pole je, tím více nahuštěné jsou k sobě siločáry. Právě jako hustotu siločar skrz malou jednotkovou plošku přímo namířenou na kouli si můžeme představit intenzitu gravitačního pole κ. Zdroj síly, tedy naši hmotnou kouli, si můžeme představit jako zdroj daného počtu siločar a tento počet stoupá s její hmotností. Pokud je siločar omezený počet daný zdrojem, pak pokud se vzdalujeme od koule, tak tato hustota klesá, protože siločáru jsou více rozptýleny do prostoru.

Počet siločar, který koule, vydává můžeme určit jako $4\pi r^2\kappa$. Ten musí zůstat konstantní. Pokud se podívám na tuto situaci ve dvou různých vzdálenostech r_0 a r, kde jsou intenzity κ_0 a κ, pak platí

$$4\pi r^2\kappa = 4\pi r_0^2\kappa_0 \quad \Rightarrow \quad \kappa = \frac{r_0^2}{r^2}\kappa_0.$$

Jak je vidět, intenzita opravdu klesá s druhou mocninou vzdálenosti, stejně jako Newtonova gravitace. Aby tyto dva modely seděly absolutně, můžu počet siločar generovaných zdrojem vyjádřit jako

$$-4\pi GM = 4\pi r^2\kappa. \tag{11}$$

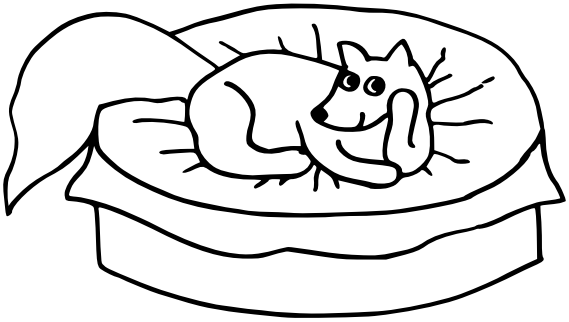
Můžete si všimnout, že jsem teď psal κ pouze skalárně. Tím jsem myslel jeho velikost a směr κ je radiální a směřující ke zdroji.

Nyní tento zákon zkusím převést do roviny. Neboť právě na bláně se koule a kuličky vlastně pohybují na rovině.

V rovině by byly zdrojem gravitačního pole nějaké planimetrické útvary. Navíc bych těmto útvarům také musel určit nějakou hmotnost. Nechť zadefinuji na membráně rozměr útvaru jako hranice doteku reálného objektu s membránou. 2D hmotnost tohoto objektu na membráně zavedu stejně jako jeho opravdovou hmotnost. V rovině opět můžu totožně zavést veličiny jako je potenciál, síla a intenzita gravitačního pole. Jediný rozdíl je v tom, že nyní neuzavírám kruhově symetrické objekty koulí o poloměru r, ale kružnicí. Musím upravit rovnici 11 do tvaru odpovídajícího rovině. Teď se nebudu bavit o ploše koule, ale o délce kružnice, skrz kterou procházejí siločáry gravitačního pole. Délka kružnice je $2\pi r$, a proto pro velikost opět radiální intenzity gravitačního pole κ platí

$$-2\pi GM = 2\pi r\kappa \quad \Rightarrow \quad \kappa = -\frac{GM}{r}.$$

Jak vidím, intenzita gravitačního pole odpovídá aproximaci síly na bláně v rovnici (10). Takto jsme se po úmorném boji dopracovali k tomu, že gravitace simulovaná na bláně se v aproximaci blíží popisu gravitace na 2D rovině. A to je velice pěkný závěr.



Seriál: Malý úvod do assembleru III

Nyní už umíme základní věci, které jsou k programování potřeba – máme základní aritmetické operace[11], umíme manipulovat s jednotlivými bity pomocí logických operací a měnit chod programu pomocí větvení a skoků. Všechno tohle dohromady je sice hezké, ale i přes to jsou všechny výpočty našeho procesoru v tuto chvíli k ničemu – neumíme je z něj dostat ven. Také časem zjistíme, že osm registrů není mnoho – zkuste si psát program o nejvýše osmi proměnných. Na některé problémy to stačí, ale brzy to bude problém – ti z vás, kteří programují, běžně používají pole o stovkách i tisících prvků. Řádově nad tisícovky se sice nedostaneme – naše šestnáctibitová architektura zatím nemá možnost, jak adresovat paměť nad adresou hodnoty `0xFFFF`[12], ale nám to bude stačit. Takže nás bude zajímat i zápis do paměti. Minule jste měli za úkol si práci s pamětí zkusit navrhnout, dneska se na ni tedy podíváme blíže.

S pamětí už budeme mít ke vstupu a výstupu blíže – podíváme se na takzvaný vstup/výstup mapovaný do paměti (dále budu pro vstup/výstup používat zkratku I/O, z anglického Input/Output), což je jedna z technik, které se dají použít například pro vykreslování pixelů na nějaké zobrazovací zařízení.

Až budeme mít vstup a výstup hotový, zamyslíme se nad problematikou periodického čtení těchto údajů, s čímž velmi úzce souvisí mechanika přerušení (anglicky interrupt).

Nuže, směle do toho. Chceme navrhnout instrukce pro práci s pamětí – konkrétně čtení a zápis.

`LOAD Rd, Rr` – tato instrukce načte z paměti hodnotu na adrese `Rr` do registru `Rd`.

`STORE Rd, Rr` – tato instrukce uloží do paměti na adresu `Rd` obsah registru `Rr`.

V našem případě je tedy práce s pamětí docela jednoduchá – při čtení nejprve uložíme do registru adresu, ze které chceme číst, a po zavolání instrukce `LOAD` si v dalším registru vyzvedneme příslušnou hodnotu. Podobně při zápisu do paměti – do jednoho registru uložíme adresu, do druhého hodnotu, kterou chceme zapsat, a instrukcí `STORE` provedeme zápis.

[11] Mimo násobení a dělení, ale ty zatím nepotřebujeme. A zájemci si je můžou dodefinovat analogicky ke sčítání.

[12] Což pořád může být docela dost paměti. Ukážeme si za chvíli.

Na procvičení si zkuste sepsat instrukce, kterými na adresy `0xF0` až `0xFA` uložíte čísla od jedné do deseti. A zkuste to napsat kratším kódem než:

```
LDI A, 0xF0
LDI B, 0x1
STORE  A, B
INC A
INC B
STORE A, B
INC A
INC B
STORE A,B
; a pokračujeme ještě 7x INC A INC B STORE A, B
```

To je nějaké podezřele jednoduché, nemyslíte? Teď se zamyslete nad odpovědí na otázku z minula: Máme oddělenou programovou paměť od paměti datové? Je to potřeba? Co když ji nemáme oddělenou? Nemůže se něco stát? Tyto otázky jsou natolik zásadní, že se u nich na chvíli zdržíme. Vezmeme to popořadě:

Máme oddělenou programovou a datovou paměť?
V našem případě ano[13]. A protože tím se nám docela podstatně mění situace, tak provedu trochu obsáhlejší paměťové shrnutí.

Do druhého dílu včetně byla situace jednoduchá. Používali jsme interní paměť procesoru, zvanou registry, a externí paměť, označovanou prostě jako paměť. Registry používáme k ukládání vnitřního stavu procesoru a na naše výpočty, z externí paměti jsme načítali instrukce, které procesor vykonával. K určení místa, ze kterého se čte instrukce k vykonání, slouží speciální registr, zvaný programový čítač.

Toto vše zůstane zachováno. Jen přejmenujeme naši externí paměť na „programovou paměť“ a zavedeme druhou externí paměť, kterou nazveme „datová paměť“ (viz Obrázek 5).

Je to potřeba?
Není, jen nám to velmi zjednoduší život a zabrání udělat spoustu chyb. Výhodou tohoto přístupu je jednoduchost a bezpečnost. Tato architektura se nazývá architekturou harvardskou. Používá se zejména u malých, často jednoúčelových čipů. Její nevýhodou je pak nemožnost měnit program za chodu[14] – co jste si do programové paměti nahráli, to máte. Občas se z ekonomických nebo bezpečnostních důvodů používá paměť pouze pro čtení – program do ní ve výrobě natvrdo

[13] Ti z vás, kteří se těšili na samomodifikující se kódy mají prozatím smůlu. Ale je to technika pro naše účely příliš pokročilá a ještě nebezpečnější.

[14] Berte s rezervou. Instrukce pro modifikaci programové paměti v některých případech existují, ale my se jimi zabývat nebudeme – to už je příliš velké sousto.

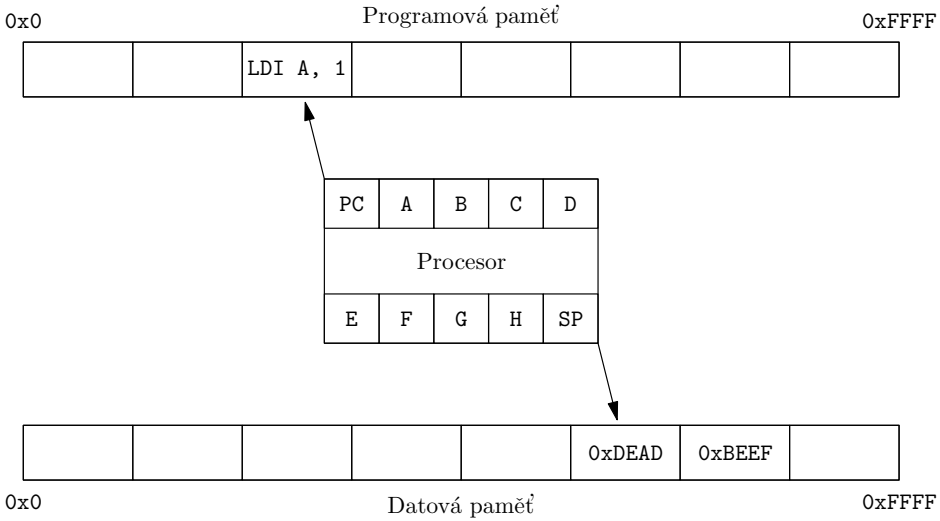
„vypálíte“ a potom jej již není možno měnit. Ani když zjistíte, že vašich deset tisíc kávovarů při stisknutí tlačítka pro presso zároveň s tlačítkem pro horkou čokoládu opaří klienta proudem klokotající vanilky.

Co když ji nemáme oddělenou – může se něco stát?
Ve chvíli, kdy nemáme oddělenou programovou a datovou paměť (tzv. von Neumannova architektura), tak se dostáváme do zajímavé situace – procesor najednou nerozlišuje mezi daty a instrukcemi. Nemá jak, protože instrukce i data jsou pořád jenom nuly a jedničky. A jediný, kdo má šanci vědět, jestli je tahle hodnota `0xDEAD` instrukcí nebo uloženou hodnotou, je programátor. Tuto architekturu používají prakticky všechny moderní počítače – operační systém je krásný příklad samomodifikujícího se kódu. Kód, který se mění za běhu, kód, který píše jiný kód – tedy takzvané metaprogramování. Kompilátory, linkery, loadery a koneckonců i sám assembler bez nutnosti psát ručně jedničky a nuly. Ach ta krása. Ach ta hrůza. Proč hrůza? Ve chvíli, kdy nemáme oddělenou programovou a datovou paměť, vzniká jedna obrovská bezpečnostní díra, se kterou doteď bojují tisíce programátorů, systémových administrátorů, návrhářů procesorů a mnozí jiní. Bez dalších opatření totiž není pro útočníka nic jednoduššího, než do paměti nahrát někam svůj kód a příští vykonávanou instrukci přepsat svou instrukcí skoku, kterou namíří na počátek tohoto kódu. A najednou je po zábavě, party skončila, jdeme domů. Způsobů, jak se tato situace s větším či menším úspěchem řeší, je nespočet, stejně tak jako je nespočet způsobů, jak tyto obrany obcházet. Je to nekonečná bitva, která s časem spíše nabírá na intenzitě. Ale to je zase na jiné povídání.

Rozhodli jsme se tedy prozatím oddělit programovou a datovou paměť a zavedli jsme instrukce `LOAD` a `STORE`. Tyto instrukce mají jednu drobnou nevýhodu – před každým čtením či zápisem musíme do registru nahrát adresu, ke které chceme přistupovat. To nás za prvé stojí čas a za druhé registr.

Nedalo by se s tím něco dělat? Zavedeme si registr, který nazveme *Stack Pointer* (zásobníkový ukazatel, značit jej budeme zkratkou `SP`) a s jeho pomocí si vytvoříme zásobník. Ha, zásobník. Programátoři zastříhají ušima, zásobník jim je důvěrně známý. Náš zásobník bude fungovat na stejném principu jako ten, který nejspíš znáte, ale bude podstatně jednodušší. Vy ostatní nezoufejte, není to složité. Tak tedy – do registru `SP` nahrajeme nějakou adresu[15]. Použijeme adresu `0xFFFF`, respektive maximální adresa paměťového rozsahu – náš zásobník totiž poroste směrem dolů. Je možno použít libovolnou jinou, ale chce to trochu opatrnosti. Tedy, `SP` ukazuje na hodnotu `0xFFFF` a toto políčko paměti se tímto stává první pozicí našeho zásobníku. Ten je zatím prázdný. Zavedu nyní dvě instrukce, můžete je vidět na Obrázku 5:

[15] Pro tuto chvíli budeme předpokládat, že se tak stane automaticky po spuštění procesoru. Inicializace zásobníku ale při běžném programování v assembleru bývá jedna z prvních instrukcí, které se provádí.



Obrázek 5: Schéma našeho modelu počítače. Procesor má krom registrů k dispozici programovou a datovou paměť. Z programové paměti procesor čte a následně vykonává instrukce, na které ukazuje Program Counter (`PC`). To, kam `PC` ukazuje, můžeme ovlivnit pomocí skoků. S datovou pamětí můžeme manipulovat buď přímo pomocí instrukcí `LOAD` a `STORE`, nebo nepřímo pomocí instrukcí `PUSH` a `POP`, které pracují se zásobníkem. K práci se zásobníkem slouží Stack Pointer (`SP`). Zásobník roste od vyšších adres směrem k nižším (tady zprava doleva) a `SP` ukazuje na první buňku paměti, která na zásobníku není (tedy nejvrchnější buňka zásobníku obsahuje hodnotu `0xBEEF`, ne `0xDEAD`).

`PUSH Rr` – tato instrukce uloží hodnotu registru `Rr` do paměti na místo určené registrem `SP` a sníží hodnotu registru `SP` o jedna

`POP Rd` – tato instrukce nejprve zvýší hodnotu registru `SP` o jedna a následně načte do registru `Rd` hodnotu z místa paměti určeného registrem `SP`.

Dalo by se říci, že `PUSH A` je ekvivalentní k zavolání `STORE SP, A` následovaným voláním `DEC SP` a stejně tak `POP Rd` můžeme nahradit sekvencí `INC SP`, `LOAD A, SP`. Proč tedy tyto instrukce zavádíme? Máme dva důvody:

1. Pohodlí – u zásobníku nemusíme řešit, odkud načítáme ani kam zapisujeme. Tyto starosti za nás řeší Stack Pointer. Zásobník se taky hodí na odkládání proměnných, pokud je za chvíli budeme znovu potřebovat – operace na zásobníku totiž mohou mít automatické zvýšení/snížení hodnoty zadrátováno přímo v čipu – takže ušetříme instrukci.

2. Síla zvyku – zásobník se běžně používá například při obsluze přerušení, takže se nám hodí se na něj podívat, abychom později nebyli překvapení.

3. Zásobník nám umožňuje mít lokální úložiště nezávislé na kontextu, ze kterého se kód zavolal, a nerozbije se ani pokud se funkce zavolá rekurzivně. Jedná se o přirozenou dynamicky alokovanou paměť.

U zásobníku musíme mít na paměti, že se jedná o strukturu typu `LIFO`, z anglického *Last In, First Out*. Prvek, který uložíme jako první, si vyzvedáváme jako poslední[16]. A ještě jedno varování – ve chvíli, kdy zavoláte instrukci `POP`, aniž by v zásobníku bylo něco uloženo, tak bude veselo. Procesor totiž instrukci ochotně vykoná, přičte do `SP` jedničku a přečte si, co vlastně?

V našem případě je u prázdného zásobníku `SP` nastaven na hodnotu `0xFFFF` – přičtením jedničky tato hodnota přeteče – na hodnotu `0x00` a vesele začneme číst paměť od začátku. Což je sice zajímavé, ale velmi, velmi nežádoucí[17]. Takže na vás mám jednu velkou prosbu – nenechávejte zásobník podtéci. I kdyby to nakrásně fungovalo, situace se dá řešit i jinak a podstatně to znepřehlední kód. Kapacita zásobníku je sice teoreticky vzato omezená pouze velikostí paměti, ale musíme pamatovat i na naši statickou paměť. Pokud bychom totiž do zásobníku dat uložili příliš mnoho, začneme si s ním přepisovat data, která jsou v paměťovém prostoru pod ním.

Instrukce `PUSH` a `POP` se občas zobecňují na instrukce použitelné pro libovolný registr, ne jen pro Stack Pointer. Zkuste si rozmyslet, jak by takové instrukce vypadaly. Kdo chce, může si je zavést a používat je – je to pohodlná alternativa například pro práci s poli. Ale nám bude současný arzenál stačit.

Je na čase se konečně podívat na vstupně-výstupní operace. Ještě než začnu, musím se přiznat – už práce s pamětí se dá považovat za takovou operaci považovat. Paměť ale bývá natolik těsně spjatá s procesorem[18], že se protokoly pro práci s pamětí nebudeme zabývat – to už bychom se nořili příliš hluboko. Přijměte prosím ono zjednodušení, že procesor pracuje s pamětí a že to umí.

No a když už jednu I/O operaci umíme, tak nejjednodušší je zneužít ji a s její pomocí udělat i ty další. Jak? V paměti si vyhradíme některé umístění – třeba hodnotu `0xAA00` – a s některým zařízením se dohodneme, že své hodnoty nalezne tam. Pro tyto účely si pořídíme abstraktní displej[19]. Náš displej v tuto chvíli oplývá úžasnou schopností zobrazit dva znaky tabulky ASCII – první z nich

[16] Pokročilejší z vás se nepochybně budou schopní dostat i k jiným prvkům zásobníku – použijete techniku používanou například v jazyce C – nahrajete si SP do jiného registru, hodnotu upravíte a dále s ní pracujete pomocí instrukcí `LOAD` a `STORE`, jako s kterýmkoliv jiným místem v paměti.

[17] Což neznamená, že někde neexistuje nějaký programátor, který s přetečením zásobníku nevytvořil něco úžasného.

[18] Některé malé čipy nemají registry – používají pouze externí paměť. Cože, programování bez registrů? Nikoliv, tyto čipy mají některé adresy externí paměti natvrdo zadrátované do instrukcí – a používají je tak jako náhražku registrů.

[19] Protože každý konkrétní displej má svůj vlastní způsob, jak si s ním procesor má povídat. A operace s displejem už navíc docela trvají – oproti procesoru i o několik řádů déle, museli bychom se tedy naučit rozumně čekat a začít více řešit například rychlost, se kterou procesor vykonává instrukce.

odpovídá vyšším osmi bitům hodnoty paměti na místě `0xAA00`, druhý si bere těch zbylých osm.

Takže když provedeme následující operace:

```
LDI A,  0xAA00   ;adresa displeje
LDI B, 0x4130    ;hodnota, kterou chceme zobrazit
STORE A, B       ;příslušný zápis do paměti
```

tak si displej přečte hodnotu `0x4130`, prvních osm bitů je `0x41` – dle ASCII se jedná o písmeno `A`. Druhých osm bitů je `0x30` – dle ASCII číslice `0`. Displej tedy zobrazí `A0`.

Úloha 6.5 – Vstup a výstup (4b)

Podúloha 6.5.1 – Hello world (2b)

Každý úvod do programovacího jazyka začíná programem `Hello world`. Nám to sice trvalo tři kapitoly, ale lepší pozdě, než nikdy. Tedy: napište program, který vypíše: „Hello, World!“.

Ale my umíme vypsat jen dva znaky? Nevadí, rozšíříme si displej na celý blok adres – jedna adresa, dvě po sobě následující písmena. Pokud budete chtít více řádků, stačí se rozhodnout, že třeba po deseti adresách (20 písmenech) ukončíte řádek a začnete nový.

Malý bonus: když budu chtít kreslit po pixelech, jak na to?

Teď už umíme výstup – a stejným způsobem můžeme data i přijímat. Domluvíme se, že naše vstupní zařízení – třeba abstraktní klávesnice – uloží hodnotu právě stisknuté klávesy na předem dohodnuté místo. A my budeme mít možnost si onu pozici v paměti přečíst instrukcí `LOAD`.

Tam, kde je výstup poměrně jasný, tam u vstupu narazíme na zajímavý problém. Konkrétně: kdy máme vstupní paměť číst?

Nejjednodušší je číst paměť opakovaně – řekněme každou milisekundu. Co v ní najdeme, to použijeme, pokud jsme změnu nestihli, asi nebyla důležitá. Toto řešení má úskalí právě v onom opakování. Aby nám totiž naše vstupní zařízení k něčemu bylo, musíme se k němu opakovaně vracet. Tento přístup má své výhody i nevýhody: Je jednoduchý na pochopení – prostě cyklicky kontrolujeme políčko paměti a když se změní, tak se podle toho zařídíme. Také můžeme přesně vědět, kdy čtení provádíme (třeba každou milisekundu), což může být v některých případech důležité. Na druhou stranu nám to může zpomalit odezvu programu, zejména pokud v něm probíhají nějaké delší výpočty při kterých na kontrolu periferií nezbývá čas. Sice možná ušetříme nějakou režii spojenou s přepínáním kontextu, ale pro uživatele to nemusí být rozhodující.

Co dělat, když se nám tento přístup nehodí? Budeme se muset naučit vstup a výstup řešit asynchronně – a k tomu budeme potřebovat mechaniku tzv. interruptů neboli přerušení.

K tomu si zavedeme ještě dva další registry: *Interrupt Cause* (`IC`) – ve kterém najdeme příčinu přerušení a *Volatile*, který budeme používat k následné obsluze – je možné jej používat i jako normální registr, ale *jeho obsah není bezpečný pokud jsou povolena přerušení.*

Dále bude potřeba si zavést nový bit registru `Flags` – konkrétně bit *Interrupt Flag* (`I`) – v našem případě se jedná o třetí bit zprava. Pokud je v bitu Interrupt Flag hodnota 0 – stav po startu procesoru, jsou přerušení zakázána. Důvod je prostý – nechceme být rušeni při inicializaci zásobníku a případných dalších operacích. Doteď jsme přerušení nepoužívali a díky tomuto bitu jsme se jím nemuseli zabývat.

My ale chceme přerušení povolit, k tomu nám bude stačit tato instrukce:

```
ORI Flags, 0b100
```

Registr `Flags` zůstane zachován, pouze přenastavíme hodnotu bitu `I`. Od této chvíle jsou přerušení povolena.

Co jsme ale povolili? Umožnili jsme procesoru reagovat na asynchronní vnější podněty. Asynchronní, tedy nezávislé na instrukcích, které procesor zrovna vykonává. Procesor obdrží přerušení a začne s jeho obsluhou, pokud bude v bitu `I` registru `Flags` hodnota 1 (tedy pokud jsou povolená přerušení) a zároveň bude libovolný bit registru `IC` nastaven na 1.

Obsluha přerušení vypadá následovně:

1. Procesor dokončí vykonávanou instrukci.
2. Zakáže přerušení (podobně jako bychom to mohli udělat my operací `ANDI FL, 0xFE`). Rozmyslete si, proč je tento krok nutný.
3. Provede `PUSH PC`.
4. Skočí na předem dohodnutou adresu. V našem případě to bude adresa `0x0` – na které jsme doteď měli svůj kód – takže máme problém.
5. A pokračuje dále v běžné činnosti – tedy ve vykonávání instrukcí. Zbytek je na programátorovi.

Programátor tedy musí napsat kus kódu, který si přečte obsah registru `IC`, a podle jeho hodnoty obslouží příslušná přerušení.

Teď si rozšíříme naši klávesnici – ta při stisku klávesy provede následující:

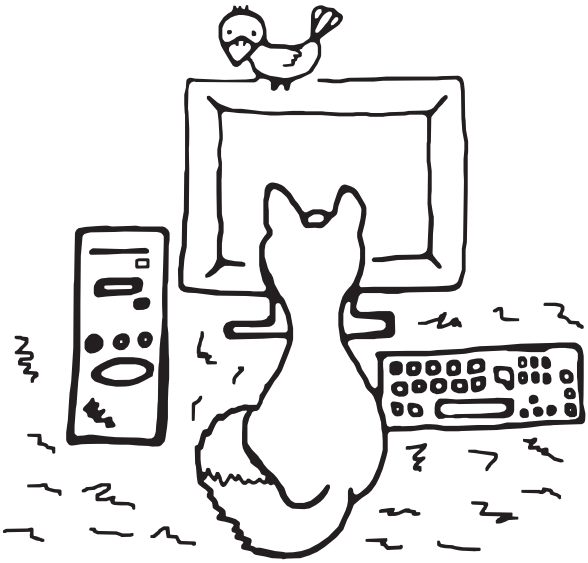
1. Uloží ASCII hodnotu stisknuté klávesy na nějakou adresu. Použijeme třeba adresu `0xBB00`.
2. Nastaví bit registru Interrupt Cause – jako by provedla `ORI IC, 0b10`.

Pokud jsou přerušení povolena, procesor začne provádět před chvílí zmíněné operace. Je na čase dohodnout se, kam skočí. A protože restart procesoru je taky přerušení, budeme skákat na začátek paměti. Na adresu `0x00`. Což nás bude stát nějaké úsilí – doteď tam totiž sídlil náš veškerý kód. Za chvíli si ukážeme příklad

možné úvodní sekvence instrukcí. Ještě před tím nás však čeká trocha papírování – najednou pro nás začalo být velmi důležité, na kterých adresách je náš kód uložen[20].

Zavedeme následující konvenci zápisu: Pokud chceme, aby blok kódu byl na konkrétní adrese, napíšeme ji před první instrukci na řádek. Další instrukce navazují, dokud nenarazíme na jinou adresu:

```
0x00  LDI A, 0x10  ;začínáme na adrese nula
      INC A        ;adresa 0x1, je to navazující adresa - nemusím
                   ;ji vypisovat
      JMP A        ;adresa 0x2, skok na adresu 0x11
                   ;tady je mezera mezi poli 0x0 a 0x11 - doporučuji
                   ;v případě mezery v paměti vynechat alespoň jeden
                   ;řádek.
0x11  JMPR zkratka ;návěstí stále fungují - píšeme je před řádek,
                   ;na který chceme skočit. JMPR zkratka tedy skočí
                   ;na adresu 0x20
zkratka:
0x20  STOP
```



A ještě jedna věc, pro přehlednost zavedeme pojmenované konstanty:

```
#define Jmeno Hodnota
```

Tedy například `#define RESTART_CAUSE 0b0001` značí, že když se v instrukci vyskytne `RESTART_CAUSE`, tak místo ní můžeme dosadit hodnotu `0b0001`. Jedná se jenom o pomůcky pro nás na úkor překladače. Protože překladač nemáme, tak to pro nás znamená přehlednost za cenu trochy práce navíc.

Ale teď už ta slíbená úvodní sekvence.

[20] Už jsme to trochu řešili, ale zatím nijak podrobně.

```
#define RESTART_CAUSE  0b0001
#define KEYBOARD_CAUSE 0b0010   ;pro tento příklad bude přerušení
                                ;způsobené restartem (a tedy i
                                ;startem) značeno bitem 1 a
                                ;přerušení vyvolané klávesnicí
;bitem 2 registru IC
#define INTERRUPT_FLAG 0x0100   ; bit I registru Flags,

0x00 MOV V, IC   ;používáme registr V - abychom náhodou nepřepsali
                 ;data našemu programu. Přerušení jsou zakázána,
                 ;takže naše práce s ním je v pořádku - nikdo nám ho
                 ;už znova nezmění.
     ANDI V, RESTART_CAUSE   ;vynulujeme všechny bity kromě posledního
     SUBI V, RESTART_CAUSE   ;pro názornost. V případě rovnosti
                             ;nastaví bit Z registru Flags - znamení
                             ;toho, že přerušení byl restart.
     BREQ restart            ;skok na obsluhu restartu.
     MOV   V, IC
     ANDI  V, KEYBOARD_CAUSE  ;analogicky pokud byla příčinou
                              ;přerušení klávesnice
     SUBI  V, RESTART_CAUSE
     BREQ  klavesnice:
                              ;v paměti cíleně nechám mezeru
                              ;na případnou obsluhu dalších
                              ;přerušení.
restart:
0xF0 LDI    SP, 0xFF
     SHL    SP, 8
     ORI    SP, 0xFF          ;inicializace stack pointeru.
     SUBI   A, RESTART_CAUSE  ;vytvoříme si bitovou masku
     AND   IC, A              ;zaznačíme, že přerušení bylo
                              ;obslouženo - smažeme příslušný
                              ;bit registru Interrupt Cause
     ORI    FL, INTERRUPT_FLAG    ;a povolíme přerušení
     JMPR main

main:
0xFF NOP          ;tělo programu
     NOP          ;já ho psáti nebudu, radši se dám na vojnu
     NOP          ;tady by byl hlavní výpočet.
     JMPR main
```

```
klavesnice:
0xF00 NOP    ;uděláme se vstupem z klávesnice to, co jsme
      NOP    ;měli v plánu, respektive uděláte
      LDI V, 0xFF
      SHL V, 8
      ORI V, 0xFF
      SUBI V, KEYBOARD_CAUSE
0xF06 ANDI IC, V   ;zaznačím, že přerušení bylo obslouženo.
                   ;sice bych ho zatím mohl rovnou vynulovat,
                   ;ale budu poctivý.
      POP V        ;popneme ze zásobníku adresu uloženou
                   ;při zahájení obsluhy přerušení
0xF08 ORI FL, INTERRUPT_FLAG   ;povolím přerušení
      JMP V   ;a vrátím se, odkud jsem přišel.
```

Pozor. Velký vykřičník. Máme v programu chybu, která je natolik závažná, že kvůli ní musíme zavést novou instrukci – jinak by se totiž téměř nedala řešit. Ne žádným rozumným způsobem.

Představte si totiž, že jste v sekci na obsluhu klávesnice, na adrese `0xF08`, dokončíte instrukci – ale někdo pořád drží klávesu stisknutou. Procesor tedy provedl bod 1 obsluhy přerušení – dokončil instrukci. A protože přerušení jsou povolená, tak pokračujeme bodem 2 – přerušení opět zakážeme. A v bodu tři se stane strašná věc – v následné obsluze přepíšeme hodnotu `V`, na kterou jsme se chtěli vrátit, hodnotou `0xF08` na které teď jsme. Jsme namydlení jako Jeníček s Mařenkou, kterým ptáci sezobali chlebové značky – nevíme, kam se vrátit.

Kudy z toho ven? Instrukcí `RETI` – *return from interrupt*. Tato instrukce provede všechny potřebné operace – tedy povolení přerušení, vyzvednutí návratové adresy ze zásobníku a skok na návratovou adresu – naráz.

Podúloha 6.5.2 – Hrejme si (2b)

Zkuste si pohrát s klávesnicí, displejem a přerušeními a něco hezkého vytvořte. I kdyby to mělo být jenom asynchronní vypisování zapsaných znaků. Kdo zvládnete něco drastického – třeba hada v ASCII artu?

Tímto dílem malý úvod do assembleru uzavírám. Pevně věřím, že jste si odnesli alespoň základní vhled do této problematiky. Zájemcům o další studium mohu doporučit nádhernou knihu o architektuře počítačů od A. Tannenbauma [1]. Kniha se assemblerem zabývá jen okrajově, ale skvěle a dopodrobna rozebírá architekturu počítačů jako takovou. Je to nádherné a vypiplané dílo – jenom varuji, že poměrně náročné. Ale ze všech knih na toto téma, které jsem zatím potkal (no, zas tolik jich nebylo), mi přišla nejzajímavější.

Tomáš Bartoněk

[1] Tanenbaum, A. S. (1976). Structured computer organization. Prentice-Hall, Pearson. ISBN 978-0132916523.

Výsledková listina 4. čísla

Poř.	Jméno	R.	$\sum_{-1}$	Úlohy r1	r2	r3	r4	s	t2	t3	k	$\sum_0$	$\sum_1$
1.	Dr.MMA. Mlezivová	3	89,0		0,5	2,9	2,9					6,3	33,5
2.	Mgr.MMK. Balej	2	32,1		3,0	1,0	3,0					7,0	32,1
3.	Dr.MMP. Turinský	4	57,4		3,0	0,7	2,5					6,2	31,9
4.	Dr.MMO. Knopp	3	55,3	3,0		3,0					15,0	21,0	28,7
5.–6.	Mgr.MMK. Rosická	2	28,5		0,5	0,5	2,0					3,0	28,5
	Mgr.MMZ. Urbanová	3	28,5		3,0	3,0	3,0		8,0	11,5		28,5	28,5
7.	Mgr.MMB. Hroncová	2	26,9	3,0	2,7		3,0					8,7	26,9
8.	Mgr.MMK. Čížková	3	29,1									0	22,6
9.	Mgr.MMR. Olšák	2	21,8			3,0	3,0					6,0	21,8
10.	Mgr.MMV. Pavlíček	1	21,3		3,0							3,0	21,3
11.	Mgr.MMM. Holubička	1	20,3						2,0			2,0	20,3
12.	Mgr.MMJ. Suchánek	3	35,1									0	19,9
13.–14.	Mgr.MMJ. Paidar	3	35,8									0	16,7
	Bc.MMK. Vnuková	2	16,7									0	16,7
15.	Dr.MMT. Domes	4	93,8		5,0		3,0					8,0	16,4
16.	Doc.MMP. Šimůnek	4	159,0					2,0				2,0	16,3
17.	Mgr.MMJ. Růžička	2	22,2	0,3		1,0	2,0					3,3	16,2
18.	Dr.MMF. Čermák	3	51,7				2,0					2,0	16,1
19.	Mgr.MML. Kundratová	2	42,7			3,0	3,0					6,0	15,4
20.	Bc.MMR. Luc	4	19,0	2,0	4,0	3,0	3,0					12,0	15,0
21.–22.	Bc.MML. Bujnovská	3	18,1			3,0	3,1					6,1	13,1
	Bc.MMZ. Lüköová	2	13,1									0	13,1
23.	Bc.MMM. Bukvaj	1	13,0									0	13,0
24.	Mgr.MMT. Hladíková	3	48,4									0	12,8
25.	Mgr.MML. Kopfová	2	31,7				2,0					2,0	12,7
26.	Mgr.MMO. Buček	3	29,6		2,0							2,0	11,3
27.	Bc.MMO. Gonzor	Z9	11,2									0	11,2
28.	Mgr.MMB. Požár	3	30,9									0	10,8
29.	Bc.MMT. Kulichová	2	10,2									0	10,2
30.	Mgr.MMJ. Pallová	2	28,8									0	9,5
31.–33.	F. Bialas	4	9,0		3,0	3,0	3,0					9,0	9,0
	Bc.MMA. Neubauerová	3	17,0									0	9,0
	V. Procházka	3	9,0									0	9,0
34.	T. Drobil	2	8,7									0	8,7
35.	Mgr.MML. Vincenová	3	45,5									0	8,4
36.	Mgr.MMD. Chytilová	3	28,8									0	8,0

Poř.	Jméno	R.	$\sum_{-1}$	Úlohy r1	r2	r3	r4	s	t2	t3	k	$\sum_0$	$\sum_1$
37.	J. Heller	4	7,5									0	7,5
38.	L. Šajnarová	3	7,3									0	7,3
39.	J. Pelc	3	7,0									0	7,0
40.	J. Šrejbr	1	6,7									0	6,7
41.	T. Dolák	4	6,3									0	6,3
42.	A. Mírková	2	6,0									0	6,0
43.	Doc.^{MM}D. Krasula	4	149,1		2,7		3,0					5,7	5,7
44.	Bc.^{MM}F. Zajíc	4	16,7	1,0								1,0	4,7
45.	Mgr.^{MM}S. Lukeš	4	42,4									0	3,2
46.	P. Martínek	2	3,1									0	3,1
47.–48.	D. Daubner	2	3,0									0	3,0
	M. Picek	2	3,0									0	3,0
49.	T. Poláková	3	2,8				0,0					0,0	2,8
50.–51.	M. Machalová	4	4,7									0	2,7
	M. Sejkorová	2	2,7									0	2,7
52.	A. Šebestíková	2	2,0									0	2,0
53.	J. Hrazdil	3	1,7	0,7			1,0					1,7	1,7
54.	F. Kmječ	1	1,5									0	1,5
55.	Mgr.^{MM}J. Vala	3	22,5				1,0					1,0	1,0
56.	L. Kubacki	4	0,0									0	0

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

Adresa redakce:

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

E-mail: mam@matfyz.cz
WWW: http://mam.matfyz.cz