STUDENTSKÝ ČASOPIS A KORESPONDENČNÍ SEMINÁŘ

Ročník XXV
Číslo 3

M&M

MATEMATIKA FYZIKA INFORMATIKA

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

Milí řešitelé,

v tomto čísle se seznámíte s novými zadáními témat. Budete zkoumat proměnlivost magnetického pole měsíce v čase a v experimentálním tématku budete zkoušet, jak moc je zkoumaný jev ovlivněn složením kapaliny. V dalším tématku se podíváme trochu více teoreticky na cestování v tramvaji. Tématko o algoritmech popisuje grafové algoritmy a fungování datových struktur jako fronta či zásobník, kryptografové si mohou v tématku počíst o módech blokových šifer. Také přibylo další matematické tématko, ve kterém se dozvíte něco o minimálním vrcholovém pokrytí grafů.

Zároveň je před námi vánoční víkendovka, která se koná 14.–16. 12. Pokud jste již přihlášení na BrNOC, nezoufejte, v sobotu je naplánován hromadný odjezd z BrNOCi na víkendovku. Registrace již byla spuštěna, přihlašovat se můžete do 13. 12. do půlnoci. Formulář najdete zde: `1url.cz/@vikendovka`

Hodně štěstí při řešení vám přejí

Vaši organizátoři

Zadání a řešení témat

Termín odeslání 3. série: 15. 1. 2019

Téma 1 – Paradoxní výsledky

Řešení 1. série

Zadání:

Co si myslíte vy, jaký byl výsledek Bartolomějova pokusu? Zmrzne teplá voda dříve než studená? Proveďte podobný pokus a ověřte, zda měl Bartoloměj pravdu. Doporučujeme to vyzkoušet nejprve s vodou, ta má víc konzistentní strukturu než Bartolomějova zmrzlina (ale nikdo vám samozřejmě nebrání něco podobného vyzkoušet). Zkuste si graficky zaznamenat závislost doby tuhnutí na počáteční teplotě jednotlivých měření, zkoumejte i závislost teploty na čase pro jednotlivá měření. Jaké parametry určují, zda je voda zmrzlá? Stanovte si okamžik, kdy prohlásíte jednotlivá měření za ukončená.

Řešení:

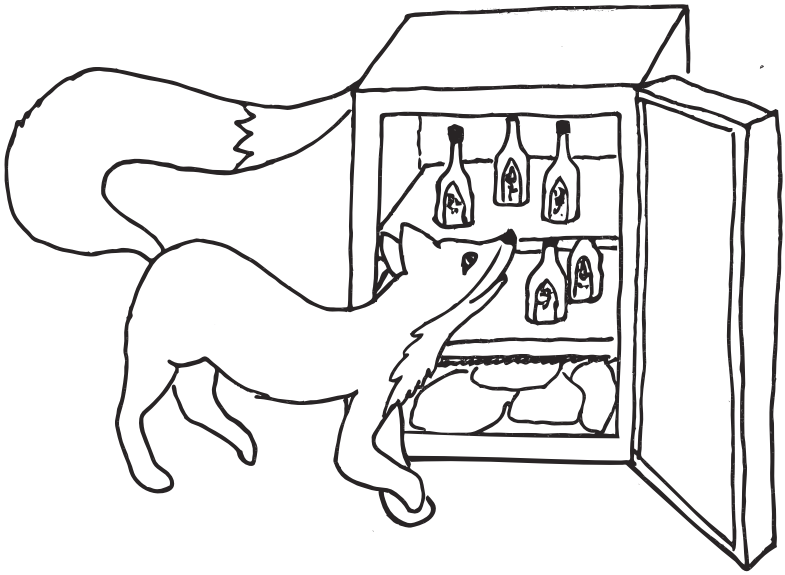
Pokud se zamyslíme nad samotnou hypotézou, tak odporuje prvotnímu logickému úsudku. Při srovnání dvou stejných nádob s vodou lišících se jen teplotou vody by nádoba s teplejší vodou měla nejprve vychladnout na teplotu té studenější vody v nádobě a pak teprve zmrznout, což neodpovídá tomu, co bychom měli experimentem dokázat.

Pro provedení tohoto experimentu je nejprve třeba definovat si podmínky, které budeme dodržovat během celého experimentu. Jako prostředí, ve kterém experiment proběhne, si zvolíme mrazák. Budeme pracovat s obyčejnou kohoutkovou vodou, kterou nejprve převaříme (např. ve varné konvici či v hrnci) a poté

necháme vychladnout na požadovanou teplotu. Tímto snížíme pravděpodobnost rozdílu v koncentraci rozpuštěných minerálů (rozpustnost látky se s teplotou rozpouštědla mění a při pozvolném chladnutí na počáteční teplotu nedojde k extrémním změnám ve složení). Můžeme použít i destilovanou vodu a tím minimalizovat anomálie způsobené složením vody úplně. Jako nádobu budeme používat odměrnou kádinku z laboratorního skla.

Z kvantitativního hlediska není dobré měřit objem, ale hmotnost kapaliny, protože objem je závislý na teplotě podle jejího koeficientu tepelné roztažnosti. Pokud však měříme kvantitu vody v objemových jednotkách, tak to pro tuto chvíli nemá vliv. Přeci jen nám zde jde o to mít u každého měření stejné množství vody a odchylky v objemu při teplotách na námi měřeném intervalu teplot od 21 °C až 82 °C jsou malé, přibližně v jednotkách mililitrů.

Měření není dobré provádět ve velkém množství, neboť se stane, že přetížíme mrazák a jeho výkon začne značně kolísat. V závislosti na velikosti mrazáku tedy budeme měřit 2 až 3 nádoby najednou.



Je dobré zvolit si, jaký typ teploměru budeme používat. Při měření bychom měli dbát na to, aby šla zavírat dvířka mrazáku na doraz a nepronikalo teplo dovnitř. Otevřené dveře by měnily výkon mrazáku a tak i celkové mikroklima a teplotu. Podobné riziko bude hrozit při každém zkontrolování teploty vody v mrazáku, kdy otevřeme dveře. Zároveň nádobu s kapalinou neumístíme do nějakého stojanu z kartonu apod., naruší nám to plynulost teplotní interakce kapaliny s okolím přes stěnu nádoby a může se stát, že naše měření nebudou vykazovat to, co by měla, kdyby tam tato bariéra nebyla.

Další bod je stanovit si moment, kdy ukončíme měření a podle čeho prohlásíme, že voda zmrzla. Zmrznutí můžeme definovat například jako moment, kdy se na vodě objeví první ledové krystaly. Nevýhoda této definice zmrznutí je ta, že se musíme spoléhat na vlastní zrak a přesně jím určit moment vzniku prvních ledových krystalků.

Jeden z problémů ovlivňujících naše výsledky je ten, že musíme experiment kontrolovat vizuálně a otevírat dveře mrazáku, a to zejména ke konci, kdy potřebujeme určit přesně okamžik ztuhnutí. Tomuto se můžeme vyhnout například tím, že si zmrznutí zadefinujeme jako moment, kdy teplota vzorku vody dosáhne teploty menší než 0 °C a zároveň použijeme teploměr s digitálními senzory, které nám budou vypisovat teplotu kapaliny pravidelně do počítače i při zavřených dveřích mrazáku. Pokud takové zařízení nemáme k dispozici, proveďme nejprve měření, podle kterého odhadneme dobu tuhnutí kapaliny a nebudeme muset mrazák otevírat už od začátku měření.

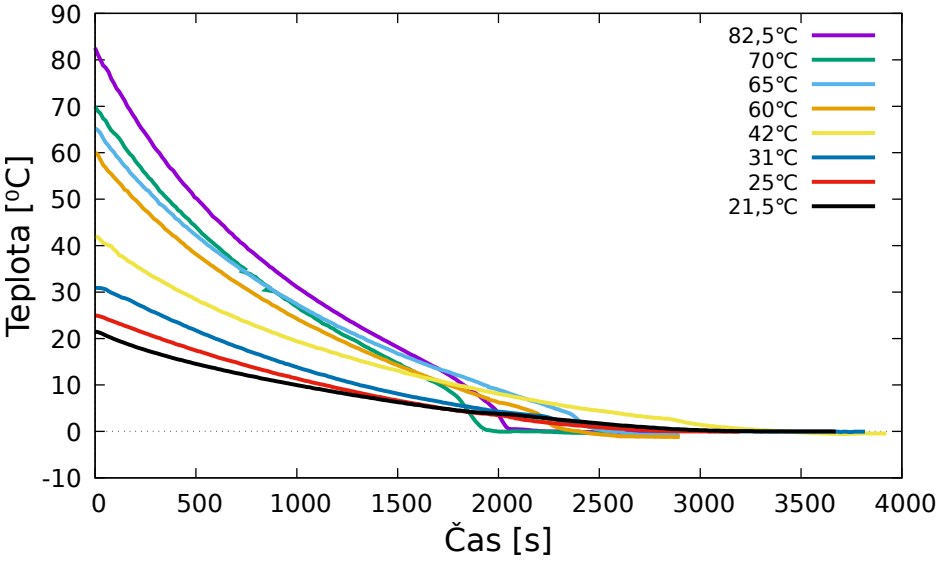
Sledování doby tuhnutí v závislosti na počáteční teplotě vody je na provedení snazší než sledovat i průběh mrznutí vody, protože v takovém případě musíme sledovat a zaznamenávat i průběžnou teplotu, ne jen čas, kdy voda zmrzne. Data z našeho experimentu můžeme vidět v tabulce 1 a na obrázcích 1 a 2 níže[1].

T_0	t_z	Δm	Úbytek ve hm. %
82,6 °C	39,28 min	7 g	7 %
69,9 °C	37,00 min	6 g	6 %
65,3 °C	42,75 min	1 g	1 %
60,0 °C	40,28 min	2 g	2 %
41,9 °C	57,43 min	3 g	3 %
30,9 °C	54,50 min	2 g	2 %
25,0 °C	51,92 min	2 g	2 %
22,1 °C	65,05 min	1 g	1 %
21,5 °C	62,48 min	1 g	1 %

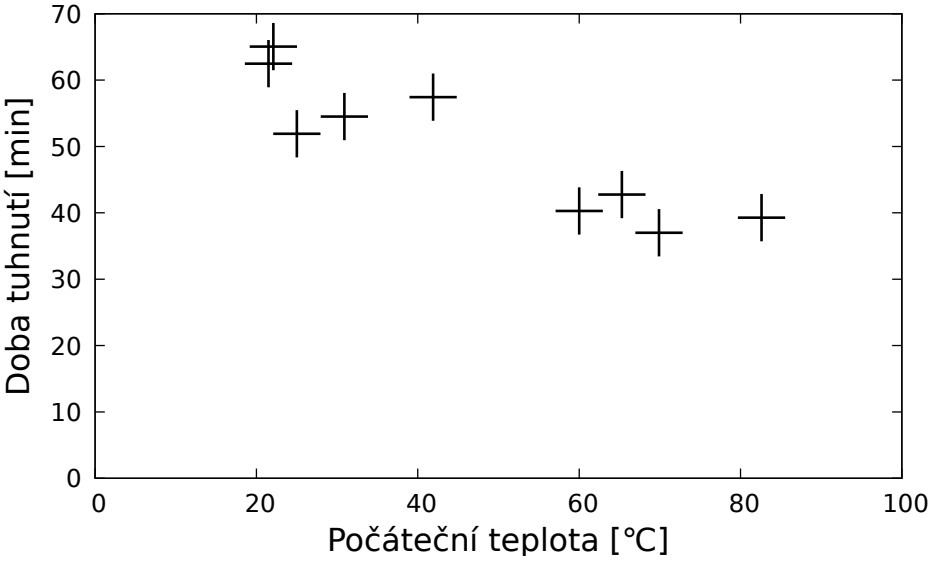
Tabulka 1: Data z experimentů pro závislost doby tuhnutí na počáteční teplotě kapaliny, kde T_0 je počáteční teplota vody, t_z je čas překročení 0 °C a Δm je odpar vody

Můžeme vidět, že s rostoucí počáteční teplotou nám klesá doba, za kterou voda ztuhne. Proč tomu asi tak je? Ono totiž *teplá voda zmrzne rychleji* je velmi zavádějící úsloví. Pojem *teplota* se zde vztahuje k *průměrné teplotě v objemu kapaliny*. Tohoto si všiml ve svém řešení Viktor Materna, který správně podotknul, že teplota v různých místech kapaliny je jiná a tento rozdíl teplot je markantnější u kapaliny s vyšší počáteční teplotou.

[1] Přehlednější grafy v barevné podobě jsou k nahlédnutí v PDF verzi čísla na našem webu.



Obrázek 1: Shrnutí měření průběhu teploty vzorku vody v čase; legenda grafu určuje počáteční teplotu v experimentech znázorněných danou křivkou



Obrázek 2: Výsledek závislosti doby tuhnutí na počáteční teplotě

Zadání 3. série

V minulých číslech jsme experimentálně dokázali teorii mladého fyzika Bartoloměje, který se domníval, že teplá voda zmrzne rychleji než studená.[2] Když jsme podle Bartolomějova vzoru vyzkoušeli naměřit i to, jak se tento jev projevuje na různých kapalinách, před námi vyvstal další problém: jak ovlivňuje průběh tuhnutí okolí? Poté, co Bartoloměj vyzkoušel, jak se tento jev projevuje na ostatních kapalinách, zkoumal dále, jak se na průběhu tuhnutí kapaliny projeví změny v okolí. Nejprve zkusil zaměnit nádoby a naměřit nějaké pokusy s alespoň dvěma nádobami s rozdílnými rozměry ze stejného materiálu. Poté zkusil zaměnit i materiál podloží nádoby[3]. Nakonec se zamyslel, jak ovlivňuje výkon mrazáku dobu tuhnutí.

Jak jste si mohli všimnout ve vzorovém řešení 1. série obsaženém v tomto čísle, průběh a doba tuhnutí závisí na způsobu odvodu tepla z kapaliny. Tento odvod tepla závisí na veličině zvané tepelný tok Q_v vyjadřující průnik množství energie válcovou nádobou se stěnou o tloušťce l přes plochu kolmou na směr proudění tepla S při rozdílu teplot Δt prostředí dotýkajících se vodícího předmětu (zde nádoby) z obou stran. Materiál, který také ovlivňuje snadnost/nesnadnost průtoku tepla předmětem, je zastoupen veličinou λ neboli součinitelem tepelné vodivosti. Vzorec pro výpočet velikosti tepelného toku pro válcovitou nádobu vypadá takto:

$$Q_V = \frac{\lambda S(t - t_{p1})}{l} + \pi \frac{r_2 + r_1}{r_2 - r_1} \lambda h (t - t_{p2}) \tag{1}$$

Doplňme, že $l = r_2 - r_1$ je zároveň rozdíl vnitřního poloměru r_1 a vnějšího poloměru r_2. První sčítanec ve vzorci reprezentuje tepelný tok přes stěnu nádoby a druhý sčítanec reprezentuje tepelný tok přes dno nádoby, které je vlastně rovinná stěna. Pokud budeme opravdu důslední, tak v prvním a druhém sčítanci bude Δt rozdílné, jednou se týká teploty vzduchu v mrazáku a podruhé teploty dna mrazáku.

Problém 1: *Změřte závislost teploty kapaliny na čase pro dvě různé nádoby a poté i pro dvě různá podloží z různých materiálů. Jak materiál nádoby a podloží ovlivňuje průběh tuhnutí? Jak závisí doba tuhnutí na výkonu mrazáku? Odhadněte číselně pro váš konkrétní mrazák. Diskutujte závislost parametrů vaší nádoby a parametrů jejího okolí na velikost tepelného toku. Pokud jste použili nádobu, která nemá válcovitý tvar, vzorec pro výpočet tepelného toku si najděte např. v odborné literatuře či na internetu.*[4]

Problém 2: *V tomto čísle vidíte poslední pevně dané zadání tohoto témátka. Pokud však máte nápady a chuť měřit a otestovat další faktory ovlivňující průběh tuhnutí kapaliny, nebojte se pustit do experimentování a posílejte nám své výsledky. Toto platí do termínu odeslání poslední série úloh 25. ročníku.*

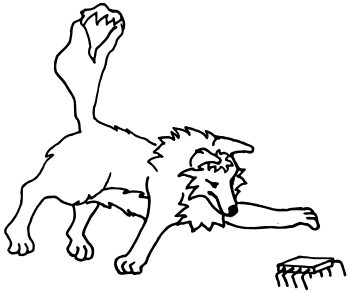
[2] Proč tomu tak je, se můžete dozvědět v řešení 1. čísla.

[3] Podložím rozumíme to, na čem vaše nádoba stojí.

[4] např. studijní text Fyzikální olympiády: `http://fyzikalniolympiada.cz/texty/texttz.pdf`

Vaše nejlepší řešení budou zveřejněna až na konci tohoto ročníku spolu s organizátorskými měřeními výsledků zadání 2. a 3. čísla, abyste měli dost času měřit vše, co bylo navrhnuto a klidně i něco navíc. Pokud máte dotazy či zajímavé připomínky, nebojte se napsat do tématkové konference nebo přímo vedoucím tohoto tématka.

Pája a Matej; pavla.trembulakova@seznam.cz
e-mailová konference: paradoxni@mam.mff.cuni.cz



Téma 2 – Principy kryptografie

Módy blokových šifer

V tomto čísle se vrátíme k blokovým šifrám a budeme řešit otázku, jak pomocí blokové šifry zašifrovat co nejlépe větší množství dat. Tento problém jsme řešili už v prvním čísle u šifry OCTA. Tam jsme zprávu rozdělili na několik kratších a ty zašifrovali nezávisle na sobě. V tomto čísle se zamyslíme, jaké takový postup má nevýhody a jak to případně udělat lépe.

Řešení 1. série

Na začátku bychom rádi připomněli úlohy z prvního čísla a vybrali z došlých příspěvků jejich pěkná řešení.

Úloha 1

Zadání:
Mějme libovolná čísla a a b. Dokážete získat výsledky operací or, xor a rotace vpravo pouze pomocí opakovaného využití operací and a not?

Řešení:
Správných řešení přišla celá řada. Například tak, že si napíšeme tabulku pravdivostních hodnot, můžeme nalézt a následně ověřit, že platí:

$$a \parallel b = \sim ((\sim x) \,\&\, (\sim y))$$

$$a \oplus b = (a \parallel b) \,\&\, (\sim (a \,\&\, b)) = \sim ((\sim x) \,\&\, (\sim y)) \,\&\, (\sim (a \,\&\, b))$$

Operaci rotace vpravo pomocí and a not vyjádřit nedokážeme. U operací and a or, a tudíž i u všech operací z nich složených, je hodnota výsledného bitu vždy závislá pouze na hodnotách bitů stejného řádu (na stejné pozici v binárním zápisu čísla). Možná i právě proto se jim říká bitové operace. U rotace vpravo je ale hodnota výsledného bitu závislá na bitu jiného řádu (obvykle o jedna vyššího).

Úloha 2

Zadání:
Dostali jste zprávu 89 19 2C 8C F9 BC FA CC 6D 6C FC BD 19 59 19 C8 29, o které víte, že je zašifrovaná šifrou OCTA pomocí klíče 4D 41 4E 44 4D (tj. MANDM). Jaká byla původní zpráva?

Řešení:
Druhá úloha vyžadovala akorát správně pochopit schéma šifry OCTA. Mnozí řešitelé se pak pouštěli do dlouhých a složitých manuálních výpočtů. Pochvalu zaslouží Lucie Kunčarová a Doc.MM Kateřina Rosická, které si šifru naprogramovaly. Mohou tak jednoduše dešifrovat i další texty.

Zašifrovaný text byl „VOMBAT CHLUPONOSY“.

Úloha 3

Zadání:
Můžete též zkusit zauvažovat nad volbou šifrovacích klíčů. Existuje klíč, který vámi zvolený otevřený text zašifruje tak, že otevřený i šifrový text budou stejné? A existuje klíč, který zašifruje libovolný otevřený text na šifrový text, který bude s otevřeným textem shodný?

Najdete otevřený text a klíč, který tento text zašifruje na samé 1, tj. FF?

Řešení:
Existuje klíč, který zvolený otevřený text zašifruje tak, že otevřený i šifrový text budou stejné? A existuje otevřený text a klíč, který ho zašifruje na FF?

Stačí si kupříkladu uvědomit, že poslední operace, ke které při šifrování dochází, je přixorování klíče K_6. Můžeme si tedy zvolit libovolný otevřený text i prvních pět rundovních klíčů a spočítat mezistav šifry před přixorováním posledního rundovního klíče. Pak už není těžké dopočítat K_6, abychom dosáhli kýženého výsledku.

Otázku, jestli existuje univerzální klíč, který zašifruje libovolný otevřený text na šifrový text, který bude s otevřeným textem shodný, necháme zatím otevřenou. Za její řešení nabízíme tři body.

Úloha 4

Zadání:
V detailním popisu AES si můžete všimnout, že v poslední rundě je vynechaná permutace MixColumns. Proč tomu tak je? A proč není vynechaná i ShiftRows?

Pokud bychom i v naší šifře OCTA vynechali v poslední rundě permutaci, změnili bychom tím obtížnost jejího prolomení?

Řešení:
Úkolem čtvrté úlohy bylo převážně dohledat na internetu informace o AES. Na detailní popis této šifry zde nemáme dostatek prostoru, takže pouze nastíníme řešení. Úpravy v poslední rundě AES jsou čistě z implementačních důvodů, aby bylo možné pomocí co nejpodobnějšího kódu šifrovat i dešifrovat zprávy. Jsou přitom udělány tak, aby to na vlastnosti šifry nemělo vůbec žádný vliv.

Podobně bychom mohli u šifry OCTA vynechat permutaci v poslední rundě. Pokud bychom navíc adekvátně zpermutovali poslední rundovní klíč, dostali bychom stejný výsledek, akorát zpermutovaný.

Problém

Zadání:
Představme si reálné využití šifry. Chcete mít aktuální přehled o tom, kolik máte doma tabulek hořké čokolády. A tak si každou hodinu posíláte do školy zprávu o jejich počtu. Protože tato informace je ale důvěrná, posíláte ji zašifrovanou pomocí šifry OCTA s klíčem, který znáte jenom vy.

Je k tomuto účelu OCTA vhodná? Pokud někdo bude odposlouchávat vaše zprávy, dokáže z nich něco zjistit? Jak by bylo možné zasílání zprávy vylepšit?

Řešení:
Jako řešení problému otiskujeme bez větších redakčních úprav příspěvek Františka Bujnovského. Pokud vám připadá, že v jeho textu něco chybí nebo s něčím nesouhlasíte, nebojte se ho doplnit vlastním článkem.

Využití šifry OCTA ke kontrole zásob čokolády (4b)

František Bujnovský

Nejdříve je třeba si uvědomit, že je více způsobů, jak počet čokolád zašifrovat. Já bych jmenoval asi tři základní:

1. Doma je 5 tabulek hořké čokolády
2. Pět
3. 5

První způsob poskytuje odposlouchači (dále budu značit *OD*) velké množství materiálu, se kterým by mohl pracovat. Obecně pro *OD* je velice obtížné zjistit, kolik rund má šifra, jaký je přesně klíč, jaká je substituce a další… A tak se bude snažit (alespoň já bych to na jeho místě dělal) přiřadit určitému znaku písmeno. Je obecně známo, že samohlásky se v textu opakují častěji než souhlásky, a např. v tomto textu se opakuje třikrát písmeno *A* a čtyřikrát *O*. Mohl by si tedy natipovat, co je tenhle znak a co zase tenhle (možná by ho trochu mohla zmást mezera – některé šifry mezery nepíšou). Taky je podezřelé, že se mění pouze jeden údaj (pár znaků v tvrzení, zbytek zůstává stejný až občas na nějaké skloňování, ale to je zanedbatelné). Časem by *OD* mohlo dojít, že je to právě číslo, které se

mění a zbytek zůstává stejný. Protože co jiného by se měnilo? Maximálně by to mohl být nějaký název, ale to je tak všechno, co mě napadá. A zde narážíme na nebezpečí číslic. Když se počet čokolád pohybuje do 10, je to v klidu, ale jak se začne zvyšovat, tak se v desítkové soustavě začnou čísla opakovat (jako v každé jiné soustavě, např. 1 se opakuje v číslech 12, 13, 14). Tedy pokud bychom k zašifrování používali pouze čísla zapsaná v číslicích, mohl by *OD* tohle pozorovat. Ze začátku by si mohl myslet (kdyby mu došlo, že je to číslo), že dvouciferné číslo je čtyřciferné číslo (nezašifrovaná 1 je v ASCII jako $(31)_{16}$), ale časem by mu došlo, že se dvojice znaků opakují a že existuje pouze deset druhů dvojic. Pak by začal tipovat a mohl by naši šifru prolomit. Ovšem na rozdíl od abecedy zde neexistují samohlásky a souhlásky, takže nelze tím, že by tam jedno číslo bylo víckrát než jednou, určit, že je to 1.[5] Tedy tenhle způsob je skoro neprolomitelný. Pouze pokud by se počet čokolád měnil vzestupně po pravidelných mezerách (např. +1) až do doby, kdy se k číslu musí přidat další cifra. V tomto případě by se dalo něco vypozorovat – určit, které znaky znamenají jaké číslo, atd.

Obtížnost šifry by se dala zvýšit odstraněním mezer mezi znaky. U naší šifry jde o to, že dva znaky značí jedno písmeno, číslo, mezeru, nebo jiné ($A = 41$ atd.). Pokud tedy necháváme za každou dvojicí mezeru, je skoro jasné, že máme pouze 10 různých číslic. Co kdybychom ale napsali číslo číslovkami? Obávám se, že je to v podstatě to samé, akorát ve větším měřítku. Budou se tu opakovat min. šestičlenné řetězce znaků (3 = tři = 54 52 49). Když *OD* spočítá počet použitých řetězců, mohl by přijít na to, že se jich opakuje pouze 10. Sice by se některá čísla v šifře nemusela vůbec objevit, ale i tak je podezřelé používat pořád stejné řetězce. Jak bychom to ale mohli ztížit? Např. šifře přidáme ještě něco jako +1. Ke všem údajům v pondělí přičteme 1, ke všem v úterý 2, atd. Nebo prostě v nějakém sledu měnit substituční tabulku. Není to nerozluštitelné, ale *OD* poté potřebuje více materiálu na luštění. Čím nepravidelněji by se tento systém měnil, tím těžší by bylo prolomit ho. Ovšem příjemce a odesílatel se v tomto případě musí domluvit, což je tady jednoduché, protože chodím každý den domů. V případě nějakých mezinárodních komunikací se ale systém nemůže moc měnit, protože oznámení o změně se nemůže poslat v šifře, u které hrozí, že byla prolomena, a tak by se nic nezměnilo – šifra by zůstala dál prolomená.

Také by se to dalo ztížit tím, že bychom 30 minut po poslání pravé poslali falešnou šifru, ve které by byly nějaké jiné údaje (možná i napsané jiným systémem).

Když už *OD* šifru prolomí, je dobré, že ze samotného čísla nic nezjistí (tedy pokud nenapíšeme „Doma je 5 tabulek čokolády“). Může si myslet, že je to počet králíků, peníze, heslo nebo jiné...

K řešení správné šifry by ale bylo dobré vytvořit si tabulku ($A = 41$, B = 42, atd.), nebo naprogramovat si program, protože přepočítání každého znaku zvlášť je časově náročné. V tomto případě by ale samozřejmě mohl *OD* tabulku ukradnout, takže nejlepší bude si ji udržet v paměti.

[5] *Poznámka redakce: Toto tvrzení obzvláště doporučujeme vaší pozornosti. Souhlasíte s ním?*

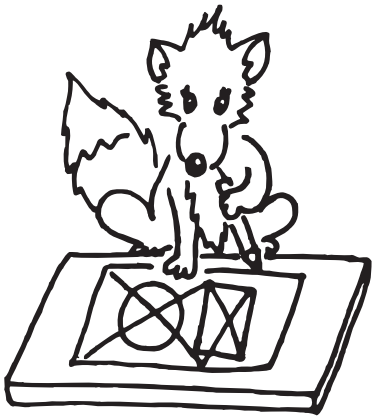
Ještě bychom mohli změnit tabulku, ale jak už jsem říkal, je blbost zkoušet různé matematické operace, aby nám vyšla ona písmena, protože je strašně mnoho možností. Daleko lehčí je prostě jenom tipovat znaky, které se opakují. Měnit tabulku tedy moc nemá smysl, pokud náhodou *OD* nepoužívá nějaký program, který zkusí několik tisíc možností za sekundu, což je mezinárodně možné, ale v tomto případě spíše nepravděpodobné.

Zadání 3. série

Šifrování delších textů

Vraťme se ke zkoumání možností, jak zašifrovat delší texty. Způsobům, jak pomocí blokové šifry s pevnou délkou bloku zašifrovat delší zprávu se říká módy, ve kterých je tato šifra použita.

Zatím jsme vždy zprávy rozdělili na několik kratších a ty šifrovali nezávisle na sobě. Tomuto postupu se říká, že šifrujeme v módu *Electronic Codebook (ECB)*. Příspěvek Františka Bujnovského ukazuje na zásadní nedostatek: pokud se nějaké bloky vstupního textu opakují, opakují se ty samé bloky i v šifrovém textu. Speciálně u šifry OCTA tedy šifrování pomocí ECB odpovídá permutaci znaků.

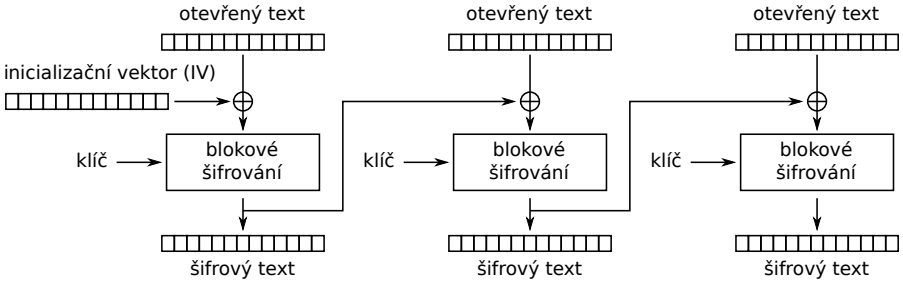


Problém 1 [3b]: *Vyberte si libovolný bitmapový obrázek (například formát BMP) a zašifrujte jeho obrazová data (tedy vše kromě hlaviček) pomocí šifry OCTA s klíčem MANDM (t.j. 4D 41 4E 44 4D) v módu ECB. Měl by opět vzniknout zobrazitelný obrázek. Jak se od toho původního liší? Původní obrázek i jeho zašifrovanou podobu nám zašlete.*

Pokud si nechcete hrát s obrázkovými formáty, tak si vzorový obrázek včetně popisu, jakou oblast je záhodno zašifrovat, můžete stáhnout z našeho webu ze stránky věnované tématku[6].

[6] `https://mam.mff.cuni.cz/problem/2203`

Proto se v praxi používají spíše jiné módy. Asi nejrozšířenějším je *Cipher Block Chaining (CBC)*. V tomto módu se k otevřenému textu vždy nejdříve přixoruje předchozí zašifrovaný blok a výsledek se následně zašifruje. Protože při šifrování prvního bloku žádný předchozí blok není k dispozici, vygeneruje se náhodný text o velikosti bloku, kterému se říká *inicializační vektor*, a místo předchozího zašifrovaného bloku se použije ten. Hodnota inicializačního vektoru se pak uloží společně se zašifrovanými daty. Schématicky je CBC mód zachycen na Obrázku 3.



Obrázek 3: Šifrování pomocí CBC módu

Problém 2 [3b]: *Zašifrujete obrázek z Úlohy 1 také pomocí šifry OCTA se stejným klíčem v CBC módu. Jaký je výsledek tentokrát? Opět nám oba obrázky pošlete.*

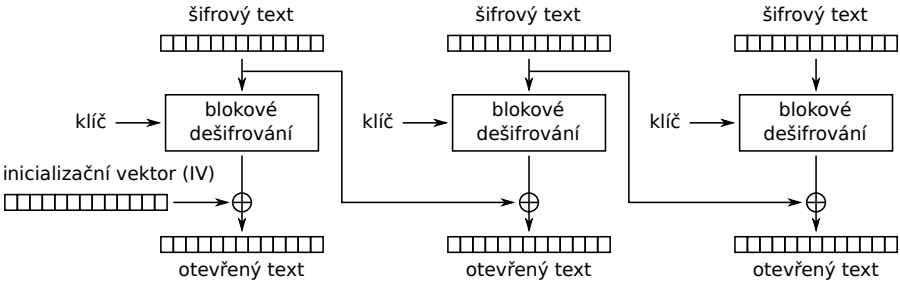
Můžeme si všimnout, že nyní úprava jednoho bloku ovlivní i všechny následující bloky. Speciálně volba inicializačního vektoru má vliv na všechny bloky šifrového textu. Pokud tedy zašifrujeme ta samá data s jiným inicializačním vektorem, dostaneme zcela odlišný šifrový text. Obvykle se volí šifrovací vektor nový pro každé šifrování. Takže když zašifrujeme vícekrát ta samá data, nebude to na šifrových textech (snadno) poznat.

Dešifrování dat zašifrovaných pomocí CBC módu probíhá obráceně než šifrování, tedy nejdříve dešifrujeme šifrový text a následně přixorujeme předchozí blok šifrového textu, případně inicializační vektor. Proces dešifrování zachycuje Obrázek 4.

Problém 3 [2+2b]: *Pokud zašifrujeme hodně dat pomocí šifry s malou velikostí bloku, musí se jednou stát, že dostaneme dva stejné bloky šifrového textu.*

Plynou z toho v případě využití CBC módu pro šifrování nějaká bezpečnostní rizika?

Kolik textu musíme průměrně zašifrovat blokovou šifrou s bloky délky 8 (to je například OCTA), než dostaneme dva stejné bloky šifrového textu? Předpokládejte, že máte náhodný vstupní text a ideální blokovou šifru.



Obrázek 4: Dešifrování dat zašifrovaných pomocí CBC módu

Problém 4 [2b+]: *Nevýhodou CBC módu je, že k zašifrování bloku potřebujeme znát hodnotu předchozího bloku šifrového textu. Nemůžeme tedy výpočty provádět paralelně. Dokážete dohledat nebo vymyslet mód, který bude zachovávat výhody CBC módu a přitom bude umožňovat výpočty provádět (alespoň částečně) paralelně?*

Pokročilejší módy

Občas od módu blokové šifry nechceme pouze, aby zamaskoval situace, kdy máme v otevřeném textu více stejných bloků nebo kdy posíláme vícekrát tu samou zašifrovanou zprávu.

Existují pokročilejší módy, pomocí kterých je možné například ověřit, že zašifrovanou zprávu nikdo neupravoval. K těmto módům se ještě ke konci témátka dostaneme. Pokud by se ale někomu chtělo si nějaké najít a napsat o nich pěkný článek, rádi tento text uveřejníme a autora adekvátně bodově oceníme.

Kuba, Káťa a Lenka; `jakub.topfer@matfyz.cz`
e-mailová konference: `krypto@mam.mff.cuni.cz`

Téma 3 – Neznámý měsíc

Řešení 1. série

Na riešenie všetkých úloh sa dajú použiť primitívne vyrobené pomôcky. Väčšinu postupov ľudia objavili už počas antického Grécka.

Úloha 1

Zadání:
Zaveďte navigační systém a navrhněte nástroje a metody pro navigaci.

Řešení:
Z poznatku, že mesiac nemá viazanú rotáciu, nám pomôže cestujúca planéta na oblohe ako orientačný bod. Z jej pohybu po oblohe si budeme môcť vytvoriť svetové strany podobne ako z hviezdy cestujúcej po oblohe. Vedeli by sme určiť smer východu a západu planéty, ale to iba v prípade, že sme na strane otočenej k planéte. Taktiež orientácia podľa hviezd pripadá v úvahu počas neprítomnosti hviezdy a planéty na oblohe.

Kompas by nám v tomto prípade veľmi pomohol a to aj primitívne vyrobený. V prípade prítomnosti magnetického poľa z mesiaca by sme mohli zaviesť svetové strany podľa neho. Ak by toto magnetické pole bolo slabé alebo neexistujúce, vieme že väčšina plynných obrov má magnetické pole a mohli by sme zaviesť svetové strany podľa magnetického poľa planéty. Bol by to komplikovanejší systém, ale bol by použiteľní.

Na orientáciu na krátku vzdialenosť by sme mohli využiť význačne body a to vysoké hory, divne tvarované skaly a prípadnú vegetáciu. Vegetácia by mohla poskytovať tiež určenie svetových strán ako je to tu na Zemi.

Úloha 2

Zadání:
Zjistěte gravitační zrychlení, velikost, parametry rotace a oběžné dráhy měsíce a planety.

Řešení:
Gravitačné zrýchlenie: Najjednoduchší experiment je volný pád na známej vzdialenosti, kde využijeme jednoduchý vzorček:

$$s = \frac{1}{2}gt^2,$$

kde s je dĺžka volného pádu, g gravitačné zrýchlenie a t čas pádu.

Zložitejší ale presnejší experiment by bolo použitie kyvadla pri malých uhloch kmitania. Matematické kyvadlo určené vzťahom:

$$T = 2\pi\sqrt{\frac{l}{g}}$$

kde T je perióda a l dĺžka kyvadla. Pre lepšie výsledky je možné počítať s fyzikálnym kyvadlom:

$$T = 2\pi\sqrt{\frac{J}{mgl}}$$

kde J je moment zotrvačnosti kyvadla, m hmotnosť kyvadla a l je dĺžka kyvadla od osi rotácie po ťažisko.

Kyvadla sú jednoduché na zostrojenie, ale ak by sme napr. chceli zistiť dobu rotácie okolo vlastnej osi mohli by sme zostrojiť Foucaultovo kyvadlo. Ide o veľmi dlhé kyvadlo, ktoré na póloch za jednu rotáciu mesiaca okolo vlastnej osi, úplne otočí svoju rovinu kmitov v smere alebo protismere hodinových ručičiek. Tento pohyb by sme nepozorovali na rovníku a na 30 ° šírky by sme pozorovali otočenie osi kmitov iba o 180 °. Riadi sa funkciou *sin* zo zemepisnej šírky. Pravdaže museli by sme mať zistenú najlepšie polohu pólov, aby sme mohli vykonať experiment. Taktiež by sme vedeli určiť polohu rovníku – kyvadlo by sa tam nestáčalo a pomohlo by nám to s orientáciou. A ak by sme mali určenú dobu rotácie, vedeli by sme z kyvadla určiť polohu na zemepisnej šírke. Ako by to bolo praktické so sebou nosiť napr. 18 kg závažie na 21 m šnúre ako Foucaultovo kyvadlo nachádzajúce sa v budove matfyzu Ke Karlovu 5 je otázne.

Veľkosť a obežné dráhy mesiaca a planéty:

Veľkosť mesiaca je možne vypočítať zo zakrivenia povrchu. Ak nájdeme dostatočne „rovnú“ plochu alebo vodnú hladinu, môžeme vypočítať veľkosť mesiaca pomocou:

$$d = \sqrt{h(2R + h)}$$

$$s = R\cos^{-1}\left(\frac{R}{R + h}\right)$$

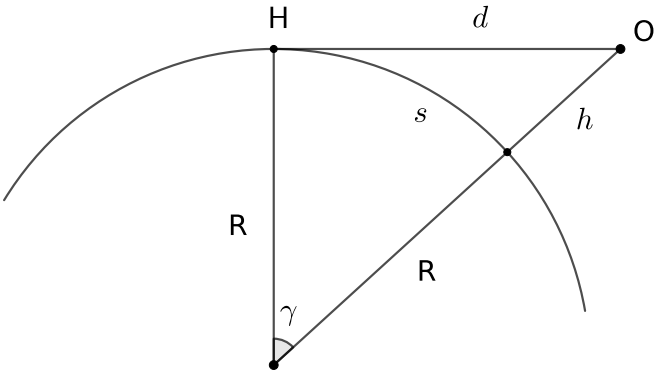
kde d je vzdialenosť pozorovateľ–objekt, h výška objektu, R polomer a s vzdialenosť na kruhovej ploche (obrazok 5).

Taktiež by sa dali použiť tiene vrhnuté rovnakým objektom v rovnaký čas na iných miestach. To by bolo dosť nepraktické pre jedného človeka. Taktiež chceme aby miesta boli kolmo (severne alebo južne) na smer pohybu hviezdy. Ale ak by človek s istotou vedel o mieste, kde v daný čas nebudú objekty vrhať tieň, teda svetlo bude dopadať kolmo, bude mu stačiť odmerať dĺžku tieňu, z ktorého zistí uhol γ pri vrchole predmetu (predpokladáme, že vieme dĺžku predmetu, ktorý vrhá tieň). Ďalej potrebuje vedieť vzdialenosť miest merania. Za predpokladu, že dopadajúce lúče sú paralelné, bude nameraný uhol rovnaký ako uhol v strede mesiaca medzi danými miestami. Zo vzorcu pre časť obvodu kružnice určíme polomer mesiaca.

$$R = \frac{s}{2\pi} \cdot \frac{360}{\gamma}$$

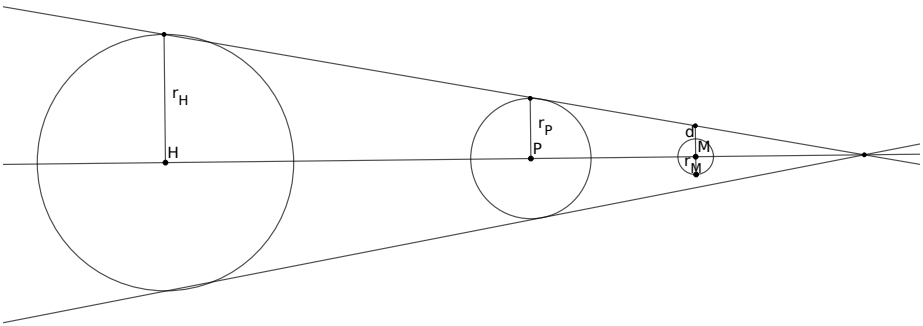
Grécky astronóm a matematik Aristarchusve vo svojej práci[7] vypočítal polomer Mesiaca, Slnka a veľkosť obežných dráh Mesiacu a Zeme okolo Slnka ako

[7] `https://en.wikipedia.org/wiki/On_the_Sizes_and_Distances_(Aristarchus)`



Obrázek 5: Nákres pre výpočet polomeru mesiaca

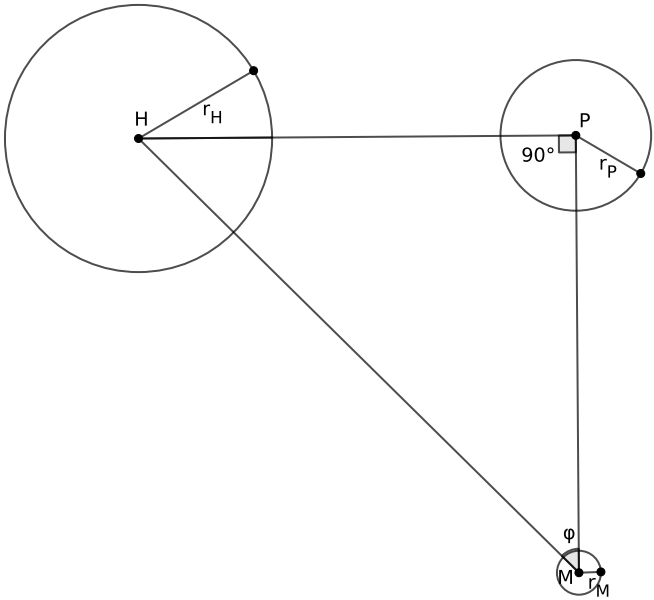
násobky Zemského polomeru alebo polomeru Mesiaca. Využíva dvoch postavení Slnko–Zem–Mesiac. Keď je mesiac v tieni Zeme (zatmenie Mesiaca) a postavenia v pravouhlom trojuholníku Slnko–Zem–Mesiac, keď je pravý uhol v Mesiaci (polovičný Mesiac). Bude potrebné presné meranie uhlov. Aristarchos sa dostal na rádový odhad, kvôli zanedbaniam v geometrii a nepresnosťou merania uhlov pod 1 °. Pre výpočet je potrebné zistiť 3 uhli, a to uhol Slnko–Zem–Mesiac pri trojuholníkovom usporiadaní φ a uhlovú veľkosť Slnka a Mesiacu θ. Nakoniec veľkosť polomeru Zemského tieňu d v priemery Mesiacu. To je možné vypočítať zo známej obežnej doby Mesiaca a času, po ktorý v nom bude schovaní.



Obrázek 6: Usporiadanie pri zatmení mesiaca

V našom prípade, bude počítať s polovičnou planétou, čo nám oproti pôvodnému príkladu nezmení výpočet z uhlu φ. Z uhlu φ zistíme pomer vzdialenosti planéta–hviezda PH ku vzdialenosti planéta–mesiac PM:

$$\frac{PH}{PM} = \tan(\varphi)$$



Obrázek 7: Usporiadanie pri polovičnej planéte

Po využití podobnosti trojuholníkov pri zatmení mesiaca dostaneme pomer polomeru hviezdy a mesiaca:

$$\frac{PH}{PM} = \frac{r_H}{r_M} = x$$

$$n = \frac{d}{r_M}$$

Kde x a n použijeme na zjednodušenie ďalších vzťahov. Nakoniec dostaneme pomery, ak si upravíme rovnice o podobnosti trojuholníkov:

$$\frac{r_M}{r_P} = \frac{1+x}{x(1+n)}$$

$$\frac{r_H}{r_P} = \frac{1+x}{1+n}$$

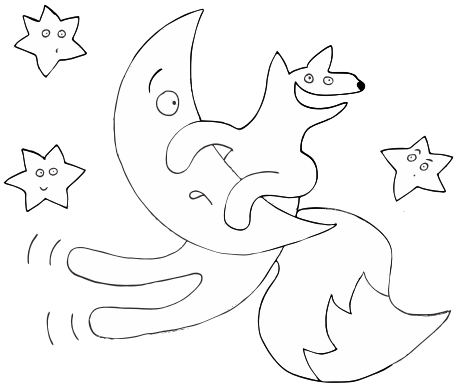
$$\frac{PM}{r_P} = 1 + \frac{1}{\mathrm{tg}(\frac{\theta_P}{2})} \approx \frac{360}{\pi\theta_P}$$

$$\frac{PH}{r_P} = \frac{r_H}{r_P}(1 + \frac{1}{\mathrm{tg}(\frac{\theta_H}{2})}) \approx \frac{r_H}{r_P}\frac{360}{\pi\theta_H}$$

kde r s indexami H, P a M sú polomery hviezdy, planéty a mesiaca. Uhly θ sú uhlové veľkosti planéty a hviezdy. Predpokladáme, že z planéty by sme namerali rovnakú uhlovú veľkosť hviezdy. Aproximácia platí iba pri malých uhloch.

Dobu obehu planéty mesiacom je jednoduché určiť z periodického striedania novu a splnu u Mesiaca. Ak odmeriame dobu medzi dvomi novmi alebo splnmi planéty, zistíme tak v prípadě nášho neznámého mesiaca dobu obehu okolo planéty. Dobu rotácie okolo vlastnej osi určíme jednoducho ak z rovnakého miesta na mesiaci uvidíme planétu v rovnakej pozícii na oblohe, a tedy napr. čas medzi dvomi západmi planéty alebo hviezdy za horizontom.

Určenie obežného času bude zložitejšie v prípade planéty. Museli by sme pozorovať polohu hviezdy a hviezdnej oblohy. Ak by sme videli zase rovnaké usporiadanie, vedeli by sme odhadnúť obežnú dráhu planéty. Mohli by sme sledovať hviezdnu oblohu v rovnakej polohe hviezda–planéta–mesiac, teda v tieni planéty. Chyba odhadu obežnej doby planéty by bola max. pomer obežných časov mesiaca a planéty. Pravdaže s veľkým počtom meraní by sme túto hodnotu vedeli určiť presne. Pre zistenie rotačnej doby planéty budeme potrebovať obežnú dobu mesiaca. Planéta má výrazné búrky ako Jupiter, čo nám dosť pomôže. Ak by sme túto búrku pozorovali na rovnakom mieste po určitej dobe, museli by sme od tejto doby odpočítať/pripočítať podľa smerov rotácie a obehu čas alebo uhol, ktorý mesiac obehol okolo planéty, za predpokladu, že búrky sa na povrchu nepohybujú.



Problém

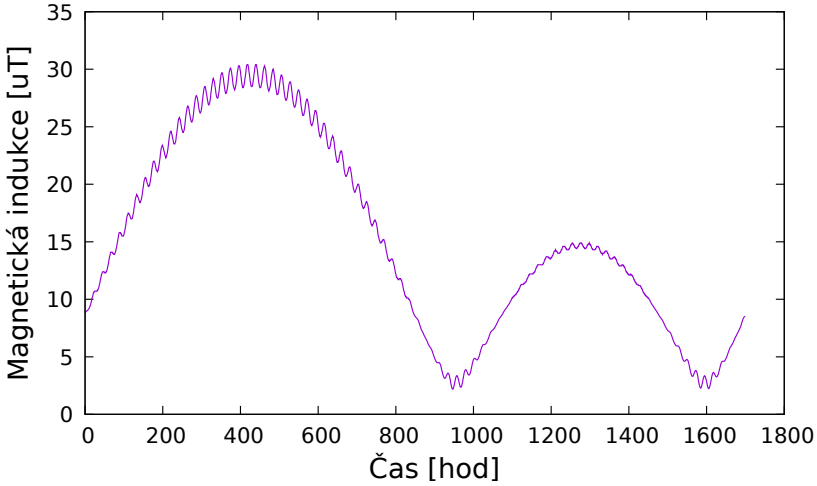
Zadání:
Můžete určit ještě něco jiného pomocí obecně dostupných předmětů – např. kdybyste měli lékárničku či nějakou kosmetiku?

Řešení:
V bežnej kozmetike a lekárničke sú rôzne chemické zlúčeniny. Nevedeli by ste pomocou nich určiť prítomnosť iných prvkov a chemických zlúčenín v pôde alebo v atmosfére? V zadaní máme, že atmosféra je rovnako veľká a hustá, ale nemusí mať rovnaké zloženie. Prítomnosť kyslíku je jednoduché určiť, ak zapálime nejakú horľavú vec. Taktiež by sme mohli určiť podľa veľkosti plameňa a rýchlosti horenia či je obsah kyslíku menší, porovnateľný alebo väčší ako u nás na Zemi. Ak by

atmosféra neobsahovala kyslík a pri malom vypustení kyslíku do atmosféry a iskre by nastalo vzplanutie vedeli by sme, že atmosféra pravdepodobne obsahuje metán a rovnakým spôsobom by sa dal potvrdiť aj prítomnosť H_2. Metán bol nájdení na Marse a veľkých mesiacoch v Slnečnej sústave. Určite existuje pomaly nekonečno možných chemických reakcii na dokázanie prítomnosti rôznych zlúčenín v pôde a atmosfére, preto by som nechal tento problém ešte stále otvorený.

Zadání 3. série

Při průzkumu měsíce jste nalezli další úlomky své lodi. Jeden obsahoval digitální magnetometr. Nechali jste ho na své základně, která je na rovníku měsíce, a zaznamenávali jste po dobu jednoho oběhu kolem planety amplitudu magnetické indukce (viz obrázek 8).



Obrázek 8: Měření magnetické indukce po dobu jednoho oběhu kolem planety

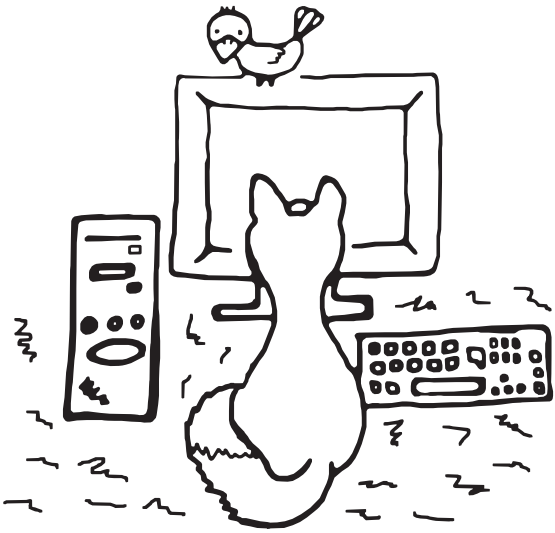
Vnější magnetické pole planety můžete modelovat pomocí dvou pólů, z nichž každý leží v polovině úsečky mezi středem planety a jedním z jejích rotačních pólů. Jeden pól bude kladný, druhý záporný. Magnetickou indukci B v určitém bodě pak počítáme stejně, jako bychom počítali intenzitu elektrického pole, kdyby to byly dva kladné náboje. Můžeme ale zanedbat konstanty, protože „magnetický náboj“ pólů Q nemá žádnou fyzikální reprezentaci. Počítáme tedy $B = \sum \frac{Q}{r^2}$, kde r je vzdálenost magnetického náboje od místa výpočtu indukce.

Co o planetě a měsíci víte předem: poloměr měsíce $R_\mathrm{m} = 5\,000\,\mathrm{km}$, poloměr planety $R_\mathrm{o} = 110\,000\,\mathrm{km}$, poloměr oběhu měsíce $a_\mathrm{m} = 400\,000\,\mathrm{km}$, sklon rotační osy planety vůči oběžné dráze měsíce $\theta = 25°$, perioda rotace měsíce 22 hod, perioda oběhu měsíce 1 700 hod.

Problém 1 [2b]: *Vysvětlete, čím je způsobený tvar grafu na obrázku 8.*

Problém 2 [5b]: *Určete amplitudu magnetické indukce na pólech měsíce, pokud zanedbáte pole způsobené planetou, a amplitudu magnetické indukce na pólech planety, pokud zanedbáte pole způsobené měsícem.*

Kuba a Kubo; kusnir.jk@gmail.com
e-mailová konference: mesic@mam.mff.cuni.cz



Téma 4 – Algoritmy od nuly (do n)

Grafové algoritmy

Zadání 3. série

Již potřetí vás vítáme u našeho informatického tématka. Opět připomínáme, že tématko velmi úzce navazuje na předchozí díly, že je určeno pouze pro řešitele, kteří nejsou dobře obeznámeni s koncepty, které se společně snažíme vymyslet, a že sepsané řešení by vždy mělo obsahovat slovní popis algoritmu a určení časové složitosti, pokud není v zadání řečeno jinak.

V dnešním dílu budeme vymýšlet grafové algoritmy. Než se do nich ale pustíme, budeme si muset krátce říct něco o tom, jak v počítači ukládat informace, abychom k nim dokázali rychle přistupovat (což je mimochodem jedním z hlavních úkolů teoretické informatiky).

Datové struktury

Datová struktura je určitý způsob ukládání dat v počítači. Z vnějšího pohledu nás zajímá především to, jaký druh dat můžeme do datové struktury ukládat (často třeba budeme chtít, abychom ukládané prvky uměli porovnávat) a jaké operace na nich můžeme provádět (to může být třeba vyhledávání nebo nalezení nejmenšího prvku). V závislosti na tom, jak datová struktura uvnitř funguje, potom mají různé operace různou časovou složitost. To zní dost abstraktně, proto si hned ukážeme příklady dvou jednoduchých datových struktur.

Zásobník je datová struktura podporující dvě operace – *push* a *pop*. Můžeme si ho představit jako srovnanou hromádku knih. Když chceme knihu na hromádku přidat, přidáme ji vždy úplně nahoru. To odpovídá operaci push, která vloží prvek do zásobníku. Pokud chceme knihu odebrat, vezmeme si samozřejmě také tu úplně nahoře (jinak by nám hromádka spadla). Tak funguje operace pop – vrátí nám poslední přidaný prvek, který jsme ještě neodebrali a odebere ho ze zásobníku.

Implementace zásobníku je jednoduchá – vytvoříme si dostatečně dlouhé pole (jeho délka je daná maximálním počtem prvků v zásobníku, který předpokládáme, že známe předem) a proměnnou `delka`, která nám bude říkat, kolik je v zásobníku aktuálně prvků. Operace push vždy uloží vkládaný prvek na pozici `delka + 1` a zvětší proměnnou `delka` o jedna. Operace pop naopak vrátí prvek z pozice `delka` a zmenší proměnnou `delka` o jedna. Náš zásobník tedy umí operace push i pop v konstantním čase.

Fronta je zásobníku dost podobná. Operace přidání a odebrání se obvykle nazývají *enqueue* a *dequeue* (čti /mˈkjuː/ a /diːˈkjuː/) a fungují tak, jak byste od poctivé fronty čekali. Enqueue tedy přidá prvek na konec fronty, dequeue odebere prvek ze začátku a vrátí ho.

Frontu můžeme implementovat třeba *spojovým seznamem*. Prvky budeme mít nyní uložené na různých místech v paměti a u každého z nich si budeme pamatovat[8], kde leží následující prvek, případně že je daný prvek poslední (v podstatě máme pro každý prvek pole délky 2, ve kterém je na první pozici uložena hodnota prvku a na druhé adresa následujícího prvku). Kromě toho si budeme v proměnných `zacatek` a `konec` pamatovat, kde leží první a poslední prvek fronty. Operace enqueue tedy vytvoří pole velikosti 2, uloží do něj nový prvek, upraví informaci u posledního prvku fronty, aby odkazoval na toto nové pole, a na závěr změní proměnnou `konec`, aby ukazovala na nový konec fronty. Operace dequeue se podívá na první prvek fronty, upraví proměnnou `zacatek`, aby odkazovala na druhý

[8] Pokud jste nečetli poslední část prvního dílu našeho tématka, tak věřte, že každá proměnná nebo pole leží někde v paměti, kterou si můžeme představit jako nekonečnou posloupnost čtverečků (*paměťových buněk*), které jsou očíslované celými čísly. Pole je prostě souvislý úsek buněk, zatímco proměnná je jedna konkrétní buňka. Pole i proměnné mají tedy vždy svou *adresu* – je to číslo jejich první buňky. Tuto adresu si tedy můžeme jako jakékoliv jiné celé číslo někam uložit, abychom věc na adrese zase znova našli, až ji budeme potřebovat. Takovéto uložené adrese říkáme *pointer* (česky *ukazatel*), protože „ukazuje“ na dané místo v paměti. Pojmenované proměnné jsou ve skutečnosti jen přezdívky za tyto ukazatele – kód s proměnnými pojmenovanými celými čísly by totiž byl velmi těžko čitelný.

prvek fronty, a první prvek fronty smaže.

Datové struktury samozřejmě můžou být mnohem složitější a mocnější, necháme si je ale do některého z dalších dílů tématka a raději se pustíme do slibovaných grafových algoritmů.

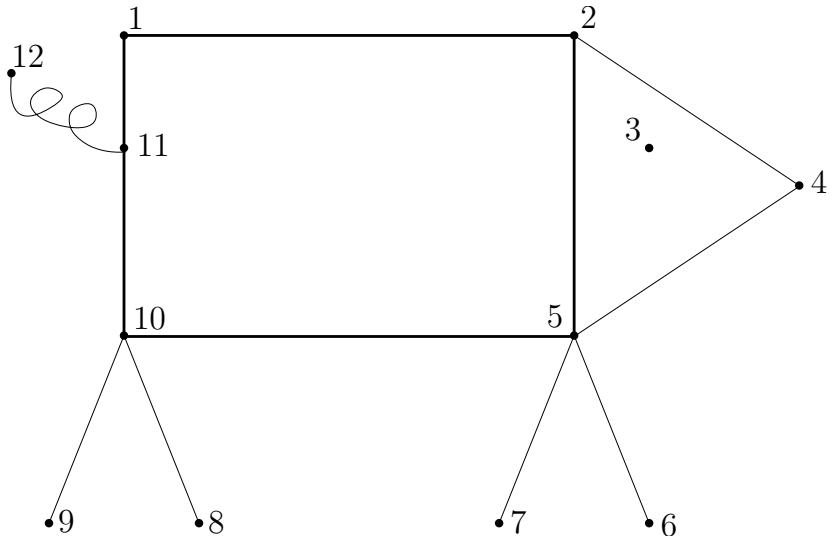
Teorie grafů

Pokud si pod pojmem *graf* představíte graf funkce v souřadném systému, můžete na tuto představu pro dnešek zapomenout. Graf z teorie grafů totiž vypadá úplně jinak – můžete si ho představit jako puntíky nakreslené na papíře, z nichž některé jsou spojeny čarami. Příkladem grafu je tedy třeba strom z minulého dílu našeho tématka nebo obrázek 9. Formálněji řečeno je graf množina *vrcholů* (značíme ji V) a množina jejich (neuspořádaných) dvojic, kterou značíme E. Těmto dvojicím vrcholů říkáme *hrany*. Vrcholy a hrany tedy odpovídají puntíkům a čarám v nakresleném grafu. Graf na obrázku 9 má tedy 12 vrcholů a 12 hran.

Počet vrcholů grafu obvykle značíme n, počet hran m (bude se nám to hodit, až budeme vyjadřovat časovou složitost grafových algoritmů). O vrcholech, které jsou s vrcholem spojeny hranou, říkáme, že jsou jeho *sousedy*. Počet sousedů vrcholu (což je počet hran z něj vedoucích) je *stupeň vrcholu*. *Cesta* v grafu je posloupnost vrcholů taková, že mezi každými dvěma po sobě jdoucími vrcholy vede hrana a žádný vrchol není v cestě obsažen více než jednou. Můžeme si to představit tak, že chodíme po grafu a zapisujeme si, kterými vrcholy jsme prošli, přičemž si dáváme pozor, abychom žádným vrcholem neprošli vícekrát. *Délkou cesty* myslíme počet jejích hran (tedy počet vrcholů mínus jedna) a *vzdálenost* dvou vrcholů je délka nejkratší cesty mezi nimi. Pokud mezi každými dvěma vrcholy v grafu vede cesta, říkáme, že je graf *souvislý*. Graf na obrázku 9 tedy souvislý není, protože mezi vrcholy 3 a 4 nevede cesta (v takovém případě říkáme, že vzdálenost vrcholů je ∞). *Cyklus* (někdy též *kružnice*) v grafu je cesta na alespoň třech vrcholech, jejíž koncový a počáteční vrchol jsou sousedi (cyklu ale náleží i hrana, spojující začátek a konec cesty). *Délka cyklu* je počet jeho hran, tedy délka příslušné cesty + 1. Na obrázku 9 je vyznačen cyklus délky 5 na vrcholech 1, 2, 5, 10 a 11. Pokud graf neobsahuje žádný cyklus a je souvislý, říkáme mu *strom* (setkali jsme se s ním v minulém dílu našeho tématka) Někdy se hodí označit ve stromě jeden z vrcholů jako *kořen*. Soused vrcholu, který leží na cestě ke kořeni je potom jeho *otec*, zbylí sousedé jsou jeho *synové* (viz obrázek v minulém dílu).

Jako informatiky by nás ještě mělo zajímat, jak takový graf reprezentovat v počítači.

Problém 1 [3b]: *Vymyslete, jak do paměti uložit graf, abychom pro každý jeho vrchol mohli v čase $\mathcal{O}$(počet jeho sousedů) najít všechny jeho sousedy a abychom zároveň uměli do grafu v konstantním čase přidat hranu či vrchol (tím už dokážeme zároveň celý graf v čase $\mathcal{O}(n+m)$ vyrobit, pokud nám někdo postupně dává jeho vrcholy a hrany). Nemusíte ošetřovat případy nevalidních vstupů, jako je třeba přidání již existujícího vrcholu.*



Obrázek 9: Graf

Otázka, kterou byste si nyní měli položit, je: A k čemu je to všechno dobré? Odpovědí je samozřejmě příliš mnoho na to, abychom je zde vypisovali – grafové problémy ve skutečnosti můžeme najít v mnoha oblastech běžného života (a navíc na ně dokážeme převést – a tím vyřešit – spoustu problémů, které na první pohled nemají s grafy nic společného). Jedna z očividných aplikací je, že si můžeme všechny silnice a křižovatky (v daném městě, státě, či na celém světě) představit jako graf. Hledání nejkratší cesty mezi dvěma místy je potom vlastně problém z teorie grafů (když zanedbáme jednosměrky, zákazy odbočení a další…)

Počítání vzdáleností

Aby nejkratší cesta v grafu odpovídala nejkratší cestě ve skutečnosti, měli bychom ještě nyní každou hranu grafu *ohodnotit* délkou odpovídající cesty v realitě. Délku cesty v grafu bychom pak nepočítali jako počet hran, ale jako součet ohodnocení všech hran na cestě. Pro dnešek si ale problém zjednodušíme a bude nás skutečně zajímat pouze počet hran. Budeme hledat nejkratší cesty z daného vrcholu v do všech zbylých vrcholů grafu (najít nejkratší cestu mezi dvěma konkrétními vrcholy stejně v nejhorším případě nejde o nic rychleji). Uděláme to tak, že nejdříve najdeme všechny vrcholy ve vzdálenosti 1 od v. To umíme jednoduše – jsou to prostě jeho sousedi, které máme u vrcholu v uložené. Jak ale pokračovat pro vrcholy ve vzdálenosti 2? Pokud budeme umět najít pro každé k všechny vrcholy ve vzdálenosti k od vrcholu v, je tím náš problém vyřešen.

Problém 2 [2b za algoritmus, 1b za složitost]: *Vymyslete způsob, jak pro každý vrchol grafu spočítat jeho vzdálenost od daného vrcholu v.*

Jakou časovou složitost má váš algoritmus? První úvaha nad nějakým problémem často vede na velmi pomalý algoritmus. V takovém případě bývá dobré položit si otázky: „Proč je algoritmus tak pomalý?“, „Čím tráví zbytečně mnoho času?“, „Nedělá něco zbytečně vícekrát?“ Odpověď nám může pomoct vymyslet nějaké vylepšení, které dá vzniknout algoritmu s rozumnou časovou složitostí.

Jedno z takových vylepšení se nazývá *memoizace*. Tato metoda spočívá v tom, že pokud v našem programu často voláme nějakou funkci, uložíme si vždy po jejím vyhodnocení, s jakými parametry jsme ji volali a jaký nám vrátila výsledek. Při každém volání si pak zkontrolujeme, jestli jsme funkci už se stejnými parametry někdy nevolali. Pokud ano, nemusíme ji znova volat – použijeme dříve uložený výsledek. Pro rozumně velké hodnoty parametrů (velké polynomiálně ve velikosti vstupu – tedy velké $\mathcal{O}(n^c)$ pro nějaké fixní c nezávislé na n) toto lze udělat v průměrném čase $\mathcal{O}(1)$.

Memoizaci můžeme použít třeba v algoritmu na hledání Fibonacciho čísel z minulého dílu našeho tématka[9]. Zatímco algoritmus, který jsme v druhém čísle předvedli, měl exponenciální časovou složitost, využitím tohoto jednoduchého triku dostaneme lineární čas. Proč lineární? V jednom zavolání funkce trávíme konstantně mnoho času a funkci voláme jen pro hodnoty menší nebo rovny n – pro každou maximálně jednou. Pokud tedy umíme uložený výsledek volání najít v konstantním čase, dává nám to složitost $\mathcal{O}(n)$.

Problém 3 [3b]: *Zkuste vylepšit memoizací časovou složitost vašeho řešení problému 2. Nezapomeňte říct, jakou datovou strukturu použijete k ukládání již spočítaných výsledků funkce při memoizaci a zohlednit to v analýze časové složitosti.*

Problém 4 (triviální) [0,5b]: *Najde váš algoritmus nejkratší cestu i v ohodnoceném grafu? (V takovém případě myslíme délkou cesty součet ohodnocení jejích hran, viz výše.) Pokud ano, vysvětlete proč. Pokud ne, najděte protipříklad.*

Prohledávání do hloubky

Dalším algoritmem, nad kterým se v této sérii budeme zamýšlet, bude takzvané *prohledávání do hloubky*. Anglicky se mu říká Depth First Search (tedy „prohledávání nejdříve do hloubky“), zkráceně DFS. Je to druhý prohledávací algoritmus, kterým se dnes zabýváme, spolu s prohledáváním do šířky, které jste si vymysleli v problémech 2 a 3 (anglicky Breadth First Search – tedy „prohledávání nejdříve do šířky“ – zkracováno BFS). Jak už název naznačuje, místo toho, aby prohledávalo graf rovnoměrně do všech stran jako BFS, půjde „někam“ (do hloubky) dokud bude moci, a když dojde do slepé uličky, bude se vracet. Poprvé byl tento algoritmus popsán francouzským matematikem jménem Charles Pierre Trémaux v 19. století, který ho navrhl jako způsob hledání cest v bludištích.

Během prohledávání grafu si budeme pro každý vrchol pamatovat jeho stav – tedy jestli je *nenavštívený*, *otevřený* nebo *uzavřený*. Pro každý otevřený vrchol si navíc budeme pamatovat, ze kterého vrcholu jsme do něj poprvé přišli.

[9] Pokud jste ho nečetli nebo si algoritmus nepamatujete, najdete ho ve 2. čísle.

Než začneme graf prohledávat, zvolíme si počáteční vrchol. Ten si označíme jako otevřený, ostatní vrcholy jako nenavštívené. Vrcholu, ve kterém se aktuálně nacházíme budeme říkat *aktuální* a tento vrchol bude vždy právě jeden a bude vždy otevřený. Na začátku je tedy aktuální počáteční vrchol. V každém kroku algoritmu zkontrolujeme, zda má aktuální vrchol nějakého nenavštíveného souseda. Pokud ano, půjdeme k němu na návštěvu – změníme tedy jeho stav na otevřený a nastavíme ho jako aktuální vrchol (vstoupíme do něj). Pokud už aktuální vrchol žádné nenavštívené sousedy nemá, označíme ho jako uzavřený a vrátíme se do vrcholu ze kterého jsme do něj poprvé přišli. Až uzavřeme počáteční vrchol, skončíme.

Zbývá si rozmyslet, jakou má DFS časovou složitost. Je jasné, že algoritmus udělá $\mathcal{O}(n)$ kroků, protože v každém kroku změníme stav vrcholu a každý vrchol může změnit svůj stav maximálně dvakrát (jednou z nenavštíveného na otevřený a podruhé z otevřeného na uzavřený). V každém kroku se musíme podívat na všechny syny daného vrcholu. Mohli bychom si tedy říct, že časová složitost je $\mathcal{O}(n\cdot$ `největší stupeň v grafu`$)$. Umíme ale udělat lepší odhad – stačí si uvědomit, kolikrát se díváme přes každou hranu. Je to maximálně čtyřikrát – pokaždé, když jsme v jednom z vrcholů, které spojuje. Kontrolu všech sousedních vrcholů tedy vždy „naúčtujeme“ hranám, přes které se koukáme. Pro každý vrchol i hranu nám tak vychází konstantní čas a celková časová složitost tedy bude $\mathcal{O}(n+m)$.

Není těžké si rozmyslet, že DFS nemusí nutně najít nejkratší cestu. Má ale oproti BFS jiné přednosti.

Problém 5 (triviální) [0,5b]: *Ukažte příklad, kdy DFS najde cestu, která nebude nejkratší.*

Kromě toho, že pomocí DFS najdeme všechny vrcholy, do kterých se lze dostat z počátečního vrcholu, tak pokud je graf souvislý, dostaneme takzvanou *kostru grafu*. Kostra grafu je strom na stejných vrcholech jako původní graf a nepoužívá hrany, které v původním grafu nejsou. Nyní si ukážeme, že když vyznačíme všechny hrany, po kterých jsme v průběhu DFS prošli, dostaneme kostru grafu. Žádné hrany, které v grafu nejsou, evidentně nepoužijeme. Abychom dokázali, že se jedná o kostru grafu, zbývá si rozmyslet následující:

Problém 6 [2b]: *Ukažte, že když je graf souvislý, navštíví DFS všechny vrcholy. Zde se vám pravděpodobně bude hodit takzvaný důkaz sporem. Předpokládejte, že (hypoteticky) existuje vrchol, který jsme v průběhu DFS nenavštívili a ukažte, že se někdy musel algoritmus zachovat jinak, než jak jsme popsali. Protože algoritmus se nechová jinak, než jsme popsali, tak z toho plyne, že takový vrchol nemůže existovat. Toto je užitečná technika matematických důkazů a, pokud ji neznáte, doporučujeme si rozmyslet, že opravdu funguje. Více si můžete o důkazu sporem přečíst na* `https://matematika.cz/dukaz-sporem`.

Problém 7 [2b]: *Mějme graf, říkejme mu G, a proveďme na něm DFS. Vezměme všechny hrany G, po kterých projdeme v průběhu DFS a ostatní vymažme. Vysvětlete, proč bude výsledný graf strom.*

Kromě hledání kostry můžeme použít DFS na „klasifikaci hran“. Hrany rozdělíme do následujících dvou kategorií[10]:

Stromové hrany jsou ty, po kterých procházíme v průběhu DFS, když navštěvujeme vrcholy. Jde tedy o hrany z problému 7 a, jak jsme si rozmysleli, opravdu tvoří strom.

Zpětné hrany jsou všechny hrany, které nejsou stromové. Vedou vždy z aktuálního vrcholu do vrcholu, který je otevřený. Kdyby totiž byl uzavřený, tak bychom z něj před jeho uzavřením vstoupili do aktuálního vrcholu, takže by hrana byla stromová a kdyby byl nenavštívený, tak bychom ho nyní navštívili a hrana by byla rovněž stromová.

Z kategorií hran lze zjistit informace o struktuře grafu. Jednu takovou aplikaci si vyzkoušíme v problému 9, ale před tím si ukážeme jednu obecnou techniku pro návrh algoritmů na stromech.

Algoritmy na stromech

Pokud máme problém na stromu, který chceme vyřešit, funguje často následující postup.

1. Pokud není zakořeněný, zakořeníme si strom v libovolném vrcholu (jinými slovy, o libovolném vrcholu prohlásíme, že je kořen).
2. Rozmyslíme si, jaké je řešení problému pro *listy* – to jsou všechny vrcholy stupně 1 kromě kořene.
3. Rekurzivně spočítáme řešení podproblémů ve všech synech kořene a vymyslíme postup, jak řešení z podstromů zkombinovat do řešení kořene.

Mohlo by se zdát, že algoritmus umí spočítat řešení jen pro kořen a to pouze pokud nějak, neznámo jak, získáme řešení pro jeho syny. Není tomu ale tak. Zde využijeme rekurzi – podstrom pod libovolným ze synů je také strom, a tedy na něm můžeme úlohu rekurzivně vyřešit stejným algoritmem.

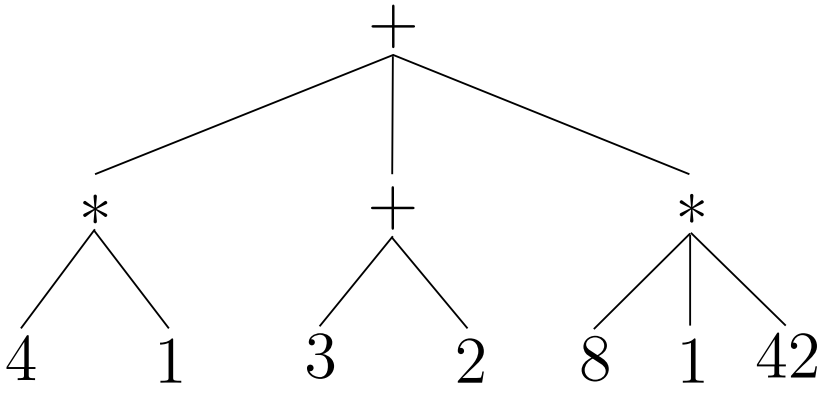
Podproblémů v rekurzi je stejně jako vrcholů, tedy n. Pokud kombinace řešení z k podstromů trvá $t(k)$ (t je tedy funkce udávající časovou složitost kombinace), pak celý algoritmus poběží v $\mathcal{O}(nt(n))$. Často lze ukázat lepší časová složitost pomocí následující analýzy. Mějme vrchol s k syny a kombinace řešení trvá $t(k)$. „Naúčtujeme“ cenu kombinace potomkům, tedy každému $t(k)/k$. Celý algoritmus běží v čase nejvýše n krát „kolik jsme nejvýše naúčtovali jednomu vrcholu“.

Tento způsob řešení problémů na stromech si můžete vyzkoušet na následujícím problému.

Problém 8 [1b za algoritmus, 2b za časovou složitost]: *Máme na vstupu zakořeněný strom reprezentující matematický výraz následujícím způsobem. Kořen odpovídá celému výrazu. V každém vnitřním vrcholu (tedy „nelistu“ – kořen je také*

[10] Pokud víte, co to je orientovaný graf, tak vězte, že pro orientované grafy jsou kategorie čtyři.

vnitřním vrcholem) je operace +, nebo ∗. V listech jsou čísla. Strom pak můžeme převést na výraz tak, že rekurzivně převedeme na výraz všechny syny aktuálního vrcholu, dáme je do závorky a na těchto podvýrazech provedeme operaci, která je v aktuálním vrcholu. Proto budeme navíc předpokládat, že každý vnitřní vrchol má alespoň 2 syny (operace + a ∗ totiž vyžadují alespoň dva operandy. Příklad takového stromu můžete vidět na obrázku 10. Popište algoritmus, který vyhodnotí zadaný strom výrazu. Nezapomeňte na časovou složitost.



Obrázek 10: Strom výrazu $(4 * 1) + (3 + 2) + (8 * 1 * 42)$

V následujícím problému budete potřebovat vědět, co je to most. *Most* je hrana v grafu, kterou když odstraníme, tak mezi jejími koncovými vrcholy nebude existovat cesta. Alternativně jde charakterizovat most jako hranu, jejímž odstraněním se zvýší počet *komponent souvislosti*, kde komponenta souvislosti je prostě „jeden souvislý kus grafu“. Pokud tedy například vezmeme mapu Prahy s křižovatkami jako vrcholy a ulicemi jako hranami, tak Karlův most není most, protože hned vedle něj je most Legií a jde to tedy obejít přes něj, Kampu a Smetanovo nábřeží.

Nyní znáte vše, co je potřeba pro vyřešení následujícího problému.

Problém 9 (těžký) [4b]: *Vymyslete lineární algoritmus, který najde všechny mosty v souvislém grafu. Poradíme, že se vám bude hodit prezentovaný způsob návrhu algoritmů pro stromy (s mírnou úpravou pro nalezení všech mostů) a klasifikace hran z DFS. Tři body dostanete za lineární řešení, které zjistí, zda nějaký most v grafu existuje. Část bodů vám přidělíme i za pomalejší algoritmus.*

Řešení 1. série

Pokud jste se už rozhodli něco odeslat, tak to byly většinou obě úlohy, víceméně správně vyřešené. Nicméně několik nešvarů se objevovalo téměř u všech řešení a bylo by dobré, kdybyste se jich do budoucna vyvarovali.

Největším nedostatkem byla absence jakéhokoliv komentáře – to, že odešlete nějaký zdrojový kód, je určitě dobře, ale neméně důležitá je také jeho srozumitelnost. A pokud nemám sebemenší ideu, jak by dané řešení mělo fungovat, tak to bývá složité rozeznat (a u chybných či složitějších zdrojových kódů i téměř nemožné). Tedy rozhodně komentujte, co by měl váš program dělat a jaká je za tím myšlenka.

Další věcí, která vám chyběla, bylo ukázat, že vaše řešení vydá správný výsledek. Protože to, že vymyslím nějaký algoritmus, nutně neznamená, že vrátí správný výsledek, nebo že vůbec někdy nějaký výsledek vydá. Je proto potřeba udělat nějaký rozbor, ve kterém tyto vlastnosti ukážete.

A ještě další důležitou věcí, která části z vás chyběla, je určit časovou složitost algoritmu. Protože to, že algoritmus skončí a vydá správný výsledek, mi nepomůže, pokud mi ho vydá až za pár miliard let :-).

Dobrý tip k psaní kódu je potom pojmenovávat rozumně proměnné (pokud proměnná vyjadřuje maximum, tak ji nepojmenuji a_2, ale *maximum*), a pokud se název skládá z více slov, tak čitelnost zvýší, když je nějak vizuálně oddělím (například *docasneMaximum* nebo *docasne_maximum*).

Tohle by bylo asi k obecným radám všechno a nyní k jednotlivým úlohám.

Úloha 1

Zadání:

Vymyslete algoritmus, který zjistí, kde se v poli nachází největší číslo a jaké číslo to je. Můžete předpokládat, že největší číslo je v poli jen jedno.

Řešení:

U první úlohy spousta z vás jako *dočasné maximum* zvolila číslo 0, se kterým potom porovnávala ostatní čísla v poli, a pokud bylo nějaké číslo větší, tak ho vzala jako nové dočasné maximum. Až potom prošla celé pole, tak vrátila *dočasné maximum* jako výsledek. Což vám bude pěkně fungovat pro čísla přirozená, zato selže pro čísla celá, s nimiž pracuje náš model. (Stačí uvážit pole pouze se zápornými čísly, kde by potom algoritmus vrátil 0.) Řešením by mohlo být dosadit tam nějaké nejmenší možné číslo, protože ale náš model pracuje se všemi celými čísly a nic jiného do proměnné ukládat neumí, tak by se to neobešlo bez komplikací (například říct, že číslo 0 je „mínus nekonečno“ a -1 je 0, -2 je -1 atd, a s tím potom pracovat – pamatovat na to při čtení vstupu, vypisování výstupu apod.). Mnohem elegantnější a jednodušší je zvolit si jako *dočasné maximum* první číslo v poli. Potom projdeme zbylé prvky v poli a postupně je porovnáme. Když budou větší, tak je vezmeme jako nové maximum a pokračujeme. Ještě je nutno si zapamatovat, na jaké pozici se číslo nachází, ale to se bude měnit stejně jako *dočasné maximum*.

Algoritmus tedy projde celé pole jednou a pro každý prvek provede porovnání, případně zapsání do proměnných, čili jeho časová složitost bude $\mathcal{O}(n)$.

Kód by potom mohl vypadat například takto (je možná až příliš pečlivě okomentovaný, rozhodně nepožadujeme zdůvodnění každého kroku, ale nějaké zdů-

vodnění se občas hodí):

Vstup: posloupnost čísel v poli a jejich počet v proměnné `n`

1. `delka_pole` ← `n` (uložíme si délku pole)
2. `docasne_maximum` ← `pole[0]` (za dočasné maximum zvolíme první prvek)
   (předpokládáme, že je v poli alespoň jeden prvek)
3. `pozice_maxima` ← 0
   (zatím je pozice největšího prvku 0, jak jsme provedli v předchozím kroku)
4. `pozice` ← 1 (pomocí proměnné pozice budeme procházet naše pole)
5. Dokud `pozice` < `delka_pole`:
   (dokud nedojdeme na konec pole, tak budeme porovnávat prvky)
6. Pokud `pole[pozice]` > `docasne_maximum`:
   (pokud jsme našli větší číslo, tak změníme naše hodnoty)
7. `docasne_maximum` ← `pole[pozice]`
8. `pozice_maxima` ← `pozice`
9. `pozice` ← `pozice` + 1 (přesuneme se na další číslo)

Výstup: proměnná `docasne_maximum`, kde je uloženo největší číslo pole, a proměnná `pozice_maxima`, kde je uložena jeho pozice.

Úloha 2

Zadání:
Vymyslete algoritmus, který setřídí pole čísel. Jinými slovy, chtěli bychom čísla seřadit podle jejich velikosti.

Řešení:
U druhé úlohy se nečitelnost kódu už projevila výrazně. Často jste totiž měli chybu ve zdrojovém kódu, takže nevracel správný výsledek. Pak bylo velmi těžké uhodnout, co by měl dělat, a pokusit se ho opravit.

Ukážeme si rovnou řešení i bonusové části. Předběhnu a prozradím, že bude mít složitost $\mathcal{O}(n^2)$, která není nejlepší možná – vymýšlení rychlejšího algoritmu jsme se věnovali v druhém čísle.

Algoritmus celkem přímočaře vychází z algoritmu na hledání maxima – nejdříve projdeme celé pole, najdeme maximum a to prohodíme s posledním prvkem. Poté projdeme pole bez posledního prvku, najdeme znovu maximum (tedy globálně druhý největší prvek) a to prohodíme s předposledním prvkem. Tak pokračujeme dále, vždy procházíme pole o délce n-k+1 a hledáme k-tý největší prvek. Od konce pole nám tak roste setřízená posloupnost. Až se dostaneme k poli délky jedna, skončíme.

V kontextu druhého dílu tématka bychom se na náš algoritmus také mohli dívat jako na rekurzivní – najdeme maximum, přehodíme ho s posledním prvkem a rekurzivně se zavoláme na pole o 1 kratší.

Pseudokód třídícího algoritmu by mohl vypadat třeba takto (část kódu shodná s úlohou 1 šedě):

Vstup: posloupnost čísel v poli a jejich počet v proměnné `n`

```
1.  delka_pole ← n                                   (uložíme si délku pole)
2.  Dokud delka_pole > 1:
                  (opakovaně hledáme maximum, dokud nám nezbude jednoprvkové pole)
3.      docasne_maximum ← pole[0]
4.      pozice_maxima ← 0
5.      pozice ← 1
6.      Dokud pozice < delka_pole:
7.          Pokud pole[pozice] > docasne_maximum:
8.              docasne_maximum ← pole[pozice]
9.              pozice_maxima ← pozice
10.         pozice ← pozice + 1
11.     pole[pozice_maxima] ← pole[delka_pole - 1]
                                  (poslední prvek uložíme na místo maxima)
12.     pole[delka_pole - 1] ← docasne_maximum
                                  (maximum uložíme na konec prohledávaného úseku)
13.     delka_pole ← delka_pole - 1          (Zmenšíme prohledávaný úsek o 1)
```

Výstup: Prvky zůstaly v zadaném poli, jen jsou teď setřízené.

Nyní nám zbývá zamyslet se nad časovou složitostí. Hledání maxima je lineární s délkou pole a hledali jsme n-krát v poli o velikosti maximálně n. Z toho je jasné, že algoritmus poběží v čase $\mathcal{O}(n^2)$. Při bližším pohledu zjistíme, že těsnější odhad najít nelze – když totiž sečteme délky polí, ve kterých jsme hledali, dostaneme výraz $1 + 2 + \cdots + (n-1) + n$, který se sečte na $\dfrac{n^2 - n}{2} \in \mathcal{O}(n^2)$.

Petr, Kuba a Tom; `domestomas+mam@gmail.com`
e-mailová konference: `algoritmy@mam.mff.cuni.cz`

Téma 5 – Přeplněná tramvaj

V předchozím čísle jste dostali zadáno zabývat se vývojem počtu cestujících uvnitř prostředku hromadné dopravy (značíme S) v závislosti na fázi cesty (konkrétněji počtu projetých stanic[11] m) s tím, že tuto závislost $S(m)$ je potřeba zkoumat jednak experimentálně, a jednak najít nějaký model, který by chování cestujících mohl popsat teoreticky[12].

K experimentální části není co dodat, jen znovu připomínám, že výsledky vlastních experimentů můžete posílat kdykoli až do uzavření tématka a že se jedná o velmi jednoduchý způsob, jak nasbírat spoustu bodů do soutěže prakticky pouze tím, že se vozíte MHD.

Ohledně hledání modelu k popisu cestujících věřím, že jste přišli s mnoha originálními nápady, avšak protože mi v době vzniku tohoto textu ještě nebyly k dispozici, budou zmíněny nejdříve v následujícím čísle. Zde předkládám vlastní řešení tohoto problému.

K popisu nejvhodnější model, který se mi podařilo nalézt, je následující: Mějme linku s N stanicemi, každá stanice v okamžiku příjezdu prostředku vygeneruje počet cestujících, který představuje náhodnou veličinu se střední hodnotou D. Každý z cestujících si v tomto okamžiku zvolí cílovou stanici své cesty, přičemž ji opět vybírá náhodně s uniformním rozdělením (tj. žádná stanice linky není upřednostněná). Leží-li zvolená stanice na lince v tom směru, kterým prostředek jede, cestující nastoupí, v opačném případě zůstává ve stanici.

K tomuto modelu hned zadávám první úkol:

Problém 1: *Spočítejte vývoj střední hodnoty počtu lidí v prostředku za předpokladu, že se lidé chovají, jak je zde popsáno, a zakreslete křivku závislosti $S(m)$. A to:*

a) Na lineární lince, tj. takové, která má začátek a konec a žádná stanice na ní není zastoupena více než jednou. [3b]

b) Na cyklické lince, tj. takové, kde se po projetí n různých stanic prostředek vrátí do první stanice a svou jízdu opakuje. Předpokládejme, že po lince jezdí vozy v obou směrech. [3b]

c) Zamyslete se, jaký vliv na platnost vašich řešení úloh a) a b) může mít počet vozů zajišťujících dopravu po lince.

Můj druhý model předpokládá, že cestující jezdí bez cíle. To znamená, že znovu každá stanice při příjezdu prostředku vygeneruje D cestujících, ti si ale nevolí cíl cesty, nýbrž volí si s pravděpodobností p, že bezdůvodně nastoupí, nebo $1-p$, že zůstanou ve stanici. Cestující uvnitř prostředku si pak obdobně s pravděpodobností q mohou zvolit, že v této stanici vystoupí, nebo $1-q$, že pojedou dál.

[11] POZOR, bylo pozměněno značení oproti předchozímu číslu. Proměnná počtu projetých stanic je od nynějška m, zatímco N odteď bude značit počet stanic na lince. Omlouváme se za zmatení, budeme se snažit, aby k dalším podobným změnám už nedocházelo.

[12] Minulé číslo najdete na adrese `https://mam.mff.cuni.cz/media/cislo/pdf/25/25-2.pdf`

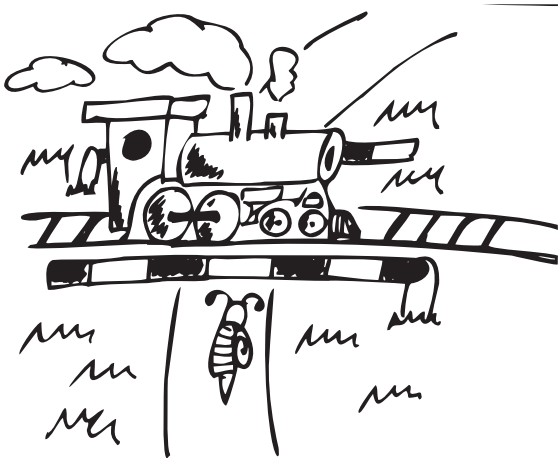
Problém 2 [3b]: *Spočítejte vývoj střední hodnoty počtu lidí v prostředku za předpokladu, že lidé cestují bez cíle, a zakreslete křivku závislosti $S(m)$. Předpokládejme velmi dlouhou lineární linku.*

Tento model je už sice o dost vzdálenější chování typického cestujícího v dopravním prostředku, přesto se nám může hodit ho uvážit. Podobně bezcílně totiž jezdí například revizoři, bloudící turisté, nebo v jistém smyslu třeba i vy, věnujete-li se právě experimentální části této úlohy. Budeme-li znát křivku $S(m)$ pro popis jak klasických, tak bezcílných cestujících, můžeme být schopni z výsledků vašich experimentů později odhadnout, jak velké procento cestujících hromadné dopravy tato bezcílná menšina tvoří.

Problém 3 (bonusový): *Zamyslete se nad tím, jaký jiný model by se dal k popisu chování cestujících použít vyjma dvou zde uvedených, a pro tento model vypočítejte křivku $S(m)$. Jedná se o prodloužení doby platnosti Úlohy 2 z minulého čísla.*

Nápověda 1: Pro zjednodušení můžete předpokládat, že je cestujících ve stanici vygenerováno vždy právě D.

Nápověda 2: Všimněte si, že v prvním modelu jsou všechny stanice rovnocenné, co se týče vytíženosti z hlediska nástupů i výstupů



Příloha

V této příloze se vám pokusím v návaznosti na předchozí číslo nastínit několik faktů a úvah z teorie pravděpodobnosti, které mohou být klíčové pro řešení výše uvedených úloh.

Především se zmíním o podmíněné pravděpodobnosti. Ta nám říká, jak spolu mohou být provázané jednotlivé jevy definované na jisté náhodné veličině X s prostorem možných výsledků $\Omega(X)$. Vezměme například obdobně k předchozímu číslu

za náhodný generátor kostku a definujme znovu jev $A = \{x_i; x_i \text{ je liché}\}$ a jev $B = \{x_i; x_i > 3\}$. Každý z těchto jevů nastane s pravděpodobností $\frac{1}{2}$. Přesto víme-li nějak, že nastane jev B, tj. padne některé z čísel 4, 5, 6, jev A už může nastat jedině tehdy, když padne 5, čemuž nyní odpovídá pravděpodobnost $\frac{1}{3}$. Zapisujeme to $P(A|B) = \frac{1}{3}$, kde výraz na levé straně čteme **pravděpodobnost *A* za předpokladu *B***.

Obecně vztah mezi pravděpodobnostmi dvou jevů udává **Bayesův vzorec**:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = P(B|A)\frac{P(A)}{P(B)}$$

Jako **nezávislé jevy** A, B označujeme takové, pro které platí $P(A|B) = P(A)$ a $P(B|A) = P(B)$. Dle Bayesova vzorce pravděpodobnost, že nastanou oba, rovna $P(A \cap B) = P(A)P(B)$. Podobně tak **nezávislými veličinami** nazýváme náhodné veličiny X, Y, jsou-li jevy $X \subseteq A$ a $Y \subseteq B$ nezávislé pro každé $A \subseteq \Omega(X), B \subseteq \Omega(Y)$. Jako takové můžeme vzít například dva po sobě jdoucí hody kostkou: pro každý hod je pravděpodobnost pádu šestky rovna $\frac{1}{6}$, takže pravděpodobnost pádu dvou šestek po sobě bude $P((\{6;6\})) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36}$.

Shrneme-li to tedy s našimi závěry z předchozího čísla, můžeme využívat dva užitečné principy:

1. U dvou nezávislých jevů je pravděpodobnost, že nastanou oba, rovna součinu jejich pravděpodobností.

2. U dvou neslučitelných jevů je pravděpodobnost, že nastane aspoň jeden, rovna součtu jejich pravděpodobností.

Dále se vám při řešení může hodit vědět, jak se chová střední hodnota složené náhodné veličiny. Nechť jsou X, Y náhodné veličiny, c je libovolná konstanta a $\mu(X)$ je střední hodnota veličiny X. Potom platí[13]:

$$\mu(c) = c$$
$$\mu(cX) = c\mu(X)$$
$$\mu(X + Y) = \mu(X) + \mu(Y)$$

Jsou-li veličiny X,Y navíc nezávislé, platí také:

$$\mu(XY) = \mu(X)\mu(Y)$$

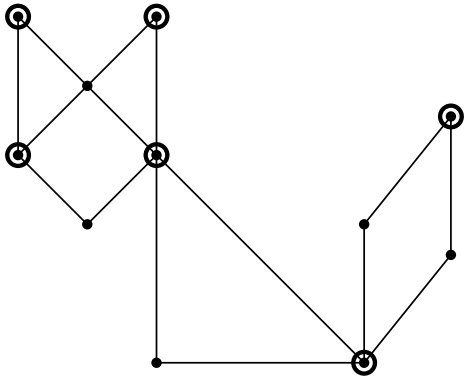
Vztahy a úvahy zde uvedené (společně s učivem SŠ) by vám měly k rozlousknutí výše zadaných problémů postačit. Přeji hodně štěstí při řešení.

Evžen; JanSkvara@email.cz
e-mailová konference: tramvaj@mam.mff.cuni.cz

[13] Veličinu cX lze chápat, jako bychom každou hodnotu z $\Omega(X)$ vynásobili konstantou c a tento výsledek prohlásili za novou náhodnou veličinu.

Téma 6 – Vrcholové pokrytí

V tomto tématku se budeme věnovat hledání minimálního vrcholového pokrytí. Máme-li graf[14] definovaný jako množinu vrcholů a množinu hran mezi nimi, tak jeho vrcholové pokrytí je taková podmnožina vrcholů, že každá hrana má alespoň jeden svůj konec v té podmnožině.



Obrázek 11: Vrcholové pokrytí grafu

Problém 1 [1b]: *Dokážete najít menší vrcholové pokrytí pro výše uvedený graf?*

Minimální vrcholové pokrytí je potom takové vrcholové pokrytí, které obsahuje minimální možný počet vrcholů. Pro hledání minimálního vrcholového pokrytí existuje spousta zajímavých triků, viz `https://en.wikipedia.org/wiki/Vertex_cover`. Na našem webu (`https://mam.mff.cuni.cz/problem/2208/`) najdete různé grafy, na kterých můžete zkusit minimální vrcholové pokrytí spočítat. Pokud pro některý z grafů naleznete minimální vrcholové pokrytí, tak nám jej pošlete. Také posílejte vaše pozorování o vlastnostech grafů, nápady na zlepšení algoritmů a obecně cokoliv, co může vám a vašim spoluřešitelům pomoct v hledání minimálních vrcholových pokrytí velkých grafů.

Matej; `lieskovsky.matej+pokryti@gmail.com`
e-mailová konference: `pokryti@mam.mff.cuni.cz`

[14] Podrobnější vysvětlení toho, co je to graf, naleznete v tématku 4 v prvním odstavci pod nadpisem „Teorie grafů“

Výsledková listina 1. čísla

Poř.	Jméno	R.	$\sum_{-1}$	Témata t1	t2	t3	t4	$\sum_0$	$\sum_1$
1.	Bc.[MM] V. Materna	3	17,0	12,0			5,0	17,0	17,0
2.	Bc.[MM] T. Flídr	1	13,0		1,0		12,0	13,0	13,0
3.	Bc.[MM] K. Hloušková	3	14,5	6,0		4,0		10,0	10,0
4.	Dr.[MM] L. Kundratová	4	69,7	9,0				9,0	9,0
5.	F. Bujnovský	1	8,8		7,0		1,8	8,8	8,8
6.–7.	Mgr.[MM] J. Pallová	4	36,8		3,0		5,0	8,0	8,0
	Mgr.[MM] M. Souza de Joode	2	27,8			8,0		8,0	8,0
8.	T. Sourada	4	7,8		1,0		6,8	7,8	7,8
9.	V. Jůzková	2	5,5				5,5	5,5	5,5
10.	J. Štěpo	Z9	5,0		5,0			5,0	5,0
11.–12.	O. Chlubna	2	4,8				4,8	4,8	4,8
	Mgr.[MM] M. Kalousková	3	30,3				4,8	4,8	4,8
13.	Mgr.[MM] O. Gonzor	2	26,2				4,5	4,5	4,5
14.	J. Kvapil	1	3,7		0,7		3,0	3,7	3,7
15.	L. Kunčarová	3	3,6		2,1		1,5	3,6	3,6
16.	J. Kováč	4	3,5				3,5	3,5	3,5
17.–18.	Dr.[MM] K. Balej	4	83,4		3,0			3,0	3,0
	Doc.[MM] K. Rosická	4	119,8		3,0			3,0	3,0
19.	R. Zavřel	3	2,3	2,0	0,3			2,3	2,3
20.	Mgr.[MM] E. Vítková	3	27,0				1,5	1,5	1,5

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.

Kontakty:

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

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