STUDENTSKÝ ČASOPIS A KORESPONDENČNÍ SEMINÁŘ

Ročník XXV
Číslo 2

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,

ve druhém čísle letošního ročníku najdete pokračování tématek z minulého čísla. K zamyšlení vám předkládáme úlohy o magnetickém poli a rotační ose měsíce, nebo si můžete vyzkoušet, zdali ostatní kapaliny mrznou podobně jako voda. V dalším tématku zabývajícím se algoritmy se podíváme na to, kde se používá rekurze, a v neposlední řadě si můžete přečíst o problémech a úlohách týkajících se hashovacích funkcí. K tématkům přibylo jedno nové matematické, ve kterém objevíte zákonitosti o cestování (nejen) tramvajemi a autobusy a budete zkoumat, jak se mění počet cestujících na trase daného dopravního prostředku.

Některé problémy z minulého čísla tohoto ročníku jsou i nadále otevřené a můžete je stále řešit. Tyto otevřené problémy poznáte podle toho, že autoři problému týkajícího se daného tématka napsali, že vzorové řešení nebude zveřejněno do konce školního roku.

V týdnu 13.–21. 10. proběho podzimní soustředění. Doufáme, že jste si užili jak přednášky a konfery, tak i hry. Pokud jste nebyli mezi účastníky, nezoufejte, před Vánoci (14.–16. 12.) se bude konat víkendovka a na jaře ve dnech 30. 3. – 7. 4. jarní soustředění.

Přejeme vám mnoho úspěchů a zábavy při řešení

Vaši organizátoři

Zadání témat

Termín odeslání 2. série: 4. 12. 2018

Téma 1 – Paradoxní výsledky

Malý Bartoloměj provedl svůj pokus a zjistil, že teplejší voda opravdu zmrzne rychleji, ale jen pokud pokus probíhá v menším měřítku (tedy pokud Bartoloměj neumístí do jednoho mrazáku mnoho nádob najednou). Rád by však přišel na to, co za to může. Nejprve se rozhodl prozkoumat vlastnosti kapaliny. Připravil si

tedy teploměr a vyzkoušel tato měření s roztokem kuchyňské soli. Svoje měření ukončil, když teplota dané kapaliny dosáhla hodnot lehce pod 0 °C. Napadlo ho, že kapalina má jinou teplotu v různých místech svého objemu. Co kdyby do ní něco vložil a zabránil tak jejímu míchání anebo materiálem vloženého tělesa (například lžičky) ovlivnil odvod tepla z kapaliny?

Problém 1: *Ověřte, zda i pro ostatní kapaliny platí, že na počátku teplejší kapalina mrzne rychleji. Vyzkoušejte např. mléko, vodu se solí atd. Na čem si myslíte, že bude výsledek měření záviset? Zkuste navrhnout ideální kapalinu, na které bude efekt nejsilnější. Zkuste změnit podmínky, za kterých voda a vámi zvolené kapaliny tuhnou, tím, že zamezíte pohybu molekul v objemu kapaliny či změníte odvod tepla tím, že do kapaliny něco vložíte. Diskutujte důsledky těchto vámi změněných podmínek.*

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

Téma 2 – Principy kryptografie

Hashovací funkce

V prvním čísle jsme se zabývali blokovými šiframi. V tomto se podíváme na další podstatný koncept, bez kterého by se kryptografie neobešla, a tím jsou hashovací neboli jednosměrné funkce.

Již nyní se můžete těšit na třetí číslo, ve kterém blokové šifry a hashovací funkce využijeme dohromady. Najdete v něm také řešení úloh z prvního i druhého čísla. Pokud jste nám zatím nezaslali řešení úloh z prvního čísla, můžete je tedy posílat i nadále.

Koncept

V kryptografii nazýváme hashovací funkcí H takovou funkci, která pro libovolný vstup vrátí výstup pevně stanovené délky a splňuje následující podmínky:

1. Pro daný hash c je obtížné najít x takové, že $H(x) = c$. Tedy pro zadaný hash je obtížné najít vzor, který je pomocí hashovací funkce na tento hash převeden.

2. Pro daný vstup x je obtížné najít y tak, aby $H(x) = H(y)$. Neboli je obtížné najít druhou vstupní hodnotu se stejným hashem.

3. Je obtížné najít vstupy x a y, aby $H(x) = H(y)$. Neboli je obtížné najít libovolné dvě vstupní hodnoty se stejným hashem.

Označením, že je něco obtížné, v tomto případě chceme vyjádřit, že toho nelze dosáhnout výrazně snáze než pomocí zkoušení variant hrubou silou. Pokud budeme mít hashovací funkci s výstupem o velikosti 64 bitů a zkusíme zahashovat

$2^{64} + 1$ hodnot, určitě najdeme dvě se stejným hashem. Dokonce s velkou pravděpodobností najdeme dvě hodnoty se stejným hashem mnohem dříve. Neměla by ale existovat žádná metoda, která je statisticky úspěšnější než zkoušet hrubou silou hashovat náhodné hodnoty.

Úloha 1 [1b]: *V kryptografii je obecně za úspěšný považován libovolný útok, který má alespoň 50% šanci na úspěch. Představme si ideální hashovací funkci, která má výstup o velikosti 64 bitů. Kolik výpočtů hashovací funkce budeme muset provést, abychom s pravděpodobností alespoň 50 % našli dvě hodnoty se stejným hashem?*

Zároveň ještě od hashovací funkce z praktických důvodů typicky chceme, aby drobná změna vstupního textu vedla k velké změně výsledného hashe a abychom z hashe nedokázali získat ani část informace o vstupu.

Využití

Hashovací funkce mají celou řadu uplatnění. Jedním z nejpodstatnějších je ověření, že nějaká data nebyla změněna. Stačí si z dat spočítat hash a ten nějakým důvěryhodným způsobem uložit. Mohu si ho třeba i elektronicky podepsat (i k tomu se v rámci tohoto tématu ještě dostaneme). Pokud chceme později ověřit, že v datech nedošlo k žádné úpravě, stačí z nich znovu spočítat hash a hashe porovnat.

Zcela jiné využití hashovacích funkcí najdeme například při ukládání hesel. Představme si webovou stránku, kam se uživatelé přihlašují pomocí jména a hesla. Aby hesla nemusela být na serveru uložena jako prostý text, který si může kdokoli přečíst, bývá zvykem na serveru ukládat pouze hashe hesel. Ve chvíli, kdy se uživatel chce přihlásit, pošle na server heslo. Na serveru je z tohoto hesla spočítán hash a porovnán s uloženým záznamem. Pokud jsou stejné, uživatel zadal nejspíš správné heslo a je přihlášen.

Problém 2 [2b+]: *Hashování hesel je určitě vhodným postupem. Pokud ale jednoduše spočítáme hash z hesla, stále můžeme při odcizení hashů čelit určitým rizikům. Napadá vás, kde jsou nedostatky takového postupu? Dokážete najít a popsat, jak ukládat hesla lépe? Napovíme, že možností na vylepšení existuje celá řada.*

Praktická realizace

Už víme, jaké vlastnosti by hashovací funkce měla mít. Zbývá vyřešit otázku, jak takovou funkci sestrojit.

V současné době můžeme za nejrozšířenější hashovací funkce považovat MD5, SHA1 nebo rodinu funkcí SHA2 (SHA256, SHA512). Ty všechny využívají pouze klasické binární operace and, or, xor, modulární sčítání a rotace, případně shifty. Více informací o těchto operacích najdete v prvním čísle. Zmíněno tam není akorát modulární sčítání. Na této operaci ale není nic složitého – jedná se o klasické sčítání s čísly pevné maximální délky, kde u nejvyššího řádu nezohledňujeme přenos výš. Vedle uživatelského vstupu (libovolné délky) funkce vždy využívají i řadu pevných konstant.

Funkce RIKSHA

Abychom si mohli s konstrukcí hashovacích funkcí trochu pohrát, tak pro vás máme naši vlastní funkci jménem RIKSHA [čti *rikša*]. Princip, jak funkce pracuje, je podobný jako u běžných hashovacích funkcí.

Nabízíme vám ji ve formě funkčního zdrojového kódu pro Python, o kterém doufáme, že je dostatečně názorný. Pokud by vám nebylo jasné, co některá část kódu dělá, nebojte se poslat dotaz do tématkové e-mailové konference.

```
#!/usr/bin/env python3
import sys

# konstanty
c0 = 0x56
c1 = 0x37
c2 = 0xa5
c3 = 0x9d
c4 = 0xbf

# inicializace casti vysledneho hashe
h0, h1, h2, h3, h4, h5, h6, h7 = c0, c1, c0, c1, c0, c1, c0, c1

# rotace vlevo o b bitu 8-bitoveho cisla n
def rotl(n, b):
    return ((n << b) | (n >> (8 - b))) & 0xff

# pridej "binarni 1", dopln 0x00 na nasobek 8 bytu
def pad(text):
    text += '80'
    text += '0' * (16 - len(text) % 16)
    return text

# uprav hash podle casti vstupu
def update(chunk):
    global h0, h1, h2, h3, h4, h5, h6, h7
    # rozdel vstup na (decimalni) cisla
    d = []
    for i in range(8):
        d.append(int(chunk[2*i:2*i+2], 16))
    # spocitej mezihodnoty
    a0 = ((d[2] << 3) ^ (d[4] << 3) ^ (d[6] << 3) ^ (c2 << 5)
         ) & 0xff
    a1 = ((d[1] << 4) ^ (d[3] << 4) ^ (d[5] << 4) ^ c3) & 0xff
    a2 = rotl(d[0], 3) ^ rotl(d[7], 5) ^ c2
```

```
    a3 = ((d[0] << 7) & (d[3] >> 2) ^ (c3 << 2)) & 0xff
    a4 = (rotl(d[0], 3) ^ rotl(d[7], 5) ^ (d[4] << 7) ^ c4
         ) & 0xff
    # uprav hodnotu hashe
    h0_puvodni = h0
    h0 = (h0 + h1 + (a0 << 5)) & 0xff
    h1 = (h1 + h2 + (a1 << 3)) & 0xff
    h2 = (h2 + h3 + (a2 << 2)) & 0xff
    h3 = (h3 + h4 + a3) & 0xff
    h4 = (h4 + h5 + a2 + a4) & 0xff
    h5 = (h5 + h6 + a3) & 0xff
    h6 = (h6 + h7 + a2 + a4) & 0xff
    h7 = (h7 + h0_puvodni + (a1 << 4)) & 0xff

# spocitej RIKSHA (vstup v hex ASCII bez mezer)
text = pad(sys.argv[1])
while len(text) > 0:
    update(text[0:16])
    text = text[16:]
print('%.2X'*8 % (h0, h1, h2, h3, h4, h5, h6, h7))
```

Zdrojový kód v elektronické podobě si můžete stáhnout z našeho webu. Na vás je prozkoumat, jestli je RIKSHA dobrou hashovací funkcí.

Úloha 3 [1b]: *Všimněte si funkce* `pad`*, která připojuje za zprávu před doplnění nulami vždy stejnou konstantu. Tento postup jsme si vypůjčili od běžně používaných hashovacích funkcí. Napadá vás, k čemu je tato konstrukce dobrá?*

Úloha 4 [2b]: *Dokážete najít dvě vstupní hodnoty, pro které je výsledný hash stejný?*

Úloha 5 [3b]: *Víte, že pomocí funkce RIKSHA byl zahashován text, který není delší než 8 bytů. Výsledný hash je* `02365E1E061E62BA`*. Dokážete najít nějaký (ne nutně stejný) vstup, který má stejný hash?*

Úloha 6 [5b]: *Existuje nějaká hodnota hashe, kterou funkce RIKSHA nevrátí pro žádný vstup? Nezapomeňte své tvrzení důkladně zdůvodnit.*

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

Téma 3 – Neznámý měsíc

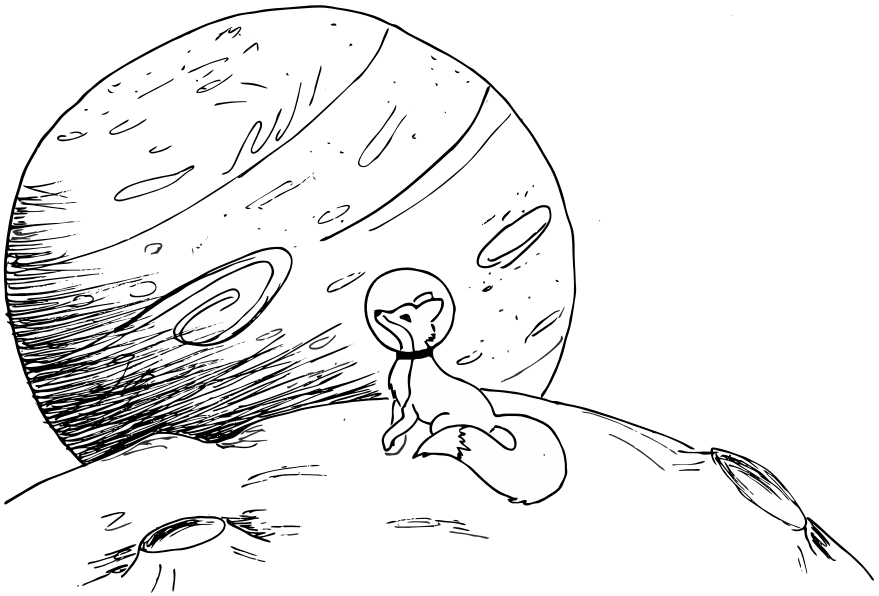
Druhá časť tématka je tu a my chceme, abyste prišli na čo najviac jednoduchých spôsobov, ako niečo zistiť o našom neznámom mesiaci.

Planéty sa v dnešnej dobe hlavne skúmajú družicami s moderným vybavením. Je však zaujímavé, ako naši predkovia v novoveku a stredoveku dokázali zistiť napríklad polomer Zeme, tiažové zrýchlenie a dobu obehu okolo Slnka a to jednoduchými a nenáročnými experimentami. Jednoduchý spôsob, ako zistiť napríklad tiažové zrýchlenie, je pozrieť sa na vzorce, kde sa objavuje tiažové zrýchlenie, a z nich ho vypočítať. Stále môžete posielať riešenia úloh a problémov z prvého čísla.

Problém 1: *Vedeli by ste určiť odklon rotačnej osi mesiaca od ekliptiky mesiaca s bežne dostupnými vecami? Ak nie, čo by ste minimálne potrebovali?*

Problém 2: *Potvrďte prítomnosť magnetického poľa na povrchu mesiaca. Vedeli by ste určiť, či toto magnetické pole generuje mesiac, planéta alebo aj obe zároveň?*

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



Téma 4 – Algoritmy od nuly (do n)

Rekurze

Slovo úvodem

Vítáme vás u dalšího dílu našeho tématka pro začínající informatiky (a také pro zvídavé fyziky a matematiky). Pokud jste zatím nečetli první díl, snažně vám doporučujeme s ním začít – budeme na něj navazovat. Znovu pro jistotu připomínáme, že cílem tématka je vymyslet si postupně základy teoretické informatiky. **Body tedy dáváme za to, že jste vymysleli řešení, a ne za sepsání algoritmu, který jste již dobře znali.** Také věříme, že M&Mko řešíte hlavně proto, abyste se něco nového naučili (a ne jen kvůli bodům). Pokud tedy řešení některých zde zmíněných problémů znáte, pusťte se raději do jiného tématka nebo jiného problému tohoto tématka, které pro vás budou větší výzvou, nebo si vhodný problém k řešení sami vymyslete. Také bychom chtěli doplnit, že u většiny problémů očekáváme především slovní popis algoritmu a ideálně i časovou složitost. Když přidáte i kód, může nám to pomoct lépe pochopit vaši myšlenku a rozhodně vám za něj body neubereme, neměl by ale nahrazovat slovní popis.

Dnes si ukážeme, jak funguje takzvaná rekurze, důležitá technika designu algoritmů. Ač, striktně vzato, není potřeba, protože libovolný algoritmus používající rekurzi můžeme přepsat v modelu, jenž jsme si ukázali minule, často výrazně zjednodušuje vymýšlení i následné programování algoritmu. Předtím si ale ještě řekneme, co jsou to funkce.

Funkce

Při programování často potřebujeme dělat nějakou věc opakovaně na různých místech v programu. Například, kdybychom programovali program na kreslení geometrických útvarů, budeme pravděpodobně často potřebovat počítat délku přepony trojúhelníku pomocí Pythagorovy věty. Proto jsou v programovacích jazycích takzvané funkce, které nám dovolují si „pojmenovat kus kódu“[1], a kdykoliv potřebujeme spočítat délku přepony, zavoláme funkci, která to za nás udělá. Taková funkce a její volání může vypadat třeba takto:

1. Funkce `délka_přepony`(x, y):
2. Vrať $\sqrt{x^2 + y^2}$
3.
4. Z $\leftarrow$ `délka_přepony`(2, 3)

Zamysleme se, jak přesně funkce fungují. Funkce se zavolá s nějakými argumenty (v našem příkladu proměnné x a y), čímž nastavíme pro funkci hodnoty

[1] Použili jsme uvozovky, protože to není úplně správný způsob, jak na funkce pohlížet, jak se vzápětí ukáže.

proměnných, které jsme definovali jako argumenty. Tyto a všechny ostatní proměnné definované ve funkci jsou lokální pro jedno zavolání (spuštění) dané funkce a když se funkce dokončí, zaniknou. Naopak k proměnným mimo funkci, které jsme nepředali jako argument, se nedostaneme. Může se tedy stát, že bude jinde v programu existovat proměnná x, která bude mít jinou hodnotu, než proměnná x lokální pro konkrétní volání funkce `délka_přepony`.

Postupně se budou vykonávat instrukce funkce řádek po řádku, jako kdekoliv jinde v programu. Co se stane, když dojdeme na konec funkce, nebo k příkazu vrať? Chtěli bychom se vrátit přesně na místo, ze kterého jsme funkci zavolali. Přesně to se stane, protože programovací jazyk si za nás zapamatoval místo, ze kterého jsme funkci zavolali, a díky tomu na něj dokáže skočit, když funkce skončí. Pokud byla funkce ukončena příkazem vrať, vrátí se daná hodnota, jak můžete vidět v příkladu, kde nám funkce vrátí hodnotu $\sqrt{13}$.

Problém 1 [4b]: *Zkuste na našem formálním modelu, který jsme si vytvořili na konci minulého dílu, vytvořit funkce. Nezapomeňte, že po svém ukončení se funkce musí vrátit hned za místo, ze kterého byla zavolána.*

Rekurze

Co se přesně bude dít při zavolání funkce `délka_přepony` by mělo být celkem intuitivní – funkce se spustí, spočte se výsledek, ten se vrátí a budeme pokračovat ve vykonávání instrukcí na místě, ze kterého jsme funkci volali. Co ale třeba následující funkce na výpočet Fibonacciho čísel[2]?

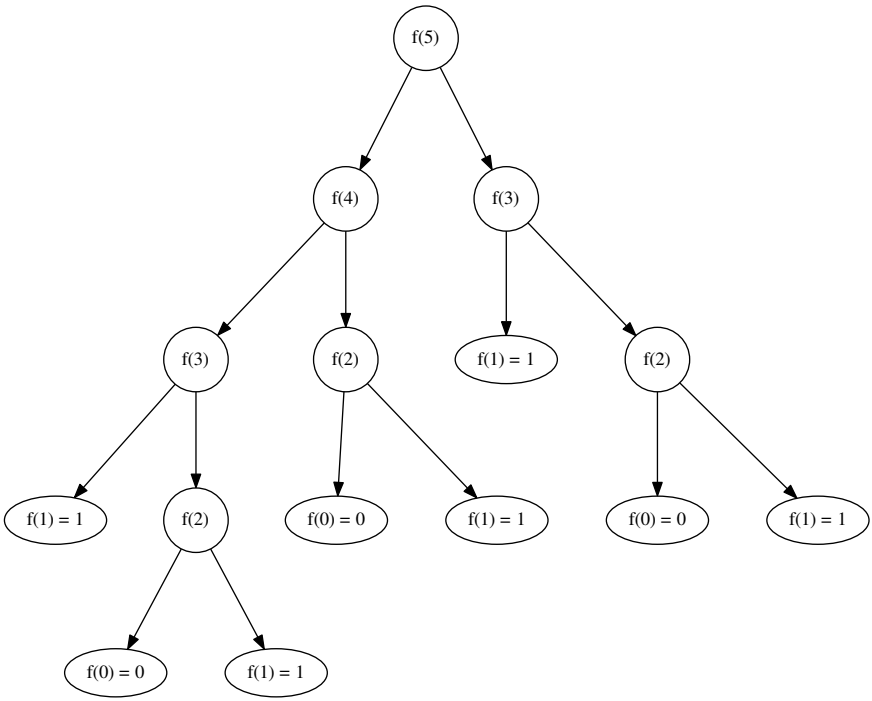
1. Funkce `fibonacciho_číslo`(n):
2. Pokud $n = 0$:
3. Vrať 0
4. Pokud $n = 1$:
5. Vrať 1
6. Vrať `fibonacciho_číslo`$(n-1)$ + `fibonacciho_číslo`$(n-2)$

Už by nám mělo být jasné, co se stane, když n bude 1, nebo 2. Když ale bude n rovno 3 nebo víc, ani jedna z prvních dvou podmínek nebude splněna a tedy se při vyhodnocování funkce dostaneme až na řádek 6. Tady by se mohlo zdát, že funkce není dobře definovaná, protože její definice závisí na sobě samé. Není tomu ale tak. Abychom pochopili, co přesně se bude dít, uvědomme si, jak funkce fungují. Když funkci zavoláme, začne se vyhodnocovat řádek po řádku a když vyhodnocování skončí, pokračuje se ve vykonávání instrukcí na místě, odkud se funkce volala. Pokud tedy funkce zavolá sebe samou, začne se se vyhodnocovat ještě jednou od začátku a když přijde k příkazu vrať, skočí na místo odkud jsme

[2]Fibonacciho posloupnost je posloupnost čísel, která začíná nulou a jedničkou a každý další člen je součtem dvou předchozích. Pokud tedy mluvíme o i-tém Fibonacciho čísle, jedná se o i-tý člen posloupnosti. Číslujeme od 0, tedy $F_0 = 0$, $F_1 = 1 \ldots$

ji zavolali, totiž do sebe samé. Hodnoty proměnných jsou lokální pro každé volání funkce, a tedy se jednotlivá volání nebudou navzájem ovlivňovat tím, že by si jednotlivá volání funkce „navzájem sahala na proměnné“.

Pokud tedy budeme chtít spočítat čtvrté Fibonacciho číslo, bude funkce `fibonacciho_číslo` zavolána celkem pětkrát. Následující diagram ukazuje, jak by výpočet probíhal. Každý bod diagramu odpovídá jednomu zavolání funkce `fibonacciho_číslo` a z každého volání vedou šipky do bodů odpovídajících vykonání funkce z něj zavolané.



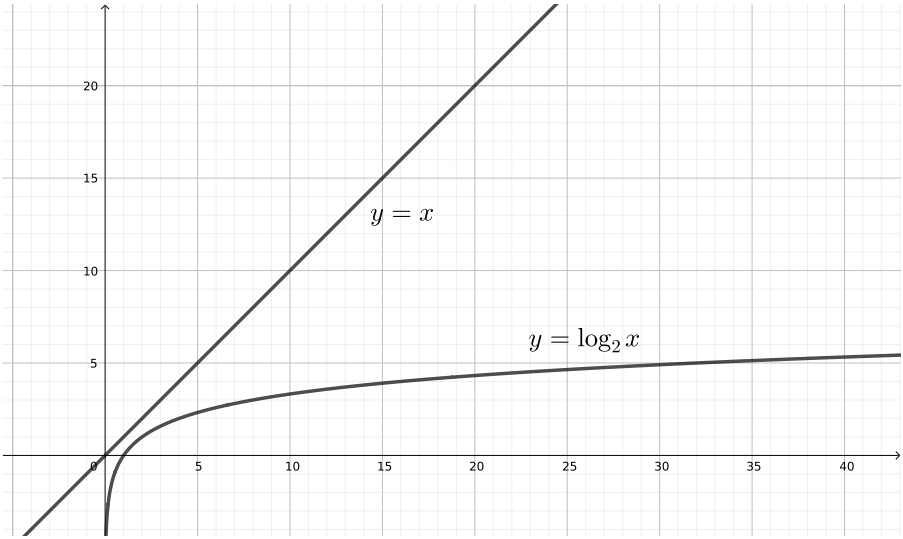
Obrázek 1: Výpočet Fibonacciho čísel

Podobným způsobem jako Fibonacciho čísla se dá rekurzivně počítat třeba i faktoriál.

Problém 2 [1b]: *Spočítejte faktoriál pomocí rekurze bez použití cyklu. Otázka, kterou je dobré si položit při řešení tohoto problému, je: „Kdybych měl spočítané* $(n-1)!$*, jak z toho spočítám* $n!$*?“. Toto je ostatně otázka, kterou je dobré si položit, kdykoliv řešíte nějaký algoritmický problém a která vede na rekurzivní algoritmus (rozmyslete si proč).*

Logaritmus

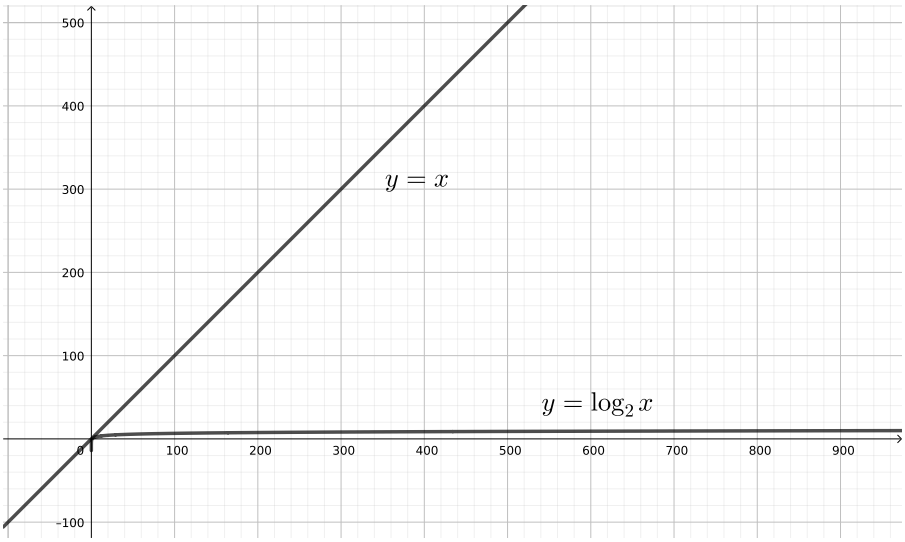
Ze školy možná znáte definici funkce zvané logaritmus. Logaritmus kladného reálného čísla x o základu a je takové reálné číslo y, pro které platí $a^y = x$. Zapisujeme $y = \log_a x$. Intuitivnější definice je, že logaritmus nám říká, kolikrát musíme základ vynásobit sebou samým, abychom dostali x. Platí tedy, že $\log_2 8 = 3$, protože $2^3 = 8$. Pokud se v informatice mluví o logaritmu, myslí se tím v drtivé většině případů logaritmus o základu 2. Stejně to bude i v našem tématku. Pokud tedy píšeme $\log n$, myslíme tím vždy $\log_2 n$.



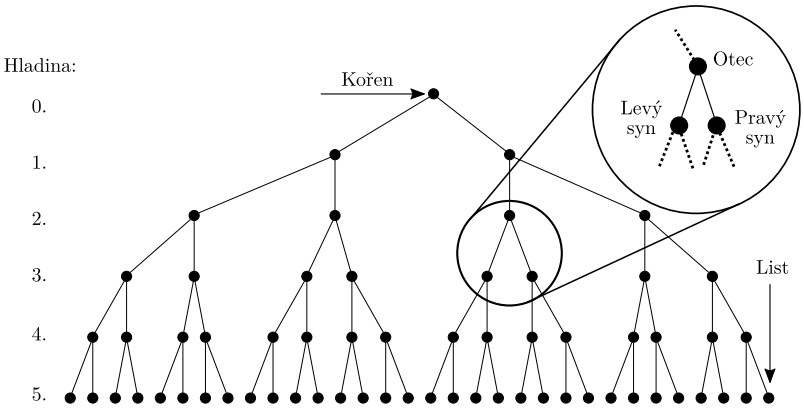
Obrázek 2: Porovnání lineání funkce s logaritmem

Na obrázku 4 můžete vidět úplný binární strom. Strom se skládá z vrcholů spojených hranami a neobsahuje cykly („větve“ se nikde nespojují). Jednotlivým „vrstvám“ vrcholů říkáme hladiny a číslujeme je od shora a od nuly. Na nulté hladině tedy leží jeden vrchol. Vrcholům, které jsou připojené pod některým vrcholem, říkáme jeho synové (podle rodokmenu, který má podobný tvar) a vrchol, který nemá ani jednoho syna, je list, zatímco vrchol bez otce se nazývá kořen (je vždy právě jeden). Strom na obrázku je navíc binární, protože každý vrchol má maximálně dva syny, a je úplný, protože každý vrchol kromě listů má dva syny a všechny listy leží na stejné hladině. Více se o stromech dozvíte v některém z dalších dílů tématka.

Můžeme si povšimnout, že na každé z hladin je dvakrát více vrcholů než na té předchozí. Na k-té hladině (počítáno shora) se tedy nachází 2^k vrcholů (na nulté hladině jich je $1 = 2^0$). Podle zmíněné definice logaritmu tedy platí, že logaritmus počtu vrcholů na dané hladině je číslo hladiny (jinak řečeno hloubka).



Obrázek 3: Porovnání lineání funkce s logaritmem – pohled z dálky



Obrázek 4: Úplný binární strom

K čemu je nám to dobré? Na stromu je hezky vidět, že logaritmus roste výrazně pomaleji než lineární funkce (když říkáme lineární funkce, budeme vždy předpokládat kladný koeficient u x). Graf těchto dvou funkcí můžete vidět na obrázcích 2 a 3. Hlavně se nám ale často stane, že něco – třeba strom všech možných průběhů algoritmu – má tvar úplného binárního stromu. Vizte následující úlohu.

Mějme panelák o n patrech (pro jednoduchost nechť n je mocnina dvojky – pokud není, tak přistavíme imaginární patra navíc aby bylo, a všimneme si,

že jsme počet pater nanejvýš zdvojnásobili). Patra jsou standardně očíslována od 0 do $n-1$. Dále mějme dostatek ideálních vajec – tedy vajec, která jsou ve všech možných fyzikálních vlastnostech shodná. Naším cílem je zjistit, z jakého nejvyššího patra můžeme ideální vejce shodit, aby se nerozbilo. Vejce se ale po každém shození rozbije nebo zakutálí, chceme proto výsledek zjistit na co nejméně pokusů. Jak budeme postupovat a kolik pokusů budeme potřebovat v nejhorším případě? Než si přečtete řešení, zkuste se nejdříve zamyslet, jak byste úlohu řešili vy.

Nabízíme následující řešení: Pokud má panelák jen jedno patro, řešení je triviální. V opačném případě vyjdeme do patra číslo $n/2$ a shodíme z něj vajíčko. Pokud se rozbije, musí se nutně rozbít i při pádu ze všech vyšších pater, tudíž nás žádné z těchto pater nezajímá. Můžeme tedy úlohu rekurzivně vyřešit pro prvních $n/2$ pater. Pokud se naopak vajíčko nerozbije, nezajímají nás žádná nižší patra – řešením je buď naše patro, nebo některé z vyšších pater. Vyřešíme tedy úlohu rekurzivně pro horní polovinu paneláku (představte si, že první polovinu paneláku odseknete, zahodíte a druhou polovinu paneláku položíte na zem – z toho by mělo být vidět, že se opravdu jedná o stejný problém poloviční velikosti). Pokud řešení nenajdeme, je řešením naše patro. Na závěr jen vrátíme číslo patra, případně informaci, že takové patro neexistuje (reprezentovanou hodnotou -1). Rozmyslete si, jak bude tato vrácená informace „vybublávat“ všemi voláními. Následuje pseudokód:

Vstup: počet pater n, funkce `shod_vajicko` která vrací 1 pro rozbité a 0 pro nerozbité vajíčko
1. Funkce `hledani_patra`(n, *prizemi*):
2. Pokud $n = 1$:
3. Pokud `shod_vajicko`(*prizemi*) $= 0$:
4. Vrať *prizemi*
5. Jinak:
6. Vrať -1

7. Pokud $n > 1$:
8. Pokud `shod_vajicko`($prizemi + n/2$) $= 1$:
9. Vrať `hledani_patra`($n/2$, *prizemi*)
10. Jinak:
11. Vrať `hledani_patra`($n/2$, $prizemi + n/2$)
Výstup: `hledani_patra`(n, 0)

Nyní nám zbývá určit časovou složitost algoritmu. Podobně jako u funkce `fibonacciho_cislo` si nakreslíme strom rekurzivních volání funkce. Je tady ale jeden zásadní rozdíl. Zatímco u Fibonacciho čísel volala funkce sama sebe vždy dvakrát, nyní se volá vždy pouze jednou. Pokud bychom tedy nakreslili, jak byla

funkce skutečně volána, dostali bychom něco, čemu se v teorii grafů říká cesta – byla by to rovná řada vrcholů spojená čarami bez jakéhokoliv větvení.

My si ale pořídíme trochu jiný strom. Nebude znázorňovat průběh jednoho spuštění algoritmu, ale všechny možné způsoby, jak může tento průběh vypadat v závislosti na vstupu. Jinými slovy, bude znázorňovat všechna rozhodnutí, která algoritmus udělal, a proto mu budeme říkat rozhodovací strom. Při každém volání se funkce v závislosti na vstupních parametrech rozhodne, zda pokračovat v hledání správného patra v dolní, nebo horní polovině paneláku. Každý z vrcholů našeho stromu značí jedno zavolání funkce a jeho dva synové tyto dvě možnosti. Pokud je funkce zavolána na vstup velikosti 1, již se dál rekurzivně nevolá a její zavolání je tak reprezentováno některým listem stromu. Tvrdíme, že tento rozhodovací strom bude mít tvar úplného binárního stromu. Proč by to tak mělo být? Když si zkusíte strom nakreslit pro nějaké malé vstupy, které jsou mocniny dvojky, zjistíte, že strom pro vstup velikosti n můžete vytvořit tak, že nakreslíte vedle sebe dvakrát rozhodovací strom pro vstup velikosti $n/2$ a přidáte nový společný kořen a z něj dvě hrany do dvou původních kořenů. Nový kořen totiž značí první volání funkce `hledani_patra`, ve kterém se rozhodneme, na kterou z polovin paneláku se rekurzivně zavolat. Další průběh algoritmu pro danou polovinu paneláku už je stejný jako pro vstup velikosti $n/2$, a proto má i stejný rozhodovací strom. Stejným způsobem si můžeme uvědomit, že počet listů stromu je vždy roven n – při zdvojnásobení vstupu se totiž zdvojnásobí i počet listů. Pokud bychom chtěli tyto vlastnosti formálně dokázat, použili bychom matematickou indukci (to je taková rekurze pro matematiky). Doufáme ale, že spojitost mezi stromem a algoritmem je vidět, a přesný důkaz zde uvádět nebudeme (pilný řešitel nám ho samozřejmě může do časopisu poslat).

Skutečný průběh algoritmu je tedy cesta z kořene do jednoho z listů. Počet zavolání funkce je tedy roven počtu hladin ve stromu, což je logaritmus počtu listů. Počet listů je ale n, čímž dostáváme, že počet volání funkce `hledani_patra` je $\log n$. Jedno volání funkce trvá konstantně dlouho, celý algoritmus tedy běží v čase $\mathcal{O}(\log n)$ a shodí $\mathcal{O}(\log n)$ vajíček.

Časovou složitost můžeme nahlédnout i bez stromu rekurze. Jak už jsme zmínili minule, každým zavoláním funkce `hledani_patra` se délka vstupu zmenší na polovinu. Když nás tedy zajímá počet volání, stačí říct, kolikrát musíme n vydělit dvěma, abychom dostali číslo 1. To je samozřejmě tolik, kolikrát musíme dvojku vynásobit samu sebou, abychom dostali n, a to je rovno $\log n$. Opět tedy dostáváme časovou složitost algoritmu $\mathcal{O}(\log n)$. Můžete si zkusit rozmyslet, že algoritmus po drobné úpravě (bude třeba $n/2$ zaokrouhlit) funguje se stejnou časovou složitostí i pro vstupy, které nejsou mocniny dvojky.

Problém 3: *Náš algoritmus na házení vajíček lze pro některé velikosti vstupů ještě o trochu zlepšit (jen o konstantu). Zkuste si rozmyslet jak a napište nám to.*

V minulém díle jsme měli algoritmus na hledání čísla v poli. To obecně nemůžeme umět rychleji než lineárně – na každé číslo se musíme alespoň jednou podívat. Co kdybychom ale měli zaručeno, že je pole setříděné (tedy čísla jsou seřazena

vzestupně)? Zkuste se inspirovat naším vajíčkovým algoritmem a vymyslete algoritmus, který v čase $\mathcal{O}(\log n)$ najde dané číslo v setříděném poli (nebo zjistí, že se v poli nenachází).

Problém 4 [2b]: *Vymyslete algoritmus, který najde v setříděném poli délky n dané číslo v čase $\mathcal{O}(\log n)$. Nezapomeňte na slovní popis algoritmu a zdůvodnění časové složitosti.*

Třídění podruhé

Nyní se zamyslíme nad tím, zda by se pomocí rekurze nedal udělat rychlejší třídící algoritmus. V minulém díle jsme vymysleli algoritmus běžící v čase $\mathcal{O}(n^2)$. Teď bychom chtěli něco rychlejšího.

Začneme s otázkou podobnou té u výpočtu faktoriálu: chceme setřídit n čísel. Předpokládejme, že už umíme setřídit $n-1$ nebo méně čísel, a zkusme pomocí toho setřídit všech n čísel. Jedním řešením by bylo setřídit prvních $n-1$ čísel a poslední číslo poté vložit do setříděné posloupnosti (zkuste si rozmyslet detaily). Prozradíme, že tak získáme algoritmus běžící v čase $\mathcal{O}(n^2)$. Je tedy třeba vymyslet jiný přístup. V následujících dvou problémech si vymyslíte, jak dosáhnout časové složitosti lepší než $\mathcal{O}(n^2)$.

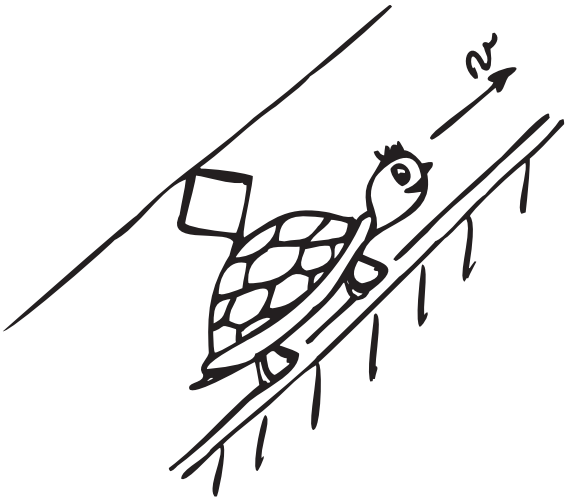
Problém 5 [3b]: *Předpokládejme, že máme dvě setříděné posloupnosti čísel (mohou být různě dlouhé). Vymyslete, jak s co nejlepší časovou složitostí z těchto dvou posloupností udělat jednu setříděnou posloupnost obsahující čísla z obou. Při vyjadřování časové složitosti použijte n jakožto celkový počet čísel. Nezapomeňte určit časovou složitost vašeho algoritmu.*

Při řešení tohoto problému si doporučujeme zkusit napsat dvě setříděné posloupnosti a napsat ručně výstupní posloupnost. Při vymýšlení algoritmů je často dobré si zkusit výsledek na malých vstupech spočítat ručně – třeba si při tom všimnete, jak to udělat efektivně.

Při řešení následujícího problému se vám bude pravděpodobně hodit využít řešení problému 4 výše. Pokud jste ho nevyřešili, můžete nyní předpokládat, že jde řešit v čase $\mathcal{O}(n)$.

Problém 6 [4b]: *Vymyslete třídící algoritmus běžící v čase lepším než kvadratickém (jinými slovy lepším než $\mathcal{O}(n^2)$). Nezapomeňte udat s odůvodněním jeho časovou složitost. Prozradíme vám, že se vám bude hodit nejen rekurze, ale i kapitola o logaritmech.*

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



Téma 5 – Přeplněná tramvaj

Určitě jste už někdy jeli vlakem nebo MHD. Možná dokonce obojím. Pravděpodobně jste si pak také všimli, že po výjezdu z výchozí stanice bývá takový vůz téměř prázdný a stejně tak bývá téměř prázdný při příjezdu do cílové stanice. Každý si asi domyslí proč. Mezi těmito dvěma body se však nachází $n-2$ dalších stanic, ve kterých se počet lidí uvnitř vozu jistým způsobem vyvíjí, pravděpodobně někde nabývá svého maxima, a tak může i vlak, který se na začátku i na konci své cesty zdá být poloprázdný, někde v polovině zcela selhat z kapacitních důvodů. Cílem každé dopravní společnosti je samozřejmě se takové věci vyvarovat a optimalizovat dopravní síť, což ovšem není možné bez předchozí analýzy vytíženosti vlaku v jednotlivých fázích cesty. Naším cílem v tomto tématku tudíž bude studium křivky popisující vývoj počtu lidí uvnitř dopravního prostředku v průběhu jeho cesty stanicemi. Budeme tedy hledat křivku, která bude mít na

ose x číslování[3] stanic na lince (pro sjednocení značení nechť je to N) a na ose y okamžitý počet lidí sedících v dopravním prostředku (nechť je to S).

Tuto úlohu budeme řešit dvěma hlavními metodami. První, jednodušší metoda, je vcelku předvídatelná: jde o experimentální stanovení naší křivky $S(N)$. V zájmu maximalizace přesnosti výsledku experimentu je pochopitelně nezbytné provést co nejvíce měření. Mnou doporučovaný postup provedení uznatelného měření spočívá v absolvování jízdy po dané lince z výchozí stanice do konečné a zaznamenávání okamžitého počtu přítomných spolucestujících mezi každými dvěma stanicemi. Kreativitě při vymýšlení jiných postupů se meze nekladou, ovšem uznány budou pochopitelně pouze ty funkční a dobře popsané.

Úloha zkoumání experimentální metodou zůstává otevřená až do odvolání nebo do uzavření tématka. Hodnocena bude podle počtu a přínosnosti provedených měření (např. z měření na lince s padesáti stanicemi lze získat o křivce $S(N)$ lepší představu než na lince s pěti stanicemi.).

Druhá metoda řešení bude metodou analytickou. Dostanete za úkol čistě matematickými metodami odvodit analytický předpis pro funkci $S(N)$. To je samozřejmě samo o sobě velmi náročným cílem, nebudeme o to tedy usilovat hned. Běžně se ve vědě totiž při hledání matematického popisu čehokoli vychází z předem známých rovnic ověřených v obecnějších případech a odvozených z nějaké prvotní úvahy. Pro situaci proměnného počtu lidí v dopravním prostředku nám žádné rovnice známy nejsou. To znamená pro vás, řešitele, příležitost si vyzkoušet pozici vědce stojícího ještě před objevením dnešních vědeckých metod a odkázaného pouze na své úvahy a svou nepodloženou představu o fungování světa. Bude zapotřebí, abyste na základě toho, jak víte, že vlaky, autobusy, tramvaje apod. fungují, vymysleli jistý zjednodušující model toho, jak se cestující dopravních prostředků chovají, aby tento model nám, teoretikům, umožňoval popsat, jakým způsobem se ve které stanici bude měnit počet lidí uvnitř prostředku. Vyvození tvaru $S(N)$ z vašeho modelu ponechávám na vás zatím čistě jako dobrovolné, samozřejmě za bonusové bodové ohodnocení.

Pozor, u hledání vhodného zjednodušujícího modelu se neočekává jednoznačné řešení! Nežádám, abyste nalezli stejný model jako já! Pokud váš model není špatný na první pohled, klidně může i on být tím správným, alespoň v jisté aproximaci. Pro rozeznání správného modelu k popisu $S(N)$ nám úvahy stačit nebudou, pro ně budeme muset využít srovnání s experimentem. Proto jestli máte nápad, nebojte se o něj podělit, třeba se právě on v některé z posledních sérií ukáže být tím pro popis nejvhodnějším. Všichni autoři nejvhodnějšího popisu pak budou odměněni body navíc.

Toto jsou vaše první úkoly k tomuto tématku:

Problém 1: *Proveďte co nejvíce měření křivky $S(N)$ dle výše uvedeného návodu na jakékoli dopravní lince autobusu, tramvaje, nebo podobného dopravního prostředku. Nezapomeňte u provedených měření zhodnotit jejich věrohodnost a dis-*

[3] Pozor, číslování se ve skutečnosti nevztahuje k samotným stanicím, nýbrž k úsekům mezi nimi (podrobnosti v příloze).

kutovat, kde mohla případná nepřesnost vzniknout. Pro přehlednost také uveďte, které linky a kdy jste zkoumali. Tento problém není časově omezený.

Problém 2: *Vymyslete jeden nebo více zjednodušujících modelů, z nichž každý nějakým způsobem popíše, dle jakých pravidel reální lidé v dopravním prostředku nastupují a vystupují, aby se na tato pravidla dalo dále navázat.*

Příloha

Teorie pravděpodobnosti

Pro snazší porozumění výše uvedené úloze je dobré zmínit zde ve stručnosti úplné základy teorie pravděpodobnosti a fyzikální statistiky.

Teorie pravděpodobnosti je odvětví matematiky založené v 17. století Blaisem Pascalem a Pierrem Fermatem za účelem popisu nedeterministických jevů, to jest takových, které při námi nerozlišitelných vstupních parametrech dávají různé výsledky. Jakým způsobem tento popis funguje, budu prezentovat na příkladu klasické šestistěnné kostky.

Příklad: Při házení férovou kostkou obyčejně výsledek hodu neumíme předem nijak odhadnout. Kostka tedy představuje *náhodný generátor* a každý hod kostkou nazýváme *pokusem*, jehož výsledkem je pak *náhodná veličina* X, což zde bude jednoduše číslo, které padne. Z formálního hlediska je to funkce $X\colon M \to \Omega$ zobrazující z nějaké množiny vstupních parametrů M (u kostky to může být například číslo, které bylo na horní stěně, když jsme kostku brali do ruky) do *prostoru možných výstupů* Ω. *Výstupem* kostky může být kterýkoli prvek $x_i \in \Omega$, zde tedy kterékoli číslo z množiny[4] $\Omega = \{1, 2, 3, 4, 5, 6\}$. Množinu $A \subseteq \Omega$ takových možných výstupů označujeme jako *jev*, který mohl nebo nemusel nastat. U kostky můžeme zadefinovat jako jevy například $A = \{x_i; x_i \text{ je liché}\} = \{1, 3, 5\}$ nebo $B = \{x_i; x_i > 3\} = \{4, 5, 6\}$.

Náhodnou veličinu charakterizuje její tzv. *pravděpodobnostní rozdělení* – funkce[5] $P\colon 2^\Omega \to [0, 1]$, která každému jevu $A \subseteq \Omega$ přiřadí číslo $P \in [0, 1]$ tak, aby:

(i) $P(\Omega) = 1$

(ii) $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

Přiřazené číslo nazýváme *pravděpodobností* daného jevu.

Můžeme si to znovu představit na příkladu kostky: Pravděpodobnostní rozdělení je zde definováno jako $P(x_i) = 1/6$ pro $i = 1 \dots 6$ (obecně je-li funkce $P(x_i)$ konstantní jako zde, příslušné rozdělení se nazývá *uniformní*).

Axiom (i) pravděpodobnostního rozdělení říká, že zcela jistě padne některý výsledek z Ω, což u kostky očividně platí, neboť $P(\{1, 2, 3, 4, 5, 6\}) = P(\{1\}) + \dots +$

[4] matematický zápis množiny, viz `https://matematika.cz/mnozinove-operace`

[5] Zápis 2^Ω se používá k označení tzv. potenční množiny, což je množina všech podmnožin Ω, viz `https://matematika.cz/mnoziny#potencni-mnozina`.

$P(\{6\}) = 1$. Axiom (ii) pak udává vztahy mezi pravděpodobnostmi jednotlivých jevů. U dvou nezávislých jevů poslední člen vypadne, pravděpodobnosti nezávislých jevů se tedy jednoduše sčítají. Platnost tohoto axiomu si můžeme znovu ověřit u kostky na dvou výše definovaných jevech A a B. Do levé strany dosadíme $P(A \cup B) = P(\{1,3,4,5,6\}) = 5/6$ a stejně tak do pravé $P(A)+P(B)-P(A \cap B) = P(\{1,3,5\}) + P(\{4,5,6\}) - P(\{5\}) = 1/2 + 1/2 - 1/6 = 5/6$.

Náhodným veličinám s číselnými výstupy bývají přiřazovány další vlastnosti, z nichž nejvýznamnější pro nás představují *střední hodnota* a *variance*. Střední hodnota $\mu(X)$ je definována jako suma součinů: $\mu(X) = \sum_{x_i \in \Omega} x_i P(x_i)$ (v případě uniformního rozdělení se tento vztah redukuje na aritmetický průměr) a má význam očekávaného[6] výsledku každého pokusu. Variance (rovněž *rozptyl*) σ^2 je potom definována velmi podobně jako $\sigma^2(X) = \sum_{x_i \in \Omega} (\mu(X) - x_i)^2 P(x_i)$. Odmocnina z variance se nazývá *směrodatná odchylka* σ a zjednodušeně řečeno má význam vzdálenosti od střední hodnoty, v níž se výsledky pokusů převážně drží.

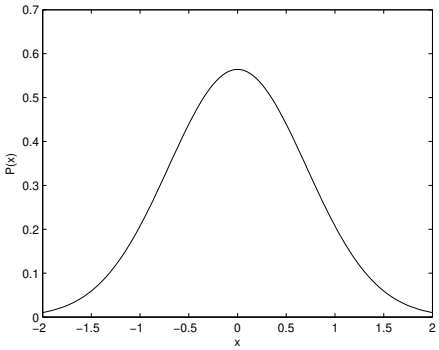
Skutečný význam pravděpodobnosti objasňuje teprve *zákon velkých čísel*, což je teorém, dle kterého se při mnohonásobném opakování pokusu relativní četnosti jednotlivých výsledků rozdělují v poměru daném pravděpodobnostmi těchto výsledků. Označíme-li tedy N počet provedených pokusů a N_i počet pokusů, které skončily výsledkem x_i, tento zákon říká, že $P(x_i) = \lim_{N \to \infty} \frac{N_i}{N}$. To jest, že i systém v principu chaotický se z dlouhodobého hlediska chová deterministicky.

Poslední zajímavou odbočku k teorii pravděpodobnosti bude v tomto stručném shrnutí představovat *Centrální limitní teorém*. Mějme náhodnou proměnnou X s libovolným pravděpodobnostním rozdělením. Nechť tato veličina vygeneruje n náhodných výsledků. Jejich součet je pak rovněž náhodnou veličinou. Označme ho $S_n(X)$. Centrální limitní teorém potom říká, že čím vyšší n zvolíme, tím více se pravděpodobnostní rozdělení S_n blíží tzv. *Gaussovu rozdělení*.

Gaussovo (tzv. normální) rozdělení je známé rozdělení zvonovitého tvaru. Právě díky Centrálnímu limitnímu teorému a jeho nezávislosti na původním rozdělení se Gaussovo rozdělení vyskytuje na mnoha místech v přírodě, a je proto dobré ho znát. Ve své nejjednodušší podobě je popsané vztahem $P(x) = \frac{1}{\sqrt{\pi}} e^{-x^2}$. Vhodně doplněnými konstantami se dá modifikovat jeho střední hodnota a variance, podoba zde uvedená se nazývá *standardní normální rozdělení*.

Problém 3 (bonusový) [3b]: *Abyste si sami prověřili, že jste tomuto článku dobře porozuměli, podívejme se nyní na případ, kdy náhodná veličina X odpovídá součtu tří nezávislých hodů šestistěnnou kostkou. Zamyslete se, co zde představuje množinu Ω, zakreslete do grafu pravděpodobnostní rozdělení a spočítejte jeho střední hodnotu a varianci.*

[6] Pozor, očekávaný výsledek nemusí být vždy ten nejpravděpodobnější. Je to výsledek, kterému se při delším opakování bude blížit průměr z jednotlivých dílčích výsledků. Například když si na strany férové mince napíšeme číslice 0 a 1, po chvíli házení se bude průměr našich výsledků blížit číslu 1/2, ačkoli pravděpodobnost, že by nám v kterémkoli hodu padla strana s takovou číslicí, je nulová.



Obrázek 5: Standardní normální rozdělení

Fyzikální statistika

Teorie pravděpodobnosti nachází dobré uplatnění mimo jiné ve fyzikální statistice, ve které samotné veličiny zkoumané sebepřesnějším experimentem představují náhodné proměnné a jejich výsledky slouží k tomu, aby za jejich pomoci byla odhadnuta střední hodnota měřené veličiny, která je teprve prohlášena za hodnotu skutečnou. Jednotlivá fyzikální měření pak z různých důvodů nevycházejí přesně a jsou od této „skutečné“ hodnoty více nebo méně odchýlena. Jejich předpokládané odchýlení je nezbytné uvést ve výsledku jako tzv. *odchylku* nebo *chybu*.

Tyto chyby se dělí na několik nejčastějších typů. Prvním druhem jsou *chyby hrubé*, které vznikají nesprávným provedením experimentu, popř. hrubým zásahem do něho. Ty je potřeba mezi výsledky rozpoznat a odstranit.

Druhou skupinu představují *chyby systematické*, vzniklé špatnou kalibrací nebo interpretací. Jejich vliv zatíží všechny výsledky stejným způsobem, je možno měření tedy dále využít, přijde-li se na to, kde a jaká chyba vznikla, a podaří-li se ji opravit.

Poslední, pro nás nejzajímavější, druh odchylek představují *odchylky statistické*, vzniklé čistě vlivy náhodnými. Statistická odchylka se po opakovaném měření projeví rozptylem hodnot, neovlivní však nijak střední hodnotu. Tyto odchylky uvádíme ve výsledku ve tvaru $x \pm \sigma_x$, kde x představuje námi odhadnutou střední hodnotu a σ_x *absolutní odchylku*, tedy informaci, o kolik se může naše hodnota lišit od hodnoty skutečné. Jak ze značení, tak z významu je zjevná její souvislost se střední kvadratickou odchylkou. Konkrétně v situaci vcelku častého standardního normálního rozdělení platí, že ve vzdálenosti nejvýš σ od střední hodnoty leží výsledek s pravděpodobností 68 %, ve vzdálenosti 2σ s pravděpodobností 95 % a ve vzdálenosti 3σ s pravděpodobností 99,7 %.

Alternativně k absolutní odchylce se využívá *odchylka relativní*, definovaná jako $\eta_x = \frac{\sigma_x}{x}$. Z ní se dá vyčíst, jak velkou nepřesnost pro nás ve skutečnosti absolutní odchylka znamená.

Ze zákona velkých čísel vyplývá, že čím více měření bude provedeno, tím blíže by střední hodnota našich výsledků měla být střední hodnotě skutečného rozdělení pravděpodobnosti. Z toho vyplývá jednak, že ve skutečně věrohodných fyzikálních experimentech je potřeba provést měření velice mnoho, a jednak, že při rostoucím počtu provedených měření by měla odchylka našeho odhadu střední hodnoty klesat. Pro naše účely stačí vědět, že tento pokles se dá odhadnout jako $\sigma_{\tilde{x}} = \frac{\sigma_x}{\sqrt{n}}$, kde n je počet měření a $\tilde{x}$ střední hodnota veličiny x (podrobnosti a odvození možno dohledat v [2]).

Stává se rovněž běžně, že se u výsledné hodnoty nastřádá statistických odchylek víc, buďto z nepřímého měření, nebo odlišných původů. Jedná-li se o *měření nepřímé*, tedy že hodnota vznikla matematickými operacemi s přímo naměřenými hodnotami, uplatníme *metodu přenosu chyb*, tedy vztah:

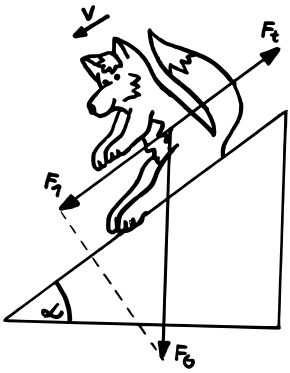
$$\sigma_{f(x_1,\ldots,x_n)^2} = \sum_{k=1}^{n} \left(\frac{df}{dx_k} \sigma_{x_k} \right)^2$$

tedy např.:

$$\sigma_{xy}{}^2 = x^2 \sigma_y{}^2 + y^2 \sigma_x{}^2$$

Dvě odchylky nezávislých původů se pak skládají na kombinovanou standardní nejistotu: $\sigma^2 = \sigma_x^2 + \sigma_y^2$.

Odvození všech těchto vztahů lze znovu dohledat v [2].



Nápověda k řešení

Zde si na závěr dovolím několik poznámek k úloze, které nepatří do zadání, přesto je však dobré si je před zahájením řešení přečíst.

Především bychom si měli sjednotit číslování stanic, které není zcela přímočaré. Už bylo ostatně nastíněno v poznámce k zadání, že se N vztahuje nikoli ke stanicím, nýbrž k úsekům mezi nimi (hodnoty však budeme stále vynášet na

celých číslech). Doporučuji úseky číslovat vždy podle stanice, z níž vůz právě vyjíždí. Dále doporučuji výchozí depo, resp. konečné depo považovat za nultou, resp. $(n+1)$-ní stanici (n nechť značí celkový počet stanic na lince). S tímto číslováním je jisté, že v $N = 0$ bude vynesena hodnota S příslušející úseku mezi depem a první stanicí, kde je samozřejmě $S = 0$, stejně tak jako v $N = n$. Křivka $S(N)$ pak bude vycházet z bodu $[0, 0]$ a časem se do nuly zase vrátí, což bude jednak estetičtější, jednak přehlednější.

Tato úloha sice není fyzikální, některé rysy fyziky však přesto vykazuje. Okamžitý počet lidí v dopravním prostředku je jistá náhodná veličina, jejíž množinou možných výsledků je $\Omega = \mathbb{N}_0$ a jejíž rozdělení pravděpodobnosti stejně jako námi hledaná střední hodnota závisí jednak na fázi trasy, v níž se prostředek právě nachází (což je jediná závislost, kterou my chceme zkoumat), mimoto ale také na denní době, vytíženosti linky, typu prostředku apod. Abychom všechny tyto vedlejší závislosti z našeho měření odfiltrovali, musíme se nad úlohou nejdřív trochu zamyslet.

Na začátku úlohy, když bylo zadáno hledání obecného tvaru křivky $S(N)$, byl tímto zadáním zároveň vysloven předpoklad, že takový obecný tvar existuje, tj. že křivka $S(N)$ si ponechává ve všech situacích tvar stejný, který se pouze jistým způsobem roztahuje a smršťuje právě v souvislosti s vytížeností linky a s počtem stanic na lince (v prvním případě se to projeví modifikací ve svislém směru, ve druhém případě ve vodorovném). Tento předpoklad není podložen ničím kromě intuice a pravděpodobně se dokonce časem ukáže, že v jistých speciálních případech neplatí.

Přesto můžeme zatím předpokládat jeho platnost alespoň v situacích vzájemně si podobných. Není tedy například nesmyslné srovnávat tvar $S(N)$ u linky s deseti a u linky s jedenácti stanicemi. Srovnání takových linek ovšem nelze provést v grafu, jehož osa x bude odpovídat přímému číslování stanic N (přinejmenším proto, že jedna z křivek bude končit o stanici dál). Proto pro hledání obecného tvaru naší křivky doporučuji přejít k redukovanému číslování stanic $\tilde{N} = \frac{N}{n+1}$ ($n + 1$ je skutečná délka trasy vynášené na grafu).

Sami si rozmyslete, jak podobným způsobem co nejlépe redukovat okamžitý počet cestujících S na $\tilde{S}$.

Nyní jsme od křivky $S(N)$ přešli ke křivce $\tilde{S}(\tilde{N})$. Ta by dle našeho předpokladu měla mít stejný tvar jako $S(N)$ (šlo jen o přenásobení konstantami) pouze s rozdílem, že jejím definičním oborem i oborem hodnot bude interval $[0, 1]$. To je pro nás úžasný výsledek, protože nyní jsme upravili naši křivku do takové podoby, která zachovává její tvar a přitom není závislá ani na počtu stanic na lince, ani na tom, kolik lidí linkou právě jede (jde jen o to, jak se tento počet relativně mění).

Tím pádem jsme se zbavili závislosti zkoumané střední hodnoty na všech výše uvedených nepodstatných parametrech a zbyla nám pouze kýžená závislost na fázi cesty. Ani ta však není zcela nezávadná. Asi každý si dovede představit, že na pevně dané lince jsou některé stanice více frekventované než jiné. Proto kdy-

bychom měření prováděli neustále na stejné lince, vyšla by nám i po redukci odlišná křivka, než kdybychom stejné měření prováděli na lince jiné. Proto vyjdeme-li z předpokladu, že rozložení frekventovaných stanic na Nespecifikované lince je náhodné, mé další doporučení ke hledání obecného tvaru bude nabádat ke zkoumání přednostně většího počtu linek jednou nežli jedné linky mnohokrát. Dodržování tohoto doporučení už bude hodnoceno skrze přínosnost provedeného měření (viz zadání).

Tímto doporučením dnešní odbočku k nápovědám k řešení zakončuji, přeji hodně štěstí při řešení jak experimentální části, tak i teoretické.

Reference

[1] Ch. M. Grinstead, J. L. Snell: Introduction to Probability, American Mathematical Society, Copyright (C) 2003

[2] J. Englich: Úvod do praktické fyziky I: Zpracování výsledků měření, MATFYZPRESS, Praha 2006

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

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