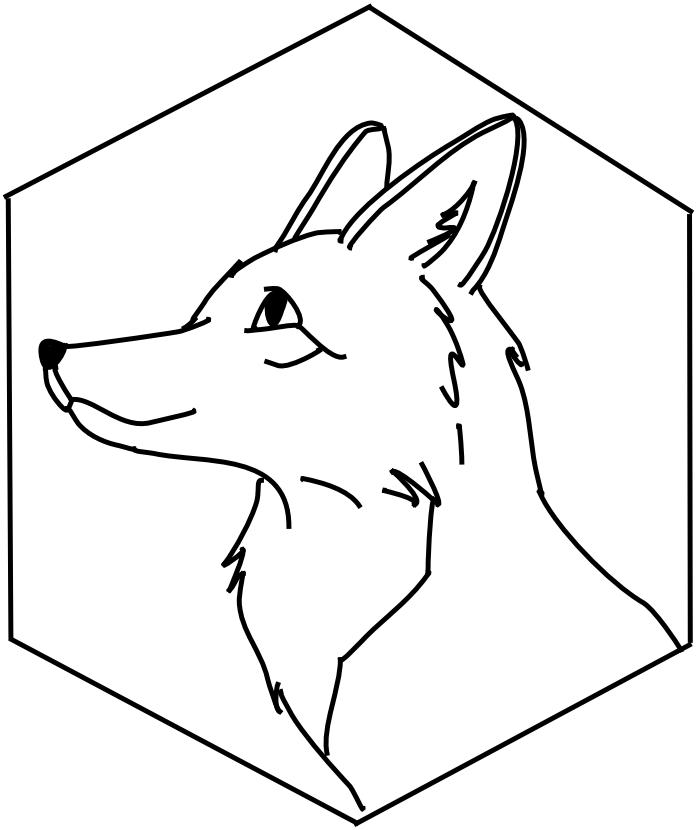
STUDENTSKÝ ČASOPIS A KORESPONDENČNÍ SEMINÁŘ

Ročník XXX
Číslo 6

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 a ta nejzajímavější z nich otiskneme. Nejlepší řešitele zveme na podzim a na jaře na soustředění.

Milí řešitelé,

s koncem školního roku přichází závěr 30. ročníku Korespondenčního semináře M&M. Léto klepe na dveře a s ním i poslední číslo našeho časopisu. Už v něm nenajdete žádná nová zadání, ale čekají na vás vzorová řešení úloh z minulého čísla témátek Lean, Termodynamika a FlatFox#. Lean navíc přináší i závěrečný díl věnující se formální verifikaci programů. Uzavírá se i témátko Hex, kde vám konečně prozradíme, ze kterého článku témátko vycházelo. A aby toho nebylo málo, můžete si přečíst vydařený řešitelský článek Měrná tepelná kapacita hliníkového válečku od Dr.MM Radima Nováka.

Rádi bychom vám všem poděkovali za účast v letošním ročníku a za všechny skvělé články a řešení, které jste nám poslali. Nejvíce bodů nakonec získala Doc.MM Jana Uglickich, a stává se tak vítězem tohoto ročníku. Moc gratulujeme! Nejlepším článkem ročníku byl vybrán článek Problém klikujícího hráče výzvy, jehož autor Mgr.MM Jáchym Löwenhöffer tak vyhrává dort. Také gratulujeme! A ostatní vyzýváme, řešte náš seminář dál!

Pro ty z vás, kteří už se nemohou dočkat dalších matematických, fyzikálních a informatických výzev, máme dobrou zprávu: na našich stránkách už je k dispozici první číslo nového, 31. ročníku, plné nových témátek a zadání.

A než se úplně rozloučíme, máme na vás ještě jednu prosbu. Pomozte nám zlepšovat náš seminář dál a věnujte chvilku vyplnění ankety o uplynulém ročníku: `https://forms.office.com/e/uXW8B1DJhC`

Přejeme vám krásné léto plné odpočinku, dobrodružství, nebo třeba hraní Hexu. Užijte si volné dny a těšíme se na vás v novém ročníku!

Vaši organizátoři

Anketa:

`https://forms.office.com/e/uXW8B1DJhC`

Obsah

Řešení témat

Téma 1 – Termodynamika

Vzorová řešení úloh 5. čísla

Níže vám přinášíme řešení Úloh 5.1, 5.2 a Problému 5.3 od Doc.^{MM} Martina Těšitele a také řešení Úlohy 5.4 od Doc.^{MM} Jany Uglickich.

Úloha 5.1

Zadání:

Máme systém 4 nerozlišitelných částic s takovou energií, která stačí k přechodu tří z nich na vyšší energetickou hladinu (což je popis makrostavu). Kolik mikrostavů se navenek projeví jako tento makrostav se třemi excitovanými částicemi? Jak se tato úloha změní, když bude mít systém 4 částic energii jen pro excitaci dvou, čtyř, nula nebo jedné částice? Částice jsou bosony, může se jich nacházet na jedné hladině více najednou.

Řešení od Doc.^{MM} Martina Těšitele:

Počet mikrostavů v makrostavu se dá popsat vzorcem:

$$K(k,n) = \frac{n!}{(n-k)! \cdot k!},$$

kde n je počet všech částic a k počet všech excitovaných částic. Vzorec

$$V(k,n) = \frac{n!}{(n-k)!}$$

je vzorec určující počet variací bez opakování a s rozlišením jednotlivých částic. Získané číslo je proto potřeba vydělit počtem permutací excitovaných částic, což nám dává výše zmíněný vzorec.

Z tohoto vzorce vyplývá, že počet mikrostavů pro k excitovaných částic ze čtyř je popisován touto tabulkou:

k	Počet mikrostavů
0	1
1	4
2	6
3	4
4	1

Víme, že tyto výpočty nejsou úplně špatně, protože součet všech mikrostavů odpovídá tomu v temátku.

Úloha 5.2

Zadání:

Uvažujte stejné podmínky pro 4 izolované částice jako v předchozí úloze. Dokažte, že počet mikrostavů je stejný pro nula a čtyři excitované částice (využijte k tomu výpočty z předchozí úlohy). Najděte makrostavy, které lze popsat největším počtem mikrostavů.

Řešení od Doc.^{MM} Martina Těšitele:

Ze vzorce pro $K(k,n)$ plyne výpočet:

$$\begin{aligned}\frac{n!}{(n-k_1)!\cdot k_1!} &= \frac{n!}{(n-k_2)!\cdot k_2!}\\ \frac{4!}{(4-4)!\cdot 4!} &= \frac{4!}{(4-0)!\cdot 0!}\\ \frac{4!}{1\cdot 4!} &= \frac{4!}{4!\cdot 1}\\ 0 &= 0,\end{aligned}$$

který je důkazem toho, že pro nula a čtyři excitované částice je počet mikrostavů stejný. Největší počet mikrostavů má makrostav, jehož k je nejblíže $n/2$, což je pro tento případ $k = 2$. Tomu odpovídá i tabulka všech makrostavů a mikrostavů pro čtyři částice z Úlohy 5.1.

Problém 5.3

Zadání:

Dokažte, že tato symetrie počtu mikrostavů v systému se 2 energetickými hladinami nezávisí na počtu částic. Nápověda: Zamyslete se nad úlohou obecně pro N částic a sepište si, jak byste spočetli počet realizací jednoho makrostavu, kde je k částic excitováno. Platí, že $k \leq N$.

Řešení od Doc.^{MM} Martina Těšitele:

Vycházejme z analogie s mincemi. Hodím-li k-krát pannu, musel jsem hodit i $(N-k)$-krát orla. Počet mikrostavů v jednom makrostavu je (pro tuto analogii) definován jako počet způsobů, kterými lze uspořádat excitované částice (panny nebo orly, je jedno, která strana mince bude reprezentovat jaký stav) v systému o N částicích.

Představme si, že jsem hodil všemi mincemi a padla mi k-krát panna. Tím pádem mi musel padnout $(N-k)$-krát orel. Řekněme, že nyní bude excitovanou částici reprezentovat panna. Když tedy hodím k-krát pannu, mám přesně $\binom{N}{k}$ způsobů, jak padlé panny uspořádat. Ale co když si v tomto případě vyberu jako reprezentanta excitovaného stavu orla? Potom mám přesně $\binom{N}{N-k}$ způsobů, jak padlé orly uspořádat. A protože jsou panna a orel jako reprezentanti stavů zaměnitelní (je jedno, jestli je reprezentant excitovaného stavu panna, nebo orel), je počet mikrostavů pro makrostavy o k a $N-k$ excitovaných částicích stejný pro jakýkoli počet částic.

Formálněji:

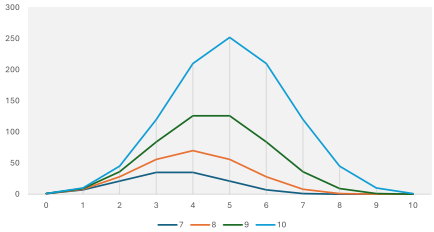
$$\binom{N}{k} = \frac{N!}{(N-k)! \cdot k!}$$

$$\binom{N}{N-k} = \frac{N!}{(N-(N-k))! \cdot (N-k)!} = \frac{N!}{k! \cdot (N-k)!}$$

$$\frac{N!}{(N-k)! \cdot k!} = \frac{N!}{k! \cdot (N-k)!}$$

$$\binom{N}{k} = \binom{N}{N-k}$$

Hezké grafické znázornění symetrie nám poslal František Nouza, který vykreslil počty mikrostavů pro jednotlivé makrostavy pro systémy se 7–10 částicemi. Pro zájemce i přikládáme odkaz na Geogebru[1], kde navíc posunul maxima nad sebe pro lepší znázornění symetrie.



Obrázek 1: Počty mikrostavů pro makrostavy 7—10 částic od F. Nouzy

[1] `https://www.geogebra.org/m/s22fwz2m`

Úloha 5.4

Zadání:

Systém A obsahuje 10 částic a 3 energetické hladiny, systém B obsahuje 10 částic a 2 energetické hladiny. Porovnejte hodnotu entropie těchto dvou systémů. Částice jsou opět bosony, může se jich nacházet na jedné hladině více najednou.

Řešení od Doc.MM Jany Uglickich:

$$S_A = k \cdot \ln w_1 = k \cdot \ln 3^{10} \approx 10{,}99k$$
$$S_B = k \cdot \ln w_2 = k \cdot \ln 2^{10} \approx 6{,}93k$$

Entropie systému A je tedy větší o $4{,}06k$, což je přibližně $5{,}6 \cdot 10^{-23}\,\mathrm{JK}^{-1}$.

Na závěr ještě přinášíme článek Dr.MM Radima Nováka o tom, jak měřil měrnou tepelnou kapacitu hliníkového válečku – najdete jej na straně 25.

Pája, Helča a Pavel; termodynamika@ledoian.cz

Téma 2 – Programování a dokazování v Leanu

Díl 6: Formální verifikace pro programátory

Poslední tři díly byly dosti matematické. Nyní si ukážeme něco málo z formální verifikace, která by mohla oslovit programátory.

Doběhne to vůbec někdy?

Možná si vzpomínáte, jak jsme v prvním díle definovali ciferný součet

```
partial def ciferny_soucet (a : Nat) : Nat :=
if a < 10
then a
else (a % 10) + ciferny_soucet (a / 10)
```

a Ackermannovu funkci[2]

```
partial def ackermann : Nat → Nat → Nat
| 0   , n   => n+1
| m+1, 0   => ackermann m 1
| m+1, n+1 => ackermann m (ackermann (m+1) n)
```

a říkali vám, ať si s tím klíčovým slovem `partial` nelámete hlavu. Klíčové slovo `partial` znamená „neslibuju, že dojdu k nějaké hodnotě“.

Lean se snaží být velmi bezpečný programovací jazyk, a tak, není-li dáno jinak, u každé funkce zaručuje, že někdy doběhne (kdyby tomu tak nebylo, mohli bychom dokonce někdy „dokázat“ nepravdivé matematické věty). Když píšeme rekurzivní funkce, většinou spadají pod některý ze speciálních případů, pro které Lean umí automaticky dokázat terminaci. Občas napíšeme funkci, která tu vlastnost, že vždy doběhne (neboli terminuje), má, ale Lean to neprozře sám od sebe.[3] V takovém případě musíme Leanu nějak pomoct. Pojďme se podívat znovu na náš ciferný součet, ale tentokrát zkusme zaručit terminaci:

```
def ciferny_soucet' (a : Nat) : Nat :=
if a < 10
then a
else (a % 10) + ciferny_soucet' (a / 10)
decreasing_by
  simp_wf
  sorry
```

[2] `https://cs.wikipedia.org/wiki/Ackermannova_funkce`

[3] poznámka pro velmi pokročilé čtenáře: To, že Halting problem `https://cs.wikipedia.org/wiki/Halting_problem` je nerozhodnutelný, platí i pro funkcionální programovací jazyky.

Kámen úrazu je, že Lean neví, že `a / 10 < a` platí při všech rekurzivních voláních. Leanu by kupříkladu pomohlo říct:

```
have : a / 10 + a / 10 ≤ a
```

Zbytek by už `linarith` obstarala sama. Snazší než dokazovat tvrzení výše je však najít v knihovně důkaz tohoto (silnějšího) tvrzení:

```
have : a / 10 * 10 ≤ a
• exact Nat.div_mul_le_self a 10
```

Celá deklarace, která se zkompiluje bez chyby, vypadá takto:

```
def ciferny_soucet' (a : Nat) : Nat :=
if a < 10
then a
else (a % 10) + ciferny_soucet' (a / 10)
decreasing_by
  simp_wf
  have : a / 10 * 10 ≤ a
  • exact Nat.div_mul_le_self a 10
  linarith
```

Problém s Ackermannovou funkcí je odlišného rázu. Zde nejde o to, že by Lean nevěděl, proč určitá kvantita klesá; problém je v tom, že Lean ani neví, která kvantita by klesat měla. Řešení je zde překvapivě snadné:

```
def ackermann' : Nat → Nat → Nat
| 0  , n   => n+1
| m+1, 0   => ackermann' m 1
| m+1, n+1 => ackermann' m (ackermann' (m+1) n)
termination_by
  ackermann' m n => (m, n)
```

Zde tvrdíme, že uspořádaná dvojice `(m, n)` postupně klesá (lexikograficky). Ptáte se, proč je to pravda? Podívejme se spolu na všechna rekurzivní volání, která tu nastávají:

Aktuální vstup	Rekurzivní volání	Zdůvodnění
`ackermann' (m+1) 0`	`ackermann' m 1`	klesl první argument
`ackermann' (m+1) (n+1)`	`ackermann' (m+1) n`	první argument zůstal stejný a druhý klesl
`ackermann' (m+1) (n+1)`	`ackermann' m k`	klesl první argument; `k` je tu libovolné číslo, často krutopřísně velké

Všechna tato pozorování udělal Lean automaticky, takže ta `decreasing_by` část tu není potřeba.

Dělá ta funkce to, co chci?

Možná si vzpomínáte, jak jsme v druhém díle naprogramovali obrácení seznamu:

```
def obrat {T : Type} : List T → List T
| [ ]           => []
| hlava :: telo => obrat telo ++ [hlava]
```

Pak jsme uvedli, že obrácení seznamu lze naprogramovat způsobem, který je těžší na pochopení, ale rychlejší na vykonání počítačem:

```
def obrat_rychl {T : Type} (pripoj : List T) : List T → List T
| [ ]           => pripoj
| hlava :: telo => obrat_rychl (hlava :: pripoj) telo

def obrat_rychle {T : Type} (seznam : List T) : List T :=
obrat_rychl [] seznam
```

Empiricky jsme si ověřili, že `obrat_rychle` dává stejné výsledky jako `obrat`, ale důkaz jsme samozřejmě nepředložili. To nás čeká dnes! A bude to dobré cvičení, protože mnohé z vás tehdy mátlo, proč vůbec `obrat_rychle` funguje. Chceme teď dokázat něco takového:

```
theorem obrat_rychle_vs_obrat {T : Type} (l : List T) :
    obrat_rychle l = obrat l := by
  sorry
```

Leanovštější je to však vyjádřit jako rovnost funkcí:

```
theorem obrat_rychle_eq_obrat {T : Type} :
    @obrat_rychle T = @obrat T := by
  sorry
```

Napsat `obrat_rychle = obrat` bohužel nestačí, protože bez přítomnosti explicitního argumentu Lean neví, jak obsadit implicitní typový argument. To je cena za to, že jsme naprogramovali funkci pro obrat jakéhokoliv seznamu, ne jen seznamu například celých čísel.

Přemýšlejme o tom v klidu tak, že dokazujeme `obrat_rychle = obrat` neboli větu ∀ `l : List T, obrat_rychle l = obrat l`. Zkuste si:

```
theorem obrat_rychle_eq_obrat {T : Type} :
    @obrat_rychle T = @obrat T := by
  ext1 l
  unfold obrat_rychle
  sorry
```

První krok hodí `l` do kontextu; cíl je `obrat_rychle l = obrat l` po tomto kroce. Druhý krok rozbalí vnější definici, která slouží jen ke snazšímu volání funkce, a očima teď vidíme, co budeme ve skutečnosti muset dokázat:

```
⊢ obrat_rychl [] l = obrat l
```

Odoláme nutkání mechanicky pokračovat `unfold obrat_rychl` (zkuste si, co by se stalo). Poněvadž je funkce `obrat_rychl` definována indukcí na druhém argumentu, mělo by být přirozené, že i důkaz provedeme indukcí. Inu, první nápad bude vypadat nějak takhle:

```
theorem obrat_rychle_eq_obrat {T : Type} :
    @obrat_rychle T = @obrat T := by
  ext1 l
  unfold obrat_rychle
  induction l with
  | nil => rfl
  | cons a z ih =>
    sorry
```

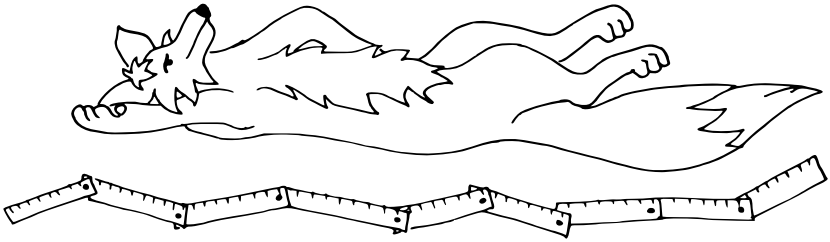
Při pohledu na cíl `obrat_rychl [] (a :: z) = obrat (a :: z)` nás napadne `unfold obrat_rychl obrat` (teď už se ví, který pattern každá definice matchne) a následný cíl `obrat_rychl [a] z = obrat z ++ [a]` nás naplní hřejivým pocitem. Nabízí se použít indukční předpoklad zprava doleva:

```
theorem obrat_rychle_eq_obrat {T : Type} :
    @obrat_rychle T = @obrat T := by
  ext1 l
  unfold obrat_rychle
  induction l with
  | nil => rfl
  | cons a z ih =>
    unfold obrat_rychl obrat
    rw [←ih]
    sorry
```

Umíme dokázat vzniklý cíl?

```
⊢ obrat_rychl [a] z = obrat_rychl [] z ++ [a]
```

Hmmm. Ta rovnost platí, to né že ne! Ale umíme ji dokázat? Ummm. Asi by bylo trapné, kdybychom neuměli! Takže, jaký je další krok? Nikdo se tu nehlásí. Tak se podívejme, co máme v šuplíku. Můžeme udělat indukci na `z`. To je koneckonců jediný seznam v současném kontextu.



```
theorem obrat_rychle_eq_obrat {T : Type} :
    @obrat_rychle T = @obrat T := by
  ext1 l
  unfold obrat_rychle
  induction l with
  | nil => rfl
  | cons a z iha =>
    unfold obrat_rychl obrat
    rw [←iha]
    induction z with
    | nil => rfl
    | cons b w ihb =>
      unfold obrat_rychl
      sorry
```

Ujistíme se, že nový cíl `obrat_rychl [b, a] w = obrat_rychl [b] w ++ [a]` určitě platí, ale protože `iha` ani `ihb` nevypadají, že by šly okamžitě použít v cíli, pokračujeme tím jediným, co máme na jazyku – indukcí na `w`. Prokřupeme prsty a jedem!

```
theorem obrat_rychle_eq_obrat {T : Type} :
    @obrat_rychle T = @obrat T := by
  ext1 l
  unfold obrat_rychle
  induction l with
  | nil => rfl
  | cons a z iha =>
    unfold obrat_rychl obrat
    rw [←iha]
    induction z with
    | nil => rfl
    | cons b w ihb =>
      unfold obrat_rychl
      induction w with
      | nil => rfl
      | cons c v ihc =>
        unfold obrat_rychl
        sorry
```

Nový cíl `obrat_rychl [c, b, a] v = obrat_rychl [c, b] v ++ [a]` vypadá, že opět platí, ale lokální kontext opět vypadá, že nám moc nepomůže v jeho důkazu. Chce se vám někomu napsat další indukci? Asi ne. Cítíme se tu jako křeček v kole[4], či spíše jako Sisyfos[5], jehož balvan se navíc stává těžším a těžším

[4] `https://dictionary.cambridge.org/dictionary/english/hamster-wheel`
[5] `https://cs.wikipedia.org/wiki/Sisyfovsk%C3%A1_pr%C3%A1ce`

po každém skoulení na zem.

V čem je tedy kámen úrazu? S každým krokem se nám mění nejen druhý argument (zkracuje se o první symbol), ale zároveň první argument (ten se prodlužuje)! Radši se vraťme na začátek:

```
theorem obrat_rychle_eq_obrat {T : Type} :
    @obrat_rychle T = @obrat T := by
  ext1 l
  unfold obrat_rychle
```

Cíl `obrat_rychl [] l = obrat l` lze také chápat jako `obrat_rychl [] l = obrat l + []`. Důkaz si usnadníme tím, že dokážeme něco silnějšího! Napadá vás nějaké užitečné zobecnění tohoto tvrzení? Prázdný seznam je speciální případ seznamu. Vyslovíme tedy lemma:

```
lemma obrat_rychl_vs_obrat {T : Type} (x y : List T) :
    obrat_rychl x y = obrat y ++ x := by
  sorry
```

To by snad mohlo platit! Než se vrhneme na důkaz lemmatu, bleskově si ověříme, že z něj plyne kýžená věta:

```
theorem obrat_rychle_eq_obrat {T : Type} :
    @obrat_rychle T = @obrat T := by
  ext1 l
  convert obrat_rychl_vs_obrat [] l
  simp
```

Za `x` jsme dosadili `[]` a za `y` jsme dosadili `l`.

Zbývá dokázat to lemma. Zkušenosti z minulého dílu, zejména pokud jste řešili poslední úlohu, nám radí, že to máme umlátit indukcí (taky jak jinak – když jsme se nakonec použití indukce vyhnuli v důkazu věty, bude patrně potřeba ji použít tady). Začavše

```
lemma obrat_rychl_vs_obrat {T : Type} (x y : List T) :
    obrat_rychl x y = obrat y ++ x := by
  induction y with
  | nil => rfl
  | cons d l ih =>
    sorry
```

se však dostaneme do podobných problémů jako minule. Pokud místo toho použijeme `induction x with`, tak se nedostaneme vůbec nikam. Potřebovali bychom indukci, která nenechává ani jeden z argumentů na místě. Jak to ale udělat? Není potřeba nový indukční princip; lze to poskládat z nástrojů, které už známe.

Mějme na paměti, že každá indukce musí někdy dojít k bázickému případu (neboli základu indukce). Středobodem důkazu tedy musí být indukce na `y`, neboť tento argument se zkracuje, když má `obrat_rychl` rekurzivní volání. Potřebujeme

však umožnit, aby se `x` mohlo při indukčním kroku prodlužovat. Jak by však šlo něco takového udělat? Třeba tak, že povolíme, aby `x` bylo jakékoliv!

Asi si říkáte, co to je za blbost, vždyť `x` není konstanta. Při volání lemmatu `obrat_rychl_vs_obrat` můžeme za `x` dosadit cokoliv! Inu, při volání můžeme. Problém je, že v okamžiku, kdy `y` máme v kontextu (což tady máme hned od hlavní dvojtečky), je hodnota `x` již zafixována. My bychom ale potřebovali, aby v okamžiku, kdy známe `y` (nebo aspoň víme, jak je dlouhé), bylo `x` stále ještě kvantifikováno. A když nám tuhle věc tactic mód nedal, tak mu ji prostě vnutíme sami! Napíšeme přesně takové pomocné tvrzení (pro každé `x'` na místě `x` a pro dané `y` z kontextu) a hned si ověříme, že lemma z pomocného tvrzení vyplývá dosazením `x` z kontextu:

```
lemma obrat_rychl_vs_obrat {T : Type} (x y : List T) :
    obrat_rychl x y = obrat y ++ x := by
  have pom : ∀ x' : List T, obrat_rychl x' y = obrat y ++ x'
  • sorry
  exact pom x
```

A teď přichází hlavní podívaná večera! Zavoláme taktiku `induction` dříve než `intro`, přestože náš cíl začíná univerzálním kvantifikátorem. Vypadá to takhle:

```
lemma obrat_rychl_vs_obrat {T : Type} (x y : List T) :
    obrat_rychl x y = obrat y ++ x := by
  have pom : ∀ x' : List T, obrat_rychl x' y = obrat y ++ x'
  • induction y with
    | nil =>
      intro x'
      sorry
    | cons d l ih =>
      intro x'
      sorry
  exact pom x
```

Tímto jsme odložili „konkretizaci“ seznamu `x'` o „jeden krok“ později. A to je přesně to, co nám stačí k tomu, aby indukce vedla tam, kam chceme! Když to napíšeme schematicky, původně jsme byli nuceni dokázat

```
∀ x, (P x l → P x y)
```

kdežto nyní dokazujeme

```
(∀ x, P x l) → (∀ x, P x y)
```

kde `y` je `d :: l` a kde `P x y` vyjadřuje `(obrat_rychl x y = obrat y ++ x)`. Když si zapřemýšlíme o rozdílech mezi uvedenými tvrzeními, shledáme, že to první tvrzení (do jehož důkazu nás taktika `induction` tlačila defaultně) je silnější, ale to druhé tvrzení nám stačí! Indukční schéma, které tu implementujeme, lze vyjádřit takto:

```
(∀ x : List T, P x []) ∧
(∀ d : T, ∀ l : List T, ((∀ x, P x l) → (∀ x, P x (d :: l)))) →
(∀ x : List T, ∀ y : List T, P x y)
```

Hlavní struktura důkazu je tímto zvolena; pojďme dořešit chybějící části:

```
lemma obrat_rychl_vs_obrat {T : Type} (x y : List T) :
    obrat_rychl x y = obrat y ++ x := by
  have pom : ∀ x' : List T, obrat_rychl x' y = obrat y ++ x'
  • induction y with
    | nil =>
      intro x'
      rfl
    | cons d l ih =>
      intro x'
      unfold obrat_rychl obrat
      rw [ih]
      exact List.append_cons (obrat l) d x'
  exact pom x
```

Předposlední řádek důkazu byl nalezen pomocí `library_search`. Na předcházejícím řádku můžeme vidět, že si taktika `rw` automaticky domyslela argument; při explicitním podání by přepsání cíle pomocí indukčního předpokladu vypadalo takto:

```
      rw [ih (d :: x')]
```

Zverifikovali jsme, že funkce `obrat` a `obrat_rychle` dělají to samé!

Ještě než se rozloučíme, zkusme ten důkaz společně zjednodušit nebo spíš zkrátit (často se tomu říká „codegolf“, nebo jen „golf“, které lze použit i jako sloveso – např: „I golfed your proof to three lines.“).

Můžeme si všimnout, že řádek deklarující `pom` vypadá dost genericky – pouze opakujeme znění lemmatu s provedením zobecnění, které jsme si podrobně vysvětlili. Takový trik není ojedinělý, a proto existuje varianta taktiky `induction`

umožňující specifikovat, které termy, jež se již vyskytují v kontextu, chceme zobecnit. Napíší se za slovo `generalizing` vložené před `with` a, je-li jich více, oddělují se jen mezerou. Tato varianta indukce nás zprostí od nutnosti vyslovit pomocné tvrzení i od důkazu, že cíl je speciálním případem tohoto zobecněného tvrzení. Důkaz pak vypadá takhle:

```
lemma obrat_rychl_vs_obrat {T : Type} (x y : List T) :
    obrat_rychl x y = obrat y ++ x := by
  induction y generalizing x with
  | nil =>
    rfl
  | cons d l ih =>
    unfold obrat_rychl obrat
    rw [ih]
    exact List.append_cons (obrat l) d x
```

Taktika `simp` pro zjednodušování výrazů toho dokáže hodně udělat za nás. Zde můžeme pomocí `simp` vyřešit bázický krok i indukční krok (musíme však explicitně uvést, které definice má `simp` rozbalovat):

```
lemma obrat_rychl_vs_obrat {T : Type} (x y : List T) :
    obrat_rychl x y = obrat y ++ x := by
  induction y generalizing x with
  | nil => rfl
  | cons d l ih => simp [obrat_rychl, obrat, ih]
```

Vzápětí, máme-li ďábelskou náladu, můžeme využít i toho, že existuje jedno volání jedné taktiky, které ve stejném znění řeší bázický i indukční krok:

```
simp_all [obrat_rychl, obrat]
```

To nám umožní použít syntax `<;>` s významem „všechny vzniklé cíle vyřeš takto“. A nakonec i důkaz hlavní věty lze zkrátit taktikou `simp`. Když provedeme všechna uvedená „zgolfení“, vypadá úplně celý důkaz takto:

```
lemma obrat_rychl_vs_obrat {T : Type} (x y : List T) :
    obrat_rychl x y = obrat y ++ x := by
  induction y generalizing x <;> simp_all [obrat_rychl, obrat]

theorem obrat_rychle_eq_obrat {T : Type} :
    @obrat_rychle T = @obrat T := by
  simp [obrat_rychle, obrat_rychl_vs_obrat]
```

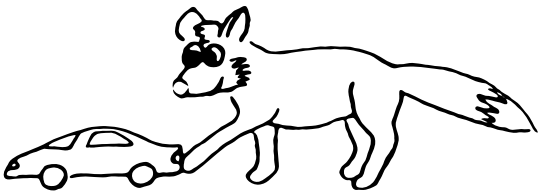
Mechanické části důkazu jsou omezeny na úplné minimum; pouze části kódu vyžadující kreativní myšlení zůstávají napsané.

A když už máme tuhle krásnou větu, můžeme si jako zákusek přidat jeden elegantní důsledek poslední úlohy:

```
theorem obrat_rychle_obrat_rychle {T : Type} (l : List T) :
    obrat_rychle (obrat_rychle l) = l := by
  rw [obrat_rychle_eq_obrat, obrat_obrat]
```

Kam dál?

Protože je to poslední díl, nenajdete tu již žádné úlohy. Uvádíme tu však pro vás několik odkazů na stránky s dalšími materiály.
Hlavní internetové fórum věnované Leanu: `https://leanprover.zulipchat.com/`
Discord server pro uživatele Leanu: `https://discord.gg/WZ9bs9UCvx`
Pokud jste ještě nehráli Natural Number Game, zahrajte si ji teď (a to obzvláště, pokud jste ignorovali poslední dva díly tématka – NNG je skvělý způsob, jak se naučit matematickou indukci v Leanu téměř bez předchozích znalostí): `https://adam.math.hhu.de/` (jsou tam také další Lean hry)
Functional Programming in Lean (elektronická kniha):
`https://leanprover.github.io/functional_programming_in_lean/`
Mathematics in Lean (elektronická kniha):
`https://leanprover-community.github.io/mathematics_in_lean/`



Řešení úloh z 5. dílu

Zadání:
Dokažte (s využitím čehokoliv, co jsme v tématku probrali):

```
def prvnich_n_krychli_sestupne : ℕ → List ℕ
| 0   => []
| n+1 => n^3 :: prvnich_n_krychli_sestupne n

def prvnich_n_krychli : ℕ → List ℕ :=
obrat ∘ prvnich_n_krychli_sestupne

theorem soucet_prvnich_n_krychli (n : ℕ) :
    soucet (prvnich_n_krychli n) = ((n-1) ^ 2 * n ^ 2) / 4 := by
  sorry
```

Příklad:

$$0^3 + 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 0 + 1 + 8 + 27 + 64 + 125 = 225 = \frac{5^2 \cdot 6^2}{4}$$

Řešení:
Protože se odčítání a dělení chová na přirozených číslech ošklivě, je klíčové najít lemma, které tyto operace nemá ve svém znění, dokázat ho indukcí a pak z něj odvodit zadanou větu. A vyhnout se v té induktivní části funkci `obrat` je také výhodné. Zkusme na to jít takhle:

```
lemma soucet_prvnich_n_krychli_sestupne (m : ℕ) :
    soucet (prvnich_n_krychli_sestupne m.succ) * 4 =
    (m ^ 2 * m.succ ^ 2) := by
  sorry
```

Plánem je dosadit `m := n - 1` a rovnici vydělit čtyřkou. Teď se dělení bude chovat správně, protože naše výrazy jsou celočíselné násobky čtyř. Než se pustíme na důkaz lemmatu, pojďme si ověřit, že zadaná věta z něj opravdu plyne:

```
theorem soucet_prvnich_n_krychli (n : ℕ) :
    soucet (prvnich_n_krychli n) = ((n-1) ^ 2 * n ^ 2) / 4 := by
  cases n with
  | zero => decide
  | succ m => sorry
```

Protože minus jednička není přirozené číslo, musíme ji ošetřit samostatně. Ostatně případ `n = 0` stejně nedává moc smysl. Není to součet `0^3` jako v případě `m = 0` v lemmatu výše, nýbrž součet přes prázdnou množinu. Takový cíl pochopitelně obsahuje pouze konstanty, takže `decide` je samozřejmou volbou. Cíl pro ostatní hodnoty je následující:

```
⊢ soucet (prvnich_n_krychli m.succ) =
  (m.succ - 1) ^ 2 * m.succ ^ 2 / 4
```

Následují dva kroky, které nás zbaví obrácení seznamu:

```
    dsimp [prvnich_n_krychli]
    rw [obrat_zachovava_soucet]
```

Využíváme tu lemma z minulého dílu. Cíl teď vypadá takto:

```
⊢ soucet (prvnich_n_krychli_sestupne m.succ) =
  (m.succ - 1) ^ 2 * m.succ ^ 2 / 4
```

To vypadá skoro přesně na to, abychom `soucet_prvnich_n_krychli_sestupne` aplikovali zprava doleva (nemáme tu vynásobení čtyřkou, takže zleva doprava aplikovat nelze). Krátká příprava a provedeme ten klíčový přepis:

```
    convert_to
      soucet (prvnich_n_krychli_sestupne m.succ) =
      m ^ 2 * m.succ ^ 2 / 4
    rw [←soucet_prvnich_n_krychli_sestupne m]
```

Cíl nyní je:

```
⊢ soucet (prvnich_n_krychli_sestupne m.succ) =
  (soucet (prvnich_n_krychli_sestupne m.succ) * 4) / 4
```

Vypadá to, že jsme skoro doma! Napíšeme `simp` a cíl je splněn.

Zbývá dokázat lemma `soucet_prvnich_n_krychli_sestupne`. Začátek toho důkazu teď již nikoho nepřekvapí:

```
lemma soucet_prvnich_n_krychli_sestupne (m : ℕ) :
    soucet (prvnich_n_krychli_sestupne m.succ) * 4 =
    (m ^ 2 * m.succ ^ 2) := by
  induction m with
  | zero => decide
  | succ n ih => sorry
```

Cíl v indukčním kroku vypadá takto:

```
⊢ soucet (prvnich_n_krychli_sestupne n.succ.succ) * 4 =
  n.succ ^ 2 * n.succ.succ ^ 2
```

Nyní bychom chtěli rozbalit jednu vrstvu definic, ale `unfold` i `dsimp` nám dávají divné výsledky. Proto specifikujeme ručně, jak má výsledek rozbalení vypadat:

```
    convert_to
      ((n+1) ^ 3 + soucet (prvnich_n_krychli_sestupne (n+1))) * 4 =
      (n+1) ^ 2 * (n+2) ^ 2
```

Indukční předpoklad vypadá takto:

```
ih : soucet (prvnich_n_krychli_sestupne n.succ) * 4 =
     n ^ 2 * n.succ ^ 2
```

Vidíme, že je potřeba roznásobit závorku nalevo:

```
    convert_to
      (n+1) ^ 3 * 4 + soucet (prvnich_n_krychli_sestupne (n+1)) * 4 =
      (n+1) ^ 2 * (n+2) ^ 2
```

Pomocný cíl splníme pomocí `ring` a konečně můžeme pomocí `rw [ih]` použít indukční předpoklad. Dojde k přepisu druhého sčítance a cíl teď vypadá takto:

```
⊢ (n + 1) ^ 3 * 4 + n ^ 2 * n.succ ^ 2 = (n + 1) ^ 2 * (n + 2) ^ 2
```

Nyní se nám nelíbí, že je napsáno `n.succ` v cíli, který obsahuje výraz `(n + 1)`. Sjednotíme to:

```
    convert_to
      (n+1) ^ 3 * 4 + n ^ 2 * (n+1) ^ 2 =
      (n+1) ^ 2 * (n+2) ^ 2
```

Taktika `ring` za nás ověří, že rovnost platí. Hotovo!

U úlohy této obtížnosti je obvyklé, že v průběhu řešení zajdeme do několika slepých uliček, ze kterých se různě dlouho vymotáváme. Důkaz, který je tady představen, taktéž nebyl napsán na první pokus bez dotyku klávesy backspace.

Nedělejte si těžkou hlavu z toho, pokud váš důkaz má několik stovek řádků. Představený důkaz je výsledek několika iterací zjednodušování, přičemž jsme se ve všech fázích vývoje omezovali na znalosti, které byly předané v časopise.

U formálních důkazů není nutné, aby byly napsány hezky. Člověk bude číst většinou jen to tvrzení; důkaz má být čitelný pro počítač. Pro vás, čtenáře, jsme však důkaz zkrášlili. Celý důkaz vypadá takto:

```
lemma soucet_prvnich_n_krychli_sestupne (m : ℕ) :
    soucet (prvnich_n_krychli_sestupne m.succ) * 4 =
    (m ^ 2 * m.succ ^ 2) := by
  induction m with
  | zero => decide
  | succ n ih =>
    convert_to
      ((n+1) ^ 3 + soucet (prvnich_n_krychli_sestupne (n+1))) * 4 =
      (n+1) ^ 2 * (n+2) ^ 2
    convert_to
      (n+1) ^ 3 * 4 + soucet (prvnich_n_krychli_sestupne (n+1)) * 4 =
      (n+1) ^ 2 * (n+2) ^ 2
    • ring
    rw [ih]
    convert_to
      (n+1) ^ 3 * 4 + n ^ 2 * (n+1) ^ 2 =
      (n+1) ^ 2 * (n+2) ^ 2
    ring

theorem soucet_prvnich_n_krychli (n : ℕ) :
    soucet (prvnich_n_krychli n) = ((n-1) ^ 2 * n ^ 2) / 4 := by
  cases n with
  | zero => decide
  | succ m =>
    dsimp [prvnich_n_krychli]
    rw [obrat_zachovava_soucet]
    convert_to
      soucet (prvnich_n_krychli_sestupne m.succ) =
      m^2 * m.succ^2 / 4
    rw [←soucet_prvnich_n_krychli_sestupne m]
    simp
```

Zadání:
Dokažte, že když dvakrát obrátíme seznam, dostaneme původní seznam.

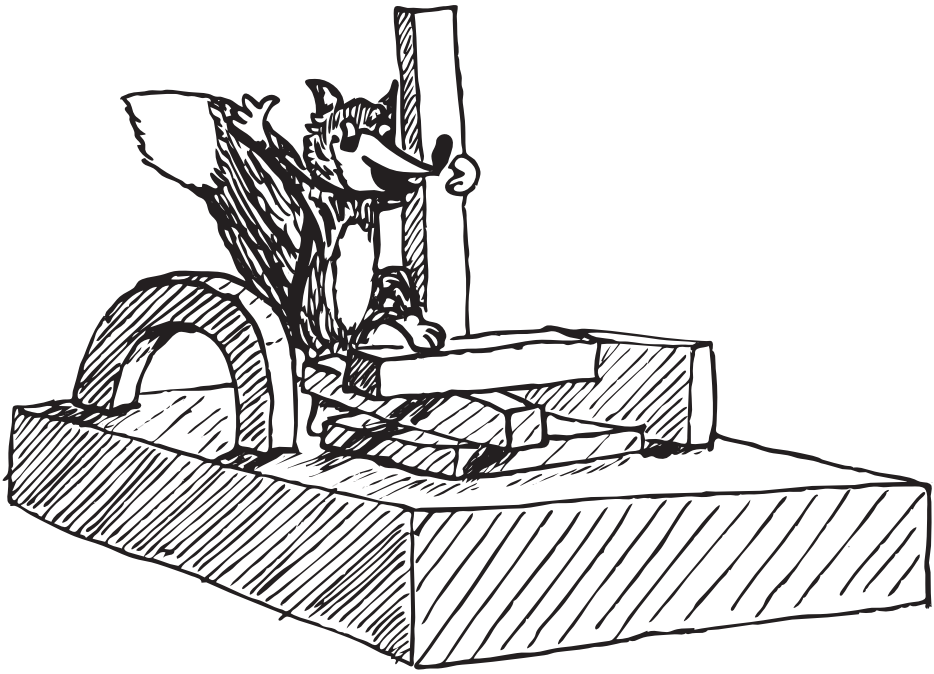
Řešení:
Tuhle úlohu šlo vyřešit dost krátkým kódem:

```
lemma obrat_pripoj {T : Type} (x : List T) (a : T) :
    obrat (x ++ [a]) = a :: obrat x := by
  induction x with
  | nil => rfl
  | cons d l ih => simp [obrat, ih]

theorem obrat_obrat {T : Type} (x : List T) :
    obrat (obrat x) = x := by
  induction x with
  | nil => rfl
  | cons d l ih => simp [obrat, ih, obrat_pripoj]
```

Taktika `simp` udělala většinu práce za nás. Na správné lemma jsme samozřejmě museli přijít sami (což tady bylo docela snadné).

Martin Dvořák; `martin.dvorak@matfyz.cz`



Téma 3 – Hex

V tomto tématku inspirovaném článkem *Infinite Hex is a draw*[6], který napsali Joel D. Hamkins a Davide Leonessi, jsme dokázali několik zajímavých poznatků o hře Hex, vymysleli a zanalyzovali spoustu zajímavých modifikací a našli vítězné strategie na konkrétních hracích plochách. Nakonec jsme se dostali i k tomu, že větší hrací plochy můžeme poskládat z menších, u kterých už vítěznou strategii známe.

Doufáme, že se vám tématko líbilo a že si třeba teď, o prázdninách, Hex s někým zahrajete. Krásné léto přejí

Bětka; betka.n@centrum.cz
Jidáš; jonas.havelka@volny.cz

Téma 5 – FlatFox#

Řešení 2. dílu

Vzhledem k tomu, že většina došlých řešení byla velmi podobná, uvádíme autorská řešení:

Úloha 5.1

Zadání:
Mějme město o 100 domech, v nich bydlí $d_1, \ldots, d_{100}$ obyvatel. Chtěli bychom najít podmnožinu domů mající dohromady přesně N obyvatel (nebo ukázat, že tam taková není). K tomu použijeme program, který součet obyvatel každé z 2^{100} podmnožin domů porovná s číslem N. Jaká je asymptotická časová složitost tohoto programu (vzhledem k velikosti N)? (Uvažujme, že sčítání a porovnávání jakýchkoliv čísel nám trvá konstantní čas.)

Porovnejte výsledek se zjištěním, že tento program prozkoumá

$$2^{100} > 1\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000$$

různých podmnožin, tedy poběží (na dnešních počítačích) určitě déle, než jak starý je vesmír.

Řešení:
Vzhledem k tomu, že pro všechna N proběhne program úplně stejně, nezávisí jeho časová složitost na N, tedy je konstantní, tj. $\mathcal{O}(1)$.

Všimněte si, že přestože má program konstantní časovou složitost, je prakticky absolutně nepoužitelný. Tudíž v praxi je potřeba dívat se i na konstanty a nejen na asymptotickou složitost.

[6] https://arxiv.org/pdf/2201.06475.pdf

Úloha 5.2

Zadání:
Jakou (asymptotickou) časovou složitost má ukázané řešení první podúlohy 4.1 (mazání registru)?

Řešení:
Lišák proběhne r-krát okolo lesa, přičemž každý oběh mu trvá konstantní počet kroků (10, poslední cyklus je pak pouze 7 kroků), a poté udělá jeden krok do cíle. Tudíž časová složitost řešení je $\mathcal{O}(10r - 2) = \mathcal{O}(r)$.

Úloha 5.3

Zadání:
Jakou (asymptotickou) časovou složitost má ukázané řešení poslední podúlohy 4.1 (zjišťování, zda je číslo záporné)?

Zkuste si řešení pustit (`https://flatfox.moznabude.cz/#MTu1.ffs`), podívat se, kterými cykly liška běhá. Pravděpodobně nejtěžší bude zjistit, kolikrát (v závislosti na r) těmito cykly liška proběhne.

Řešení:
Řešení sestává ze dvou cyklů, mezi kterými se přepíná tam a zpátky. Při podrobnějším prozkoumání zjistíme, že cykly zajišťují, aby se červený registr vrátil zpět na svoji hodnotu a pak se červený registr buď sníží nebo zvýší (záleží na tom, jestli jsme v horním nebo dolním cyklu) o 1, 2, 3, ... Ve chvíli, kdy narazíme na 0 v červeném registru, končíme. To bude tehdy, když snížíme/zvýšíme červený registr o $|r|$, tedy oba cykly spustíme $|r|$-krát. Po každém spuštění proběhne liška cyklem maximálně $2|r|$-krát (vrácení červeného registru na původní hodnotu a jeho zmenšení/zvětšení maximálně o $|r|$) a každý cyklus má konstantně mnoho kroků. Tedy asymptotická časová složitost je $\mathcal{O}(|r|^2)$.

Úloha 5.4

Zadání:
Najděte rychlejší řešení zápornosti a určete jeho časovou složitost.

Řešení:
Jedním řešením (se kterým však nikdo nepřišel) je nezmenšovat/nezvětšovat červený registr pouze o 1 oproti jeho předchozímu minimu/maximu („nejít o 1 dál, než jsme byli naposledy“), ale zmenšovat/zvětšovat ho vždy o 1, 2, 3, ... víc („jít o 1, 2, 3, ... dál“). To znamená, že na 0 narazíme už po $\sqrt{|r|}$ cyklech (až na konstantu), neboť po n spuštěních cyklu jsme se pokusili zmenšit/zvětšit registr už o $1+2+3+\ldots+n = \frac{n(n+1)}{2} > \frac{n^2}{2}$. Tedy máme časovou složitost $\mathcal{O}(|r| \cdot \sqrt{|r|})$.

Ještě rychlejší je však řešení, které vždy provede dvojnásobné (oproti minulému) zvětšení/zmenšení červeného registru. To znamená, že na nulu natrefí už při $\log_2(|r|)$ provedení cyklů, neboť při n-tém spuštění cyklu snížíme/zvýšíme červený registr o 2^n. Tedy $\mathcal{O}(|r| \cdot \log|r|)$ je jistě časová složitost tohoto řešení.

Ale my můžeme provést ještě lepší odhad. V n-tém spuštění cyklu provedeme (až na konstanty) 2^n kroků, tudíž počet kroků, které provedeme do n-tého spuštění (včetně) bude $1 + 2 + 4 + 8 + \ldots + 2^n = 2 \cdot 2^n - 1$. A jelikož na nulu narazíme tehdy, když bude 2^n poprvé větší než $|r|$, tak 2^n je při posledním spuštění cyklu až na konstantu rovné $|r|$, tudíž počet kroků je $\mathcal{O}(|r|)$.

Úloha 5.5

Zadání:
Spojitá verze poslední podúlohy 4.1: Představte si nekonečný rovný plot, ve kterém je právě jedna díra ve vzdálenosti D od místa, kde stojíte. Vy ale neznáte ani D, ani směr, kterým díra je. Jak byste takovou díru hledali a jaká je (asymptotická, tj. v $\mathcal{O}$ notaci, a v závislosti na D) ušlá vzdálenost? (Čím pomaleji poroste funkce odhadující tuto vzdálenost, tím více bodů dostanete.)

Řešení:
Úloha byla zařazena spíše jako jiný pohled na předchozí úlohu. Tedy stačí říct, že řešení je stejné jako při hledání, zda je číslo záporné. Zajímavá tato úloha začne být až, když bychom hledali nejlepší konstantu, tedy poměr mezi ušlou vzdáleností a vzdáleností díry. Ale do toho nebudeme zabíhat.

Úloha 5.6

Zadání:
Proč nemůže program na počítání druhé mocniny (používající pouze `@+-^>v<#`) běžet rychleji než v $\mathcal{O}(r^2)$?

Řešení:
Jediné, co je k této úloze potřeba, jsou pozorování, že neumíme registr změnit o více než 1 a zároveň potřebujeme zvýšit registr s výsledkem alespoň na n^2. Tudíž potřebujeme alespoň n^2 kroků.

Problém 4.7

Zadání:
Naprogramujte další zajímavé programy. Nabízí se například nalezněte n-té (nebo ověřte, zda číslo je): prvočíslo, Fibonacciho číslo, dokonalé číslo, šťastné číslo.

Řešení:
Poté, co Doc.[MM] Jana Uglickich naprogramovala program ověřující prvočíselnost čísla (`#JUp7.ffs`[†]), naučila naši lišku také hledat n-té prvočíslo: `#JUprv.ffs`[†]. Načež, inspirována nápadem Dr.[MM] Lucie Zůnové, napsala program, který dosti rychlým způsobem ověřuje, zda je číslo druhá mocnina celého čísla[7]: `#JU2.ffs`[†].

[†]Pro zpřehlednění píšeme místo `https://flatfox.moznabude.cz/#nazevsouboru.ffs` pouze `#nazevsouboru.ffs`. V elektronické verzi jsou názvy souborů prokliknutelné. Děkujeme za pochopení.

[7]Dokumentaci k programu ověřujícímu, zda je číslo druhá mocnina, si můžete přečíst na: `https://flatfox.moznabude.cz/JU2.pdf`

Mgr.[MM] Miroslav Stýskala zase spojil své znalosti kombinatoriky a FlatFox#, čímž získal praktické prográmky na výpočet permutací, kombinací a variací:

- permutace bez opakování (aneb faktoriál): `#MS1.ffs`† (červená vstup, zelená výstup);
- variace bez opakování (aneb $n!/(n-k)!$): `#MS2.ffs`† (červená n, zelená k, modrá výstup);
- variace s opakováním (aneb n^k): `#MS3.ffs`† (červená n, zelená k, modrá výstup);
- kombinace bez opakování (aneb $\binom{n}{k}$): `#MS4.ffs`† (červená n, zelená k, modrá výstup)
- kombinace s opakováním (aneb $\binom{n+k-1}{k}$): `#MS5.ffs`† (červená n, zelená k, modrá výstup)
- permutace 3 prvků s opakováním (aneb $(k_1+k_2+k_3)!/(k_1!\cdot k_2!\cdot k_3!)$): `#MS6.ffs`† (červená, zelená a modrá k_1 až k_3, fialová výstup).

Díl třetí: Rozloučení

Vyzkoušeli jsme si programování v netradičním programovacím jazyce (lišák Riki řídící se příkazy poskládanými do dvojrozměrného lesa), který nám pak umožnil jednoduše provádět časovou analýzu (jako počet kroků lišáka), které se žádný student informatiky nevyhne.

Reálné programovací jazyky mají o mnoho složitější sadu příkazů a funkcí, pro programátora je však důležitější umět si rozmyslet, jak problém řešit, implementace v konkrétním programovacím jazyce už je jen implementační detail.

Teď už však vypněme počítač a pojďme se koupat či si jinak užívat léta.

Jidáš; `jonas.havelka@volny.cz`

†Pro zpřehlednění píšeme místo `https://flatfox.moznabude.cz/#nazevsouboru.ffs` pouze `#nazevsouboru.ffs`. V elektronické verzi jsou názvy souborů prokliknutelné. Děkujeme za pochopení.

Měrná tepelná kapacita hliníkového válečku 6b

Dr.^{MM} *Radim Novák*

Teoretická část

Kalorimetrická rovnice

Jsou-li dvě tělesa s různou teplotou v kontaktu, dochází k tepelné výměně. Pokud jsou v izolované soustavě, tak platí zákon zachování energie. Chceme-li vyjádřit teplo odevzdané či přijaté, použijeme vzorec

$$Q = m_1 c_1 (t - t_1),$$

kde m_1 je hmotnost tělesa, c_1 je jeho měrná tepelná kapacita, t_1 jeho počáteční teplota a t je koncová teplota. Pokud těleso teplo přijímá, je teplo kladné, a pokud teplo odevzdává, je teplo záporné. Pro dvě tělesa pak dostaneme rovnici

$$m_1 c_1 (t - t_1) = m_2 c_2 (t_2 - t),$$

kde levá strana představuje studenější těleso a pravá těleso teplejší. Na pravé straně máme výraz $(t_2 - t)$, kde jsou prohozená „téčka“ – je to proto, aby obě strany rovnice vycházely kladně a rovnice platila.[8] Obecně do rovnice můžeme přidávat takový počet členů, jaký má naše soustava.

Někdy potřebujeme znát tepelnou kapacitu určitého objektu. Tu dostaneme následujícím vztahem:

$$C_1 = m_1 c_1,$$

kde m_1 je hmotnost tělesa a c_1 je jeho měrná tepelná kapacita.

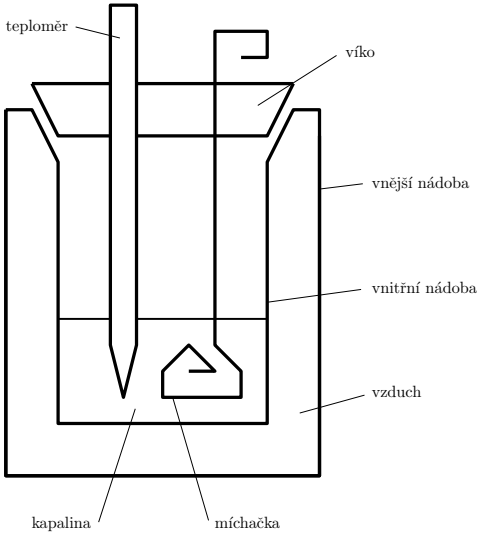
Kalorimetr

Kalorimetr se používá k měření tepelných kapacit těles. Na to ovšem potřebujeme znát tepelnou kapacitu kalorimetru. Pokud známe měrnou tepelnou kapacitu kapaliny (většinou vody), můžeme tepelnou kapacitu kalorimetru spočítat podle vztahu:

$$C_k = \frac{c_1 m_1 (t_1 - t)}{t - t_k}, \tag{1}$$

kde indexem 1 označujeme kapalinu, indexem k kalorimetr a t je koncová teplota.

[8] Pozn. redakce: Prohození nastává kvůli tomu, že u jednoho tělesa počítáme teplo odevzdané, zatímco u druhého teplo přijaté.



Obrázek 2: Kalorimetr

Výsledky měření

Kalorimetr

Nejdříve jsme změřili tepelnou kapacitu kalorimetru C_k tak, že jsme do něj nalili vodu o teplotě 76,6 °C a hmotnosti 249,6 g a čekali, až se teplota ustálí. Počáteční teplotu kalorimetru jsme naměřili 23,5 °C. Pak ze vztahu (1) vyšlo:

$$C_k = 49{,}4 \pm 0{,}1\,\mathrm{J{\cdot}K^{-1}}.$$

Hliníkový váleček

Do kalorimetru vložíme hliníkový váleček o hmotnosti 42 g a zalijeme horkou vodou o teplotě 73,2 °C. Kalorimetr i váleček mají počáteční teplotu 23,5 °C. Po ustálení měla soustava teplotu 69,3 °C. Hodnoty naměřené i spočtené jen dosadíme do rovnice popisující naši soustavu:

$$c_1 m_1 (t_1 - t) = c_2 m_2 (t - t_2) + C_k (t - t_k),$$

kde indexem k označujeme kalorimetr, indexem 1 vodu a indexem 2 hliníkový váleček. Po dosazení vyšlo:

$$c_2 = 941{,}6 \pm 0{,}1\,\mathrm{J{\cdot}kg^{-1}K^{-1}}.$$

Diskuze výsledků

Naměřená hodnota se od tabulkové liší asi o $50\,\mathrm{J{\cdot}kg^{-1}K^{-1}}$. To může být způsobeno nedokonale izolovanou soustavou, konkrétně únikem tepla. Rovnice počítá s izolovanou soustavou a proto to vypadá, že si váleček hliníku bere tepla víc než ve skutečnosti. V kalorimetru byla neucpaná díra pro teploměr, kterou teploměr zcela neutěsnil. Ještě je problém v počáteční teplotě kalorimetru v druhé části. Horkou vodu jsme vylili a kalorimetr zchladili pod proudem studené vody, následně jsme počkali, až se teplota kalorimetru ustálí, ale neznali jsme přesnou teplotu kalorimetru, jen přibližnou pokojovou teplotu.

Závěr

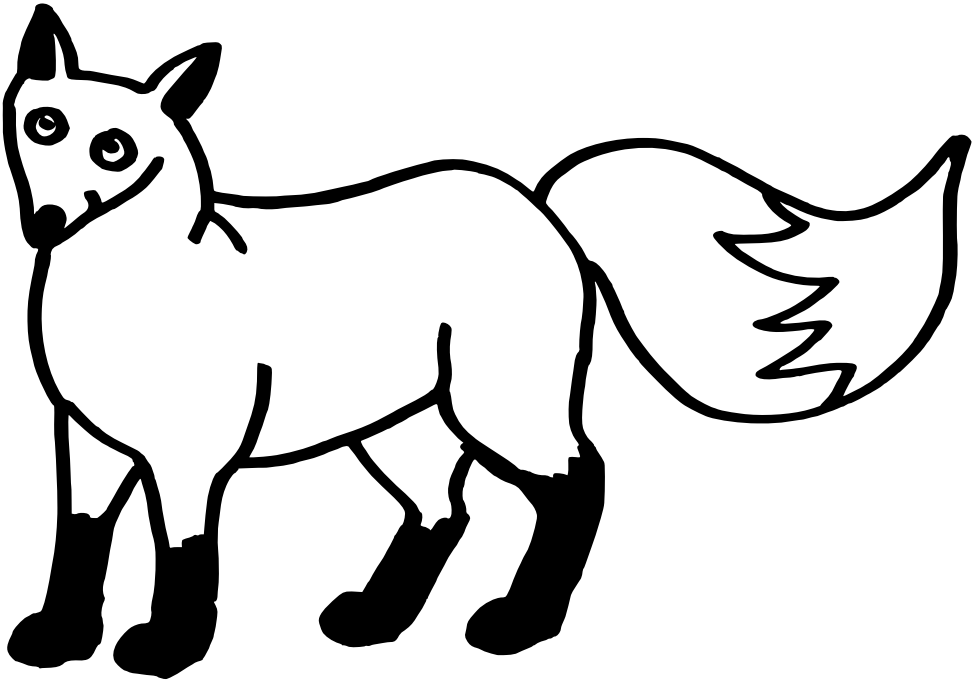
Změřili jsme tepelnou kapacitu kalorimetru a měrnou tepelnou kapacitu hliníku:

$$C_k = 49{,}4 \pm 0{,}1\,\mathrm{J{\cdot}K^{-1}}$$
$$c_2 = 941{,}6 \pm 0{,}1\,\mathrm{J{\cdot}kg^{-1}K^{-1}}.$$

Výsledek se neshoduje s tabulkovou hodnotou.

Seznam použité literatury

MIKULČÁK, Jiří. Matematické, fyzikální a chemické tabulky a vzorce pro střední školy. Praha: Prometheus, 2003. ISBN 80-7196-264-3



Výsledková listina 2. deadlinu 4. čísla a obou deadlinů 5. čísla

Poř.	Jméno	R.	$\sum_{-1}$	Témata 1	2	3	4	5	O	$\sum_0$	$\sum_1$
1.	Doc.MM J. Uglickich	2	328,3	6,0	14,0		3,5	10,5		34,0	165,5
2.	Doc.MM M. Těšitel	3	229,8	7,0	8,0		7,6	7,4	6,0	36,0	144,3
3.	Dr.MM L. Zůnová	3	183,2	5,0	6,6		0,4	1,0		13,0	120,9
4.	Dr.MM M. Ambros	1	110,7	10,0	2,0		9,9	7,0		28,9	110,7
5.	Mgr.MM B. Vosáhlová	4	94,4								94,4
6.	Doc.MM O. Nevěřil	2	209,4	4,0			7,6	8,2		19,8	87,5
7.	Mgr.MM M. Drexlerová	4	86,1								86,1
8.	Mgr.MM A. Bakoč	3	82,4					37,0		37,0	82,4
9.	Doc.MM A. Žák	4	236,7		3,0			29,9		32,9	82,3
10.	Mgr.MM M. Kadlec	2	84,6	5,5			11,8	11,5		28,8	77,1
11.	Mgr.MM L. Votrubová	3	95,6	6,0	2,0		7,5			15,5	70,9
12.	Doc.MM J. Klementová	2	209,8	2,0				9,8		11,8	70,4
13.	Doc.MM O. Sedláček	3	232,5					5,0		5,0	66,0
14.	Dr.MM R. Novák	4	169,5								65,3
15.	Mgr.MM K. Kučerová	Z8	72,6								64,1
16.	Mgr.MM J. Jedlička	2	61,2	6,0				7,0		13,0	61,2
17.	Mgr.MM M. Stýskala	3	62,5					6,0		6,0	56,5
18.	Bc.MM J. Kadlec	3	47,3	5,5				17,0		22,5	47,3
19.	Dr.MM M. Urbanová	1	108,7	5,5						5,5	44,4
20.	Dr.MM K. Plchová	4	105,6	5,0			4,6			9,6	44,2
21.	Doc.MM V. Tichý	4	353,3								43,0
22.	Dr.MM V. Bartáková	4	102,6	6,0				4,0		10,0	42,3
23.	Mgr.MM A. Stará	3	89,3								41,9
24.	Dr.MM M. Jarvis	2	188,2					8,5		8,5	38,4
25.–26.	Bc.MM O. Hrabě	4	31,5								31,5
	Mgr.MM M. Púll	4	85,1								31,5
27.	Bc.MM A. Bihun	4	29,2								29,2
28.	Bc.MM E. Sabol	4	28,7								28,7
29.	Bc.MM K. Maxera	3	31,5								28,5
30.	Bc.MM A. Trnková	3	27,9		19,0			8,9		27,9	27,9
31.–32.	Bc.MM T. Pazourek	2	27,5								27,5
	Dr.MM D. Kaňka	2	144,6					13,0		13,0	27,5
33.	Bc.MM J. Koška	4	27,3								27,3
34.	Bc.MM K. Česká	3	27,2								27,2
35.	Mgr.MM M. Ulumbekov	2	88,1								26,4
36.	Bc.MM K. Vomelová	4	31,7								24,5

Poř.	Jméno	R.	$\sum_{-1}$	Témata 1	2	3	4	5	O	$\sum_0$	$\sum_1$
37.	Mgr.[MM] M. Rybecký	2	62,4	3,5				4,0		7,5	21,2
38.–39.	Bc.[MM] P. Barták	Z9	20,4								20,4
	Mgr.[MM] J. Löwenhöffer	3	89,9								20,4
40.	Mgr.[MM] M. Jursová	4	51,5								19,8
41.	V. Sklenár	4	18,5								18,5
42.	M. Skýpala	3	17,5								17,5
43.	Mgr.[MM] V. Kučera	2	50,6								17,0
44.	Dr.[MM] O. Popovský	4	104,2								16,5
45.	Bc.[MM] L. Poljaková	4	41,6								14,7
46.	L. Trojan	4	14,0								14,0
47.	A. Weberová	4	11,6								11,6
48.	Dr.[MM] V. Menšíková	2	140,1								11,3
49.	Mgr.[MM] V. Vybíral	2	51,9			1,5				1,5	9,5
50.	Dr.[MM] J. Tregler	4	116,3								9,4
51.	A. Stýskala	4	13,9								8,4
52.	M. Ambrosová	Z8	7,8								7,8
53.	A. Migel	1	7,3								7,3
54.	L. Šírová	2	5,9								5,9
55.–56.	Bc.[MM] V. Verner	3	48,9								4,0
	F. Nouza	2	4,0	4,0						4,0	4,0
57.	R. Zelený	3	3,8								3,8
58.–59.	L. Trochová	1	3,0								3,0
	Bc.[MM] R. Petit	3	37,0								3,0
60.	Bc.[MM] K. Menšíková	1	24,9								1,9
61.	S. Teodorovičová	3	8,1								1,0

Sloupeček $\sum_{-1}$ je součet všech bodů získaných v našem semináři, $\sum_0$ je součet bodů v těchto deadlinech a $\sum_1$ součet všech bodů v tomto ročníku. Sloupec **O** symbolizuje **Ostatní**, obvykle body za články. Tituly uvedené v předchozím textu slouží pouze pro účely M&M.

Výsledková listina 30. ročníku

Poř.	Jméno	R.	$\sum_{-1}$	Číslo 1	2	3	4	5	$\sum_{1}$
1.	Doc.MM J. Uglickich	2	328,3	45,1	24,1	32,3	42,0	22,0	165,5
2.	Doc.MM M. Těšitel	3	229,8	21,5	37,3	29,5	32,6	23,4	144,3
3.	Dr.MM L. Zůnová	3	183,2	24,0	31,0	22,0	38,9	5,0	120,9
4.	Dr.MM M. Ambros	1	110,7	25,7	25,1	17,0	27,9	15,0	110,7
5.	Mgr.MM B. Vosáhlová	4	94,4	32,6	36,1	20,0	5,7		94,4
6.	Doc.MM O. Nevěřil	2	209,4	18,9	24,8	8,0	23,6	12,2	87,5
7.	Mgr.MM M. Drexlerová	4	86,1	22,9	37,5	19,0	6,7		86,1
8.	Mgr.MM A. Bakoč	3	82,4	14,4	16,0	15,0	28,5	8,5	82,4
9.	Doc.MM A. Žák	4	236,7	21,5	24,4	3,5	24,5	8,4	82,3
10.	Mgr.MM M. Kadlec	2	84,6	13,5	34,8		23,3	5,5	77,1
11.	Mgr.MM L. Votrubová	3	95,6	25,5	24,1	5,8	7,5	8,0	70,9
12.	Doc.MM J. Klementová	2	209,8	14,2	30,4	3,5	17,5	4,8	70,4
13.	Doc.MM O. Sedláček	3	232,5	22,2	16,5	15,5	6,8	5,0	66,0
14.	Dr.MM R. Novák	4	169,5	18,1	18,7	9,0	19,5		65,3
15.	Mgr.MM K. Kučerová	Z8	72,6	18,0	17,8	3,5	24,8		64,1
16.	Mgr.MM J. Jedlička	2	61,2			21,0	30,2	10,0	61,2
17.	Mgr.MM M. Stýskala	3	62,5	19,5	8,5	8,5	20,0		56,5
18.	Bc.MM J. Kadlec	3	47,3	12,5	8,5	3,8	17,0	5,5	47,3
19.	Dr.MM M. Urbanová	1	108,7	23,7	5,7	4,5	5,0	5,5	44,4
20.	Dr.MM K. Plchová	4	105,6	8,7	9,0	8,0	13,5	5,0	44,2
21.	Doc.MM V. Tichý	4	353,3	19,5			23,5		43,0
22.	Dr.MM V. Bartáková	4	102,6	13,1	7,0	5,0	7,2	10,0	42,3
23.	Mgr.MM A. Stará	3	89,3	11,8	11,5	0,3	18,3		41,9
24.	Dr.MM M. Jarvis	2	188,2	20,9	9,0			8,5	38,4
25.–26.	Bc.MM O. Hrabě	4	31,5	17,5			14,0		31,5
	Mgr.MM M. Púll	4	85,1	31,5					31,5
27.	Bc.MM A. Bihun	4	29,2	29,2					29,2
28.	Bc.MM E. Sabol	4	28,7	28,7					28,7
29.	Bc.MM K. Maxera	3	31,5	20,0	8,0	0,5			28,5
30.	Bc.MM A. Trnková	3	27,9					27,9	27,9
31.–32.	Bc.MM T. Pazourek	2	27,5	14,7	12,8				27,5
	Dr.MM D. Kaňka	2	144,6	5,5	9,0		13,0		27,5
33.	Bc.MM J. Koška	4	27,3	27,3					27,3

Poř.	Jméno	R.	$\sum_{-1}$	Číslo 1	2	3	4	5	$\sum_{1}$
34.	Bc.[MM] K. Česká	3	27,2	9,2	5,0	1,5	11,5		27,2
35.	Mgr.[MM] M. Ulumbekov	2	88,1	8,7	17,7				26,4
36.	Bc.[MM] K. Vomelová	4	31,7	24,5					24,5
37.	Mgr.[MM] M. Rybecký	2	62,4	1,7	7,7	4,3	4,0	3,5	21,2
38.–39.	Bc.[MM] P. Barták	Z9	20,4	9,9			10,5		20,4
	Mgr.[MM] J. Löwenhöffer	3	89,9	13,4	7,0				20,4
40.	Mgr.[MM] M. Jursová	4	51,5			12,1	7,7		19,8
41.	V. Sklenár	4	18,5	12,5	6,0				18,5
42.	M. Skýpala	3	17,5	17,5					17,5
43.	Mgr.[MM] V. Kučera	2	50,6	6,2	10,8				17,0
44.	Dr.[MM] O. Popovský	4	104,2	11,5	5,0				16,5
45.	Bc.[MM] L. Poljaková	4	41,6	14,7					14,7
46.	L. Trojan	4	14,0				14,0		14,0
47.	A. Weberová	4	11,6	11,6					11,6
48.	Dr.[MM] V. Menšíková	2	140,1	11,3					11,3
49.	Mgr.[MM] V. Vybíral	2	51,9	6,3	0,5	1,2	1,5		9,5
50.	Dr.[MM] J. Tregler	4	116,3	9,4					9,4
51.	A. Stýskala	4	13,9	8,4					8,4
52.	M. Ambrosová	Z8	7,8		7,8				7,8
53.	A. Migel	1	7,3	7,3					7,3
54.	L. Šírová	2	5,9			5,9			5,9
55.–56.	Bc.[MM] V. Verner	3	48,9	4,0					4,0
	F. Nouza	2	4,0					4,0	4,0
57.	R. Zelený	3	3,8	3,8					3,8
58.–59.	L. Trochová	1	3,0	3,0					3,0
	Bc.[MM] R. Petit	3	37,0	3,0					3,0
60.	Bc.[MM] K. Menšíková	1	24,9	1,9					1,9
61.	S. Teodorovičová	3	8,1	1,0					1,0

Časopis M&M je zastřešen Matematicko-fyzikální fakultou Univerzity Karlovy. S obsahem časopisu je možné nakládat dle licence CC BY 4.0. Autory textů jsou, není-li uvedeno jinak, organizátoři M&M. Realizace projektu byla podpořena Ministerstvem školství, mládeže a tělovýchovy. Pokud si časopis nepřejete dále dostávat v tištěné podobě, zrušte si prosím jeho odběr v nastavení svého účtu na webu.

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

matfyz
M&M