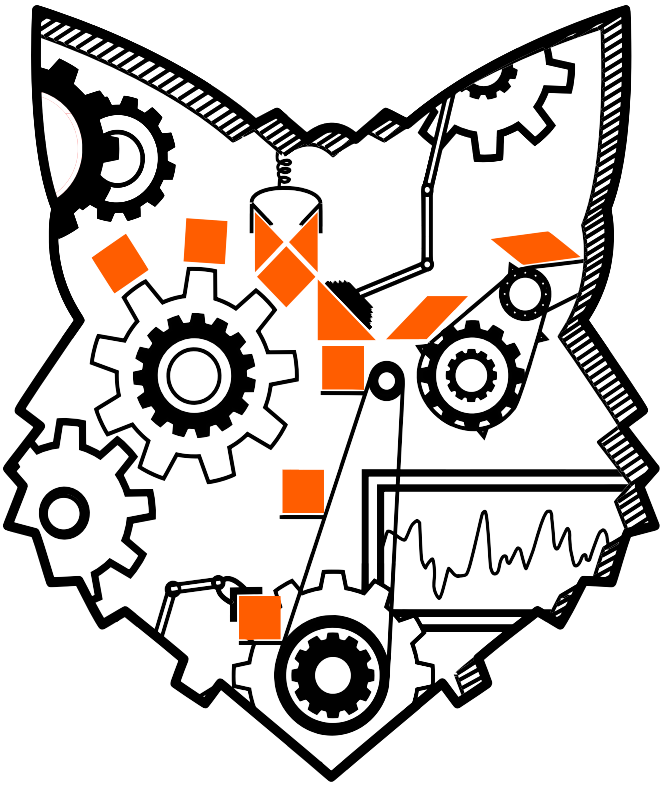
STUDENTSKÝ ČASOPIS A KORESPONDENČNÍ SEMINÁŘ

Ročník XXX
Číslo 5

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.

Milý řešiteli,

vítej v dalším čísle! Konec roku se pomalu blíží, v M&Mku se ale ještě děje spousta věcí. Již 20. dubna začíná jarní soustředění – pokud jsme Tě pozvali, dej nám vědět, zda dorazíš, a pokud ne, nezoufej a něco vyřeš, ať s námi můžeš jet třeba na podzim.

Soustředění ale není naše jediná akce. Ať už jsi ve výsledkovce vysoko, nebo úplně dole, můžeš s námi ještě před prázdninami vyrazit na víkendovku, která se koná 24.–26. května. Půjdeme na výlet, budeme jíst, hrát deskovky a na kytaru a všeobecně se veselit. Přihlašování najdeš již brzy ve své e-mailové schránce!

A teď již k obsahu samotného čísla. V tématku *Termodynamika* najdeš tentokrát kraťoučký (ale o to zajímavější) text o entropii a k němu zbrusu nové problémy a úlohy. Tématko *Genetika* bohužel již končí, můžeš si ale alespoň projít otištěná vzorová řešení. Spoustu nového obsahu najdeš v tématku *FlatFox#*, kde se kromě vzorových řešení dozvíš něco o časové složitosti, lišák Riki tak bude moci skákat mezi bludnými kořeny efektivněji než kdy dříve. No a *Lean* samozřejmě nezůstane pozadu; ukážeme si v něm, jak ve formálních důkazech používat matematickou indukci.

Přejeme Ti hodně zápalu do řešení a na viděnou na soustředění či víkendovce!

Tvoji organizátoři

Obsah

Zadání a řešení témat

1. deadline: 30. dubna 2024 | 2. deadline: 28. května 2024

Téma 1 – Termodynamika

Entropie

Entropie je pojem, který se často vyskytuje v popularizační vědě. Tato fyzikální veličina je nějakým způsobem spojena s chaosem a neuspořádaností systému. Ne vždy je ale vysvětlena úplně pěkně a do detailu. My se na ni podíváme z pohledu statistické termodynamiky známé také jako statistická mechanika. Tento vědní obor se snaží veličiny popisující jednotlivé částice systému převést na veličiny, které popisují systém jako celek pomocí jedné hodnoty. Například z rychlosti každé částice a z její pozice v systému (např. v nádobě) lze vypočítat jeho tlak, teplotu a mimo jiné i entropii. My se zaměříme právě na to, jak statistická termodynamika popisuje a vypočítá entropii nějakého systému a co to o tom systému vypovídá.

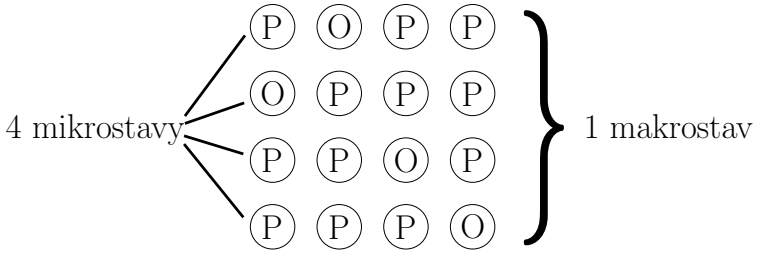
Panna nebo orel? Excitace či základní stav?

Pro začátek použijeme analogii hodů mincemi. Abychom měli alespoň nějak velký systém, zvolíme si tyto mince čtyři. Počet mincí symbolizuje počet částic v systému, který je konstantní. Další předpoklady v tomto systému jsou 2 možné stavy – panna nebo orel pro každou minci. Kolika způsoby dopadne hod 4 mincemi? Za předpokladu, že jsou mince stejné, dojdeme k $2^4 = 16$ možnostem.

Jedna z možných interpretací tohoto popisu mincemi ve fyzice a chemii je dělení částice mezi jednu základní energetickou hladinu a hladinu pro excitovaný stav. Pro každou částici tedy existují 2 energetické hladiny, ve kterých se může vyskytovat. Jde o popis systému se 4 částicemi, které se snažíme rozdělit na 2 energetické hladiny například ozářením elektromagnetickým zářením vhodné vlnové délky.

Kolika způsoby se tedy částice mohou uspořádat do systému se 2 hladinami energií? Za předpokladu, že částice jsou stejné a jsou to bosony[1], lze říct, že můžeme pozorovat právě oněch 16 možností umístění. Těmto 16 možnostem se říká mikrostavy. Za předpokladu, že mince jsou stejné (částice jsou nerozlišitelné), kombinace panna–panna–orel–panna, orel–panna–panna–panna, panna–panna–panna–orel a panna–orel–panna–panna jsou stejné (viz obrázek 1). Počtu těchto realizací odpovídá počet mikrostavů, které se navenek jeví stejně (padne 1 orel). Jinými slovy, každý tento mikrostav mající stejný výsledek (jeden hozený orel) je pro pozorovatele ve výsledku stejný. Dohromady tyto „stejné“ mikrostavy dávají makrostav, který je degenerovaný. Náš makrostav hodu mincí je popsán spojením „hodíme 1 orla, a je jedno, na které minci padne“.

[1] Bosony jsou částice, které se mohou nacházet v nějakém vybraném stavu ve větším počtu než jeden. Příkladem je foton nebo gluon. Toto mimo jiné neplatí pro elektrony a protony, ty patří mezi fermiony. Více viz: `https://www.energy.gov/science/doe-explainsbosons-and-fermions`



Obrázek 1: Analogie mikrostavů a makrostavů systému pro hod 4 dvoustěnnými mincemi. P – panna, O – orel.

Úloha 5.1 [2b]: *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.*

Všimněte si, že doteď nebylo řečeno, zda vyšší energetické hladině částic v naší mincové analogii odpovídá panna nebo orel. Je to proto, že bez ohledu na přiřazení excitované hladiny panně nebo orlovi bude celý systém 4 mincí popsán stejným počtem mikrostavů a budou symetricky rozdělené.

Úloha 5.2 [1b]: *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.*

Po spočítání počtu mikrostavů pro všech 5 makrostavů (nula až všechny čtyři excitované částice) a jejich sečtení dostaneme oněch $2^4 = 16$ možností mikrostavů.

Problém 5.3: *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 $k \leq N$.*

Pokud bychom z počtu mikrostavů chtěli počítat entropii S, lze použít vztah $S = -k \ln(w)$, kde w je počet všech mikrostavů ve všech makrostavech a $k = 1{,}380649 \cdot 10^{-23}\,\mathrm{J} \cdot \mathrm{K}^{-1}$ je Boltzmannova konstanta.

Ponožky v šuplíku a plyn v nádobě na závěr

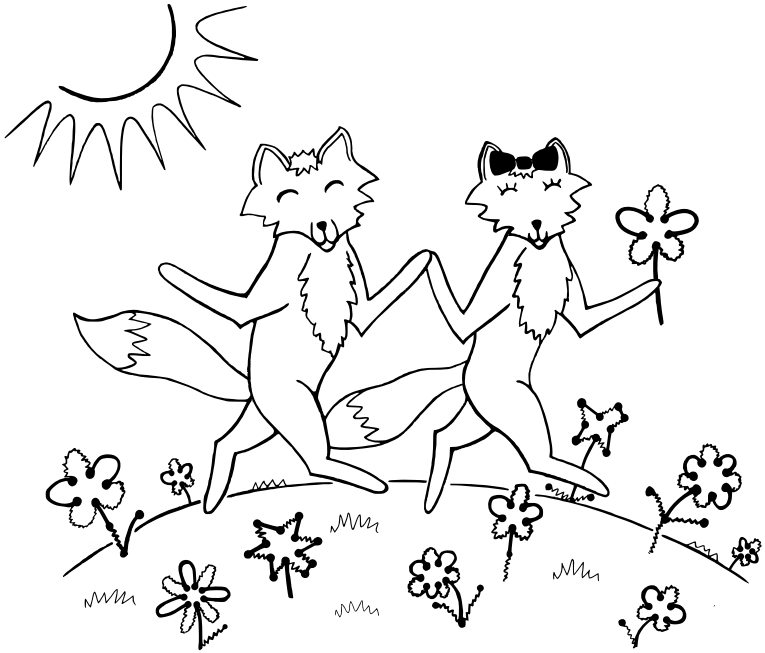
Z tohoto textu je tedy poznat, že vysvětlení entropie jako míra nepořádku ve vašem pokoji není dostatečná. Entropie pokoje jako systému je vysoká právě tehdy, když v něm existuje spousta možností, kterými můžete skladovat vaše věci (počet

mikrostavů). Pokud tedy máte spoustu možností, kam zaházet svoje oblečení, entropie systému (vašeho pokoje) je vysoká. Ponožky se můžou nacházet v šuplíku, na stole, pod postelí, na lustru, na parapetu, pod polštářem... Pokud jste ale pořádkumilovní a nepřipouštíte jiné místo pro ponožky než je jejich místo v šuplíku, tak nemáte mnoho možností realizovat ponožkové mikrostavy a entropie pokoje bude malá.

Úloha 5.4 [2b]: *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.*

Závěrem chceme dodat, že změna entropie mezi dvěma stavy systému je kritérium, které rozhoduje o směru děje. Přidám místa na zahození ponožek a najednou se nepořádek v pokoji zvětší, při samovolném rozpínání plynu mají částice více možností, kde se vyskytovat, při smíchávání dvou látek tyto látky nezůstávají odděleny, ale mísí se, proteiny se sbalují do trojrozměrných struktur...

Pája, Helča a Pavel; `termodynamika@ledoian.cz`
odevzdávejte do odevzdávátka



Téma 2 – Programování a dokazování v Leanu

Úloha 5.1 [3b]: *Theorem z Úlohy 4.10, kterému jsme věnovali velkou část minulého čísla, je známá matematická věta. Jak se tato věta obvykle nazývá? Napište jeden až dva odstavce o její historii. Zjistěte, kde bychom tuto větu (či nějakou její blízkou obdobu) našli v knihovně Mathlib (vložte odkaz na konkrétní řádek kódu). Tuto úlohu doporučujeme řešit pomocí ChatGPT* [2] *nebo podobného nástroje (klidně můžete celou odpověď vygenerovat použitím umělé inteligence).*

Díl 5: Matematická indukce

Občas je příliš těžké dokázat matematické tvrzení o přirozeném číslu, které je univerzálně kvantifikováno, přímo. Matematická indukce nám umožní nahradit tento složitý důkaz (či dokonce nemožný důkaz) dvěma jednoduššími důkazy. Nejprve dokážeme, že dané tvrzení platí pro nulu. Následně dokážeme, že za předpokladu, že dané tvrzení platí pro o jedna menší číslo, platí i pro naše číslo.

O matematické indukci jste se mohli dozvědět v poznámce redakce ke konci 3. dílu tématka.[3] Nyní se ji naučíte používat v Leanu.

Pokud máme v kontextu `(n : ℕ)` a jsme v tactic módu, můžeme dělat indukci na `n` takto (pozor na to, že následující syntax je potřeba opsat dost přesně):

```
induction n with
| zero => důkaz cíle, kde je za n dosazeno 0
| succ m ih => důkaz cíle, kde je za n dosazeno m + 1
```

Pokud se `n` nachází i jinde než v cíli, jsou patřičné hodnoty za `n` dosazeny na všech místech. Ve větvi `succ m ih` dostáváme (a to je ten důvod, proč indukci používáme) tvrzení `ih` v kontextu (tedy, je to lokální předpoklad), které říká to samé, co je náš cíl, ale je v něm za `n` dosazeno `m` (tzv. indukční předpoklad[4]).

Podobně se píše indukce pro seznamy. Pokud máme v kontextu `(x : List ℕ)` a jsme v tactic módu, můžeme dělat indukci na `x` takto (opět je potřeba syntax opsat dost přesně):

```
induction x with
| nil => důkaz cíle, kde je za x dosazeno []
| cons d l ih => důkaz cíle, kde je za x dosazeno d :: l
```

Stejně se píše indukce i pro seznamy jiného typu. Term `d` bude v takovém případě jiného typu než přirozené číslo.

[2] `https://chat.openai.com/`

[3] `https://mam.mff.cuni.cz/media/cislo/pdf/30/30-3.pdf`

[4] anglicky „induction hypothesis“ (pozor na to, že v matice má slovo „hypothesis“ jiný význam než ve vědě)

Připomeňme si pár definic z druhého dílu (ta notace není úplně stejná jako teď, když používáme knihovnu Mathlib, ale můžeme je bez problému používat):

```
def prvnich_n_lichych_sestupne : Nat → List Nat
| 0   => []
| n+1 => (2 * n + 1) :: (prvnich_n_lichych_sestupne n)

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

def prvnich_n_lichych : Nat → List Nat :=
obrat ∘ prvnich_n_lichych_sestupne

def soucet : List Nat → Nat
| [ ]           => 0
| hlava :: telo => hlava + soucet telo
```

Nyní bychom chtěli dokázat, že součet prvních n lichých čísel je roven n^2. Zadání zní takhle:

```
theorem soucet_prvnich_n_lichych (n : ℕ) :
    soucet (prvnich_n_lichych n) = n * n := by
  sorry
```

Obvykle se stává, že matematickou indukci nelze nebo není vhodné použít okamžitě. Většinou potřebujeme vyslovit vhodné lemma, které dokážeme indukcí. To lemma pak použijeme k důkazu kýžené věty. Zde tím hlavní lemmatem je:

```
lemma soucet_prvnich_n_lichych_sestupne (n : ℕ) :
    soucet (prvnich_n_lichych_sestupne n) = n * n := by
  sorry
```

Důkaz začíná takhle:

```
lemma soucet_prvnich_n_lichych_sestupne (n : ℕ) :
    soucet (prvnich_n_lichych_sestupne n) = n * n := by
  induction n with
  | zero => decide
  | succ m ih => sorry
```

Taktika `decide` říká: „Počítači, vyčísli to!“ Používáme ji k důkazům cílů, které se skládají pouze z konstant. Když se teď podíváme na cíl v indukčním kroku, zarmoutí nás, že levá ani pravá strana `ih` nic nematchuje. Proto ho převedeme na vhodnější tvar:

```
    convert_to 2 * m + 1 + soucet (prvnich_n_lichych_sestupne m) =
               m * m + 2 * m + 1
```

Nyní si dokonce můžeme vybrat, jakým směrem `ih` použijeme. Tímto řádkem provedeme přepis zleva doprava:

```
    rw [ih]
```

Zbytek důkazu nám najde `library_search`. Celý důkaz lemmatu je takhle:

```
lemma soucet_prvnich_n_lichych_sestupne (n : ℕ) :
    soucet (prvnich_n_lichych_sestupne n) = n * n := by
  induction n with
  | zero => decide
  | succ m ih =>
    convert_to 2 * m + 1 + soucet (prvnich_n_lichych_sestupne m) =
               m * m + 2 * m + 1
    • convert_to (m + 1) * (m + 1) = m * m + 2 * m + 1
      ring
    rw [ih]
    exact add_comm (2 * m + 1) (m * m)
```

Dále se hodí toto lemma:

```
lemma obrat_zachovava_soucet (x : List ℕ) :
    soucet (obrat x) = soucet x := by
  sorry
```

Na jeho důkaz se podívejte do repozitáře. Nyní dokážeme větu:

```
theorem soucet_prvnich_n_lichych (n : ℕ) :
    soucet (prvnich_n_lichych n) = n * n := by
  dsimp [prvnich_n_lichych]
  rw [obrat_zachovava_soucet]
  apply soucet_prvnich_n_lichych_sestupne
```

Hotovo! Pro větší stručnost jsme mohli `soucet_prvnich_n_lichych_sestupne` použít uvnitř toho `rw` řádku.

Další příklad, který najdete v repozitáři, je následující nerovnost (podstatně méně zajímavá než ostatní příklady, které vám tu ukazujeme – každý matematik přeci ví, že exponenciální funkce roste rychleji než kvadratická, a dosazováním hodnot se lze snadno přesvědčit, že toto tvrzení začíná platit hned za čtyřkou):

```
theorem dva_na_vs_na_druhou (n : ℕ) (aspon_pet : n ≥ 5) :
    2^n > n^2 := by
  sorry
```

Abychom nemuseli důkaz začínat únavným rozborem případů, vyslovíme lemma, které snadno dokážeme indukcí:

```
lemma dva_na_vs_na_druhou_aux (n : ℕ) :
    2 ^ (n+5) > (n+5) ^ 2 := by
  induction n with
  | zero => decide
  | succ m ih =>
    convert_to 2 ^ (m+5) * 2 > (Nat.succ (m+5)) ^ 2
    have : (m+5) * (m+5) > 3 * (m+5)
    • nlinarith
    linarith
```

Teď už si akorát musíme dát pozor na to, že odčítání přirozených čísel nikdy nejde do záporu. Převod věty na dokázané lemma provedeme takto:

```
theorem dva_na_vs_na_druhou (n : ℕ) (aspon_pet : n ≥ 5) :
    2^n > n^2 := by
  have minus5_plus5 : n = (n - 5) + 5
  • exact Nat.eq_add_of_sub_eq aspon_pet rfl
  rw [minus5_plus5]
  exact dva_na_vs_na_druhou_aux (n - 5)
```

Tadáá!

Na závěr si ukažme jeden složitější příklad. Dokážeme si Binetův vzorec[5]. Ten vyjadřuje n-té Fibonacciho číslo (v následujícím vzorci označené jako F_n) v uzavřeném tvaru takto:

$$F_n = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

Nejprve si připomeňme, jak jsme v prvním díle zadefinovali Fibonacciho čísla:

```
def fibonacci : Nat → Nat
| 0   => 0
| 1   => 1
| n+2 => fibonacci n + fibonacci (n+1)
```

Nyní formálně vyslovíme větu, kterou chceme dokázat:

```
theorem binetuv_vzorec (n : ℕ) :
  fibonacci n = (1 / Real.sqrt 5) * (
      ((1 + Real.sqrt 5) / 2) ^ n -
      ((1 - Real.sqrt 5) / 2) ^ n )
```

Binetův vzorec je opravdu zajímavý. Fibonacciho posloupnost je vyjádřena jako součet dvou exponenciálních posloupností; kvocientem první posloupnosti je zlatý řez; kvocientem druhé posloupnosti je opačná převrácená hodnota zlatého řezu.

[5] Jacques Philippe Marie Binet nebyl první, kdo vzorec nalezl. Znal ho už Abraham de Moivre.

Proč zrovna tato dvě čísla? Ono totiž neexistuje mnoho exponenciálních posloupností, které splňují rekurentní vztah $a_{n+2} = a_n + a_{n+1}$. Existují jen dvě hodnoty q splňující $q^{n+2} = q^n + q^{n+1}$. K důkazu se nám bude hodit následující lemma:

```
lemma fibonacci_aux {x : ℝ} (predpoklad : x * x = x + 1) (m : ℕ) :
    x ^ (m+1) = x * fibonacci (m+1) + fibonacci m := sorry
```

Lemma dokážeme matematickou indukcí dle `m`. Bázický krok necháme dokázat automaticky (avšak `decide` zde nestačí, protože hodnoty závisí na `x`):

```
lemma fibonacci_aux {x : ℝ} (predpoklad : x * x = x + 1) (m : ℕ) :
    x ^ (m+1) = x * fibonacci (m+1) + fibonacci m := by
  induction m with
  | zero => simp [fibonacci]
  | succ n ih => sorry
```

Indukční krok má následující kontext:

```
x : ℝ
predpoklad : x * x = x + 1
n : ℕ
ih : x ^ (n + 1) = x * ↑(fibonacci (n + 1)) + ↑(fibonacci n)
⊢ x ^ (Nat.succ n + 1) =
  x * ↑(fibonacci (Nat.succ n + 1)) + ↑(fibonacci (Nat.succ n))
```

Šipka nahoru označuje, že se zde přirozené číslo používá jako reálné číslo. Ty šipky by správně měly být i ve znění věty a lemmatu, ale Lean si to domyslel sám. Nyní převedeme cíl do tvaru, ve kterém se nám bude dobře aplikovat indukční předpoklad:

```
  convert_to x * x ^ (n+1) =
             x * (fibonacci n + fibonacci (n+1)) + fibonacci (n+1)
```

Všimněte si, že jsme expandovali výraz `fibonacci (n+2)` z daného cíle na výraz `fibonacci n + fibonacci (n+1)`. Následně na levé straně rovnosti dosadíme indukční předpoklad do pravého činitele:

```
    rw [ih]
```

Následkem je tento cíl:

```
⊢ x * (x * ↑(fibonacci (n+1)) + ↑(fibonacci n)) =
  x * (↑(fibonacci n) + ↑(fibonacci (n+1))) + ↑(fibonacci (n+1))
```

Jak dál? Všimněte si, že jsme doposud nevyužili `predpoklad`. Připravme tedy cíl na jeho aplikaci. Taktika `convert_to` nám opět dobře poslouží. Zbytek je jen několik algebraických manipulací, které nám `ring` dobře automatizuje. Celý důkaz je takhle:

```
lemma fibonacci_aux {x : ℝ} (predpoklad : x * x = x + 1) (m : ℕ) :
    x ^ (m+1) = x * fibonacci (m+1) + fibonacci m := by
  induction m with
  | zero => simp [fibonacci]
  | succ n ih =>
    convert_to x * x ^ (n+1) =
             x * (fibonacci n + fibonacci (n+1)) + fibonacci (n+1)
    • simp [fibonacci]
    rw [ih]
    convert_to (x * x) * fibonacci (n+1) + x * fibonacci n =
             x * (fibonacci n + fibonacci (n+1)) + fibonacci (n+1)
    • ring
    rw [predpoklad]
    ring
```

Důkaz hlavní věty je přímočarý, ale pracný. Můžete se na něj podívat do repozitáře.[6]

Úloha 5.2 [12b]: *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:

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

Nápověda: Najděte vhodné lemma, které nepoužívá odčítání, dělení, ani funkci `prvnich_n_krychli`.

Úloha 5.3 [4b]: *Dokažte (s využitím čehokoliv, co jsme v tématku probrali):*

```
theorem obrat_obrat {T : Type} (x : List T) :
    obrat (obrat x) = x := by
  sorry
```

Nápověda: Taktika `simp` toho dokáže udělat fakt hodně. Do hranatých závorek ji můžete dát definice, lemmata i lokální tvrzení, která má používat.

[6] https://github.com/madvorak/lean-mam/blob/main/mam/Cislo5.lean

Řešení úloh ze 4. dílu

Zadání:

Důkaz následujícího tvrzení napište pouze pomocí základních taktik (bez automatizace).

Řešení:

```
example (P Q R : Prop) : P ∧ (Q ∨ R) ↔ (P ∧ Q) ∨ (P ∧ R) := by
  constructor
  • intro predpoklad
    obtain ⟨hp, hqr⟩ := predpoklad
    cases hqr with
    | inl hq =>
      left
      constructor
      • exact hp
      • exact hq
    | inr hr =>
      right
      constructor
      • exact hp
      • exact hr
  • intro predpoklad
    cases predpoklad with
    | inl hpq =>
      obtain ⟨hp, hq⟩ := hpq
      constructor
      • exact hp
      • left
        exact hq
    | inr hpr =>
      obtain ⟨hp, hr⟩ := hpr
      constructor
      • exact hp
      • right
        exact hr
```

Celý tenhle důkaz by bylo možné nahradit zavoláním taktiky `tauto` nebo sáhnutím do standardní knihovny, ale to nebylo podstatou tohoto cvičení. Smyslem cvičení bylo naučit se důkazovou techniku, která bude potřebná na složitější důkazy, kde automatizace nepomůže (respektive pomůže až v „listech důkazu“; tu základní strukturu musí napsat člověk).

Zadání:
Důkaz následujícího tvrzení napište pouze pomocí základních taktik (bez automatizace).

Řešení:
Pokud půjdeme vyloženě „za nosem“, napíšeme důkaz nějak takhle:

```
example (R : ℕ → ℕ → ℕ → ℕ → ℕ → Prop) :
    (∃ x, ∀ y, ∀ z, ∀ b, ∃ a, R a b x y z) →
    (∀ z, ∀ y, ∃ x, ∀ w, ∃ v, R v w x y z) := by
  intro predpoklad
  obtain ⟨x, hx⟩ := predpoklad
  intro z
  intro y
  use x
  intro w
  obtain ⟨b, hb⟩ := hx y z w
  use b
  exact hb
```

Všimněte si, že `b` dělá to samé jako `w` a že `a` dělá to samé jako `v`, takže s těmito čtyřmi kvantifikátory není potřeba nijak manipulovat. Když to vezmeme v potaz, vypadá důkaz takhle:

```
example (R : ℕ → ℕ → ℕ → ℕ → ℕ → Prop) :
    (∃ x, ∀ y, ∀ z, ∀ b, ∃ a, R a b x y z) →
    (∀ z, ∀ y, ∃ x, ∀ w, ∃ v, R v w x y z) := by
  intro predpoklad
  obtain ⟨x, hx⟩ := predpoklad
  intro z
  intro y
  use x
  exact hx y z
```

Stejný důkaz lze napsat zkratkovitě:

```
example (R : ℕ → ℕ → ℕ → ℕ → ℕ → Prop) :
    (∃ x, ∀ y, ∀ z, ∀ b, ∃ a, R a b x y z) →
    (∀ z, ∀ y, ∃ x, ∀ w, ∃ v, R v w x y z) := by
  intro ⟨x, hx⟩ z y
  exact ⟨x, hx y z⟩
```

Dokonce lze úplně to samé napsat jako term (tj. bez použití `by`):

```
example (R : ℕ → ℕ → ℕ → ℕ → ℕ → Prop) :
    (∃ x, ∀ y, ∀ z, ∀ b, ∃ a, R a b x y z) →
    (∀ z, ∀ y, ∃ x, ∀ w, ∃ v, R v w x y z) :=
  fun ⟨x, hx⟩ z y => ⟨x, hx y z⟩
```

Zadání:
Dokažte, že složení surjektivních funkcí dá surjektivní funkci.

Řešení:

```
theorem slozSurjektivni {A B C : Type} {f : A → B} {g : B → C}
    (hf : Surjektivni f) (hg : Surjektivni g) :
    Surjektivni (g ∘ f) := by
  intro z
  obtain ⟨y, hy⟩ := hg z
  obtain ⟨x, hx⟩ := hf y
  use x
  rw [←hy, ←hx]
  rfl
```

Důkaz lze řádek-po-řádku přeříkat slovy:

1. Hodnotu zadanou nepřítelem pojmenujeme `z`. Pro ni musíme najít nějaký `(g ∘ f)`-vzor.
2. Hodnotu `z` zadáme do předpokladu `hg` a dostaneme vzor `y` spolu s důkazem, že `g` zobrazí `y` na `z`.
3. Hodnotu `y` zadáme do předpokladu `hf` a dostaneme vzor `x` spolu s důkazem, že `f` zobrazí `x` na `y`.
4. Jako svědka použijeme `x`.
5. Teď využijeme, že `f` zobrazí `x` na `y` a že `g` zobrazí `y` na `z`.
6. Zbytek je reflexivita (to znamená, že se strany rovnají přímo z definice).[7]

Zadání:
Dokažte, že složení bijektivních funkcí dá bijektivní funkci.

Řešení:

```
theorem slozBijektivni {A B C : Type} {f : A → B} {g : B → C}
    (hf : Bijektivni f) (hg : Bijektivni g) :
    Bijektivni (g ∘ f) := by
  obtain ⟨prosta_f, surjektivni_f⟩ := hf
  obtain ⟨prosta_g, surjektivni_g⟩ := hg
  constructor
  • exact slozProsta prosta_f prosta_g
  • exact slozSurjektivni surjektivni_f surjektivni_g
```

[7] Tenhle krok většinou udělá Lean sám od sebe, jakmile dostane příležitost. Na čem závisí, že se to někdy (třeba tady) nestane automaticky a Lean je potřeba pošťouchnout, přesahuje rámec tohoto textu.

Toto by byl zcela elementární důkaz. Ovšem díky tomu, že v těchto úlohách jste již měli povoleno využívat automatické taktiky, jste mohli napsat důkaz takto krátce:

```
theorem slozBijektivni {A B C : Type} {f : A → B} {g : B → C}
    (hf : Bijektivni f) (hg : Bijektivni g) :
    Bijektivni (g ∘ f) := by
  simp_all [Bijektivni, slozSurjektivni, slozProsta]
```

Zadání:
Dokažte, že každý prvek s lichým počtem předchůdců má nějakého rodiče a ten má sudý počet předchůdců.

Řešení:

```
lemma LichaGenerace.existuje_rodic {f : A → B} {g : B → A}
    {a : A} (lichaGen : LichaGenerace f g a) :
    ∃ p : B, g p = a ∧ SudaGenerace g f p := by
  have ⟨n, hn⟩ := lichaGen
  cases hn with
  | nasl p hp hpg =>
    use p
    constructor
    • exact hp
    • use n
      exact hpg
```

Zadání:
Dokažte, že pokud prvek má lichý počet předchůdců, jeho potomek má sudý počet předchůdců.

Řešení:

```
lemma LichaGenerace.pristiSudaGenerace {f : A → B} {g : B → A}
    {a : A} (lichaGen : LichaGenerace f g a) :
    SudaGenerace g f (f a) := by
  have ⟨n, hn⟩ := lichaGen
  use n+1
  right
  • rfl
  convert hn
```

Zadání:
Dokažte, že pokud prvek má sudý počet předchůdců, jeho potomek má lichý počet předchůdců.

Řešení:

```
lemma SudaGenerace.pristiLichaGenerace {f : A → B} {g : B → A}
    {a : A} (sudaGen : SudaGenerace f g a) :
    LichaGenerace g f (f a) := by
  have ⟨n, hn⟩ := sudaGen
  use n
  right
  • rfl
  exact hn
```

Zadání:
Dokažte, že pokud nesirotek má sudý počet předchůdců, jeho rodič má lichý počet předchůdců.

Řešení:

```
lemma SudaGenerace.predchoziLichaGenerace {f : A → B} {g : B → A}
    {a : A} (sudaGen : SudaGenerace g f (f a)) (hf : Prosta f) :
    LichaGenerace f g a := by
  have ⟨n, hn⟩ := sudaGen
  cases n with
  | zero =>
    cases hn with
    | nula sirotek =>
      exfalso
      apply sirotek
      use a
  | succ k =>
    use k
    cases hn with
    | nasl p hpa hp =>
      have p_je_a : p = a
      • by_contra kdyby
        exact hf p a kdyby hpa
      rw [p_je_a] at hp
      exact hp
```

Zadání:
Dokažte, že funkce g^{-1} *se tam, kde je definována, chová jako prostá funkce.*

Řešení:

```
lemma inverze_jakoProsta {f : A → B} {g : B → A}
    {x y : A} (hxy : x ≠ y)
    (hx : LichaGenerace f g x) (hy : LichaGenerace f g y) :
    inverze hx ≠ inverze hy := by
  intro rovnost
  have potom : g (inverze hx) = g (inverze hy)
  • exact congr_arg g rovnost
  rw [hx.g_inverze, hy.g_inverze] at potom
  exact hxy potom
```

Zadání:
Dokončete důkaz věty.

Řešení:

```
theorem jsouStejneVelke :
    (∃ f : A → B, Prosta f) ∧ (∃ g : B → A, Prosta g) →
      StejneVelke A B := by
  intro ⟨⟨f, hf⟩, ⟨g, hg⟩⟩
  classical
  let F : A → B := fun a =>
      if ha : LichaGenerace f g a then inverze ha else f a
  use F
  constructor
```

V tuto chvíli se otevřely dva cíle:

```
⊢ Prosta F
⊢ Surjektivni F
```

Pojďme se podívat na první cíl. Jak už jsme to dělali dříve, použijeme:

- `intro x y hxy`

Cíl se změnil na:

```
⊢ F x ≠ F y
```

Následuje rozbor případů podle toho, zda je `x` v liché generaci a zda je `y` v liché generaci. Tohle je kostra důkazu:

```
    if hx : LichaGenerace f g x
    then
      if hy : LichaGenerace f g y
      then
        sorry
      else
        sorry
    else
      if hy : LichaGenerace f g y
      then
        sorry
      else
        sorry
```

Uvedeme si tu pouze důkaz posledního případu (na zbytek se podívejte do repozitáře). Využijeme teď toho, že se funkce `F` chová v tomto případě stejně jako funkce `f` (a o ní máme předpoklad, že je prostá):

```
convert_to f x ≠ f y
• exact dif_neg hx
• exact dif_neg hy
apply hf
exact hxy
```

Jak už tomu bývá zvykem, řádky začínající puntíkem následované `exact` důkazem jsme si nechali vygenerovat pomocí `library_search`.

A teď surjektivita:

- `intro z`

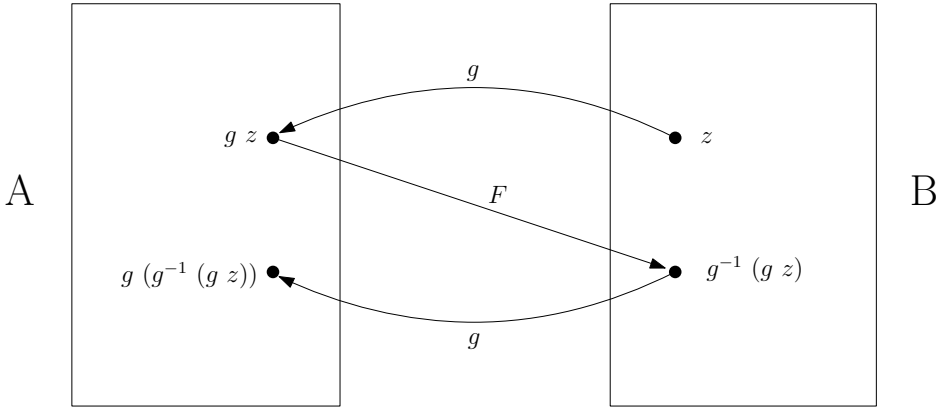
Je potřeba dokázat:

```
⊢ ∃ x : A, F x = z
```

Uvědomíme si, že pro `z` sudé generace je `g z` liché generace, a tudíž je v tomto případě `g z` vzorem `z`. Proto náš důkaz pokračuje takto:

```
if hz : SudaGenerace g f z
then
  use g z
  have hgz : LichaGenerace f g (g z)
  • exact hz.pristiLichaGenerace
```

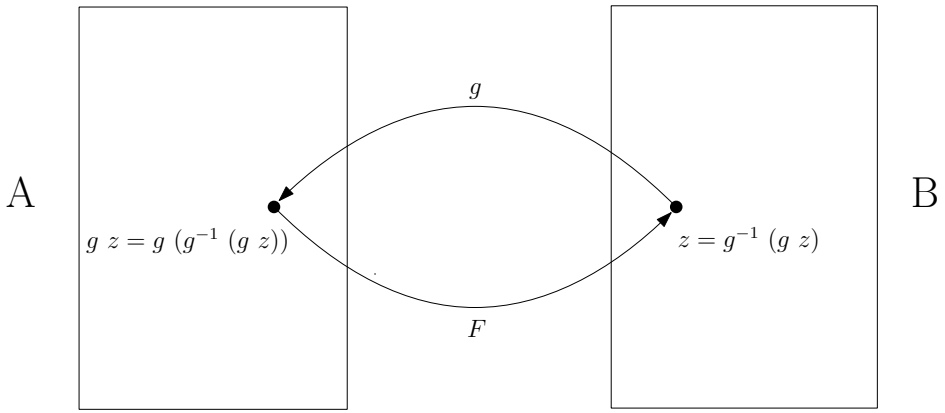
Nyní musíme dokázat `F (g z) = z` za této situace:



Momentálně nemáme k dispozici rovnost mezi puntíky napravo.[8] Naštěstí nám term `hgz.g_inverze` dává rovnost mezi puntíky nalevo. Nadchází chvíle, kdy využijeme, že `g` je prostá funkce. Takhle je zbytek důkazu v této větvi (kvůli způsobu, jak je definována prostá funkce, používáme na konci důkaz sporem):

```
convert_to inverze hgz = z
• exact dif_pos hgz
by_contra pro_spor
exact hg (inverze hgz) z pro_spor hgz.g_inverze
```

Pro rekapitulaci, situace z minulého obrázku vypadá dle našich závěrů takhle:



Důkaz v `else` větvi provedeme sporem. Kdyby `z` nemělo rodiče, patřilo by do nulté (a tedy sudé) generace:

```
by_contra vnejsi_spor
apply hz
use 0
constructor
```

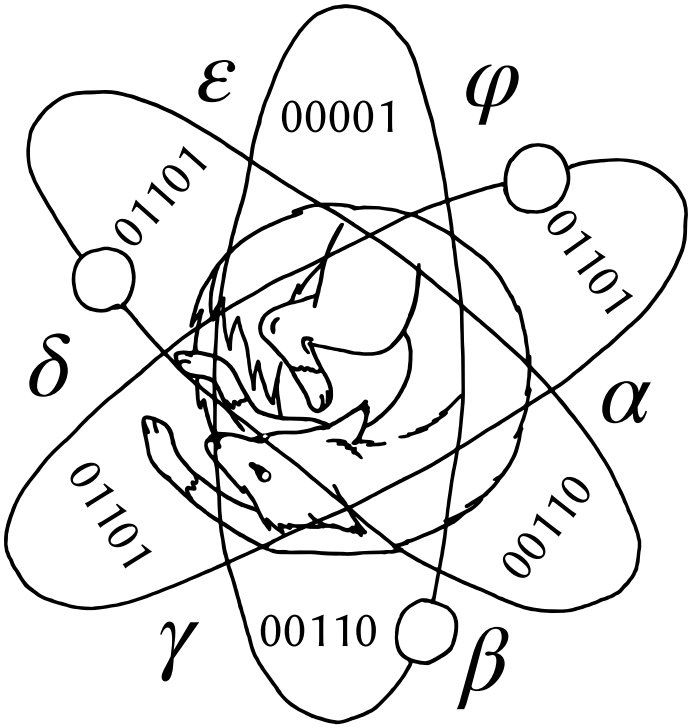
Poslední tři položky Infoview jsou teď klíčové:

```
hz : ¬ SudaGenerace g f z
vnejsi_spor : ¬ ∃ x, F x = z
⊢             ¬ ∃ b, f b = z
```

[8] Nenechte se zmást zkratkou $^{-1}$ používanou pouze v neformálním popisu. Rozmyslete si, že ze způsobu, jak jsme inverzi definovali, plyne, že `g` zavolaná na výsledek $\mathtt{g}^{-1}$ se navzájem vyruší, ale o $\mathtt{g}^{-1}$ zavolané na výsledek `g` to zatím nevíme.

Zbývá už jen trocha nudné manipulace se symboly, abychom Lean přesvědčili, že s využitím `hz` jsou poslední dvě položky to samé. Celý důkaz najdete v repozitáři.[9]

Martin Dvořák; `martin.dvorak@matfyz.cz`
odevzdávejte do odevzdávátka



[9] `https://github.com/madvorak/lean-mam/blob/main/mam/Reseni4.lean`

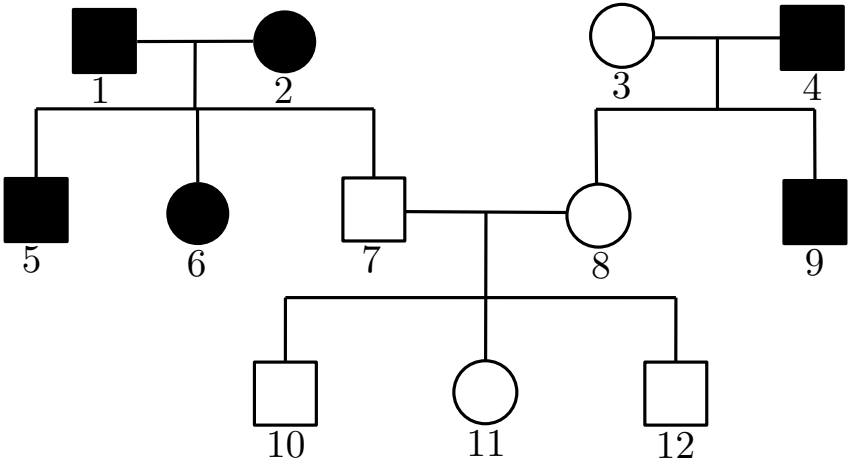
Téma 4 – Genetika

Milí řešitelé, tématko Genetika se s vámi loučí. Níže si můžete přečíst vzorová řešení minulého dílu. Kdybyste ještě měli nějaké dotazy, určitě nám napište na mam@matfyz.cz, moc rádi je zodpovíme.

Vzorová řešení úloh 4. čísla

Úloha 4.1

Zadání:
Poznej, jakému typu dědičnosti patří rodokmen na obrázku 2. Odůvodni svoje řešení a uveď genotypy jedinců 1, 6 a 12.



Obrázek 2: Rodokmen k úloze 4.1

Řešení:
Nejdříve si všimneme, že nakažení jedinci 1, 2 mají zdravého potomka, a tedy nemůže být onemocnění recesivní. To by tito jedinci neměli zdravé alely, které by mohli předat svým potomkům.

Tím pádem jedinci 7, 8 musejí mít obě alely zdravé, a tím pádem oba od svého otce zdědili bezproblémovou alelu. V jednom případě se ale jedná o chlapce, ve druhém o dívku, a tedy v jednom případě byl předán zdravý X chromozom, v druhém pak chromozom Y. Onemocnění tedy nemůže být vázané ani na jeden z pohlavních chromozomů a je tedy autozomální.

Nyní již víme, že je onemocnění autozomálně dominantní a můžeme pokračovat s určováním genotypů. U jedince 1 se nemoc projevuje a musí tedy mít alespoň jednu alelu postiženou. Zároveň ale má zdravého potomka, kterému musel předat zdravou alelu a má tedy genotyp $A_n a$. Jeho potomek 6 je opět nemocný, ale od

svých rodičů mohl druhou alelu zdědit jak zdravou, tak i chybnou, a tedy jeho genotyp může být jak $A_n a$, tak i $A_n A_n$. Jedinec 12 je pak ze všech nejjednodušší, jelikož se u něho neprojevuje nemoc, která je dominantní, tak musí mít obě alely zdravé aa.

Úloha 4.2

Zadání:
S jakou pravděpodobností se dvěma jedincům s achondroplázií narodí dítě se stejným postižením?

V této úloze bylo důležité si uvědomit, že achondroplázie, stejně jako naprostá většina dominantních nemocí, je v homozygotní formě smrtelná. Můžeme tak předpokládat, že oba rodiče jsou heterozygoti. Úlohu správně vyřešila například Mgr.[MM] Anežka Stará:

> Achondroplázie je autozomálně dominantní nemoc, tudíž ji může zdědit dítě nehledě na pohlaví. Homozygoti ($A_n A_n$) se s největší pravděpodobností narodí již mrtví, takže počítám s tím, že oba rodiče jsou heterozygoti – $A_n a$. Takže
>
> - 25 % dětí se nejčastěji vůbec nenarodí, jelikož genotyp $A_n A_n$ bývá neslučitelný se životem;
> - 50 % dětí se narodí se stejným postižením;
> - 25 % dětí se narodí zcela zdravých.

Úloha 4.3

Zadání:
S jakou pravděpodobností budou trpět hemofilií A děti zdravého muže a ženy přenašečky? Jak se procenta změní u dětí muže hemofilika a zdravé ženy?

Řešení podle Mgr.[MM] Veroniky Bartákové:

> V prvním případě dochází ke křížení jedinců $X_n X$ a XY . Nemocí budou trpět pouze mužští potomci $X_n Y$, tedy 25 % dětí. Přenašečkami pak budou ženy $X_n X$, což opět odpovídá 25 % potomků. Zbylých 50 % bude zdravých.
>
> Ve druhém případě dochází ke křížení jedinců XX, $X_n Y$. Všichni potomci zdědí od matky zdravý dominantní chromozom X, žádný potomek tedy nebude nemocný. Veškeré dcery ale budou přenašečkami ($X_n X$).

My ještě poukážeme, k čemu jsme to vlastně došli. Zjistili jsme, že zdravému páru se může narodit nemocné dítě, zatímco u potomků nemocného otce se nemoc projevit nemůže (pokud žena není přenašečka), což bychom intuitivně rozhodně neočekávali.

Úloha 4.4

Zadání:
S jakou pravděpodobností budou postiženy děti gonozomálně recesivním X-vázaným onemocněním, jestliže:

1. *matka je heterozygot Xx_n a otec hemizygot x_nY;*
2. *matka je zdravá XX a otec hemizygot x_nY;*
3. *matka je heterozygot Xx_n, otec zdráv XY;*
4. *matka je dominantní homozygot XX, otec zdravý XY;*
5. *matka je dominantní homozygot XX, otec hemizygot x_nY.*

Zde si stačilo vytvořit Punnettovy čtverce, pěkné řešení měla třeba Doc.[MM] Jana Uglickich:

Onemocnění je gonozomálně recesivní a X-vázané.

- Pár Xx_n, x_nY – pravděpodobnost 0.5, s pravděpodobností 0.25 se narodí přenašečka.

	X	x_n
x_n	x_nX	x_nx_n
Y	XY	x_nY

Tabulka 1: Pár Xx_n, x_nY.

- Pár XX, x_nY – pravděpodobnost 0, s pravděpodobností 0.5 se narodí přenašečka.

	X	X
x_n	x_nX	x_nX
Y	XY	XY

Tabulka 2: Pár XX, x_nY.

- Pár Xx_n, XY – pravděpodobnost 0.25, se stejnou pravděpodobností se narodí přenašečka.

	X	x_n
X	XX	x_nX
Y	XY	x_nY

Tabulka 3: Pár Xx_n, XY.

- Pár XX, XY – pravděpodobnost 0, stejně jako pro narození přenašečky.

	X	X
X	XX	XX
Y	XY	XY

Tabulka 4: Pár XX, XY.

- Pár XX, x_nY – pravděpodobnost 0, pravděpodobnost 0.5 pro narození přenašečky.

	X	X
x_n	x_nX	x_nX
Y	XY	XY

Tabulka 5: Pár XX, x_nY.

Juli; julie.krimska@mensa.cz
Fofík; martin.fof1@seznam.cz
Ludmila; bujnlu@gmail.com
Jirka; polachj5@gmail.com
odevzdávejte do odevzdávátka

Téma 5 – FlatFox#

Řešení 1. dílu

Úloha 4.1

Zadání:
Za pomoci `@+-^>v<#` *naprogramujte (tj. navrhněte les tak, aby liška skončila na zajímavé úložce a splnila následující; je-li napsáno „nějaké číslo“, „$R = r$“ nebo „$r > 0$“, je cílem napsat jeden program, který danou podúlohu dělá pro všechna taková čísla):*

0.5b Vynulování kladného registru: liška si po opuštění nory (`@`) pamatuje pro červenou nějaké kladné číslo ($R = r > 0$); zařiďte, aby si po dosažení zajímavé úložky (`#`) pamatovala pro červenou nulu ($R = 0$).

1b Přesun registru: liška si po opuštění nory pamatuje pro červenou nějaké kladné číslo ($R = r > 0$); po zastavení si má pamatovat toto číslo v zeleném registru ($G = r$).

1b Kopírování registru[10]: liška si na začátku pamatuje $R = r > 0$ a na konci si chce pamatovat toto číslo v zeleném a modrém registru ($G = B = r$).

1.5b Součet dvou kladných čísel: $R = r > 0$, $G = g > 0$, zařiďte, aby $B = r + g$. (Rozmyslete si i rozdíl.)

1.5b Porovnání dvou kladných čísel: $R = r > 0$, $G = g > 0$, nastavte modrý registr na 1 ($B = 1$), pokud je $r < g$, jinak nastavte modrý na 0 ($B = 0$).

1b Vydělte se zbytkem číslo třemi (a popište, jak by se dělilo větší konstantou): máme $R = r > 0$, chceme G nastavit na celočíselný podíl a B na zbytek po dělení r třemi.

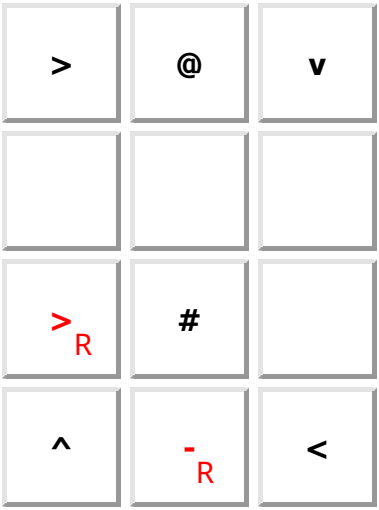
2b Vynásobte dvě kladná čísla.

2.5b Vydělte dvě kladná čísla se zbytkem.

3b Zda je číslo záporné: máme $R = r$, nastavte $G = 1$, pokud $r < 0$, jinak $G = 0$. (Nezapomeňte na to, že se liška musí zastavit.) (V této části úlohy je nejlepší časová složitost docela složitá věc, tedy zde nepožadujeme se jí snažit optimalizovat.)

Řešení:
V první podúloze (vynulování) stačilo z úvodního příkladu programu ve FlatFox#u umazat zelená plus (Obrázek 3). Mgr.[MM] Miroslav Stýskala a Dr.[MM] Radim Novák však přišli na řešení s o dost menší prostorovou i (ne-asymptotickou) časovou složitostí: `https://flatfox.moznabude.cz/#1.ffs`

[10] V ukázkové úloze je docela hloupé, že liška během násobení dvěma zapomene původní číslo. To můžeme vyřešit tím, že si ho nejdříve zkopírujeme.



Obrázek 3: Přímočaré řešení vynulování registru. (Ne však nejrychlejší.)

Druhá podúloha (přesun) šla taktéž řešit smazáním tentokráte jednoho plus. Kreativní a velmi rychlé řešení pak navrhla Dr.^{MM} Michaela Urbanová: `#MU1.ffs`†.

Třetí podúloha (kopírování) šla stále řešit upravením ukázky, v tomto případě nahrazením zeleného plus za modré. Složitostně nejlépe si však zase vedl Dr.^{MM} Radim Novák: `#3.ffs`†.

Hezké kompaktní řešení další podúlohy (sčítání) odevzdali Doc.^{MM} Ondřej Sedláček spolu s Lukášem Trojanem: `#OSLT4.ffs`†. Hezké na pohled (ale pomalejší) řešení pak odevzdala Dr.^{MM} Lucie Zůnová: `#LZ4.ffs`†.

Porovnat čísla (pátá podúloha) v nejkratším čase se podařilo Mgr.^{MM} Barboře Vosáhlové: `#BVu1.ffs`†. Dr.^{MM} Lucie Zůnová i zde odevzdala vizuálně pěkný program: `#LZ5.ffs`†.

Pro podúlohu dělení třemi se zbytkem (šestá podúloha) odkážeme na třetí číslo 20. ročníku[11], kde je úsporné a rozšiřitelné (na dělení libovolnou konstantou) řešení (Mateje Lieskovského): `#6.ffs`†.

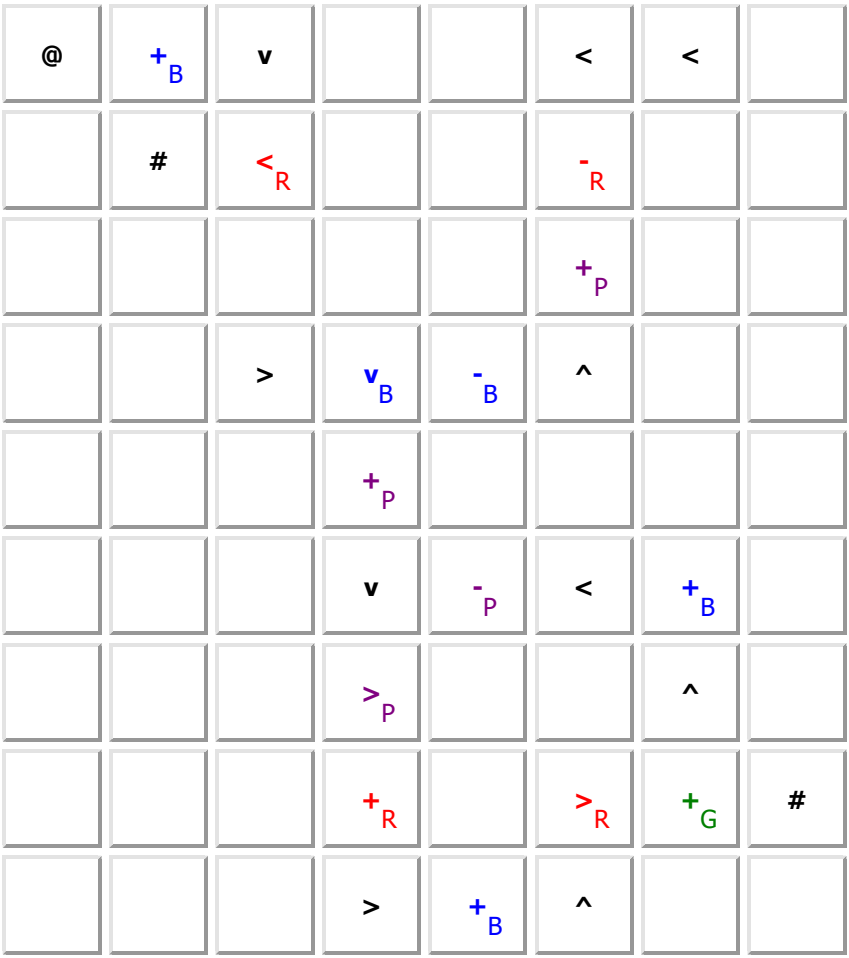
Násobení zvládli nejkompaktněji Doc.^{MM} Ondřej Sedláček spolu s Lukášem Trojanem. Jejich řešení je na první pohled nepřehledné, když ho však pustíte, zjistíte, že jsou to 3 jednoduché cykly („okruhy“, po kterých vždy liška chvíli běhá, načež se přesune k dalšímu): `#OSLT7.ffs`†.

Jednoduché řešení dělení registrem se zbytkem (osmou podúlohu) odevzdala Bc.^{MM} Karolína Česká, které k dělení stačily pouze dva cykly: `#KCu1.ffs`†.

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

[11] `https://mam.mff.cuni.cz/media/cislo/pdf/20/20-3.pdf#page=13`

Poslední podúloha (zjišťování, zda je číslo záporné) byla o dost těžší. Přišlo nám jediné řešení, které tuhle úlohu řeší „rychleji“, ale to tu s ohledem na úlohy v tomto čísle nebudeme ukazovat. Myšlenka všech ostatních řešení byla k danému číslu postupně přičítat a odčítat od něj čím dál větší (vždy o 1) číslo, dokud nenarazíme na nulu. Ze všech těchto řešení je nejpřehlednější to Dr.^{MM} Martina Těšitele (obrázek 4): `https://flatfox.moznabude.cz/#MTu1.ffs`



Obrázek 4: Řešení poslední podúlohy (zjišťování, zda je číslo záporné) od Dr.^{MM} Martina Těšitele. Viz `https://flatfox.moznabude.cz/#MTu1.ffs`

Problém 4.2, 4.3 a 4.7

Zadání:
Dle svého uvážení (budeme bodovat kreativitu) naprogramujte další zajímavé operace.

Zadání:
Vyberte si nějakou podúlohu z úlohy 4.1 (případně vaši operaci z problému 4.2), která přímo nedělá některou z pokročilejších operací FlatFox#u (např. přesun registru, jeho kopírování, nebo zjišťování, zda je číslo záporné) a vyřešte ji snadněji za pomocí nových operací.

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

Řešení:
Bc.[MM] Jan Jedlička kreativně využil násobení a dalších operací z úlohy 4.1 a naprogramoval faktoriál $n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$, který je sice neprakticky pomalý, ale funkční a naprogramovaný jen pomocí `@+-^>v<#`. Jeho řešení můžete nalézt na `#JJ!.ffs`[†]. Výpočet pak zjednodušil (a velmi zrychlil) pomocí dalších operací FlatFox#u: `#JJ!2.ffs`[†].

Dr.[MM] Radim Novák použil pokročilé operace k jednoduchému kopírování registru (to byla jedna z motivací k zadání problému 4.3): `#RNp3.ffs`[†]. Obdobně Dr.[MM] Lucie Zůnová naprogramovala přesun registru: `#LZp3.ffs`[†]. Dr.[MM] Ondřej Nevěřil kromě těchto dvou operací naprogramoval ještě porovnání prvků (za pomoci dělení): `#ONp3.ffs`[†]

Za pomoci dalších operací Doc.[MM] Jana Uglickich vytvořila jednoduchý prográmek na nalezení n-tého Fibonacciho čísla: `#JUfib.ffs`[†]

Doc.[MM] Václav Tichý rozšířil své řešení poslední podúlohy 4.1 na zjišťování, zda je číslo ve Fibonacciho posloupnosti[12]: `#VTp7.ffs`[†]

Nakonec Doc.[MM] Jana Uglickich odevzdala řešení „ověřte, zda číslo je prvočíslo“, které bravurně pracuje s bludnými kořeny, a tak je i moc hezké na pohled: `#JUp7.ffs`[†]

Problém 4.4

Zadání:
K čemu je dobrý bludný kořen? Aneb co je (v programování) funkce, a jak souvisí s bludným kořenem? A proč bychom měli funkce používat?

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

[12]Nastavení hodnot – všechno 0 – kromě barvy turquoise, to je číslo, pro které zkoušíme, zda je ve Fibonacciho posloupnosti. Po doběhnutí programu je barva green nastavena na 0, pokud je číslo součástí posloupnosti, když není, tak je nastavena na 1.

Řešení od Doc.MM Jany Uglickich

> Bludný kořen lišku okamžitě převede z jednoho pole na druhé, což znamená, že ji pomocí dvou bludných kořenů můžu odvést z nějaké části lesa do jiné, nechat ji tam vykonat nějaké příkazy, a zase ji vrátit na původní místo. To by odpovídalo situaci, kdy v „normálním“ programovacím jazyce zavolám funkci.
>
> Využívání bludného kořenu (a funkcí) taky zpřehledňuje a zestručňuje kód jako celek, protože se v něm neopakuje dokola ten samý úsek (špagetový kód), ale (u FlatFox#u) jen jedno pole s odkazem na úsek s funkcí. Kromě zpřehlednění tak ušetřím i instrukce.

Doc.MM Václav Tichý také nadhodil, že by bylo dobré umět skočit na políčko se souřadnicemi danými v registrech, protože pak bychom si mohli uložit, kam se po ukončení funkce vrátit.

To by však ještě zesložitilo už tak složité programovací prostředí, proto to dělat nebudeme. Určitý „workaround“ je očíslovat si místa, odkud funkci voláme. Toto číslo si uložit třeba do registru R a pak mít na konci funkce dlouhou šňůru: červená šipka na bludný kořen, červené mínus; červená šipka, červené mínus; ... (To ale nemusí kód zpřehlednit a navíc ho to může zpomalit.)

Úloha 4.5

Zadání:
Vypište „Hello world!“.

Řešení:
Dr.MM Lucie Zůnová přišla se zajímavě upraveným řešením: `https://flatfox.moznabude.cz#LZu5.ffs`

Přímočaré, hezky čitelné řešení odevzdal Dr.MM Ondřej Nevěřil: `https://flatfox.moznabude.cz#ONu5.ffs` (Podobné řešení odevzdali i Dr.MM Martin Těšitel a Doc.MM Jana Uglickich, kteří využili, že ve slově `Hello` se opakuje `l`.)

Nápadité řešení měl i Dr.MM Radim Novák: `https://flatfox.moznabude.cz#RNu5.ffs`

A stejně tak tu máme jedno velice kompaktní řešení od organizátorů: `https://flatfox.moznabude.cz#ORGu5.ffs`

Úloha 4.6

Zadání:
Načtěte číslo v desítkové soustavě. Tj. na vstup napíšeme číslo v desítkové soustavě, úkolem je mít v R toto číslo jako hodnotu.

Řešení:
Skvělé řešení předvedl Dr.MM Radim Novák, který použil registry jako parametry a umožnil načítat i v jiných soustavách než v desítkové: `https://flatfox.moznabude.cz#RNu6.ffs` (Bohužel v ASCII nejsou čísla a písmena za sebou, tedy fungují pouze soustavy o základu menším než 11.)

Díl druhý: Časová složitost

Při programování nás kromě toho, že program dělá, co má,[13] často zajímá, jak dlouho poběží. Protože mít program, který určitě dá správnou odpověď, ale poběží déle než existuje vesmír, je jaksi nepraktické.

Přístupů k tomu, jak se vypořádat s dobou běhu programu, je více. Samozřejmě můžeme program spustit a měřit jak dlouho poběží. To se většinou dělá, když už máme nějaký větší (složitější) kus softwaru a chceme ho zrychlit, speciálně pokud víme, na jakém počítači a za jakých podmínek program poběží. Tedy například pokud se nějaká stránka na `https://mam.matfyz.cz` načítá příliš pomalu.

To ale nejenže je nepřesné a musíme mít tolik vstupních dat, abychom pokryli všechna použití; ale hlavně takové měření závisí na hardwaru (počítači, konkrétně hlavně na procesoru, paměti a grafické kartě) a na dalších programech, které v danou chvíli na daném počítači běží. Zároveň to také závisí na implementaci (tj. „naprogramování“) funkcí daného jazyka. (Například naše implementace FlatFox# při běhu programu vykresluje jeho stav, což znamená, že s větším lesem krok lišky trvá (nepatrně) déle, přestože při jiné implementaci by se tohle určitě dít nemuselo.)

Proto se při programování základních programů a funkcí díváme spíše na to, jak počet operací (pro nás počet kroků lišky[14]) závisí na vstupu – většinou na jeho velikosti, což obvykle znamená např. počet čísel, v našem případě ale budeme hledat závislost spíše na velikosti čísla/čísel v registru/registrech při opuštění nory (`@`).

Například ukázkový program (z minulého dílu, také ho najdete přímo na `https://flatfox.moznabude.cz`), který dvojnásobek červené uloží do zelené, udělá přesně $10 \cdot R - 3 + 1$ kroků. To je ale příšerně detailní. Koho zajímá, jestli na konci udělá o krok víc nebo méně, a jak už jsme řekli, každý počítač dělá operace jinak rychle, tak k čemu je dobré vědět, že jich udělá 10krát víc? K tomu se nám bude hodit tzv. $\mathcal{O}$ notace („óčková notace“, také velké O notace, omikron notace, nebo velké omikron notace).

$\mathcal{O}$ notace

Ve skutečnosti je tato notace původně spjatá výhradně s matematikou, proto nebudeme zabíhat do detailů a obecnějších definic, ale spíše se zaměříme na to, co je potřeba k časové složitosti.[15] Pro nás je důležité to, že celá $\mathcal{O}$ notace je

[13] Což samozřejmě není vůbec jednoduchá věc, ale FlatFox# není moc uzpůsobený tomu, že bychom mohli psát testy.

[14] To můžeme, protože náš programovací jazyk je jednoduchý. V běžných programovacích jazycích musíme počítat s tím, že ne každá operace trvá stejně dlouho. Odstrašujícím příkladem nechť je násobení čísel v Pythonu (závisí na velikosti čísel) nebo spojování řetězů (často funguje tak, že se místo přidání druhého řetězce na konec prvního oba řetězce nakopírují do řetězce nového, což je při skládání řetězce po malých kouskách velký rozdíl).

[15] Detailně byla tato notace (známá také jako Landauova) probírána v pátém čísle 28. ročníku: `https://mam.mff.cuni.cz/media/cislo/pdf/28/28-5.pdf#page=18`.

o třídách (něco jako množina, neboli „skupinka“) funkcí $\mathcal{O}(\dots)$ a o dvou funkcích $f, g : \mathbb{N} \to \mathbb{N}$ řekneme, že $f \in \mathcal{O}(g)$ (f náleží do třídy $\mathcal{O}(g)$), pokud „f roste (až na konstantu) nejvýše tak rychle jako g“. Formálně $f \in \mathcal{O}(g)$, jestliže existuje $C > 0$ tak, že pro všechna n od nějakého n_0 je $f(n) < C \cdot g(n)$.[16]

Co si pod tím představit? Tak za prvé, vůbec nás netrápí, jaké hodnoty f a g mají pro „malá“ $\mathbb{N}$. Zároveň nás nezajímá vynásobení konstantou (číslem nezávisejícím na velikosti vstupu). Tedy pokud $f \in \mathcal{O}(g)$, pak i $10000 \cdot f \in \mathcal{O}(3.14159 \cdot g)$. Klesajícími a nekladnými funkcemi se vůbec nebudeme zajímat, tedy nejpomaleji rostoucí je pro nás „konstantní složitost“, tj. $\mathcal{O}(1)$, kam patří všechny omezené funkce (speciálně klesající a konstantní). Typickou „velmi pomalu rostoucí“ funkcí je logaritmus[17] (samozřejmě existují i pomaleji rostoucí funkce, např. $\log \log n$, iterovaný logaritmus nebo inverzní Ackermannova funkce). Více než logaritmus roste libovolná (kladná) mocnina, tj. n^r, navíc větší mocnina roste rychleji než menší. Nezapomeňte, že $n^{1/2} = \sqrt{n}$, což je typicky nejmenší mocnina, se kterou se setkáme. Rychleji pak roste první mocnina, tj. n, které se říká lineární složitost, a potom druhá mocnina (n^2), které se říká kvadratická. Nakonec naopak nejrychleji rostoucími běžnými (alespoň co se týče časové složitosti) funkcemi jsou funkce exponenciální, tedy např. 2^n.

Ještě si můžeme všimnout, že nezáleží na „pomaleji rostoucích částech funkce“, tj. např. $n^1 + n^5 + \log n + 1000 \in \mathcal{O}(n^5)$. Pro součin zas platí, že násobíme vnitřek $\mathcal{O}$, tj. například z toho, že $\log n \in \mathcal{O}(\sqrt{n})$ (logaritmus roste pomaleji než libovolná mocnina) můžeme odvodit $n \cdot \log n \in \mathcal{O}(n \cdot \sqrt{n})$.

Pár slov o konstantní složitosti

Naše implementace FlatFox# má vůči tomuto počítání časové složitosti, zvanému asymptotická časová složitost, jednu zásadní vadu. Možná jste si už všimli, že velikost čísel, která si liška je schopna pamatovat, je nejvýše $9223372036854775807 = 2^{63} - 1$, tedy omezená. Stejně tak počet registrů (alespoň pro jeden konkrétní program) je omezený. Tím pádem velikost vstupu je omezená a existuje tedy vstup, na kterém náš program poběží nejdéle. Tudíž by se dalo říct, že programy v naší implementaci běží v $\mathcal{O}(1)$, případně že asymptotická časová složitost zde nedává smysl, jelikož $\mathcal{O}$ notace zkoumá chování pro „velké“ vstupy a těch zde „nemůžeme dosáhnout“. Proto budeme pro tento díl uvažovat, že liška je schopna si zapamatovat libovolně velké číslo.

Úloha 5.1 [1b]: *Mějme město o 100 domech, v nich bydlí $d_1, \dots, d_{100}$ 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.)*

[16] Také se občas hodí třídy $\Theta(\dots)$, $\Omega(\dots)$, $o(\dots)$, $\omega(\dots)$, které znamenají stejně rychle, alespoň tak rychle, (ostře) pomaleji a (ostře) rychleji.

[17] Logaritmus n je zjednodušeně: kolikrát musíme vydělit n třeba 2 (logaritmy o různém základu se liší jen konstanta-krát), než dostaneme číslo menší rovno jedné. Např. $\log_2(100) \approx 7$.

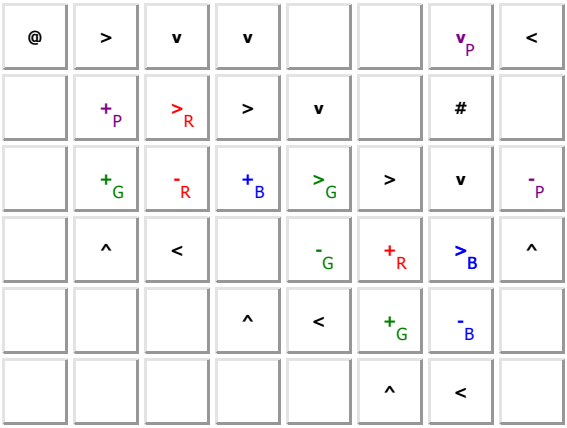
Porovnejte výsledek se zjištěním, že tento program prozkoumá

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

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

Shrnutí

(Asymptotickou) časovou složitost zde počítáme jako počet kroků (s použitím $\mathcal{O}$ notace) v závislosti na velikosti vstupních čísel (která pro naše účely mohou být libovolně velká, i když to naše implementace neumožňuje).



Obrázek 5: Program počítající druhou mocninu. Druhý a třetí sloupec zkopírují červenou do zelené a fialové. Pět dalších sloupců je cyklus opakující se fialově-krát (proto jsme si červenou nakopírovali do fialové). Z toho třetí a čtvrtý sloupec programu přesune zelenou do modré, aby ji pak mohly pátý a šestý sloupec při navracení do zelené přičíst do červené. (Tj. r krát přičte r do červené, kde r je původní hodnota červené.)

Příklad (časová složitost počítání druhé mocniny): Vezměme si program[18] z obrázku 5. V něm nejprve liška udělá 3 kroky k první „rozbočce“ (červená šipka). Potom udělá $(R =)\, r$-krát 8 kroků, následně se jedním krokem přesune do velkého cyklu. Ten se provede $(V =)\, r$-krát a každé jeho provedení sestává z: 2 kroky do druhé (zelené) rozbočky, pak $(G =)\, r$-krát 8 kroků na přesunutí $G \to B$. Následně 3 kroky do třetí (modré) rozbočky, kde program pokračuje $(B =)\, r$-krát 8 kroků na přičtení B do R (a vrácení $B \to G$). Nakonec 9 kroků zase na začátek, krom posledního cyklu, kdy je to o tři kroky méně (k zajímavé úložce). Tedy

$$\underbrace{3 + 8r + 1 + r \cdot (2 + 8r + 3 + 8r + 9) - 3}_{\in\, 3+8r+1+r\cdot\mathcal{O}(r)-3} = \underbrace{1 + 22r + 16r^2}_{\in\, \mathcal{O}(r^2)}.$$

[18] `https://flatfox.moznabude.cz#ctverec.ffs`

Není třeba počítat časovou složitost přesně. Stačí si uvědomit, že v některých místech děláme lineárně (tedy $\mathcal{O}(r)$) kroků, tudíž můžeme rovnou zapomenout na konstantní počty (+3, +1, +2, atd.) kroků. Také si můžeme všimnout, že můžeme zvlášť „zasymptotit“ vnitřek velkého cyklu a až pak se podívat na celkovou asymptotickou složitost a dostaneme stejný (správný) výsledek $\mathcal{O}(r^2)$.

Úloha 5.2 [1b]: *Jakou (asymptotickou) časovou složitost má ukázané řešení první podúlohy 4.1 (mazání registru)?*

Úloha 5.3 [2b]: *Jakou (asymptotickou) časovou složitost má ukázané řešení poslední podúlohy 4.1 (zjišťování, zda je číslo záporné)?*

Zkuste si řešení pustit, podívat se, kterými cykly liška běhá. Pravděpodobně nejtěžší bude zjistit, kolikrát (v závislosti na r) těmito cykly liška proběhne.

Úloha 5.4 [1b+1b]: *Najděte rychlejší řešení zápornosti a určete jeho časovou složitost.*

Úloha 5.5 [bonus]: *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.)*

Úloha 5.6 [1b]: *Proč nemůže program na počítání druhé mocniny (používající pouze `@+-^>v<#`) běžet rychleji než v $\mathcal{O}(r^2)$?*

Problém 4.7: *Naprogramujte další zajímavé programy. Nabízí se například nalezněte n-té (nebo ověřte, zda číslo je): prvočíslo, Fibonacciho číslo, dokonalé číslo, šťastné číslo.*

Můžete u nich také určit časovou složitost.

Problém 4.8: *Co všechno náš programovací jazyk dokáže? Je opravdu třeba všech instrukcí (nestačilo by `@+-^>v<#`)?*

Už se objevilo řešení tohoto problému mluvící o Turingově stroji. Můžete se zamyslet, jaký má vliv na tento problém, zda jsou registry omezené (naše implementace), či neomezené (v tom počítáme časovou složitost).

Jidáš; `jonas.havelka@volny.cz`
odevzdávejte do odevzdávátka
kód nejlépe ve formátu .ffs,
tak, jak ho generuje web `https://flatfox.moznabude.cz`

Výsledková listina 2. deadlinu 3. čísla a 1. deadlinu 4. čísla

				Témata							
Poř.	**Jméno**	**R.**	$\sum_{-1}$	**1**	**2**	**3**	**4**	**5**	**O**	$\sum_{0}$	$\sum_{1}$
1.	Doc.[MM] J. Uglickich	2	294,3	0,5	2,0	2,5	7,8	20,2		33,0	131,5
2.	Dr.[MM] M. Těšitel	3	193,8	12,0	17,5			20,0		49,5	108,3
3.	Dr.[MM] L. Zůnová	3	170,2	10,5	2,0		9,6	21,3		43,4	107,9
4.	Mgr.[MM] B. Vosáhlová	4	94,4		11,5			5,7		17,2	94,4
5.	Mgr.[MM] M. Drexlerová	4	86,1	7,5	4,8	2,7	10,7			25,7	86,1
6.	Mgr.[MM] M. Ambros	1	81,8	3,5	13,5		0,0	14,0		31,0	81,8
7.	Dr.[MM] O. Nevěřil	2	189,6		0,0	1,5		16,0		17,5	67,7
8.	Dr.[MM] R. Novák	4	169,5					19,5	9,0	28,5	65,3
9.	Mgr.[MM] K. Kučerová	Z8	72,6		1,5		12,3	14,5		28,3	64,1
10.	Doc.[MM] O. Sedláček	3	227,5					6,8		6,8	61,0
11.	Dr.[MM] J. Klementová	2	198,0	0,0			10,5			10,5	58,6
12.	Mgr.[MM] L. Votrubová	3	80,1	4,5	1,3					5,8	55,4
13.	Mgr.[MM] M. Stýskala	3	56,5		8,5			14,0		22,5	50,5
14.	Doc.[MM] A. Žák	4	203,8	3,0	0,5					3,5	49,4
15.	Mgr.[MM] M. Kadlec	2	55,8								48,3
16.	Bc.[MM] J. Jedlička	2	48,2	11,5	9,5		8,2	19,0		48,2	48,2
17.	Bc.[MM] A. Bakoč	3	45,4		15,0					15,0	45,4
18.	Doc.[MM] V. Tichý	4	353,3					23,5		23,5	43,0
19.	Mgr.[MM] A. Stará	3	89,3				17,6	1,0		18,6	41,9
20.	Dr.[MM] M. Urbanová	1	103,2				6,5	3,0		9,5	38,9
21.	Mgr.[MM] K. Plchová	4	96,0	1,5			8,9			10,4	34,6
22.	Mgr.[MM] V. Bartáková	4	92,6				12,2			12,2	32,3
23.–24.	Bc.[MM] O. Hrabě	4	31,5					14,0		14,0	31,5
	Mgr.[MM] M. Púll	4	85,1								31,5
25.	Dr.[MM] M. Jarvis	2	179,7								29,9
26.	Bc.[MM] A. Bihun	4	29,2								29,2
27.	Bc.[MM] E. Sabol	4	28,7								28,7
28.	Bc.[MM] K. Maxera	3	31,5		0,5					0,5	28,5
29.	Bc.[MM] T. Pazourek	2	27,5								27,5
30.	Bc.[MM] J. Koška	4	27,3								27,3
31.	Bc.[MM] K. Česká	3	27,2	0,0	1,5			11,5		13,0	27,2
32.	Mgr.[MM] M. Ulumbekov	2	88,1								26,4
33.	Bc.[MM] J. Kadlec	3	24,8	1,5			2,3			3,8	24,8
34.	Bc.[MM] K. Vomelová	4	31,7								24,5
35.–36.	Bc.[MM] P. Barták	Z9	20,4				10,5			10,5	20,4
	Mgr.[MM] J. Löwenhöffer	3	89,9								20,4

Poř.	Jméno	R.	$\sum_{-1}$	Témata 1	2	3	4	5	O	$\sum_0$	$\sum_1$
37.	Mgr.[MM] M. Jursová	4	51,5	0,5			7,7			8,2	19,8
38.	V. Sklenár	4	18,5								18,5
39.	M. Skýpala	3	17,5								17,5
40.	Mgr.[MM] V. Kučera	2	50,6								17,0
41.	Dr.[MM] O. Popovský	4	104,2								16,5
42.	Bc.[MM] L. Poljaková	4	41,6								14,7
43.	Dr.[MM] D. Kaňka	2	131,6								14,5
44.	L. Trojan	4	14,0				7,2	6,8		14,0	14,0
45.	Mgr.[MM] M. Rybecký	2	54,9	1,5			2,8			4,3	13,7
46.	A. Weberová	4	11,6								11,6
47.	Dr.[MM] V. Menšíková	2	140,1								11,3
48.	Dr.[MM] J. Tregler	4	116,3								9,4
49.	A. Stýskala	4	13,9								8,4
50.	Mgr.[MM] V. Vybíral	2	50,4	0,0		1,2				1,2	8,0
51.	M. Ambrosová	Z8	7,8								7,8
52.	A. Migel	1	7,3								7,3
53.	L. Šírová	2	5,9	2,5	3,4					5,9	5,9
54.	Bc.[MM] V. Verner	3	48,9								4,0
55.	R. Zelený	3	3,8								3,8
56.–57.	L. Trochová	1	3,0								3,0
	Bc.[MM] R. Petit	3	37,0								3,0
58.	Bc.[MM] K. Menšíková	1	24,9								1,9
59.	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 příspěvky za články. Tituly uvedené v předchozím textu slouží pouze pro účely M&M.

Kontakty:

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

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

matfyz
M&M