67. ročník Matematické olympiády – 2017/2018

Úlohy krajského kola kategorie P

Krajské kolo 67. ročníku MO kategorie P se koná v úterý 23. 1. 2018 v dopoledních hodinách. Na řešení úloh máte 4 hodiny čistého času. V krajském kole MO-P se neřeší žádná praktická úloha, pro zajištění rovných podmínek řešitelů ve všech krajích je použití počítačů při soutěži zakázáno. Zakázány jsou rovněž jakékoliv další pomůcky kromě psacích potřeb (např. knihy, výpisy programů, kalkulačky, mobilní telefony).

Řešení každé úlohy vypracujte na samostatný list papíru.

Řešení každé úlohy musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, *argumenty zdůvodňující jeho správnost* (případně důkaz správnosti algoritmu), diskusi o efektivitě vašeho řešení (časová a paměťová složitost). Slovní popis řešení musí být jasný a srozumitelný i bez nahlédnutí do samotného zápisu algoritmu (do programu). Není možné odkazovat se na vaše řešení úloh domácího kola, opravovatelé je nemusí mít k dispozici.
- **Zápis algoritmu.** V úlohách **P-II-1**, **P-II-2** a **P-II-3** je třeba uvést zápis algoritmu v nějakém dostatečně srozumitelném pseudokódu (případně v programovacím jazyce Pascal nebo C/C++). Nemusíte detailně popisovat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů, vyhledávání v poli, třídění apod. V řešení úlohy **P-II-4** sestrojte požadované funkce a zapište je takovým způsobem, jak je ukázáno ve studijním textu.

Za každou úlohu můžete získat maximálně 10 bodů. Hodnotí se nejen správnost řešení, ale také kvalita jeho popisu a efektivita zvoleného algoritmu. Algoritmy posuzujeme podle jejich časové složitosti, tzn. závislosti doby výpočtu na velikosti vstupních dat. Záleží přitom pouze na řádové rychlosti růstu této funkce. V zadání každé úlohy najdete přibližné limity na velikost vstupních dat. Efektivním vyřešením úlohy rozumíme to, že váš program spuštěný s takovými daty na současném běžném počítači dokončí výpočet během několika sekund.

Vzorová řešení úloh naleznete krátce po soutěži na webových stránkách olympiády *http://mo.mff.cuni.cz/*. Na stejném místě bude zveřejněn i seznam úspěšných řešitelů krajského kola a seznam řešitelů postupujících do ústředního kola.

P-II-1 Trénink

Jirkovi se v programátorské olympiádě příliš nedařilo, a proto se rozhodl, že začne trénovat – každý den vyřeší několik úloh ze starších ročníků.

Učitel informatiky pozoroval trénujícího Jirku během několika po sobě jdoucích dní. Nevíme přesně, kolik dní to bylo, ale určitě jich bylo alespoň k.

Na Vánoce potom učitel daroval svému oblíbenci Jirkovi ručně pletený svetr s nápisem „v jednom dni jsem vyřešil x úloh“. Samozřejmě, že číslo x bylo nejvyšší mezi všemi počty úloh vyřešených v jednom dni, které učitel viděl.

Soutěžní úloha

Na vstupu je dána posloupnost kladných celých čísel $a_1, \ldots, a_n$. Pro jednoduchost budeme předpokládat, že všechna tato čísla jsou *navzájem různá*.

Z dané posloupnosti viděl učitel *souvislý* úsek tvořený *aspoň* k hodnotami a následně určil maximum z tohoto úseku. Napište program, který spočítá, kolik různých výsledků mohl učitel dostat.

Formát vstupu a výstupu

Na prvním řádku vstupu je číslo n a číslo k. Na druhém řádku jsou čísla $a_1, \ldots, a_n$.

Na výstup vypište jedno číslo udávající počet hodnot, kterým se může rovnat maximum z úseku dané posloupnosti délky alespoň k.

Omezení a hodnocení

Můžete předpokládat, že platí $1 \leq k \leq n$ a že se všechna a_i vejdou do běžných celočíselných proměnných.

Za řešení, jež efektivně vyřeší libovolný vstup s $n \leq 500$, získáte aspoň 3 body.

Za řešení, jež efektivně vyřeší libovolný vstup s $n \leq 5000$, získáte aspoň 4 body.

Za řešení, jež efektivně vyřeší libovolný vstup s $n \leq 100\,000$ a $k \leq 100$, získáte aspoň 5 bodů.

Za řešení, jež efektivně vyřeší libovolný vstup s $n \leq 100\,000$ a libovolným k, získáte aspoň 8 bodů.

Na plných 10 bodů nalezněte a popište optimální řešení úlohy.

Příklad

Vstup:	*Výstup:*
`8 3` `47 30 20 10 80 60 70 50`	`4`

Maximum úseku tvořeného aspoň třemi po sobě jdoucími čísly může nabývat následujících hodnot:

- *80 (např. vezmeme celou posloupnost),*
- *70 (poslední tři prvky posloupnosti),*
- *47 (např. první čtyři prvky), nebo*
- *30 (druhý až čtvrtý prvek).*

Všimněte si, že ačkoliv je číslo 60 třetí nejvyšší hodnotou v posloupnosti, je obklopené většími hodnotami, takže neexistuje dostatečně dlouhý úsek, jehož maximum by bylo rovno 60.

P-II-2 Telenovela II

Možná si pamatujete, jak v domácím kole Jakub sledoval telenovelu. Ondra je mnohem zkušenějším divákem telenovel. Při sledování telenovely dodržuje následující zásady:

1. Úplně na začátku je třeba vidět první díl, v němž se seznámíme s hlavními postavami.
2. Není nutné vidět všechny díly. Všechno důležité se opakuje, takže můžeme libovolné množství dílů vynechat.
3. Je ale důležité sledovat díly ve správném pořadí. Jinými slovy, posloupnost čísel zhlédnutých epizod musí být *rostoucí*.
4. Jelikož Ondra je už zkušený divák, může si dovolit *nejvýše jednu výjimku:* nejvýše jednou se mu může stát, že se podívá na takové dva díly po sobě, že druhý z nich nemá větší číslo než první z nich.
5. Úplně na konci je třeba vidět poslední díl, ve kterém přijde happyend.

Příklad: Představme si, že máme telenovelu s 50 díly. Když se Ondra postupně podívá na osm dílů v pořadí 1, 2, 7, 13, 2, 8, 15 a 50, dodrží všechna popsaná pravidla.

Všimněte si, že některé díly může zhlédnout dvakrát – v našem příkladu je to díl 2.

Soutěžní úloha

Je dána posloupnost $a_1, \ldots, a_n$ čísel těch dílů telenovely, na jejichž vysílání by se Ondra mohl dívat. Zjistěte, zda může zhlédnout telenovelu tak, aby dodržel všechny výše uvedené zásady. Pokud ano, určete také, kolik nejvýše dílů může vidět. Jestliže bude Ondra sledovat některý díl dvakrát, každé sledování takového dílu počítáme zvlášť. Posloupnost dílů, na které se bude dívat, musí začínat prvním a končit posledním dílem a až na nejvýše jednu výjimku musí být ostře rostoucí.

Formát vstupu a výstupu

Na prvním řádku vstupu jsou dvě celá čísla n (počet dílů, které Ondra může sledovat) a e (počet všech epizod telenovely). Epizody jsou očíslovány od 1 do e. Na druhém řádku je posloupnost n celých čísel $a_1, \ldots, a_n$: čísla epizod v chronologickém pořadí, jak po sobě následují ve vysílání.

Program vypíše jedno číslo: maximální počet dílů, které může Ondra vidět při správném sledování telenovely. Jestliže telenovelu nemůže sledovat správným způsobem, program vypíše číslo -1.

Omezení a hodnocení

Vaše řešení musí obsahovat popis celého algoritmu. Jestliže chcete nějak upravit algoritmus z řešení domácího kola, *nestačí* odvolat se na toto řešení, je třeba explicitně popsat, jak a proč tento algoritmus funguje.

Za jakékoliv správné řešení získáte aspoň 3 body.

Za řešení, jež efektivně vyřeší libovolný vstup s $n \leq 500$, získáte aspoň 6 bodů.

Za řešení, jež efektivně vyřeší libovolný vstup s $n \leq 5000$, získáte aspoň 8 bodů.

Za řešení, jež efektivně vyřeší libovolný vstup s $n \leq 100\,000$, můžete získat plných 10 bodů.

Příklady

Vstup:	*Výstup:*
`15 50` `1 4 2 7 1 13 11 2 8 8 3 15 50 42 47`	`8`

Jedním možným optimálním řešením je podívat se na následujících osm dílů:

$$\boxed{1}\ 4\ \boxed{2}\ \boxed{7}\ 1\ \boxed{13}\ 11\ \boxed{2}\ 8\ \boxed{8}\ 3\ \boxed{15}\ \boxed{50}\ 42\ 47$$

Všimněte si, že jediné porušení správného pořadí nastalo, když po dílu 13 Ondra viděl (znovu) díl 2.

Vstup:	*Výstup:*
`10 5` `1 1 2 2 3 3 4 4 5 5`	`6`

Zde je optimálním řešením sledovat díly například v pořadí 1, 1, 2, 3, 4, 5 nebo v pořadí 1, 2, 2, 3, 4, 5.

P-II-3 Vaření

Honza se rozhodl, že si uvaří večeři. Jeho kuchařské schopnosti jsou ale značně omezené. Zvládá používat jen velmi jednoduché recepty, v nichž ze dvou jídel vznikne nějaké třetí. Navíc nemá moc peněz, a tak by byl rád, kdyby ho celé vaření stálo co nejméně.

Soutěžní úloha

Existuje n druhů jídla. Pro potřeby této úlohy si je očíslujeme od 1 do n. Honza si chce připravit jednu porci jídla číslo 1.

Každé jídlo lze přímo koupit v obchodě. Jedna porce jídla číslo i má v obchodě cenu c_i, budeme jí říkat *nákupní cena* jídla. Od každého druhu jídla je možné koupit libovolný počet porcí.

Honza má kuchařskou knihu, která obsahuje r jednoduchých receptů. Každý z receptů má následující tvar: „Z jedné porce jídla s_i a jedné porce jídla t_i připravíš jednu porci jídla u_i.“

Napište program, který spočítá, za jakou nejnižší cenu lze získat jednu porci jídla číslo 1.

Formát vstupu a výstupu

Na prvním řádku vstupu jsou čísla n a r: počet druhů jídla a počet receptů.

Na druhém řádku je n kladných celých čísel c_1, ..., c_n: nákupní cena jedné porce pro každé jídlo.

Zbytek vstupu tvoří r řádků, z nichž každý má tvar „s_i t_i `->` u_i“ a popisuje jeden recept.

Program vypíše jedno číslo: nejmenší celkovou cenu jedné porce jídla číslo 1.

Omezení a hodnocení

Za libovolné správné řešení získáte aspoň 4 body.

Za libovolné správné řešení, jehož časová složitost je polynomiální vzhledem k n a r a které obsahuje také korektní důkaz správnosti, získáte aspoň 7 bodů.

Vzorové řešení vyřeší efektivně libovolný vstup, pro který platí $1 \leq n \leq 100\,000$ a $0 \leq r \leq 100\,000$.

Příklady

Vstup:	*Výstup:*

`5 3`
`47 1 10 20 4700`
`4 5 -> 1`
`2 3 -> 4`
`3 2 -> 5`
	`22`

Kdyby Honza přímo koupil jednu porci jídla číslo 1, stálo by ho to 47. Existují ale levnější postupy. Nejlepší z nich vypadá takto:

Za 1 + 1 + 10 + 10 koupí Honza dvě porce jídla 2 a dvě porce jídla 3. Potom z jedné dvojice jídel 2 a 3 uvaří jídlo 4 a z druhé dvojice uvaří jídlo 5. Na závěr z jedné porce jídla 4 a jedné porce jídla 5 uvaří požadované jídlo 1.

Vstup:

```
4 3
1000 1000 1000 10
2 2 -> 1
3 3 -> 2
4 4 -> 3
```

Výstup:

```
80
```

Tentokrát je nejlepší jednu porci jídla číslo 1 postupně uvařit z osmi porcí jídla číslo 4.

P-II-4 Stavebnice funkcí

K této úloze se vztahuje studijní text uvedený na následujících stranách. Studijní text je shodný se studijním textem z domácího kola, až na to, že na jeho konci je doplněn jeden nový oddíl o predikátech.

Jednotlivé části soutěžní úlohy můžete řešit v libovolném pořadí. Každá část je hodnocena zvlášť. Ve svém řešení můžete využívat:

- *Všechny funkce sestrojené ve studijním textu.*
- *Funkce z domácího kola: mul (násobení), p (předchůdce) a sub (odčítání, které nepodteče pod nulu).*
- *Všechny konstantní funkce libovolné arity. Konstantní funkci s a vstupy, která vždy vrátí hodnotu b, si označíme k_b^a. (Z domácího kola již víme, jak bychom sestrojili kteroukoliv z těchto funkcí.)*
- *Funkce sestrojené v předcházejících částech této úlohy, a to i v případě, že jste tyto části nevyřešili.*

Konstrukci dříve sestrojených funkcí nerozepisujte. Například funkci $f(x) = 2x$ stačí uvést ve tvaru $K[v_1^1, v_1^1, add]$, není třeba znovu popisovat celou konstrukci funkce add.

Úkol A: *(3 body)* Sestrojte binární funkci *pow* takovou, že $\forall x, y : pow(x, y) = x^y$. Zvláště upozorňujeme, že $pow(0, 0) = 1$.

Úkol B: *(2 body)* Sestrojte unární funkci *sgn* (signum) takovou, že $sgn(0) = 0$ a $\forall x > 0 : sgn(x) = 1$.

Úkol C: *(2 body)* Sestrojte predikát *geq* (greater or equal): binární funkci, pro kterou platí, že když $x \geq y$, pak $geq(x, y) = 1$, jinak je $geq(x, y) = 0$.

Úkol D: *(3 body)* Tomáš včera sestrojil dvě různé n-ární funkce f_0 a f_1 a jeden n-ární predikát g.

Uvažujme funkci h definovanou následujícím předpisem:

$$h(x_1, \ldots, x_n) = \begin{cases} f_0(x_1, \ldots, x_n) & \text{jestliže } g(x_1, \ldots, x_n) = 0 \\ f_1(x_1, \ldots, x_n) & \text{jestliže } g(x_1, \ldots, x_n) = 1 \end{cases}$$

Funkce h tedy odpovídá větvení (příkazu „if“) v běžném programovacím jazyce: podle toho, zda je splněna podmínka g, počítáme výstup funkce h buď funkcí f_0, nebo funkcí f_1.

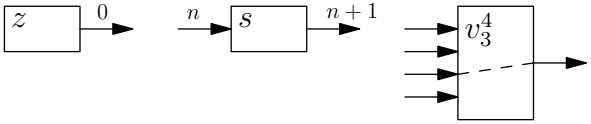
Dokažte nebo vyvraťte tvrzení: funkci h lze vždy sestrojit z Tomášovy stavebnice (samozřejmě za předpokladu, že je možné sestrojit funkce f_0, f_1 a g).

Studijní text

Tomáš dostal k narozeninám zvláštní dárek: stavebnici funkcí. Když balíček rozbalil, našel v něm hned několik různých věcí. Nejprve uviděl tři sáčky s hotovými funkcemi. Každá funkce je malá krabička, která má několik vstupů a právě jeden výstup.

- V prvním sáčku byla jediná funkce. Jmenovala se z (z anglického „zero“, tedy nula), neměla žádné vstupy a na výstupu vracela stále číslo 0.
- Také ve druhém sáčku byla jen jedna funkce. Tato se nazývala s (z anglického „successor“, tedy následník). Měla jeden vstup a jeden výstup. Když na vstupu dostala číslo n, vrátila nám na výstupu číslo $n + 1$.
- Třetí sáček byl o něco plnější – bylo v něm nekonečně mnoho funkcí. Pro každé k a n (takové, že $1 \leq k \leq n$) tam byla funkce v_k^n („vyber k-tý z n vstupů“), která měla n vstupů a na výstup vždy vrátila tu hodnotu, kterou dostala na svém k-tém vstupu.

  Na obrázku jsou znázorněny funkce z, s a v_3^4.



Zbytek balíčku obsahoval dva přístroje, které zjevně slouží k výrobě nových funkcí. Na jednom z nich byl nápis *Kompozitor,* na druhém *Cyklovač.* Každý z přístrojů funguje tak, že dovnitř vložíme ve správném pořadí nějaké funkce, zatočíme klikou a vypadne nám nová funkce. Ta je vhodně poskládána z funkcí, které jsme do přístroje vložili. Než si podrobněji popíšeme fungování Kompozitoru a Cyklovače, potřebujeme si o našich funkcích něco říci formálněji.

V této úloze považujeme nulu za přirozené číslo. Přirozená čísla pro nás tedy tvoří množinu $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.

Všechny funkce, s nimiž budeme pracovat, budou definovány na celém oboru přirozených čísel. Na vstupu budeme tedy funkci zadávat přirozená čísla a pro každý možný vstup nám funkce vrátí na výstupu jedno přirozené číslo.

Počet vstupů funkce se nazývá *arita.* Například funkce s jedním vstupem se nazývá *unární,* funkce se dvěma vstupy *binární,* atd. Funkce v_2^7 má aritu 7. Funkce z má aritu 0. Aritu funkce budeme někdy explicitně zapisovat jako horní index. Mohli bychom tedy například říci, že v prvním sáčku byla funkce z^0 a ve druhém funkce s^1. U vybíracích funkcí v_k^n musíme aritu uvádět vždy, jelikož třeba v_1^4 a v_1^7 jsou dvě různé funkce.

Jakmile nějakou funkci vytvoříme, máme navždy k dispozici libovolné množství jejích kopií. Speciálně platí, že když funkci použijeme při výrobě jiné, složitější funkce, původní funkci tím neztratíme.

Kompozitor

Kompozitor umí funkce skládat (v běžném matematickém smyslu tohoto slova). Musíme ovšem dávat pozor na správné arity funkcí. Použití Kompozitoru se skládá z následujících kroků:

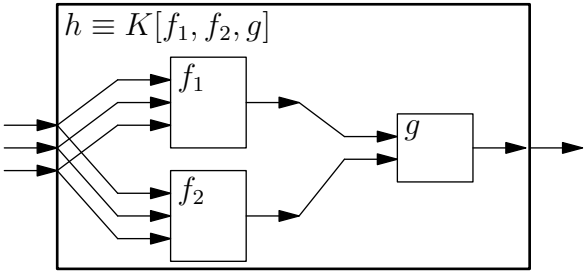
1. Zvolíme si aritu $a \geq 0$ funkce, kterou chceme vytvořit.
2. Zvolíme si funkci g^b (tedy funkci g s nějakou aritou b, třeba i jinou než a), kterou použijeme ve druhé fázi výpočtu. Hodnota b musí být kladná – jinými slovy, funkce g musí mít aspoň jeden vstup.
3. Zvolíme si b funkcí $f_1^a, \ldots, f_b^a$, které použijeme v první fázi výpočtu.
4. Zatočíme klikou na Kompozitoru a vypadne nám z něj nová funkce h definovaná následujícím pseudokódem:

```
def h ( x_1, ..., x_a ):
    tmp_1 = f_1 ( x_1, ..., x_a )
    tmp_2 = f_2 ( x_1, ..., x_a )
    ...
    tmp_b = f_b ( x_1, ..., x_a )

    return g ( tmp_1, ..., tmp_b )
```

   Jak vidíte, funkce h vezme a vstupů, které dostala. Pomocí funkcí f_1 až f_b z nich vypočítá b pomocných hodnot. Nakonec pomocí funkce g spočítá z pomocných hodnot výstup celé funkce h.

Na obrázku je graficky znázorněna funkce vyrobená Kompozitorem pro $a = 3$ a $b = 2$.



Cyklovač

Cyklovač umí provádět for-cykly. Také při jeho použití musíme dbát na správné arity funkcí a pořadí jejich parametrů. Správné použití Cyklovače vypadá takto:

1. Zvolíme si aritu $a \geq 1$ funkce, kterou chceme vytvořit. Prvním parametrem této funkce (označíme ho x) bude speciální proměnná, která určuje, kolikrát se má for-cyklus provést. Ostatní parametry (označíme je $y_1, \ldots, y_{a-1}$) budou zůstávat beze změny.
2. Zvolíme si funkci f^{a-1}, jejímž provedením bude výpočet začínat.

3. Zvolíme si funkci g^{a+1}, která počítá, co se stane při jedné iteraci cyklu. Funkce g má $a+1$ vstupů: všechny proměnné, které bude mít i výsledná funkce, a navíc ještě hodnotu, která byla výstupem předchozí iterace cyklu.
4. Zatočíme klikou na Cyklovači a vypadne nám z něj nová funkce h definovaná následujícím pseudokódem:

```
def h ( x, y_1, ..., y_{a-1} ):
    tmp = f ( y_1, ..., y_{a-1} )
    for i = 0 to x-1:
        tmp = g ( i, y_1, ..., y_{a-1}, tmp )
    return tmp
```

Výpočet tedy začne tím, že funkcí f spočítáme (z ostatních parametrů) výstup pro $x = 0$. Z něho potom funkcí g vypočítáme výstup pro $x = 1$, z něj opět funkcí g výstup pro $x = 2$, a tak dále až po požadovanou hodnotu prvního parametru.

Značení

Rovnost dvou funkcí budeme zapisovat symbolem $\equiv$. Zápis $f \equiv g$ tedy znamená, že funkce f a g mají stejnou aritu a na každém vstupu dávají stejný výstup.

Funkci, která vznikne Kompozitorem z funkcí $f_1, \ldots, f_b$ a g, budeme označovat $K[f_1, \ldots, f_b, g]$.

Funkci, která vznikne Cyklovačem z funkcí f a g, budeme značit $C[f, g]$.

Příklad 1

Pojďme se nyní společně podívat na to, jak si Tomáš začal vytvářet nové funkce. Pěknou jednoduchou funkcí je například identita: unární funkce i taková, že pro každé n je $i(n) = n$. Víte, jak ji vyrobit?

To byla triková otázka. Identitu vytvářet nepotřebujeme, dostali jsme ji ve třetím sáčku. Identitou je totiž funkce v_1^1. Můžeme proto psát $i \equiv v_1^1$.

Příklad 2

Vyrobíme si funkci j^0: konstantní funkci, která nemá žádný vstup a na výstupu vrací hodnotu 1.

Tuto funkci vytvoříme Kompozitorem. V prvním kroku použijeme funkci z, která nemá žádný vstup a na výstupu vrátí 0. Tuto 0 potom „pošleme dále“ do funkce s, která ji zvýší na 1.

Dostáváme tedy $j^0 \equiv K[z, s]$.

Příklad 3

Unární funkci *plus3*, která svůj jediný vstup zvýší o 3, získáme například jako $K[K[s, s], s]$. Nejprve si tedy vytvoříme funkci $K[s, s]$, která svůj vstup zvýší o 2, a tuto funkci potom opět vložíme do Kompozitoru s další funkcí s.

Příklad 4

Nyní si ukážeme, jak si Tomáš může vyrobit sčítání – tedy binární funkci *add* takovou, že $\forall x, y : add(x, y) = x + y$.

Základním pozorováním je, že sčítání je vlastně opakované použití funkce „+1“, tedy následníka. Výpočet $x + y$ si můžeme zformulovat takto: „začni s hodnotou y a potom na ni x-krát použij funkci s“. Vypadá to jako cyklus, takže na výrobu sčítání budeme chtít použit Cyklovač.

Podívejme se na pseudokód funkce, kterou vytvořil Cyklovač (s tím, že si ho už upravíme konkrétně na funkci se dvěma vstupy).

```
def add(x,y):
    tmp = f(y)
    for i = 0 to x-1:
        tmp = g(i,y,tmp)
    return tmp
```

Jaké funkce f a g potřebujeme vložit do Cyklovače, když chceme dostat funkci pro sčítání?

Funkce f má jednoduše vrátit hodnotu y, kterou dostala na vstupu – takže f bude identita.

Funkce g má v každé iteraci cyklu vzít starou hodnotu (uloženou v proměnné `tmp`) a zvýšit ji použitím funkce s. Potřebujeme tedy funkci se třemi vstupy, která první dvě vstupní hodnoty ignoruje, na třetí použije funkci s a vrátí výsledek této operace. Takovou funkci sice ještě nemáme, ale umíme si ji snadno vytvořit Kompozitorem: je to funkce $K[v_3^3, s]$.

Dohromady tedy dostáváme $add \equiv C[\, v_1^1,\ K[v_3^3, s]\,]$.

Příklad 5

Další jednoduchou funkcí je unární konstantní nula, tedy funkce zz^1, která má jeden vstup a na výstupu vždy vrací nulu. (Formálně: $\forall n : zz^1(n) = 0$.)

Než si ukážeme, jak zz^1 vyrobit, poznamenejme, že zz^1 a z^0 (což je funkce z, kterou jsme dostali v prvním sáčku) jsou dvě různé funkce.

Zdálo by se, že funkci zz^1 půjde nějak vyrobit z funkce z^0 pomocí Kompozitoru. Jenže jak? Kdybychom použili funkci z^0 v prvním kroku, tak bez ohledu na to, jakou funkci použijeme ve druhém kroku, určitě získáme funkci s aritou 0. V druhém kroku funkci z^0 použít nesmíme, neboť funkce použitá v druhém kroku musí mít kladnou aritu.

Přes Kompozitor tedy cesta nevede. Ukážeme si ale, že funkci zz^1 dokážeme vytvořit pomocí Cyklovače. Nejprve se opět podíváme, jak to vypadá, když chceme pomocí Cyklovače vyrobit unární funkci. My dodáme funkce f^0 a g^2 a Cyklovač nám z nich sestaví funkci h^1 definovanou následovně:

```
def h(x):
    tmp = f()
    for i = 0 to x-1:
        tmp = g(i,tmp)
    return tmp
```

Jak zvolíme funkce f a g, aby funkce h pro každý vstup vracela nulu? Zjevně musíme zvolit $f \equiv z^0$, aby bylo $h(0) = 0$. Za funkci g můžeme jednoduše vzít $g \equiv v_2^2$.

Tím se z příkazu `tmp = g(i,tmp)` stane příkaz `tmp = tmp`, a proto v proměnné `tmp` zůstane stále nula bez ohledu na hodnotu x.

Touto poměrně umělou konstrukcí jsme si tedy ukázali, že funkci zz^1 můžeme sestrojit jako $C[z, v_2^2]$.

Predikáty

Predikát je odborné označení funkce, jejíž návratovou hodnotou je logická hodnota: pravda nebo nepravda. Naše funkce takové hodnoty sice vracet nedovedou, pomůžeme si ale stejně, jako to udělali kdysi například autoři programovacího jazyka C: prohlásíme 0 za nepravdu a 1 za pravdu.

Funkci tedy budeme nazývat predikát, jestliže:

- tato funkce pro každý vstup vrátí jednu z hodnot 0 nebo 1,
- chceme zdůraznit, že na tyto hodnoty má smysl pohlížet jako na logické hodnoty namísto číselných.

Příklady: Funkce $eq(x, y)$, která vrátí 1, pokud se její vstupy rovnají, zatímco v ostatních případech vrátí 0, je predikát. Predikátem je také funkce $isprime(x)$, která vrátí hodnotu 1, jestliže x je prvočíslem, a hodnotu 0, pokud x prvočíslem není. Na unární konstantní funkci zz^1 se můžeme dívat jako na predikát, který vždy vrací hodnotu „nepravda“.