Priebeh krajského kola

Krajské kolo 33. ročníka Olympiády v informatike, kategória A, sa koná 23. januára 2018 v dopoludňajších hodinách. Na riešenie úloh majú súťažiaci **4 hodiny čistého času**. Rôzne úlohy riešia súťažiaci na samostatné listy papiera. Akékoľvek pomôcky okrem písacích potrieb (napr. knihy, výpisy programov, kalkulačky) sú zakázané.

Čo má obsahovať riešenie úlohy?

- Slovne popíšte algoritmus.
  Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného algoritmu/programu.
- Zdôvodnite správnosť vášho algoritmu.
- Uveďte a zdôvodnite jeho časovú a pamäťovú zložitosť.
- Podrobne uveďte dôležité časti algoritmu, ideálne vo forme programu v nejakom bežnom programovacom jazyku (napr. C++, Python, Java, Pascal).
- V prípade, že používate vo svojom programovacom jazyku knižnice, ktoré obsahujú implementované dátové štruktúry a algoritmy (napr. STL pre C++), v popise algoritmu stručne vysvetlite, ako by ste napísali program s rovnakou časovou zložitosťou bez použitia knižnice.

Hodnotenie riešení

Za každú úlohu môžete získať od 0 do 10 bodov.

Pokiaľ nie je v zadaní povedané ináč, najdôležitejšie dve kritériá hodnotenia sú v prvom rade **správnosť** a v druhom rade **efektívnosť** navrhnutého algoritmu. Na výslednom počte bodov sa môže prejaviť aj kvalita popisu riešenia a zdôvodnenie tvrdení o jeho správnosti a efektívnosti.

Efektívnosť algoritmu posudzujeme vypočítaním jeho časovej zložitosti – funkcie, ktorá hovorí, ako dlho vykonanie algoritmu trvá v závislosti od veľkosti vstupných parametrov. Nezávisí pri tom na konštantných faktoroch, len na rádovej rýchlosti rastu tejto funkcie.

V zadaní úloh uvádzame časť „Hodnotenie“, v ktorej nájdete približné limity na veľkosť vstupných údajov. Pod pojmom „efektívne vyriešiť“ chápeme to, že váš program spustený na modernom počítači by mal dať odpoveď nanajvýš do niekoľkých sekúnd.

Údaje z tejto časti zadania by mali slúžiť hlavne na to, aby ste o riešení, ktoré vymyslíte, vedeli približne povedať, koľko bodov zaň dostanete.

A-II-1 Tréning

Samkovi sa v OI príliš nedarilo, a tak sa rozhodol, že bude trénovať – na stránke Rýchlostného programovania[1] si každý deň vyrieši niekoľko starých úloh.
Učiteľ Oňo trénujúceho Samka niekoľko po sebe idúcich dní pozoroval. Nevieme, koľko presne tých dní bolo, ale určite ich bolo aspoň k.
Na Vianoce potom Oňo daroval Samkovi ručne pletený sveter s nápisom „v jeden deň som vyriešil x úloh“. Samozrejme, číslo x bolo najväčšie spomedzi čísel, ktoré Oňo videl.

Súťažná úloha

Na vstupe je daná postupnosť kladných celých čísel $a_1, \ldots, a_n$. Pre jednoduchosť budeme predpokladať, že všetky tieto čísla sú **navzájom rôzne**.
Z danej postupnosti Oňo videl **súvislý** úsek tvorený **aspoň k** hodnotami a následne našiel maximum vybratého úseku. Napíšte program, ktorý vypočíta, koľko rôznych výsledkov mohol Oňo dostať.

Formát vstupu a výstupu

V prvom riadku vstupu je číslo n a číslo k. V druhom riadku vstupu sú čísla $a_1, \ldots, a_n$.
Na výstup vypíšte jeden riadok a v ňom počet hodnôt, ktoré môže mať maximum dostatočne dlhého úseku danej postupnosti.

Obmedzenia a hodnotenie

Môžete predpokladať, že platí $1 \leq k \leq n$ a že sa všetky a_i zmestia do bežných celočíselných premenných.

Riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 500$, môže získať aspoň 3 body.
Riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 5000$, môže získať aspoň 4 body.
Riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 100\,000$ a $k \leq 100$, môže získať aspoň 5 bodov.
Riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 100\,000$ a ľubovoľným k, môže získať aspoň 8 bodov.
Na plných 10 bodov nájdite a implementujte optimálne riešenie tejto úlohy.

Príklady

vstup	výstup
8 3 47 30 20 10 80 60 70 50	4

Maximum úseku tvoreného aspoň tromi po sebe idúcimi prvkami môže nadobúdať nasledovné hodnoty:

- *80 (napr. zoberieme celú postupnosť),*
- *70 (posledné tri prvky),*
- *47 (napr. prvé štyri prvky), alebo*
- *30 (druhý až štvrtý prvok).*

Všimnite si, že hoci je číslo 60 treťou najväčšou hodnotou v postupnosti, je obklopené väčšími hodnotami, a teda neexistuje dostatočne dlhý úsek, ktorý by mal maximum 60.

[1] http://ksp.sk/~acm

A-II-2 Telenovela II

Možno si pamätáte, ako v domácom kole Jaro sledoval telenovelu. Ondro žije na ďalekom východe a je omnoho skúsenejším divákom telenoviel. Ondro má pri pozeraní telenovely nasledovné zásady:

1. Úplne na začiatku treba pozrieť prvý diel, v ktorom sa zoznámime s hlavnými postavami.
2. Nie je potrebné vidieť všetky diely. Všetko dôležité sa opakuje, takže vieme ľubovoľne veľa dielov vynechať.
3. Aby sa človek nestratil v deji, je veľmi dôležité sledovať diely v správnom poradí. Inými slovami, postupnosť čísel pozretých epizód musí byť **rastúca**.
4. Ale keďže Ondro už je skúsený, môže si dovoliť **nanajvýš jednu výnimku**: nanajvýš raz sa mu môže stať, že si po sebe pozrie dva diely, z ktorých druhý nemá väčšie číslo ako prvý.
5. Na úplný záver je potrebné vidieť posledný diel, v ktorom príde happyend.

Príklad: Predstavme si, že máme telenovelu s 50 dielmi. Ak si Ondro postupne pozrie osem dielov, a to postupne diely 1, 2, 7, 13, 2, 8, 15 a 50, tak dodrží všetky vyššie popísané pravidlá.
(Všimnite si, že niektoré diely mohol vidieť aj dvakrát – v našom príklade je to diel 2.)

Súťažná úloha

Daná je postupnosť $a_1, \ldots, a_n$ čísel dielov, ktorých vysielanie by Ondro stíhal sledovať. Zistite, či si vie pozrieť telenovelu tak, aby dodržal vyššie uvedené pravidlá. Ak áno, zistite tiež, koľko najviac dielov vie vidieť. (Ak bude pozerať ten istý diel dvakrát, každé pozretie rátame zvlášť.) Postupnosť dielov, ktoré si Ondro pozrie, musí začínať prvým dielom, končiť posledným, a až na nanajvýš jednu výnimku musí byť ostro rastúca.

Formát vstupu a výstupu

V prvom riadku vstupu sú celé čísla n (počet dielov, ktoré Ondro stíha sledovať) a e (počet epizód telenovely). Epizódy sú očíslované od 1 po e.
V druhom riadku je postupnosť n celých čísel $a_1, \ldots, a_n$: čísla odvysielaných epizód v chronologickom poradí.

Vypíšte jeden riadok a v ňom jedno číslo: maximálny počet dielov, ktoré vie Ondro vidieť pri správnom pozretí tejto telenovely. Ak Ondro správnym spôsobom telenovelu pozrieť nevie, vypíšte namiesto toho číslo -1.

Obmedzenia a hodnotenie

Vaše riešenie by malo obsahovať implementáciu a popis celého algoritmu. Špeciálne zdôrazňujeme, že ak chcete nejak upraviť algoritmus z riešení domáceho kola, **nestačí** sa odvolať na toto riešenie, je potrebné explicitne popísať, ako a prečo tento algoritmus funguje.
Ľubovoľné korektné riešenie môže získať aspoň 3 body.
Riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 500$, môže získať aspoň 6 bodov.
Riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 5000$, môže získať aspoň 8 bodov.
Riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 100\,000$, môže získať plných 10 bodov.

Príklady

vstup	výstup
15 50 1 4 2 7 1 13 11 2 8 8 3 15 50 42 47	8

Jedno možné optimálne riešenie je pozrieť nasledujúcich osem dielov:
[1] 4 [2] [7] 1 [13] 11 [2] 8 [8] 3 [15] [50] 42 47
Všimnite si, že jediné porušenie správneho poradia nastalo, keď po diele 13 Ondro pozrel (znova) diel 2.

vstup	výstup
10 5 1 1 2 2 3 3 4 4 5 5	6

Tu je optimálne pozrieť diely napr. v poradí 1, 1, 2, 3, 4, 5 alebo v poradí 1, 2, 2, 3, 4, 5.

A-II-3 Varenie

Miško sa rozhodol, že uvarí Baške večeru. Jeho kuchárske schopnosti však nie sú nijak úžasné, zvláda len veľmi jednoduché recepty, pri ktorých z dvoch vecí vznikne tretia. A navyše teraz Miško nemá peňazí nazvyš, a tak by bol rád, keby ho celé varenie stálo čo najmenej.

Súťažná úloha

Existuje n druhov jedla. Pre potreby tejto úlohy si ich očíslujeme od 1 po n. Miško chce Baške navariť jednu porciu jedla číslo 1.
Každé jedlo sa dá priamo kúpiť v obchode. Jedna porcia jedla číslo i má v obchode cenu c_i. Z každého jedla vie Miško nakúpiť ľubovoľne veľa porcií.
Miško má kuchársku knihu, ktorá obsahuje r jednoduchých receptov. Každý z receptov má nasledovný tvar: „Z jednej porcie jedla s_i a jednej porcie jedla t_i takto vyrobíš jednu porciu jedla u_i.“
Napíšte program, ktorý vypočíta, za akú najmenšiu cenu sa dá získať jedna porcia jedla číslo 1.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla n a r: počet druhov jedla a počet receptov.
V druhom riadku je n kladných celých čísel $c_1, \ldots, c_n$: obchodná cena jednej porcie pre každé jedlo.
Zvyšok vstupu tvorí r riadkov, každý z nich má tvar „s_i t_i `->` u_i“ a popisuje jeden recept.

Vypíšte jeden riadok a v ňom jedno číslo: najmenšiu celkovú cenu jednej porcie jedla číslo 1.

Obmedzenia a hodnotenie

Ľubovoľné korektné riešenie môže získať aspoň 4 body.
Ľubovoľné korektné riešenie, ktorého časová zložitosť je polynomiálna od n a r a ktoré obsahuje korektný dôkaz správnosti, môže získať aspoň 7 bodov.
Vzorové riešenie zvláda efektívne vyriešiť ľubovoľný vstup v ktorom platí $1 \leq n \leq 100\,000$ a $0 \leq r \leq 100\,000$.

Príklady

vstup

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

výstup

```
22
```

Ak by Miško priamo kúpil jednu porciu jedla číslo 1, stálo by ho to 47. Existujú však aj lacnejšie postupy. Najlepší spôsob vyzerá nasledovne:
Za 1+1+10+10 kúpi dve porcie jedla 2 a dve porcie jedla 3. Potom z jednej dvojice jedál 2 a 3 uvarí jedlo 4 a z druhej dvojice uvarí jedlo 5. No a na záver z jednej porcie jedla 4 a jednej porcie jedla 5 uvarí želané jedlo 1.

vstup

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

výstup

```
80
```

Tentokrát je najlepšie jednu porciu jedla číslo 1 postupne navariť z ôsmich porcií jedla číslo 4.

A-II-4 Stavebnica funkcií

Za zadaním súťažnej úlohy nájdete študijný text k nej. Študijný text je identický s textom z domáceho kola, až na to, že na jeho konci je doplnený jeden nový odsek o ***predikátoch****.*

Jednotlivé podúlohy súťažnej úlohy môžete riešiť v ľubovoľnom poradí. Každá podúloha je hodnotená zvlášť. Pri riešení konkrétnej podúlohy môžete využívať:

- *Všetky funkcie zostrojené v študijnom texte.*
- *Funkcie z domáceho kola: mul (násobenie), p (predchodca) a sub (odčítanie, ktoré nepodtečie pod nulu).*
- *Všetky konštantné funkcie ľubovoľnej arity. Konštantnú funkciu s a vstupmi ktorá vždy vráti hodnotu b si označíme k_b^a. (Z domáceho kola tiež vieme, ako by sme vyrobili hociktorú z týchto funkcií.)*
- *Funkcie zostrojené v predchádzajúcich podúlohách, a to aj ak ste tieto podúlohy nevyriešili.*

Konštrukciu skôr zostrojených funkcií nerozpisujte. Príklad: funkciu $f(x) = 2x$ stačí uviesť v tvare $K[v_1^1, v_1^1, add]$, netreba znova rozpisovať celú konštrukciu funkcie add.

- **Podúloha A (3 body).** Zostrojte binárnu funkciu *pow* takú, že $\forall x, y : pow(x, y) = x^y$.
  Špeciálne upozorňujeme, že $pow(0, 0) = 1$.
- **Podúloha B (2 body).** Zostrojte unárnu funkciu *sgn* (signum) takú, že $sgn(0) = 0$ a $\forall x > 0 : sgn(x) = 1$.
- **Podúloha C (2 body).** Zostrojte predikát *geq* (greater or equal): binárnu funkciu, pre ktorú platí, že ak $x \geq y$ tak $geq(x, y) = 1$ a inak je $geq(x, y) = 0$.
- **Podúloha D (3 body).** Miška včera zostrojila dve rôzne n-árne funkcie f_0 a f_1 a jeden n-árny predikát g.
  Uvažujme funkciu h definovanú nasledovným predpisom:

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

  Slovne, funkcia h zodpovedá vetveniu (príkazu „if“) v bežnom programovacom jazyku: podľa toho, či je splnená podmienka g, počítame výstup funkcie h buď funkciou f_0 alebo funkciou f_1.

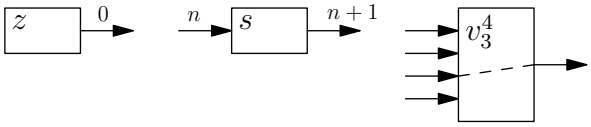
  Dokážte alebo vyvráťte: funkcia h sa vždy dá postaviť z Miškinej stavebnice (samozrejme za predpokladu, že sa dajú postaviť f_0, f_1 aj g).

Študijný text: stavebnica funkcií

Miška dostala na narodeniny zvláštny darček: stavebnicu funkcií. Keď darček rozbalila, našla v ňom hneď niekoľko rôznych vecí. Ako prvé jej oko padlo na tri sáčky s hotovými funkciami. Každá funkcia je malá škatuľka, ktorá má niekoľko vstupov a práve jeden výstup.

- V prvom sáčku bola jediná funkcia. Volala sa z (z anglického „zero“, čiže nula), nemala žiadne vstupy a na výstupe vracala číslo 0.
- Aj v druhom sáčku bola len jedna funkcia. Táto sa volala s (z anglického „successor“, čiže nasledovník). Mala jeden vstup a jeden výstup. Keď na vstup dostala číslo n, vrátila nám na výstupe číslo $n + 1$.
- Tretí sáčok bol o čosi plnší – bolo v ňom nekonečne veľa funkcií. Pre každé k a n (také, že $1 \leq k \leq n$) tam bola funkcia v_k^n („vyber k-ty z n vstupov“), ktorá mala n vstupov a na výstup vždy vrátila hodnotu, ktorú dostala na k-tom vstupe.

Na obrázku sú znázornené funkcie z, s a v_3^4.



Zvyšok balíčka tvorili dva prístroje, ktoré zjavne slúžia na výrobu nových funkcií. Na jednom z nich sa skvel nápis Kompozítor, na druhom zase Cyklovač. Každý z nich funguje tak, že do vnútra v správnom poradí vložíme nejaké funkcie, zatočíme kľukou a vypadne nám nová funkcia. Tá je vhodne poskladaná z funkcií, ktoré sme vložili dovnútra. Ale skôr ako si podrobnejšie popíšeme fungovanie Kompozítora a Cyklovača, potrebujeme si o naších funkciách povedať čosi formálnejšie.

V tejto úlohe považujeme nulu za prirodzené číslo. Prirodzené čísla sú teda pre nás množina $\mathbb{N} = \{0, 1, 2, 3, \dots\}$. Všetky funkcie, ktoré si budeme vyrábať, budú totálne funkcie na prirodzených číslach. Ako vstupy budeme teda funkcii dávať prirodzené čísla a pre každý možný vstup nám funkcia vráti na výstup jedno prirodzené číslo. Počet vstupov funkcie sa nazýva *arita*. Napr. funkcie s jedným vstupom odborne voláme *unárne*, funkcie s dvoma vstupmi *binárne*, atď. Funkcia v_2^7 má aritu 7. Funkcia z má aritu 0. Aritu občas budeme explicitne písať ako horný index. Mohli by sme teda napr. hovoriť, že v prvom sáčku bola funkcia z^0 a v druhom zas funkcia s^1. (U vyberacích funkcií v_k^n aritu musíme uvádzať vždy, keďže napr. v_1^4 a v_1^7 sú dve rôzne funkcie.)

Akonáhle už nejakú funkciu vyrobíme, máme navždy k dispozícii ľubovoľne veľa jej kópií. Špeciálne platí, že ak funkciu použijeme pri výrobe inej, zložitejšej, nestratíme ju tým.

Kompozítor

Kompozítor vie funkcie skladať (v bežnom matematickom zmysle tohto slova). Je však trochu háklivý na správne arity. Použitie Kompozítora sa skladá z nasledujúcich krokov:

1. Vyberieme si aritu $a \geq 0$ funkcie, ktorú chceme vyrobiť.
2. Zvolíme si funkciu g^b (čiže funkciu g s nejakou aritou b, možno inou ako a), ktorú použijeme v druhom kroku výpočtu.

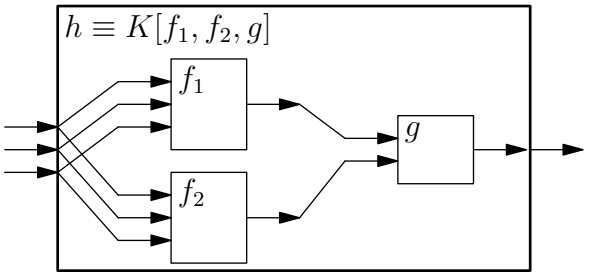
   Hodnota b musí byť kladná. (Inými slovami, funkcia g musí mať aspoň jeden vstup.)
3. Zvolíme si b funkcií $f_1^a, \dots, f_b^a$, ktoré použijeme v prvom kroku výpočtu.
4. Zatočíme kľukou na boku Kompozítora. Na výstupe nám vypadne nová funkcia h definovaná nasledujúcim pseudokódom:

```
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 )
```

   Slovne: Funkcia h zoberie a vstupov, ktoré dostala. Pomocou funkcií f_1 až f_b z nich vypočíta b pomocných hodnôt. No a na záver pomocou funkcie g vypočíta z pomocných hodnôt výstup celej funkcie h.

Na obrázku je graficky znázornená funkcia vyrobená Kompozítorom pre $a = 3$ a $b = 2$.



Cyklovač

Cyklovač vie vyrábať for-cykly. Aj on je však trochu háklivý na správne arity funkcií a poradie ich parametrov. Dajte si na ne pozor, keď ho budete kŕmiť. Správne použitie Cyklovača vyzerá nasledovne:

1. Vyberieme si aritu $a \geq 1$ funkcie, ktorú chceme vytvoriť.

   Prvý parameter tejto funkcie (označíme ho x) bude špeciálna premenná, ktorá hovorí, koľkokrát sa má for-cyklus vykonať. Ostatné parametre (označíme ich $y_1, \ldots, y_{a-1}$) budú ostávať nezmenené.

2. Zvolíme si funkciu f^{a-1}, ktorou výpočet cyklu začne.

3. Zvolíme si funkciu g^{a+1}, ktorá počíta, čo sa stane počas jednej iterácie cyklu. Funkcia g má až $a+1$ vstupov: všetky premenné, ktoré bude mať aj výsledná funkcia a navyše hodnotu ktorá bola výstupom predchádzajúcej iterácie cyklu. (Ak to nedávalo zmysel, pozri pseudokód uvedený v ďalšom kroku.)

4. Zatočíme kľukou na boku Cyklovača. Na výstupe nám vypadne nová funkcia h definovaná nasledujúcim pseudokódom:

   ```
   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
   ```

   Slovne: Výpočet začne tým, že funkciou f vypočítame (z ostatných parametrov) výstup pre $x = 0$. Z neho potom funkciou g vypočítame výstup pre $x = 1$, z toho znova funkciou g výstup pre $x = 2$, a tak ďalej až po požadovanú hodnotu prvého parametra.

Notácia

Rovnosť dvoch funkcií budeme značiť $\equiv$. Zápis $f \equiv g$ teda znamená, že funkcie f a g majú rovnakú aritu a na každom vstupe dávajú ten istý výstup.
Funkciu ktorá vznikne Kompozítorom z funkcií $f_1, \ldots, f_b$ a g budeme značiť $K[f_1, \ldots, f_b, g]$.
Funkciu ktorá vznikne Cyklovačom z funkcií f a g budeme značiť $C[f, g]$.

Príklady

Uf, to vyzerá komplikovane. Poďme sa preto spolu pozrieť na to, ako si Miška začala vyrábať nejaké nové funkcie.

Príklad 1. Čím by sme tak začali? Peknou jednoduchou funkciou je napríklad identita: unárna funkcia i taká, že pre každé n je $i(n) = n$. Vidíte, ako ju vyrobiť?
Toto bola triková otázka. Identitu vyrábať nepotrebujeme, dostali sme ju totiž v treťom sáčku. Identitou je totiž funkcia v_1^1. Môžeme teda písať $i \equiv v_1^1$.

Príklad 2. Vyrobme si funkciu j^0: konštantnú funkciu, ktorá nemá žiadny vstup a na výstupe vracia hodnotu 1. Túto funkciu si vieme vyrobiť Kompozítorom. V prvom kroku použijeme funkciu z, ktorá nemá žiaden vstup a na výstupe vráti 0. Túto 0 potom „pošleme ďalej“ do funkcie s, ktorá ju zväčší na 1.
Dostávame teda, že $j^0 \equiv K[z, s]$.

Príklad 3. Unárnu funkcia $plus3$, ktorá svoj jediný vstup zväčší o 3, vieme vyrobiť napríklad ako $K[K[s, s], s]$. Teda najskôr si vyrobíme funkciu $K[s, s]$, ktorá svoj vstup zväčší o 2, a túto potom opäť vložíme do Kompozítora s ďalšou funkciou s.

Príklad 4. Teraz si ukážeme, ako si Miška vie vyrobiť sčítanie – teda binárnu funkciu add takú, že $\forall x, y : add(x, y) = x + y$.
Základným pozorovaním je, že sčítanie je vlastne opakované použitie funkcie „+1“, teda nasledovníka. Výpočet $x + y$ si teda môžeme sformulovať nasledovne: „začni s hodnotou y a potom na ňu x-krát použi funkciu s“. No a keďže toto vyzerá ako cyklus, na výrobu sčítania budeme chcieť použiť Cyklovač.

Pozrime sa teda na pseudokód funkcie, ktorú vyrobí Cyklovač (s tým, že si ho už upravíme konkrétne na funkciu s dvoma vstupmi).

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

Aké funkcie f a g potrebujeme vhodiť do Cyklovača ak chceme dostať program pre sčítanie?
Funkcia f má jednoducho vrátiť hodnotu y, ktorú dostala na vstupe – čiže f má byť identita.
Funkcia g má v každej iterácii cyklu zobrať starú hodnotu (uloženú v premennej `tmp`) a inkrementovať ju použitím funkcie s. Potrebujeme teda funkciu s tromi vstupmi, ktorá prvé dva odignoruje, na tretí použije s a vráti výsledok. Takúto funkciu síce ešte nemáme, ale vieme si ju ľahko vyrobiť Kompozítorom: je to funkcia $K[v_3^3, s]$.
Dokopy teda dostávame, že $add \equiv C[\, v_1^1,\, K[v_3^3, s]\,]$.

Príklad 5. Ďalšou jednoduchou funkciou je unárna konštantná nula, teda funkcia zz^1, ktorá má jeden vstup a na výstupe vždy vracia nulu. (Formálne: $\forall n : zz^1(n) = 0$.)
Skôr, než si ukážeme, ako zz^1 vyrobiť, podotknime, že zz^1 a z^0 (čo je funkcia z, ktorú sme dostali v prvom sáčku) sú dve rôzne funkcie.
Zdalo by sa, že zz^1 musí ísť nejak vyrobiť zo z^0 pomocou Kompozítora. Lenže ako? Ak by sme chceli z^0 použiť v prvom kroku, tak bez ohľadu na to, akú funkciu použijeme v druhom kroku, určite vyrobíme funkciu s aritou 0. No a v druhom kroku z^0 použiť nesmieme, lebo funkcia použitá v druhom kroku musí mať kladnú aritu.
Cez Kompozítor teda cesta nevedie. Ukážeme si však, že zz^1 vieme vyrobiť pomocou Cyklovača. Opäť začnime tým, že sa pozrieme, ako to vyzerá, keď chceme pomocou Cyklovača vyrobiť unárnu funkciu. My dodáme funkcie f^0 a g^2 a Cyklovač nám z nich vyrobí funkciu h^1 definovanú nasledovne:

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

Ako máme zvoliť funkcie f a g, ak chceme, aby h na každom vstupe vracala nulu? Zjavne musíme zvoliť $f \equiv z^0$, aby bolo $h(0) = 0$. No a jedna možnosť ako teraz zvoliť funkciu g je jednoducho zobrať $g \equiv v_2^2$. Tým sa z príkazu `tmp = g(i,tmp)` stane príkaz `tmp = tmp`, a teda v premennej `tmp` ostane stále nula, bez ohľadu na hodnotu x.
Touto pomerne obskurnou konštrukciou sme si teda ukázali, že funkciu zz^1 vieme zostrojiť ako $C[z, v_2^2]$.

Predikáty. Predikát je odborné meno pre funkciu, ktorej návratová hodnota je logická hodnota: pravda alebo nepravda. Naše funkcie takéto hodnoty síce vracať nevedia, pomôžeme si ale rovnako ako voľakedy napríklad autori programovacieho jazyka C: prehlásime 0 za nepravdu a 1 za pravdu.
Funkciu teda budeme volať predikát vtedy, ak:

- táto funkcia na každom možnom vstupe vráti jednu z hodnôt 0 a 1
- chceme zdôrazniť, že na tieto hodnoty sa má zmysel dívať ako na logické hodnoty namiesto číselných

Príklady: Funkcia $eq(x, y)$, ktorá vráti 1 ak sa jej vstupy rovnajú a 0 inak, je predikát. Predikátom je aj funkcia $isprime(x)$ ktorá vráti hodnotu 1 ak x je prvočíslo a 0 inak. Na unárnu konštantnú funkciu zz^1 sa môžeme dívať ako na predikát, ktorý vždy vracia hodnotu „nepravda“.

TRIDSIATY TRETÍ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2018