Příloha časopisu
MATEMATIKA – FYZIKA – INFORMATIKA
Ročník 26 (2017), číslo 4

Úlohy I. kola (domácí část) 67. ročníku MO (kategorie P)

Úlohy P-I-1 a P-I-2 jsou praktické, vaším úkolem v nich je vytvořit a odladit efektivní program v jazyce Pascal, C nebo C++. Řešení těchto dvou úloh odevzdávejte ve formě zdrojového kódu přes webové rozhraní přístupné na stránce *http://mo.mff.cuni.cz/submit/,* kde také naleznete další informace. Odevzdaná řešení budou automaticky vyhodnocena pomocí připravených vstupních dat a výsledky vyhodnocení se dozvíte krátce po odevzdání. Pokud váš program nezíská plný počet 10 bodů, můžete své řešení opravit a znovu odevzdat.

Úlohy P-I-3 a P-I-4 jsou teoretické a skládají se z několika částí. Hodnocení jednotlivých částí je uvedeno přímo v zadání úlohy, celkem lze za každou úlohu získat až 10 bodů. Řešení úlohy P-I-3 bude obsahovat požadované výsledky, případně popis použitých algoritmů, zdůvodnění jejich správnosti a odhad časové a paměťové složitosti. Řešení úlohy P-I-4 bude popisovat, jak se sestrojí požadované funkce pomocí prostředků uvedených ve studijním textu. Řešení obou teoretických úloh odevzdávejte ve formě souboru typu PDF přes výše uvedené webové rozhraní.

Řešení všech úloh můžete odevzdávat do 15. listopadu 2017. Opravená řešení a seznam postupujících do krajského kola najdete na webových stránkách olympiády na adrese *http://mo.mff.cuni.cz/,* kde jsou také k dispozici další informace o kategorii P.

P-I-1 Trojice

Martin si vyrobil n kartiček a na každou napsal jedno přirozené číslo. Potom se postupně podíval na všechny možné (neuspořádané) trojice kartiček. Pro každou trojici spočítal součet jejich čísel a výsledek si zapsal na papír.

Soutěžní úloha

Je dáno přirozené číslo n a seznam čísel na kartičkách. Dále je dáno rozumně malé přirozené číslo k. Určete k-té nejmenší číslo zapsané na Martinově papíru – tedy k-tý nejmenší součet mezi všemi součty trojic čísel na kartičkách.

Pokud vznikne nějaký součet více různými způsoby, počítáme každý jeho výskyt zvlášť (viz první příklad).

Formát vstupu a výstupu

Program čte data ze standardního vstupu. Na prvním řádku vstupu jsou dvě přirozená čísla n a k. Na druhém řádku vstupu je n navzájem různých přirozených čísel $a_1, \ldots, a_n$. Čísla jsou uspořádána od nejmenšího k největšímu.

Program vypíše na standardní výstup jediný řádek obsahující jedno číslo: k-tý nejmenší mezi všemi součty trojic.

Hodnocení

Váš program bude testován na deseti sadách testovacích vstupů, za každou můžete získat 1 bod.

Ve všech sadách platí $3 \leq n \leq 100\,000$, $1 \leq k \leq 1\,000\,000$, $k \leq \binom{n}{3}$ a $1 \leq a_1 < a_2 < \cdots < a_n \leq 10^{12}$. Pozor, na uložení hodnot a_i použijte proměnné s dostatečným rozsahem!

V prvních třech sadách platí $n \leq 500$. V dalších třech sadách platí $n \leq 7\,000$.

Příklady

Vstup:	*Výstup:*
`5 4` `1 2 3 4 5`	`8`

Toto je všech 10 možných součtů: 6, 7, 8, 8, 9, 9, 10, 10, 11, 12. Pro $k = 3$ i pro $k = 4$ je tedy správným výstupem číslo 8. Pro $k = 9$ by bylo správným výstupem číslo 11.

Vstup:	*Výstup:*
`5 4` `1 10 100 1000 10000`	`1110`

Tentokrát jsou všechny součty trojic navzájem různé.

P-I-2 Telenovela

V televizi vysílají novou telenovelu. Jakub by ji chtěl sledovat, ale má hodně práce. Ještě že všechny díly ve vysílání tak často opakují. A tak

Jakub sedí, v jedné ruce má svůj diář, v druhé televizní program, a vymýšlí, na které díly se má vlastně dívat.

Při sledování telenovely platí následující zásady:

- Úplně na začátku je třeba vidět první díl, v němž se seznámíme s hlavními postavami.
- Není nutné vidět všechny díly. Všechno důležité se opakuje, takže můžeme libovolné množství dílů vynechat.
- Je ale důležité sledovat díly ve správném pořadí. Jakmile shlédnete nějaký díl telenovely mimo pořadí (tedy nejprve pozdější a potom teprve dřívější díl), už nikdy se nezorientujete v tom, co se vlastně kdy stalo.
- Žádný díl nechceme sledovat dvakrát.
- Úplně na konci je třeba vidět poslední díl, ve kterém přijde happyend.

Soutěžní úloha

Je dána posloupnost $a_1, \ldots, a_n$ čísel těch dílů telenovely, na jejichž vysílání by se Jakub mohl dívat. Některá čísla se mohou v posloupnosti opakovat (když má Jakub čas sledovat více vysílání téže epizody), některá mohou v posloupnosti chybět (když Jakub nemá čas sledovat ani jedno vysílání příslušné epizody). Zjistěte, zda může Jakub shlédnout telenovelu tak, aby dodržel výše uvedené zásady. Pokud ano, určete také, kolik nejvýše dílů může vidět.

Formát vstupu a výstupu

Program čte data ze standardního vstupu. Na prvním řádku vstupu jsou dvě celá čísla n (počet dílů, které Jakub 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, …, a_n: čísla epizod v chronologickém pořadí, jak po sobě následují ve vysílání.

Program vypíše na standardní výstup jediný řádek a na něm jedno číslo: maximální počet dílů, které může Jakub vidět při správném sledování telenovely. Jestliže telenovelu nemůže sledovat správným způsobem, program vypíše číslo -1.

Hodnocení

Váš program bude testován s deseti sadami testovacích vstupů, za každou můžete získat 1 bod.

Ve všech sadách platí $1 \leq n \leq 100\,000$, $2 \leq e \leq 10^9$ a pro každé i platí $1 \leq a_i \leq e$.

Různé sady mají různě velkou (postupně rostoucí) maximální hodnotu n. Speciálně v druhé sadě je $n = 10$, ve třetí sadě je $n = 14$, v páté sadě je $n = 100$, v sedmé sadě je $n = 7\,000$.

V sadách s lichým číslem platí, že v každém testovacím vstupu jsou všechna a_i navzájem různá.

Příklady

Vstup:	*Výstup:*
`5 50` `33 1 50 20 47`	`2`

Jakub se podívá na první a poté na poslední epizodu.

Vstup:	*Výstup:*
`8 50` `1 3 3 3 3 50 1 4`	`3`

Jakub se podívá na epizodu 1, na jedno ze čtyř vysílání epizody 3 a nakonec na epizodu 50.

Vstup:	*Výstup:*
`6 50` `47 4 6 3 5 1`	`-1`

Když se Jakub podívá na první epizodu, už nestihne žádnou další. Navíc nikdy neuvidí poslední epizodu (s číslem 50).

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

Z epizod 3 a 4 si Jakub vybere jednu, ostatní uvidí všechny.

P-I-3 Převracení prefixů

Ve všech částech této úlohy máme na začátku pole $A[1..n]$, v němž je uložena nějaká permutace čísel od 1 do n. Jinými slovy, každé z čísel od 1 do n se v poli A nachází právě jednou.

Obsah pole A můžeme měnit pouze pomocí operace *flip*(k). Hodnota k přitom musí být z rozmezí od 1 do n (včetně). Provedením této operace převrátíme pořadí prvních k prvků v poli A.

Např. z pole $A = (1, 2, 3, 5, 4)$ dostaneme operací *flip*(3) pole $(3, 2, 1, 5, 4)$.

Část A (2 body) Nalezněte algoritmus s polynomiální časovou složitostí, který libovolné vstupní pole A uspořádá.

Část B (2 body) David se rozhodl, že bude stále opakovat operaci *flip*$(A[1])$. V každém kroku se tedy podívá na první prvek pole A a následně převrátí pořadí tolika prvků, jakou hodnotu měl první prvek pole. Když tedy například na začátku pole bude číslo 4, provede *flip*(4).

Davidovi se zdá, že ať začne s jakýmkoliv polem, vždy se časem stane, že se na začátek pole dostane číslo 1. Tím samozřejmě všechna zábava končí, protože od tohoto okamžiku už bude David opakovat jenom operaci *flip*(1), která obsah pole nijak nemění.

Ověřte tuto hypotézu pro $n = 10$. Zjistěte, pro kterou z 10! počátečních permutací provede David nejvíce kroků, než se na začátek pole dostane jednička.

Část C (3 body) Pro $n = 47$ určete nějakou permutaci, pro kterou David provede velké množství kroků, než poprvé nastane $A[1] = 1$.

Za permutaci, pro kterou vykoná aspoň 500 kroků, dostanete 1 bod.

Za permutaci, pro kterou vykoná aspoň 650 kroků, dostanete 2 body.

Za permutaci, pro kterou vykoná aspoň 750 kroků, dostanete 3 body.

Část D (3 body) Platí skutečně, že se pro každou hodnotu n a pro každou permutaci čísel v poli A dostane časem na začátek pole A číslo 1? Dokažte toto tvrzení, nebo nalezněte protipříklad.

P-I-4 Stavebnice funkcí

K této úloze se vztahuje studijní text uvedený na následujících stranách. Doporučujeme vám nejprve prostudovat studijní text a až potom se vrátit k samotným soutěžním úlohám.

Jednotlivé části úlohy lze řešit v libovolném pořadí. Požadovanou funkci můžete sestrojit „ve více krocích“ – tedy nejprve si vytvořit nějaké jednodušší pomocné funkce.

Část A (2 body)

Sestrojte funkci zzz^3: funkci se třemi vstupy, která vždy vrátí na výstupu nulu. Formálně řečeno, musí platit

$$\forall a, b, c\colon zzz^3(a, b, c) = 0.$$

Část B (2 body)

Máte k dispozici nějakou funkci φ^4, ale nevíte nic o tom, co počítá. Sestrojte funkci ψ^3 definovanou následovně:

$$\forall a, b, c\colon \psi^3(a, b, c) = \varphi^4(b, a, c, a)$$

Část C (2 body)

Sestrojte násobení: binární funkci *mul* takovou, že pro všechna čísla a, b platí $mul(a, b) = a \cdot b$.

Část D (2 body)

Sestrojte předchůdce: unární funkci p takovou, že $p(0) = 0$ (neboť záporná čísla nemáme) a $\forall n\colon p(n + 1) = n$.

Část E (2 body)

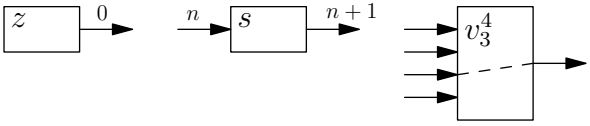
Sestrojte odčítání, resp. funkci, která se co nejvíce podobá odčítání. Přesně řečeno, sestrojte binární funkci *sub* takovou, že pro $x > y$ je $sub(x, y) = x - y$ a pro $x \leq y$ je $sub(x, y) = 0$.

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 $\mathbb{N} = \{0, 1, 2, 3, \dots\}$.

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:

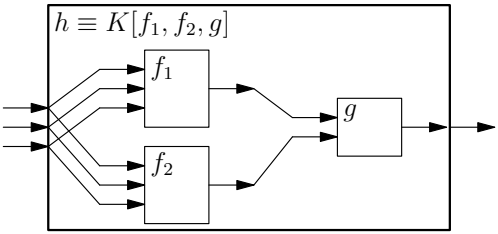
1. Zvolíme si aritu $a \geq 0$ funkce, kterou chceme vytvořit.
2. Zvolíme si funkci g^b (tedy funkci g s nějakou aritou b, třeba i jinou než a), kterou použijeme ve druhé fázi výpočtu. Hodnota b musí být kladná – jinými slovy, funkce g musí mít aspoň jeden vstup.
3. Zvolíme si b funkcí $f_1^a, \dots, f_b^a$, 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 $K[v_1^3, s]$.

Dohromady tedy dostáváme

$$add \equiv C\left[v_1^1,\, K[v_1^3, s]\right].$$

Příklad 5

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

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 $C[z, v_2^2]$.