67. ročník Matematické olympiády – 2017/2018

Úlohy ústředního kola kategorie P – 1. soutěžní den

Na řešení úloh máte 4,5 hodiny čistého času. Řešení každé úlohy pište na samostatný list papíru. Při soutěži je zakázáno používat jakékoliv pomůcky kromě psacích potřeb (tzn. knihy, kalkulačky, mobily, apod.).

Řešení každého příkladu musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, argumenty zdůvodňující jeho správnost, 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 povoleno odkazovat se na Vaše řešení předchozích kol, opravovatelé je nemají k dispozici; na autorská řešení se odkazovat můžete.
- **Zápis algoritmu.** V úlohách **P-III-1** a **P-III-2** je třeba uvést zápis algoritmu, a to buď ve tvaru zdrojového textu nejdůležitějších částí programu v jazyce Pascal nebo C/C++, nebo v nějakém dostatečně srozumitelném pseudokódu. Nemusíte detailně popisovat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů, vyhledávání v poli, třídění apod. V řešení úlohy **P-III-3** 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 (včetně zdůvodnění správnosti) a efektivita zvoleného algoritmu. Algoritmy posuzujeme zejména podle jejich časové složitosti, tzn. podle 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.

P-III-1 Kapitánka hledá posádku

Žilo jednou jedno děvčátko se spoustou dětských snů. Jednoho dne vyrostlo a rozhodlo se, že si jeden svůj sen splní a stane se pirátskou kapitánkou. Když už sehnala klobouk, pásku přes oko a loď, chybělo jí jediné: spolehlivá posádka. A tak nejprve zakotvila v přístavu Port Royal, kde chce nějakou posádku naverbovat.

Jakmile se rozkřiklo, že se verbuje nová posádka, sešlo se na molu mnoho mladých nadějných pirátů, kteří se všichni chtěli přihlásit. Kapitánka si je očíslovala od 1 do n v tom pořadí, jak stojí v řadě vedle sebe. O každém z nich si zapsala číslo p_i udávající, jak je to dobrý pirát (čím vyšší hodnota, tím lépe).

Kapitánka ví, že při výběru posádky jsou důležité dvě věci: musí to být dostatečně dobří piráti a musí zároveň tvořit dobrou partu. Naverbovat dobrou partu je snadné: piráti, kteří spolu dobře vycházejí, určitě stojí v řadě vedle sebe. Stačí tedy, když jako posádku vezme nějaký neprázdný souvislý úsek uchazečů.

Dobrá parta vám ale námořní bitvu nevyhraje a naverbovat dostatečně schopnou posádku vůbec není jednoduché. Důležitá je při tom tzv. Denisova hranice. Denis je známým archetypem jen-tak-tak použitelného piráta. Nic mu nejde nijak hvězdně, ale všechny podstatné pirátské úkoly nakonec nějak zvládne.

Při vyhodnocení toho, zda je posádka schopná plavby, je důležitý koncept tzv. *středního piráta:* kdyby se všichni piráti z posádky seřadili podle svých schopností, je to ten, který by stál uprostřed. Je-li pirátů sudý počet, středním pirátem je *lepší* z těch dvou, kteří stojí uprostřed pořadí.

Posádka je plavbyschopná právě tehdy, když je její střední pirát aspoň tak dobrým pirátem, jako je Denis.

Soutěžní úloha

V řadě stojí n pirátů očíslovaných od 1 do n v tom pořadí, jak stojí vedle sebe. Známe čísla $p_1, \ldots, p_n$, která o každém z nich udávají, jak dobrým je pirátem. Dále známe číslo d udávající totéž o Denisovi. (Samotný Denis není mezi uchazeči a nelze ho naverbovat do posádky.)

Kapitánka si chce vybrat *neprázdný souvislý úsek* uchazečů tak, aby dostala plavbyschopnou posádku. Jinými slovy, střední pirát ve vybraném úseku musí být aspoň tak dobrým pirátem, jako je Denis.

Zjistěte, kolika způsoby si může Kapitánka vybrat svoji posádku.

Formát vstupu a výstupu

Na prvním řádku vstupu jsou dvě celá čísla n a d. Na druhém řádku jsou celá čísla $p_1, \ldots, p_n$.

Na výstup vypište počet způsobů, jak lze vybrat plavbyschopnou posádku.

Omezení a hodnocení

Můžete předpokládat, že d i všechny hodnoty p_i se vejdou do běžných celočíselných proměnných.

Za postup, který efektivně vyřeší libovolný vstup s $n \leq 500$, můžete získat nejvýše 3 body.

Za postup, který efektivně vyřeší libovolný vstup s $n \leq 5000$, můžete získat nejvýše 5 bodů.

Za postup, který efektivně vyřeší libovolný vstup s $n \leq 100\,000$, můžete získat 8 až 10 bodů podle jeho přesné asymptotické časové složitosti.

Příklad

Vstup:	*Výstup:*
`4 30` `10 40 20 30`	`7`

Kapitánka má následující možnosti: vzít jako posádku pouze piráta 2, pouze piráta 4, libovolné dva sousední piráty, piráty 2+3+4, nebo všechny čtyři piráty.

Všimněte si, že kdyby vzala piráty 2+3+4, středním pirátem v posádce by nebyl pirát 3 (který nyní stojí uprostřed vybraného úseku), ale byl by to pirát 4. Pirát 4 bude středním pirátem také v případě, že Kapitánka vezme jako posádku všechny čtyři uchazeče.

P-III-2 Závaží

Jarka našla na půdě sadu n exotických závaží. Jednotlivá závaží mají hmotnosti $m_1, \ldots, m_n$. Měla radost, že si pomocí nich bude moci cokoliv zvážit. Samozřejmě, že místo jednoho závaží může Jarka použít také libovolnou jejich podmnožinu a odvážit tak i jiné hmotnosti.

Soutěžní úloha

Celkem existuje 2^n podmnožin závaží. Představme si, že jsme je všechny seřadili podle celkové hmotnosti a očíslovali jsme je, počínaje od jedné. Má-li více podmnožin stejnou hmotnost, můžeme je seřadit libovolně.

Pro dané n, jednotlivé hmotnosti závaží a zadané relativně malé číslo k určete hmotnost množiny s pořadovým číslem k.

Formát vstupu a výstupu

Na prvním řádku vstupu jsou dvě kladná celá čísla n a k. Na druhém řádku jsou *kladná* celá čísla $m_1, \ldots, m_n$.

Na výstup vypište hmotnost podmnožiny, která ve výše definovaném očíslování dostane číslo k.

Omezení a hodnocení

O číslech m_i předpokládejte, že jsou kladná a jejich součet se vejde do běžné celočíselné proměnné. Můžete také předpokládat, že hmotnosti závaží jsou na vstupu uspořádané, tedy že platí $m_1 \leq m_2 \leq \cdots \leq m_n$.

Za postup, který efektivně vyřeší libovolný vstup s $n \leq 20$, můžete získat 2 body.

Za postup, který efektivně vyřeší libovolný vstup s $n \leq 500$ a $k \leq 500$, můžete získat 5 bodů.

Za postup, který efektivně vyřeší libovolný vstup s $n \leq 5\,000$ a $k \leq 5\,000$, můžete získat 7 bodů.

Za postup, který efektivně vyřeší libovolný vstup s $n \leq 100\,000$ a $k \leq 100\,000$, můžete získat 10 bodů.

Příklady

Vstup:	*Výstup:*
`3 5` `1 1 2`	`2`

Jestliže si závaží označíme A, B a C, máme osm množin závaží:

$$\emptyset,\ \{A\},\ \{B\},\ \{C\},\ \{A,B\},\ \{A,C\},\ \{B,C\},\ \{A,B,C\}.$$

Jejich hmotnosti jsou 0, 1, 1, 2, 2, 3, 3, 4, uvedené pořadí je již uspořádáno podle hmotnosti. Pořadové číslo $k = 5$ tedy odpovídá hmotnosti 2.

Vstup:	*Výstup:*
`4 4` `1 10 100 1000`	`11`

Všimněte si, že dvouprvková množina obsahující první a druhé závaží je tentokrát lehčí než některé jednoprvkové množiny.

Vstup:	*Výstup:*
`10 512` `7 7 7 7 7 7 7 7 7 7`	`35`

P-III-3 Stavebnice funkcí

K této úloze se vztahuje studijní text uvedený na následujících stranách. Studijní text je shodný se studijním textem z krajského kola.

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

- Všechny funkce definované nebo sestrojené ve studijním textu.
- Všechny konstantní funkce libovolné arity. Konstantní funkci s a vstupy, která vždy vrátí hodnotu b, označujeme k_b^a.
- Funkce z domácího kola: *mul* (násobení), *p* (předchůdce) a *sub* (odčítání, které nepodteče pod nulu).
- Funkce z krajského kola: *pow* (umocnění), *sgn* (signum) a *geq* (predikát „větší nebo rovno“).
- Funkci *not* (logická negace) ze vzorového řešení krajského kola.

Konstrukci dříve sestrojených funkcí zbytečně nerozepisujte. Upozorňujeme ale, že konstrukce využívající myšlenku z poslední části soutěžní úlohy krajského kola (větvení, „if“) *je nutné rozepsat.*

Úkol A: *(2 body)* Sestrojte binární funkce *min* a *max* počítající minimum a maximum.

Úkol B: *(2 body)* Sestrojte unární funkci *last*, která vrátí poslední cifru čísla na vstupu. Formálně zapsáno, musí platit $\forall n : last(n) = n \bmod 10$.

Úkol C: *(3 body)* Sestrojte unární funkci *sqrt*, která spočítá dolní celou část odmocniny. Dolní celá část je největší celé číslo nepřevyšující přesný výsledek. Například $sqrt(99) = 9$ a $sqrt(100) = 10$.

Úkol D: *(3 body)* Fibonacciho čísla jsou definována rekurzívně: $F_0 = 0$, $F_1 = 1$ a $\forall n \geq 2 : F_n = F_{n-1} + F_{n-2}$. Sestrojte unární funkci *fib*, která pro vstup n vrátí na výstupu hodnotu F_n.

Abychom vám ušetřili práci, v části D nemusíte formálně rozepisovat použití Kompozitoru. Přesněji, kdykoliv chcete nějakou novou funkci vyrobit Kompozitorem z dříve vytvořených funkcí, stačí novou funkci korektně matematicky definovat. (Novou funkci samozřejmě musí být možné získat složením jiných funkcí, které již máte vytvořené, nemůžete si jen tak vyčarovat úplně nové funkce.)

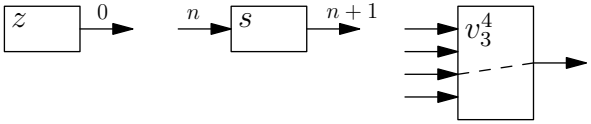
V řešení této části úlohy tedy můžete například napsat: „Z funkce *pow* si snadno vyrobíme funkci f takovou, že $\forall n : f(n) = (n + 4)^7$.“

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:

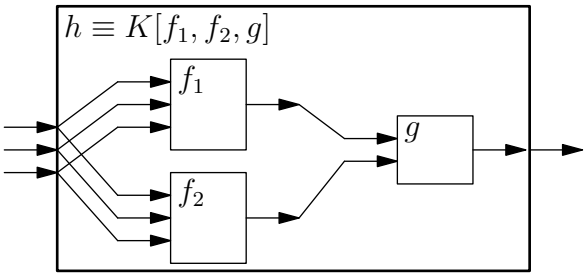
1. Zvolíme si aritu $a \geq 0$ funkce, kterou chceme vytvořit.
2. Zvolíme si funkci g^b (tedy funkci g s nějakou aritou b, třeba i jinou než a), kterou použijeme ve druhé fázi výpočtu. Hodnota b musí být kladná – jinými slovy, funkce g musí mít aspoň jeden vstup.
3. Zvolíme si b funkcí $f_1^a, \ldots,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