Priebeh krajského kola

Krajské kolo 33. ročníka Olympiády v informatike, kategória B, 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.

B-II-1 Návrat do špajze

Babička samozrejme prišla na to, že jej Peťa zo špajze vyjedá džem. Ale nehnevala sa, však kto iný by Peťku rozmaznával, ak nie ona. Dokonca sa ponúkla, že ju naučí piecť svoju preslávenú čerešňovú bublaninu. Peťa sa tomu nápadu veľmi potešila a navrhla babke, aby tie bublaniny upiekli rovno tri – jednu pre rodičov, druhú pre kamarátov a tretiu pre ňu.
Ako prvé bolo treba zo špajze doniesť čerešňový kompót. Babka preto poslala Peťu, aby doniesla tri poháre s rovnakým počtom čerešní – jednu do každého koláča. V špajzi však Peťa so zdesením zistila, že v každom pohári je iný počet čerešní.
Začala teda rozmýšľať, ako to vyrieši. Napadlo ju, že by mohla vybrať nejaký pohár s c čerešňami a k nemu nájsť dva ďalšie poháre, pričom v jednom bude o k čerešní viac a v druhom o k čerešní menej ako c. Potom ľahko preloží tých k čerešní naviac do pohára kde chýbajú, čím dostane tri poháre, každý po c čerešniach. Ktoré poháre však zvoliť?

Súťažná úloha

Petina babka má v špajzi n pohárov čerešní. V každom z nich **je iný počet čerešní**. Vašou úlohou je zistiť, koľko rôznych trojíc pohárov si môže Peťa vybrať. Hľadáte teda všetky trojice pohárov, v ktorých je c, $c+k$ a $c-k$ čerešní pre nejaké kladné čísla c a k. Pozor, c ani k nie je zadané, výsledok máte vrátiť pre všetky možné hodnoty c a k.

Formát vstupu a výstupu

Na prvom riadku vstupu je číslo n – počet pohárov s čerešňami.
Na druhom riadku vstupu je n navzájom rôznych celých kladných čísel: $a_1, a_2, \ldots, a_n$. Môžete predpokladať, že tieto čísla sa zmestia do bežných celočíselných premenných vo vašom programovacom jazyku.
Na výstup vypíšte jediné číslo – počet trojíc zadaných čísel, ktoré majú tvar $c-k$, c a $c+k$ pre nejaké c a k.

Obmedzenie a hodnotenie

Plný počet bodov môžu získať len riešenia, ktorých časová zložitosť (v najhoršom prípade) bude nanajvýš kvadratická od počtu pohárov. Inými slovami, ak na vstupe dostanete n čísel, tak váš program môže spraviť nanajvýš rádovo n^2 krokov výpočtu.
Riešenia, ktoré toto len tesne nespĺňajú, môžu získať nanajvýš 7 bodov. Takéto riešenie by si na bežnom počítači malo za sekundu poradiť s ľubovoľným vstupom v ktorom platí $n \leq 5000$.
Za ľubovoľné korektné riešenie môžete, bez ohľadu na časovú zložitosť, získať aspoň 3 body. Pomalé korektné riešenia spravidla získajú 3 až 4 body.

Príklad

vstup	výstup
8 2 3 15 4 6 7 13 9	4

Nasledovné trojice pohárov vyhovujú podmienkam zadania – $(2,3,4)$, $(2,4,6)$, $(3,6,9)$ *a* $(3,9,15)$.

B-II-2 Zoznamovačka

Na začiatku každého tábora či sústredenia je dôležité, aby sa účastníci navzájom spoznali. Predsa len spolu strávia celý týždeň a oslovovať sa „Hej ty tam!“ je mierne nepraktické. Preto sa na úvod sústredenia zaraďuje takzvaná zoznamovačka – hra, ktorej cieľom je, aby sa účastníci navzájom spoznali.
Zoznamovačka samozrejme nemá presne určený formát a organizátori s ňou môžu rôzne experimentovať. Na poslednom sústredení Korešpondenčného seminára z programovania to skúsili nasledovne. V chate vyhradili jednu špeciálnu miestnosť, kde sa účastníci mohli zoznamovať. Keď účastník prišiel na sústredenie, vošiel do tejto miestnosti, chvíľu sa venoval aktivite, ktorá sa vnútri odohrávala, a potom sa išiel vybaľovať na izbu. Počas aktivity účastník spoznal všetkých ostatných ľudí, ktorí aspoň chvíľu boli v miestnosti spolu s ním.
Hlavný vedúci sústredenia si ku každému účastníkovi poznačil, kedy do miestnosti vošiel a kedy z nej vyšiel. Teraz by ich zaujímalo, koľko iných účastníkov každý z nich pri tejto hre spoznal.

Súťažná úloha

Na sústredenie prišlo n účastníkov. Pre každého z nich poznáme čas z_i, v ktorom vošiel do zoznamovacej miestnosti a čas k_i, v ktorom z tejto miestnosti vyšiel. Dvaja účastníci sa spoznali, ak sa stretli v tejto miestnosti, teda ak existuje moment, v ktorom v nej boli obaja. Pre jednoduchosť môžete predpokladať, že všetky čísla z_i a k_i sú rôzne a teda sa nestalo, že nejaký účastník do miestnosti vošiel v tom istom okamihu ako z nej iný vychádzal.
Pre každého účastníka spočítajte koľko iných účastníkov spoznal počas tejto zoznamovačky.

Formát vstupu a výstupu

Na prvom riadku vstupu je číslo n – počet účastníkov, ktorý prišli na sústredenie.
Nasleduje n riadkov, každý z nich obsahuje dve kladné celé čísla z_i a k_i – čas vstupu a výstupu z miestnosti. Pre každé i platí $z_i < k_i$. Všetkých $2n$ čísel je navzájom rôznych.
Účastníkov číslujeme od 1 po n v poradí, v akom sa objavili na vstupe.
Na výstup vypíšte n čísel, každé na samostatný riadok. Číslo v i-tom riadku je počet účastníkov, ktoré počas zoznamovačky spoznal účastník i.

Obmedzenie a hodnotenie

Na získanie plného počtu bodov musí vaše riešenie na bežnom počítači vyriešiť za sekundu ľubovoľný vstup kde $n \leq 100\,000$ a $z_i, k_i \leq 10^9$.
Za riešenie, v ktorom budete predpokladať, že $n \leq 100\,000$ a $z_i, k_i \leq 10^6$, môžete získať až 8 bodov.
Až 5 bodov môžete získať za riešenie, ktoré zvládne vstupy, kde $n \leq 5\,000$ a $z_i, k_i \leq 10^9$.
Za ľubovoľné korektné riešenie môžete, bez ohľadu na časovú zložitosť, získať aspoň 3 body.

Príklad

vstup

```
5
4 16
8 12
2 5
9 20
21 22
```

výstup

```
3
2
1
2
0
```

Ako prvý vošiel do miestnosti účastník číslo 3. Po ňom vošiel účastník 1 a navzájom sa spoznali. Účastník 3 následne odišiel, spoznal teda iba jediného účastníka.
Kým bol v miestnosti účastník 1, vošli tam ešte účastníci 2 a 4. Všetci traja sa navzájom spoznali a postupne z miestnosti odišli. Účastníci 2 a 4 teda spoznali dvoch iných ľudí, účastník 1 ich vďaka účastníkovi 3 spoznal až troch.
Nakoniec vošiel do miestnosti účastník 5, kým však bol v miestnosti, nikto iný tam nebol a preto nikoho nespoznal.

B-II-3 Stolná hra

Jonáš je fanúšik všemožných hier a hlavolamov. Jeho kamarátov už omrzelo neustále s ním prehrávať, a tak Jonáša veľmi potešilo, keď na Vianoce dostal hru pre jedného hráča.
Na hracom pláne tejto hry je n políčok očíslovaných od 1 po n. Počas prípravy hry Jonáš na každé políčko položí kartičku a na tú napíše nejaké číslo od 1 po n. (Môže pritom niektoré čísla nikdy nepoužiť a iné napísať viackrát.) Potom Jonáš na každé políčko položí jednu figúrku: na políčko s číslom i položí figúrku s číslom i. Okamih, kedy už sú všetky figúrky na svojich miestach, nazveme čas 0.
Hra prebieha v taktoch. Každý takt zodpovedá jednotke času. V každom takte sa všetky figúrky **naraz presunú**: každá figúrka skočí na políčko, ktorého číslo je napísané na kartičke, pri ktorej práve stojí. Ak je napríklad na políčku číslo 3 položená kartička s číslom 1, znamená to, že všetky figúrky, ktoré sú v čase i na políčku 3 budú v čase $i+1$ na políčku 1. Na každom políčku pritom môže naraz byť ľubovoľne veľa figúrok.
Celé prázdniny Jonáš sedel za stolom a hral sa. Nech však kartičky rozostavil akokoľvek, stále mal pocit, že hra sa začala po nejakom čase opakovať. Je však tento stav opakovania nutný? A ak áno, kedy nastane?

Podúloha A (4 body)

Dokážte, že pre ľubovoľné začiatočné rozloženie Jonášovej hry sa hra časom začne dookola pravidelne opakovať. (Nezabudnite, že figúrky sú očíslované a teda vieme rozlíšiť jednu od druhej.)

Podúloha B (6 bodov)

Na vstupe dostanete popis kartičiek v Jonášovej hre. Pre každé z políčok 1 až n teda dostanete číslo p_i ($1 \leq p_i \leq n$), ktoré označuje, že z políčka s číslom i sa figúrka posunie na políčko s číslom p_i.
Navrhnite algoritmus, ktorý vypočíta najmenšie t také, že v čase t bude rozloženie figúrok presne rovnaké ako v nejakom skoršom čase t'. (Môžete predpokladať, že sa vám hodnota t zmestí do bežnej celočíselnej premennej.)

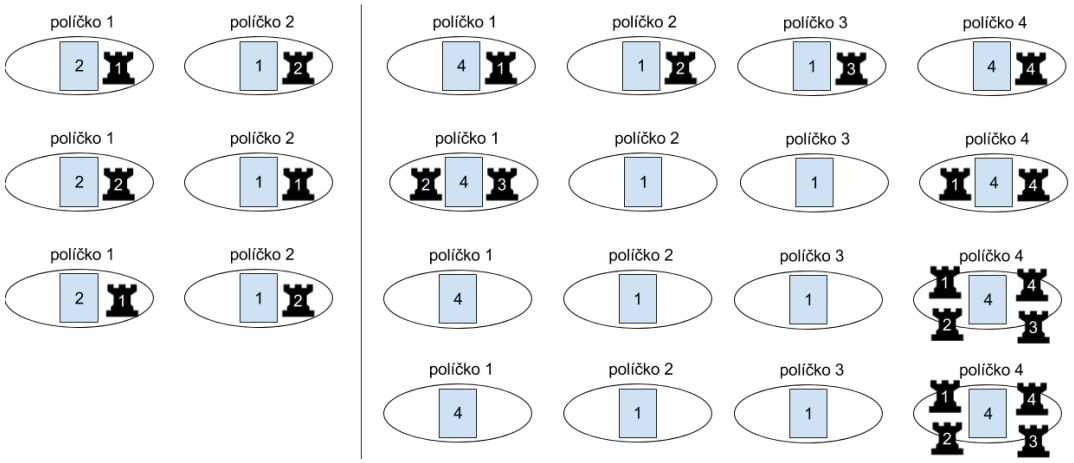
Na získanie plného počtu bodov za túto podúlohu musí vaše riešenie na bežnom počítači vyriešiť za sekundu ľubovoľný vstup kde $n \leq 1\,000\,000$.
Za riešenie, v ktorom budete predpokladať, že $n \leq 5\,000$, môžete získať až 8 bodov.
Za ľubovoľné korektné riešenie môžete, bez ohľadu na časovú zložitosť, získať aspoň 3 body.

Príklady k podúlohe B

vstup	výstup
2 2 1	2

vstup	výstup
4 4 1 1 4	3

Na obrázku nižšie sú znázornené oba vzorové príklady. Riadky postupne zodpovedajú času 0, 1, 2 a 3. Čísla v obdĺžnikoch sú kartičky, ktoré udávajú pohyb figúrok. V prvom príklade bude situácia v čase 2 rovnaká ako na začiatku (t.j. v čase 0). V druhom príklade bude v čase 3 rozmiestnenie figúrok rovnaké ako v čase 2.



B-II-4 Reálne čísla

Táto súťažná úloha nadväzuje na úlohu z domáceho kola. Podstatné časti textu zadania domáceho kola nájdete pod zadaním súťažnej úlohy.

Súťažná úloha

Podúloha A (2 body): Ako vyzerá číslo jedna sedmina zapísané v dvojkovej sústave?

Podúloha B (2 body): Aká hodnota bude naozaj uložená v premennej x typu single, keď sa do nej pokúsime uložiť presnú hodnotu jednej sedminy? (Odpoveď stačí uviesť v dvojkovej sústave.)

Podúloha C (6 bodov): Ďalej uvažujme nasledujúci jednoduchý program: Zoberieme premennú y typu single, nastavíme ju na 0 a potom premennú y zvyšujeme o 1 až kým nenastane zdanlivo nemožná podmienka: $y = y+1$. Príklad programu v jazyku C++:

```
float y = 0;
while (y != y+1) y += 1;
```

Aj keď to tak vyzerá, nejde o nekonečný cyklus. Ale prečo? Vysvetlite, prečo tento program nutne skončí po konečnom počte krokov. Odhadnite, aké číslo bude v premennej y, keď program skončí. Zdôvodnite svoj odhad.

Nasleduje text z domáceho kola o ukladaní reálnych čísel vo formáte s plávajúcou bodkou. Tento formát funguje podobne, ako keď číslo 1 234 567 zapíšeme v tvare 1.234567×10^6. Samozrejme, počítače nepoužívajú desiatkovú sústavu ale dvojkovú – teda jediné dve cifry sú 0 a 1. V nasledujúcom texte preto budeme občas aj my používať dvojkovú sústavu. Na miestach, kde nebude jasné, ktoré čísla sú v dvojkovej a ktoré v desiatkovej sústave, použijeme dolné indexy. Napr. 11_{10} je jedenásť, zatiaľ čo 11_2 je tri.

To, že počítače používajú dvojkovú sústavu namiesto desiatkovej, ich nijak neobmedzuje. Totiž podobne ako v desiatkovej sústave, aj v dvojkovej vieme zapisovať aj necelé čísla: 0.1_2 je jedna polovica, 0.01_2 je jedna štvrtina, a tak ďalej. Teda napríklad 0.1001_2 je polovica plus šestnástina, čiže 0.5625_{10}.
Samozrejme, niektoré reálne čísla majú konečný zápis, zatiaľ čo iné nie. Všetky iracionálne čísla, ako napríklad π, e, alebo $\sqrt{2}$, majú aj v dvojkovej sústave nekonečný zápis, ktorý navyše nie je periodický. Ako sme už videli vyššie, niektoré racionálne čísla majú v dvojkovej sústave konečný zápis. Ale nie všetky. Napríklad jedna tretina v dvojkovej sústave je $0.\overline{01} = 0.01010101\ldots$
Zaujímavé je tiež, že každé racionálne číslo, ktoré má konečný zápis v dvojkovej sústave, má konečný zápis aj v desiatkovej sústave. (Viete, prečo je to tak? Skúste si to dokázať.) Naopak to však neplatí. Napríklad jedna pätina je v desiatkovej sústave 0.2, zatiaľ čo v dvojkovej je to $0.\overline{0011}$.

V desiatkovej sústave niekedy používame tzv. vedecký zápis reálnych čísel. Pri tomto zápise posunieme desatinnú bodku za najvýznamnejšiu cifru čísla a zvlášť uvedieme, akou mocninou 10 treba toto číslo vynásobiť. Vyššie sme si už spomínali jeden príklad: číslo 1 234 567 zapíšeme ako 1.234567×10^6. Číslo 0.0047 vieme zase zapísať ako 4.7×10^{-3}.
Toto isté ideme teraz spraviť v dvojkovej sústave. Tam je to ešte o čosi jednoduchšie – totiž v každom čísle okrem nuly má jeho najvýznamnejšia cifra hodnotu 1. Každé číslo okrem nuly teda vieme v dvojkovej sústave zapísať v nasledovnom tvare: $\pm 1.m \times 2^e$.
V tomto zápise m (nazývané *mantisa*) predstavuje ľubovoľnú, konečnú alebo aj nekonečnú, postupnosť cifier, zatiaľ čo e (nazývané *exponent*) je nejaké celé, možno aj záporné číslo.
Napríklad uvažujme číslo $x = 5.125_{10}$. Jeho klasický zápis v dvojkovej sústave je $x = 101.001_2$. Toto isté číslo ale vieme zapísať aj nasledovne: $x = 1.01001 \times 2^2$. Pre toto x teda máme mantisu 01001 a exponent 2.
Číslo $y = -0.125_{10} = -0.001_2$ v tomto tvare zapíšeme nasledovne: $y = -1. \times 2^{-3}$.

Keď sa reálne čísla ukladajú do pamäte počítača vo formáte s plávajúcou bodkou, používame presne tento formát, s jedinou zmenou: pamäť počítača je bohužiaľ konečná. Preto v tomto formáte nevieme reprezentovať ani zďaleka všetky reálne čísla. Presne vieme uložiť len tie, ktoré majú v dvojkovej sústave konečný a dostatočne

krátky zápis (t.j. krátku mantisu) a zároveň dostatočne malý exponent. Ak chceme uložiť nejaké iné číslo, máme smolu. Najlepšie, čo vieme spraviť (a čo sa aj v skutočnosti v počítači stane) je, že nastane *zaokrúhľovacia chyba* a namiesto skutočnej hodnoty si počítač zapamätá najbližšiu hodnotu, ktorú skutočne uložiť vie. Inými slovami, zapamätá si z mantisy len toľko *najvýznamnejších bitov*, koľko sa mu zmestí do pamäte.
(Drobný technický detail: Ak sa presná ukladaná hodnota nachádza presne v strede medzi dvoma hodnotami, ktoré si vieme v pamäti počítača reprezentovať, takúto hodnotu vždy zaokrúhlime na „okrúhlejšie“ číslo – inými slovami, vyberie sa tá možnosť, pri ktorej uložená mantisa končí nulou.)
Ak si teda v takomto formáte uložíme hodnotu $4/3 = 1.\overline{01}$, nebude v pamäti počítača uložená presná hodnota štyroch tretín, ale nejaké číslo, ktoré sa od nich o maličkú trošku líší. Ak by sme napríklad v pamäti mali miesto len na sedem bitov mantisy, uložili by sme si namiesto presných štyroch tretín číslo 1.0101011×2^0. Všimnite si, že uložená mantisa (0101011) končí cifrou 1, keďže toto číslo je ku štyrom tretinám bližšie ako 1.0101010×2^0.

Dva najbežnejšie používané verzie formátu s plávajúcou bodkou sa volajú *single precision* a *double precision* (stručne len *single* a *double*, voľne preložené: obyčajná a dvojnásobná presnosť). Číslo uložené vo formáte single zaberá 4 bajty pamäte, a to nasledovne: 1 bit je znamienko (0 je plus, 1 je mínus), potom 8 bitov exponent, a na záver 23 bitov tvorí mantisa. Číslo vo formáte double zaberá dvakrát toľko pamäte, teda 8 bajtov. Opäť je tam jeden bit na znamienko, potom máme 11-bitový exponent a na záver 52-bitovú mantisu.
Bežné formáty „reálnych“ čísel v programovacích jazykoch zvyknú zodpovedať týmto dvom verziám: single precision je napríklad `float` v Jave aj C++ a `single` vo FreePascale, double precision je napríklad `double` v Jave, C++, aj FreePascale a `float` v Pythone.
Pri ukladaní čísla v tvare $(\pm 1.m \times 2^e)$ ukladáme ako mantisu len m. Jednotku pred ňou ukladať netreba, keďže tú majú všetky čísla rovnakú. Ak teda máme 23-bitovú mantisu, znamená to, že si do pamäte uložíme *24 najvýznamnejších bitov* ukladaného čísla. A keďže $2^{24} > 10^7$, neformálne to znamená, že v desiatkovej sústave by sa nám aj napriek zaokrúhľovacím chybám malo zachovať aspoň sedem najvýznamnejších cifier.
Príklad: číslo $z = 1.234567890_{10}$ má v dvojkovej sústave rozvoj $z = 1.00111100000011001010010000101\ldots_2$. Ak by sme z uložili do premennej typu single, počítač by si zapamätal len prvých 24 bitov tohto čísla. Skutočná hodnota uložená v pamäti by teda nebola presne z. (Pre zaujímavosť, bola by to presne hodnota $z' = 1.2345678806304931640625$. Všimnite si, o koľko sa z a z' líšia.)
Pre formát double je 2^{53} o čosi menej ako 10^{16}. Ak teda reálne číslo reprezentujeme vo formáte double, chyba sa pohybuje na úrovni zhruba sedemnástej cifry v jeho desiatkovom zápise.

Pozorného čitateľa už isto nejakú dobu trápi otázka čo s nulou. Nulu totiž nevieme zapísať v tvare $(\pm 1.m \times 2^e)$, a teda ju je nutné ošetriť ako špeciálny prípad. Konkrétne riešenie otázky ako uložiť nulu je prekvapivo zložité. V skutočnosti totiž nula nie je jediným špeciálnym prípadom. Pri návrhu formátov single a double sa totiž ich autori rozhodli pridať aj ďalšie špeciálne hodnoty: plus aj mínus nekonečno, malé hodnoty veľmi blízke nule, aj špeciálne hodnoty predstavujúce, že výsledok operácie nie je číslo. (Napríklad pokus v reálnych číslach odmocniť -1 vedie k takejto hodnote.) Pre účely tejto úlohy nepotrebujete vedieť technické detaily, postačí poznamenať, že všetky tieto špeciálne prípady sa riešia tak, že sú pre ne rezervované dve konkrétne hodnoty exponentu.

Zaokrúhľovacie chyby vznikajú aj pri aritmetických operáciách. Aj číslo 10, aj číslo 3 vieme uložiť presne, ale keď vypočítame ich podiel, dostaneme číslo 10/3, ktoré už presne uložiť nevieme. A aj ak máte doma hocijakú jednoduchú starú kalkulačku, môžete si vyskúšať, že keď 10 vydelíme a následne vynásobíme tromi, na displeji sa nám namiesto desiatky zjaví 9.9999999.
Iný príklad: predstavme si, že človek, ktorý si vie pamätať len sedem cifier z každého čísla, dostane za úlohu sčítať čísla 1 000 000 a 1 234.567. Čo dostane ako výsledok? Namiesto presného 1 001 234.567 si zvládne zapamätať iba zaokrúhlenú hodnotu 1 001 235. A presne to isté sa stane pri aritmetických operáciách v počítači: môže sa stať, že výsledok operácie nebudeme vedieť uložiť presne, a teda jeho zaokrúhlením vznikne ďalšia chyba.

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