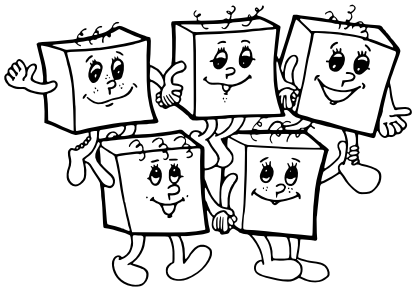
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH

http://oi.sk/



tridsiaty štvrtý ročník
školský rok 2018/2019

zadania celoštátneho kola, deň 1
kategória A

Priebeh celoštátneho kola

Celoštátne kolo 34. ročníka Olympiády v informatike, kategórie A, sa koná v dňoch 27.-30. marca 2019. Na riešenie úloh prvého, teoretického dňa majú súťažiaci 4,5 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.
- 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 prvého (teoretického) dňa

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 úlohy môžu byť uvedené limity na veľkosť premenných. Tieto môžete použiť na odhad toho, ako dobré vaše riešenie je. Na počítači, ktorý vykoná miliardu inštrukcií za sekundu, vyrieši vzorové riešenie ľubovoľný povolený vstup nanajvýš za niekoľko sekúnd.

A-III-1 ... a princeznú za ženu

Keď Janko zabil zlého draka, jednou z odmien, ktoré dostal, bola tradičná polovica kráľovstva.
Janko dobre vie, čo v kráľovstve prináša hodnotu (do kráľovskej pokladnice): krčmy. A taktiež vie, čo hodnotu odnáša: uhoľné bane, ktoré sa starý kráľ nezmyselne zaviazal dotovať ešte bohvie koľko rokov. Pre jednoduchosť predpokladajme, že profit z ľubovoľnej jednej krčmy je rovný dotácii pre ľubovoľnú jednu baňu.
Kráľovstvo je rovina a krčmy aj bane sú body v nej. Pre jednoduchosť môžete predpokladať, že **žiadne tri neležia na priamke**.
Janko si pol kráľovstva vyberie tak, že si zvolí dva rôzne body P a Q a od starého kráľa dostane všetko, čo leží ostro naľavo od (orientovanej) priamky PQ. Poraďte mu, ktorú priamku si má zvoliť, ak chce maximalizovať svoj profit.

Formát vstupu a výstupu

V prvom riadku vstupu je počet bodov n. V každom z ďalších n riadkov je popis jedného bodu: jeho súradnice x_i, y_i a jeho typ. Typ je buď `K` (krčma) alebo `B` (baňa).
Na výstup vypíšte štyri reálne čísla: najskôr súradnice zvoleného bodu P a potom súradnice zvoleného bodu Q. Janko dostane všetko, čo leží v polrovine, ktorú vidíme naľavo, keď stojíme na bode P a hľadíme na bod Q. (Ak priamka PQ prechádza niektorými z daných n bodov, tieto Janko nedostane.)

Obmedzenia a hodnotenie

Všetky súradnice na vstupe sú rozumne malé celé čísla. Pri písaní programu môžete predpokladať, že výpočty s nimi sú presné a že sa medzivýsledky zmestia do bežných premenných. (Netreba teda napr. uvažovať a ošetrovať zaokrúhľovacie chyby.)
Plný počet bodov získa korektné riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 5\,000$.

Príklad

vstup

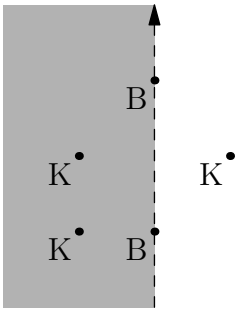
```
5
0 0 K
0 1 K
1 0 B
1 2 B
2 1 K
```

výstup

```
1.0 -1.0 1.0 3.0
```

Príklad vstupu a výstupu je znázornený na obrázku vpravo. Priamka popísaná v príklade výstupu prechádza oboma baňami. Janko teda dostane dve krčmy a žiadnu baňu. Toto je zjavne optimálne: získať tri krčmy a žiadnu baňu sa nedá.

Existujú aj iné optimálne riešenia. Janko by si mohol napríklad vybrať body $P = (1, 1.2)$ a $Q = (0, 1.1)$, dostal by tri krčmy a jednu baňu.



A-III-2 Podpostupnosť

Reťazec S voláme podpostupnosťou reťazca T, ak platí, že keby bol T napísaný na tabuli, tak by sme vedeli zmazať nejaké (možno žiadne, možno všetky) písmená T tak, aby sme dostali S.
Napríklad reťazec „`zaba`“ je podpostupnosťou reťazca „`koziabrada`“, ale nie je podpostupnosťou „`bazar`“.

V škole na tabuli je momentálne text T. Móricko sa už teší, ako po hodine časť textu zotrie a vyrobí tak žartovnú správu Z. Učiteľ však ešte rozpráva. A počas toho občas na tabuli nejaké miesta zotrie a napíše tam niečo iné. Občas tým Móricka poteší (lebo sa z textu na tabuli bude dať vyrobiť Z), občas ho zase sklame.

Súťažná úloha

Dostanete dlhý text T a krátku žartovnú správu Z. Zistite, či je Z podpostupnosťou textu T.
Následne dostanete postupnosť zmien, ktoré robil učiteľ. Každá zmena má tvar „písmeno na indexe a_i sa zmenilo na p_i“. Zmeny sú inkrementálne, teda každú aplikujeme na text ktorý vznikol predchádzajúcou zmenou.
Po každej zmene opäť zistite, či sa správa Z (stále tá istá) dá vyrobiť z aktuálneho textu.

Formát vstupu a výstupu

V prvom riadku vstupu je reťazec T. V druhom riadku je reťazec Z. V treťom riadku je číslo n: počet postupných zmien. Zvyšok vstupu tvorí n riadkov, i-ty z nich obsahuje index a_i a písmenko p_i. (Pozície v T indexujeme od nuly.)
Na výstup vypíšte $n + 1$ riadkov: jeden po prečítaní T a Z a jeden po každej zmene. Zakaždým vypíšte „`ano`“ alebo „`nie`“ podľa toho, či v danej chvíli bol alebo nebol reťazec Z podpostupnosťou reťazca na tabuli.

Obmedzenia a hodnotenie

Pri implementácii predpokladajte, že všetky písmená vo všetkých reťazcoch sú malé písmená anglickej abecedy. Pri odhade časovej a pamäťovej zložitosti počet rôznych písmen označte s. (Anglická abeceda má $s = 26$.) Dĺžky reťazcov T a Z označíme t a z.
Plný počet bodov dostanete za riešenie, ktoré si efektívne poradí s ľubovoľným vstupom, v ktorom platí $z \leq 100$, $t \leq 1\,000\,000$ a $n \leq 100\,000$.

Príklady

vstup

```
koziabrada
zaba
4
7 z
2 n
3 z
3 z
```

výstup

```
ano
ano
nie
ano
ano
```

Ako sme už spomenuli v zadaní, `zaba` *je podpostupnosťou* `koziabrada`*. Prvou zmenou sme vyrobili na tabuli text* `koziabrzda`*, ktorý stále obsahuje podpostupnosť* `zaba`*. Druhou zmenou vznikne reťazec* `koniabrzda`*, ktorý už podreťazec* `zaba` *neobsahuje. Tretia zmena zmení text na tabuli na* `konzabrzda` *a z tohto textu má Móricko opäť radosť. Štvrtá zmena v skutočnosti nič nezmení, text na tabuli aj odpoveď sú rovnaké ako po tretej zmene.*

A-III-3 Online algoritmy 3

Hráme sa hru, pri ktorej zbierame kartičky. Každá kartička má na sebe jeden druh ovocia: Ananás, Banán, Citrón, Dyňu alebo Egreš. Na začiatku hry už máme **po dve kartičky** z každého druhu. Samotná hra potom pozostáva z vopred neznámeho počtu kôl. V každom kole nám súper ponúkne dve **rôzne** kartičky a my si z nich jednu vezmeme. Naše skóre na konci hry je rovné najväčšiemu počtu kartičiek rovnakého typu v našej zbierke.

Príklad: V prvom kole nám súper ponúkol kartičky A a B, my sme si vybrali A. V druhom kole nám opäť ponúkol A a B a my sme si opäť vybrali A. V treťom kole nám ponúkol B a C, my sme si vybrali B. Potom súper ohlásil koniec hry. Máme 4 ananásy, 3 banány, 2 citróny, 2 dyne a 2 egreše, čiže naše skóre je 4. Všimnite si, že keby sme dopredu poznali všetky dvojice, ktoré nám súper ponúkne, dalo sa získať skóre 5.

Pri maximalizačných problémoch počítame kompetitívny pomer opačne ako pri minimalizačných. Napr. 3-kompetitívna stratégia pre túto hru je taká stratégia, ktorá vždy zaručene nahrá aspoň tretinu maximálneho skóre, ktoré sa dalo nahrať, keby sme vopred presne vedeli, aké dvojice nám súper ukáže.

Podúloha A (1 bod): Dokážte, že úplne každá možná stratégia pre túto hru je 7-kompetitívna alebo lepšia.
Podúloha B (5 bodov): Nájdite stratégiu, ktorá bude 2-kompetitívna, a dokážte to o nej.
Podúloha C (4 body): Dokážte, že neexistuje c-kompetitívna stratégia pre $c < 2$.

Študijný text: online algoritmy

V úlohách našej olympiády sa zväčša stretnete s úlohami, v ktorých dostanete naraz celý vstup a z neho máte vypočítať optimálny výstup. V praxi sa však občas vstup dozvedáme postupne a náš algoritmus musí už priebežne robiť nejaké rozhodnutia. Takýmto problémom hovoríme *online* problémy.

Príklad: Stránkovanie

Počítače majú len obmedzené množstvo fyzickej pamäte. Aby na nich mohlo bežať viac procesov naraz, rieši sa to tak, že majú väčšie množstvo pamäte virtuálnej. Tej si každý program alokuje, koľko potrebuje. V prípade, že programy spotrebujú virtuálnej pamäte viac ako je dostupnej fyzickej, prichádza následne k *swapovaniu*: vždy, keď nejaký program potrebuje pristupovať ku svojej pamäti, presunieme príslušný kus údajov do fyzickej pamäte. Ak je tá ale už plná, musíme najskôr uvoľniť nejaké miesto v nej.
My sa podrobnejšie pozrieme na jeden konkrétny spôsob, ako robiť takúto správu pamäte. Pôjde o tzv. *stránkovanie*. Fyzickú pamäť (napr. RAM) si rozdelíme na k rovnakých kusov nazývaných *stránky*. Hodnotu k budeme považovať za pevne zvolenú konštantu. Vo virtuálnej pamäti (napr. na pevnom disku) máme priestor pre ľubovoľne veľa stránok. Stránky vo virtuálnej pamäti si označíme prirodzenými číslami.
Naša úloha teraz bude vyzerať nasledovne: Predpokladajme, že na začiatku je celá fyzická pamäť prázdna. Postupne nám budú chodiť požiadavky. Každá požiadavka bude tvaru „virtuálnu stránku a_i treba dať do fyzickej pamäte“. Ak tam táto stránka momentálne už je, nemusíme robiť nič, ak tam nie je, musíme na ňu spraviť miesto (ak už je fyzická pamäť plná, niektorú stránku z nej odswapovať na disk) a následne ju tam nahrať. Toto je pomalé, preto by sme radi **minimalizovali počet nahratí stránky do fyzickej pamäte**.
Príklad: Uvažujme situáciu v ktorej máme len $k = 2$ stránky fyzickej pamäte. Postupne nám prídu požiadavky 7, 2, 3, 2. Jedna možnosť, ako ich môžeme spracovať, vyzerá nasledovne:

- Potrebujeme virtuálnu stránku 7. Nahráme ju do fyzickej pamäte na prvú pozíciu.
  Fyzická pamäť teda vyzerá nasledovne: (7, nič).
- Potrebujeme virtuálnu stránku 2. Nahráme ju na druhú pozíciu. Fyzická pamäť: (7, 2).
- Potrebujeme virtuálnu stránku 3. Odswapujeme stránku 7 a nahráme stránku 3. Fyzická pamäť: (3, 2).
- Potrebujeme virtuálnu stránku 2. Tú ešte stále máme vo fyzickej pamäti, nemusíme teda nič robiť.

Dokopy sme teda 3× museli nahrávať stránku do fyzickej pamäte. Všimnite si, že ak by sme v kroku 3 namiesto virtuálnej stránky 7 vyhodili virtuálnu stránku 2, museli by sme stránku do fyzickej pamäte nahrávať až 4×.

Ktorý online algoritmus je lepší?

Zamyslime sa nad príkladom, ktorý sme práve videli. Keby sme vopred poznali celú postupnosť požiadaviek, je zjavné, že by sme si vybrali prvé riešenie – to, ktoré potrebovalo len trikrát nahrávať stránku. Problém je ale v tom, že ju nevieme. Keď prišla tretia požiadavka, musíme sa hneď vtedy rozhodnúť, ktorú stránku z fyzickej pamäte vyhodiť. A nech sa rozhodneme akokoľvek, vždy bude možné, že ako ďalšia príde práve tá požiadavka, ktorá to celé pokazí. Online algoritmy teda zväčša nebudú vedieť zaručiť, že zostroja optimálne riešenie.
Keď chceme porovnať dva online algoritmy, potrebujeme vedieť nejak odmerať, ktorý z nich je lepší. Intuitívne by sme čakali, že lepší bude ten, ktorý sa od optimálneho riešenia bude menej líšiť. Ako ale túto intuíciu zachytiť formálnou definíciou?
Podobne ako pri časovej zložitosti, aj tu sa sústredíme na najhorší prípad: bude nás zaujímať *koľkokrát horšie* môže byť riešenie vyrobené online algoritmom v porovnaní s tým, ktoré by vyrobil offline, teda „vševedúci“, algoritmus *pre ten istý vstup*.
Uvažujme optimalizačné problémy, v ktorých je naším cieľom minimalizovať hodnotu riešenia. Majme konkrétny online algoritmus. Pre daný vstup v si označme $ALG(v)$ hodnotu riešenia, ktoré vyrobí on, a $OPT(v)$ hodnotu riešenia, ktoré by vyrobil optimálny offline algoritmus. Ak existuje reálna konštanta c taká, že pre úplne všetky v platí $ALG(v) \leq c \cdot OPT(v)$, tak hovoríme, že náš online algoritmus je *c-kompetitívny*. A samozrejme, čím je c bližšie k 1, tým je ten algoritmus lepší.

Poznámka o determinizme

Vyššie uvedenú definíciu a aj analýzu konkrétnych algoritmov by nám výrazne skomplikovalo, keby sme im dovolili používať náhodné čísla. Keďže vás tým nechceme trápiť, používanie náhody pre istotu úplne zakážeme a budeme sa zaoberať len deterministickými algoritmami. Podotýkame, že taktiež je zakázané používať generátor náhodných čísel pri riešení zadaných súťažných úloh.

Algoritmus „zásobník“

Stále sa zaoberáme problémom stránkovania, ktorý sme si popísali vyššie. V tejto časti sa pozrieme na konkrétny online algoritmus. Bude veľmi jednoduchý: vždy, keď je fyzická pamäť plná a algoritmus do nej potrebuje nahrať stránku, ktorá tam nie je, tak vyhodí tú stránku, ktorá tam je najkratšie.
Je tento algoritmus c-kompetitívny pre nejaké c? Ukážeme si, že nie.
Všimnime si vstup dĺžky $n > k$ ktorý vyzerá nasledovne:

$$1,\ 2,\ \ldots,\ k-1,\ k,\ k+1,\ k,\ k+1,\ \ldots,\ k,\ k+1,\ \ldots$$

Čo spraví algoritmus „zásobník“ pre tento vstup? Prvými k požiadavkami si zaplní fyzickú pamäť a potom bude na striedačku vyhadzovať stránku k a $k+1$ a nahrádzať ju tou druhou. Dokopy teda bude nahrávať stránku do fyzickej pamäte až n-krát (čiže v úplne každom kroku).
Ako vyzerá optimálne riešenie pre tento istý vstup? Zjavne po rovnakých prvých k krokoch stačí v $(k+1)$. kroku vyhodiť z pamäte ľubovoľnú stránku inú ako k. Od tej chvíle už budeme mať vo fyzickej pamäti aj stránku k, aj stránku $k+1$, a teda už nebudeme potrebovať nikdy nič ďalšie nahrávať. Dokopy teda nahráme stránku do fyzickej pamäte len $(k+1)$-krát.
No a teda tento algoritmus nemôže byť c-kompetitívny pre žiadne reálne c, lebo čím väčší vstup zoberieme, tým väčší bude pomer $n/(k+1)$. Teda ak by napríklad niekto tvrdil, že algoritmus „zásobník“ je 1000-kompetitívny, vieme mu to vyvrátiť napríklad tak, že zoberieme vstup vyššie popísaného tvaru v ktorom $n = 1001(k+1)$.

Algoritmus „fronta“

Pozrime sa teraz na druhý veľmi jednoduchý algoritmus. Od algoritmu „zásobník“ sa líši len v tom, že vždy, keď potrebuje uvoľniť miesto vo fyzickej pamäti, vyhodí tú stránku, ktorú má vo fyzickej pamäti *najdlhšie*.
Dokážeme si, že algoritmus „fronta“ je k-kompetitívny.
Uvažujme ľubovoľný vstup. Nech optimálne riešenie nahráva nejakú stránku do pamäte presne m-krát, a nech je to po požiadavkách s indexmi $i_1, \ldots, i_m$. (Všimnite si, že nutne platí $i_1 = 1$, keďže na začiatku je pamäť prázdna a teda prvú požadovanú stránku do nej musíme nahrať.)
Rozdeľme si teraz vstup na m úsekov, pričom j-ty úsek tvoria požiadavky s číslami i_j až $i_{j+1} - 1$. (Posledný

úsek je od i_m po koniec vstupu.)
Všimnime si, že úseky môžu síce byť ľubovoľne dlhé, ale v každom úseku sa môže vyskytovať len nanajvýš k rôznych stránok. Totiž keďže počas j-teho úseku optimálny algoritmus nahráva do pamäte novú stránku len v kroku i_j, musia všetky požiadavky v tomto úseku byť len na tie stránky, ktoré má náš optimálny algoritmus práve v pamäti po spracovaní požiadavky s indexom i_j.
To ale znamená, že algoritmus „fronta“ počas spracúvania tohto úseku nahrá stránku do pamäte nanajvýš k-krát. Totiž počas spracúvania úseku stretneme požiadavky na nanajvýš k rôznych stránok, a ľahko nahliadneme, že každú z nich počas spracúvania úseku nanajvýš raz nahráme do pamäte. Prečo? Lebo na to, aby nám pri použití algoritmu „fronta“ nejaká stránka z pamäte vypadla, potrebujeme po nej nahrať do pamäte iných k stránok, a to počas tohto úseku nenastane.
Inými slovami, na ľubovoľnom vstupe, na ktorom optimálny algoritmus nahráva stránku do pamäte m-krát, nahráva algoritmus „fronta“ stránku do pamäte nanajvýš km-krát. A z toho priamo podľa definície vidíme, že algoritmus „fronta“ je k-kompetitívny.

Optimálnosť algoritmu „fronta“

V tejto časti si dokážeme, že pre online problém stránkovania nemôže existovať lepší ako k-kompetitívny algoritmus – a teda že algoritmus „fronta“ je v tomto zmysle optimálnym online algoritmom.
Uvažujme úplne ľubovoľný deterministický online algoritmus pre problém stránkovania. Začneme tým, že špeciálne preň zostrojíme vstup, ktorý ho donúti veľakrát nahrávať stránku do pamäte.
Vstup budeme zostrojovať priebežne počas behu samotného algoritmu. Prvých k požiadaviek budú stránky s číslami 1 až k. Každú ďalšiu požiadavku zvolíme nasledovne: pozrieme sa na to, ktorých k stránok má náš algoritmus práve v pamäti, a ako požiadavku mu zadáme najmenšie číslo stránky, ktorú v pamäti nemá. (Kvôli neskoršej analýze si všimnite, že toto číslo bude zjavne vždy z rozsahu 1 až $k+1$.)
Zjavne k ľubovoľnému online algoritmu a číslam n a k vieme takto zostrojiť jeden konkrétny vstup dĺžky n, pre ktorý tento online algoritmus bude musieť nahrávať stránku do pamäte v úplne každom kroku. Pre tento vstup v teda platí $ALG(v) = n$.
Čo vieme povedať o hodnote $OPT(v)$? Uvažujme nasledovný (nie nutne optimálny) offline algoritmus: Prvých k krokov má vynútených, rovnako ako v predchádzajúcom riešení musí nahrať do pamäte stránky s číslami 1 až k a zatiaľ nemá zmysel žiadnu z nich vyhadzovať. Od tohto okamihu bude náš offline algoritmus postupovať nasledovne: vždy, keď treba nejakú stránku z pamäte vyhodiť, pozrieme sa do budúcnosti na k požiadaviek začínajúc práve spracúvanou a vyhodíme z pamäte ľubovoľnú stránku, ktorá sa medzi nimi nenachádza.
Všimnime si, že v požiadavkach, ktoré spracúvame, sa používa len $k+1$ rôznych stránok. Preto vždy v okamihu, keď nejakú stránku z pamäte vyhodíme a nahradíme inou, budeme mať v pamäti všetkých k stránok iných od tej práve vyhodenej. To ale znamená, že vždy, keď náš algoritmus vyhodí z pamäte stránku, ktorú k krokov nebude potrebovať, tak má v pamäti všetky stránky, ktoré potrebovať bude. Inými slovami, vyššie popísaný algoritmus pre náš vstup v vždy po nahraní stránky do pamäte bude mať aspoň $k-1$ krokov, počas ktorých zaručene do pamäte nič nahrávať nebude.
Inými slovami, na vstupe dĺžky n tento offline algoritmus nahrá do pamäte stránku nanajvýš $(k + \lceil (n-k)/k \rceil)$-krát. A keďže optimálny algoritmus je aspoň taký dobrý ako hocijaký iný, platí $OPT(v) \leq k + \lceil (n-k)/k \rceil$.
Platí teda:

$$\frac{ALG(v)}{OPT(v)} \geq \frac{n}{k + \lceil (n-k)/k \rceil} \geq \frac{n}{(n+k^2)/k} = k \cdot \frac{n+k^2-k^2}{n+k^2} = k \cdot \left(1 - \frac{k^2}{n+k^2}\right)$$

Všimnite si, že pre dostatočne dlhé vstupy (t.j. dostatočne veľké n) bude výraz na pravej strane nadobúdať hodnoty ľubovoľne blízke k. To ale znamená, že žiaden online algoritmus pre problém stránkovania nemôže byť c-kompetitívny pre žiadne $c < k$, čo je presne to, čo sme chceli dokázať.

TRIDSIATY ŠTVRTÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Bui Truc Lam, Michal Forišek, Samuel Sládek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2019