Priebeh krajského kola

Krajské kolo 37. ročníka Olympiády v informatike, kategória A, sa koná 18. 1. 2022 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 Hladná myška

Katka má v labáku bludisko pre myšky. Bludisko je tvorené n miestnosťami a m chodbičkami medzi nimi. Miestnosti sú očíslované od 0 po $n-1$. V miestnosti 0 je vchod do bludiska, v miestnosti $n-1$ je východ. Všetky chodbičky sú obojsmerné.
V niektorých miestnostiach sú momentálne rozmiestnené rôzne veľké kúsky syra. Syr je veľmi chutný a myšky mu nevedia odolať. Akonáhle myška príde do miestnosti so syrom, okamžite ho celý zožerie.
Tu však môže pre myšky nastať istý problém: čím viac syra myška zožerie, tým bude ťažšia (a širšia). No a potom sa jej môže stať, že sa do niektorých chodbičiek nezmestí. Pre každú chodbičku poznáme maximálnu hmotnosť myšky, ktorá ňou ešte prejde.

Súťažná úloha

Algernon je múdra myška. Vie, že zajtra ho Katka dá do bludiska a on sa bude musieť dostať von. Pozná celé bludisko a vie aj to, ako je v ňom rozmiestnený syr. Algernon by chcel vedieť, ako veľmi sa dnes večer môže najesť, ak chce zajtra úspešne prejsť bludiskom. Napíšte program, ktorý to pre neho zistí. (Pri prechode bludiskom si Algernon môže zvoliť, cez ktoré chodbičky a miestnosti sa bude pohybovať, ale musí zjesť všetok syr, ktorý na zvolenej ceste stretne.)

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla n a m: počet miestností v bludisku a počet chodbičiek.
V druhom riadku vstupu je n celých čísel $s_0, \ldots, s_{n-1}$, pričom $s_0 = s_{n-1} = 0$. Číslo s_i udáva hmotnosť syra v miestnosti i. (Keď myška zožerie syr v miestnosti i, stúpne hmotnosť myšky o s_i.)
Zvyšok vstupu tvorí m riadkov, každý z nich popisuje jednu chodbičku. Popis chodbičky tvoria čísla a_j, b_j miestností, ktoré spája, a číslo c_j: maximálna hmotnosť myšky, ktorá sa ešte cez túto chodbičku zmestí.

Nech h je hmotnosť, ktorú má Algernon v okamihu, kedy ho Katka vloží do miestnosti 0. Algernon má odtiaľ prejsť bludiskom do miestnosti $n-1$. Nájdite a vypíšte najväčšiu hodnotu h, pre ktorú to vie spraviť.
Ak neexistuje žiadna kladná hmotnosť h s touto vlastnosťou, vypíšte namiesto toho číslo -1.

Obmedzenia a hodnotenie

Plný počet bodov je za riešenia efektívne pre $n \leq 200\,000$, $m \leq 500\,000$, $s_i \leq 10^9$ a $c_i \leq 10^9$.
Nanajvýš 8 bodov je za riešenia efektívne pre $n \leq 50\,000$, $m \leq 200\,000$, $s_i \leq 10^6$ a $c_i \leq 10^6$.
Nanajvýš 6 bodov je za riešenia efektívne pre $n \leq 5\,000$, $m \leq 20\,000$, $s_i \leq 10^6$ a $c_i \leq 10^6$.
Nanajvýš 4 body sú za riešenia efektívne pre $n \leq 500$, $m \leq 1000$, $s_i \leq 100$ a $c_i \leq 100$.

Príklady

vstup

```
3 2
0 100 0
0 1 470
1 2 100
```

výstup

```
-1
```

Bludisko tvoria tri miestnosti, široká chodbička medzi miestnosťami 0-1 a užšia chodbička medzi miestnosťami 1-2. V miestnosti 1 je 100 gramov syra. Algernon sa do cieľa bludiska nedostane, lebo aj keby začínal len s hmotnosťou 1 gram, akonáhle príde do miestnosti 1 a zožerie syr, bude mať viac ako 100 gramov a už sa nezmestí do druhej chodbičky.

vstup

```
3 2
0 47 0
0 1 470
1 2 100
```

výstup

```
53
```

Do bludiska príde 53-gramový Algernon, v miestnosti 1 zje syr a ešte sa tesne zmestí do druhej chodbičky.

vstup

```
5 6
0 10 10 10 0
0 1 135
0 2 115
2 1 247
1 3 126
2 3 150
3 4 147
```

výstup

```
117
```

Algernon má možnosť ísť cestami 0-1-3-4 alebo 0-2-3-4, pri ktorých zožerie len 20 gramov syra. Najlepšou možnosťou je však, možno prekvapivo, ísť cestou 0-1-2-3-4 a zožrať až 30 gramov syra. Táto cesta má totiž o čosi širšie chodbičky, a tak môže na začiatku vážiť o čosi viac.

A-II-2 Červené jabĺčko

Janko má 1000 jabĺk. Jedno z nich je červené a všetky ostatné sú zelené. Janko to o nich vie.
Janko má vrodenú vadu: červeno-zelenú farbosleposť. Všetky jablká sa mu zdajú rovnaké.
Janko má už od rána chuť na červené jablko.
Janko má Peťku, ktorá vie rozlíšiť červenú od zelenej.
Peťka má chuť Janka potrápiť a odmieta mu priamo povedať, ktoré jablko je červené.
Peťka má v pláne odpovedať Jankovi len na „áno/nie“ otázky.
Janko má napr. povolené položiť konkrétnych 147 jabĺk na stôl a opýtať sa Peťky „je červené jablko na stole?“
Peťka má názor, že by to takto Janko mal ešte stále príliš ľahké.
Peťka má plán, ako to Jankovi sťažiť: oznámila mu, že môže *nanajvýš v jednej svojej odpovedi* klamať.

Súťažná úloha

Janko musí zistiť, ktoré z jeho jabĺk je to jediné červené. Musí to spraviť tak, že Peťke postupne kladie otázky. Otázky musí formulovať tak, aby Peťka na každú vedela logicky odpovedať „áno“ alebo „nie“. Otázky bude klásť postupne – teda ďalšiu otázku si vie vždy zvoliť podľa toho, ako Peťka dovtedy odpovedala.
Janko môže pri hľadaní červeného jablka použiť veľa rôznych stratégií. Nás budú zaujímať len *korektné stratégie*, teda také, pri ktorých je zaručené, že Janko si na konci bude istý, ktoré jablko je červené.
Pri vyhodnocovaní toho, ktorá korektná stratégia je ako dobrá, nás bude zaujímať jej *najhorší možný prípad* – teda najväčší počet otázok, ktoré môže Janko potrebovať položiť Peťke, kým nájde správne jablko. Čím je tento počet menší, tým je stratégia lepšia.
Nájdite čo najlepšiu korektnú stratégiu pre Janka. Stratégiu dostatočne exaktne popíšte, najlepšie vo forme pseudokódu. (Prípadne je možné stratégiu uviesť aj v podobe programu vo vašom obľúbenom programovacom jazyku. V takom prípade samozrejme očakávame aj slovný popis, rovnako ako v iných úlohách.)
Nezabudnite zdôvodniť, že je vaša stratégia korektná, a uviesť, koľko otázok potrebuje v najhoršom prípade.

Obmedzenia a hodnotenie

Počet bodov, ktoré dostanete, závisí od hodnoty x: počtu otázok, ktoré vaša stratégia potrebuje položiť v najhoršom prípade. Čím menšie x máte, tým viac bodov môžete získať. Na plný počet bodov nájdite stratégiu, ktorej stačí $x \leq 14$ otázok. Za korektnú stratégiu s $x = 21$ môžete získať 6 bodov. Za korektnú stratégiu s $x > 900$ môžete získať 3 body.

Príklad komunikácie

```
Janko (s dvomi jablkami v rukách): Je jedno z týchto dvoch jabĺk červené?
Peťka: Áno.
Janko (ukazujúc jablko v pravej ruke): Je toto jablko červené?
Peťka: Nie.
Janko (znovu ukazujúc jablko v pravej ruke): Je toto jablko červené?
Peťka: Áno.
Janko (ukazujúc jablko v ľavej ruke): Je toto jablko červené?
Peťka: Nie.
Janko: Som si istý, že jablko v mojej pravej ruke je červené. Potreboval som 4 otázky.
```

Neúplný príklad popisu nejakej (nie nutne dobrej) stratégie

```
Nachystaj si prázdne vrece.
Kým máš nejaké jablká mimo vreca, opakuj:
    Vezmi dve z jabĺk mimo vreca do rúk, ukáž ich Peťke.
    Opýtaj sa, či je niektoré z nich červené.
    Ak odpovie "nie":
        daj obe do vreca
    inak:
        ...
```

A-II-3 Turbojazdec

Jazdec je klasická šachová figúrka. Pohybuje sa tak, že ho zdvihneme, posunieme o dve políčka niektorým smerom (v riadku alebo v stĺpci), potom o jedno políčko v kolmom smere a tam ho položíme.
Turbojazdec sa pohybuje podobne ako jazdec, len rýchlejšie: najskôr ho posunieme o **viac ako dve** políčka jedným smerom a potom o **viac ako jedno** druhým smerom (kolmým na prvý).

Máme obriu šachovnicu s r riadkami a s stĺpcami. Na každom políčku je napísané jedno písmeno.
Chceme po nej preskákať turbojazdcom tak, aby postupne navštívil písmenká daného slova w. (Začať môžeme na ľubovoľnom políčku obsahujúcom prvé písmeno w, každým ďalším ťahom sa musíme presunúť na jeho nasledujúce písmeno.)
Koľkými rôznymi spôsobmi sa to dá spraviť?
Namiesto správnej odpovede stačí, keď vypočítate, aký zvyšok dáva po delení $10^9 + 7$. (Správna odpoveď totiž môže byť veľmi veľká.)

Formát vstupu a výstupu

V prvom riadku vstupu sú rozmery šachovnice: r a s. V druhom riadku je slovo w. Zvyšok vstupu tvorí popis písmen na šachovnici: r riadkov a v každom z nich s písmen.
Na výstup vypíšte jeden riadok so správnou odpoveďou.

Obmedzenia a hodnotenie

Môžete predpokladať, že všetky písmená na vstupe sú veľké písmená anglickej abecedy. Veľkosť abecedy môžete pri analýze algoritmu považovať za konštantu.
Plný počet bodov môžu dostať riešenia, ktoré efektívne vyriešia vstupy s $r, s \leq 1000$ a $|w| \leq 100$.
Nanajvýš 8 bodov môžu dostať riešenia, ktoré efektívne vyriešia vstupy s $r, s, |w| \leq 100$.
Nanajvýš 6 bodov môžu dostať riešenia, ktoré efektívne vyriešia vstupy s $r, s, |w| \leq 40$.
Nanajvýš 3 body môžu dostať riešenia, ktoré efektívne vyriešia vstupy v ktorých $r, s, |w| \leq 40$ a navyše máme zaručené, že pre každé i existuje nanajvýš 10^6 platných spôsobov preskákania prvých i písmen slova w.

Príklady

vstup

```
3 4
OI
KIVI
MALO
OBAL
```

výstup

```
1
```

Na takto malej šachovnici vie turbojazdec spraviť jediný typ pohybu: z niektorého rohu do protiľahlého rohu. Jediná možnosť, pri ktorej takto navštívi písmená slova OI, začína v ľavom dolnom rohu a pokračuje skokom do pravého horného rohu.

vstup

```
3 5
BRAT
BANAN
NEMAL
SUPKU
```

výstup

```
0
```

Keďže na šachovnici nemáme žiadne R ani T, je zjavné, že po písmenách slova BRAT sa preskákať nedá.

vstup

```
3 5
UUUUUU
UUKUU
HESLO
UUPUU
```

výstup

```
84
```

Ak riadky a stĺpce číslujeme od 0 vľavo hore, jedna prípustná možnosť skákania je $(0,0) \to (2,4) \to (0,1) \to (2,4) \to (0,0) \to (2,3)$.

vstup

```
6 8
KSP
KSPPSKSP
PPSKSKSP
SKKSKSKP
PPPSKSKK
KKSKSKPP
PPPSKSKP
```

výstup

```
414
```

Jedno prípustné riešenie je začať na K úplne vľavo hore, odtiaľ skočiť na S zhruba v strede dole a odtiaľ na P úplne vpravo hore.

A-II-4 Pokazený rover kóduje čísla

K tejto úlohe patrí študijný text uvedený na nasledujúcich stranách. Odporúčame najskôr prečítať ten a až potom sa vrátiť k samotným súťažným úlohám.
Na konci študijného textu nájdete pridaný prehľad makier počítajúcich funkcie, ktoré sme si naprogramovali v domácom kole. Aj tieto môžete používať pri riešení aktuálnych súťažných úloh.

Podúloha A (1 bod).
Majme nasledovné poradie dvojíc nezáporných čísel: $(0,0)$, $(0,1)$, $(1,0)$, $(0,2)$, $(1,1)$, $(2,0)$, $(0,3)$, $(1,2)$, $(2,1)$, $(3,0)$, $(0,4)$, $(1,3)$, ...
Rozmyslite si, ako v tomto poradí pokračovať tak, aby každú dvojicu (x,y) obsahovalo práve raz. Nájdite matematický predpis pre funkciu *spoj* takú, že $spoj(x,y)$ je pozícia dvojice (x,y) v tomto poradí. Pozície číslujeme od nuly, teda napr. má platiť $spoj(3,0) = 9$.

Podúloha B (2 body).
Napíšte pre rover makro `spoj X Y Z` počítajúce funkciu *spoj* z podúlohy A.
V lokalitách `X` a `Y` sú ľubovoľné počty kamienkov, ktoré si označíme x a y. Keď makro skončí, v lokalite `Z` má byť toľko kamienkov, aká je pozícia dvojice (x,y) v našom zozname.

Podúloha C (4 body).
Napíšte pre rover makro `prvy Z X`.
V lokalite `Z` je nejaký neznámy počet kamienkov, ktorý si označíme z. Nech (x,y) je dvojica, ktorá je v našom poradí na pozícii z. Vaše makro má zistiť hodnotu x a do lokality `X` uložiť x kamienkov.

Podúloha D (3 body).
Doteraz sme mohli každú lokalitu chápať ako premennú, do ktorej vieme uložiť nezáporné celé číslo. Pomocou funkcie `spoj` však vieme do jednej lokality uložiť dve čísla – na lokalitu, v ktorej je `spoj(x,y)` kamienkov, sa môžeme dívať ako na premennú, v ktorej sú uložené čísla x a y.
Nám však dve čísla nestačia a chceli by sme viac: do jednej lokality chceme uložiť ľubovoľne veľa čísel naraz!
Toto spravíme nasledovne:

- Prázdnej postupnosti čísel priradíme ako kód číslo 0.
- Predpokladajme, že už poznáme kód k postupnosti $x_1, \ldots, x_n$. Kódom postupnosti $x_0, x_1, \ldots, x_n$ bude hodnota $1 + spoj(x_0, k)$.

Napíšte pre rover makro `dlzka Z X`.
V lokalite `Z` je z kamienkov. Číslo z je kódom nejakej postupnosti. Lokalita `X` je prázdna. Vaše makro má do nej uložiť dĺžku postupnosti, ktorej kódom je číslo z.
Pri písaní tohto makra môžete používať makrá `spoj` a `prvy` (aj ak ste nevyriešili predchádzajúce podúlohy). Tiež môžete používať makro `druhy`, ktoré funguje podobne ako `prvy`, len namiesto hodnoty x vypočíta hodnotu y.

Študijný text: Pokazený rover

Na Marse máme rover. Voľakedy sme mali veľké plány, no po tom, ako ho zasiahla piesočná búrka a pokazila mu skoro všetky periférie, ostal rover takmer nepoužiteľný. Jediné, čo ešte stále vie robiť, je presúvať sa medzi lokalitami a nosiť z miesta na miesto nejaké kamienky. Našou úlohou v tejto sérii úloh bude naučiť rover robiť aspoň nejaké použiteľné výpočty. **Nebude nám pri tom záležať na časovej zložitosti** programov, len na ich korektnosti.

Do roveru vieme na diaľku nahrať program: konečnú postupnosť inštrukcií. Niektoré inštrukcie môžu mať návestia (labels) – symbolické mená, pomocou ktorých sa na ne vieme odkazovať.

Rover pozná na Marse dve špeciálne lokality: *jedáleň* a *kameňolom*. Pre stručnosť ich budeme označovať `J` a `K`. V jedálni je momentálne práve jeden kamienok, inak je to lokalita ako každá iná. V kameňolome je vždy k dispozícii dostatočne veľa kamienkov.

Každému inému reťazcu má rover priradenú nejakú unikátnu lokalitu na Marse, kam sa dajú ukladať kamienky. Ak nie je povedané ináč, predpokladáme, že všetky takéto lokality sú prázdne – neobsahujú žiadne kamienky.

Jediná inštrukcia, ktorú ešte rover vie vykonávať, vyzerá nasledovne:

1. Príď do lokality `Y`. Spočítaj si do pomocnej premennej, koľko je tam kamienkov.
2. Príď do lokality `X`. Pokús sa nabrať taký počet kamienkov aký máš v pomocnej premennej.
3. Ak sa to podarilo, odnes tieto kamienky do lokality `Z`, tam ich vysyp a pokračuj nasledujúcou inštrukciou.
4. Ak sa to nepodarilo (t.j. v lokalite `X` nie je dosť kamienkov), vráť lokalitu `X` do pôvodného stavu a pokračuj inštrukciou s návestím `N`.

Program, ktorý pošleme roveru, bude teda postupnosťou takýchto inštrukcii. Inštrukciu vrátane návestia budeme zapisovať nasledovne:

```
navestie: prenes X Y Z N
```

V ľudskej reči môžeme jednotlivé parametre tejto inštrukcie čítať nasledovne:

```
prenes (odkial) (kolko) (kam) (co robit ak sa nepodarilo)
```

Ako štvrtý parameter môžeme písať `-` (pomlčku) ak chceme, aby vykonávanie programu aj v prípade neúspechu pokračovalo nasledujúcou inštrukciou.

Lokality `X`, `Y` a `Z` nemusia byť navzájom rôzne. Jediné obmedzenie je, že kameňolom (kde je „nekonečne veľa“ kamienkov) nesmieme použiť ako lokalitu `Y`.

Vykonávanie programu sa skončí, keď sa dostane na neexistujúcu inštrukciu – teda buď keď vykonáme poslednú inštrukciu a máme pokračovať nasledujúcou po nej, alebo keď skočíme na neexistujúce návestie.

Príklad 1: príkazy, ktoré skoro nič nerobia

Čo spraví rover, keď mu dáme príkaz `prenes X X X I`? Spočíta kamienky v lokalite `X`, potom ich všetky naberie, potom ich na tom istom mieste zase všetky vysype a bude pokračovať nasledujúcou inštrukciou. Takáto inštrukcia teda Mars vôbec nezmení.

Čo spraví rover, keď mu dáme príkaz `prenes X Y X I`? Aj tentokrát sa počet kamienkov v žiadnej lokalite nezmení, sú však už dva možné priebehy výpočtu: ak bolo v lokalite `Y` viac kamienkov ako v lokalite `X`, výpočet bude pokračovať inštrukciou `I`.

Príklad 2: vyprázdni lokalitu

Rozmyslite si sami, akým príkazom vieme z lokality odstrániť všetky kamienky.

Riešenie: Na vyprázdnenie lokality `X` môžeme použiť príkaz `prenes X X K -`. Rover spočíta kamienky v lokalite `X`, všetky naberie a vysype ich v kameňolome.

Príklad 3: naučíme rover sčítať

V lokalitách `A` a `B` máme nejaké neznáme množstvá kamienkov, lokalita `C` je prázdna. Chceli by sme v lokalite `C` mať počet kamienkov rovný súčtu lokalít `A` a `B`. Lokality `A` a `B` pritom chceme nechať nezmenené.

Opäť, skôr ako si nižšie prečítate riešenie, skúste si ho sami vymyslieť.
Riešenie: Stačí príslušné počty kamienkov presypať z kameňolomu do lokality C. Jedným korektným riešením je teda nasledovný program:

```
prenes K A C -
prenes K B C -
```

Príklad 4: naučíme rover odčítať

V lokalitách A a B máme nejaké neznáme počty kamienkov, ktoré označíme a a b. Lokalita C je prázdna. Chceli by sme v lokalite C mať počet kamienkov rovný $a - b$, resp. rovný nule ak $b > a$. Lokality A a B pritom chceme nechať nezmenené.
(Na rozdiel od inštrukcie prenes, ktorá odoberie kamienky len ak vie odobrať všetky, chceme pri našom odčítaní vždy odobrať koľko ide.)
Riešenie: Do lokality C dáme a kamienkov a potom sa pokúsime odobrať b. Ak sa nám to podarí, sme hotoví, ak nie, tak ešte celú lokalitu C vyprázdnime do kameňolomu.
Program:

```
          prenes K A C -
          prenes C B K nulovanie
          prenes prazdna J K koniec
nulovanie: prenes C C K
```

Všimnite si, že v treťom kroku (ktorý sa vykoná, ak sme z lokality C úspešne odobrali b kamienkov) sa pokúsime z prázdnej lokality presunúť jeden kamienok. To sa nám zaručene nepodarí, program teda skočí na neexistujúce návestie „koniec“ a tým skončí. Štvrtá inštrukcia sa teda vykoná len vtedy, ak na ňu skočíme z druhej.

Príklad 5: naučíme rover násobiť

V lokalitách A a B máme nejaké neznáme počty kamienkov, ktoré označíme a a b. Lokalita C je prázdna. Chceli by sme v lokalite C mať počet kamienkov rovný $a \cdot b$. Lokality A a B pritom chceme nechať nezmenené.
Toto už nevieme spraviť v konštantnom čase. Násobenie je však len opakované sčítanie: kôpku ab kamienkov vyrobíme tak, že na ňu a-krát prinesieme b kamienkov.
Program:

```
        prenes K A Akopia -
cyklus: prenes Akopia J K koniec
        prenes K B C -
        prenes prazdna J K cyklus
```

Na začiatku si vyrobíme kópiu lokality A, ktorú potom počas výpočtu zničíme. Dokola opakujeme: zober jeden kamienok z kópie lokality Akopia a pridaj b kamienkov do lokality C.

Makrá

Roveru sa dajú definovať makrá: zoberieme postupnosť príkazov a dáme jej symbolické meno, aby sme nemuseli tú istú postupnosť príkazov rozpisovať viackrát. Makro môže mať parametre: pri rôznych použitiach môže robiť tie isté inštrukcie ale pre iné lokality a iné návestia.
Vo vnútri makra môžeme používať aj pomocné lokality. Lokality, ktoré dáme pri jeho definícii do hranatých zátvoriek, budú pri každom použití makra nahradené inou lokalitou, ktorá sa nikde inde v programe nepoužíva. Toto isté sa automaticky stane aj so všetkými návestiami, ktoré dáme inštrukciám v definícii makra. Každé takéto návestie je teda akoby viditeľné len z jedného konkrétneho použitia makra.
Pri definícii makra smieme používať aj iné makrá, ktoré sme už skôr zadefinovali.
Nižšie uvádzame niekoľko definícií makier. Riadky začínajúce mriežkou sú komentáre.

```
# instrukcia, ktora nic nezmeni, len jeden krok caka
MAKRO cakaj
    prenes J J J -
END

# makro s dvoma parametrami-lokalitami: do Y prida tolko kamienkov, kolko je v X
MAKRO pridaj X Y
    prenes K X Y -
END
```

```
# makro s jednym parametrom-navestim: skoci na dane navestie
MAKRO skoc N
    prenes [nova_prazdna_lokalita] J K N
END

# makro pre nasobenie: do Z prida sucin poctov kamienkov z X a Y
MAKRO vynasob X Y Z
            pridaj X [Xkopia]
    cyklus: prenes [Xkopia] J K koniec
            prenes K Y Z -
            skoc cyklus
    koniec: cakaj
END
```

Všimnite si, že keď sme vyššie písali samostatný program pre násobenie, stačilo nám, že návestie `koniec` nikde neexistuje a skokom naň sme ukončili program. Pri písaní makra toto návestie dáme pred prázdnu inštrukciu na konci makra, keďže chceme, aby po ukončení násobenia program ešte pokračoval ďalej.

Príklad 5: naučíme rover počítať tretiu mocninu

Pomocou vyššie uvedených makier ľahko napíšeme program počítajúci tretiu mocninu. V lokalite A máme nejaký neznámy počet kamienkov, ktorý označíme a. Lokalita B je prázdna a chceme do nej uložiť a^3 kamienkov. Program:

```
vynasob A A pomocna_lokalita
vynasob A pomocna_lokalita B
```

Pre názornosť ešte ukážeme, ako by mohol vyzerať ten istý program, ak by sme všetky makrá nahradili ich definíciami.

```
            prenes  K         A                 Akopia1           -
   cyklus1: prenes  Akopia1   J                 K                 koniec1
            prenes  K         A                 pomocna_lokalita  -
            prenes  hocico1   J                 K                 cyklus1
   koniec1: prenes  J         J                 J                 -
            prenes  K         A                 Akopia2           -
   cyklus2: prenes  Akopia2   J                 K                 koniec2
            prenes  K         pomocna_lokalita  B                 -
            prenes  hocico2   J                 K                 cyklus2
   koniec2: prenes  J         J                 J                 -
```

Riešenia domáceho kola

Nasleduje zoznam makier, ktoré počítajú funkcie, ktoré sme si naprogramovali v riešeniach domáceho kola. Tieto makrá môžete používať vo svojich riešeniach bez potreby rozpisovať ich implementáciu.

```
MAKRO vynuluj X           # vynuluje obsah X
MAKRO nulove X N          # ak X obsahuje nulu, skoci na navestie N
MAKRO rovnake X Y N       # ak su hodnoty X a Y rovnake, skoci na navestie N
MAKRO zapis X Y           # hodnotu X zapise do Y, pricom prepise povodny obsah Y
MAKRO vydel X Y P Z       # hodnotu X vydeli hodnotou Y, do P zapise podiel, do Z zvysok
MAKRO nedelitelne X Y N   # ak X nie je delitelne Y, skoci na navestie N
MAKRO nsd X Y Z           # do Z ulozi najvacsieho spolocneho delitela X a Y
```

TRIDSIATY SIEDMY ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2021