### Priebeh celoštátneho kola

Celoštátne kolo 37. ročníka Olympiády v informatike, kategórie A, sa koná v dňoch 23.-26. 3. 2022. 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 Ihrisko

Veľký vezír Miško sa rozhodol, že Absurdistan sa stane novou veľmocou v metlobale.
Na to by sa zišlo mať v krajine nejaké ihrisko. Ihrisko musí byť obdĺžnikového tvaru, musí mať strany v smeroch hlavných svetových strán a musí mať v dvoch protiľahlých rohoch stĺp umelého osvetlenia.
Stĺpy umelého osvetlenia sú momentálne všade vypredané. Našťastie po Absurdistane ich už $n$ stojí. Miško bude musieť využiť dva z týchto $n$ stĺpov.
Stĺpy sú už trochu staršie a keby sa nimi hýbalo, mohli by sa poškodiť. Miško ich teda musí nechať stáť tam, kde práve stoja, a ihrisko proste postaviť tak, aby malo dva z nich v protiľahlých rohoch. Aha, a nech nezabudneme: na ihrisku pochopiteľne nesmú trčať žiadne ďalšie stĺpy s osvetlením, nech sa hráči nezranía.

### Súťažná úloha

Daných je $n$ bodov v rovine. Pre pohodlie môžete predpokladať, že **žiadne dva nie sú na tej istej vodorovnej ani zvislej súradnici**.
Napíšte algoritmus, ktorý zistí, koľkými spôsobmi vieme v tejto rovine vyznačiť obdĺžnik, ktorého strany sú rovnobežné so súradnicovými osami, v dvoch protiľahlých vrcholoch má dva zo zadaných bodov a celé jeho vnútro je prázdne (ostatných $n - 2$ bodov leží mimo neho).

### Formát vstupu a výstupu

V prvom riadku vstupu je celé číslo $n$. V každom zo zvyšných $n$ riadkov sú súradnice $x_i$, $y_i$ jedného z bodov. (Všetky $x_i$ sú navzájom rôzne celé čísla. Všetky $y_i$ sú navzájom rôzne celé čísla.)
Na výstup vypíšte jedno číslo: počet možných ihrísk.

### Príklad

vstup

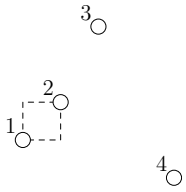
```
4
2 2
3 3
4 5
6 1
```

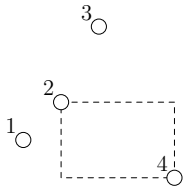
výstup

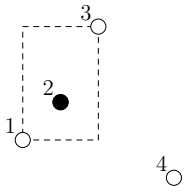
```
5
```

*Ak by sme vybrali prvý a tretí stĺp osvetlenia, bol by na ihrisku druhý stĺp, táto možnosť teda nevyhovuje. Každá zo zvyšných piatich možností je OK.*

*Na ľavom a strednom obrázku sú dva z piatich dobrých obdĺžnikov. Na pravom obrázku je jediný zlý obdĺžnik: obdĺžnik, ktorý má v protiľahlých rohoch stĺpy 1 a 3 nevyhovuje, keďže obsahuje stĺp 2.*




### Obmedzenia a hodnotenie

Za riešenia efektívne pre $n \leq 500$ môžete získať nanajvýš 2 body.
Za riešenia efektívne pre $n \leq 5000$ môžete získať nanajvýš 6-7 bodov, pričom maximum závisí od presnej asymptotickej časovej zložitosti riešenia.
Za riešenia efektívne pre $n \leq 100\,000$ môžete získať nanajvýš 10 bodov.

## A-III-2 Horiaci strom

Na lúke stojí krásny košatý strom. Má $n$ vrcholov, očíslovaných od 0 po $n-1$. Medzi vrcholmi vedú hrany. Každá hrana má nejakú kladnú dĺžku. (Vrcholy sú všetky konce stromu a všetky miesta, kde sa vetví. Hrany sú jednotlivé konáre a kusy kmeňa stromu.)
Do stromu práve udrel krásny košatý blesk. Strom následne v niektorých vrcholoch začal horieť.
Oheň sa šíri spojito po všetkých hranách stromu, a to rýchlosťou 1 cm za sekundu. Teda keď začne horieť nejaký vrchol, o 4.7 sekundy horí všetko vo vzdialenosti nanajvýš 4.7 cm od neho.
Keď sa na strom pozrieme v nejakom konkrétnom okamihu, vidíme, že niektoré súvislé časti stromu už horia a iné súvislé časti stromu ešte nehoria. Každú maximálnu súvislú časť stromu, ktorá ešte nehorí, budeme volať *oblasť*. Bude nás zaujímať ich počet.

### Súťažná úloha

Daný je popis stromu a zoznam vrcholov, kde začal (vo všetkých naraz) horieť. Napíšte program, ktorý zistí, na koľko najviac nehoriacich oblastí bol strom rozdelený niekedy počas toho, ako horel.

### Formát vstupu a výstupu

V prvom riadku vstupu sú celé čísla $n$ a $h$: počet vrcholov stromu a počet tých z nich, ktoré na začiatku horia.
Druhý a tretí riadok popisujú tvar nášho stromu.
V druhom riadku vstupu sú celé čísla $p_1, \ldots, p_{n-1}$, pre ktoré platí $0 \leq p_i < i$.
V treťom riadku vstupu sú kladné celé čísla $d_1, \ldots, d_{n-1}$.
Pre každé $i$ platí, že jedna z hrán stromu spája vrcholy $i$ a $p_i$. Táto hrana má dĺžku $d_i$ cm.
V poslednom riadku vstupu sú čísla $z_1, \ldots, z_h$: čísla na začiatku horiacich vrcholov.

Na výstup vypíšte jediné číslo: maximálny počet disjunktných nehoriacich oblastí, ktoré existovali niekedy počas požiaru stromu.

### Obmedzenia a hodnotenie

Plný počet bodov môžu dostať riešenia, ktoré úlohu efektívne vyriešia pre $n \leq 200\,000$ a $d_i \leq 10^9$, a to vrátane riešení, ktoré budú mať asymptotickú časovú zložitosť mierne horšiu ako vzorové riešenie.
Riešenia, ktoré úlohu efektívne vyriešia pre $n \leq 5\,000$ a $d_i \leq 10^9$ môžu dostať 6 bodov.
Riešenia, ktoré úlohu efektívne vyriešia pre $n \leq 50$ a $d_i \leq 10$ môžu dostať 3 body.
Pre pohodlie môžete predpokladať, že všetky čísla $d_i$ sú párne.

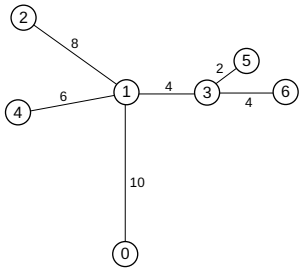
### Príklady

vstup

```
7 7
0 1 1 1 3 3
10 8 4 6 2 4
0 1 2 3 4 5 6
```

výstup

```
6
```



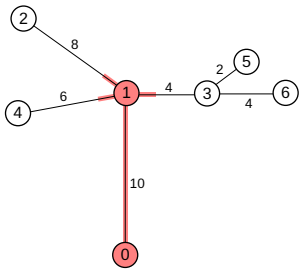

*Toto je 7-vrcholový strom, ktorý začal naraz horieť vo všetkých vrcholoch. Tesne po tom, ako začal horieť, existovalo 6 nehoriacich oblastí: jedna na každej hrane. Po 1 sekunde už horela celá hrana 3-5, od toho okamihu už bolo len 5 oblasti. Ako oheň postupoval, oblastí postupne ubúdalo, až nakoniec horel celý strom a už neexistovali žiadne nehoriace oblasti.*

vstup

```
7 1
0 1 1 1 3 3
10 8 4 6 2 4
0
```

výstup

```
4
```



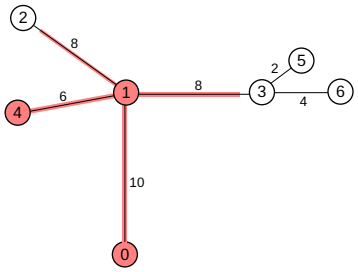

*Ten istý strom ako v predchádzajúcom príklade, ale začal horieť len vo vrchole 0. Zo začiatku je len jedna nehoriaca oblasť – všetko, kam ešte neprišli plamene, drží po kope. Po 10 sekundách sa oheň dostane do vrcholu 1, od tej chvíle máme 3 samostatné nehoriace oblasti. Na obrázku je situácia po zhruba 11 sekundách, ešte stále v nej vidíme tri oblasti.*
*Po 14 sekundách od začiatku oheň po hrane 1-3 dosiahne vrchol 3. Od tejto chvíle máme až 4 samostatné oblasti, každú tvorí časť jednej hrany. Od tejto chvíle už budú oblasti len ubúdať.*

vstup

```
7 1
0 1 1 1 3 3
10 8 8 6 2 4
0
```

výstup

```
3
```




*Ten istý strom ako v predchádzajúcom príklade, len tentokrát má hrana 1-3 dĺžku až 8. V 16. sekunde požiaru sa oheň dostane do vrcholu 4 a tým počet nehoriacich oblastí klesne z 3 na 2. Na obrázku je situácia niekedy okolo 17. sekundy.*
*Po 18 sekundách požiaru sa oheň naraz dostane do vrcholov 2 a 3. Tým ďalšia oblasť zmizne (dohorela celá hrana 1-2) a zároveň sa iná doteraz súvislá oblasť (doteraz obsahujúca vrcholy 3, 5, 6 a časť hrany 1-3) rozpadne na dve menšie.*

## A-III-3 Pokazený rover spracúva polia

*K tejto úlohe patrí študijný text uvedený na nasledujúcich stranách. Obsahuje všetko, čo obsahoval v krajskom kole, a navyše obsahuje časť zadania krajského kola, v ktorej definujeme kódovanie postupností do čísel.*
*Na konci študijného textu nájdete pridaný prehľad makier počítajúcich funkcie, ktoré sme si naprogramovali v domácom a krajskom kole. Aj tieto môžete používať pri riešení aktuálnych súťažných úloh.*

**Podúloha A (2 body).**
V lokalite A máme neznámy počet kamienkov, ktorý označíme $a$. Napíšte makro `prvocislo A B` pre rover, ktoré do lokality B uloží kamienok práve vtedy, ak $a$ je prvočíslo. V lokalite A musí na konci byť pôvodná hodnota $a$.

**Podúloha B (3 body).**
V lokalite X je $x$ kamienkov. Číslo $x$ je kódom nejakej postupnosti $P_x$. V lokalite A je $a$ kamienkov. Napíšte makro `append X A` pre rover, ktoré **na koniec** postupnosti uloženej v lokalite X pridá prvok s hodnotou $a$.
Formálne, nech $y$ je kód postupnosti, ktorú dostaneme, ak na koniec postupnosti $P_x$ pridáme prvok $a$. Napr. ak $x$ je kódom postupnosti $(4, 7, 0)$ a $a = 13$, tak $y$ má byť kódom postupnosti $(4, 7, 0, 13)$. Keď vaše makro skončí, v lokalite X by malo byť $y$ kamienkov.

**Podúloha C (5 bodov).**
V lokalite X je $x$ kamienkov. Číslo $x$ je kódom nejakej postupnosti $P_x$. Napíšte makro `sort X` pre rover, ktoré túto postupnosť usporiada.
Formálne, označme $y$ počet kamienkov v lokalite X keď vaše makro skončí. Hodnota $y$ musí byť kódom postupnosti, ktorá vznikne usporiadaním $P_x$. Napr. ak $x$ je kódom postupnosti $(4, 7, 0, 4)$, tak $y$ by malo byť kódom postupnosti $(0, 4, 4, 7)$.

Pripomíname, že pri hodnotení vašich riešení tejto úlohy nám nebude záležať na ich časovej zložitosti.

## 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 predchádzajúcich kôl

Nasleduje zoznam makier, ktoré počítajú funkcie, ktoré sme si naprogramovali v riešeniach domáceho a krajského 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

MAKRO spoj X Y Z           # do Z ulozi hodnotu (x+y)(x+y+1)/2 + x ktora je kodom dvojice (x,y)
MAKRO prvy Z X             # ak Z obsahuje kod dvojice (x,y), toto makro do X ulozi x
MAKRO druhy Z Y            # ak Z obsahuje kod dvojice (x,y), toto makro do Y ulozi y
MAKRO dlzka Z X            # do X ulozi dlzku postupnosti, ktorej kodom je cislo v Z
```

Ako sme sa už naučili v krajskom kole, pomocou makier `spoj`, `prvy` a `druhy` vieme dve čísla zakódovať do jedného a následne ich opäť dekódovať. Následne sme sa naučili, že dokonca vieme do jedného čísla zakódovať ľubovoľne dlhú konečnú postupnosť čísel. Toto robíme 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 $1 + spoj(x_0, k)$.

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 2022