71. ročník Matematické olympiády – 2021/2022

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.** Ve všech úlohách 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.

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 Hřiště

Vládce Absurdistánu rozhodl, že se jeho země stane novou světovou velmocí v metlobalu. K dosažení tohoto cíle bude ale zapotřebí mít v zemi nějaké hřiště. Hřiště na metlobal musí být obdélníkového tvaru, musí mít strany ve směru hlavních světových stran a ve dvou protilehlých rozích musí stát sloup umělého osvětlení.

Sloupy umělého osvětlení jsou momentálně zcela vyprodané. Naštěstí jich ale v Absurdistánu už n stojí. Budeme tedy muset využít dva z těchto n sloupů. Sloupy jsou už dost staré a při jakémkoliv přemístění by se mohly poškodit. Necháme je proto stát tam, kde právě stojí, a hřiště postavíme tak, aby mělo dva ze sloupů ve svých protilehlých rozích. Na hřišti pochopitelně nesmí trčet žádné další sloupy s osvětlením, aby se hráči nezranili.

Soutěžní úloha

Je dáno n bodů v rovině. Můžete předpokládat, že *žádné dva body nemají stejnou vodorovnou ani svislou souřadnici.*

Navrhněte algoritmus, který zjistí, kolika způsoby můžeme v rovině vyznačit obdélník, jehož strany jsou rovnoběžné s osami souřadnic, ve dvou protilehlých vrcholech má dva ze zadaných bodů a uvnitř obdélníka žádné ze zadaných bodů neleží (tzn. zbývajících $n - 2$ bodů leží vně obdélníka).

Formát vstupu a výstupu

Na prvním řádku vstupu je jedno celé číslo n. Každý ze zbývajících n řádků vstupu obsahuje souřadnice x_i, y_i jednoho z bodů. Všechna x_i jsou navzájem různá a všechna y_i jsou také navzájem různá.

Na výstup vypište jedno číslo: počet možných obdélníků.

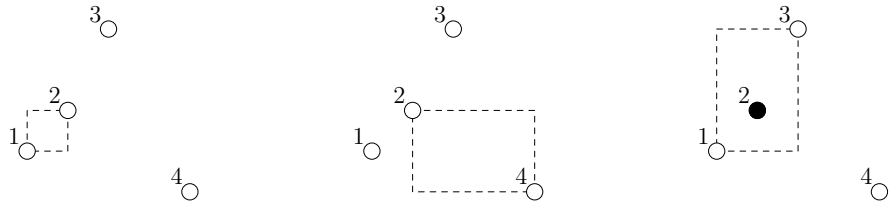
Příklad

Vstup:	*Výstup:*

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

Kdybychom vybrali první a třetí sloup osvětlení, stál by na hřišti druhý sloup, tato možnost tedy nevyhovuje. Všech pět zbývajících možností je v pořádku.

Na levém a prostředním obrázku jsou dva z pěti dobrých obdélníků. Na pravém obrázku je jediný nevyhovující obdélník: obdélník, který má v protilehlých rozích sloupy 1 a 3 nevyhovuje, protože obsahuje sloup 2.



Bodování

Za řešení, které efektivně vyřeší vstupy pro $n \leq 500$, můžete získat nejvýše 2 body.

Za řešení, které efektivně vyřeší vstupy pro $n \leq 5000$, můžete získat 6–7 bodů, přičemž maximum závisí na přesné asymptotické časové složitosti tohoto řešení.

Za řešení, které efektivně vyřeší vstupy pro $n \leq 100\,000$, můžete získat 10 bodů.

P-III-2 Hořící strom

Na louce stojí krásný košatý strom. Má n vrcholů očíslovaných od 0 do $n-1$, mezi kterými vedou hrany. Každá hrana má nějakou kladnou délku. (Vrcholy jsou všechny konce stromu a všechna místa, kde se strom větví. Hrany jsou jednotlivé větve a kusy kmenu stromu.)

Do stromu právě udeřil blesk a některé vrcholy stromu začaly hořet. Oheň se šíří spojitě po všech hranách stromu, a to rychlostí 1 cm za sekundu. Když tedy začne hořet nějaký vrchol, o 4.7 sekundy později hoří všechno ve vzdálenosti nejvýše 4.7 cm od něho.

Pokud se na strom podíváme v nějakém konkrétním okamžiku, vidíme, že některé souvislé části stromu už hoří a jiné souvislé části stromu ještě nehoří. Každou maximální souvislou část stromu, která ještě nehoří, nazveme *oblast*. Bude nás zajímat počet těchto oblastí.

Soutěžní úloha

Je dán popis stromu a seznam vrcholů, kde strom začal hořet (ve všech těchto vrcholech začal hořet zároveň). Napište program, který zjistí, na kolik nejvýše nehořících oblastí byl strom rozdělen během doby, kdy hořel.

Formát vstupu a výstupu

Na prvním řádku vstupu jsou dvě celá čísla n a h: počet vrcholů stromu a počet těch z nich, které na začátku hoří.

Druhý a třetí řádek vstupu popisují tvar stromu. Na druhém řádku jsou uvedena celá čísla $p_1, \ldots, p_{n-1}$, pro něž platí $0 \leq p_i < i$. Na třetím řádku vstupu jsou kladná celá čísla $d_1, \ldots, d_{n-1}$. Pro každé i platí, že jedna z hran stromu spojuje vrcholy i a p_i. Tato hrana má délku d_i cm.

Na posledním řádku vstupu jsou čísla $z_1, \ldots, z_h$: čísla vrcholů, které začínají hořet hned na začátku.

Na výstup vypište jediné číslo: maximální počet disjunktních nehořících oblastí, které existovaly někdy během požáru stromu.

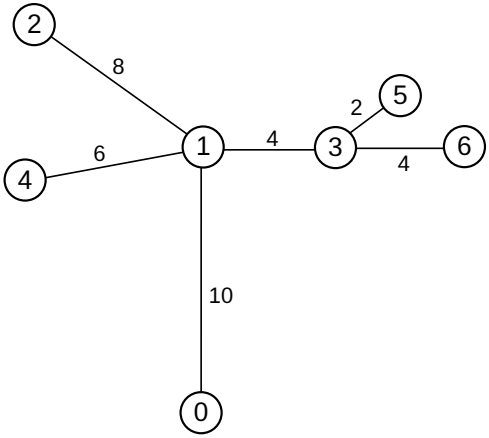
Příklady

Vstup:

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

Výstup:

```
6
```



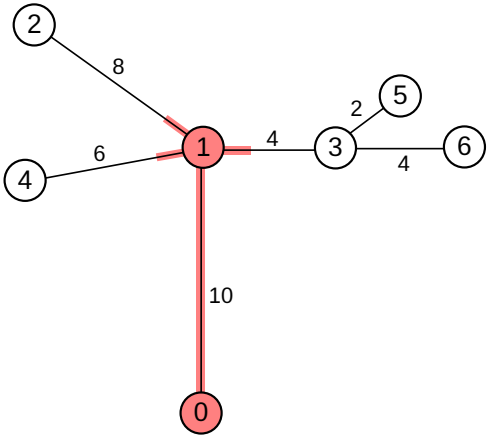
Toto je 7-vrcholový strom, který začal najednou hořet ve všech vrcholech. Těsně poté, jak začal hořet, existovalo 6 nehořících oblastí: jedna na každé hraně. Po 1 sekundě už hořela celá hrana 3–5, od tohoto okamžiku už bylo jen 5 oblastí. Jak oheň postupoval, oblastí postupně ubývalo, až nakonec hořel celý strom a už neexistovaly žádné nehořící oblasti.

Vstup:

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

Výstup:

```
4
```



Tentýž strom jako v předchozím příkladu, ale začal hořet jenom ve vrcholu 0. Ze začátku je jen jedna nehořící oblast – všechno, kam se ještě nedostaly plameny, drží pohromadě. Po 10 sekundách se oheň dostane do vrcholu 1, od té chvíle máme 3 samostatné nehořící oblasti. Na obrázku je znázorněna situace po zhruba 11 sekundách, ještě stále v ní vidíme tři oblasti.

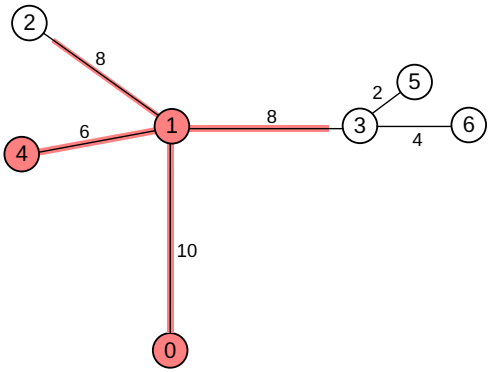
Po 14 sekundách od začátku se oheň dostane po hraně 1–3 do vrcholu 3. Od této chvíle máme 4 samostatné oblasti, každou tvoří část jedné hrany. Později už budou nehořící oblasti pouze ubývat.

Vstup:

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

Výstup:

```
3
```



Stejný strom jako v předchozím příkladu, pouze hrana 1–3 má tentokrát délku 8. V 16. sekundě požáru se oheň dostane do vrcholu 4 a tím počet nehořících oblastí klesne ze 3 na 2. Na obrázku je zachycena situace někdy kolem 17. sekundy.

Po 18 sekundách požáru se oheň dostane zároveň do vrcholů 2 a 3. Tím další oblast zanikne (dohořela celá hrana 1–2) a zároveň se jiná dosud souvislá oblast (obsahující vrcholy 3, 5, 6 a část hrany 1–3) rozpadne na dvě menší.

Bodování

Plný počet bodů můžete získat za řešení, které úlohu efektivně vyřeší pro vstupní data $n \leq 200\,000$ a $d_i \leq 10^9$, a to i za řešení, které bude mít asymptotickou časovou složitost mírně horší než vzorové řešení.

Až 6 bodů získáte za řešení, které úlohu efektivně vyřeší pro $n \leq 5\,000$ a $d_i \leq 10^9$.

Nejvýše 3 body dostanete za řešení, které úlohu efektivně vyřeší pro $n \leq 50$ a $d_i \leq 10$.

Pro jednoduchost můžete předpokládat, že všechna čísla d_i jsou sudá.

P-III-3 Vozidlo zpracovává pole

K této úloze se vztahuje studijní text uvedený na následujících stranách. Doporučujeme vám nejprve si přečíst studijní text a až potom se vrátit k samotným soutěžním úkolům. Text je zcela shodný se studijním textem, který znáte již z domácího a krajského kola soutěže. Navíc obsahuje část zadání úlohy z krajského kola, v níž definujeme kódování posloupností do čísel.

Na konci studijního textu najdete přidaný přehled maker, která počítají funkce naprogramované v domácím a krajském kole. Můžete je používat při řešení soutěžních úloh ústředního kola.

a) *(2 body)*

V lokalitě `A` máme neznámý počet kamenů, který označíme a. Napište pro vozidlo makro `prvočíslo A B`, které do lokality `B` uloží kámen právě tehdy, je-li a prvočíslo. V lokalitě `A` musí na konci zůstat zachována původní hodnota a.

b) *(3 body)*

V lokalitě `X` je x kamenů. Číslo x je kódem nějaké posloupnosti P_x. V lokalitě `A` je a kamenů. Napište pro vozidlo makro `append X A`, které *na konec* posloupnosti uložené v lokalitě `X` přidá prvek s hodnotou a.

Formálně, nechť y je kód posloupnosti, kterou dostaneme, když na konec posloupnosti P_x přidáme prvek a. Například je-li x kódem posloupnosti $(4, 7, 0)$ a $a = 13$, potom y bude kódem posloupnosti $(4, 7, 0, 13)$. Když vaše makro skončí, v lokalitě `X` musí být y kamenů.

c) *(5 bodů)*

V lokalitě `X` je x kamenů. Číslo x je kódem nějaké posloupnosti P_x. Napište pro vozidlo makro `sort X`, které tuto posloupnost uspořádá.

Formálně, označme y počet kamenů v lokalitě `X` po skončení vašeho makra. Hodnota y musí být kódem posloupnosti, která vznikne uspořádáním posloupnosti P_x. Například je-li x kódem posloupnosti $(4, 7, 0, 4)$, potom y musí být kódem posloupnosti $(0, 4, 4, 7)$.

Připomínáme, že při hodnocení této úlohy *nezáleží na časové složitosti* zvoleného řešení.

Studijní text

Na Mars jsme dopravili průzkumné terénní vozidlo. Měli jsme s ním velké plány, ale zasáhla ho písečná bouře a zničila mu skoro všechna periferní zařízení, takže vozidlo zůstalo téměř nepoužitelné. Nyní se může jenom přesouvat mezi různými lokalitami a vozit z místa na místo kameny. Naším úkolem bude naučit vozidlo provádět alespoň některé použitelné výpočty. **Nebude nám přitom záležet na časové složitosti** programů, pouze na jejich korektnosti.

Do vozidla můžeme na dálku nahrát program: konečnou posloupnost příkazů. Některé příkazy mohou mít návěstí (labels) – symbolická jména, pomocí nichž se na příkazy můžeme odkazovat.

Vozidlo zná na Marsu dvě speciální lokality: *jídelnu* a *kamenolom.* Pro jednoduchost je budeme značit `J` a `K`. V jídelně je momentálně právě jeden kámen, jinak je to lokalita jako každá jiná. V kamenolomu je vždy k dispozici dostatečné množství kamenů.

Ke každému jinému řetězci má vozidlo přiřazenu nějakou unikátní lokalitu na Marsu, kam je možné ukládat kameny. Není-li řečeno jinak, předpokládáme, že všechny tyto lokality jsou prázdné – neobsahují žádné kameny.

Vozidlo umí provádět jediný příkaz, který vypadá následovně:

1. Jeď do lokality `Y`. Spočítej si do pomocné proměnné, kolik je tam kamenů.
2. Jeď do lokality `X`. Zkus tam naložit tolik kamenů, kolik udává pomocná proměnná.
3. Pokud se to podařilo, odvez tyto kameny do lokality `Z`, tam je vysyp a pokračuj následujícím příkazem.
4. Pokud se to nepodařilo (tzn. v lokalitě `X` není dost kamenů), uveď lokalitu `X` do původního stavu a pokračuj příkazem s návěstím `N`.

Program, který nahrajeme do vozidla, bude tedy posloupností takovýchto příkazů. Příkaz včetně návěstí budeme zapisovat takto:

```
návěstí: přenes X Y Z N
```

Jednotlivé parametry tohoto příkazu můžeme číst následovně:

```
přenes (odkud) (kolik) (kam) (co dělat při neúspěchu)
```

Na místě čtvrtého parametru můžeme napsat `-` (pomlčku), jestliže chceme, aby výpočet programu i v případě neúspěchu pokračoval následujícím příkazem.

Za `X`, `Y` a `Z` můžete dosadit libovolné lokality, dokonce nemusí být navzájem různé. Jediným omezením je, že kamenolom (kde je „nekonečně mnoho“ kamenů) nesmíme použít jako lokalitu `Y`.

Výpočet programu skončí, když se dostane na neexistující příkaz – tedy buď provedeme poslední příkaz programu a máme pokračovat následujícím příkazem, nebo skočíme na neexistující návěstí.

Příklad 1: příkazy, které skoro nic nedělají

Co udělá vozidlo, když mu dáme příkaz `přenes X X X I`? Spočítá kameny v lokalitě `X`, potom je všechny naloží, na stejném místě je zase všechny vysype a bude pokračovat následujícím příkazem. Tímto příkazem tedy Mars vůbec nezměníme.

Co udělá vozidlo, když mu dáme příkaz `přenes X Y X I`? Také tentokrát se počet kamenů v žádné lokalitě nezmění, existují ale dva možné průběhy výpočtu: jestliže bylo v lokalitě `Y` více kamenů než v lokalitě `X`, výpočet bude pokračovat příkazem s návěstím `I`.

Příklad 2: vyprázdni lokalitu

Rozmyslete si sami, jakým příkazem můžeme z lokality odstranit všechny kameny.

Řešení: Pro vyprázdnění lokality `X` můžeme použít příkaz `přenes X X K -`. Vozidlo spočítá kameny v lokalitě `X`, všechny je naloží a vysype je v kamenolomu.

Příklad 3: naučíme vozidlo sčítat

V lokalitách `A` a `B` máme nějaké neznámé počty kamenů, lokalita `C` je prázdná. Chtěli bychom mít v lokalitě `C` tolik kamenů, kolik jich je v lokalitách `A` a `B` dohromady. Lokality `A` a `B` přitom chceme ponechat beze změny.

Dříve než si přečtete řešení, zkuste ho opět sami vymyslet.

Řešení: Stačí příslušné počty kamenů převézt z kamenolomu do lokality `C`. To zajistíme následujícím programem:

```
přenes K A C -
přenes K B C -
```

Příklad 4: naučíme vozidlo odčítat

V lokalitách `A` a `B` máme nějaké neznámé počty kamenů, které označíme a a b. Lokalita `C` je prázdná. Chtěli bychom mít v lokalitě `C` počet kamenů rovný $a - b$, resp. rovný nule, pokud $b > a$. Lokality `A` a `B` přitom chceme ponechat beze změny.

(Na rozdíl od příkazu `přenes`, který odebere kameny pouze tehdy, pokud může vzít celý požadovaný počet, při našem odčítání chceme odebrat vždy tolik kamenů, kolik je možné.)

Řešení: Do lokality `C` přivezeme z kamenolomu a kamenů a potom se pokusíme odebrat z nich b. Když se nám to podaří, jsme hotovi, jinak všechny kameny z lokality `C` odvezeme zpět do kamenolomu.

```
            přenes K A C -
            přenes C B K nulování
            přenes prázdná J K konec
nulování:   přenes C C K -
```

Všimněte si, že ve třetím kroku (který se provádí, jestliže jsme z lokality `C` úspěšně odebrali b kamenů) se pokusíme z prázdné lokality odvézt jeden kámen. To se nám zaručeně nepodaří, program proto skočí na neexistující návěstí „konec“ a tím

výpočet skončí. Čtvrtý příkaz se tedy provede jenom tehdy, když na něj skočíme z druhého příkazu.

Příklad 5: naučíme vozidlo násobit

V lokalitách `A` a `B` máme nějaké neznámé počty kamenů, které označíme a a b. Lokalita `C` je prázdná. Chtěli bychom mít v lokalitě `C` počet kamenů rovný $a \cdot b$. Lokality `A` a `B` přitom chceme ponechat beze změny.

Toto už nedokážeme provést v konstantním čase. Násobení je ale jenom opakované sčítání: hromadu ab kamenů získáme tak, že na ni a-krát přivezeme b kamenů.

```
        přenes K A Akopie -
cyklus: přenes Akopie J K konec
        přenes K B C -
        přenes prázdná J K cyklus
```

Nejprve si vytvoříme kopii lokality `A`, kterou potom během výpočtu zničíme. Dále opakujeme, dokud to jde: vezmi jeden kámen z lokality `Akopie` a přidej b kamenů do lokality `C`.

Makra

Při programování našeho průzkumného vozidla můžeme definovat makra: vezmeme nějakou posloupnost příkazů a přiřadíme jí symbolické jméno, abychom nemuseli tutéž posloupnost příkazů rozepisovat vícekrát. Makro může mít parametry – při různých použitích takového makra se budou provádět stejné příkazy, ale pro jiné lokality a jiná návěstí.

Uvnitř makra můžeme používat také pomocné lokality. Lokality, které zapíšeme v definici makra do hranatých závorek, budou při každém použití makra nahrazeny jinou lokalitou, která se nikde jinde v programu nepoužívá.

Totéž se automaticky stane i se všemi návěstími, která dáme příkazům v definici makra. Každé takové návěstí je tedy viditelné jen z jednoho konkrétního použití makra.

V definici makra můžeme používat i jiná makra, která jsme definovali dříve.

Ukážeme si nyní několik příkladů definice makra. Řádky začínající mřížkou obsahují komentáře.

```
# příkaz, který nic nezmění, pouze jeden krok čeká
MAKRO čekej
        přenes J J J -
KONEC

# makro se dvěma parametry - lokalitami:
# do Y přidá tolik kamenů, kolik jich je v X
MAKRO přidej X Y
        přenes K X Y -
KONEC

# makro s jedním parametrem - návěstím: skočí na dané návěstí
MAKRO skoč N
        přenes [nová_prázdná_lokalita] J K N
KONEC
```

```
# makro pro násobení: do Z přidá součin počtů kamenů v X a Y
MAKRO vynásob X Y Z
          přidej X [Xkopie]
  cyklus: přenes [Xkopie] J K konec
          přenes K Y Z -
          skoč cyklus
   konec: čekej
KONEC
```

Všimněte si, že když jsme původně psali samostatný program pro násobení, stačilo nám, že návěstí `konec` neexistuje a skokem na něj jsme ukončili program. Při psaní makra toto návěstí umístíme před prázdný příkaz na konci makra, neboť chceme, aby po ukončení násobení program pokračoval dále ve výpočtu.

Příklad 6: naučíme vozidlo počítat třetí mocninu

Pomocí výše uvedených maker snadno napíšeme program, který bude počítat třetí mocninu. V lokalitě `A` máme nějaký neznámý počet kamenů, který označíme a. Lokalita `B` je prázdná a chceme do ní dovézt a^3 kamenů.

```
vynásob A A pomocná_lokalita
vynásob A pomocná_lokalita B
```

Pro názornost ještě ukážeme, jak by vypadal stejný program, kdybychom všechna makra nahradili jejich definicemi.

```
          přenes K A Akopie1 -
 cyklus1: přenes Akopie1 J K konec1
          přenes K A pomocná_lokalita  -
          přenes cokoliv1 J K cyklus1
  konec1: přenes J J J -
          přenes K A Akopie2 -
 cyklus2: přenes Akopie2 J K konec2
          přenes K pomocná_lokalita B -
          přenes cokoliv2 J K cyklus2
  konec2: přenes J J J -
```

Řešení předcházejících kol

Následuje seznam maker, která počítají funkce naprogramované v řešeních domácího a krajského kola. Tato makra můžete používat ve svých řešeních bez nutnosti rozepisovat jejich implementaci.

```
MAKRO vynuluj X          # vynuluje obsah X
MAKRO nulové X N         # pokud X obsahuje nulu, skočí na návěstí N
MAKRO stejné X Y N       # pokud jsou hodnoty X a Y stejné, skočí na návěstí N
MAKRO zapiš X Y          # hodnotu X zapíše do lokality Y; původní obsah Y přepíše
MAKRO vyděl X Y P Z      # hodnotu X vydělí hodnotou Y, do P uloží podíl, do Z zbytek
MAKRO nedělitelné X Y N  # pokud X není dělitelné hodnotou Y, skočí na návěstí N
MAKRO nsd X Y Z          # do Z uloží největšího společného dělitele čísel X a Y

MAKRO spoj X Y Z         # do Z uloží (x+y)(x+y+1)/2 + x, tedy kód dvojice (x,y)
MAKRO první Z X          # když Z obsahuje kód dvojice (x,y), toto makro do X uloží x
MAKRO druhý Z Y          # když Z obsahuje kód dvojice (x,y), toto makro do Y uloží y
MAKRO délka Z X          # do X uloží délku posloupnosti, jejímž kódem je číslo v Z
```

V krajském kole jsme se naučili, jak pomocí maker `spoj`, `první` a `druhý` můžeme dvě čísla zakódovat do jednoho a následně je opět dekódovat. Dále jsme se naučili, že do jednoho čísla dokážeme zakódovat dokonce libovolně dlouhou konečnou posloupnost čísel. To provedeme následovně:

- Prázdné posloupnosti čísel přiřadíme jako kód číslo 0.
- Předpokládejme, že už známe kód k posloupnosti $x_1, \ldots, x_n$. Kódem posloupnosti $x_0, x_1, \ldots, x_n$ bude hodnota $1 + \mathit{spoj}(x_0, k)$.