71. ročník Matematické olympiády – 2021/2022

Úlohy krajského kola kategorie P

Krajské kolo 71. ročníku MO kategorie P se koná v úterý 18. 1. 2022 v dopoledních hodinách. Na řešení úloh máte 4 hodiny čistého času. V krajském kole MO-P se neřeší žádná praktická úloha, pro zajištění rovných podmínek řešitelů ve všech krajích je použití počítačů při soutěži zakázáno (s výjimkou nahrávání řešení do odevzdávacího systému). I v případě distanční formy soutěže ve vašem kraji je tedy zakázáno psát řešení přímo na počítači. Zakázány jsou rovněž jakékoliv další pomůcky kromě psacích potřeb (např. knihy, výpisy programů, kalkulačky, mobilní telefony).

Řešení každé úlohy vypracujte na samostatný list papíru. V případě distanční organizace krajského kola poté řešení oskenujte či nafoťte a nejpozději do 20 minut po skončení soutěže nahrajte ve formátu PDF do odevzdávacího systému, který najdete na *https://mo.mff.cuni.cz/osmo/*.

Řešení každé úlohy musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, *argumenty zdůvodňující jeho správnost* (případně důkaz správnosti algoritmu), diskusi o efektivitě vašeho řešení (časová a paměťová složitost; to se netýká úlohy P-II-4). Slovní popis řešení musí být jasný a srozumitelný i bez nahlédnutí do samotného zápisu algoritmu (do programu). Není možné odkazovat se na vaše řešení úloh domácího kola, opravovatelé je nemusí mít k dispozici.
- **Zápis algoritmu.** Ve všech úlohách je třeba uvést zápis algoritmu v nějakém dostatečně srozumitelném pseudokódu (případně v programovacím jazyce Pascal, C/C++ nebo Python). 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 a efektivita zvoleného algoritmu. Algoritmy posuzujeme podle jejich časové složitosti, tzn. 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.

Vzorová řešení úloh naleznete krátce po soutěži na webových stránkách olympiády *https://mo.mff.cuni.cz/*. Na stejném místě bude zveřejněn i seznam úspěšných řešitelů krajského kola a seznam řešitelů postupujících do ústředního kola.

P-II-1 Halušky

Michal je vášnivý turista a úloha z domácího kola v něm vzbudila touhu znovu podniknout výlet do Tater. Tatry si pro účely úlohy představme jako n míst očíslovaných čísly $0, 1, \ldots, n-1$, z nichž mezi některými dvojicemi se dá obousměrně pohybovat po vyznačených stezkách. A protože jsou to vysokohorské stezky, na každé z nich se nacházejí i různé žebříky, řetězy a kramle, které mají omezenou nosnost. U každé stezky Michal ví, kolik nejvíce kilogramů může člověk i s vybavením vážit, aby ji mohl bezpečně projít.

Michal je zároveň vášnivý milovník halušek. Pro každé místo v Tatrách si předem zjistil, jak dobré tam mají halušky, a tudíž kolik kilogramů minimálně přibere, pokud toto místo navštíví.

Jediné, co Michal zatím naplánoval, je, že túru začne v místě 0 a zakončí v místě $n-1$. Poraďte Michalovi, kolik maximálně může i s výbavou vážit, až bude začínat v místě 0, aby se mohl dostat do místa $n-1$ a dodržel přitom hmotnostní omezení na všech stezkách, které projde.

Soutěžní úloha

Dostanete graf o n vrcholech číslovaných $0, 1, \ldots, n-1$ a m hranách reprezentující místa v Tatrách a stezky mezi nimi. K vrcholu i dostanete číslo h_i, totiž minimální počet kilogramů, které Michal určitě přibere, pokud do tohoto vrcholu (místa) vstoupí (platí $h_0 = h_{n-1} = 0$). Ke každé hraně dostanete číslo c_i, totiž nejvyšší hmotnost, s níž je bezpečné touto hranou (stezkou) projít. Najděte největší W takové, že existuje cesta z vrcholu 0 do vrcholu $n-1$, jíž může Michal bezpečně projít tehdy, když začne ve vrcholu 0 s hmotností W.

Formát vstupu

Na prvním řádku dostanete čísla n a m: počet vrcholů (míst) a počet hran (stezek). Na druhém řádku dostanete n nezáporných celých čísel $h_0, \ldots, h_{n-1}$, kde h_i je hmotnost, kterou Michal ve vrcholu i přibere. Platí $h_0 = h_{n-1} = 0$. Zbytek vstupu tvoří m řádků, každý z nich obsahuje tři čísla a_j, b_j, c_j popisující, že mezi vrcholy a_j a b_j vede oboustranná stezka, jíž lze bezpečně projít s hmotností nejvýše c_j.

Formát výstupu

Vypište $W > 0$, totiž největší hmotnost, s níž může Michal v místě 0 začít tak, aby existovala bezpečná cesta do místa $n-1$. Pokud žádné takové kladné W neexistuje, vypište místo toho -1.

Příklady

Vstup:	*Výstup:*
3 2 0 100 0 0 1 470 1 2 100	-1

Tatry tvoří tři místa, stezka s vysokou nosností mezi místy 0 a 1 *a stezka s nízkou nosností mezi místy* 1 a 2. *Na místě* 1 *přibere Michal 100 kg. Proto nemá možnost se do cíle dostat: I kdyby začal s hmotností* 1 *kg, jakmile přijde do místa* 1, *bude vážit více než* 100 *kg, a tudíž nebude moci bezpečně projít druhou stezkou.*

Vstup:	*Výstup:*
3 2 0 47 0 0 1 470 1 2 100	53

Do Tater přijede 53kilogramový Michal, v místě 1 přibere 47 kg a ještě bude těsně moci bezpečně projít stezkou na místo 2.

Vstup:	*Výstup:*
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	117

Michal může jít cestami 0–1–3–4 nebo 0–2–3–4, při nichž přibere jen 20 kg. Nejlepší možností ale je, možná překvapivě, zvolit cestu 0–1–2–3–4 a přibrat 30 kg. Stezky na této cestě mají totiž o něco větší nosnost, a tudíž může Michal na začátku vážit o trochu více.

Bodování

Ve všech vstupech platí $n \geq 2$.

Plný počet bodů získáte za řešení, která efektivně vyřeší vstupy, kde $n \leq 200\,000$, $m \leq 500\,000$, $h_i \leq 10^9$ pro každé $0 \leq i < n$ a $c_j \leq 10^9$ pro každé $0 \leq j < m$.

Až 8 bodů získáte za řešení, která efektivně vyřeší vstupy, kde $n \leq 50\,000$, $m \leq 200\,000$, $h_i \leq 10^6$ pro každé $0 \leq i < n$ a $c_j \leq 10^6$ pro každé $0 \leq j < m$.

Až 6 bodů získáte za řešení, která efektivně vyřeší vstupy s $n \leq 5\,000$, $m \leq 20\,000$, $h_i \leq 10^6$ pro každé $0 \leq i < n$ a $c_j \leq 10^6$ pro každé $0 \leq j < m$.

Až 4 body získáte za řešení, která efektivně vyřeší vstupy, kde $n \leq 500$, $m \leq 1000$, $h_i \leq 100$ pro každé $0 \leq i < n$ a $c_j \leq 100$ pro každé $0 \leq j < m$.

P-II-2 Červené jablíčko

Pepa má 1000 jablek a ví, že jedno z nich je červené a všechna ostatní jsou zelená. Jelikož je však od narození červeno-zeleně barvoslepý, všechna jablka pro něj vypadají stejně. Už od rána má chuť na červené jablko a naštěstí k němu právě na návštěvu dorazil Vašek, který červené jablko od zelených rozlišit umí.

Vašek chce ale Pepu trochu potrápit, a tak se rozhodl, že mu neřekne přímo, které jablko je červené. Místo toho může Pepa ukázat na kteroukoliv množinu jablek a Vašek mu prozradí, zda mezi nimi je to červené. Aby to měl Pepa ještě těžší, Vašek mu řekl, že může v **nejvýše jedné odpovědi** lhát.

Soutěžní úloha

Pepa musí zjistit, které z jeho jablek je to jediné červené, a to tak, že postupně bude ukazovat na nějaké množiny jablek a ptát se Vaška, zda mezi nimi to červené je. Speciálně tedy každou další množinu může zvolit podle toho, jak Vašek do té doby odpovídal.

Na toto může Pepa zvolit veliké množství různých strategií. Nás budou zajímat pouze ty *korektní,* tj. takové, u nichž je zaručené, že si Pepa na konci bude jistý, které jablko je červené (a bude si jistý správně). Při vyhodnocování toho, jak dobrá nějaká strategie je, nás bude zajímat její *nejhorší možný případ,* tj. největší počet otázek, které může Pepa potřebovat, než najde správné jablko. Čím je tento počet nižší, tím je strategie lepší.

Najděte co nejlepší korektní strategii pro Pepu. Tuto strategii dostatečně exaktně popište (např. formou pseudokódu), samozřejmě nezapomeňte ani na zdůvodnění, že je strategie korektní, a na odvození, kolik otázek v nejhorším případě potřebuje.

Příklad komunikace

Pepa *(s jedním jablkem v každé ruce):* Je jedno z těchto dvou jablek červené?

Vašek: Ano.

Pepa *(ukazuje jablko v pravé ruce):* Je toto jablko červené?

Vašek: Ne.

Pepa *(znovu ukazuje jablko v pravé ruce):* Je toto jablko červené?

Vašek: Ano.

Pepa *(ukazuje jablko v levé ruce):* Je toto jablko červené?

Vašek: Ne.

Pepa: Jsem si jistý, že jablko v mé pravé ruce je červené. Potřeboval jsem 4 otázky.

Neúplný příklad popisu nějaké (ne nutně korektní či dobré) strategie

1. Nachystej si prázdnou kapsu.
2. Dokud máš nějaké jablko mimo kapsu, opakuj:
3. Vezmi nějaká dvě jablka mimo kapsu do ruky a ukaž je Vaškovi.
4. Zeptej se, zda je některé z nich červené.

```
5.      Pokud odpoví NE:
6.          Dej obě do kapsy.
7.      Jinak:
8.          ...
```

Bodování

Počet bodů, které dostanete, záleží na hodnotě x, totiž počtu otázek, které vaše strategie potřebuje položit v nejhorším případě. Čím menší bude x, tím více bodů můžete získat.

Plný počet bodů můžete získat za korektní strategii s $x \leq 14$.

Za korektní strategii s $x \leq 21$ můžete získat až 6 bodů.

Až 3 body získáte za libovolnou korektní strategii.

P-II-3 Turbojezdec

Jezdec je klasická šachová figurka. Pohybuje se tak, že ho zvedneme, posuneme jej o dvě políčka některým směrem (v řádku nebo sloupci), potom o jedno políčko tím druhým směrem a tam ho položíme.

Turbojezdec se pohybuje podobně jako jezdec, jen rychleji: Nejprve jej posuneme o **více než dvě** políčka jedním směrem a potom o **více než jedno** políčko tím druhým.

Máme obří šachovnici s r řádky a s sloupci, na jejímž každém políčku je napsané jedno písmeno. Chceme po ní přeskákat turbojezdcem tak, aby se postupně zastavil přesně na písmenkách daného slova w. (Začít můžeme na libovolném políčku obsahujícím první písmeno w, každým dalším tahem se musíme přesunout na políčko obsahující následující písmeno slova w.) Kolika různými způsoby to můžeme udělat?

Jelikož správná odpověď může být opravdu velká, počítejte pouze její zbytek po dělení číslem $10^9 + 7$.

Formát vstupu

Na prvním řádku dostanete dvě mezerou oddělená celá čísla r a s. Na druhém řádku dostanete slovo w. Na i-tém z následujících r řádků dostanete řetězec délky s popisující, která písmena jsou na i-tém řádku šachovnice.

Formát výstupu

Vypište řádek obsahující jedno celé číslo, totiž zbytek počtu různých způsobů, jak vyskákat na šachovnici slovo w, po dělení číslem $10^9 + 7$.

Příklady

Vstup:	*Výstup:*

`3 4`
`HA`
`HLOH`
`NENI`
`AKAT`
	`1`

Na této šachovnici se turbojezdec umí pohybovat jen mezi levým horním a pravým spodním rohem a mezi levým spodním a pravým horním rohem.

Vstup:	*Výstup:*

`3 5`
`OREL`
`VASEK`
`HRAJE`
`SACHY`
	`0`

Jelikož na šachovnici není žádné O ani L, je zřejmé, že se slovo OREL poskládat nedá.

Vstup:

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

Výstup:

```
84
```

Pokud bychom řádky a sloupce číslovali od nuly zleva nahoře, jedna z přípustných možností skákání 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 přípustné řešení je začít na K úplně vlevo nahoře, odtud se na S zhruba uprostřed dole a odtud na P úplně vpravo nahoře

Bodování

Můžete předpokládat, že všechna písmena na vstupu jsou velká písmena anglické abecedy. Speciálně tedy velikost abecedy můžete považovat za konstantu.

Plný počet bodů může získat řešení, které efektivně vyřeší vstupy s $r, s \leq 1000$ a $|w| \leq 100$.

Až 8 bodů může získat řešení, které efektivně vyřeší vstupy s $r, s, |w| \leq 100$.

Až 6 bodů může získat řešení, které efektivně vyřeší vstupy s $r, s, |w| \leq 40$.

Až 3 body může získat řešení, které efektivně vyřeší vstupy, v nichž platí $r, s, |w| \leq 40$ a navíc je zaručené, že pro každé i existuje nejvíce 10^6 různých způsobů, jak po šachovnici vyskákat prvních i písmen slova w.

P-II-4 Vozidlo kóduje čísla

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 kola soutěže. Jediným rozdílem je, že na konci studijního textu najdete přidaný přehled maker, která počítají funkce naprogramované v domácím kole. Můžete je používat při řešení soutěžních úloh krajského kola.

a) *(1 bod)*

Uvažujme následující pořadí dvojic nezáporných čísel: $(0,0)$, $(0,1)$, $(1,0)$, $(0,2)$, $(1,1)$, $(2,0)$, $(0,3)$, $(1,2)$, $(2,1)$, $(3,0)$, $(0,4)$, $(1,3)$, ...

Rozmyslete si, jak v tomto pořadí pokračovat, aby obsahovalo každou dvojici čísel (x,y) právě jednou. Nalezněte matematický předpis pro funkci *spoj* takovou, že $spoj(x,y)$ určuje pozici dvojice (x,y) v uvedeném pořadí. Pozice číslujeme od nuly, takže například platí $spoj(3,0) = 9$.

b) *(2 body)*

Napište pro naše vozidlo makro `spoj X Y Z` počítající funkci *spoj* z podúlohy a).

V lokalitách `X` a `Y` jsou libovolné počty kamenů, které označíme x a y. Když makro skončí, v lokalitě `Z` má být tolik kamenů, jaká je pozice dvojice (x,y) ve výše uvedeném seznamu dvojic.

c) *(4 body)*

Napište pro vozidlo makro `první Z X`.

V lokalitě `Z` je nějaký neznámý počet kamenů, který si označíme z. Nechť (x,y) je dvojice čísel, která je v našem pořadí umístěna na pozici z. Vaše makro má zjistit hodnotu x a do lokality `X` uložit x kamenů.

d) *(3 body)*

Dosud jsme mohli každou lokalitu chápat jako proměnnou, do níž dokážeme uložit nezáporné celé číslo. Nyní pomocí funkce *spoj* dokážeme do jedné lokality uložit dvě čísla – na lokalitu, v níž je umístěno $spoj(x,y)$ kamenů, se můžeme dívat jako na proměnnou, ve které jsou uložena čísla x a y. Nám ale dvě čísla nestačí a chtěli bychom ještě více: do jedné lokality chceme uložit libovolně mnoho čísel najednou!

Uděláme to následovně:

- Prázdné posloupnosti čísel přiřadíme kód 0.
- Předpokládejme, že již známe kód k posloupnosti $x_1, \ldots, x_n$. Potom kódem posloupnosti $x_0, x_1, \ldots, x_n$ bude hodnota $1 + spoj(x_0, k)$.

Napište pro naše vozidlo makro `délka Z X`.

V lokalitě `Z` je z kamenů. Číslo z je kódem nějaké posloupnosti. Lokalita `X` je prázdná. Vaše makro má do lokality `X` uložit tolik kamenů, jaká je délka posloupnosti s kódem z.

Při psaní tohoto makra můžete používat makra `spoj` a `první`, a to i v případě, pokud jste nevyřešili předchozí podúlohy. Můžete používat také makro `druhý`, které funguje podobně jako makro `první`, jenom místo hodnoty x počítá hodnotu y.

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 domácího kola

Následuje seznam maker, která počítají funkce naprogramované v řešeních domácího 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
```