# 74. ročník Matematické olympiády – 2024/2025

*Úlohy domácího kola kategorie P*

---

Řešení úloh odevzdávejte pomocí webového rozhraní, které je dostupné na stránkách olympiády *https://mo.mff.cuni.cz/*. Tam také najdete podrobnější instrukce k odevzdávání a další informace o kategorii P.

Úlohy P-I-1 a P-I-2 jsou praktické. Vaším úkolem v nich je vytvořit a odladit efektivní program v jednom z následujících programovacích jazyků: C, C++ 17, Python 3.11, C# 11, Java 11 nebo Pascal. Řešení těchto dvou úloh odevzdávejte přes webové rozhraní ve formě zdrojového kódu. Odevzdaná řešení budou automaticky vyhodnocena pomocí připravených vstupních dat a výsledky vyhodnocení se dozvíte krátce po odevzdání. Pokud váš program nezíská plný počet 10 bodů, můžete své řešení opravit a znovu odevzdat.

Vzhledem k různé výkonnosti stejného algoritmu implementovaného v různých programovacích jazycích nezaručujeme, že je ve všech podporovaných jazycích možné získat plný počet bodů – může se stát, že program nestihne doběhnout do časového limitu, i když implementuje algoritmus s optimální časovou složitostí. Časové limity jsou (s rezervou) nastavené dle autorských řešení implementovaných v C++.

Úloha P-I-4 je také praktická, k jejímu řešení je ale potřeba využít model výpočtu a odpovídající programovací jazyk detailně popsaný ve studijním textu, který naleznete na konci zadání. Ohledně odevzdávání a vyhodnocování jinak platí obdobné instrukce, jako pro první dvě úlohy.

Úloha P-I-3 je teoretická, neodevzdává se tedy program, ale písemné řešení obsahující popis algoritmu, který tuto úlohu efektivně vyřeší. Součástí řešení také musí být zdůvodnění správnosti tohoto algoritmu a odhad jeho časové a paměťové složitosti. Do řešení nemusíte psát odpovídající program, algoritmus stačí zapsat ve vhodném pseudokódu nebo dokonce jenom slovně, je-li popis dostatečně podrobný a srozumitelný. Hodnotí se nejen správnost, ale také efektivita zvoleného postupu řešení. Řešení této teoretické úlohy odevzdávejte ve formě souboru typu PDF přes výše uvedené webové rozhraní.

Řešení všech úloh můžete odevzdávat do 15. listopadu 2024. Opravená řešení a seznam postupujících do krajského kola najdete na webových stránkách olympiády o několik týdnů poté.

## P-I-1 Vlaky

České dráhy se rozhodly zvýšit kvalitu svých služeb a přepracovat jízdní řády tak, aby cestujícím více vyhovovaly. Jednou z nejčastějších stížností cestujících bylo, že vlaky do podobně znějících stanic vyjíždí ve skoro stejný čas, a snadno se tak stane, že člověk omylem nastoupí do špatného vlaku. Na základě těchto stížností vedení ČD vypracovalo seznam požadavků ve tvaru „vlak $A$ musí odjíždět o alespoň $m$ minut později, než vlak $B$." Na vás je připravit jízdní řád splňující tyto požadavky.

Situaci vám trochu komplikuje to, že se do požadavků vloudil drobný šotek a čísla $m$ v některých požadavcích mohou být záporná. Vedení ČD ale pochopitelně trvá na tom, že požadavky už opravit nejde a je nutné je splnit; tj. je třeba stanovit časy $T_A$ a $T_B$ odjezdů vlaků $A$ a $B$ tak, aby $T_A \geq T_B + m$, bez ohledu na to, zda $m$ je kladné či záporné (nebo nula).

### Soutěžní úloha

Na vstupu dostanete seznam omezení na časy odjezdů jednotlivých vlaků. Nalezněte jízdní řád (přiřazení časů odjezdu k jednotlivým vlakům), který tato omezení splňuje, nebo rozhodněte, že žádný neexistuje. Čas odjezdu každého vlaku (v minutách od nějakého pevně zvoleného počátečního času) musí být celé číslo, jehož absolutní hodnota je nejvýše $10^9$.

Vaše řešení by mělo číst vstupní data ze standardního vstupu a výsledky vypisovat na standardní výstup; více detailů naleznete na *https://mo.mff.cuni.cz/p/otazky_a_odpovedi.html*.

### Formát vstupu

Na prvním řádku vstupu jsou dvě nezáporná celá čísla $t$ a $o$ ($1 \leq t \leq 3 \cdot 10^3$, $0 \leq o \leq 5 \cdot 10^4$) oddělená mezerou, udávající počet vlaků a počet omezení. Vlaky jsou číslovány od 1 do $t$. Každý z $o$ následujících řádků obsahuje tři mezerou oddělená celá čísla $A$, $B$ a $m$ ($1 \leq A, B \leq t$, $A \neq B$, $-10^5 \leq m \leq 10^5$) popisující jedno omezení: Čas odjezdu vlaku $A$ musí být alespoň o $m$ větší, než čas odjezdu vlaku $B$.

### Formát výstupu

Existuje-li přiřazení časů odjezdů vlakům, které splňuje všechna omezení, výstup tvoří jediný řádek obsahující $t$ celých čísel (v rozsahu od $-10^9$ do $10^9$) oddělených mezerami; $i$-té z nich udává čas odjezdu vlaku číslo $i$. Existuje-li více řešení, můžete vypsat libovolné z nich.

Jestliže žádné takové přiřazení neexistuje, na výstup vypište řetězec „`nelze`".

**Bodování**

Pro všechny vstupy platí $1 \leq t \leq 3 \cdot 10^3$ a $1 \leq o \leq 5 \cdot 10^4$. Nechť $A_i, B_i, m_i$ jsou hodnoty $i$-tého omezení ve vstupu. Je celkem šest sad testovacích vstupů:

| *sada* | *bodů* | *omezení* |
|---|---|---|
| 1 | 1 | $t = o \leq 100$, $A_i = i, B_i = i + 1$ pro všechna $1 \leq i < t$ a jinak $A_t = t, B_t = 1$ |
| 2 | 2 | pro všechna $i$ platí $m_i > 0$ |
| 3 | 1 | $t \leq 10, o \leq 20$ |
| 4 | 2 | $t \leq 100, o \leq 300$ |
| 5 | 2 | $t \leq 500, o \leq 1\,000$ |
| 6 | 2 | žádná další omezení |

**Příklady**

| *Vstup:* | *Výstup:* |
|---|---|
| `3 4`<br>`2 1 -4`<br>`1 2 -4`<br>`3 1 2`<br>`2 3 1` | `0 4 2` |

*První dvě omezení vynucují, že časy $T_1$ a $T_2$ odjezdů vlaků 1 a 2 se liší nanejvýš o 4. Další dvě nám říkají, že $T_1 < T_3 < T_2$ a $T_3$ musí být alespoň o 2 větší než $T_1$. Jiným správným řešením by tedy bylo například* `-1 2 1`.

| *Vstup:* | *Výstup:* |
|---|---|
| `3 3`<br>`2 1 0`<br>`3 2 0`<br>`1 3 1` | `nelze` |

*Uvedené požadavky říkají, že $T_3 \geq T_2 \geq T_1 > T_3$, což nelze splnit.*

## P-I-2 Správa skladu

Quark Ferengijec zakládá nový obchod se vším možným. Během dne za ním chodí prodejci a zájemci a dávají mu nabídky typu „prodám ti warp cívku za 5 ks zlatem raženého latinia“ či „koupil bych krabičku sardinek za 100 ks zlatem raženého latinia.“ Quark může kteroukoliv z těchto nabídek přijmout, či odmítnout – samozřejmě s tím, že může prodat jen něco, co aktuálně vlastní.

Jeho obchod je ale v úplných začátcích a má velmi omezenou kapacitu skladu: Vejde se mu do něj vždy pouze jedna položka! Pokud tedy už něco na skladě má a koupí něco jiného, starou položku musí vyhodit a už se k ní nemůže vrátit. To mu přirozeně obchodování výrazně komplikuje. Aby alespoň něco vydělal, používá ke správě skladu sofistikovaný AI kvantový software čtvrté generace, který mu po každé nabídce řekne, zda ji má přijmout, nebo odmítnout.

Quark si ale není jistý, jak moc dobře tento software funguje. Zajímalo by ho tedy, kolik by nejvíc mohl vydělat, kdyby měl předem přesné informace o nabídkách a poptávkách. Jelikož je velmi zaneprázdněný, nabídl vám krabičku sardinek za pomoc s řešením tohoto problému.

### Soutěžní úloha

Na vstupu dostanete seznam nabídek a poptávek tak, jak je po sobě Quark dostával. Určete, jakého největšího možného zisku mohl dosáhnout dle výše popsaných pravidel, jestliže začínal s prázdným skladem.

Vaše řešení by mělo číst vstupní data ze standardního vstupu a výsledky vypisovat na standardní výstup; více detailů naleznete na *https://mo.mff.cuni.cz/p/otazky_a_odpovedi.html*.

### Formát vstupu

Na první řádce vstupu je jedno přirozené číslo $n$ ($1 \leq n \leq 2 \cdot 10^5$) udávající celkový počet nabídek a poptávek. Každý z $n$ následujících řádků popisuje nabídku či poptávku ve formátu „$X$ $t$ $c$“, kde $X$ je buď znak `N` (nabídka), nebo znak `P` (poptávka), $t$ ($1 \leq t \leq n$) je přirozené číslo udávající kód prodávaného či kupovaného předmětu a $c$ ($1 \leq c \leq 10^3$) je přirozené číslo udávající cenu, za kterou je tento předmět nabízen či poptáván. Nabídky a poptávky jsou na vstupu v pořadí, v jakém je Quark dostával.

### Formát výstupu

Vypište jedno nezáporné celé číslo: největší možný zisk (rozdíl cen přijatých poptávek a nabídek), kterého Quark mohl dosáhnout.

**Bodování**

Pro všechny vstupy platí $1 \leq t \leq n \leq 2 \cdot 10^5$ a $1 \leq c \leq 10^3$.

Jsou celkem čtyři sady testovacích vstupů:

| *sada* | *bodů* | *omezení* |
|---|---|---|
| 1 | 2 | po první poptávce již nejsou žádné nabídky |
| 2 | 2 | $t = 1$ (všechny předměty jsou stejného druhu) |
| 3 | 3 | $n \leq 2\,000$ |
| 4 | 3 | žádná další omezení |

**Příklady**

| *Vstup:* | *Výstup:* |
|---|---|
| `6`<br>`N 1 15`<br>`N 1 10`<br>`N 2 1`<br>`P 1 30`<br>`P 2 20`<br>`P 1 20` | `20` |

*Quark může přijmout nabídku předmětu 1 za 10 a poté ho prodat za 30. Kvůli omezené kapacitě skladu přitom bohužel nemůže využít nabídku a poptávku předmětu 2, kterou by mohl vydělat 19. Tento vstup by se mohl objevit v první sadě.*

| *Vstup:* | *Výstup:* |
|---|---|
| `2`<br>`P 1 1000`<br>`N 1 1` | `0` |

*Před poptávkou po předmětu 1 Quark nemá možnost ho pořídit.*

| *Vstup:* | *Výstup:* |
|---|---|
| `4`<br>`N 3 70`<br>`P 3 80`<br>`N 3 10`<br>`P 3 20` | `20` |

*Quark dvakrát prodá předmět 3 a při každé z těchto transakcí vydělá 10.*

## P-I-3 Stavba

Tonda si založil stavební společnost Korporátní Stavební Podnik.™ Nakoupil pozemky na jedné ulici a udělal průzkum trhu. Nicméně zjistil, že splnit místní stavební předpisy bude složitější, než se na první pohled zdálo ...

Ulice má $N$ pozemků na přímce, kde $i$-tý pozemek sousedí s pozemky $i-1$ a $i+1$. (Pozemky 1 a $N$ mají jen jeden sousední pozemek.) Na každém pozemku lze postavit nejvýše jeden domek. Dále máme $N$ zákazníků, kde $i$-tý zákazník si chce koupit dům s výškou $h_i$. Nicméně je mu jedno, kde na ulici bude tento dům stát.

Kvůli protimrakodrapovým předpisům se musí výšky dvou domků na sousedních pozemcích lišit nejvýš o 1. Pozemek bez domku má výšku 0. Pozemky 1 a $N$ navíc sousedí se silnicí, která má výšku 0.

Tonda ještě potřebuje sehnat další povolení na úřadě, takže nechal plánování na vás. Kolik nejvíce domků dokážete prodat?

### Formát vstupu

Na prvním řádku vstupu dostanete číslo $N$ – počet pozemků i zákazníků. Na dalším řádku dostanete $N$ kladných celých čísel $h_1, \ldots, h_N$, kde $h_i$ udává výšku požadovanou $i$-tým zákazníkem.

### Formát výstupu

Vypište $N$ čísel $d_1, \ldots, d_N$ oddělených mezerami, kde $d_i$ značí výšku domku postaveného na $i$-tém pozemku (nebo 0, pokud dům nebudete stavět). Musíte naplánovat co nejvíce domků, které požadují zákazníci. Zároveň je nutné splnit protimrakodrapové regulace, tedy $d_1, d_N \leq 1$ a $|d_i - d_{i-1}| \leq 1$ pro každé $i$.

### Bodování

Řešení s časovou složitostí $\mathcal{O}(N)$, $\mathcal{O}(N \log N)$ či obdobnou (tj. efektivní pro $N \leq 10^6$) získají 9–10 bodů v závislosti na přesné časové složitosti. Řešení s časovou složitostí $\mathcal{O}(N^2)$ (tj. efektivní pro $N \leq 5\,000$) získají 6 bodů. Libovolné správné řešení bez ohledu na jeho časovou složitost může získat až 3 body. Připomínáme, že nezbytnou součástí řešení je zdůvodnění jeho správnosti.

### Příklad

| *Vstup:* | *Výstup:* |
|---|---|
| `6`<br>`1 7 3 1 3 1` | `1 2 3 3 2 1` |

*Prodáme domky na pozicích 1, 3, 4 a 6. Lze dokázat, že více jak 4 domky prodat nejde.*

| *Vstup:* | *Výstup:* |
|---|---|
| `6`<br>`1 1 1 4 5 6` | `1 2 1 2 1 1` |

*Zájemcům můžeme prodat například domky na pozicích 3, 5 a 6.*

## P-I-4 Počítáme s tříděním

*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. Ze stejného studijního textu budou vycházet i úlohy v dalších kolech soutěže.*

Toto je praktická úloha. Máte za úkol napsat a odevzdat tři programy pro Ondrův procesor, které budou automaticky otestovány na sadě vstupních dat. Tyto programy můžete napsat ručně, nebo si můžete napsat program, který je vygeneruje.

V tomto kole budete psát programy pro `OP2048`, Ondrův nízkorozpočtový model s $N = 2048$ jádry. Váš program smí obsahovat nejvýše 256 instrukcí, jinak nebude uznán.

Jednotlivé podúlohy jsou nezávislé, můžete vyřešit podúlohu, i když nevyřešíte předchozí. Řešení podúloh můžete ve webovém rozhraní odevzdat postupně, nebo všechny najednou. Pokud vaše řešení nezíská plný počet bodů, můžete ho opravit a odevzdat znovu. Pokud již máte nějakou podúlohu vyřešenou, tak její řešení nemusíte odevzdávat znovu, když odevzdáváte jinou podúlohu.

### Úlohy

a) Minimum (*3 body*) (`minimum.txt`)

Napište program, který spočítá minimum z řady čísel.

Každé jádro dostane jako vstup jedno číslo. Výstupem jádra s identifikačním číslem 0 by mělo být nejmenší z čísel na vstupu.

b) Nejčetnější číslo (*3 body*) (`nejcetnejsi.txt`)

Napište program, který najde nejčetnější číslo v seřazené posloupnosti.

Každé jádro dostane jako vstup jedno číslo. Posloupnost těchto čísel bude neklesající, tedy napravo od každého jádra (kromě posledního) se bude nacházet jádro se stejným nebo větším vstupem. Výstupem jádra s identifikačním číslem 0 by mělo být to číslo, které se na vstupu vyskytuje nejvícekrát. Zaručujeme, že se jedno z čísel bude vyskytovat častěji, než jakékoliv jiné.

c) Součet (*4 body*) (`soucet.txt`)

Napište program, který spočítá součet řady čísel.

Každé jádro dostane jako vstup jedno číslo. Výstupem jádra s identifikačním číslem 0 by měl být součet všech čísel na vstupu. Všechna čísla na vstupu budou mezi $-10^{12}$ a $10^{12}$ včetně, jejich součet se tedy do 64-bitového čísla s přehledem vejde.

### Simulátor

Abyste mohli programy pro Ondrův procesor vyzkoušet, můžete použít simulátor. Simulátor spustí daný program na daném vstupu a vypíše vám jeho výstup. Také umí vygenerovat tabulku s výsledky všech instrukcí.

Simulátor existuje ve dvou verzích. První z nich má podobu webového rozhraní, dostupného na adrese *https://mo.mff.cuni.cz/sort-machine/*. To vám umožní

spustit zadaný program na našem serveru. Druhá z nich má podobu aplikace pro příkazový řádek. Z adresy *https://mo.mff.cuni.cz/sort-machine/download* si můžete stáhnout buď zkompilovanou binárku pro x86-64 Windows nebo Linux, nebo zdrojový kód v jazyce Rust, který můžete přeložit pro (téměř) libovolný operační systém a architekturu pomocí nástroje Cargo.

V každém případě musíte zadat program a vstup. Formát programu je popsaný níže. Vstupní soubor musí obsahovat tolik řádků, jako má každé z jader vstupů. Na prvním řádku by měl být první vstup všech jader od prvního po poslední, na druhém druhý vstup všech jader, a tak dále. Hodnoty v rámci řádku mohou být odděleny jednou nebo více mezerami či tabulátory. Pokud chcete simulovat program, který nemá žádný vstup, musíte specifikovat počet jader (a vstup můžete nechat prázdný). V opačném případě počet jader uvádět nemusíte.

Simulátor nekontroluje limity ze zadání, jader tedy nemusí být nutně 2048 a program může být delší než 256 instrukcí. Webová verze simulátoru omezuje délku programu a vstupu. Verze ke stažení žádná omezení nevymáhá, ale upozorňujeme, že ke spuštění programu s $M$ instrukcemi na $N$ jádrech budete potřebovat přibližně $8 \cdot N \cdot M$ bajtů paměti.

**Studijní text**

Ondra si před deseti lety během zkoušky z algoritmizace vylosoval třídicí algoritmy, které se nenaučil, a ze zkoušky dostal čtyřku. Tehdy přísahal, že se třídicím algoritmům pomstí. Pomstít se algoritmu ale není vůbec jednoduché, protože algoritmus není možné zničit ani uvěznit. Algoritmus může porazit jedině tím, že ho překoná.

Proto Ondra posledních deset let strávil vývojem úplně nového modelu procesoru, který umí třídit v konstantním čase. Je si jistý, že za pár let bude počítač s Ondrovým procesorem v každé domácnosti. Na třídicí algoritmy pak všichni do pár dalších let zajisté zapomenou!

Stroje na masovou výrobu integrovaných obvodů jsou ale velmi drahé, takže Ondra potřebuje pár ukázkových programů, aby získal přízeň investorů. Protože Ondra nikdy algoritmizaci nedostudoval, chce, abyste mu s tím pomohli.

**Ondrův procesor**

Ondrův procesor má $N$ jader, každé má jiné identifikační číslo od 0 do $N-1$. Jádra jsou uspořádána v řadě. Na začátku jsou seřazená podle identifikačních čísel s jádrem 0 vlevo, jejich pořadí se ale v průběhu výpočtu může měnit.

Programy pro něj jsou tvořeny posloupností instrukcí, které všechna jádra vykonávají souběžně, tedy nejprve všechna jádra současně vykonají první instrukci, poté všechna jádra vykonají druhou instrukci, a tak dále. Existuje mnoho různých instrukcí, jejich seznam najdete níže. Instrukce číslujeme od jedné.

Každá instrukce spočítá hodnotu, které budeme říkat výsledek. Každé jádro si pamatuje výsledky všech instrukcí, které za běh programu vykonalo. Většina instrukcí pracuje s výsledky předchozích instrukcí, například instrukce `+ 1 2` sečte výsledek první a druhé instrukce.

Jádra mohou číst paměť svých sousedů pomocí speciálních instrukcí `left` a `right`. Instrukce `left` $i$ přečte výsledek $i$-té instrukce vykonané jádrem nalevo od jádra, co ji vykoná, obdobně `right` $i$ přečte výsledek $i$-té instrukce jádra napravo. Čtení probíhá cyklicky, tedy první jádro je napravo od posledního.

Nakonec tu máme nejdůležitější instrukci, a to `sort`, která změní pořadí jader. Každé jádro si nejprve zvolí číslo zvané klíč (argument instrukce `sort`). Poté procesor pomocí komplikovaných obvodů jádra seřadí. Jádro s nejmenším klíčem skončí nalevo a jádro s největším klíčem skončí napravo. Třídění je stabilní, tj. jádra se stejným klíčem si zachovají vzájemné pořadí. Jádra si při přesuvání zachovávají svoje identifikační čísla i historii výsledků instrukcí, jádro s identifikačním číslem 0 tedy po přesunutí už nemusí být nalevo.

**Aritmetika**

Ondrův procesor používá 64-bitová znaménková čísla ve dvojkovém doplňku, muže tedy reprezentovat čísla v rozsahu $-2^{63}$ až $2^{63}-1$. Pokud se výsledek nějaké operace nevejde do 64 bitů, pak budou přebytečné nejvýznamnější bity zahozeny.

**Vstup a výstup**

Každé jádro má několik vstupů (počet záleží na problému) očíslovaných od 1. Vstup číslo $k$ může jádro přečíst pomocí instrukce `input` $k$. Za výstup jádra považujeme výsledek poslední instrukce, kterou vykoná.

**Instrukční sada**

K dispozici jsou následující instrukce. V popisu instrukcí označíme výsledek $i$-té instrukce pro specifikované jádro $r_i$.

| | |
|---|---|
| `const` $x$ | Vrátí číslo $x$. |
| `id` | Vrátí identifikační číslo jádra. |
| `input` $k$ | Vrátí $k$-tou vstupní hodnotu tohoto jádra. |
| `copy` $i$ | Vrátí $r_i$. |
| `sort` $i$ | Seřadí jádra podle klíče $r_i$, vrátí $r_i$. |
| `left` $i$ | Vrátí $r_i$ jádra nalevo. |
| `right` $i$ | Vrátí $r_i$ jádra napravo. |
| `+` $a$ $b$ | Vrátí $r_a + r_b$. |
| `-` $a$ $b$ | Vrátí $r_a - r_b$. |
| `*` $a$ $b$ | Vrátí $r_a \cdot r_b$. |
| `/` $a$ $b$ | Vrátí dolní celou část $r_a \div r_b$, pokud $r_b \neq 0$, jinak vrátí 0. |
| `%` $a$ $b$ | Vrátí zbytek po dělení $r_a \div r_b$, pokud $r_b \neq 0$, jinak vrátí 0. |
| `\|` $a$ $b$ | Vrátí bitovou disjunkci (OR) $r_a$ a $r_b$. |
| `&` $a$ $b$ | Vrátí bitovou konjunkci (AND) $r_a$ a $r_b$. |
| `^` $a$ $b$ | Vrátí bitovou exkluzivní disjunkci (XOR) $r_a$ a $r_b$. |
| `=` $a$ $b$ | Vrátí 1, pokud $r_a = r_b$, jinak vrátí 0. |
| `!=` $a$ $b$ | Vrátí 1, pokud $r_a \neq r_b$, jinak vrátí 0. |
| `<` $a$ $b$ | Vrátí 1, pokud $r_a < r_b$, jinak vrátí 0. |
| `>` $a$ $b$ | Vrátí 1, pokud $r_a > r_b$, jinak vrátí 0. |
| `<=` $a$ $b$ | Vrátí 1, pokud $r_a \leq r_b$, jinak vrátí 0. |
| `>=` $a$ $b$ | Vrátí 1, pokud $r_a \geq r_b$, jinak vrátí 0. |
| `if` $c$ $a$ $b$ | Pokud $r_c \neq 0$, pak vrátí $r_a$, jinak vrátí $r_b$. |

**Formát programu**

Programy pro Ondrův procesor se zapisují do textového souboru. Každá instrukce je zapsaná na svém vlastním řádku. Jméno instrukce a její argumenty jsou oddělené alespoň jednou mezerou či tabulátorem. V kódu mohou být prázdné řádky, které nic nedělají a nepočítají se mezi instrukce. Program může obsahovat i komentáře, které začínají znakem „`#`“ a končí na konci řádku. Například program pro výpočet druhé mocniny čísla procesoru můžeme zapsat takto:

```
id    # Získáme identifikační číslo procesoru
* 1 1 # Vynásobíme ho sebou samým
```

Instrukce je možné pojmenovat; stačí na začátek řádku připsat jméno následované znakem „`:`“. Jméno může obsahovat písmena, číslice a podtržítka, nesmí se skládat jen z číslic. Jméno instrukce je pak možné použít místo jejího čísla:

```
procesor: id
mocnina:  * procesor procesor
```

Stejné jméno můžete použít vícekrát, v tom případě jméno vždy označuje poslední přecházející takto pojmenovanou instrukci. Instrukce také může mít více jmen. Například program pro sečtení tří vstupů můžete zapsat takto:

```
součet: const 0
vstup:  input 1
součet: + součet vstup
vstup:  input 2
součet: + součet vstup
vstup:  input 3
součet: + součet vstup
```

**Příklad #1: hledání minima**

Každé jádro dostane jako vstup jedno číslo, každé jádro jiné. Jádro s nejmenším číslem by mělo mít výstup 1, ostatní jádra výstup 0.

*Řešení:*

```
vstup:        input 1
              sort vstup
vstup_nalevo: left vstup
              > vstup_nalevo vstup
```

Jádra seřadíme podle čísla na vstupu, čímž dostaneme jádro s nejmenším vstupem na začátek řady. Poté každé jádro zjistí, jestli má jádro nalevo od něj větší vstup než ono samo. Pokud tomu tak je, pak se musí nacházet na začátku řady, a tedy mít nejmenší vstup.

**Příklad #2: hledání duplikátů**

Každé jádro dostane jako vstup dvě čísla. Jádro musí odpovědět 1, pokud existuje jiné jádro se stejným vstupem, jinak musí odpovědět 0.

*Řešení:*

```
vstup_1:  input 1
vstup_2:  input 2
          sort vstup_2
          sort vstup_1

soused_1: left vstup_1
soused_2: left vstup_2
stejné_1: = soused_1 vstup_1
stejné_2: = soused_2 vstup_2
vlevo:    & stejné_1 stejné_2

soused_1: right vstup_1
soused_2: right vstup_2
stejné_1: = soused_1 vstup_1
stejné_2: = soused_2 vstup_2
vpravo:   & stejné_1 stejné_2

          | vlevo vpravo
```

Jádra seřadíme nejprve podle druhého vstupu, poté podle prvního vstupu. Díky tomu, že třídění zachovává pořadí jader se stejným klíčem, budou nakonec jádra seřazená primárně podle prvního vstupu a sekundárně podle druhého vstupu (a terciárně podle identifikačního čísla). Jádra se stejnými vstupy tedy skončí u sebe. Poté stačí, aby se každé jádro podívalo na levého a pravého souseda, a zjistilo, jestli některý má stejný vstup.