74. ročník Matematické olympiády – 2024/2025

Úlohy krajského kola kategorie P

Krajské kolo 74. ročníku MO kategorie P se koná v úterý 21. 1. 2025 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). Zakázány jsou rovněž jakékoliv další pomůcky kromě psacích potřeb (např. knihy, výpisy programů, kalkulačky, mobilní telefony). Řešení každé úlohy vypracujte na samostatný list papíru.

Řešení každé úlohy má 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). Není možné odkazovat se na vaše řešení úloh domácího kola.

Do řešení nemusíte psát odpovídající program, algoritmus stačí zapsat ve vhodném pseudokódu nebo dokonce jenom slovně, je-li popis dostatečně podrobný a srozumitelný. 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/p/*. Na stejném místě bude zveřejněn i seznam úspěšných řešitelů krajského kola a seznam řešitelů postupujících do ústředního kola. Svá opravená řešení s komentáři opravovatelů najdete v OSMO.

P-II-1 Největší čtverec

Adam se rozhodl namalovat čtvercové umělecké dílo na část chodníku skládající se z $n \times m$ dlaždic. Jak už to ale bývá, některé z těchto dlaždic jsou rozbité.

Adam samozřejmě chce mít co největší umělecké dílo, ale kdyby obsahovalo moc rozbitých dlaždic, velmi by se za sebe styděl. Nastavil si tedy jako limit k rozbitých dlaždic, při kterých ještě tuto ostudu ustojí. Poradíte Adamovi, kterou čtvercovou sekci chodníku má pomalovat, aniž by tato část obsahovala více než k dlaždic a navždy ho zostudila?

Soutěžní úloha

Je zadána mřížka $n \times m$ dlaždic a Adamova tolerance na rozbité dlaždice k. Vašim úkolem je najít co největší čtvercovou část chodníku, která obsahuje nanejvýš k rozbitých dlaždic (je zaručeno, že alespoň jedna taková část existuje). Je-li jich více, stačí odpovědět libovolnou z nich.

Formát vstupu

Na prvním řádku dostanete celá čísla n, m a k ($1 \leq n,\ m \leq 5000$, $0 \leq k \leq nm$), popisující rozměry chodníku a Adamovu toleranci na rozbité dlaždice.

Následuje n řádků, z nichž každý obsahuje řetězec délky m. Každý znak v těchto řetězcích je buď `+` reprezentující nepoškozenou dlaždici, nebo `@` reprezentující rozbitou dlaždici.

Formát výstupu

Na jediném řádku výstupu vypište tři celá čísla r, s, a popisující vybraný čtverec. Čísla r a s ($1 \leq r \leq n$, $1 \leq s \leq m$) určují řádek a sloupec levého horního rohu vybraného čtverce, indexováno od jedné. Číslo a ($1 \leq a \leq n$) určuje délku strany tohoto čtverce v dlaždicích. Souřadnice pravého dolního rohu takového čtverce jsou tedy $r + a - 1$ a $s + a - 1$.

Bodování

Řešení s časovou složitostí $\mathcal{O}(nm)$, $\mathcal{O}(nm \log(n + m))$ či obdobnou, tedy efektivní pro $n, m \leq 5000$, získá 9 až 10 bodů v závislosti na přesné časové složitosti.

Řešení s časovou složitostí $\mathcal{O}(nm \min(n, m))$ či obdobnou, tedy efektivní pro $n, m \leq 500$, dostane alespoň 6 bodů.

Libovolné správné řešení dostane alespoň 3 body.

Příklady

Vstup:	*Výstup:*

`4 5 0`
`@+@+@`
`+++@+`
`+++++`
`@+@+@`
	`2 1 2`

Čtverec s levým horním rohem v druhém řádku a prvním sloupci s délkou strany 2 neobsahuje žádné rozbité dlaždice. Žádný větší takový čtverec nalézt nelze. Dalším správným řešením by bylo „`2 2 2`“.

Vstup:	*Výstup:*

`5 5 2`
`@+++@`
`+@+@+`
`++@++`
`+@+++`
`@+++@`
	`3 3 3`

P-II-2 Vynášení odpadků

Organizátoři jedné nejmenované programovací soutěže se rozhodli oslavit úspěšné vymyšlení úloh do dalšího kola soutěže. Oslava se samozřejmě protáhla dlouho do noci. Když se kolem poledne organizátoři probudili, nikdo už si nepamatoval, jak zněla zadání vymyšlených úloh, natožpak jejich řešení. Rozhodně ale po celonoční oslavě zbyl velký nepořádek.

Organizátoři musí co nejrychleji vymyslet nové úlohy, ale také je zapotřebí vyčistit zahradu od všech odpadků, které se na ní válí. Na tento úklid byl vytažen robot s krycím názvem Krajně Suprový Pomocník.

Jelikož se ale jedná jen o prototyp, robot v jeden okamžik unese pouze jeden kus odpadu. Odpadků jsou naštěstí jen dva druhy (papír a plast), ale pro každý druh je k dispozici jen jedna popelnice. Každý kus odpadu je nutné zatřídit do popelnice, do které patří. Poraďte Krajně Suprovému Pomocníkovi, jak se má do úklidu pustit, aby najezdil co nejmenší vzdálenost.

Soutěžní úloha

V zahradě se nachází m kusů papírového odpadu a n kusů plastového odpadu. Každý odpadek je zadán svými souřadnicemi v zahradě: dvojice celých čísel (x_i, y_i).

Dále se v zahradě nachází popelnice na papír se souřadnicemi $(x_{\text{papír}}, y_{\text{papír}})$ a popelnice na plast se souřadnicemi $(x_{\text{plast}}, y_{\text{plast}})$. Robot Krajně Suprový Pomocník je na začátku přesně u popelnice na papír, tedy na souřadnicích $(x_{\text{papír}}, y_{\text{papír}})$.

Vaším úkolem je spočítat délku nejkratší možné cesty, během které zvládne robot roztřídit všechen odpad na zahradě do správných popelnic. Nachází-li se robot na stejných souřadnicích jako nějaký kus odpadu a zatím žádný odpadek nenese, může ho nabrat. Naopak, nachází-li se na stejných souřadnicích jako popelnice odpovídající právě převáženému druhu odpadu, může ho vyhodit.

Formát vstupu

Na prvním řádku dostanete šest celých čísel: m, n, $x_{\text{papír}}$, $y_{\text{papír}}$, x_{plast} a y_{plast}. Pak následuje $m + n$ řádků udávající polohy odpadků, a to vždy jako dvojici celých čísel x_i a y_i. Prvních m dvojic udává souřadnice papírových odpadků, pak následuje n dvojic souřadnic plastových odpadků.

Formát výstupu

Vypište délku nejkratší možné trasy, která je potřeba na úklid zahrady, jako desetinné číslo. Vaše odpověď se může lišit maximálně o 10^{-6} od správné odpovědi.

Bodování

Nechť $N = n + m$.

- Řešení s časovými složitostmi $\mathcal{O}(N)$, $\mathcal{O}(N \log N)$ či obdobnými, tedy efektivní pro $N \leq 1\,000\,000$, získají 9 až 10 bodů v závislosti na přesné časové složitosti.
- Až 6 bodů mohou získat řešení s časovou složitostí $\mathcal{O}(N^2)$, tedy efektivní pro $N \leq 50\,000$.

- Až 3 body mohou získat efektivní řešení, která předpokládají $n \leq 1$.
- Až 2 body může získat libovolné správné řešení.

Příklady

Vstup:	*Výstup:*

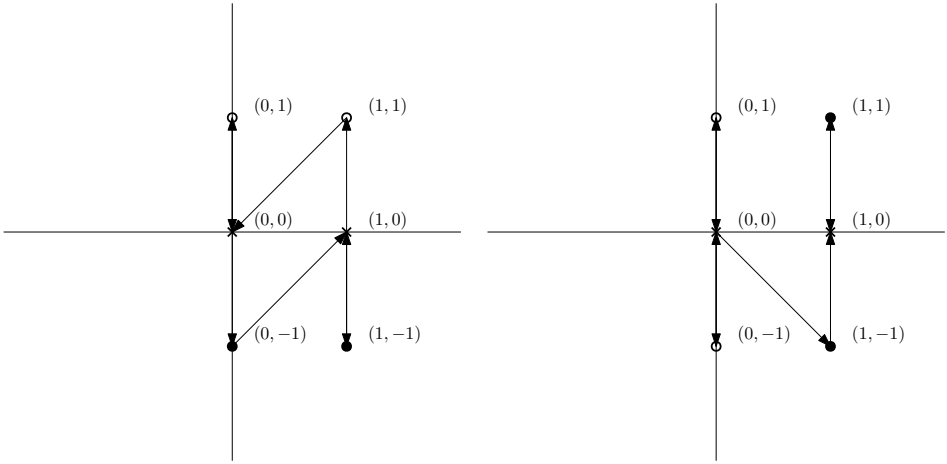
`2 2 0 0 1 0`
`0 1`
`1 1`
`0 -1`
`1 -1`
	`8.828427`

Nejprve vyhodíme papírový odpad na $(0, 1)$*, pak plastový odpad na* $(0, -1)$*, pak plastový odpad na* $(1, -1)$ *a nakonec papírový odpad na* $(1, 1)$*. Toto bude dohromady vzdálenost* $2 + (1 + \sqrt{2}) + 2 + (1 + \sqrt{2}) = 6 + 2\sqrt{2} \approx 8.828427$*.*

Vstup:	*Výstup:*

`2 2 0 0 1 0`
`0 1`
`0 -1`
`1 1`
`1 -1`
	`8.414213`

Nejprve vyhodíme papírový odpad na $(0, 1)$*, pak papírový odpad na* $(0, -1)$*, pak plastový odpad na* $(1, -1)$ *a nakonec plastový odpad na* $(1, 1)$*. Toto bude dohromady vzdálenost* $2 + 2 + (\sqrt{2} + 1) + 2 = 7 + \sqrt{2} \approx 8.414213$*.*



P-II-3 Venčení

Od té doby, co se domácí mazlíčci naučili používat mobilní telefony, je život pejskařů nesmírně obtížný. Chlupáčci si postahovali mapy okolí a když zjistili, jak obrovský je venkovní svět, začali se dožadovat delších a delších procházek ... Nakonec se jim podařilo vyvinout aplikaci, která jim nejdelší trasu počítá automaticky.

Nejhůře jsou na tom nyní občané České Lípy, která obsahuje opravdu dlouhé cesty. Každodenní venčení se tu někdy protáhne třeba i na 25 hodin! Městská rada se proto rozhodla vydat mimořádné opatření: na jednom z náměstí bude zbourána 5G věž. Tomuto místu se bude jistě každý pes vyhýbat, jelikož by jinak po cestě nemohl v rozšířené realitě lovit virtuální veverky. Poradíte radě, kterou věž má obětovat, aby co nejvíce zkrátila nejdelší možné venčení?

Soutěžní úloha

Česká Lípa sestává z n náměstí očíslovaných 1 až n. Mezi jejich dvojicemi vede celkem $n - 1$ stejně dlouhých, obousměrných ulic.

Cestou rozumíme nějakou posloupnost různých náměstí takovou, že mezi každou po sobě jdoucí dvojicí náměstí vede ulice. S jistotou víme, že z každého náměstí lze právě jednou cestou dojít do libovolného jiného. (V teorii grafů bychom řekli, že Česká Lípa je strom).

Rada si přeje zbourat 5G věž na jednom z náměstí. Možným venčením rozumíme jakoukoliv cestu, která neobsahuje toto zvolené náměstí. Vaším úkolem je vybrat takové náměstí, aby bylo nejdelší možné venčení co nejkratší.

Formát vstupu

Na prvním řádku dostanete číslo n ($2 \leq n \leq 1\,000\,000$), udávající počet náměstí v České Lípě. Na dalších $n - 1$ řádcích jsou vždy dvě čísla náměstí, mezi kterými vede ulice.

Formát výstupu

Vypište číslo takového náměstí, na kterém zbourání 5G věže co nejvíce zkrátí nejdelší možné venčení. Pokud takových náměstí existuje více, vypište číslo libovolného z nich.

Bodování

- Až 10 bodů může získat řešení s časovou složitostí $\mathcal{O}(n)$, tedy efektivní pro $n \leq 1\,000\,000$. Až 9 bodů dostane také řešení s časovou složitostí $\mathcal{O}(n \log n)$ či obdobnou.
- Až 7 bodů obdrží řešení efektivní pro $n \leq 70\,000$ na vstupech s dodatečným předpokladem, že vždy existuje takové náměstí, po jehož zvolení by bylo nejdelší možné venčení kratší než $c = 100$. Chcete-li tento předpoklad využít ve svém řešení, důrazně to do něj napište. Toto řešení by mělo mít časovou složitost $\mathcal{O}(nc)$.
- Až 5 bodů obdrží řešení s časovou složitostí $\mathcal{O}(n^2)$, tedy efektivní pro $n \leq 5\,000$.
- Až 2 body může získat libovolné správné řešení.

Příklady

Vstup:	*Výstup:*
5 1 2 2 3 3 4 3 5	3

Zboříme-li věž na třetím náměstí, nejdelší možné venčení bude sestávat pouze ze dvou náměstí (1, 2). V ostatních případech půjde vždy o právě tři náměstí.

Vstup:	*Výstup:*
7 1 2 1 3 2 4 2 5 3 6 3 7	2

Pro tento vstup lze zbořit věž na náměstí 1, 2, nebo 3.

P-II-4 Počítáme s tříděním

K této úloze se vztahuje studijní text uvedený na následujících stranách, který je stejný jako v domácím kole.

Ondrova řada procesorů se tentokrát vrací v teoretické úloze. Nebudete tedy psát program pro konkrétní model procesoru s pevným počtem jader, ale pro obecný procesor s N jádry. Nemusí být možné napsat jeden program, který by fungoval pro libovolné N, takže máte za úkol popsat, jak byste program pro konkrétní N napsali. Můžete si představit, že píšete program pro běžný počítač, který píše program pro Ondrův procesor. Můžete předpokládat, že N je mocnina dvojky.

Vaše řešení budeme hodnotit podle asymptotické složitosti délky programu. Ve většině podúloh byste měli cílit na *polylogaritmickou* složitost vzhledem k N, tedy složitost $\mathcal{O}((\log N)^c)$ pro libovolnou konstantu c.

Obdobně jako v běžných úlohách můžete předpokládat, že Ondrův procesor dokáže pracovat s čísly polynomiální velikosti vzhledem k N a číslům na vstupu. Například součet všech čísel na vstupu tedy výsledkem instrukce být může, jejich součin ale určitě ne.

Ve vašem popisu řešení můžete místo zápisu programu použít vhodný pseudokód, zvlášť pro jednoduché logické nebo aritmetické operace. Musí ale být jasné, jak probíhá samotné třídění a komunikace mezi jádry.

Jednotlivé podúlohy budou hodnoceny nezávisle, můžete je řešit v libovolném pořadí. Ve svém řešení se můžete odkazovat na autorské řešení úloh z domácího kola.

Úlohy

a) Průměrnost (*2 body*)

Každé jádro dostane jako vstup jedno číslo. Z těchto čísel bychom mohli spočítat průměr. Výstupem jádra má být:

- -1, pokud je jeho vstup menší než průměr,
- 0, pokud je jeho vstup rovný průměru, nebo
- 1, pokud jeho vstup větší než průměr.

Plný počet bodů dostane řešení v $\mathcal{O}(\log N)$, libovolné polylogaritmické řešení dostane 1 bod.

Ukázkový vstup:
```
2 6 8 7 3 5 1 8
```
Ukázkový výstup:
```
-1 1 1 1 -1 0 -1 1
```

b) Prefixový součet (*3 body*)

Každé jádro dostane jako vstup jedno číslo. Výstupem jádra by měl být součet jeho vstupu a vstupů všech jader s nižším identifikačním číslem.

Plný počet bodů dostane řešení v $\mathcal{O}(\log N)$, libovolné polylogaritmické řešení dostane 2 body.

Ukázkový vstup:
```
2 1 4 0 4 3 2 5
```
Ukázkový výstup:
```
2 3 7 7 11 14 16 21
```

c) Pořadí (*5 body*)

Každé jádro dostane jako vstup jedno číslo od 0 do K, každé jádro jiné. Výstupem jádra s nejmenším vstupem by mělo být 0, výstupem jádra s druhým nejmenším vstupem 1, a tak dále. Výstupem jádra by tedy měla být jeho pozice, kdybychom jádra seřadili podle jejich vstupu.

Hodnotu K budete při psaní programu znát, může na ní tedy záviset vaše složitost délky programu. Plný počet bodů dostane řešení v $\mathcal{O}(\log K)$, libovolné polylogaritmické (ať už v N nebo K) řešení dostane 3 body a řešení v $\mathcal{O}(N)$ dostane 1 bod.

Ukázkový vstup:
```
15 10 7 14 12 4 0 9
```
Ukázkový výstup:
```
7 4 2 6 5 1 0 3
```

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.