# Informácie a pravidlá

### Pre koho je súťaž určená?

- Do **kategórie B** sa smú zapojiť len tí žiaci základných a stredných škôl, ktorí ešte ani v tomto, ani v nasledujúcom školskom roku nebudú končiť strednú školu.
- Do **kategórie A** sa môžu zapojiť všetci žiaci (základných aj) stredných škôl.

### Odovzdávanie riešení domáceho kola

Riešitelia domáceho kola odovzdávajú riešenia sami, v elektronickej podobe, a to priamo na stránke olympiády: http://oi.sk/. Odovzdávanie riešení bude spustené najneskôr do konca septembra.

### Termíny 40. ročníka OI

- Riešenia domáceho kola je potrebné v kategórii A odovzdať najneskôr **15. 11. 2024**, v kategórii B najneskôr **30. 11. 2024**.
- Krajské kolo oboch kategórií prebehne 21. 1. 2025 zvlášť v každom kraji.
- Celoštátne kolo kategórie A sa bude konať v dňoch 19.-22. 3. 2025 (súťaží sa vo štvrtok a piatok).
- Detailnejšie informácie o priebehu súťaže nájdete na webe OI: https://oi.sk/?s=pravidla

### Ako majú vyzerať riešenia úloh?

V **praktických úlohách** je vašou úlohou vytvoriť program, ktorý bude riešiť zadanú úlohu. Program musí byť v prvom rade korektný a funkčný, v druhom rade sa snažte aby bol čo najefektívnejší.

V kategórii B môžete použiť ľubovoľný programovací jazyk.

V kategórii A musíte riešenia praktických úloh písať v jednom z podporovaných jazykov (napr. C++, Python alebo Java). Presný zoznam podporovaných jazykov vrátane konkrétnych verzií ich kompilátorov a interpreterov nájdete v systéme na odovzdávanie riešení. Taktiež tam nájdete presnejší popis toho, ako sa majú vaše programy správať, napr. detaily realizácie vstupu a výstupu.

Odovzdaný program bude automaticky otestovaný na viacerých vopred pripravených testovacích vstupoch. Podľa toho, na koľko z nich dá správnu odpoveď, vám budú pridelené body. Výsledok testovania sa dozviete krátko po odovzdaní. Ak váš program nezíska plný počet bodov, budete ho môcť vylepšiť a odovzdať znova, až do uplynutia termínu na odovzdávanie.

Ak nie je v zadaní povedané ináč, riešenia **teoretických úloh** musia obsahovať:

- podstatné časti algoritmu ako program (v ľubovoľnom bežnom prog. jazyku) alebo ekvivalentný pseudokód
- podrobný slovný popis použitého algoritmu
- zdôvodnenie jeho správnosti
- diskusiu o efektivite zvoleného riešenia (odhad časovej a pamäťovej zložitosti)

Ak používate v programe netriviálne algoritmy alebo dátové štruktúry (napr. rôzne súčasti STL v C++), súčasťou slovného popisu algoritmu by mal byť stručný popis ich implementácie.

### Usporiadateľ súťaže

Olympiádu v informatike (OI) vyhlasuje *Ministerstvo školstva SR* v spolupráci so *Slovenskou informatickou spoločnosťou* (odborným garantom súťaže) a *Slovenskou komisiou Olympiády v informatike*. Súťaž organizuje *Slovenská komisia OI* a v jednotlivých krajoch ju riadia *krajské komisie OI*. Na jednotlivých školách ju zaisťujú učitelia informatiky. Celoštátne kolo OI, tlač materiálov a ich distribúciu po organizačnej stránke zabezpečuje NIVAM v tesnej súčinnosti so Slovenskou komisiou OI.

## A-I-1 Dejepis

*Toto je **praktická úloha**. Pomocou webového rozhrania odovzdaj **funkčný, odladený program**.*

Miška sa snaží umiestniť $t$ historických udalostí (očíslovaných od 1 po $t$) na časovú os – presnejšie, chce každej udalosti $j$ priradiť konkrétny rok $r_j$, kedy nastala.

Miška si síce nepamätá vôbec žiadne roky, ale zo súvislostí si vie odvodiť rôzne vzťahy medzi udalosťami. Presnejšie, prišla na $n$ rôznych vzťahov. Každý z týchto vzťahov má tvar nerovnosti: „udalosť $a_i$ musí byť aspoň $d_i$ rokov po udalosti $b_i$“.

V niektorých nerovnostiach môže byť $d_i$ aj záporné. Napr. ak $d_i = -10$, tak si je Miška istá, že udalosť $a_i$ mohla byť 10 rokov pred $b_i$ alebo neskôr, ale určite nie skôr. (Teda ak by pre toto $d_i$ bola udalosť $b_i$ v roku 1980, udalosť $a_i$ by mohla byť v ľubovoľnom roku väčšom alebo rovnom 1970.)

Napíš program, ktorý určí roky $r_j$ tak, aby všetky Miškine nerovnosti platili – alebo zistí, že sa to nedá.

### Formát vstupu a výstupu

V prvom riadku vstupu sú celé čísla $t$ a $n$. Zvyšok vstupu tvorí $n$ riadkov. V $i$-tom z nich sú tri celé čísla popisujúce jeden vzťah: $a_i$, $b_i$ a $d_i$. V každom vzťahu platí $1 \leq a_i, b_i \leq t$ a $a_i \neq b_i$.

Ak riešenie neexistuje, vypíš jeden riadok a v ňom reťazec „`neexistuje`“.

Inak vypíš ľubovoľné jedno riešenie: riadok obsahujúci $t$ medzerou oddelených celých čísel $r_1, \ldots, r_t$. Inými slovami, postupne pre každú udalosť $j$ vypíš rok $r_j$, kedy sa mohla stať. Všetky vypísané hodnoty $r_j$ musia byť z rozsahu od $-10^9$ po $10^9$ vrátane a pre každé $i$ musí platiť $r_{a_i} \geq r_{b_i} + d_i$.

### Obmedzenia a hodnotenie

Vo všetkých vstupoch platí $1 \leq t \leq 1000$, $1 \leq n \leq 10\,000$ a v každom vzťahu platí $|d_i| \leq 100\,000$.

Testovacie vstupy sú rozdelené do šiestich sád. Pre každú sadu platia iné dodatočné obmedzenia. Body za každú sadu dostaneš, ak tvoj program správne vyrieši všetky vstupy, ktoré do nej patria.

| sada | body | dodatočné obmedzenia |
|---|---|---|
| 1 | 1 | $t \leq 100$, $n = t$, pre každé $i$ $(1 \leq i \leq n-1)$ vo vzťahu $i$ platí $a_i = i$ a $b_i = i+1$ a v poslednom vzťahu platí $a_t = t$ a $b_t = 1$ (vzťahy teda tvoria jeden orientovaný cyklus) |
| 2 | 2 | pre každé $i$ $(1 \leq i \leq n)$ vo vzťahu $i$ platí $d_i > 0$ |
| 3 | 1 | $t \leq 10$, $n \leq 20$ |
| 4 | 2 | $t \leq 100$, $n \leq 300$ |
| 5 | 2 | $t \leq 500$, $n \leq 1\,000$ |
| 6 | 2 | |

### Príklady

vstup

```
3 3
3 2 0
2 1 0
1 3 1
```

výstup

```
neexistuje
```

*Prvé dva vzťahy nám hovoria $r_3 \geq r_2$ a $r_2 \geq r_1$. Tretí vzťah nám hovorí $r_1 \geq r_3 + 1$. Tieto nerovnosti naraz zjavne nemôžu platiť.*

vstup

```
3 4
2 1 -4
1 2 -4
3 1 2
2 3 1
```

výstup

```
100 104 102
```

*Z prvých dvoch vzťahov vyplýva, že roky udalostí 1 a 2 sa líšia nanajvýš o štyri. Tretí a štvrtý vzťah hovoria, že udalosť 3 je aspoň dva roky po udalosti 1 a udalosť 2 aspoň rok po udalosti 3.*

*Vypísaná trojica rokov tieto obmedzenia spĺňa. Iným správnym riešením je napr. postupnosť rokov „−1 2 1“.*

## A-I-2 Obchodovanie

*Toto je **praktická úloha**. Pomocou webového rozhrania odovzdaj **funkčný, odladený program**.*

Malcolm má malú vesmírnu loď Serenity. Lieta ňou sem a tam vesmírom a obchoduje. V lodi sa nachádza malý úkryt, v ktorom sa dá prevážať jeden kus *zaujímavého* tovaru (teda takého, ktorý by veľmi zaujímal colníkov). Občas sa niekde naskytne možnosť nejaký zaujímavý tovar kúpiť a občas zase možnosť predať.

Malcolm si za posledný rok poctivo zapisoval všetky ponuky na nákup a predaj zaujímavého tovaru, ktoré stretol. Napíš program, ktorý z týchto poznámok vypočíta, koľko najviac mohol Malcolm zarobiť, ak by sa vždy optimálne rozhodoval, kedy niečo kúpiť a kedy to zase predať. (Na nákup má vždy dosť peňazí. V lodi nesmie nikdy mať viac ako jeden tovar naraz, takže kým nejaký tovar vezie, nesmie kúpiť ďalší.)

### Formát vstupu a výstupu

V prvom riadku vstupu je číslo $n$ ($1 \leq n \leq 200\,000$) udávajúce počet udalostí.
Zvyšok vstupu tvorí $n$ riadkov. Každý z nich popisuje jednu udalosť, a to v poradí, v ktorom nastali.
Popis $i$-tej udalosti má tvar „$u_i$ $t_i$ $c_i$“, kde:
- $u_i$ je písmeno udávajúce typ udalosti: `N` ak ide o možný nákup alebo `P` ak ide o možný predaj
- $t_i$ je prirodzené číslo ($1 \leq t_i \leq n$) udávajúce typ tovaru, ktorého sa ponuka týka
- $c_i$ je prirodzené číslo ($1 \leq c_i \leq 1\,000$) udávajúce cenu za daný typ tovaru

Na výstup vypíšte jeden riadok a v ňom jedno celé číslo: Malcolmov maximálny možný zisk.
(Zisk je rozdiel cien, za ktoré Malcolm tovary predal, a cien, za ktoré ich predtým nakúpil.)

### Obmedzenia a hodnotenie

Vstupy sú rozdelené do štyroch testovacích sád s nasledovnými dodatočnými obmedzeniami:
- v sade 1 (za 2 body) každý vstup obsahuje najskôr ponuky na nákup a potom ponuky na predaj,
- v sade 2 (za 2 body) v každom vstupe pre všetky ponuky platí $t_i = 1$ (je len jeden druh tovaru),
- v sade 3 (za 3 body) v každom vstupe platí $n \leq 2\,000$,
- v sade 4 (za 3 body) platia len vyššie uvedené všeobecné obmedzenia.

### Príklady

vstup

```
6
N 1 15
N 1 10
N 2 1
P 1 30
P 2 20
P 1 20
```

výstup

```
20
```

*Optimálne je prijať druhú ponuku (kúpiť tovar typu 1 za 10 kreditov) a potom štvrtú ponuku (predať tovar typu 1 za 30 kreditov). Tovar typu 2 si pri tomto scenári kúpiť nevieme, keďže v čase tretej ponuky naň nie je v lodi miesto.*

vstup

```
3
P 1 1000
N 1 1
P 2 1000
```

výstup

```
0
```

*Tovar typu 1 nevieme predať skôr ako ho kúpime. Taktiež nevieme využiť tretiu ponuku: nevieme predať tovar typu 2, keďže sme nikdy nemali príležitosť takýto tovar kúpiť. Optimálne je teda nerobiť nič.*

vstup

```
4
N 3 70
P 3 80
N 3 10
P 3 20
```

výstup

```
20
```

*Tu je optimálne postupne využiť všetky štyri ponuky.*

## A-I-3 Domy

*Toto je **teoretická úloha**. Pomocou webového rozhrania odovzdaj **súbor vo formáte PDF**, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách.*

Pozdĺž nedávno vyasfaltovanej ulice sa nachádza $n + 2$ pozemkov, za radom očíslovaných od 0 po $n + 1$. Pre každé $i$ od 1 po $n$ teda platí, že pozemok $i$ susedí s pozemkami $i - 1$ a $i + 1$.

Pozemky 0 a $n + 1$ ti nepatria. Sú prázdne a musia zostať prázdne. Všetky ostatné pozemky sú tvoje. Na každý pozemok sa zmestí jeden dom. Chceš tieto domy postaviť a potom čo najviac z nich predať.

Tvoja spoločnosť má $z$ zákazníkov, očíslovaných od 1 po $z$. Každý zákazník si chce kúpiť práve jeden dom. Zákazník $j$ chce dom s presne $p_j$ poschodiami.

Hlavný architekt mesta vydal nariadenie, že kvôli estetike musí pre každé dva susedné pozemky platiť, že stavby na nich majú podobnú výšku: rozdiel smie byť nanajvýš jedno poschodie. Toto nariadenie musí dodržať celá tvoja ulica, a to vrátane koncov – vedľa prázdneho pozemku smieš postaviť nanajvýš 1-poschodový dom.

Navrhni algoritmus, ktorý pre danú dĺžku ulice a dané požiadavky zákazníkov zistí, ako postaviť domy tak, aby sa ich následne predalo čo najviac.

### Formát vstupu a výstupu

V prvom riadku vstupu sú celé čísla $n$ a $z$. V druhom riadku vstupu sú kladné celé čísla $p_1, \ldots, p_z$.

Na výstup vypíš $n$ čísel $v_1, \ldots, v_n$, kde $v_i$ je počet poschodí domu, ktorý postavíš na pozemku $i$. Tieto výšky musia spĺňať podmienky zo zadania a musia predstavovať jedno možné optimálne riešenie – teda počet domov, ktoré vieme predať zákazníkom, musí byť maximálny.

### Obmedzenia a hodnotenie

Riešenia efektívne pre $n \leq 10^6$ môžu získať nanajvýš 9-10 bodov podľa presnej časovej zložitosti: riešenia s časovou zložitosťou $O(n)$ môžu získať plný počet bodov, riešenia so zložitosťou $O(n \log n)$ alebo podobnou nanajvýš 9.

Riešenia efektívne pre $n \leq 5\,000$, teda s časovou zložitosťou $O(n^2)$ alebo podobnou, môžu získať nanajvýš 5 bodov. Ľubovoľne pomalé korektné riešenie môže získať 3 body.

Dôležitou súčasťou riešenia je **dôkaz jeho správnosti**.

### Príklady

vstup

```
6 7
1 7 3 1 3 12 1
```

výstup

```
1 2 3 3 2 1
```

*Ak postavíme domy podľa ukážkového výstupu, vieme predať štyri z nich: napr. zákazníkom 1, 3, 4 a 5 predáme domy na pozemkoch 1, 3, 6 a 4. Dá sa dokázať, že bez ohľadu na to, aké domy postavíme, nebudeme vedieť predať viac ako štyri.*

*Existujú aj iné optimálne riešenia. Tvoj program môže vypísať ľubovoľné z nich.*

vstup

```
6 3
1 1 1
```

výstup

```
1 2 1 2 1 1
```

*Na takto postavenej ulici vieme ľubovoľné tri zo štyroch 1-poschodových domov predať našim trom zákazníkom. Viac ako tri domy predať nevieme, keďže nemáme viac zákazníkov.*

## A-I-4 Triedička

*Toto je, netradične,* ***praktická úloha****. V každej podúlohe odovzdaj príslušný program pre triedičku. K tejto úlohe patrí študijný text uvedený na nasledujúcich stranách. Aj v nasledujúcich kolách tohto ročníka OI stretneš úlohu nadväzujúcu na tento študijný text.*

V každej podúlohe domáceho kola je tvojou úlohou napísať korektný program pre triedičku, ktorý obsahuje **nanajvýš 256 inštrukcií** a rieši zadanú súťažnú úlohu.

Odovzdaný program spustíme na viacerých vopred pripravených testoch. V každom teste bude tvoj program spustený na triedičke, ktorá má **presne 2048 jadier**. Ak všetky testy správne vyrieši, dostaneš príslušné body.

Programy pre triedičku môžeš napísať ručne alebo si ich vygenerovať iným programom, ktorý si na to napíšeš v bežnom programovacom jazyku – to už je na tebe.

V každej podúlohe v domácom kole platí, že nám záleží len na výstupe jadra s ID = 0. Výstupy všetkých ostatných jadier môžu byť ľubovoľné.

**Podúloha A (3 body): minimum**

Každé jadro dostane na vstupe jeden člen postupnosti: celé číslo od 0 po $10^9$. Napíš program, ktorý vypočíta minimum tejto postupnosti. Presnejšie, jadro s ID=0 má na výstup dať najmenšie z čísel na vstupe.

**Podúloha B (3 body): najčastejší prvok**

Každé jadro dostane na vstupe jeden člen postupnosti: celé číslo od 0 po $10^9$. O tejto postupnosti je zaručené, že je usporiadaná v neklesajúcom poradí. Navyše platí, že niektoré číslo má v tejto postupnosti ostro viac výskytov ako každé iné – inými slovami, najčastejší prvok vstupnej postupnosti je unikátny. Napíš program, ktorý tento najčastejší prvok nájde. Jadro s ID=0 má na výstup dať jeho hodnotu.

**Podúloha C (4 body): súčet**

Každé jadro dostane na vstupe jeden člen postupnosti: celé číslo od 0 po $10^9$. Napíš program, ktorý vypočíta súčet tejto postupnosti. Jadro s ID=0 má tento súčet dať na výstup.

V podúlohách A a B môžeš dostať čiastočné body, ak tvoj program funguje na všeobecných vstupoch ale nesprávne ošetruje nejaké okrajové prípady.

## Študijný text: Triedička

Benjamín si vo voľnom čase doma poskladal nový typ paralelného počítača: triedičku.

Triedička obsahuje $n$ procesorových jadier. Jadrá majú navzájom rôzne celočíselné ID od 0 po $n-1$.

Jadrá triedičky sú vždy usporiadané do jedného radu vedúceho zľava doprava. Na začiatku výpočtu sú zoradené podľa ich ID (0 je vľavo).

Program pre triedičku tvorí postupnosť inštrukcií. Túto postupnosť vykonávajú všetky jadrá naraz – teda najskôr všetky naraz vykonajú prvú inštrukciu, potom všetky naraz druhú, a tak ďalej. Zoznam povolených inštrukcií nájdete nižšie.

Každá inštrukcia spočíta hodnotu, ktorú budeme volať výsledok. Každé jadro si pamätá všetky výsledky, ktoré počas výpočtu dostalo. Väčšina inštrukcií sa odvoláva na skôr spočítané výsledky. Napr. inštrukcia „`+ 1 2`“ sa pozrie na výsledok prvej a druhej inštrukcie a vypočíta ich súčet.

Jadrá sa navyše vedia pozerať aj do pamäte svojich susedov pomocou špeciálnych inštrukcií: inštrukcia „`left` $i$“, resp. „`right` $i$“, prečíta $i$-ty výsledok uložený v ľavom, resp. pravom, susedovi jadra, ktoré túto inštrukciu vykoná. Susednosť je pritom interpretovaná cyklicky – napr. najľavejšie jadro inštrukciou „`left`“ prečíta príslušnú hodnotu z najpravejšieho jadra.

Najdôležitejšou inštrukciou triedičky je inštrukcia „`sort i`“. Pri vykonávaní tej si každé jadro ako svoj kľúč zoberie svoj $i$-ty výsledok a potom sa všetky jadrá (v konštantnom čase) fyzicky preusporiajú podľa kľúča – jadro

s najmenším kľúčom skončí úplne vľavo, jadro s najväčším kľúčom úplne vpravo. Toto triedenie je stabilné: ak majú nejaké jadrá rovnaký kľúč, zachovajú si poradie medzi sebou (teda jadro, ktoré bolo pred inštrukciou viac vľavo, je aj po inštrukcii viac vľavo). Jadrá si aj po preusporiadaní zachovajú svoje pôvodné ID – po inštrukcii „`sort`“ teda už nemusí platiť, že najľavejšie jadro má ID 0.

### Inštrukčná sada

Nižšie uvádzame pre každú inštrukciu jej syntax a jej význam. Výsledok $i$-tej inštrukcie označujeme $r_i$. Inštrukcie, ktoré vyhodnocujú logické podmienky, majú výsledok 1 ak je podmienka splnená a výsledok 0, ak splnená nie je.

| **inštrukcia** | **výsledok** | **komentár** |
|---|---|---|
| `sort` $i$ | $r_i$ | jadrá sa navyše preusporiadajú podľa $r_i$ |
| `id` | ID tohto jadra | |
| `input` $i$ | $i$-ta z hodnôt na vstupe tohto jadra | |
| `const` $x$ | $x$ | |
| `copy` $i$ | $r_i$ | |
| `left` $i$ | $r_i$ jadra (cyklicky) naľavo | |
| `right` $i$ | $r_i$ jadra (cyklicky) napravo | |
| `+` $i$ $j$ | $r_i + r_j$ | |
| `-` $i$ $j$ | $r_i - r_j$ | |
| `*` $i$ $j$ | $r_i \cdot r_j$ | |
| `/` $i$ $j$ | celá časť podielu $r_i / r_j$ | ak $r_j = 0$, výsledkom je 0 |
| `%` $i$ $j$ | zvyšok, ktorý dá $r_i$ po delení $r_j$ | ak $r_j = 0$, výsledkom je 0 |
| `\|` $i$ $j$ | bitový or čísel $r_i$ a $r_j$ | |
| `&` $i$ $j$ | bitový and čísel $r_i$ a $r_j$ | |
| `^` $i$ $j$ | bitový xor čísel $r_i$ a $r_j$ | |
| `!` $i$ | logický not čísla $r_i$ | 1 ak $r_i = 0$, 0 inak |
| `=` $i$ $j$ | $r_i = r_j$ | |
| `!=` $i$ $j$ | $r_i \neq r_j$ | |
| `<` $i$ $j$ | $r_i < r_j$ | |
| `>` $i$ $j$ | $r_i > r_j$ | |
| `<=` $i$ $j$ | $r_i \leq r_j$ | |
| `>=` $i$ $j$ | $r_i \geq r_j$ | |
| `if` $i$ $j$ $k$ | ak $r_i \neq 0$ tak $r_j$, inak $r_k$ | |

### Aritmetika

Jadrá pracujú so 64-bitovými celými číslami so znamienkom, uloženými v tzv. doplnkovom kóde (two's complement). Každé číslo je teda z rozsahu od $-2^{63}$ po $2^{63} - 1$, vrátane, a je v pamäti uložené tým istým spôsobom ako 64-bitové celočíselné premenné vo všetkých bežných programovacích jazykoch.

Všetky aritmetické operácie sa robia modulo $2^{64}$. Inými slovami, ak nastane pretečenie (t.j., výsledok operácie leží mimo rozsahu uložiteľných hodnôt), postupne zahadzujeme najvýznamnejšie bity až kým nedostaneme uložiteľnú hodnotu.

### Vstup a výstup

Každé jadro môže mať viacero vstupov. Tie sú očíslované od 1 a vieme k nim pristupovať pomocou inštrukcie „`input`“. (V úlohách domáceho kola má každé jadro len jeden vstup, ten dostaneme zavolaním „`input 1`“.)

Za výstup jadra považujeme výsledok poslednej inštrukcie, ktorú príslušné jadro vykonalo.

### Formát programu

Program zapisujeme do textového súboru. Každá inštrukcia musí byť celá v jednom riadku. Názov inštrukcie a jej argumenty musia byť od seba oddelené bielymi znakmi (napr. medzerou alebo tabom). Prázdne riadky nič

nerobia a nepočítajú sa za inštrukcie. Znak # označuje začiatok komentára: všetko od tohto znaku po najbližší koniec riadku sa pri vykonávaní programu ignoruje.

Napr. nasledovný program s dvoma inštrukciami počíta druhú mocninu ID jadra:

```
id      # prvý výsledok je ID jadra vykonávajúceho program
* 1 1   # druhý výsledok je prvý výsledok vynásobený samým sebou
```

Inštrukcie si môžete pomenovať a následne v programe namiesto čísla inštrukcie použiť ako argument jej meno. Meno inštrukcie ide na začiatok riadku a môže ním byť ľubovoľný neprázdny reťazec znakov, v ktorom aspoň jeden znak nie je cifra.[1] Bezprostredne za menom inštrukcie musí byť dvojbodka.

Vyššie uvedený program teda môžeme upraviť nasledovne:

```
ídečko: id                 # prvý výsledok je ID jadra vykonávajúceho program
štvorec: * ídečko ídečko   # druhý výsledok je prvý výsledok vynásobený samým sebou
```

Ak použijete to isté meno viackrát, neskoršie použitie prepíše to skoršie. Inými slovami, ak meno inštrukcie použijeme ako argument, myslí sa tým najbližšia skoršia inštrukcia s daným menom.

### Príklad #1: súčet troch vstupov

V tejto úlohe má každé jadro tri vstupy a má za úlohu vypočítať ich súčet. V našom riešení použijeme pomenované inštrukcie, aby sme si ukázali, ako funguje používanie toho istého mena.

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

### Príklad #2: kto má minimum?

V tejto úlohe má každé jadro jeden vstup, pričom je zaručené, že máme viac ako jedno jadro a že najmenší zo všetkých vstupov je unikátny. Jadro, ktoré dostalo toto minimum, má na výstup dať hodnotu 1. Každé iné jadro má na výstup dať hodnotu 0.

```
vstup:       input 1
             sort vstup
vstup_vľavo: left vstup          # čiže výstup inštrukcie označenej "vstup" u ľavého suseda
             > vstup_vľavo vstup
```

Každé jadro sa pozrie na svoj vstup a potom sa preusporiadajú podľa jeho hodnoty. Jadro, v ktorom je minimum, sa teda presunie na začiatok radu.

V treťom kroku sa každé jadro pozrie na vstup svojho aktuálneho ľavého suseda a vo štvrtom túto hodnotu porovná so svojou. Vďaka usporiadaniu v kroku 2 je zjavné, že jediné jadro, ktoré naľavo od seba uvidí väčšiu hodnotu, je jadro obsahujúce minimum – jeho ľavý sused je jadro na konci radu obsahujúce maximum.

Cvičenie: Upravte riešenie tejto úlohy tak, aby každé jadro navyše skončilo na mieste, na ktorom začínalo.

### Príklad #3: detekcia duplikátov

V tejto úlohe každé jadro dostane dva vstupy, pričom je zaručené, že máme viac ako jedno jadro. Každé jadro má za úlohu zistiť, či existuje iné jadro, ktoré dostalo presne tie isté dva vstupy ako ono. Na výstup má vrátiť 1 ak duplikát jeho vstupov existuje a 0 ak nie.

[1] Názvy inštrukcií nesmú obsahovať biele znaky. Odporúčame v nich používať len znaky s ASCII hodnotami 33-126, ale malo by fungovať všetko rozumné vrátane písmen s diakritikou.

```
môj_vstup_1:    input 1
môj_vstup_2:    input 2
                sort môj_vstup_2
                sort môj_vstup_1

ľavý_vstup_1:   left môj_vstup_1
ľavý_vstup_2:   left môj_vstup_2
pravý_vstup_1:  right môj_vstup_1
pravý_vstup_2:  right môj_vstup_2

eq1:            = môj_vstup_1 ľavý_vstup_1
eq2:            = môj_vstup_2 ľavý_vstup_2
vľavo_rovnaké:  & eq1 eq2

eq1:            = môj_vstup_1 pravý_vstup_1
eq2:            = môj_vstup_2 pravý_vstup_2
vpravo_rovnaké: & eq1 eq2

odpoveď:        | vľavo_rovnaké vpravo_rovnaké
```

Jadrá sme si usporiadali najskôr podľa ich druhého a potom podľa ich prvého vstupu. Pripomenieme, že používame stabilné triedenie, takže pri druhom triedení jadrá, ktoré majú rovnakú hodnotu kľúča (teda prvého vstupu) zostanú v poradí, v ktorom boli pred týmto krokom (teda usporiadané podľa druhého vstupu). Inými slovami, po štvrtej inštrukcii máme celé pole usporiadané podľa usporiadaných dvojíc (prvý vstup, druhý vstup).

Jadrá, ktorých hodnoty vstupov sú duplikáty, teraz nutne tvoria v usporiadanom súvislé úseky. Inými slovami, pre každé jadro platí, že ak v poli existujú nejaké jeho duplikáty, tak po usporiadaní určite s nejakým svojim duplikátom susedí. Preto stačí, aby každé jadro porovnalo svoje vstupy so svojimi susedmi.

### Interpreter

Svoje programy si môžete aj doma otestovať použitím interpretera, ktorý sme si pre vás pripravili. Nájdete ho tu: https://oi.sk/apps/triedicka/

Interpreter dostane na svojom vstupe program, ktorý má odsimulovať, a vstup, na ktorom ho má spustiť. Interpreter najskôr skontroluje syntax všetkých inštrukcií, a ak bolo všetko v poriadku, tak postupne všetky vykoná a oznámi vám ich výsledok.

Interpreter môžete používať buď priamo online (na vyššie uvedenej stránke) alebo si ho stiahnuť do svojho počítača a spúšťať lokálne. Online verzia je obmedzená v tom, ako veľa vstupov a inštrukcií môžete zadať. Lokálna verzia takéto obmedzenia nemá. Na spustenie interpretera na svojom počítači budete potrebovať Python 3.

**ŠTYRIDSIATY ROČNÍK OLYMPIÁDY V INFORMATIKE**

Príprava úloh: Michal Anderle, Michal Forišek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: NIVAM – Národný inštitút vzdelávania a mládeže, Bratislava 2024