Priebeh celoštátneho kola

Celoštátne kolo 40. ročníka Olympiády v informatike, kategórie A, sa koná v dňoch 19.-22. 3. 2025. Na riešenie úloh druhého, praktického dňa majú súťažiaci 4,5 hodiny čistého času. Akékoľvek pomôcky okrem písacích potrieb (napr. knihy, výpisy programov, kalkulačky) sú zakázané.

Čo má obsahovať riešenie úlohy?

- Skompilovateľný program v podporovanom programovacom jazyku. Ak sa váš program nepodarí na našom testovacom počítači skompilovať a spustiť, bude automaticky hodnotený 0 bodmi.

Odovzdávanie riešení

Riešenia praktického dňa sa odovzdávajú v systéme OIsubmit: https://oi.sk/oisubmit/
Priamo v systéme je pri každej úlohe uvedený presný **časový a pamäťový limit** pre jej riešenia.

Hodnotenie riešení druhého (praktického) dňa

Sú tri úlohy. Ku každej úlohe máme pripravených niekoľko sád testovacích vstupov. Sada vstupov pozostáva z jedného alebo viacerých testovacích vstupov. Za každú sadu vstupov, ktorej každý vstup váš program správne vyrieši, získate v zadaní uvedený počet bodov. Dokopy sa za každú úlohu dá získať 10 bodov.
Testovanie na každom vstupe prebieha samostatne. Spustíme váš program a na štandardný vstup mu dáme konkrétne vstupné údaje. Hovoríme, že váš program daný vstup vyriešil, ak splní nasledujúce kritériá:

- Skončí skôr ako uplynie stanovený časový limit.
- Neprekročí stanovený pamäťový limit.
- Skončí korektne, nie chybou počas behu.
- Dáta, ktoré vypíše na štandardný výstup, tvoria korektný výstup, zodpovedajúci danému vstupu.
- Nebude používať žiadne funkcie zakázané kvôli bezpečnosti testovacieho systému.

Počas súťaže môžete priebežne odovzdávať svoje riešenia. Odovzdané riešenie bude otestované a dozviete sa svoj bodový zisk. Výsledný počet bodov za každú úlohu bude rovný **maximu počtov bodov**, ktoré získali jednotlivé vaše odovzdané riešenia tejto úlohy.
(V prípade preťaženia testovača môžu organizátori obmedziť toto priebežné testovanie na vhodnú podmnožinu všetkých testovacích dát a po skončení súťaže všetky riešenia dotestovať. V prípade technickej chyby, ktorá naruší plánované vyhodnocovanie riešení, môžu organizátori všetky odovzdané riešenia pretestovať.)

A-III-4 Farebné oázy

Utopistan je tvorený n izolovanými oázami. Oázy majú čísla od 0 po $n-1$. Existuje n farieb ktoré tiež majú čísla od 0 po $n-1$. Každá oáza má svoju vlajku, pričom platí, že oáza i má vlajku farby i.
Veľký vezír sa rozhodol, že v krajine postupne spraví q zmien, a potom to celé oslávi majestátnou cestou po krajine. Každá zmena je jedného z dvoch typov:

1. Vezír zvolí dve rôzne farby i a j. Prefarbíme v celej krajine všetky vlajky, ktoré majú farbu i, farbou j.
2. Vezír zvolí dve rôzne farby i a j. Pre každú dvojicu oáz, v ktorej má momentálne jedna vlajku farby i a druhá farby j, kúpime karavánu, ktorá bude odteraz už navždy premávať medzi týmito dvoma oázami.

Po vykonaní poslednej zmeny sa vezír bude chcieť previezť krajinou, a to z oázy u do oázy v. Aby ukázal, aké skvelé zmeny spravil, chce celú cestu z u do v prejsť tak, že postupne pocestuje s niektorými z karaván, ktoré počas zmien nakúpil.
Samozrejme, čím rýchlejšie sa vezír dostane z u do v, tým viac budú obyvatelia zmenami nadšení. Preto bude treba vezírovi nájsť **najkratšiu cestu** z u do v – teda takú, na ktorej postupne použije čo najmenej karaván.
Napíš program, ktorý načíta popis všetkých zmien a následne nájde dĺžku najkratšej cesty z u do v.

Formát vstupu a výstupu

V prvom riadku vstupu sú kladné celé čísla n a q: n je počet oáz (a zároveň farieb) a q je počet zmien.
V druhom riadku sú celé čísla u a v ($0 \leq u, v < n$, $u \neq v$): oázy, medzi ktorými chce na konci vezír cestovať.
Zvyšok vstupu tvorí q riadkov – jeden pre každú zmenu v chronologickom poradí. Riadok tvaru „`p` i j“ predstavuje prvý typ zmeny (prefarbenie z i na j), riadok tvaru „`c` i j“ predstavuje druhý typ zmeny (nové karavány medzi farbami i a j). V každej zmene platí $0 \leq i, j < n$ a $i \neq j$.
Je zaručené, že vstupy sú zvolené tak, aby sa v každom dalo nejako dostať karavánami z u do v.

Na výstup vypíš jeden riadok a v ňom jedno celé číslo: minimálnu vzdialenosť (meranú počtom karaván, ktoré treba použiť) medzi oázami u a v vo výslednej krajine.

Obmedzenia a hodnotenie

Vstupy sú rozdelené do sád. V každej sade platia iné obmedzenia. Body za sadu dostaneš, ak tvoj program správne a dostatočne efektívne vyrieši všetky vstupy z danej sady. Údaje o sadách sú v tabuľke nižšie.

č. sady	body	max n	max q	ďalšie obmedzenia
1	2	200 000	300 000	farby len miznú, všetky zmeny sú druhého typu
2	1	200 000	300 000	farby len miznú, najskôr sa robia zmeny prvého, a potom druhého typu
3	1	300	900	farby len miznú
4	2	10 000	30 000	farby len miznú
5	2	200 000	300 000	farby len miznú
6	2	200 000	300 000	

„Farby len miznú“ znamená, že pri každom prefarbení „`p` i j“ existuje v krajine aspoň jedna oáza farby i aj aspoň jedna oáza farby j. V týchto vstupoch je teda zaručené, že v každej zmene typu `p` niečo prefarbíme, a tiež že akonáhle niekedy prefarbíme oázy z farby i na farbu j, už nikdy nebude existovať oáza farby i.

Príklady

vstup

```
6 5
0 4
c 0 1
p 4 5
c 3 5
p 1 2
c 2 5
```

výstup

```
2
```

V prvej zmene pridáme karavánu medzi oázami 0 a 1. V tretej pridáme dve karavány: 3-4 a 3-5. V piatej pridáme štyri karavány: 1-4, 1-5, 2-4 a 2-5.
Z oázy 0 do oázy 4 teda neexistuje priamo idúca karavána, ale vieme tam vezíra dostať pomocou dvoch karaván: jednou z 0 do 1 a potom druhou z 1 do 4.

vstup

```
6 6
0 5
c 1 0
c 1 2
c 2 3
c 2 1
c 3 4
c 4 5
```

výstup

```
5
```

Jediná možnosť je poslať vezíra cestou $0 \to 1 \to 2 \to 3 \to 4 \to 5$.

(Medzi oázami 1 a 2 premávajú dve karavány, ale to je nám zjavne jedno. Vezíra môžeme poslať ľubovoľnou z nich.)

vstup

```
4 4
1 2
p 0 1
p 1 2
p 2 0
c 3 0
```

výstup

```
2
```

Po prefarbeniach majú oázy 0, 1 aj 2 vlajku farby 0. Následne pridáme karavány medzi nimi a oázou 3. Najkratšia cesta z oázy 1 do oázy 2 je teda $1 \to 3 \to 2$.

(Tento príklad by mohol byť len v sade 6.)

A-III-5 Jednosmerná jednokoľajka

Z eurofondov sa spravila veľkolepá rekonštrukcia železnice z Jablonice do Brezovej pod Bradlom. Po koľaji z Jablonice do Brezovej teraz premávajú až tri typy vlakov: osobné, rýchliky a expresy.
Nik bohužiaľ pri rekonštrukcii nemyslel na to, postaviť kdekoľvek po ceste viac ako jednu koľaj, a tak sa často stáva, že rýchlejší vlak ostane trčať za pomalším a naberie meškanie.
V súčasnom grafikone je na úseku z Jablonice do Brezovej naplánovaných n vlakov. O každom z nich poznáme čas p_i jeho plánovaného odchodu z Jablonice a jeho typ. Navyše poznáme hodnoty $t_{Os} \geq t_R \geq t_{Ex}$: časy, ktoré potrebuje osobák, rýchlik a expres na prejdenie trate z Jablonice do Brezovej, ak má celý čas voľnú koľaj.
Nedávno ťa najali ako výpravcu v Jablonici. Po pár dňoch práce ťa osvietilo: možno vieš zlepšiť situáciu s meškaniami! Akonáhle už v nejakom poradí pustíš vlaky na trať, v tom poradí na nej ostanú až do cieľa cesty – nemajú sa kde predbiehať. Trik je v tom, že môžeš niektoré vlaky pozdržať už priamo v Jablonici a na trať ich vypraviť až **neskôr** ako v ich plánovaný čas odchodu. Paradoxne tak môžeš dosiahnuť, že zmenšíš celkové meškanie. (Na zrekonštruovanej stanici v Jablonici je dosť koľají na to, aby sa tam dalo pozdržať aj všetky vlaky naraz.)

Technické detaily

Vlaky budeme pre jednoduchosť považovať za body. Ľubovoľne veľa vlakov sa môže nachádzať na trati tesne za sebou (akoby v tom istom bode, ale stále majú fixné poradie, ktoré sa už nevie zmeniť). Špeciálne je možné aj vypraviť na trať naraz viacero vlakov, pričom ak to spravíš, môžeš si pre ne zvoliť ľubovoľné poradie, v ktorom na trať tesne po sebe vojdú.
Keďže sa vlaky nevedia predbiehať, ak rýchlejší vlak, ktorý by do cieľa dorazil v čase a, dobehne pomalší vlak, ktorý má do cieľa príchod v čase $b > a$, pôjdu ďalej za sebou rýchlosťou pomalšieho vlaku a teda oba dorazia do cieľa v čase b (samozrejme, ak cestou nestretnú ešte pomalší vlak).
Ak má vlak typu X_i v grafikone odchod v čase p_i, tak je jeho plánovaný príchod do Brezovej v čase $q_i = p_i + t_{X_i}$. Ak do cieľa skutočne dorazí až v čase r_i, tak jeho meškanie je $r_i - q_i$.

Celkové meškanie je rovné súčtu meškaní všetkých n vlakov. Tvojou úlohou je zistiť, aké najmenšie celkové meškanie vieš ako výpravca v Jablonici dosiahnuť.

Formát vstupu a výstupu

V prvom riadku vstupu je číslo n udávajúce počet vlakov.
V druhom riadku sú tri kladné celé čísla: t_{Os}, t_R a t_{Ex}. Platí pre ne $1 \leq t_{Ex} \leq t_R \leq t_{Os} \leq 10^5$.
Zvyšok vstupu tvorí n riadkov popisujúcich jednotlivé vlaky. Popis každého vlaku má najskôr jeho plánovaný čas odchodu (celé číslo p_i spĺňajúce $0 \leq p_i \leq 10^9$) a potom jeho typ (reťazec X_i, ktorý je jedným z reťazcov „`Os`“, „`R`“ a „`Ex`“).
Vlaky sú zoradené podľa ich času odchodu. Vlaky s rovnakým časom odchodu môžu byť na vstupe uvedené v ľubovoľnom poradí.

Na výstup vypíš jeden riadok a v ňom jedno celé číslo: najmenšie dosiahnuteľné celkové meškanie.

Obmedzenia a hodnotenie

Vstupy sú rozdelené do ôsmich sád. V každej z nich platia iné obmedzenia. Za každú sadu sa dajú získať nejaké body. Body za sadu dostaneš, ak tvoj program správne a dostatočne efektívne vyrieši všetky vstupy z danej sady. Údaje o jednotlivých sadách sú v tabuľke nižšie.

č. sady	body	max n	ďalšie obmedzenia
1	1	8	–
2	1	14	–
3	1	40	–
4	1	100	$t_{Os} \leq 4$, $t_R = 1$, žiadne expresy, jedinečné časy
5	1	100	$p_i \leq 1000$, $t_{Os} \leq 100$, $t_R \leq 100$, $t_{Ex} \leq 100$
6	2	100	žiadne expresy
7	1	100	–
8	2	400	–

Obmedzenie „jedinečné časy“ znamená, že žiadne dva vlaky nemajú ten istý plánovaný čas odchodu p_i.

Príklady

vstup

```
5
12 8 4
3 Os
5 R
7 Os
9 Ex
11 Os
```

výstup

```
6
```

Prvé dva vlaky vypravíme v plánovaných časoch.

Osobák má prázdnu trať, do Brezovej príde v čase 15.

Rýchlik by tam bol v čase 13, cestou ale dobehne osobák a dorazí tesne za ním v čase 15 (s meškaním 2).

V čase 9 vypravíme najskôr načas expres a potom s dvojminútovým meškaním druhý osobák. Expres má tiež plánovaný príchod v čase 13, namiesto toho ale časom dobehne oba skôr vypravené vlaky a príde do cieľa s nimi – tiež s meškaním 2. Osobák si svoje dvojminútové meškanie udrží až po Brezovú, kam dorazí v čase 21.

Tretí osobák vypravíme načas v čase 11 a do Brezovej príde tiež načas v čase 23.

vstup

```
5
4 1 1
3 R
5 Os
6 R
7 R
8 R
```

výstup

```
3
```

A-III-6 Triedička III

K tejto úlohe patrí študijný text, ktorý nájdeš na konci zadaní aj na stránke s interpreterom (viď časť Interpreter nižšie). Študijný text je identický s tým z domáceho a krajského kola.

Opäť ideme programovať triedičku. Dostaneš zadanú jednu konkrétnu úlohu, ale s viacerými rôzne ťažkými sadami vstupov. Naprogramovať a odovzdať máš tentokrát **generátor** programu pre triedičku – teda program v bežnom programovacom jazyku, ktorý na vstupe dostane číslo n udávajúce počet jadier triedičky (a nejaké ďalšie užitočné informácie, ktoré si popíšeme neskôr) a na výstup vypíše program pre n-jadovú triedičku, ktorý rieši zadanú úlohu.

Súťažná úloha

Každé jadro i dostane na vstupe dve čísla.
Prvé z nich (označíme si ho s_i) bude číslo jadra, ktoré jadro i *sleduje*. Toto číslo bude z rozsahu od 0 po $n-1$.
Druhé z nich (to označíme t_i) budeme volať *tajomstvo* jadra i. Tajomstvami môžu byť ľubovoľné 64-bitové čísla.
Tvojou úlohou je napísať program, ktorým každé jadro zistí a dá na výstup tajomstvo jadra, ktoré sleduje.
Formálne, jadro i má na výstup dať hodnotu t_{s_i}.

Príklad vstupu a výstupu pre triedičku

vstup

```
1 78
5 38
5 60
5 57
6 -3
0 48
5 48
7 35
```

výstup

```
38 48 48 48 13 78 48 35
```

V i-tom riadku vstupu (číslované od 0) sú vstupy pre jadro i. V ukážkovom výstupe sú správne výstupy jadier v poradí od 0 po 7.

Pre ďalšie príklady vstupu viď sekciu Interpreter.

Vstup a výstup generátoru

Na vstupe dostane tvoj generátor jeden riadok a v ňom medzerou oddelené tri údaje: číslo n, znak d a znak p.
Znaky d a p popisujú obmedzenia, ktoré v aktuálnej sade platia. Tvoj generátor môže (ale samozrejme nemusí) túto informáciu využiť na to, aby pre rôzne sady vygeneroval rôzne programy.
Pre každú použitú kombináciu n, d a p tvoj generátor raz spustíme a jeho výstup použijeme ako program pre triedičku pre všetky vstupy spĺňajúce príslušné obmedzenia.
Znak d je '`S`', '`R`' alebo '`-`'. Znak p je '`P`' alebo '`-`'. Pomlčka má v oboch prípadoch význam „žiadne ďalšie obmedzenie“. Ostatné znaky majú nasledovný význam:

- `S` (spoločný cieľ sledovania): všetky jadrá sledujú to isté jadro (t.j. majú tú istú hodnotu s_i)
- `R` (rôzne ciele sledovania): žiadne dve jadrá nemajú tú istú hodnotu s_i
- `P` (parita sledovania): párne jadrá sledujú nepárne a naopak – teda i má vždy opačnú paritu ako s_i

Na výstup musí tvoj generátor vypísať program vo formáte, ktorý bol používaný aj vo všetkých skorších úlohách.
Vypísaný program smie mať **nanajvýš 2500 inštrukcií**.

Obmedzenia a hodnotenie

V každom vstupe platí, že $n \geq 2$ a n je **mocninou dvoch**.
Je sedem bodovaných sád vstupov. Informácie o nich sú v nasledujúcej tabuľke.

sada	body	max n	d	p	komentár
0	0	2^1	-	-	jediný vstup „`2 - -`“ z príkladu nižšie
1	1	2^7	-	-	
2	1	2^{14}	S	-	
3	2	2^{14}	R	P	
4	1	2^{14}	-	P	
5	3	2^{14}	R	-	
6	1	2^{11}	-	-	
7	1	2^{14}	-	-	

Interakcia s testovačom

Generátor má časový limit 1 sekunda a pamäťový limit 1 GB. Ak nezbehne úspešne, dostaneš príslušnú chybovú hlášku (napr. „prekročený časový limit“ alebo „chyba počas behu“).
Ak generátor úspešne zbehne, dostaneš buď verdikt „OK“ (ak máš plný počet bodov) alebo verdikt „zlá odpoveď“ (vo všetkých ostatných prípadoch). V stĺpci „čas behu“ bude ukázaný čas behu *generátora*. (Čas behu *triedičky* nemeriame, keďže tá už má obmedzený počet inštrukcií.)
Navyše v detailoch pre každú neúspešne vyriešenú sadu uvidíš, aký typ chyby pri jej riešení nastal – teda buď sa dozvieš, že tvoj generátor nevyrobil korektný program a prečo, alebo sa dozvieš, že vygenerovaný program nejaký test nevyriešil správne.

Príklad vstupu a výstupu pre generátor

vstup

```
2 - -
```

výstup

```
sledujem:  input 1
tajomstvo: input 2
moje_id:   id
egoista:   = sledujem moje_id
ľavé_t:    left tajomstvo
výstup:    if egoista tajomstvo ľavé_t
```

Generátor by mal byť program v ľubovoľnom podporovanom programovacom jazyku. Pre $n = 2$ by mohol vypísať program uvedený vyššie – dá sa ukázať, že pre dve jadrá tento program vždy dá správnu odpoveď.

Úloha sa dá vyriešiť aj tak, že namiesto programovania generátora ručne vyrobíš jeden program pre triedičku. Tento program musí byť napísaný tak, aby fungoval pre všetky povolené počty jadier. Ak takýto program ručne vyrobíš, odovzdať ho vieš napr. tak, že mu na začiatok pridáš riadok `print("""` a na koniec riadok `""")`. Takto dostaneš generátor v Pythone 3, ktorý tvoj program vypíše.

Interpreter

Svoje programy si môžeš otestovať použitím interpretera. Nájdeš ho tu: https://oi.sk/apps/triedicka/ck/.
Interpreter je identický s tým z domáceho kola. 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 ti ich výsledok.
Interpreter môžeš 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 lokálne spustenie interpretera treba použiť `python3`.
Na stránke interpretera nájdeš tiež niekoľko rôznych príkladov vstupu s $n = 8$ ktoré zodpovedajú rôznym sadám testov.

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:

```
ídéčko: id                  # prvý výsledok je ID jadra vykonávajúceho program
štvorec: * ídéčko ídéčko    # 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.

[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.

```
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.

```
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.

ŠTYRIDSIATY ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Forišek
Recenzia: Michal Anderle, Tomáš Belan, Paulína Smolárová
Slovenská komisia Olympiády v informatike
Vydal: NIVAM – Národný inštitút vzdelávania a mládeže, Bratislava 2025