Priebeh krajského kola

Krajské kolo 40. ročníka Olympiády v informatike, kategória A, sa koná 21. 1. 2025 v dopoludňajších hodinách. Na riešenie úloh majú súťažiaci **4 hodiny čistého času**. Rôzne úlohy riešia súťažiaci na samostatné listy papiera. Akékoľvek pomôcky okrem písacích potrieb (napr. knihy, výpisy programov, kalkulačky) sú zakázané.

Čo má obsahovať riešenie úlohy?

- Slovne popíšte algoritmus.
  Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného algoritmu/programu.
- Zdôvodnite správnosť vášho algoritmu.
- Uveďte a zdôvodnite jeho časovú a pamäťovú zložitosť.
- Podrobne uveďte dôležité časti algoritmu, ideálne vo forme programu v nejakom bežnom programovacom jazyku (napr. C++, Python, Java, Pascal).
- V prípade, že používate vo svojom programovacom jazyku knižnice, ktoré obsahujú implementované dátové štruktúry a algoritmy (napr. STL pre C++), v popise algoritmu stručne vysvetlite, ako by ste napísali program s rovnakou časovou zložitosťou bez použitia knižnice.

Hodnotenie riešení

Za každú úlohu môžete získať od 0 do 10 bodov.

Pokiaľ nie je v zadaní povedané ináč, najdôležitejšie dve kritériá hodnotenia sú v prvom rade **správnosť** a v druhom rade **efektívnosť** navrhnutého algoritmu. Na výslednom počte bodov sa môže prejaviť aj kvalita popisu riešenia a zdôvodnenie tvrdení o jeho správnosti a efektívnosti.

Efektívnosť algoritmu posudzujeme vypočítaním jeho časovej zložitosti – funkcie, ktorá hovorí, ako dlho vykonanie algoritmu trvá v závislosti od veľkosti vstupných parametrov. Nezávisí pri tom na konštantných faktoroch, len na rádovej rýchlosti rastu tejto funkcie.

V zadaní úloh uvádzame časť „Hodnotenie“, v ktorej nájdete približné limity na veľkosť vstupných údajov. Pod pojmom „efektívne vyriešiť“ chápeme to, že váš program spustený na modernom počítači by mal dať odpoveď nanajvýš do niekoľkých sekúnd.

Údaje z tejto časti zadania by mali slúžiť hlavne na to, aby ste o riešení, ktoré vymyslíte, vedeli približne povedať, koľko bodov zaň dostanete.

A-II-1 Hrozienka nemajú mať nožičky!

Aby ježibaba rýchlejšie vykŕmila Janka a Marienku, upiekla im v peci koláč s hrozienkami. Teda, to bol pôvodný plán. Ježibaba už totiž horšie vidí a nevšimla si, že sa jej do hrnčeka s hrozienkami nanosili pavúky.
Výsledkom pečenia je koláč rozmerov $n \times n$. Môžeme si ho predstaviť ako mriežku tvorenú n^2 štvorcovými políčkami. Na každom políčku je buď hrozienko alebo pavúk.
Ježibaba správne odhaduje, že deťom sa príliš nebude chcieť do koláča, ak na ňom bude *priveľa* pavúkov. Potrebovala by preto odkrojiť nejaké okraje tak, aby ostal **čo najväčší štvorcový** koláč, na ktorom bude nanajvýš p pavúkov.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla n a p ($1 \leq p \leq n^2$). Zvyšok vstupu tvorí $n \times n$ znakov: P je políčko s pavúkom, H je políčko s hrozienkom.
Na výstup vypíšte jeden riadok a v ňom jedno číslo: najväčšie k, pre ktoré sa niekde na ježibabinom koláči nachádza štvorec rozmerov $k \times k$ obsahujúci nanajvýš p pavúkov.

Obmedzenia a hodnotenie

Pri spisovaní riešení tejto úlohy nezabudnite okrem dôkazu správnosti spraviť poriadny **horný odhad časovej zložitosti** v najhoršom prípade, a taktiež tento odhad **dostatočne zdôvodniť**.
Pri hodnotení vašich riešení tejto úlohy budeme primárne prihliadať na ten odhad časovej zložitosti, ktorý sa vám podarí korektne zdôvodniť. (Teda napr. ak niekto navrhne algoritmus s kvadratickou časovou zložitosťou, ale zdôvodní len horný odhad časovej zložitosti $O(n^3)$, môže toto riešenie byť ohodnotené ako kubické.)

Plný počet bodov môžu získať riešenia s časovou zložitosťou kvadratickou od n.
Riešenia s časovou zložitosťou medzi kvadratickou a kubickou od n môžu získať nanajvýš 9 bodov.
Riešenia s kubickou časovou zložitosťou môžu získať nanajvýš 6 bodov.
Riešenia s časovou zložitosťou $O(n^4)$ alebo ešte horšou môžu získať nanajvýš 2 body.

Príklady

vstup

```
4 3
PPPH
HHPP
PHHP
HPHP
```

výstup

```
3
```

Štvorec rozmerov 3×3 v ľavom dolnom rohu koláča obsahuje len tri pavúky.

vstup

```
4 15
PPPH
HHPP
PHHP
HPHP
```

výstup

```
4
```

Ak sú deti ochotné zjesť až 15 pavúkov, môžeme im dať celý koláč – ten ich má len 9.

A-II-2 Recyklujeme

Na lúke za mestom sa práve skončil festival. Po nezodpovedných návštevníkoch ostal kopec odpadkov, ktoré treba odpratať: f prázdnych plastových fliaš a p prázdnych hliníkových plechoviek. Na lúke sa tiež nachádza jeden kôš na plasty a jeden na hliník. (Lúku si môžeme predstaviť ako nekonečnú vodorovnú rovinu a všetky objekty ako body na nej.)
Ale nebojte sa, nebudeme odkázaní na manuálnu prácu. Na upratanie lúky máme maličkého robota, ktorého môžeme naprogramovať, aby všetko pozbieral za nás.
Robot začína na rovnakých súradniciach ako kôš na plasty. Vie sa pohybovať ľubovoľným smerom. Keď príde na súradnice nejakého odpadu, vie ho zdvihnúť, a keď odpad prinesie na súradnice správneho koša, vie ho tam vyhodiť. Naraz vie robot niesť vždy len jeden odpad – teda keď nejaký zodvihne, musí ho najskôr zahodiť do koša a až potom môže ísť zodvihnúť ďalší.
Robot potrebuje 1 sekundu na prejdenie 1 metra. Z Pytagorovej vety teda vieme, že na prechod zo súradníc (x_1, y_1) na (x_2, y_2) potrebuje robot $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ sekúnd. Všetky ostatné úkony sa dejú okamžite.
Napíšte program, ktorý nájde najrýchlejší spôsob, ako dostať všetky odpadky z lúky do správnych košov.

Formát vstupu a výstupu

V prvom riadku vstupu sú súradnice koša na plast (a teda aj miesta, kde robot začína). V druhom riadku sú súradnice koša na hliník. Nasleduje popis fliaš: najskôr riadok s číslom f a potom f riadkov so súradnicami jednotlivých fliaš. Zvyšok vstupu tvorí popis plechoviek v rovnakom formáte: najskôr p, potom súradnice. Všetky súradnice sú celé čísla v metroch.
Na výstup vypíšte jedno reálne číslo: minimálny čas, za ktorý vie náš robot dostať všetky odpadky do správnych košov.

Obmedzenia a hodnotenie

Plný počet bodov môžu získať riešenia, ktoré efektívne vyriešia ľubovoľný vstup s $f + p \leq 100\,000$ (a to aj ak majú časovú zložitosť o trochu horšiu od optimálnej).

Za riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $f + p \leq 1000$, sa dá získať 7 bodov.
Za riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $f + p \leq 20$, sa dá získať 5 bodov.
Za riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $f + p \leq 10$, sa dá získať 3 body.

Príklady

vstup

```
7 5
17 15
3
6 5
6 6
7 6
0
```

výstup

```
6.828427124746
```

Bez ohľadu na to, v akom poradí bude robot zbierať tieto tri fľaše, dokopy prejde $4 + 2\sqrt{2}$ metrov a to mu bude trvať $4 + 2\sqrt{2}$ sekúnd.

vstup

```
0 0
7 -7
1
6 -6
1
4 -3
```

výstup

```
19.8994949366
```

Robot prejde z $(0, 0)$ ku plechovke, zdvihne ju a prenesie ju do koša na hliník. Odtiaľ prejde ku fľaši, zdvihne ju a odnesie do koša na plasty.
Tentoraz na poradí záleží: keby išiel najskôr vyhodiť fľašu, potreboval by na upratanie lúky približne o 7.071 sekundy viac.

A-II-3 Decentralizácia

V Absurdistane je n oáz. Sú očíslované od 0 po $n-1$. Medzi oázami vedú trasy karaván. Trás je presne $n-1$. Každá spája nejakú konkrétnu dvojicu oáz, a to tak, že sa karaváno dá po týchto trasách prejsť z každej oázy do každej inej. (Sieť trás karaván má teda stromovú topológiu.)
Dá sa ukázať, že pre každú dvojicu oáz musí existovať práve jeden spôsob, ako sa karaváno vie dostať z jednej oázy do druhej bez zbytočných zachádzok. Celkovú dĺžku trás, ktoré treba prejsť, aby sme sa dostali z jednej oázy do druhej, nazveme ich *vzdialenosťou*.
Priemer Absurdistanu je rovný najväčšej zo všetkých vzdialeností medzi jeho oázami.

Veľký vezír sa rozhodol, že si v niektorej oáze postaví palác. Táto oáza sa stane hlavnou oázou Absurdistanu. Akonáhle sa tak stane, zvyšok krajiny bude prirodzene rozdelený na niekoľko *okresov*. Dve oázy sú v tom istom okrese, ak vieme po cestách prejsť z jednej do druhej bez toho, aby sme navštívili hlavnú oázu. (Hlavná oáza nepatrí do žiadneho okresu.)
Veľký vezír by chcel hlavnú oázu zvoliť tak, aby žiaden okres nebol príliš veľký. Formálne, pre každé konkrétne rozdelenie Absurdistanu na okresy môžeme jeho *rozsahom* nazvať najväčší z priemerov jeho okresov.[1] Veľký vezír by chcel zo všetkých možných rozdelení zvoliť to, ktorého rozsah je najmenší možný.

Formát vstupu a výstupu

V prvom riadku vstupu je číslo n udávajúce počet oáz. Zvyšok vstupu tvorí $n-1$ riadkov, každý popisujúci jednu cestu. Popis cesty pozostáva z čísel oáz, ktoré spája, a kladného celého čísla udávajúceho jej dĺžku.
Na výstup vypíšte dva riadky: v prvom najmenší možný rozsah rozdelenia Absurdistanu a v druhom jednu možnú voľbu hlavnej oázy, ktorá povedie k tomuto rozsahu.

Obmedzenia a hodnotenie

Plný počet bodov môžu získať riešenia, ktoré efektívne vyriešia ľubovoľný vstup s $n \leq 1\,000\,000$.
Do 7 bodov môžete dostať za riešenie, ktoré je efektívne na vstupoch v ktorých platí $n \leq 70\,000$, každá cesta medzi dvoma oázami má dĺžku 1 a celý Absurdistan má priemer nanajvýš 200. Ak chcete odovzdať takéto riešenie, výrazne to uveďte na začiatku popisu.
Riešenia, ktoré efektívne vyriešia ľubovoľný vstup s $n \leq 3000$, môžu získať 5 bodov.
Riešenia, ktoré efektívne vyriešia ľubovoľný vstup s $n \leq 100$, môžu získať 2 body.

Príklady

vstup

```
5
0 1 100
1 2 100
2 3 100
2 4 100
```

výstup

```
100
2
```

Ak vyhlásime oázu 2 za hlavnú, vzniknú nám 3 okresy: jeden tvoria oázy 0 a 1 (priemer 100), druhý oáza 3 (priemer 0) a tretí oáza 4 (priemer 0). Každá iná hlavná oáza by viedla k rozsahu ostro väčšiemu ako 100.

vstup

```
5
0 1 100
1 2 10
2 3 10
2 4 10
```

výstup

```
20
1
```

Tentokrát je optimálne vyhlásiť za hlavnú ľubovoľnú z oáz 0 alebo 1. Dostaneme síce rôzne okresy (v jednom prípade len jeden okres, v druhom dva), v oboch prípadoch však bude toto rozdelenie na okresy mať rozsah 20. Ak by bola hlavná oáza inde, skončili by 0 a 1 v tom istom okrese a ten by mal priemer aspoň 100.

[1] Priemer okresu definujeme rovnako ako priemer celého Absurdistanu, len pochopiteľne tentokrát berieme do úvahy len oázy tvoriace príslušný okres a trasy medzi nimi.

A-II-4 Triedička II

V každej podúlohe krajského kola je tvojou úlohou napísať korektný program pre triedičku, ktorá má **presne $n = 2^{20}$ jadier**. Na plný počet bodov je potrebné napísať správny program, ktorý obsahuje **nanajvýš 10 000 inštrukcií** (teda rádovo menej ako n). Korektné programy s viac inštrukciami budú hodnotené menej bodmi.

Súčasťou riešenia každej podúlohy by mal byť aj program, aj jeho slovný popis.
Ak váš program obsahuje nejaké opakujúce sa časti, stačí to samozrejme vhodne popísať – napríklad označiť blok inštrukcií a uviesť počet opakovaní. V prípade komplikovanejšej štruktúry vášho programu pre triedičku je najlepšie popísať ho tak, že uvediete pseudokód, ktorý ho vygeneruje.
Vo všetkých podúlohách krajského kola platí, že nám záleží len na výstupoch jednotlivých jadier – každé jadro má dať výstup zodpovedajúci jeho ID. Poradie jadier na konci výpočtu môže byť ľubovoľné.

Podúloha A (3 body): si priemerný?
Každé jadro dostane na vstupe jeden prvok postupnosti: celé číslo od 0 po 10^9. Napíš program, ktorý dosiahne, že každé jadro dá na výstup hodnotu -1, ak je jeho vstup podpriemerný (t.j. menší ako priemer celej postupnosti), hodnotu 0, ak je jeho vstup priemerný, alebo hodnotu $+1$, ak je jeho vstup nadpriemerný.

Podúloha B (3 body): prefixové súčty
Každé jadro dostane na vstupe jeden prvok postupnosti: celé číslo od 0 po 10^9. Napíš program, ktorý dosiahne, že každé jadro dá na výstup súčet vstupov všetkých jadier, ktoré majú menšie alebo rovné ID. (Teda jadro s ID 0 má dať na výstup svoj vstup, jadro s ID 1 súčet vstupov z jadier 0 a 1, a tak ďalej.)

Podúloha C (4 body): koľký som v poradí
Každé jadro dostane na vstupe jeden prvok postupnosti: celé číslo od 0 po 10^9. Tieto čísla sú zaručene **všetky navzájom rôzne**.
Napíš program, ktorý dosiahne, že každé jadro zistí, koľký najmenší (číslujúc od nuly) je jeho vstup a dá toto jeho poradové číslo na výstup. Teda jadro s najmenším vstupom má dať na výstup 0, jadro s druhým najmenším 1, atď.

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.[2] 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.

```
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.

[2] 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.

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