Priebeh krajského kola

Krajské kolo 36. ročníka Olympiády v informatike, kategória A, sa koná 19. januára 2021 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 Kozmonauti

Kocúrkovská amatérska vesmírna agentúra KASA sa rozhodla, že pošle do vesmíru dvoch kozmonautov. Medzi obyvateľmi Kocúrkova sa našlo n dobrovoľníkov, ktorí sa chcú tohto letu zúčastniť. Každý dobrovoľník dostal poradové číslo od 1 po n a v agentúre si ho odmerali a odvážili. Dobrovoľník i má výšku h_i a hmotnosť w_i.
Každému je určite jasné, že Kremienok a Chocholúšik ani Ant-Man a Hulk nie sú ideálnou dvojicou kozmonautov. Omnoho lepšie je, ak majú kozmonauti podobnú aj výšku, aj váhu – aby každý vedel v prípade potreby používať výbavu toho druhého.
KASA sa preto rozhodla vybrať takú dvojicu kozmonautov, ktorá bude spĺňať nasledujúce podmienky:

- Rozdiel ich výšok je nanajvýš rovný d_h.
- Rozdiel ich hmotností je nanajvýš rovný d_w.
- Aspoň v jednej z týchto nerovností musí nastať presná rovnosť. (KASA chce zistiť, či je stanovená hranica vhodná a či takto odlišní kozmonauti ešte naozaj dokážu spolu fungovať.)

Súťažná úloha

Navrhnite algoritmus, ktorý spočíta, koľkými rôznymi spôsobmi vieme spomedzi dobrovoľníkov vybrať dvojic kozmonautov.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla n, d_h a d_w. V i-tom z nasledujúcich n riadkov sú čísla h_i a w_i. Všetky údaje na vstupe sú kladné celé čísla.
Na výstup vypíšte jeden riadok a v ňom hľadaný počet dvojíc kompatibilných kozmonautov.

Hodnotenie

Vo všetkých vstupoch platí $2 \leq n$ a $1 \leq d_h, d_w, h_1, \ldots, h_n, w_1, \ldots, w_n \leq M$.

10 bodov: riešenie, ktoré efektívne vyrieši vstupy s $n \leq 10^5$ a $M \leq 10^9$.
6 bodov: riešenie, ktoré efektívne vyrieši vstupy s $n \leq 10^5$ a $M \leq 5000$.
3 body: riešenie, ktoré efektívne vyrieši vstupy s $n \leq 5000$ a $M \leq 10^9$.

Príklady

vstup

```
3 3000 300
1 1
3001 301
1001 301
```

výstup

```
2
```

Vyhovujúce dvojice dobrovoľníkov sú (1,2) a (1,3). V dvojici (2,3) sú oba rozdiely primalé.

vstup

```
5 10 10
100 100
100 110
110 100
110 110
12345 12345
```

výstup

```
6
```

Ľubovoľní dvaja spomedzi prvých štyroch dobrovoľníkov tvoria dobrú posádku. Piaty je smutný, že je veľký a ťažký a nikam nepoletí.

vstup

```
5 5 5
100 100
100 110
110 100
110 110
105 105
```

výstup

```
4
```

Piaty dobrovoľník určite poletí, spoločnosť mu môže robiť ľubovoľný zo zvyšných štyroch.

A-II-2 Laboratórne protokoly

Mišo a Rišo robili laboratórnu prácu z fyziky. Aj jednému aj druhému na konci vyšla nejaká postupnosť n celočíselných výsledkov. Mišove výsledky sú $m_1, \ldots, m_n$, Rišove sú $r_1, \ldots, r_n$.
Ideálne by samozrejme bolo, keby im obom vyšlo to isté. Keďže sa tak ale nestalo, bolo by dobré výsledky kde-tu trošku upraviť, aby sa na seba viac podobali. Ale len trošku, nech to zase nie je podozrivé.

Súťažná úloha

Dané sú obe postupnosti nameraných hodnôt a ďalej číslo k. Postupne nanajvýš k-krát môžeme zmeniť niektorý prvok niektorej postupnosti o 1.
Odchýlku pre konkrétny výsledok budeme merať ako štvorec rozdielu Mišovej a Rišovej výslednej hodnoty. Celková odchýlka bude súčtom týchto odchýlok. Teda ak Mišo a Rišo postupne upravia svoje postupnosti na $m'_1, \ldots, m'_n$ a $r'_1, \ldots, r'_n$, tak celková odchýlka bude $(m'_1 - r'_1)^2 + \cdots + (m'_n - r'_n)^2$.
Navrhnite algoritmus, ktorý určí, ako majú postupnosti upraviť, aby bola na konci celková odchýlka najmenšia možná.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla n a k, v druhom $m_1, \ldots, m_n$, v treťom $r_1, \ldots, r_n$. Všetky m_i aj r_i sú **celé čísla**.
Na výstup vypíšte jeden riadok a v ňom najmenšiu možnú celkovú odchýlku po úpravách postupností.
Pri písaní programu môžete predpokladať, že celočíselné premenné vo vašom jazyku majú dostatočný rozsah na to, aby sa do takejto premennej pohodlne zmestila odchýlka zadaných dvoch postupností.

Hodnotenie

10 bodov: riešenie, ktoré efektívne vyrieši vstupy s $1 \leq n \leq 10^5$ a $0 \leq k \leq 10^{18}$.
7 bodov: riešenie, ktoré efektívne vyrieši vstupy s $1 \leq n \leq 10^5$ a $0 \leq k \leq 10^5$.
4 body: riešenie, ktoré efektívne vyrieši vstupy s $1 \leq n \leq 1000$ a $0 \leq k \leq 1000$.

Nezabudnite, že potrebnou súčasťou riešenia je aj dôkaz správnosti použitého algoritmu.

Príklady

vstup

```
1 8
10
5
```

výstup

```
0
```

Napr. Mišo postupne upraví svoju jedinú hodnotu z 10 na 9, z 9 na 8 a z 8 na 7 a potom Rišo svoju z 5 na 6 a zo 6 na 7. Ešte by sme mohli spraviť tri ďalšie úpravy, ale už zjavne nič lepšie nedosiahneme.

vstup

```
3 5
10 20 30
10 20 20
```

výstup

```
25
```

Napr. vyrobia postupnosti 10 20 30 a 10 20 25.
Súčet odchýlok bude $(10-10)^2 + (20-20)^2 + (30-25)^2 = 0^2 + 0^2 + 5^2 = 25$.

vstup

```
5 2
10 9 10 9 10
9 10 9 10 9
```

výstup

```
3
```

Napr. vyrobia postupnosti 10 10 10 9 10 a 9 10 9 9 9.

A-II-3 Babičky

Bývaš na dedine. V dedine je n domov. Niektoré sú prázdne, niektoré sú obývané rodinami a v niektorých bývajú babičky. Ty sa čoskoro budeš chcieť dostať z prázdneho domu číslo 1 do prázdneho domu n.
Je zima a všade je kopec snehu, pohybovať sa vieš len tunelmi vyhrabanými doň. Tých je m. Tunel číslo i spája domy s číslami x_i a y_i. Všetky tunely sú obojsmerné.
Je po Vianociach. Vždy, keď budeš prechádzať okolo nejakého rodinou obývaného domu, nanútia ti nejaké vianočné koláče, ktoré musíš zjesť. (Ak pôjdeš okolo toho istého domu viackrát, koláče ti nanútia pri každom prechode okolo.)
Rodinami obývaných domov zvládneš navštíviť nanajvýš k za sebou, potom už budeš mať plné brucho. Našťastie sa s tým dá niečo robiť: Vždy, keď prejdeš okolo prázdneho domu, ti vytrávi dostatočne na to, aby si zase zvládol ďalších nanajvýš k obývaných domov za sebou.
Horšie je to u domov, v ktorých bývajú babičky. Keď ťa nejaká babička uvidí pri svojom dome, vyhlási, že trpíš podvýživou, zavolá ťa dnu a napchá ťa tak, že už sa do teba ani hlbší nádych nezmestí. Toto sa už ani rozchodiť nedá, už až do konca cesty nesmieš nič jesť lebo by ťa roztrhlo.
No a úplne na záver ešte pridáme jednu katastrofu. U niektorej babičky (zatiaľ nevieme u ktorej) je na návšteve dedo Jozef. Tomu dlžíš stovku a rozhodne ho nechceš stretnúť.

Súťažná úloha

Tesne pred tým, ako sa vydáš na cestu, si zistíš, u ktorej babičky je dedo Jozef. Hovoríme, že dedina je *priechodná*, ak už teraz vieš zaručiť, že sa budeš vedieť dostať do domu číslo n bez ohľadu na to, čo sa dozvieš o polohe deda Jozefa.
Na vstupe je daný popis tvojej dediny. Napíš algoritmus, ktorý zistí, či je priechodná.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla n, m a k. V druhom riadku sú čísla $v_1, \ldots, v_n$ popisujúce typy domov v dedine: pri $v_i = 0$ je dom i prázdny, $v_i = 1$ je dom s rodinou a $v_i = 2$ dom s babičkou. Domy 1 aj n sú prázdne a existuje aspoň jeden dom s babičkou.
Zvyšok vstupu tvorí m riadkov, každý z nich obsahuje dve čísla domov prepojených tunelom v snehu.

Na výstup vypíšte, či je dedina zadaná na vstupe priechodná.

Príklady

vstup

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

výstup

```
ANO
```

Ak bude dedo Jozef v dome 2, pôjdeme cestou 1-3-4-5. Ak bude v dome 3, pôjdeme 1-2-4-5.

vstup

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

výstup

```
NIE
```

Dedo Jozef bude buď u babičky v dome 3 alebo u babičky v dome 4. Ak nastane tá druhá situácia, nemáme ho ako obísť.

vstup

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

výstup

```
NIE
```

Do domu 5 vedie jediná cesta a pozdĺž nej sú tri rodinné domy za sebou. Tvoj žalúdok však zvláda najvýš $k = 2$ takéto domy za sebou a nie je kde si oddýchnuť.

vstup

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

výstup

```
NIE
```

Ak bude dedo Jozef v dome číslo 2, musíme ho obísť cestou 1-3-5-6. Toto ale tiež nie je správne riešenie, lebo po návšteve babičky už až do konca cesty nesmieš nič jesť. (V dome 3 nás babička napchá na prasknutie. Následne potrebujeme ešte prejsť okolo domu 5, kde by nám nanútili ďalšie koláče a my by sme praskli.)

vstup

```
7 5 2
0 1 1 0 1 2 0
1 2
2 3
3 4
3 5
5 7
```

výstup

```
ANO
```

Priama cesta z domu 1 do domu 7 je 1-2-3-5-7. Túto cestu ale nemôžeš použiť – postupne navštívi až tri rodinné domy za sebou, zatiaľ čo ty zvládneš len dva. Našťastie však existuje riešenie: pôjdeš trasou 1-2-3-4-3-5-7. Od domu 3 si teda „odskočíme“ ku prázdnemu domu 4, tam nám vytrávi a zvládneme nasledovne prejsť popri domoch 3 a 5 do cieľa.

Hodnotenie

10 bodov: riešenie, ktoré efektívne vyrieši vstupy s $2 \leq n \leq 10^5$, $0 \leq m \leq 10^5$ a $1 \leq k \leq 10$.
5 bodov: tie isté obmedzenia, ale navyše predpokladáme, že existuje nanajvýš 10 babičiek
aspoň 2 body: ľubovoľné korektné riešenie

A-II-4 Triedenie v externej pamäti

Táto súťažná úloha nadväzuje na úlohu z domáceho kola. Za samotným zadaním úlohy nájdete študijný text. Ten je totožný so študijným textom zo zadaní domáceho kola.

V oboch podúlohách budeme riešiť tú istú úlohu: Na disku máme uloženú postupnosť položiek $x_1, \ldots, x_n$. Všetky položky majú jednotkovú veľkosť, každý blok disku teda obsahuje presne b položiek. Túto postupnosť chceme usporiadať.
Ľubovoľné dve položky vieme porovnať v konštantnom čase. Môžete predpokladať, že žiadne dve položky nie sú rovnaké a že výsledky porovnania sú tranzitívne (teda z $x < y$ a $y < z$ vieme, že nutne aj $x < z$).
V podúlohách A aj B predpokladajte, že do pamäte, ktorú máte k dispozícii, sa naraz zmestí len **konštantne veľa** blokov – teda m je $O(b)$.

Podúloha A (3 body).
Selection sort je algoritmus, ktorý funguje nasledovne: postupne pre každé k od 1 po n nájdeme najmenší spomedzi prvkov $x_k, \ldots, x_n$ a vymeníme ho s prvkom x_k.
Napíšte rozumnú implementáciu programu, ktorý použije tento algoritmus na usporiadanie postupnosti uloženej na disku. Analyzujte časovú a komunikačnú zložitosť vašej implementácie.
Pripomíname, že do pamäte sa naraz zmestí len konštantný počet blokov.

Podúloha B (5 bodov).
Navrhnite efektívny algoritmus pre triedenie postupnosti uloženej na disku. Snažte sa dosiahnuť čo najmenšiu komunikačnú zložitosť.
Pre jednoduchosť môžete v tejto podúlohe predpokladať, že n je násobkom veľkosti bloku.
Tiež je povolené predpokladať, že za vstupnou postupnosťou sú na disku ďalšie prázdne bloky, ktoré môžete využívať. Existujú riešenia, ktoré si vystačia s miestom na disku, ktoré zaberá vstup, ale aj riešenia, ktoré pre pohodlie použijú ďalších $O(n/b)$ blokov disku, budeme hodnotiť rovnako.

Podúloha C (2 body).
Popíšte, ako zlepšiť vaše riešenie podúlohy B v prípade, že máte k dispozícii viac pamäte. Vyjadrite komunikačnú zložitosť vášho vylepšeného riešenia ako funkciu n, b a m.

Študijný text: Externá pamäť

V tomto ročníku olympiády budeme pracovať s dátami, ktoré sú tak veľké, že sa nám naraz všetky nezmestia do pamäte. Budeme teda písať programy pre počítač, ktorý má internú aj externú pamäť. Tie budeme pre jednoduchosť volať *pamäť* a *disk*. Práca s diskom je omnoho pomalšia ako práca s pamäťou.
K dátam v pamäti môžeme pristupovať priamo, tak ako sme zvyknutí z bežných algoritmov. Pamäť je však obmedzená, naraz sa do nej zmestí len m *údajov*. Každý údaj je nejaké rozumne veľké celé číslo.[1]
Disk je neobmedzene veľký. Je rozdelený na *bloky*. Každý blok obsahuje b údajov. K týmto údajom nevieme pristupovať priamo. Vieme robiť len dva typy operácií: načítať blok z disku na konkrétne miesto do pamäte, a uložiť konkrétny kus pamäte ako blok na disk.
Ak nebude povedané ináč, tak jediné, čo môžete predpokladať o parametroch m a b, je, že pamäť je dostatočne veľká na to, aby sa do nej naraz zmestilo konštantne veľa blokov. Ľubovoľný program, v ktorom použijete $O(b)$ pamäte, je teda v poriadku.

[1] Údaj si môžete pre jednoduchosť predstavovať ako 32-bitovú celočíselnú premennú. Formálne zvykneme dovoliť do týchto premenných ukladať čísla ktorých veľkosť je nanajvýš polynomiálna od veľkosti vstupu – teda napr. číslo n^3 sa ešte do premennej zmestí ale 2^n už nie.

Práca s diskom a iné technické detaily

Konštantu b (veľkosť bloku) poznáme a môžeme ju používať v programoch.
Na čítanie z disku budeme v programoch používať funkciu `Read(blok)`, ktorá dostane poradové číslo bloku, ktorý má načítať, načíta jeho obsah do poľa a toto pole vráti na výstupe.
Na zápis na disk budeme používať funkciu `Write(blok, pole)`, ktorá obsah daného b-prvkového poľa zapíše do daného bloku na disku.

Analýza algoritmov

Pri analýze algoritmov v tomto modeli nás budú zaujímať dva hlavné druhy zložitosti. Prvým je klasická *časová zložitosť* – počet krokov výpočtu. Druhým je *komunikačná zložitosť* – počet volaní funkcií `Read` a `Write`.
Budeme sa snažiť hľadať algoritmy, ktoré budú mať (asympoticky) rovnakú časovú zložitosť, ako by mal optimálny algoritmus na klasickom počítači (t.j. taký, ktorý má k dispozícii neobmedzené množstvo pamäte). Samozrejme, ak máme dva takéto algoritmy, lepší z nich je ten, ktorý má (asympoticky) menšiu komunikačnú zložitosť.

Aby sme sa nemuseli zaoberať malými patologickými prípadmi, stačí sa pri analýze algoritmov zaoberať situáciou, kde veľkosť vstupu je rádovo väčšia ako veľkosť jedného bloku.
Teda ak napríklad spravíme nanajvýš $n/b+3$ volaní funkcie `Read`, môžeme $+3$ prehlásiť za zanedbateľne malé v porovnaní s n/b a konštatovať, že komunikačná zložitosť je $O(n/b)$.

Zvlášť vás ešte chceme upozorniť na programy používajúce rekurziu. Nezabudnite, že každé volanie funkcie potrebuje záznam na zásobníku, a ten sa tiež nachádza v pamäti. Pri rekurzívnych algoritmoch treba teda aj hĺbku rekurzie rátať do pamäťovej zložitosti a hlavne si treba dať pozor na to, aby ste samotnou rekurziou neminuli priveľa pamäte.

Príklad: maximum

Prácu s diskom si vyskúšame na jednom z najjednoduchších možných príkladov. V pamäti máme číslo n. Na začiatku disku máme postupnosť n kladných celých čísel. Našou úlohou je nájsť najväčšie z čísel na vstupe.
Pre názornosť explicitne pripomíname, že postupnosť na disku je rozdelená do blokov: v prvom bloku je prvých b čísel, v druhom ďalších b, a tak ďalej. Dokopy teda vstup zaberá $\lceil n/b \rceil$ blokov na disku.
Riešenie bude priamočiare: postupne po jednom prejdeme všetky bloky. Každý blok načítame do pamäte a prejdeme všetky údaje v ňom. Následne ho už nebudeme potrebovať, takže pamäť, kde bol uložený, jednoducho prepíšeme nasledujúcim blokom.
Nižšie uvádzame ukážkovú implementáciu tohto riešenia v jazyku Python. (Vo svojich riešeniach môžete použiť ľubovoľný bežný programovací jazyk, detaily použitia funkcií `Read` a `Write` si vhodne zvoľte.)

```
# v pamäti už máme:
#     - premennú N obsahujúcu počet údajov na vstupe
#     - konštantu B označujúcu veľkosť bloku na disku

s = 0          # počet už spracovaných blokov zo vstupu
odpoved = 0    # doterajšie maximum

while s*B < N:                # kým sme ešte nespracovali celý vstup
    data = Read(s)            # načítaj ďalší blok z disku
    pocet = min( B, N - s*B ) # zisti, koľko údajov z neho ešte patrí do vstupu
    for i in range(pocet):    # prejdi tie údaje a porovnaj s doterajším maximom
        odpoved = max( odpoved, data[i] )
    s += 1                    # posuň sa na nasledujúci blok

print(odpoved)
```

O tomto programe vieme povedať nasledovné:

- Jeho časová zložitosť je $O(n)$, teda lineárna od veľkosti vstupu. Na každý prvok vstupu sa raz pozrieme.
- Jeho komunikačná zložitosť je presne $\lceil n/b \rceil$.
- Naraz máme v pamäti len jeden blok a konštantne veľa pomocných premenných, pamäťovú zložitosť teda máme naozaj len $O(b)$.

Príklad: test symetrie

V pamäti máme číslo n. Na začiatku disku máme postupnosť n kladných celých čísel. Našou úlohou je zistiť, či ide o palindróm – teda či táto postupnosť čítaná odzadu vyzerá rovnako ako odpredu.
Ak by n bolo násobkom b, bola by implementácia riešenia celkom príjemná: stačilo by porovnať prvý blok s posledným, potom druhý s predposledným, a tak ďalej.
So všeobecným prípadom bude o čosi väčšia oštara. Nech teda n dáva po delení b nejaký nenulový zvyšok z. Potom prvý blok chceme porovnať sčasti s poslednými z údajmi (na začiatku posledného bloku) a sčasti s predchádzajúcimi $b - z$ údajmi (na konci predposledného bloku vstupu).
Predstavme si, že máme dvoch pracovníkov. Prvý číta vstup zľava doprava, druhý zároveň rovnakým tempom sprava doľava. Vždy, keď obaja prečítajú číslo, porovnajú si ich. Takto implementujeme aj naše riešenie, s tým, že každý pracovník si vždy, keď sa mu minú čísla v pamäti, vyžiada príslušný ďalší blok z disku. Prestaneme, keď sa obaja pracovníci stretnú v strede postupnosti.
Takéto riešenie bude zjavne mať časovú zložitosť $O(n)$, pamäťovú zložitosť $O(b)$ a komunikačnú zložitosť $O(n/b)$.

Príklad: reverz postupnosti

Na záver študijného textu sa pozrieme na príklad podobný tomu predchádzajúcemu. Ako by to vyzeralo, keby sme chceli postupnosť, zadanú na vstupe, v pamäti obrátiť? Zjavne stačí robiť to isté ako v predchádzajúcom riešení, s dvoma úpravami. Namiesto porovnania si pracovníci svoje čísla vymenia. A vždy pred tým, ako by načítali ďalší blok z disku, tak najskôr do toho minulého zapíšu čísla, ktoré dostali od kolegu.
Ešte si všimnime, že musíme byť opatrní pri spracovaní stredu postupnosti, aby sme zapísali správne vyzerajúci prostredný blok.
Nižšie uvádzame jednu možnú implementáciu jednoduchšej verzie tejto úlohy. V tejto implementácii predpokladáme, že n je násobkom veľkosti bloku, a teda len reverzujeme a medzi sebou vymieňame celé bloky.

```
assert N % B == 0
pocet_blokov = N // B   # N celočíselne vydelené B

# kým máme aspoň dva bloky, vymeníme a reverzneme prvý a posledný z nich
for i in range(pocet_blokov // 2):
    lavy = Read(i)
    pravy = Read(pocet_blokov-1-i)
    Write( i, reversed(pravy) )
    Write( pocet_blokov-1-i, reversed(lavy) )

# ak nám v strede zvýšil jeden blok, ten len reverzneme
if pocet_blokov % 2 == 1:
    stredny = Read( pocet_blokov // 2)
    Write( pocet_blokov // 2, reversed(stredny) )
```

TRIDSIATY ŠIESTY ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2021