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 niekedy v septembri.

Riešenia kategórie A je potrebné odovzdať najneskôr **15. novembra 2020**.
Riešenia kategórie B je potrebné odovzdať najneskôr **30. novembra 2020**.

Priebeh súťaže

Za každú úlohu domáceho kola sa dá získať od 0 do 10 bodov. Na základe bodov domáceho kola stanoví Slovenská komisia OI (SK OI) pre každú kategóriu bodovú hranicu potrebnú na postup do **krajského kola**. Očakávame, že táto hranica bude približne rovná **tretine maximálneho počtu bodov**.

V krajskom kole riešitelia riešia štyri teoretické úlohy, ktoré môžu tematicky nadväzovať na úlohy domáceho kola. V kategórii B súťaž týmto kolom končí.

V kategórii A je približne najlepších 30 riešiteľov krajského kola (podľa počtu bodov, bez ohľadu na kraj, v ktorom súťažili) pozvaných do **celoštátneho kola**. V celoštátnom kole účastníci prvý deň riešia teoretické a druhý deň praktické úlohy. Najlepší riešitelia sú vyhlásení za víťazov. Približne desať najlepších riešiteľov následne SK OI pozve na týždňové výberové sústredenie. Podľa jeho výsledkov SK OI vyberie družstvá pre Medzinárodnú olympiádu v informatike (IOI) a Stredoeurópsku olympiádu v informatike (CEOI).

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++, Pascal alebo Java). 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.

Presný popis, ako majú vyzerať riešenia praktických úloh (napr. realizáciu vstupu a výstupu), nájdete na webstránke, kde ich budete odovzdávať.

Ak nie je v zadaní povedané ináč, riešenia teoretických úloh musia v prvom rade obsahovať **podrobný slovný popis použitého algoritmu, zdôvodnenie jeho správnosti** a diskusiu o efektivite zvoleného riešenia (t. j. posúdenie časových a pamäťových nárokov programu). Na záver riešenia uveďte program. Ak používate v programe netriviálne algoritmy alebo dátové štruktúry (napr. rôzne súčasti STL v C++), súčasťou popisu algoritmu musí byť dostatočný 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 IUVENTA v tesnej súčinnosti so Slovenskou komisiou OI.

Covid-19

Krajské kolo súťaže je plánované na 19. januára 2021 a celoštátne kolo na 24.-27. marca 2021. V súčasnosti ešte nevieme zaručiť, či tieto kolá prebehnú tradičnou prezenčnou formou.

Slovenská komisia OI dôsledne sleduje aktuálny vývoj epidemiologickej situácie a v prípade potreby mu prispôsobí organizáciu súťaže. Domáce kolo súťaže určite prebehne v tradičnej podobe, keďže naň v súčasnosti platné opatrenia nemajú žiaden vplyv. O presnej forme a spôsobe organizácie ďalších kôl súťaže budú postupujúci súťažiaci informovaní priebežne.

A-I-1 Zatopené mesto

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

Kocúrkovom sa prehnala búrka a časť mesta zatopila. Kubík sa teraz potrebuje dostať zo školy domov. Plávanie zatopenými časťami mesta mu nerobí problém, ale nechce prechladnúť. Presnejšie, nesmie byť príliš dlho mokrý vonku – keď už sa raz namočí, potrebuje sa ponáhľať domov.

Kocúrkovo pozostáva z n križovatiek (očíslovaných 0 až $n-1$) a m obojsmerných ulíc. Každá ulica spája dve križovatky. Prejdenie ľubovoľnej ulice trvá 1 minútu.

Kubík sa potrebuje dostať z križovatky 0 na križovatku $n-1$. Niektoré križovatky (možno aj 0 a/lebo $n-1$) sú zatopené vodou.

Ďalej je dané číslo k, hovoriace, že akonáhle sa Kubík prvýkrát namočí, musí sa dostať domov za nanajvýš k minút. (Napr. ak $k = 1$, tak na Kubíkovej ceste domov môžu byť zatopené nanajvýš posledné dve križovatky, pričom poslednou je určite križovatka číslo $n-1$.)

Zistite, či sa Kubík vie dostať domov, a ak áno, ako najrýchlejšie to vie spraviť.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla n, m a k.

Druhý riadok obsahuje čísla $z_0, z_1, \ldots, z_{n-1}$, pričom $z_i = 0$ ak je križovatka i suchá a $z_i = 1$ ak je zatopená.

Zvyšok vstupu tvorí m riadkov, v každom sú čísla dvoch križovatiek spojených ulicou. Každá ulica spája dve rôzne križovatky. Každá dvojica križovatiek je spojená nanajvýš jednou ulicou.

Vypíšte jeden riadok a v ňom jedno celé číslo: ak sa Kubík vie požadovaným spôsobom dostať domov, vypíšte najmenší počet minút, za ktorý to ide, inak vypíšte -1.

Obmedzenia a hodnotenie

Vo všetkých vstupoch platí $n \geq 2$, $m \geq 0$ a $1 \leq k \leq n$.

Odovzdané riešenie bude otestované na šiestich sadách vstupov. Body a maximálne hodnoty n, m, k pre ne udávame v tabuľke.

sada	body	n	m	k	ďalšie obmedzenia
#1	2	10	45	10	
#2	1	10^5	10^6	1	
#3	1	10^5	10^6	10^5	$m \cdot k \leq 10^6$
#4	1	10^5	10^6	10^5	nanajvýš 100 zatopených križovatiek
#5	1	10^5	10^6	10^5	$m \cdot k \leq 10^6$ a nanajvýš 100 zatopených križovatiek
#6	4	10^5	10^6	10^5	

Príklady

vstup

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

výstup

```
-1
```

Zo školy domov vedie Kubíkovi len jedna cesta. Na križovatke 1 sa namočí a následne sa nevie dostatočne rýchlo dostať domov ($k = 2$, ale mokrému Kubíkovi to bude trvať 3 minúty).

vstup

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

výstup

```
4
```

Tento raz je Kubíkova cesta domov v poriadku, mokrý bude len minútu. Celkový čas cesty sú 4 minúty.

vstup

```
7 7 1
0 1 0 0 1 0 0
0 1
1 2
2 6
0 3
3 5
4 5
4 6
```

výstup

```
4
```

Zo školy domov vedú dve cesty. Cesta 0-1-2-6 je kratšia, ale nevyhovuje – Kubík bude pridlho mokrý vonku. Musí preto ísť dlhšou cestou: 0-3-5-4-6.

A-I-2 Román

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

Roman práve dopísal svoj prvý román. Keď vydavateľ uvidel rukopis, zalomil rukami a povedal, že je toho tak veľa, že to budú musieť rozdeliť na viac kníh a vydať postupne. Tu ale vyvstala závažná otázka: ako presne Romanov román rozdeliť?
Román má n kapitol. Každá z nich je veselá, smutná, alebo neutrálna. Vydavateľ si myslí, že je najlepšie, keď je kniha vyvážená – teda keď veselých kapitol obsahuje rovnako veľa ako smutných.
Vašou úlohou je rozdeliť celý Romanov román na knihy tak, aby čo najviac z nich bolo vyvážených.

Formát vstupu a výstupu

V prvom riadku vstupu je číslo n. V druhom riadku vstupu je postupnosť n čísel popisujúca jednotlivé kapitoly: 1 je veselá, 0 je neutrálna a -1 je smutná kapitola. Kapitoly sú dané v správnom poradí a toto poradie nesmiete meniť.

Na výstup vypíšte jediný riadok a v ňom jedno číslo: najväčší dosiahnuteľný počet vyvážených kníh.

Obmedzenia a hodnotenie

Odovzdané riešenie otestujeme na štyroch sadách vstupov. Body a maximálne hodnoty n pre ne sú nasledovné:

sada	body	n	ďalšie obmedzenia
#1	1	2	
#2	2	500	žiadna kapitola nie je smutná
#3	3	500	
#4	4	10^6	

Príklady

vstup

```
4
-1 -1 1 1
```

výstup

```
1
```

Nevieme vyrobiť viac ako jednu vyváženú knihu. Sú dva rôzne optimálne spôsoby delenia: buď dáme všetky kapitoly do jednej knihy, alebo román rozdelíme takto: `-1 | -1 1 | 1`.

vstup

```
10
0 1 1 0 -1 -1 -1 -1 1 -1
```

výstup

```
3
```

Jedno optimálne delenie na knihy: `0 | 1 1 0 -1 -1 | -1 -1 | 1 -1`. *Tretia kniha nie je vyvážená ale zvyšné tri sú. Viac ako tri vyvážené knihy sa nedá dosiahnuť.*

A-I-3 Celta

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

Na trhovisku stojí n stánkov v jednom rade. Ich priečelia sú rôzne široké: i-ty stánok zľava má a_i centimetrov. Máme celtu obdĺžnikového tvaru, ktorou by sme chceli stánky zakryť pred dažďom. Celta má práve takú šírku, aby dosiahla od prednej hrany stánkov po zadnú. Dĺžka celty je k centimetrov. Celtu nie je dovolené nijak deliť.

Súťažná úloha

Zadané sú čísla $a_1, \ldots, a_n$ a číslo k. Chceme úplne zakryť čo najviac stánkov. Navrhnite čo najefektívnejší algoritmus, ktorý nájde ľubovoľné jedno optimálne riešenie.

Formát vstupu a výstupu

V prvom riadku vstupu sú kladné celé čísla n a k. V druhom riadku vstupu sú medzerou oddelené kladné celé čísla $a_1, \ldots, a_n$. Môžete predpokladať, že ich súčet sa zmestí do bežnej celočíselnej premennej.

Vypíšte jeden riadok a v ňom dve čísla. Ak sa nedá zakryť žiadny stánok, vypíšte „0 0“. Ak je optimálne zakryť d stánkov začínajúc ℓ-tým zľava, vypíšte „d ℓ“.

Obmedzenia a hodnotenie

Plných 10 bodov môžu získať len riešenia s optimálnou časovou zložitosťou.
(Nezabudnite na zdôvodnenie správnosti a odhad časovej zložitosti.)

Za riešenia, ktoré efektívne vyriešia ľubovoľný vstup s $n \leq 10^6$, môžete získať do 7 bodov.
Za riešenia, ktoré efektívne vyriešia ľubovoľný vstup s $n \leq 5000$, môžete získať nanajvýš 4 body.
Ľubovoľné funkčné riešenie, bez ohľadu na rýchlosť, môže dostať aspoň 2 body.

Príklady

vstup

```
7 600
200 100 200 300 200 200 100
```

výstup

```
3 2
```

Stánky číslo 2, 3 a 4 majú dokopy šírku $100 + 200 + 300 = 600$ *cm, máme teda presne dosť celty na ich zakrytie. Iné tiež správne výstupy pre tento vstup sú „3 1“ a „3 5“. V oboch týchto prípadoch ide o trojicu susedných stánkov, ktoré majú dokopy len 500 cm.*

vstup

```
3 10
100 100 100
```

výstup

```
0 0
```

A-I-4 Externá pamäť

Toto je teoretická úloha. Pomocou webového rozhrania odovzdajte súbor vo formáte PDF, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách. K tejto úlohe patrí študijný text uvedený na nasledujúcich stranách. Odporúčame najskôr prečítať ten a až potom sa vrátiť k samotným súťažným úlohám.

V pamäti máme číslo n. Na začiatku disku máme uloženú maticu rozmerov $n \times n$. Matica je uložená po riadkoch: teda najskôr sú uložené prvky z jej prvého riadku (zľava doprava), potom prvky z druhého riadku (opäť zľava doprava), a tak ďalej. Pre jednoduchosť môžete predpokladať, že n je násobkom b (veľkosti bloku).

Podúloha A (2 body). Rozcvička: vypíšte prvky tejto matice po riadkoch.

Podúloha B (3 body). Vypíšte prvky tejto matice po stĺpcoch – teda najskôr jej prvý stĺpec zhora dole, potom druhý stĺpec zhora dole, a tak ďalej.

Podúloha C (5 bodov). Otočte celú maticu o 90 stupňov doprava. Teda napr. jej ľavý horný prvok má skončiť v pravom hornom rohu, pravý horný prvok má skončiť vpravo dole, a tak ďalej. Na konci musí byť otočená matica uložená na disku tam, kde je teraz uložená pôvodná matica.
Ak vám to pri písaní programu pomôže, môžete v tejto podúlohe predpokladať, že do pamäte sa vám zmestí až rádovo b blokov – teda že môžete použiť až $O(b^2)$ pamäte.

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.

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.

[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.

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 2019