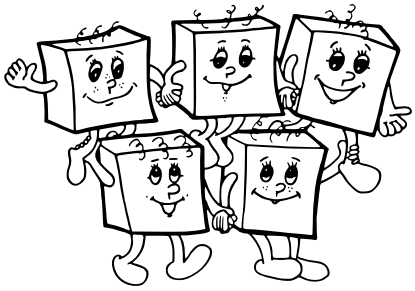
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH

http://oi.sk/



tridsiaty šiesty ročník
školský rok 2020/2021

zadania celoštátneho kola, deň 1
kategória A

Priebeh celoštátneho kola

Celoštátne kolo 36. ročníka Olympiády v informatike, kategórie A, sa koná v dňoch 25.-26. marca 2021. Na riešenie úloh prvého, teoretického dňa majú súťažiaci 4,5 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.
- 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 prvého (teoretického) dňa

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 úlohy môžu byť uvedené limity na veľkosť premenných. Tieto môžete použiť na odhad toho, ako dobré vaše riešenie je. Na počítači, ktorý vykoná miliardu inštrukcií za sekundu, vyrieši vzorové riešenie ľubovoľný povolený vstup nanajvýš za niekoľko sekúnd.

A-III-1 Dobíjacie stanice

V Absurdistane je n osád. Sú očíslované od 1 po n. Medzi niektorými dvojicami osád vedú cesty. Ciest je presne $n-1$ a sú postavené tak, aby sa po celom Absurdistane dalo po nich cestovať – majú teda stromovú topológiu. Všetky cesty majú približne rovnakú dĺžku.
Šejk Milko si kúpil elektromobil. Ten vie na jedno nabitie prejsť nanajvýš po k cestách. V niektorých osadách sú dobíjacie stanice. V hociktorej takej osade si Milko môže svoje auto dobiť a potom pokračovať v ceste.

Navrhnite čo najefektívnejší algoritmus, ktorý spočíta, koľko existuje usporiadaných dvojíc (x, y) osád takých, že $x \neq y$ a ak Milko začína v osade x s plne nabitým autom, vie sa dostať do osady y. (Táto cesta môže zahŕňať ľubovoľne veľa zastávok na dobitie auta. Začiatočná osada x nemusí mať dobíjaciu stanicu.)

Formát vstupu a výstupu

V prvom riadku vstupu sú celé čísla n, k a d. V druhom riadku je d rôznych celých čísel: čísla osád, v ktorých sú dobíjacie stanice. Zvyšok vstupu tvorí zoznam ciest – pre každú sú uvedené čísla osád, ktoré spája.
Na výstup vypíšte jedno číslo: hľadaný počet dvojíc, odkiaľ kam sa dá dostať elektromobilom.

Príklady

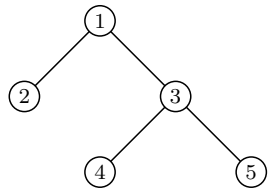
vstup

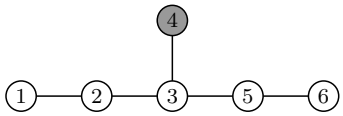
```
5 2 0

1 2
1 3
3 4
3 5
```

výstup

```
16
```

Obrázok vľavo. Nemáme žiadne dobíjacie stanice. Dvojice $(2,4)$, $(2,5)$, $(4,2)$ a $(5,2)$ sú zlé: napr. ak Milko začne v osade 4 a chce sa dostať do osady 2, musel by postupne prejsť po troch cestách (4–3–1–2), aj plne nabité auto však zvládne len dve cesty.





vstup

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

výstup

```
30
```

Obrázok vpravo. V tejto krajine sa dá elektromobilom cestovať medzi ľubovoľnými dvomi mestami. Občas (napr. pri ceste z 1 do 6) to vyžaduje zastávku v osade 4 na dobitie auta.

Obmedzenia a hodnotenie

Môžete predpokladať, že $n \geq 2$, $1 \leq k \leq n-1$ a $0 \leq d \leq n$. Môžete tiež predpokladať, že k je malé: $k \leq 100$.

Na plný počet bodov stačí nájsť riešenie dostatočne efektívne pre $n \leq 300\,000$. (Existujú aj ešte lepšie riešenia, ktoré nepotrebujú predpoklad, že k je malé – za takéto riešenie však už nedostanete žiadne body navyše.)
Za riešenie efektívne pre $n \leq 10\,000$ bude nanajvýš 6 bodov, za riešenie efektívne pre $n \leq 1000$ nanajvýš 4 body.
Za efektívne riešenie pre veľké n, ktoré ale funguje len za predpokladu $d = 0$ (žiadne dobíjacie stanice), viete získať nanajvýš 3 body.

A-III-2 Základne na Marse

Navzdory všetkým očakávaniam bola misia na Mars úspešná: kozmonauti kocúrkovskej amatérskej vesmírnej agentúry KASA spravili ďalší veľký krok pre ľudstvo.
Teraz už majú na Marse postavených v vysielačov a prišla doba, kedy treba niekde postaviť základňu. Aby mala základňa dobré spojenie so Zemou, musíme ju postaviť vo vnútri nejakého trojuholníka, ktorý má vo vrcholoch vysielače, a následne všetky tri tieto vysielače prepojiť so základňou.
Na vstupe dostanete $v + z$ bodov v rovine: najskôr súradnice všetkých vysielačov a následne z možných polôh základne. Pre každú polohu základne buď nájdite ľubovoľné tri vysielače také, že základňa leží v nimi určenom trojuholníku, alebo podajte správu, ak taká trojica vysielačov neexistuje.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla v a z. Zvyšok vstupu tvoria súradnice jednotlivých bodov (najskôr vysielače, potom potenciálne základne).
Môžete predpokladať, že zadané body sú vo všeobecnej polohe: žiadne dva nie sú totožné a žiadne tri neležia na priamke.

Pre každú potenciálnu základňu vypíšte buď trojicu čísel z množiny $\{1, \ldots, v\}$ (poradové čísla vysielačov, ktoré tvoria jeden možný hľadaný trojuholník) alebo správu „neexistuje“.

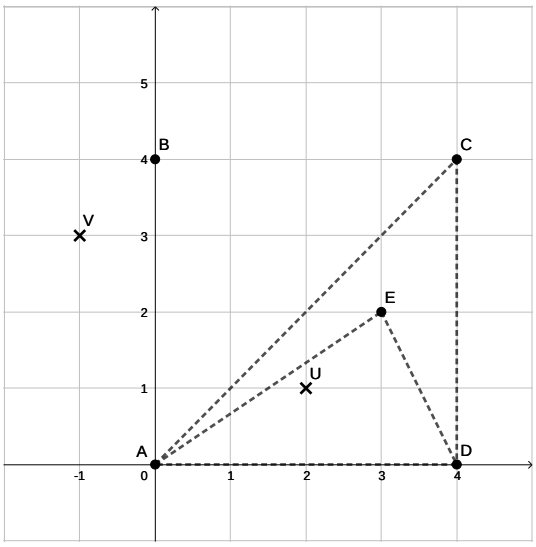
Príklady

vstup

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

output

```
1 5 4
neexistuje
```



Bodky na obrázku predstavujú vysielače, krížiky sú potenciálne základne. Základňa U leží napr. v trojuholníku ADE (čo sú základne číslo 1, 4 a 5) alebo v trojuholníku ACD (1, 3, 4). Základňa V neleží vo vnútri žiadneho trojuholníka z vysielačov.

Obmedzenia a hodnotenie

Môžete predpokladať, že $v \geq 3$ a že všetky súradnice sú celé čísla v absolútnej hodnote neprevyšujúce 10^9.
Plných 10 bodov môžete dostať za riešenie efektívne zvládajúce vstupy s $v, z \leq 100\,000$.
Nanajvýš 7 bodov môžete dostať za riešenie efektívne zvládajúce vstupy s $v \leq 100\,000$ a $z \leq 50$.
Tiež nanajvýš 7 bodov môžete dostať za riešenie efektívne zvládajúce vstupy s $v \leq 5\,000$ a $z \leq 100\,000$.
Nanajvýš 5 bodov môžete dostať za riešenie efektívne zvládajúce vstupy s $v \leq 5000$ a $z \leq 10$.
Nanajvýš 2 body môžete dostať za riešenie efektívne zvládajúce vstupy s $v \leq 100$ a $z \leq 10$.

A-III-3 Hľadanie v externej pamäti

Za zadaním úlohy nájdete študijný text. Je totožný so študijným textom zo zadaní domáceho a krajského kola.

V tejto úlohe budeme vyhľadávať v množine prvkov, ktorá je uložená na disku. Na začiatku dostanete množinu M prvkov. Tú si môžete predspracovať a uložiť na disk. Následne bude prichádzať veľa otázok. Každá otázka sa bude pýtať, či sa nejaký prvok y nachádza v našej množine M.

Presnešie, na začiatku dostanete množinu $M = \{x_1, \ldots, x_n\}$ prvkov nejakého konkrétneho ale neznámeho typu. Prvky tohto typu vieme medzi sebou porovnávať: pre každé dva prvky vieme v konštantnom čase zistiť, ktorý je menší, resp. že sú rovnaké. Porovnávanie je tranzitívne: ak $x < y$ a $y < z$, tak aj $x < z$.
O prvkoch množiny M nepredpokladajte nič ďalšie. Špeciálne upozorňujeme, že nemusí ísť o čísla, a teda nevieme pracovať s ich hodnotami.
Každý prvok zaberá konštantne veľa miesta, takže do jedného bloku disku sa ich zmestí b.

Podúloha A (3 body).
Základné riešenie nášho problému vyzerá tak, že si množinu M na disk uložíme v usporiadanom poradí a potom v nej budeme binárne vyhľadávať. Binárne vyhľadávanie má časovú zložitosť $O(\log n)$. Akú najlepšiu komunikačnú zložitosť vieme dosiahnuť vhodnou prácou s diskom počas binárneho vyhľadávania, ako a prečo?

Podúloha B (5 bodov).
Nájdite lepšie riešenie. Je nutné **zachovať asymptotickú časovú zložitosť** binárneho vyhľadávania, ale zároveň chcete dosiahnuť **čo najlepšiu komunikačnú zložitosť** hľadania.
Predspracovanie množiny M môže mať ľubovoľnú polynomiálnu časovú aj komunikačnú zložitosť v závislosti od veľkosti M. (Teda môžete s množinou M spraviť čokoľvek, čo vieme spraviť rozumne efektívne.) Detaily predspracovania netreba rozpisovať, stačí nám, keď popíšete výsledné uloženie dát na disku a následne algoritmus vyhľadávania v nich. Zdôvodnite, že váš algoritmus má požadovanú časovú zložitosť a odhadnite jeho komunikačnú zložitosť.

Podúloha C (2 body).
Dokážte, že žiaden iný deterministický algoritmus nemôže mať rádovo lepšiu komunikačnú zložitosť ako vaše riešenie podúlohy B. (Ak ste v podúlohe B nenašli optimálny algoritmus, v tejto podúlohe ešte stále môžete získať body za dôkaz dolného odhadu optimálnej komunikačnej zložitosti.)

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.

[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.

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.

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 pamati uz mame:
#     - premennu N obsahujucu pocet udajov na vstupe
#     - konstantu B oznacujucu velkost bloku na disku

s = 0          # pocet uz spracovanych blokov zo vstupu
odpoved = 0    # doterajsie maximum

while s*B < N:                # kym sme este nespracovali cely vstup
    data = Read(s)            #nacitaj dalsi blok z disku
    pocet = min( B, N - s*B ) # zisti, kolko udajov z neho este patri do vstupu
    for i in range(pocet):    # prejdi tie udaje a porovnaj s doterajsim maximom
        odpoved = max( odpoved, data[i] )
    s += 1                    # posun sa na nasledujuci 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