70. ročník Matematické olympiády – 2020/2021

Úlohy školního kola kategorie P

Úlohy P-I-1 a P-I-2 jsou praktické, vaším úkolem v nich je vytvořit a odladit efektivní program v jazyce Pascal, C nebo C++. Řešení těchto dvou úloh odevzdávejte ve formě zdrojového kódu přes webové rozhraní přístupné na stránce *https://mo.mff.cuni.cz/submit/,* kde také naleznete další informace. Odevzdaná řešení budou automaticky vyhodnocena pomocí připravených vstupních dat a výsledky vyhodnocení se dozvíte krátce po odevzdání. Pokud váš program nezíská plný počet 10 bodů, můžete své řešení opravit a znovu odevzdat.

Úlohy P-I-3 a P-I-4 jsou teoretické, za každou z nich lze získat až 10 bodů. Řešení musí obsahovat popis algoritmu, zdůvodnění jeho správnosti a odhad časové a paměťové složitosti, v úloze P-I-4 též komunikační složitost. Nemusíte psát program, algoritmus stačí zapsat ve vhodném pseudokódu nebo dokonce jenom slovně, ale v tom případě dostatečně podrobně a srozumitelně. Hodnotí se nejen správnost, ale také efektivita zvoleného postupu řešení. Řešení obou teoretických úloh odevzdávejte ve formě souboru typu PDF přes výše uvedené webové rozhraní.

Řešení všech úloh můžete odevzdávat do 15. listopadu 2020. Opravená řešení a seznam postupujících do krajského kola najdete na webových stránkách olympiády na adrese *https://mo.mff.cuni.cz/,* kde jsou také k dispozici další informace o kategorii P.

Další programovací jazyky: V letošním ročníku MO-P je praktické úlohy možné odevzdávat i v jazycích Python 3 a Java 11. Nezaručujeme však, že v těchto jazycích bude možné získat plný počet bodů – je možné, že program nestihne doběhnout do časového limitu, byť používá algoritmus s optimální časovou složitostí. Také neslibujeme, že tyto jazyky budou k dispozici v dalších kolech soutěže.

P-I-1 Přívalový déšť

Kocourkovem se přehnal přívalový déšť, který Vašíka dočasně uvěznil ve škole. Teď by Vašík rád vyrazil domů, ale potřebuje k tomu vaši pomoc, některé křižovatky jsou totiž zatopené. To samo o sobě pro závodního plavce Vašíka není nepřekonatelný problém, jenže jakmile se Vašík namočí, musí se rychle dostat domů do tepla, jinak dostane rýmu. Pomůžete mu najít bezpečnou cestu domů?

Soutěžní úloha

Kocourkov sestává z n křižovatek označených čísly $0, \ldots, n-1$, z nichž mezi některými vede celkem m obousměrných ulic (tj. také si ho můžete představit jako neorientovaný graf na n vrcholech s m hranami). Škola se nachází na křižovatce číslo 0, Vašík bydlí na křižovatce číslo $n-1$ a některé z křižovatek jsou zatopené (zatopené mohou být i křižovatky 0 a $n-1$). Vaším úkolem je najít délku (počet ulic) nejkratší cesty z křižovatky 0 na křižovatku $n-1$ takové, která poté, co navštíví nějakou zatopenou křižovatku, použije nejvýše k dalších ulic (například, pokud $k = 1$, tak na takové cestě mohou být zatopené jen poslední dvě křižovatky, přičemž poslední je $n-1$), nebo vypsat, že taková neexistuje.

Formát vstupu

Na prvním řádku dostanete mezerami oddělená tři přirozená čísla n, m a k. Na druhém řádku dostanete n mezerou oddělených čísel $z_0, z_1, \ldots, z_{n-1}$, kde pro každé i platí $z_i \in \{0, 1\}$, a $z_i = 1$ právě když je křižovatka číslo i zatopená. Na každém z dalších m řádků dostanete mezerou oddělená dvě různá přirozená čísla a_i a b_i, kde $0 \leq a_i, b_i \leq n-1$, která značí, že mezi křižovatkami číslo a_i a b_i vede obousměrná ulice.

Ve všech vstupech bude platit, že $n \geq 2$, $m \geq 0$, $1 \leq k \leq n$, žádná ulice nespojuje nějakou křižovatku samu se sebou (tj. $a_i \neq b_i$) a žádnou ulici nezadáme vícekrát (tj. pokud $i \neq j$, potom $\{a_i, b_i\} \neq \{a_j, b_j\}$). Další omezení jsou popsána v sekci Bodování.

Formát výstupu

Na výstup vypište počet ulic nejkratší cesty z křižovatky 0 na křižovatku $n-1$, která splňuje podmínky, anebo -1, pokud taková neexistuje.

Příklady

Vstup:	*Výstup:*
5 4 2 0 1 0 0 0 0 1 1 2 2 3 3 4	-1

Ze školy domů vede pouze cesta délky 4, jejíž druhá křižovatka je zatopená. To znamená, že se s $k = 2$ Vašík domů nedostane.

Vstup:	*Výstup:*
5 4 1 0 0 0 1 1 0 1 1 2 2 3 3 4	4

Stejné město jako dříve, ale zatopené jsou jiné křižovatky a $k = 1$. V tomto případě Vašík zvládne dojít domů.

Bodování

Následující tabulka popisuje, kolik bodů dostanete za jednotlivé testovací sady a jaké podmínky v nich platí (hodnoty n, m a k jsou horní meze):

body	n	m	k	*další*
2	10	45	10	
1	10^5	10^6	1	
1	10^5	10^6	10^5	$mk \leq 10^6$
1	10^5	10^6	10^5	nejvýše 100 zatopených křižovatek
1	10^5	10^6	10^5	$mk \leq 10^6$ a nejvýše 100 zatopených křižovatek
4	10^5	10^6	10^5	

P-I-2 Román

Filip se na stará kolena rozhodl dát na spisovatelství a zrovna dokončuje svůj první román. Teď přemýšlí, jak své veledílo rozdělit do kapitol. Z internetového tutoriálu o psaní románů se Filip dozvěděl, že správná kapitola by měla mít dobrou rovnováhu veselých a smutných částí – nepíše přeci humoresku ani žalozpěv.

Filipův román má n odstavců, každý z nich je veselý, smutný, nebo neutrální. Filip by chtěl odstavce rozdělit do kapitol tak, aby co nejvíc kapitol bylo *vyvážených:* Řekneme, že kapitola je vyvážená, pokud obsahuje stejně veselých jako smutných odstavců. Pomozte Filipovi dokončit jeho bestseller.

Soutěžní úloha

Je zadána posloupnost n celých čísel $a_1, \ldots, a_n$, kde a_i popisuje i-tý odstavec:

- pokud $a_i = 1$, jedná se o veselý odstavec,
- pokud $a_i = -1$, jedná se o smutný odstavec,
- jinak $a_i = 0$ a jedná se o neutrální odstavec.

Vaším cílem je najít rozdělení posloupnosti takové, že má co nejvíc vyvážených kapitol, tj. že co nejvíc částí obsahuje stejný počet hodnot -1 a 1. Vypište, kolik lze při nejlepším možném rozdělení vytvořit vyvážených kapitol. Pořadí odstavců nesmíte měnit.

Formát vstupu

První řádek obsahuje číslo $n \geq 2$. Druhý řádek obsahuje n mezerou oddělených čísel $a_1, \ldots, a_n$, kde každé a_i je -1, 0, nebo 1.

Formát výstupu

Vypište jedno číslo: nejvyšší dosažitelný počet vyvážených kapitol.

Příklady

Vstup:	*Výstup:*
`4` `-1 -1 1 1`	`1`

Nelze vytvořit víc než jednu vyváženou kapitolu. Abychom získali jednu, můžeme buď vzít všechny odstavce jako jedinou kapitolu, nebo odstavce rozdělit takto: `-1 | -1 1 | 1`.

Vstup:	*Výstup:*
`10` `0 1 1 0 -1 -1 -1 -1 1 -1`	`3`

Můžeme odstavce rozdělit takto: `0 | 1 1 0 -1 -1 | -1 -1 | 1 -1`. *První, druhá a čtvrtá kapitola jsou vyvážené. Všimněte si, že vyvážené jsou i kapitoly neobsahující smutné ani veselé odstavce. Vyššího počtu vyvážených kapitol dosáhnout nelze.*

Bodování

Následující tabulka popisuje, kolik bodů dostanete za jednotlivé testovací sady a jaké podmínky v nich platí.

body	n	*další*
1	2	
2	500	*Knihy neobsahují žádné smutné odstavce.*
3	500	
4	10^6	

P-I-3 Veletrh dortů

Radek se letos jako každý rok vypravil na veletrh dortů. Ten tvoří řada n stánků očíslovaných 1 až n. Pravý důvod Radkovy přítomnosti je, že za malý poplatek a_i Kč si Radek v i-tém stánku může ochutnat představovaný dort. Radek by samozřejmě rád ochutnal co nejvíce dortů, jenže má to dva háčky. Za prvé Radek disponuje pouze sumou k Kč. Za druhé moc dobře ví, že jakmile poprvé ochutná dort v nějakém stánku i, nedá mu to a ochutná poté dort ve vedlejší stánku $i + 1$, následně ve stánku $i + 2$ a tak dále, než mu dojdou peníze nebo než dojde na konec řady (tehdy skončí, i kdyby mu ještě zbývalo plno peněz). Ve kterém stánku má Radek začít, aby ochutnal co nejvíce dortů?

Soutěžní úloha

Je zadána posloupnost n kladných celých čísel $a_1, a_2, \ldots, a_n$, kde a_i je cena i-tého dortu, a kladné celé číslo k, tedy počet korun, které Radek má k dispozici. Zjistěte, kolik dortů může nejvíce Radek sníst a u kterého stánku má začít, aby toho dosáhl. Vaším cílem je vypsat dvě čísla d a p taková, že platí následující:

1. $1 \leq d \leq n$ a $1 \leq p \leq n - d + 1$.
2. $a_p + a_{p+1} + \ldots + a_{p+d-1} \leq k$, tedy lze ochutnat d dortů, začne-li Radek na pozici p.
3. pro všechna $1 \leq p' \leq n - d$ platí $a_{p'} + a_{p'+1} + \ldots + a_{p'+d} > k$, tedy více než d dortů ochutnat nelze.

Pokud existuje více vyhovujících pozic p, můžete vypsat kteroukoliv z nich.

Formát vstupu

První řádek obsahuje dvě kladná celá čísla, n a k. Druhý řádek obsahuje n kladných celých čísel $a_1, a_2, \ldots, a_n$.

Formát výstupu

Výstup je tvořen dvěma čísly d a p, kde d je největší počet dortů, které Radek může ochutnat, a p je číslo stánku, u něhož má začít, aby d dortů snědl.

Příklady

Vstup:	*Výstup:*
`7 6` `2 1 2 3 2 2 1`	`3 2`

Radek může ochutnat tři dorty, pokud začne ochutnávkou druhého dortu: za dorty zaplatí $1+2+3 = 6$ *Kč. Jiné správné řešení by bylo začít na první, nebo páté pozici.*

Vstup:	*Výstup:*
`3 10` `100 100 100`	`0 1`

V tomto případě nemůže Radek ochutnat ani jeden dort, prázdná řada dortů pak může začínat na pozici 1, 2, *nebo* 3.

Bodování

Plných 10 bodů získá řešení s optimální časovou složitostí.

Za řešení, které na běžném počítači vyřeší úlohu pro $n = 10^6$ během několika sekund, dostanete až 7 bodů.

Za řešení dostatečně efektivní pro $n = 5000$ dostanete až 4 body.

Za jakékoli funkční řešení dostanete až 2 body.

P-I-4 Matice na disku

K této úloze se vztahuje studijní text uvedený na následujících stranách. Doporučujeme vám nejprve prostudovat studijní text a až potom se vrátit k samotným soutěžním úlohám.

Mějme matici (dvojrozměrné pole čísel) velikosti $N \times N$ uloženou na disku po řádcích. Tím myslíme, že nejprve jsou uloženy všechny prvky prvního řádku zleva doprava, pak všechny prvky druhého atd. Můžete předpokládat, že N je násobkem velikosti bloku B a že velikost paměti M je mnohem menší než počet prvků matice N^2.

a) *(2 body)* Vypište prvky matice po řádcích (řádky shora dolů, v každém řádku prvky zleva doprava).

b) *(3 body)* Vypište prvky matice po sloupcích (sloupce zleva doprava, v každém sloupci prvky shora dolů).

c) *(5 bodů)* Otočte prvky v matici po směru hodinových ručiček. Například levý horní prvek se dostane do pravého horního rohu, pravý horní prvek skončí vpravo dole atd. Pokud vám to pomůže, můžete předpokládat, že $M \geq B^2$. Výsledná matice bude na disku uložena tam, kde byl uložen vstup.

U všech algoritmů analyzujte časovou a komunikační složitost.

Studijní text

V tomto ročníku olympiády se budeme zabývat počítáním s obrovským množstvím dat. Budeme zpracovávat vstupy, které se nám ani celé nevejdou do hlavní paměti počítače. Proto naše programy budou muset pracovat s daty uloženými na disku a vyrovnat se s tím, že disk je mnohem pomalejší než hlavní paměť. Budeme přitom uvažovat následující zjednodušený model disku.

Náš počítač má k dispozici *vnitřní a vnější paměť*. Pro zjednodušení budeme vnitřní paměti říkat prostě *paměť* a té vnější *disk*.

K datům ve (vnitřní) pamětí můžeme přistupovat přímo – tak, jak jsme zvyklí v běžných algoritmech. Tato paměť ale má jenom omezenou kapacitu: M položek, kde M je nějaký parametr, který určíme později. Do každé položky můžeme uložit jedno „rozumně velké“ číslo. Tím myslíme čísla, která nejsou řádově větší než čísla na vstupu.

Disk je neomezeně velký. Je rozdělený na *bloky* o velikosti B položek (další parametr). S těmito položkami ale nelze počítat přímo. Můžeme pouze načíst celý blok do paměti, anebo naopak B po sobě jdoucích položek paměti zapsat do jednoho bloku na disku. Čtení budeme v programech zapisovat jako volání funkce READ(b), která dostane pořadové číslo bloku b a vrátí obsah bloku. Podobně WRITE(b, *obsah*) zapíše blok na disk.

Efektivitu algoritmů posuzujeme pomocí dvou druhů složitosti: *časové složitosti* výpočtů v paměti a *komunikační složitosti,* což je počet přenosů bloků (čtení a zápisů) mezi pamětí a diskem.

Budeme se snažit hledat algoritmy, které dosahují stejné časové složitosti jako na klasickém počítači, a k tomu mají co nejlepší komunikační složitost. Nebude-li řečeno jinak, naše algoritmy by měly fungovat pro libovolné hodnoty parametrů M a B. Můžeme přitom předpokládat, že $\mathcal{O}(B)$ položek se do paměti vejde. Speciálně nám tedy paměť vystačí na konstantní počet číselných proměnných. Ale pokud bychom vytvořili rekurzivní funkci, musíme si dát pozor na prostor potřebný na zásobník.

Příklad: Sekvenční průchod

Práci s diskem si vyzkoušíme na jednoduchém příkladu. Na disku dostaneme posloupnost N celých čísel. Čísla budou rozdělena do bloků: v prvním bloku prvních B čísel, pak dalších B atd. Celkem tedy vstup zabírá $\lceil N/B \rceil$ bloků. Naším úkolem bude najít mezi zadanými čísly to největší.

Půjdeme na to jednoduše: budeme postupně procházet bloky vstupu jeden po druhém. Každý blok načteme do paměti, projdeme všechny jeho prvky, přičemž si udržujeme průběžné maximum. Poté už blok nebudeme potřebovat, takže tutéž paměť můžeme využít na další blok. V pseudokódu by to vypadalo takto:

1. $i \leftarrow 0$ ◁ *Aktuální pozice ve vstupu*
2. $m \leftarrow -\infty$ ◁ *Průběžné maximum*
3. Dokud $i < N$: ◁ *Ještě něco zbývá*
4. $\quad b \leftarrow \textsc{Read}(i/B)$ ◁ *Přečteme další blok vstupu*
5. $\quad n \leftarrow \min(B, N - i)$ ◁ *Kolik v něm je položek?*
6. $\quad$ Pro j od 0 do $n - 1$: ◁ *Projdeme položky*
7. $\quad\quad m \leftarrow \max(m, b[j])$
8. $\quad i \leftarrow i + z$
9. Vypíšeme m.

Časová složitost tohoto algoritmu je jistě lineární s velikostí vstupu, tedy $\mathcal{O}(N)$. Jak je to s komunikační složitostí? Každý blok vstupu načteme právě jednou, takže přeneseme $\lceil N/B \rceil \leq N/B + 1$ bloků. To je jistě nejlepší možné, protože na každý blok se musíme alespoň jednou podívat.

Ještě musíme ověřit, kolik potřebujeme vnitřní paměti. Spotřebujeme konstantní počet položek na proměnné a B položek na právě zpracovávaný blok. To se určitě vejde do $\mathcal{O}(B)$, a tím pádem i do paměti.

Příklad: Test symetrie

Opět bude zadána nějaká posloupnost čísel. Tentokrát budeme chtít zjistit, zda je symetrická, tedy zde první prvek je roven poslednímu, druhý předposlednímu a tak dále.

Kdyby byla délka vstupu dělitelná B, stačilo by porovnat první blok s posledním (první má být zrcadlovým obrazem posledního), druhý s předposledním atd. Přitom bychom každý blok přečetli právě jednou.

Pokud délka vstupu dává po dělení B nějaký nenulový zbytek $z = N \bmod B$,

je to trochu složitější: zrcadlový obraz prvního bloku bude zčásti v z položkách posledního bloku a zčásti v $B - z$ položkách předposledního.

Představíme si proto, že spustíme současně dva programy: jeden bude číst vstup od začátku do konce, druhý od konce k začátku. Každý z nich si bude udržovat svůj aktuální blok a čas od času přečte nový. Pokaždé porovnáme aktuální položku přečtenou oběma programy.

První program se bude chovat stejně jako sekvenční čtení z minulého příkladu, druhý bude číst tytéž bloky pozpátku. Dohromady tedy přečtou nejvýše $2\lceil N/B \rceil \leq 2N/B + 2$ bloků. Dokonce se mohou zastavit po zpracování $N/2$ párů prvků, čímž se komunikační složitost zlepší na $N/B + 2$. Časová složitost je stále $\mathcal{O}(N)$.

Dodejme ještě, že zapisování komunikační složitosti pomocí $\mathcal{O}$-čkové notace je poněkud zrádné. Funkci $2N/B + 2$ totiž nelze jen tak zjednodušit na $\mathcal{O}(N/B)$ – pro funkce dvou (a více) proměnných už neplatí, že přičítanou konstantu můžeme jen tak „schovat do $\mathcal{O}$“.* Nejjednodušší možný zápis tedy je $\mathcal{O}(N/B + 1)$.

Příklad: Otočení posloupnosti

Pokračujeme v předchozím příkladu: co kdybychom chtěli posloupnost čísel otočit? Tedy přesunout první prvek na poslední místo, druhý na předposlední a tak dále.

Opět nejprve uvažujme případ, kdy je délka posloupnosti dělitelná velikostí bloku. Tehdy můžeme postupovat po blocích. Nejprve načteme do paměti první a poslední blok, otočíme obsah obou bloků a zapíšeme je zpět na disk v opačném pořadí. Pak provedeme totéž s druhým a předposledním blokem atd. A nakonec, máli posloupnost lichý počet bloků, přečteme prostřední blok, otočíme ho a zapíšeme zpět.

Všimněte si, že takto každý blok jednou přečteme a jednou zapíšeme, tudíž komunikační složitost činí opět $\mathcal{O}(N/B)$.

V případě, kdy zbytek N mod B není nulový, použijeme osvědčenou úvahu se současně běžícími programy, které si vyměňují prvky. Komunikační složitost pak bude $\mathcal{O}(N/B + 1)$.

* Notace $f(n) = \mathcal{O}(g(n))$ totiž znamená, že existuje konstanta c taková, že pro všechna n kromě konečně mnoha výjimek platí $f(n) \leq c \cdot g(n)$. U funkcí více proměnných definujeme $\mathcal{O}$ obdobně a povolíme konečně mnoho výjimek pro každou proměnnou. Jenže zlomek N/B může být libovolně malý i po zakázání konečně mnoha hodnot N i B, takže $+1$ nepřebijeme sebevětší konstantou c.