68. ročník Matematické olympiády – 2018/2019

Úlohy krajského kola kategorie P

Krajské kolo 68. ročníku MO kategorie P se koná v úterý 22. 1. 2019 v dopoledních hodinách. Na řešení úloh máte 4 hodiny čistého času. V krajském kole MO-P se neřeší žádná praktická úloha, pro zajištění rovných podmínek řešitelů ve všech krajích je použití počítačů při soutěži zakázáno. Zakázány jsou rovněž jakékoliv další pomůcky kromě psacích potřeb (např. knihy, výpisy programů, kalkulačky, mobilní telefony). Řešení každé úlohy vypracujte na samostatný list papíru.

Řešení každé úlohy musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, *argumenty zdůvodňující jeho správnost* (případně důkaz správnosti algoritmu), diskusi o efektivitě vašeho řešení (časová a paměťová složitost). Slovní popis řešení musí být jasný a srozumitelný i bez nahlédnutí do samotného zápisu algoritmu (do programu). Není možné odkazovat se na vaše řešení úloh domácího kola, opravovatelé je nemusí mít k dispozici.
- **Zápis algoritmu.** Ve všech úlohách je třeba uvést zápis algoritmu v nějakém dostatečně srozumitelném pseudokódu (případně v programovacím jazyce Pascal, C/C++ nebo Python). Nemusíte detailně popisovat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů, vyhledávání v poli, třídění apod.

Za každou úlohu můžete získat maximálně 10 bodů. Hodnotí se nejen správnost řešení, ale také kvalita jeho popisu a efektivita zvoleného algoritmu. Algoritmy posuzujeme podle jejich časové složitosti, tzn. závislosti doby výpočtu na velikosti vstupních dat. Záleží přitom pouze na řádové rychlosti růstu této funkce. V zadání každé úlohy najdete přibližné limity na velikost vstupních dat. Efektivním vyřešením úlohy rozumíme to, že váš program spuštěný s takovými daty na současném běžném počítači dokončí výpočet během několika sekund.

Vzorová řešení úloh naleznete krátce po soutěži na webových stránkách olympiády *http://mo.mff.cuni.cz/*. Na stejném místě bude zveřejněn i seznam úspěšných řešitelů krajského kola a seznam řešitelů postupujících do ústředního kola.

P-II-1 Tulipány

Radovan má dlouhý (ale úzký) záhonek, který si rozdělil na n po sobě jdoucích čtverečků a do každého z nich vysadil tulipán. Po pár dnech zjistil, že některé z vysazených tulipánů mu sežrali slimáci a místo jedné souvislé řady tulipánů jich má několik skupin oddělených posloupnostmi vyžraných čtverečků. Radovan se zaradoval, že se mu alespoň vyřešilo, jak tulipány rozdělit do kytic: pokaždé, když bude potřebovat kytici, sklidí jednu takovou skupinu.

Pak si ale uvědomil, že v kytici má být lichý počet květin. Rád by tedy na některé vyžrané čtverečky dovysadil tulipány tak, aby jich každá skupina měla lichý počet. Všiml si ovšem, že takovým dovysazením se víc skupin může spojit v jednu (například pokud dvě skupiny jsou oddělené jen jedním vyžraným čtverečkem, na který dovysadí tulipán), což celou situaci komplikuje. Navíc chce za nové sazenice utratit co nejméně.

Soutěžní úloha

Pro každý z n čtverečků víte, zda je na něm tulipán nebo ne. Určete nejmenší počet tulipánů, které je potřeba vysadit na prázdné čtverečky, aby ve výsledku každá maximální souvislá posloupnost čtverečků s tulipány měla lichou délku.

Formát vstupu

Vstupem je řetězec liché délky n (kde $n \leq 100\,000$), jehož i-té písmeno je `T`, je-li na i-tém čtverečku tulipán, a `P`, je-li čtvereček prázdný.

Formát výstupu

Vypište jedno přirozené číslo: nejmenší počet čtverečků, na něž je potřeba vysadit tulipány tak, aby jich v každé souvislé skupině byl lichý počet (jelikož n je liché, nějaké řešení vždy existuje).

Příklad

Vstup:	*Výstup:*
`TPTTPTTPPTT`	`2`

Tulipán stačí vysadit na 5. a 9. čtvereček.

P-II-2 Ohrada

Kromě trojúhelníkové zahrádky by Novákovi na svém pozemku rádi měli ohradu na slepice ve tvaru konvexního čtyřúhelníka. Na pozemku stojí n stromů, stačí si tedy 4 z nich vybrat, natáhnout kolem nich pletivo a je hotovo. Paní Nováková ale rozhodně nechce, aby uvnitř ohrady byl nějaký jiný strom (jednak by se z něj špatně sklízelo ovoce, jednak by se o něj mohla zranit při honění slepic).

Soutěžní úloha

Máte dáno n bodů v rovině, z nichž žádné tři neleží na stejné přímce. Najděte 4 z nich tvořící rohy konvexního čtyřúhelníka, v jehož vnitřku neleží žádný ze zadaných bodů.

Formát vstupu

Na prvním řádku vstupu je přirozené číslo n (kde $4 \leq n \leq 100\,000$). Na každém dalším řádku jsou dvě přirozená čísla x a y, udávající souřadnice jednoho z bodů.

Formát výstupu

Vypište souřadnice čtyř ze zadaných bodů (v libovolném pořadí) splňujících podmínky zadání, nebo řetězec `nelze`, pokud žádné takové body neexistují.

Příklady

Vstup:

```
5
0 0
10 0
5 10
5 1
6 -1
```

Výstup:

```
0 0
6 -1
10 0
5 1
```

Vstup:

```
4
0 0
10 0
5 10
5 1
```

Výstup:

```
nelze
```

P-II-3 Úřady

V zemi krále Miroslava je mnoho měst propojených silnicemi a kanály. Města jsou očíslována od 1 do n v rostoucím pořadí dle jejich nadmořské výšky. Mezi každými dvěma městy vede nejvýše jeden kanál a dá se po něm plavit pouze po proudu, tedy z města z vyšším číslem do města s nižším číslem. Mezi každými dvěma městy také vede nejvýše jedna cesta, kterou lze použít obousměrně. Mezi dvěma městy může vést zároveň cesta i kanál. Dvě města jsou *sousední,* pokud mezi nimi vede kanál nebo cesta.

Král Miroslav chce zřídit úřady dohlížející na obchodníky v jeho městech. Každý úřad může dohlížet na obchodníky ve městě ve kterém sídlí, a dále ve městech, do kterých se z něj dokáží dostat kurýři na koních (přes posloupnost navazujících cest) nebo na lodích (po proudu přes posloupnost navazujících kanálů). Kurýr nemůže přesednout z koně na loď ani naopak.

Je samozřejmě potřeba, aby na obchodníky v každém z měst dohlížel alespoň jeden úřad. Král Miroslav ale nechce, aby úřady sídlily v sousedních městech (úředníci v takových sousedních městech by se buď hádali, nebo proti králi organizovali spiknutí).

Soutěžní úloha

V zadané síti měst propojených kanály a cestami najděte podmnožinu měst U takovou, že žádná dvě města v U nejsou sousední a do každého města mimo U se dá dostat z nějakého města z U buď po navazující posloupnosti cest, nebo po navazující posloupnosti kanálů používaných pouze po proudu.

Formát vstupu

Na prvním řádku vstupu jsou přirozená čísla n, k a c (kde $1 \leq n \leq 100\,000$ a $0 \leq k, c \leq 100\,000$) udávající počet měst, kanálů a cest. Na k dalších řádcích jsou dvě přirozená čísla m_1 a m_2 ($1 \leq m_1 < m_2 \leq n$), značící, že z města číslo m_2 vede kanál do města číslo m_1. Na c dalších řádcích jsou dvě přirozená čísla m_1 a m_2 ($1 \leq m_1 < m_2 \leq n$), značící, že mezi městy s čísly m_1 a m_2 vede cesta.

Formát výstupu

Vypište čísla měst tvořících nějakou podmnožinu U splňující podmínky zadání. Čísla můžete uvést v libovolném pořadí. Pokud žádná taková podmnožina neexistuje, vypište řetězec `nelze`.

Příklad

Vstup:	*Výstup:*

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

Jiná správná řešení jsou například 2 3, 2 4 či 2 6 4.

P-II-4 Doprava

K této úloze se vztahuje studijní text uvedený na následujících stranách, tentýž jako v domácím kole.

Našli jste si práci ve vedlejším městě, do kterého dojíždíte vlakem. Nicméně váš nový šéf je známý svou impulzivitou a kdykoliv vás může vyhodit. Každé ráno si tedy přečtete e-maily a zjistíte tak, zda ještě máte zaměstnání. Pokud ano, koupíte si jízdenku a jedete.

Normální jízdenka stojí 2 Kč. Každý večer máte možnost si koupit za $a \geq 1$ Kč slevovou kartu; máte-li ji, každá další jízdenka vás stojí pouze 1 Kč. Slevovou kartu si můžete koupit i večer před prvním dnem svého zaměstnání (ale i ráno před prvním dnem zaměstnání můžete dostat výpověď).

V okamžiku, kdy vás vyhodí z práce, byste chtěli mít utraceno za dojíždění (jízdenky plus případně slevová karta) co nejméně. Navrhněte algoritmus s co nejlepším poměrem ceny vůči optimálnímu řešení (které dopředu ví, kdy vás vyhodí).

Studijní text

V olympiádě se většinou zabýváme úlohami, v nichž dopředu známe celá vstupní data a na jejich základě produkujeme celý výstup. Snadno si ale lze představit situace, v nichž se vstupní data dozvídáme postupně a výstup musíme vyrábět průběžně, pouze s ohledem na část vstupů, které již známe. O takových problémech říkáme, že jsou *on-line.*

Příklad: Externí paměť

Počítač má pracovní paměť omezené velikosti – vejde se do ní nejvýše k bloků dat – a výrazně větší externí paměť, skládající se ze stejně velkých bloků identifikovaných přirozenými čísly. Je-li potřeba zpracovat nějaký blok dat, musí být nejprve nahrán do pracovní paměti. Jestliže je v tomto okamžiku pracovní paměť plná, musíme se rozhodnout, který z aktuálně nahraných bloků z ní vyhodíme (byl-li modifikován, zkopírujeme ho přitom zpět do externí paměti – nemusíme se tedy při volbě vyhozeného bloku bát, že bychom o nějaká data přišli).

Naše úloha je tedy následující: Na začátku je pracovní paměť prázdná. Postupně dostáváme čísla bloků z externí paměti, které je třeba zpracovat. Pokud daný blok už v pracovní paměti je, neděláme nic. Pokud v ní není, ale v pracovní paměti je méně než k nahraných bloků, zadaný blok bude nahrán na jedno z volných míst. Je-li pracovní paměť plná, vybereme, který z nahraných bloků vyhodíme, a zadaný blok bude nahrán na tímto uvolněné místo.

Jelikož přesuny mezi externí a pracovní pamětí jsou pomalé, chceme volit vyhozené bloky tak, abychom minimalizovali počet nahrání bloků do pracovní paměti.

Například, nechť $k = 2$ a postupně dostáváme požadavky 1 2 3. Při prvních dvou požadavcích nic nerozhodujeme, do pracovní paměti se

nahrají bloky 1 a 2. Při třetím požadavku se musíme rozhodnout, který z bloků v pracovní paměti vyhodíme – třeba blok 2, takže poté budeme v pracovní paměti mít bloky 1 a 3. Přijde-li nám nyní požadavek 1, nemusíme nic dělat a celkově tedy proběhnou 3 nahrání bloků do pracovní paměti. Kdyby nám ale místo toho přišel požadavek 2, pak musíme uvolnit místo v pracovní paměti a blok 2 znovu nahrát, celkově by tedy proběhly 4 nahrání bloků do pracovní paměti.

Jak srovnávat a vyhodnocovat algoritmy řešící on-line problémy? Mohli bychom samozřejmě zjišťovat, jak kvalita jejich řešení závisí (v nejhorším případě) třeba na délce vstupu, to ale typicky není příliš informativní. Například, ve výše uvedeném příkladu pro vstup 1 2 3 ... n nutně musíme provést n nahrání bloků, a to nezávisle na tom, jaký algoritmus pro výběr vyhozených bloků použijeme – dokonce i kdybychom vstup znali celý předem, nemohli bychom dosáhnout lepšího výsledku.

Jako zajímavější možnost se nabízí srovnávat náš on-line algoritmus na každém vstupu s optimálním algoritmem, který zná celý vstup dopředu. Nechť v je libovolný vstup (v diskutovaném příkladu je v posloupnost požadavků). Jakožto $\mathrm{OPT}(v)$ si označme hodnotu optimálního řešení pro vstup v. Jakožto $\mathrm{ALG}(v)$ si označme hodnotu řešení získaného uvažovaným on-line algoritmem, který vstup dostává postupně a výstup produkuje průběžně. Pokud pro nějaké číslo c platí, že $\mathrm{ALG}(v) \leq c \cdot \mathrm{OPT}(v)$ pro každý vstup v, říkáme, že uvažovaný algoritmus je *c-kompetitivní.*

Příklad: Fronta a zásobník

Uvažujme algoritmus Fronta, který má bloky v pracovní paměti setříděné v pořadí, v němž byly nahrány, a vždy vyhazuje nejstarší z nich. Nechť $v = p_1, p_2, \ldots, p_n$ je libovolná posloupnost požadavků, na které optimální algoritmus nahrává blok do pracovní paměti pro i_1-tý, i_2-tý, až i_m-tý z nich; tj. $\mathrm{OPT}(v) = m$. Zjevně $i_1 = 1$, jelikož na začátku je pracovní paměť prázdná. Uvažujme úsek u mezi dvěma požadavky, pro něž optimální algoritmus nahrává blok, tedy $u = p_{i_j}, \ldots, p_{i_{j+1}-1}$ pro nějaké $j \in \{1, \ldots, m\}$. V rámci úseku u mohou být požadavky na pouze k různých bloků (těch, které má optimální algoritmus v pracovní paměti po vyřízení požadavku p_{i_j}).

Algoritmus Fronta proto na úseku u nahraje do pracovní paměti blok nejvýše k-krát (poté, co jednou nahraje blok číslo b, ho vyhodí až když nahraje k dalších různých bloků, což na úseku u nenastane). Stejnou úvahu můžeme použít na každém takovém úseku, proto algoritmus Fronta na vstupu v nahraje nejvýše $k \cdot m$ bloků. Algoritmus Fronta je tedy k-kompetitivní.

Oproti tomu uvažme algoritmus Zásobník, který vždy vyhazuje nejnovější načtený blok. Tento algoritmus na vstupu $v = 1, \ldots, k, k+1, k, k+1, k, k+1, \ldots, k, k+1$, kde úsek $k, k+1$ se opakuje n-krát, bude na střídačku vyhazovat a nahrávat bloky k a $k+1$, celkem tedy nahraje $2n + k - 1$ bloků do pracovní paměti. Oproti tomu optimální algoritmus při prvním

požadavku $k+1$ vyhodí blok 1 a od té chvíle má bloky k i $k+1$ v paměti, provede tedy celkem $k+1$ nahrání bloků. Poměr mezi počtem nahrání bloků algoritmu Zásobník a optimálního algoritmu tedy může být libovolně velký.

Naším cílem je samozřejmě získat pro zadaný problém algoritmus, který je c-kompetitivní pro co nejmenší možné c (oproti tomu se nebudeme příliš zabývat časovou a paměťovou složitostí těchto algoritmů, nemusíte ji tedy v řešeních vašich úloh určovat). Často se stane, že pro danou úlohu umíme dokonce ukázat, že lepší než c-kompetitivní algoritmus existovat nemůže.

Příklad: Fronta je optimální

Uvažujme libovolný algoritmus ALG pro diskutovaný problém správy pracovní paměti. Zkonstruujeme vstup v délky n následujícím způsobem: začneme požadavky $1, \ldots, k$. Poté se vždy podíváme na obsah pracovní paměti a budeme požadovat ten blok z $\{1, \ldots, k+1\}$, který v ní není. Algoritmus ALG tedy bude nahrávat blok při každém požadavku, celkem tedy bude nahrávat $\mathrm{ALG}(v) = n$-krát.

Oproti tomu optimální algoritmus se (po prvních k vynucených krocích), vždy když musí vyhodit blok z pracovní paměti, podívá na $k-1$ následujících požadavků a vyhodí nějaký blok, který se mezi nimi nevyskytuje. Tak si zajistí, že v $k-1$ následujících požadavcích nebude muset nahrávat nový blok do pracovní paměti. Celkem tedy nahraje $\mathrm{OPT}(v) \leq k + \lceil (n-k)/k \rceil \leq (n+k^2)/k$ bloků do paměti.

Máme tedy

$$\frac{\mathrm{ALG}(v)}{\mathrm{OPT}(v)} \geq \frac{n}{(n+k^2)/k} = k \cdot \frac{n+k^2-k^2}{n+k^2} = k \cdot \left(1 - \frac{k^2}{n+k^2}\right).$$

Pro dostatečně dlouhý vstup (velké n) je tento poměr libovolně blízko k. Žádný algoritmus ALG tedy nemůže být c-kompetitivní pro $c < k$. Výše popsaný algoritmus Fronta tedy má nejlepší možný kompetitivní poměr.

Mohli bychom namítnout, že možná není v pořádku konstruovat vstup průběžně tak, aby se na něm algoritmus ALG choval co nejhůře. Nicméně vzhledem k tomu, že algoritmus ALG je deterministický, bude na takto získaném vstupu fungovat vždy stejně – čili takto obdržíme pevný vstup v, pro nějž $\mathrm{ALG}(v)/\mathrm{OPT}(v)$ je blízko k.

Pozorný čtenář si možná povšimne, že úvaha z předchozího odstavce nefunguje pro pravděpodobnostní algoritmy, které využívají náhodná čísla. Analýza chování pravděpodobnostních algoritmů je výrazně obtížnější, omezíme se proto pouze na deterministické algoritmy. Ve vašich řešeních tedy nesmíte používat generátor náhodných čísel.