68. ročník Matematické olympiády – 2018/2019

Úlohy ústředního kola kategorie P – 1. soutěžní den

Na řešení úloh máte 4,5 hodiny čistého času. Řešení každé úlohy pište na samostatný list papíru. Při soutěži je zakázáno používat jakékoliv pomůcky kromě psacích potřeb (tzn. knihy, kalkulačky, mobily, apod.).

Řešení každého příkladu musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, argumenty zdůvodňující jeho správnost, 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 povoleno odkazovat se na Vaše řešení předchozích kol, opravovatelé je nemají k dispozici; na autorská řešení se odkazovat můžete.
- **Zápis algoritmu.** Ve všech úlohách je třeba uvést zápis algoritmu, a to buď ve tvaru zdrojového textu nejdůležitějších částí programu v jazyce Pascal nebo C/C++, nebo v nějakém dostatečně srozumitelném pseudokódu. 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 (včetně zdůvodnění správnosti) a efektivita zvoleného algoritmu. Algoritmy posuzujeme zejména podle jejich časové složitosti, tzn. podle 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.

P-III-1 Půl království

Honza zabil draka a nyní má dostat půl království (a princeznu za ženu, ale s tím není problém). Jelikož království je nekonečně velké a zabírá celou rovinu, není úplně jasné, jak tento slib realizovat. Po konzultaci s právníky se král rozhodl, že Honza si může vybrat libovolnou polorovinu. Během svého dobrodružství Honza získal mnoho přátel, ale i mnoho nepřátel. Rád by si vybral polorovinu, v níž je rozdíl počtu přátel a nepřátel co největší.

Soutěžní úloha

Máte dáno n bodů v rovině, z nichž žádné tři neleží na stejné přímce. Pro každý z těchto bodů máte také dáno, zda se na něm nachází Honzův přítel či nepřítel. Nalezněte polorovinu, v níž je rozdíl počet přátel mínus počet nepřátel největší možný. Počítají se pouze přatelé či nepřátelé žijící uvnitř této poloroviny, ne tedy ti, kteří žijí přesně na přímce tvořící hranici této poloroviny.

Formát vstupu

Na prvním řádku vstupu je přirozené číslo n (kde $1 \leq n \leq 1\,000$). Na každém z n dalších řádků jsou dvě celá čísla x a y a znak p popisující, že v bodu se souřadnicemi (x, y) žije Honzův přítel (jestliže p je `P`) či nepřítel (jestliže p je `N`).

Formát výstupu

Vypište čtyři reálná čísla x_1, y_1, x_2, y_2 popisující polorovinu ze zadání úlohy: navzájem různé body (x_1, y_1) a (x_2, y_2) leží na přímce tvořící hranici poloroviny a díváme-li se z bodu (x_1, y_1) směrem k bodu (x_2, y_2), pak tato polorovina leží vlevo od nás.

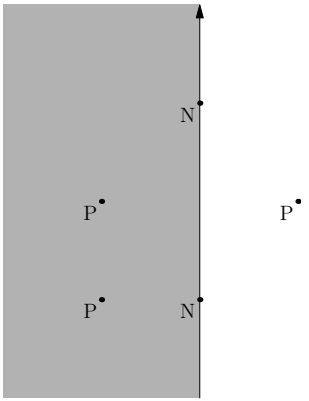
Příklad

Vstup:

```
5
0 0 P
0 1 P
1 0 N
1 2 N
2 1 P
```

Výstup:

```
1 -1 1 3
```



Příklad odpovídá situaci na obrázku. V Honzově polorovině žijí dva přátelé a žádný nepřítel (nepřátelé žijící na hraniční přímce se nepočítají). Jiné možné řešení je 2 1.5 1 1.5, v této polorovině žijí tři přátelé a jeden nepřítel, rozdíl je tedy také 2.

P-III-2 Pomluvy

Na internetu se začaly šířit nactiutrhačské texty o našem králi. Úřady by proti nim samozřejmě rázně zakročily, ale nactiutrhači naneštěstí používají několik triků, kterými se vyhýbají dopadení. Jednak své texty prokládají znaky navíc, aby se v nich nactiutrhačská hesla obtížněji hledala. Také tyto texty pravidelně obměňují a úředníci je nestíhají číst.

Soutěžní úloha

Máte zadáno nactiutrhačské heslo, skládající se z nejvýše 100 malých písmen. Dále máte dán dlouhý text tvořený malými písmeny a posloupnost v něm prováděných změn; změny jsou ve tvaru „písmeno na i-té pozici nahraď písmenem p“. Po každé změně rozhodněte, zda lze z textu smazat písmena tak, aby vzniklo zadané heslo.

Formát vstupu

Na prvním řádku je řetězec h délky mezi 1 a 100 včetně a skládající se z malých písmen, udávající hledané nactiutrhačské heslo. Na druhém řádku je řetězec délky m, kde $1 \leq m \leq 1\,000\,000$, skládající se z malých písmen a udávající počáteční stav textu. Na třetím řádku je přirozené číslo n, kde $1 \leq n \leq 100\,000$. Na každém z dalších n řádků je jedno přirozené číslo i (kde $1 \leq i \leq m$) a jedno malé písmeno p, popisující, že v textu se má písmeno na i-té pozici nahradit písmenem p.

Formát výstupu

Výstup se skládá z n řádků. Na j-tém z nich vypište `ano`, jestliže po j-té změně lze z textu smazat písmena tak, aby vzniklo heslo h, a `ne` jinak.

Příklad

Vstup:	*Výstup:*
`kralkrade`	`ano`
`xkxxrarylxkraxdxex`	`ne`
`3`	`ano`
`1 p`	
`5 p`	
`8 a`	

Po první změně je text `pkxxrarylxkraxdxex`*, z nějž lze smazat všechna písmena* `p` *a* `x` *a podřetězec* `ry` *a získat tak požadované heslo* `kralkrade`*. Po druhé změně je text* `pkxxparylxkraxdxex`*, který hledané heslo neobsahuje. Po třetí změně je text* `pkxxparalxkraxdxex`*, z nějž znovu lze smazáním písmen dostat požadované heslo.*

P-III-3 Kartičky

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

Hrajeme následující hru s kartičkami pěti barev (které si označme A, B, C, D, E). Na začátku máme od každé barvy dvě kartičky. V každém kole hry nám soupeř nabídne na výběr dvě kartičky různých barev a my si z nich jednu vybereme a vezmeme. Až se soupeř rozhodne, hra skončí a naše výsledné skóre je rovno největšímu počtu kartiček, který máme v jedné z barev.

Příklad: V prvním kole nám soupeř nabídne kartičky barev A a B, my si vybereme A. Ve druhém kole nám nabídne opět A a B, vybereme si opět A. Ve třetím kole nám nabídne B a C a my si vybereme B. Pak soupeř ohlásí konec hry. Máme 4 A, 3 B a 2 C, D a E, počet námi získaných bodů je maximum z těchto čísel, tedy 4.

Chceme navrhnout strategii, která je co nejlepší v porovnání s optimálně hrajícím „podvodníkem“, který předem zná soupeřovy tahy.

Příklad: Ve výše uvedené situaci by si podvodník mohl vybrat kartičku barvy B v každém kole a získal by tak 5 bodů. Zvládli jsme tedy získat 4/5 z možných bodů.

a) *(5 bodů)* Nalezněte strategii, která zaručí, že počet námi získaných bodů bude alespoň 1/2 počtu bodů získaných podvodníkem.

b) *(5 bodů)* Ukažte, že pro žádné $r > 1/2$ neexistuje strategie, která by zaručila, že počet námi získaných bodů bude alespoň r krát počet bodů získaných podvodníkem.

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, …, 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.