Příloha časopisu
MATEMATIKA – FYZIKA – INFORMATIKA
Ročník 27 (2018), číslo 4

Úlohy I. kola (domácí část) 68. ročníku MO (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 `http://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 úlohy P-I-3 musí obsahovat popis algoritmu, zdůvodnění jeho správnosti a odhad časové a paměťové složitosti. 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. Úloha P-I-4 je tvořena třemi samostatnými podúlohami. Část bodů dostanete, i když vyřešíte správně jenom některou z nich. Řešení obou teoretických úloh odevzdávejte ve formě souboru typu PDF přes výše uvedené webové rozhraní.

Řešení všech úloh můžete odevzdávat do 15. listopadu 2018. Opravená řešení a seznam postupujících do krajského kola najdete na webových stránkách olympiády na adrese `http://mo.mff.cuni.cz/`, kde jsou také k dispozici další informace o kategorii P.

P-I-1 Bourání města

Starosta Kocourkova si jednoho dne uvědomil, že kulturní úroveň města se zajisté zvýší, bude-li na každém rohu stát sud. Do takového útulného sudu, ideálně orientovaného na východ, se jistě brzy nastěhuje filosof libu-

jící si ve skrovném, leč klidném a každé ráno dobře osvětleném příbytku. Starostovo nařízení *Každá křižovatka bude mít sud postavený k východu* však bylo špatně vyslechnuto a k odpovědným radním se dostalo nařízení *Každá křižovatka bude mít sudý počet východů.* Poctiví radní se tedy rozhodli zbourat některé kocourkovské ulice tak, aby každá křižovatka sousedila se sudým počtem ulic. Než se do toho ale dali, radní se rozhodli vybrat některé důležité ulice s neobyčejnou kulturní hodnotou, které by se bourat neměly (shodou okolností jsou to zároveň ulice, ve kterých radní bydlí).

Soutěžní úloha

Je zadána síť ulic v Kocourkově sestávající z N křižovatek očíslovaných od 1 do N a s M obousměrnými ulicemi spojujícími některé dvojice křižovatek. Některé z ulic jsou důležité a nemůžou se zbořit. Ze zbylých ulic vyberte některé tak, aby po jejich zboření každá křižovatka sousedila se sudým počtem ulic. Speciálně křižovatka podmínku splňuje, i pokud po zboření vybraných ulic nesousedí s žádnou ulicí (jsme přeci v Kocourkově).

Formát vstupu

Program čte vstupní data ze standardního vstupu. První řádek obsahuje tři celá čísla N, M a K ($1 \leq N$, $0 \leq K \leq M$) oddělená mezerou. Každý z následujících řádků obsahuje dvě celá čísla od 1 do N udávající dvojici křižovatek, mezi kterými je ulice. Můžete předpokládat, že mezi každou dvojicí křižovatek vede nejvýše jedna ulice a že neexistují ulice s oběma konci na stejné křižovatce. Prvních K ulic je důležitých a nesmějí být zbourány.

Formát výstupu

Program vypíše na standardní výstup několik řádků. Na prvním řádku bude jedno celé číslo $0 \leq L \leq M - K$, počet nedůležitých ulic, které jste se rozhodli zbourat. Následuje L řádků, přičemž každý z nich obsahuje dvojici čísel od 1 do N udávajících ulici, kterou jste se rozhodli zbourat. Pořadí těchto dvou čísel se může lišit od pořadí, v jakém jsou zadána na vstupu. Každá ulice se ve výčtu objeví nejvýše jednou. Jestliže řešení neexistuje, vypište `-1`.

Příklady

Vstup:	*Výstup:*

`3 2 2`
`1 2`
`2 3`
	`-1`

V tomto případě žádnou ulici zbořit nesmíme. Křižovatka 1 sousedí s lichým počtem ulic, takže žádné řešení neexistuje.

Vstup:	*Výstup:*

`3 3 2`
`1 2`
`1 3`
`2 3`
	`0`

V tomto případě je správným řešením zbořit 0 ulic, neboť všechny křižovatky již sousedí se sudým počtem ulic.

Vstup:	*Výstup:*

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

Jiným správným řešením by bylo zbořit obě ulice sousedící s křižovatkou 3.

Bodování

Plných 10 bodů získá řešení, které efektivně vyřeší libovolný vstup s $N \leq 1\,000\,000$ a $M \leq 5\,000\,000$.

Částečný počet bodů nicméně lze získat za vyřešení některých podúloh. Váš program bude spuštěn na deseti testovacích sadách a za každou správně vyřešenou dostanete 1 bod. Následující tabulka obsahuje seznam podmínek, které dané sady splňují.

sady	*omezení*
1	$K = M$ a dále $N \leq 2\,000,\ M \leq 10\,000$
2	$K = M$
3	$K \leq 1$ a dále $N, M \leq 20$
4	$K \leq 1$ a dále $N \leq 2\,000,\ M \leq 10\,000$
5	$K \leq 1$
6	$N, M \leq 20$
7	$N \leq 2\,000,\ M \leq 10\,000$
8, 9, 10	bez přidaných omezení

P-I-2 Bonbóny

Rodiče se rozhodli motivovat své líné děti – Aničku a Honzíka – ke sportování, konkrétně k jízdě na kole. Za každý úsek, který ujedou, jim dají bonbóny. Nicméně ne všechny úseky jsou stejně těžké a navíc se Anička a Honzík neshodnou na jejich obtížnosti; Anička raději jezdí po rovině, Honzík má rád úseky lesem, kde je stín, a tak dále. Proto za každý úsek mohou děti dostat různé počty bonbónů.

Tato idea se celkem osvědčila, má ale jeden problém. Pokud na konci výletu mají Anička a Honzík různé počty bonbónů, navzájem si závidí a perou se. Rodiče se proto pokusili pozměnit odměny tak, aby při každém výletu začínajícím a končícím na stejném místě získali Anička a Honzík stejný počet bonbónů. Na vás je, abyste ověřili, zda se jim to podařilo.

Soutěžní úloha

Je dána síť M jednosměrných cyklostezek, spojujících křižovatky číslované od 1 do N, a pro každou z nich víte, kolik bonbónů za její projetí dostane Anička a Honzík. Cyklostezky se protínají se pouze na křižovatkách a neexistuje žádná cyklostezka se stejným začátkem a koncem, ale mezi dvěma křižovatkami může vést i více cyklostezek. Uzavřený výlet je posloupnost navazujících cyklostezek začínající a končící na stejném místě; v rámci výletu je možné projet některou cyklostezku i vícekrát a Anička a Honzík v tomto případě dostanou bonbóny za každé její projetí. Rozhodněte, zda existuje uzavřený výlet, na němž Anička a Honzík dostanou celkově různé počty bonbónů, a případně nějaký takový vypište.

Formát vstupu

Program čte vstupní data ze standardního vstupu. První řádek obsahuje dvě celá čísla N a M ($1 \leq N, M \leq 1\,000\,000$) oddělená mezerou, kde M udává počet cyklostezek a N počet křižovatek. Na každém z M následujících řádků jsou čtyři celá čísla z, d, a, h ($1 \leq z, d \leq N$, $1 \leq a, h \leq 1\,000$) oddělená mezerami. Tato čísla udávají, že z křižovatky číslo z na křižovatku číslo d vede cyklostezka, za jejíž projetí dostane Anička a bonbónů a Honzík h bonbónů. Trasy jsou číslované od 1 do M dle jejich pořadí na vstupu.

Formát výstupu

Existuje-li uzavřený výlet, na němž Anička a Honzík dostanou různé počty bonbónů, vypište libovolný takový výlet jako posloupnost čísel cyklostezek $t_1, \ldots, t_n$ oddělených mezerami takových, že pro $i = 1, \ldots, n-1$ cyklostezka číslo t_i končí na křižovatce, na níž cyklostezka číslo t_{i+1} za-

číná, a cyklostezka číslo t_n končí na křižovatce, na níž cyklostezka číslo t_1 začíná. Jinak vypište pouze číslo 0.

Příklady

Vstup:	*Výstup:*

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

Po projetí cyklostezek číslo 1 *(z* 1 *do* 2*),* 4 *(z* 2 *do* 4*) a* 5 *(z* 4 *zpět do* 1*) má Anička 3 bonbóny a Honzík* 4*. Jsou i jiná správná řešení, například 4 5 1 nebo 1 2 3 1 4 5.*

Vstup:	*Výstup:*

`3 3`
`1 2 1 1`
`2 3 1 2`
`3 1 2 1`
	`0`

Anička a Honzík mohou pouze kroužit po okruhu cyklostezek 1 2 3, vždy když se dostanou do bodu, v němž začali, mají stejný počet bonbónů (čtyřikrát tolik, kolikrát okruh objeli).

Vstup:	*Výstup:*

`3 2`
`1 2 1 1`
`2 3 1 2`
	`0`

Žádný výlet začínající a končící na stejném místě neexistuje, nehrozí tedy, že by na něm Anička a Honzík dostali různé počty bonbónů.

Bodování

Plných 10 bodů získá řešení, které efektivně vyřeší libovolný vstup s $N, M \leq 1\,000\,000$.

Částečný počet bodů nicméně lze získat za vyřešení některých podúloh. Váš program bude spuštěn na deseti testovacích sadách a za každou správně vyřešenou dostanete 1 bod. Následující tabulka obsahuje seznam podmínek, které dané sady splňují.

sady	*omezení*
1,2,3	$M, N \leq 20$
4	$M \leq 1\,000$ a nejvýše 6 křižovatek sousedí s více než dvěma cyklostezkami
5	$M \leq 1\,000\,000$ a nejvýše 6 křižovatek sousedí s více než dvěma cyklostezkami
6	$M, N \leq 1\,000$ a z každé křižovatky se po cyklostezkách dá dojet na libovolnou jinou křižovatku
7	$M, N \leq 1\,000$
8	z každé křižovatky se po cyklostezkách dá dojet na libovolnou jinou křižovatku
9, 10	bez přidaných omezení

P-I-3 Zahrádka

Novákovi si v malebné vesničce za Prahou koupili malý rodinný domek s velkým pozemkem. Na pozemku však roste jen tráva, což jejich zemědělské ambice neukojilo, a proto se rozhodli vytvořit si trojúhelníkovou zahrádku. Jako první krok vyběhli na pozemek a vyznačili si n možných pozic vrcholů zahrádky. Nyní zbývá už jen vybrat vhodné tři vrcholy, záhonek zrýt, zasít, hnojit, kropit, vytrhat plevel, kropit, odehnat slimáky, vytrhat plevel, odehnat slimáky, zbavit se krtků, kropit, ... a nakonec možná sklidit, pokud se urodí. Prostě čím větší zahrádka bude, tím víc s ní bude práce. Pan Novák by proto rád vybral takové tři vrcholy, aby trojúhelník jimi vyznačený měl co nejmenší obsah. Zjistil ale, že bodů vyznačili moc a že takový trojúhelník neumí najít. Pomůžete mu?

Soutěžní úloha

Na vstupu dostanete n bodů v rovině v obecné poloze (tj. žádné tři neleží na přímce). Vaším cílem je najít tři z nich, které tvoří trojúhelník s nejmenším obsahem. Pokud existuje více řešení, vypište kterékoliv z nich. Ve svém řešení se nemusíte zabývat zaokrouhlovacími chybami – můžete předpokládat, že výpočty probíhají s neomezenou přesností.

Formát vstupu

Na prvním řádku dostanete jedno přirozené číslo n, počet vyznačených vrcholů. Na každém z dalších n řádků dostanete dvě mezerou oddělená celá čísla x_i a y_i, souřadnice i-tého vyznačeného vrcholu.

Formát výstupu

Vypište řádek obsahující souřadnice tří vrcholů trojúhelníka s nejmenším obsahem. Souřadnice jednotlivých vrcholů oddělte středníky.

Příklad

Vstup:	*Výstup:*

`4`
`0 0`
`0 1`
`1 0`
`1 1`
	`0 0; 0 1; 1 0`

V tomto případě mají všechny trojúhelníky stejný obsah, takže lze vypsat kteroukoli trojici.

Vstup:	*Výstup:*

`4`
`0 0`
`2 0`
`0 1`
`100 100`
	`0 0; 0 1; 2 0`

Bodování

Pokud váš algoritmus rychle a správně odpoví na vstupy s $n \leq 100$, může získat až 4 body. Plných 10 bodů získá řešení, které rychle a správně odpoví na vstupy s $n \leq 1\,000$, tj. s časovou složitostí $O(n^2 \log n)$ nebo lepší.

P-I-4 Dva lupiči

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.

Dva lupiči rozdělují lup. Postupně věci (různých cen) vytahují z pytle a o každé z nich rozhodují, komu připadne. Chtějí, aby věci byly rozděleny co nejrovnoměrněji – budou tedy minimalizovat sumu cen věcí na dražší hromádce.

a) *(2 body)* Ukažte, že algoritmus, který dá všechny věci prvnímu lupiči, je 2-kompetitivní.
b) *(4 body)* Nalezněte 3/2-kompetitivní algoritmus.
c) *(4 body)* Ukažte, že žádný algoritmus nemůže být lepší než 3/2-kompetitivní.

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. Na-

příklad, 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 $\mathrm{OPT}(v) \leq$ $\leq k + \lceil (n-k)/k \rceil \leq (n+k^2)/k$ 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.