59. ročník Matematické olympiády – 2009/2010

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 použitého 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 vhodné odkazovat se na Vaše řešení předchozích kol, opravovatelé je nemají k dispozici; na autorská řešení se odkazovat můžete.
- **Program.** V úlohách **P-III-1** a **P-III-2** je třeba uvést dostatečně podrobný zápis algoritmu, např. ve tvaru pseudokódu nebo zdrojového textu nejdůležitějších částí programu v programovacím jazyce Pascal nebo C/C++. Ze zápisu můžete vynechat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů apod. V řešení úlohy **P-III-3** je nutnou součástí řešení program pro počítač Kvak.

Za každou úlohu můžete získat 0 až 10 bodů. Hodnotí se nejen správnost programu, ale také efektivita zvoleného algoritmu a kvalita popisu řešení.

P-III-1 Znovu čokoláda

Jeníček bude mít zanedlouho narozeniny. Jeho sestra Mařenka si ještě dobře pamatuje na olámanou čokoládu, kterou od Jeníčka dostala před několika měsíci. Rozhodla se proto, že mu dá k narozeninám podobný dárek. Zašla do sklepa a ze své tajné skrýše vzala čokoládu, kterou si tam kdysi ukryla. Myši již stihly ohryzat i tuto čokoládu, ale to Mařence nevadilo – stačí přece ohryzané části olámat.

Mařenka je ovšem šikovnější než Jeníček a uvědomila si, že nemusí lámáním vytvořit čtverec. Čokolády mají přece často obdélníkový tvar. Tím získá mnoho nových možností, jak lze vyrobit dárek pro Jeníčka.

Soutěžní úloha:

Je dán původní počet řádků R a počet sloupců S čokolády a matice $R \times S$ nul a jedniček udávající, která políčka čokolády zůstala zachována celá. Určete, kolika způsoby může Mařenka uskutečnit svůj plán. Jinými slovy, spočítejte, kolika způsoby lze ve zbytku čokolády vyznačit obdélník bez děr. Všechny hrany obdélníka musí samozřejmě ležet na hranách políček. Stejně velké obdélníky umístěné na různých místech původní čokolády považujeme za různá řešení.

Formát vstupu:

Na prvním řádku vstupu jsou dvě celá čísla R a S oddělená mezerou. Následuje R řádků, přičemž na r-tém z nich je S mezerami oddělených celých čísel $a_{r,1}, \ldots, a_{r,S}$. Je-li políčko čokolády na souřadnicích (r, s) celé, bude $a_{r,s} = 1$, v opačném případě $a_{r,s} = 0$.

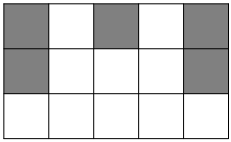
Formát výstupu:

Program vypíše na výstup jediné číslo – hledaný počet obdélníků.

Příklad:

Vstup:
```
3 5
0 1 0 1 0
0 1 1 1 0
1 1 1 1 1
```

Výstup:
```
33
```



Na obrázku vpravo je zobrazena čokoláda popsaná na vstupu. Šedou barvou jsou vyznačena políčka, která už myši stihly poškodit.

Obdélník 1×1 lze na této čokoládě vyznačit deseti způsoby, 2×1 pěti, 1×2 šesti, 3×1 dvěma, 1×3 čtyřmi, 1×4 dvěma, 2×2 dvěma, 1×5 jedním, a 2×3 také jedním způsobem.

P-III-2 Šachovnice

„Šach-mat," oznámil s úšklebkem Jeníček. Mařenka měla sice dosud šachy velmi ráda, ale už ji to přestává bavit: právě s Jeníčkem prohrála sedmnáctou partii po sobě. Vymyslela si proto novou, vlastní hru, v níž Jeníčka určitě porazí.

Hracím plánem je šachovnice, která je směrem doprava a nahoru nekonečná. Každé políčko této šachovnice můžeme označit dvojicí nezáporných celých čísel (x, y). Políčko v levém dolním rohu šachovnice má označení $(0, 0)$, směrem doprava roste souřadnice x, směrem nahoru vzrůstá souřadnice y.

Na šachovnici je rozmístěno N šachových koní. Na začátku i kdykoliv během hry může stát více koní na témže políčku. Koně se pohybují podle šachových pravidel. Jsou povoleny pouze takové tahy, při nichž kůň neopustí šachovnici a navíc klesne součet obou souřadnic (viz obrázek na následující straně).

Hráč, který je na tahu, si vybere několik koní (může si jich vybrat, kolik chce, ale musí vždy aspoň jednoho) a každým z nich provede jeden tah. Hráči se ve hře pravidelně střídají. Prohrává ten, kdo nemůže provést další tah (tzn. nemůže pohnout podle pravidel žádným koněm).

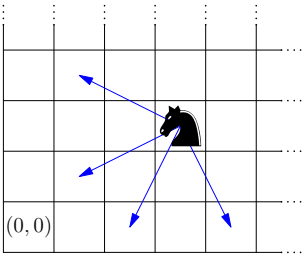
Mařenka si už napsala program, který za ni bude hrát tuto hru optimálním způsobem. Jeníček ale programovat neumí, a proto by přivítal vaši pomoc.

Soutěžní úloha:

Je dán počet koní N a jejich počáteční rozmístění na šachovnici. První tah provádí Jeníček.

Napište program, který zjistí, kdo vyhraje, když budou oba hráči hrát optimálně. Pokud zvítězí Jeníček, váš program by mu měl také poradit první tah libovolné vyhrávající strategie.

Jestliže úlohu nedokážete vyřešit pro obecné N, část bodů dostanete i v případě, že ji vyřešíte pro jednoho koně, případně pro dva koně začínající na souřadnicích mezi $(0, 0)$ a $(100, 100)$.



Všechny povolené tahy koněm na souřadnicích $(3, 2)$

Formát vstupu:

První řádek vstupu obsahuje počet koní N. Na každém z následujících N řádků jsou dvě čísla r_i a s_i, která určují řádek a sloupec, kde se nachází i-tý kůň na začátku hry.

Formát výstupu:

První řádek výstupu bude obsahovat jméno hráče, který vyhraje (`Jenicek` nebo `Marenka`). Jestliže vyhraje Jeníček, vypište pro každého koně, kterým má Jeníček v úvodním tahu táhnout, řádek s čísly r_a, s_a, r_b, s_b – původní a nová poloha koně. Na pořadí těchto řádků nezáleží.

Příklad 1:

Vstup:	*Výstup:*
`1`	`Marenka`
`0 4`	

Jeníček musí táhnout na $(1, 2)$*, odtud Mařenka táhne koněm na* $(0, 0)$ *a vyhraje.*

Příklad 2:

Vstup:	*Výstup:*
`2`	`Jenicek`
`3 1`	`3 1 1 0`
`2 1`	`2 1 0 0`

Po uvedeném tahu Jeníčka Mařenka ihned prohraje, neboť ani jedním koněm už nemůže pohnout.

Všimněte si, že kdyby Jeníček nechal koně z políčka $(3, 1)$ *na původním místě, prohraje. Prohrál by i v případě, že by tohoto koně přesunul na políčko* $(1, 2)$*, bez ohledu na to, zda by druhým koněm pohnul, nebo ne.*

P-III-3 Počítač Kvak

V tomto ročníku olympiády se setkáváme se speciálním počítačem nazvaným Kvak. Ve studijním textu uvedeným za zadáním úlohy je popsáno, jak tento počítač funguje. Studijní text je shodný s textem z domácího a krajského kola.

Soutěžní úloha:

a) *(3 body)* V rouře počítače je posloupnost kladných celých čísel. Označme si je $a_1, a_2, \ldots, a_N$ v pořadí, v němž se v rouře nacházejí. Napište program, který zkontroluje, zda je tato posloupnost rostoucí. Pokud ano, program ukončí výpočet, aniž by cokoliv vypsal. Jestliže posloupnost není rostoucí, program zjistí a vypíše nejmenší i takové, že $a_i \geq a_{i+1}$.

b) *(7 bodů)* V rouře počítače je posloupnost kladných celých čísel. Víte, že jedno z těchto čísel má v rouře nadpoloviční většinu – toto číslo se tedy v rouře vyskytuje vícekrát, než všechna ostatní čísla dohromady. Napište program, který toto číslo najde a vypíše.

Studijní text:

V letošním ročníku olympiády se budeme setkávat se speciálním počítačem zvaným Kvak.

Jediný datový typ, se kterým Kvak pracuje, se nazývá `number`, což je celé číslo z rozsahu od 0 do 65 535 včetně.* Všechny matematické výpočty provádí Kvak modulo 65 536, takže například hodnotou výrazu $65530 + 10$ je 4.

Kvak používá 26 proměnných, které nazýváme *registry*. Registry jsou označeny písmeny `a` až `z` a v každém z nich může být uložena jedna hodnota typu `number`. Na začátku výpočtu jsou ve všech registrech nuly.

Kromě registrů má Kvak ještě jednu *jednosměrnou rouru* neomezené délky, do které se mohu ukládat hodnoty typu `number`. Je to jediná datová struktura, kterou Kvak používá. S rourou lze provádět dvě operace:

- vložit do ní číslo z registru X příkazem `put` X,
- z opačného konce roury odebrat číslo a uložit ho do registru X příkazem `get` X.

Čísla se v rouře počítače nemohou předbíhat, Kvak je tedy bude odebírat ve stejném pořadí, v jakém je do roury vložil.** Roura má neomezenou kapacitu, lze do ní vložit libovolné množství čísel. Není-li řečeno jinak, roura je na začátku výpočtu prázdná.

Počítač Kvak má také možnost vypisovat čísla (výsledky výpočtu) na výstup.

Příkazy

V následující tabulce jsou shrnuty všechny příkazy, které Kvak umí provádět a které tedy můžete používat v programech.

---

* $65\,535 = 2^{16} - 1$, typ `number` je tedy přesně to, co znáte jako 16-bitové celé číslo bez znaménka.

** Takovou datovou strukturu obvykle nazýváme *fronta*.

příkaz	*význam příkazu*
`get` X	Kvak odebere jedno číslo z roury a uloží ho do registru X.
`put` X	Kvak vloží do roury číslo z registru X.
`put` *číslo*	Kvak vloží dané číslo do roury.
`print`	Kvak odebere jedno číslo z roury a vypíše ho na výstup.
`add`	sčítání: Kvak odebere dvě čísla z roury a vloží do roury jejich součet.
`sub`	odčítání: Kvak odebere dvě čísla z roury a vloží do roury jejich rozdíl (první minus druhé).
`mul`	násobení: Kvak odebere dvě čísla z roury a vloží do roury jejich součin.
`div`	dělení: Kvak odebere dvě čísla z roury a vloží do roury celou část jejich podílu (první lomeno druhé).
`mod`	zbytek: Kvak odebere dvě čísla z roury a vloží do roury zbytek, který dá první z nich po celočíselném dělení druhým.
`label` L	návěstí: Toto místo v programu dostane označení L (kde L může být libovolný řetězec). Stejné návěstí nesmí být v programu vícekrát.
`jump` L	skok: Kvak bude pokračovat v provádění programu od místa, které má označení L.
`jz` X L	skok jestliže nula: Je-li v registru X nula, Kvak provede příkaz `jump` L.
`jeq` X Y L	skok jestliže se rovnají: Je-li v registrech X a Y stejná hodnota, Kvak provede příkaz `jump` L.
`jgt` X Y L	skok jestliže je větší: Je-li v registru X větší hodnota než v registru Y, Kvak provede příkaz `jump` L.
`jempty` L	skok jestliže je prázdná: Není-li v rouře žádné číslo, Kvak provede příkaz `jump` L.
`stop`	konec: Kvak ukončí svůj výpočet.

Pokud se během výpočtu stane, že se pokusíme odebrat číslo z roury počítače a roura přitom bude prázdná, nastane chyba. Chyba nastane také tehdy, když se pokusíme dělit nulou, počítat zbytek po dělení nulou, nebo skočit na neexistující místo v programu. Dojde-li výpočet programu na konec, Kvak po provedení posledního příkazu korektně skončí (jako kdyby na konci programu byl ještě příkaz `stop`.)

V zápisu programu můžeme psát více příkazů na jeden řádek, v takovém případě je od sebe oddělujeme středníkem.

Příklad 1

Následující program spočítá a vypíše součet všech čísel od 1 do 20.

```
put 20
put 0
```

```
label start
get a
jz a end
put a ; put a ; put 1
add
sub
get b ; put b
jump start
label end
print
```

Pokaždé, když se Kvak při provádění programu dostane ke třetímu řádku (`label start`), budou v rouře právě dvě čísla. Jestliže první z nich označíme N, hodnota druhého bude rovna součtu $S = (N + 1) + \cdots + 20$. Poté načteme N do registru `a`. Je-li $N = 0$, máme v rouře hledaný součet, můžeme ho vypsat na výstup a skončit. V opačném případě chceme provést dvě věci: Přičíst N k dosud získanému součtu, a následně N zmenšit o 1. Po provedení řádku šest (tři příkazy `put`) máme v rouře postupně čísla: S, N, N, 1. Příkaz `add` sečte první dvě, po jeho provedení bude v rouře trojice čísel N, 1, $N+S$. Po vykonání dalšího příkazu `sub` budou v rouře hodnoty $N + S$ a $N - 1$. To už je téměř to, co potřebujeme, jenom v opačném pořadí. Proto první z nich načteme do registru `b` a znovu vložíme do roury.

Příklad 2

V rouře je neprázdná posloupnost čísel. Napíšeme program, který spočítá a vypíše na výstup jejich součet. (Přesněji, jeho zbytek po dělení 65 536.)

Budeme stále opakovat následující postup: Zjistíme, zda jsou v rouře aspoň dvě čísla. Jestliže ano, některá dvě z nich sečteme a nahradíme je jejich součtem. Pokud tam už dvě čísla nejsou, zůstalo tam tady už jenom jediné a to zjevně součtem všech původních čísel. V programu pro počítač Kvak můžeme tuto myšlenku implementovat například následovně:

```
label cyklus
get a
jempty konec
put a
add
jump cyklus

label konec
put a
print
```

Na začátku každé iterace odebereme z roury jedno číslo a vložíme ho do registru `a`. Pokud se tím roura vyprázdnila, máme v registru `a` hledaný součet, stačí ho už jenom vypsat. Pokud ne, číslo z registru `a` vrátíme zpět do roury. V tom okamžiku jsou v rouře alespoň dvě čísla a můžeme tedy bez obav provést příkaz `add`.

Časová složitost tohoto řešení je lineární vzhledem k počtu čísel, která byla na začátku výpočtu v rouře. Každá iterace cyklu totiž provádí jen konstantní počet příkazů a zmenší nám o jedno počet čísel v rouře.