54. ročník Matematické olympiády – 2004/2005

Úlohy domácího kola kategorie P

Řešení každého příkladu musí obsahovat podrobný popis použitého algoritmu, zdůvodnění jeho správnosti a diskusi o efektivitě zvoleného řešení (tzn. posouzení časových a paměťových nároků programu).

V praktických úlohách P-I-1, P-I-2 a P-I-3 je třeba k řešení také připojit odladěný program zapsaný v jazyce Pascal, C nebo C++. Program se odevzdává v písemné formě (jeho výpis je tedy součástí řešení) i na disketě, aby bylo možné otestovat jeho funkčnost. Slovní popis řešení musí být ovšem jasný a srozumitelný, aniž by bylo nutno nahlédnout do zdrojového textu programu. V úloze P-I-4 zapište navržený algoritmus ve formě programu v jazyce Alik.

Řešení úloh domácího kola MO kategorie P vypracujte a odevzdejte nejpozději do 15. 11. 2004. Vzorová řešení úloh naleznete po tomto datu na Internetu na adrese *http://mo.mff.cuni.cz/*. Na stejném místě jsou stále k dispozici veškeré aktuální informace o soutěži a také archiv soutěžních úloh a výsledků minulých ročníků.

P-I-1 Prádelna

Bořivoj se rozhodl, že začne podnikat – a to ve velkém. Jednou v noci, poté co upadl v koupelně, dostal skvělý nápad: otevře si prádelnu. Každý, kdo přijde, si bude moci za malý obnos půjčit pračku, pokud bude nějaká volná, a vypere si své prádlo. Ihned se tedy zeptal svých přátel, zda by chtěli jeho prádelnu navštěvovat, a zjistil, že zájem je opravdu veliký. Brzy ani nevěděl, kolik praček bude vůbec potřebovat, aby se dostalo na všechny zákazníky. A proto se rozhodl obrátit se na vás s prosbou o pomoc.

Soutěžní úloha: Na vstupu dostanete počet zakázek, které Bořivoj na jeden konkrétní den obdržel. U každé zakázky víte čas příchodu zákazníka a dobu, na jakou si chce pronajmout jednu pračku. Požadavky zákazníků nejsou uvedeny v žádném konkrétním pořadí.

Vaším úkolem je zjistit, kolik nejméně praček bude Bořivoj potřebovat, aby si každý zákazník mohl pronajmout pračku na celou požadovanou dobu od svého příchodu. Kromě minimálního počtu praček musíte pro Bořivoje vytvořit ještě seznam, podle kterého bude posílat zákazníky k volným pračkám.

Formát vstupu: První řádka textového souboru `pradelna.in` obsahuje jediné přirozené číslo $N \leq 10\,000$ – počet zákazníků. Dalších N řádek obsahuje informace o jednotlivých zákaznících: na i-té z těchto řádek je uveden čas T_i, kdy chce zákazník přijít, a doba T_i', na kterou si chce pronajmout pračku. Můžete předpokládat, že T_i a T_i' jsou celá čísla od 1 do $1\,000\,000\,000$.

Formát výstupu: První řádka textového souboru `pradelna.out` obsahuje jediné číslo P – nejmenší možný počet praček, s nimiž může Bořivoj obsloužit všechny zákazníky. Dalších N řádek bude obsahovat N čísel a_1 až a_N, přičemž a_i je číslo pračky, kterou má použít i-tý zákazník. Předpokládejte, že pračky budou očíslovány od 1 do P.

Příklad:

`pradelna.in`

```
4
1000 1000
1900 900
1500 700
2000 500
```

`pradelna.out`

```
3
2
1
3
2
```

P-I-2 Závody

Letos se po několika letech opět konají slavné závody švábů. Závody probíhají na pečlivě připravené překážkové dráze obsahující takové záludnosti, jako je třeba mistička s cukrem. Švábí závodníci jsou na trať vypouštěni v minutových intervalech a aby je bylo možno v cíli rozeznat, má každý závodník k sobě připevněnu cedulku s minutou startu (první šváb má tedy číslo nula, druhý jedna, atd.). Organizátory závodu by zajímalo, jak moc se švábi během tréninkového běhu promíchali. Pokud by se totiž promíchali hodně, bylo by třeba prodloužit intervaly mezi jednotlivými závodníky, aby se při běhu tolik neovlivňovali. Jako míra promíchanosti závodníků byl stanoven počet dvojic závodníků, kteří doběhli do cíle v opačném pořadí, než v jakém vyběhli na trať. Spočítat míru promíchanosti pro dané pořadí švábů v cíli je již úloha pro vás.

Váš program dostane na vstupu počet švábů N a pořadí, v jakém švábi doběhli do cíle (tedy nějakou permutaci čísel $0, \ldots, N-1$). Na výstup má váš program vypsat míru promíchanosti závodníků.

Formát vstupu: Vstupní textový soubor `zavody.in` obsahuje dva řádky. Na prvním řádku je uvedeno jedno celé číslo N, $1 \leq N \leq 30\,000$. Na druhém řádku je N různých celých čísel z intervalu $0, \ldots, N-1$ oddělených jednou mezerou.

Formát výstupu: Výstupní textový soubor `zavody.out` obsahuje jediný řádek s jedním celým číslem – počtem dvojic závodníků, kteří doběhli v opačném pořadí, než v jakém vystartovali.

Příklad:

`zavody.in`

```
5
1 0 4 2 3
```

`zavody.out`

```
3
```

P-I-3 Fylogenetika

Fylogenetika je obor biologie zabývající se rozpoznáváním vývojových vztahů mezi organismy. Často používanou metodou je srovnávání genetického kódu. V této úloze se budeme zabývat výrazně zjednodušenou variantou tohoto problému.

Genetický kód budeme mít uložen jako řetězec skládající se z písmen '`A`', '`C`', '`G`' a '`T`'. Budeme předpokládat, že vývoj nového druhu probíhá tak, že se na začátek nebo na konec genetického kódu připojí nové geny – to je samozřejmě pouze idealizace (čti: úplný nesmysl). Dostanete zadán genetický kód několika organismů, vaším úkolem je nalézt mezi nimi všechny dvojice předek – potomek, tj. takové, že genetický kód předka je souvislým podřetězcem genetického kódu potomka.

Formát vstupu: Vstupní textový soubor `fylogen.in` obsahuje několik řetězců složených z písmen '`A`', '`C`', '`G`' a '`T`', reprezentujících genetické kódy jednotlivých organismů. Organismy jsou očíslovány $1, 2, \ldots, n$; na i-tém řádku se nachází kód i-tého organismu. Můžete předpokládat, že řetězců je nejvýše 50, každý z nich má nejvýše 50 znaků a žádné dva řetězce nejsou stejné.

Formát výstupu: Výstupní textový soubor `fylogen.out` tvoří seznam všech dvojic předek – potomek. Každá řádka výstupního souboru popisuje jednu z těchto dvojic a sestává z čísla předka následovaného číslem potomka. Dvojice mohou být uvedeny v libovolném pořadí, nesmějí se však opakovat.

Příklad: Pro vstup

```
ATAT
CATATG
CATATGA
CATATGG
```

je jedním z možných správných řešení výstup

```
1 2
1 3
1 4
2 3
2 4
```

P-I-4 ALIK

Studijní text:

Aritmeticko-logický integerový kalkulátor (zkráceně Alik) je počítací stroj pracující s W-bitovými celými čísly v rozsahu 0 až $2^W - 1$ včetně; kdykoliv budeme hovořit o *číslech,* půjde o tato čísla. Budeme je obvykle zapisovat ve dvojkové soustavě polotučnými číslicemi a vždy si na začátek dvojkového zápisu doplníme příslušný počet nul, aby číslic (bitů) bylo právě W. Většinou také nebudeme rozlišovat mezi číslem a jeho dvojkovým zápisem, takže i-tým bitem čísla budeme rozumět i-tý bit jeho dvojkového zápisu (bity číslujeme zprava doleva od 0 do $W - 1$).

Paměť stroje je tvořena 26 *registry* pojmenovanými a až z. Každý registr vždy obsahuje jedno číslo.

Alik se řídí programem, což je posloupnost přiřazovacích příkazů typu *registr* := *výraz*, přičemž *výraz* může obsahovat konstanty (čísla zapsaná ve dvojkové soustavě), registry, závorky a následující operátory (řecká písmena značí podvýrazy, v pravém sloupci jsou priority operátorů):

$\alpha + \beta$	sečte čísla α a β. Pokud je výsledek větší než $2^W - 1$, číslice vyšších řádů odřízne. Jinými slovy, počítá součet modulo 2^W.	4
$\alpha - \beta$	odečte od čísla α číslo β. Pokud je $\alpha < \beta$, spočte $2^W + \alpha - \beta$, čili rozdíl modulo 2^W.	4
$\neg\, \alpha$	spočte bitovou negaci čísla α, což je číslo, jehož i-tý bit je **0** právě tehdy, je-li i-tý bit čísla α roven **1**, a naopak.	9
$\alpha \wedge \beta$	bitové operace: *and, or* a *xor.* Vyhodnocují se tak, že se i-tý bit výsledku spočte z i-tého bitu	8
$\alpha \vee \beta$	čísla α a i-tého bitu čísla β podle následujících tabulek:	7
$\alpha \oplus \beta$		7

$$\begin{array}{lll} \mathbf{0} \wedge \mathbf{0} = \mathbf{0} & \mathbf{0} \vee \mathbf{0} = \mathbf{0} & \mathbf{0} \oplus \mathbf{0} = \mathbf{0} \\ \mathbf{0} \wedge \mathbf{1} = \mathbf{0} & \mathbf{0} \vee \mathbf{1} = \mathbf{1} & \mathbf{0} \oplus \mathbf{1} = \mathbf{1} \\ \mathbf{1} \wedge \mathbf{0} = \mathbf{0} & \mathbf{1} \vee \mathbf{0} = \mathbf{1} & \mathbf{1} \oplus \mathbf{0} = \mathbf{1} \\ \mathbf{1} \wedge \mathbf{1} = \mathbf{1} & \mathbf{1} \vee \mathbf{1} = \mathbf{1} & \mathbf{1} \oplus \mathbf{1} = \mathbf{0} \end{array}$$

$\alpha << \beta$	posune číslo α o β bitů doleva, čili doplní doprava β nul a odřízne prvních β bitů zleva, aby byl výsledek opět W-bitový.	2
$\alpha >> \beta$	posune číslo α o β bitů doprava, čili doplní doleva β nul a odřízne posledních β bitů vpravo, aby byl výsledek opět W-bitový.	2

Pokud závorky neurčí jinak, vyhodnocují se operátory s vyšší prioritou před operátory s nižší prioritou. V rámci stejné priority se pak vyhodnocuje zleva doprava (s výjimkou operátoru ¬, který je unární, a tudíž se musí vyhodnocovat zprava doleva).

Příklad 0: (jak fungují operátory; zde máme $W = 4$)

$a + b \wedge c + d = (a + (b \wedge c)) + d$ zde zafungují priority operátorů

$\mathbf{0101} + \mathbf{1110} = \mathbf{0011}$	nejvyšší bit výsledku **10011** se již oříznul
$\mathbf{0001} - \mathbf{1111} = \mathbf{0010}$	odčítáme modulo 16 = **10000**
$\mathbf{0101} \wedge \mathbf{0011} = \mathbf{0001}$	takto funguje *and*
$\mathbf{0101} \vee \mathbf{0011} = \mathbf{0111}$	takto *or*
$\mathbf{0101} \oplus \mathbf{0011} = \mathbf{0110}$	a takto *xor*
$(\mathbf{1} \ll \mathbf{11}) - \mathbf{1} = \mathbf{1000} - \mathbf{1} = \mathbf{0111}$	jak vyrobit pomocí $\ll$ posloupnost jedniček
$a \vee \neg\, a = \mathbf{1111}$	jak získat z čehokoliv samé jedničky

Výpočet probíhá takto: Nejprve se do registru x nastaví vstup (to je vždy jedno číslo) a do ostatních registrů nuly. Poté se provedou všechny příkazy v pořadí, v jakém jsou v programu uvedeny, přičemž vždy se nejprve vyhodnotí *výraz* na pravé straně a teprve poté se jeho výsledek uloží do *registru*, takže uvnitř výrazu je ještě možné pracovat s původní hodnotou registru. Po dokončení posledního příkazu se hodnota v registru y interpretuje jako výsledek výpočtu. Hodnoty v ostatních registrech mohou být libovolné.

Často budeme potřebovat, aby program mohl pracovat s většími čísly, než je číslo na vstupu, takže budeme rozlišovat velikost vstupu N (tj. počet bitů potřebných k zapisu vstupní hodnoty) a velikost W registrů a mezivýsledků, kterou si při psaní programu sami určíme. Pokud bychom ovšem povolili exponenciálně velká čísla (tedy $W = 2^N$), mohli bychom cokoliv spočíst v konstantním čase – stačilo by do jedné dlouhatánské konstanty uvedené v programu zakódovat všechny možné výsledky programu pro všechny hodnoty vstupu. Tak dlouhé registry lze však stěží považovat za realistické, proto přijmeme omezení, že W musí být polynomiální ve velikosti vstupu, čili že existuje konstanta k taková, že pro každé N je $W \leq N^k$.

Ne vždy si ovšem vystačíme s jedním programem, který funguje pro všechny velikosti vstupu – mnohdy potřebujeme podle N měnit hodnoty pomocných konstant v programu, někdy také nějakou operaci opakovat vícekrát v závislosti na velikosti vstupu. Povolíme si tedy programy zapisovat obecněji, a to tak, že uvedeme seznam pravidel, jež nám pro každé N vytvoří program, který počítá správně pro všechny vstupy velikosti N. [Formálně bychom tato pravidla mohli zavést třeba jako programy v nějakém klasickém programovacím jazyce. My si ale formalismus odpustíme a budeme je popisovat slovně.]

Při řešení úloh budeme chtít, aby časová složitost vygenerovaných programů, tedy jejich délka v závislosti na N, byla co nejmenší. Mezi stejně rychlými programy je pak lepší ten, který si vystačí s kratšími čísly, čili s menším W (to je analogie prostorové složitosti). Podobně jako u klasických programů ovšem budeme v obou případech přehlížet multiplikativní konstanty.

Příklad 1: Sestrojte program pro ALIK, který dostane na vstupu nenulové číslo a vrátí výsledek 1 právě tehdy, je-li toto číslo mocninou dvojky, jinak vrátí nulu.

Řešení: Nejdříve si všimněme, že mocniny dvojky jsou právě čísla, která obsahují právě jeden jedničkový bit. Sledujme chování následujícího jednoduchého programu.

Zmiňme ale ještě konvence, které budeme používat při psaní všech ukázkových programů: V levém sloupci naleznete jednotlivé příkazy, v pravém sloupci obecný tvar spočítané hodnoty pro libovolné N. Pokud se nějaká číslice nebo skupina číslic opakuje vícekrát, značíme opakování exponentem, tedy $\mathbf{0}^8$ je osm nul, $(\mathbf{01})^3$ je zkratka za **010101**. Řeckými písmeny značíme blíže neurčené skupiny bitů.

	$x = \alpha\mathbf{10}^i$
$a := x - 1$	$a = \alpha\mathbf{01}^i$
$b := x \wedge a$	$b = \alpha\mathbf{00}^i$

Číslo v registru a se od x vždy liší tím, že nejpravější **1** se změní na **0** a všechny **0** vpravo od ní se změní na **1**. Proto $b = x \wedge a$ se musí od x lišit právě přepsáním nejpravější **1** na **0**. (To proto, že bity vlevo od této **1** jsou stále stejné a $\alpha \wedge \alpha = \alpha$, zatímco ve zbytku čísla se vždy *and*uje **0** s **1**, což dá nulu.) A jelikož mocniny dvojky jsou právě čísla, v jejichž dvojkovém zápisu je právě jedna **1**, spočte náš program v b nulu právě tehdy, je-li x mocninou dvojky (nebo nulou, což jsme si ale zakázali).

Zbývá tedy vyřešit, jak z nuly udělat požadovanou jedničku a z nenuly nulu. K tomu si zavedeme operaci $r := \mathit{if}(s, t, u)$, která bude realizovat podmínku: pokud $s \neq 0$, přiřadí $r := t$, jinak $r := u$. Provedeme to jednoduchým trikem: rozšíříme si registry o jeden pomocný bit vlevo, nastavíme v r tento bit na jedničku a sledujeme, zda se zmenšením vzniklého čísla o jedničku tento bit změní na nulu nebo ne:

$v := s \vee \mathbf{10}^N$	$v = \mathbf{1}s$
$v := v - 1$	$v = \mathbf{1}s'$ (je-li $s \neq 0$), jinak $\mathbf{01}^N$
$v := v \wedge \mathbf{10}^N$	$v = \mathbf{10}^N$ nebo $\mathbf{00}^N$
$v := v \gg N$	$v = \mathbf{0}^N\mathbf{1}$ nebo $\mathbf{0}^N\mathbf{0}$
$v := v - 1$	$v = \mathbf{0}^{N+1}$ nebo $\mathbf{1}^{N+1}$
$r := (u \wedge v) \vee (t \wedge \neg\, v)$	$r = t$ nebo u

Stačí tedy na konec našeho programu přidat

$y := \mathit{if}(b, 0, 1)$	$y = \mathbf{0}$ nebo $\mathbf{1}$

a máme program, který rozpoznává mocniny dvojky v konstantním čase a používá k tomu čísla o $N + 1 = O(N)$ bitech.

Ještě si ukažme, jak bude probíhat výpočet pro dva konkrétní 8-bitové vstupy (tehdy je $N = 8$ a $W = 9$):

	$x = \mathbf{001011000}$	$x = \mathbf{000100000}$
$a := x - 1$	$a = \mathbf{001010111}$	$a = \mathbf{000011111}$

$b := x \wedge a$	$b = \mathbf{001010000}$	$b = \mathbf{000000000}$
$v := b \vee \mathbf{100000000}$	$v = \mathbf{101010000}$	$v = \mathbf{100000000}$
$v := v - 1$	$v = \mathbf{101001111}$	$v = \mathbf{011111111}$
$v := v \wedge \mathbf{100000000}$	$v = \mathbf{100000000}$	$v = \mathbf{000000000}$
$v := v >> 8$	$v = \mathbf{000000001}$	$v = \mathbf{000000000}$
$v := v - 1$	$v = \mathbf{000000000}$	$v = \mathbf{111111111}$
$y := (\mathbf{000000001} \wedge v) \vee (\mathbf{000000000} \wedge \neg\, v)$	$y = \mathbf{000000000}$	$y = \mathbf{000000001}$

Příklad 2: Sestrojte program pro Alik, který spočte *binární paritu* vstupního čísla, čili vrátí 0 nebo 1 podle toho, zda je v tomto čísle sudý nebo lichý počet jedničkových bitů.

Řešení: Binární parita $P(x)$ čísla $x = x_{N-1} \ldots x_1 x_0$ je podle definice rovna $x_0 \oplus x_1 \oplus \ldots \oplus x_{N-1}$. Jelikož operace $\oplus$ je asociativní ($\alpha \oplus (\beta \oplus \gamma) = (\alpha \oplus \beta) \oplus \gamma$) a komutativní ($\alpha \oplus \beta = \beta \oplus \alpha$), můžeme tento vztah pro $N = 2^k$ (to opět můžeme bez újmy na obecnosti předpokládat) přeuspořádat na

$$P(x) = (x_0 \oplus x_{N/2}) \oplus (x_1 \oplus x_{N/2+1}) \oplus \ldots \oplus (x_{N/2-1} \oplus x_{N-1}),$$

což je ovšem parita čísla vzniklého vyxorováním horní a dolní poloviny čísla x. Takže výpočet parity N-bitového čísla můžeme na konstantní počet příkazů převést na výpočet parity $N/2$-bitového čísla, ten zase na výpočet parity $N/4$-bitového čísla atd., až po $\log_2 N$ krocích na paritu 1-bitového čísla, která je ovšem rovna číslu samému.

Paritu tedy vypočteme na logaritmický počet příkazů pracujících s N-bitovými čísly takto:

$p := x >> N/2$	$p =$ horních N/2 bitů x
$q := x \wedge \mathbf{1}^{N/2}$	$q =$ dolních N/2 bitů x
$x := p \oplus q$	$x = N/2$-bitové číslo s paritou jako původní x
$x := (x >> N/4) \oplus (x \wedge \mathbf{1}^{N/4})$	$x = N/4$-bitové … (můžeme psát zkráceně)
…	
$x := (x >> 1) \oplus (x \wedge \mathbf{1})$	$x =$ 1-bitové …
$y := x$	$y = x$ (už jen zkopírovat výsledek)

Náš programovací jazyk samozřejmě $\mathbf{1}^{N/2}$ a podobné operace nemá, ale to vůbec nevadí, protože je vždy používáme jen na podvýrazy závisící pouze na N, takže je v programu můžeme pro každé N uvést jako konstanty. Například pro $N = 8$ bude výpočet probíhat takto:

	$x = \mathbf{00110110}$
$p := x >> 4$	$p = \cdots\cdot\mathbf{0011}$
$q := x \wedge \mathbf{1111}$	$q = \cdots\cdot\mathbf{0110}$
$x := p \oplus q$	$x = \cdots\cdot\mathbf{0101}$
$x := (x >> 2) \oplus (x \wedge \mathbf{11})$	$x = \cdots\cdot\mathbf{00}$
$x := (x >> 1) \oplus (x \wedge \mathbf{1})$	$x = \cdots\cdot\mathbf{0}$
$y := x$	$y = \mathbf{00000000}$

Soutěžní úlohy:

a) Sestrojte program pro Alik, jehož výsledkem bude počet jedničkových bitů ve dvojkovém zápisu čísla na vstupu.

b) Sestrojte program pro Alik, který k zadanému číslu x spočte nejbližší větší číslo, v jehož dvojkovém zápisu je stejný počet jedniček jako v zápisu x. Pokud takové číslo neexistuje, výsledek může být libovolný.