54. ročník Matematické olympiády – 2004/2005

Úlohy krajského kola kategorie P

Krajské kolo 54. ročníku MO kategorie P se koná v úterý 11. 1. 2005 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.

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).
- **Program.** V úlohách **P-II-1**, **P-II-2** a **P-II-3** je třeba uvést dostatečně podrobný zápis algoritmu, nejlépe ve tvaru zdrojového textu nejdůležitějších částí programu v jazyce Pascal nebo C. Nemusíte podrobně popisovat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů, vyhledávání v poli, třídění apod. V úloze **P-II-4** zapište navržený algoritmus ve formě programu v jazyce Alik.

Hodnotí se nejen správnost programu, ale také kvalita popisu řešení a efektivita zvoleného algoritmu.

Vzorová řešení úloh naleznete krátce po soutěži na Internetu na adrese *http://mo.mff.cuni.cz/*. Na stejném místě bude na konci února zveřejněn i seznam postupujících do celostátního kola. Naleznete zde také popis prostředí, v němž budete na celostátním kole řešit praktické úlohy.

P-II-1 Prádelní salón

Z Bořivoje se stal díky vaší pomoci úspěšný podnikatel a jeho klientela zahrnuje i bohatší a malichernější zákazníky. Pokud totiž nějaký zákazník uvidí dva různé zákazníky používat stejnou pračku, nebude už tuto prádelnu dále navštěvovat: „No považte, přece nelze prát prádlo s lidmi, kteří nemají na to, aby si zaplatili pračku sami pro sebe!“

Soutěžní úloha: Na vstupu dostanete $N \leq 10\,000$ – počet zákazníků, kteří navštíví Bořivojovu prádelnu během jednoho dne. U každého zákazníka je zadán čas jeho příchodu a doba, na jakou si chce pronajmout pračku (obojí jsou celá čísla mezi 1 a $1\,000\,000\,000$). Požadavky zákazníků nejsou uvedeny v žádném konkrétním pořadí.

Vaším úkolem je zjistit, kolik nejméně praček Bořivoj potřebuje, aby všichni jeho zákazníci byli zcela spokojeni. Zákazník bude spokojen, pokud si bude moci pronajmout pračku od okamžiku příchodu na dobu, kterou požaduje (je samozřejmé, že jednu pračku nemohou používat dva různí zákazníci současně), a navíc během doby, kdy bude prát, nebude žádnou pračku využívat více zákazníků po sobě.

Kromě určení minimálního počtu praček musíte pro Bořivoje vytvořit ještě seznam, podle kterého bude posílat zákazníky k volným pračkám.

Příklad: Pro 5 zákazníků, jejichž příchody a doby praní jsou

```
1000 1000
3000 2000
4500 500
1500 500
2000 2000
```

jsou potřeba alespoň 3 pračky a přiřazení praček zákazníkům například takové:

```
zákazník 1 bude u pračky 2
zákazník 2 bude u pračky 3
zákazník 3 bude u pračky 1
zákazník 4 bude u pračky 3
zákazník 5 bude u pračky 2
```

Všimněte si, že zákazníci 3 a 5 nemohou dostat stejnou pračku, protože by je viděl zákazník 2 pracovat u stejné pračky.

P-II-2 Zakázané rozdíly

Mějme dáno celé kladné číslo N, $N \geq 2$, a soustavu podmínek tvaru $x_i - x_j \neq a_{i,j}$, kde $x_1, \ldots, x_{N+1}$ jsou proměnné, $a_{i,j}$ jsou celá čísla mezi 0 a $N-1$ a pro každou dvojici indexů i a j, $1 \leq i < j \leq N+1$ soustava obsahuje právě jednu podmínku.

Soustavu budeme řešit modulo modulo zadané číslo N, tj. všechny aritmetické operace jsou prováděny modulo N. Připomeňme si, že výsledkem aritmetické operace provedené modulo N je zbytek po dělení původního výsledku číslem N, např. $(2+3) \bmod 4 = 1$, $(2-3) \bmod 4 = 3$, $(3 \cdot 2) \bmod 5 = 1$, $(3 \cdot 4) \bmod 6 = 0$, atd. Všimněte si zejména způsobu počítání, pokud je původní výsledek operace záporný.

Nalezněte algoritmus, který pro zadané N a čísla $a_{i,j}$ zjistí, zda zadaná soustava podmínek má řešení, tzn. zda existují čísla $x_1, \ldots, x_{N+1} \in \{0, \ldots, N-1\}$ taková, že rozdíl $x_j - x_i - a_{i,j}$ není dělitelný N pro žádné i a j, $1 \leq i < j \leq N+1$. Pokud má soustava řešení, algoritmus musí také libovolné její řešení nalézt a vypsat.

Příklad 1: Pro $N = 3$, máme zadány následující podmínky:

$$\begin{aligned} x_1 - x_2 &\neq 1 \\ x_1 - x_3 &\neq 2 \\ x_1 - x_4 &\neq 2 \\ x_2 - x_3 &\neq 2 \\ x_2 - x_4 &\neq 1 \\ x_3 - x_4 &\neq 0 \end{aligned}$$

Soustava má řešení, např. $x_1 = x_2 = x_4 = 2$ a $x_3 = 1$.

Příklad 2: Pro $N = 2$, máme zadány následující podmínky:

$$\begin{aligned} x_1 - x_2 &\neq 1 \\ x_1 - x_3 &\neq 0 \\ x_2 - x_3 &\neq 1 \end{aligned}$$

Pokud $x_1 = 0$, pak $x_2 = 0$ podle první podmínky a $x_3 = 1$ podle druhé podmínky. Potom ale třetí podmínka není splněna. Podobně pokud $x_1 = 1$, x_2 musí být rovno 0 a x_3 rovno 1 a třetí podmínka opět není splněna. Zadaná soustava podmínek tedy nemá řešení.

P-II-3 Redundantní redundance

Úřad pro potírání redundantních repetic (zřízený Komisí pro likvidaci redundantních úřadů) se zabývá odstraňováním zbytečně opakovaných dokumentů v archivech. Procházení archivů je samozřejmě velmi nudná práce, která odvádí úředníky od jiných, mnohem zajímavějších a jistě i prospěšnějších využití jejich pracovního času. Velmi by je proto potěšilo, kdybyste pro ně napsali program řešící následující úlohu:

Je dáno přirozené číslo k a nějaký znakový řetězec T. Určete souvislý podřetězec délky k, který se v T nejvíce opakuje, a také počet jeho výskytů R. Jednotlivé výskyty tohoto řetězce se mohou částečně překrývat. V případě, že existuje více řetězců, které se opakují R-krát, vypište jeden libovolný z nich.

Příklad: Pro vstup `abababa` a $k = 3$ je nejčastějším řetězcem `aba` opakující se 3-krát.

P-II-4 ALÍK

Definici stroje Alik naleznete ve studijním textu za touto úlohou. Od domácího kola se liší tím, že přibyly operace násobení, dělení a zbytku po dělení a Příklad 3 na tyto operace.

Soutěžní úloha:

Sestrojte program pro Alik, který k zadanému číslu $x = x_{N-1} \ldots x_1 x_0$ nalezne zrcadlové číslo $y = x_0 x_1 \ldots x_{N-1}$, tj. číslo, jehož dvojkový zápis vznikne zapsáním N-bitového dvojkového zápisu čísla x (včetně případných počátečních nul) pozpátku.

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
$\alpha * \beta$	vynásobí dvě čísla, výsledek opět modulo 2^W.	6
$\alpha \,/\, \beta$	vydělí číslo α číslem β; dělení nulou dá vždy výsledek 0.	6
$\alpha \,\%\, \beta$	vrátí zbytek po dělení čísla α číslem β, čili $\alpha - \beta * (\alpha \,/\, \beta)$; pokud je $\beta = 0$, je výsledek roven α.	6
$\neg\, \alpha$	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 $\neg$, 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*
$(\mathbf{1} << \mathbf{11}) - \mathbf{1} = \mathbf{1000} - \mathbf{1} = \mathbf{0111}$	jak vyrobit pomocí << 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 $\mathbf{00}^N$
$v := v >> N$	$v = \mathbf{0}^N\mathbf{1}$ 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

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

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

Příklad 3: Ve vzorovém řešení úlohy P-I-4 b) jsme potřebovali přesunout posloupnost jedniček na konec čísla, tedy číslo tvaru $\mathbf{0}^i\mathbf{1}^j\mathbf{0}^k$ převést na $\mathbf{0}^i\mathbf{0}^k\mathbf{1}^j$. To je pomocí dělení možné provést v konstantním čase třeba takto:

	$x = \mathbf{0}^i\mathbf{1}^j\mathbf{0}^k$
$a := x \wedge (x - 1)$	$a = \mathbf{0}^i\mathbf{1}^{j-1}\mathbf{00}^k$ (viz Příklad 1)
$b := x \oplus a$	$b = \mathbf{0}^i\mathbf{0}^{j-1}\mathbf{10}^k$
$y := x \,/\, b$	$b = \mathbf{0}^i\mathbf{0}^k\mathbf{1}^j$

Zde jsme využili toho, že dělení mocninou dvojky je možné použít jako bitový posun doprava, ovšem zadaný místo počtu bitů, o které se má posouvat, číslem majícím **1** na pozici, která se má po posunu objevit úplně vpravo.