MATEMATICKÁ OLYMPIÁDA NA STREDNÝCH ŠKOLÁCH

54. ročník, školský rok 2004/2005

Zadania úloh 2. kola kategórie P

Druhé kolo 54. ročníka MO kategórie P sa koná 11. januára 2005 v dopoludňajších hodinách. Na riešenie úloh máte 4 hodiny čistého času. Riešenie každého príkladu musí obsahovať (pokiaľ nie je v zadaní uvedené ináč):

- **Popis riešenia,** to znamená slovný popis použitého algoritmu, argumenty zdôvodňujúce jeho správnosť (prípadne dôkaz správnosti algoritmu), diskusiu o efektivite vášho riešenia (časová a pamäťová zložitosť). Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného zápisu algoritmu (do programu).

- **Program.** V úlohách **P-II-1**, **P-II-2** a **P-II-3** treba uviesť dostatočne podrobný zápis algoritmu, najlepšie v tvare zdrojového textu najdôležitejších častí programu v jazyku Pascal alebo C/C++. Zo zápisu môžete vynechať jednoduché operácie ako vstupy, výstupy, implementáciu jednoduchých matematických vzťahov a pod.

  Triedenie už **nie je** jednoduchá operácia, preto ak píšete program v Pascale a používate triedenie, nezabudnite stručne uviesť aj jeho algoritmus. V jazyku C/C++ môžete namiesto toho použiť knižničné funkcie na triedenie, v takom prípade stačí uviesť porovnávaciu funkciu.

  Ak píšete program v C++ a používate STL, súčasťou popisu vášho algoritmu by mal byť dostatočne podrobný popis implementácie dátových štruktúr, ktoré používate.

  V úlohe **P-II-4** je potrebné uviesť program pre ALIK.

Hodnotí se nielen správnosť programu, ale tiež kvalita popisu riešenia a efektivita zvoleného algoritmu.

P-II-1

Z Kleofáša sa stal vďaka vašej pomoci úspešný podnikateľ a jeho klientela zahŕňa bohatších a malichernejších zákaznikov. Ak totiž nejaký zo súčasných zákazníkov uvidí dvoch rôznych zákaznikov používať rovnakú práčku, nebude už túto práčovňu ďalej navštevovať – povie si: „Pochopte predsa, že nebudem prať bielizeň s ľuďmi, ktorí nemajú na to, aby si zaplatili práčku sami pre seba!“

Súťažná úloha: Na vstupe dostanete $N \leq 10\,000$ – počet zákazníkov, ktorí navštívia Kleofášovu práčovňu počas jedného dňa. Pre každého zákazníka je zadaný čas jeho príchodu a doba, na akú si chce prenajať práčku (obidve sú celé čísla medzi 1 a $1\,000\,000\,000$). Požiadavky zákazníkov nie sú uvedené v žiadnom konkrétnom poradí.

Vašou úlohou je zistiť, koľko najmenej práčok Kleofáš potrebuje, aby všetci jeho zákazníci boli úplne spokojní. Zákazník bude spokojný, ak si bude môcť prenajať práčku od okamihu príchodu na dobu, ktorú požaduje (je samozrejmé, že jednu práčku nemôžu používať dvaja rôzni zákazníci súčasne), a naviac počas doby, kedy bude prať, nebude žiadnu práčku postupne využívať viac zákazníkov.

Okrem určenia minimálneho počtu práčok musíte pre Kleofáša vytvoriť aj zoznam, podľa ktorého bude posielať zákazníkov k voľným práčkam.

Príklad:

vstup	**výstup**
5 zákazníkov	Treba aspoň 3 práčky.
časy príchodu a doby:	Priradenie práčok zákazníkom:
`1000 1000`	– zákazník 1 bude pri práčke 2
`3000 2000`	– zákazník 2 bude pri práčke 3
`4500 500`	– zákazník 3 bude pri práčke 1
`1500 500`	– zákazník 4 bude pri práčke 3
`2000 2000`	– zákazník 5 bude pri práčke 2

(Všimnite si, že zákazníci 3 a 5 nemôžu dostať tú istú práčku, pretože by ich videl zákazník 2 pracovať pri rovnakej práčke.)

P-II-2

Majme dané celé kladné číslo N, $N \geq 2$, a sústavu podmienok tvaru $x_i - x_j \neq a_{i,j}$, kde $x_1, \ldots, x_{N+1}$ sú premenné, $a_{i,j}$ sú celé čísla medzi 0 a $N-1$ a pre každú dvojicu indexov i a j, $1 \leq i < j \leq N+1$ sústava obsahuje práve jednu podmienku.

Aby nám odčítaním nemohli vzniknúť na ľavej strane niektorých podmienok záporné čísla, budeme túto sústavu riešiť modulo zadané číslo N.

Teda všetky aritmetické operácie sú vykonané modulo N. Pripomeňme si, že výsledkom aritmetickej operácie vykonanej modulo N je zvyšok po delení pôvodného výsledku číslom N, napríklad $(2 + 3) \bmod 4 = 1$, $(2 - 3) \bmod 4 = 3$, $(3 \cdot 2) \bmod 5 = 1$, $(3 \cdot 4) \bmod 6 = 0$, atď. Všimnite si hlavne spôsob počítania v prípade, že je pôvodný výsledok záporný.

Nájdite algoritmus, ktorý pre zadané N a čísla $a_{i,j}$ zistí, či zadaná sústava podmienok má riešenie – teda či existujú nezáporné celé čísla $x_1, \ldots, x_{N+1}$ také, že rozdiel $x_j - x_i - a_{i,j}$ nie je deliteľný N pre žiadne i a j, $1 \leq i < j \leq N + 1$. Ak sústava má riešenie, algoritmus musí tiež (jedno ľubovoľné) jej riešenie nájsť a vypísať.

Príklad 1: Pre $N = 3$ máme zadané nasledujúce podmienky:

$x_1 - x_2 \neq 1$ $\quad$ $x_2 - x_3 \neq 2$ $\quad$ $x_3 - x_4 \neq 0$
$x_1 - x_3 \neq 2$ $\quad$ $x_2 - x_4 \neq 1$
$x_1 - x_4 \neq 2$

Sústava má riešenie, napr. $x_1 = x_2 = x_4 = 2$ a $x_3 = 1$.

Príklad 2: Pre $N = 2$ máme zadané nasledujúce podmienky:

$x_1 - x_2 \neq 1$ $\quad$ $x_2 - x_3 \neq 1$
$x_1 - x_3 \neq 0$

Ak $x_1 = 0$, tak potom $x_2 = 0$ podľa prvej podmienky a $x_3 = 1$ podľa druhej podmienky. Potom ale tretia podmienka nie je splnená. Podobne, ak $x_1 = 1$, x_2 musí byť rovné 0 a x_3 rovné 1. Tretia podmienka ale opäť nie je splnená. Zadaná sústava podmienok teda nemá riešenie.

P-II-3

Úrad pre potláčanie redundantných repetícií (zriadený Komisiou pre likvidáciu redundantných úradov) sa zaoberá odstraňovaním zbytočne opakovaných dokumentov v archívoch.

Prechádzanie archívov je samozrejme veľmi nudná práca, ktorá odvádza úradníkov od iných, omnoho zaujímavejších a iste prospešnejších využití ich pracovného času. Veľmi by ich preto potešilo, keby ste im napísali program riešiaci nasledovnú úlohu:

Súťažná úloha: Je dané prirodzené číslo k a reťazec znakov T. Nájdite súvislý podreťazec dĺžky k, ktorý sa v T najviackrát opakuje, a tiež počet jeho výskytov R. Jednotlivé výskyty tohto reťazca sa môžu čiastočne prekrývať. V prípade, že existuje viac reťazcov, ktoré sa opakujú R-krát, vypíšte ľubovoľný z nich.

Príklad:

Pre $T =$ `abababa` a $k = 3$ je riešením podreťazec `aba`, opakuje sa 3-krát.

P-II-4

Popis zariadenia ALIK nájdete v študijnom texte za touto úlohou. Od domáceho kola sa líši tým, že pribudli operácie násobenia, delenia a zvyšku po delení a Príklad 3 na tieto operácie.

Súťažná úloha

Zostrojte program pre ALIK, ktorý k zadanému číslu $x = x_{N-1} \ldots x_1 x_0$ nájde zrkadlové číslo $y = x_0 x_1 \ldots x_{N-1}$, t. j. číslo, ktorého dvojkový zápis vznikne zapísaním celého čísla x (vrátane prípadných núl na jeho začiatku) odzadu.

Študijný text

Aritmeticko-logická integerová kalkulačka (skratka ALIK) je výpočtové zariadenie pracujúce s W-bitovými celými číslami v rozsahu 0 až $2^W - 1$ vrátane; v ďalšom texte pod *číslami* rozumieme vždy takéto čísla. Budeme ich obvykle zapisovať v dvojkovej sústave hrubým písmom a na začiatok dvojkového zápisu vždy doplníme príslušný počet núl, aby počet číslic (bitov) bol presne W. Väčšinou tiež nebudeme rozlišovať mezi číslom a jeho dvojkovým zápisom, takže i-tym bitom čísla budeme rozumieť i-ty bit jeho dvojkového zápisu (bity číslujeme zprava doľava od 0 po $W - 1$).

Pamäť stroja tvorí 26 *registrov* pomenovaných a až z. Každý register vždy obsahuje jedno číslo.

ALIK se riadi programom, čo je postupnosť priraďovacích príkazov typu *register* := *výraz*, kde *výraz* môže obsahovať konštanty (čísla zapísané v dvojkovej sústave), registre, zátvorky a nasledujúce operátory (grécke písmená označujú podvýrazy):

- $\boxed{\alpha + \beta \text{ (priorita 4)}}$ sčíta čísla α a β. Ak je výsledok viac ako $2^W - 1$, číslice vyšších rádov odreže. Inak povedané, počíta súčet modulo 2^W.
- $\boxed{\alpha - \beta \text{ (priorita 4)}}$ odčíta od čísla α číslo β. Ak je $\alpha < \beta$, spočíta $2^W + \alpha - \beta$, teda rozdiel modulo 2^W.
- $\boxed{\alpha * \beta \text{ (priorita 6)}}$ vynásobí dve čísla, výsledok opäť modulo 2^W.
- $\boxed{\alpha \,/\, \beta \text{ (priorita 6)}}$ vydelí číslo α číslom β; delenie nulou dá vždy výsledok 0.
- $\boxed{\alpha \,\%\, \beta \text{ (priorita 6)}}$ vráti zvyšok po delení čísla α číslom β, teda $\alpha - \beta * (\alpha \,/\, \beta)$; ak $\beta = 0$, výsledok sa rovná α.
- $\boxed{\neg\, \alpha \text{ (priorita 9)}}$ spočíta bitovú negáciu čísla α, čo je číslo, ktorého i-ty bit je **0** práve vtedy, keď i-ty bit čísla α sa rovná **1**, a naopak.

- $\alpha \wedge \beta$ (priorita 8), $\alpha \vee \beta$, $\alpha \oplus \beta$ (priorita 7) bitové operácie: *and, or* a *xor.* Vyhodnocujú sa tak, že sa i-ty bit výsledku spočíta z i-teho bitu čísla α a i-teho bitu čísla β podľa nasledujúcich tabuliek:

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

- $\alpha << \beta$ (priorita 2) posunie číslo α o β bitov doľava, teda doplní na jeho koniec β núl a odreže prvých β bitov, aby bol výsledok opäť W-bitový.
- $\alpha >> \beta$ (priorita 2) posunie číslo α o β bitov doprava, teda doplní na jeho začiatok β núl a odreže posledných β bitov, aby bol výsledok opäť W-bitový.

Ak zátvorky neurčia inak, vyhodnocujú sa operátory s vyššiou prioritou pred operátormi s nižšou prioritou. V rámci rovnakej priority sa vyhodnocuje zľava doprava (s výnimkou operátora $\neg$, ktorý je unárny, a teda sa musí vyhodnocovať zprava doľava).

Príklad: ako fungujú operátory; predpokladáme $W = 4$

$$a + b \wedge c + d = (a + (b \wedge c)) + d$$
$$\mathbf{0101} + \mathbf{1110} = \mathbf{0011}$$
$$\mathbf{0001} - \mathbf{1111} = \mathbf{0010}$$
$$\mathbf{0101} \wedge \mathbf{0011} = \mathbf{0001}$$
$$\mathbf{0101} \vee \mathbf{0011} = \mathbf{0111}$$
$$\mathbf{0101} \oplus \mathbf{0011} = \mathbf{0110}$$

Ako vyrobiť pomocou $<<$ postupnosť jednotiek:

$$(\mathbf{1} << \mathbf{11}) - \mathbf{1} = \mathbf{1000} - \mathbf{1} = \mathbf{0111}$$

Ako získať z čohokoľvek samé jednotky:

$$a \vee \neg\, a = \mathbf{1111}$$

Výpočet prebieha takto: Najprv sa do registra x nastaví vstup (vždy jedno číslo) a do ostatných registrov nuly. Potom sa vykonajú všetky príkazy *register* := *výraz* v poradí, v akom sú v programe uvedené, pričom sa vždy najskôr vyhodnotí *výraz* na pravej strane a až potom sa jeho výsledok uloží do *registra*, takže vo vnútri výrazu je ešte možné pracovať s pôvodnou hodnotou registra. Po dokončení posledného príkazu sa hodnota v registri y interpretuje ako výsledok výpočtu. Hodnoty v ostatných registroch môžu byť ľubovoľné.

Veľkosť vstupu zadefinujeme ako počet bitov N, ktoré potrebujeme na reprezentáciu vstupu. Naše programy budú často potrebovať registre, do ktorých možno uložiť čísla väčšie ako $2^N - 1$. Počet bitov, ktoré potrebujeme na reprezentáciu týchto čísel budeme volať *veľkosť registra* W. Samozrejme, vždy musí platiť $W \geq N$.

Nie vždy je možné použiť ten istý program pre všetky možné veľkosti vstupu N. V závisloti od N a W je napríklad potrebné zmeniť hodnoty pomocných konštánt v programe, počet opakovaní niektorých operácií v programe a podobne. Preto pri riešení úloh pre každú hodnotu N musíme popísať:

- veľkosť registrov W, ktorú program bude používať, v závislosti od N
- ako pre každú konkrétnu hodnotu N zostaviť program, ktorý funguje pre všetky vstupy veľkosti N

Časovou zložitosťou programu v závislosti od N budeme rozumieť počet inštrukcií, ktoré potrebujeme pre danú veľkosť vstupu. Veľkosť registra W v závislosti od N budeme nazývať *pamäťovou zložitosťou* (keďže počet registrov je obmedzený na 26, veľkosť registra skutočne určuje množstvo pamäte). Tak ako pri časovej a pamäťovej zložitosti bežných programov, pri odhadoch budeme zanedbávať multiplikatívne konštanty (môžeme teda používať O-notáciu). Budeme však vyžadovať, aby **veľkosť registra W bola polynomiálne závislá od veľkosti vstupu** N (t.j. existuje konštanta k, pre ktorú $W \leq N^k$ pre všetky $N \geq 2$).

Pri riešení úloh budeme chcieť, aby časová zložitosť vygenerovaných programov v závislosti na N bola čo najmenšia. Medzi rovnako rýchlymi programami je potom lepší ten s menšou pamäťovou zložitosťou.

Príklad 1: Zostrojte program pre ALIK, ktorý dostane na vstupe nenulové číslo a vráti výsledok 1 práve vtedy, ak je toto číslo mocninou dvojky, inak vráti nulu.

Riešenie: Najprv si všimnime, že mocniny dvojky sú práve čísla, ktoré obsahujú práve jeden jednotkový bit. Sledujme chovanie nasledujúceho jednoduchého programu.

Vo všetkých ukážkových programoch budeme v ľavom stĺpci uvádzať jednotlivé príkazy a v pravom stĺpci všeobecný tvar spočítanej hodnoty pre ľubovoľné N. Ak sa nejaká číslica alebo skupina číslic opakuje viackrát, označíme opakovanie exponentom. Teda $\mathbf{0}^8$ je osem núl, $(\mathbf{01})^3$ je skratka za $\mathbf{010101}$. Gréckymi písmenami budeme označovať bližšie neurčené skupiny bitov.

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

Číslo v registri a sa od x vždy líši tým, že najpravejšia **1** sa zmení na **0** a všetky **0** vpravo od nej sa zmenia na **1**. Preto $b = x \wedge a$ sa musí od x líšiť práve prepísaním najpravejšej **1** na **0**. (To preto, že bity naľavo od tejto **1** sú stále rovnaké a $\alpha \wedge \alpha = \alpha$, kým vo zvyšku čísla sa vždy *and*uje **0** s **1**, čo dá nulu.) A keďže mocniny dvojky sú práve čísla, v ktorých dvojkovom zápise je práve jedna **1**, spočíta náš program v b nulu práve vtedy, ak je x mocnina dvojky (alebo nula, čo sme ale zakázali).

Zostáva teda vyriešiť, ako z nuly spraviť požadovanú jednotku a z nenuly nulu. K tomu si zavedieme operáciu $r := \mathit{if}(s, t, u)$, ktorá bude realizovať podmienku: ak $s \neq 0$, priradí $r := t$, inak $r := u$. Spravíme to jednoduchým trikom: rozšíríme si registre o jeden pomocný bit vľavo, nastavíme v r tento bit na jednotku a sledujeme, či sa zmenšením vzniknutého čísla o jednotku tento bit zmení na nulu alebo nie:

$v := s \vee \mathbf{10}^N$	$v = \mathbf{1}r$
$v := v - 1$	$v = \mathbf{1}r'$ (ak $r \neq 0$), inak $\mathbf{01}^N$
$v := v \wedge \mathbf{10}^N$	$v = \mathbf{10}^N$ alebo $\mathbf{00}^N$
$v := v >> N$	$v = \mathbf{0}^N\mathbf{1}$ alebo $\mathbf{0}^N\mathbf{0}$
$v := v - 1$	$v = \mathbf{0}^{N+1}$ alebo $\mathbf{1}^{N+1}$
$r := (u \wedge v) \vee (t \wedge \neg\, v)$	$s = t$ alebo u

Stačí teda na koniec programu pridať

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

a máme program, ktorý rozpoznáva mocniny dvojky v konštantnom čase a používa na to čísla s $N + 1 = O(N)$ bitmi.

Ešte si ukážme, ako bude prebiehať výpočet pre dva konkrétne 8-bitové vstupy (teda $N = 8$ a $W = 9$):

	$x = \mathbf{001011000}$	$x = \mathbf{000100000}$
$a := x - 1$	$a = \mathbf{001010111}$	$a = \mathbf{000011111}$
$b := x \wedge a$	$b = \mathbf{001010000}$	$b = \mathbf{000000000}$
$v := b \vee \mathbf{100000000}$	$v = \mathbf{101010000}$	$v = \mathbf{100000000}$
$v := v - 1$	$v = \mathbf{101001111}$	$v = \mathbf{011111111}$
$v := v \wedge \mathbf{100000000}$	$v = \mathbf{100000000}$	$v = \mathbf{000000000}$
$v := v >> 8$	$v = \mathbf{000000001}$	$v = \mathbf{000000000}$
$v := v - 1$	$v = \mathbf{000000000}$	$v = \mathbf{111111111}$
$y := (\mathbf{000000001} \wedge v)$ $\vee(\mathbf{000000000} \wedge \neg\, v)$	$y = \mathbf{000000000}$	$y = \mathbf{000000001}$

Príklad 2: Zostrojte program pre ALIK, ktorý spočíta *binárnu paritu* vstupného čísla, teda vráti 0 nebo 1 podľa toho, či má toto číslo párny alebo nepárny počet jednotkových bitov.

Riešenie: Bez ujmy na všeobecnosti môžeme predpokladať, že N je mocnina dvoch. Binárna parita $P(x)$ čísla $x = x_{N-1} \ldots x_1 x_0$ sa podľa definície rovná $x_0 \oplus x_1 \oplus \ldots \oplus x_{N-1}$. Keďže operácia $\oplus$ je asociatívna ($\alpha \oplus (\beta \oplus \gamma) = (\alpha \oplus \beta) \oplus \gamma$) a komutatívna ($\alpha \oplus \beta = \beta \oplus \alpha$), môžeme tento vzťah preusporiadať na

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

čo je ale parita čísla, ktoré vznikne vyxorovaním ľavej a pravej polovice čísla x. Takže výpočet parity N-bitového čísla môžeme konštantným počtom príkazov previesť na výpočet parity $N/2$-bitového čísla, ten zase na výpočet parity $N/4$-bitového čísla atď., až po $\log_2 N$ krokoch na paritu 1-bitového čísla, ktorá sa ovšem rovná číslu samotnému.

Paritu teda spočítame nasledujúcim programom. Jeho časová zložitosť tohto programu je $O(\log N)$ krokov, pamäťová zložitosť je $O(N)$ bitov.

$p := x >> N/2$	$p =$ horných $N/2$ bitov x
$q := x \wedge \mathbf{1}^{N/2}$	$q =$ dolných $N/2$ bitov x
$x := p \oplus q$	$x = N/2$-bitové číslo s paritou ako pôvodné x
$x := (x >> N/4) \oplus (x \wedge \mathbf{1}^{N/4})$	$x = N/4$-bitové číslo s rovnakou paritou (všimnite si skrátený zápis výrazu)
...	...
$x := (x >> 1) \oplus (x \wedge \mathbf{1})$	$x =$ 1-bitové číslo
$y := x$	$y = x$ (skopírovať výsledok)

Náš programovací jazyk samozrejme žiadne celé časti čísel a podobné operácie nemá, ale to vôbec nevadí, pretože ich vždy používame len na podvýrazy závisiace iba na N, takže ich v programe môžeme pre každé N uviesť ako konštanty. Napríklad pre $N = 8$ bude výpočet prebiehať takto:

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

Príklad 3: Vo vzorovom riešení úlohy P-I-4 b) sme potrebovali presunúť postupnosť jednotiek na koniec čísla, teda číslo tvaru $\mathbf{0}^i\mathbf{1}^j\mathbf{0}^k$ previesť na $\mathbf{0}^i\mathbf{0}^j\mathbf{1}^k$. To je pomocou delenia možné spraviť v konštantnom čase napríklad takto:

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

Tu sme využili to, že delenie mocninou dvojky je možné použiť ako bitový posun doprava, ale namiesto počtu bitov, o ktoré sa má posúvať, zadáme číslo majúce **1** na pozícii, ktorá sa má po posune objaviť úplne vpravo.

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY

54. ROČNÍK MATEMATICKEJ OLYMPIÁDY

Zadania 2. kola kategórie P

Vydala IUVENTA pre vnútornú potrebu Ministerstva školstva SR
Zodpovedný redaktor: M. Forišek
Sadzba programom LaTeX

© Slovenská komisia Matematickej olympiády, 2004