MATEMATICKÁ OLYMPIÁDA NA STREDNÝCH ŠKOLÁCH

54. ročník, školský rok 2004/2005

Zadania úloh 3. kola kategórie P

1. súťažný deň

Na riešenie úloh máte 4.5 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-III-1** a **P-III-2** 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.

  Ak vo vašom riešení využívate netriviálne dátové štruktúry alebo algoritmy, ktoré sú obsiahnuté v knižniciach k vášmu programovaciemu jazyku (napr. STL pre C++), súčasťou riešenia by mal byť dostatočne podrobný popis implementácie použitých dátových štruktúr a algoritmov.

  V úlohe **P-III-3** uveďte príslušný program pre Alik-a.

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

P-III-1

Pánko Lektor je vášnivý zberateľ náhrdelníkov. Náhrdelníky, ktoré zbiera, sa navzájom líšia počtom, poradím a druhmi použitých drahých kameňov. Ako každý zberateľ, ani Lektor nechce vyhadzovať peniaze za náhrdelníky, ktoré už v zbierke má. Vymyslel si preto nasledujúce kódovanie: Každému druhu drahokamov priradil jedno písmeno abecedy. Náhrdelník teraz zapísal tak, že začal od niektorého drahokamu a napísal si kódy všetkých drahokamov v poradí, v akom sa na náhrdelníku nachádzali. Navyše si kúpil stroj, ktorý mu pre daný kód povie, či už taký náhrdelník má v zbierke alebo nie.

Klenotníci si však rýchlo všimli slabinu v jeho postupe – keď predávaný náhrdelník pootočili, prípadne ho obrátili hore nohami, jeho kód sa tým zmenil. Takto si pánko Lektor nakúpil zopár duplikátov – napríklad náhrdelník `ABCA` si kúpil aj ako `AABC` a `ACBA`.

Niet divu, že by chcel svoj stroj vylepšiť tak, aby rozoznal aj takéto situácie. Vašou úlohou bude napísať program, ktorý to dokáže.

Súťažná úloha:

Program dostane na vstupe niekoľko kódov náhrdelníkov $x_1, \ldots, x_N$ ($1 \leq N \leq 1\,000\,000$). O každom z nich by mal vypísať, či ho má pánko Lektor kúpiť alebo nie – podľa toho, či už predtým rovnaký náhrdelník kúpil. Ak je náhrdelník x_i rôzny od každého spomedzi náhrdelníkov $x_1, \ldots, x_{i-1}$ (vrátane ich rotácii a preklápania), mal by byť i-ty riadok výstupu `Kup ho!`, inak by mal byť i-ty riadok výstupu `Ten uz mas.`

Váš program môže využívať Lektorov starý stroj ako *čiernu skrinku*, ktorá si vie pamätať množinu kódov náhrdelníkov. Na začiatku behu programu je táto množina prázdna. Vo svojom programe môžete volať funkcie `Pridaj` a `MamHo`. Prvá z nich pridá nový kód náhrdelníka do pamätanej množiny, druhá vráti `true` alebo `false` (v C/C++ 1 alebo 0) podľa toho, či sme predtým niekedy zavolali `Pridaj` s daným kódom náhrdelníka.

Hlavičky pomocných funkcií:

```
int MamHo (const char *kod);   /* v C */
void Pridaj (const char *kod);
function MamHo (var kod:string):boolean;  { v Pascale }
procedure Pridaj (var kod:string);
```

Hodnotenie riešení:

Pánko Lektor má nasledujúce požiadavky: Pamäťová zložitosť vášho programu **nesmie** závisieť od počtu testovaných náhrdelníkov. Časová zložitosť vášho programu by mala byť čo najnižšia.

Volania funkcií `Pridaj` a `MamHo` majú časovú zložitosť lineárnu od dĺžky reťazca `kod`.

Spomedzi dvoch programov s rovnakou časovou zložitosťou je lepší ten, ktorý potrebuje menej volaní funkcií `Pridaj` a `MamHo`.

Príklad:

vstup	**výstup**
`ABCA`	`Kup ho!`
`ACBA`	`Ten uz mas.`
`AABC`	`Ten uz mas.`
`ABAC`	`Kup ho!`

P-III-2

Nedávno bolo otvorené nové Magické Observatórium v Polomokrej chňapke (MO-P). Podnebie v Polomokrej chňapke je také polomokré, preto sa vedenie MO-P rozhodlo, že všetky svoje budovy prepojí krytými nadzemnými mostami. A aby zostalo peňazí aj na iné účely, bolo by vhodné, aby celková dĺžka týchto mostov bola najmenšia možná. Navyše kvôli častým obdobiam, kedy tam fúka silný severák, musia všetky mosty viesť zo severu na juh.

Súťažná úloha:

Vašou úlohou je navrhnúť algoritmus, ktorý pre zadanú mapu areálu MO-P navrhne najlepšie možné prepojenie budov. Ak sa budovy pospájať nedajú, program by to mal zistiť a podať o tom správu.

Mapa areálu MO-P je zadaná ako štvorcová sieť s N riadkami a M stĺpcami. Budovy sú vyznačené znakmi `x`, okolité močiare bodkami. Ak dve políčka s písmenom `x` majú spoločnú hranu, patria do tej istej budovy a pracovníci MO-P medzi nimi môžu prejsť. Vašou úlohou je dokresliť do mapy čo najmenej severojužných (t.j. zvislých) mostov tak, aby sa medzi každými dvomi budovami dalo prejsť bez stúpenia do močiara. Políčka mostov sa smú používať len v severojužnom smere, t.j. prísť naň aj odísť z neho smieme len v týchto smeroch.

Pri riešení tejto úlohy (a zároveň pri jeho popise) sa sústreďte na efektívne nájdenie optimálneho prepojenia budov a na zdôvodnenie správnosti daného algoritmu.

Príklad 1:

vstup	**výstup**
$M = 8$, $N = 5$	`..xxxxx.`
`..xxxxx.`	`...|..|x`
`.......x`	`...x..|x`
`...x...x`	`.x.x..||`
`.x.x....`	`.xxx..xx`
`.xxx..xx`	

Vysvetlenie príkladu: Areál je tvorený štyrmi budovami (znaky `x` v pravom stĺpci patria do inej budovy ako znaky `x` v prvom riadku). Keďže mosty môžeme používať len v severojužnom smere, je nutné postaviť aj most v pravom stĺpci, bez neho by jedna budova nebola pripojená.

Príklad 2:

vstup	**výstup**
$M = 6$, $N = 5$	`xxxxxx`
`xxxxxx`	`x.|..x`
`x....x`	`x.xx.x`
`x.xx.x`	`x....x`
`x....x`	`xxxxxx`
`xxxxxx`	

Príklad 3:

vstup	**výstup**
$M = 5$, $N = 4$	`Areal sa neda pospajat.`
`xxx..`	
`.....`	
`xxx..`	
`...xx`	

P-III-3

Napíšte program pre Alik-a, ktorý vypočíta dvojkový logaritmus zadaného nenulového čísla x – teda pozíciu najľavejšej jednotky, keď x zapíšeme v dvojkovej sústave. (Pozície bitov rastú sprava doľava, napravejší bit je na pozícii 0.)

Príklad:

Dvojkový logaritmus dvojkového čísla **00110001** je 5, dvojkový logaritmus z **00000001** je 0.

Študijný text

(Študijný text je rovnaký ako v krajskom kole, len vysvetlenia pri príkladoch boli z typografických príčin skrátené.)

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.
- $\boxed{\alpha \wedge \beta \text{ (priorita 8)}}$, $\alpha \vee \beta$, $\alpha \oplus \beta$ (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:

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

- $\boxed{\alpha << \beta \text{ (priorita 2)}}$ 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.
- $\boxed{\alpha >> \beta \text{ (priorita 2)}}$ 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:

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.

Zavedieme operáciu $r := if(s,t,u)$, 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

Potom nasledujúci program rieši zadanú úlohu:

	$x = \alpha\mathbf{10}^i$
$a := x - 1$	$a = \alpha\mathbf{01}^i$
$b := x \wedge a$	$b = \alpha\mathbf{00}^i$
$y := if(b,0,1)$	$y = \mathbf{0}$ alebo $\mathbf{1}$

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

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

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.

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:

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

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 3. kola kategórie P

1. súťažný deň

Vydala IUVENTA pre vnútornú potrebu Ministerstva školstva SR

Zodpovedný redaktor: M. Forišek

Sadzba programom LaTeX