Slovenská komisia Matematickej olympiády
FMFI UK, Mlynská dolina, 842 48 Bratislava 4

50. ročník Matematickej olympiády, školský rok 2000/2001

Riešenia úloh celoštátneho kola kategórie P
1. súťažný deň

P-III-1 *(Veže)*

Na úvod si treba uvedomiť, že jednotlivé súradnice pozícií veží môžeme voliť nezávisle na sebe. To preto, že voľba stĺpca nijako neobmedzuje rozsah riadkov, v ktorých môže byť veža umiestnená a naopak. Môžeme teda pre každú vežu najskôr zvoliť stĺpec, v ktorom sa bude nachádzať a potom pre každú zvoliť riadok. Tým sme pôvodnú úlohu previedli do jedného rozmeru. Jednotlivé obdĺžniky sa nám premietnu na intervaly a v každom intervale chceme vybrať jedno číslo tak, aby sa žiadne dve vybrané čísla nezhodovali.

Čísla budeme hľadať nasledujúcim spôsobom: Budeme postupne prechádzať čísla od 1 do N a budeme si udržiavať informáciu, ktoré intervaly začínajúce pred týmto číslom ešte nemajú pridelené žiadne číslo. Z týchto intervalov vyberieme najskôr končiaci a priradíme mu aktuálne číslo. Pokiaľ vybraný interval už prideľované číslo neobsahuje (skončil skôr), tvrdíme, že úloha nemá riešenie.

Majme nejaké korektné rozostavenie veží R. Indukciou dokážeme, že každé takéto rozostavenie vieme upraviť na rozostavenie, ktoré navrhne náš algoritmus. Tým teda ukážeme, že ak existuje korektné rozostavenie veží, tak náš algoritmus jedno korektné rozostavenie nájde.

1° Nech C je najmenšie číslo, ktoré chce náš algoritmus priradiť nejakému (konkrétne najskôr začínajúcemu) intervalu I. Z popisu algoritmu je zrejmé, že menšie číslo sa v žiadnom rozostavení nemôže vyskytovať. Pokiaľ sa v R nevyskytuje ani číslo C, môžeme R upraviť tak, že intervalu I priradíme namiesto jeho súčasného čísla číslo C. Tým sme dostali korektné rozostavenie, ktoré sa navyše v prvom čísle zhoduje s riešením, ktoré nájde náš algoritmus. Ak je v R číslo C už priradené nejakému intervalu J, môžeme intervalu J priradiť číslo, ktoré mal doteraz priradené interval I a intervalu I priradiť číslo C. Keďže interval I končí zo všetkých intervalov obsahujúcich C najskôr, bude toto rozostavenie opäť korektné.

2° Z indukčného predpokladu vieme, že existuje rozostavenie, ktoré sa s rozostavením navrhnutým našim algoritmom zhoduje v prvých k číslach, chceme ukázať, že existuje rozostavenie, ktoré sa s ním zhoduje v prvých $k+1$ číslach. Myšlienka je úplne rovnaká ako v prvom kroku indukcie, preto túto časť dôkazu nechávame na čitateľa.

Priama implementácia algoritmu vedie k riešeniu s časovou zložitosťou $O(N^2)$. My implementáciu zrýchlime nasledujúcim spôsobom: Aby sme vedeli rýchlo upravovať informáciu o tom, ktoré intervaly obsahujú práve spracúvané číslo, zoradíme si ich najskôr podľa začiatku. Aby sme boli schopní rýchlo nájsť najskôr končiaci z nich, budeme si intervaly obsahujúce spracúvané číslo, udržiavať v halde podľa ich koncov. Vylepšená implementácia teda vyzerá nasledovne:

Prechádzame postupne čísla od 1 do N. Pre každé si do haldy pridáme všetky intervaly, začínajúce týmto číslom. Potom z haldy (ak je neprázdna) vyberieme najskôr končiaci interval a pridelíme mu toto číslo. Pokiaľ vybraný interval už toto číslo neobsahuje, vyhlásime, že rozostavenie neexistuje. S týmito vylepšeniami má algoritmus časovú zložitosť $O(N.\log N)$ a pamäťovú $O(N)$.

Program je priamou implementáciou algoritmu. Halda je implementovaná v poli, kde prvok na pozícii i má synov na pozíciách $2i$ a $2i+1$.

```
program Veze;                    {P-III-1}
const
  MAXN = 100;
type
  {Popis jednoho intervalu}
  Int = record
          s, e : Integer;        {Počátek a konec intervalu}
          n : Integer;           {Číslo věže, které interval patří}
        end;
  {Popis jednoho bodu}
  Point = array [1..2] of Integer;
  {Popis jednoho obdélníka}
  Rectangle = record
                a, b : Point;    {Levý horní a pravý dolní roh}
              end;
  {Intervaly v jedné souřadnici}
  IntList = array[1..MAXN] of Int;
  CmpProc = function(A, B : Int) : ShortInt;
var
  N : Integer;  {Pocet vezi}
  Rec : Array[1..MAXN] of Rectangle;    {Obdélníky}
  T : Array[1..MAXN] of Point;          {Umístění věží}

{Načte vstup}
procedure ReadInp;
var
  i : Integer;
begin
  Write('Pocet vezi: ');
  Read(N);
  WriteLn('Souradnice obdelniku:');
  for i := 1 to N do
    Read(Rec[i].a[1], Rec[i].a[2], Rec[i].b[1], Rec[i].b[2]);
end;

{Přidá interval do haldy}
procedure AddHeap(var N : Integer; var H : IntList; I : Int;
                  Cmp : CmpProc);
```

```
var
  A : Integer;
  Tmp : Int;
begin
  Inc(N);
  H[N] := I;
  A := N;
  while A <> 1 do begin {Nejsme na vrcholu}
    if Cmp(H[A], H[A div 2]) <> -1 then
      break;  {Je splněna podmínka haldy?}
    {Zaměníme prvky, aby byla podmínka splněna}
    Tmp := H[A];
    H[A] := H[A div 2];
    H[A div 2] := Tmp;
    A := A div 2; {Posun o úroveň výše}
  end;
end;

{Odebere z haldy minimum}
procedure GetHeapMin(var N : Integer; var H : IntList; Cmp : CmpProc;
                     var Res : Int);
var
  A, M : Integer;
  Tmp : Int;
begin
  Res := H[1];
  H[1] := H[N];
  Dec(N);
  A := 1;
  while A * 2 < N do begin      {Nejsme na dně?}
    {Nalezneme menšího ze synů}
    if Cmp(H[A*2], H[A*2+1]) = -1 then
      M := A*2
    else
      M := A*2+1;
    if Cmp(H[M], H[A]) <> -1 then  {Podmínka haldy splněna?}
      break;
    {Zaměníme prvky, aby byla podmínka splněna}
    Tmp := H[M];
    H[M] := H[A];
    H[A] := Tmp;
    A := M; {Posun o úroveň níže}
  end;
end;

{Porovná dva intervaly podle počátku}
```

```
function CmpInt(a, b : Int) : ShortInt; far;
begin
  if (a.s < b.s) or ((a.s = b.s) and (a.e < b.e)) then
    CmpInt := -1
  else if (a.s = b.s) and (a.e = b.e) then
    CmpInt := 0
  else
    CmpInt := 1;
end;

{Setřídí pole intervalů}
procedure SortInts(var A : IntList);
var
  C, i : Integer;
  H : IntList;
begin
  C := 0;
  for i := 1 to N do
    AddHeap(C, H, A[i], CmpInt);
  for i := 1 to N do
    GetHeapMin(C, H, CmpInt, A[i]);
end;

{Srovná intervaly podle konce}
function CmpBack(a, b : Int) : ShortInt; far;
begin
  if a.e < b.e then
    CmpBack := -1
  else if a.e = b.e then
    CmpBack := 0
  else
    CmpBack := 1;
end;

{Spočte jednu souřadnici pro každou věž}
function CountCoord(c : Integer) : Boolean;
var
  Ints : IntList;  {Intervaly}
  i, a : Integer;
  ActInts : IntList;    {Halda intervalů k uplatnění}
  ActIntsN : Integer;   {Počet intervalů v haldě}
  Tmp : Int;
begin
  {Vytvoří pole intervalů z obdélníku}
  for i := 1 to N do begin
    Ints[i].s := Rec[i].a[c];
```

```
    Ints[i].e := Rec[i].b[c];
    Ints[i].n := i;
  end;
  SortInts(Ints); {Setřídí intervaly}
  {Určí souřadnici věží}
  ActIntsN := 0;
  a := 1;
  for i := 1 to N do begin
    {Přidá do haldy všechny intervaly začínající na aktuální pozici}
    while (a <= N) and (Ints[a].s = i) do begin
      AddHeap(ActIntsN, ActInts, Ints[a], CmpBack);
      Inc(a);
    end;
    if ActIntsN > 0 then begin
      {Vybereme nejdříve končící interval}
      GetHeapMin(ActIntsN, ActInts, CmpBack, Tmp);
      if Tmp.e < i then begin  {Už skončil?}
        CountCoord := False;
        Exit;
      end;
      T[Tmp.n][c] := i;
    end;
  end;
  CountCoord := True;
end;

{Vytiskne souřadnice věží}
procedure Print;
var
  i : Integer;
begin
  WriteLn('Souradnice vezi jsou:');
  for i := 1 to N do
    WriteLn(T[i][1], ' ', T[i][2]);
end;

begin
  ReadInp;              {Načte vstup}
  if CountCoord(1) and CountCoord(2) then
    Print   {Určí souřadnice věží - vešly se?}
  else
    WriteLn('Rozmisteni vezi neexistuje.');
end.
```

P-III-2 *(Nuly a jednotky)*

Najskôr ukážeme, že pre každé N existuje prirodzené číslo x, ktoré má cifry len 0 a 1 a je deliteľné N. Označme $x_1 = 1$, $x_2 = 11$, $x_3 = 111$ atď. Ďalej označme $m_i = x_i \bmod N$. Čísla m_i môžu nadobúdať iba hodnoty od 0 do $N-1$, a preto sa aspoň dve z čísel m_1 až m_{N+1} rovnajú. Nech sú to m_i a m_j $(i < j)$. Potom ale číslo $x = x_j - x_i$ je deliteľné N a jeho zápis sa zjavne skladá len z cifier 0 a 1.

Naďalej budeme uvažovať len čísla zložené z cifier 0 a 1. Predchodcom čísla x budeme volať číslo $x \div 10$ (teda x bez poslednej cifry) a nasledovníkmi čísla $10x$ a $10x+1$ (teda tie, ktoré z x dostaneme pridaním ďalšej cifry). Číslo x budeme nazývať minimálnym číslom pre zvyšok z, ak $x \bmod N = z$, x obsahuje vo svojom zápise len cifry 0 a 1 a je zo všetkých takýchto čísel najmenšie. Teraz si dokážeme jednoduchú pomocnú vetu:

Lemma: Nech x je minimálne číslo pre zvyšok z, nech x' je predchodca x; označme $z' = x' \bmod N$. Potom x' je minimálne číslo pre zvyšok z'.

Dôkaz: Sporom. Predpokladajme, že x' nie je minimálne číslo pre zvyšok z', čiže existuje $x'' < x'$ také, že $x'' \bmod N = x' \bmod N$. Nech c je posledná cifra čísla x. Keďže $(10x'+c) \bmod N = z$ a $x' \bmod N = x'' \bmod N$, musí nutne platiť aj $(10x''+c) \bmod N = z$, ale potom by číslo x nemohlo byť minimálne pre zvyšok z, lebo číslo $10x''+c$ má požadované vlastnosti a je menšie. Tým sme dospeli k sporu, preto nutne x' je minimálne číslo pre zvyšok z', q.e.d.

Podľa tejto pomocnej vety budeme počítať minimálne čísla pre rôzne zvyšky. Pre daný zvyšok z sa dá spočítať minimálne číslo x tak, že budeme postupne testovať v rastúcom poradí nasledovníkov minimálnych čísel pre ostatné zvyšky a prvý nasledovník y nejakého minimálneho čísla, pre ktorého platí $y \bmod N = z$, je zjavne hľadaným minimálnym číslom pre zvyšok z.

Týmto postupom nájdeme minimálne čísla pre všetky zvyšky, pre ktoré existujú. Položme $l_1 = 1$ a keď už vieme $l_1, \ldots, l_k$, položme l_{k+1} rovné najmenšiemu nasledovníkovi niektorého z čísel $l_1, \ldots, l_k$, pre ktorého platí, že $l_{k+1} \bmod N$ je rôzne od všetkých $l_i \bmod N$ pre $1 \le i \le k$. Z predchádzajúcich úvah ale vyplýva, že jednotlivé čísla l_i sú minimálne čísla pre zvyšky $z_i = l_i \bmod N$. Naviac podľa prvého odstavca sa niektoré z čísel z_i rovná 0.

Náš algoritmus bude pracovať presne podľa vyššie uvedeného popisu. V poli `fronta` si budeme pamätať hodnoty z_i a na m-tej pozícii poľa `predchadzajuci` si budeme pamätať zvyšok predchodcu minimálneho čísla pre zvyšok m – z hodnôt v tomto poli sme schopní ľahko zostrojiť minimálne číslo pre zadaný zvyšok. Postupne budeme brať hodnoty z poľa `fronta`, zrátame hodnoty $(10 * z_i) \bmod N$ a $(10 * z_i + 1) \bmod N$ (čo sú zvyšky, ktoré dávajú nasledovníci minimálneho čísla pre zvyšok z_i) a ak sme doteraz nenašli číslo, ktorého zvyšok by bola jedna z týchto hodnôt, tak tento zvyšok pridáme na koniec fronty a príslušne zmeníme pole `predchadzajuci`. Časová aj pamäťová zložitosť algoritmu je zjavne $O(N)$.

```
program nulajed; { P-III-2 }
const MAXN=1000;
var N:word;  { zadané číslo N }
    predchozi:array[0..MAXN-1] of integer;
```

```
                { v slovenskom komentari sa vola predchadzajuci}
        { zbytek předchůdce minimálního čísla s daným zbytkem
       Hodnoty se zvláštním významem:
       -2 ... dosud nenalezeno minimální číslo pro tento zbytek
       -1 ... nemá předchůdce (číslo 1)
     }
     fronta:array[0..MAXN-1] of word;
     { fronta použitá pro generování minimálních čísel }
     ukazatel:word;
     { právě zpracovávaný prvek v poli fronta }
     vefronte:word;
     { počet prvků v poli fronta }
procedure pridej(puvodni,novy:word);
begin    { přidá minimální prvek pro zbytek novy do fronty;
       jeho předchůdce je minimální pro zbytek puvodni }
 if predchozi[novy]<>-2 then exit;
 fronta[vefronte]:=novy;
 inc(vefronte);
 predchozi[novy]:=puvodni;
end;
procedure vypis(p:word);
begin    { vypíše číslo se zadaným zbytkem }
 if predchozi[p]>=0 then
  begin
   vypis(predchozi[p]);
   if (10*predchozi[p]) mod N=p then write(0) else write(1)
  end
 else
  write(1)
end;
begin
 readln(N);  { načteme číslo N a inicializujeme pole predchozi }
 for ukazatel:=0 to N-1 do predchozi[ukazatel]:=-2;
 fronta[0]:=1 mod N;
 predchozi[fronta[0]]:=-1;
 ukazatel:=0; vefronte:=1;
 while (predchozi[0]=-2) do
  begin    { generujeme čísla s různými zbytky... }
   pridej(fronta[ukazatel],(10*fronta[ukazatel]) mod N);
   pridej(fronta[ukazatel],(10*fronta[ukazatel]+1) mod N);
   inc(ukazatel);
  end;
 vypis(0);   { vypíšeme výsledek a odřádkujeme }
 writeln
end.
```

P-III-3 *(Dlaždice)*

Aby sme nemuseli všetko popisovať zbytočne zložito, zaveďme si jednoduché označenia: *Postupnosti* budú vždy zložené len z núl a jednotiek a všetky budú mať n prvkov. Budeme ich značiť tučnými písmenami a ich prvky písmenami s indexmi, teda napríklad $\mathbf{x}$ je postupnosť, obsahujúca prvky $x_1, \ldots, x_n$. Počtu jednotiek v postupnosti $\mathbf{x}$ budeme hovoriť jej *váha* a značiť $\#\mathbf{x}$.

Postupnos5 $\mathbf{y}$ je zotriedením postupnosti $\mathbf{x}$ práve vtedy, keď $y_1 \leq \ldots \leq y_n$ (teda $\mathbf{y}$ je neklesajúca) a $\#\mathbf{y} = \#\mathbf{x}$.

Zaoberajme sa najskôr tým, ako overiť druhú podmienku. Mohli by sme postupovať podobne ako v krajskom kole – v každom riadku vydláždenia odstrániť po jednej jednotke z oboch postupností a to opakovať tak dlho, kým nebudú obe obsahovať len nuly, čo ľahko overíme farbou spodného okraja steny. Tým úlohu vyriešime s lineárnou zložitosťou, čo však nie je (ako si ďalej ukážeme) optimálne. Zostaneme však pri myšlienke zjednodušovania testovaných postupností.

Tvrdenie: $\#\mathbf{x} = \#\mathbf{y} \iff \#\mathbf{x} \bmod 2 = \#\mathbf{y} \bmod 2 \ \wedge \ \#\hat{\mathbf{x}} = \#\hat{\mathbf{y}}$, kde $\hat{\mathbf{x}}$ je postupnosť, ktorú dostaneme z $\mathbf{x}$ prepísaním každej druhej jednotky na nulu (t.j. prvú prepíšeme, druhú necháme, tretiu prepíšeme atď.)

Dôkaz: Ak $\#\mathbf{x} = \#\mathbf{y}$, tak iste $\#\mathbf{x} \bmod 2 = \#\mathbf{y} \bmod 2$, ale keďže $\#\hat{\mathbf{x}} = \lfloor \#\mathbf{x}/2 \rfloor$, musí byť aj $\#\hat{\mathbf{x}} = \#\hat{\mathbf{y}}$. Naopak ak $\#\hat{\mathbf{x}} = \#\hat{\mathbf{y}}$, môžu sa váhy $\mathbf{x}$ a $\mathbf{y}$ líšiť najviac o 1, a keďže dávajú rovnaký zvyšok po delení 2, nutne sa rovnajú.

Vytvorme najskôr dlaždicový podprogram, ktorý pre danú postupnosť $\mathbf{x}$ zadanú farbami horného okraja spočíta $\hat{\mathbf{x}}$ na dolnom okraji a $\#\mathbf{x} \bmod 2$ na pravom okraji. (V tom zmysle, že vydláždenie bude existovať práve vtedy, keď farby týchto okrajov spĺňajú dané podmienky.) Toto vydláždenie bude mať jediný riadok. Predpokladajme, že ľavý okraj má farbu 0. Budeme potrebovať nasledovnú sadu dlaždíc:

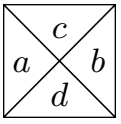
$$T_0 = \left\{ \begin{array}{|cc|}\hline & 0 \\ 0 & 0 \\ \hline & 0 \\ \hline \end{array}\ , \begin{array}{|cc|}\hline & 1 \\ 0 & 1 \\ \hline & 0 \\ \hline \end{array}\ , \begin{array}{|cc|}\hline & 0 \\ 1 & 1 \\ \hline & 0 \\ \hline \end{array}\ , \begin{array}{|cc|}\hline & 1 \\ 1 & 0 \\ \hline & 1 \\ \hline \end{array} \right\}$$

Ak si označíme z_i farbu pravej hrany a y_i farbu dolnej hrany i-tej dlaždice vydláždeného riadku (z_0 nech je 0). Ľahko ukážeme, že $z_i = \left(\sum_{k=1}^{i} x_k\right) \bmod 2$. Indukciou – pre $i = 0$ platí, pre $i > 0$ je

$$z_i = (z_{i-1} + x_k) \bmod 2 = \left(\left(\sum_{k=1}^{i-1} x_k\right) \bmod 2 + x_i\right) \bmod 2 = \left(\sum_{k=1}^{i} x_k\right) \bmod 2.$$

Taktiež si treba uvedomiť, že $y_i = 1 \iff x_i = 1 \ \wedge \ z_{i-1} = 1$, inými slovami práve vtedy, keď je x_i jednotka, pred ktorou bol nepárny počet jednotiek, teda keď táto je párna, čiže ju máme nechať v $\hat{\mathbf{x}}$.

Skombinujme teraz dva takéto programy tak, aby pracovali naraz – jeden s postupnosťou $\mathbf{x}$ a druhý s $\mathbf{y}$ (postupnosti sú kódované dvojicami (x_i, y_i)). Budeme požadovať, aby vypočítali rovnako kódované postupnosti $\hat{\mathbf{x}}$ a $\hat{\mathbf{y}}$ aj zvyšky $\#\mathbf{x} \bmod 2$ a $\#\mathbf{y} \bmod 2$. Na to stačí zostrojiť množinu T_1 obsahujúcu „súčiny“ dvojíc dlaždíc z množiny T_0, to znamená pre každé dve dlaždice

$A =$ 

a $B =$ 

z T_0 do T_1 pridať dlaždicu

,

kde ae, bf, cg a dh sú usporiadané dvojice farieb. Ak existuje vydláždenie riadku, ktorý má na hornej hrane dvojice $(x_1, y_1), \ldots, (x_n, y_n)$, na dolnej $(\hat{x}_1, \hat{y}_1), \ldots, (\hat{x}_n, \hat{y}_n)$, na ľavej $(0, 0)$ a na pravej (z_x, z_y) dlaždicami z T_1, musí vďaka tomu, ako sme si tieto dlaždice zadefinovali, existovať aj vydláždenie riadku s hornou hranou $x_1, \ldots, x_n$, dolnou $\hat{x}_1, \ldots, \hat{x}_n$, ľavou 0 a pravou z_x, ako aj analogické vydláždenie pre $\mathbf{y}$. A naopak, ak existujú tieto dve vydláždenia, existuje aj ich zloženie zostavené z dlaždíc z množiny T_1. Preto $z_x = \#\mathbf{x} \bmod 2$, $z_y = \#\mathbf{y} \bmod 2$ a spodná hrana naozaj obsahuje dvojice prvkov postupností $\hat{\mathbf{x}}$ a $\hat{\mathbf{y}}$.

My však potrebujeme akceptovať práve tie vydláždenia, u ktorých $z_x = z_y$, teda s pravým okrajom $(0, 0)$ aj $(1, 1)$. K dispozícii však máme len jednu farbu pravého okraja. Preto rozšírime množinu T_1 na T_2 tak, že ku každému typu dlaždice z T_1 tvaru

alebo 

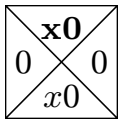
.

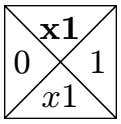
pridáme do T_2 typ

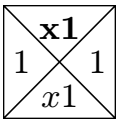
,

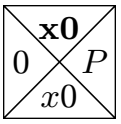
kde P je farba pravého okraja, ktorá sa nezhoduje s žiadnou inou farbou. Keďže sa táto dlaždica môže vyskytnúť len tesne pri pravej stene (keďže žiadna dlaždica nemá ľavú hranu farby P), zodpovedajú korektné vydláždenia pomocou tejto sady dlaždíc práve tým vydláždeniam sadou T_1, pri ktorých je $z_x = z_y$.

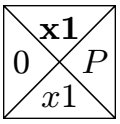
Zostáva nám ešte pridať test, či je postupnosť $\mathbf{y}$ neklesajúca, čo vyriešime pridaním dlaždíc typov:

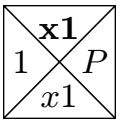
$T_z = \Bigg\{$ 

, 

, 

, 

, 

, 

;

$x \in \{0, 1\}, \mathbf{x} =$ „tučná verzia“ $x \Bigg\}$,

kde **00**, **01**, **10** a **11** sú farby dvojíc kódujúcich zadané postupnosti $\mathbf{x}$ a $\mathbf{y}$ a 00, 01, 10 a 11 analogické kódy používané všetkými ostatnými doteraz zadefinovanými dlaždicami. Tým z T_2 vznikne množina T, o ktorej tvrdíme, že s farbou ľavého okraja 0, pravého P a spodného 00 rieši našu úlohu v čase $O(\log n)$. Toto tvrdenie teraz dokážeme.

Prvý riadok musí obsahovať len dlaždice z T_z, lebo žiadne iné nemajú na svojích horných hranách farby zodpovedajúce tým, ktorými je kódovaný vstup. Ako sme už ukázali v domácom kole, táto sada overí, že postupnosť $\mathbf{y}$ je neklesajúca. Navyše na spodnom okraji tohto riadku dostaneme pôvodné postupnosti zo vstupu, len ináč kódované. Všetky ostatné riadky obsahujú len dlaždice z T_2, pričom každý riadok prepíše postupnosti $\mathbf{x}_i$ a $\mathbf{y}_i$ zadané na svojom hornom okraji na postupnosti $\mathbf{x}_{i+1} = \hat{\mathbf{x}}_i$ a $\mathbf{y}_{i+1} = \hat{\mathbf{y}}_i$ a overí, či $\#\mathbf{x}_i \bmod 2 = \#\mathbf{y}_i \bmod 2$. Ak $\#\mathbf{x} = \#\mathbf{y}$, tak po maximálne $\lceil \log_2 n \rceil$ riadkoch budú obe postupnosti zredukované na nuly, čo zodpovedá farbe spodného okraja. Čiže v takomto prípade vydláždenie existuje a má hĺbku $\leq 1 + \lceil \log_2 n \rceil$. Ak $\#\mathbf{x} \neq \#\mathbf{y}$, vydláždenie nemôže existovať, lebo v aspoň jednom kroku by sa $\#\mathbf{x}_i \bmod 2$ nerovnalo $\#\mathbf{x}_i \bmod 2$ (viď Tvrdenie).

Poznámka: Lepšiu hĺbku než $\Omega(\log n)$ sa nedá dosiahnuť. To môžeme zdôvodniť podobne, ako sme v krajskom kole dokazovali, že na overenie symetrie potrebujeme lineárnu hĺbku. Opäť budeme počítať možné ofarbenia stredného stĺpca – tentokrát si uvedomíme, že tieto ofarbenia musia byť rôzne pre každé dva rôzne počty jednotiek v $x_1, \ldots, x_{n/2}$ – ak sú jednotky len medzi týmito prvkami a $x_{n/2+1}, \ldots, x_n$ sú nulové, musí zotriedená postupnosť $\mathbf{y}$ obsahovať naopak vo svojej ľavej polovici samé nuly a v pravej rovnako jednotiek ako $\mathbf{x}$ v ľavej. Možných počtov jednotiek medzi $x_1, \ldots, x_{n/2}$ je $n/2+1$, možných ofarbení stredného stĺpca f^h, kde f je počet nami použitých farieb a h je výška steny, teda zložitosť programu. Z toho dostávame:

$$b^h \geq n/2 + 1 \quad \Longrightarrow \quad b^h > n/2 \quad \Longrightarrow \quad h > \log_b n - \log_b 2 \quad \Longrightarrow \quad h = \Omega(n).$$

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY

50. ROČNÍK MATEMATICKEJ OLYMPIÁDY

Vzorové riešenia celoštátneho kola kategórie P

Vydala IUVENTA – zariadenie pre voľný čas detí, mládeže i dospelých MŠ SR
pre vnútornú potrebu Ministerstva školstva SR
Programom TEX sadzbu pripravil Michal Forišek

Autori príkladov: Jan Kára
Daniel Kráľ
Martin Mareš