Slovenská komisia Matematickej olympiády
FPEDaS ŽU, Veľký diel, 010 26 Žilina

MATEMATICKÁ OLYMPIÁDA NA STREDNÝCH ŠKOLÁCH

52. ročník, školský rok 2002/2003

Riešenia úloh 1. kola kategórie P

Tento pracovný materiál nie je určený priamo študentom — účastníkom olympiády. Má pomôcť učiteľom na školách pri príprave konzultácií a pracovných seminárov pre riešiteľov súťaže, členom krajských výborov MO slúži ako podklad pre opravovanie úloh domáceho kola MO kategórie P. Študentom možno tieto komentáre poskytnúť až po termíne stanovenom pre odovzdanie riešení úloh domáceho kola MO kategórie P ako informáciu, ako bolo treba úlohy správne riešiť a pre ich odbornú prípravu na účasť v krajskom kole súťaže.

P-I-1

Pri riešení úlohy budeme používať terminológiu z teórie grafov. Významné miesta na čajovníku budeme nazývať *vrcholy*, časti kmeňa čajovníku medzi dvoma významnými miestami *hrany*. Vrcholy spolu s hranami pomenujeme *strom* (stromom sa v teórii grafov nazýva súvislý graf, ktorý nemá žiadny cyklus). Počet hrán, ktoré vedú z nejakého vrcholu v (teda vlastne počet častí kmeňa, ktoré vedú z významného miesta v), nazveme *stupňom* vrcholu v.

Najskôr si všimnime, že každý strom s aspoň dvomi vrcholy má aspoň jeden vrchol stupňa jeden (takýto vrchol sa nazýva *list*). Tento vrchol môžeme nájsť takto. Začneme strom prehľadávať v ľubovoľnom vrchole. Ak ešte nie sme v liste, prejdeme do ľubovoľného susedného (t.j. pripojeného hranou) vrcholu, v ktorom sme ešte neboli. Keďže v strome nie sú cykly, musí takýto vrchol vždy existovať. Pretože vrcholov je konečný počet, musíme raz skončiť – a to môžeme iba v liste.

Teraz ukážeme, že súčet stupňov všetkých vrcholov v ľubovoľnom strome je $2N-2$ (kde N je počet vrcholov). Naopak platí, že ak máme N kladných celých čísel so súčtom $2N-2$, tak existuje strom s N vrcholmi majúcimi tieto stupne. Z toho už je jasné, že stačí zistiť, či čísla na vstupe majú súčet $2N-2$, a podľa výsledku vypísať príslušnú správu.

Prvé tvrdenie dokážeme indukciou vzhľadom na počet vrcholov. Strom s dvoma vrcholmi obsahuje iba jednu hranu. Súčet stupňov vrcholov je teda $1+1=2$ a naše tvrdenie platí. Ak má strom viac vrcholov, podľa predchádzajúceho pozorovania vieme, že má list. Ak tento list odoberieme (teda zrušíme vrchol a hranu, ktorá ho pripojuje k zvyšku stromu), získame zjavne opäť strom. Pre nový strom z indukčného predpokladu platí, že súčet stupňov je $2\cdot(N-1)-2=2N-4$. Pretože pôvodný strom mal o jeden list viac a jeden jeho jeden vrchol mal stupeň o jedna vyšší (ten, ku ktorému byl pripojený list), je súčet stupňov v pôvodnom strome $2N-4+2=2N-2$. Tým je prvé tvrdenie dokázané.

Druhé tvrdenie dokážeme indukciou podľa počtu členov postupnosti: Nech máme postupnosť dvoch kladných celých čísel, ktorých súčet je $2\cdot 2-2=2$. Tieto čísla teda môžu byť iba dve jednotky. Pre ne sú zrejme zodpovedajúcim stromom dva vrcholy spojené hranou. Ak má postupnosť viac ako dve čísla, musí zjavne obsahovať aspoň jednu jednotku (inak by súčet N čísel bol aspoň $2N$ a nie $2N-2$). Analogicky musí tiež obsahovať aspoň jedno číslo väčšie ako jedna. Ak z postupnosti vypustíme jednu jednotku a jedno z čísel väčších ako jedna znížime o jedna, získame postupnosť čísel o jedna kratšiu so súčtom $2N-2-2=2\cdot(N-1)-$

2. Z indukčného predpokladu teda existuje strom s $N-1$ vrcholmi s príslušnými stupňami vrcholov. Keď do stromu pridáme jeden list a pripojíme ho hranou k vrcholu, ktorý zodpovedá číslu, ktoré sme zmenšovali o jedna, získame presne strom pre našu pôvodnú postoupnosť. Tým je dokázané druhé tvrdenie.

Časová zložitosť algoritmu je $O(N)$, pamäťová $O(1)$.

```
program Caj;
var
  N, i, Suma, Casti : Integer;
  vstup, vystup : Text;
begin
  Assign(vstup, 'caj.in'); Assign(vystup, 'caj.out');
  Reset(vstup); Rewrite(vystup);
  Suma := 0;
  ReadLn(vstup, N);
  for i := 1 to N do begin
    Read(vstup, Casti);
    Suma := Suma + Casti;
  end;
  if Suma = 2 * (N - 1) then WriteLn(vystup, 'EXISTUJE')
    else WriteLn(vystup, 'NEEXISTUJE');
  Close(vstup); Close(vystup);
end.
```

P-I-2

Ukážeme si dve možné riešenia tejto úlohy. Obe sú založené na metóde nazývanej dynamické programovanie: úloha sa najskôr vyrieši pre podúlohu veľkosti 1. Toto riešenie se použije na nájdenie riešenia podúlohy veľkosti 2. Takto nájdené riešenie sa použije na vyriešenie podúlohy veľkosti 3, atď. V našom prípade bude veľkosť podúlohy určená počtom kníh, ktoré chceme do skrine umiestiť.

Prvé riešenie používa dvojrozmerné pole A veľkosti $N \times V$, kde N je celkový počet kníh, ktoré máme do skrine umiestiť a V je maximálna výška skrine (V je v našom prípade 250 podľa zadania úlohy). Hodnota $A[i,j]$, $0 \leq i \leq N, 1 \leq j \leq V$ udáva minimálnu možnú šírku skrine výšky j, do ktorej je možné umiestiť prvých i kníh. Ak do skrine výšky j prvých i kníh nemožno umiestiť, t.j. niektorá z týchto kníh je vyššia ako $j-2$ cm, tak je hodnota $A[i,j]$ rovná nejakej špeciálnej hodnote, napríklad -1.

Popíšeme si, ako spočítať hodnotu $A[i_0,j_0]$ v čase $O(N)$, ak už máme spočítané hodnoty $A[i,j]$ pre $i < i_0$. Ak $i_0 = 0$, tak zjavne $A[i_0,j_0] = 0$ cm. Ak existuje i, $1 \leq i \leq i_0$, také, že výška v_i i-tej knihy je viac ako $j_0 - 2$ cm, tak prvých i kníh nemožno do skrine výšky j_0 umiestiť a $A[i_0,j_0]$ bude rovné -1. V ostatných prípadoch určíme hodnotu $A[i_0,j_0]$ nasledovne. Pre $0 \leq i < i_0$ skúsime umiestiť na poslednú poličku skrine $(i+1)$-vú až i_0-tu knihu a prvých i kníh dáme na predchádzajúce poličky; výška poslednej poličky by teda musela byť aspoň $v = \max_{i+1 \leq k \leq i_0} v_k$ a môžeme predpokladať, že je práve v. Šírka tejto poličky musí byť aspoň $i_0 - i$. Ak sa $A[i, j_0 - v - 1]$ rovná -1, tak sa nedá vytvoriť skriňa výšky j_0, ktorá by obsahovala prvých i_0 kníh a na poslednej poličke by z nich mala posledných $i_0 - i$. V opačnom prípade sa najmenšia šírka skrine výšky j_0, ktorá obsahuje prvých i_0 kníh a na poslednej poličke má z nich umiestnených posledných $i_0 - i$, rovná $\max\{A[i, j_0 - v - 1], i_0 - i\}$. Najmenší z týchto výrazov pre $0 \leq i < i_0$ je hľadanou hodnotou $A[i_0,j_0]$. Vyššie popísaný výpočet sa dá vykonať v čase

$O(N)$, ak postupujeme od $i = i_0 - 1$ k $i = 0$. V takom prípade je možné $v = \max_{i+1 \le k \le i_0} v_k$ spočítať z v pre hodnotu i o 1 väčšiu v konštantnom čase. Hodnota poľa $A[N, 250]$ je hľadanou minimálnou možnou šírkou skrine.

Ak chceme zároveň nájsť aj rozmiestnenie kníh do skrine a výšky jednotlivých poličiek, zavedieme si ešte pomocné pole $B[i, j]$, $0 \le i \le N, 1 \le j \le V$ a pri výpočte hodnoty $A[i_0, j_0]$ si do položky $B[i_0, j_0]$ uložíme to i, pre ktoré je šírka skrine minimálna pri výške j_0. Z hodnoty $B[N, 250]$ určíme počet kníh, ktoré sú v optimálnom riešení na poslednej poličke; táto hodnota nám umožní spočítať výšku skrine bez poslednej poličky a počet kníh v ostatných poličkách dohromady. Z príslušnej hodnoty v poli B určíme počet kníh na predposlednej poličke a takto postupujme, až kým nedosiahneme prvú poličku. Vzhľadom na veľkosť poľa A je časová zložitosť práve popísaného algoritmu $O(VN^2)$ a pamäťová zložitosť $O(VN)$.

Teraz popíšeme druhé možné riešenie. Najprv ukážeme, ako sa dá v čase $O(N^2)$ rozhodnúť, či sa dajú knihy umiestniť do skrine šírky s a výšky V. K tomu si vytvoríme pomocné pole $A[i]$, $0 \le i \le N$, ktoré obsahuje minimálnu výšku skrine šírky s, do ktorej je možné umiestiť prvých i kníh. Ak $A[N] > V$, tak sa knihy nedajú umiestiť do skrine šírky s a výšky V; v opačnom prípade sa umiestniť do skrine s týmito rozmery dajú. Na určenie hodnôt v poli A opäť použijeme dynamické programovanie. Hodnota $A[0]$ je 1 cm, čo je špeciálny prípad všeobecného vzťahu pre výšku skrine "súčet výšiek poličiek + (počet poličiek + 1) × 1 cm". Popíšeme, ako určiť hodnotu $A[i_0]$, ak poznáme hodnoty $A[i]$ pre $0 \le i < i_0$. Zvoľme $i_0 - s \le i < i_0$; na poslednú poličku chceme umiestiť v takomto prípade posledných $i_0 - i$ kníh (preto podmienka $i_0 - s \le i$). Výška skrine je potom $A[i] + 1 + \max_{i<k\le i_0} v_k$; najmenší z týchto výrazov pre i, $i_0 - s \le i < i_0$ je hľadaná hodnota $A[i_0]$. Hodnotu $A[i_0]$ možno spočítať v čase $O(N)$, ak budeme postupovať od $i = i_0 - 1$ k $i = i_0 - s$ (potom je možné spočítať $\max_{i<k\le i_0} v_k$ z hodnoty pre $i + 1$ v konštantnom čase). Popísaná procedúra v čase $O(N^2)$ s pamäťou $O(N)$ rozhodne, či sa zadaných N kníh dá umiestiť do skrine šírky s a výšky V.

Zostáva popísať, ako sa dá táto procedúra použiť na vyriešenie pôvodnej úlohy. Najprv skontrolujeme, že výška všetkých kníh je najviac $V - 2$ cm = 248 cm a teda že knihy sa vôbec dajú umiestiť do nejakej skrine výšky V. Na určenie minimálnej šírky skrine s_0 použijeme metódu nazývanú binárne vyhľadávanie. Budeme si udržovať dve premenné $s_1 \le s_2$, ktoré nám budú ohraničovať možný interval, v ktorom je hľadaná šírka s_0, t.j., $s_1 \le s_0 \le s_2$. Najprv položíme $s_1 = 1$ a $s_2 = N$. V každom kroku zvolíme $s = \lfloor (s_1 + s_2)/2 \rfloor$ a pomocou vyššie popísanej procedúry skontrolujeme, či sa dá umiestniť našich N kníh do skrine výšky V a šírky s. Ak sa knihy dajú do takej skrine umiestiť, priradíme do s_2 hodnotu s; v opačnom prípade priradíme do s_1 hodnotu $s + 1$. Celý postup opakujeme, kým sa hodnoty s_1 a s_2 líšia, t.j. až kým nenájdeme hľadanú hodnotu s_0. Všimnite si, že v každom kroku sa rozdiel $s_2 - s_1$ zmenší aspoň o 1 (ak by sme pri voľbe s použili hornú celú časť namiesto dolnej celej časti, nebolo by toto tvrdenie pravdivé) a tento rozdiel se zmenší zhruba na polovicu. Teda po $O(\log N)$ krokoch nájdeme hľadanú optimálnu šírku skrine s_0. Výšky poličiek a rozmiestnenie kníh je možné nájsť podobne ako v predchádzajúcom algoritme zavedením pomocného poľa B, do ktorého si budeme ukladať počet kníh na poslednej poličke v optimálnom riešení. Celková časová zložitosť práve popísaného algoritmu je teda $O(N^2 \log N)$ a pamäťová zložitosť je $O(N)$.

Zostáva vyriešiť otázku, ktorý z dvoch popísaných algoritmov je lepší. Odpoveď je, že ani jeden. Vzhľadom k zadaniu úlohy, kde V je obmedzené, je časová zložitosť prvého algoritmu síce $O(N^2)$ a pamäťová iba $O(N)$, ale multiplikatívna konštanta skrytá vo "veľkom O" je lineárna s V; na druhej strane pamäťová zložitosť druhého algoritmu je iba $O(N)$, kde multiplikatívna konštanta nezáví od výšky. Podľa vyššie popísaného postupu druhý algoritmus potrebuje polia, ktoré sú 250-krát menšie. Rovnako v časovej zložitosti bude člen $\log N$ menší ako člen V vyskytujúci sa v časovej zložitosti prvého algoritmu. Prvý algoritmus je teda asymptoticky lepší pre obmedzenú výšku V, ale v skutočnosti bude lepší ako druhý popísaný algoritmus až pre veľmi

veľké hodnoty N. Dá sa teda povedať, že druhý algoritmus je použiteľnejší.

```
program p_1_2;
{ Riešenie úlohy P-I-2 verzia 1 }
const MAXN = 100;
      VYSKA_MIESTNOSTI = 250;
var vyska : array[1..MAXN] of word;      { výšky kníh }
    n : word;                            { počet kníh }
    A : array[0..MAXN, 1..VYSKA_MIESTNOSTI] of integer;
                                         { pole minimálnych šíriek skrine }
    B : array[0..MAXN, 1..VYSKA_MIESTNOSTI] of word;
                                         { počty kníh na poslednej poličke
                                           v optimálnom riešení }
function max(a, b : longint) : longint;
begin
  if a < b then max := b else max := a
end;
procedure vypis(n : word; v : word);
var i, k : word;
begin
  if n = 0 then begin
    writeln('Výška skrine: ', VYSKA_MIESTNOSTI - v + 1, ' cm');
    exit;
  end;
  k := 0;
  for i := n - B[n, v] + 1 to n do k := max(k, vyska[i]);
  vypis(n - B[n, v], v - k - 1);
  writeln;
  writeln('Výška poličky: ', k, ' cm');
  write('Knihy na poličke:');
  for i := n - B[n, v] + 1 to n do write(' ', i, '(', vyska[i], ' cm)');
  writeln;
end;

var i, j, k : word;
    maxvyska : word;
begin
  readln(n);
  for i := 1 to n do read(vyska[i]);
  for i := 1 to n do
    if vyska[i] > VYSKA_MIESTNOSTI - 2 then begin
      writeln('Pre zadané rozmery kníh neexistuje skriňa!');
      halt;
    end;
  for j := 1 to VYSKA_MIESTNOSTI do A[0, j] := 0;
  for i := 1 to n do
    for j := 1 to VYSKA_MIESTNOSTI do begin
      maxvyska := vyska[i];
      A[i, j] := -1;
```

```
        for k := 1 to i do begin
          maxvyska := max(maxvyska, vyska[i - k + 1]);
          if maxvyska + 2 > j then break;
          if A[i - k, j - maxvyska - 1] = -1 then continue;
          if (A[i, j] = -1) or (A[i, j] > max(A[i - k, j - maxvyska - 1], k)) then begin
            A[i, j] := max(A[i - k, j - maxvyska - 1], k);
            B[i, j] := k;
          end;
        end;
      end;
    if A[n, VYSKA_MIESTNOSTI] = -1 then begin
      writeln('Pre zadané rozmery kníh neexistuje skriňa!');
      halt;
    end;
    writeln('Optimálna šírka skrine je ', A[n, VYSKA_MIESTNOSTI], ' cm.');
    vypis(n, VYSKA_MIESTNOSTI);
end.
```

```
program p_1_2;
{ Riešenie úlohy P-I-2 verzia 2 }
const MAXN = 1000;
      VYSKA_MIESTNOSTI = 250;
var vyska : array[1..MAXN] of word;      { výšky kníh }
    n : word;                            { počet kníh }
function mozna_skrina(s : word; v : word; vypisovat : boolean) : boolean;
  var A : array[0..MAXN] of word;        { pole s minimálnymi výškami skríň }
      B : array[1..MAXN] of word;        { pole s počtami kníh na poličkách }
      maxvyska : word;
      i, j : word;
  begin
    A[0] := 1;
    for i := 1 to n do
      begin
        maxvyska := vyska[i];
        A[i] := A[i - 1] + maxvyska + 1;
        B[i] := 1;
        j := i - 1;
        while (j > 0) and (i - j < s) do
          begin
            if maxvyska < vyska[j] then maxvyska := vyska[j];
            if A[i] > A[j - 1] + maxvyska + 1 then
              begin
                A[i] := A[j - 1] + maxvyska + 1;
                B[i] := i - j + 1;
              end;
            dec(j);
          end
      end;
```

```
    mozna_skrina := A[n] <= v;
    if not(vypisovat) then exit;
    i := n;
    writeln('Výška skrine: ', A[n], ' cm.');

writeln('Výšky poličiek v skrini a ich naplnenie knihami od spodku knižnice:');
    while i > 0 do
      begin
        writeln;
        writeln('Výška poličky: ', A[i] - A[i - B[i]] - 1, ' cm');
        write('Knihy v poličke:');
        for j := i - B[i] + 1 to i do
          write(' ', j, '(', vyska[j], ' cm)');
        writeln;
        i := i - B[i];
      end
  end;
var i : word;
    s1, s2 : word;
begin
  readln(n);
  for i := 1 to n do read(vyska[i]);
  for i := 1 to n do
    if vyska[i] > VYSKA_MIESTNOSTI - 2 then
      begin
        writeln('Pre zadané rozmery kníh neexistuje skriňa!');
        halt;
      end;
  s1 := 1; s2 := n;
  while s1 < s2 do
    if mozna_skrina((s1 + s2) div 2, VYSKA_MIESTNOSTI, false) then
      s2 := (s1 + s2) div 2
    else
      s1 := (s1 + s2) div 2 + 1;
  writeln('Optimálna šírka skrine je ', s1, ' cm.');
  mozna_skrina(s1, VYSKA_MIESTNOSTI, true);
end.
```

P-I-3

Poznáme posledný stĺpec zotriedenej tabuľky. Základná myšlienka celého riešenia spočíva v tom, že tým je jednoznačne určený aj prvý stĺpec – stačí zoradiť písmená posledného stĺpca podľa abecedy. Teraz využijeme to, že jednotlivé riadky vznikli rotáciou pôvodného reťazca. Teda ak je na začiatku niektorého riadku písmeno x a na jeho konci písmeno y, znamená to, že v pôvodnom reťazci bolo písmeno x tesne za písmenom y. ("Tesne za" berieme cyklicky, t.j. prvé písmeno reťazca je tesne za posledným.) Keďže každé písmeno sa v reťazci vyskytuje najviac raz, je n riadkami tabuľky určené poradie písmen v celom reťazci, až na rotáciu. Naviac vieme, v ktorom riadku sa nachádza naše slovo, odkiaľ vieme jeho prvé písmeno.

Zostrojiť algoritmus podľa tejto myšlienky je už jednoduché. Písmená zadaného reťazca si

zoradíme podľa abecedy (keďže sú z obmedzeného rozsahu, vieme to spraviť v lineárnom čase) a vyrobíme si tabuľku, v ktorej bude každému písmenu reťazca priradené to, ktoré nasleduje v reťazci tesne za ním. Potom začneme od písmena, o ktorom vieme, že je prvé a postupujeme od neho podľa tabuľky až kým nevypíšeme celý reťazec.

Časová aj pamäťová zložitosť tohoto algoritmu sú zjavne lineárne od dĺžky vstupu.

```
program bw;

const MAX = 100;
type slovo = array[1..MAX] of char;
var prvy_stlpec, posledny_stlpec : slovo;
    riadok, dlzka, i, l : integer;
    buckets : array[char] of boolean;
    nasledovnik : array[char] of char;
    s : string;
    ch : char;

begin
  readln (s); dlzka := length (s); { načítanie zadania }
  for i := 1 to dlzka do posledny_stlpec[i] := s[i];
  readln (riadok);
  for ch := #0 to #255 do buckets[ch] := false; { bucket sort }
  for i := 1 to dlzka do buckets[posledny_stlpec[i]] := true;
  l := 0;
  for ch := #0 to #255 do if buckets[ch] then begin
    inc (l);
    prvy_stlpec[l] := ch;
  end;
  for i := 1 to dlzka do { určenie nasledovníkov }
    nasledovnik[posledny_stlpec[i]] := prvy_stlpec[i];
  ch := prvy_stlpec[riadok]; { výpis }
  for i := 1 to dlzka do begin
    write (ch);
    ch := nasledovnik[ch];
  end; writeln;
end.
```

P-1-4

Najjednoduchšie riešenie je prejsť pole prvok po prvku, avšak takéto riešenie je pomerne pomalé a nijakým spôsobom nevyužíva, že je pole zotriedené. Ponúka sa nám myšlienka binárneho vyhľadávania. Skúsime teda spraviť jeho reverzibilnú verziu. Binárne vyhľadávanie býva tradične implementované ako while-cyklus. Takýto cyklus však nemáme k dispozícii, preto si budeme musieť pomôcť rekurziou.

Zavedieme si podprocedúru `Hladaj(var l,p : word)`, ktorá bude hľadať hodnotu `co` v úseku `X[l]`, `X[l+1]` až `X[p]` a výsledok pripočíta k premennej `kde`. Bude fungovať tak, že najskôr si spočíta pozíciu `m` prostredného prvku zadaného úseku (ak má úsek párnu dĺžku, zaokrúhlime nadol) a podľa hodnoty príslušného prvku zistí, v ktorej polovici úseku má hľadanie

pokračovať: ak `X[m]<co`, tak od `m+1` po `p`, ak `X[m]>co`, tak od `l` po `m-1`. Na tento úsek sa rekurzívne zavoláme. Po návrate z rekurzie nesmieme zabudnúť `m` vynulovať.

Ak niekedy nastane pri porovnávaní rovnosť, práve sme hodnotu `co` našli, pripočítame `m` ku `kde` a už sa hlbšie nevnárame. Ak dospejeme v rekurzii k úseku nulovej dĺžky (`r<l`), hodnota `co` sa v poli nenachádza, preto sa už len vynoríme z rekurzie bez toho, aby sme menili obsah premennej `kde`.

Podľa tohoto algoritmu už ľahko napíšeme program, kvôli prehľadnejšiemu zápisu používame príkaz `wrap`. Z takto zapísaného programu je zjavná jeho reverzibilnosť.

Vždy, keď sa v rekurzii vnoríme o úroveň hlbšie, zmenší sa prehľadávaný úsek na polovicu, takže po najviac $\log_2 n$ volaniach hľadanú hodnotu buď nájdeme, alebo zistíme, že v poli nie je. Na každé vnorenie spotrebujeme konštantne veľa pamäte. Preto časová aj pamäťová zložitosť je $O(\log n)$, čiže logaritmická.

```
procedure Najdi(var n : word; var X : array [1..n] of word; var co, kde : word);
var one : word;

  procedure Hladaj(var l, r : word);
  var m : word;
  begin
    if l <= r then { úsek nie je prázdny }
      wrap m += (l + r) div 2 { spočítame stred }
      on
        if X[m] = co then { našli sme }
          kde += m
        else
          if X[m] < co then begin { postúpime do pravého úseku }
            wrap m += 1 on Hladaj(m, r)
          end else begin { do ľavého }
            wrap m -= 1 on Hladaj(l, m)
          end
  end;

begin
  wrap one += 1
  on Hladaj(one, n)
end;
```

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY
52. ROČNÍK MATEMATICKEJ OLYMPIÁDY
Riešenia 1. kola kategórie P

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