**Slovenská komisia Matematickej olympiády**
MFF UK, Mlynská dolina, 842 48 Bratislava

50. ročník Matematickej olympiády, školský rok 2000/2001

# Riešenia úloh I. kola kategórie P

**Tento pracovný materiál nie je určený priamo žiakom stredných škôl – účastníkom olympiády.** Má pomôcť učiteľom na školách pri príprave konzultácií a pracovných seminárov pre riešiteľov súťaže, členom oblastných výborov MO slúži ako podklad pre opravovanie úloh domáceho kola MO kategórie P. Žiakom je možné tieto komentáre poskytnúť až po termíne stanovenom pre odovzdanie riešení úloh domáceho kola MO kategórie P ako informáciu, ako bolo možné úlohy správne riešiť, a pre ich odbornú prípravu na účasť v oblastnom kole súťaže.

## P-I-1

*(Jan Kára)*

Je jasné, že za každé dva uzly hĺbky $K, K > 0$ musí existovať jeden uzol hĺbky $K - 1$, ktorý nie je listom – každé dva uzly musíme pod nejaký uzol „zavesiť“. Pretože naviac pod každý uzol môžeme zavesiť iba žiadny alebo dva uzly, musí byť počet uzlov hĺbky $K$ párny. Vyššie uvedené pozorovania nám už dávajú návod na zostrojenie algoritmu. Pre každú hĺbku si budeme pamätať počet listov a počet ostatných uzlov. Budeme postupovať od uzlov s najväčšou hĺbkou. Pre každú hĺbku $K, K > 0$ skontrolujeme, či je počet uzlov v nej párny. Ak nie je, strom neexistuje. Ak je počet uzlov párny, za každé dva uzly umiestnené v strome v danej hĺbke pridáme do predchádzajúcej hĺbky jeden uzol. Keď sa takto dostaneme až k uzlom hĺbky 0, stačí overiť, či v této hĺbke leží práve jeden uzol, ako sa vyžaduje v de nícii binárneho stromu. Ak nie je, strom neexistuje. To plynie z toho, že ak neexistuje uzol hĺbky 0, nebol zadaný žiadny list, a teda strom nemôže mať žiadne uzly. To je ale v spore s požiadavkou, že každý strom musí mať aspoň koreň. Ak je uzlov hĺbky 0 naopak viac, strom neexistuje, pretože všetky uzly, ktoré sme pridávali, boly vynútené a každý strom s danými počami listov teda musí mať v jednotlivých hĺbkach aspoň toľko uzlov ako náš strom. Ak existuje práve jeden uzol hĺbky 0, jednoducho už zo spočítaných počtov listov a ostatných uzlov v jednotlivých hĺbkach vytvoríme požadovaný zápis stromu. Jednoducho vypíšeme toľko `L`, koľko je počet listov danej hĺbky, a toľko `U`, koľko je počet ostatných uzlov danej hĺbky. Algoritmus má časovú i pamäťovú zložitosť $O(N)$. Program je priamou implementáciou algoritmu.

```
program STROMY;
const
  MAXV = 10000;
var
  Vrcholu : Array[0..MAXV, 1..2] of Word; {Počty listov a nelistov
                                           jednotlivých hĺbok}
  V : Word;                    {Počet listov}
  Existuje : Boolean;          {Môže zadaný strom existovať?}
```

```
procedure Vstup;
{Načíta vstup zo súboru}
var
  Inp : Text;
  i, H : Word;
begin
  Assign(Inp, 'stromy.in');
  Reset(Inp);
  ReadLn(Inp, V);
  for i := 1 to V do begin
    Vrcholu[i,1] := 0;
    Vrcholu[i,2] := 0;
  end;
  for i := 1 to V do begin
    Read(Inp, H);
    if H >= V then begin
      Existuje := False;
      break;
    end;
    Inc(Vrcholu[H,1]);
  end;
  Close(Inp);
end;

procedure Vypis(Existuje : Boolean);
{Vypíše nájdený strom alebo správu, že neexistuje}
var
  Out : Text;
  i,j : Word;
begin
  Assign(Out, 'stromy.out');
  Rewrite(Out);
  if Existuje then
    for i := 0 to V-1 do begin
      for j := 1 to Vrcholu[i,2] do
        Write(Out, 'U');
      for j := 1 to Vrcholu[i,1] do
        Write(Out, 'L');
      WriteLn(Out);
    end
  else
    WriteLn(Out, 'Zodpovedajuci strom neexistuje.');
  Close(Out);
end;

function NajdiStrom : Boolean;
{Pokúsi se nájsť strom s danými hĺbkami listov}
var
  i : Word;
begin
  {Prejdeme všetky možné hĺbky listov}
  for i := V-1 downto 0 do begin
    {Je počet vrcholov na nejakej úrovni nepárny?}
```

```
    if (Vrcholu[i+1,1] + Vrcholu[i+1,2]) mod 2 = 1 then begin
      Existuje := False;       {Úroveň nad nami nemôže byť korektná ...}
      break;
    end;
    {Pod každým vrcholom visia vždy práve dva vrcholy}
    Vrcholu[i,2] := (Vrcholu[i+1,1] + Vrcholu[i+1,2]) div 2;
  end;
  {Je na prvej úrovni viac než jeden vrchol?}
  if Existuje and (Vrcholu[0,1] + Vrcholu[0,2] > 1) then
    Existuje := False;
end;

begin
  Existuje := True;
  Vstup;
  if Existuje then
    NajdiStrom;
  Vypis(Existuje);
end.
```

## P-I-2 *(Jan Kára)*

Algoritmus riešiaci túto úlohu sa dá rozdeliť do troch fáz. V prvej fáze sa pre každé mesto spočíta, aká je jeho vzdialenosť od nepriateľom obsadených miest (v zmysle de nície uvedenej v zadaní). V druhej fáze sa zistí, akú maximálnu vzdialenosť od nepriateľských miest dokážeme udržať pri ceste z počiatočného do cieľového mesta. V tretej fáze potom nájdeme najkratšiu z trás vedúcich z počiatočného do cieľového mesta, ktoré udržujú spočítanú vzdialenosť.

Prvá fáza: Vzdialenosť od obsadených miest budeme hľadať pomocou prehľadávania do šírky. Pre každé mesto si budeme udržovať informáciu, či sme v ňom už boli (na počiatku bude nastavené práve u všetkých obsadených miest) a jeho vzdialenosť od nepriateľa. Pre mestá obsadené nepriateľom bude táto vzdialenosť rovná 0. Ďalej si budeme udržovať rad (frontu) miest na spracovanie, do ktorého na začiatku uložíme všetky nepriateľské mestá. V každom kroku výpočtu vždy vezmeme jedno mesto z radu a u všetkých jeho susedov, v ktorých sme doteraz neboli, nastavíme vzdialenosť o jedna väčšiu, než je vzdialenosť vybraného mesta. U všetkých týchto susedov tiež označíme, že sme v nich už boli, a pridáme ich na koniec radu. Prvá fáza výpočtu končí, keď sa vyprázdni rad. Vtedy sme prešli všetky mestá a určili sme vzdialenosť každého z nich od nepriateľa.

Druhá fáza: V tejto fáze si budeme udržovať radov hneď niekoľko, pre každú vzdialenosť od nepriateľských miest jeden. Ďalej si pre každé mesto budeme zaznamenávať, či sme v ňom už boli. Tiež si budeme pamätať doposiaľ najväčšiu nájdenú vzdialenosť, ktorú dokážeme udržať od nepriateľa. Na začiatku nastavíme udržateľnú vzdialenosť od nepriateľa na hodnotu vzdialenosti kráľovského mesta od nepriateľa a toto mesto vložíme do radu pre príslušnú vzdialenosť. Pre toto mesto takisto nastavíme, že sme v ňom už boli. Výpočet prebieha tak, že postupne vyberáme mestá z radu pre aktuálnu udržateľnú vzdialenosť, dokým sa tento rad nevyprázdni. Keď sa rad vyprázdni, znížime udržateľnú vzdialenosť

o jedna a opäť začneme vyberať mestá z príslušného radu. Vždy, keď vezmeme nejaké mesto z radu, prejdeme všetkých jeho susedov, u doteraz nenavštívených z nich nastavíme príznak, že už sme ich navštívili, a pridáme ich do radu – ak je vzdialenosť takéhoto mesta od nepriateľských miest väčšia ako aktuálna udržateľná vzdialenosť, pridáme vrchol do radu zodpovedajúceho aktuálnej udržateľnej vzdialenosti, inak mesto pridáme do radu zodpovedajúceho jeho vzdialenosti od nepriateľských miest. Druhá fáza končí hneď ako vyberieme z radu cieľové mesto. Aktuálna udržateľná vzdialenosť je potom výslednou udržateľnou vzdialenosťou.

Tretia fáza: Táto fáza predstavuje opäť prosté prehľadávanie do šírky. Pre každé mesto si pamätáme, či sme v ňom už boli, a ak áno, zaznamenáme si to mesto, z ktorého sme do neho prišli. Opäť používame rad na doteraz nespracované mestá. Na začiatku vložíme do radu cieľové mesto. U neho nastavíme, že sme v ňom už boli, a ako jeho predchodcu nastavíme jeho samé. V každom kroku výpočtu potom vezmeme jedno mesto z radu a prejdeme všetkých jeho susedov. Každého suseda, ktorého sme doteraz nenavštívili a ktorého vzdialenosť od nepriateľských miest je väčšia alebo rovná výslednej udržateľnej vzdialenosti, označíme ako navštíveného a pridáme ho na koniec radu. Tiež u neho ako mesto, z ktorého sme prišli, nastavíme práve vybrané mesto. Prehľadávanie končí vo chvíli, keď je z radu vybrané počiatočné (kráľovské) mesto. Potom už len prejdeme cestu z počiatočného do cieľového mesta (to je veľmi jednoduché vďaka odkazom na mestá, odkiaľ sme do nich pri prehľedávaní prišli) a cestu vypíšeme.

Algoritmus má časovú zložitosť $O(M+N)$, kde $M$ je počet ciest a $N$ je počet miest.

Správnosť algoritmu budeme ukazovať opäť po fázach. To, že algoritmus spočíta správne vzdialenosti od nepriateľských miest v prvej fáze, plynie z nasledujúceho: Na počiatku majú všetky vrcholy so vzdialenosťou nula túto vzdiaenosť priradenú. V okamihu, keď sú spracované všetky vrcholy vzdialenosti nula, prešli sme všetkých ich susedov, priradili sme im vzdialenosť jedna a zaradili ich do radu. Pretože iné vrcholy vzdialenosť jedna mať nemôžu, je vzdialenosť jedna priradená práve všetkým správnym vrcholom. Túto úvahu ide jednoducho zovšeobecniť pre ľubovoľnú vzdialenosť $D$. Prehľadávanie teda skutočne určí vzdialenosti od nepriateľských miest správne.

V druhej fáze sa správne spočíta maximálna udržateľná vzdialenosť od obsadených miest. Sledujeme v nej totiž súbežne všetky možné trasy vedúce z počiatočného mesta tak dlho, kým dokážeme udržať vzdialenosť počiatočného mesta (výsledná vzdialenosť od nepriateľa zrejme nemôže byť väčšia než vzdialenosť počiatočného mesta). Keď už neexistuje mesto s dostatočne veľkou vzdialenosťou, do ktorého by sme mohli ísť, znížime udržateľnú vzdialenosť o jedna. Všetky vrcholy so vzdialenosťou o jedna nižšou, do ktorých ss dokážeme dostať cez vrcholy s doterajšou udržateľnou vzdialenosťou, máme už pripravené v príslušnom rade a začneme teda prehľadávať z nich. Pretože udržateľnú vzdialenosť znižujeme až keď sme sa už dostali všade, kam to bolo možné, jej výsledná hodnota bude zrejme najvyššia možná.

To, že v tretej fáze nájdeme najkratšiu trasu s danou vzdialenosťou, je zrejmé.

Robíme totiž jednoduché prehľadávanie do šírky s tým, že ignorujeme mestá s príliš malou vzdialenosťou od nepriateľa. Nájdeme teda určite trasu s dostatočnou vzdialenosťou od nepriateľa. Skutočnosť, že to bude trasa najkratšia možná, plynie z vlastností prehľadávania do šírky uvedených v prvej časti dôkazu.

Program je priamou implementáciou uvedeného algoritmu.

```
program POSOL;
const
  MAXV = 100;                  {Maximálny počet miest}
type
  MestT = record
            Byl : Boolean;     {Navštívili sme už pri poslednom
                                prehľedávaní toto mesto}
            HC : Byte;         {Počet ciest z daného mesta}
            H : Array[1..MAXV] of  Byte;     {Susedia daného mesta}
            NepritelD : Byte;  {Vzdialenosť od mesta obsadeného nepriateľom}
          end;
  MestP = Array[1..MAXV] of Byte;   {Typ pre pole s číslami miest}
var
  VC : Byte;                   {Počet miest}
  V : Array[1..MAXV] of MestT; {Jednotlivé mestá}
  NVC : Byte;                  {Počet miest obsadených nepriateľom}
  NV : MestP;                  {Čísla miest obsadených nepriateľom}
  Start, Cil : Byte;           {Počiatočné a cieľové mesto}

procedure Nacti;
var
  Inp : Text;
  i : Word;
  M : Word;                    {Počet všetkých existujúcich ciest}
  A, B : Byte;                 {Počiatočné a koncové mesto cesty}
begin
  Assign(Inp, 'posel.in');
  Reset(Inp);
  Read(Inp, VC);
  for i := 1 to VC do
    V[i].HC := 0;
  ReadLn(Inp, M);
  for i := 1 to M do begin
    ReadLn(Inp, A, B);
    {Pridáme cestu k obom mestám}
    Inc(V[A].HC);
    V[A].H[V[A].HC] := B;
    Inc(V[B].HC);
    V[B].H[V[B].HC] := A;
  end;
  {Načítame obsadené mestá}
  ReadLn(Inp, NVC);
  for i := 1 to NVC do
    Read(Inp, NV[i]);
  {Načíta počiatočné a cieľové mesto}
  Read(Inp, Start);
  ReadLn(Inp, Cil);
```

```
  Close(Inp);
end;

procedure NeprVzdal;
{Spočíta vzdialenosť jednotlivých miest od nepriateľských}
var
  S, N : Byte;                  {Prvé a posledné mesto v rade}
  F : Array[1..MAXV] of Byte;  {Rad miest}
  A : Byte;                     {Aktuálne mesto}
  I : Byte;
begin
  {Inicializujeme hodnoty Byl}
  for i := 1 to VC do begin
    V[i].Byl := False;
    V[i].NepritelD := VC;
  end;
  N := 0;
  S := 1;
  {Naplníme rad nepriateľskými mestami}
  for i := 1 to NVC do begin
    Inc(N);
    F[N] := NV[i];
    V[NV[i]].Byl := True;
    V[NV[i]].NepritelD := 0;
  end;
  {Kým je niečo v rade}
  while N >= S do begin
    A := F[S];
    Inc(S);
    for i := 1 to V[A].HC do begin
      if not V[V[A].H[i]].Byl then begin
        {Nastavíme vzdialenosť od nepriateľa}
        V[V[A].H[i]].Byl := True;
        V[V[A].H[i]].NepritelD := V[A].NepritelD + 1;
        {Pridáme mesto do radu}
        Inc(N);
        F[N] := V[A].H[i];
      end;
    end;
  end;
end;

function ZjistiMinVzdal : Byte;
{Zistí, akoú vzdialenosť si dokážeme udržať od nepriateľa}
var
  S : Array[1..MAXV] of Byte;  {Začiatky jednotlivých radov}
  N : Array[1..MAXV] of Byte;  {Posledné mestá v jednotlivých radoch}
  F : Array[1..MAXV, 1..MAXV] of Byte;  {Rady}
  A : Byte;                     {Aktuálne mesto}
  ActD : Byte;                  {Aktuálna vzdialenosť od nepriateľa}
  I : Byte;
begin
  for i := 1 to VC do begin
    S[i] := 1;
```

```
    N[i] := 0;
    V[i].Byl := False;
  end;
  {Uložíme do radu počiatočné mesto}
  Inc(N[V[Start].NepritelD]);
  F[V[Start].NepritelD,N[V[Start].NepritelD]] := Start;
  V[Start].Byl := True;
  {Prejdeme všetky možné vzdiaenosti}
  for ActD := V[Start].NepritelD downto 0 do begin
    {Kým je vo fronte niečo na spracovanie}
    while S[ActD] <= N[ActD] do begin
      A := F[ActD,S[ActD]];
      if A = Cil then begin    {Našli sme cestu až do cieľového mesta?}
        ZjistiMinVzdal := ActD;
        break;
      end;
      Inc(S[ActD]);
      for i := 1 to V[A].HC do begin
        if not V[V[A].H[i]].Byl then begin  {Ešte sme v tomto meste neboli?}
          V[V[A].H[i]].Byl := True;
          if V[V[A].H[i]].NepritelD >= ActD then begin
                                              {Môžeme do neho vojsť teraz?}
            {Pridáme mesto do radu}
            Inc(N[ActD]);
            F[ActD,N[ActD]] := V[A].H[i];
          end
          else begin           {Do mesta teraz vstúpiť nemôžeme}
            {Pridáme ho do príslušného radu do budúcna}
            Inc(N[V[V[A].H[i]].NepritelD]);
            F[V[V[A].H[i]].NepritelD,N[V[V[A].H[i]].NepritelD]] := V[A].H[i];
          end;
        end;
      end;
    end;
    if A = Cil then
      break;
  end;
end;

procedure VypisCestu(Prev : MestP);
{Vypíše nájdenú trasu}
var
  A : Byte;                    {Aktuálne mesto}
  Out : Text;
begin
  Assign(Out, 'posel.out');
  Rewrite(Out);
  {Prejdeme celú nájdenú trasu}
  A := Start;
  while A <> Cil do begin
    Write(Out, A, ' ');
    A := Prev[A];
  end;
  WriteLn(Out, A);
```

```
  Close(Out);
end;

procedure NajdiCestu(D : Byte);
{Nájde najkratšiu trasu cez mestá vzdialenosti aspoň D}
var
  S, N : Byte;                  {Začiatok a koniec radu}
  F : Array[1..MAXV] of Byte;  {Rad na mestá}
  Prev : MestP;                 {Mestá, odkiaľ sme prišli}
  A : Byte;                     {Aktuálne mesto}
  i : Byte;
begin
  for i := 1 to VC do
    V[i].Byl := False;
  {Zaradíme počiatočné mesto do radu}
  S := 1;
  N := 1;
  F[N] := Cil;
  V[Cil].Byl := True;
  Prev[Cil] := Cil;
  while S <= N do begin
    {Vyberieme mesto z radu}
    A := F[S];
    if A = Start then
      break;
    Inc(S);
    for i := 1 to V[A].HC do   {Prejdeme vštky cesty z mesta}
      {Ešte sme v danom meste neboli a je dosť ďaleko od nepriateľov?}
      if (not V[V[A].H[i]].Byl) and (V[V[A].H[i]].NepritelD >= D) then begin
        {Zaradíme mesto do radu}
        V[V[A].H[i]].Byl := True;
        Inc(N);
        F[N] := V[A].H[i];
        Prev[V[A].H[i]] := A;  {Poznamenáme, odkiaľ sme prišli}
      end;
  end;
  VypisCestu(Prev);             {Vypíšeme nájdenú cestu}
end;

begin
  Nacti;       {Načíta vstup}
  NeprVzdal;   {Spočíta vzdialenosti od nepriateľa}
  {Zistí, akú vzdialenosť dokážeme mať od nepriateľa a nájdeme najkratšiu cestu}
  NajdiCestu(ZjistiMinVzdal);
end.
```

## P-I-3 *(Daniel Kráľ)*

Riešenie úlohy rozdelíme na niekoľko častí – najprv sformulujeme nutné a postačujúce podmienky pre to, aby skupina mohla prejsť trasu podľa podmienok v zadaní úlohy, potom vyriešime úlohu a) a nakoniec nájdeme riešenie úlohy b).

Plánovanou trasou sa dá prejsť práve vtedy, keď táto trasa nie je dlhšia ako vzdialenosť, ktorú skupina prejde za deň, t.j. $L \leq K$, alebo keď sú splnené zároveň všetky štyri nasledujúce podmienky:

1. Na trase je aspoň jeden kemp.
2. Vzdialenosť prvého kempu od začiatku trasy je najviac $K$, tj. $l_1 \leq K$.
3. Vzdialenosť ľubovolných dvoch po sebe nasledujúcich kempov nie je väčšia než $K$, tj. $l_{i+1} - l_i \leq K$ pro $1 \leq i \leq N-1$.
4. Vzdialenosť posledného kempu od konca trasy je najviac $K$, tj. $L - l_N \leq K$.

Nutnosť všech uvedených podmienok je zrejmá; pokiaľ sú tieto podmienky splnené, potom plán cesty, v ktorom skupina bude cestovať $N+1$ dní a $i$-tý den prespí v $i$-tom kempe, spĺňa podmienky zo zadania úlohy.

Teraz vyriešime úlohu a). Na chvíľu si predstavme, že sme každému kempu priradili číslo $d_i$, ktoré udáva, koľký deň najskôr môžeme do tohoto kempu prijsť. Číslo $d_i$ pridenené kempu $i$ zrejme splňuje jednu z následujúcich dvoch podmienok:

1. Buď je $d_i = 1$ a $l_i \leq K$ – do kempu sa dá prijsť hneď prvý deň.
2. Existuje $j < i$ také, že $l_i - l_j \leq K$ a $d_j = d_i - 1$ (do $i$-teho kempu prijdeme tak, že deň pred tým prespíme v $j$-tom kempe), ale neexistuje $j < i$ také, že $l_i - l_j \leq K$ a $d_j < d_i - 1$ (inak by bolo možné do $j$-tého kempu prijsť už skôr).

Podľa týchto podmienok by bolo možné spočítať všetky $d_i$ v čase $O(N)$, náš program však $d_i$ počítať nebude. Plán cesty spĺňajúci podmienku a) by mohol vyzerať napríklad tak, že $h$-tý deň skupina prespí v $i$-tom kempe, pokiaľ $d_i = h$ a $d_{i+1} = h+1$; skupina naviac prespí v $N$-tom kempe, pokiaľ plán cesty bez tohto kempu nespĺňa podmienku obmedzujúcu maximálnu vzdialenosť, ktorú je možné prejsť za jeden deň. Budeme priamo vytvárať takýto plán cesty – ak doposiaľ vytvorený plán cesty končí kempom vo vzdialenosti $l$ od začiatku výletu a $L - l > K$ (nedá sa bez prespania prijsť na koniec trasy), potom ďalší deň skupina prespí v $i$-tom kempe, pokiaľ $l_i - l \leq K$ a $i$ je maximálne s touto vlastnosťou. Vytvoriť program pracujúci podľa práve popísaného postupu je triviálne; časová zložitosť algoritmu je $O(N)$.

Ďalej vyriešime úlohu b). Každému kempu priradíme číslo $e_i$, ktoré určuje, koľko by cyklisti museli zaplatiť za prespanie na ceste z $i$-teho kempu na koniec výletu (vrátane poplatku za prespanie v $i$-tom kempe). Čísla $e_i$ náš algoritmus spočíta postupne pre $i = N$ až $i = 1$; okrem týchto čísel si pre každý kemp uložíme informáciu, do ktorého následujúceho kempu sa z neho máme vybrať, aby sme za nocľah zaplatili optimálnu cenu $e_i$. Čísla $e_i$ sa dajú spočítať podľa nasledujúceho predpisu:

1. Ak $L - l_i \leq K$, potom $e_i = c_i$; z $i$-tého sa kempu dá prijsť na koniec trasy behom jedného dňa.
2. Ak $L - l_i > K$, potom $e_i = min_j(c_i + e_j) = c_i + min_j e_j$, kde minimum sa počíta cez všetky $j > i$ také, že $l_j - l_i \leq K$; to $j$, kde sa nadobúda minimum, určuje poradie kempu, v ktorom prespíme ten deň, keď vyjdeme z $i$-teho kempu.

Prvý deň, ak $L > K$, prespíme v kempe s číslom $i$ s minimálnym $e_i$, pre ktorý platí $l_i \leq K$. Ako bude algoritmus pracovať je teraz už jasné. Zostáva ešte určiť, ako rýchlo sa dá nájsť $j$, ktoré minimalizuje vzťah v druhom bode.

Na rýchle nájdenie indexu $j$ použijeme dátovú štruktúru, ktorá sa nazýva halda. Halda je dátová štruktúra, ktorá umožňuje v konštantnom čase určiť najmenší z prvkov v nej; prvok do haldy pridať alebo vybrať najmenší prvok dokáže v čase logaritmickom od počtu prvkov obsiahnutých v halde. V halde si budeme udržovať čísla kempov zotriedené podľa hodnôt $e_i$; na začiatku budeme mať v halde naviac koniec trasy, ktorého hodnotu budeme považovať za rovnú nule (bude najmenším prvkom haldy). Nájdenie vhodného $j$ bude probiehať nasledovne: Zistíme, či najmenší prvok haldy je od $i$-teho kempu vzdialený najviac o $K$ – ak áno, našli sme príslušné $j$, v opačnom prípade z haldy odstránime tento prvok a celý postup zopakujeme. Potom do haldy priradíme $i$-ty kemp. Za predpokladu logaritmického času pridania prvku do haldy a vybratia najmenšieho prvku z haldy je celková doba behu algoritmu $O(N \log N)$.

Haldu budeme reprezentovať v poli. Ak bude v halde $n$ prvkov, potom jej prvky budú uložené v poli na poziciách s číslami 0 až $n-1$. Pre prvok $x$ s indexom $k$ budeme prvky na poziciách $2k+1$ a $2k+2$ nazývať synmi prvku $x$ a prvok $x$ budeme nazývať otcom týchto prvkov. Všimnite si, že každý prvok okrem prvku na pozícii 0, má práve jedného otca. Pri práci s haldou budeme dodržovať nasledujúci invariant: Každý prvok je väčší ako jeho otec. Najmenším prvkom v halde je preto prvok na nultej pozícii; zistenie najmenšieho prvku haldy sa dá teda spraviť v konštantnom čase. Pridanie prvku do haldy bude prebiehať nasledovne: Ak je v halde $n$ prvkov, potom nový prvok umiestime na pozíciu $n$; ak je jeho otec väčší, vymeníme pridávaný prvok s jeho otcom a celý postup opakujeme tak dlho, pokiaľ nový prvok nie je nultým prvkom alebo jeho otec nie je menší než on. V každom kroku sa index nového prvku v poli zmenší aspoň na polovicu a teda sa po logaritmicky veľa krokoch zastavíme. Odobranie prvku z haldy bude probiehať podobne: Ak je v halde $n$ prvkov, potom najmenší prvok haldy nahradíme prvkem z $(n-1)$-tej pozície; tento prvok porovnáme s oboma jeho synmi a prípadne ho vymeníme s menším z oboch jeho synov. Skončíme, ak je tento prvok menší než obaja jeho synovia. Pretože sa v každom kroku posunieme na prvok s aspoň dvojnásobným indexom, odobranie najmenšieho prvku z haldy bude trvať čas logaritmický od počtu prvkov v nej.

```
program vylet;
const MAXN=10000;
var l: array[1..MAXN] of word;  { vzdialenosti kempov }
    c: array[1..MAXN] of word;  { ceny za nocľah }
    delka: word;                { celková dĺžka výletu }
    K: word;                    { maximálna vzdálenostosť denne }
    N: word;                    { počet kempov }
    f: text;


function ma_reseni:boolean;
var i: word;
begin
```

```
  if (delka<=K) then
    ma_reseni:=true
  else
    begin
      ma_reseni:=false;
      if (l[1]>K) then exit;
      for i:=1 to N-1 do if (l[i+1]-l[i]>K) then exit;
      if (delka-l[N]>K) then exit;
      ma_reseni:=true;
    end
end;

procedure vyres_a;
var opt: array[1..MAXN] of word;  { optimálne riešenie }
    optpocet: word;               { počet kempov v poli opt }
    optcena: word;                { cena za prespanie v kempoch podľa poľa opt }
    vzdalenost: word;             { vzdialenosť posledného kempu v poli opt }
    i: word;
begin
  vzdalenost:=0;
  optpocet:=0;
  optcena:=0;
  i:=1;
  while (i<N) do
    begin
      if (l[i+1]-vzdalenost>K) then
        begin
          inc(optpocet);
          opt[optpocet]:=i;
          inc(optcena,c[i]);
          vzdalenost:=l[i];
        end;
      inc(i)
    end;
  if (delka-vzdalenost>K) then
    begin
      inc(optpocet);
      opt[optpocet]:=i;
      inc(optcena,c[i]);
    end;
  assign(f,'VYLET-A.OUT');
  rewrite(f);
  writeln(f,optpocet,' ',optcena);
  for i:=1 to optpocet do writeln(f,opt[i]);
  close(f)
end;

procedure vyres_b;
var halda: array[0..MAXN] of word;        { pomocná halda }
    vhalde: word;                         { veľkosť pomocnej haldy }
    optcena: array[0..MAXN] of word;      { minimálna cena cesty
                                            z daného kempu do konca trasy }
    nasledujici: array[0..MAXN] of word;  { nasledujúcí kemp na opt. trase }
 { Označenie: 1 až N - kempy, 0 - konec trasy }
```

```
function cena(p:word):word; { vracia opt. cenu cesty z kempu p do konce trasy }
begin
  if p=0 then
    cena:=0
  else
    cena:=optcena[p]
end;
function vzdalenost(p:word):word; { vracia vzdialenosť kempu p }
begin
  if p=0 then
    vzdalenost:=delka
  else
    vzdalenost:=l[p]
end;
procedure vyjmi; { vyberie najmenší prvok z haldy }
var i, j, k: word;
begin
  dec(vhalde);
  halda[0]:=halda[vhalde];
  i:=0;
  while (((i shl 1)+1<vhalde) and (cena(halda[i])>cena(halda[(i shl 1)+1]))) or
        (((i shl 1)+2<vhalde) and (cena(halda[i])>cena(halda[(i shl 1)+2]))) do
   begin
     j:=i; { vybereme větev, do které půjdeme }
     if (((i shl 1)+1<vhalde) and (cena(halda[j])>cena(halda[(i shl 1)+1]))) then
       j:=(i shl 1)+1;
     if (((i shl 1)+2<vhalde) and (cena(halda[j])>cena(halda[(i shl 1)+2]))) then
       j:=(i shl 1)+2;
     { ... a vymeníme prvky }
     k:=halda[i]; halda[i]:=halda[j]; halda[j]:=k
   end
end;
procedure pridej(p:word); { pridá kemp p do haldy }
var i, j, k: word;
begin
  i:=vhalde; halda[i]:=p; inc(vhalde);
  while (i>0) and (cena(halda[i])<cena(halda[(i-1) shr 1])) do
    begin
      j:=(i-1) shr 1;
      k:=halda[j]; halda[j]:=halda[i]; halda[i]:=k;
      i:=j
    end
end;
var i,j: word;
begin
  vhalde:=1;
  halda[0]:=0;
  for i:=n downto 1 do
    begin
      while (vzdalenost(halda[0])-l[i]>K) do vyjmi;
      optcena[i]:=c[i]+cena(halda[0]);
      nasledujici[i]:=halda[0];
      pridej(i);
    end;
```

```
  while (vzdalenost(halda[0])>K) do vyjmi;
  optcena[0]:=cena(halda[0]);
  nasledujici[0]:=halda[0];
  { spočítame počet kempov na ceste }
  i:=0; j:=0;
  while (nasledujici[i]<>0) do
    begin
      inc(j);
      i:=nasledujici[i];
    end;
  { vypíšeme riešenie }
  assign(f,'VYLET-B.OUT');
  rewrite(f);
  writeln(f,j,' ',optcena[0]);
  i:=0;
  while (nasledujici[i]<>0) do
    begin
      i:=nasledujici[i];
      writeln(f,i);
    end;
  close(f)
end;

var i: word;
begin
  assign(f,'VYLET.IN');
  reset(f);
  readln(f,delka,K,N);
  for i:=1 to N do readln(f,l[i],c[i]);
  close(f);
  if not ma_reseni then
     begin
      assign(f,'VYLET-A.OUT');
      rewrite(f);
      writeln(f,'Trasu nemozno prejst.');
      close(f);
      assign(f,'VYLET-B.OUT');
      rewrite(f);
      writeln(f,'Trasu nemozno prejst.');
      close(f);
    end
   else
    begin
      vyres_a;
      vyres_b;
    end
end.
```

## P-I-4 *(Martin Mareš)*

**a)** Majme zadané nejaké dvojkové číslo $\langle x_1, \ldots, x_n \rangle$, o ktorom máme rozhodnúť, či je deliteľné piatimi. Zostrojíme sadu dlaždíc, ktorými bude možné vydláždiť iba stenu s jediným riadkom, a to tak, aby farba pravej hrany $i$-tej dlaždice

(tú budeme značiť $p_i$) odpovedala zvyšku po delení dvojkového čísla $\langle x_1, \ldots, x_i \rangle$ piatimi. Keď naviac zvolíme farbu pravého okraja steny tak, aby zodpovedala zvyšku 0, pôjde stena vydláždiť práve vtedy, keď je zadané číslo deliteľné piatimi, a to je presne to, čo potrebujeme.
Použijeme dlaždice nasledujúcich typov:

$$T = \left\{ \begin{array}{|c|}\hline y \backslash\, / \\ x \quad z \\ \bullet \\ \hline \end{array} \;;\; 0 \leq x \leq 4, 0 \leq y \leq 1, z = (2x + y) \bmod 5 \right\},$$

ľavý aj pravý okraj budoú mať farbu 0 a dolný okraj farbu •. Keďže farba • sa nevyskytuje na hornej hrane žiadnej dlaždice, musí byť každé korektné vydláždenie tvorené jediným riadkom. Zostáva dokázať, že farby pravých hrán dlaždíc zodpovedajú zvyškom, čo urobíme indukciou:

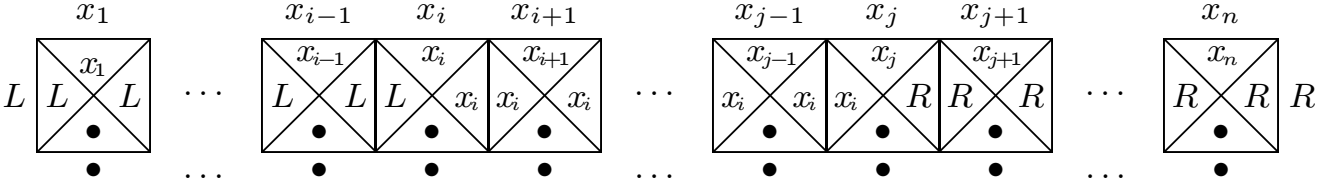
- $p_1 = \langle x_1 \rangle = \langle x_1 \rangle \bmod 5$ (existuje práve jedna dlaždica, ktorá môže být na prvom políčku – tá, ktorá má na ľavom okraji nulu a na hornom okraji $x_1$).
- Ak je $p_i = \langle x_1, \ldots, x_i \rangle \bmod 5$, môže byť na $i+1$-vom políčku iba jediná dlaždica (majúca na ľavom okraji $p_i$ a na hornom okraji $x_{i+1}$) a jej pravý okraj má farbu $p_{i+1} = (2p_i + x_{i+1}) \bmod 5 = (2(\langle x_1, \ldots, x_i \rangle \bmod 5) + x_{i+1}) \bmod 5 = (2 \langle x_1, \ldots, x_i \rangle + x_{i+1}) \bmod 5 = \langle x_1, \ldots, x_{i+1} \rangle \bmod 5$.

Z toho plynie, že popísaný dlaždicový program rieši zadanú úlohu so zložitosťou $O(1)$.

**b)** Použijeme nasledujúce typy dlaždíc:

- „ľavé“ dlaždice $\left\{ \begin{array}{|c|}\hline x \backslash\, / \\ L \quad x \\ \bullet \\ \hline \end{array} \;;\; 0 \leq x \leq 9 \right\}$
- „pravé“ dlaždice $\left\{ \begin{array}{|c|}\hline y \backslash\, / \\ x \quad R \\ \bullet \\ \hline \end{array} \;;\; 0 \leq x, y \leq 9, x \neq y \right\}$
- „opakovacie“ dlaždice $\left\{ \begin{array}{|c|}\hline x \backslash\, / \\ L \quad L \\ \bullet \\ \hline \end{array}, \begin{array}{|c|}\hline x \backslash\, / \\ R \quad R \\ \bullet \\ \hline \end{array}, \begin{array}{|c|}\hline x \backslash\, / \\ y \quad y \\ \bullet \\ \hline \end{array} \;;\; 0 \leq x, y \leq 9 \right\}$

Ľavý okraj steny ofarbíme farbou $L$, pravý farbou $R$ a spodný farbou •. Keďže farba • sa nevyskytuje na hornej hrane žiadnej dlaždice, musí byť každé korektné vydláždenie tvorené jediným riadkom.
Z toho, ako sme si typy dlaždíc nade novali, ihneď plynie, že každé vydláždenie steny musí vyzerať takto:




(zľava doprava: najprv [možno i prázdna] postupnosť dlaždíc opakujúcich $L$, potom jedna ľavá dlaždica, nasleduje opäť niekoľko opakovacích dlaždíc, jedna

pravá dlaždica a prípadne opakovanie $R$). Také vydláždenie je ale korektné práve vtedy, ak bolo možné nájsť indexy $i$ a $j$ také, že $x_i \neq x_j$ (vďaka de nícii farieb hrán pravej dlaždice), takže náš dlaždicový program rieši zadanú úlohu so zložitosťou $O(1)$.

Poznámka: Naše riešenie využíva fakt, že dlaždicové programy sú nedeterministické (to znamená, že nemajú pevne de novaný priebeh výpočtu a miesto toho pripúšťajú viac rôznych výpočtov s tým, že odpoveďou programu je ÁNO, ak existuje aspoň jeden korektný výpočet) a že nám nedeterminizmus „uhádne“ polohu nejakých dvch rôznych prvkov vstupnej postupnosti.

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY

**50. ROČNÍK MATEMATICKEJ OLYMPIÁDY**

Riešenia úloh prvého kola kategórie P

Vydala IUVENTA – zariadenie pre voľný čas detí, mládeže i dospelých MŠ SR
pre vnútornú potrebu Ministerstva školstva SR
Náklad: 275 výtlačkov
Programom TEX sadzbu pripravil Richard Kráľovič

Autori príkladov: Jan Kára
Daniel Kráľ
Martin Mareš