Korešpondenčný seminár z programovania
XXII. ročník, 2004/05
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporujú: aSc – Applied Software Consultants spol. s r.o.
MICROSTEP-MIS spol. s r.o.

Vzorové riešenia 2. kola letnej časti, kat. Z

Všetko dobré sa raz pominie, nuž sa pominul aj ďalší ročník KSP. Na mladších z vás sa tešíme v budúcom ročníku, na starších na matfyze... no a na najšikovnejších z vás sa tešíme na jesennom sústredku, ktoré bude po všetkých stránkach ešte úžasnejšie ako všetky doteraz :-). Pred tým, než si pozrieš výsledkovú listinu, pozorne si prečítaj všetky vzorové riešenia!

No dobre, stačí aj **po** pozretí na výsledkovku...

KSPáci

1. Zvrátená učiteľka

opravoval MMx
(max. 15 bodov)

Skoro všetky riešenia tohto príkladu boli správne, aj keď niektorí z vás sa pri jeho riešení pekne zamotali. Pozrime sa teraz na vzorové riešenie.

Na vstupe sme dostali permutáciu, ktorá nám nejako preusporiada deti sediace na stoličkách. Našou úlohou je nájsť permutáciu, ktorá preusporiada deti do pôvodného stavu (nazýva sa inverzná). Postupovať budeme nasledovne: Po načítaní prvého čísla A_1 už vieme, že dieťa sediace na prvej stoličke si po tlesknutí sadne na A_1-tu stoličku zľava. Pri retrospektívnom tlesknutí si teda musí sadnúť naspäť, čiže vo výslednej postupnosti musí byť na A_1-tom mieste číslo 1 ($B_{A_1} = 1$). Rovnako budeme postupovať pre všetky ostatné čísla na vstupe. Keďže každé číslo je na vstupe práve raz a sú tam všetky čísla od 1 po n, takýmto postupom dostaneme výslednú permutáciu.

Časová zložitosť tohto algoritmu je $O(N)$, pretože každé číslo na vstupe spracujeme v konštantnom čase. Pamäťová zložitosť je tiež $O(N)$, pretože si potrebujeme pamätať pole s výslednou permutáciou.

Za takéto riešenie sa dalo získať 13 bodov. Za riešenie v čase $O(N \log N)$ bolo 10 bodov a za kvadratické riešenie 9 bodov. Po jednom bode sa dalo získať za popis, ktorý vysvetľoval, prečo váš algoritmus funguje správne a za odhad zložitosti.

Bonusový bod za najlepší názov dostala Lucia Schlosáriková za názov „Záhada Blair Teach alebo Učiteľ tlieska dvakrát“ a Tomáš Pavlíček za názov „Čo sa za mlada na(m)učíš, pri KSP využiješ!“

Listing programu:

```
const max=100;
var b       :array[1..max] of integer;
    n, i, a :integer;

begin
  write('N = '); readln(n);
  for i:=1 to n do
  begin
    read(a);
    b[a]:=i;
  end;
```

```
  for i:=1 to n-1 do write(b[i],' ');
  writeln(b[n]);
end.
```

opravoval KuK
(max. 15 bodov)

2. Z kalendára

Fajn príklad... mal aj myšlienku... škoda, že sa nám ju nepodarilo nejak obstojne sprostredkovať ;-). Väčšina z vás uvažovala (a to nie veľmi chybne) asi takto: načítanie trvá lineárny čas, máme iba jedinkú otázku, tak sa nám zložitosť nepokazí, keď jednoducho prejdeme celý zoznam a vypíšeme, čo treba.

Tak toto ste chceli, Slováci? Nuž pôvodne sme chceli tých otázok o dosť viac. Chyba je teda na našej strane, a preto som hodnotil veľmi umiernene (a ak nie, tak vedzte, že na mňa je to umiernené).

Hodnotenie: Lineárne riešenie „se fším fšudy“ – 15 bodov[1]. Za zlý odhad zložitosti (časovej a pamäťovej) som stŕhal po bode; takisto som výnimočne stŕhal aj za (výnimočne) „grcné“ riešenia; pritom stačilo niečo jednoduché štýlu

```
for i:=1 to n do
  if ((den='*') or (den=db_den[i]))
     and ((mesiac='*') or (mesiac=db_mesiac[i]))
     and ((rok='*') or (rok=db_rok[i])) then treba ho vypísať
```

Taktiež som stŕhal za chýbajúci popis. Vyskytlo sa aj konkurenčné riešenie s veľkým poľom, ktoré však zabúdalo, že sa stávajú aj také veci, že sa narodia viacerí ľudia v jeden deň (napr. také dvojičky). Za to som udeľoval základ 11 bodov (a z nich veselo stŕhal ďalej ;-)).

Ďalej sa už budeme venovať poupravenému zadaniu – otázok môže byť vééeľa.

Ako sa to teda mohlo riešiť: V takomto prípade existuje viac dobrých (aj zlých) riešení[2]. Napríklad to, ktoré zopár z vás poslalo: spravíme si veľké trojrozmerné pole,

```
db : array [1..31, 1..12, 1950..2050] of ...
```

takže ľubovoľný dátum vieme nájsť hneď, a to na pozícií [deň, mesiac, rok] – tam bude napísané meno, kto sa vtedy narodil. Keď budeme chcieť napr. vypísať všetkých ľudí, čo sa narodili v danom dni a mesiaci (rok je hviezdička), prejdeme for-cyklom všetky roky, teda budeme sa pozerať na všetky pozície [deň, mesiac, *].

Aby sme ešte oštrili prípad, že sa viacero ľudí narodí jeden deň, nebude v každom políčku iba meno, ale celý zoznam mien ľudí, čo sa vtedy narodili. Nadáva sa tomu spájaný zoznam alebo v iných zemiach aj linked list: okrem samotného mena si vždy zapamätáme aj odkaz na nasledujúceho, takže údaje tvoria takého „hada“; niečo také sa väčšinou robí ako cvičenie, keď sa preberajú pointre alebo, ak chcete, ale najskôr asi nie, smerníky; už tak je tento komentár dosť dlhý, tak ak to neviete, niekde si to iniciatívne pozrite.

Jediná chybička krásy na tomto riešení je, že to žerie vééeľa pamäte: okrem naozaj „využitej“ pamäte, kde si pamätáme mená a. pod. potrebujeme ešte $31 \times 12 \times$ [počet možných rokov] políčok tabuľky. Tu sa už len ťažko dá alibisticky priblblo usmievať, že „hm, a fšak

[1] – je pomlčka, mínus vyzerá takto: −; za lineárne riešenie bolo naozaj +15 bodov

[2] to je veta

konštanta“. Podobne je to aj s časovou zložitosťou: jeden dátum nájdeme hneď – s konštantnou zložitosťou; najhoršie, čo sa môže stať je, že sú zadané 3 hviezdičky, prejdeme celú(!) tabuľku, a tam skoro nič. Môžeme sa opäť tváriť, že konštanta, nuž ale, inu, dosť veľká.

Trie: To, čo som vskutku naprogramoval a čo si môžete pozrieť po tomto pokece, je niečo podobné riešeniu „veľkým poľom“ – tu bol jediný vážny problém, že pole bolo naozaj veľké a tak ho trochu zmenšíme. Nadviažem na minulé kolo, kde vám Tonko vysvetlil písmenkové stromy, tzv. trie[3]. Ten sa dá celkom úspešne použiť aj tu: budeme mať tri poschodia/úrovne: jedno poschodie bude rockové, druhé mesačné a to tretie dňové. Trie je strom, ktorý sa skladá z vrcholov; my budeme mať jeden pomyselný koreň, v ktorom začneme – odtiaľ pôjde veľa hrán do rockových vrcholov, z rockových vrcholov pôjdu hrany do mesačných vrcholov a odtiaľ do dňových – v tých už bude iba spájaný zoznam mien, ktoré sa narodili v tomto dátume. Keď budeme chcieť nájsť nejaký konkrétny dátum, prídeme najskôr do vrcholu, ktorý predstavuje daný rok, odtiaľ sa poberieme po hrane do vrchola, ktorý reprezentuje daný mesiac a tam už nájdeme zadaný deň a vypíšeme zoznam mien, ktoré tam nájdeme. Ak je namiesto konkrétneho údaju, napr. konkrétneho mesiaca, zadaná hviezdička, prejdeme for-cyklom všetky hrany, t.j. navštívime všetky mesiace. Časová zložitosť bude rovnaká ako u veľkého poľa, tu sme si nepomohli, ale pamäťová zložitosť bude ozaj lineárna, pričom alibistická konštanta bude menšia ako v poli.

Pre tých, ktorý do toho už trochu vidia: takýto strom vlastne predstavuje ono trojrozmerné pole zo začiatku riešenia: rockové vrcholy sú akoby prvý index, mesačné akoby druhý a denné akoby tretí index do onoho veľkého poľa. Jediný rozdiel je, že v strome vrcholy vytvárame, až keď ich naozaj potrebujeme; berte to teda ako príklad, aby sme si precvičili trie, keď sme si ho už teda minule vysvetlili a príklad toho, že keď sa chce, dá sa program dokopať do takého stavu, aby žral len toľko pamäte, koľko potrebuje (čo ale do KSP až tak netreba, bo to zneprehľadňuje zdrojáky).

Haluze: Iný možný postup je rozdeliť si úlohu na osem podúloh: hľadáme konkrétny dátum – deň, mesiac, rok (DMR), alebo taký, čo má niekde hviezdičku (*MR, D*R, DM*), alebo dve (**R, *M*, D**), alebo tri (***). No a teraz si stačí zobrať nejakú namakanú dátovú štruktúru (čítaj „čiernu krabičku“), do ktorej vieme rýchlo vložiť a rýchlo hľadať. Riešenie bude fungovať tak, že budeme mať 8 takýchto krabičiek. Na začiatku do každej vložíme zadané údaje. Keď dostaneme napr. zadaný deň a rok, pričom mesiac je hviezdička, spýtame sa krabičky D*R, ktorá nám dá rýchlo odpoveď. Inými slovami, v každej krabičke budú rovnaké údaje, ale každá sa bude špecializovať na iný druh otázky, aby vedela na tú-ktorú otázku rýchlo odpovedať.

Ostáva poriešiť, ako naprogramujeme tieto krabičky. Jeden jednoduchý spôsob je mať osem polí. Do každého nasúkame zadané údaje a potom prvé pole utriedime podľa dátumu, mesiaca a roku, druhé iba podľa mesiaca a roku, tretie iba podľa dňa a roku atď. Teraz keď budeme chcieť hľadať podľa daného mesiaca a roku, budeme hľadať v druhom poli (to sme podľa mesiaca a roku triedili). Ako to vieme spraviť rýchlo? Spomeňme si na hru „myslím si číslo od 1 do 100; nooo, už hádaj, konečneee, budem ti hovoriť, či viac alebo menej“. Ako je najlepšie hádať, aby sme sa spýtali čo najmenej otázok? Zrejme najlepšie je tipnúť 50, potom 25 alebo 75, podľa toho, či menej alebo viac ako 50 atď., vždy strelíme do polovice (keby sme nešli do polovice, na just by to mohlo byť v tej väčšej časti[4] a potrebovali by sme viac otázok. Nuž a tento štýl hádania môžeme použiť aj pri hľadaní: pozrieme sa na číslo v polovici poľa; ak je väčšie, ako hľadané pokračujeme v prvej polovici, inak v druhej. Tí, ktorí dačo takéto počujú prvýkrát a z tohto krátkeho popisu to nepochopili, nech si to naštudujú odinakiaľ; volá sa to binárne vyhľadávanie alebo binary search; budete to ešte potrebovať...

[3] vraj z anglického retrieval

[4] áno, zarazil som sa pri fráze „väčšej polovice“

Lepšia možnosť ako takúto krabičku naprogramovať sa volá hašovanie[5]. Rôznych dátumov môže byť veľa, zatiaľčo naozajstných záznamov v tabuľke môže byť málo – preto sa kvôli pamäti neoplatí riešenie typu „spravme veľké pole, kde každý dátum bude mať vlastný chlievik“. Hašovanie je práve o tom. Nebudeme mať veľké pole, ale také normálne, aby nám stačilo a budeme mať hašovaciu funkciu (čítaj „mlynček“), ktorý nám dátum pomelie a povie, kam do poľa ho máme uložiť. Keďže je pole také normálne a dátumov tak veľa, občas sa stane, že mlynček dvom dátumom priradí ten istý chlievik v poli – niečo takéto katastrofické sa nazýva kolízia. Možných riešení je viacej, spomeňme napríklad riešenie spájaným zoznamom: v chlieviku nebudeme mať uložené iba jedno meno, ale celý zoznam mien. A aký mlynček je dobrý? Nuž od takej „kvalitnej“ funkcie by sme chceli, aby údaje rovnomerne porozhadzovala po poli, aby kolízie nevznikali „príliš často“. Potom sa dá očakávať, že hašovacia tabuľka bude fungovať veľmi dobre, priam v konštantnom čase.

Svoje tláchanie skúsim podložiť príkladom. Dajme tomu, že vieme, že naša databáza bude potrebovať pojať 1000 údajov (napríklad, hej?). Tak si vezmeme nejaké väčšie prvočíslo (prečo prvočíslo? ... nuž, to je alchýmia), napr. 1021 a spravíme si pole od 0 po 1020. Teraz keď dostaneme dátum, nech d, m, r sú postupne deň mesiac a rok, spravíme najskôr z týchto čísel jedno. Ako? Nuž vieme, že $0 \leq d-1 < 31$ a $0 \leq m-1 < 12$; stačí teda vypočítať napr. $31 \cdot (12 \cdot r + m - 1) + d - 1$ a je to. Toto jedno číslo už iba vydelíme 1021. Zvyšok po delení bude niečo od 0 po 1020 – a to sme chceli. Ešte raz: keď dostaneme dátum, vypočítame z neho jediné číslo a z toho spočítame zvyšok po delení (aby sme boli v rozsahu poľa) – na toto políčko uložíme daný údaj, resp. na tomto políčku budeme hľadať hľadané... Takáto hašovacia tabuľka sa dala použiť pri riešení – mali by sme ich osem a bolo by nám sveta žiť.

Listing programu:

```
unit databaza;
interface

const hocico = 0; { wildcard, hviezdicka }
procedure vloz (meno : string; d, m, r : integer);
procedure hladaj (d, m, r : integer);

implementation

type ptrie = ^trie;
     trie = array [1..31] of pointer;

    plist = ^list;
     list = record
        m : string; { meno }
        n : plist;  { dalsi }
     end;

var db : array [1950..2050] of ptrie;

function novy : ptrie; { vytvori novy vrchol }
var tmp : ptrie;
    i : integer;
begin
    new (tmp); for i := 1 to 31 do tmp^[i] := nil;
    novy := tmp;
end;
```

[5] z anglického hash – hašé (pokrm z vareného mletého mäsa a zemiakov, príp. inej zeleniny)..... aaaale, Kubo, Kubo, zase fabuluješ a mystifikuješ...

```
procedure vloz (meno : string; d, m, r : integer);
var l : plist;
    t : ptrie;
begin
  { ak vrcholy este nejestvuju, vytvorime nove}
  if db[r] = nil then db[r] := novy;
  if db[r]^[m] = nil then db[r]^[m] := novy;
  t := db[r]^[m];
  { vytvorime novy zaznam a zaradime ho pred zoznam }
  new (l); l^.m := meno; l^.n := t^[d];
  { opravime ukazovatel, aby ukazoval na (novy) zaciatok zoznamu }
  t^[d] := l;
end;

procedure vypis (p : plist); { dany je pointer na zoznam }
begin                        { tak ten zoznam vypis, prosim ta }
  while p <> nil do begin
    writeln (p^.m);
    p := p^.n;
  end;
end;

procedure den (p : ptrie; d : integer);
var i : integer;
begin
  if d = hocico then for i := 1 to 31 do vypis (p^[i])
                else vypis (p^[d]);
end;

procedure mesiac (p : ptrie; d, m : integer);
var i : integer;
begin
  if m = hocico then for i := 1 to 12 do den (p^[i], d)
                else den (p^[m], d);
end;

procedure hladaj (d, m, r : integer);
var i : integer;
begin
  if r = hocico then for i := 1950 to 2050 do mesiac (db[i], d, m)
                else mesiac (db[r], d, m);
end;

end.
```

opravoval Peťo

3. Zachránená škôlka

(max. 15 bodov)

Graf je taká pekná matematická štruktúra pozostávajúca z vrcholov a hrán. Múdri ľudia dávno pred nami vymysleli kopu užitočných vecí o grafoch a preto sa občas oplatí ich využiť. Prenesme si našu úlohu do teórie grafov: každé dievča bude jeden vrchol grafu a každá možnosť podplatenia dvoch dievčat bude hrana medzi tými dievčatami=vrcholmi. Počet cukríkov potrebný na podplatenie budeme volať dĺžka hrany. My chceme nájsť pre Quida také hrany, aby sa pomocou nich dalo prejsť medzi každými dvoma vrcholmi a aby tam neboli žiadne zbytočné hrany. Takýto útvar sa volá *kostra* grafu. A keďže Quido nechce minúť veľa cukríkov tak to musí byť najlacnejšia kostra.

Ako som zistil pri opravovaní vašich riešení, tak spôsobov je naozaj veľa. Náš vzorový algoritmus je nasledovný: V každom kroku máme nejaké vrcholy už zaradené do čiastočnej kostry a ostatné sú ešte nikam nezaradené. Pre každý z nezaradených vrcholov vieme jeho najkratšiu hranu, ktorá by ho spojila s množinou zaradených vrcholov. Teda jeho vzdialenosť od čiastočnej kostry. Vyberieme ten, ktorý mal najkratšiu vzdialenosť spomedzi všetkých, pridáme ho medzi zaradené a tú najkratšiu hranu pridáme do kostry. Potom však musíme pozrieť všetky jeho hrany, ktoré vedú medzi nezaradené vrcholy a vrcholom upraviť vzdialenosti od čiastočnej kostry (už aj s našim vrcholom) tak, aby boli zase najmenšie. Teraz môžeme vybrať ďalší najbližší vrchol. Takto postupujeme kým nezistíme, že kostra už obsahuje všetky vrcholy.

Prečo to funguje? Vždy vyberieme tú najvhodnejšiu hranu h, ktorá v najlacnejšej kostre určite bude. Pridávame vrchol v. Cesta do kostry cez nejaké ešte nepridané vrcholy určite nebude lacnejšia, pretože ak by sme výhodne pospájali nejaké vrcholy spomedzi ešte nezaradených, musíme potom pripojiť tento komponent k pôvodnej čiastočnej kostre a to by sme aj tak urobili tou najkratšou možnou hranou, teda hranou h. A tou ideme práve pridať v do kostry takže sa to oplatí a nič tým nepokazíme. Keď už pridáme v a h do kostry, tak nám to žiadne vzťahy nenaruší pretože kostra už bola robená tak, že pridanie každej hrany sa oplatilo. To znamená, že neexistuje hrana, ktorá by bola nahraditeľná práve pridanou hranou h s použitím ešte nejakej inej. Potom by totiž jedna z nich musela byť kratšia ako tá, ktorú chceme nahradiť, a teda by sme ju použili skôr.

Inak tento algoritmus je známy ako *Primov* algoritmus. Poznáme takisto *Kruskalov* algoritmus, ale ten je výhodný, ak je nejak dobre ohraničený počet hrán. Časová zložitosť *Primovho* algoritmu je nasledovná. Na pridanie každého z $n-1$ vrcholov treba: nájsť najbližší spomedzi n vrcholov a potom treba pozrieť ešte maximálne $n-1$ hrán, ktoré môžu z neho vychádzať, teda: $T(n) = (n-1) \times (n+n) = 2n^2 - 2n$ a to je $O(n^2)$. Pamäťová zložitosť vzorového riešenia bude $O(n^2)$ kvôli tomu, že používame maticu susednosti. Dali by sa použiť spájané zoznamy a tam si pamätať iba tie hrany, ktoré existujú, ale to by zlepšilo iba na $O(m)$, čo sa pri zahustenom grafe blíži k nášmu n^2.

Spomeniem také dve chybičky, ktoré boli vo vašich riešeniach najčastejšie. Popis, ktorý píšete nemá byť popis programu, ale *algoritmu*! Občas môžete spomenúť, čo sa ako volá vo vašom programe, ale nepíšte, čo robí ten-ktorý riadok programu. Tak isto by tam mal byť dôkaz správnosti resp. funkčnosti hlavne pri „neštandardných“ riešeniach. Opravovateľ by nemal krvopotne hľadať dôvod, prečo to máte dobre. A keď píšete odhad zložitosti, pozor na písmenká. Keď je n vrcholov a m hrán, tak to tak musí byť aj v odhade (všimnite si rozdiel medzi hodnotením n^2 a m^2 ;-). No, a body:

- $O(n^2)$ – 15 bodov,
- $O(m \log m)$, $O(m \log n)$ – 13 bodov,
- $O(m \times n)$ – 11 bodov,
- $O(n^3)$ – 9 bodov; $O(m^2)$ – 8 bodov; $O(n^2 \times m)$ – 7 bodov; $O(m^3)$ – 6 bodov;
- tradične za slabý až žiadny popis sa stŕhali 1 až 2 bodov,
- za nesprávny alebo žiadny odhad zložitosti 1 bod

Listing programu:

```
const max=10;

type Tvrchol=record
   vzdialenost,odkial:integer;
end;

var i,j,n,m,a,b,c,max_cena,cukriky,min,minv:integer;
   hrany:array[1..max,1..max] of integer;
   vrchol:array[1..max] of Tvrchol;
```

```
begin
   max_cena:=0;
   readln(n,m); {nacitaj m,n z f}
   for i:=1 to n do
      for j:=1 to m do hrany[i][j]:=-1;
      {ak neexistuje hrana z i do j tak je tam -1}
   for i:=1 to m do begin
      readln(a,b,c);
      hrany[a][b]:=c;
      hrany[b][a]:=c;
      if (c>=max_cena) then max_cena:=c+1;
      {max_cena je vacsie ako hoci ktora cena
     takze to bude vzdialenost vrcholu do ktoreho este neviedla hrana}
   end;
   cukriky:=0;
   vrchol[1].vzdialenost:=0;
   for i:=2 to max do
      if(hrany[1][i]<>-1)then begin
         vrchol[i].vzdialenost:=hrany[1][i];
         vrchol[i].odkial:=1;
      end
      else vrchol[i].vzdialenost:=max_cena;
   {v prvom vrchole uz som a ostatnym som nastavil vzdialenost od prveho
   ak existuje taka hrana}

   for i:=1 to n-1 do begin {treba pridat n-1 hran do kostry}
      min:=max_cena;
      for j:=2 to n do
         if (vrchol[j].vzdialenost>0) and (vrchol[j].vzdialenost<min) then begin
            min:=vrchol[j].vzdialenost;
            minv:=j;
         end;
      {nasiel som najblizsi vrchol}
      writeln(vrchol[minv].odkial,' ',minv);
      cukriky:=cukriky+min;
      {vypisal som hranu a pripocital cenu hrany}
      vrchol[minv].vzdialenost:=0;
      for j:=2 to n do
         if (hrany[minv][j]>-1)
            and (hrany[minv][j]<vrchol[j].vzdialenost) then begin
            vrchol[j].vzdialenost:=hrany[minv][j];
            vrchol[j].odkial:=minv;
         end;
      {upravil som vzdialenosti a mozem ist odznova}
   end;
   writeln('A cena bude ',cukriky);
   {nebolo to v zadani ale vypisat to mozeme ;-) }
end.
```

4. Zobudené veveričky

opravoval Lukáš
(max. 15 bodov)

Táto úloha patrí do teórie grafov. Domčeky veveričiek sú vrcholy a konáre medzi nimi sú hrany. Navyše podľa zadania je daný graf strom (veď veveričky bývajú na stromoch). Strom je taký súvislý graf, ktorý nemá cyklus. Zároveň medzi každými dvomi vrcholmi existuje práve jedna cesta a celý graf má práve $N - 1$ hrán. Ešte sa oplatí spomenúť listy. To sú také

vrcholy, ktoré majú len jedného suseda. Strom, ktorý má aspoň dva vrcholy, má aspoň dva listy (to sa dá ľahko ukázať napríklad sporom).

Toľko definície a teraz sa pozrime na našu úlohu. Predstavme si, že z grafu vyberieme nejakú hranu. Graf sa rozpadne na dve časti (tie voláme komponenty), pričom v jednej bude k orechov chýbať a v druhej bude zase k orechov naviac. Keď hranu dáme späť, zistíme, že cez ňu určite pôjde k veveričiek, lebo si pôjdu po svoj orech. Ak by tadiaľ išlo viac veveričiek, dostali by sme horšie riešenie. Toto je základná myšlienka nášho algoritmu, zostáva nám len vyriešiť počítanie orechov v komponentoch.

Náš strom si zakoreníme. Dá sa to jednoducho predstaviť tak, že strom zavesíme na stenu za ľubovoľný vrchol a zatrasieme. Vrcholy sa tak preusporiadajú do úrovní podľa vzdialenosti od koreňa. Zavedieme ešte jedno označenie. Nech uv je hrana medzi vrcholmi u a v a vrchol u je bližšie ku koreňu ako vrchol v. Ak z grafu odoberieme hranu uv, vzniknú dva komponenty. Ten, v ktorom je vrchol v budeme volať podstrom s koreňom v. Keď sa totiž lepšie pozrieme na tento komponent, vidíme, že všetky vrcholy sú od koreňa ďalej ako vrchol v.

Možno už niektorí z vás vedia, že najlepšie sa strom zakoreňuje prehľadávaním do hĺbky. Implementuje sa pomocou rekurzie. Funkcia `prehladaj(i)` berie ako parameter číslo vrchola, z ktorého sa má vnoriť hlbšie do stromu. Pozrie sa na všetkých susedov vrchola i a rekurzívne sa na nich zavolá. Funkcia sa, samozrejme, volá len na ešte nenavštívené vrcholy. Naša funkcia bude zároveň vracať počet orechov, ktoré chýbajú v podstrome s koreňom i.

Susedov vrchola i označme $a_1, \ldots, a_m$ (nie je medzi nimi vrchol, z ktorého sme do vrchola i vošli). Všimnime si číslo $s = \mathit{prehladaj}(a_1) + \cdots + \mathit{prehladaj}(a_m)$, ktoré určuje počet chýbajúcich orechov v podstrome s koreňom i. Túto hodnotu vráti funkcia `prehladaj(i)` a zároveň je to počet veveričiek, ktoré pôjdu po hrane z vrchola i smerom ku koreňu. Stačí nám spočítavať tieto čísla pre všetky vrcholy a dostaneme správny výsledok.

Algoritmus už máme, ešte musíme vyriešiť pamätanie stromu, aby sme mali optimálnu pamäťovú a časovú zložitosť. Pre každý vrchol si budeme pamätať zoznam jeho susedov. Využijeme spájaný zoznam, aby sme mali pamäťovú zložitosť $O(N)$. Spájaný zoznam je dátová štruktúra, kde si každý objekt pamätá svojho nasledovníka. Každý vrchol si pamätá ukazovateľ na svojho prvého suseda. Prvý sused si pamätá ukazovateľ na druhého suseda atď. Takto máme každú hranu v pamäti dvakrát, lebo každá hrana má dva konce. Prehľadávanie do hĺbky navštívi každý vrchol práve raz, preto pracuje v čase $O(N)$.

Nakoniec pár slov k hodnoteniu. Za optimálne riešenie pracujúce v čase a pamäti $O(N)$ ste mohli dostať 15 bodov, za riešenia v $O(N^2)$ 12 bodov. Za pomalšie ešte menej a za nefunkčné najmenej.

Listing programu:

```
program vevericky;
type
  ukaz=^zoznam;
  zoznam=record
    i: integer;
    dalsi: ukaz;
  end;

var s, j, n, a, b: integer;
    p: array[1..100] of integer;
    bol: array[1..100] of boolean;
    l: array[1..100] of ukaz;
    x, y: ukaz;

function prehladaj(i: integer) : integer;
var u:integer;
    y:ukaz;
begin
```

```
  bol[i]:=true;
  u:=p[i]; y:=l[i];
  while y<>nil do begin
    if not bol[y^.i] then inc(u, prehladaj(y^.i));
    y:=y^.dalsi;
  end;

  inc(s, abs(u));
  prehladaj:=u;
end;

begin
  readln(n);
  for j:=1 to n do begin
    read(p[j]); dec(p[j]);{každá veverička zje jeden orech}
    bol[j]:=false;          {ak ho nemá, bude v zápore, ale to nevadí}
    l[j]:=nil;
  end;
  readln;

  for j:=1 to n-1 do begin
    readln(a, b);{načítanie hrán do spájaného zoznamu}
    new(x); x^.i:=b; x^.dalsi:=l[a];
    l[a]:=x;
    new(x); x^.i:=a; x^.dalsi:=l[b];
    l[b]:=x;
  end;

  s:=0;
  prehladaj(1);{strom zakoreníme z prvého vrchola}
  writeln(s*2);

  for j:=1 to n do begin {čistenie pamäte}
    x:=l[j];
    while x<>nil do begin
      y:=x;
      x:=x^.dalsi;
      dispose(y);
    end;
  end;
end.
```

5. Zamyslite sa

opravoval Tono
(max. 15 bodov)

Hneď na úvod poviem, že ak by ste poslali zadanie tohto príkladu ako jeho riešenie, nedostali by ste veľa bodov. Zadanie tohto príkladu totiž zakazovalo poslať zadanie tohto príkladu (t.j. samo seba) s riešením ako riešenie. To je pre opravovateľa značne nevýhodné, pretože inak by sa našiel riešiteľ, ktorý by napísal vzorové riešenie a ušetril tak opravovateľovi kopu roboty.[6]

Opravovanie tohto príkladu bola ale vcelku zábava. Došla kopa výtvorov, niektoré zaujímavé, niektoré aspoň s rozsiahlou epickou rozprávkou. No nie všetci ste pochopili, o čo presne išlo. Príklady ste si vymysleli všetci. Tiež ste ich všetci vyriešili. Tu ale väčšina z vás

[6] Ak by ste zadanie upravili tak, že môžete poslať aj jeho riešenie ako riešenie, dostali by ste viacej bodov. Tu nám vzniká taká kuriózna situácia... Nestáva sa často, aby niekto poslal vzorové riešenie a dostal zaň menej bodov ako za niečo iné. Už ste sa zamotali?

skončila. Zabudli ste totiž vzorové riešenie *vysvetliť* a jasne určiť, ako sa príklad bude opravovať. Možno stále ešte niektorí nečítate vzoráky... v každom prípade máte teraz exkluzívnu šancu v skratke dozvedieť sa, ako u nás vznikajú príklady.

Fáza[7] 1 – vymýšľanie príkladu

Jedného pekného dňa sa celý seminár stretne (všetci dôjdu, načas, nie je treba sa nikomu vyhrážať). Každý prezentuje *nápad* na nový príklad. Takýto nápad sa skladá z približného určenia problému (zatiaľ ešte bez všetkých rozprávok okolo) a myšlienky jeho riešenia. Väčšinou je problém zaujímavý a riešenie pekné, elegantné a nie veľmi prácne. Niekto rád vymýšľa originálne príklady, niekto drsne ťažké – každý ale s cieľom poskytnúť riešiteľovi čo najväčšiu radosť z riešenia. Všetci sa po búrlivom diskutovaní dohodnú na sade príkladov, tie sa potom spíšu a posunú do oddelenia rozprávok.

Fáza 2 – písanie zadaní

Teraz už jednotliví ľudia dávajú príkladom čitateľnú formu. Úloha to nie je ľahká. Musí sa napísať dostatočne pútavá rozprávka, ktorá ale nie je zbytočne prešpekulovaná a zdĺhavá. Taktiež sa musí dbať na presnosť a jednoznačnosť zadania. Nakoniec sa ešte aj musí vymyslieť vhodný názov.[8] Následne sa zadania dajú prečítať každému, kto príde pod ruku, nech sa vyčistia od všemožných chýb, či už pravopisných alebo iných. Potom sa vytlačia, s láskou zabalia, previažu veľkou červenou stuhou a pošlú do sveta. (Podľa správ od riešiteľov stuha vraj zvykne po ceste k nim zmiznúť. Situáciu prešetríme.)

Fáza 3 – opravovanie a riešenia

Hneď, ako dôjdu vaše riešenia, pustíme sa do horlivého opravovania. Opravovateľ všetky prezrie a zistí, ako veľmi rôznorodé sú. Potom určí spôsob hodnotenia a riešenia oboduje. Zároveň napíše *vzorové riešenie*. V ňom musí byť čo najzrozumiteľnejšie vysvetlená myšlienka optimálneho riešenia (samozrejme, väčšinou aj spolu s odhadmi časovej a pamäťovej zložitosti programu), prípadne myšlienky viacerých možných optimálnych riešení. Taktiež musí obsahovať popis bodovania, v ktorom budú určené body za riešenia s rôznymi zložitosťami. Nakoniec, ak si to príklad vyžaduje, treba ešte priložiť zdrojový kód programu. Tak ako zadania, aj vzoráky sa opakovaným čítaním vyčistia od chýb, vytlačia, navoňajú a pošlú k vám.

Ako sa hodnotilo? Za každú fázu, hoci rôzne časovo a intelektuálne náročnú, ste mohli získať najviac 5 bodov. Dôležitá bola originalita príkladu, elegancia možného riešenia, čitateľnosť rozprávky, jednoznačnosť zadania, prítomnosť vzorového riešenia, jeho zrozumiteľnosť a správnosť, spôsob bodovania a iné. Zavážiť mohol aj celkový dojem, za ktorý ste mohli získať inde stratené body.

[7] fasa, skákať po nej dá sa

[8] **Z**amyslite sa, ako sa pozná vhodný názov.

Výsledková listina po 2. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda		21	22	23	24	25	Σ
1	Danilák Michal	Gym. Hubeného BA	2	65	15	15	13	13	15	136
2	Králik Martin	Gym. Grösslingová BA	3	67	14	11	8	12	15	127
3	Danko Juraj	Gym. P. de Coubertina Piešťany	2	64	15	10	15	6	14	124
4	Pavlíček Tomáš	SPŠE Piešťany	2	55	16	15	15	2	15	118
5	Kováč Michal	Gym. Grösslingová BA	3	62	14	15	10	2	13	116
6	Mikuláš Ondrej	Gym. Haličská Lučenec	2	55	15	15	15	12		112
6	Sucha Martin	Gym. Jura Hronca BA	1	69	15	15	13			112
8	Zajacová Danica	Gym. Jura Hronca BA	2	43	14	15	15	2	14	103
9	Korcsok Peter	Gym. Mládežnícka Šahy	1	57	15	12	10			94
9	Tříska Martin	Gym. P. de Coubertina Piešťany	2	46	15	15	6		12	94
11	Košdy Martin	Gym. Jura Hronca BA	0	52	14	15	11			92
12	Kucharík Marcel	Gym. Športová Nové Mesto n.V.	3	39	15	15	5		13	87
12	Petrucha Michal	Gym. Metodova BA	0	47	15	15	10			87
14	Nikodem Peter	Gym. Školská Spiš. Nová Ves	1	43	14	15			9	81
15	Fedáková Dominika	Gym. Stará Ľubovňa	1	26	15	15	11		13	80
15	Švec Marcel	Gym. Jura Hronca BA	2	32	14	13	10	11		80
17	Hlavatý Peter	Gym. Jura Hronca BA	1	31	9	13	2	12	12	79
17	Nagy Kristián	Gym. Jura Hronca BA	0	29	14	13		12	11	79
19	Takács Michal	Gym. Tajovského B. Bystrica	3	40	14	15	8			77
20	Pavlovský Martin	Gym. Jura Hronca BA	2	17	14	15	10	2	11	69
20	Uherčík Tomáš	Gym. P. de Coubertina Piešťany	2	41	0	10	7	2	9	69
22	Berthoty Adam	Gym. Ľudovíta Štúra Trenčín	3	21	15	15	2		15	68
23	Beňo Fratišek	SPŠ Humenné	1	28	13	13			12	66
23	Betík Roman	SPŠ Levice	3	26	15	15	10			66
25	Panák Pavol	Gym. Jura Hronca BA	3	31	13	12	8			64
26	Vojtko Jakub	Gym. Jura Hronca BA	0	18	15	15			15	63
27	Kollár Jakub	Gym. Jura Hronca BA	3	29	14	12				55
28	Orihel Lukas	Gym. Jura Hronca BA	3	31	13		8			52
29	Okruhlica Adam	Gym. Jura Hronca BA	2	48						48
30	Blaho Pavol	Gym. Jura Hronca BA	1	0	14	15	8	10		47
31	Hegedušová Monika	Gym. P. Horova Michalovce	3	8	11	15		1	10	45
31	Vido Rudolf	Gym. Jura Hronca BA	2	4	14	15			12	45
33	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	1	44						44
33	Schuster Vladimír	Gym. Jura Hronca BA	1	14	15			15		44
33	Vlček Tomáš	Gym. Šk. Bratov BA	2	15	15	14				44
36	Mego Marek	Gym. Jura Hronca BA	3	8	15	15	5			43
37	Beno Miroslav	Gym. Jura Hronca BA	3	14		15		12		41
37	Kvasnička Igor	Gym. Jura Hronca BA	1	3	12	15		11		41
39	Kostolányi Peter	Gym. Jura Hronca BA	0	12	13	15				40
40	Schlosáriková Lucia	Gym. P. de Coubertina Piešťany	2	0	15	13			10	38
41	Holla Kamila	Gym. Jura Hronca BA	3	14		12		11		37
41	Mosný Vladimír	Gym. Jura Hronca BA	2	14	13	10				37
43	Vnuk Marek	Gym. Jura Hronca BA		0	13	12	1		8	34
44	Mazal Tomáš	Gym. Jura Hronca BA	2	33						33
45	Synak Peter	Gym. Jura Hronca BA	1	31						31
46	Kováčovský Tomáš	Gym. Jura Hronca BA	1	0		15	13			28
47	Bílka Ondřej		3	0	14	5		8		27
47	Dittrich Andrej	Gym. Jura Hronca BA	1	7	8	12				27
49	Vojtko Martin	Gym. Jura Hronca BA	2	26						26
50	Dobiaš Michal	Gym. Jura Hronca BA	3	24						24

	Meno a priezvisko	Škola	Trieda		21	22	23	24	25	Σ
50	Lachata Adrián	Gym. Svidník	3	24						24
52	Baumann Martin	Gym. Jura Hronca BA	2	10			13			23
53	Kuboň Luboš	Gym. Jura Hronca BA	0	21						21
54	Doležalková Andrea	Gym. Jura Hronca BA	2	7		13				20
54	Gálik Marian	Gym. Jura Hronca BA	1	20						20
56	Mazák Tomáš	Gym. Jura Hronca BA	2	0	13		6			19
57	Jarabek Tomas	Gym. Jura Hronca BA	1	18						18
58	Trnovec Matúš	Gym. Jura Hronca BA	1	17						17
59	Sabo Michal	Gym. Jura Hronca BA	1	15						15
59	Šuranyi Peter	Gym. Jura Hronca BA	3	15						15
61	Fojtík Michal		3	14						14
61	Jarabek Michal	Gym. Jura Hronca BA	3	14						14
61	Struhár Filip	Gym. Jura Hronca BA	2	14						14
64	Jakabovic Juraj	Gym. Jura Hronca BA		0		13				13
64	Janoskova Michaela	Gym. Jura Hronca BA	1	13						13
64	Martinovic Miroslav	Gym. Jura Hronca BA	1	13						13
64	Morvay Peter	Gym. Jura Hronca BA	2	13						13
64	Ružička Peter	Gym. Jura Hronca BA	4	0	13					13
69	Kolesár Štefan	Gym. Alejová Košice	3	0		11				11
70	Meszaros Milan	Gym. Jura Hronca BA	1	10						10
71	Dillinger Viliam	Gym. Jura Hronca BA	3	8						8
72	Malícková	Gym. Jura Hronca BA	1	7						7
72	Pilák Otto	Gym. Jura Hronca BA	0	7						7
72	Slobodník	Gym. Jura Hronca BA	1	7						7
72	Travniček Bohuš	Gym. Jura Hronca BA	2	7						7
72	Zalesakova Dada	Gym. Jura Hronca BA	1	7						7
77	Gruska Jonáš	Gym. Jura Hronca BA	0	6						6
77	Truben Martin	Gym. Jura Hronca BA	1	6						6
77	Zvac Peter	Gym. Jura Hronca BA	2	6						6
80	Svoboda Boris	Gym. Jura Hronca BA	1	5						5
81	Koval Michal	Gym. Jura Hronca BA	1	4						4
81	Lesnák Stefan	Gym. Jura Hronca BA	1	4						4
83	Hamid	Gym. Jura Hronca BA	0	3						3
84	Petro Juraj	Gym. Jura Hronca BA	1	1						1
85	Kundis Tomáš	SOU, Kukučínova 483, Poprad	2	0						0
85	Lobotková Martina	Gym. Grösslingová BA	3	0						0