**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

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. O aritmetickej postupnosti

opravovala Julka
(max. 15 bodov)

Riešenia tohto príkladu boli veľmi rozmanité. Skoro žiadne dve riešenia nemali rovnakú časovú zložitosť a ani jedno riešenie sa nevyrovnalo vzorovému `:-)`. Našli sa riešenia, ktoré predpokladali, že všetky čísla sú celé, napriek tomu, že to nikde v zadaní nebolo napísané. Toto niektorým trochu zmenilo ideu, ale snažila som sa to vždy prerobiť tak, aby nikto nebol veľmi ukrátený o body (t.j. za ten predpoklad som strhávala 2 body a potom už ďalej bodovala, akoby to ten človek riešil aj pre reálne čísla).

Bodovanie bolo nasledovné:

- $O(N^2/\log N)$ – za vzorové riešenie sa dalo získať 15 bodov,
- $O(N^2)$ a pamäť $O(N^2)$ – 13 bodov,
- $O(N^2 \log N)$ – 11 bodov,
- $O(N^3)$ a pamäť $O(N)$ – 9 bodov,
- $O(N^3)$ a pamäť $O(N^2)$ – 8 bodov,
- $O(N^3 \log N)$ a pamäť $O(N)$ – 7 bodov,
- za predpoklad, že na vstupe sú iba celé čísla sa väčšinou strhali 2 body,
- za nesprávny alebo žiadny odhad zložitosti 1 bod

A teraz už prejdime k vzorovému riešeniu. Najprv vstup utriedime. Nič tým nezhoršíme, iba polepšíme. Časovú zložitosť nám to aj tak nezhorší. Budeme skúšať možné dĺžky postupnosti a pre danú dĺžku budeme v lineárnom čase vedieť povedať, či sa tam nachádza aritmetická postupnosť danej dĺžky (teda aj danej diferencie).

Najprv si treba uvedomiť, že stačí skúmať postupnosti dĺžky $q$, kde $q-1$ je prvočíslo. Ak by existovala aritmetická postupnosť dĺžky $a \cdot b + 1$ s diferenciou $A/ab$, tak musí nutne existovať aj aritmetická postupnosť dĺžky $a+1$ s diferenciou $A/a$.

Príklad: $7 = 2 \cdot 3 + 1$, takže môžeme zobrať prvý, štvrtý a siedmy prvok (teda každý tretí).

Z tohto dôvodu nám teda stačí skúmať iba uvedené dĺžky. Jeden slávny ujo dokázal, že prvočísel menších ako $N$ je rádovo $N/\log N$. Z toho bude vyplývať aj časová zložitosť.

A teraz ako sa dá v lineárnom čase zistiť, či existuje postupnosť danej dĺžky s danou diferenciou. Odpoveď je: Dynamické programovanie. V poli na $i$-tom mieste si budeme pamätať, aká najdlhšia aritmetická postupnosť s danou diferenciou sa nám končí v $i$-tom mieste.

Budeme mať dva ukazovatele. Jeden sa bude posúvať po poli vždy práve o 1 a bude ukazovať na prvok (nazvime ho aktuálny), ktorému práve hodnotu najdlhšej postupnosti rátame. Druhý pointer sa tiež bude posúvať po prvkoch iba jedným smerom (upozorňujem, že to pole je už utriedené). Bude vždy ukazovať na najmenší taký prvok, ktorý je už väčší alebo rovný ako aktuálny mínus diferencia. V prípade, ak je rovný, znamená to, že pridaním aktuálneho prvku sa nám postupnosť zväčší o 1 od najdlhšej postupnosti pre prvok, na ktorý nám ukazuje druhý pointer. V prípade, že je väčší, znamená to, že sa v poli nenachádza prvok s hodnotou aktuálneho − diferencia. Najdlhšia postupnosť, končiaca v aktuálnom prvku, je teda 1.

Akonáhle sme počas rátania našli postupnosť danej dĺžky, môžeme skončiť a vyhlásiť, že taká postupnosť sa tam nachádza. V opačnom prípade ideme ďalej a skúšame možné dĺžky postupnosti až do čísla $N$.

Spracovávaných dĺžok bolo rádovo $N/\log N$ a spracovanie nám trvalo $O(N)$. Výsledná časová zložitosť je teda $O(N^2/\log N)$. Pamätali sme si iba lineárne veľa čísel, takže pamäťová zložitosť je $O(N)$.

## Listing programu:

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>

#define eps 0.000000001

int p[100];
double vs[100];
int dyn[100];

int double_cmp(const void *va, const void *vb) {
  double *ia=(double *)va, *ib=(double *)vb;
  if (*ia>*ib) return 1;
  if (*ia==*ib) return 0;
  return -1;
}

int sito(int n) {
  int pom;
  n=n+1;
  bzero(p,n*sizeof(int));
  int odmoc=(int)sqrt(n);
  for (int i=2; i<odmoc; i++) {
    if (p[i]==0){
      pom=i;
      while (i+pom<n) {
        pom+=i;
        p[pom]=1;
      }
    }
  }
  return 0;
  //teraz bude v poli p[i]=1 ak i nie je prvocislo a
  //p[i]=0 ak i prvocislo je
}


int main() {
  int hotovo=0;
```

```
  int akt, moz, n;
  double A;
  scanf("%d ",&n);
  scanf("%lf ", &A);
  for (int i=0; i<n; i++) {
    scanf("%lf ", &vs[i]);
  }
  qsort(vs, n, sizeof(vs[0]), double_cmp);
  sito(n);
  //tipneme si dlzku postupnosti
  for (int i=3; i<=n && hotovo==0; i++) {
    //ak i-1 je prvocislo, teda v p[i-1] je 0
    if (!p[i-1]) {
      //ideme ratat dynamiku
      double dif=A/(i-1);
      dyn[0]=1;
      moz=0;
      for (akt=1; akt<n && hotovo==0; akt++) {
        while (vs[moz]+dif<vs[akt] && moz<akt-1) moz++;
        if ((vs[moz]+dif+eps>=vs[akt]) && (vs[moz]+dif-eps<=vs[akt]))
          dyn[akt]=dyn[moz]+1;
        else dyn[akt]=1;
        if (dyn[akt]==i) hotovo=1;
      }
    }
  }
  if (hotovo) printf("ANO EXISTUJE\n"); else printf("NIE NEEXISTUJE\n");

  return 0;
}
```

## 2. O štatistikárení

opravoval Mirko
(max. 15 bodov)

Tento príklad riešilo len málo ľudí. Ale pripisujem to skôr tomu, že je už koniec školského roka, ako tomu, že by bol ťažký. Idea riešenia je veľmi jednoduchá: stačilo si uvedomiť, kde sa vyskytuje takzvaná subštruktúra problému, ktorá je základom aplikácie dynamického programovania. Subštruktúra v tomto kontexte voľne povedané znamená, že optimálne (t.j. to, ktoré chceme) riešenie celého problému obsahuje optimálne riešenia istej množiny podproblémov. Problém resp. podproblém je tu chápané ako vstup resp. časť vstupu. Navyše uvažovaný podproblém musí spĺňať určité vlastnosti vzhľadom na isté parametre. Tieto vlastnosti a parametrizácia spolu presne opisujú subštruktúru problému. Poďme si teda povedať, ako vyzerá subštruktúra v tejto úlohe.

Riešenie problému je vlastne postupnosť čísel ľudí, ktorá udáva, kto kedy tancoval, s najmenším počtom výmen. Podproblémy budú iba také, že obsahujú všetkých ľudí a pre každého všetky tancovania v čase menšom alebo rovnom $t$ a nech v riešení podproblému tancuje ako posledný človek $m$. Takže máme parametre $t$ a $m$. Nech $B[t][m]$ je najmenší počet výmen pre podproblém daný parametrami $t$ a $m$. Predstavme si teraz nejaké optimálne riešenie, teda majme postupnosť ľudí, ktorá vedie k najmenšiemu počtu výmen. Teraz ak v tejto postupnosti je na konci minúty $t_1$ človek $m_1$, tak počet výmen, ktoré sa vyskytli do konca minúty $t_1$ je $B[t][m]$. Toto je dosť dôležité tvrdenie (dobre si uvedomte, čo tvrdíme). Toto tvrdenie je nám zatiaľ nanič, najprv ho musíme vedieť použiť. A na to, aby sme ho mohli použiť, musíme ho najprv dokázať.

Takže ak by $B[t][m]$ bolo menej ako počet výmen v najlepšom riešení do konca minúty $t_1$, mohli by sme tuto postupnosť (riešenie) upraviť tak, aby sa jej začiatok zhodoval s riešením (postupnosťou) pre podproblém s parametrami $t = t_1$ a $m = m_1$. Tým by sme ale znížili počet výmen aj v celkovom riešení, čo je spor s jeho minimalitou. Opačná nerovnosť je zrejmá. V tomto dôkaze však chýba pár vecí, napríklad nie je zrejmé, ako chceme postupnosť upraviť, a že či potom dostaneme platné riešenie. Taktiež chýba aj formalistická zakončovacia perlička typu: „ak neplatí, že $a > b$, ani $a < b$, tak $a = b$, čo bolo treba dokázať“. Navyše bol ukrátený o mnohé iné formalizmy na úkor jeho zrozumiteľnosti.

A na čo je to všetko dobré? No predsa vďaka tomu, že sme si vhodne zvolili subštruktúru, vieme teraz jednotlivé podproblémy riešiť pomocou riešení menších podproblémov. A takto sa vieme od triviálneho riešenia dopracovať k celkovému riešeniu. Postup je nasledovný: Nech $K$ je počet ľudí skáčucich v minúte $t$, $B[1][m]$ počet ľudí skáčucich v prvej minúte.

- ak $t = 0$, tak $B[0][x] = 0$ pre všetky $x$; pre $t \geq 1$:
- ak $K = 0$, tak $B[t][m] = B[t][m-1]$,
- ak $K > 0$ a $m$ neskákal v $t$-tej minúte, tak $B[t][m] = \infty$
- ak $K \geq 1$ a $m$ skákal v $t$-tej minúte, tak

$$B[t][m] = \min(B[t-1][m] + K - [K = 1], \\ \min_{m' \neq m} (B[t-1][m'] + K - [m' \text{ skákal } t\text{-tu minútu}])),$$

kde $[P] = 1$, ak $P$ je pravdivé, inak $[P] = 0$; teda napr. $[K = 1]$ je 1, ak $K = 1$, inak 0.

To znamená, že zoberieme minimum zo všetkých takýchto možností:

1. Týpek $m$ bol posledný v $t-1$ minúte a nie je sám v $t$-tej. Vtedy necháme tancovať $K-1$ ľudí a nakoniec týpka $m$, čo je $K$ výmen.
2. Týpek $m$ bol posledný v $t-1$ minúte a v $t$-tej je sám ($K = 1$). Potom nerobíme žiadnu výmenu.
3. Nech $m'$ je ľubovoľný týpek rôzny od $m$, potom vieme urobiť také usporiadanie, že dáme $K-1$ ľudí tancovať a týpka $m$ na koniec, čo je $K$ výmen. Ak ale týpek $m'$ tancuje aj v minúte $t$, tak ho necháme tancovať na jej začiatku a ušetríme si tak počet výmen o jedna.

V týchto troch prípadoch sú zrejme zahrnuté všetky usporiadania vedúce k optimálnemu riešeniu, lebo optimálne riešenie musí vychádzať z optimálnych riešení podproblémov, a poradie v akom ľudí umiestnime v rámci jednej minúty nie je dôležité; dôležité je iba to, kto je posledný a kto je prvý. A toho, kto je posledný, máme určeného v parametri $t$ a prvého má zmysel dávať jedine toho, čo bol posledný v predchádzajúcej minúte, a také možnosti prechádzame všetky. Všimnime si, že totožnosť týpka $m'$ je vo väčšine prípadov zbytočná, ide iba o to, či tancoval v $t$-tej minúte, alebo nie, a či bol rôzny od $m$. S použitím tohto poznatku vieme zlepšiť priamu implementáciu uvedeného postupu z $O(TN^2)$, kde by sme kvôli každej hodnote $B[t][m]$ prešli všetkých týpkov, na $O(NT)$.

V podstate si pre každý čas $t$ potrebujeme vyrátať par vecí: $A$, $A_m$, kde $A$ je minimálna hodnota $B[t-1][m]$ a človek $m$ netancoval v čase $t$, a $A_m$ je človek ktorý má túto hodnotu $A = [t-1][A_m]$. A ešte aj hodnoty $X$, $Y$ a $X_m$, $Y_m$, kde $X$, $Y$ sú 2 najmenšie hodnoty (môžu byť aj rovnaké), $X = B[t-1][X_m]$, $Y = B[t-1][Y_m]$, pričom $X_m$ sa nerovná $Y_m$, a ľudia $X_m$ a $Y_m$ tancujú v minúte $t$, a $X \leq Y$. teraz môžeme $B[t][m]$ vyrátať nasledujúcim spôsobom: ak $m$ je rôzne od $X_m$, tak $B[t][m] = \min(B[t-1][m] + K - [K = 1], X + K - 1, A + K)$, ak ale $m$ je rovnaké ako $X_m$, tak nemôžeme použiť $X$, lebo človek nemôže byť naraz aj prvý aj posledný (jedine že by $K = 1$, ale vtedy je $B[t-1][m] = X$, takže ok), takže musíme použiť $Y$ namiesto $X$. Tak to je všetko k riešeniu.

Riešenia sa bodovali nasledovne: Za časovú zložitosť $O(TN)$ bolo 15 bodov, za $O(TN^2)$ 12 bodov a za nefunkčné riešenie 8 bodov. Ďalej sa 1 bod strhával za formálne chyby, 3 body

za chýbajúci, respektíve nedostatočný dôkaz správnosti a 2 body aj za nedostatočný popis algoritmu. A samozrejme aj bonus $-x$ bodov podľa uváženia. Samozrejme, nefunkčné riešenie väčšinou nemôže mať dobrý dôkaz správnosti, a aj takú formálnu chybu, že nefunguje.

## Listing programu:

```
#include<cstdio>
using namespace std;
int H[20001][51]; //pole kde si znacim ze kedy kto tancoval;
int B[20001][51]; //tabulka ktoru vyplname
int n;
int infty=10000000;
int main(){
  scanf("%d",&n);
  for(int i = 0;i < n;i++) for(int j = 0; j < 20001; j++) H[j][i]=0;
  for(int i = 0;i < n;i++){
    while(1){int a;scanf("%d\n",&a);if (a==0) break; else H[a][i]=1;}
  }

  for(int i=0;i<n;i++) B[0][i]=0; //inicializujeme hodnoty na 0

  for(int t = 1; t < 20001; t++){
    int K = 0;
    for(int i=0; i < n; i++) K+=H[t][i];  //H[t][i] je vzdy 0 alebo 1
    int A,X,Y,X_m; //A_m a Y_m vlastne nepotrebujeme

    //najdeme najmensi prvok z B[t-1][i],ak i netancuje minutu t;
    A=infty;
    for(int i = 0; i < n; i++)
      if (!H[t][i]) A <?= B[t - 1][i];

    //najmensi taky,ze tancoval
    X=infty;
    for (int i = 0; i < n; i++)
      if (H[t][i] && (X==infty || X>B[t - 1][i] ) ) {X=B[t - 1][i];X_m=i;}

    //najdeme dalsi najmensi
    Y=infty;
    for (int i = 0; i < n; i++)
      if (H[t][i] &&  i != X_m && (Y == infty || Y > B[t - 1][i])) {Y = B[t
- 1][i];}

    for (int m = 0; m < n; m++){
      int C = X;
      //zabezpecime vyber vhodnej hodnoty
      if (m == X_m) C = Y;

      //spocitame nas najdvolezitejsi vyraz
      B[ t ][ m ] = ( B[t - 1][ m ] + K - (K==1) ) <? ( C + K - 1 ) <? ( A +
K ) ;

      if (K > 0 && !H[ t ][ m ] ) B[ t ][ m ] = infty;

      if (K == 0) B[ t ][ m ] = B[ t - 1  ][ m ];
    }
  }

  int najmenej=infty;
  for(int i = 0; i < n; i++) najmenej <?= B[20000][i];
  printf("najmensi mozny pocet vymien je: %d\n",najmenej );
```

```
  return 0;
}
```

# 3. O maximalizácií radov

opravoval Palo
(max. 15 bodov)

Tento príklad patril medzi tie ľahšie. Základom bolo uvedomiť si, že nepotrebujeme žiadnu haldu ani inú podobnú štruktúru, ale stačia nám spájané zoznamy, ktorých operácie majú konštantnú časovú zložitosť. Pozrime sa priamo na vzorové riešenie:

Označme si čas od spustenia programu ako *aktcas*. V poli *casy* si budeme na $(aktcas + i) \bmod M$-tom mieste pamätať spájaný zoznam všetkých prestávok, ktoré sa začnú $i$ sekúnd od *aktcas*. Ďalej si budeme pre každú prestávku pamätať ukazateľ do zoznamu, v ktorom sa nachádza (prípadne nulový ukazateľ) a zoznam ešte nevyužitých mien pre prestávky. Mená prestávok budú čísla od 1 po $N$. Ako budú vyzerať implementácie jednotlivých príkazov?

- SET $i$: Zo zoznamu voľných mien pre prestávky jednu zmažeme a pridáme ju do spájaného zoznamu v poli *casy* na $(aktcas + i) \bmod M$-te miesto a zapamätáme si, kam sme ju pridali.
- DEL $i$: Pre prestávku $i$ máme ukazateľ do zoznamu prestávok, ktoré majú začať v rovnakom čase ako $i$. Z tohto zoznamu prestávku $i$ zmažeme a pridáme $i$ do zoznamu nevyužitých mien pre prestávky.
- TICK: Zvýšime *aktcas* o jedna. Vypíšeme všetky prestávky v zozname prestávok, ktoré sa začnú *aktcas* sekúnd od spustenia programu, tento zoznam vyprázdnime a všetky prestávky z neho znovu pridáme do zoznamu nevyužitých mien pre prestávky.

Je zrejmé, že operácie SET a DEL vieme implementovať v $O(1)$ a operáciu TICK v čase $O(K)$, kde $K$ je počet prestávok začínajúcich v čase *aktcas*, teda v najhoršom prípade $O(N)$. Ešte nám chýba upraviť jednu drobnosť – *aktcas* nemusíme zväčšovať do nekonečna, stačí si vždy pamätať hodnotu *aktcas* mod $M$.

Na začiatku musíme inicializovať zoznam nevyužitých mien pre prestávky a vynulovať všetky spájané zoznamy prestávok. Toto vieme spraviť v čase $O(N + M)$. Jedna operácia nám trvá v najhoršom čase $O(N)$ (konkrétne operácia TICK), takže ak program dostane $K$ príkazov, jeho časová zložitosť je $O(N + M + KN)$. Pamäťová je zjavne $O(N + M)$, pretože si pre každú prestávku pamätáme konštantné množstvo údajov.

Náš odhad časovej zložitosti však nie je tesný. Dá sa zlepšiť nasledovnou úvahou: Počet príkazov DEL a výpisov BZZZ nemôže byť väčší ako počet príkazov SET. To znamená, že ak máme na vstupe $S$ príkazov SET, tak spravíme maximálne $2S = O(S)$ operácií. Inými slovami: Ak máme na vstupe $K$ (ľubovoľných) operácií, tak z nich je maximálne $K$ SETov a teda urobíme maximálne $2K = O(K)$ operácií. Celková časová zložitosť nášho algoritmu spolu s inicializáciou je $O(N + M + K)$. Všimnime si jednu zaujímavú vec: Aj keď jedna operácia TICK môže byť pomalá (až $O(N)$), $K$ operácií trvá len $O(K)$.

Poznámky k implementácii: Nepotrebujeme si pri implementácii spájaného zoznamu dynamicky alokovať a dealokovať pamäť. Môžeme si prestávky pamätať v jednom poli veľkosti $N$ a v ňom si pre každú prestávku pamätať index predchádzajúcej a nasledujúcej prestávky. Takto je napísané aj vzorové riešenie nižšie.

Na záver niekoľko slov o hodnotení: Za program s časovou zložitosťou $O(N + M + K)$ sa dalo získať 15 bodov, s časovou zložitosťou $O(N + M + K \log N)$ maximálne 12 bodov a za časovú zložitosť $O(N + M + KN)$ najviac 10 bodov. Body ste väčšinou stratili za vyššie pamäťové nároky alebo nedostatočný popis programu.

## Listing programu:

```
const
    MAXN=10000;
    MAXM=10000;
var
    c: char;
```

```
    i, numids, kedy, aktcas, co, n, m: integer;
    next, prev, kde, ids: array[1..MAXN] of integer;
    casy: array[0..MAXM-1] of integer;

procedure pridaj(kam, co: integer);
begin
    if casy[kam]=-1 then begin
        casy[kam]:=co;
        prev[co]:=co;
        next[co]:=co;
        kde[co]:=kam;
    end else begin
        next[co]:=casy[kam];
        prev[co]:=prev[casy[kam]];
        next[prev[casy[kam]]]:=co;
        prev[casy[kam]]:=co;
        casy[kam]:=co;
        kde[co]:=kam;
    end;
end;

procedure zober(co: integer);
begin
    if next[casy[kde[co]]]=casy[kde[co]] then begin
        casy[kde[co]]:=-1;
    end else begin
        if co=casy[kde[co]] then casy[kde[co]]:=next[casy[kde[co]]];
        prev[next[co]]:=prev[co];
        next[prev[co]]:=next[co];
    end;
end;

begin
    readln(n,m);
    aktcas:=0;
    numids:=n;
    for i := 1 to n do ids[i]:=i;
    for i := 0 to m-1 do casy[i]:=-1;
    while true do begin
        read(c);
        if c='T' then begin
            aktcas:=(aktcas+1) mod m;
            while casy[aktcas]<>-1 do begin
                writeln('BZZZ ', casy[aktcas]);
                inc(numids);
                ids[numids]:=casy[aktcas];
                zober(casy[aktcas]);
            end;
        end else if c='S' then begin
            read(kedy);
            pridaj((aktcas+kedy) mod m, ids[numids]);
            writeln(ids[numids]);
            dec(numids);
        end else if c='D' then begin
            read(co);
            inc(numids);
            ids[numids]:=co;
            zober(co);
        end;
        readln;
```

```
    end;
end.
```

## 4. O reforme ciest do práce

opravoval Bee
(max. 15 bodov)

Prišli veľmi zaujímavé a navzájom väčšinou rôzne riešenia. Väčšina z vás síce uhádla správnu odpoveď, ale dokázal ju málokto a väčšina sa pustila cestou propagandy, sľubov a vyhrážok. Za funkčné, dokázané riešenie v $O(n)$ ste mohli získať plný počet bodov. Za funkčné, nedokázané riešenie v $O(n)$ ste získali toľko bodov, koľko ste si vymohli svojou propagandou, sľubmi a vyhrážkami. Iné riešenia – ak ma pamäť neklame – v podstate ani neprišli. Nič to za to, poďme na to ...

Ako som už povedal, väčšina z vás prišla na to, že odpoveď bude $\lceil \frac{l}{2} \rceil$, $l$ je počet listov v grafe. Vo všetkých vašich riešeniach sa slovo list vyskytlo, takže ho nebudem vysvetľovať.

Ako sa to dokáže? Väčšina z vás prišla aj na základnú myšlienku. To, že je to minimálny počet je celkom zrejmé, pretože vo výslednom grafe určite nesmú byť listy, a kedže jednou hranou sa zbavíme najviac dvoch listov, tak sa výsledok črtá už celkom jasne.

Že je to aj postačujúci počet, už vôbec také zrejmé nie je a hoci dôkaz nie je až taký extrémne ťažký, väčšina z vás má predsa len radšej to bohapusté presviedčanie.

Najprv zopár intuitívnych definícií, aby sa nám ľahšie pracovalo. Pojmy označené hviezdičkou sú moje vlastné, preto ich radšej doma nepoužívajte. Ich akákoľvek podobnosť so skutočnými pojmami je čisto náhodná.

- *Stupeň vrchola* $v$ je počet vrcholov spojených s vrcholom $v$ hranou. Budeme ho označovať $\deg(v)$
- *Cesta* je postupnosť vrcholov $v_1, v_2 \ldots, v_n$ taká, že susedné vrcholy v postupnosti sú spojené hranou a každý vrchol grafu je v nej najviac raz.
- *Lúč vrcholu* $v^*$ je cesta $v, v_1, v_2, \ldots, v_n$, kde $v_n$ je list a $\deg(v_1) = \deg(v_2) = \ldots = \deg(v_{n-1}) = 2$. Voľne povedané: z lúča nevychádzajú žiadne iné hrany.
- *Komponent* je maximálny súvislý podgraf grafu, teda medzi každými dvoma vrcholmi v komponente existuje cesta, ale do žiadneho vrcholu nepatriaceho do tohto komponentu nevedie žiadna cesta .
- *Vetva vrcholu* $v^*$ je taký podgraf daného grafu, že ak z grafu odoberieme vrchol $v$, tak vetva sa stane komponentom.
- *Strom* je taký graf, že všetky jeho vrcholy sú spojené pravé jednou cestou.
- *Slnečník*$^*$ je vrchol v strome, ktorého všetky vetvy sú lúče.
- *Slnečnica*$^*$ je strom, ktorý obsahuje slnečník.
- *Dvojsúvislosť* je vlastnosť grafu. Graf ju má, ak medzi každými dvoma vrcholmi existujú dve disjunktné cesty.
- *Cyklus* alebo *Kružnica* je postupnosť vrcholov $v_1, v_2, \ldots, v_n, v_1$ taká, že susedné vrcholy v postupnosti sú spojene hranou a každý vrchol z grafu okrem $v_1$ sa v postupnosti nachádza najviac raz. Zrejme cyklus je dvojsúvislý.
- *Kontrakcia cyklu*$^*$ je náhrada cyklu jedným vrcholom tak, že tento vrchol bude susediť so všetkými vrcholmi s ktorými susedil cyklus a nepatrili do cyklu.
- *Extrakcia cyklu*$^*$ je opačný proces ako kontrakcia. Vrchol vzniknutý kontrakciou nahradíme pôvodným cyklom.

Dajme sa teda dokazovať, najprv jedna malá lama. Ech, lema ...

**Lema.** *Každý strom, ktorý nie je slnečnica, obsahuje aspoň 2 vrcholy so stupňom aspoň 3. Navyše potom existujú aspoň dva také vrcholy že majú lúč.*

*Dôkaz.* Ak by strom obsahoval najviac jeden vrchol stupňa aspoň 3, tak všetky ostatné vrcholy majú stupeň najviac 2 a teda tvoria už len lúče; čiže strom je slnečnica. To, že existujú aspoň dva vrcholy, ktoré majú lúč je zrejmé. Ak by mal totiž najviac jeden vrchol stupňa aspoň 3 lúč, tak vetvy toho vrcholu, ktoré nie sú lúčmi, by nemohli obsahovať listy a teda tieto podgrafy by neboli stromami. □

Nech graf nie je slnečnica. Podľa lemy existujú aspoň dva vrcholy $s$ a $t$ stupňa aspoň 3 a navyše ich vieme vybrať tak, že budú mať lúče. V týchto lúčoch sú zrejme listy a práve tie teraz spojíme hranou. Vznikne nám cyklus, obsahujúci pridanú hranu, lúče a cestu medzi vrcholmi $s$ a $t$. Pretože $\deg(s) \geq 3$, $\deg(t) \geq 3$, tak existujú aspoň dve hrany, ktoré nepatria do cyklu. Skontrahovaním tohto cyklu dostaneme teda nový vrchol, z ktorého vychádzajú aspoň dve hrany a teda to nie je list. Takto sme dostali nový strom, ktorý ma o dva listy menej.

Tu je dôležité podotknúť, prečo sme sa takto trápili: ak by sme spojili len tak nejaké dva listy, tak kontrakciou môže vzniknúť list, čo znamená, že pridaním hrany, sme sa zbavili len jedného listu.

Takže back to dokazovanie. Tento postup môžeme opakovať dovtedy, kým nedostaneme slnečnicu. Tú raz dostaneme určite, lebo kontrakciou premieňame cyklus na vrchol, teda znižujeme počet vrcholov a po konečnom počte krokov sa dostaneme k takému počtu vrcholov, že graf už musí byť slnečnicou (dokážte na domácu úlohu: ak má strom najviac štyri vrcholy, tak je to slnečnica).

Teraz, ak je strom slnečnica, tak ho vieme ľahko doplniť na dvojsúvislý a to tak, že pospájame jeho listy. Ak má nepárny počet listov, tak posledný list spojíme s nejakým už spojeným listom.

Zrejme každým spojením vznikne jedna kružnica a táto bude vždy prechádzať slnečníkom. Ak sa chceme dostať z vrcholu $s$ do vrcholu $t$, pričom tieto ležia na jednej kružnici, tak medzi nimi existujú dve disjunktné cesty (obidve ležiace na kružnici). Ak ležia na rôznych kružniciach, tak existujú dve disjunktné cesty z $s$ do slnečníka a tiež dve disjunktné cesty z $t$ do slnečníka. Z nich už ľahko skonštruujeme dve disjunktné cesty z $s$ do $t$.

Ak sú v tomto výslednom grafe aj vrcholy, ktoré vznikli kontrakciou, tak ich budeme postupne extrahovať. Treba dokázať, že týmto sa nám nepokazí dvojsúvislosť. Označme vrchol, ktorý budeme extrahovať $v$ a vyextrahovaný cyklus $C$. Majme dve disjunktné cesty medzi vrcholmi $s$ a $t$.

- Ak neprechádza ani jedna z nich cez $v$, tak nič nepokazíme.
- Ak prechádza jedna, tak prídeme z $s$ do nejakého vrcholu $a$, ktorý leží na $C$ a z $t$ do nejakého $b$, ktorý leží na $C$. Medzi $a$ a $b$ už existuje cesta.
- Ak prechádzajú dve disjunktné cesty cez $v$, tak potom z $s$ prídeme do vrcholov $a$ a $b$ z $C$ a z $t$ prídeme do $c$ a $d$ z $C$ ($a$, $b$, $c$ a $d$ nemusia byť rôzne vrcholy) Pretože $C$ je cyklus, tak vieme isto nájsť disjunktné cesty buď z $a$ do $c$ a z $b$ do $d$, alebo z $a$ do $d$ a z $b$ do $c$. Takže aj v novom grafe vieme skonštruovať dve disjunktné cesty medzi $s$ a $t$.

Tak to by sme mali. Správnym riešením je teda obyčajný algoritmus v $O(n)$, ktorý nám spočíta počet listov.

## Listing programu:

```
#include<cstdio>
using namespace std;
const int MAX=100000;
int main(){
  int n;
  scanf("%d",&n);
  int v[MAX]={0};
  for(int i=1;i<n;i++){
    int a, b;
```

```
    scanf("%d %d", &a, &b);
    v[a]++; v[b]++;
  }
  int list=n;
  for(int i=0;i+n;n--) if(v[n]-1) list--;
  printf("Treba pridat %d hran/u/y\n",(list+1)/2);
  return 0;
}
```

opravoval MišoF.
(max. 15 bodov)

## 5. Ostávame pri kryptológii

Nuž, správnych riešení bolo síce poskromne, ale nezúfajte, chyba nie je vo vás, úloha bola naozaj ťažká. (Aby ste mali predstavu: s podobnými úlohami ako bola časť bé sa môžete stretnúť tak vo štvrtom ročníku na matfyze...)

Pozrime sa teda na to, ako sa to celé malo riešiť.

**Protokol a)**

1. $A \to B : \{N_A, A\}_{K_B}$
2. $B \to A : \{N_A, N_B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$

Správu $\{MSG\}_{K_X}$ vie vyrobiť hocikto, prečítať len $X$.

**Útok na protokol a)**

Alica práve chce komunikovať s Oskarom a poslala mu prvú správu:

1. $A \to O : \{N_A, A\}_{K_O}$

A čo spraví Oskar? Pošle Bobovi správu, ktorá sa tvári, že je od Alice.

1.′ $O(A) \to B : \{N_A, A\}_{K_B}$

Nič netušiaci Bob odpovie Alici, správu si ale odchytí Oskar.

2.′ $B \to O(A) : \{N_A, N_B\}_{K_A}$

Oskar túto správu síce nevie prečítať, vie ju ale poslať Alici ako druhý krok protokolu medzi Alicou a ním. (T.j. akoby si aj on zvolil ten istý reťazec $N_B$.)

2. $O \to A : \{N_A, N_B\}_{K_A}$

Alica Oskarovi odpovie.

3. $A \to O : \{N_B\}_{K_O}$

Normálne by si teraz Oskar mal overiť, či $N_B$ sedí s tým, ktoré si zvolil. V skutočnosti sa však stalo niečo úplne iné – Oskar sa dozvedel reťazec $N_B$, ktorý si zvolil Bob. Pomocou neho teraz Boba definitívne presvedčí, že je Alica.

3.′ $O(A) \to B : \{N_B\}_{K_B}$

**Zabezpečenie protokolu a)**

Problém tohto protokolu, ktorý ukazuje náš útok, spočíva v tom, že správu $\{N_A, N_B\}_{K_A}$ mohol Oskar použiť v inej komunikácii ako v tej, do ktorej patrila. Tohto problému sa vieme zbaviť tak, že do tejto správy pridáme aj ID oboch komunikujúcich strán. Opravený protokol:

1. $A \to B : \{N_A, A\}_{K_B}$
2. $B \to A : \{A, B, N_A, N_B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$

**Protokol b)**

1. $A \to B : A, N_A$
2. $B \to T : B, N_B, \{A, N_A\}_{K_B}$
3. $T \to A : N_B, \{B, K, N_A\}_{K_A}, \{A, K, N_B\}_{K_B}$
4. $A \to B : \{A, K, N_B\}_{K_B}, \{N_B\}_K$

Správu $\{MSG\}_{K_X}$ vie vyrobiť aj prečítať len $X$ a $T$.

**Útok na protokol b)**

Zo zadania máme začiatok útoku:

1. $A \to O(B) : A, N_A$

$1'$. $O(B) \to A : B, N_A$

$2'$. $A \to O(T) : A, N'_A, \{B, N_A\}_{K_A}$

Ukážeme, že bez toho, aby Bob čokoľvek tušil a akokoľvek sa zapojil, vie Oskar dokončiť úspešne celý beh protokolu s Alicou.

V 2. kroku normálneho protokolu by mal Bob poslať Trudy nejaké údaje. Toto ale Oskar bez zapojenia Boba nedokáže – časť z tých údajov musí totiž byť šifrovaná kľúčom $K_B$. Druhý krok sa ale Alice netýka, tá očakáva až v treťom kroku správu od Trudy. No a tú musí teraz Oskar nejako sfalšovať.

Pozrime sa, čo má k dispozícii. Tým, že začal celý protokol ešte raz, podarilo sa mu v kroku $2'$ dostať od Alice správu $\{B, N_A\}_{K_A}$. Hm... nevedel by Oskar pomocou nej presvedčiť Trudy, že je Alica? A naozaj:

$2''$? $O(A) \to T : A, X, \{B, N_A\}_{K_A}$

Z pohľadu Trudy je toto korektná správa od Alice, ktorá chce komunikovať s Bobom. Preto odpovie Bobovi... teda vlastne Oskarovi, ktorý si správu opäť odchytí. Všimnite si, že $X$, ktoré Oskar poslal, môže byť ľubovoľný reťazec. Trudy na správu takéhoto tvaru odpovie:

$3''$? $T \to O(B) : X, \{A, K, N_A\}_{K_B}, \{B, K, X\}_{K_A}$

Oskar teda vie od Trudy „vymámiť“ nejaké správy. Ale aké vlastne potrebuje? Správa, ktorú by mala v 3. kroku poslať Trudy Alici vyzerá nasledovne:

3.? $T \to A : N_B, \{B, K, N_A\}_{K_A}, \{A, K, N_B\}_{K_B}$

$N_B$ si má voliť Bob, z pohľadu Alice môže byť ľubovoľné. Ako $N_B$ môže Oskar teda použiť čokoľvek, čo sa mu práve hodí. No a navyše si môže vhodne zvoliť $X$ v správe, ktorú pošle Trudy... a vyhrali sme.

Útok teda pokračuje nasledovne:

$2''$. $O(A) \to T : A, N_A, \{B, N_A\}_{K_A}$

$3''$. $T \to O(B) : N_A, \{A, K, N_A\}_{K_B}, \{B, K, N_A\}_{K_A}$

A teraz už nič nebráni Oskarovi zobrať dve časti správy, ktoré práve dostal, zatváriť sa, že je Trudy a pokračovať v pôvodnej komunikácii s Alicou:

3. $O(T) \to A : N_A, \{B, K, N_A\}_{K_A}, \{A, K, N_A\}_{K_B}$

Alica si odšifruje svoju časť správy, spokojne skontroluje, že s ňou chce hovoriť Bob a že $N_A$ sedí s tým, čo si zvolila... a pošle štvrtú správu, ktorú Oskar len zahodí.

4. $A \to O(B) : \{A, K, N_A\}_{K_B}, \{N_A\}_K$

**Zabezpečenie protokolu b)**

V prvom rade musíme zodpovedať otázku: Je toto vlastne útok? Protokol síce úspešne skončil, ale Oskar sa predsa kľúč $K$ nedozvedel... Naozaj, táto slabina nie je až taká veľká. Oskar v podstate nezískal takmer nič navyše oproti tomu, čo by získala Eva, keby len odpočúvala komunikáciu Alice, Boba a Trudy. Jediný rozdiel je teda v tom, že Alica je presvedčená, že Bob pozná $K$ – hoci v skutočnosti tomu tak nie je. Aj z tohto môžu občas vzniknúť problémy... tie už ale nechám na vašu predstavivosť.

Slabina protokolu je opäť v jeho „symetrii“, teda v podobnosti častí použitých správ. Všimnite si, že v kroku 3 Oskar použil dve časti správy (ktoré nevedel prečítať) v opačnom poradí ako ich dostal v kroku $3''$. Takéto symetrie sa väčšinou dajú nejako zneužiť, preto je vhodné, aby sa o každej správe v protokole vedelo, „kam patrí“.

V našom prípade si vieme pomôcť jednoducho:

1. $A \to B : A, N_A$
2. $B \to T : B, N_B, \{A, N_A\}_{K_B}$
3. $T \to A : N_B, \{0, B, K, N_A\}_{K_A}, \{1, A, K, N_B\}_{K_B}$
4. $A \to B : \{1, A, K, N_B\}_{K_B}, \{N_B\}_K$

A to už je na dnes všetko, dobrú noc :-)

# Výsledková listina po 2. kole kategórie KSP

| | Meno a priezvisko | Škola | Trieda | | 21 | 22 | 23 | 24 | 25 | Σ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Mikuláš Ján | Gym. Haličská Lučenec | 3 | 67 | 13 | 8 | 15 | 15 | 6 | 124 |
| 2 | Fedák Matúš | Gym. Stará Ľubovňa | 3 | 61 | 8 | 3 | 15 | 10 | 15 | 112 |
| 3 | Bundala Daniel | Gym. Jura Hronca BA | 3 | 47 | 13 | 9 | 12 | 15 | 6 | 102 |
| 3 | Imriška Jakub | Gym. Jura Hronca BA | 3 | 51 | 9 | 13 | 15 | 14 | | 102 |
| 5 | Bílka Ondřej | | 3 | 39 | 12 | 11 | 14 | 8 | 13 | 97 |
| 6 | Sudolský Michal | Gym. Tajovského B. Bystrica | 2 | 63 | 5 | | 8 | | 7 | 83 |
| 7 | Herman Peter | Gym. Jura Hronca BA | 2 | 44 | 12 | 5 | 10 | | 6 | 77 |
| 8 | Jerguš Ján | Gym. Alejová Košice | 2 | 50 | 11 | 7 | 8 | | | 76 |
| 9 | Krchniak Matej | Gym. Jura Hronca BA | 3 | 25 | 9 | 8 | 14 | 10 | 9 | 75 |
| 10 | Petruchová Zuzana | Gym. Grösslingová BA | 3 | 47 | | 10 | 9 | | | 66 |
| 11 | Ďuďák Juraj | Gym. Golianova Nitra | 3 | 21 | 5 | | 15 | 10 | 7 | 58 |
| 12 | Pančík Andrej | Gym. Tajovského B. Bystrica | 2 | 29 | 7 | | 8 | 4 | 6 | 54 |
| 12 | Rampášek Ladislav | Gym. Jura Hronca BA | 2 | 33 | 2 | 6 | 5 | | 8 | 54 |
| 14 | Okruhlica Adam | Gym. Jura Hronca BA | 2 | 43 | | | | | | 43 |
| 15 | Perešíni Peter | Gym. Tajovského B. Bystrica | 3 | 0 | 8 | 13 | 15 | | 6 | 42 |
| 16 | Bodnár Jozef | Gym. Fiľakovo | 4 | 22 | | | | | | 22 |
| 17 | Schuster Vladimír | Gym. Jura Hronca BA | 1 | 15 | 4 | | | | | 19 |
| 18 | Buštor Ivan | Gym. Jura Hronca BA | 2 | 0 | 4 | | 9 | | 5 | 18 |
| 19 | Ďurfina Lukáš | Gym. Golianova Nitra | 4 | 15 | | | | | | 15 |
| 20 | Lobotková Martina | Gym. Grösslingová BA | 3 | 0 | | | 9 | | | 9 |
| 21 | Zaťko Maroš | Gym. Ľudovíta Štúra Trenčín | 1 | 7 | | | | | | 7 |