Korešpondenčný seminár z programovania
XXV. ročník, 2007/08
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporujú: MICROSTEP-MIS spol. s r.o.
Whitestein Technologies spol. s r.o.

Vzorové riešenia 1. kola zimnej časti

Milé naše riešiteľky, milí naši riešitelia,

práve sa Vám dostávajú do rúk vzorové riešenia prvého kola nášho skvelého seminára na neuveriteľných deviatich listoch. To však ale nevadí, však každý večer je čoraz dlhší a dlhší... Užite teda naše vzoráky podľa Vášho najlepšieho vedomia a svedomia a nezabudnite poslať riešenia ďalšej série.

Komu sa nelení, tomu treba roboty naložiť.

KSPáci

1. Zrátajme si jedničky

opravoval KuKo & Peto
(max. 15 bodov)

Poďme na to hneď zhurta; bez okolkov a zbytočnej omáčky. Väčšina riešení mala vyzerať a veru sa aj podobala na niečo takéto:

```
var n, h : longint;

begin
  readln (n); h := 0;
  while (n > 0) do begin
    h := h + (n mod 2); { alebo if (n mod 2 = 1) then inc (h); }
    n := n div 2;
  end;
  writeln ('Cislo ma ', h, ' jedniciek.');
end.
```

Ako to funguje? Nuž ak ciferný zápis čísla N v dvojkovej sústave je $a_m \dots a_2 a_1 a_0$ (teda $a_i \in \{0,1\}$, $0 \leq i \leq m$), potom ak ho (celočíselne so zvyškom) vydelíme dvojkou, dostaneme $a_m \dots a_2 a_1$ a zvyšok a_0. Prečo? Lebo číslo N sa dá zapísať takto:

$$\begin{aligned} N &= a_m \cdot 2^m + \dots + a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0 \\ &= 2 \times (a_m \cdot 2^{m-1} + \dots + a_3 \cdot 2^2 + a_2 \cdot 2^1 + a_1 \cdot 2^0) + a_0, \end{aligned}$$

kde a_0 je posledná cifra dvojkového zápisu. Takto teda vieme zistiť poslednú cifru a zvyšok čísla (bez tejto cifry). Opakovaním tohto postupu vieme postupne zisťovať všetky cifry a počítať jednotky.

Mimochodom, takýmto spôsobom môžeme robiť ciferný súčet v ľubovoľnej sústave: delíme základom sústavy a sčítavame zvyšky (predstavte si to v desiatkovej sústave).

Časová zložitosť je úmerná *počtu cifier* čísla N (nie číslu N samotnému). ((Keďže m-ciferné binárne čísla sú od 2^{m-1} po 2^m, číslo N má rádovo $\lg N$ cifier, ale to nebolo treba, preto je to až v dvoch zátvorkách)). Pamäťová zložitosť je, samozrejme, konštantná.

Ak ste napísali toto všetko, stačilo to na plných 15 (so 4 jednotkami) bodov. Niektorí to však mali trochu pomalšie, alebo si pamätali celý binárny zápis v stringu. Niektorým chýbal

popis, niektorí mali zlý odhad zložitosti, ba čo horšie, niektorí mali bug a ešte horšie, niektorí aj odpisovali. Za to sa stŕhali body.

Toto bol prvý príklad, mienený ako najľahší a podľa toho sa aj bodovalo. Uvedené riešenie však nie je najlepšie možné riešenie – odvážni a zvedaví riešitelia a iní dobrodinci, čítajte ďalej!

Čo budeme na lepšie riešenia potrebovať? Okrem klasického sčítania a odčítania budeme používať tzv. *bitové operácie* (také, čo pracujú s bitmi – jednotlivými ciframi čísla), ako je and, or a bitové posuny shl a shr[1]. Každý z vás zrejme pozná logické spojky and a or. Ak zapíšeme pravdivostné hodnoty ako 0 a 1, platí

$$x \text{ and } y = \begin{cases} 1 & \text{ak } x = 1 \text{ aj } y = 1 \\ 0 & \text{inak} \end{cases} \qquad x \text{ or } y = \begin{cases} 1 & \text{ak } x = 1 \text{ alebo } y = 1 \\ 0 & \text{inak} \end{cases}$$

(Dôležité pozorovanie je, že x and 0 je vždy 0 a x and $1 = x$.) Obdobné operácie existujú aj pre čísla, pričom sa and-ujú / or-ujú príslušné bity. Skúste si napríklad v Pascale napísať writeln(22 and 27)[2] – vypíše sa 18 (lebo binárne zápisy sú 10110 a 11011, ak and-neme príslušné bity, dostaneme 10010). Podobne 18 or 11 je 27 (lebo 10010 or 01011 je 11011). Operácie shl a shr[3] posúvajú bity doľava, resp. doprava (o daný počet bitov), pričom prázdne miesto sa doplní nulami. Napríklad ak posunieme postupnosť 0001 0111 doľava o 1, dostaneme 0010 1110, ak doprava o 1, dostaneme 0000 1011. Teda napríklad $5 \text{ shl } 1 = 10$, $5 \text{ shl } 2 = 20$, $5 \text{ shr } 1 = 2$, $5 \text{ shr } 2 = 1$ a $5 \text{ shr } 3 = 0$. Všimnite si, že $x \text{ shl } 1$ je to isté ako $2 \cdot x$, resp. $x \text{ shl } k$ je to isté ako $2^k \cdot x$. Podobne $x \text{ shr } 1 = \lfloor x/2 \rfloor$ a všeobecne $x \text{ shr } k = \lfloor x/2^k \rfloor$. Pracujeme predsa v dvojkovej sústave.

Ako toto využijeme? Nuž keď počítač pracuje v dvojkovej sústave a všetky čísla sú uložené v dvojkovej sústave, malo by sa dať na číslo jednoducho „pozrieť a vidieť“, nie pracne ich deliť 2. To sa naozaj dá: na posledný bit (poslednú cifru v dvojkovej sústave) čísla N sa vieme „pozrieť“ tak, že vypočítame N and 1. Prečo? Rozmyslite si, čo sa stane. Číslo 1 je vlastne 0000 0001, takže všetky bity čísla N sa budú and-ovať s 0 (a výsledok bude 0), až na posledný bit, ktorý sa and-uje s 1. Keď zistíme posledný bit, stačí číslo posunúť o 1 doprava. Takto dostávame program č. 2:

```
while (n > 0) do begin
  h := h + (n and 1); { pozrieme sa na posledny bit}
  n := n shr 1;       { posunieme cislo doprava }
end;
```

Tento program nie je lepší ako náš prvý (je to iba rozcvička). Jedinou výhodou je, že namiesto div a mod používa oveľa rýchlejšie operácie shr a and. Avšak každý čo len trochu optimalizujúci kompilátor túto zámenu spraví za vás.

Trochu lepší program dostaneme pomocou nasledovnej finty: vezmime si číslo N a $N - 1$ v dvojkovej sústave (napríklad 0101 1000 a 0101 0111). Čím sa líšia tiet dve čísla? Pri odčítaní jednotky sa najpravejší jednotkový bit čísla N sa zmení na nulu a všetky nuly vpravo od neho sa zmenia na jednotky.[4] Ak teraz obe čísla z-and-ujeme, dostame $N' = 0101\,0000$, čo je číslo N s vynulovaným najpravejším jednotkovým bitom! [5] Ak tento postup zopakujeme, dostaneme $N' - 1 = 0100\,1111$ a $N'' = N' \text{ and } (N' - 1) = 0100\,0000$ (číslo N' bez najpravejšej jednotky). Ak to zopakujeme ešte raz, dostaneme nulu. Takýmto spôsobom

[1] shift left, shift right

[2] v C-éčku sa bitový and a or zapisujú & a |

[3] v C je to $\ll$ a $\gg$

[4] Podobne je to aj v iných sústavách; napríklad v desiatkovej sa posledná nenulová cifra zníži o 1 a všetky nuly vpravo od nej sa zmenia na deviatky, napr. $12300 - 1 = 12299$

[5] Rozmyslite si to.

teda vieme vynulovať najpravejší jednotkový bit v konštantnom čase. Stačí teda počítať počet opakovaní, kým sa nedostaneme na číslo bez jednotiek – na nulu.

```
while (n > 0) do begin
  n := n and (n-1); { odstranime poslednu jednotku }
  h := h + 1;       { zapocitame ju }
end;
```

Tento program je trochu rýchlejší: nezávisí od dĺžky čísla, ale od počtu jednotiek. Odvážni riešitelia dočítavší sa až sem sa, dúfam, dozvedeli čo-to o bitových operáciach. Toto však stále nie je najlepšie riešenie. Tí dobrodružnejší z vás, čítajte ďalej.

Ukážeme si riešenie, ktoré behá v čase úmernom logaritmu počtu cifier čísla. Mimochodom tento príklad sa vyskytol aj v domácom kole MOP-ky v roku 2004, takže podrobnejšie vysvetlenie nájdete na `http://www.ksp.sk/oi/archive.php?year=2004`.

Jednotlivé cifry budeme sčítavať v kolách. Bity si rozdelíme do blokov, pričom v každom bloku bude počet jednotiek na danom úseku. V prvom kole sčítame naraz(!) všetky dvojice susedných bitov a dostaneme tak bloky dĺžky 2, v ktorých je vždy počet jednotiek v príslušnom úseku dĺžky 2. V druhom kole naraz sčítame všetky susedné dvojice dvojbitových blokov a dostaneme tak štvorbitové bloky. V treťom kole sčítame dvojice 4-bitových blokov a dostaneme 8-bitové bloky. Potom 16-bitové, 32-bitové bloky, atď. (kým nám neostane jeden jediný blok).

Ukážme si túto myšlienku na príklade. Na vstupe je číslo 10110110, ktoré má 5 jednotiek. Toto jedno číslo vždy rozdelíme do niekoľkých skupiniek (v príklade sú oddelené čiarou), pričom v každej skupinke je počet jednotiek v danom úseku. (Na pravej strane sú jednotlivé čísla prepísané do desiatkovej sústavy.)

```
1|0|1|1|0|1|1|0        1|0|1|1|0|1|1|0
 01| 10| 01| 01          1|  2|  1|  1
   0011|   0010             3|      2
       00000101                      5
```

Ostáva ukázať, ako sčítame naraz susedné bloky. Urobíme to nasledovne: Pomocou and-u vieme vybrať vhodné bity. Napríklad párne bity vyberieme tak, že číslo z-and-ujeme s maskou $\ldots 0101\,0101$[6] (označme výsledok p). Nepárne bity dostaneme andom s maskou $\ldots 1010\,1010$ (výsledok označme q). Následne zarovnáme bity pod seba – číslo q shiftneme o 1 doprava a sčítame. Teda $p \leftarrow (N \text{ and } 0101\,0101)$, $q \leftarrow (N \text{ and } 0101\,0101)$ a $N' \leftarrow p + (q \text{ shr } 1)$. Dostávame bloky dĺžky 2.

S blokmi dĺžky 2 to bude podobné: tie párne vytiahneme andovaním s $\ldots 0011\,0011$, tie nepárne andovaním s maskou $\ldots 1100\,1100$. Posunieme o 2 a sčítame. Tak dostaneme bloky dĺžky 4. Párny získame andovaním s maskou $\ldots 0000\,1111$, nepárny získame andovaním s maskou $\ldots 1111\,0000$. Posunieme o 4 a sčítame. Nasledovný zdroják je pre 16-bitový pascalovský word. Pre 32-bitové čísla by pribudol ešte jeden riadok a pre 64-bitové ešte ďalší riadok navyše a trochu by sa zmenili konštanty (jednotlivé masky) – tie sú kvôli prehľadnosti(?)[7] zapísané v šestnástkovej sústave (v Pascale sa také to čísla označujú dolárom). Výhodou šestnástkovej sústavy je, že každá cifra zodpovedá štyrom bitom (takže jednak vieme čísla ľahko zapisovať a čítať, jednak je takýto hexadecimálny zápis omnoho kratší (napríklad a

[6] na začiatku čísla sú tri bodky, pretože v skutočnosti máme na mysli číslo, kde sa striedajú nuly a jednotky potrebnej dĺžky – podľa toho, či pracujeme s 16-, 32-, alebo 64-bitovými číslami (alebo s číslami inej dĺžky)

[7] ako pre koho?

zodpovedá 1010, 3 zodpovedá 0011 a c zodpovedá 1100)).

```
var n : word;
begin
  readln (n);
  n := (n and $5555) + ((n and $aaaa) shr 1);
  n := (n and $3333) + ((n and $cccc) shr 2);
  n := (n and $0f0f) + ((n and $f0f0) shr 4);
  n := (n and $00ff) + ((n and $ff00) shr 8);
  writeln ('Cislo ma ', n, ' jedniciek.');
end.
```

Časová zložitosť je logaritmus dĺžky premennej, s akou pracujeme. (Napríklad keby sme pracovali s 1024-bitovými premennými, stačilo by nám 10 takých príkazov). Priznám sa, že nepoznám nič s lepšou zložitosťou. (Presnejšie nič lepšie v modeli AC^0-RAM. To je taký, kde môžeme používať napríklad operácie $+$, $-$, and, or a bitové posuny v konštantnom čase, ale nie násobenie, delenie, modulo...) Ak by sme povolili ľubovoľne dlhé registre a násobenie v konštantnom čase, vedeli by sme to aj v konštantnom čase. (Ale od praxe to má ďaleko, potrebovali by sme veľmi veľké čísla – na sčítanie 8 bitov by sme potrebovali 64 bitové a na sčítanie 64 bitov 4096-bitové.)

2. Zajačia Farma

Opravoval Hermi & Mic
(max. 15 bodov)

Nuž, holúbätká, musím priznať, že sa vašich riešení veru požehnane u nás zišlo. Najprv si povieme ako sa hodnotilo, čo to poznamenám k niektorým veciam, ktoré bolo vhodné si uvedomiť, a vysvetlím vám, prečo rekurzia nebola to pravé orechové.

V zásade prichádzali 3 typy riešení:

- Čas $O(N)$, pamäť $O(1)$. Toto bolo to najlepšie, na čo ste sa zmohli a vyslúžilo to autorovi 14 bodový základ.
- Čas $O(N)$, pamäť $O(N)$. Za takúto nádielku ste dostávali 12 bodov. No a nakoniec:
- Čas exponenciálny, pamäť $O(1)$ (V skutočnosti $O(N)$, lebo sa volania funkcie ukladajú na stack, ale nechcel som vás už moc deptať). Toto riešenie dostalo štedrých 9 bodov... (a automaticky aj bod dole kvôli zlému odhadu)

Iste si mnohí z vás kladú otázku, kde sa vzal ten exponenciálny čas. Veľa z vás proste slepo prepísalo definíciu do rekurzie a myslelo si, že má vyhrané. Neuvedomili ste si však, že vďaka tomu, že si vypočítané hodnoty nikam neukladáme, rátame niektoré veci viackrát. Pozrime sa teraz, ako sa táto funkcia správa. Na vyrátanie F_0 a F_1 nám stačí jeden krok, čo taký výpočet F_N? Nóó, to čo robíme je, že sa zavoláme rekurzívne na 2 predchádzajúce hodnoty, čiže potrebujeme vypočítať F_{N-2} a F_{N-1}, čiže počet krokov potrebných na výpočet F_N je rovný súčtu počtu krokov potrebných na vypočítanie dvoch predchádzajúcich plus jedna operácia na sčítanie týchto hodnôt. Možno vám to niečo začína pripomínať. A možno ste si už uvedomili, že Fibonacciho postupnosť rastie exponenciálne. A presnejšie je časová zložitosť $O((\frac{1+\sqrt{5}}{2})^N)$. A problém je na svete...

A čo s tou pamäťou? Stačí sa zamyslieť a hneď si uvedomíme, že vždy pri počítaní rátame najviac s dvomi poslednými hodnotami, všetky ostatné sú nám už na nič. Preto je vskutku veľký luxus pamätať si toľko čísiel. Preto si budeme pamätať iba 2 predchádzajúce čísla.

Za čo sa dali stratiť ďalšie body? Veľa z vás si našlo v príklade iba lakonickú poznámku „preteká“. Úlohou bolo vypísať Fibonacciho číslo modulo M. Drvivá väčšina z vás toto interpretovala tak, že vyrátala to, čo bolo treba a pred vypísaním aplikovala modulo. Čo sa však nestane? Fibonacciho čísla rýchlo rastú. A už také päťdesiate sa nám nezmestí do premennej. Čo môžeme robiť je, že čísla modulujeme priebežne. To znamená, že všetky

operácie robíme modulo M, tým dosiahneme to, že celý čas pracujeme s číslami rádovo tak veľkými ako M. (Za domácu úlohu je rozmyslieť si, prečo si to môžeme dovoliť).

Ďalej sa mohli stratiť body za zlý odhad, poprípade bug v programe (niektorí z vás pre malé N dávali chybné výstupy).

Korektné 14 bodové riešenie v $O(N)/O(1)$ malo teda vyzerať zhruba takto:

```
var a : array[0..2] of integer;
    n, m, i : integer;

begin
  read (n, m);
  if (n < 2) then writeln (n mod m)
  else begin
    a[0] := 0; a[1] := 1;
    for i := 1 to n-1 do begin
      a[2] := (a[0]+a[1]) mod m;
      a[0] := a[1]; a[1] := a[2];
    end;
    writeln (a[2]);
  end;
end.
```

Teraz poďme posunúť hranice ešte o kúsok ďalej. Verte tomu, alebo nie, príklad sa dá vyriešiť aj v $O(\log n)$! Možno ste sa už stretli so vzorčekom na výpočet N-tého Fibonacciho čísla.

$$F_N = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^N - \left(\frac{1-\sqrt{5}}{2} \right)^N \right)$$

Hovoriť o tom, ako sme sa k tomuto vzorcu dostali je bohužiaľ nad rámec tohto vzoráku. Tí, čo predsalen majú záujem o dôkaz, nech si pozrú `http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fibformproof.html`. Sú tam hneď dva pekné dôkazy toho vzorčeka. Pozrime sa však na čo sa nám náš problém zredukoval – potrebujeme umocniť číslo na N-tú. A tu príde veľká finta pánov informatikov. Vyrátať N-tú mocninu čisla vieme v čase $O(\log N)$, bližšie to bude popísané na konci vzoráku.

Problém s reálnymi číslami je, že nevieme priebežne robiť modulo. To znamená, že nie sme schopní vypočítať výsledok pre také veľké N, ako keď sme používali predchádzajúci lineárny algoritmus (reálne čísla v počítači nie sú implementované úplne presne, takže pre veľké N už nedostaneme presný výsledok). Preto budeme hľadať riešenie, ktoré je v čase $O(\log N)$ a zároveň využíva iba celé čísla.

K logaritmickému riešeniu s použitím len celých čísiel nám pomôžu matice a násobenie matíc. Presnejšie to, čo by sme chceli je zobrať nejakú štvorcovú maticu veľkosti 2×2, umocniť ju na N a potom z nej odčítať N-té Fibonacciho číslo. Najskôr by sme si však mali povedať, ako sa také matice násobia. Dve matice veľkosti 2×2 vynásobíme takto:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

Všimnime si teraz nasledovné:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} a+c & b+d \\ a & b \end{pmatrix}$$

Teraz poďme trošku rozmýšľať. Keď si pozrieme prvý stĺpec výsledku, tak vidíme, že zo stĺpca (a, c) sme dostali stĺpec $(a+c, a)$. Teda v prvom stĺpci sa nám generujú Fibonacciho

čísla. To isté sa nám deje v druhom stĺpci. Ďalšie pozorovania:

$$\begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} F_{N+1} & F_N \\ F_N & F_{N-1} \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F_{N+2} & F_{N+1} \\ F_{N+1} & F_N \end{pmatrix}$$

To znamená nasledovnú vec:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^N = \begin{pmatrix} F_{N+1} & F_N \\ F_N & F_{N-1} \end{pmatrix}$$

Keď N krát vynásobím našu maticu, tak v nej budem mať N-té Fibonacciho číslo. Teraz už len ako umocňovať. Podotýkam, že na umocňovanie matíc bude fungovať úplne rovnaký algoritmus ako na umocňovanie obyčajných čísiel, iba s tým rozdielom, že namiesto obyčajného násobenia, budeme násobiť matice.

Teraz nám už iba ostáva vyriešiť umocňovanie v $O(\log N)$. Pre jednoduchosť sa budeme tváriť, že umocňujeme normálne číslo. Čo vieme robiť ľahko je umocnenie na druhú, tzn. vynásobenie exponentu dvomi (umocňovať na mocninu dvojky). Čiže všetky exponenty, ktoré su moncinou dvojky vieme dostať triviálne na pár umocnení. A vieme aj, že pri násobení sa nám stane taká habaďúra $a^x \cdot a^y = a^{x+y}$. Čo nám teda stačí urobiť? Stačí nám poskladať číslo z mocnín dvojky. To je jednoduché, proste vezmeme zápis N v dvojkovej sústave, a ak tam je i-tom mieste zprava 1ka vieme, že 2^i-tu mocninu treba pridať. Napríklad ak chceme umocniť číslo a na 12, tak umocníme na 4 a na 8 a potom výsledky znásobíme a teda dostaneme $a^8 \cdot a^4 = a^{8+4} = a^{12}$.

Bonus: N-tú mocninu reálneho čísla ide vypočítať aj v pseudo-$O(1)$ čase. Pseudo preto, lebo síce to bude na počítači bežať v konštantnom čase, ale len vďaka tomu, že má obmedzený rozsah premenných. Na to nám pomôžu funkcie exp a ln. $exp(x)$ je e^x a ln je prirodzený logaritmus. Obe tieto funkcie sú v počítači implementované v konštantnom čase (ale využívajú to, že rozsah reálnych čísiel je limitovaný). Ako sa to dá využiť? Iste mi všetci veríte, že platí, ak napíšem $e^{ln(a)} = a$. Teda keď napíšem $e^{ln(a)x} = a^x$ tak mi tiež uveríte. Preto keď chcem vypočítať a^x, stačí napísať $exp(x \cdot ln(a))$.

Listing programu:

```
type TMatica = array [1..2,1..2] of longint;
var Fib: TMatica;
    n, m: longint;

function multiply (x, y: TMatica) : TMatica;
begin
  multiply[1,1]:=(x[1,1]*y[1,1]+x[1,2]*y[2,1]) mod m;
  multiply[1,2]:=(x[1,1]*y[1,2]+x[1,2]*y[2,2]) mod m;
  multiply[2,1]:=(x[2,1]*y[1,1]+x[2,2]*y[2,1]) mod m;
  multiply[2,2]:=(x[2,1]*y[1,2]+x[2,2]*y[2,2]) mod m;
end;

function pow (x: TMatica; n: longint) : TMatica;
var res: TMatica;
begin
  res[1,1]:=1;
  res[1,2]:=0;
  res[2,1]:=0;
  res[2,2]:=1;
  while (n > 0) do begin
```

```
    if (n mod 2 = 1) then res:=multiply(res,x);
    x := multiply(x,x);
    n := n div 2;
  end;
  pow := res;
end;

begin
  Fib[1,1]:=1;
  Fib[1,2]:=1;
  Fib[2,1]:=1;
  Fib[2,2]:=0;
  readln (n, m);
  writeln(n,m);
  writeln (pow(Fib, n)[1,2]);
end.
```

3. Zase tie preteky...

Opravoval Stano
(max. 15 bodov)

Tento príklad sa mi nezdal veľmi ťažký, preto ma prekvapilo malé množstvo riešiteľov. Najprv načrtnem jednoduché riešenie s časovou zložitosťou $O(N^2)$. Pripomeniem, že ak N je počet vrcholov v strome, potom hrán je $N-1$ a medzi každou dvojicou vrcholov vedie práve jedna cesta.

Ako nájsť najdlhšiu cestu v strome? Vysvetlím algoritmus prehľadávania do hĺbky. Začnem v nejakom vrchole a zaznačím si, že mám cestu dĺžky 0. Zároveň si ho zapíšem do pomocného poľa P na prvú pozíciu a postupne do poľa budem zapisovať ďalšie vrcholy, cez ktoré som prešiel. Ďalej si zaznačím, že som už tento vrchol navštívil, keby som doňho mohol prísť znova, tak to už nespravím. Čo sa nám na strome nestane, ale na iných grafoch môže.

Potom sa rekurzívne zavolám na svojich susedov a dám ako parameter aj dĺžku cesty+1.

Ak prídem na vrchol ktorý nemá neprehľadaných potomkov, znamená to, že cez tento vrchol už neviem nijak predĺžiť cestu z počiatočného vrcholu. Tak sa len pozriem, či je týmto vrcholom ukončená cesta dlhšia ako doteraz najdlhšia nájdená. Ak áno, zapamätám si ju. Teraz sa potrebujem z vrcholu vrátiť. Vyhodím z poľa P posledný vrchol a zaznačím, že som skrátil cestu o 1.

Treba dávať pozor, aby sme mali rozumne implementovaný graf. V tomto príklade je ideálny zoznam susedov vrcholu, lebo tak hľadaním susedných vrcholov pri prehľadaní celého grafu minieme len čas úmerný počtu hrán, ktorý je v tomto príklade $O(N)$.

Implementovať sa dá napríklad poľom dynamických polí, alebo aj v poli $N \times N$. Alebo aj v jednom poli, tak ako je vo vzoráku.

Nevhodná je matica susednosti, teda pole $N \times N$, kde je uložené 1 ak sú vrcholy susedné a 0 inak. Hľadanie susedov jedného vrcholu spotrebuje až $O(N)$ operácií, teda celkovo až $O(N^2)$!

Najdlhšia cesta sa začína aj končí v nejakom vrchole. Keď teda spustím prehľadávanie z každého vrcholu, určite začnem v nejakom, v ktorom začína najdlhšia cesta. Pri prehľadávaní z tohto vrcholu sa ale dostanem aj tam, kde najdlhšia cesta končí a preto ju nájdem.

To celé iba za čas $O(N^2)$!

Avšak existuje aj rýchlejšie riešenie, s časovou zložitosťou iba $O(N)$. Ako na to?

Zoberieme si ľubovoľný vrchol v a označíme si ho ako koreň. Zavedieme si funkciu $f(v)$, ktorá bude znamenať dĺžku cesty od koreňa do vrchola v. Ďalej budeme používať označenie *hore*, ak ideme z vrcholu v_1 do v_2, tieto dva vrcholy sú susedné a platí $f(v_1) > f(v_2)$.

Označenie *dole*, ak $f(v_1) < f(v_2)$. Ak sa hýbeme iba smerom hore, navštívime iba svojich *predkov*. Naopak, ak sa hýbeme iba dole, navštívime iba svojich *potomkov*.

Zadefinujme si teraz ešte jeden pojem. Nech $w_1, w_2, \ldots, w_k$ je najdlhšia cesta taká, že sa začína vo vrchole w_1 a platí $f(w_i) < f(w_{i+1})$ pre všetky i. Inými slovami je to najdlhšia cesta, ktorá ide z vrcholu w_1 iba dole. Za vrchol w_1 zvoľme postupne každý vrchol a dĺžku takejto najdlhšej cesty, označme $d(w_1)$.

Predpokladajme, že y je vrchol najdlhšej cesty v celom grafe taký, že je najbližšie ku koreňu. Máme 2 možnosti, ako môže vyzerať táto najdlhšia cesta. Buď sa vo vrchole y začína, a teda pokračuje ďalej iba smerom dole. Takáto cesta má dĺžku $d(y)$. Môžete si rozmyslieť, že táto hodnota bude dĺžkou najdlhšej cesty iba ak y je koreň. V druhej možnosti sa vrchol y nachádza niekde vnútri cesty $u_1, \ldots, u_l, v, u_{l+1}, \ldots, u_k$. Uvedomme si, že úseky $u_1, \ldots, u_l$ a $u_{l+1}, \ldots, u_k$ musia byť najdlhšie možné, lebo inak by táto cesta nebola najdlhšia. Preto pre tieto dva vrcholy u_l a u_{l+1} platí, že $d(u_l)$ a $d(u_{l+1})$ sú najväčšie spomedzi všetkých potomkov vrchola y. Akú dĺžku má cesta v tomto druhom prípade? Spolu je to $d(u_l) + d(u_{l+1}) + 1$.

Problém je teda vyriešený, stačí zistiť hodnoty $d(v_i)$ pre každého potomka vrchola v. Ale to môžeme spraviť jednoduchým prehľadávaním do hĺbky. Ak vrchol v nemá potomka (teda je list), potom $d(v) = 1$. Ak má potomkov, rekurzívne sa na všetkých zavoláme. Hodnota $d(v)$ bude $d(v_{\max}) + 1$, kde $d(v_{\max})$ je najväčšia hodnota spomedzi potomkov.

Zároveň v každom vrchole kontrolujeme, či je tento vrchol na ceste dlhšej ako bola doteraz najlepšia nájdená. Ak áno, zapamätáme si tento vrchol.

A teraz sa pozrite na vzorák.

Listing programu:

```
const MAX=100;
var index, count, cesty, bol : array [0..MAX+1] of integer;
 graf : array [1..2*MAX] of integer;
 graf_temp, dlzky, kde : array [0..MAX,1..2] of integer;
 i, a, b, N, best : integer;

procedure vymen (var x, y : integer);
var t : integer; begin t:=x; x:=y; y:=t; end;

function search (v : integer) : integer;
var i, max1, max2, kde1, kde2, temp : integer;
begin
  bol[v]:=1; max1:=0; max2:=0; kde1:=0; kde2:=0;
  for i:=index[v] to index[v+1]-1 do begin
    if (bol[graf[i]]=1) then continue;
    temp:=search(graf[i])+1; {zapamatam si 2 najvacsie}
    if (max2<=temp) then begin max2:=temp; kde2:=graf[i]; end;
    if (max2>=max1) then begin vymen(max2,max1); vymen(kde1,kde2); end;
  end;
  dlzky[v,1]:=max1; dlzky[v,2]:=max2;
  kde[v,1]:=kde1;   kde[v,2]:=kde2;
  if (dlzky[best,1]+dlzky[best,2]<max1+max2) then best:=v;
  search:=max1; bol[v]:=0;
end;

procedure najdicestu(v,p:integer);
begin
  cesty[p]:=v;
  if ((dlzky[v,1]>0) and (kde[v,1]>0)) then najdicestu(kde[v,1],p+1);
end;

begin
  readln(N);
```

```
  for i:=1 to N do count[i]:=0;
  for i:=1 to N-1 do
  begin
    readln(a,b);
    inc(count[a]); inc(count[b]);
    graf_temp[i,1]:=a; graf_temp[i,2]:=b;
  end;

  index[1]:=0;{cesty z prveho vrchola budu zacinat na 0}
  for i:=1 to N do begin
    index[i+1]:=index[i]+count[i];
    count[i]:=0;{pole count zneuzijem na pocet uz zapisanych ciest z vrcholu i}
  end;

  for i:=1 to N-1 do begin
    a:=graf_temp[i,1]; b:=graf_temp[i,2];
    graf[index[a]+count[a]]:=b; inc(count[a]);
    graf[index[b]+count[b]]:=a; inc(count[b]);
  end;

  best:=0; dlzky[0,1]:=0;dlzky[0,2]:=0;
  search(1);
  najdicestu(kde[best,1],1);
  for i:=dlzky[best,1] downto 1 do begin write(cesty[i],' '); end;
  write(best);
  najdicestu(kde[best,2],1);
  for i:=1 to dlzky[best,2] do begin write(' ',cesty[i]); end;
end.
```

..... Tu sa začínajú riešenia KSP-O .....

4. O potápaní

opravovala Elφnka
(max. 15 bodov)

Každú alianciu si budeme pamätať ako strom, v ktorom vrcholy predstavujú jednotlivé lodičky ktoré do aliancie patria. Tú lodička, ktorá je koreňom stromu budeme nazývať šéfom aliancie . Každý vrchol si bude pamätať svojho rodiča. Keďže koreň nemá rodiča, tak si bude pamätať počet lodičiek v aliancii (počet vrcholov v strome). Aby sa nám to neplietlo s číslom svojho rodiča, tak si bude pamätať ako záporné číslo. Inými slovami, ak si vrchol pamätá kladné číslo, tak je to číslo jeho rodiča, ak si pamätá záporné číslo, tak je to koreň.

Budeme využívať nasledujúce 2 funkcie:

- *find* nám nájde koreň stromu v ktorom sa nachádza daný vrchol. Toto sa dá dosiahnuť tak, že prechádza po rodičoch až kým nedorazí ku koreňu stromu.
- *union* spojí dva stromy tak, že pod koreň prvého zavesí druhý strom.

Na začiatku bude každá lodička v aliancii iba sama so sebou. Vždy, keď dve lodičky uzavrú alianciu, najprv pomocou *find* zistíme, či majú ten istý koreň. Ak nie, stromy spojíme pomocou *union*. Keď by sme však vešali pod seba stromy náhodne, mohlo by sa stať, že nám vznikne strom takej hĺbky, koľko v ňom máme vrcholov, a vtedy by nám nájdenie koreňa trvalo lineárne od počtu vrcholov v strome. Aby sme tomu zabránili, budeme napájať vždy menší strom pod väčší. Pri spojení dvoch rôzne veľkých stromov sa nám hĺbka nezmení, ak spájame dva rovnako veľké stromy, zvýši sa o 1. Ak chceme vedieť, či sú 2 loďky spolu v aliancii, stačí pomocou *find* zistiť, či majú spoločný koreň.

Stromy budú mať hĺbku najviac $\log N$, a preto pobeží takéto riešenie v čase $O(M \log N)$. Skúsme ešte zlepšiť funkciu *find*. Keď budeme hľadať cestu od vrchola ku koreňu, tak nastavíme všetkým vrcholom na tejto ceste ako rodiča koreň dého stromu(toto môžeme spraviť tak, že najprv nájdeme koreň a potom prejdeme túto trasu znova a už meníme vrcholom rodiča na koreň). Prečo to robíme? Keď sme spustili *find*, tak sme sa dozvedeli šéfa aliancie pre všetky lodičky, ktoré sme počas nej spracovali. Aby sme to nemuseli robiť nabudúce, tak každému z nich nastavíme ako rodiča nájdeného šéfa, čím každému z nich znížime hĺbku na 1. Toto zrýchlenie nám pomôže k zložitosti $O(N \log^* N)$.

A čo to vlastne $\log^*$ je? Predstavme si čísla 1, 2, 2^2, $2^{(2^2)} = 2^4$, $2^{(2^{(2^2)})} = 2^{(2^4)} = 2^{16}$, $2^{(2^{(2^{(2^2)})})} = 2^{(2^{16})} = 2^{65536}$, ... $\log^* N$ je definovaný ako prvé číslo K také, že K-te číslo z hore vypísaných čísel je väčšie alebo rovné ako N. Ukážme si to teda na príklade:

$\log^* N = 1$ ak $0 \leq N \leq 2$

$\log^* N = 2$ ak $3 \leq N \leq 4$

$\log^* N = 4$ ak $17 \leq N \leq 65536$

Optimálne riešenia dostali 15 bodov, za $O(M \log N)$ ste mohli získať 14 bodov. Za lineárne 12, kvadratické maximálne 10 a kubické 6.

Listing programu:

```
const MAX = 10000;

var n, m, w, a, b, koren_a, koren_b, i:integer;
    size:array[1..MAX] of integer;
    parent:array[1..MAX] of integer;

{ vytvori nam strom pre jednu lodku }
procedure makeset(n:integer);
begin
    parent[n] := n;
    size[n] := 1;
end;

procedure merge(a:integer; b:integer);
var c : integer;
begin
    { OPTIMALIZACIA 1: chceme mat v B cislo korena stromu s mensim poctom
potomkov }
    if size[a] < size[b]  then
      begin c := a; a := b; b := c; end;
    { otec vrcholu B bude A, pretoze strom korena B ma menej vrcholov }
    parent[b] := a;
    { a zvysime pocet potomkov stromu korena A }
    size[a] := size[a] + size[b];
end;

function find(n:integer):integer;
var root, par:integer;
begin
    root := n;
    { najdeme koren }
    while root <> parent[root] do
        root := parent[root];

    { OPTIMALIZACIA 2: nastavime vsetkym predkom vrcholu N ako otca ROOT }
    while n <> parent[n] do begin
        par := parent[n];
```

```
        parent[n] := root;
        n := par;
    end;
    find := root;
end;

begin
    readln(n,m);
    for i := 1 to n do makeset(i);
    for i := 1 to m do begin
        readln(w, a, b);
        if(w = 0) then begin
            { najdeme korene stromov }
            koren_a := find(a);
            koren_b := find(b);
            { a zavolame merge, ak su stromy disjunktne }
            if koren_a <> koren_b then
                merge(koren_a, koren_b);
        end
        else begin
            { pozrieme sa, ci su lodky v tom istom strome }
            if(find(a) = find(b)) then writeln('ANO')
                                  else writeln('NIE');
        end;
    end;
end.
```

Opravoval Laci & Apir
(max. 15 bodov)

5. Ooo, aký mocný odvar!

Rozprávač: Kdesi ďaleko za vrátnicou matfyzu, za siedmimi akvárkami sedeli dve bukvice Apir s Lacim v T2 a hútorili:

Laci: A koľko že prišlo tých riešení čo máme opraviť?

Apir: Mocné množstvo, skoro 90!

Laci: Ej bisťu, tak to bol dáky ľahký príklad, nie?

Apir: Bol. Ale keby poriadne čítali zadanie...

Laci: A čože si to nevšimli?

Apir: No predsa že čarodejníci odmietajú pracovať s desatinnými číslami. Chúďatká...

Laci: Čo si sa zbláznil? Jak to mali odmocniť?

Apir: A vravel im niekto že to majú odmocniť? No nevravel.

Laci: Aháá čiže ty vravíš že to nemali odmocniť... a čo mali potom robiť? Umocňovať? Ha-Ha-Ha!

Apir: No veď samozrejme...

Rozprávač: Prenesme sa teraz ďalej v deji a prejdime k vzoráku:

Na úvod si rozoberme úlohu. Ňou je zoradiť N prirodzených čísel od najmenšieho po najväčšie, pričom niektoré sú pod treťou resp. druhou odmocninou, v prípade rovnosti rozhoduje stupeň odmocniny. **Problém** #1 sú odmocniny, **problém** #2 je triedenie prvkov (pod pojmom triedenie rozumieme zoradenie prvkov podľa voľajakej špecifickej vlastnosti).

Problém #1: Takže najprv sa potrebujeme zbaviť odmocnín, čo mnohí z vás skúšali rôznymi (ne)sofistikovanými spôsobmi a pritom to vôbec nie je také zložité ako sa môže zdať. Keďže nemôžeme pri výpočte pracovať s desatinnými číslami (teda v podstate ani odmocňovať, keďže výsledkom tejto operácie je vo všeobecnosti reálne číslo) a na vstupe sú prirodzené čísla, tak môžeme aspoň umocňovať keďže umocnenie kladných čísiel nezmení

ich vzájomné poradie. Výsledkom umocnenia prirodzeného čísla na prirodzené číslo je vždy číslo prirodzené, čo, ako sa ukáže, je kľúčom k riešeniu. Stačí ak si napíšeme nerovnicu a umocníme na 6-tu, a všetko nám už bude jasné:

Ak a, b, c sú ľubovoľné prirodzené čísla a platí: $\sqrt[3]{a} < \sqrt{b} < c$ tak aj po umocnení na 6 musí platiť $\sqrt[3]{a}^6 < \sqrt{b}^6 < c^6 \Leftrightarrow a^2 < b^3 < c^6$. Naopak ak po umocnení platí $\sqrt[3]{a}^6 < \sqrt{b}^6 < c^6$ tak pred umocnením platilo $\sqrt[3]{a} < \sqrt{b} < c$

Tým sme sa zbavili odmocnín, nutno však poznamenať, že x^6 je nepríjemne rýchlo rastúca funkcia a preto by sme ľahko mohli vyčerpať rozsah dátových typov, na čo treba pamätať (ideálne by bolo spraviť si vlastnú aritmetiku) ale zapodievať sa tým nebudeme.

Problém #2: Triedenie. Ukážeme si dva rôzne rýchle algoritmy na triedenie:

Bubblesort: Uvádzame ho tu len pre jeho jednoduchosť, pretože jeho implementácia je dvojriadková (keď ste leniví alebo netreba rýchle triedenie). Už názov napovedá ako funguje „bublinkové“ triedenie. Prechádzame pole a kontrolujeme susedné prvky či spĺňajú kritérium nerovnosti, ak nie, vymeníme ich, z toho je zrejme že najväčší prvok „prebubleme“ na koniec. Ak tento postup aplikujeme N krát, je isté, že dostaneme utriedenú postupnosť. Časová zložitosť je $O(N^2)$, keďže N-krát prejdeme N-prvkové pole. Čo sa pamäte týka, vystačíme si s jedným jednorozmerným poľom, čo znamená, že pamäťová zložitosť je $O(N)$, kde N je počet prvkov. A je pole o N prvkoch indexovaných od 0 po $N-1$.

```
for i:=1 to N do
    for j:=1 to N-i do if A[j-1]>A[j] then vymen(A[j-1],A[j]);
```

```
for(int i=0;i<N;i++)
   for(int j=1;j<N-i;j++) if(A[j-1]>A[j]) swap(A[j-1],A[j]);
```

Quicksort: Quicksort sa zakladá na princípe rozdeľuj a panuj (rozdeľ ťažký problém na viacero malých, ktoré však už vieš ľahko riešiť). Proces triedenia pre pole $A[l \dots r]$ pozostáva z troch krokov:

- Rozdeľ: Pole $A[l \dots r]$ je prerozdelené do dvoch neprázdnych podpolí $A[l \dots q]$ a $A[q+1 \dots r]$ tak, že každý prvok z $A[l \dots q]$ je menší alebo rovný ako ľubovoľný prvok $A[q+1 \dots r]$. Index q je vypočítaný počas tejto procedúry. Rozdelíme to tak, že si vyberieme jeden reprezentatívny prvok (nazveme ho *pivot*) a rozdelíme pole tak, aby v ľavej časti poľa boli prvky menšie alebo rovné ako *pivot* a v pravej časti aby boli prvky väčšie alebo rovné ako *pivot*. Ako vybrať *pivota*? V istej miere je nám to jedno, ale ideálne by sme chceli, aby *pivot* bol medián (prvok, čo je po utriedení poľa v strede). Toto však nie je jednoduché zaručiť, preto my budeme vyberať pivota zo stredu postupnosti[8].
- Panuj: Dve podpolia $A[l \dots q]$ a $A[q+1 \dots r]$ sú utriedené rekurzívnymi volaniami procedúry Quicksort. To znamená, že vďaka rekurzii sa nám počiatočné pole rozdelí na mnoho jednoprvkových podpolí, ktoré sú sami o sebe usporiadané (toto nazývame triviálny prípad). V ideálnom prípade sa procedúra Quicksort volá na obe polovice počiatočného poľa, potom na štvrtiny, šestnástiny, dvaatridsatiny... Spôsob, akým ich vytvárame nám zaručuje, že aj počiatočné pole bude utriedené.
- Skombinuj: Keďže podpolia sú utriedené na mieste, netreba už nič robiť, celé pole $A[l \dots r]$ je teraz utriedené.

Odporúčame si tento algoritmus ručne demonštrovať na papier, alebo navštíviť stránku na wikipédii `http://cs.wikipedia.org/wiki/Quicksort` kde sa nachádza nápomocná animácia (aj implementácia). Implementáciu v pascale uvádzame aj vo vzorovom riešení.

Analyzujme teraz časovú zložitosť ideálneho prípadu, keď sa pole delí na polovičky (za pivota vyberáme medián). Predstavme si, že máme úsečku a na nej je rovnomerne nanesených N bodov (to je naša paralela s N prvkovým poľom). Následne rozdelíme túto úsečku v

[8] Existuje aj znáhodnelá verzia Quiksortu, ktorá za pivota vyberie náhodný prvok

bode, ktorý je v jej polovici na dve polovičné úsečky. V druhom kroku tieto polovice opäť rozdelíme v strednom bode na štvrtiny pôvodnej úsečky, v treťom kroku opäť rozdelíme štvrtiny na polovice (vzniknú šestnástiny pôvodnej úsečky) a takto pokračujeme, až máme N nedeliteľných úsečiek. Otázka teda znie, ako dlho takto môžeme deliť, resp. koľko krát musíme deliť N dvojkou, aby sme dostali 1? Táto otázka je ekvivalentná s otázkou: „Dva na koľkú je N"? A na to je zrejmá odpoveď: dvojkový logaritmus z N. Ako dlho trvá jedno rozdelenie na polovice? Každú časť musíme rozdeliť na polovice podľa jej mediána, čiže musíme naľavo poupratovať menšie prvky a napravo väčšie prvky, počet všetkých prvkov vo všetkých častiach je N, takže čas rozdelenia je $O(N)$. Ako sme si neformálne dokázali o niečo vyššie, týchto rozdelení je $O(\log N)$ a keďže každé trvá $O(N)$ a výsledná časová zložitosť je počet rozdelení krát časová náročnosť rozdelenia, v ideálnom prípade je to teda $O(N \log N)$.

Analyzujme teraz časovú zložitosť v najhoršom prípade, a to keď sa pole rozdelí na jeden prvok a na všetky ostatné prvky (vtedy sme vybrali veľmi zlého pivota). Podobnou úvahou s úsečkou ako v predchádzajúcom odstavci vieme ukázať, počet rozdelení je až $O(N)$. Čas potrebný na jedno rozdelenie sa nám nezmenil, takže časová zložitosť v najhoršom prípade je $O(N \cdot N)$, teda $O(N^2)$. Našťastie takýchto prípadov je vskutku málo.

Časová zložitosť quicksortu v očakávanom[9] prípade je tiež $O(N \log N)$, dôkaz už kôli jeho zložitosti neuvádzame. Mnohí z vás sa teraz iste pýtajú, prečo je quicksort považovaný za jedno z najrýchlejších triedení, keď niekedy môže čas jeho výpočtu byť až kvadratický, pritom heapsort alebo mergesort sú stále $O(N \log N)$. Dôvodom sú konštanty nezávislé od vstupu, ktoré nemenia rádovú zložitosť, čiže ak quicksort je v priemernom prípade $k \cdot (N \log N)$ a mergesort je $l \cdot (N \log N)$ tak $k < l$.

Pamäťová zložitosť je vždy rovnaká – lineárne závislá od N, čiže $O(N)$, kde N je počet prvkov.

Takže už vieme všetko čo potrebujeme k vyriešeniu našej úlohy. Nezabudnime ešte, že ak pri porovnávaní dvoch odvarov dôjde k zhode ich absolútnej mocnosti, rozsúdime ich podľa odmocnenosti, čo nám len mierne skomplikujem porovnávanie, keďže pribudne jedna podmienka v prípade rovnosti absolútnej mocnosti.

K bodovaniu:

- 15 bodov $O(N \log N)$ riešenia
- 12 bodov $O(N^2)$ riešenia
- za použitie reálnych čísel bola odmena -6 bodov
- po -1 bode sme strhávali za absenciu popisu/odhadu časovej zložitosti/odhadu pamäťovej zložitosti
- za nefunkčné riešenie bolo maximálne 6 bodov
- a individuálne

Riešenia s inou ako lineárnou pamäťovou zložitosťou neprišli, tak sa o nich vo vzorovom riešení ani nezmieňujeme.

Vzorové riešenie, ktoré má časovú zložitosť $O(N \log N)$ a pamäťovú zložitosť $O(N)$, sme implementovali v jazyku Pascal, kde si môžete pozrieť aj implementáciu quicksortu a vyššie zmienenú porovnávaciu funkciu.

Listing programu:

```
const MAX=1000;
type Todvar=record M,O,ABS:int64; end;
var odvary:array [1..MAX] of Todvar;
    i,N:integer;

function umocni(zaklad:int64; exponent:integer):int64; {jednoduche umocnovanie}
```

[9] Očakávaný prípad znamená, ako dlho to bude trvať na priemernom vstupe.

```
  var i:integer;
      vysledok:int64;
  begin
    vysledok:=zaklad;
    for i:=1 to exponent-1 do vysledok:=vysledok*zaklad;
    result:=vysledok;
  end;

function ostro_vacsie(a,b:Todvar):boolean; {vrati true ak (a > b)}
  begin
    if (a.ABS>b.ABS) or ((a.ABS=b.ABS) and (a.O<b.O)) then result:=true
    else result:=false;
  end;

procedure qsort(left,right:integer);
var i,j,pivot:integer;
    temp:Todvar;
begin
  i:=left;
  j:=right;
  { volba pivota je v podstate lubovolna, idealne by bolo keby bol median,
    my si tipneme prvok v strede triedeniej casti pola }
  pivot:=(left + right) div 2;
  repeat
    { v lavej casti pola najde prvok vacsi/rovny ako pivot }
    while ostro_vacsie(odvary[pivot], odvary[i]) do inc(i);
    { v pravej casti pola najde prvok mensi/rovny ako pivot }
    while ostro_vacsie(odvary[j], odvary[pivot]) do dec(j);
    { ak je naozaj vacsi/rovny nalavo od mensieho, vymen ich }
    if (i<=j) then
    begin
      temp:=odvary[i];
      odvary[i]:=odvary[j];
      odvary[j]:=temp;
      inc(i);
      dec(j);
    end;
  until i > j;
  { rekurzivne volanie }
  if left < j then qsort(left,j);
  if i < right then qsort(i,right);
end;

begin
    readln(N);
    for i:=1 to N do
        begin
            readln(odvary[i].M,odvary[i].O);
            case (odvary[i].O) of
                    0:   odvary[i].ABS:=umocni(odvary[i].M,6);
                    1:   odvary[i].ABS:=umocni(odvary[i].M,3);
                    2:   odvary[i].ABS:=umocni(odvary[i].M,2);
            end;
         end;
    qsort(1,N);
    for i:=1 to N do writeln(odvary[i].M,' ',odvary[i].O);
end.
```

..... Tu sa končia riešenia KSP-Z .....

Opravoval Maty
(max. 15 bodov)

6. Ohrozená posádka

Na začiatok vzoráku by sme sa vám chceli ospravedlniť za nie moc presne formálne popísanie zadanie. Napríklad tam chýbal predpoklad, že sa na začiatku loď a asteroid nepretínajú. S tým však nebol problém, lebo niektorí z vás si to uvedomili a pridali si to do predpokladov a iní sa tým vôbec nezaoberali. Ďalším hlavným problémom bolo to, že sa loď a asteroid mohli zraziť v minulosti. Teda ak by mala loď opačnú rýchlosť, tak by sa zrazili. Na to už mnohí z vás zabudli a stratili za to 3 body. Vaše riešenia mali buď lineárnu $\Theta(M+N)$ alebo kvadratickú $\Theta(M^2+N^2)$ zložitosť. Lineárne mohli dostať 15 bodov, kvadratické 12. Za neošetrenie niektorých špeciálnych prípadov polohy asteroidu a lode som strhával 1–2 body.

A teraz k vzorovému riešeniu. Na začiatku predpokladáme, že sa asteroid a loď nepretínajú. Ak si predstavíme, ako letí loď, tak si všimneme, že vytvorí v rovine pás (teda celý pás by vytvorila, keby sa hýbala aj v opačnom smere). Tento pás vieme reprezentovať pomocou jeho hraničných priamok a tie sú jednoznačne určené bodmi loďe, ktoré sú v istom zmysle najviac na krajoch. Najviac na krajoch znamená že majú najväčšiu, resp. najmenšiu (orientovanú) vzdialenosť od priamky tvorenej bodom $(0,0)$ a vektorom v, pričom vpravo je kladná vzdialenosť a vľavo záporná. Prístupov ako nájsť tieto 2 body je viacero. Prvým je vyjadriť vzorec pre vzdialenosť bodu od priamky určenej bodom $(0,0)$ a vektorom v. Šikovnejšie je však celú situáciu otočiť tak, aby náš pás bol rovnobežný s x-ovu osou. Otáčanie okolo bodu $(0,0)$ o uhol α proti smeru hodinových ručičiek robíme na základe vzorca

$$(x,y) \mapsto (x\cos\alpha + y\sin\alpha, -x\sin\alpha + y\cos\alpha).$$

K rátaniu tohto vzorca potrebujeme poznať $\sin\alpha$ a $\cos\alpha$. Tie zrátame pomocou vektora pohybu lode (x,y) a nasledujúcich vzťahov:

$$d = \sqrt{x^2+y^2} \qquad \sin\alpha = y/d \qquad \cos\alpha = x/d.$$

Takto si môžeme otočiť všetky body loďe a asteroidu.

Celá situácia hľadania krajných bodov sa nám teraz podstatne zjednodušila. Vieme, že vektor pohybu je teraz násobok $(1,0)$ a teda hraničné priamky pásu nám určujú už iba y-ové súradnice bodov lode, konkrétne bod s maximálnou a minimálnou y-ovou súradnicou. Aby asteroidu nehrozila zrážka s loďou, musia byť všetky jeho body nad alebo pod naším pásom. Teda ak nájdeme minimálnu a maximálnu y-ovú súradnicu asteroidu a lode, musí platiť, že[10] $lmin.y > amax.y$ alebo $lmax.y < amin.y$. Vtedy sa loď určite asteroidu vyhne.

V opačnom prípade platí, že asteroid leží v páse tvorenom loďou. Posledná vec, ktorú musíme overiť je, či sa asteroid nachádza pred alebo za loďou. Ak vylúčime možnosť, že asteroid a loď už majú spoločný prienik, potom stačí zobrať jeden bod asteroidu, ktorý je v páse a pozrieť sa, či je vľavo alebo vpravo od lode. Loď totiž môže nabúrať len do bodov v páse a vďaka konvexnosti asteroidu sa nemôže stať, že jeden bod je vľavo a iný vpravo od lode (podľa definície konvexnosti by musel obsahovať aj všetky body medzi nimi a mal by teda prienik s loďou, čo je spor s našim predpokladom). Jeden bod v páse teda dokáže rozhodnúť.

Ako ho nájdeme? Buď je to *amin*, alebo *amax*, alebo si ho musíme dopočítať. V tomto treťom prípade musí platiť, že $amax.y > lmax.y$ a $amin.y < lmin.y$. Keďže je však asteroid konvexný, úsečka tvorená bodmi *amax* a *amin* leží celá v asteroide a má neprázdny

[10] Poznámka k označeniu: l znamena loď, a znamená asteroid, max znamená najvyšší boj, a min znamená najnižší bod.

prienik s naším pásom. Stačí teda bodmi $amin$, $amax$ preložiť priamku a spočítať prienik s nejakou priamkou nášho pásu. Všeobecná rovnica priamky je $ax + by + c = 0$, kde (a, b) je normálový vektor priamky a teda $(-b, a)$ je smerový vektor priamky, ktorý dostaneme ako rozdiel $(-b, a) = (amax.x - amin.x, amax.y - amin.y)$. Posledný parameter z rovnice priamky, ktorý potrebujeme dopočítať je c a ten získame tak, že dosadíme za (x, y) jeden z bodov $amin$, $amax$. Keď už máme všeobecnú rovnicu priamky, stačí nám dosadiť napríklad $y = lmin.y$ a dostaneme bod, ktorý leží na našej priamke a zároveň v páse (konkrétne na spodnej hranici).

Ostáva zistiť, či je bod asteroidu vpravo alebo vľavo od lode. Dokonca stačí zistiť, či je vpravo alebo vľavo od úsečky tvorenej bodmi $lmax$, $lmin$ (táto úsečka vytvára rovnaký pás ako celá loď) – to si môžeme dovoliť opäť vďaka konvexnosti lode a predpokladu, že nemá s asteroidom prienik.

Zistiť to vieme dvoma spôsobmi: Jeden je, že vypočítame bod na úsečke $lmin$–$lmax$ s rovnakou y-ovou súradnicou, ako má bod asteroidu a porovnáme ich x-ové súradnice. Druhý je, že vypočítame orientovanú veľkosť vektorového súčinu a rozhodneme sa podľa znamienka. Nech $v_1 = lmax - lmin$ je vektor reprezentujúci loď a aa nech je bod asteroidu. Nech $v_2 = aa - lmin$. Potom ak $v_1 \times v_2 < 0$, tak v_2, resp. aa je vpravo od v_1, ak $v_1 \times v_2 > 0$, tak v_2, resp. aa je vľavo od v_1. Orientovanú veľkosť vektorového súčinu vypočítame ako $v1 \times v2 = v1.x \cdot v2.y - v1.y \cdot v2.x$.

Časová aj pamäťová zložitosť tohto riešenia je $\Theta(N + M)$.

Listing programu:

```
#include <cstdio>
#include <cstdlib>
#include <cmath>
using namespace std;
#define MAX 1000
#define REP(i,n) for (int i=0; i<(n); ++i)
#define VEDLA { printf ("Uff, to bolo tesné!\n"); return 0; }

struct Point { double x,y; };
int N, M;
double d, sina, cosa;
Point l[MAX], a[MAX], v;

Point otoc (Point p) {
  Point r;
  r.x = p.x*cosa + p.y*sina;
  r.y = -p.x*sina + p.y*cosa;
  return r;
}

double jevpravo (Point a, Point b, Point c) {
  Point v1, v2;
  v1.x = b.x - a.x; v1.y = b.y - a.y;
  v2.x = c.x - a.x; v2.y = c.y - a.y;
  return v1.x*v2.y - v1.y*v2.x;
}

int main() {
  scanf ("%d", &N); REP(i,N) scanf("%lf %lf", &l[i].x, &l[i].y);
  scanf ("%d", &M); REP(i,M) scanf("%lf %lf", &a[i].x, &a[i].y);
  scanf ("%lf %lf", &v.x, &v.y);
  if ((v.x==0) && (v.y==0)) VEDLA;

  d = sqrt (v.y*v.y + v.x*v.x);
```

```
  sina = v.y/d; cosa = v.x/d;
  REP(i,N) l[i] = otoc(l[i]);
  REP(i,M) a[i] = otoc(a[i]);
  v = otoc(v);

  Point lmin=l[0], lmax=l[0], amin=a[0], amax=a[0];
  REP(i,N) { if (lmax.y < l[i].y) lmax=l[i]; if (lmin.y > l[i].y) lmin=l[i]; }
  REP(i,M) { if (amax.y < a[i].y) amax=a[i]; if (amin.y > a[i].y) amin=a[i]; }
  if ((amin.y > lmax.y) || (amax.y < lmin.y)) VEDLA;

  Point aa;
  double a, b, c, x;
  if (amax.y <= lmax.y) aa = amax;
  else if (amin.y >= lmin.y) aa = amin;
  else { //bod asteroidu musime doratat
    a = amax.y - amin.y;
    b = amin.x - amax.x;
    c = -a*amin.x-b*amin.y; //c z vseobecnej rovnice priamky priemeru
    aa.x = (-b*lmin.y-c)/a; // bod asteroidu a zaroven v poli lode
    aa.y = lmin.x;
  }
  if (jevpravo (lmin,lmax,aa)<0) printf("Je to hotová samovražda!\n"); else VEDLA;
  return 0;
}
```

7. Ošemetná postupnosť

opravoval Mišo
(max. 15 bodov)

Ako prvé si treba uvedomiť, že postupnosť sa začína nulou a každý člen sa môže zvýšiť maximálne o 1 a minimálne o -1, takže súčet je niekde medzi $-N(N-1)/2$ a $N(N-1)/2$ (vďaka vzorcu na súčet aritmetickej postupnosti[11]).

Keby sme každý ďalší člen postupnosti zvyšovali o 1, celkový súčet by bol $N(N-1)/2$. Keď máme už nejakú postupnosť a zmeníme rozdiel medzi dvomi susednými, teda medzi i-tym a $i+1$-vým z 1 na -1, zmení sa každý člen počnúc $i+1$-vým a končiac N-tým. Presnejšie: každý bude menší o 2. Takže celkové zmenšenie je $2(N-i)$. Kedže máme daný súčet S, vieme, že celkovo máme súčet zmenšiť o $N(N-1)/2-S$ oproti čisto rastúcej postupnosti. Vidno, že všetky zmeny, ktoré sme schopní spraviť, sú párne. Z toho vyplýva, že $N(N-1)/2-S$ musí byť párne. V opačnom prípade úloha nemá riešenie. Vieme ale spraviť ľubovoľné S spĺňajúce predchádzajúce podmienky? Odpoveď je áno, a aby ste mi nemuseli slepo veriť, tak si to aj dokážeme.

Nech $K = N(N-1)/2 - S$. Vieme, že K je párne[12]. Ďalej vieme, že K je najviac $N(N-1)$. To, čo vieme, je znížiť súčet postupnosti najviac raz o 2, najviac raz o 4, ..., najviac raz o $2(N-1)$. Teda chceme vyskladať K z čísiel $2, 4, \cdots, 2(N-1)$ pričom každé použijeme najviac raz. Iste ste si všimli, že všetky čísla sú párne a preto ich môžeme vydeliť dvojkou. Takže teraz sa pýtame, či dokážeme z čísiel $1, 2, \cdots, N-1$ vyskladať každé číslo z $0, 1, \cdots, N(N-1)/2$. Toto vieme a dokážeme si to indukciou.

1. Ak $N = 0$ alebo $N = 1$, tak je to triviálne. Ak $N = 2$, tak máme na výber číslo 1 a potrebujeme vyskladať sumy 0 a 1, čo prirodzene vieme.
2. Teraz predpokladáme, že to vieme vyskladať pre N. Podľa indukčného predpokladu z čísiel $1, 2, \cdots, N-1$ vieme vyskladať všetky čísla od 0 po $N(N-1)/2$. Platí tvrdenie

[11] Pre neznalých, súčet čísiel od 1 po N je $N(N+1)/2$

[12] Ako sme si ukázali vyššie, pre nepárne sa to nedá

pre $N+1$? Pribudlo nám číslo N, ktoré môžeme použiť, a chceme navyše vyskladať čísla $N(N-1)/2+1, \cdots, N(N+1)/2$ (menšie vyskladať vieme). Keď ale zoberieme rozdiel $N(N+1)/2 - N(N-1)/2 - 1 = N-1$, tak zistíme, že keď použijeme číslo N, tak dostaneme číslo z intervalu $< 0, N(N-1)/2 >$, ale to vieme podľa indukčného predpokladu vyskladať z čísiel $1, \cdots, N-1$.

Teraz sme si pekne dokázali, že to vieme, tak sa vrhnime na samotný algoritmus.

V premennej `rozdiel` si pamätáme, o koľko sa nám líši súčet momentálnej postupnosti a S. Je to vždy párne číslo, ktoré je na začiatku rovné $N(N-1)/2 - S$, keďže na začiatku uvažujeme postupnosť, ktorá stále rastie. Pôjdeme postupne a niektoré členy budeme meniť. Prvý člen zmeníme, ak je `rozdiel` väčší alebo rovný $2(N-1)$. Zároveň rozdiel zmenšíme o $2(N-1)$. Vo všeobecnosti ak po i krokoch je `rozdiel` väčší ako $2(N-i)$, zmeníme rozdiel i-teho a $i+1$-vého člena a premennú `rozdiel` rozdiel zmenšíme o $2(N-i)$.

Celý program bude pozostávať z jediného cyklu, ktorý bude od začiatku prechádzať a prípadne meniť postupnosť. Zároveň však, keďže vieme rozdiel i-tého a $i+1$-vého člena, vieme členy priebežne rátať a vypisovať ich, teda vystačíme s konštantnou pamäťou. Časová zložitosť je lineárna.

Bodovanie: Za riešenie v čase $O(N)$ a s pamäťou $O(1)$ bolo 13 bodov, s lineárnou pamäťou za 12. Za čas $O(N^3)$ 10, za čas $O(2^N)$ som dával len 8 bodov. Za správne odhady zložitostí bol bonus 2 body[13].

Listing programu:

```
#include<cstdio>
using namespace std;

int main() {
  int N, S, rozdiel;
  scanf("%d %d",&N,&S);
  rozdiel = N*(N-1)/2 - S;
  if (rozdiel<0 || rozdiel>N*(N-1) || rozdiel%2==1)
    printf("Riesenie neexistuje\n");
  else {
    int last=0;
    printf("0 ");
    for (int i=1; i<N; i++) {
      if(rozdiel>=2*(N-i)) { --last; rozdiel -= 2*(N-i); }
      else ++last;
      printf("%d ",last);
    }
    printf("\n");
  }
  return 0;
}
```

..... Tu sa začínajú riešenia KSP-T .....

8. O nekompromisnom Maroškovi

Opravoval Zemčo
(max. 15 bodov)

Vitajte vo vzoráku, ktorý dodrží štandardnú vnútornú kompozíciu drámy, pretože bude veľmi dramatický.

[13] takže predsa sa tých 15 bodov získať dalo

Úvod (expozícia): Len jedno vzorové riešenie! Len jedno číslo 15 za tento príklad vo výsledkovke! Musíte sa viac snažiť! Pri zbežnom prezretí riešení sa síce vyskytlo viacero so vzorovým časom, tajomstvo vášho úspechu však spočívalo v zlom časovom odhade. Ale o tom potom. Poďme si rozobrať, čo ste robili a ako sa to líši od toho, čo ste mali robiť.

Zápletka (kolízia): Ako prvá každého iste napadla nenápaditá priamočiara simulácia postupu. V každom momente skontrolujeme vzdialenosti medzi každými dvoma bodmi a vyberieme, ktoré zlúčime. Vzdialenosti porátame v kvadratickom čase od počtu bodov, pričom zlučujeme $(N-1)$-krát. Výsledný čas tak bude $O(N^3)$.

O rád rýchlejšie riešenie si už vyžadovalo určitý programátorský kumšt. Privedie nás k nemu otázka „Nedá sa troška šikovnejšie vyberať najbližšiu dvojicu v každom okamihu?“ Keď si vezmeme dve po sebe idúce zlučovania a zamyslíme sa, aké všetky vzdialenosti kontrolujeme, uvedomíme si, že robíme strašne veľa zbytočnej práce, pretože väčšina bodov ostáva na mieste. To naznačuje veľký potenciál na zlepšenie. Použijeme haldu, do ktorej nahádžeme na začiatku všetky dvojice zariadení. Netreba zabudnúť, čo robiť, ak sa nám vzdialenosti rovnajú a podľa toho upraviť porovnávanie v halde. Na toto ste ale pamätali všetci. V halde máme N^2 prvkov, ale výber a vkladanie nás stojí stále $O(\log N)$ času[14]. Vyberieme najbližšiu dvojicu a zlúčime, čím dostaneme nový bod. Teraz do haldy vložíme všetky dvojice nového bodu s tými, čo ostali. Niekoho možno zarazí, že v halde ostali aj neplatné dvojice – také, ktorých aspoň jedna zložka je už neexistujúci bod. Tie nás ale trápiť nebudú. V každom ďalšom kroku budeme vyberať dovtedy, kým nevyberieme platnú dvojicu. Čas nám to nepokazí, pretože v halde bude v každom momente stále len $O(N^2)$ prvkov. Na začiatku spravíme rádovo N^2 vložení a potom každým krokom rádovo N vložení, pričom výberov nebude viac ako $O(N^2)$, keďže viac prvkov v halde nemôže byť. Takže dostávame čas $O(N^2 \log N)$.

Vyvrcholenie deja (kríza): Vzorové riešenie je až na kľúčový trik vcelku jednoduché. Je založené opäť na simulácii. V každom momente budeme zlučovať dva body na jeden. Na začiatku porátame ku každému bodu najbližší a to si zapamätáme. Nebudeme si pamätať informáciu o každej dvojici, pretože to nepotrebujeme a ušetríme pamäť. Následne v lineárnom čase nájdeme dvojicu, ktorú ideme rušiť. Teraz vypočítame súradnice nového bodu. Jeden zo starých nahradíme novým, druhý označíme za neaktívny a v ďalšom postupe ho budeme ignorovať. Údaje v poli o najbližších bodoch teraz môžu byť neplatné a aby sme ich dostali opäť do konzistentného stavu, musíme riešiť nasledovné prípady:

1. Dorátať najbližší bod k novému bodu.
2. Skontrolovať, či nový bod nebude nabližší nejakému bodu.
3. Nájsť nové najbližšie body k tým, ktoré mali ako najbližší bod jeden zo zrušených.

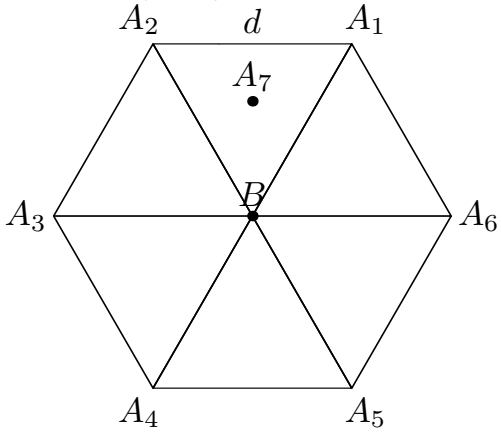
Nájsť najbližší bod k nejakému vieme v lineárnom čase od počtu aktívnych bodov. S prvým bodom teda nemáme problém. Rovnako aj druhý bod stíhame lineárne. Ten tretí bude zaujímavejší. Jeden by povedal, že bodov, ktorých sa to týka môže byť veľa a ak pre každý stratíme lineárny čas, môže sa nám jedno zlučovanie zhoršiť na kvadratický a celý algoritmus na kubický čas. Ukážem však, že počet bodov, ktorým bol zrušený bod najbližší sa bude dať zhora ohraničiť nejakou konštantou.

Vyvrcholenie vyvrcholenia deja (kríza krízy): Tvrdenie dokážeme pre 6 bodov. Predstavme si ľubovoľný bod B a okolo neho 6 bodov $A_1 \dots A_6$, pre ktoré platí, že B je najbližší bod ku každému z nich. Pre jednoduchosť situácie môžeme zabudnúť na sekundárne kritéria pri rovnosti vzdialeností a predpokladajme, že sa rozhodujeme náhodne. Dôkazu to neublíži. Body $A_1 \dots A_6$ budú rozložené niekde okolo bodu B a ked po nich prejdeme v smere hodinových ručičiek a pospájame ich, tak dostaneme 6-uholník, ktorý musí byť pravidelný. To vyplýva z toho, že oproti najväčšej strane je v trojuholníku najväčší uhol. Keď si vezmeme každý zo šiestich trojuholníkov tvoriaci náš šesťuholník, musí byť vzdialenosť medzi bodmi A_i a A_j aspoň taká ako vzdialenosti A_i od B a A_j od B. To znamená, že uhol pri B musí

[14] pretože $O(\log N^2) = O(2 \log N) = O(\log N)$.

byť aspoň taký ako hociktorý z tých zvyšných dvoch a z toho vyplýva, že nie je menší ako 60 stupňov. Keďže ale toto musí platiť pre všetkých 6 trojuholníkov, bude to práve 60. Takže trojuholníky budú rovnostranné a celý 6-uholník pravidelný. Dĺžku jeho strany označíme d.

Skúsme teraz niekam umiestniť bod A_7, pre ktorý by platilo, že najbližší k nemu je B a zároveň aby B ostalo najbližším aj pre všetky A_i. A_7 nemôže ležať mimo 6-uholníka, pretože by zjavne malo bližšie k nejakému bodu A_i. Keby sme ale umiestnili A_7 do vnútra alebo na okraj nejakého z trojuholníkov, potom by bod B určite prestal byť najbližší k bodom, ktoré tvorili tento trojuholník, pretože by sa ním stal práve A_7. Takže A_7 nie sme schopní umiestniť do situácie tak, aby každý z bodov $A_1 \dots A_7$ mal ako najbližší bod B. Bodov, ktoré majú ako najbližší bod B tak môže byť najviac 6.



Obrat deja (peripetia): Bodov, ktorým musíme hľadať nový najbližší bod je teda len konštante veľa. Dokopy strávime riešením ich všetkých len lineárny čas, takže každé zlučovanie nás stojí lineárny čas a celý postup bude bežať v čase $O(N^2)$ a s pamäťou $O(N)$.

Záver (ozajstná katastrofa): Inými slovami – ako som hodnotil vaše snaženie. Maximum bolo 15 bodov, za riešenie s haldou som dával najviac 12 a za kubické riešenie 8 bodov. Štandardne som strhol polovicu bodov za absenciu kódu a bod za zlé odhady, ktorých bolo tentokrát prekvapujúco veľa. Presnejšie: mnohí s riešením za 8 bodov si ho odhadli ako riešenie za 15 bodov, ale nevyšlo im to a skončili len so siedmimi bodmi.

Po hre nasleduje vysvetlenie od kritika Mišofa: V geometrii platí nepísaná zásada: „Všetko ide spočítať v čase $O(N \log N)$." Tak je to aj v tejto úlohe. Existujú dátové štruktúry, ktoré vedia v logaritmickom čase robiť každú z operácií „vlož bod", „vyber bod" a „nájdi dva najbližšie body". Jeden možný prístup je založený na tom, že keď máme množinu bodov, vieme ju predspracovať tak, že nájdeme vzdialenosť δ takú, aby existovala podmnožina bodov, ktorá obsahuje približne polovicu bodov a platí, že každé dva body z tejto podmnožiny majú vzdialenosť aspoň δ. Podobne si vybudujeme jemnejšie delenie zvyšných bodov. S takto predspracovanými bodmi sa dá efektívne spraviť celkom veľa operácií vloženia a výberu bez toho, aby sa nám predrátané informácie príliš pokazili. Keď sa nám už pokazia, tak si opäť všetko pekne prerátame odznova a veselo pokračujeme ďalej.

Detaily sú príliš komplikované na to, aby sme ich sem písali. Kto si chce pozrieť, ako takáto hrôza vyzerá, dočasne nájde na adrese `https://foja.dcs.fmph.uniba.sk/~misof/clanok.pdf` 37-stranový článok, v ktorom to je vysvetlené poriadne. Odporúčame len prelistovať, radšej sa to nesnažte pochopiť.

Listing programu:

```
#include <cstdio>
#include <cmath>
#define MAXN 1000
#define VELA 999999
#define EPS 1e-7
```

```
struct TPoint{double x,y;};

TPoint body[MAXN];
int je[MAXN],skym[MAXN],N;
double najblizsi[MAXN];

double dist(TPoint a, TPoint b){
    return sqrt( (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y) );
}

void najdinajblizsi(int k){
    najblizsi[k] = VELA;
    for (int i=0;i<N;i++)if (je[i]==1 && i!=k){
        if (  fabs(najblizsi[k] - dist(body[k],body[i])) < EPS ){
            //ak nerozhodne vzdialenost, musia suradnice
            if ( fabs(body[i].x - body[skym[k]].x) < EPS){
                if (body[i].y < body[skym[k]].y){
                    skym[k]=i;
                }
            }
            else if (body[i].x < body[skym[k]].x){
                skym[k] = i;
            }
        }
        else if ( najblizsi[k] > dist(body[i],body[k])){
            najblizsi[k] = dist(body[i],body[k]);
            skym[k]=i;
        }
    }
    return;
}

int main(){
    scanf("%d ",&N);
    for (int i=0;i<N;i++){
        scanf("%lf %lf ",&body[i].x,&body[i].y);
        je[i]=1;
    }
    for (int i=0;i<N;i++){//pohladame pre kazdy bod najblizsi
        najdinajblizsi(i);
    }
    for (int j=0;j<N-1;j++){
        int ktory = -1, druhy = -1;
        double min = VELA;
        for (int i=0;i<N;i++)if (je[i]==1){//najdeme najblizsiu dvojicu
            if (najblizsi[i] < min){ktory = i; min = najblizsi[i]; druhy = skym[i];}
        }
        //zrusime
        double nx = (body[ktory].x+body[druhy].x)/2.0;
        double ny = (body[ktory].y+body[druhy].y)/2.0;
        je[druhy]=0;
        body[ktory].x = nx; body[ktory].y = ny;
        //najdeme najblizsi pre novy bod
        najdinajblizsi(ktory);
        //pobehame, ci novy bod nieje nejakemu bodu najblizsi
        for (int i=0;i<N;i++)if (i!=ktory)if(je[i]==1){
            if (fabs(dist(body[ktory],body[i]) - najblizsi[i]) < EPS){
                if (  fabs( body[skym[i]].x - body[ktory].x) < EPS ){
                    if (body[skym[i]].y > body[ktory].y){skym[i] = ktory;}
                }
```

```
                else{
                    if (body[skym[i]].x > body[ktory].x){skym[i] = ktory;}
                }
            }
            else if (dist(body[ktory],body[i]) > najblizsi[i]){
                najblizsi[i] = dist(body[ktory],body[i]); skym[i]=ktory;
            }
        }
        //pobehame, ci niekde neostal stary bod uvedeny ako najblizsi
        for (int i=0;i<N;i++)if (i!=ktory && i!=druhy && je[i]==1){
            if (skym[i]==ktory || skym[i]==druhy){
                //toto moze nastat len konstantny pocet krat
                najdinajblizsi(i);
            }
        }
    }
    for (int i=0;i<N;i++)if (je[i]==1){
        printf("Vysledne zariadenie bude v bode [%.3lf,%.3lf].\n",body[i].x,body[i].y);
    }
    return 0;
}
```

..... Tu sa končia riešenia KSP-O .....

Opravoval Mic
(max. 15 bodov)

9. Trápenie kráľa Artuša

Ach jaj:-(. Veľmi ste ma nepotešili, ba priam naopak. Dostal som iba 7 riešení. A z toho fungovalo iba jedno. Ale aspoň bolo vzorové. To dostalo 15 bodov. Ostatné riešenia dostali tak 3 body. O čo boli slabšie riešenia, o to treba lepší vzorák:-)

Väčšina z vás si všimla, že na vstupe ste dostali graf. Vrcholmi boli rytieri. A hrana medzi rytiermi bola práve vtedy keď sa nemajú radi. Ale s takýmto grafom sa nám robí veľmi ťažko, preto si ho transformujme na taký graf, že hrana medzi rytiermi vedie práve vtedy, keď sa dvaja rytieri majú radi. To znamená, že v našom novom grafe bude viesť hrana medzi vrcholmi a a b práve vtedy, keď v pôvodnom grafe vrcholy a a b neboli spojené hranou (takýto graf sa nazýva **komplement grafu**). Odteraz automaticky budeme pracovať s **komplementom** grafu. Našou úlohou je nájsť vrcholy, ktoré sa nedajú umiestniť na žiadnu kružnicu nepárnej dĺžky, ktorá má aspoň 3 vrcholy. Pozrime sa najskôr na prvú podmienku, teda na kružnice. S tým nám pomôže nasledovný pojem: **dvojsúvislý komponent**.

Čo to je? Dá sa to jesť? Dvojsúvislý komponent je taká množina vrcholov K, že ľubovoľné dva vrcholy z K ležia na kružnici (pričom celá kružnica leží v K). Dá sa to povedať aj tak, že medzi každými dvoma vrcholmi existujú dve cesty také, že okrem počiatočného a cieľového vrcholu nemajú žiadne vrcholy spoločné[15]. Zároveň sa K nedá pridaním vhodných vrcholov zväčšiť To znamená, že keď zoberieme dva vrcholy, ktoré neležia v tom istom dvojsúvislom komponente, tak neležia na spoločnej kružnici. Z toho vyplýva, že všetky kružnice, ktoré vieme vytvoriť ležia celé v nejakom dvojsúvislom komponente. To znamená, že sa nám stačí pozerať zvlášť na dvojsúvislé komponenty. Nech K je nejaký dvojsúvislý komponent. Ak veľkosť K je menšia ako 3, tak tento komponent môžeme rovno zahodiť (kružnicu dĺžky 3 tam nenájdeme). Teda zmysel sa má pozerať len na komponenty veľkosti 3 a viac.

Zoberme dvojsúvislý komponent K, ktorý obsahuje aspoň 3 vrcholy. Kedy ho môžeme určite zahodiť? No predsa vtedy, ak v ňom neexistuje cyklus nepárnej dĺžky. Taký komponent

[15] Spojením týchto dvoch ciest dostaneme spomínanú kružnicu.

potom nazývame **bipartitný** [16]. Keďže túto situáciu riešiť vieme, tak môžeme predpokladať, že náš komponent obsahuje kružnicu nepárnej dĺžky. Zoberme preto nejakú ľubovoľnú nepárnu kružnicu z komponentu K. Rozmyslite si, že táto kružnica bude mať dĺžku aspoň 3. Chcem teraz ukázať, že ľubovoľný vrchol z komponentu leží na nejakej kružnici nepárnej dĺžky. Inými slovami: chcem ukázať, že vieme pre každý vrchol nájsť kružnicu nepárnej dĺžky, na ktorej leží. Zoberme teda vrchol z K, ktorý **neleží** na našej vybranej kružnici (pre ležiace dôkaz prenechávam na čitateľa) a označme ho v. Ak taký vrchol v neexistuje, to znamená, že každý vrchol z dvojsúvislého komponentu leží na nájdenej kružnici a teda nemáme čo dokazovať. Z toho, že komponent je dvojsúvislý vyplýva, že existujú aspoň dve cesty také, že začínajú vo vrchole v, každá končí v nejakom vrchole (pre každú cestu iný koncový vrchol) na našej kružnici, pričom tieto cesty nemajú spoločný vrchol (okrem vrchola v) a tiež nemajú spoločný vrchol s kružnicou (okrem koncových vrcholov). Označme koncové vrcholy a a b. Spojením našich dvoch ciest vieme vytvoriť cestu z a do b. Všimnite si, že vrcholy a a b nám rozdelili kružnicu na dve polovice[17]. Jedna z nich má párnu dĺžku a druhá nutne nepárnu. Niektorí z Vás už určite tušia. Ak je cesta z a do b cez v nepárnej dĺžky, tak ju spojíme s párnou časťou kružnice. Ak je tá cesta párnej dĺžky, tak ju spojíme s nepárnou časťou kružnice. V oboch prípadoch sme vytvorili nepárnu kružnicu, na ktorej leží vrchol v. To znamená, že ak dvojsúvislý komponent nie je bipartitný, tak každý vrchol leží na nejakej kružnici nepárnej dĺžky.

Ako teda bude vyzerať náš algoritmus? Načíta graf a vytvorí jeho **komplement**. Potom ho rozdelí na **dvojsúvislé komponenty** a potom komponenty, ktoré majú dva a menej vrcholov a tie, čo sú **bipartitné** zahodí. Ostanú nejaké komponenty, pričom každý vrchol nachádzajúci sa v nejakom z pozostalých komponent je dobrý, pretože sa nachádza na kružnici nepárnej dĺžky (a dlhšej než 2). A zároveň žiadny iný. Teda našiel som „dobrých“ rytierov, takže „zlí“ sú nutne tí ostatní (aspoň v Artušových očiach).

A čo časová zložitosť? Nech N je počet rytierov a M je počet konfliktov. Zostrojiť komplement grafu viem v čase $O(N^2)$. Teraz dostanem graf s počtom hrán rádovo $N^2 - M$. Zostrojiť dvojsúvislé komponenty zvládam v čase $O(N + N^2 - M) = O(N^2)$. Zistiť, či je komponent bipartitný zvládam v lineárnom čase od veľkosti komponentu a preto celková časová zložitosť je $O(N^2)$. Keďže si graf pamätám ako maticu susednosti, tak pamäťová zložitosť algoritmu je tiež $O(N^2)$.

Nasledujúce časti vzoráku sa venujú už len nachádzaniu dvojsúvislých komponentov a kontroly bipartitnosti grafu. Kto vie ako na to, tak to môže preskočiť.

Ako nájsť dvojsúvislé komponenty? Budeme prehľadávať graf do hĺbky[18]. Keď sa počas prehľadávania dostaneme do vrcholu, kde sme už raz boli, tak sme našli nejaký cyklus. Teda všetky vrcholy, ktoré sa nachádzajú na nájdenom cykle patria do toho istého dvojsúvislého komponentu. Ako sa to ale dané vrcholy dozvedia? Najjednoduchší spôsob je taký, že každému vrcholu počas prehľadávania pripravíme číslo, ktoré sa bude rovnať poradiu v ktorom sme do daného vrcholu vošli (odteraz tento pojem budeme označovať ako poradie vrchola). Vždy, keď sa počas prehľadávania zavoláme na vrchol, v ktorom sme už boli, tak vrátime jeho poradie. Teda keď nájdeme cyklus, tak sa nám vráti nejaké poradie, ktoré je menšie ako naše poradie. Budeme si pamätať najmenšie poradie, ktoré sme takto videli (teda ako najviac naspäť sa vieme dostať). Aby sme aj ostatným vrcholom na našom cykle povedali, že sú na cykle, tak keď sa budeme vracať naspäť tak vrátime najmenšie poradie, ktoré sme videli[19]. Aby sa to dalo ľahšie predstaviť, tak som pre vás nakreslil obrázok. Prvé

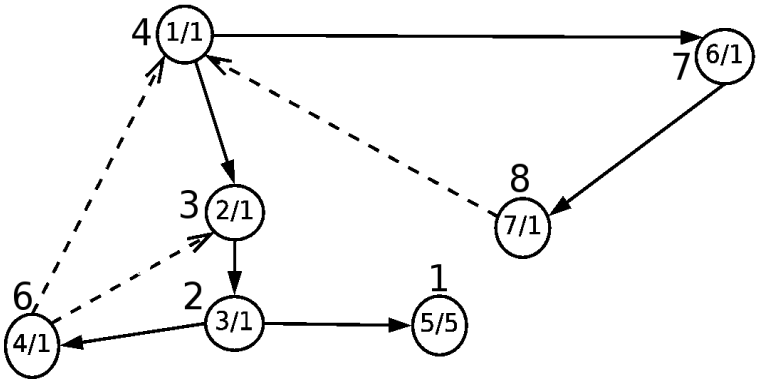
[16] Graf je bipartitný práve vtedy, ak sa jeho vrcholy dajú rozdeliť do dvoch disjunktých množín, pričom každá hrana spája vrcholy z rôznych množín.

[17] Ospravedlňujem sa za použitie výrazu polovica, aj keď nie sú rovnaké.

[18] Uvedený postup pripomenie hľadanie silne súvislých komponentov v orientovanom grafe alebo artikulácii v neorientovanom.

[19] Voľne preložené, ako najviac naspäť sa vieme vrátiť.

číslo vo vrchole znamená poradie, v ktorom sme sa do toho vrcholu dostali a druhé číslo je najmenšie číslo (pardón, poradie vrchola) ktoré sme videli. Číslo pri vrchole znamená číslo vrchola (jeho jednoznačný identifikátor).



Ako vyzeralo prehľadávanie do hĺbky? $4 \to 3 \to 2 \to 6 \to 3 \leftarrow 6 \to 4 \leftarrow 6 \leftarrow 2 \to 1 \leftarrow 2 \leftarrow 3 \leftarrow 4 \to 7 \to 8 \to 4 \leftarrow 8 \leftarrow 7 \leftarrow 4$ ($\to$ znamená vnorenie a $\leftarrow$ znamená vynorenie). Vrcholy sme objavili v poradí $4, 3, 2, 6, 1, 7, 8$, a podľa toho im prislúcha poradie (prvé číslo v krúžku). Teraz sa pozrime na to, keď sme sa dostali druhý krát do vrchola 3. Vtedy sme tam už boli, tak vrátime naspäť najmenšie poradie vrchola ktoré sme videli, čo bude v tomto prípade 2. Keď pôjdeme následne do vrchola 4, tak vrátime číslo 1. To znamená, že vrchol číslo 6 videl nasledujúce čísla: $1, 2$ a 6 (sám seba samozrejme vidí). Keďže najmenšie číslo, ktoré videl vrchol číslo 6 bolo 1, tak prirodzene vráti 1. Intuitívne by sa mohlo zdať, že vrcholy, ktoré videli ako najmenší vrchol ten istý budú patriť do toho istého dvojsúvislého komponentu (v dvojsúvislom komponente sú každé dva vrcholy na nejakej kružnici). Ale už na obrázku je znázornené, že to tak nie je. Keď si pozriete vrchol 4, tak ten patrí do dvoch dvojsúvislých komponentov. Takže iba takéto obyčajné prehľadávanie nám nestačí.

Treba to ešte akosi vylepšiť. Poďme si všímať čísla, ktoré sme porátali počas prehľadávania. Pozrime sa na to, čo sa nám počas prehľadávania vrátilo. Ak sa nám vrátilo to isté číslo ako my, tak to znamená, že práve sme prehľadali jeden celý dvojsúvislý komponent (napríklad keď sa z trojky do štvorky vrátila jednotka). Ale ako zistíme, ktoré vrcholy do toho dvojsúvislého komponentu patria? Tu nám dá pomocnú ruku jedna pekná dátová štruktúra, ktorá sa volá stack (po slovensky zásobník). Vždy, keď vojdeme prvý krát do vrcholu, tak číslo vrchola (pozor, nie poradie!) vhodíme na vrchol zásobníka. Ak sa nám vráti poradie, ktoré je rovnaké, ako naše poradie, tak vyberieme prvky zo zásobníka až po číslo vrchola v ktorom sme. Tieto prvky aj s aktuálnym vrcholom patria do jedného dvojsúvislého komponentu[20]. A pokiaľ sa nám vráti číslo väčšie ako naše poradie, tak povyhadzujeme prvky zo zásobníka až po aktuálny vrchol (zamyslite sa a uvedomte si, že v skutočnosti stačí vyhodiť jeden vrchol).

Akú má tento zázrak časovú zložitosť? V princípe je to iba trošku upravené prehľadávanie do hĺbky, takže celková časová zložitosť je $O(N + M)$, ak N je počet vrcholov a M je počet hrán.

Ako zistiť, či je komponent bipartitný? Táto časť je celkom jednoduchá. Komponent je bipartitný práve vtedy, keď neobsahuje kružnicu nepárnej dĺžky. To znamená, že všetky kružnice majú párnu dĺžku. Náš algoritmus bude teda vyzerať ako obyčajné prehľadávanie do hĺbky, ibaže s tým, že budeme vrcholy striedavo farbiť dvoma farbami. Raz červenou, raz čiernou. Ak prídeme do vrcholu v ktorom sme boli, tak skontrolujeme, či ho nechceme ofarbiť na inú farbu, ako má a ak hej, tak sme našli nepárny cyklus a teda komponent nemôže byť bipartitný[21]. Ak sa nám toto nestalo, tak je komponent bipartitný, pretože sme nenašli

[20] Ešte prečítajte nasledujúcu vetu a potom sa poriadne zamyslite, že je to tak.

[21] Nakreslite si to, a poriadne rozmyslite, že je to naozaj tak!

kružnicu, ktorá by nám to kazila. Tento algoritmus má zložitosť jednoduchého prehľadávania do hĺbky Keď má komponent N vrcholov a M hrán, tak je zložitosť $O(N + M)$.

Tak a teraz si môžete pokojne preštudovať vzorák, aby ste nabudúce také poslali :-)

Listing programu:

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

int N,M;
vector<int> V;
vector<int> was;
vector<vector<int> > D; //dvojsuvisle komponenty
vector<vector<bool> > G;
vector<int> F,R;
stack<int> S;
int poc=1;

int dvojsuvis(int v,int p){
    if(was[v])return V[v];
    was[v]=true;
    V[v]=poc++;
    int mn=V[v];
    S.push(v);
    for(int i=0;i<N;i++){
        if(G[v][i]&&i!=p&&i!=v){
            int k=dvojsuvis(i,v);
            if(k>V[v])S.pop();
            if(k==V[v]){
                vector<int> a;
                while(S.top()!=v){
                    a.push_back(S.top());
                    S.pop();
                }
                a.push_back(v);
                D.push_back(a);
            }
            mn=min(mn,k);
        }
    }
    return mn;
}


int je_bipartitny(int k,int v,int f){
    if(F[v]>0){
        if(f!=F[v])return false;
        return true;
    }
    F[v]=f;
    f=f%2+1;
    bool result=true;
    for(unsigned int i=0;i<D[k].size();i++){
        if(v!=D[k][i]&&G[v][D[k][i]])
            result&=je_bipartitny(k,D[k][i],f);
    }
    return result;
}
```

```
int main(){
    cin >> N >> M;
    V.resize(N,0);
    was.resize(N,false);
    G.resize(N,vector<bool>(N,true));
    for(int i=0;i<M;i++){
        int a,b;
        cin >> a >> b;
        a--;b--;
        G[a][b]=G[b][a]=false;
    }
    int zlych=0;
    //rozbi to na komponente
    for(int i=0;i<N;i++)
        if(!was[i])dvojsuvis(i,-1);
    //bipartitni
    F.resize(N);
    R.resize(N,0);
    for(unsigned int i=0;i<D.size();i++){
        if(D[i].size()<3)continue;
        for(unsigned int j=0;j<D[i].size();j++){
            F[D[i][j]]=0;
        }
        if(!je_bipartitny(i,D[i][0],1)){
            for(unsigned int j=0;j<D[i].size();j++){
                R[D[i][j]]=1;
            }
        }
    }
    for(int i=0;i<N;i++)if(R[i]==0)zlych++;
    cout << zlych << endl;
    return 0;
}
```

10. Táničkin tajomný trik

opravoval Lukáš
(max. 15 bodov)

Podľa počtu riešení usudzujem, že ste sa tohto príkladu všetci zľakli. A pritom sa dal vyriešiť bez rozmýšľania pomocou Gaussovej eliminácie v čase $O(N^3)$. Ak ste porozmýšľali, dala sa dosiahnuť zložitosť $O(P^2)$.

Najprv si niečo povieme o modulárnej aritmetike (všetky operácie robíme modulo prvočíslo P). Operácie plus, mínus a krát fungujú úplne prirodzene. Ale čo s delením? Musíme sa zmieriť s tým, že nulou nebudeme vedieť deliť. Nech teda chceme vyrátať a/b. Platí tiež $a = a \cdot 1$, teda $a/b = a \cdot (1/b)$. Týmto trikom sme dosiahli, že nám stačí, ak budeme vedieť vyjadriť $1/b$ pre hocijaké b a delenie potom prevedieme na násobenie $a \cdot (1/b)$. Ďalej budeme namiesto $1/b$ používať označenie b^{-1} (inverzný prvok k b). Aké vlastnosti musí mať b^{-1}? Napríklad $b^{-1} \cdot b = 1/b \cdot b = 1$. Ukážeme, že pre každé $b > 0$ existuje také b^{-1}, že $b^{-1}b = 1$. Vezmime si postupnosť $0, b, 2b, \ldots, (P-1)b$. Jej hodnoty sú medzi 0 a $P-1$ (keďže rátame modulo P). Zároveň sa žiadne hodnoty v postupnosti neopakujú, lebo $kb = lb$ by znamenalo, že $l = k$ (dôkaz máte na domácu úlohu – stačí využiť fakt, že P je prvočíslo). Máme teda P rôznych celých čísel medzi 0 a $P-1$, preto sa tam každé číslo nachádza práve raz. To znamená, že v postupnosti $0, b, 2b, \ldots, (P-1)b$ je 1 a teda existuje b^{-1} také, že $b^{-1}b = 1$.

Teraz sa budeme chvíľu hrať s polynómami. Predstavme si, že čísla na tabuli sú postupne hodnoty nejakého polynómu $Q(0), Q(1), \ldots, Q(P-1)$. Keď zopakujeme Táničkinu procedúru, teda zapíšeme do kruhu rozdiely susedných hodnôt, dostaneme $Q(1)-Q(0), Q(2)-Q(1), \ldots, Q(0)-Q(P-1)$. Nech $Q'(x) = Q(x+1)-Q(x)$, potom predchádzajúca postupnosť je rovnaká ako $Q'(0), Q'(1), \ldots, Q'(P-1)$. Všimnime si, že $Q'(x)$ je tiež polynóm. A nie hocijaký. Má stupeň o jedna menší ako $Q(x)$. Prečo? Rozmýšľajte na domácu úlohu.

Ďalej dokážeme nasledovné tvrdenie: Nech sú dané čísla $x_0, \ldots, x_n$ a $y_0, \ldots, y_n$, potom existuje práve jeden polynóm $Q(x)$ stupňa najviac n taký, že $Q(x_i) = y_i$ pre všetky i. Ukážeme si rovno, ako taký polynóm zostrojiť. Tento algoritmus sa volá Newtonova interpolácia. Nech $Q_i(x)$ je taký polynóm, že pre $j \leq i$ platí $Q_i(x_j) = y_j$. Takto sa dajú postupne vypočítať polynómy $Q_0(x), \ldots, Q_n(x)$:

$$Q_0(x) = y_0$$

$$R_{i+1}(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_i)}{(x_{i+1}-x_0)(x_{i+1}-x_1)\ldots(x_{i+1}-x_i)}$$

$$Q_{i+1}(x) = Q_i(x) + (y_{i+1} - Q_i(x_{i+1}))R_{i+1}(x)$$

Indukciou dokážeme, že každý polynóm $Q_i(x)$ má požadované vlastnosti. Pre $i = 0$ je tvrdenie zrejmé. Ďalej využijeme indukčný predpoklad. Všimnime si, že polynóm $R_{i+1}(x)$ je nulový pre všetky x_j, kde $j < i+1$. Preto platí $Q_{i+1}(x_j) = Q_i(x_j) = y_j$ pre všetky $j < i+1$ (vyplýva to z indukčného predpokladu). Navyše platí $R_{i+1}(x_{i+1}) = 1$. Preto $Q_{i+1}(x_{i+1}) = Q_i(x_{i+1}) + (y_{i+1} - Q_i(x_{i+1})) = y_{i+1}$. Teda $Q_{i+1}(x_j) = y_j$ pre všetky $j \leq i+1$. Po n krokoch vyrátame polynóm $Q_n(x)$, ktorý je hľadaným polynómom $Q(x)$.

Práve sme ukázali, že existuje aspoň jeden polynóm s danou vlastnosťou. Ukážeme, že je práve jeden. Sporom. Nech existuje ešte druhý polynóm $S(x)$ taký, že $S(x_i) = y_i$ pre všetky i. Potom $S(x) - Q(x)$ je tiež polynóm a platí $S(x_i) - Q(x_i) = 0$ pre všetky i. To znamená, že $S(x) - Q(x)$ má $n+1$ koreňov. Ale $S(x) - Q(x)$ je najviac stupňa n, preto musí platiť $S(x) - Q(x) = 0$ a teda $S(x) = Q(x)$. A to je spor s tým, že existuje polynóm rôzny od $Q(x)$ s danou vlastnosťou.

Sumá sumárum, existuje práve jeden polynóm $Q(x)$ najviac stupňa $P-1$ taký, že na tabuli sú čísla $Q(0), Q(1), \ldots, Q(P-1)$. A podľa toho, čo sme si povedali vyššie, jedným prepísaním znížime stupeň polynómu na tabuli. Teda ak to spravíme $(P-1)$-krát, musíme dostať polynóm stupňa nula, takže na tabuli budú rovnaké čísla. Ak operáciu zopakujeme ešte raz, budú všade nuly a zostanú tam, nech robíme, čo robíme.

Už máme vybudovaný teoretický základ, môžeme začať vymýšľať algoritmus. Newtonovou interpoláciou zostrojíme polynóm z hodnôt, čo sú na tabuli. Nájdeme tak jediný polynóm stupňa najviac $P-1$. Každé prepísanie tabule zníži stupeň polynómu. Preto po N prepísaniach tabule dostaneme polynóm stupňa najviac $P-1-N$. Ak sme interpoláciou dostali polynóm vyššieho stupňa ako $P-1-N$, tak postupnosť nemohla vzniknúť N prepísaniami tabule, takže je zlá. V opačnom prípade dorátame všetky hodnoty, ktoré Tomáš z tabule zotrel (na vstupe sú nahradené otáznikmi). Dorátanie jednej hodnoty trvá čas $O(P)$.

Vyrátať koeficienty polynómu v Newtonovej interpolácii vieme v čase $O(P^2)$. Vyrátať inverzné prvky vieme v čase $O(P \log P)$ alebo dokonca v $O(P)$, ale v programe nájdete iba algoritmus v čase $O(P^2)$. Záujemcov odkazujem na Malú Fermatovu vetu alebo Rozšírený Euklidov algoritmus. Pomocou obidvoch vieme vyrátať jeden inverzný prvok v čase $O(\log P)$. Pre algoritmus v $O(P)$ sa hodia vedomosti z vysokoškolskej matematiky – generátory, rád prvku v grupe, atď. Nakoniec ešte potrebujeme dorátať zmazané hodnoty na tabuli – to vieme v čase $O(P^2)$. Ostatné časti programu trvajú $O(P)$. Pamäťová zložitosť je $O(P)$, keďže si pamätáme len pár polí.

Toto všetko nájdete implemetované vo vzoráku. Poďme sa však ešte trochu zamyslieť, ako sa dá zlepšiť časová zložitosť. Vieme, že výsledný polynóm bude stupňa nanajvýš $P-1-N$. Preto potrebujeme najviac $P-N$ hodnôt na to, aby sme určili jeho koeficienty. Teda výpočet

koeficientov zvládneme v čase $O((P-N)^2)$. Ďalej potrebujeme dorátať čísla na tabuli, ktoré Tomáš zmazal. Rovnako však musíme vyrátať čísla, ktoré Tomáš nezotrel – ak sa totiž číslo na tabuli nebude zhodovať s hodnotou polynómu, polynóm bude mať určite vyšší stupeň ako $P-1-N$. Dorátanie týchto N hodnôt bude trvať $O((P-N)N)$ (keďže polynóm má stupeň $P-1-N$). Spolu dostávame časovú zložitosť $O((P-N)^2+(P-N)N) = O((P-N)P)$.

Listing programu:

```
#include <string>
#include <cstdio>
#include <iostream>
#include <vector>
using namespace std;

int p;
//modulárna aritmetika začiatok
vector<int> inv;
struct Mod {
    int x;
    Mod(int xx) : x(xx) {}
};
Mod operator+(Mod a, Mod b) {
    return Mod((a.x+b.x)%p);
}
Mod operator-(Mod a, Mod b) {
    return Mod((a.x-b.x+p)%p);
}
Mod operator*(Mod a, Mod b) {
    return Mod((a.x*b.x)%p);
}
Mod operator/(Mod a, Mod b) {
    return Mod((a.x*inv[b.x])%p);
}
//modulárna aritmetika koniec

Mod hodnota(vector<Mod> f, Mod x) {
    //hodnota polynómu f(x)
    Mod res = 0;
    for (int i = f.size()-1; i >= 0; i--)
        res = res * x + f[i];
    return res;
}
vector<Mod> operator*(vector<Mod> x, Mod y) {
    //vynásobenie polynómu konstantou
    vector<Mod> res;
    for (unsigned i = 0; i < x.size(); i++)
        res.push_back(x[i] * y);
    return res;
}
vector<Mod> operator+(vector<Mod> x, vector<Mod> y) {
    //sčítanie dvoch polynómov
    vector<Mod> res = x;
    for (unsigned i = 0; i < y.size(); i++)
        if (i < res.size()) res[i] = res[i] + y[i];
        else res.push_back(y[i]);
    return res;
}
ostream& operator<<(ostream& out, vector<Mod> x) {
    //vypísanie vektora
```

```
    for (unsigned i = 0; i < x.size(); i++) {
        out<<x[i].x;
        if (i + 1 < x.size()) out<<" ";
    }
    return out;
}
vector<Mod> Newton(vector<Mod> x, vector<Mod> y) {
    //Newtonova interpolácia
    int n = x.size();
    vector<Mod> f(1, Mod(1)), res;
    for (int i = 0; i < n; i++) {
        Mod h = y[i] - hodnota(res, x[i]);
        for (int j = 0; j < i; j++)
            h = h / (x[i] - x[j]);
        res = res + f * h;
        vector<Mod> tmp = f; tmp.insert(tmp.begin(), Mod(0));
        f = tmp + f * (Mod(0) - x[i]);
    }
    return res;
}
int main() {
    int n;
    cin>>p>>n;
    inv.assign(p, 0);
    for (int i = 1; i < p; i++)
        for (int j = 1; j < p; j++) if (i*j%p == 1)
            inv[i] = j;

    vector<string> a(p);
    vector<Mod> x, y;
    for (int i = 0; i < p; i++) {
        cin>>a[i];
        if (a[i] != "?") {
            int tmp;
            sscanf(a[i].c_str(), "%d", &tmp);
            x.push_back(Mod(i));
            y.push_back(Mod(tmp));
        }
    }

    vector<Mod> poly = Newton(x, y);
    while (!poly.empty() && poly.back().x == 0)
        poly.pop_back();
    n = min(n, p);//nemá zmysel brať väčšie n ako p

    if ((int)poly.size() > p-n)
        cout<<"Ziadne riesenie neexistuje!"<<endl;
    else {
        vector<Mod> res(p, Mod(0));
        for (int i = 0; i < p; i++)
            res[i] = hodnota(poly, Mod(i));
        cout<<res<<endl;
    }
}
```

Výsledková listina po 1. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda	1	2	3	4	5	Σ
1	Kekely Michal	Gym. Varšavská Žilina - Vlčince	1	15	11	15	12	7	60
2	Kováč Jakub	Gym. sv. Cyrila a Metoda Nitra	1	14	13	8	12	12	59
3	Meravý Jan	SPŠ Dubnica nad Váhom	4	12	13	12	10	9	56
4	Bačo Ladislav	Gym. Poštová Košice	3	15	13		12	15	55
5	Kikta Michal	Gym. Tatarku Poprad	1	15	12		12	15	54
5	Kuzma Tomáš	Gym. Alejová Košice	3	14	14		12	14	54
7	Štrbka Dávid	Gym. Grösslingová BA	4	14	11	3	10	15	53
8	Kubina Filip	Gym. Dolný Kubín	4	15	13		12	12	52
8	Kvasnička Igor	Gym. Jura Hronca BA	4	13	12		12	15	52
10	Polák Lukáš	Gym. Jura Hronca BA	2	14	13	11	13		51
11	Demovič Ľuboš	Gym. Grösslingová BA	3	14	11	9	8	8	50
12	Gajdoš Tomáš	Gym. Alejová Košice	2	14	13	11	10	1	49
12	Hozza Ján	Gym. Jura Hronca BA	1	15	14	9	5	6	49
14	Novella Tomas	Gym. Alejová Košice	2	14	13	11	8	1	47
15	Rohár Pavol	Gym. Alejová Košice	2	14	13	9	8	2	46
15	Sabo Michal	Gym. Jura Hronca BA	4	14	12		12	8	46
17	Košdy Martin	Gym. Jura Hronca BA	3	15	13		12	4	44
17	Miklovič Tomáš	Gym. Nové Zámky	2	14	13		10	7	44
19	Dolák Peter	Gym. Pezinok	1	14	13		11	4	42
19	Habovštiak Martin	Gym. Tvrdošín	2	15	13		9	5	42
19	Proksa Ondrej	Gym. Párovská Nitra	3	15	11	9	5	2	42
22	Kalugerov Samuel	Gym. Jura Hronca BA	?	15	12		14		41
22	Szabó Erich	Gym. Senica nad Myjavou	3	14	12	5	6	4	41
22	Uherčík Maroš	Gym. P. de Coubertina Piešťany	3	15	11		8	7	41
25	Goga Rudolf	Gym. Jura Hronca BA	?	15	12		13		40
25	Chlebíková Andrea	Dorothy Stringer High School Brighton	0	15	13			12	40
25	Kušnier Juraj	Gym. Bánovce nad Bebravou	4	15	7		10	8	40
28	Jursa Jakub	Gym. Alejová Košice	3	14	13		12		39
28	Mikláš Ján	Gym. P. de Coubertina Piešťany	3	14	11		12	2	39
28	Sebechlebský Ján	Gym. Žiar nad Hronom	1	15	13		9	2	39
31	Csiba Peter	Gymnázium	4	11	13		12	2	38
31	Jančigová Jarmila	Gym. Jura Hronca BA	4	14	11		6	7	38
31	Vaňo Maroš	SPŠ Martin	4	14	11	3	1	9	38
34	Matušov Izidor	Gym. A.Sládkoviča B. Bystrica	4	14	13		1	9	37
34	Rohaľ Matúš	Gym. Slovenská Bardejov	4	12	7		11	7	37
34	Vavrík Boris	Gym. Jura Hronca BA	1	17	12		5	3	37
37	Čerman Ondrej	Gym. Partizánske	0	15	10			11	36
37	Hašík Juraj	Gym. Grösslingová BA	?	15	11	4		6	36
37	Paulech Matej	Gym. P. de Coubertina Piešťany	3	14	13		4	5	36
40	Tóth Dávid	Gym. Daxnera V.n. Topľou	3	15	13	2	1	3	34
41	Ďurech Dušan	Gym. 17. novembra Topoľčany	?	14	7		10	2	33
42	Tóth Beáta	Gym. maďarské Šahy	3	14	12		6		32
43	Macháč Juraj	Gym. Jura Hronca BA	1	14	10		7		31
43	Piovarči Michal	Gym. Hronská BA	3	14	12		1	4	31
45	Čaniga Lukáš	Gym. 17. novembra Topoľčany	2	13	11			6	30
45	Páleník Martin	Gymnázium Levice	?	14	12			4	30
47	Majdiš Mojmír	Gym. Dolný Kubín	2	14	10		1	4	29
47	Sayedová Monika	Gym. Grösslingová BA	4	14	13			2	29
47	Stopka Martin	Gym. Š. Moyzesa Ružomberok	4	15	10		1	3	29
50	Dittrich Andrej	Gym. Jura Hronca BA	4	15	13				28

	Meno a priezvisko	Škola	Trieda	1	2	3	4	5	Σ
50	Hozzánková Hana	Gym. Jura Hronca BA	2		13		11	4	28
50	Masár Juraj	Gymnázium	1	14	13			1	28
50	Polačko Martin	Gym. Alejová Košice	3	15	13				28
50	Popovič Viktor	Gymnázium J.A. Raymana	2	15	13				28
55	Boško Lukáš	Gym. Jura Hronca BA	3	14	13				27
55	Hagara Michal	Gym. Jura Hronca BA	2	15	12			0	27
55	Kohút Matej	Gym. Jura Hronca BA	3	14	13				27
55	Matyó Hajnalka	Gym. maďarské Šahy	3	14	13				27
55	Ort Miroslav	Gym. Pankúchova Bratislava	3	8	11		5	3	27
55	Pásztor Bálint	Gym. maďarské Šahy	3	14	12		1		27
55	Phuong Mariana	Gym. Jura Hronca BA	1	14	12		1		27
55	Rigdová Emília	Gym. Popradské nábr. Poprad	2	14	13				27
55	Suchetková Mária	Gym. maďarské Šahy	3	14	13				27
55	Szabolcs Szabó	Gym. maďarské Šahy	3	14	7		6		27
55	Šutý Karol	Gym. Jura Hronca BA	4	15	12				27
55	Toman Viktor	Gym. Golianova Nitra	1	14	12			1	27
55	Ziman Michal	Gym. Haličská Lučenec	2	14	13				27
68	Drábiková Juliana	Gym. Čadca	?	13	13				26
68	Haffner Oto	Gym. Hlohovec	4	14	12				26
68	Mudrík Richard	Gym. Čadca	4	14	12				26
68	Štajer Andrej	Gym. Dolný Kubín	2	14	11		1		26
68	Vasko Martin	Gym. Alejová Košice	3	14	12				26
73	Arpáš Jozef	Gym. Piaristická Nitra	3	8	7		7	3	25
73	Kovaľ Anton	Gym. L. Stöckela Bardejov	2	10	12			3	25
73	Kudláč Jakub	Gym. Bilíkova BA	3	15	10				25
73	Kuncová Alexandra	Gym. Alejová Košice	3	12	13				25
73	Mikulec Marek	SPŠE Nové Zámky	2	11	10		1	3	25
73	Špano Marek	Gym. Jura Hronca BA	1	14	10			1	25
73	Štubňa Igor	Gym. Golianova Nitra	1	14	11				25
80	Cudrák Miloš	Gym. Čadca	4	14	10				24
80	Fedoročko Marek	Gym. Alejová Košice	3	14	10				24
80	Hulák Filip	Gym. Golianova Nitra	2	13	11				24
80	Korbaš Rafael	Gym. Hronská BA	1	11	12			1	24
80	Michalek Martin	Gym. P. de Coubertina Piešťany	3	11	10			3	24
85	Konečný Jakub	Gym. Grösslingová BA	2	9	13			1	23
85	Škorec Lubomír	Gymnázium Považská Bystrica	4	11	12				23
85	Zboja Tomáš	Gym. Hronská BA	3	11	12			0	23
88	Baxová Katarína	Gym. Ľudovíta Štúra Trenčín	3	8	13	1			22
89	Bača Rastislav	Gym. Alejová Košice	3	14	7				21
89	Dobiaš Martin	Gym. Hlohovec	4	11	10				21
89	Galbavý Ondrej	Gym. Golianova Nitra	2	14	7				21
89	Iváncai Rastislav	Gym. Hlohovec	4	11	10				21
89	Ivanecký Jakub	Gym. Alejová Košice	3	14	7				21
89	Kriško Boris	Gym. Alejová Košice	3	11	10				21
89	Liščinský Miroslav	Gym. Alejová Košice	3	14	7				21
96	Chrást Lukáš	Gym. Golianova Nitra	3	11	8			1	20
96	Kučera Erik	Gym. Hlohovec	4	9	11	0			20
96	Sajko Miroslav	Gym. Alejová Košice	?	10	10				20
96	Schumann Viktor	Gym. Alejová Košice	3	14	6				20
100	Kuchyňár Martin	Gym. Jura Hronca BA	2	6	13				19
101	Éliás Tibor	Stredné odborné učilište strojárske Komárno	?	11	7				18
101	Jarabek Tomas	Gym. Jura Hronca BA	4	12			6		18
101	Sevela Richard	Gym. Jura Hronca BA	2	8	10				18

	Meno a priezvisko	Škola	Trieda	1	2	3	4	5	Σ
101	Šrámek Martin	Gym. Alejová Košice	3	11	7				18
101	Žák Samuel	Gym. Rajec	2	11	7		0		18
106	Berente František	SPŠ Revúca	3	12	1			4	17
106	Fogaš Pavol	Gym. Alejová Košice	3	10	7				17
106	Chmelář Lukáš	Gym. Alejová Košice	?	10	7				17
106	Šmahovský Marek	Gym. Jura Hronca BA	2	10	7				17
110	Zaťková Michaela	Gym. Párovská Nitra	1		11			5	16
111	Valachovič Peter	Gym. Hronská BA	4	13					13
112	Holas Juraj	Gym. Jura Hronca BA	1		12				12
112	Luščon Michal	Gym. Hlohovec	4		12				12
114	Gabčík Juraj	Gym. Rajec	4	11					11
114	Harminc Pavol	Gym. Alejová Košice	3	11					11
116	Siebenstich Erich	Gym. Jura Hronca BA	?	6					6

Výsledková listina po 1. kole kategórie KSP-O

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
1	Ondrúška Peter	SPŠ Dubnica nad Váhom	4	15	15	15	15	15	75
2	Hapák Samuel	Gym. Grösslingová BA	4	15	15	12	15	10	67
2	Kekely Lukáš	Gym. Varšavská Žilina - Vlčince	4	12	15	13	15	12	67
4	Kostolányi Peter	Gym. Jura Hronca BA	4	15	15	13	15	8	66
5	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	4	15	7	12	14	12	60
5	Fulla Peter	SPŠ Spišská Nová Ves	3	14	15	5	14	12	60
5	Hozza Michal	Gym. Jura Hronca BA	3	12	15	12	13	8	60
8	Varga András	Gym. H. Selyeho Komárno	4	10	15	12	14	8	59
9	Hudec Tobiáš	Gym. Partizánske	2	12	15	12	10	7	56
10	Belan Tomáš	Škola pre mim. nadané deti BA	2	12	11	15	15		53
11	Košinárová Alena	Gym. Grösslingová BA	4	15	15	12	10		52
12	Tureková Katarína	Gym. Tajovského B. Bystrica	4	4	12	12	15	5	48
13	Kušnier Juraj	Gym. Bánovce nad Bebravou	4	10	8	12	10	7	47
14	Fedáková Dominika	Gym. Stará Ľubovňa	4	12	9	8	10	7	46
15	Hojčka Michal	Gym. Partizánske	3	8	15		14	8	45
15	Matula Peter	Gym. Školská Turč. Teplice	4	9	7	8	14	7	45
17	Nagy Kristián	Gym. Jura Hronca BA	4	15	6	15		8	44
18	Boža Vladimír	Gym. Tatarku Poprad	4	14	15	14			43
19	Kukan Matúš	Gym. Grösslingová BA	4	1	15	12	13		41
19	Petrucha Michal	Gym. Metodova BA	3	12	15		14		41
21	Csiba Peter	Gymnázium	4	12	2	11	15		40
21	Kočiský Tomáš	Gym. Grösslingová BA	4	15	9	10		6	40
23	Starovská Mária	Gym. Grösslingová BA	4	1	15	12		8	36
24	Kudláč Matúš	Gym. Bilíkova BA	4	12	12		3	8	35
25	Fekiač Jozef	Gym. Grösslingová BA	3	2	15		15		32
25	Meravý Jan	SPŠ Dubnica nad Váhom	4	10	9		5	8	32
25	Proksa Ondrej	Gym. Párovská Nitra	3	5	2	8	10	7	32
28	Simanová Lucia	Gym. Grösslingová BA	4	2	6		15	7	30
28	Vojtko Jakub	Gym. Jura Hronca BA	3	10	6		14		30
30	Hozza Ján	Gym. Jura Hronca BA	1	5	6		10	8	29
31	Bara Martin	SPŠE Nové Zámky	4	6	9	13			28
32	Bačo Ladislav	Gym. Poštová Košice	3	12	15				27
32	Kikta Michal	Gym. Tatarku Poprad	1	12	15				27
32	Kvasnička Igor	Gym. Jura Hronca BA	4	12	15				27
35	Kuzma Tomáš	Gym. Alejová Košice	3	12	14				26
36	Končok Jakub	Gym. Haličská Lučenec	4			10	15		25
36	Štrbka Dávid	Gym. Grösslingová BA	4	10	15				25
38	Kováč Jakub	Gym. sv. Cyrila a Metoda Nitra	1	12	12				24
38	Kubina Filip	Gym. Dolný Kubín	4	12	12				24
38	Vaňo Maroš	SPŠ Martin	4	1	9	4	3	7	24
41	Korcsok Peter	Gym. Mládežnícka Šahy	4	12	10				22
41	Miňo Igor	Gym. Sobrance	4		5	10		7	22
43	Ďurech Dušan	Gym. 17. novembra Topoľčany	?	10	2			8	20
43	Sabo Michal	Gym. Jura Hronca BA	4	12	8				20
43	Truben Martin	Gym. Jura Hronca BA	4	4	8		8		20
46	Kekely Michal	Gym. Varšavská Žilina - Vlčince	1	12	7				19
46	Matejovičová Lenka	Gym. Grösslingová BA	4		4	10	5		19
46	Matušov Izidor	Gym. A.Sládkoviča B. Bystrica	4	1	9		9		19
49	Rohaľ Matúš	Gym. Slovenská Bardejov	4	11	7				18
50	Miklovič Tomáš	Gym. Nové Zámky	2	10	7				17

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
51	Demovič Ľuboš	Gym. Grösslingová BA	3	8	8				16
51	Košdy Martin	Gym. Jura Hronca BA	3	12	4				16
51	Saleh Adam	Gym. Jura Hronca BA	4	12	4				16
54	Dolák Peter	Gym. Pezinok	1	11	4				15
54	Hozzánková Hana	Gym. Jura Hronca BA	2	11	4				15
54	Uherčík Maroš	Gym. P. de Coubertina Piešťany	3	8	7				15
57	Habovštiak Martin	Gym. Tvrdošín	2	9	5				14
57	Kalugerov Samuel	Gym. Jura Hronca BA	?	14					14
57	Mikláš Ján	Gym. P. de Coubertina Piešťany	3	12	2				14
57	Šutý Karol	Gym. Jura Hronca BA	4				14		14
61	Goga Rudolf	Gym. Jura Hronca BA	?	13					13
61	Jančigová Jarmila	Gym. Jura Hronca BA	4	6	7				13
61	Polák Lukáš	Gym. Jura Hronca BA	2	13					13
64	Herencsár Albert	Gym. maď. Galanta	3	12					12
64	Chlebíková Andrea	Dorothy Stringer High School Brighton	0		12				12
64	Jursa Jakub	Gym. Alejová Košice	3	12					12
67	Čerman Ondrej	Gym. Partizánske	0		11				11
67	Gajdoš Tomáš	Gym. Alejová Košice	2	10	1				11
67	Sebechlebský Ján	Gym. Žiar nad Hronom	1	9	2				11
70	Arpáš Jozef	Gym. Piaristická Nitra	3	7	3				10
70	Novella Tomas	Gym. Alejová Košice	2	8	1	1			10
70	Rohár Pavol	Gym. Alejová Košice	2	8	2				10
70	Szabó Erich	Gym. Senica nad Myjavou	3	6	4				10
74	Paulech Matej	Gym. P. de Coubertina Piešťany	3	4	5				9
74	Phuong Mariana	Gym. Jura Hronca BA	1	1			8		9
76	Ort Miroslav	Gym. Pankúchova Bratislava	3	5	3				8
76	Trančík Ivan	Gym. Jura Hronca BA	4	8					8
76	Vavrík Boris	Gym. Jura Hronca BA	1	5	3				8
79	Macháč Juraj	Gym. Jura Hronca BA	1	7					7
80	Čaniga Lukáš	Gym. 17. novembra Topoľčany	2		6				6
80	Hašík Juraj	Gym. Grösslingová BA	?		6				6
80	Jarabek Tomas	Gym. Jura Hronca BA	4	6					6
80	Stachová Paula	Gym. Bilíkova BA	4	6					6
80	Szabolcs Szabó	Gym. maďarské Šahy	3	6					6
80	Tóth Beáta	Gym. maďarské Šahy	3	6					6
86	Majdiš Mojmír	Gym. Dolný Kubín	2	1	4				5
86	Piovarči Michal	Gym. Hronská BA	3	1	4				5
86	Zaťková Michaela	Gym. Párovská Nitra	1		5				5
89	Berente František	SPŠ Revúca	3		4				4
89	Mikulec Marek	SPŠE Nové Zámky	2	1	3				4
89	Páleník Martin	Gymnázium Levice	?		4				4
89	Stopka Martin	Gym. Š. Moyzesa Ružomberok	4	1	3				4
89	Tóth Dávid	Gym. Daxnera V.n. Topľou	3	1	3				4
94	Kovaľ Anton	Gym. L. Stöckela Bardejov	2		3				3
94	Michalek Martin	Gym. P. de Coubertina Piešťany	3		3				3
96	Sayedová Monika	Gym. Grösslingová BA	4		2				2
97	Chrást Lukáš	Gym. Golianova Nitra	3		1				1
97	Konečný Jakub	Gym. Grösslingová BA	2		1				1
97	Korbaš Rafael	Gym. Hronská BA	1		1				1
97	Masár Juraj	Gymnázium	1		1				1
97	Pásztor Bálint	Gym. maďarské Šahy	3	1					1
97	Špano Marek	Gym. Jura Hronca BA	1		1				1
97	Štajer Andrej	Gym. Dolný Kubín	2	1					1

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
97	Toman Viktor	Gym. Golianova Nitra	1		1				1
105	Hagara Michal	Gym. Jura Hronca BA	2		0				0
105	Zboja Tomáš	Gym. Hronská BA	3		0				0
105	Žák Samuel	Gym. Rajec	2	0					0

Výsledková listina po 1. kole kategórie KSP-T

	Meno a priezvisko	Škola	Trieda	8	9	10	Σ
1	Ondrúška Peter	SPŠ Dubnica nad Váhom	4	15	15	13	43
2	Hapák Samuel	Gym. Grösslingová BA	4	10		13	23
3	Fulla Peter	SPŠ Spišská Nová Ves	3	12	3		15
4	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	4	12			12
4	Kekely Lukáš	Gym. Varšavská Žilina - Vlčince	4	12			12
6	Kudláč Matúš	Gym. Bilíkova BA	4	8	3		11
7	Boža Vladimír	Gym. Tatarku Poprad	4			10	10
7	Proksa Ondrej	Gym. Párovská Nitra	3	7	3		10
7	Vaňo Maroš	SPŠ Martin	4	7	3	0	10
10	Ďurech Dušan	Gym. 17. novembra Topoľčany	?	8			8
10	Hojčka Michal	Gym. Partizánske	3	8			8
10	Hozza Ján	Gym. Jura Hronca BA	1	8			8
10	Hozza Michal	Gym. Jura Hronca BA	3	8			8
10	Kostolányi Peter	Gym. Jura Hronca BA	4	8			8
10	Meravý Jan	SPŠ Dubnica nad Váhom	4	8			8
10	Nagy Kristián	Gym. Jura Hronca BA	4	8			8
10	Starovská Mária	Gym. Grösslingová BA	4	8			8
10	Varga András	Gym. H. Selyeho Komárno	4	8			8
19	Fedáková Dominika	Gym. Stará Ľubovňa	4	7			7
19	Hudec Tobiáš	Gym. Partizánske	2	7			7
19	Kušnier Juraj	Gym. Bánovce nad Bebravou	4	7			7
19	Matula Peter	Gym. Školská Turč. Teplice	4	7			7
19	Miňo Igor	Gym. Sobrance	4	7			7
19	Simanová Lucia	Gym. Grösslingová BA	4	7			7
25	Kočiský Tomáš	Gym. Grösslingová BA	4	6			6
26	Tureková Katarína	Gym. Tajovského B. Bystrica	4	5			5
27	Hozzánková Hana	Gym. Jura Hronca BA	2		3		3
27	Košdy Martin	Gym. Jura Hronca BA	3		3		3