Korešpondenčný seminár z programovania
XXIII. ročník, 2005/06
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

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

Vzorové riešenia 2. kola letnej časti, kat. Z

Milé naše riešiteľky, milí naši riešitelia!

Letné prázdniny sú už na dohľad, tak sme pre vás opravili poslednú sériu KSP-Z. Taktiež sme pre vás aj vzorové riešenia napísali. Tak si ich chytro bežte prečítať a potom hor sa na prázdniny. Dúfame ale, že na nás nezabudnete, a budete nás riešiť aj v ďalšom školskom roku. A taktiež dúfame, že poniektorí z vás začnú riešiť aj „ostrú“ kategóriu. Tak sa tešte na ďalší ročník, budú zaujímavé príklady...

Viete, že už celkom dlho neboli použité tieto priania na sústredení?

KSPáci

1. Zirakrigilská knižnica

opravoval Stano
(max. 15 bodov)

Najskôr si asi povieme, ako som bodoval. Označme si N počet riadkov a M počet stĺpcov.

- Riešenia v čase $O(NM)$ a pamäti $O(M)$ mohli dostať 15 bodov.
- Riešenia v čase $O(NM)$ a pamäti $O(NM)$ mohli dostať 12 bodov.
- Riešenia v čase $O(NM \min(N, M))$ mohli dostať 10 bodov.
- Riešenia v čase $O((NM)^2)$, resp $O(NM \min(N, M)^2)$ mohli dostať 8 bodov.
- Pomalšie riešenia mohli dostať 6 bodov.
- Riešeniam písaným v Befunge som strhával body podľa nálady, najmä za nepoužívanie vyššieho programovacieho jazyka.
- Bodíky sa ešte dali stratiť za nedostatočný popis, či odhady zložitosti.
- Nefunkčné riešenia dostali málo bodov.

Spôsobov, ako túto úlohu riešiť, bolo viacero. Azda najjednoduchšie riešenie je prejsť každé políčko (toto bude pravý dolný roh) a ku každému nájsť príslušnú maximálnu veľkosť. Všetkých políčok je $N \cdot M$, štvorec má veľkosť maximálne N (až M) a treba skontrolovať, či je celý z jednotiek. Takýto program (je to onen 8-bodový) má teda zložitosť $O((MN)^2)$ a je dosť pomalý.

Všimnime si však, že veľmi veľa času nám trvá kontrola, či štvorce obsahujú iba jednotky. Hneď, ako zistíme pre jedno políčko najväčší štvorec, ktorý tam má pravý dolný roh a posunieme sa na vedľajšie políčko, začneme robiť celú tú robotu úplne od začiatku. Dosť by nám pomohlo, keby sme si pri práci nejaké zaujímavé informácie aj zapamätali, aby sme ich mohli využiť neskôr.

Tu sa ku slovu dostáva metóda, ktorá sa nazýva dynamické programovanie. Používa sa vtedy, keď sa riešenie problému dá vyskladať z nejakých menších riešení toho istého problému. Ukážme si toto na našom konkrétnom príklade.[1]

Predstavme si našu mapu v poli A. Do poľa B si do políčka $B[i, j]$ zapamätáme, aký je najväčší štvorec, ktorý má v ňom pravý dolný roh. Ako nám táto informácia pomôže? Nuž

[1] Touto metódou sa dal riešiť aj 4. príklad, kde si môžete pozrieť iný príklad na využitie tejto metódy.

ak vieme, aké veľké sú štvorce, ktoré majú pravý dolný roh na políčku vľavo, hore a vľavo hore, vieme *priamo vypočítať*, aký veľký bude maximálny štvorec, ktorý má pravý dolný roh na tomto políčku. Nakreslite si to!

Ako teda vypočítame číslo $B[i,j]$? Ak $i = 1$, alebo $j = 1$, t.j. pre prvý riadok a stĺpec je to jednoduché – ak máme na vstupe 0, tak v poli B je 0, pretože taký štvorec nie je; ak je $A[i,j] = 1$, tak najväčší štvorec, ktorého pravý dolný roh má súradnice $[i,j]$ má veľkosti 1, teda $B[i,j] = 1$, keďže sme na okraji.

Pozrime sa teraz na ostatné miesta. Pokiaľ je na $A[i,j]$ číslo 0, na tomto políčku nemôže končiť žiaden štvorec a teda aj $B[i,j] = 0$. Ak je na políčku $A[i,j]$ číslo 1, tak určite štvorec končiaci na tomto políčku nebude viac ako o 1 väčší ako štvorec, ktorý končí na políčku nad ním. Platí teda $B[i,j] \leq B[i-1,j] + 1$. Dôkaz tohto faktu je sporom: keby bol štvorec končiaci na $[i,j]$ väčší, mohli by sme zväčšiť aj štvorec, ktorý má pravý dolný roh $[i-1,j]$, t.j. údaj $B[i-1,j]$ by nebol pravdivý, čo je spor. Rovnako platí, že štvorec nebude viac ako o 1 väčší ako štvorec končiaci na políčku vľavo od neho, teda $B[i,j] \leq B[i,j-1] + 1$. A netreba zabudnúť ani na políčko vľavo hore: $B[i,j] \leq B[i-1,j-1] + 1$.

No to je toho... Vieme, že veľkosť štvorca nemôže prekročiť žiadne z čísel $B[i-1,j] + 1$, $B[i,j-1] + 1$ a $B[i-1,j-1] + 1$... Pravda je však taká, že keď si z týchto čísel vyberieme minimum, tak takýto štvorec bude obsahovať samé jednotky! (Nakreslite si to.) Ak by obsahoval nejakú nulu, potom aj niektorý zo štvorcov, ktoré končia na susednom políčku by musel obsahovať nulu, čo je opäť spor. Teda $B[i,j]$ vieme vypočítať jednoducho (a veľmi rýchlo) ako $\min(B[i-1,j], B[i,j-1], B[i-1,j-1]) + 1$. Na záver stačí z hodnôt $B[i,j]$ vybrať maximum.

Takto pre každé políčko (ktorých je $N \cdot M$) zrátame len jedno minimum (spravíme konštantný počet krokov), teda časová zložitosť je vzorových $O(NM)$. A keď sa na to tak pozeráme a tešíme sa z rýchlosti programu, mohlo by nám ešte zísť na um, že nie je potrebné pamätať si celé pole, pretože vždy využívame iba posledné dva riadky. Ak si budeme pamätať iba tie, zlepšíme pamäťové nároky na $O(M)$.

A teraz si poďte pozrieť vzorák...

Listing programu:

```
const SIZE=1000;
var a, b : array [1..SIZE] of integer;
  {a,b by sa dali nahradit jednym polom a[0..1, 1..SIZE]
   potom a[j]=a[i mod 2,j]; b[j]=a[i+1 mod 2,j], ale nie je to prehladne}
  i, j, N, M, vys : integer;

function min(x,y:integer):integer; begin if x<y then min:=x else min:=y; end;
function max(x,y:integer):integer; begin if x>y then max:=x else max:=y; end;

begin
  vys:=0;
  readln(N,M);

  for i:=1 to N do begin
    for j:=1 to M do begin
      read(b[j]);
      if ((i>1) and (j>1) and (b[j]<>0)) then
        b[j]:=min(min(a[j-1],a[j]),b[j-1])+1;
      vys:=max(vys,b[j]);
    end;
    for j:=1 to M do a[j]:=b[j];
  end;
  writeln(vys);
end.
```

A jeden pre priaznivcov C++:

Listing programu:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int N,M,vys=0;
  cin >> N >> M;
  vector <int> a(M), b(M);

  for(int i=0;i<N;i++) {
    for(int j=0;j<M;j++) {
      cin >> b[j];
      vys>?=b[j]*=(i*j) ? (b[j-1] <? a[j-1] <? a[j])+1 : 1;
    }
    a=b;
  }
  cout << vys << endl;
}
```

opravovala *Lφnka*
(max. 15 bodov)

2. Z dražby

Tento príklad bol naozaj ľahký, ako ste si určite všimli aj vy, keďže mi prišla väčšina dobrých riešení. Strhávala som body takto:

- ak ste zabudli ošetriť prípad so samými 9-tkami
- za lineárne počítanie mocnín desiatky
- nuž a samozrejme dalo sa po bode stratiť za zlé/neodhadnuté zložitosti

Pozrime sa teda na riešenie. Najprv si treba uvedomiť, že na to, aby sme vedeli jednoznačne určiť palindróm, nám stačí vedieť prvú alebo druhú polovicu cifier (v prípade, že je číslo nepárne, tak tú väčšiu polovicu) a ich počet. Druhá časť je potom už jednoznačne určiteľná (ak hľadáme 5-ciferný palindróm a vieme, že prvé tri cifry sú 156, potom hľadaný palindróm bude určite 15651). To znamená, že nám stačí pracovať s polovicou čísla.

Majme teda na vstupe palindróm N a chceme nájsť nasledujúci väčší palindróm M. Budeme uvažovať všeobecný prípad, keď palindróm N nie je zložený len zo samých cifier 9. Ten by mal vyzerať tak, že s N má prvých pár cifier rovnakých a potom sa tieto dva palindrómy začnú líšiť. Ako sme si už povedali, prvá (väčšia) polovica cifier (tam kde sú cifry vyšších rádov) určuje druhú polovicu cifier v palindróme a preto rozdiel medzi palindrómmi N a M musí nastať už niekde v prvej polovici cifier (cifry pre vyššie rády). Kedže chceme, aby tieto palindrómy nasledovali po sebe, tento rozdiel musí byť minimálny. To ale znamená, že sa prvá polovica cifier v N a v M líši len o jednotku.[2]

Teraz sa ešte zamyslime nad prípadom, že číslo N je zložené zo samých 9-tok. Je to špeciálny prípad, keď sa mení dĺžka čísla. Tento prípad sa dá špeciálne riešiť tak, že urobíme $N + 2$ a dostaneme opäť palindróm. [3]

[2] premyslite si, ako toto vyzerá s palindrómami nepárnej dĺžky.

[3] premyslite si, ako upravit všeobecné riešenie pre tento prípad.

Túto úvahu teraz použijeme pri hľadaní nasledujúceho väčšieho palindrómu: Najprv zistíme počet cifier pôvodného čísla a overíme, že či ide o všeobecný prípad. Ak je počet cifier párny, zoberieme číslo zložené z prvých $N/2$ cifier, zvýšime ho o 1 a vypíšeme vzniknuté číslo spolu s jeho zrkadlovým obrazom. Ak je počet cifier nepárny, zoberieme aj strednú cifru a toto číslo zväčšujeme o 1. Pri výpise však strednú cifru zrkadlovo nevypisujeme. Ak sa vstupné číslo N skladá len zo samých cifier 9, vieme to overiť a vypísať $N + 2$.

Pamäťová zložitosť tohto prístupu je konštantná, pretože si pamätáme len pár čísel. Časová zložitosť je $O(\log N)$, pretože počet cifier čísla N je logaritmický od jeho veľkosti – $(\lfloor \log N \rfloor + 1)$.

Listing programu:

```
var cislo,cislo2:longint;
    cifry,i:integer;
    same9:boolean;

begin
    readln(cislo);
    cislo2:=cislo;
    cifry:=0;
    while cislo2>0 do begin
        inc(cifry);
        cislo2:=cislo2 div 10;
    end;
    cislo2:=cislo;
    same9:=true;
    for i:=1 to cifry do begin
        if cislo2 mod 10 <> 9 then same9:=false;
        cislo2:=cislo2 div 10;
    end;
    if same9 then begin
        inc(cislo,2);
        writeln(cislo);
    end else begin
        for i:=1 to cifry div 2 do
            cislo:=cislo div 10;
        inc(cislo);
        write(cislo);
        if odd(cifry) then cislo:=cislo div 10;
        while cislo > 0 do begin
            write(cislo mod 10);
            cislo:=cislo div 10;
        end;
        writeln;
    end;
end.
```

3. Zamilovaní KSPáci

opravoval Miňo
(max. 15 bodov)

Vaše riešenia ma pri opravovaní veľmi potešili, všetky riešenia mali dobrú ideu. Kôpky správnych riešení boli tento raz tri – riešenia fungujúce v ideálnom $O(M + N)$ (max. 15 bodov), $O(N^2)$ (max. 12 bodov) a $O(N^3)$ (max. 10 bodov), pamäťové zložitosti boli $O(M + N)$ a $O(N^2)$. Za pamäť $O(N^2)$ išiel dole jeden bodík, za chýbajúce odhady takisto po bode, za zlý alebo chýbajúci popis sa dali stratiť maximálne 2 body. Riešeniam obsahujúcim funkčnú chybu som strhal body podľa závažnosti chyby.

Samotné vzorové riešenie sa dá popísať veľmi jednoducho. Zo všetkých KSPákov, ktorí sú k dispozícii, vyberieme tých, ktorí nikoho neľúbia. Títo sa dostanú do neba a zostane nám nejaká podmnožina KSPákov. Na nich môžeme opäť použiť takéto isté pravidlo, a celé to opakovať, pokiaľ máme koho odobrať. Ak už nemôžeme odobrať KSPáka, ale nejakí zostali na Zemi, tak je v ich vzťahoch „cyklus“ a teda všetci KSPáci sa do neba nedostanú.

Toto celé sa dalo realizovať v časovej aj pamäťovej zložitosti $O(M+N)$ – KSPákov sme si vedeli pamätať v poli od 1 po N, a pre každého z nich sme si pamätali KSPákov, ktorí ho majú radi a počet KSPákov, ktorých má rád. Keď bolo všetko načítané, bolo treba už len nájsť KSPákov, ktorým na nikom nezáleží a tých odstraňovať – jednoduchším algoritmom v čase $O(N^2)$. (Pozrieme sa, či vybranému KSPákovi záleží na niekom a ak nie, tak ho môžeme odobrať a ostatným KSPákom za odstráneného KSPáka znížiť počet KSPákov, ktorých má rád a sú ešte na Zemi.)

Prefíkanejšie riešenie je v čase $O(M+N)$. Nekontrolujú sa zakaždým všetci KSPáci, ale len tí, ktorým sa zmenil počet ľudí, ktorých majú radi (a sú ešte na Zemi). Toto sa dá spraviť viacerými spôsobmi, my si ukážeme tzv. prehľadávanie do šírky, na ktoré potrebujeme *rad* (alebo front).

Rad je dátová štruktúra, ktorá funguje podobne, ako rad v reálnom živote – na jeden jeho koniec sa pridávajú prvky a z druhého konca sa prvky odoberajú. Vo vzorovom riešení na začiatku pridáme do radu všetkých KSPákov, ktorým už na nikom nezáleží a potom ich postupne posielame do neba. Za každého takto do neba poslaného KSPáka treba ostatným (pozemským) KSPákom, ktorí ho mali radi, znížiť počítadlo. Ak sa stane, že niekomu z pozemských KSPákov už nezáleží na nikom z ostávajúcich, tak ho jednoducho vložíme do radu (na jeho koniec). Takto pokračujeme, až kým nemáme prázdny rad – potom stačí už len zistiť, či všetci KSPáci odišli a napísať výsledok.

Listing programu:

```
const kspakov = 100;
type KSPak = record
       maju_radi: array[1..kspakov] of Integer;
       maju_radi_poc: Integer;
       ma_rad: Integer;
     end;

var N, M, i, kto, koho, zf, kf:Integer;
    ksp: array[1..kspakov] of KSPak;
    fronta: array[1..kspakov] of Integer;

begin
  readln(N, M); zf := 1; kf := 1;

  for i:=1 to M do begin
    readln(kto, koho);
    Inc(ksp[kto].ma_rad); Inc(ksp[koho].maju_radi_poc);
    ksp[koho].maju_radi[ksp[koho].maju_radi_poc] := kto;
  end;

  for i:=1 to N do
    if ksp[i].ma_rad = 0 then begin fronta[kf] := i; Inc(kf); end;
  { Neexistuje KSPak, koho nikto nema rad }
  if (kf = 1) then begin writeln('KSPaci skoncia v pekle'); exit; end;

  while (kf - zf) > 0 do begin
    kto := fronta[zf];
    for i:=1 to ksp[kto].maju_radi_poc do begin
      koho := ksp[kto].maju_radi[i]; Dec(ksp[koho].ma_rad);
```

```
      if ksp[koho].ma_rad = 0 then begin
        fronta[kf] := koho; Inc(kf);
        Dec(ksp[koho].ma_rad); {ma_rad=-1 - v nebi}
      end;
    end;
    Inc(zf);
  end;
  { niektori KSPaci zostali pred branou }
  if kf <> N + 1 then writeln('KSPaci skoncia v pekle')
  else for i := 1 to N do write(fronta[i], ' '); { poradie KSPakov }
end.
```

4. Závislosť na hranolkách

opravoval Zemčo
(max. 10 bodov)

Súdiac podľa vašich komentárov a počtu riešení, ktoré prišli, som vycítil, že pred týmto príkladom ste nemali veľký rešpekt. Väčšina z vás ho naozaj zvládla vcelku dobre, pretože pre niekoho, kto sa už niekedy stretol s kombinatorikou na hodinách matematiky, bol naozaj veľmi jednoduchý. Takže na záver úvodného slova trikrát Hurá poznatkom z matematiky!

Ako sa teda dal riešiť tento príklad? Veľmi pomaly, stredne efektívne, rýchlo a optimálne. Veľmi pomaly si spomínať nebudeme, vyskytlo sa len párkrát a dotyční jedinci sú vyrozumení. Stredne efektívny postup sa dá nazvať aj typická ukážka dynamického programovania – postupu, ktorý v našom prípade síce nebol optimálny, ale vo všeobecnosti je veľmi často používaný. Jeho idea je rozdeliť si problém na menšie podproblémy a tie na ešte menšie..., až kým nie sme schopní tieto inštancie riešiť triviálne a tieto riešenia triviálne spojiť do výsledného.

Asi ste z tejto definície veľmi nezmúdreli, takže si to teda skúsme aplikovať na náš problém. Nakreslime si mapu matfyzu na štvorčekový papier a zaznačme si prednáškovú sálu a bufet. Dostali sme mriežku, kde si v každom bode budeme písať číselko, koľko rôznych ciest existuje do tohto bodu. V bode, označujúcom bufet bude teda riešenie. A teraz si všimnime takúto vec: počet ciest do bodu $[x, y]$ nie je nič iné ako súčet počtu ciest do bodu $[x-1, y]$ a do bodu $[x, y-1]$. Pretože z oboch týchto bodov sa vieme dostať do bodu $[x, y]$ jednou cestou a zároveň sú to všetky cesty do bodu $[x, y]$ (ako bolo aj v zadaní poradené, naše cesty pozostávajú len z pohybov dvoch smerov). Z toho vyplýva, že na získanie počtu ciest do nejakého bodu potrebujeme len hodnoty jeho dvoch susedov, ktorí sú bližšie k začiatku. No a ak sme si predstavili toto, tak už ľahko vymyslíme algoritmus v čase $O(M \cdot N)$. Budeme mať pole, v ktorom si budeme pamätať pre každý bod počet ciest a iba ho postupne od začiatku vyplníme. Použijeme vzťah pole$[x, y]$:=pole$[x-1, y]$+pole$[x, y-1]$, samozrejme, ošteríme okraje a tak... Toto riešenie sa dá ešte zlepšiť, ak si uvedomíme, že pri zápise do nejakého riadku už nepotrebujeme to, čo sme počítali dávno – vždy potrebujeme len aktuálny riadok a ten bezprostredne pod nami. Preto môžeme prepisovaním nepotrebných údajov dosiahnúť lineárnu pamäť namiesto pôvodnej $O(N \cdot M)$.

Toto riešenie je pekné, ale nie najlepšie. Kto to nevymyslel rovno, mohol si všimnúť zaujímavé veci na práve popísanom postupe. Konkrétne, že počet ciest do nejakého bodu počítame ako súčet nejakých dvoch hodnôt vypočítaných predtým. Nepripomína nám to niečo? Ak si začnete tento postup naozaj poctivo písať a potom správne papier natočíte, budete si môcť všimnúť Pascalov trojuholník. Teraz aplikujeme hore-spomenuté poznatky z kombinatoriky. Pre tých, čo sa ešte nestretli, alebo o tom nevedia, si najprv Pascalov trojuholník predstavíme. Sú to do trojuholníkovej formácie usporiadané prirodzené čísla. Na ich vrchu je jednotka, na nasledujúcom riadku sú dve jednotky, o riadok nižšie je $1, 2, 1$. Vo všobecnosti – riadok má o jedno číslo viac ako ten predchádzajúci a každé jeho číslo sa získa

súčtom dvoch čísel nad ním (a na kraje riadku dáme číslo 1). V Pascalovom trojuholníku sa dá zbadať veľa vecí, my si ale všimneme, že číslo na n-tom riadku a k-tom mieste je kombinačné číslo $\binom{n}{k}$ (obe veličiny číslujeme od 0), a to je definované ako $n!/(k! \cdot (n-k)!)$. Skúsenejší za problémom okamžite videli kombinácie alebo permutácie s opakovaním, čo je ten istý vzorec. Výsledok sa nachádza na $(N+M)$-tom riadku a na M-tej pozícii (alebo N-tej od konca). Takže na výsledok nám stačí len vypočítať vzorec. Niektorí z vás sa uspokojili, spravili si funkciu faktoriál a v čase $O(N+M)$ získali riešenie, ktoré v analógii zo začiatku druhého odstavca získava prívlastok rýchle.

Ak sa pýtate, ako je možné dosiahnúť niečo lepšie alebo sa domnievate, že už nie je možnosť, ako to zlepšiť, tak je nasledujúci odstavec určený práve vám. K optimálnemu riešeniu nás privedie len jedno jediné pozorovanie, a to, že vo vzorci na výpočet kombinačného čísla sa nám dosť vecí vyškrtá. Ako som už spomenul, čísla M a N môžeme zameniť, na výsledok to nemá vplyv. My totiž počítame $(M+N)$ faktoriál, a dole faktoriál N a faktoriál M. Títo obaja sú určite súčasťou čitateľa. Tak načo počítať celého čitateľa, potom spočítať celého menovateľa a potom to aj tak z väčšej časti škrtnúť? My to spravíme múdro a budeme počítať len to, čo sa neškrtne. A keď si zvolíme to väčšie číslo, nech je to N, tak po škrtnutí ním zostane v menovateli $M!$ a v čitateli M súčiniteľov, ktorí zostali po vykrátení $N!$ z čísla $(N+M)!$. No a po tejto optimalizácii dostávame zložitosť $O(\min(N,M))$. A lepšiu si už asi naozaj nemôžeme želať. Táto optimalizácia nám ako vedľajší produkt prináša trocha väčší rozsah vstupov, lebo toho násobíme menej a dostávame o niečo menšie čísla. Aj tak ale veľkosť faktoriálov zostáva v princípe neriešiteľným problémom, pretože už faktoriál 20 je 19-ciferné číslo. Ešte musím povedať, že sa mi páčilo, ak ste si dokázali, že výsledok po tomto vykrátení je celé číslo.

Nakoniec pár slov k bodovaniu – za $O(N \cdot M)$ sa dávalo 7 bodov, za optimalizáciu pamäte mohla byť pochvala. Za $O(N+M)$ bolo 9 a za $O(\min(N,M))$ sa dalo získať 10. Iné riešenie dostalo iný počet bodov. A klasické postihy boli za zlý odhad alebo nedostatočný popis. Ešte sa chcem ospravedlniť za troška viac počarbané riešenia, pretože som si neskoro všimol, že príklad je 10 bodový.

Listing programu:

```
program hranolky;
var m,n,i:integer;
    vysledok:longint;
begin
  readln(n);
  readln(m);
  if n>m then begin i:=n; n:=m; m:=i; end;
  vysledok:=1;
  for i:=1 to n do
  begin

    vysledok:=vysledok*(m+i);
    vysledok:=vysledok div i;

  end;
  writeln(vysledok);
  readln;
end.
```

opravoval Mic
(max. 15 bodov)

5. Zaplnená chladnička

Takže deti, poďme si povedať, ako som bodoval.

- 15 bodov dostali tí, ktorí dosiahli časovú zložitosť $O(NK)$
- 13 bodov bolo za $O(NK \log N)$.
- 10 bodov som udeľoval za $O(N^2K)$ riešenie.
- ostatné riešenia dostali menej bodov.
- strhával som za nedostatočný popis, zlý odhad zložitosti a iné nedostatky.

Teraz keď už viete, ako som bodoval, vrhnime sa na vzorák. Riešenia prišli všakovaké, ale tu si ukážeme také to najsamlepšie a pritom sa naučíme trošičku pracovať s novou dátovou štruktúrou a to konkrétne s písmenkovým stromom (v angličtine nazývaným trie). Čo to je? Načo je to dobré? Dá sa to jesť? No, jesť sa to nedá, ale ináč je to celkom dobrá vec. Čo je to vlastne strom? Strom má v prvom rade koreň. Z koreňa vedú nejaké vetvy, ktoré končia v nejakých vrcholoch. Z nich potom môžu ísť ďalšie vetvy do ďalších vrcholov. Ak vrchol už nemá žiadne vetvy, ktoré by do neho viedli, tak ho nazývame list. A ako vyzerá taký písmenkový strom? Na každej vetve je napísané nejaké písmenko. Čo znamenajú tie písmenká? Začnime z koreňa. Keď sa posunieme po vetve s písmenkom A, tak to znamená, že si zapíšeme písmenko A. Potom sa vydáme po vetve s H, tak si zapíšeme H. Potom pôjdeme po O, a nakoniec po J. Keď v tomto vrchole skončíme, tak to znamená, že sme dostali slovo AHOJ. Čiže keď vypíšeme písmenká z cesty z koreňa do vrcholu, tak dostaneme slovo, ktoré je uložené v danom vrchole. Ako nám to pomôže v riešení? V každom vrchole si budeme pamätať navyše aj počet výskytov daného slova. Vždy, keď načítame nové slovo, tak si ho vložíme do písmenkového stromčeka a zvýšime výskyt toho slova vo vhodnom vrchole. Na konci prejdeme celý strom a vypíšeme tie slová, ktoré sa tam nachádzajú aspoň raz aj s počtom ich výskytov. V ďalších odstavcoch vám poviem, ako spraviť jednotlivé časti.

Ako vkladať do takéhoto stromčeka? Vytvoríme si vlastný typ TVrchol. Bude obsahovať počet výskytov slova vo vrchole a smerníky na ďalšie vrcholy. Smerník s číslom x označuje že ten vrchol je spojený s vetvou, ktorá má namaľované x-té písmenko. Ak je tam hodnota *nil*, tak tadiaľ žiadna vetva nevedie. Takže najskôr si vytvoríme koreň. Počet výskytov bude mať nastavený na nula a na začiatku z neho nepovedie žiadna vetva. Ako do nášho stromčeka vložíme nejaké slovo? Veľmi jednoducho. Budeme si pamätať vrchol, v ktorom sa aktuálne nachádzame. Na začiatku sme v koreni. Pôjdeme postupne po všetkých písmenkách slova, ktoré pridávame. Zoberieme písmenko a pokúsime sa posunúť hlbšie v stromčeku. Ak vetva z vrcholu V po danom písmenku neexistuje, tak vytvoríme nový vrchol, vynulujeme ho a pripojíme ho k vrcholu V a tým vytvoríme vetvu pre dané písmenko. Teraz, keď vetva s písmenkom existuje, tak sa po nej posunieme ďalej a zoberieme ďalšie písmenko zo slova. Takto postupujeme až kým nevyčerpáme všetky písmenká slova. Keď nám už neostane žiadne písmenko, tak vo vrchole, v ktorom sme zostali (slovo uložené v tom vrchole je slovo, ktoré sme chceli vložiť do slovníka) zvýšime počet výskytov o 1. Tým je vkladanie do stromčeka skončené.

Takto sme do písmenkového stromčeku povkladali všetky slová a dokonca tam máme aj uložené počty, ale ako to odtiaľ dostať naspäť? Na to budeme musieť ten strom prehľadať, a použijeme na to niektorými ľuďmi zatracovanú, inými oslavovanú ... rekurziu. Ako nám táto vec pomôže? Ako to bude vyzerať? Naša rekurzívna procedúra bude mať za parameter smerník na nejaký vrchol. Jej úlohou bude vypísať tie slová, ktoré začínajú na slovo, ktoré je uložené v tom vrchole (samozrejme len tie, čo sa vyskytujú viac ako nula krát). Takže najkôr si skontrolujeme, či sa slovo, ktoré je uložené v danom vrchole, nachádza aspoň raz. Ak hej, tak ho vypíšem. Potom sa rekurzívne zavoláme do vetiev, ktoré vedú z vrchola. Zjavne týmto postupom prejdeme všetky vrcholy a navyše každý práve raz. Už iba stačí doriešiť, ako zistíme, ktoré slovo je uložené v nejakom vrchole? Musel by som kvôli tomu prejsť predsa celú cestu až ku koreňu a potom to odzadu vypísať. Ale použijeme nasledovnú fintu. V jednej globálne premennej si budeme pamätať slovo, ktoré je uložené v aktuálnom vrchole. Ako? Jednoducho. Vždy keď sa vnoríme hlbšie po vetve s nejakým písmenkom, tak to písmenko pridáme k slovu, ktoré si aktuálne pamätáme. Keď sa vynoríme z rekurzie, tak

ho zase vymažeme. Takto budeme vedieť okamžite povedať, aké slovo je uložené vo vrchole, v ktorom sme.

Teraz si ešte povedzme niečo o časovej a pamäťovej zložitosti. Vždy, keď pridáme slovo do stromčeka, tak urobíme rádovo toľko operácií, aké dlhé je to slovo. Pri vypisovaní je to podobne. Vzhľadom na to, že najdlhšie slovo môže mať dĺžku najviac K, tak časová zložitosť je $O(NK)$. Pamäťová zložitosť je na tom podobne, pre každé písmenko si budeme v najhoršom prípade pamätať konštantne veľa pamäte, takže pamäťová zložitosť je $O(NK)$. A teraz si bežte kód prečítať.

Listing programu:

```
program pismenkovy_strom;

type PVrchol=^TVrchol;
     TVrchol=record
       pocet:integer;
       pismenka:array[0..25]of PVrchol;
     end;

var koren:PVrchol;
    s:string;
    i,N,K:integer;

  procedure vloz(slovo:string);
    var vrch,p:PVrchol;
        i,j:integer;
  begin
    vrch:=koren;
    for i:=1 to ord(slovo[0]) do begin
      if vrch^.pismenka[ord(slovo[i])-ord('a')]=nil then begin
        new(p);
        p^.pocet:=0;
        for j:=0 to 25 do p^.pismenka[j]:=nil;
        vrch^.pismenka[ord(slovo[i])-ord('a')]:=p;
      end;
      vrch:=vrch^.pismenka[ord(slovo[i])-ord('a')];
    end;
    Inc(vrch^.pocet);
  end;

  procedure vypis(p:PVrchol);
    var i:integer;
  begin
    if p^.pocet>0 then writeln(p^.pocet,' ',s);
    for i:=0 to 25 do
      if p^.pismenka[i]<>nil then begin
        setlength(s,length(s)+1);
        s[length(s)]:=chr(i+ord('a'));
        vypis(p^.pismenka[i]);
        setlength(s,length(s)-1);
      end;
  end;

begin
  new(koren);
  for i:=0 to 25 do koren^.pismenka[i]:=nil;
  koren^.pocet:=0;
  readln(N,K);
  for i:=1 to N do begin
```

```
      readln(s);
      vloz(s);
    end;
    s:='';
    writeln(s);
    vypis(koren);
end.
```

Výsledková listina po 2. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda		21	22	23	24	25	Σ
1	Hapák Samuel	Gym. Grösslingová BA	2	62	15	15	15	10	15	132
2	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	2	62	10	12	15	10	13	122
3	Nagy Kristián	Gym. Jura Hronca BA	2	58	6	15	13	9	11	112
4	Herencsár Albert	Gym. maď. Galanta	1	54	7	13	11	10	13	108
5	Hlavatý Peter	Gym. Jura Hronca BA	2	36	15	15	15	10	14	105
6	Kostolányi Peter	Gym. Jura Hronca BA	2	51	2	15	11	9	14	102
7	Sucha Martin	Gym. Jura Hronca BA	2	36	12	15	15	10	12	100
8	Korcsok Peter	Gym. Mládežnícka Šahy	2	42	12	15	10	9	10	98
9	Fecko Stanislav	Gym. Pankúchova Bratislava	3	51	5	15	11	5	10	97
10	Košdy Martin	Gym. Jura Hronca BA	1	36	12	15	9	10	12	94
11	Košinárová Alena	Gym. Grösslingová BA	2	37	10	15	11	10	10	93
12	Synak Peter	Gym. Jura Hronca BA	2	42	11	9	11	8	10	91
13	Křemenová Lucia	Neznama skola	1	28	12	15	12	7	15	89
13	Uherčík Tomáš	Gym. P. de Coubertina Piešťany	3	43	2	15	7	9	13	89
15	Švec Marcel	Gym. Jura Hronca BA	3	36	12	15		9	13	85
16	Magyar Vladimír	SPŠE Nové Zámky	2	36	5	15	12	7	9	84
17	Kočiský Tomáš	Gym. Grösslingová BA	2	32	15		12	10	14	83
18	Kvasnička Igor	Gym. Jura Hronca BA	2	36		13	11	10	10	80
19	Fedáková Dominika	Gym. Stará Ľubovňa	2	35		13	11	7	10	76
20	Saleh Adam	Gym. Jura Hronca BA	2	15	12	15	12	8	10	72
21	Mészároš Roman	SPŠ Levice	3	26	7	15	9	1	13	71
22	Sabo Michal	Gym. Jura Hronca BA	2	32	6	9	7	7	9	70
23	Szabados Michal	Škola pre mim. nadané deti BA	3	0	15	15	12	10	15	67
24	Frankovská Veronika	Gym. Mercury BA	2	0	13	13	15	9	14	64
25	Gálik Marian	Gym. Jura Hronca BA	2	0	12	15	12	10	14	63
26	Jančigová Jarmila	Gym. Jura Hronca BA	2	21		7	11	10	9	58
26	Jenis Jakub	Gym. Cyrila a Metoda Nitra	1	26		13		9	10	58
28	Vojtko Jakub	Gym. Jura Hronca BA	1	57						57
29	Simanová Lucia	Gym. Grösslingová BA	2	16	15	15		10		56
30	Bachratý Martin	ZŠ Gaštanová Žilina	1	16		11	10	5	9	51
30	Bimbo Miroslav	Gym. Jura Hronca BA	2	0	12	15	9	6	9	51
32	Kučin Alexander	Gym. Lipany	2	23		15			9	47
33	Blaho Pavol	Gym. Jura Hronca BA	2	40						40
34	Hojčková Martina	Gym. Jura Hronca BA	3	0	3	15	9	10		37
35	Balint Pavel	Gym. Alejová Košice	3	0		13		9	13	35
35	Hanes Filip	Gym. Tajovského B. Bystrica	3	35						35
35	Svitek Andrej	Gym. Jura Hronca BA	1	35						35
38	Košdy Ivan	Gym. Jura Hronca BA	1	0	7	4	7	6	9	33
39	Jagoš Ľubomír	Neznama skola	1	0	4	10		9	8	31
39	Liska Igor	Gym. Jura Hronca BA	1	0	5	11		7	8	31
41	Matula	Gym. Jura Hronca BA	2	0		13	8		8	29
42	Dullová Veronika	Gym. Ul. L. Sáru BA	3	19		9				28
42	Hozza Michal	Gym. Jura Hronca BA	1	9				7	12	28
44	Beňo Fratišek	SPŠ Humenné	2	26						26
45	Dittrich Andrej	Gym. Jura Hronca BA	2	8		7		8		23
46	Nikodem Peter	Gym. Školská Spiš. Nová Ves	2	22						22
47	Tomkovič Viki	Gym. Jura Hronca BA	1	0		13		8		21
48	Popovič Viktor	Gym. Mudroňova Prešov	1	11				9		20
49	Truben Martin	Gym. Jura Hronca BA	2	19						19
50	Tóth Ladislav	SPŠE Nové Zámky	3	18						18

	Meno a priezvisko	Škola	Trieda		21	22	23	24	25	Σ
51	Trančík Ivan	Gym. Jura Hronca BA	2	0				6	8	14
52	Buchovecká Simona	Gym. Medzilaborce	3	7		6				13
53	Korenek Roman	Gym. Jura Hronca BA	1	0		4		7		11
54	Trnovec Matúš	Gym. Jura Hronca BA	2	8	2					10
55	Jarabek Tomas	Gym. Jura Hronca BA	2	0				9		9
56	Janočko Ján	Gym. Jura Hronca BA	1	0				8		8
57	Čevorová Kristína	Škola pre mim. nadané deti BA	3	0	1					1