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

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

Vzorové riešenia 1. kola zimnej časti

A sú tu vzoráky! Kto nevie čo s nimi, nech si naplní ústa vodou a môže začať čítať.

A pozor, mokrá dlážka!!!

KSPáci

1. Ohne na Kapingamarangi

opravoval Lukáš
(max. 15 bodov bodov)

Táto úloha bola ľahká, o čom svedčí aj priemerný bodový zisk. Za riešenie s časovou zložitosťou $O(MN)$ a pamäťovou $O(M)$ ste mohli získať 15 bodov, za riešenie s časovou zložitosťou $O(M^2N^2)$ a pamäťovou $O(MN)$ ste mohli získať 9 bodov. Riešenia medzi tým dostali niečo medzi tým. Pre riešenie úlohy si bolo treba uvedomiť, že každý oheň prepneme najviac raz a nezáleží na poradí prepínania ohňov.

Prečo každý oheň prepneme najviac raz? Označme stav nejakého ohňa x, pričom $x \in \{0, 1\}$. Jeho prepnutý stav nie je nič iné, ako znegované x alebo $x \oplus 1$. Znakom $\oplus$ sa označuje operácia xor, pričom platí $0 \oplus 0 = 1 \oplus 1 = 0$ a $1 \oplus 0 = 0 \oplus 1 = 1$. Ďalej využijeme, že platí $a_1 \oplus a_2 \oplus \ldots \oplus a_n = (a_1 + a_2 + \ldots + a_n) \bmod 2$ a naviac $x \oplus y = z$ je to isté ako $x = y \oplus z$. Keby sme prepli oheň so súradnicou $[i, j]$ viac ako raz, pre všetky ohne, ktorých sa týka kliatba, by to znamenalo prixorovanie hodnoty $y = 1 \oplus 1 \ldots 1 \oplus 1$. Ale z vlastností xoru vieme, že y môže byť nula alebo jednotka, takže všetky ďalšie zmeny sú zbytočné a len nám zhoršujú riešenie.

Prečo nezáleží na poradí prepínania ohňov? Opäť sa pozrieme na vlastnosti xoru. Každý oheň vplyvom kliatby alebo prepnutím zmení svoj stav niekoľkokrát, teda jeho stav je výsledkom xorovania začiatočného stavu a niekoľkých jednotiek. Xor je však komutatívna operácia, teda $x \oplus y = y \oplus x$ a zmeny môžeme vykonávať v ľubovoľnom poradí.

Zoberme si teraz štvoricu ohňov so stavmi $a_{i-1,j-1}, a_{i-1,j}, a_{i,j-1}, a_{i,j}$. Označme počet zmien, ktorými prešli tieto ohne $p_{i-1,j-1}, p_{i-1,j}, p_{i,j-1}, p_{i,j}$. Predstavme si, že vieme hodnoty $p_{i-1,j-1}, p_{i-1,j}, p_{i,j-1}$ a chceme sa rozhodnúť, či zapálime oheň $[i, j]$. Po chvíli trápenia zistíme, že počet zmien, ktorými zatiaľ prešiel náš oheň, je $z = p_{i-1,j} + p_{i,j-1} - p_{i-1,j-1}$. Je to dané tým, že zmeny na týchto troch ohňoch sa prejavia aj na našom ohni. Aby sme zarátali každú zmenu práve raz, musíme použiť hore uvedený vzorec, ktorý vyplýva z toho, že vlastne hľadáme zjednotenie množín a platí $|A \cup B| = |A| + |B| - |A \cap B|$ (skúste si nakresliť). My však už vieme, že hodnota z zmení náš oheň, iba keď je nepárna.

Takže chceme vedieť hodnotu $(p_{i-1,j} + p_{i,j-1} - p_{i-1,j-1}) \bmod 2$. Označme zvyšky čísel $p_{i-1,j}, p_{i,j-1}, p_{i-1,j-1}$ po delení dvomi s, t, u. Potom $(p_{i-1,j} + p_{i,j-1} - p_{i-1,j-1}) \bmod 2 = (s + t - u) \bmod 2 = (s + t - u + 2u) \bmod 2 = (s + t + u) \bmod 2 = s \oplus t \oplus u$. No dobre, ale odkiaľ zistíme hodnoty s, t, u? Musíme si uvedomiť, že pre všetky i, j platí $(p_{i,j} \bmod 2) \oplus a_{i,j} = 1$, keďže po prixorovaní všetkých zmien chceme, aby oheň horel. Takže $(p_{i,j} \bmod 2) = a_{i,j} \oplus 1$ a konečne $s \oplus t \oplus u = a_{i-1,j} \oplus a_{i,j-1} \oplus a_{i-1,j-1} \oplus 1 \oplus 1 \oplus 1 = a_{i-1,j} \oplus a_{i,j-1} \oplus a_{i-1,j-1} \oplus 1$.

Vráťme sa k nášmu ohňu $[i, j]$. Vieme už, či sa zmenil vplyvom zmien naľavo a hore od neho. Jeho momentálny stav je teda $a_{i-1,j} \oplus a_{i,j-1} \oplus a_{i-1,j-1} \oplus 1 \oplus a_{i,j}$. Musíme sa rozhodnúť,

či vykonáme zmenu ($z = 1$) alebo nevykonáme zmenu ($z = 0$) jeho stavu, pričom musí platiť $a_{i-1,j} \oplus a_{i,j-1} \oplus a_{i-1,j-1} \oplus 1 \oplus a_{i,j} \oplus z = 1$ a $z = a_{i-1,j} \oplus a_{i,j-1} \oplus a_{i-1,j-1} \oplus a_{i,j}$.

Vyšlo nám, že oheň musíme prepnúť, ak xor začiatočných stavov troch okolitých ohňov a ohňa samého je 1. Algoritmus je priamočiary – prejdeme všetky štvorice $[i, j], [i-1, j], [i, j-1], [i-1, j-1]$, zoxorujeme ich stavy a z toho už vieme, či máme daný oheň prepnúť. Je ľahké si uvedomiť, že na výpočet si musíme pamätať iba dva riadky, takže pamäťová náročnosť bude $O(M)$. Pri každom ostrove vykonáme konštantný počet operácií, preto časová náročnosť je $O(MN)$. Z technických príčin si okolo mapy doplníme jednotky, aby sme nemuseli ošetrovať všelijaké okrajové prípady.

Listing programu:

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

int main() {
    int n, m; cin>>n>>m;
    bool a[100][2];
    int pocet=0;
    for (int i=0; i<=m; i++) a[i][0]=a[i][1]=1;

    for (int i=1; i<=n; i++) for (int j=1; j<=m; j++) {
        cin>>a[j][i%2];
        pocet+=a[j-1][i%2]^a[j][i%2]^a[j-1][(i+1)%2]^a[j][(i+1)%2];
    }
    cout<<pocet<<endl;
}
```

2. O Semaforovej rally Kocúrkovo 2004

opravoval Palo
(max. 15 bodov bodov)

Tento príklad patril medzi tie ľahšie, o čom ma presvedčila drvivá väčšina z vás, ktorá poslala vzorové riešenie alebo jemu podobné. Riešenia s časovou zložitosťou $O(N^2)$ mohli získať 15 bodov, v čase $O(M \log N)$ 14 bodov, v čase $O(MN)$ 11 bodov, riešenia v exponenciálnej zložitosti maximálne 8 bodov. Body sa dali stratiť za chýbajúci popis a odhad zložitosti alebo za chyby v programe.

Všetci ste prišli na to, že ulice a križovatky tvoria graf a že v tomto grafe treba nájsť najkratšiu cestu medzi dvomi zadanými vrcholmi - štartom v_s a cieľom v_c. Najprv zabudnime na semafory a tvárme sa, že tam vôbec nie sú. V tomto prípade je to klasická úloha a algoritmus, ktorý ju rieši dostal meno podľa uja Dijkstru[1] a funguje nasledovne:

V každom vrchole v si budeme pamätať doteraz nájdenú najkratšiu cestu od štartu v_s. Vrcholy budeme mať rozdelené do dvoch skupín: V prvej skupine budú vrcholy (pracovne si ich označme ako ofarbené), ktorým už vieme skutočnú najkratšiu cestu od štartu. V druhej skupine budú všetky ostatné vrcholy (tie nebudú zatiaľ ofarbené), ktorým vždy budeme vedieť najkratšiu takú cestu od štartu, ktorá vedie len cez ofarbené vrcholy, prípadne u nich budeme mať zaznačené, že taká cesta neexistuje.

Náš algoritmus bude fungovať tak, že si vždy vyberieme nejaký neofarbený vrchol v, ten ofarbíme a aktualizujeme údaje o najkratších cestách ostatným vrcholom. Keď ofarbíme aj cieľ, tak budeme pre neho vedieť najkratšiu cestu od štartu. Pozrime sa lepšie na taký neofarbený vrchol v, ktorý je najbližšie štartu. My vieme z neho najkratšiu takú cestu c_f do

[1] Pozor na pravopis

štartu, ktorá ide len cez ofarbené vrcholy. Ako vyzerá najkratšia cesta c_v, ktorá môže viesť cez ľubovolné vrcholy? Tvrdím, že dĺžka c_v je rovná dĺžke c_f. Rozoberme dva prípdady:

- c_v prechádza len cez ofarbené vrcholy. Najkratšia taká cesta je c_f a c_v od nej nemôže byť dlhšia.
- c_v prechádza aj cez nejaký neofarbený vrchol u, tzn. c_v má tvar: $v_s, \ldots, u, \ldots v$. To ale znamená, že neofarbený vrchol u je bližšie štartu ako v, čo je v spore s tým, že v je najbližší taký vrchol, teda tento prípad nemôže nastať.

Zistili sme, že už vlastne poznáme najkratšiu cestu od štartu po v, teda môžeme v spokojne ofarbiť. Teraz musíme ešte aktualizovať údaje o najkratších cestách nejakým iným neofarbeným vrcholom, pretože sme zmenili množinu ofarbených vrcholov. Všimnime si, že najkratšia cesta od štartu idúca len po označených vrcholoch sa mohla zmeniť len susedom vrchola v. Takže ich musíme všetkých prejsť a pozrieť sa, či cesta prechádzajúca cez vrchol v nie je kratšia ako doteraz nájdená.

Všimnime si, že si nemusíme pamätať celú najkratšiu cestu od štartu v každom vrchole, ale stačí si nám pamätať dĺžku najkratšej cesty a predchádzajúci vrchol na ceste. Z týchto informácií vieme už cestu ľahko odzadu zostrojiť.

Takže náš algoritmus bude v skratke fungovať takto: Na začiatku budú všetky vrcholy neofarbené. Každému vrcholu okrem štartu priradíme najkratšiu vzdialenosť nekonečno. Najkratšia vzdialenosť štartu od štartu je zjavne nulovej dĺžky. Potom budeme pracovať vo fázach. V každej fáze nájdeme najbližší neofarbený vrchol. Ten ofarbíme a jeho susedom aktualizujeme údaj o nájdenej najkratšej ceste v prípade, že sa nám oplatí ísť cez práve ofarbený vrchol. Toto opakujeme, až kým neofarbíme cieľ, alebo keď zistíme, že už žiadny vrchol nejde ofarbiť (t.j. neexistuje žiadna cesta medzi štartom a cieľom).

Čo robiť, keď sú tam semafory? Dôležité pozorovanie je, že sa nám vždy oplatí prísť do nejakého vrchola čo najskôr. Vtedy môžeme z neho vyjsť skôr (ale nemusíme keď svieti červená), ale nemôže sa nám stať, že by sme z neho museli vyjsť neskôr. Takže jediná zmena oproti klasickej implementácii Dijkstrovho algoritmu je v aktualizovaní vzdialeností susedov práve ofarbeného vrchola v. Keď svieti vo v červená, tak sa budeme tváriť, že hrany do susedných vrcholov sú dlhšie o čas, po ktorý červená svieti. Keď vo v svieti zelená, tak aktualizovanie funguje rovnako ako v pôvodnom Dijkstrovom algoritme. To, či svieti červená a koľko bude ešte svietiť vieme zistiť ľahko v konštantnom čase[2].

Ešte sme zabudli na odhady zložitostí: Graf si budeme pamätať tak, že pre každý vrchol si budeme pamätať zoznam z neho vychádzajúcich hrán. Navyše potrebujeme ešte nejaké tie pomocné polia veľkosti N. Takže pamäťová zložitosť nášho algoritmu je $O(N + M)$. Každý vrchol ofarbíme práve raz, teda na každú z neho vychádzajúcu hranu sa pozrieme práve raz, čiže urobíme maximálne $O(M)$ aktualizácií najkratšej vzdialenosti od štartu. Najbližší vrchol od štartu vieme zistiť ľahko v čase $O(N)$ prejdením cez všetky vrcholy. Teda výsledná časová zložitosť je $O(N^2 + M) = O(N^2)$.

Naše riešenie nie je optimálne a dá sa ešte zlepšiť. Ak by graf na vstupe mal málo hrán, tak si môžeme neofarbené vrcholy pamätať v halde, celková časová zložitosť nášho algoritmu by bola $O(M \log N)$. Keďže hrán môže byť však až $O(N^2)$, toto riešenie má v najhoršom prípade horšiu časovú zložitosť. Najlepšia známa implementácia Dijkstrovho algoritmu má časovú zložitosť $O(M + N \log N)$. Je však príliš komplikovaná a ak by ju niekto napísal, dostal by aj viac bodov ako 15.

Listing programu:

```
const MAXN=1000;
var
   sus: array[1..MAXN,1..MAXN] of integer;
```

[2] Nepovinná domáca úloha

```
    vzd: array[1..MAXN,1..MAXN] of integer;
    pocsus,semafor,cesta,cas,trvaly,vysl: array[1..MAXN] of integer;
    start,ciel,n,m: integer;
    a,b,d,min,i,pvysl: integer;
function update_cas(cas,vrchol: integer): integer;
begin
    if ((cas div semafor[vrchol]) mod 2 = 0) then update_cas:=cas
    else update_cas:=(cas div semafor[vrchol])*(semafor[vrchol]+1);
end;

begin
    read(n,m,start,ciel);
    for i := 1 to n do begin
        cas[i]:=maxint;
        cesta[i]:=-1;
        pocsus[i]:=0;
        trvaly[i]:=0;
    end;
    for i := 1 to m do begin
        read(a,b,d);
        pocsus[a] := pocsus[a] + 1;
        pocsus[b] := pocsus[b] + 1;
        sus[a][pocsus[a]]:=b; sus[b][pocsus[b]]:=a;
        vzd[a][pocsus[a]]:=d; vzd[b][pocsus[b]]:=d;
    end;
    for i := 1 to n do read(semafor[i]);
    cas[start]:=0;
    cesta[start]:=0;
    while true do begin
        min := -1;
        for i := 1 to n do
            if (trvaly[i] = 0) and ((min = -1) or (cas[i]<cas[min])) then min := i;
        if (min = -1) or (cas[min] = maxint) then break;
        trvaly[min]:=1;
        for i := 1 to pocsus[min] do
            if (update_cas(cas[min],min) + vzd[min][i] < cas[sus[min][i]]) then begin
                cas[sus[min][i]] := update_cas(cas[min],min) + vzd[min][i];
                cesta[sus[min][i]] := min;
            end;
    end;
    if (cesta[ciel] = -1) then begin writeln('Ziadna cesta neexistuje.');
    end else begin
        i := ciel;
        pvysl:=0;
        while true do begin
            pvysl := pvysl + 1;
            vysl[pvysl] := i;
            i := cesta[i];
            if (i = 0) then break;
        end;
        for i := pvysl downto 1 do write(vysl[i], ' ');
        writeln;
    end;
end.
```

3. O papagájoch

opravoval Bee
(max. 15 bodov)

Veľmi ma potešilo, že prišlo práve toľko riešení, koľko prišlo. Všetci ste sa pekne snažili, ale väčšina z vás bohužiaľ neprišla na optimálne riešenie. Takže aké riešenia vlastne prišli? V prvom rade väčšinou nefunkčné a tie mohli dostať bez výnimky maximálne 4 body. Ale poďme sa zamerať na tých, čo si dali záležať viac a namiesto vymýšľania úžasných, ale nefunkčných programov sa uspokojili s vymyslením obyčajného, ale funkčného. Presne tak, uhádli ste. Hovorím o backtracku, ktorý mohol dostať bodov 8. Ak viete, že váš program nefunguje, alebo si nie ste istý, tak radšej pošlite riešenie horšie, ale funkčné.

Ešte spomeňme riešenia polynomiálne. Tie sa samozrejme zakladajú na niektorej implementácii dynamického programovania. Dá sa spraviť triviálne riešenie v $O(n^4)$, maximálne za 10 bodov, skúšaním od najmenších zhodných dvojíc, ale tým sa nebudeme zaoberať, lebo to zďaleka nie je optimálne. Rozhodne najviac funkčných riešení bolo v $O(n^3)$, ktoré mohli dostať 12 bodov, ale to tiež nebudeme veľmi rozoberať, pretože už sme len krôčik od zložitosti optimálnej, 15-bodovej. Takže poďme na to.

Budeme potrebovať dynamické programovanie, takže si vysvetlíme aspoň podstatu. Jednoducho povedané, postupujeme od menších problémov k väčším, pričom ak riešime nejaký problém, tak predpokladáme, že všetky menšie problémy už máme vyriešené a ich riešenia využijeme pri riešení nášho veľkého problému. Takže toľko o dynamike všeobecne a pozrime sa na praktický príklad:

Naše slovo bude $a_1a_2\ldots a_n$, kde n je dĺžka slova. Označme $P_{i,j}$ počet palindrómov v slove $a_i, a_{i+1}, \ldots, a_j$, kde $1 \leq i \leq j \leq n$. Ďalej si označme $Q_{i,j}$ počet tých palindrómov v slove $a_i, a_{i+1}, \ldots, a_j$, ktoré sa začínajú písmenkom a_i.

Tak sa pozrime, koľko je $P_{i,j}$. Palindrómy v slove $a_i, \ldots, a_j$ sú dvoch typov. Tie, čo začínajú písmenkom a_i a tie ostatné. Tých, čo začínajú písmenkom a_i je $Q_{i,j}$. No a tie ostatné sú presne tie v slove $a_{i+1}, \ldots, a_j$, ktorých je $P_{i+1,j}$. Teda máme $P_{i,j} = Q_{i,j} + P_{i+1,j}$. No a koľko by tak mohlo byť $Q_{i,j}$? To zistíme podobnou úvahou. Budeme rozlišovať dva prípady. Ak $a_i \neq a_j$ tak tie palindrómy nevyužívajú písmenko a_j, teda sú to presne tie, čo sú v slove $a_i, \ldots, a_{j-1}$, ktorých je $Q_{i,j-1}$. Ak $a_i = a_j$ tak tie palindrómy sú troch typov. Tie, čo nevyužívajú písmenko a_j, tých je $Q_{i,j-1}$, samotné a_ia_j a ešte všetky palindrómy tvaru a_iwa_j, kde w je palindróm v slove $a_{i+1}, \ldots a_{j-1}$, tých je $P_{i+1,j-1}$. Dokopy to je $Q_{i,j-1}+P_{i+1,j-1}+1 = P_{i,j-1}+1$. Ešte je dúfam zrejmé, že $P_{i,i} = 1$ a $Q_{i,i} = 1$.

Pomocou získaných rekurentných vzťahov budeme teda počítať hodnoty $P_{i,j-1}$ a $Q_{i,j}$ postupne pre $j = 1, 2, \ldots, n$, pričom pre dané j budeme brať $i = j, j-1, \ldots, 1$. Nakoniec sa dopracujeme k hodnote $P_{1,n}$, čo je riešenie úlohy. Ešte je dobré všimnúť si, že počas výpočtu si nemusíme pamätať vsetky $P_{i,j}$ resp. $Q_{i,j}$, ale len tie, ktoré majú j o jedno menšie. Celé sa to teda dá urobiť v čase $O(n^2)$ a pamäti $O(n)$.

No ale už vás nechám so vzorákom, aby ste si tieto pekné myšlienky mohli vychutnať v praxi.

Listing programu:

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

const int MAX=10000;

int main(){
  cout<<"Zadaj slovo: ";
  string w; cin>>w;
```

```
  //pridame na koniec slova este jeden odlisny znak, aby sa nam
  //lahsie pocitalo, vid dalej
  w+='#'; int l=w.length();

  //vo v[j] je pocet palindromov zacinajucich na j a
  //konciacich nie za i, pricom toto i posuvame ku koncu slova
  //(teda v[j]=Q[j,i])
  int v[MAX];
  for(int i=0;i<l;i++){
    //k bude pocet palindromov v intervale <j,i-1> plus 1
    //(teda bude k=P[j,i-1]+1)
    //na zaciatku je ten interval prazdny (ziadne palindromy tam nie su)
    int k=1; v[i]=1;

    //teraz je v[j]=Q[j,i-1] (okrem v[i])
    //v tomto vykle ho postupne zmenim na v[j]=Q[j,i]
    for(int j=i-1;j>=0;j--){
      //teraz je k=P[j+1,i-1]+1
      k+=v[j];
      //a teraz uz je k=P[j,i-1]+1

      //upravime v[j] podla toho, ci su krajne pismenka rovnake
      //(vid. popis)
      if(w[i]==w[j])v[j]=k;
    }
    //teraz uz je v[j]=Q[j,i]

    //pretoze sme na koniec slova pridali este jeden znak, tak na konci
    //mame v k pocet palindromov v povodnom slove + 1
    if(i==l-1)cout<<"Pocet palindromov je "<<k-1<<endl;
  }
}
```

opravoval Tono a Mirko

4. O hre Scrobbli

(max. 15 bodov)

Najprv si povieme ako vyzerá riešenie, ktoré, hoci nie optimálne, je jednoduché. Ako prvé si treba uvedomiť, že v našom prípade stačí pri porovnávaní dvoch slov len skontrolovať, či sa skladajú z rovnakých písmen. Pridŕžajúc sa tohto, vhodne si pripravíme slovník. Pre každé slovo si vieme v čase lineárnom od K vytvoriť tabuľku počtu výskytov jednotlivých písmen abecedy v ňom. Keďže veľkosť abecedy je konštantná, tieto hodnoty zaberú pre každé slovo len konštantnú pamäť. Príprava zaberie čas $O(NK)$. Pre každý vytiahnutý reťazec (vždy si stačí pamätať len ten aktuálny) si tiež vytvoríme takúto tabuľku. Potom prejdeme postupne celý slovník a budeme porovnávať tieto počty. Ak nájdeme slovo s rovnakou tabuľkou ako má hľadaný reťazec, vieme že sa skladajú z rovnakých písmenok a toto slovo teda môžme vypísať. Vytiahnutých reťazcov je M, pre každý spravíme $O(K + N)$ operácií (+výpis). Celková časová zložitosť teda bude $O((N + M)K + MN + out)$, kde out je veľkosť výstupu (maximálne môže byť $O(MNK)$).

Optimálne riešenie Skúsme teraz vymyslieť riešenie, ktoré by malo časovú zložitosť rovnú veľkosti vstupu a výstupu – $O(M + N)K + out$.

Tentokrát dve slová nebudeme porovnávať podľa počtu výskytov písmeniek, ale podľa ich **anagramov**. **Anagram** nejakého slova je také slovo, ktoré vznikne utriedením jeho písmeniek. Platí teda, že dve slová sa dajú premeniť jedno na druhé práve vtedy, keď sa rovnajú ich anagramy. Tiež platí, že keď v slove ľubovolne pomiešame písmenká, jeho anagram sa

nezmení. Znovu využijeme, že máme len konečne veľkú abecedu – písmenka v slove môžme utriediť countsortom v čase $O(K)$. Ako si ale vhodne pamätať anagramy slovníka? Aby riešenie bolo optimálne, potrebujeme štruktúru, do ktorej vloženie N slov bude trvať $O(NK)$ a v ktorej budeme slovo vedieť vyhľadať v čase $O(K)$. Presne na to máme **písmenkový strom**(aka **trie** alebo **radix tree**).

Takýto strom má ohodnotené hrany, vždy jedným znakom z abecedy. Z každého vrcholu potom vychádza pre každý znak najviac jedna hrana. Vo vrchole si pamätáme, či v ňom končí nejaké slovo(čo v našom prípade nemusíme). Ak áno, je to slovo, ktoré vznikne postupným vypísaním písmeniek na hranách cesty od koreňa k vrcholu. Slovo pridávame tak, že sa v každom vrchole podľa ďalšieho písmenka rozhodneme, ktorou hranou pôjdeme. Ak táto hrana neexistuje, vytvoríme nový vrchol a pokračujeme. Po poslednom písmenku už len vrcholu zapíšeme, že v ňom končí slovo. Na každé písmenko slova pozrieme len raz, zložitosť pridávania je lineárna od dĺžky slova. Vyhľadávanie slova je podobné – podľa ďalšieho znaku sa rozhodujeme, ktorou hranou pôjdeme. Ak neexistuje, slovo určite v strome nie je. Po poslednom písmenku skontrolujeme, či sa vo vrchole končí nejaké slovo. Ak áno, je to naše hľadané. Zložitosť je tiež lineárna od dĺžky vstupu. Pre každý znak v slove sa nám vytvorí najviac jeden nový vrchol – pamäť je lineárna od počtu vložených znakov.

Anagramy slov v slovníku povkladáme do písmenkového stromu. Keďže dve slová môžu mať rovnaký anagram, v každom vrchole si ešte budeme pamätať v spájanom zozname všetky slová, ktorých anagramy ňom končia. Do zoznamu môžme pridávať (po vložení anagramu) na začiatok v konštantnom čase, celkovú časovú zložitosť to nijako nezhorší. Každé slovo má práve jeden anagram – nachádza sa len v jednom spájanom zozname a teda pamäťová zložitosť sa nezmení. Písmenkový strom so spájanými zoznamami vytvoríme v čase $O(NK)$. Potom budeme postupne čítať reťazce. Z každého vždy vytvoríme jeho anagram, ktorý vyhľadáme v strome (v čase $O(K)$). Ak sme boli úspešní, vypíšeme všetky slová zo slovníku s takýmto anagramom – vypíšeme celý zoznam pod príslušným vrcholom. Celková časová zložitosť bude teda $O((M+N)K+out)$, čo je presne to, čo sme chceli.

Hodnotenie Riešenia bežiace v optimálnom čase dostali plných 15 bodov. Riešenia, ktoré mali časovú zložitosť $O((M+N)K+MN+out)$ dostali 12 bodov. Riešenia, so zložitosťou $O(MNK)$ dostali 10 bodov , a tie so zložitosťou $O(MNK^2)$ si zaslúžili najviac 8 bodov.

Iné funkčné riešenia dostali 7 až 14 bodov, podľa časovej zložitosti. Body ste ešte mohli stratiť za chýbajúci, prípadne nedostatočný popis a odhad zložitosti. Bod sme strhli aj za zlú pamäťovú zložitosť – tým, ktorí si zbytočne pamätali všetky vytiahnuté reťazce.

Listing programu:

```
#include<stdio.h>
#include<string.h>
#define maxN 100
#define maxK 100
#define maxM 100
int T[maxK*maxM+maxM][256]; // struktura pre pismenkovy strom, staci mi len
  // vediet, z ktoreho vrcholu ide ktora hrana kam, 0 znamena, ze nejde nikam
int kod[maxK*maxM];  // kod slova, nic dvolezite..., slova s rovnakym kodom
  // maju rovnaky anagram, kod je vlastne cislo vrchola, ktory tym slovam prislucha
char slova[maxN][maxK]; // vstupne slova
int cnt[maxN*maxK+2],cnt2[256],indexy[maxN]; // polia pouzite na count sorty
int treesize=0,root; // premenne sluziace na absluhu datovej struktury

// prida do pismenkoveho stromu slovo
int add(int x,char *a,int slovo){
  // ak neexistuje vrchol, vytvorime ho
  if (!x){ x=++treesize;for(int i=0;i<256;i++) T[x][i]=0;}
  if (*a){ // podmienka je splnena, ak niesme na konci retazca (kazdy retazec konci 0)
    T[x][(int)*a]=add(T[x][(int)*a],a+1,slovo); // ideme na dalsi vrchol, cez hranu
```

```
    // predstavujucu prvy znak v pridavanom slove, a odstranime prve pismenko slova
  } else {
    kod[slovo]=x; // zapametame si, k slovu i jeho kod, slova s = kodom maju =
anagram
  }
  return x;
}
void find(int x,char *a){
  // ak sme na konci vypiseme vsetky slova s kodom x
  if (!*a){
    for(int i=cnt[x];i<cnt[x+1];i++) printf("%s\n",slova[indexy[i]]);
  }  else find(T[x][(int)*a],a+1);
}
int main(){
  int m,n,k,i,j;
  char q[maxK],p[maxK];
  p[0]=0;
  root=add(0,p,(int)3.14);
  scanf("%d %d %d",&m,&n,&k);
  for( i=0;i<n;i++) {
    scanf("%s\n",slova[i]);
    strcpy(p,slova[i]);
    // utriedi slovo count sortom
    for(j=0;j<256;j++) cnt2[j]=0;
    for(j=0;j<k;j++) cnt2[(int)p[j]]++;
    for(j=1;j<256;j++) cnt2[j]+=cnt2[j-1];
    for(j=0;j<k;j++) p[--cnt2[(int)slova[i][j]]]=slova[i][j];
    add(root,p,i);
  }
  // utriedim slova podla ich kodov, cnt[x] hovori kde je ulozene prve slovo s kodom x
  // nebudem triedit samotne slova, len si v poli indexy[i] zapametam, ktore slovo by
  // bolo na i-tom mieste
  // nasledujuce riadky su tiez count sort, takze ho mame 3 krat v programe
  for(i=0;i<=treesize;i++) cnt[i]=0;
  for(i=0;i<n;i++) cnt[kod[i]]++;
  for(i=1;i<=treesize;i++) cnt[i]+=cnt[i-1];
  for(i=0;i<n;i++) indexy[--cnt[kod[i]]]=i;
  for(i=0;i<m;i++){
    scanf("%s\n",p);
    // count sort
    for(j=0;j<256;j++) cnt2[j]=0;
    for(j=0;j<k;j++)   cnt2[(int)p[j]]++;
    for(j=1;j<256;j++) cnt2[j]+=cnt2[j-1];
    for(j=0;j<k;j++)   q[ --cnt2[(int)p[j]] ] = p[j];
      q[k]=0;
      printf("odpovede na:%s\n",p);
      find(root,q);
    }
  return 0;
}
```

5. Ovez fmtkojgjbdz

opravoval MišoF.
(max. 15 bodov)

Ako hovorí názov zadania, budeme si rozprávať niečo o tajoch kryptológie. V prvej sérií sme začali niečím, čomu sa odborne hovorí *secret sharing scheme*, resp. *schéma na zdieľanie tajomstva*.

Podúloha a.

Zjavne stačí zaoberať sa prípadom, keď je ľudí práve K – ak ich je viac, jednoducho vyberú K spomedzi seba.

Na chvíľu si predstavme, že nepočítame modulo P, ale normálne v reálnych číslach. V matematike je dobre známou skutočnosťou, že funkčnými hodnotami v K bodoch je jednoznačne určený práve jeden polynóm stupňa menšieho ako K. (Tento polynóm sa volá *interpolačný*.)

Prečo je tomu tak? Sporom. Majme dva rôzne polynómy $f(x)$ a $g(x)$, ktoré sú stupňa menšieho ako K a zhodujú sa v K bodoch. Potom polynóm $f(x) - g(x)$ je tiež stupňa menšieho ako K. V každom z našich K bodov je však nulový. A keďže nenulový polynóm stupňa d má najviac d koreňov, musí byť $f(x) - g(x) = 0$, čož spor.

Ako takýto interpolačný polynóm nájsť? Jedna možnosť je nasledujúca: Nech sú predpísané body (a_1, b_1) až (a_K, b_K). Všimnime si polynóm $f_1(x) = (x - a_2)(x - a_3)\dots(x - a_K)$. Pre ľubovoľnú z hodnôt a_2 až a_K vráti 0, pre a_1 je nenulový. Označme $c_1 = f_1(a_1)$. Podobne zostrojíme zvyšné polynómy f_i a čísla c_i. No a z nich si teraz výsledný polynóm poskladáme. Bude:

$$f(x) = \frac{b_1 f_1(x)}{c_1} + \frac{b_2 f_2(x)}{c_2} + \dots + \frac{b_K f_K(x)}{c_K}$$

Zjavne všetky f_i sú polynómy stupňa n, preto f je stupňa najviac n. Navyše ľahko zistíme, že vo všetkých zadaných bodoch nadobúda správnu hodnotu. Takže už by len stačilo spočítať $f(0)$ a vyhrali sme.

Pozrime sa teraz, čo sa zmení, ak budeme počítať modulo P. V zadaní bol uvedený hint, že vieme „deliť“. Je tomu naozaj tak? Zoberme nejaké $x \in \{1, \dots, P-1\}$. Všimnime si dva jeho násobky kx a lx. Kedy dávajú rovnaký zvyšok po delení P? Vtedy, keď P delí ich rozdiel $kx - lx = (k-l)x$. Ale keďže P a x sú nesúdeliteľné (lebo P je prvočíslo), znamená to, že P delí $k - l$. Preto čísla x, $2x$, …, $(P-1)x$ dávajú navzájom rôzne zvyšky po delení P. Nutne práve jeden z týchto zvyškov je 1. Nech je to pre yx. Potom y budeme značiť x^{-1} a volať inverzný prvok k x.

Ľahko overíme, že aj keď rátame modulo P, vieme každý polynóm jednoznačne rozložiť na súčin koreňových činiteľov. Preto naďalej platí, že polynóm stupňa d má najviac d rôznych koreňov. A teda naďalej funguje aj náš dôkaz, že danými bodmi je určený práve jeden polynóm stupňa menšieho ako K – ten, ktorý máme nájsť.

Nájdeme ho presne rovnako. Keď si všimneme polynóm $c_i^{-1} f_i(x)$, v bode a_i nadobúda hodnotu 1, v ostatných 0. Preto hľadaný polynóm je

$$f(x) = b_1 c_1^{-1} f_1(x) + b_2 c_2^{-1} f_2(x) + \dots + b_K c_K^{-1} f_K(x)$$

A už naozaj stačí len spočítať hodnotu $f(0)$ mod P a vyhrali sme.

Iné riešenie. Dívajme sa na koeficienty a_i polynómu zo zadania ako na neznáme. Pre každého z K ľudí dostávame dosadením jeho hodnôt jednu rovnicu pre tieto neznáme. Dokopy teda dostaneme K rovníc s K neznámymi. Dá sa ukázať (napr. pomocou determinantov), že táto sústava má práve jedno riešenie. (Pozor, toto nie je úplne zjavné! My o tej sústave vieme, že má aspoň jedno riešenie, ale potrebujeme dokázať, že ich nie je viac – inými slovami, že žiadna rovnica nie je lineárnou kombináciou ostatných.)

Podúloha b.

Opäť, zjavne sa stačí zaoberať prípadom, keď je ľudí práve $K - 1$ – ak ich je menej, neuškodí, ak si zavolajú ďalšieho. Ukážeme silnejšie tvrdenie ako to v zadaní. Dokážeme totiž, že každé heslo je dokonca (na základe toho, čo vedia) rovnako pravdepodobné. Prečo je tomu tak?

Jednoducho si predstavte, že si zvolíme heslo h. Tým sme akoby pridali K-teho človeka, ktorý si pamätá čísla 0 a h. V podúlohe a sme ukázali, že v tejto situácií vieme jednoznačne spočítať polynóm f.

To ale znamená, že našich $K-1$ ľudí je v situácií, kedy je možných polynómov práve P a každý zodpovedá inému heslu. A oni nemajú ako zistiť, ktorý z nich je ten správny.

Opäť pohľad cez rovnice. Rovnakým postupom ako v časti a sa dá ukázať, že keď zoberieme $K-1$ rovníc pre ľudí a pridáme rovnicu $h =?$, dostaneme sústavu, ktorá má jediné riešenie. Každé heslo je teda možné a rovnako pravdepodobné.

Pozor, nestačí povedať, že keď máme menej ako K rovníc, sústava má viac riešení! Čo ak náhodou vo všetkých dostaneme do isté heslo? Napr. sústava $x+y+z=47$, $2x+y+z=47$ má v reálnych číslach nekonečne veľa riešení, ale x je vždy 0.

Podúloha c.

Toto je len jeden z viacerých bezpečnostných problémov tejto schémy. O ostatných problémoch ani o tom, ako sa im dá sa mu zabrániť, tu písať nebudeme, aj tak je toto riešenie už dlhé.

Predstavme si, že účastník 1 chce vykradnúť trezor. Nič ľahšie. Stretne sa s $K-1$ inými a pokúsia sa otvoriť trezor. Každý zverejní svoje zapamätané hodnoty, len účastník 1 zverejní namiesto svojho t_i náhodnú hodnotu. Heslo spočítajú zle, trezor sa im otvoriť nepodarí. (Takže zistia, že niekto podvádzal, ale nevedia, kto to bol.) Účastník 1 však už má dosť informácií na to, aby trezor otvoril.

(Podotknime ešte, že na to mu stačí vedieť jeho hodnotu t_1, hodnotu t_1', ktorú im oznámil, zle spočítané heslo h' a poradové čísla ostatných $K-1$ účastníkov – **nepotrebuje** vedieť ich hodnoty t_i! Teda takýto útok by fungoval aj vtedy, ak by napr. postupne po jednom zadávali tajne svoje hodnoty do počítača a ten im nakoniec oznámil spočítané heslo. Zamyslite sa, ako na to.)

Bodovanie.

Ak ste sa dostali zhruba tam, kam ja v tomto vzorovom riešení, bolo vám nadelených 15 bodov. Ak nie, tak primerane menej.

Na záver...

...už len jedna perla z nemenovaného riešenia: „Keby chcel jeden člen rady ostatných podviesť, mohol by napríklad namontovať do miestnosti s trezorom wifi kameru, ktorá by sledovala displej, pomocou ktorého trezor oznamuje členom rady čísla, zamaskoval by ju za kvetináč...“

Výsledková listina po 1. kole kategórie KSP

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
1	Simančík František	Gym. Grösslingová BA	4	15	15	15	15	15	75
2	Nánási Michal	Gym. Jura Hronca BA	4	15	15	14	15	10	69
2	Perešíni Peter	Gym. Tajovského B. Bystrica	3	15	15	12	15	12	69
4	Takáč Slavomír	Gym. Nové Zámky	3	15	15	15	14	9	68
5	Bulánek Jan	Gymnázium Klatovy	4	15	15	15	14		59
6	Bundala Daniel	Gym. Jura Hronca BA	3	15	15	12	14		56
7	Fedák Matúš	Gym. Stará Ľubovňa	3	14	14	7	12	7	54
8	Cicko Miroslav	Gym. Tajovského B. Bystrica	4	15	14	12	12		53
8	Imriška Jakub	Gym. Jura Hronca BA	3	12	15	12	14		53
10	Bílka Ondřej	Gymnázium	3	15	14	8	15		52
11	Plžík Milan	Gym. Ľ. Štúra Zvolen	4	15	15	4	15		49
12	Petruľák Matúš	Gym. Grösslingová BA	4	15	15	3	15		48
13	Ďuďák Juraj	Gym. Golianova Nitra	3	14	15		13	5	47
14	Zeman Marek	Gym. Jura Hronca BA	4	15	11		12	6	44
15	Palenčár Ján	Gym. Malá Hora Martin	4	9	12	8	14		43
15	Špalek Lukáš	Gym. Čadca	4	14	15	1	13		43
17	Sudolský Michal	Gym. Tajovského B. Bystrica	2	8	8	8	12	6	42
18	Mikuláš Ján	Gym. Haličská Lučenec	3	14	15	2	10		41
19	Beleš Lukáš	Gym. Čadca	4	12	15	3	8		38
19	Buštor Stano	Gym. Jura Hronca BA	4	10	15	6	7		38
21	Dzurenko Miroslav	Gym. Ľ. Štúra Zvolen	4	9	15	8	5		37
22	Kundrák Ľubomír	Ev. lýceum BA	2	7	8	7	12		34
23	Dzetkulič Michal	Gym. P. Horova Michalovce	4	15	10	7			32
24	Mindek Peter	Škola pre mim. nadané deti BA	3	8		8	7	7	30
25	Krištof Peter	Gym. Haličská Lučenec	4	7	6	7	8		28
25	Paulis Peter	Gym. Cyrila a Metoda Nitra	4	7	10	0	10	1	28
27	Ambrošová Lucia	Gym. Grösslingová BA	3	7	10		10		27
28	Krchniak Matej	Gym. Jura Hronca BA	3	11		3	12		26
29	Beran Jakub	Gym. Alejová Košice	3	7		8	2	7	24
29	Domány Dušan	Gym. P. Horova Michalovce	4	14			10		24
31	Janík Oliver	Gym. L. Stöckela Bardejov	5	13			10		23
32	Ďurfina Lukáš	Gym. Golianova Nitra	4	7		7	8		22
32	Okruhlica Adam	Gym. Jura Hronca BA	2	9		4	9		22
34	Kuštárová Tamara	Bilingválne gym. Sučany	4	11			10		21
34	Trnovcová Zuzana	Gym. Jura Hronca BA	4	9	12				21
36	Bodnár Jozef	Gym. Fiľakovo	4	15					15
37	Šuster Vladimír	Gym. Jura Hronca BA	2				10		10