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 letnej časti

„Človek, ktorý nedokáže hovoriť tak, aby mu ostatní rozumeli, je idiot. Rozumieš?“
„Nie, tati.“
Tak či onak, kto nečíta vzoráky, je hnilé vajce (veľkonočné).

1. O hvezdárovi bez oka

opravoval Bee
(max. 15 bodov)

Na tomto príklade v zásade nebolo čo riešiť. Ak ste si neuvedomili nič, tak sa dal spraviť v $O(n^2)$ za 8 bodov. Ak ste si niečo uvedomili, tak to išlo v $O(n \log n)$ za 11. No a napokon, ak ste si uvedomili všetko, tak sa dalo už triviálne napísať lineárne riešenie za 15 bodov. Takže poďme na to ...

Poďme sa venovať riadku prislúchajúcemu k-tej najbližšej hviezde. Zrejme $k-1$ hviezd je bližšie, to znamená, že v tom riadku bude $k-1$ rôznych hodnôt (pretože žiadne dve nemajú rovnakú vzdialenosť) a $n-k$ hviezd bude ďalej, čiže v tomto riadku budeme mať $n-k$ rovnakých hodnôt, ktoré sú zároveň najväčšími hodnotami v riadku [1]

Tieto úvahy už môžeme použiť na skonštruovanie algoritmu. Pamätáme si najväčšiu hodnotu v riadku a ak sa vyskytne viac ako jedenkrát, tak to isto bude hľadaná hodnota. Ešte treba rozobrať prípad, keď je v riadku každá hodnota najviac raz. Zrejme sa potom jedná o jednu z dvoch najväčších hviezd a ich vzdialenosti nevieme z našich údajov zistiť.

Listing programu:

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

const int maxN=1000;

int main(){
  int pole[maxN][maxN];
  int k; cin>>k; k--;
  int maximum=0;
  for(int i=0;i<maxN;i++){
    if(pole[k][i]==maximum){ cout<<"Vzdialenost je "<<maximum<<endl; return 0; }
    maximum>?=pole[k][i];
  }
  cout<<"Vzdialenost sa neda urcit"<<endl;
}
```

2. Ortonormálny teleport

opravoval KuKi
(max. 15 a viac bodov)

Orto-čo? No nič. Na úvod asi nič moc. Tento príklad nebol až taký ťažký... len si pozrite vzoráky a posúďte sami ;-)

[1] Rozmyslite si prečo.

Chceme zistiť, koľko sledov dĺžky K existuje v danom grafe. Toť v zásade úloha celkom kombinátorického rázu. Pri slove 'kombinátorika' vám zrovna oči žiariť nemusia, ale akýsi inštinkt, či skôr skúsenosť, alebo iný podvedomý živel by sa mohol vzchopiť a začať tak jemne nabádať: „kombinátorika, kombinátorika, to bude určite dynamika... haló, počuje ma niekto? dynamicky... no tak...".

No a tí z vás, ktorí si to uvedomili, neboli už veľmi ďaleko od riešenia. Vezmime si nejaký vrchol a pozrime sa na jeho susedov. Každý sled dĺžky ℓ, ktorý začína v tomto vrchole sa skladá z jednej hrany do nejakého suseda a pokračuje sledom dĺžky $\ell - 1$ hrán z tohoto suseda. Ak ste na riešenie ešte neprišli, skúste sa v tomto bode ešte (aspoň) raz zamyslieť nad dynamickým riešením – je to už naozaj priamočiare.

...keby sme teda mali pre každý vrchol v spočítaný počet sledov dĺžky $\ell - 1$, ktoré v tomto vrchole začínajú (označme ho $P^{\ell-1}(v)$), potom počet sledov dĺžky ℓ začínajúcich vo vrchole u je

$$P^{\ell}(u) = \sum_{(u,v)\in E} P^{\ell-1}(v),$$

teda počet sledov o jedna menšej dĺžky z prvého, plus počet sledov z druhého, až posledného suseda. Pôjdeme teda od dĺžky 0 (z každého vrcholu existuje práve jeden sled dĺžky nula), pričom vždy prebehneme všetky hrany a spočítame počet sledov o jedna dlhších. Všimnime si, že si stačí pamätať iba dva riadky tabuľky, resp. iba starú a novú hodnotu pre vrchol. Algoritmus sa teda dá implementovať s pamäťovou zložitosťou $\Theta(M + N)$. Časová zložitosť bude $\Theta(MK)$ – v každej fáze prebehneme všetky hrany (všimnite si ešte, že je jedno, v akom poradí preberám hrany, takže mi na hrany stačí jedno preobyčajné pole).

Niekoľkí z vás programovali na túto úlohu backtrack[2] – i tí boli celkom blízko: stačilo si všimnúť, že ste napísali procedúru s dvoma parametrami, ktorá často počíta to isté, zadeklarovať si jedno pole a výsledky memoizovať (t.j. výsledok rekurzie si zapamätáme a druhýkrát ho už nepočítame, iba vyhľadáme v tabuľke).

Počet bodov za také riešenie, fakt, že sa vzorák ešte neskončil a možno nejaké vyššie spomínané podvedomé živly vás medzičasom už nepochybne presvedčili, že existuje aj čosi „lepšie". Jediný problém s vyššie uvedeným riešením je ten, že čas je lineárne závislý od K, ktoré (ako sa vás snažili presvedčiť dobre mienené príklady hneď pod zadaním) mohlo byť dosť veľké. V čom je háčik? Nuž všimnite si, že ako vstup musíme okrem M-ka zadať aj M hrán, teda keď je čas lineárne závislý od M-ka, je lineárne závislý od veľkosti vstupu. Keď je však algoritmus lineárne závislý od K-čka, je vlastne exponenciálne závislý od veľkosti vstupu (lebo číslo vieme zakódovať v nejakej pozičnej sústave).

Ako sa to teda dá lepšie ako lineárne od K? Poďme opäť dynamicky, ale trošku inak: nebudeme si pamätať pre každý vrchol, koľko je ciest z neho, ale pre každú dvojicu vrcholov, koľko je ciest z prvého do druhého (nejakej dĺžky). Všimnite si, že počet sledov jednotkovej dĺžky medzi každými dvoma vrcholmi je matica susednosti (t.j. pole, kde sú jednotky tam, kde je hrana medzi dvoma vrcholmi (mimochodom hrana je sled dĺžky 1)). Majme teda zapamätané počty sledov dĺžky $\ell-1$, ako zistíme počty dĺžky ℓ? Aby som sa mohol vyjadririť, nech u a v sú vrcholy, pre ktoré chceme zistiť počet sledov dĺžky ℓ. Ako vyzerá každý sled z u do v? No začína sa v u, ide do nejakého suseda w a odtiaľ sú to už staré známe sledy z w do v dĺžky $\ell - 1$. Teda počet sledov dĺžky ℓ z u do w (označme ho $P^{\ell}_{u,v}$) je súčet počtov

[2] ...pričom nesprávne odhadovali časovú a pamäťovú zložitosť; z každého vrcholu môže ísť najviac $N - 1$ hrán; hľadáme sledy dĺžky K, teda backtrack sa po K vnoreniach zastaví – preto je časová zložitosť $O(N^K)$ – je to taký stromček, ktorý sa v každom vrchol rozvetví na N haluzí a má hĺbku K; chybičky sa vlúdili aj do odhadov pamäťovej zložitosti: pri rekurzii sa na stacku ukladá návratová adresa (kde bude program pokračovať, keď procedúra skončí) a parametre; pri K rekurzívnych volaniach teda potrebujeme navyše $\Theta(K)$ pamäte, na ktorú sa pri odhadoch často zabúdalo... tak nezabúdať...

sledov dĺžky $\ell - 1$ zo susedov u do w, teda

$$P^{\ell}_{u,v} = \sum_{(u,w)\in E} P^{\ell-1}_{w,v}.$$

Ešte trochu ináč: „spočítať všetkých susedov“ znamená, že susedov do súčtu započítame raz, ostatných nulakrát. Môžeme si teda na pomoc vziať maticu susednosti A ($A_{uv} = 1$, ak u a v sú susedia, inak nula – presne to potrebujeme). Sumu teda môžeme zapísať takto:

$$P^{\ell}_{u,v} = \sum_{w\in V} A_{u,w} \cdot P^{\ell-1}_{w,v}$$

hej? No a to už dúfam dostatočne pripomína násobenie matíc. Pre tých, ktorý sa matice ešte neučili, matica $n \times n$ je to, čo v programovaní nazývame dvojrozmerné pole a súčin matíc $C = AB$ vypočítame takto:

$$c_{i,j} = \sum_{k} a_{i,k} b_{k,j},$$

teda pôjdeme jedným prstom po i-tom riadku prvej matice a zároveň druhým prstom po j-tom stĺpci druhej matice, dané políčka násobíme a výsledky sčitujeme (takto vypočítame jedno políčko)[3]. Jednotkou vzhľadom na násobenie (t.j. taká matica M, že pre ľubovoľnú maticu A platí $AM = MA = A$, teda M sa správa ako jednotka ($1 \cdot a = a \cdot 1 = a$)) je jednotková matica, ktorá má jednotky na diagonále a nuly inde (napr. pre $n = 2$ je to $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$; takúto maticu označujeme I; na DÚ si dokážte, že naozaj $IA = AI = A$ pre ľubovoľnú[4] maticu A).

Ešte si prezrite tú úžasnú podobnosť a pekne si to zrekapitulujeme: *keď máme maticu, kde sú uložené počty sledov dĺžky ℓ a vynásobíme ju maticou susednosti daného grafu, dostaneme maticu, kde sú uložené počty sledov dĺžky $\ell + 1$.* Jediné, čo nám teda ostáva, je vypočítať maticu A^K. Násobenie matice vieme spraviť priamočiaro v čase $\Theta(N^3)$ (viď zdroják), finteno aj o niečo lepšie. Ako vypočítame A^K? Priamočiaro môžeme K-krát vynásobiť jednotkovú maticu A-čkom – teda so zložitosťou $\Theta(KN^3)$. No to sme sa zbavili toho K-áčka... Počkať, počkať; matice tu nekvákam zbytočne, vzorák stále nekončí. Všimnite si, že keď chceme vypočítať A^{16}, nemusíme maticu 16-krát násobiť – stačí ju 4-krát umocniť na druhú[5]. Takto by sa vyriešili mocniny dvojky – $A^{(2^N)}$ spočítame tak, že A umocníme N-krát na druhú. Čo s nemocninami? Nuž každé číslo sa dá napísať do dvojkovej sústavy, teda ako súčet mocnín dvojky, teda napr. A^{22} môžeme vypočítať ako $A^{2+4+16} = I \cdot A^2 \cdot A^4 \cdot A^{16}$. Prevod do dvojkovej sústavy dúfam nemusím veľmi komentovať – postupne K-čko delíme dvojkou a všímame si zvyšok.

Poznámka. Riešenie v čase logaritmickom od K sa dalo vymyslieť aj ak ste doteraz slovo *matica* nepočuli. Základná myšlienka zostáva rovnaká – rekurzívnym volaním spočítame počty sledov (medzi všetkými dvojicami vrcholov) s dĺžkou $\lfloor K/2 \rfloor$, ak je K nepárne, spočítame z toho počet sledov s dĺžkou $\lceil K/2 \rceil$ a následne z týchto počtov sledov poskladáme výsledok. Riešenie pomocou matíc má ale tú výhodu, že ho vieme ďalej zlepšiť fintami, ktoré si teraz ukážeme.

Už teda iba pár slov ku zložitosti: Maticu A nanajvýš $\log K$-krát umocníme na druhú a nanajvýš toľkokrát ňou vynásobíme výsledok, teda zložitosť bude $\Theta(N^3 \log K)$. Dá sa aj

[3] ak nie, pozrite si zdroják – malo by sa to dať pochopiť

[4] nie celkom ľubovoľnú, pretože násobiť vieme iba také matice, kde šírka prvej a výška druhej sú rovnaké, ale tu sa zaoberáme len peknými štvorcovými maticami a nechcel som vás tým pliesť...

[5] vyplýva to z toho, že násobenie matíc je asociatívne, teda môžeme ho ľubovoľne uzátvorkovať; ak sa vám zdá že to je samozrejmé a táto poznámka je zbytočná, tak vám len s potešením oznámim, že násobenie matíc *nie je* komutatívne, teda činitele si nemôžeme medzi sebou ľubovoľne poprehadzovať... to len tak, aby sa niekto neunáhľoval...

lepšie a síce tak, že matice vieme násobiť aj rýchlejšie ako v $\Theta(N^3)$ – napríklad Strassenovým algoritmom. Jedna malá vsuvka o komplexných číslach: ako vynásobíme dve komplexné čísla? Klasicky $(a+ib)(c+id) = ac+iad+ibc+i^2bd = (ac-bd)+i(ad+bc)$ (lebo $i^2 = (\sqrt{-1})^2 = -1$). Iný spôsob: vypočítajme si napríklad $F = (a-b)(c+d)$, $G = ad$ a $H = bc$. Všimnime si, že $F = ac+ad-bc-bd$, teda $(a+ib)(c+id) = (ac-bd)+i(ad+bc) = (F-G+H)+i(G+H)$. Aaha. A čo sme tým dosiahli? Výsledok sme zistili iba pomocou troch násobení (pri počítaní F, G a H; samozrejme použili niekoľko sčítaní, ale sčítavať vieme väčšinou rýchlejšie ako násobiť). Druhá vsuvka, o divide & conquer[6]: ako vynásobíme dve veľké celé čísla[7]? Priamočiaro školácky v $O(N^2)$, kde N je počet cifier (pripomeňme, že sčítavanie školácky je lineárne). Trikovejší postup: rozdeľme si čísla na dve časti (tak polovičnej dĺžky), teda zapíšme si a ako $\bar{a}\cdot 10^k + \underline{a}$ (napríklad 31415 si napíšeme ako $314\cdot 10^2+15$), rovnako $b$ee. Chceme teda vypočítať $(\bar{a}\cdot 10^k + \underline{a})(\bar{b}\cdot 10^k + \underline{b})$. Roznásobením dostávame $\bar{a}\cdot\bar{b}\cdot 10^{2k} + (\bar{a}\cdot\underline{b} + \underline{a}\cdot\bar{b})\cdot 10^k + \underline{ab}$, pričom k je zhruba polovica. Ak sa rekurzívne zavoláme na násobenie $\bar{a}\cdot\bar{b}$, $\bar{a}\cdot\underline{b}$ atď., dostaneme štyri násobenia a rekurziu $T(n) = 4T(n/2) + O(n)$ (čítaj zhruba: čas na vynásobenie dvoch čísel je 4-krát čas na vynásobenie čísel polovičnej dĺžky + dačo lineárne (sčítanie, násobenie mocninou desiatich). Táto rekurzia má riešenie $T(n) = O(n^2)$, teda veľmi sme si nepomohli (na DÚ dokázať indukciou). Pozrite si ešte raz fintu s komplexnými číslami a určite nebude problém vymyslieť rekurziu iba s troma násobeniami – a to už je rozdiel, pretože $T(n) = 3T(n/2) + O(n) = \Theta(n^{\log_2 3}) = O(n^{1.59})$. No a na podobnom princípe funguje aj Strassenov algoritmus, ibaže maticu rozseká na 4 časti a namiesto 8 použije iba 7 násobení – a to už je rozdiel, lebo $T(n) = 7T(n/2)+\Theta(n^2) = \Theta(n^{\lg 7}) = O(n^{2.81})$. Keby ste sa hecli, tak sa hecnem a napíšem aj Strassena, ale nehecli ste sa, tak bude možno lepší jednoduchší vzorák. Existujú aj lepšie algoritmy, o ktorých sa vie (všimnite si ten trpný rod – ja to nie som), napr. taký Coppersmithov a Winogradov algoritmus, ktorý beží v čase $O(n^{2.38})$. Pre „riedke“ matice existujú rýchlejšie algoritmy, ktoré majú určite tiež svoju teoretickú hodnotu (môžeme biť do pŕs, že sa približujeme k tomu $O(n^2)$)... Dosť bolo blábolu, hodnotil som nasledovne:

- backtrack v $O(N^K)$ – 6 bodov
- maticami v $\Theta(N^3K)$ – 9 bodov
- dynamika v $\Theta(MK)$ – 12 bodov
- maticami v $\Theta(N^3 \log K)$ – 15 bodov
- maticami v $o(N^3 \log K)$ (t.j. lepšie) – viac, ale nevyskytlo sa

Body som drsne stŕhal za odhad časovej zložitosti – bodík za čas, bodík za pamäť.

Listing programu:

```
#include <cstdio>
#define MAX 100

long a[MAX][MAX], // matica susednosti
     r[MAX][MAX], // vysledok
   pom[MAX][MAX]; // pomocne pole pre sucin
int n, m, k, x, y;
long s=0;

int mul (long c[][MAX], long a[][MAX], long b[][MAX]) { // vypocita c <- a*b
  for (int i=0; i<n; i++)
    for (int j=0; j<n; j++) {
      pom[i][j] = 0;
      for (int k=0; k<n; k++) pom[i][j] += a[i][k]*b[k][j];
    }
  for (int i=0; i<n; i++) for (int j=0; j<n; j++) c[i][j] = pom[i][j];
```

[6] alebo, ako hovorí Dávidko, command & conquer

[7] teraz nemám na mysli Fourierovú transformáciu, ktorou to vieme spraviť rýchlejšie ;-)

```
}

int main () {
  scanf ("%d %d", &n, &k);
  for (int i=0; i<n; i++) for (int j=0; j<n; j++) {
    a[i][j] = 0;              // vynulujeme a
    r[i][j] = (i==j) ? 1 : 0; // r bude jednotkova matica, t.j. bude mat jednotky
  }                           // iba na diagonale, inde nuly
  while (scanf ("%d %d", &x, &y) == 2) a[x][y] = 1; // nacitame maticu susednosti

  while (k>0) {
    if (k%2) mul (r, r, a); // r <- r*a
    mul (a, a, a);          // a <- a*a
    k /= 2;
  }

  for (int i=0; i<n; i++) for (int j=0; j<n; j++) s += r[i][j]; // spocitame sledy
  printf ("Existuje plus-minus %ld sledov\n", s);

  return 0;
}
```

3. O mrakodrapoch

opravoval Lukáš
(max. 15 bodov)

Najprv si ukážeme riešenie podobnej, ale ľahšej úlohy. Namiesto partície čísla máme nájsť k-tu permutáciu čísel 1 až n. Ak by bolo prvé číslo 1, k by muselo byť menšie alebo rovné $(n-1)!$, lebo práve toľko je permutácií tvaru $1, a_1, \ldots, a_{n-1}$. Ak by malo byť prvé číslo 2, tak platí $(n-1)! < k \leq 2(n-1)!$. Všetky permutácie začínajúce jednotkou sú totiž lexikograficky menšie, ako permutácie začínajúce dvojkou. Takto sa vieme dostať až k hľadanému prvému číslu permutácie. Rovnako vieme určiť aj ďalšie čísla permutácie, len si musíme pamätať čísla, ktoré sme už použili.

Podobný spôsob generovania využijeme aj my. Pred hľadaním postupnosti však odpočítame od m číslo $n \cdot 1$, teda pre každý mrakodrap odpočítame jednotku. Vieme, že každý mrakodrap musí mať nenulovú dĺžku. Týmto odčítaním znížime výšku každého o jedna a pripustíme tak aj nulovú výšku, čo nám neskôr uľahčí výpočty. Náš postup bude kopírovať hľadanie k-tej permutácie. Postupne budeme zľava doprava zisťovať cifry, ale na hľadanie nemôžeme použiť tak jednoduché číslo ako $n!$. Ak dáme na prvé miesto jednotku, preskočíme všetky postupnosti, ktoré sa začínajú nulou. Ale koľko je takých, čo sa začínajú nulou?

Na výpočet použijeme dynamické programovanie. Označme $p(n, m)$ počet rozostavení m kociek do n mrakodrapov, pričom zľava doprava tvoria výšky mrakodrapov neklesajúcu postupnosť. Ak bude mať prvý mrakodrap výšku 0, tak takýchto rozostavení bude rovnako veľa ako $p(n-1, m)$. Ak bude prvý mrakodrap vyšší ako 0, možných rozostavení bude $p(n, m-n)$. Totiž ak každý mrakodrap znížime o 1, dostaneme menší problém s $m-n$ kockami a n mrakodrapmi. Znížiť mrakodrapy môžeme vďaka neklesajúcosti postupnosti a výške prvého mrakodrapu.

Teraz už vieme zostrojiť algoritmus na hľadanie všetkých hodnôt $p(n, m)$. Pre $n = 1$ triviálne platí $p(1, m) = 1$. Inak $p(n, m) = p(n-1, m) + p(n, m-n)$. Výsledky si budeme pamätať v dvojrozmernom poli a výpočet spravíme pre všetky možné n a m, takže dokopy dostávame časovú aj pamäťovú zložitosť $O(NM)$.

Pokračujeme hľadaním výslednej postupnosti. Ak by bola prvá nula, muselo by platiť $k \leq p(n-1, m)$, teda k je menšie ako počet rozostavení dĺžky $n-1$ z m kociek. Ak by bola prvá jednotka, platilo by $p(n-1, m) < k \leq p(n-1, m-n) + p(n-1, m)$. Výraz na pravej

strane zodpovedá počtu postupností začínajúcich jednotkou alebo nulou. Pre hľadané číslo i, ktoré dáme na prvé miesto postupnosti, bude platiť

$$\sum_{j=0}^{i-1} p(n-1, m-n\cdot j) < k \leq \sum_{j=0}^{i} p(n-1, m-n\cdot j)$$

Získané číslo i dáme na prvé miesto postupnosti. Každý ďalší mrakodrap bude mať výšku minimálne i, preto môžeme problém redukovať na $n-1$ mrakodrapov a $m-n\cdot i$ kociek. Samozrejme, ku každému ďalšiemu mrakodrapu musíme neskôr pripočítať hodnotu i, ktorú sme teraz odčítali. Zložitosť tohto hľadania bude v najhoršom prípade $O(M)$, čo je však o dosť menej, ako počítanie rozostavení.

Výsledná časová a pamäťová zložitosť je teda $O(NM)$ a jedine za ňu ste mohli získať plných 15 bodov. Pochvala patrí Vladimírovi Schusterovi, lebo iba on mal optimálnu zložitosť. Ostatné riešenia boli v čase $O(NK)$, za tie ste mohli dostať 9 bodov. K môže totiž byť veľmi veľké číslo a jeho veľkosť závisí rádovo exponenciálne od M a N.

Listing programu:

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

int main() {
   int m, n, k; cin>>m>>n>>k;
   int p[100][100];
   fill(p[1], &p[1][m], 1);
   for (int i=2; i<n; i++) for (int j=0; j<=m; j++) {
      p[i][j]=p[i-1][j];
      if (j>=i) p[i][j]+=p[i][j-i];
   }

   int s=1;//výška aktuálneho mrakodrapu
   m-=n;
   for (int i=n; i>1; i--) for (int j=0; j<=m/i; j++) {
      //nájdeme vyhovujúce j
      if (p[i-1][m-i*j]>=k) {
         m-=i*j; s+=j;
         cout<<s<<" ";
         break;
      }
      else k-=p[i-1][m-i*j];
   }
   //vypíšeme zvyšok
   cout<<s+m<<endl;
}
```

4. Opäť v 47. hyperpriestorovej dimenzii

opravoval Tom
(max. 15 bodov)

Tento príklad podľa mňa nebol moc ťažký, ale na počte vašich riešení sa to neodrazilo. A ani tie, čo prišli, neboli všetky funkčné.

Poďme sa pozrieť na bodovanie. To bolo jednoduché. Bolo treba zistiť tri veci, teda za každú sa dalo získať niekoľko bodov. Za zistenie, či je niektorá parcela prázdna sa dalo získať 0 bodov (za nefunkčné riešenie), alebo 3 body (za funkčné riešenie). Za zistenie, či sa nejaké

dve parcely prekrývajú, sa dalo získať 0 (za nefunkčné), 3 (za lineárny čas, ale v pamäti 10^{17}), alebo 6 (za kvadratický čas s malou konštantou) bodov. No a za zistenie, či tie parcely pokrývajú celý priestor ste mohli získať 0, 3 (tiež za konštantu 10^{17}), alebo 6 bodov. Body za jednotlivé časti sa sčítali a takto získaná hodnota sa vynásobila koeficientom $(\alpha + \beta)$, kde α (v rozsahu 0 až 0.6) bolo za popis a β (v rozsahu 0 až 0.4) bolo za implementáciu. Výsledok je počet bodov za vaše riešenie.

Predtým, než si ukážeme vzorové riešenie, musím povedať niečo tým, ktorí vo svojich riešeniach používali 17-rozmerné polia a podobné veci. KSP je síce teoretická súťaž, ale 10^{17} je veľa. Skúšali ste si tie programy aj spustiť? Veď toľko pamäti pokope ešte pekných pár rokov neuvidíte. A ďalší dôvod, prečo takéto riešenia dostali málo bodov je, že ste si pamätali celý priestor. Takýto prístup nie je vždy najvhodnejší a v tomto prípade podľa mňa ani nebol. Veď ten rozsah mohol byť kľudne aj trebárs 3000, alebo to mohli byť racionálne čísla (hoci len do 10). V takom prípade by už vaše riešenia také efektívne neboli.

Dosť bolo veľkých polí, poďme sa teraz pozrieť na vzorové riešenie. Najprv si všimnime, ako taká hyperparcela vyzerá. Každá hyperparcela je vlastne hyperkváder, čiže množina (bodov) určená 17-timi nerovnicami tvaru $x_L \leq x \leq x_H$ (kde x je a, b, ..., q). Jednoduchý dôkaz nechávam ako cvičenie pre čitateľa. Načítanie vstupu bude teda jednoduché. Po načítaní podmienky napr. $x <= x_P$ si x_H upravíme na $\min(x_H, x_P)$. Podobným spôsobom spracujeme aj ostatné podmienky. No a toto celé urobíme pre každú parcelu.

No a ako teraz zistíme, či je parcela neprázdna? Nuž jednoducho. Parcela je totiž prázdna práve vtedy, keď pre nejakú os x platí $x_L > x_H$. Toto sa dá pre každú parcelu overiť v lineárnom čase.

Či sa nejaké dve parcely prekrývajú overíme tiež jednoducho. Stačí si uvedomiť, že to nastane práve vtedy, keď sa prekrývajú ich priemety do každej osi. Teda keď pre každú os x platí $x_H^{(1)} \geq x_L^{(2)}$ a súčasne $x_H^{(2)} \geq x_L^{(1)}$, pričom prvá parcela má horný index (1) a druhá (2). Toto vieme pre každú dvojicu parciel overiť v kvadratickom čase.

Zostáva už len overiť, či parcely pokrývajú celý hyperkváder. Tu bolo kľúčové uvedomiť si, že toto sa testuje, len ak sa tie parcely neprekrývajú. V takomto prípade totiž vypĺňajú celý hyperkváder ak súčet ich objemov sa rovná objemu celého hyperkvádra. No a toto sa tiež dá jednoducho overiť v lineárnom čase.

Celková časová zložitosť vzorového riešenia je teda $O(N^2)$ a pamäťová je $O(N)$.

Listing programu:

```
#include <stdio.h>

#define MAXN 147

int L[MAXN][17], H[MAXN][17];  //rozsahy parciel
int N;  //pocet parciel

int plne()
{
  int i,j;
  for(i=0;i<N;i++)for(j=0;j<17;j++)
    if(L[i][j]>H[i][j])
    {
      printf("Parcela %d je prazdna !!!\n",i+1); //ak sa najde parcela a os na ktorej
      return 0;                                  //je ta parcela prazdna tak je zle
    }
  return 1;
}

int prekryv2(int i,int j)
{
```

```
  int k;
  for(k=0;k<17;k++)if((H[i][k]<L[j][k])||(H[j][k]<L[i][k]))return 0;
    //ak sa najde os na ktorej sa neprekryvaju sa sa neprekryvaju vobec
  return 1;
}

int prekryv()
{
  int i,j;
  for(i=0;i<N;i++)for(j=i+1;j<N;j++)if(prekryv2(i,j))
  {
    printf("Parcely %d a %d sa prekryvaju !!!\n",i+1,j+1);
    return 1;   //ak sa najdu 2 parcely co sa prekryvaju tak je zle
  }
  return 0;
}

int vyplnaju()
{
  long long objem=505447028499293771;  //11^17 = objem celeho hyperkvadra
  long long ob;
  int i,k;
  for(i=0;i<N;i++)
  {
    ob=1;
    for(k=0;k<17;k++)ob*=H[i][k]-L[i][k]+1;
    objem-=ob;   //odratame objem kazdej parcely
  }
  if(objem!=0)printf("Nekompletne pokrytie !!!\n");
  return objem==0;  //ak to vyslo presne tak je dobre
}

void nacitaj()
{
  int i,j,k,r;
  char S[7];
  char x;
  scanf("%d ",&N);  //pocet parciel
  for(i=0;i<N;i++)
  {
    for(k=0;k<17;k++){L[i][k]=0; H[i][k]=10;}
    scanf("%d ",&j);  //pocet podmienok na parcelu
    while(j--)
    {
      scanf("%c %s %d ",&x,S,&r);
      k=x-'a';
      switch(S[0])  //podla znamienka ...
      {
        case '=':
          if(H[i][k]>r)H[i][k]=r;
          if(L[i][k]<r)L[i][k]=r;
          break;
        case '<':
          if(S[1]!='=')r--;
          if(H[i][k]>r)H[i][k]=r;
          break;
        case '>':
          if(S[1]!='=')r++;
          if(L[i][k]<r)L[i][k]=r;
          break;
```

```
      }
    }
  }
}

int main()
{
  nacitaj();
  if(!plne())return 0;
  if(prekryv())return 0;
  if(!vyplnaju())return 0;
  printf("OK\n");
  return 0;
}
```

5. Opatrne s XORovaním!

opravoval MišoF.
(max. 20 bodov)

Úloha a)

Už v zadaní sme si povedali, že xor (značíme ho $\oplus$) je komutatívny a že $\forall x;\ x \oplus x = 0$. Ešte potrebujeme ukázať, že je aj asociatívny. To sa dá rozobraním všetkých prípadov, prípadne nasledujúcou úvahou:

Zápis $a \oplus b$ nám vlastne hovorí: zober a a invertuj tie bity, ktoré má b nastavené. (Prípadne naopak, začni s b a zmeň mu bity v a.) Potom už je zjavné, že $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ – začneme s b a postupne mu zmeníme bity, ktoré sú jednotkové v a a c, pričom zjavne nezáleží na poradí, v akom to robíme.

Potom už môžeme dokázať, že Bob na konci naozaj dostane pôvodnú správu. Simuláciou postupu zo zadania zistíme, že výsledná správa je $V = (((M \oplus K_A) \oplus K_B) \oplus K_A) \oplus K_B$. Keďže $\oplus$ je asociatívna a komutatívna operácia, môžeme zátvorky vynechať a poradie operácií preusporiadať. Dostávame $V = M \oplus (K_A \oplus K_A) \oplus (K_B \oplus K_B) = M \oplus 0 \oplus 0 = M$, q.e.d.

Za túto časť sa dali získať dva body. Dostali ste ich, ak som vo vašom riešení stretol slovo „asociatívny“. Úvahy smerom „kuknem a vidím“ dostali bod.

Úloha b)

Majme teda výslednú správu S a rovnako dlhú správu M', z ktorej chceme S vyrobiť. Koľko je takých kľúčov, ktoré toto spôsobia?

Pre každý taký kľúč K musí platiť: $K \oplus M' = S$. Prexorujeme obe strany číslom M', dostávame $K = S \oplus M'$. Ale $S \oplus M'$ je jedna konkrétna hodnota. Preto pre kľúč K máme len túto jednu možnosť. Na druhej strane ľahko nahliadneme, že tento kľúč K naozaj robí to, čo má.

Iné možné riešenie bolo úvahou. Kľúč K nám vlastne hovorí, ktoré bity M' máme zmeniť, aby sme dostali S – a toto je zjavne jednoznačne určené tým, ako obe správy vyzerajú.

Za túto časť sa dali získať tri body. Bol som trochu prísnejší – ak z vami uvedených argumentov vyplývalo iba to, že počet vyhovujúcich kľúčov je ≥ 1 (prípadne ≤ 1), strhal som bod. Poučenie: Aj keď je niečo zjavné, občas nie je také ľahké poriadne to zapísať. Pozor na to.

Úloha c)

Ľahko si všimneme, v čom je problém. Nech sú tri poslané správy $S_1 = M \oplus A$, $S_2 = M \oplus A \oplus B$ a $S_3 = M \oplus B$. Potom zjavne $S_1 \oplus S_2 \oplus S_3 = M \oplus A \oplus M \oplus A \oplus B \oplus M \oplus B = M$. Takže stačí všetky tri správy prexorovať a máme riešenie, „Strc prst skrz krk!“.

Fasovalo sa po 4 body. Prémiové 2 body získal Matúš Fedák za nasledujúce pozorovanie: Všimnime si, že prexorovaním S_1 a S_2 sa poštárka dozvie kľúč K_B. Ten teraz môže použiť na to, aby tretiu Alicinu správu nahradila svojou, ktorú Bobovi podstrčí.

Za domácu úlohu si rozmyslite, že chudák Bob nemá tento podvod ako odhaliť. Na podobné útoky sa poriadnejšie pozrieme v štvrtej sérii, máte sa na čo tešiť.

Úloha d)

Snáď všetci ste prišli na nasledujúcu myšlienku: Smith po ceste odchytí Quileniovu[8] zamknutú truhlicu, umiestni na ňu svoj zámok a pošle ju späť. Nič netušiaci Quilenius dá dole svoj zámok, opäť pošle truhlicu a Smith si ju odchytí a hravo ju otvorí, lebo už je na nej len jeho zámok. Medzičasom môže Smith urobiť (s inou truhlicou) prvé dva kroky komunikácie s Jonesom, aby tomu nič neprišlo podozrivé.

Dalo sa dostať päť bodov, prípadne štyri, ak ste úplne odignorovali Jonesa.

Uvedomte si, že podobný problém by vznikol vždy – aj keby sme takto používali šifry iné ako xor, ktorého slabinu sme si ukázali v úlohe c). Aktívny útočník by sa vždy mohol vložiť doprostred komunikácie a oklamať oboch jej účastníkov. Tomuto sa odborne hovorí *man in the middle* útok.

Existujú aj spôsoby, ako mu zabrániť, ale to by bolo na dlhé rozprávanie. Snáď najjednoduchší: Vždy, keď Alici alebo Bobovi príde správa, zdvihne telefón, zavolá tomu druhému a porovnajú si jej hashovaciu hodnotu. (Je samozrejme jasné, že príliš praktický tento spôsob nie je, ale aspoň funguje.)

Úloha e)

Bez výnimky všetky úspešné útoky boli ručné. Začnime tým, že si uvedomíme, že máme dve správy: $A \oplus K$ a $B \oplus K$. Ich prexorovaním dostaneme správu $A \oplus B$. Zbavili sme sa teda úplne kľúča K. Ostáva nám uhádnuť správy A, B. Tu je stále veľa možností, no pomôže, že vieme, že obsahujú zmysluplné slovenské slová.

Všimneme si, že xorom dvoch písmeniek dostávame relatívne malú hodnotu, zatiaľ čo xorom písmenka a medzery hodnotu väčšiu. Takto vieme zhruba odhadnúť, kde sú v niektorej zo správ medzery. Vidíme teda, že jedna začína štvorpísmenovým slovom. Po uhádnutí „Ahoj␣“ dostávame v druhej správe „Absol“, čo nie je „Absolvent␣“, ale „Absolutne␣“. V prvej správe to vedie k „Ahoj moja␣“ a veselo pokračujeme ďalej. Výsledné správy:

„`Ahoj moja najdrahsia, toto nikto nerozlusti.`“

„`Absolutne bezpecna sifra odola vasim utokom!`“

Riešenie nie je jednoznačné, napr. nevieme určiť, ktoré písmená sú veľké, aj posledný znak sme si len tipli, mohli byť napr. aj naopak – výkričník v prvej vete, bodka v druhej. Napriek tomu sa správa dala zrekonštruovať snáď až príliš ľahko :) Poučenie: Xor dvoch otvorených textov v sebe obsahuje viac informácie, než by nám bolo milé.

Za vyriešenie + postup sa dalo získať 5 bodov, za zostrojenie $A \oplus B$ dva body.

Ešte k automatickej metóde lámania takejto „šifry“ – keď si niekde tipneme slovo (a dve medzery okolo neho), vieme dopočítať zodpovedajúcu časť druhého slova a následne hľadať v slovníku slová, ktoré do nej pasujú.

Prekvapujúco účinný môže byť aj nasledujúci postup: Spravíme si pomocou nejakého dlhého textu štatistiku počtu výskytov *tetragramov* (štvoríc po sebe idúcich písmen). Pomocou tejto štatistiky sa dá o danom reťazci povedať, nakoľko „znie po slovensky“. Začneme s ľubovoľnou nezmyselnou prvou správou, postupne ju budeme skúšať meniť tak, aby obe výsledné správy zneli „čo najviac slovensky“ a vypisovať si priebežne najlepšie dosiahnuté riešenie.

(Podobná metóda sa dá napr. použiť aj na automatické riešenie substitučných šifier, je oveľa účinnejšia ako „tradičné slovníkové“ metódy – tie totiž majú problém, akonáhle je v šifre veľa slov, ktoré v slovníku nemáme.)

[8] Takto sa to skloňuje, žiadne „Quileniusovu“ :)

Výsledková listina po 1. kole kategórie KSP

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
1	Mikuláš Ján	Gym. Haličská Lučenec	3	15	12	9	15	16	67
2	Sudolský Michal	Gym. Tajovského B. Bystrica	2	11	11	8	15	18	63
3	Fedák Matúš	Gym. Stará Ľubovňa	3	15	12	9	6	19	61
4	Imriška Jakub	Gym. Jura Hronca BA	3	15		13	15	8	51
5	Jerguš Ján	Gym. Alejová Košice	2	11	5	9	8	17	50
6	Bundala Daniel	Gym. Jura Hronca BA	3	15	4	2	15	11	47
6	Petruchová Zuzana	Gym. Grösslingová BA	3	12	4	9	9	13	47
8	Herman Peter	Gym. Jura Hronca BA	2	11	9	8		16	44
9	Okruhlica Adam	Gym. Jura Hronca BA	2	11	9	2	8	13	43
10	Bílka Ondřej	Gymnázium	3	7	13	10	2	7	39
11	Rampášek Ladislav	Gym. Jura Hronca BA	2	11	10	2		10	33
12	Pančík Andrej	Gym. Tajovského B. Bystrica	2	11	4	2		12	29
13	Krchniak Matej	Gym. Jura Hronca BA	3	10		2	2	11	25
14	Bodnár Jozef	Gym. Fiľakovo	4	8				14	22
15	Ďuďák Juraj	Gym. Golianova Nitra	3	8		2		11	21
16	Ďurfina Lukáš	Gym. Golianova Nitra	4	11				4	15
16	Schuster Vladimír	Gym. Jura Hronca BA	1			15			15
18	Zaťko Maroš	Gym. Ľudovíta Štúra Trenčín	1					7	7