Korešpondenčný seminár z programovania
XXVII. ročník, 2009/10
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporujú: MICROSTEP-MIS spol. s r.o.
Agentúra na podporu výskumu a vývoja

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 24 stranách. Užite si ich :)

Matika sa neučí, matika sa chápe.

KSPáci

1. Zmenená endianita

opravoval JMK a Feki
(max. 10 bodov)

Úvodom si pripomeňme, že vstup už máme načítaný a nerátame ho do pamäťovej zložitosti. Ak by bol vstup N, potom by pamäťové nároky na jeho uloženie boli $O(\log N)$, pretože počet cifier čísla v desiatkovom zápise je približne rovný jeho dekadickému logaritmu[1]. Kebyže do výslednej zložitosti rátame aj premennú na výstup, potom by sme opäť automaticky dostali pamäťové nároky algoritmu minimálne $O(\log N)$, pretože úplne všetky algoritmy produkujú daný výstup. Ak premennú na výstup nebudeme počítať do pamäťovej zložitosti, dostaneme odhad, ktorý sa bude týkať čisto činnosti algoritmu ako takého a preto môže byť potenciálne menší ako $O(\log N)$. Takto budeme môcť odlíšiť lepší algoritmus od horšieho.

Máme teda číslo v desiatkovej sústave. Jedna z jej vlastností je, že je pozičná, teda hodnota (významnosť) danej cifry závisí od jej pozície v čísle. Konkrétne hodnota i-tej číslice sprava je 10^i. Číslo, ktoré chceme získať, má mať tieto hodnoty pre dané cifry vymenené. Teda prvá číslica sprava má mať hodnotu prvej zľava atď. Najjednoduchší spôsob, ako poskladať výsledné číslo, by bol postupne ho rekonštruovať po cifrách – vziať číslicu a vynásobiť ju k-krát číslom 10, kde k je jej pozícia v novom čísle sprava. Takto realizovaný postup by mal ale vysokú časovú zložitosť. Najlepšie by bolo získavať cifry nie po jednej, ale násobiť ich nejak spolu.

Vezmime prvú cifru sprava a vynásobme ju desiatimi. Aby sme ju dostali na správne miesto vo výslednom čísle, ešte ju potrebujeme vynásobiť toľkokrát, koľko násobení potrebujeme na presunutie druhej číslice. Môžeme teda túto druhú cifru pripočítať k násobku prvej cifry desiatimi a násobiť tieto dve čísla spolu. Po ďalšom vynásobení desiatimi pripočítame tretiu cifru, po ďalšom štvrtú a takto pokračujeme, až kým nepripočítame poslednú cifru pôvodného čísla. Takto budeme postupne dostávať výsledné číslo od najvýznamnejších cifier.

Na tento algoritmus by sme ale potrebovali rýchlo získavať jednotlivé cifry pôvodného čísla postupne v poradí od najmenej významnej. Tie budeme získavať opačným postupom k tomu, ktorým sme vytvárali nové číslo. Všimnime si, že keď číslo vydelíme hodnotou 10^s, tak zvyšok po delení desiatimi tohto predeleného čísla bude práve s-tá cifra sprava, ak by sme

[1] Logaritmus čísla N so základom 10 je exponent, na ktorý musíme umocniť desiatku, aby sme dostali N. Platí teda: $\log x = y$ vtedy, keď $10^y = x$.

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

tieto cifry číslovali od nuly. Tieto číslice teda môžeme získať tak, že ho budeme postupne deliť desiatimi a všímať si zvyšky po delení desiatimi (to je len inými slovami povedaná posledná cifra).

Pokiaľ by sme si najprv vytvorili všetky cifry a až potom ich začali násobiť, náš algoritmus by potreboval pamäť $O(\log N)$, čo je zbytočne veľa. Keďže ich nepotrebujeme všetky naraz, tak ich budeme vytvárať postupne až vtedy, keď ich budeme potrebovať. Jeden krok výsledného algoritmu bude teda pozostávať zo získania zvyšku po delení vstupného čísla desiatimi, jeho predelení desiatimi, pripočítaní zvyšku k výslednému číslu a ak nejde o posledný krok, tak aj jeho vynásobenie desiatimi.

Tento krok vykonáme $(\log N)$-krát, časová zložitosť je teda $O(\log N)$, kde N je dané číslo. V pamäti máme uložených niekoľko premenných, ktorých veľkosť je konštantná (počet bajtov, ktoré si musíme pamätať, nezávisí od vstupného čísla ani od dĺžky jeho zápisu), pamäť je teda $O(1)$.

Bodovalo sa takto:

- optimálne riešenie s dobrým popisom – 10 bodov,
- čas $O(\log N)$, pamäť $O(\log N)$ s dobrým popisom – 9 bodov,
- čas $O(\log^2 N)$, pamäť $O(1)$ s dobrým popisom – 8 bodov,
- čas $O(\log^2 N)$, pamäť $O(\log N)$ s dobrým popisom – 7 bodov,
- programy, ktoré načítali reťazec a vypisovali ho odzadu dostali aj s dobrým popisom málo bodov,
- odhady boli za bod,
- za chýbajúci popis bola polovica bodov dole,
- za chyby v kóde, ak ich bolo viac, a za slabšie popisy sa dávalo primerane menej bodov.

PS: keďže tento príklad je len jednotka, bol hodnotený mierne. Nabudúce si musíte dať pozor na odhady zložitosti a popis algoritmu, pretože pri vyšších úlohách sú postihy za tieto nedostatky citeľnejšie.

Listing programu:

```
var pom, cislo: integer;

begin
  readln(cislo);
  pom := 0;
  while cislo > 0 do begin
    pom := pom * 10;
    pom := pom + (cislo mod 10);
    cislo := cislo div 10;
  end;
  cislo := pom;
  writeln(cislo);
end.
```

2. Zo schodiska bude panelák!

opravoval Tommy, vzorák písal Imp
(max. 10 bodov)

Na začiatok si uvedomme, že sa od nás nepožaduje nič ohľadom počtu poschodí paneláku, iba to, aby boli aspoň dve. A všade tam, kde sa dá postaviť trojposchodový či viacposchodový panelák, dá sa postaviť aj dvojposchodový (ak existuje postupnosť p, $p + K$, $p + 2K$, ..., potom existuje aj dvojprvková postupnosť p, $p+K$). Nič teda nepokazíme, ak budeme hľadať

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

len výšku, v ktorej sa dá postaviť dvojposchodový panelák. Máme však nájsť čo najmenšiu takú výšku. To znamená najmenšie také p, že v zadanej postupnosti je aj číslo $p + K$.

Riešenia v $O(N^2)$ a $O(N \log N)$

Toto riešenie je najjednoduchšie na napísanie a zároveň najmenej efektívne. Zadanú postupnosť prejdeme dvoma do seba vnorenými cyklami tak, že vonkajším si postupne určujeme jednotlivé prvky ako p a vnútorným hľadáme v poli číslo $p + K$. Oba cykly prechádzajú rádovo N prvkov, preto má toto riešenie časovú zložitosť $O(N^2)$. To sa dá zlepšiť tak, že číslo $p + K$ nebudeme hľadať ďalším cyklom, ale binárnym vyhľadávaním. Pozrieme sa na prvok v strede poľa. Ak má väčšiu hodnotu ako hľadáme, hľadaný prvok môže byť len naľavo od neho, ak má menšiu, hľadaný prvok môže byť len napravo (keďže prvky v poli tvoria neklesajúcu postupnosť). Tou polovicou poľa, o ktorej už vieme, že tam hľadaný prvok nemôže byť, sa ďalej nezaoberáme a postup opakujeme na tej druhej. Hľadanie ukončíme, ak prvok nájdeme, alebo sme ho nenašli a skúmaný úsek poľa už má dĺžku len 1, teda ho nevieme ďalej rozdeliť. Binárne vyhľadávanie má v poli veľkosti N časovú zložitosť $O(\log N)$ a spúšťame ho N-krát (pre všetky možné p), preto výsledná časová zložitosť je $O(N \log N)$.

Optimálne riešenie v $O(N)$

Na napísanie optimálneho riešenia si musíme uvedomiť, že predchádzajúce metódy mnohé dvojice porovnávajú zbytočne. Ak totiž máme dva prvky a, b také, že $b - a > K$, potom žiaden z prvkov postupnosti naľavo od a určite netvorí riešenie so žiadnym prvkom napravo od b. Využijúc tento poznatok nám stačí zadanú postupnosť len lineárne prehľadať. Na prechádzanie postupnosti použijeme dva ukazovatele[2]. Prvý ukazovateľ nastavíme na prvý prvok postupnosti a druhým ukazovateľom prehľadávame ďalšie prvky len dovtedy, kým je rozdiel menší alebo rovný K. Potom posunieme prvý ukazovateľ o jeden prvok ďalej a s druhým pokračujeme (rozmyslite si, že sa s ním nemusíme vracať naspäť) opäť dovtedy, pokiaľ je rozdiel menší alebo rovný číslu K. Tento postup opakujeme, kým sa nedostaneme druhým ukazovateľom až za koniec postupnosti (vtedy riešenie neexistuje), alebo kým nenastane situácia, že rozdiel hodnôt pod ukazovateľmi je presne K (vtedy sme našli riešenie; všimnite si, že prvé riešenie, ktoré nájdeme pri prehľadávaní zľava doprava, zodpovedá najnižšie postavenému paneláku). Keďže oba ukazovatele prejdú postupnosťou len raz, časová zložitosť tohto algoritmu je $O(N)$. Inými slovami, v každom momente spravíme s nejakým ukazovateľom krok a tých krokov je dokopy najviac $2N$.

Za optimálne riešenie, t.j. v $O(N)$, sa dalo získať 10 bodov, za $O(N \log N)$ 9 bodov a za jednoduché kvadratické $O(N^2)$ riešenie 7 bodov. Bolo si treba uvedomiť, že výška najvyššieho schodu môže byť *veľmi* veľká, a teda robiť niečo s poľom takejto veľkosti môže byť preto veľmi pomalé. Ďalšie body sa dali stratiť za zlý, chýbajúci či žiadny popis, prípadne odhady. Našlo sa aj väčšie množstvo drobných chýb v programoch, a niekto dokonca i opisoval (za to sa, podľa pravidiel, body rozdeľovali).

Listing programu:

```
program panelak;
const MAXN = 1000;
var schody: array [1..MAXN] of integer;
    N, K, i, j: integer;
begin
  Read(N, K);
  for i := 1 to N do
    Read(schody[i]);
  j := 1;
```

[2]Niekedy sa tomu hovorí aj *metóda dvoch bežcov*.

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
  for i := 1 to N do
    while schody[j] - schody[i] <= K do begin
      if schody[j] - schody[i] = K then begin
        WriteLn(i);
        Exit;
      end;
      if j = N then begin
        WriteLn('Panelak sa neda postavit.');
        Exit;
      end;
      j := j + 1;
    end;
end.
```

Opravovalo sa samo, vzorák písal Stano
(max. 10 bodov)

3. Zabudnutá kalkulačka

Tento príklad nebol tažký, čomu nasvedčovalo aj veľké množstvo riešení za plný počet bodov. Ako sa dal riešiť? Na optimálne riešenie ste potrebovali iba konštantné množstvo pamäte a spracovávať vstup postupne, tak ako prichádza na vstup. Vždy, keď príde nová číslica, tak si zmeníme pomocnú premennú, aby sme v nej mali aktuálne zapísané číslo. A ak nám príde nová operácia, tak už máme celé číslo, ktoré sme načítavali. Teraz môžeme spočítať predchádzajúcu operáciu a zapamätať si tú novú.

Rozoberme si podrobnejšie 3 prípady, ktoré nastanú pri spracovávaní:

1. Zo vstupu sme prečítali číslicu – jediné čo treba, je vynásobiť predtým zapamätané číslo 10-tkou a pripočítať k nemu novú číslicu.
2. Zo vstupu sme prečítali operáciu. Ak to bola prvá, tak si iba zapamätáme, že medzivýsledok je predchádzajúce načítané číslo, zapamätáme si operáciu a pokračujeme. Ak to bola neskoršia operácia, tak vykonáme tú predchádzajúcu, zapíšeme si medzivýsledok a aj novú operáciu.
3. Prečítali sme znak =. V podstate je to rovnaká situácia ako v bode 2, vyhodnotíme poslednú operáciu, vypíšeme výsledok a môžeme skončiť.

Listing programu:

```
var c, operacia: char;
    cislo, medzivysledok: integer;

begin
  medzivysledok := 0;
  cislo := 0;
  operacia := 'n';
  repeat
    read(c);
    if (ord(c) >= ord('0')) and (ord(c) <= ord('9'))  then begin
      cislo := cislo * 10 + ord(c) - ord('0');
      continue;
    end;

    case operacia of
      '+': medzivysledok := medzivysledok + cislo;
      '-': medzivysledok := medzivysledok - cislo;
      '*': medzivysledok := medzivysledok * cislo;
```


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
      '/': medzivysledok := medzivysledok div cislo;
      'n': medzivysledok := cislo;
    end;
    operacia := c;
    cislo := 0;
  until c = '=';
  writeln(medzivysledok);
end.
```

Opravovala Dominika

4. Zostroj ten string!

(max. 15 bodov)

Tento príklad nebol tažký, preto som strhávala body aj za menšie nedostatky, horšie časové zložitosti a samozrejme, ak vám chýbal odhad zložitostí. Správne riešenie má konštantnú časovú aj pamäťovú zložitosť $O(1)$ (ak zanedbáme načítavanie vstupu, ktorý nakoniec ani nepotrebujeme).

Ak na vstupe dostaneme všetky možné reťazce dĺžky m (teda platí $n = 2^m$), tak určite neexistuje reťazec, ktorý by sa zhodoval so všetkými vstupnými reťazcami aspoň v jednom bite. Použijeme dôkaz sporom: povedzme, že daný výsledný reťazec existuje. Potom k nemu vytvoríme tzv. opačný reťazec (vznikne bitovou negáciou pôvodného reťazca). Tento opačný reťazec sa nachádza medzi vstupnými reťazcami (keďže na vstupe sú všetky možnosti) a zároveň nemá žiaden spoločný bit s výsledným reťazcom, pretože je jeho negáciou. Teda nastal *spor*, lebo všetky vstupné reťazce sa majú zhodovať v aspoň jednom bite s výsledným reťazcom. Teda ak platí $n = 2^m$, odpoveď bude „nie“.

Ak $n < 2^m$, tak stačí nájsť aspoň jeden reťazec, ktorý medzi vstupnými chýba a znegovať ho. Tento opačný reťazec sa musí zhodovať so všetkými reťazcami na vstupe v aspoň jednom bite. Toto platí, pretože existuje len jeden reťazec, s ktorým sa nezhoduje v žiadnom bite (jeho negácia), ale ten sa na vstupe nenachádza. Teda ak platí $n < 2^m$, tak odpoveď bude „ano“. Na toto riešenie prišla prevažná väčšina. Nad čím sa treba zamyslieť je, ako najrýchlejšie vyrobiť 2^m. Jedna možnosť je robiť to v cykle, ale to je dosť pomalé. Preto môžeme využiť bitový posun doľava (shift left), ktorý pracuje v $O(1)$ a hodí nám presne 2^m.

Ak niekto o tomto ešte nepočul, tak si vysvetlíme ako to funguje: shift left pracuje v dvojkovej sústave. Ak hodíme shift left na nejaké číslo, tak tento zoberie všetky bity, posunie ich doľava a dá za ne na najpravejšie miesto nulu (najľavejší bit čísla, ktorý sa nám do daného rozsahu teraz už nezmestí, jednoducho zahodíme). V našom prípade chceme použiť shift left na číslo jedna a posun o m bitov (napr. 1 `shl` $5 = 100000_2 = 2^5_{10} = 32_{10}$). Teda shift left je v podstate násobenie dvojkou. Ak to aplikujeme na číslo 1 m-krát, dostaneme 2^m.

Listing programu:

```
program zostrojstring;
var m, n, i: integer;

begin
  readln(n, m);
  {pre správnosť vstupu načítame n stringov}
  for i := 1 to n do
    readln();
  if 1 shl m = n then
    writeln('nie')
  else
```


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
    writeln('ano');
end.
```

5. Opravovacia lotéria

Opravoval Mišo
(max. 15 bodov)

Lotéria je tu opäť s výsledkami a vysvetlením. Každý, kto napísal nepárne číslo väčšie ako 3, získal aspoň 1 bod. Prečo je to tak? Pozrime sa na funkciu `magia(a,b)`: pre `k` od 2 vyššie delíme `a` aj `b` číslom `k`. Robíme to, kým sa dá deliť bez zvyšku. Keď sa číslo `k` zvýši na nejaký svoj násobok, tak sa áčko ani béčko už nebudú deliť, lebo sa nedelili ani pri `k`. To je veľmi dôležitá idea, lebo z nej vyplýva, že nás zaujímajú len prvočíselné hodnoty `k`. Keď si teda predstavíme `a,b` ako súčin mocnín prvočísel, tak robíme to, že pre jednotlivé prvočísla znižujeme ich exponenty v rozklade oboch čísel. A to robíme, až kým nie je jeden z tých exponentov nulový. Ak ostane nenulový exponent v áčku, tak riadok `while a mod k = 0 do a := a div k;` ho vynuluje. Ak je nenulový exponent v béčku, tak tam ostane už naveky. Funkcia vráti `false` práve vtedy, keď sú všetky exponenty nulové. To znamená, že v áčku bol pri každom prvočísle väčší alebo rovný exponent. A tomuto javu hovoríme, že `b` delí `a :-)`. Takže `magia(a,b) = not(b delí a)`.

A čo robí funkcia `odpoved(n)`? Pre premennú `skusam` od `n` do 2 skúša, či nedelí `n+1`. V premennej `vysledok` bude uložené posledné (najmenšie) také číslo, ktoré nedelí `n+1`. Keď ste si overovali svoje tipy, tak dobré zlepšenie bolo otočiť tento cyklus, aby sa skúšalo od 2 vyššie, či je to deliteľ daného čísla.

Počet bodov je `odpoved(N-1) div 2`. Na 15 bodov treba, aby `odpoved` vrátila 30. Takže premenná `vysledok` má byť 31. Premenná `n` musí byť deliteľná číslami od 2 po 30. Ale na vstup do funkcie `odpoved` dávame `N-1`, takže treba pripočítať 1. Správne číslo je napríklad $1 + NSN(2..30)$. Môžete porozmýšľať, prečo funkcia `odpoved` nemohla vrátiť číslo väčšie ako 30.

Príklad sa opravoval celkom jednoducho. Zistiť počet bodov sa dá tu:

http://people.ksp.sk/~miso/zadania/magia.php. K bodovaniu dodám akurát toľko, že ak ste zabudli pripočítať 1 a namiesto 15 bodov vám lotéria vyhodila 0, tak som trochu prižmúril oči a dostali ste nejaké body podľa kvality popisu. V tomto prípade ale zabudnite na hazard, lebo prehráte aj z koláča 4 diery.

6. O N mešcoch

opravoval Bob
(max. 20 bodov)

Niektorí z vás predpokladali, že sa Hermimu vždy oplatí zobrať ten väčší z krajných mešcov. Tento greedy prístup však nevedie k správnemu riešeniu. Napríklad pre dvoch hráčov a mešce $1, 1, 10, 2$ je optimálne zobrať ľavý mešec s hodnotou 1, lebo potom nám súper musí odkryť mešec s hodnotou 10.

Istotou je simulovanie všetkých spôsobov, ako môže hra prebiehať. Keď je na ťahu Hermi, má dve možnosti: zobrať ľavý alebo pravý mešec. Po ňom nasleduje $K-1$ ostatných KSPákov, tí si tiež vyberajú z dvoch možností. Pre Hermiho však nie je vôbec zaujímavé vedieť, ktorý zo súperov si zobral ktorý mešec. Podstatné je len to, ako bude úsek mešcov vyzerať, keď sa znova dostane na ťah Hermi. Týchto možností je K: ubudne 0 mešcov zľava a $K-1$ sprava, ubudne 1 mešec zľava a $K-2$ sprava, ..., ubudne $K-1$ mešcov zľava a 0 sprava. V každom prípade bude dĺžka úseku mešcov v ďalšom Hermiho ťahu o K menšia oproti aktuálnemu Hermiho ťahu.

http://www.ksp.sk/ksp2.0

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Naprogramovať sa to dá jednoducho pomocou rekurzie. Budeme mať funkciu $f(i,j)$, ktorá vráti najväčšiu možnú výhru, ktorú je Hermi schopný dosiahnuť, keď dostane vo svojom ťahu postupnosť mešcov dĺžky j začínajúcu i-tym mešcom. Hodnotu $f(i,j)$ vypočítame nasledovne: Hermi má k dispozícii mešce $i,\ldots,i+j-1$. Označme počet peňazí v a-tom mešci p_a. Keď si potiahne ľavý mešec, získa

$$L = p_i + \min(f(i+1,j-K), f(i+2,j-K), \ldots, f(i+K,j-K))$$

peňazí, lebo predpokladáme ten najhorší prípad, teda že mu súperi nechajú práve ten najmenej výnosný úsek. Keď si potiahne pravý mešec, získa

$$R = p_{i+j-1} + \min(f(i,j-K), f(i+1,j-K), \ldots, f(i+K-1,j-K))$$

peňazí. Z týchto dvoch možností vyberieme tú s vyššou výhrou, teda $f(i,j) = \max(L,R)$. Keď $j \leq K$, potom je Hermi vo svojom poslednom ťahu, oplatí sa mu zobrať väčší z krajných mešcov ($f(i,j) = \max(p_i, p_{i+j-1})$) a nemusí uvažovať nasledujúce ťahy súperov. Toto riešenie je ale pomalé. Jedno volanie funkcie f sa vetví na $2K$ ďalších volaní až do hĺbky rovnej počtu Hermiho ťahov, čo je $\lfloor \frac{N}{K} \rfloor$ (tento zápis znamená dolnú celú časť a je ekvivalentný výrazu N div K). V prvom kole sa teda rozvetvíme na $2K$ vetiev, tie sa v druhom kole rozvetvia na $(2K)^2$ vetiev a tak ďalej. Časová zložitosť je preto exponenciálna.

Keď si všimneme, že hodnotu $f(i,j)$ pre konkrétne i,j počítame viackrát, núka sa jednoduché zrýchlenie. Vždy, keď dopočítame $f(i,j)$, uložíme si jej hodnotu do dvojrozmerného poľa na pozíciu $M_{i,j}$. Nabudúce, keď ju budeme znova potrebovať, netreba ju rátať; stačí sa pozrieť do poľa na zapamätanú hodnotu. Tejto technike sa hovorí memoizácia. Počet rôznych parametrov funkcie f je rádovo $N \cdot \lfloor \frac{N}{K} \rfloor$ a pre konkrétne i,j trvá výpočet $O(K)$, preto je časová zložitosť tohto riešenia $O(N^2)$. Ak tomuto odhadu neveríte, tak si uvedomte, že keď vypočítame nejakú hodnotu $f(i,j)$ a uložíme si ju, potom sa všetky volania na hodnotu $f(i,j)$ budú riešiť už v konštantnom čase. Možno že sa budeme na túto hodnotu volať ešte veľakrát, ale každé toto zavolanie bude potrebné pre výpočet nejakej hodnoty f, ktorá je na hodnote $f(i,j)$ závislá. A každú hodnotu vyhodnocujeme presne raz, takže toto zavolanie bude súčasťou času potrebného na výpočet inej hodnoty funkcie f. Preto bude celková práca algoritmu naozaj kvadratická.

Dĺžky úsekov, ktoré môže Hermi počas hry dostať vo svojom ťahu, dávajú rovnaký zvyšok po delení číslom K, takže v M máme niektoré stĺpce[3] zbytočne (tie pre j s iným zvyškom po delení K). Definujme si preto nové pole D tak, že $D_{i,j}$ bude obsahovať najväčšiu možnú výhru, ktorú je Hermi schopný dosiahnuť, keď dostane vo svojom j-tom ťahu postupnosť mešcov začínajúcu i-tym mešcom. Všimnite si, že dĺžka postupnosti mešcov v j-tom Hermiho ťahu sa dá priamo vypočítať. Pretože na zistenie hodnoty $D_{i,j}$ potrebujeme poznať iba hodnoty pre väčšie j, budeme pole D vypĺňať „odzadu“. Pre posledný ťah je hľadaná hodnota maximum z najľavejšieho a najpravejšieho mešca. Keď už máme vypočítané hodnoty $D_{i,j}$ pre všetky (rozumné) i a pre konkrétne j, pokračujeme rátaním hodnôt pre $j-1$. Tento spôsob riešenia sa nazýva dynamické programovanie. Očividne je rovnako rýchly ako memoizácia; zmenilo sa len poradie, v akom chceme počítať jednotlivé hodnoty (ešte by sa dala ušetriť pamäť v porovnaní s $O(N^2)$ pamäťou pri memoizácii tak, že by sme si pamätali len aktuálny a predchádzajúci stĺpec).

No a nakoniec ešte jedno fintové zrýchlenie. Čítame si zdrojový kód dynamického programovania a spokojne sa usmievame nad jeho eleganciou. Ako sa tie cykly krásne cyklia... Ako sa hľadá to minimum... No ale, hej! Veď minimum (najhorší spôsob, ako môžu ťahať súperi) pre dva úseky mešcov rovnakej dĺžky so začiatkom posunutým o 1 sa počíta

[3] Alebo riadky. Záleží od uhla pohľadu.

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

viac-menej z tých istých hodnôt! Presnejšie, na výpočet $D_{i,j}$ chceme vedieť minimum z čísel $D_{i+1,j+1}, \ldots, D_{i+K,j+1}$ (keď berieme ľavý mešec) a z čísel $D_{i,j+1}, \ldots, D_{i+K-1,j+1}$ (keď berieme pravý mešec). Úsek $D_{i+1,j+1}, \ldots, D_{i+K-1,j+1}$ je spoločný, minimum z neho stačí vypočítať iba raz. Na výpočet $D_{i+1,j}$ potrebujeme opäť minimum z dvoch postupností, ktorých spoločným úsekom je $D_{i+2,j+1}, \ldots, D_{i+K,j+1}$. Teda spoločný úsek pre výpočet $D_{i+1,j}$ sa líši od spoločného úseku pre výpočet $D_{i,j}$ iba v tom, že v ňom už nie je $D_{i+1,j+1}$ a je v ňom navyše $D_{i+K,j+1}$.

Ako využiť toto pozorovanie? Budeme si udržiavať zoznam tých prvkov z aktuálneho spoločného úseku, ktoré sa ešte môžu stať minimom. Nazvime ich kandidáti. Pamätať si ich budeme v dátovej štruktúre, ktorá umožňuje vkladanie prvkov na koniec radu a odstraňovanie prvkov zo začiatku aj konca radu v konštantnom čase. Taká štruktúra sa dá ľahko implementovať napríklad pomocou spájaného zoznamu, v STL je na to ako stvorená `deque`. Kandidáti budú v tomto rade usporiadaní podľa ich indexu, teda kandidát $D_{a,j+1}$ bude bližšie k začiatku radu ako kandidát $D_{b,j+1}$ pre $a < b$.

Keď budeme počítať minimum spoločného úseku pre $D_{i,j}$ a na začiatku radu bude kandidát $D_{i,j+1}$, tak ho odstránime, pretože ten už do spoločného úseku nebude patriť. Na koniec radu chceme pridať nového kandidáta, $D_{i+K-1,j+1}$. Predtým však z konca radu povyhadzujeme všetkých tých kandidátov, ktorí sú väčší ako pridávaný kandidát. Môžeme to urobiť preto, že kandidáti s menším indexom vypadnú z radu skôr a po celý čas, kým budú v rade, bude pridávaný kandidát menší ako oni (takže už nebudú mať šancu stať sa minimom). Takto budú v rade naozaj iba tí kandidáti, ktorí môžu byť minimom. Všimnite si ale, že sú usporiadaní nielen podľa indexu, ale aj podľa hodnoty tak, že najmenší prvok je na začiatku radu (takéto usporiadanie vznikne práve vďaka odstraňovaniu kandidátov z konca radu pred pridaním nového). No a to je presne výherca – skutočné minimum spoločného úseku. Každého kandidáta vložíme aj odstránime z radu práve raz, preto je časová zložitosť udržiavania radu počas výpočtu hodnôt D pre konkrétne j rovná $O(N)$. S trochou námahy a za cenu straty elegancie dokážeme určiť pri výpočte $D_{i,j}$ hľadané minimum v konštantom čase. Celková časová zložitosť algoritmu sa zlepší na $O(N \cdot \lfloor \frac{N}{K} \rfloor)$.

Za nekorektné riešenie ste mohli získať najviac 2 body. Simulácie všetkých možností priebehu hry s exponenciálnou zložitosťou si vybojovali 8 bodov. 16 bodmi som hodnotil memoizácie a dynamiky v čase $O(N^2)$ a vzorové riešenie by dostalo 20 bodov. Za chýbajúce odhady zložitosti a nedostatočný popis sa dalo stratiť po jednom bode.

Listing programu:

```
#include <iostream>
#include <vector>
#include <deque>
#define INF 1234567890
using namespace std;

// kvôli kratšiemu kódu číslujem mešce od 0, Hermiho ťahy odzadu začínajúc od 0
// keď K delí N, ako posledný Hermiho ťah rátam ten, keď dostane prázdny úsek mešcov

int N, K;

int posledny(int prvy, int uroven){ // vráti index posledného mešca v rade
    return prvy + uroven * K + N % K - 1;
}

int main(){
    cin >> N >> K;
    vector<int> A(N);
```


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
    for(int i = 0; i < N; ++i)
        cin >> A[i];

    vector<vector<int> > D(N / K + 1, vector<int>(N, 0));

    if(N % K == 0) // prípad, keď Hermi nemá čo ťahať v poslednom kole
        for(int i = 0; i < N; ++i)
            D[0][i] = 0;
    else // inak vezme ten väčší mešec
        for(int i = 0; posledny(i, 0) < N; ++i)
            D[0][i] = max(A[i], A[posledny(i, 0)]);

    for(int j = 1; j <= N / K; ++j){ // j = číslo ťahu
        deque<int> Q;
        for(int i = 1; i < K - 1; ++i){
            while(!Q.empty() && D[j - 1][Q.back()] > D[j - 1][i])
                Q.pop_back();
            Q.push_back(i);
        }

        for(int i = 0; posledny(i, j) < N; ++i){ // i = index prvého mešca v rade
            while(!Q.empty() && Q.front() < i + 1)
                Q.pop_front(); // vyhodíme pristaré prvky
            while(!Q.empty() && D[j - 1][Q.back()] > D[j - 1][i + K - 1])
                Q.pop_back(); // vyhodíme väčšie prvky
            Q.push_back(i + K - 1);

            int spolocne = D[j - 1][Q.front()];
            int zlava = A[i] + min(spolocne, D[j - 1][i + K]);
            int sprava = A[posledny(i, j)] + min(spolocne, D[j - 1][i]);
            D[j][i] = max(zlava, sprava);
        }
    }

    cout << D[N / K][0] << endl;

    return 0;
}
```

Opravovala Halucinka, vzorák U$Ama
(max. 20 bodov)

7. Obchodovanie s DNA

Najprv si odmyslíme možnosť používania špinavých obchodov. Ako by vyzeralo riešenie takejto úlohy? Gény by tvorili vrcholy grafu a obchody medzi nimi by tvorili hrany. A v tomto grafe by sme hľadali najkratšiu cestu. A na to už poznáme algoritmus (konkrétne detaily si povieme neskôr).

Teraz pridajme možnosť špinavých obchodov. Potom si uvedomme, že každý gén môžeme získať buď tak, že predtým využijeme práve 0, 1, alebo 2 špinavé obchody. Dá sa z tohoto vyrobiť graf? Zoberme si ako vrchol takúto dvojicu: (gén, počet použitých čistiacich prostredkov). Teraz do toho ešte doplníme hrany. Nečestný obchod bude spájať vrcholy, ktoré majú patričný gén a rovnaký počet čistiacich prostriedkov. Špinavý obchod spojí vrcholy s príslušným génom, ale zvýši počet čistiacich prostriedkov (musíme použiť orientované hrany). Napr. špinavý obchod AA BB vyrobí nasledovné:

$(AA, 0) \rightarrow (BB, 1)$

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

$(AA, 1) \rightarrow (BB, 2)$
$(BB, 0) \rightarrow (AA, 1)$
$(BB, 1) \rightarrow (AA, 2)$

Takže sme si vyrobili graf. Počet jeho vrcholov je trojnásobok počtu rôznych génov a počet hrán je tiež nejaký násobok pôvodného počtu obchodov. Keďže nám nezáleží, koľko čistiacich prostriedkov použijeme na získanie požadovaného génu, riešením pôvodnej úlohy je najkratšia z najkratších ciest do všetkých troch vrcholov prislúchajúcich požadovanému génu.

Ako nájsť najkratšiu cestu? Vieme, že štartovací vrchol má vzdialenosť od začiatku rovnú 0. Všetci jeho susedia budú mať teda vzdialenosť 1. Všetci susedia týchto susedov, o ktorých nič nevieme (to sú tie vrcholy, u ktorých sme zatiaľ nezistili, že majú vzdialenosť 0 alebo 1) budú mať vzdialenosť 2. A tak ďalej. Keď vieme všetkých, čo majú vzdialenosť menšiu alebo rovnú ako N, tak vieme povedať aj všetkých tých, čo majú vzdialenosť rovnú $N + 1$. Takže môžeme začať tým, že máme jeden vrchol so vzdialenosťou 0. Nájdeme všetky so vzdielenosťou 1 (tú k nim zapíšeme). Tie vrcholy, ktoré sme našli si niekde zapamätáme. A z nich nájdeme všetky so vzdielenosťou 2 (a dáme si pozor, aby sme brali len tie, ktorým sme ešte vzdialenosť neurčili). Zase si ich zapamätáme. A tak ďalej. Toto má lineárnu časovú zložitost od počtu vrcholov a hrán (na každú hranu sa pozrieme práve raz z každého konca). Ešte sa to dá vylepšiť na to, aby sa to lepšie kódilo. Miesto toho, aby sme mali jeden chlievik na vrcholy so vzdielenosťou 0, ďalší na vzdialenosť 1, ... použijeme jednu frontu. Uvedomte si, že do fronty vždy najskôr príde vrchol s menšou vzdialenosťou a teda aj skôr vyjde a nič sa nám neporuší. Takže celkový postup vyzerá takto:

1. Vlož štartovací vrchol do fronty.
2. Vyber vrchol z fronty.
3. Urči vzdialenosť jeho susedov (teda tých, ktorých vzdialenosť ešte nie je určená).
4. Každého suseda, o ktorom sme sa práve dozvedeli vzdialenosť, vlož do fronty.
5. Ak fronta nie je prázdna, vráť sa na krok 2.

Ešte niekoľko poznámok k načítaniu vstupu. Hlúpy prístup štýlu „dostanem hranu tak porovnám tie stringy so stringmi všetkých vrcholov“ má zložitosť $O(NML)$ (N - počet vrcholov, M - počet hrán (počet obchodov dokopy), L - dĺžka génov). To nie je nič pekné. Toto chcelo vylepšiť. Prvé riešenie je využiť to, čo máme v niektorých jazykoch zadarmo. Napríklad mapu. Tento binárny strom nám pre každý reťazec vie povedať ktorý to je vrchol (čiže jeho číslo) v čase $O(L \log N)$. Tak isto dlho trvá aj vkladanie. To už je o dosť lepšie. Ďalšou možnosťou je použiť hashmap. Vzhľadom na to, že využíva hashovanie, trvá vyhľadávanie a vkladanie čas $O(L)$[4]. Ešte ďalšou možnosťou je použiť písmenkový strom. Ako si každý domyslel, tak je to strom. Jeho vrcholy reprezentujú nejaké stringy, resp. ich prefixy. A vrchol x je potomok y práve vtedy ak sa v y nachádza prefix toho, čo je v x a string v x je o 1 dlhší. Napr. ak máme stringy *abc*, *abd*, *acd*, tak náš strom bude mať vrcholy *a*, *ab*, *abc*, *abd*, *ac*, *acd*. Vrchol *a* bude syn koreňa stromu, jeho synovia budú *ab* a *ac*. *ab* bude mať ešte synov *abc*, *abd* a *ac* ešte syna *acd*. Rozmyslite si, ako v tejto štruktúre v lineárnom čase vyhľadávať nejaký reťazec, resp. ako v linerárnom čase doplniť nejaký reťazec. Takže časová zložitosť na spracovanie reťazca dĺžky L bude $O(L)$. Celková časová zložitosť je teda $O(N + ML)$. A pamäťové nároky sú tiež $O(N + ML)$.

Ak ste mali riešenie, ktoré fičí, ale je pomalé, určite ste nedostali pod 10 bodov. V závislosti od vašej časovej zložitosti ste poväčšinou dostali cca 16 bodov. Naopak, ak ste mali

[4]Pri použití hashovania tento čas v skutočnosti môže narásť, ak máme smolu, na oveľa väčší. Avšak toto prakticky nastáva len s veľmi malou pravdepodobnosťou a môžeme počítať s dobrým časom. Čo presne znamená *mať smolu* presahuje náplň tohto vzoráku.

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

riešenie, ktoré nefičí, tak to už bolo horšie. Veľmi som sa potešila nakódeným písmenkovým stromčekom, hoci som si všímala hlavne algoritmus prechádzania vášho grafu. Každopádne si myslím, že ste sa nad touto úlohou zamysleli, ale prečítanie vzoráku považujem u každého za prospešné. Totiž veľa z vás si povedalo, že veď urobím graf, dostanem sa zo začiatočného génu do konečného a počet prostriedkov už nejako zrátam. A práve tam začala väčšina problémov. Buď to váš program zbytočne spomalilo alebo to vôbec nerobilo, čo malo. Dúfam, že nabudúce to už budete vedieť. Tešíme sa na vaše ďalšie riešenia.

Listing programu:

```
#include <cstdio>
#include <vector>
#include <queue>
#include <algorithm>

using namespace std;

#define INF 1000000000

struct Pismenko
{
    int next[26];
    int v;
};

struct Vrchol
{
    vector<int> next;
    int vzd;
};

vector<Pismenko> pismenka;
vector<Vrchol> vrcholy;
int V;

Pismenko prazdne()
{
    Pismenko p;
    for(int i = 0; i < 26; i++)
        p.next[i]=-1;
    p.v = -1;
    return p;
}

void initPismenkac()
{
    Pismenko p = prazdne();
    pismenka.push_back(p);
}

int najdiSlovo(char *slovo)
{
    int cur = 0;
    for(int i = 0; slovo[i]; i++) {
        if(pismenka[cur].next[slovo[i]-'A']==-1) {
            Pismenko p = prazdne();
            pismenka[cur].next[slovo[i]-'A'] = pismenka.size();
```


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
            pismenka.push_back(p);
        }
        cur = pismenka[cur].next[slovo[i]-'A'];
    }
    if(pismenka[cur].v==-1) {
        pismenka[cur].v=V;
        V++;
    }
    return pismenka[cur].v;
}

int najdiVrchol(int v)
{
    if(vrcholy.size() <= 3*v) {
        Vrchol x;
        x.vzd = INF;
        for(int i = 0; i < 3; i++)
            vrcholy.push_back(x);
    }
    return 3*v;
}

int main()
{
    initPismenkac();
    char buf[1000];
    scanf("%s", buf);
    int start = najdiVrchol(najdiSlovo(buf));
    scanf("%s", buf);
    int end = najdiVrchol(najdiSlovo(buf));

    int a, b, x1, x2;
    scanf("%d %d", &a, &b);
    for(int i = 0; i < a; i++) {
        scanf("%s", buf);
        x1 = najdiVrchol(najdiSlovo(buf));
        scanf("%s", buf);
        x2 = najdiVrchol(najdiSlovo(buf));

        for(int j = 0; j < 3; j++){
            vrcholy[x1+j].next.push_back(x2+j);
            vrcholy[x2+j].next.push_back(x1+j);
        }
    }

    for(int i = 0; i < b; i++) {
        scanf("%s", buf);
        x1 = najdiVrchol(najdiSlovo(buf));
        scanf("%s", buf);
        x2 = najdiVrchol(najdiSlovo(buf));

        vrcholy[x1].next.push_back(x2+1);
        vrcholy[x1+1].next.push_back(x2+2);

        vrcholy[x2].next.push_back(x1+1);
        vrcholy[x2+1].next.push_back(x1+2);
    }
```


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
    vrcholy[start].vzd = 0;
    queue<int> fr;
    fr.push(start);
    int x;
    while(!fr.empty()) {
        x = fr.front();
        fr.pop();
        for(int i = 0; i < vrcholy[x].next.size(); i++) {
            if(vrcholy[vrcholy[x].next[i]].vzd==INF) {
                vrcholy[vrcholy[x].next[i]].vzd = vrcholy[x].vzd+1;
                fr.push(vrcholy[x].next[i]);
            }
        }
    }

    int vzd = min(vrcholy[end].vzd, min(vrcholy[end+1].vzd, vrcholy[end+2].vzd));
    if(vzd==INF)
        printf("Ani tam nechod\n");
    else
        printf("Bude to trvat aspon %d minut\n", vzd*10);
    return 0;
}
```

Opravoval Zemčo
(max. 25 bodov)

8. Taká stanovačka...

Táto úloha bola veľmi zradná. K veľkej miere opatrnosti vás mal priviesť už fakt, že aj v minulej sérii sa tento príklad objavil, avšak nebol vyriešený korektne nikým. Tentokrát už nejaké korektné riešenia prišli, ale boli len približne dve. Osobitnú pochvalu získava Tobiáš, ktorý dosiahol lepšiu časovú zložitosť ako náš pôvodný vzorák.

Mnohé riešenia sa pokúšali o dôkladný rozbor prípadov a tak skúšali vyjadriť celkový počet kosoštvorcov priamo matematicky. Ďalší sa oklamali v myšlienke, že budú pre každé k od 1 po N počítať, koľko kosoštvorcov sa zmestí do štvorca so stranou k tak, že budú využívať aspoň jeden z dvoch rozmerov úplne. Toto je často užitočný spôsob uvažovania, avšak tu sa do riešenia s dobrým časom nedal doviesť. Najväčší problém ale bol, že v tejto úlohe sa bolo veľmi ťažké zorientovať a uvedomiť si jeho podstatu[5]. Nekorektným riešeniam som dával od 0 po 2 body, v závislosti od úplnosti a od toho, aké myšlienky sa v ňom vyskytli. Ukázalo sa, že čistá matematika a rozbor prípadov napokon k úspechu neviedol a preto skúsme rozmýšľať ako programátori a zapojiť do práce aj počítač. Keďže uvádzam viaceré riešenia s rôznymi myšlienkami, koncentroval som sa práve na vysvetlenie myšlienky. Technické detaily nechávam čitateľom na premyslenie.

Prezeranie všetkých možností a kubické riešenie

Prvú vec, ktorú sme si mohli uvedomiť je, že v momente, keď poznáme tri body kosoštvorca, štvrtý si už vieme ľahko dorátať (napríklad kvôli osovej súmernosti kosoštvorca). Ďalej bolo užitočné si uvedomiť, že nám stačí uvažovať kosoštvorce s najnižším bodom (alebo jedným z najnižších) v bode $[0, 0]$, keďže posunutím nám nový kosoštvorec nevznikne. Dalo sa teda napríklad skúšať si generovať všetky dvojice bodov s celočíselnými súradnicami s jednou súradnicou nezápornou a druhou medzi N a $-N$. Takto spolu s bodom v $[0, 0]$ dostávame tri body, z ktorého jednoznačne vypočítame štvrtý – overíme, či existuje, či je celočíselný

[5] Schválne, našlo tvoje riešenie kosoštvorec $[0, 0] \to [8, 1] \to [12, 8] \to [4, 7] \to [0, 0]$? A to je N len 12...

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

a či sa výsledný kosoštvorec zmestí do zadaného štvorca. Ešte si treba uvedomiť, že niečo vypočítame viackrát a zariadiť sa podľa toho. Máme hotové prvé riešenie v čase $O(N^4)$, pretože dvojíc bodov je rádovo toľkoto a každý riešime konštantne. Možností ako docieliť podobnú časovú zložitosť však bolo viacero. Skúsme teraz naše riešenie vylepšiť.

Opäť uvažujme bod $[0,0]$ a nejaký iný zvolený bod $[k,l]$, kde l je nezáporné. Teraz vieme dĺžku strán kosoštvorcov, ktorých súčasťou bude táto úsečka. Tento fakt môžeme využiť. Pre daný bod $[k,l]$ vezmeme kružidlo, zapichneme ho do $[0,0]$ a nastavíme polomer na vzdialenosť $[k,l]$. Teraz narysujeme polkružnicu a pozrieme sa, koľkokrát a kde preťala bod s celočíselnými súradnicami.

Predstavme si body so súradnicami $[a,y]$ pre nejaké pevne zvolené a. Určite existuje najviac jedno $y, y \geq 0$ také, že úsečka z $[0,0]$ do $[a,y]$ má dĺžku rovnakú ako zvolená úsečka z $[0,0]$ do $[k,l]$. No a navyše platí, že keď sme v bode $[a,y]$ s požadovanou vzdialenosťou od $[0,0]$ a $a < 0$, potom ak a zvýšime o jedna, tak ak chceme nájsť ďalší taký bod, tak musíme hľadať medzi vyššími y. Opačne, ak je $a > 0$ a a zvýšime, potom musíme hľadať medzi bodmi s nižšími y.

Na začiatku sa teda postavíme úplne naľavo a budeme sa po jednom kroku posúvať doprava. Vždy si pritom budeme udržiavať y súradnicu tak, aby sme išli podľa kružnice a kontrolovali tak všetky podozrivé body. Keď nájdeme bod s celočíselnou súradnicou y, overíme len, či je (jednoznačný) štvrtý bod celočíselný a či sa nový stan zmestí do štvorca so stranou N. Treba si dať ale pozor, aby sme žiadny stan nezapočítali dva krát. Celý prechod bude lineárny (pretože spravíme najviac $2N$ krokov doprava a najviac $2N$ krokov dole alebo hore) a celkový čas teda $O(N^3)$. Hlavná myšlienka je, že prechodom po kružnici sme sa zbavili veľa zbytočnej práce. Toto nie je ale ani zďaleka jediné kubické riešenie, ktoré sa dalo pri tejto úlohe vymyslieť.

Rozdeľovanie úsečiek do vedierok

Ďalšie zaujímavé riešenie bol náš pôvodný vzorák a v trošku modifikovanej podobe prišlo aj od Tomiho Belana. Uvažujme opäť úsečky, ktoré začínajú v bode $[0,0]$ a smerujú niekam do bodu $[k,l]$. Každá takáto úsečka má nejakú dĺžku, ktorej druhá mocnina je prirodzené číslo. Vezmime si teraz N^2 vedier, očíslovaných od 1 po N^2. Preberieme úsečky pre všetky k,l také, že $-N \leq k \leq N, 0 \leq l \leq N$, a každú úsečku hodíme do vedra, ktoré má číslo druhej mocniny jej dĺžky. Úsečky, ktorých dĺžka je väčšia ako N, sa nám neoplatí uvažovať, pretože stan zložený z nich sa určite nezmestí do štvorca s požadovanou veľkosťou[6]. Na konci budeme mať v každom vedre všetky úsečky danej dĺžky. V mnohých vedrách nebude ani jedna úsečka, v niektorých budú dve a v niektorých bude aj viac.

O každom stane musí platiť, že je tvorený úsečkami z rovnakého vedra, pretože kosoštvorec má všetky strany rovnako dlhé. To znamená, že výsledok môžeme dostať tak, že pôjdeme po vedrách a pri každom budeme uvažovať, aké všetky kosoštvorce môžeme z úsečiek z daného vedra zostaviť a pre všetky vedrá toto sčítame.

Teraz treba už len vyriešiť, koľkými spôsobmi sa dá pre dané vedro s úsečkami s rovnakou dĺžkou zostaviť stan. Úsečky, tvoriace kosoštvorec nemôžu byť len tak hocijaké – dve protiľahlé musia byť rovnaké. Z toho nám vychádza idea, že pre každú dvojicu úsečiek vo vedre môžeme dostať nejaký tvar stanu. Pozor si len musíme dať na to, že nesmieme použiť dve rovnaké úsečky. Ďalej si treba dať pozor, aby sme nič nezarátali dvakrát a samozrejme, aby sme dodržali obmedzenie o zmestiteľnosti stanu do štvorca so stranou N. Rovnako nesmieme do vedra dostať dvojicu opačných úsečiek, avšak toto nastáva len pri úsečkách s druhou súradnicou 0.

Nech K je navyšší počet úsečiek v nejakom vedre. Ak budeme pre každé vedro skúmať každú dvojicu úsečiek osobitne, celkový čas sa bude dať zhora odhadnúť $O(N^2K^2)$, keďže

[6] Prečo?

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

každú uvažovanú dvojicu úsečiek riešime konštantne. A aké veľké je to K? Prax ukazuje, že pre $N = 500$ je to 48 a pre $N = 5000$ je to 96. Takže sa zdá, že nebude vzhľadom k N veľké. Ďalej treba poznamenať, že presnejší odhad celkovej zložitosti by bol $O(N^2 + \sum_{i=1}^{N^2} b_i^2)$, kde b_i je veľkosť vedierka s úsečkami, ktorých druhá mocnina dĺžky je i. A to je oveľa menej ako prvý odhad, kedže $\sum_{i=1}^{N^2} b_i = O(N^2)$, čo znamená, že v priemere je v každom vedre konštantne málo úsečiek (presnejšie okolo $\frac{\pi}{2}$). Toto riešenie je teda celkom dobré a jeho hlavné pozitívum je, že sme s použitím vedier (opäť) zbavili veľa práce pre hlúpe kombinácie úsečiek. Negatívum tohto riešenia je ale vyššia pamäťová náročnosť – až kvadratická.

V prípade, že by sme do vedier dávali len úsečky smerujúce do bodov, ktoré majú obe súradnice nezáporné, mohli by sme naše riešenie zlepšiť, ak by sme úsečky v každom vedre utriedili a dostali ich do poradia, v ktorom bude jedna zo súradníc rásť a druhá klesať[7]. V takomto usporiadaní potom nemusíme uvažovať každú dvojicu úsečiek vo vedre osobitne. Pre nejakú zvolenú úsečku totiž máme všetkých jej potenciálnych partnerov z hľadiska zmestiteľnosti do štvorca so stranou N v úseku za sebou a keď zvolenú úsečku posúvame, tak sa tento úsek prijateľných druhých úsečiek rozširuje tak, že je stále v usporiadaní súvislý. Preto je celá táto fáza uskutočniteľná pomocou posúvania ukazovateľov do zoznamu úsečiek lineárne od veľkosti vedra, aj keď si musíme dať pozor a zarátavať niektoré kosoštvorce viackrát. V celom algoritme je teda najpomalšia fáza triedenie, ktorého celkový čas je zhora ohraničiteľný ako $O(N^2 \log N)$, pretože triedime N^2 prvkov, aj keď nie naraz, ale po častiach (to dokázateľne nie je pomalšie).

Na záver poznamenávame, že myšlienka toho riešenia sa dá realizovať aj v čase $O(N^2)$. Predstavme si, že máme všetky úsečky v jednom veľkom poli. Keď teraz utriedime tieto úsečky podľa primárneho kritéria dĺžky a podľa sekundárneho kritéria veľkosti prvej súradnice, dostali by sme vedrá v tomto poli za sebou a mohli by sme rovnako úspešne realizovať tento algoritmus. Ak by sme použili Heapsort alebo iný efektívny algoritmus, dostali by sme čas $O(N^2 \log N)$. Pointa ale je, že toto triedenie dokážeme spraviť aj v čase $O(N^2)$. Aj dĺžka, aj veľkosť prvej súradnice sú totiž celé čísla od 1 do N^2. Keď máme K čísel od 1 po K, potom ich dokážeme pomocou Countsortu utriediť v čase $O(K)$. Číslo do veľkosti K^2 si môžeme predstaviť ako dvojciferné číslo s ciframi najviac K a tu môžeme použiť Radixsortovú myšlienku (v podstate dva krát zopakovať Countsort). Preto môžeme Radixsortom všetky úsečky utriediť pre naše potreby v čase $O(N^2)$. Jednoduchší postup ale nájdete v nasledujucom odstavci.

Tobiášov pohľad cez uhlopriečky

Vzorové riešenie je viac geometrické. Najprv si uvedomme, že uhlopriečky každého kosoštvorca sú na seba kolmé a rozpoľujú sa. Ďalej si všimnime, že ak majú byť body kosoštvorca na celočíselných súradniciach, nesmie byť bod prieniku týchto uhlopriečok len tak hocikde. Musí to byť bod, ktorý bude ako stred symetrie zobrazovať celočíselné (mrežové) body na iné celočíselné body, pretože len tak budeme mať všetky body kosoštvorca na celočíselných súradniciach. Po jednoduchom zamyslení vylúčime všetky body, ktorých obe súradnice nemajú tvar $[k + d_1, l + d_2]$, kde d_1, d_2 sú 0.5 alebo 0 a k, l sú celočíselné. Navyše sa dá dokázať, že $d_1 = d_2$. Ak by totiž jedno bolo 0 a druhé 0.5, potom by na jednej uhlopriečke ležali celočíselné body práve vtedy, keď na druhej neležali. Na demonštráciu tohto si stačí napísať rovnice uhlopriečiek. Záver teda je, že prienik uhlopriečiek môže byť len v mrežových bodoch alebo v strede štvorcov medzi mrežovými bodmi.

Keď posunieme prienik uhlopriečok každého kosoštvorca do bodu $[0, 0]$, ostanú nám jeho body alebo v celočíselných súradniciach, alebo v strede štvorcov medzi mrežovými bodmi.

[7] Stačí triediť tak, aby jedna súradnica rástla. Druhá podmienka bude vzďadom k rovnakej dĺžke úsečiek vo vedrách splnená automaticky.

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Každý kosoštvorec je teda alebo jedného alebo druhého typu. Každý z typov skúsime zrátať osobitne.

Kosoštvorce prvého typu budú mať po posunutí prieniku ich uhlopriečok do stredu súradnicovej sústavy všetky body na celočíselných súradnidiach. Každý kosoštvorec bude mať v každom kvadrante práve jeden bod (hraničné čiary priradíme kvadrantom tak, aby mal každý jednu).

Uvažujme teraz nejaký takýto kosoštvorec. Pozrime sa na priamku, ktorá vznikne predĺžením jeho uhlopriečky idúcej cez prvý kvadrant. Všimnime si, že každý celočíselný bod z prvého kvadrantu, ktorý táto priamka preťala, môžeme použiť na vznik nového kosoštvorca, ktorý bude oproti pôvodnému sploštený podľa tejto uhlopriečky. Treba si totiž uvedomiť, že všetky tieto body majú svoj zrkadlový obraz aj v treťom kvadrante. Stačí, keď bod z prvého kvardantu posunieme do tohto bodu a zodpovedajúco posunieme aj bod v treťom kvadrante. Samozrejme, uvažujeme len body so súradnicami do $N/2$.

A ako to bude vyzerať v druhom a štvrtom kvadrante? Presne symetricky, pretože uhlopriečky sú na seba kolmé. Predstavme si teda, že v prvom kvadrante preťala k bodov so súradnicami do $N/2$. Potom aj kolmá priamka pretína k bodov v druhom kvadrante. Preto si môžeme vybrať ľubovoľné body spomedzi prvých a druhých a dostaneme vyhovujúci kosoštvorec. Dostávame teda, že počet kosoštvorcov, ktorý je tvorený uhlopriečkami s daným zvoleným smerom, je k^2.

A ako máme naprogramovať algoritmus, aby bol dosť rýchly? Budeme spracovávať body v prvom kvadrante. Pôjdeme od tých s najmenšími súradnicami a pre bod $[k, l]$ sa budeme pýtať, koľko bodov bude ležať na priamke, ktorá vznikne predĺžením úsečky $[0, 0] \rightarrow [k, l]$. Všetky celočíselné body na tejto priamke totiž nájdeme ľahko – budeme skúšať násobky prvého bodu, teda body $[kd, ld]$ pre $d = 1, 2, \ldots$. Akonáhle niektorá súradnica presiahne $N/2$, môžeme skončiť, pretože už ďalšie vhodné body nenájdeme. Takýmto spôsobom spočítame počet bodov na priamke a už vieme, že k výsledku musíme pripočítať jeho druhú mocninu. Keď nejaký bod zarátame, poznačíme si ho. Keď k nemu prídeme neskôr, budeme vedieť, že ho už nemáme riešiť, pretože smer k nemu už sme uvažovali. Na toto nám postačí jedno dvojrozmerné pole typu boolean.

Ostáva nám spočítať kosoštvorce druhého typu. Tieto budú mať po presunutí prieniku uhlopriečok body v strede mrežových bodov (každý bod bude mať každú súradnicu celé číslo plus polovica). Avšak použiť modifikovaný spôsob ako pri prvom type nám nič nebráni. Obe fázy trvajú $O(N^2)$ čas, pretože každý z $O(N^2)$ bodov nás stojí konštantý počet operácii. Potrebujeme bohužiaľ aj $O(N^2)$ pamäť (bez pomoci pamäte by nám časová zložitosť narástla, ak by sme chceli zachovať korektnosť).

Listing programu:

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

#define FOR(i,n) for(int i=0;i<(n);i++)
#define MAXN 1000
//pole na zapametanie si, ktore body sme uz riesili
bool P[MAXN][MAXN];
int res,N;

int main(){
  cin >> N;
  //vycistime si pole
  FOR(i,N/2+1)FOR(j,N/2+1) P[i][j] = false;
  //prva faza - prienik uhlopriecok je mrezovy bod
  FOR(i,N/2+1)FOR(j,N/2+1)if (P[i+1][j] == false){
```


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
    int bodov = 1;
    int px=i+1; int py=j;
    while(px <= N/2 && py <= N/2){
      P[px][py]=true; bodov++; px+=(i+1); py+=(j);
    }
    res+=(bodov-1)*(bodov-1);
  }
  //znovu vycistime pred druhou fazou - prenik je polciselny bod
  FOR(i,N/2+1)FOR(j,N/2+1) P[i][j] = false;
  FOR(i,N/2+1)FOR(j,N/2+1)if (P[i][j] == false){
    int bodov = 1;
    //tato zvlastna delta kvoli tomu, ze kazdy druhy bod je necelociselny
    //a neuvazujeme ho
    double dx = 2*i+1; double dy = j*2+1;
    double px = 0.5 + i; double py = 0.5 + j;
    while (px <= (double)N/2 && py <= (double)N/2){
      P[(int)px][(int)py]=true; bodov++; px+=dx; py+=dy;
    }
    res+=(bodov-1)*(bodov-1);
  }
  cout << res << endl;
  return 0;
}
```

Opravovala kewo
(max. 25 bodov)

9. Tanečná

Príklad sa tešil nesmiernej obľube, našli sa aj správne riešenia[8].

bNa začiatku máme nálepky. Tie nám vyjadrujú permutáciu dĺžky N, ktorú vieme rozdeliť na c cyklov ($c \leq N$) dĺžky $l_i, 1 \leq i \leq c, 1 \leq l_i \leq N, \sum l_i = N$. Cyklus dĺžky l sa počas tanca môže nachádzať (nečakane) v l rôznych posunutiach ($p \in 0, 1, \ldots (l-1)$).

Stav každého z cyklov v K-tom kole vieme jednoznačne vyjadriť ako $p_i = K \bmod l_i, 1 \leq i \leq c$ alebo tiež $K \equiv p_i \pmod{l_i}$. Notácia $a \equiv b \pmod{M}$ značí, že „a a b sú kongruentné modulo M“ čo v praxi znamená, že a a b po delení M dávajú rovnaký zvyšok. (Uvedomte si, že keď máme napr. cyklus dĺžky 3 a vykonali sme K krokov, tak posunutie cyklu bude skutočne $K \bmod 3$).

Poznáme cykly, poznáme aktuálny stav parketu. Čo nepoznáme a chceme poznať je K alebo $K \bmod P$ (kde P bude najmenší spoločný násobok $l_1, l_2, ...l_c$). Čo nám k tomu chýba? Vzhliadnime o pár riadkov vyššie na asi jediný vzťah, kde sa nám K nachádza: $K \equiv p_i \pmod{l_i}$, $1 \leq i \leq c$. Počet cyklov vieme, dĺžku cyklov tiež (tu vieme zistiť napr. jednoduchým ofarbovaním v $O(N)$), čo nám chýba sú posunutia p_i, ktoré v podstate tiež vieme zo zadania. Celkový čas na toto bude opäť $O(N)$, najprv predrátame pre každý cyklus posunutia všetkých bodov oproti nejakému bodu na cykle a potom v konštantnom čase vieme povedať o koľko sa kto posunul, vypočítaním rozdielu novej a pôvodnej pozície. Ešte samozrejme ak sa niekto dostal do úplne iného cyklu, tak prehlásime, že sa niekto pomýlil.

Máme sústavu kongruencií (tých bude maximálne N, občas pre niektorý cyklus ich bude viac, ale to nám až tak nevadí) popisujúcich stav parketu - situácia na parkete mohla nastať práve vtedy, keď spomínaná sústava kongruencií má riešenie. Už ich len vyriešiť.

[8] a to nám došli aj z Fínska, Číny či gym. Pankúchová...

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Tu nám pomôže napr. metóda postupnej substitúcie[9], ktorá sa vyskytla v spomínaných fínskych a čínskych riešeniach:

V každom kroku táto metóda zoberie 2 kongruencie $K \equiv a \pmod{b}$ a $K \equiv c \pmod{d}$ a spraví ich riešenie v tvare: $K \equiv e \pmod{f}$. Z prvej rovnice vieme povedať, že: $K = nb + a$ (kde n je nejaké celé číslo). Toto dosadíme do druhej: $nb + a \equiv c \pmod{d}$. Túto upravíme na $nb \equiv c - a \pmod{d}$. Teraz si uvedomne, že ak sú d, b nesúdeliteľné, tak určite b má inverzný prvok modulo d (inverzný prvok je číslo b^{-1}, pre ktoré platí $bb^{-1} \equiv 1 \pmod{d}$. Takýto prvok vieme nájsť napríklad rozšireným euklidovým algoritmom[10]). Potom týmto prvkom môžeme kongruenciu opäť upraviť. A získáme priamo hodnotu n. Pokiaľ sú d, b súdeliteľné, tak ich najväčší spoločný deliteľ musí deliť $a - c$. Ak nedelí, tak táto kongruencia nemá riešenie. Ak delí, môžeme obe strany aj modulo vydeliť týmto deliteľom a sme tam kde sme boli. Teraz dostaneme $n = x \pmod{y}$. A toto dosadíme do pôvodnej veci a máme $K = b(my + x) + a = bym + bx + a$, čiže $K \equiv bx + a \pmod{by}$.

Keď máme sústavu viacerých kongruencií, tak najprv poriešime prvé dve. Potom výsledok spojíme s ďalšou a takto pokračujeme ďalej.

Čas jedného kroku je konštantný až na hľadanie inverzných prvkov a najväčších spoločných deliteľov. Ten je rovný logaritmu menších prvkov, ktoré vchádzajú do daného algoritmu, čiže zložitosť každého kroku je $O(\log l_i)$. Keďže $\log l_i < l_i$, tak celkovú zložitosť vieme odhadnúť ako $O(l_1 + l_2 + ... + l_c) = N$.

Ešte ako poslednú vec potrebujeme zistiť, že kto kde skončí. Dostali sme ako výstup $K = X \pmod{Y}$. Pokiaľ dĺžka cyklu delí Y, tak vieme zistiť, kde sa daný človek nachádza (zistíme $X \bmod l_i$). Ináč nevieme.

Suma sumárov: Pamäť: $O(N)$ Čas: $O(N)$

Pozn.: Ticho predpokladáme, že výsledné čísla nám budú výchadzať rozumne veľké. Reálne ale môžu narásť na dosť veľké. Potom by ale zložitosti násobení v riešení sústavy už neboli konštantné.

Dobrú chuť, zdroják Vám posiela USAma.

Listing programu:

```
#include <stdio.h>
#include <vector>
#include <algorithm>

using namespace std;

int n;
int send[1000000];        //kam nas to posiela
int num[1000000];         //cislo pozicie v cykle
int len[1000000];         //dlzka cyklu, kde je policko
int numc[1000000];        //cislo cyklu
int out[1000000];         //vystup

vector<pair<int,int> > ekv;

typedef long long vlong;

pair<vlong, pair<vlong, vlong> > egcd(vlong a, vlong b) {   //rozsireny euklid
    if (b==0) {
        return make_pair(a, make_pair(1,0));
    }
```

[9] http://en.wikipedia.org/wiki/Method_of_successive_substitution

[10] http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
    pair<vlong, pair<vlong, vlong> > tmp=egcd(b, a%b);
    return make_pair(tmp.first, make_pair(tmp.second.second, tmp.second.first -
tmp.second.second * (a/b)));
}

vlong inv(vlong x, vlong MOD) { //rata inverzny prvok
    return (egcd(x, MOD).second.first + MOD)%MOD;
}

//x = first mod second
pair<vlong, vlong> solve(pair<vlong, vlong> y, pair<vlong, vlong> x)
{
    vlong a, b, c, d, k;
    a = y.second;
    b = y.first;
    c = x.first;
    d = x.second;//a*k + b = c mod d
    b%=d;
    c = (c-b+d)%d;//a*k = c mod d
    vlong gc = __gcd(a, d);
    if(c%gc!=0) return make_pair(-1,-1);
    c/=gc;
    a/=gc;
    d/=gc;
    k = c*inv(a, d);
    k %= d;
    return make_pair(y.first+y.second*k, y.second*d);
}

int main()
{
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
    {
        scanf("%d", &send[i]);
        send[i]--;
        num[i]=-1;
    }
    int l, cur;
    int nc = 0;
    for(int i = 0; i < n; i++)//spocitame cykly
    {
        if(num[i]!=-1) continue;//uz sme tu boli
        l = 0;
        cur = i;
        while(num[cur]==-1)
        {
            num[cur]=l;
            l++;
            cur=send[cur];
        }
        cur = i;
        do {
            len[cur]=l;
            numc[cur]=nc;
            cur=send[cur];
        }while(cur!=i);
        nc++;
```


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
    }
    int a,roz;
    for(int i = 0; i < n; i++)
    {
        scanf("%d", &a);
        if(a==-1) continue;
        a--;
        if(numc[i]!=numc[a])//niekto stoji uplne mimo
        {
            printf("Niekto sa pomylil\n");
            return 0;
        }
        roz = num[a]-num[i];
        if(roz<0) roz+=len[i];
        ekv.push_back(make_pair(roz, len[i]));
    }
    pair<int,int> sol = make_pair(0,1);
    for(int i = 0; i < ekv.size(); i++)
    {
        sol = solve(ekv[i], sol);
        if(sol.second==-1){
            printf("Niekto sa pomylil.\n");
            return 0;
        }
    }
    //sol.first == pocet otociek mod sol.second
    for(int i = 0; i < n; i++)
        out[i]=-1;

    int g;
    for(int i = 0; i < n; i++)
    {
        if(out[i]!=-1) continue;//uz sme pisali
        if(sol.second%len[i]!=0) continue;
        g = sol.first%len[i];
        cur = i;
        for(int j = 0; j < g; j++)//posunieme cloveka o g pozicii dopredu
            cur = send[cur];
        l = i;
        while(1)
        {
            if(out[cur]!=-1) break;//zapisali sme vsetko
            out[cur]=l+1;
            l = send[l];
            cur = send[cur];
        }
    }
    for(int i = 0; i < n; i++)
        printf("%d ", out[i]);
    printf("\n");
}
```

+

http://www.ksp.sk/ksp2.0

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Opravoval PPershing
(max. 25 bodov)

10. Totálne zle naplánovaný výlet

Úlohu vyriešime dvoma rôznymi spôsobmi, každý s inou časovou zložiťostou. Keď potom poskladáme tieto dva spôsoby dokopy do jedného programu, ktorý si vyberie pre daný vstup tú rýchlejšiu možnosť, dostaneme vzorové riešenie. Prvý spôsob je rýchly, ale funguje len občas. Konkrétne, pre funkčnosť predpokladajme podmienku $K \geq NSN(A,B)$. Ako nájdeme riešenie? V prvom rade, nech $d = NSD(A,B) = X_A \cdot A + X_B \cdot B$. Danú trojicu d, X_A, X_B nájdeme napríklad rozšíreným Euklidovým algoritmom pre najväčšieho spoločného deliteľa[11]. Nech ďalej platí $A = d \cdot a$ a $B = d \cdot b$. Čísla a, b sú nesúdeliteľné a preto $NSN(A,B) = d \cdot a \cdot b$ (pretože $NSN(A,B) \cdot NSD(A,B) = A \cdot B$ z čoho dostaneme $NSN(A,B) = \frac{(a \cdot d) \cdot (b \cdot d)}{d}$). Ďalej nech $k = \lfloor \frac{K}{d} \rfloor$. Potom tvrdíme, že existuje riešenie, v ktorom počet použitých bubákov je práve $K_{max} = d \cdot k$. Zjavne totiž každé riešenie $A \cdot S_A + B \cdot S_B = d \cdot (A \cdot a + B \cdot b)$ je deliteľné d a K_{max} je najväčší násobok d menší resp. rovný K.

Dôkaz: V prvom rade, v celom zvyšku vzoráku budeme uvažovať predpoklad $d = 1$. Ako domácu úlohu si dôkaz urobte so všeobecným d alebo si rozmyslite, čo sa stane ak A, B, K predelíme hodnotou d.

Zoberme ako riešenie $S_A = k \cdot X_A, S_B = k \cdot X_B$, kde X_A, X_B dostaneme z rozšíreného Euklidovho algoritmu. Platí $K_{max} = A \cdot S_A + B \cdot S_B$. Čo nám môže pokaziť jeho správnosť? Napríklad taká zápornosť. Nikto nám nezaručuje, že obe čísla X_A, X_B budú kladné - dokonca ani nebudú. Bez ujmy na všeobecnosti nech $S_A < 0$. Potom vieme spraviť nasledujúci trik - zväčšime S_A o b a zmenšime S_B o a. Hodnota K_{max} sa pri tejto úprave nezmení, pretože $a \cdot (S_A + b) + b \cdot (S_B - a) = S_A \cdot a + b \cdot a + S_B \cdot b - a \cdot b = S_A \cdot A + S_B \cdot B = K_{max}$. Toto môžeme opakovať pokiaľ $S_A < 0$ a potom zastať. Otázkou ostáva, či v tomto momente $S_B \geq 0$. Tu príde na pomoc predpoklad $K_{max} \geq NSN(a,b)$ (rozmyslite si, že to vyplýva z $K > NSN(a,b)$ a deliteľnosti K_{max} číslom d), pričom $NSN(A,B) = a \cdot b$. Nech $0 \leq S_A < b$ a $S_B < 0$. Potom $K_{max} = a \cdot S_A + b \cdot S_B < a \cdot b \leq K_{max}$. Spor.

Máme teda nejaké riešenie. Je optimálne? Nemusí byť. Opakujme však náš trik až do momentu, kedy máme ešte nezáporné S_B ale ďalšou iteráciou by už nebolo. Potom tvrdím, že S_A, S_B je optimálne riešenie.

Dôkaz: Nech existuje nejaké S'_A, S'_B také že $aS_A + bS_B = K_{max} = aS'_A + bS'_B$ a zároveň $0 \leq S_B < a$ a $S'_A > S_A$ a $S'_B \geq 0$. Potom $X = S'_A - S_A, Y = S_B - S'_B$ je riešenie rovnice $aX = bY$, pričom $X > 0, Y < a$. Lenže ak $Y < a$, tak potom bY je číslo deliteľné b a zároveň aj a, ktoré je menšie ako $NSN(a,b) = ab$, čo je spor.

Časová zložitosť tohoto riešenia je $O(log(A+B))$ pretože na začiatku potrebujeme rozšírený Euklidov algoritmus a zvyšok vieme v konštantnom čase (namiesto opakovania triku si spočítame počet iterácii obyčajným delením).

Myšlienka druhého prípadu bude nasledujúca: S_A môže nadobúdať hodnoty medzi 0 a $\lfloor K/A \rfloor$. Pre každú z týchto hodnôt vieme dopočítať S_B jednoducho ako $\lfloor \frac{K - A \cdot S_A}{B} \rfloor$. Ak sa pozrieme na hodnotu K_{max}, určite sa bude nachádzať medzi $A \cdot \lfloor \frac{K}{A} \rfloor$ a K. Inak povedané, zaujíma nás zvyšok $K - A \cdot S_A$ modulo B. Spomedzi všetkých riešení chceme vybrať také, kde je tento zvyšok najnižší a ak je takých viacej, tak to, ktoré má najväčšie S_A. Ako toto spraviť prefíkane? Uvažujme zvyšky pre $S_A = 0, 1, \ldots$ a nakreslime si úsečky s ich dĺžkami za sebou do jedného dlhého riadku. Uvažujme nejaké číslo t (neskôr si ho upresníme) a po každej t-tici týcto úsečiek si nakreslime zvislú čiaru[12]. Našim cieľom bude vyhnúť sa počítaniu všetkých „chlievikov“ okrem toho prvého a nájsť v ních minimum inteligentnejšie. Formálne zapísané, zvyšok vieme iteratívne počítať nasledovne: $ZV_{S_A+1} = (ZV_{S_A} + C)$ mod B kde

[11] REA si môžu záujemcovia nájsť na `http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm` a v našom kóde na konci tohto vzoráku.

[12] to akože oddeľovač medzi chlievikmi

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

$C = (-A \bmod \mathrm{B})$. Vypočítajme si prvých t členov. Ako zistíme ďalších t členov? $ZV_{i+t} = ZV_i + C_t$ kde $C_t = (-t \cdot A \bmod \mathrm{B})$. Na prvý pohľad sme si neušetrili žiadnu prácu. Ale pozor, nás predsa nazaujímajú tie členy. Nás zaujíma ich minimum. A toto môžeme nájsť veľmi efektívne. Predstavme si, že sme dané zvyšky pre hodnoty $0, 1, \ldots, t-1$ (prvý chlievik) utriedili. Nájsť minimum pre zvyšky $t, t+1, \ldots, 2t-1$ (druhý chlievik) je jednoduché – stačí, ak binárne vyhľadáme prvú hodnotu väčšiu alebo rovnú $B - C_t$ – táto hodnota $+C_t$ (skok na ďalší chlievik) je prvá hodnota väčšia rovná $B - C_t + C_t \bmod \mathrm{B} = 0$. Inými slovami, minimálna možná.

Takto vieme v čase $O(\log t)$ vyhľadať maximum pre každú ďalšiu t-ticu zvyškov. Celkovo máme vstupnú investíciu $O(t \log t)$ na výpočet a utriedenie prvého chlievika. Ostatok vieme dopočítat v čase $O(\frac{K}{At} \log t)$, pretože máme najviac $\lceil \frac{K}{At} \rceil$ chlievikov. Ak položíme $t = \sqrt{K/A}$, dostávame čas výpočtu $O(\sqrt{K/A} \log \sqrt{K/A}) = O(\sqrt{K/A} \log K/A)$.

Dokopy teda vieme, že ak $K \geq NSN(A, B)$, výpočet vieme spravit rýchlo (logaritmicky). Ak $K < NSN(A, B)$, tak $K/A < B$ a teda $\sqrt{K/A} \log (K/A) < \sqrt{B} \log B$. Preto môžeme výslednú časovú zložitosť programu zapísať jednotne ako $O(\sqrt{\min(A, B)} \log \min(A, B))$.

Listing programu:

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <cmath>
#include <vector>
#include <utility>
using namespace std;
typedef long long int ll;
typedef pair<ll, ll> PLL;
#define INF 10000000000000ll

// returns (x,y) such that x * a + y * b == gcd(a, b)
PLL extended_gcd_(ll a, ll b) {
    if (a % b == 0) return make_pair(0, 1);

    PLL tmp = extended_gcd_(b, a % b);
    return make_pair(tmp.second, tmp.first - tmp.second * (a / b));
}

// First strategy, use when K>=AB
PLL strategy1(ll K, ll A, ll B) {
    PLL g = extended_gcd_(A,B);
    // gcd
    ll d = g.first * A + g.second * B;
    ll k = K / d;
    ll sa = k * g.first;
    ll sb = k * g.second;

    A /= d;
    B /= d;

    ll t = sb / A;
    sa += B * t;
    sb -= A * t;
    // if sb was negative, now it is in range -a..0, fix that
    if (sb<0) {
        sa -= B;
```


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
        sb += A;
    }
    return make_pair(sa, sb);
}

// Second strategy
PLL strategy2(ll K, ll A, ll B, bool maximize) {
  ll block_size = 1 + (ll) sqrt(K / A);
  vector<PLL> data(block_size);
  // direction for sorting
  int direction = maximize ? -1 : 1;

  for (int i = 0; i < block_size; i++) {
    data[i] = make_pair((K - A * i) % B, direction * i);
  }
  sort(data.begin(), data.end());

  ll bestA = -1;
  ll bestZV = INF;
  ll current_block = 0;
  ll C = 0;

  while (current_block + block_size <= K/A) {
    vector<PLL>::iterator pos = lower_bound(data.begin(), data.end(),
              make_pair(B - C, -INF));
    if (pos == data.end()) pos=data.begin();

        if ((pos->first + C) % B < bestZV ||
            ((pos->first + C) % B == bestZV && maximize)) {
                bestZV = (pos->first + C) % B;
                bestA = abs(pos->second) + current_block;
            }
    C = (B+ (C - block_size * A) % B) % B;
    current_block += block_size;
  }

  // finish last block
  for (int i = current_block; i <= K/A; i++) {
    if ((K - A * i) % B < bestZV ||
        ((K - A * i) % B == bestZV && maximize)) {
        bestZV = (K - A*i) %B;
        bestA = i;
    }
  }

  assert(bestA != -1);
  return make_pair(bestA, (K - bestA * A) / B);
}

int main() {
    ll A, B, K;
    PLL result;

    cin >> K >> A >> B;

    if (K >= A * B) {
        result = strategy1(K, A, B);
    } else {
```


Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

```
        if (A > B) {
            result = strategy2(K, A, B, true);
        } else {
            result = strategy2(K, B, A, false);
            swap(result.first, result.second);
        }
    }
    cout << "Best solution is "<< result.first
         << ", " << result.second << " " <<
         result.first*A + result.second*B <<    endl;
}
```

http://www.ksp.sk/ksp2.0

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Výsledková listina po 1. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda	1	2	3	4	5	Σ
1	Balog Matej	Gym. Grösslingová BA	3	10	9	10	15	15	59
1	Brandys Jozef	Gym. Námestovo	2	9	10	10	15	15	59
1	Rohár Pavol	Gym. M. R. Štefánika, Košice	4	10	9	10	15	15	59
4	Horňák Marián	Gym. Párovská Nitra	2	10	7	10	13	15	55
4	Mariš Andrej	Gym. Piaristická Nitra	2	10	8	10	12	15	55
4	Pulmann Jan	Gym. Grösslingová BA	3	8	7	10	15	15	55
4	šinogl jakub	Gym. Žiar nad Hronom	3	10	8	10	12	15	55
8	Birkus Róbert	SPŠE Nové Zámky	3	9	7	10	13	15	54
9	Sedláček Lukáš	Gym. Žiar nad Hronom	3	10	9	10	13	8	50
9	Smolik Michal	Gym. Grösslingová BA	1	10	6	10	11	13	50
11	Vargovčík Matej	Gym. Sabinov	3	9	9	2	13	15	48
12	Magdolen Matej	Gym. Golianova Nitra	4	10	7	10	11	8	46
12	Novella Tomáš	Gym. Alejová Košice	4	9	9	10	11	7	46
14	Babiak Jakub	Gym. Žiar nad Hronom	3	10	9	10	12	4	45
15	Greššák Jerguš	Gymnázium J.A. Raymana	1	10	6	7	10	11	44
16	Klc Dano	GLN Tomášikova BA	4	10	5	10	9	8	42
16	Kováč Ondrej	SKŠ Nitra, Gym. sv. Cyrila a Metoda	3	10	7		12	13	42
16	Porubsky Martin	SKŠ Nitra, Gym. sv. Cyrila a Metoda	3	8	10	10	9	5	42
16	Čačko Marek	Gym. M. Galandu Turčianske Teplice	4	8	7	10	9	8	42
20	Sebechlebský Ján	Gym. Žiar nad Hronom	3			10	15	15	40
21	Mutný Mojmír	Gym. Jura Hronca BA	2	9	4	10	0	15	38
22	Jurových Jakub	Gym. Okružná Zvolen	3	9	7	10	5	5	36
23	Šuppa Marek	SKŠ Nitra, Gym. sv. Cyrila a Metoda	1	8	6	7	9	5	35
24	Jankovič Radovan	Gymnázium Levice	3	8	10	10		6	34
24	Jurenka Vladimír	Gym. Grösslingová BA	4	8	7	10	9		34
26	Anderle Michal	Gym. Haličská Lučenec	3	10	6		12	5	33
26	Masár Juraj	Gym. Jura Hronca BA	3	10	6	6		11	33
26	Minárik Jakub	Gym. Jura Hronca BA	3	7	3	0	8	15	33
26	Takács Gabriel	Gym. Fándlyho Šaľa	2	8	7	10		8	33
30	Livora Tomáš	Gym Javorová Spiš. Nová Ves	3	7	9	10		6	32
31	Kutaj Tomáš	Gym. Jura Hronca BA	3	8	3	0	12	8	31
32	Krumpal Rudo	Gym. Jura Hronca BA	3	4	2		8	15	29
33	Pokorný Fridolín	Gym. Haličská Lučenec	4	9	4	0	9	6	28
34	Ivanov Marián	Gym. Jura Hronca BA	1			10		15	25
34	Kučera Martin	Gym. Golianova Nitra	3			10	9	6	25
34	Pinter Martin	Gym. Jura Hronca BA	3	9	10	2		4	25
34	Toman Viktor	Gym. Golianova Nitra	3	9	5	2	9		25
34	Šormanová Mária	Škola pre mim. nadané deti BA	3	8	7	10		0	25
39	Lami Jozef	Gym. Poštová Košice	2	9	5	10			24
40	Vozárová Zuzana	Gym. Jura Hronca BA	2	8		0		15	23
41	Marko Jozef	SPŠ Myjava	3	6	3	0	11		20
41	Petrucha Jaroslav	Gym. Metodova BA	1		10	10			20
43	Chudjak Martin	SPŠ Martin	4	10	5	2			17
44	Plavák Dušan	Gym. Trstená	3	9	4			3	16
45	Anderko Maroš	Gym. Konštantínova Prešov	4	9	6	0			15
45	Hornyák Zsolt	Gymnázium	4	8		0	7		15
47	Krajčovič Matej	Gym. Jura Hronca BA	1	8	6	0			14
47	Krasnayová Dáša	Gym. Alejová Košice	3	8	6				14
47	Viskup Michal	Gym. Jura Hronca BA	3	6		0		8	14

http://www.ksp.sk/ksp2.0

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

	Meno a priezvisko	Škola	Trieda	1	2	3	4	5	Σ
50	Bezek Matúš	Gym. Jura Hronca BA	1	7				5	12
51	Durčák Dávid	Gym. Námestovo	4	10					10
51	Ille Ondrej	Gym. Jura Hronca BA	3	2		0		8	10
51	Matuš Juraj	Gym. Jura Hronca BA	2	6	3		1		10
51	Václavík Lukáš	SSOĽ Via Humana	2	5		5	0		10
51	Šeliga Adam	Súkromná SOŠ Humanus Via	2	5		5			10
56	Dang Quoc Trung	Gym. Jura Hronca BA	2	5		0		4	9
57	Kucerová Bibiana	Gym. Alejová Košice	3	7				1	8
57	Pistrakova Alexandra	Gym. Poštová Košice	2	8	0				8
59	Krotký Miroslav	Gym. Spišská Stará Ves	2	6				1	7
60	Bobuľa Daniel	Spojená škola Kráľovnej pokoja Žilina	2	6					6
60	Mesároš Tomáš	Gym. Jura Hronca BA	3	4		2			6
62	Lieskovsky Adam	Gym. Jura Hronca BA	3	2	2			1	5
62	Orenič Jozef	Gym. Slovenská Bardejov	4	5			0		5
64	Šafin Jakub	Gym. P. Horova Michalovce	1	4	0	0		0	4
65	Kollár Dan	Gym. Grösslingová BA	2	3				0	3

http://www.ksp.sk/ksp2.0

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

Výsledková listina po 1. kole kategórie KSP-O

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
1	Hudec Tobiáš	Gym. Partizánske	4	15	15	16	20	25	91
2	Belan Tomáš	Škola pre mim. nadané deti BA	4	15	15	16	17	20	83
3	Rohár Pavol	Gym. M. R. Štefánika, Košice	4	15	15	15	20	2	67
4	Hozza Jan	Gym. Jura Hronca BA	3	14		16	20		50
5	Sebechlebský Ján	Gym. Žiar nad Hronom	3	15	15	7	12		49
6	Mariš Andrej	Gym. Piaristická Nitra	2	12	15	16	5		48
7	Pitoňák Martin	Gym. Tajovského B. Bystrica	4	15	15		12	2	44
8	Korbaš Rafael	Gym. Hronská BA	3	12	15	1	13		41
9	Kekely Michal	Gym. Varšavská Žilina - Vlčince	3	15	15	2	6	1	39
10	Varga Mátyás	Gym. H. Selyeho Komárno	3	15	15	7		1	38
11	Mrocková Mária	Gym. Jura Hronca BA	3	12	15		7	1	35
12	Cuc Bruno	Gym. Grösslingová BA	4	7	11	1	13	1	33
13	Balog Matej	Gym. Grösslingová BA	3	15	15				30
13	Brandys Jozef	Gym. Námestovo	2	15	15				30
13	Dresslerova Anna	Gym. Jura Hronca BA	3	15	15				30
13	Gandžala Jozef	Gym. Tajovského B. Bystrica	4	15	15				30
13	Habovštiak Martin	Gym. Tvrdošín	4	15	3		12		30
13	Pulmann Jan	Gym. Grösslingová BA	3	15	15				30
19	Birkus Róbert	SPŠE Nové Zámky	3	13	15				28
19	Horňák Marián	Gym. Párovská Nitra	2	13	15				28
19	Kučera Martin	Gym. Golianova Nitra	3	9	6		13		28
19	Vargovčík Matej	Gym. Sabinov	3	13	15				28
19	Večerík Matej	Škola pre mim. nadané deti BA	3	13	15				28
24	Miklovič Tomáš	Gym. Nové Zámky	4	12	15				27
24	šinogl jakub	Gym. Žiar nad Hronom	3	12	15				27
26	Kováč Ondrej	SKŠ Nitra, Gym. sv. Cyrila a Metoda	3	12	13				25
27	Smolik Michal	Gym. Grösslingová BA	1	11	13				24
28	Krumpal Rudo	Gym. Jura Hronca BA	3	8	15				23
28	Minárik Jakub	Gym. Jura Hronca BA	3	8	15				23
30	Greššák Jerguš	Gymnázium J.A. Raymana	1	10	11				21
30	Sedláček Lukáš	Gym. Žiar nad Hronom	3	13	8				21
32	Kutaj Tomáš	Gym. Jura Hronca BA	3	12	8				20
33	Magdolen Matej	Gym. Golianova Nitra	4	11	8				19
33	Novella Tomáš	Gym. Alejová Košice	4	11	7		1		19
35	Jankovič Radovan	Gymnázium Levice	3		6	2	9	1	18
36	Anderle Michal	Gym. Haličská Lučenec	3	12	5				17
36	Klc Dano	GLN Tomášikova BA	4	9	8				17
36	Ziman Michal	Gym. Haličská Lučenec	4	12	5				17
36	Čačko Marek	Gym. M. Galandu Turčianske Teplice	4	9	8				17
40	Babiak Jakub	Gym. Žiar nad Hronom	3	12	4				16
40	Vozárová Zuzana	Gym. Jura Hronca BA	2		15			1	16
42	Ivanov Marián	Gym. Jura Hronca BA	1		15				15
42	Kovár Martin	Gym. Jura Hronca BA	3		15				15
42	Mutný Mojmír	Gym. Jura Hronca BA	2	0	15				15
42	Pokorný Fridolín	Gym. Haličská Lučenec	4	9	6				15
46	Porubsky Martin	SKŠ Nitra, Gym. sv. Cyrila a Metoda	3	9	5				14
46	Spano Marek	Gym. Jura Hronca BA	3	14					14
46	Šuppa Marek	SKŠ Nitra, Gym. sv. Cyrila a Metoda	1	9	5				14
49	Marko Jozef	SPŠ Myjava	3	11					11

http://www.ksp.sk/ksp2.0

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
49	Masár Juraj	Gym. Jura Hronca BA	3		11				11
51	Jurových Jakub	Gym. Okružná Zvolen	3	5	5				10
52	Jurenka Vladimír	Gym. Grösslingová BA	4	9					9
52	Toman Viktor	Gym. Golianova Nitra	3	9					9
54	Holas Juraj	Gym. Jura Hronca BA	3	0	8				8
54	Ille Ondrej	Gym. Jura Hronca BA	3		8				8
54	Takács Gabriel	Gym. Fándlyho Šaľa	2		8				8
54	Viskup Michal	Gym. Jura Hronca BA	3		8				8
58	Hornyák Zsolt	Gymnázium	4	7					7
58	Hruby Tomas	Gym. Jura Hronca BA	3	5	2				7
60	Livora Tomáš	Gym Javorová Spiš. Nová Ves	3		6				6
61	Bezek Matúš	Gym. Jura Hronca BA	1		5				5
62	Dang Quoc Trung	Gym. Jura Hronca BA	2		4			0	4
62	Pinter Martin	Gym. Jura Hronca BA	3		4				4
64	Plavák Dušan	Gym. Trstená	3		3				3
65	Krotký Miroslav	Gym. Spišská Stará Ves	2		1				1
65	Kucerová Bibiana	Gym. Alejová Košice	3		1				1
65	Lieskovsky Adam	Gym. Jura Hronca BA	3		1				1
65	Matuš Juraj	Gym. Jura Hronca BA	2	1					1
69	Kollár Dan	Gym. Grösslingová BA	2		0				0
69	Orenič Jozef	Gym. Slovenská Bardejov	4	0					0
69	Václavík Lukáš	SSOĽ Via Humana	2	0					0
69	Šafin Jakub	Gym. P. Horova Michalovce	1		0				0
69	Šormanová Mária	Škola pre mim. nadané deti BA	3		0				0

Výsledková listina po 1. kole kategórie KSP-T

	Meno a priezvisko	Škola	Trieda	8	9	10	Σ
1	Belan Tomáš	Škola pre mim. nadané deti BA	4	20	14		34
2	Hudec Tobiáš	Gym. Partizánske	4	25			25
3	Jankovič Radovan	Gymnázium Levice	3	1	23		24
4	Rohár Pavol	Gym. M. R. Štefánika, Košice	4	2	6		8
5	Novella Tomáš	Gym. Alejová Košice	4			5	5
6	Pitoňák Martin	Gym. Tajovského B. Bystrica	4	2			2
7	Cuc Bruno	Gym. Grösslingová BA	4	1			1
7	Kekely Michal	Gym. Varšavská Žilina - Vlčince	3	1			1
7	Mrocková Mária	Gym. Jura Hronca BA	3	1			1
7	Varga Mátyás	Gym. H. Selyeho Komárno	3	1			1
7	Vozárová Zuzana	Gym. Jura Hronca BA	2	1			1
12	Dang Quoc Trung	Gym. Jura Hronca BA	2	0			0
12	Lampanen Mika	Gym. Pankúchova Bratislava	3		lolo:)	NOOB	0

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. LPP-0103 -09