Korešpondenčný seminár z programovania
XXIV. ročník, 2006/07
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.
Gnome spol. s r.o.
Whitestein Technologies spol. s r.o.

Vzorové riešenia 1. kola zimnej časti

Zima by sa k nám mala pomaličky blížiť, ale nejako sa jej nechce... Nič ale nemení na tom, že sme vám opravili vaše riešenia (niektoré boli naozaj originálne) a napísali pre vás vzoráky, aby ste boli múdrejší a nabudúce mali ešte viac bodov ako teraz. Tak šup šup do čítania, aby ste nedopadli ako my v Budapešti...

Jahody su milé zvieratká. Pokiaľ vás nemienia zjesť...

KSPáci

1. Zaveštíme si trošičku

opravoval Stano
(max. 15 bodov)

Príkladov prišlo požehnane, kopa z vás mala riešenie podobné vzorovému. Ale aj napriek tomu sa v riešeniach vyskytovali isté chyby. Spomeniem tie najčastejšie:

Niektorí zabudli na 0 a záporné čísla (tie sa však nepočítali ako chyba `:-)`). Pre záporné čísla stačí z N vyrobiť jeho absolútnu hodnotu, a nulu sa občas oplatí ošetriť samostatne.

Začnime najskôr riešeniami, ktoré sa uspokojili s tým, že simulovali postup zo zadania. Aké rýchle je takéto riešenie?

Potrebujeme spočítať všetky cifry čísla N. Koľko ich má? No rádovo ich bude $\log_{10} N$. (Ako na toto prísť? Nuž, číslo 10^K je najmenšie s $K+1$ ciframi...) Ak sa teda dohodneme, že veci ako delenie so zvyškom (teda odtrhnutie poslednej cifry) vieme robiť v konštantnom čase, potrebujeme na nasčítavanie cifier čísla N spraviť $O(\log N)$ operácií.

Postup budeme možno opakovať niekoľkokrát, no všetky ďalšie čísla, s ktorými ho budeme robiť, sú oproti tomu zo vstupu zanedbateľne malé. Takže nad nimi hodíme rukou a uspokojíme sa s tým, že takéto riešenie má časovú zložitosť $O(\log N)$.

No a teraz sa už vrhnime na riešenie. Najprv si dokážeme malú lamu (teda vlastne lemu):

Lema: *Číslo modulo 9 je rovné (ciferném súčtu čísla) modulo 9.*

Dôkaz: Majme číslo $A = \overline{a_n a_{n-1} \cdots a_1 a_0}$. Môžeme ho napísať ako

$$\begin{aligned} A &= 10^n a_n + 10^{n-1} a_{n-1} + \cdots + 10^2 a_2 + 10^1 a_1 + 10^0 a_0 \\ &= (10^n - 1)a_n + a_n + \cdots + (100 - 1)a_2 + a_2 + (10 - 1)a_1 + a_1 + a_0 \\ &= [(10^n - 1)a_n + (10^{n-1} - 1)a_{n-1} + \cdots + 99a_2 + 9a_1] + [a_0 + a_1 + a_2 + \cdots + a_{n-1} + a_n] \end{aligned}$$

Číslo $(10^i - 1)$ je číslo tvaru $99\cdots9$, ktoré sa skladá so samých deviatok, takže je vždy deliteľné číslom 9. Preto A dáva rovnaký zvyšok po delení 9 ako súčet zvyšných členov. To je ale práve súčet $a_0 + \cdots + a_n$, teda ciferný súčet čísla A.

Keď už máme túto lemu, môžeme ju použiť ako kladivo na príklad. Vieme, že spravením jednej operácie zo zadania sa nám nezmení zvyšok, ktorý dáva naše číslo po delení deviatimi. Aj výsledné MVČ, ktoré dostaneme niekoľkonásobným zopakovaním operácie zo zadania, bude teda dávať po delení deviatimi rovnaký zvyšok ako N. Tým je ale MVČ jednoznačne

určené: Pre $N = 0$ je to zjavne 0, pre kladné N je aj MVČ kladné, a teda je to jedno z čísel od 1 do 9. Ak je $N \bmod 9 = 0$, MVČ je rovné 9, ináč je rovné ($N \bmod 9$).

Listing programu:

```
var n:integer;
begin
  readln(n);
  if (n<0) then n:=-n;
  if (n<>0) then if (n mod 9=0) then n:=9 else n:=n mod 9;
  writeln(n);
end.
```

2. Zničený autobus

opravoval Luki
(max. 10 bodov)

Tento príklad nebol príliš ťažký a väčšina z vás si s ním hravo poradila, keďže je veľmi ľahké predstaviť si, ako sa autobusy kazia. Jediné, čo bolo potrebné naprogramovať, je spôsob, ako sa má kaziť zadaný autobus, keď máme zadanú spoľahlivosť N.

Najjednoduchšie riešenia vyzerali nasledovne: N krokov sa autobus nepokazí a následne sa raz pokazí a zastane. Tento postup sa stále opakoval. Toto riešenie som ohodnotil na 7 bodov.

Lepšie riešenia využívali náhodné čísla, ale častokrát neriešili prípad, keď autobus je spoľahlivý na 100%, takéto riešenia boli ohodnotené na 8 bodov.

A najlepšie riešenia boli asi takéto: v každom kroku vygenerujeme náhodné číslo z intervalu $1, \ldots, 100$ a porovnáme ho so zadanou spoľahlivosťou. Ak je vygenerované číslo väčšie ako naša spoľahlivosť, tak sa pokazí. Teda čím je väčšie N, tým je menej čísel, ktoré sú väčšie ako N a menšie alebo rovné 100. Až keď je $N = 100$, tak neexistuje žiadne také číslo, a teda autobus bude stále fungovať. A toto sa vykonávalo, až pokiaľ nebol stlačený kláves. Kto napísal toto alebo niečo podobné, dostal 10 bodov.

Bodovanie bolo veľmi mierne, keďže väčšina riešení spĺňala požiadavky zadania. Body sa dali stratiť aj za to, že autobus išiel a potom sa pokazil a koniec. Za takúto chybu som dával -1 bod. Keď sa vyskytla nejaká chyba alebo bol nedostatočný popis, tak som podľa vážnosti strhol nejaké body.

Pamäťová zložitosť je konštantná $O(1)$, lebo nepoužívame žiadne pole, ale iba konštantný počet jednoduchých premenných. Časová zložitosť je taktiež konštantná $O(1)$ – nezáleží od zadanej spoľahlivosti.

Nasleduje MišoFova drobná vsuvka na dlhé zimné večery. Ak nebudete mať niekedy čo robiť, skúste sa začítať, povieme si o tejto zdanlivo ľahkej úlohe niečo viac `:-)`.

Na tomto mieste by bolo vhodné položiť si otázku: a ako sa naozaj kazia autobusy? Podobá sa realita nami zvolenému modelu? To, čo by sme mohli spraviť, keby sme neboli leniví, je zájsť do dopravného podniku, pozrieť si knihy opráv jednotlivých autobusov, zaznačiť si do grafu jednotlivé intervaly medzi poruchami, a potom odhadnúť a preložiť tým nejakú čo najjednoduchšiu krivku.

Keďže ale leniví sme, opýtame sa uja Googla. Ak napríklad budeme merať, koľkokrát sa nám autobus pokazí za jeden rok, dostaneme podobné rozdelenie hodnôt, ako keby sme merali, koľkokrát sa nám vypáli ktorá žiarovka, koľko áut prejde ktorý deň medzi treťou a štvrtou popred školu, či koľko ľudí denne navštívi náš server. Všetky tieto náhodné premenné majú rozloženie pravdepodobnosti, ktoré sa volá Poissonovo, a dočítate sa o ňom viac na adrese `http://en.wikipedia.org/wiki/Poisson_distribution`.

No ale nás zaujíma niečo trochu iné ako to, koľkokrát sa nám autobus pokazí za rok. My by sme chceli vedieť generovať vierohodne vyzerajúce čísla, ktoré budú hovoriť, kedy sa

ktorý autobus pokazí prvýkrát. Teda zaujíma nás náhodná premenná, ktorá hovorí, koľko času uplynie, kým prvýkrát nastane jav, ktorý má Poissonovo rozloženie pravdepodobnosti. Na vyššie uvedenej adrese sa môžeme dočítať, že takýto jav má rozloženie pravdepodobnosti, ktoré voláme exponenciálne. Opäť, detaily a vzorce vám prezradí Wikipedia na adrese `http://en.wikipedia.org/wiki/Exponential_distribution`.

No a dočítame sa tam aj toľko, že exponenciálne rozdelenie pravdepodobnosti má aj brata, ktorý nie je spojitý. Keď budeme hádzať mincou, ktorá má pravdepodobnosť p že padne znak, a rátať počet hodov do prvého znaku, dostaneme približne exponenciálne rozdelenie so strednou hodnotou $1/p$.

No a na záver, keď sme sa už pekne potešili, že naozajstné autobusy sa kazia rovnako ako naše, ukážeme si ešte, ako si môžeme pomôcť tým, že teraz toho o nich vieme viac. Keby sme robili veľkú počítačovú hru, bolo by zbytočne nešikovné v každom takte si pre každý autobus hádzať mincou či sa pokazí. Lepšie je si pre každý autobus v okamihu poruchy rovno vygenerovať, kedy bude mať najbližšiu.

A tu prichádza k slovu exponenciálne rozloženie pravdepodobnosti. Premennú p s takýmto rozložením totiž vieme vygenerovať z „obyčajnej“ náhodnej premennej $r \in \langle 0, 1)$ takto: $p = -\ln r$. No a teraz to už len naškálujeme podľa toho, aký má byť autobus spoľahlivý, a máme to.

Koniec vsuvky, ďakujeme za pozornosť.

Listing programu:

```
program zniceny_autobus;
uses crt;
const
  autobus_vyska: integer = 4;
  autobus_art: array[1..2,1..4] of string = ((
  '    .----------------.  ',
  ' ___[I_I_I_|39|I_I_I_|  ',
  '(  _ ; :          _  |  ',
  ' -(0)-=----------(0)-=  '),
  (
  '   .----------------.   ',
  '___[\_\X\_\_\#\_\_\_|   ',
  '|  _ ; :          _  |  ',
  ' -cob-~~-u-|6E|/\\cxw-= '  ));

type TStav = (Funguje, Pokazeny);

var spolahlivost : integer;
    x, y : integer;

procedure autobus (x, y : integer; s: TStav);
var tmp : integer;
begin
  for tmp := 1 to autobus_vyska do begin
    gotoxy(x, y + tmp - 1);
    if s = Funguje then write(autobus_art[1][tmp])
                   else write(autobus_art[2][tmp]);
  end;
end;

begin
  readln(spolahlivost);
  randomize;
  clrscr;
  x := 56; y := 1;
```

```
  while not keypressed do begin
    if (random(100)+1) > spolahlivost then begin
      autobus(x, y, Pokazeny);
      delay(1000);
    end else begin
      x := x - 1;
      if x = 0 then begin
        clrscr;
        x := 57;
        y := y+1;
      end;
      if y >= 47 then y := 1;
      autobus(x, y, Funguje);
      delay(50);
    end;
  end;
end.
```

3. Zúfalí Zimbabwejskí zvedavci

Opravovala Elφnka
(max. 15 bodov)

Koalície budeme hľadať nasledovne. Vyskúšame všetky možné kombinácie strán do koalície a tie, ktoré nám budú vyhovovať, vypíšeme. Také skúšanie všetkých možností nazývame „backtrack“ a robí sa takto: Budeme postupne vyrábať koalície tak, že určíme prvú stranu, ktorá tam bude patriť, a potom k nej skúsime pridať ako druhú stranu všetky možné ďalšie. Toto sa robí rekurziou. Na začiatku kroku máme už nejaký zoznam strán, ktoré skúšame dať spolu do koalície. Potom prejdeme všetky ostatné strany a skúsime ich pridať, t.j. for-cyklom prejdeme všetky ešte nevyskúšané strany, a pre každú spravíme nasledovné: pridáme ju ku koalícii a zavoláme znova našu procedúru. Ďalší krok pôjde už s touto novou stranou.

Pri tom musíme dať pozor na 3 veci:

1. aby boli spolu iba strany, ktoré môžu,
2. aby mali správny počet hlasov,
3. aby sme nemali nejakú stranu navyše,
4. a aby sme nevypísali nejakú koalíciu viackrát

Poďme sa teda pozrieť na to, ako sa vysporiadať s jednotlivými problémami:

1. Keď už máme potenciálnu koalíciu a chceme k nej pridať ďalšiu stranu, overíme, či nemá psychické problémy s nejakou inou, ktorá už v danej koalícii je; a ak nie, tak ju pridáme.
2. Overíme jednoducho – stále si počítame, koľko hlasov má doteraz vytvorená koalícia a skúsime ju vypísať, až keď ich bude mať dosť.
3. Ak už máme nájdenú potenciálnu koalíciu so správnym počtom hlasov, treba overiť, či v nej všetky strany ozaj musia byť. Ak náhodou nie (t.j. existuje strana, ktorá keby tam nebola, tak bude počet hlasov stále vyhovujúci), tak ju nevypíšeme. Všimnite si, že určite nájdeme koalíciu, ktorá by vznikla vynechaním nepotrebnej strany – dostaneme sa k nej totiž inou vetvou prehľadávania.
4. Na to, aby sme našli všetky potenciálne koalície nám stačí, aby sme hľadali iba koalície usporiadané podľa čísel strán, t.j. nájdeme len koalíciu 1, 2, 6, ale už nie 1, 6, 2. Na toto nám stačí, aby sme hľadali ku koalícii len strany, ktoré majú väčšie poradové číslo ako posledná pridaná strana.

Prehľadávame všetky možnosti. V najhoršom prípade vyskúšame všetky možné podmnožiny, tých je 2^N a pre každú spravíme navyše rádovo N operácií (overenie, či môžu byť spolu a či ich nestačí menej). Časová zložitosť je preto $O(N2^N)$. Aby sme mohli v konštantnom

čase určiť, či môžu byť dve strany v koalícii, použijeme pole (`strany`), kde máme pre každú dvojicu uložené `true`/`false` – či môžu, alebo nie. Pamäťová zložitosť teda bude $O(N^2)$.

Listing programu:

```
const max=100;

var strany: array[1..max,1..max] of boolean;
    hlasy: array[1..max] of integer;
    sucet,N: integer;
    i,j: integer;
    koalicia: array[1..max] of integer;
    overujeme: boolean;

procedure over(pocet,hlasov:integer);
var dobra: boolean;
    i,j: integer;
begin
  dobra:=true;
  for i:=1 to pocet do
    if hlasov-hlasy[koalicia[i]]>=sucet then dobra:=false;
  if dobra then begin
    for i:=1 to pocet do
      write(koalicia[i],' ');
    writeln;
  end;
end;

procedure hladaj(hlasov,ktore,kolko:integer);
var i,j:integer;
begin
  if (hlasov>=sucet) then over(kolko,hlasov)
  else
    if (ktore<N) then begin
      for i:=ktore+1 to N do begin
        overujeme:=true;
        for j:=1 to kolko do
          if strany[i][koalicia[j]] then overujeme:=false;
        if (overujeme) then begin
          koalicia[kolko+1]:=i;
          hladaj(hlasov+hlasy[i],i,kolko+1);
        end;
      end;
    end;
end;

begin
  readln(N);
  sucet:=0;
  for i:=1 to N do begin
    read(j);
    sucet:=sucet+j;
    hlasy[i]:=j;
    read(j);
    while j<>0 do begin
      strany[i][j]:=true;
      strany[j][i]:=true;
      read(j);
    end;
  end;
```

```
  sucet:=sucet div 2 + 1;
  hladaj(0,0,0);
end.
```

opravoval Mišo & Maják
(max. 15 bodov)

4. Zlostný generál

Väčšina z vás tento príklad vyriešila veľmi jednoducho – keďže sa hľadá N-tý najmenší prvok dvoch usporiadaných polí, v čase $O(N)$ ste ich spojili tak, ako sa to robí v mergesorte (dáme pointer na začiatok každého poľa; v každom kroku porovnáme dva prvky, na ktoré ukazujú, vyberieme do výsledného poľa menší z nich a príslušný pointer posunieme na ďalší prvok), prípadne to utriedili pomocou horších triediacich algoritmov ($N \log N, N^2, N^3$). Avšak na plný počet bodov bolo treba viac.

Logaritmická zložitosť sa dala dosiahnuť napríklad takto: pre medián oboch polí platí, že je N-tý v poradí, t.j. $N-1$ prvkov je od neho menších a N prvkov väčších. Túto podmienku dokážeme pre ľubovoľný prvok overiť v konštantnom čase. Využijeme pri tom fakt, že polia sú už usporiadané. Ak je náš overovaný prvok v prvom poli i-ty, tak pred ním je v tom poli $i-1$ menších prvkov. Z toho vyplýva, že aby to bol medián, v druhom poli ich musí byť práve $N-1-(i-1) = N-i$ menších. To znamená, že $(N-i)$-ty bude menší, ale $(N-i+1)$-vý už musí byť väčší (alebo rovný).

A ako nájdeme ten prvok, pre ktorý toto platí? Binárnym vyhľadávaním. Funguje nasledove: Overíme si stredný prvok prvého poľa, ak to preň platí, tak je to náš hľadný medián, a vyhrali sme. Ak je od neho menších prvkov viac ako $N-1$, tak je príliš veľký, a budeme ďaľej hľadať naľavo od neho. Ak je menších menej ako $N-1$, tak je príliš malý, a preto ho budeme ďalej hľadať v pravej polovici poľa. Tým sa nám úsek poľa, v ktorom sa má náš medián nachádzať, scvrkol na polovicu. A zase použijeme rovnaký postup – pozrieme sa na stredný prvok a podľa neho sa nám aj tento úsek poľa scvrkne na polovicu, a ten na polovicu, a ten na polovicu... Áno, je to rekurzia (našťastie nie nekonečná – končí, keď nám podmienka pre ten stredný prvok vyhovuje). Ak nám nakoniec ostal už len jeden prvok, a ani preň naša podmienka nevyhovuje, znamená to, že medián je v druhom z polí. V tom prípade zistíme medián presne rovnakým spôsobom, len s tou výnimkou, že binárne vyhľadávať budeme v druhom poli.

Aký rýchly je tento postup? Veľmi. Nazvime „pozri sa do stredu, ak to nie je medián hľadaj v ľavej alebo pravej polovici“ jeden krok. Keby bolo $N = 1$, stačí nám na nájdenie mediána (v jednom poli) jeden krok. Keby sme však mali k dispozícií kroky dva, vedeli by sme úlohu riešiť pre $N = 3 = 1+1+1$. Na tri kroky už vieme riešiť úlohu pre $N = 7 = 3+1+3$ (ak to nie je stredný prvok, budeme riešiť úlohu pre $N = 3$ a to vieme na dva kroky). Keby sme mohli počítať štyri kroky, vedeli by sme úlohu riešiť pre $N = 15 = 7+1+7$ (jediným krokom totiž buď nájdeme medián alebo zmenšíme pole na polovicu). Pridaním ďalšieho kroku sa nám N, pre ktoré vieme úlohu riešiť, približne zdvojnásobí. Stačí k krokov na to, aby sme vedeli vyriešiť úlohu (v prvom poli) pre $N \leq 2^k - 1$. V najhoršom prípade budeme musieť celý tento postup zopakovať pre druhé pole, v každom prípade však bude N rádovo 2^k. To je exponenciálne veľa! V takejto situácií hovoríme, že zložitosť je logaritmická, $O(\log N)$. Na nájdenie mediána potrebujeme rádovo $\log_2 N$ krokov.

Okrem polí zo vstupu sme použili iba konštantný počet premenných naviac, a áno, hádate správne, pamäťová zložitosť bude teda $O(1)$.

Riešenie v čase $O(\log N)$ bolo za 15 bodov, riešenie v čase $O(N)$ bolo za 12 bodov, horšie riešenie za 8 bodov. Za chýbajúce odhady zložitosti sme strhávali po jednom bode. Za chýbajúci/nedostatočný popis sme strhávali najviac polovicu zvyšných bodov.

Listing programu:

```
const max=100;
type pole=array[1..max] of integer;
var A,B:pole;
    N,i,vysledok:integer;

function binSearch(A,B:pole):integer;
var l,r,s:integer;
begin
   l:=1;r:=N;
   while l <= r do begin
     s := ((l+r) div 2);
     if (s=N)and(B[N-s+1]<A[s]) then r:=s-1
     else if (s=N)and(B[N-s]>A[s]) then begin
       binSearch:=A[s];
       exit;
     end
     else if (B[N-s]>A[s])and(B[N-s+1]>A[s]) then l:=s+1
     else if (B[N-s]<A[s])and(B[N-s+1]<A[s]) then r:=s-1
     else begin
       binSearch:=A[s];
       exit;
     end;
   end;
   binSearch:=-1;
end;

begin
  readln(N);
  for i:=1 to N do read(A[i]);
  for i:=1 to N do read(B[i]);
  vysledok:=binSearch(A,B);
  if vysledok=-1 then writeln('Median je ',binSearch(B,A)) else writeln('Median je
',vysledok);
end.
```

5. Zase raz Kapingamarangi

opravoval w
(max. 15 bodov)

Po prečítaní zadania si asi väčšina z vás preložila zadanie na *odskúšaj všetky obdĺžniky a vypíš koľko si videl tých s maximálnym zážitkom*.

Štandardný postup rátania zážitku pre obdĺžnik – vyberiem si obdĺžnik a pustím po jeho hranách cykly, ktoré zrátajú celkový zážitok. Možných obdĺžnikov je zhruba $R^2 \cdot S^2$. Pre každý obdĺžnik ešte musíme prejsť po jeho obvode (v čase $O(R+S)$), odkiaľ dostávame odhad zložitosti triviálneho riešenia: $O(R^2 \cdot S^2 \cdot (R+S))$.

Možné zlepšenie – pokiaľ by sme vedeli zážitok v ľubovoľnom obdĺžniku povedať v konštantnom čase, potom zjavne vylepšíme časovú zložitosť na $O(R^2 \cdot S^2)$. Uvedieme len stručnú myšlienku jedného možného takéhoto riešenia: Prepíšeme každú 0 na -1. Pre každé $[i,j]$ spočítame súčet hodnôt v obdĺžniku s rohmi $[1,1]$ a $[i,j]$. (Toto vieme ľahko spraviť v čase $O(R^2 \cdot S^2)$, ale ide to dokonca spraviť v $O(RS)$.) Pomocou týchto hodnôt vieme ku ľubovoľnému obdĺžniku spočítať v čase $O(1)$ súčet hodnôt v ňom. (Ak neviete ako, nakreslite si ľubovoľný obdĺžnik a predĺžte jeho hrany až po okraj. Potom už len pozerať, kým to neuvidíte.) A zážitok pre obdĺžnik je predsa (súčet celého obdĺžnika) mínus (súčet jeho vnútra).

Ale poďme sa pozrieť na vzorové riešenie, ktoré bude ešte rýchlejšie. Použijeme pri tom techniku, ktorú nazývame dynamické programovanie. Táto technika je v princípe veľmi jednoduchá. Dá sa použit práve vtedy, keď vieme náš problém vypočítať z menších podproblémov.

Pozrime sa na to, ako vyzerá taký priemerný obdĺžnik. Taký hranatý. Rozdeľme ho na dva útvary a to tak, že zoberieme spodnú hranu. Ostane nám také prevrátené U a spodná hrana. Zjavne keď zrátame zážitok na prevrátenom U so zážitkom na spodnej hrane, tak dostaneme zážitok na celom obdĺžniku.

Zjednodušme si úlohu, a predstavme si, že máme k dispozící milú vílu Amálku[1], ktorá nám prezradí, kde má náš hľadaný obdĺžnik ľavý a pravý okraj. Ako by sme vedeli využiť túto vedomosť?

Budeme postupne spracúvať kusy riadkov, ktoré ležia v páse medzi poradeným ľavým a pravým stĺpcom. Predstavme si, že pre $i-1$ riadok vieme povedať aké najlepšie U končí na riadku $i-1$. Aký najlepší obdĺžnik končí na i-tom riadku? Označme U_k zážitok najlepšieho U, ktoré končí na riadku k a nech St_k je zážitok na spodnej strane obdĺžnika, ktorý končí na k-tom riadku[2]. Potom cena najlepšieho obdĺžnika končiaceho na i-tom riadku je $\max(St_i, U_{i-1} + St_i)$, teda buď zoberieme najlepšie U a pridáme k nemu náš riadok, alebo zoberieme iba riadok, lebo U je veľmi nudné. Ešte by sme ale radi vyrátali cenu U_i. Zrátame cenu predĺženého U_{i-1}, čiže k U_{i-1} pridáme začiatok a koniec i-teho riadku. U_i sa bude rovnať maximu z tejto hodnoty a St_i. Prečo? No buď zoberieme najlepšie U a predĺžime ho, alebo ešte za U môžeme prehlásiť i-ty riadok, lebo je vačší ako predĺžené U.

Náš algoritmus je už jasný. Zoberieme prvý riadok a prehlásime ho za U_1 aj za najlepší obdĺžnik. Každý ďalší riadok spracujeme podľa predchádzajúceho postupu. Ale ešte nám tu zopár vecí chýba. Napríklad rátanie počtu takých obdĺžnikov. To ale nie je zložité. Potrebujeme vedieť akurát, koľko existuje najzaujímavejších U nado mnou. Vždy, keď nájdem rovnako zaujímavý obdĺžnik ako zatiaľ najzaujímavejší, tak tento počet prirátam k počtu najzaujímavejších obdĺžnikov.

Ďalší detail, na ktorý sa nesmie zabudnúť, je to, že by sa patrilo rátať zážitok v riadku v konštantnom čase. To vyriešime nasledovným trikom. Pre každý riadok budeme mať vyrátaný nasledovný ťahák: $T_{i,j}$ je zážitok pre úsek i-teho riadku, ktorý obsahuje stĺpce 1 až j. Pomocou ťaháku ľahko vyrátam zážitok pre úsek od stĺpca a po stĺpec b: je to $T_{i,b} - T_{i,a-1}$.

Pozrime sa na celé riešenie. V čase $O(R \cdot S)$ vieme vyrátať náš ťahák. V čase $O(R)$ vieme nájsť najzaujímavejšie obdĺžniky pre jednu dvojicu stĺpcov. Takých dvojíc je $S \cdot (S-1)/2$. Ak vyskúšame všetky rozumné dvojice[3] stĺpcov, tak sa dostaneme k algoritmu so zložitosťou $O(S^2 \cdot R)$, čo však nemusí zaručene byť minimálne možné. Čo ak $S > R$? Nie je nám potom výhodnejšie rátať algoritmom s vymenenými riadkami za stĺpce? Je. Dokonca si to môžme dovoliť, lebo počty jednotlivých zážitkov tým nezmeníme. Tzn. v prípade potreby načítame mapu „pootočenú“ a dostávame výslednú časovú zložitosť $O(R \cdot S \cdot \min(R,S))$ a pamäťovoú zložitosť $O(R \cdot S)$.

Bodovanie: Za riešenie $O(R^2 \cdot S^2 \cdot (R+S))$ sa dalo získať max. 10 bodov. Za riešenie $O(R^2 \cdot S^2)$ sa dalo získať 12 bodov. Riešenie $O(R \cdot S^2)$ alebo $O(R^2 \cdot S)$, no nie $O(R \cdot S \cdot \min(R,S))$ bolo ohodnotené najviac 14 bodmi a za riešenie s min sa dalo získať 15 bodov. -5 bodov za absenciu zdrojáku, po -1 bodov za zlý odhad zložitosti, po -2 bodov za žiadny.

Listing programu:

```
program zazitky;

const iMax = 100;
      minNek = -32000;
```

[1] nemýliť si s istým vedúcim KSP

[2] Uvedomte si, že je to tiež obdĺžnik, a dokonca je to aj také degenerované U.

[3] keďže vílu Amálku máme len v našich predstavách :-(

```
var cena, zazitok: Array[1..100, 1..100] of Integer;
    r, s, i, j, k, l, maxZ, pocet, uVal, uPoc, hrana: Integer;
    f: Text;

begin
  readln(r, s);

  if (s < r) then { riadkov je viac ako stlpcov }
    for j := 1 to r do begin
      for i := 1 to (s - 1) do begin
        read(zazitok[i,j]);
        zazitok[i,j] := zazitok[i,j]*2 - 1;
        cena[i,j] := zazitok[i,j];
      end;
      readln(zazitok[s,j]);
      zazitok[s,j] := zazitok[s,j]*2 - 1;
      cena[s,j] := zazitok[s,j];
    end
  else { stlpcov je aspon tolko, co riadkov }
    for j := 1 to r do begin
      for i := 1 to (s - 1) do begin
        read(zazitok[j,i]);
        zazitok[j,i] := zazitok[j,i]*2 - 1;
        cena[j,i] := zazitok[j,i];
      end;
      readln(zazitok[j,s]);   zazitok[j,s] := zazitok[j,s]*2 - 1;   cena[j,s] :=
zazitok[j,s];
    end;

  if (s>r) then  begin i := s; s := r; r := i; end;

  maxZ := minNek; { dlzka trasy s najvacsim zazitkom }
  pocet := 0;      { pocet tras s maximalnym zazitkom }

  { predratame silu zazitku po riadkoch zlava }
  for j := 1 to r do
    for i := 2 to s do
      cena[i,j] := cena[i-1,j] + cena[i,j];


  for i := 1 to s do { lavy okraj pasu  }
    for j := i to s do begin { pravy okraj pasu }
      { prvy riadok }
      { hrana = rozdiel medzisuctov zlava + ten vrchol, co sme odratali 2x }
      if (i <> j) then uVal := cena[j,1] - cena[i,1] + zazitok[i,1]
                  else uVal := zazitok[i,1];

      uPoc := 1;
      if (uVal > maxZ) then begin { nove najlepsie riesenie }
        maxZ := uVal;
        pocet := 1;
      end else
        if (uVal = maxZ) then  { rovnake ako zatial najlepsie }
          inc(pocet);

      for k := 2 to r do begin
        { rozdiel medzisuctov + 2x odratany vrchol }
        if (i <> j) then
          hrana := cena[j,k] - cena[i,k] + zazitok[i,k]
```

```
        else
          hrana := zazitok[i,k];

        { ak by sa to tymto riadkom malo koncit }
        if (uVal + hrana > hrana) then begin
          if (uVal + hrana = maxZ) then   { a navyse je to maximalny zazitok }
            inc(pocet, uPoc)              { priratame pocet moznych tras }
          else
            if (uVal + hrana > maxZ) then { lepsi ako maximalny zazitok }
            begin   maxZ := uVal + hrana;   pocet := uPoc;   end
        end else begin
          if (hrana > maxZ) then begin
            maxZ := hrana;
            pocet := 1;
          end else if (hrana = maxZ) then inc(pocet);
        end;

        if (i <> j) then l := uVal + zazitok[i,k] + zazitok[j,k]
                    else l := uVal + zazitok[i,k];

        if (l > hrana) then uVal := 1 { volba noveho Ucka }
        else begin
          if (l = hrana) then begin { mame o jednu trasu s rovnakym zazitkom viac }
            uVal := l;
            inc(uPoc);
          end else begin { oplati sa mi tu zacat odznova }
            uVal := hrana;
            uPoc := 1;
          end;
        end;
      end;
    end;

  writeln(maxZ, ' ', pocet);
end.
```

opravoval Zemčo
(max. 15 bodov)

6. O Novom Jorku a teroristoch

Dostavilo sa toho ku mne vcelku dosť, no akosť ma veľmi neuspokojila. Väčšina z vás vyprodukovala len najjednoduchšie riešenie. Iba zopár sa pokúsilo vymyslieť niečo lepšie a niekedy to aj prekvapujúco dobre dopadlo. Takže na úvod pochvala všetkým tým, čo poslali vzorové riešenie alebo niečo, čo sa naň ponášalo. Ako si ukážeme, nebolo to nič extra zložité.

Poďme sa ale najprv pozrieť na tú prvú, početnejšiu skupinu. Vás pracovne nazveme simulanti, a to preto, lebo celý dejový vývoj v Novom Jorku simulujete. Máte jedno pole, v ktorom si pamätáte výšky budov. Update veľkosti budovy takto riešite v konštantom čase, keď ale letí lietadlo, prebehnete celé pole a hľadáte dotyčnú budovu, ktorej dá prelietavajúce lietadlo pusu. Toto nás stojí lineárny čas od počtu budov. Netvrdím, že toto riešenie nemá svoje pozitíva, pretože sa naozaj ľahko implementuje, ale práve ten lineárny čas nám pri ňom vadí. A tým sa dostaneme k vzorovému postupu.

Vzorák[4] a jeho časová zložitosť je prudko závislá na použitej dátovej štruktúre. V našom prípade to bude intervalový strom. Poďme si teda ukázať, ako také niečo vyzerá a funguje. Predstavme si pole, v ktorom máme uložené výšky budov presne ako v jednoduchom riešení.

[4] podobne ako väčšina úloh kde máte spracovávať viac typov požiadaviek

Čo musíme spraviť najsamprvé je, že jeho dĺžku zväčšíme na najbližšiu mocninu dvojky. Bude to najviac dvojnásobok pôvodnej dĺžky, a preto nám to nijak veľmi nevadí a ani nedegraduje pamäťovú zložitosť. V umelo pridanej časti budeme mať konštantne nuly. Teraz nad týmto poľom vytvoríme úplný binárny strom – prvky pôvodného poľa budú listy[5] a nad každou dvojicou políčok pôvodného poľa dáme nové políčko. Vyššie poschodie má presne polovicu prvkov ako to nižšie. Takto to budeme postupne robiť, až kým nedôjdeme do koreňa. A keďže máme vhodnú dĺžku pôvodného poľa, akurát nám to pekne vyjde.

Teraz si povieme, čo si budeme v jednotlivých vrcholoch stromu pamätať. Podľa tohto poznáme viacero typov intervalových stromov. Ten náš bude maxový, to znamená, že ak má vrchol deti, tak si pamätá v sebe maximum z týchto dvoch detí. V liste je uložená informácia o výške budovy na danom mieste. Keď si takto predstavíme strom, vidíme, že v každom vrchole je uložená výška najvyššej budovy v intervale, ktorý leží pod týmto vrcholom – preto sa strom volá intervalový. V takomto strome budeme schopní efektívne odpovedať na požadované otázky.

Jednoduchšia operácia je zmena výšky budovy. Upravíme príslušný list. Čo sa nám ale teraz mohlo pokaziť? Informácia v niektorých vrcholoch nado mnou. Avšak len v tých, ktoré sú na ceste od zmeneného listu ku koreňu – premyslite si, že naozaj len v tých. Teraz sa už len teda stačí vyšplhať hore[6] a pekne skontrolovať a prípadne upraviť. Pre každé poschodie hĺbky stromu robíme konštante veľa operácií, a preto je náročnosť úmerná hĺbke stromu, čo je v prípade úplných binárnych stromov dvojkový logaritmus z počtu prvkov N.

Ako si máme ale poradiť s letiajúcim lietadlom? Prvé, čo môžeme spraviť, je pozrieť na koreň stromu, kde máme zaručene výšku najvyššej budovy v meste. V prípade, že je táto menšia ako výška letu lietadla, to určite preletí. Ak nie, pokračujeme. Predpokladajme teda, že lietadlo niekde narazí. Pozrieme sa v koreni na svoje dve deti, najprv nás bude zaujímať to ľavé. To zahŕňa tú polovicu Nového Jorku, ktorú lietadlo prelieta skôr. No a z informácie o najvyššej budove v tejto časti vieme, či tam narazí alebo nie. Ak vieme, že tam lietadlo narazí, budeme sa ďalej venovať len tejto časti mesta, v opačnom prípade musí naraziť v tej pravej a budeme sa venovať tej. Takto sme schopní v každom vrchole identifikovať tú polovicu intervalu, kde lietadlo narazí a potom sa presunúť do príslušného dieťaťa a pokračovať v rozhodovaní tam. Takto sa posúvame dole, až raz prídeme k listu a keď sa dobre zamyslíte, tak vám z toho vyjde, že to je tá inkriminovaná budova, ktorá príde do fyzického kontaku s lietadlom. A keďže opäť každé poschodie riešime v konštantom čase, celková náročnosť tejto operácie je $O(\log N)$.

Toto riešenie je oveľa silnejšie ako to z prvého odstavca. My síce nevieme, koľko bude ktorých operácií, ale vypočítajte si, o koľko viac by muselo byť úprav budov od preletov lietadla, aby nám to jednoduché riešenie vyhovovalo viac. Logaritmus je totiž najmä pre veľké N veľmi lenivo rastúca funkcia.

Toľko riešenie, ktoré ste dúfam prežuli. Ešte pár poznámok k implementácii. Ono to totiž na prvý pohľad vyzerá asi dosť zložito. Tak vedzte, že až také tragické to nie je. Intervalový strom, keďže je úplný binárny strom, sa nám pohodlne implementuje podobne ako napríklad halda v lineárnom poli. Vrchol, ktorý je v poli s indexom i má svojho rodiča na políčku s indexom $\lfloor i/2 \rfloor$ a svoje decká na miestach $2 \cdot i$ a $2 \cdot i + 1$. Zamyslime sa teraz, koľko pamäte spotrebujeme uchovaním si takéhoto stromu. Prvé poschodie je dlhé N, to nad ním má dĺžku $N/2$, to nad ním $N/4$ a tak ďalej. Súčet niečoho takéhoto je $2N - 1$, takže dostávame pamäť $O(N)$. Keď sme v nejakom liste, napríklad keď zisťujeme, kde lietadlo narazí a vieme, že index v poli je i, tak políčko budovy získame ľahko, keď si uvedomíme, že všetky poschodia nad dolným zaberajú presne $N - 1$ článkov poľa. Preto pre index i dostávame list $i - N + 1$.

Ešte veľmi rýchlo bodovanie – za ľahké riešenie bolo maximum 8 bodov a za vzorové ste mohli získať 15. Prišli ešte aj iné špekulantské riešenia v rôznych časoch, ktorým nechý-

[5] Také vrcholy, ktoré nemajú deti. Z botanickej analógie – list je niečo, kde sa strom končí (alebo začína?)

[6] už ste niekedy šplhali od listov ku koreňu?

bala nápaditosť, no chýbala im efektivita. Tie dostávali počet primeraný ich porovnaniu so štandardnými dvoma. Trestala sa samozrejme absencia kódu, nedostatočný popis a chybný odhad. To je všetko, majte sa krásne. Alebo sa aspoň majte.

Listing programu:

```
program Teroristi;
const MAX = 10000;
type TIntervalTree = object
   data : array[1..MAX] of integer;
   pocet : integer;
   procedure initialize(p:integer);
   procedure update(x,y:integer);
   function get(x:integer):integer;
end;

procedure TIntervalTree.initialize(p:integer);
begin
   pocet:=p;
end;

procedure TIntervalTree.update(x,y:integer);
{kde x, na kolko y}
var where:integer;
begin
   where := pocet - 1 +  x;
   data[where]:=y;
   where:=where div 2;
{list bol updatovany, prebubleme hore a upravujeme}
   while where > 0 do
   begin
     if data[where * 2 + 1]>data[where * 2] then data[where]:=data[where * 2+1]
     else data[where]:=data[where * 2];
     where := where div 2;
   end;
end;

function TIntervalTree.get(x:integer):integer;
var where:integer;
begin
   if data[1]<x then begin get := 0; exit; end;
   where := 1;
{Bubleme dole, kym nenajdeme list - prislusna budova}
   while (where*2+1) <= (pocet*2 - 1) do
   begin
      if data[where * 2]>=x then where:=where * 2
      else where:= where *  2 + 1;
   end;
   get :=  where - pocet + 1;
end;

var x,y,z,i,n:integer;
    strom: TIntervalTree;
begin
   readln(n);
   i:=1; while i<n do i:=i*2;
   strom.initialize(i);
   read(x);
   while x<>0 do
   begin
```

```
      read(y);
      if x = 2 then begin readln(z); strom.update(y,z); {writeall(n);} end;
      if x = 1 then
      begin
         readln;
         z:= strom.get(y);
         if z = 0 then writeln('Preleti') else writeln(z);
      end;
      read(x);
   end;
   readln;
end.
```

7. O VMVČ

opravoval KuKo
(max. 15 bodov)

Vo väčšine riešení, ktoré mi prišli, ste prišli na to, ako z najkratších ciest dĺžky k vypočítať najkratšie cesty dĺžky $k+1$. Podľa toho, ako prefíkane ste to robili, ste dostali riešenia

- chybné – 1 bod
- backtracky skúšajúce všetky možné cesty – max. 4 body
- horšie polynomiálne – do 8 bodov
- riešenia v čase $O(n^3(k+n))$ – 11 bodov
- a najlepšie riešenia v čase $O(n^3 \log k)$ – 15.

Ešte skôr ako ma odnesú aj so stoličkou od počítača (že vraj týram deti), by som vám chcel porozprávať trošku o maticiach. Ono sa to nezdá, ale matice sú úplne almighty záležitosť. Taká matica je niečo ako dvojrozmerné pole. My budeme pracovať iba so štvorcovými maticami, takže to bude niečo ako pole $n \times n$.

Matici si navyše vymysleli, ako takéto matice[7] sčítať a násobiť: ak $C_{i,j}$ označíme číslo v i-tom riadku a j-tom stĺpci matice C, tak súčet matíc $C = A + B$ je taká matica, ktorej prvky sú $C_{i,j} = A_{i,j} + B_{i,j}$ – teda sčítavame prislúchajúce políčka.

Sčítanie matíc bolo len tak na rozcvičku. Oveľa zaujímavejšia operácia s maticami je násobenie. Súčin matíc $D = A \cdot T$ vypočítame tak, že

$$D_{i,j} = A_{i,1} \cdot T_{1,j} + A_{i,2} \cdot T_{2,j} + A_{i,3} \cdot T_{3,j} + \cdots + A_{i,n} \cdot T_{n,j}.$$

To znamená, že políčko s indexami i a j vypočítame tak, že z prvej matice si posvietime na i-ty riadok, z druhej matice si posvietime na j-ty stĺpec a začneme násobiť: prvé číslo v i-tom riadku s prvým číslom v j-tom stĺpci, druhé s druhým, atď. – všetky tieto súčiny nakoniec sčítame. Skúste dobre požuť.

Vezmime si teraz nejaký graf MHD. Ten bude zadaný dvoma maticami susednosti – $A_{u,v}$ = [medzi zastávkami u a v premáva autobus] a $T_{u,v}$ = [medzi zastávkami u a v premáva trolejbus] ($A_{u,v}$ a $T_{u,v}$ je `true` alebo `false`). Ako zistíme, či sa dá z nejakej zastávky i dostať na zastávku j tak, že pôjdeme najskôr autobusom a potom trolejbusom? No dá sa tak dostať vtedy, keď existuje zastávka k taká, že medzi i a k ide autobus (t.j. $A_{i,k}$) a medzi k a j ide trolejbus ($T_{k,j}$). Teda

$$D_{i,j} = (A_{i,1} \wedge T_{1,j}) \vee (A_{i,2} \wedge T_{2,j}) \vee (A_{i,3} \wedge T_{3,j}) \vee \cdots \vee (A_{i,n} \wedge T_{n,j}).$$

Nepripomína vám to niečo?

[7] Pozor, matici a matice sú dva rôzne živočíšne druhy, nie samci a samice toho istého druhu!

Iný príklad. Nech $A_{u,v}$ a $T_{u,v}$ je počet autobusových, resp. trolejbusových liniek z u do v (teda $A_{u,v}, T_{u,v} \in \mathrm{N}$). Ako zistíme, koľkými spôsobmi sa dá dostať z i do j, ak pôjdeme najskôr autobusom a potom trolejbusom? Ak je k medzizastávka, potom $A_{i,k}$ spôsobmi sa dostaneme do k a $T_{k,j}$ spôsobmi do j. Spolu $A_{i,k} \cdot T_{k,j}$, pričom k môže byť ľubovoľné (1 až n). Teda

$$D_{i,j} = (A_{i,1} \cdot T_{1,j}) + (A_{i,2} \cdot T_{2,j}) + (A_{i,3} \cdot T_{3,j}) + \cdots + (A_{i,n} \cdot T_{n,j}).$$

Nepripomína vám to niečo?

Posledný príklad. Nech $A_{u,v}$ a $T_{u,v}$ je čas, ktorý to trvá autobusu, resp. trolejbusu z u do v (teda $A_{u,v}, T_{u,v} \in \mathrm{R}_0^+ \cup \{\infty\}$ – čas je nezáporné reálne číslo alebo nekonečno, ak medzi danými zastávkami spoj nejde). Ako najrýchlejšie sa vieme dostať z i do j, ak chceme ísť najskôr autobusom a potom trolejbusom (zanedbajme čakanie na spoj)? Opäť všetky trasy vyzerajú tak, že ideme z i do nejakého k a potom do j, teda trasa bude trvať $A_{i,k} + T_{k,j}$. Z týchto všetkých časov treba nájsť minimum, teda

$$D_{i,j} = \min\{A_{i,1} + T_{1,j},\ A_{i,2} + T_{2,j},\ A_{i,3} + T_{3,j},\ \ldots,\ A_{i,n} + T_{n,j}\}.$$

... alebo, napíšem to ešte sugestívnejšie: Nech $a \downarrow b = \min(a, b)$, teda $\downarrow$ je taký infixový zápis binárneho minima (wtf?), potom:

$$D_{i,j} = (A_{i,1} + T_{1,j}) \downarrow (A_{i,2} + T_{2,j}) \downarrow (A_{i,3} + T_{3,j}) \downarrow \cdots \downarrow (A_{i,n} + T_{n,j}).$$

Nepripomína vám to niečo? Tentokrát si dokonca aj odpoviem: Musí! V každom príklade bolo vlastne to isté – násobenie matíc. Iba v prvom prípade boli prvkami matice pravdivostné hodnoty, namiesto $\cdot$ bol $\wedge$ a namiesto $+$ bol $\vee$. Druhý prípad bolo násobenie matíc s prvkami z N a tretí prípad bolo opäť v zásade to isté, iba prvky boli z $\mathrm{R}_0^+ \cup \{\infty\}$, namiesto $\cdot$ bolo $+$ a namiesto $+$ bolo $\downarrow$. Taký algebraik[8] by si s tým vôbec nelámal hlavu – v každom prípade máme nejaké prvky, ktoré svojským spôsobom „sčítavame“ a „násobíme“. Všade však „násobíme“ dvojice (i, k) a (k, j) a tieto (svojské) „súčiny“ (svojsky) „sčitujeme“.

To by bolo k násobeniu matíc a rôznym úlohám, ktoré sa na ňu dajú previesť. Na maticiach môžeme definovať aj umocnenie matice na k-tu: $A^1 = A$, $A^2 = A \cdot A$, $A^3 = A \cdot A \cdot A$, všeobecne

$$A^k = A \cdot A^{k-1} = \underbrace{A \cdot A \cdot A \cdots A}_{k}.$$

Napríklad v prvom prípade nám matica $A^2 = A \cdot A$ hovorí, z ktorého vrcholu do ktorého sa dá dostať dvoma cestami autobusom. V druhom prípade je to počet rôznych možností, ako ísť dvoma autobusmi. V treťom je to najkratší čas, ako sa dostať z jednej zastávky na druhú, ak pôjdeme práve dvoma autobusmi. Podobne A^3 je to isté, ale autobusom pôjdeme trikrát. Všeobecne A^k nám povie odpovede na naše úlohy, akurát nepôjdeme autobusom a potom trolejbusom, ale k-krát autobusom.

Mimochodom $A^0 = I$ je tzv. jednotková matica. Jednotková matica má všade nuly, iba na hlavnej diagonále má jednotky (zľava hore doprava dole). Hovorí sa jej jednotková, pretože sa správa podobne ako číslo 1. Jednotka je také číslo, že $1 \cdot x = x \cdot 1 = x$. To isté platí pre jednotkovú maticu – $I \cdot X = X \cdot I = X$.

V jednotlivých príkladoch, kde sme mali svojské $\cdot$ a $+$ máme aj svojských náhradníkov za 0 a 1 (pre nulu platí $x + 0 = 0 + x = x$ a $0 \cdot x = x \cdot 0 = 0$). V prvom prípade hrá úlohu 0 `false` a 1 `true`. V treťom prípade nám funkciu nuly spĺňa ∞ (naozaj $x \downarrow \infty = x$ a $x + \infty = \infty$) a úlohu jednotky hrá nula (naozaj $0 + x = x$). Preto v našom prípade vyzerá

[8] algebraici sú jeden veľmi živočíšny druh matikov

jednotková matica tak živočíšne ako vyzerá (pozri funkciu `id`). Ale to len tak na okraj a na popletenie.

Po tomto siahodlhom výklade by ste teda mohli vedieť riešiť úlohy, kde sa v správny moment v správnom kontexte použijú slová ako „graf“, „zistite pre každý vrchol“ a „cesta dĺžky práve k“.[9] Jednoduché riešenie úlohy 7. O VMVČ by teda mohlo vyzerať nejak takto: Žiadna cesta nemôže pozostávať z viac ako $k + n$ letov (potom by existoval cyklus dĺžky najviac n a jeho odstránením by sme získali čosi kratšie). Vypočítame teda postupne A^k, A^{k+1}, A^{k+2}, ..., A^{k+n} (pričom namiesto $\cdot$ bude + a namiesto + bude $\downarrow$). Matica A^k bude tabuľka najlacnejších ciest pozostávajúcich z k letov, A^{k+1} z $k + 1$ letov atď. Teraz už iba pre každú dvojicu miest z týchto tabuliek vyberieme minimum.

Násobiť matice vieme v čase $O(n^3)$ (v zdrojáku je to funkcia `mul`; teoretické výsledky hovoria, že zložitejšie sa to dá aj o čosi rýchlejšie). Takto sme teda dostali algoritmus bežiaci v čase $O((n + k)n^3)$. Toto ste dostali viacerí, či už nepoužijúc a či použijúc nevedomky násobenie matíc.

Dá sa to zlepšiť. Už vieme riešiť „nájdi medzi každými dvoma vrcholmi najkratšiu cestu dĺžky k“. To, čo ste určite všetci vedeli aj predtým a zvádzalo vás to ku všakovakým riešeniam je „nájdi medzi každými dvoma vrcholmi najkratšiu cestu (s ľubovoľným počtom hrán, to znamená 0 a viac)“. Akiste tiež viete, že sa táto úloha dá riešiť v čase $O(n^3)$ pomocou Floydovho-Warshallovho algoritmu (kto nie, nájsť a naštudovať).

Nech teda $K = A^k$ je tabuľka najkratších ciest na práve k letov a M je tabuľka najkratších ciest vôbec (z i do i má najkratšia cesta dĺžku 0). Na vyriešenie VMVČákov sa dajú riešenia spomínaných úloh použiť. Každá cesta, ktorá ma aspoň k hrán (letov) sa skladá z k-hranovej cesty a zvyšku (0 a viac hrán). Ba čo viac, najkratšia cesta dĺžky aspoň k sa skladá z najkratšej k-hranovej cesty a najkratšej cesty, ako prejsť zvyšok (iba sme vhodne doplnili slovo „najkratší“). Teda keď chceme ísť z i do j (na aspoň k letov), existuje nejaké mesto ℓ, kde budeme po k letoch; z i do ℓ sa najlacnejšie dostaneme za $K_{i,\ell}$ bubákov a z ℓ do j na $M_{\ell,j}$. Zo všetkých ℓ treba vybrať to najlepšie. Dúfam, že nemusím znovu vypisovať ten vzorec – výsledok dostaneme jednoducho tak, že „vynásobíme“ matice K a M.

K vypočítame v čase $O(kn^3)$ a, ako už odznelo, M vypočítame v $O(n^3)$, spolu $O(kn^3)$. Lepšie. Toto sa snáď dalo aj bez matíc, prečo som tu teda toľko vyvetľoval tie matice? Ukázali sme si, ako s nimi vyriešime viacero príkladov. Dokonca aj najkratšie cesty na $\leq k$ hrán vieme vyriešiť tak, že povolíme prechody z v do v a umocníme na k: $(I + A)^k$ a najkratšie cesty ľubovoľnej dĺžky vieme vypočítať tak, že $(I + A)$ umocníme na n-tú – i keď to bude pomalšie a trochu zložitejšie, dať sa to dá. Bolo to teda preto, že matice sú almighty.

Ten druhý dôvod bol ten, že ich asi naozaj potrebujeme ku vzorovému riešeniu. Maticu K vieme vypočítať aj rýchlejšie ako pomocou k násobení. Napríklad x^{19} nebudeme počítať ako $x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x$. Stačí vypočítať $x^2 = x\cdot x$, $x^4 = x^2\cdot x^2$, $x^8 = x^4\cdot x^4$, $x^{16} = x^8\cdot x^8$ a keďže $19 = 16+2+1$, $x^{19} = x^{16+2+1} = x^{16}\cdot x^2\cdot x$. Vo všeobecnosti takto vieme vypočítať mocninu v logaritmickom čase – postupne umocňujeme na druhú a dostávame $x, x^{2^1}, x^{2^2}, x^{2^3}, \ldots$ a potom tie „správne“ mocniny vynásobíme. Ktoré sú tie správne? To nám prezradí dvojkový zápis čísla – napr. $19 = (10011)_2 = 2^4 + 2^1 + 2^0 = 16 + 2 + 1$. Rovnako to funguje pri maticiach. Budeme teda postupne umocňovať na druhú a zároveň k-čko deliť dvojkou – ak je posledný bit 1 (nepárne číslo), mocninu k výsledku prinásobíme, inak nie (pozrite si funkciu `pow`). Výsledok: elegantné riešenie v čase $O(n^3 \log k)$.

Listing programu:

```
#include <cstdio>
#include <cstring>
#define INF 9999
```

[9] všade kde sa tu rozpráva o cestách máme v skutočnosti na mysli tzv. sledy – po hranách a do vrcholov sa dá ísť aj viackrát

```
#define MAX 100
#define REP(i,n) for (int i=0; i<(n); ++i)
typedef long matrix [MAX][MAX];

int n, k;
matrix L, // ceny (priamych) letov,
       K, // najkratsie cesty dlzky K
       M, // najkratsie cesty
       R; // vysledna tabulka

void id (matrix A) { // A <- I_n
  REP(i,n) REP(j,n) A[i][j] = INF;
  REP(i,n) A[i][i] = 0;
}

void copy (matrix B, matrix A) { // B <- A
  REP(i,n) REP(j,n) B[i][j] = A[i][j];
}

// nasobenie matic
void mul (matrix C, matrix B, matrix A) { // C <- A * B
  matrix TMP;
  REP(i,n) REP(j,n) {
    TMP[i][j] = INF;
    REP(k,n) if (TMP[i][j] > A[i][k]+B[k][j]) TMP[i][j] = A[i][k]+B[k][j];
  }
  copy (C, TMP);
}

// umocnenie matice
void pow (matrix B, matrix A, int k) { // B <- A^k
  matrix P;
  id (B); copy (P, A);
  while (k) {
    if (k % 2) mul (B, B, P); // B <- B * P
    mul (P, P, P); // P <- P * P
    k /= 2;
  }
}

// floydov-warshallov algoritmus
void fw (matrix A) {
  REP(k,n) REP(i,n) REP(j,n)
    if (A[i][j] > A[i][k]+A[k][j]) A[i][j] = A[i][k]+A[k][j];
}

void print (matrix A) {
  REP(i,n) {
    REP(j,n) if (i == j) printf (" - ");
                    else printf ("%3d ", A[i][j]);
    printf ("\n");
  }
}

int main() {
  // nacitaj
  scanf ("%d %d\n", &n, &k);
  REP(i,n) REP(j,n)
    if (i == j) L[i][j] = INF;
           else scanf ("%d ", &L[i][j]);
```

```
  // najkratsie cesty dlzky k
  pow (K, L, k);
  // najkratsie cesty
  copy (M, L); REP(i,n) M[i][i] = 0;
  fw (M);
  // skombinujeme
  mul (R, K, M);
  print (R);
  return 0;
}
```

8. Otravná aritmetická postupnosť

opravoval Mic
(max. 15 bodov)

Tento príklad je veľmi pekný. Vzorové riešenie je veľmi krátke, dá sa rýchlo nakódiť a dokonca aj jednoducho dokázať. Kde je teda pes zakopaný? Ťažké to bolo vymyslieť, pretože riešenie nie je intuitívne[10]. Ale najskôr vám poviem, ako som hodnotil.

- 15 bodov som udeľoval riešeniam bežiacim v čase $O(N \log N)$.
- 13 bodov dostalo kubické riešenie.
- Za riešenia skúšajúce všetky možnosti som dával maximálne 8 bodov.
- Za nefunkčné riešenia som dával maximálne 5 bodov.
- Za nedostatočný popis, chýbajúci dôkaz, chybu v kóde som strhával maximálne 2 body za každý prehrešok.

Naozaj to chcete počuť? Áno? Tak poďme na to. Poďme si všetko rátať v dvojkovej sústave, lebo tam sa nám to robí veľmi dobre a jednoducho. Čím je význačná každá aritmetická postupnosť? Predsa má svoju diferenciu. Tak si nejakú zoberme, napríklad d a zapíšme si d v dvojkovej sústave. Ako vyzerá? Nech $k > 0$ je také číslo, že na $k-1$ najnižších bitoch v d sú nuly a na na k-tom bite je jednotka. Zjavne platí, že prvky ľubovoľnej aritmetickej postupnosti s diferenciou d majú prvých $k-1$ bitov rovnakých, tie nás preto nebudú zaujímať.

Teraz si zoberme nejaké číslo z aritmetickej postupnosti. Nech má na k-tom bite nulu. Potom ďalšie číslo z danej postupnosti musí mať na k-tom bite jednotku, a ďalšie tam opäť musí mať nulu. Toto vyplýva zo spôsobu, akým sa v dvojkovej sústave sčítavajú čísla. V prípade, že má prvé číslo na k-tom bite jednotku, tak druhé tam má nulu, a tretie jednotku. Všimnime si, že na k-tom bite sa striedajú jednotky a nuly[11]. Ale veď my nehľadáme aritmetické postupnosti! My tam žiadne nechceme. Takže ak zabránime, aby sa na tom k-tom bite mohli striedať jednotky a nuly, potom tam taká postupnosť ani nemôže vzniknúť. Môžeme to napríklad urobiť tak, že tie čísla, ktoré majú na k tom bite nulu pôjdu vždy pred číslami ktoré majú na k-tom bite jednotku.

Tak teda poďme triediť. Keďže prvých $k-1$ bitov musí byť rovnakých, tak začnime od $k = 1$, pre ktoré to zjavne platí. Takže utriedime číselká podľa najmenej významného bitu. Čísla sa nám rozdelia na dve polovice. Na tie, čo majú na najmenej významnom bite jednotku a tie, čo majú na najmenej významnom bite nulu. Zjavne nemôže existovať aritmetická podpostupnosť (dĺžky aspoň 3), ktorá má nejaké prvky z prvej kopy a ďalšie z druhej. To znamená, že teraz stačí už len usporiadať každú polovicu zvlášť. Tie tentokrát s $k = 2$, čím sa nám to zase rozdelí a budeme mať dokopy 4 časti, ktoré treba zotriediť. Keď budeme takto

[10] stretol som sa dokonca aj s názorom, že je to kontraintuitívne

[11] ale na vyšších bitoch to nemusí a ani nebude platiť

pokračovať, tak zjavne dôjdeme k takému usporiadaniu, že tam nebude existovať aritmetická podpostupnosť[12].

Teraz sa ale poriadne na 5 minút zamyslite, čo vlaste dané usporiadanie spraví. Hotovo? Určite ste si všimli, že my vlastne triedime čísla. Najkôr z každého čísla spravíme binárny reverz, tieto čísla utriedime, a potom z nich spravíme binárny reverz znova. Čo je to binárny reverz? Dá sa to jesť? Odpoveď je jednoduchá. Je to číslo, ktoré vznikne zapísaním pôvodného čísla v binárnej sústave a prečítaním odzadu[13]. Dôkaz toho, že toto tvrdenie platí, si spravte na domácu úlohu. Ja len dodám, že dôkaz nebude príliš iný[14] s tým, čo ste už čítali v tomto vzoráku.

Ešte by som rád poznamenal, že uvedený postup nefunguje, ak na vstupe dostaneme 3 a viac rovnakých prvkov. To je jediný prípad, kedy sa postupnost nedá utriediť tak, aby tam nebola aritmetická podpostupnosť.

Ešte by sme si mohli niečo povedať o časovej zložitosti. Časová zložitosť použitého triedenia je $O(N \log N)$. Časová zložitosť urobenia reverzu čísla je konštantná. Preto časová zložitosť riešenia je $O(N \log N + N) = O(N \log N)$.

Listing programu:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
#define REP(i,n) for (int i=0; i<(n); ++i)

unsigned rev(unsigned a){
    unsigned b=0;
    REP(i,8) {
      b = (b << 1) | (a&1);
      a >>= 1;
    }
    return b;
}

int main(){
    int N;
    cin >> N;
    vector<unsigned> A(N);
    REP(i,N) cin >> A[i];
    REP(i,N) A[i] = rev(A[i]);
    sort(A.begin(),A.end());
    REP(i,N) A[i] = rev(A[i]);
    REP(i,N-2) {
        if(A[i]==A[i+1] && A[i]==A[i+2]){
            cout << "Neda sa :-(" << endl; return 0;
        }
    }
    REP(i,N) cout << A[i] << endl;
    return 0;
}
```

[12] za predpokladu, že tam neexistujú tri rovnaké čísla

[13] všimnite si, že keď toto aplikujeme na číslo a a potom ešte raz na výsledok, dostaneme číslo a

[14] ak nie totožný:-)

9. Trpasličie hry

opravoval Lukáš
(max. 15 bodov)

Podľa počtu riešení to vyzerá tak, že ste sa tohto príkladu všetci zľakli. Nakoniec to ale nebolo až také ťažké. Vyskytli sa dva typy riešení. Prvé boli pomalé a hľadali všetky cesty. Tie potom zotriedili a vypísali K-tu najmenšiu cestu. Ciest však môže byť veľmi veľa – počet ciest pod uhlopriečkou zodpovedá Katalánovym číslam[15]. N-té Katalánove číslo je približne $4^N/N^{3/2}\sqrt{\pi}$, čo je veľmi veľa. Tieto riešenia mohli dostať 6 bodov. Ďalšie riešenia už pracovali s časovou aj pamäťovou zložitosťou $O(N^2)$ a boli odmenené 15 bodmi. A práve takéto riešenie si popíšeme.

Veľmi ľahko si všimneme, že každá cesta z $[1,1]$ do $[N,N]$ má dĺžku $2N-1$, dokonca je to napísané aj v zadaní. Naše riešenie bude fungovať tak, že postupne budeme určovať $2N-1$ písmen výsledného slova zľava doprava. Ešte si dovolím upozorniť, že ďalej budem niekedy zamieňať slovo a cestu. Totiž každá cesta nám určuje slovo. Naopak to už ale neplatí, preto budem opatrný. Prvé písmeno je jasné, lebo musíme začať v ľavom hornom rohu. Odtiaľ máme dve možnosti, ako pokračovať. Nech na pozícii $[1,2]$ je písmeno a a na pozícii $[2,1]$ písmeno b. Ak platí $a=b$, potom je druhé písmeno jasné a pokračujeme ďalej dvomi vetvami. Inak môžeme bez ujmy na všeobecnosti predpokladať, že $a<b$. Počet ciest z obidvoch políčok na políčko $[N,N]$ je rovnaký. Označme ho p. Potom ak by bolo prvé písmeno b, preskočili by sme p slov (lebo všetky slová začínajúce sa na b sú za slovami začínajúcimi sa na a). Na vstupe je zadané číslo K, ktoré udáva poradové číslo slova, ktoré chceme nájsť. Preto stačí porovnať čísla K a p a z toho už vieme, či druhé písmeno výsledného slova je a alebo b.

Podobne budeme postupovať aj ďalej. Predstavme si, že sme už úspešne určili prvých i písmen výsledného slova – $w_1w_2\ldots w_i=w$. Môže sa nám stať (podobne ako sa nám stalo pri druhom písmene, že $a=b$), že budeme naraz na viacerých písmenách v mriežke. Inými slovami, ciest popísaných slovom w môže byť viac a môžu sa končiť na rôznych pozíciách v mriežke. Označme tieto pozície $[x_1,y_1],\ldots,[x_m,y_m]$. Navyše, do nejakej pozície $[x_j,y_j]$ môže viesť viacero ciest popísaných slovom w. Z pozície $[x_j,y_j]$ sa môžeme pohnúť na najviac dve políčka – dole a doprava. Nemôžeme však vyskočiť z mriežky ani prekročiť uhlopriečku. Dostaneme tak najviac $2m$ kandidátov na ďalšie písmeno. Pre každé takéto písmeno z si spočítame, koľko je ciest, ktoré sa začínajú slovom $w_1w_2\ldots w_iz$ a končia v pravom dolnom rohu. K tomuto počítaniu sa ešte vrátime. Teraz mi na chvíľu uverte, že to vieme ľahko a rýchlo zrátať. Keď teda poznáme všetky tieto počty ciest, vieme učiť $(i+1)$-vé písmeno slova.

Z predchádzajúcich výpočtov vieme, koľko ciest sme už preskočili tým, že sme zvolili cestu $w_1w_2\ldots w_i$. Nech kandidáti na nové písmeno sú podľa abecedy $a_1,a_2,\ldots a_l$. Keby sme zvolili písmeno a_1, nepreskočili by sme žiadne ďalšie cesty. Keby sme zvolili písmeno a_2, preskočili by sme navyše všetky cesty začínajúce sa na $w_1w_2\ldots w_ia_1$. Všeobecne ak zvolíme písmeno a_j, preskočíme aj cesty $w_1w_2\ldots w_ia_o(\forall o, o<j)$. Ako sme si povedali pred chvíľou, všetky tieto počty ciest už poznáme. Vieme teda povedať, ktoré z písmen $a_1,a_2,\ldots a_l$ si vyberieme tak, aby sme nepreskočili veľa alebo málo ciest. Povedané formálnejšie – s vybraným písmenom a_q sme preskočili menej ako K ciest a zároveň ak by sme zobrali a_{q+1}, dokopy by sme už preskočili aspoň K ciest.

Teraz sa vráťme späť k tomu, ako vyrátať počet ciest začínajúcich sa slovom $w_1w_2\ldots w_iz$, prechádzajúcich cez políčko $[x,y]$ a končiacich sa v $[N,N]$. Zistíme najprv počet ciest začínajúcich sa na $w_1w_2\ldots w_iz$ a prechádzajúcich cez políčko $[x,y]$ a toto číslo vynásobime počtom ciest z $[x,y]$ do $[N,N]$. Pri hľadaní $(i+1)$-vého písmena už poznáme počet ciest $w_1\ldots w_i$ pre jednotlivé políčka. Potom stačí pripočítať danú hodnotu na ďalšie políčko, kam sa pohneme.

Označme $p(x,y)$ počet ciest z $[x,y]$ do $[N,N]$ takých, že neprekračujú uhlopriečku. V našom programe ich budeme rátať tak, že najprv vyrátame hodnoty pod uhlopriečkou a potom priradíme $p(x,y)$ do $p(y,x)$, lebo hodnoty sú zjavne symetrické. Triviálne platí $p(N,N)=1$.

[15] `http://en.wikipedia.org/wiki/Catalan_numbers`

Hodnotu $p(x, y)$ vieme vyrátať z $p(x, y+1)$ a $p(x+1, y)$ tak, že ich sčítame. Ak by sme však mali prekročiť uhlopriečku alebo vyjsť z mriežky, jednu z dvoch hodnôt nepripočítame. V programe sa o počítanie tejto hodnoty stará funkcia `rataj`.

Takže algoritmus máme hotový a už sa len ubezpečíme, že pracuje v čase $O(N^2)$. Každé políčko spracúvame ako kandidáta na ďalšie písmeno najviac raz. V ňom sa pozrieme na najviac dvoch susedov, čiže táto fáza má kvadratickú zložitosť. Keď vyberáme i-te písmeno slova máme naraz najviac $N+1$ kandidátov na pokračovanie (písmeno na uhlopriečke môže byť medzi kandidátmi aj dvakrát, lebo sme naňho mohli prísť zhora aj zdola). V tejto fáze ešte potrebujeme pre každé písmeno zrátať počet ciest a vybrať i-te písmeno. Počet ciest vieme zrátať v čase lineárnom od počtu kandidátov. Výber písmena potom závisí len od veľkosti abecedy, čo je v našom prípade 26, teda konštanta. Kandidátov hľadáme $(2N-1)$-krát, preto je zložitosť tiež kvadratická. No a rátanie $p(x, y)$ vieme ľahko spraviť tiež v kvadratickom čase.

V tomto príklade je roboty ako na kostole. Ak vás to neodrádza, pozrite si aj vzorový program.

Listing programu:

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

typedef long long ll;
ll kol[32][32];
struct Tri {
    //objekt, v ktorom si pamätáme súradnice políčka
    //a či je nad uhlopriečkou alebo pod uhlopriečkou
    int x, y;
    bool dole;
    Tri(int xx, int yy, bool d) :
        x(xx), y(yy), dole(d) {};
};
int t, n;
long long rataj(int i, int j) {
    //zrátame počet ciest z [i, j] do [n-1, n-1]
    if (kol[i][j]!=-1) return kol[i][j];
    if (i<j) return kol[i][j]=rataj(j, i);

    if (i==n-1) return kol[i][j]=1;

    kol[i][j]=0;
    if (i+1<n) kol[i][j]+=rataj(i+1, j);
    if (j+1<=i) kol[i][j]+=rataj(i, j+1);
    return kol[i][j];
}
char c[32][32];
ll poc[32][32][2];
bool mam[32][32][2];
string vysl;
void prehl(vector<Tri> a, ll m) {
    if ((int)vysl.size()==2*n-1) {//ak už máme dosť dlhé slovo, skončíme
        cout<<vysl<<endl;
        return;
    }
    vector<Tri> nove;
    //v poli nove budeme mať kandidátov na ďalšie písmeno
    vector<ll> q(26, 0);
```

```
    for (unsigned i=0; i<a.size(); i++)
        for (int j=1; j<=2; j++) {
            //pohneme sa dole alebo doprava
            int s=a[i].x+j/2, t=a[i].y+j%2;
            if (s>=n || t>=n ||
                    (s<t && a[i].dole) || (s>t && !a[i].dole)) continue;

            poc[s][t][a[i].dole]+=poc[a[i].x][a[i].y][a[i].dole];
            q[ c[s][t]-'A' ]+=poc[a[i].x][a[i].y][a[i].dole]*rataj(s, t);
            if (!mam[s][t][a[i].dole]) {
                nove.push_back(Tri(s, t, a[i].dole));
                mam[s][t][a[i].dole]=true;
            }
        }

    char pismeno='A';
    for (int i=0; i<26; i++) if (q[i]) {
        //ideme "preskakovať" cesty
        if (m<q[i]) {
            //vybratím tohto písmena by sme už preskočili veľa,
            //takže ho vyberieme
            pismeno=i+'A';
            break;
        }
        else m-=q[i];
    }

    vector<Tri> res;
    for (unsigned i=0; i<nove.size(); i++) {
        if (c[nove[i].x][nove[i].y]==pismeno)
            res.push_back(nove[i]);
    }
    vysl+=pismeno;
    prehl(res, m);
}
int main() {
    ll m;
    cin>>n>>m;
    memset(kol, 0xff, sizeof(kol));
    memset(poc, 0x00, sizeof(poc));
    memset(mam, 0x00, sizeof(mam));
    m%=2*rataj(0, 0);

    for (int i=0; i<n; i++) cin>>c[i];
    vector<Tri> a(1, Tri(0, 0, 0));//z prvého políčka ideme hore...
    a.push_back(Tri(0, 0, 1));//a dole

    vysl=c[0][0]; poc[0][0][0]=poc[0][0][1]=1;
    prehl(a, m);
}
```

10. TurboTrúba

opravoval Ppershing
(max. 15 bodov)

Vzorák by som začal asi klasickým porekadlom „Nie je všetko Huffman čo sa blyští“. Ako bolo treba postupovať pri riešení tohto príkladu? Pozriem sa a vidím, že ľahšia úloha zo zadania

je obyčajné Huffmanovo kódovanie.[16] Tak čo, skúsim ho vylepšiť. Potiaľto sa dostala väčšina z malej hŕbky riešiteľov. Ale riešenie príkladu by malo pokračovať otázkou: „Prečo je to sakra 10. úloha, tyč, a ja som ju dal tak rýchlo? Tu niečo nehrá!“ Na túto otázku si pekne krásne odpovedal opravovateľ asi po trojhodinovej snahe dokázať funkčnosť vašich riešení, bezúspešne. Prišla druhá fáza – google.

Tam zistíme, že „Huffman coding with unequal letter costs“ nemá známe polynomiálne riešenie. To je už dosť dobrý fakt na to, aby sme od optimizmu ustúpili. Takže, zrejme je to problém, ktorý je veľmi špecifický. Treba naň vymyslieť niečo, čo nie je bežné (preto 10. príklad). Poďme sa teda pustiť do rozmýšľania – predpokladajme, že by nám niekto podvrhol správny prefixový strom. V tom prípade by sme to vedeli spraviť ľahko: listy ohodnotíme pravdepodobnosťami tak, aby písmenku s menšou pravdepodobnosťou prislúchala väčšia vzdialenosť od koreňa, a teda dlhšie kódové slovo.

Našou starosťou je teda nájsť najlepší taký strom. Aby sme sa nehrali s cenami jednotlivých hrán, jednoducho sa dohodneme, že ľavý potomok je vždy o 1 a pravý o 2 úrovne nižšie (tým vyriešime problém nerovnakej dĺžky prenosu „tú“ a „húúú“).

Stromy budeme tvoriť od koreňa smerom nadol, pričom použijeme dynamické programovanie. Inými slovami, nebudeme skúšať úplne všetky stromy, ale skúsime niektoré prehlásiť za dostatočne podobné.

Čo bude stav nášho výpočtu? Budeme si pamätať hĺbku aktuálneho stromu (pričom hĺbka koreňa je 1). Ďalej potrebujeme: koľko listov je na spodnej úrovni, koľko ich je o úroveň vyššie, a aký je celkový počet listov. (Z toho vieme povedať, koľko listov je na všetkých vyšších úrovniach dokopy.)

Listy na spodných dvoch úrovniach budeme ešte môcť „rozvíjať“ ďalej, tie vyššie už prehlásime za definitívne. Každému listu priradíme jedno písmeno, pekne podľa pravdepodobnosti ich výskytu. (Niektoré písmená ostanú ešte nepriradené, a niektoré písmená sa ešte budú posúvať hlbšie, keď „rozvinieme“ list, kde teraz sú.)

Cena konkrétneho „nehotového stromu“ bude teda súčet hodnôt (pravdepodobnosť písmena) krát (hĺbka v ktorej je), pričom sčítame cez všetky listy v ňom.

Uvedomte si, že danému stavu výpočtu môže zodpovedať veľa rôznych nehotových stromov. Z nášho hľadiska sú všetky ekvivalentné – ak by sme k jednému nejako doplnili zvyšok stromu, presne rovnaký zvyšok vieme doplniť ku každému inému z nich. Preto nás pre každý stav výpočtu bude spomedzi všetkých možných stromov zaujímať len ten dovtedy najlepší.

Inými slovami, pre každý stav výpočtu budeme chcieť spočítať, akú najlepšiu cenu môže mať nehotový strom, ktorý tomuto stavu zodpovedá.

Začínať budeme z triviálneho stromčeku pre 2 písmenka, teda zo stavu [3][1][1][2]. Do akých stavov sa môžeme dostať? Dohodnime sa, že budeme robiť iba takúto operáciu – zober k listov v hĺbke $hĺbka-2$ a zmeň ich na k triviálnych stromčekov (1 list vľavo, 1 vpravo v hĺbke 2). Tvrdím, že opakovaním tohoto postupu vieme určite vyrobiť optimálny strom. Dôkaz – ukážte si, že s optimálnym stromom vždy vieme spraviť reverznú operáciu, t.j. z najhlbšej úrovne odstrániť všetky listy aj s ich prarodičmi, teda zmazať celé triviálne stromčeky.

Zmena stavu nám zmení cenu stromu. Ako? Z úrovne $hĺbka-2$ presunieme k najmenej častých písmen o úroveň nižšie, a na úroveň $hĺbka$ pridáme k nových písmen. Zo stavu $[x][a][b][c]$ sme prešli do stavu $[x+1][b+k][k][c+k]$.

Optimálnu dosiahnuteľnú cenu stromu získame ako minimum z hodnôt [*][*][*][počet písmenok]. Ak si navyše budeme pre každý stav pamätať predchádzajúci, vieme spätne z minima vybudovať strom, ohodnotiť ho a vypísať. Máme teda algoritmus. Aká je jeho zložitosť? Čas $O(n^4$ stavov $\cdot$ veľa roboty v každom stave$) = O(n^6)$, pamäť $O(n^4)$.

Ale správnemu KSPákovi takáto zložitosť nedá spať. Preto si ukážeme, ako spraviť riešenie v čase $O(n^4)$ a pamäti $O(n^3)$. Skúsime sa zbavíme niečoho „nepotrebného“ – hĺbky.

[16] Ak nevieš, čo to je, pozri napr. `http://en.wikipedia.org/wiki/Huffman_coding`

Jediná vec, kde ju používame je výpočet zmeny ceny pri prechode do nového stavu, presnejšie pri pridávaní nových písmen.

Nedá sa to ale obísť? Isteže dá, stačí si vhodne zmeniť definíciu ceny stromu: Nech cena nehotového stromu je $\sum$ pravdepodobnosť použitého písmenka · jeho hĺbka+ $\sum$ pravdepodobnosť nepoužitého písmenka · aktuálna hĺbka stromu.

Potom naša operácia prechodu medzi stavami spraví toto: zväčší o 1 cenu každého písmena, ktoré posúvame o úroveň hlbšie, a zväčší o 1 cenu každého písmena, ktoré sme ani v tomto kroku ešte nepriradili.

Ako ale vieme, nepoužité písmenká sú tie s najmenšími pravdepodobnosťami. Ak si teda predrátame súčet prvých k najmenších pravdepodobností, vieme druhý člen súčtu zistiť v konštantnom čase. Takisto vieme zistiť pravdepodobnosť písmenok, ktoré „presúvame“ o poschodie nižšie. Teda prechod do ďalšieho stavu vieme spraviť v konštantnom čase, v každom stave prechádzame do $O(n)$ ďalších stavov a stavov je $O(n^3)$. Finálna zložitosť je $O(n^4)$ a pamäťová $O(n^3)$.

Domáca úloha – riadne si nakresliť a stráviť, čo tento algoritmus robí a domyslieť detaily (prípadne porovnať zo zdrojákom).

Bodovanie: Za vylepšeného Huffmana sa dalo získať 10 bodov, za zadrôtovanú tabuľku 7. Neuvedenie zdrojového kódu bolo odmenené -3 bodmi.

Listing programu:

```
#include <stdio.h>
#include <algorithm>
#include <list>
#include <queue>
#include <string>
using namespace std;

#define FOR(q,n) for(int q=0;q<n;q++)
#define MAX 50
#define INFINITY 1000000.0
typedef double ld;

ld pravdep[MAX];
ld pravdep_sum[MAX];
int n;
ld stav[MAX][MAX][MAX];
int back[MAX][MAX][MAX][3]; // informacie pre spatne rekonstruovanie stromu

int lavy[MAX*3]; // na strom
int pravy[MAX*3];
int strom_size=1; // indexujeme od 1
list<int> l1,l2,l3; // zoznamy vrcholov o level1,2 nizsie
string tabulka[MAX*3];
string vysledok[MAX];

void init(){
  scanf("%d",&n);
  FOR(q,n) scanf("%lf",&pravdep[q]); // nacitam pravdepodobnosti
  sort(pravdep,pravdep+n);           // a usortim ich

  FOR(q,n+2)
    FOR(w,n+2)
     FOR(e,n+2) {
        stav[q][w][e]=INFINITY;
        FOR(f,3) back[q][w][e][f]=-1;
     }
```

```
  if (n<2) { printf("mala abeceda\n"); exit(0); }

  FOR(q,n) pravdep_sum[q+1]=pravdep_sum[q]+pravdep[q];
  // sum[k]= suma pravdepodovnosti prvych k-1 pismenok
}

void solve(){
 // stav[a][b][c]=d oznacuje
 // ak mam strom, ktory ma na level-2 a listov
 // na level-1 b listov, ma dokopy c listov, tak minimalna
 // hodnota je d
 stav[1][1][2]=pravdep_sum[n]*2-pravdep[n-1];
  // akoby vsetky pismenka boli na leveli 2
 for (int pismenok=2;pismenok<=n;pismenok++) // budujeme podla poctu pismenok
   FOR(a,n+1)
     FOR(b,n+1) { // mame pismenok pismenok, a na level-2 b na level
       if (stav[a][b][pismenok]=INFINITY) continue; // nevieme sa tam dostat
       ld zmena=0; // kolko pridame, ked pridame vrchol
       ld zmena2=0; // + vsetky nepouzite pismenka hodime o level vyssie
       ld stav_hodnota=stav[a][b][pismenok];
       for (int rozsirime=1;rozsirime<=a;rozsirime++) {
         zmena+=pravdep[n-1-pismenok+b+rozsirime];
         zmena2=zmena+pravdep_sum[n-pismenok];
    if ( stav[b+rozsirime][rozsirime][pismenok+rozsirime]>stav_hodnota+zmena2){
         stav[b+rozsirime][rozsirime][pismenok+rozsirime]=stav_hodnota+zmena2;
         back[b+rozsirime][rozsirime][pismenok+rozsirime][0]=a;
         back[b+rozsirime][rozsirime][pismenok+rozsirime][1]=b;
         back[b+rozsirime][rozsirime][pismenok+rozsirime][2]=pismenok;
         }
       }
     }
}

void sprav_strom(int a,int b,int pismenok){
  if (a==1 && b==1 && pismenok==2) {
    lavy[1]=2;       // dame pociatocny stav
    pravy[1]=3;      //   1
    pravy[3]=4;      //  / \
    strom_size=5;    // 2   3
                     //      \
                     //       4
    l1.push_back(4);
    l2.push_back(2);
    return ;
  }

  int a1=back[a][b][pismenok][0];
  int b1=back[a][b][pismenok][1];
  int p1=back[a][b][pismenok][2];
  sprav_strom(a1,b1,p1); // ideme zo stavu a1,b1,p1
  // updatneme veci
  l3=l2; // odlozime level-2
  l2=l1; // odlozime level-1
  l1.clear(); // aktualna level-1 je prazdna
  // a ideme spracovat danu zmenu
  FOR(q,pismenok-p1) { // rozsirili sme o pismenok-p1
    int vrchol=*l3.begin(); // zobereme vrchol z povodnej level-2
    l3.erase(l3.begin());
    lavy[vrchol]=strom_size; // a nastavime mu potomkov
    l2.push_back(strom_size);
```

```
    strom_size++;
    pravy[vrchol]=strom_size;
    pravy[strom_size]=strom_size+1;
    strom_size+=2;
    l1.push_back(strom_size-1);
  }
}

void generate_tree(){
  int a=0;
  int b=0;
  FOR(q,n+1)
    FOR(w,n+1)
     if (stav[a][b][n]>stav[q][w][n]) a=q, b=w; // najdeme minimum

  printf("vygenerovana priemerna dlzka %lf \n",stav[a][b][n]);
  back[n][n][n][0]=a; // nastavime, aby sme nam funkcia sprav_strom
  back[n][n][n][1]=b; // automaticky prehladala cely strom
  back[n][n][n][2]=n; // aj s poslednym levelom !
  sprav_strom(n,n,n);
}

void generate_table(){
  queue<int> fronta;
  fronta.push(1);
  tabulka[1]=string("");

  int pismenko=n-1;
  while (!fronta.empty()) { // prehladavame strom do sirky
    int v=fronta.front(); fronta.pop();
    if (lavy[v]) {
       int i=lavy[v];
       tabulka[i]=tabulka[v]+".";
       fronta.push(i);
    }
    if (pravy[v]) {
       int i=pravy[v];
       if (tabulka[v][(int)tabulka[v].size()-1]=='+') { // + je pomocny medziznak
          tabulka[i]=tabulka[v];                        // na polovicu -
         tabulka[i][(int)tabulka[i].size()-1]='-';
       } else tabulka[i]=tabulka[v]+"+";
       fronta.push(i);
    }
    if (lavy[v]==pravy[v] && lavy[v]==0)
      vysledok[pismenko--]=tabulka[v];  // sme na konci
  }
  // vypis vygenerovanu tabulku
  FOR(q,n) printf("%.3lf %s\n",pravdep[q],vysledok[q].c_str());
}

int main(){
 init();
 solve();
 generate_tree();
 generate_table();
}
```

Výsledková listina po 1. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda	1	2	3	4	5	Σ
1	Ondruška Peter	SPŠ Dubnica nad Váhom	3	8	10	15	12	12	57
1	Varga András	Gym. H. Selyeho Komárno	3	13	10	15	12	7	57
3	Kostolányi Peter	Gym. Jura Hronca BA	3	15	10		15	9	49
4	Herencsár Albert	Gym. maď. Galanta	2	14	10		12	8	44
5	Hagara Michal	Gym. Jura Hronca BA	1	15	10		13		38
5	Simanová Lucia	Gym. Grösslingová BA	3	14	9	2	13		38
7	Končok Jakub	Gym. Haličská Lučenec	3	15	10		12		37
7	Matejovičová Lenka	Gym. Grösslingová BA	3	13	10	2	12		37
7	Súkeník Ján	Gym. Lettricha Martin	3	7	10		12	8	37
7	Štefanišin Peter	Gym. Svidník	4	12	8	5	12		37
11	Fecko Stanislav	Gym. Pankúchova Bratislava	4	13	10		12		35
12	Saleh Adam	Gym. Jura Hronca BA	3	13			12	9	34
13	Piter Miroslav	Gym. Svidník	0	12	7		12		31
14	Bene Daniel	Gym. Haličská Lučenec	3	8	10		12		30
14	Molčan Dávid	Gym. Slovenská Bardejov	4	7	7	10	6		30
16	Chrzan Martin	Gym. Ľ. Štúra Zvolen	2	5	9		10	4	28
16	Chudý Lukáš	Gym. Liptovský Hrádok	2	7	8		10	3	28
18	Bachratý Martin	Gym. Veľká Okružná Žilina	2	6	10		11		27
18	Kompan Martin	Gym. Partizánske	4	5	10		12		27
20	Boško Lukáš	Gymnázium Považská Bystrica	0	7	7		12		26
20	Kudláč Matúš	Gym. Bilíkova BA	3	11			10	5	26
20	Kušnier Juraj	Gym. Bánovce nad Bebravou	3	7	7		12		26
23	Fekiač Jozef	Gym. Grösslingová BA	2	5	8		12		25
23	Jenis Jakub	Gym. Cyrila a Metoda Nitra	3	7	6		12		25
25	Halabuk Milan	Gymnázium Levice	4	6	6		12		24
26	Hozza Michal	Gym. Jura Hronca BA	2	5	8		10		23
26	Knotek Tomáš	SPŠ Nitra	4	7		6	10		23
26	Polák Lukáš	Gym. Jura Hronca BA	1	5	10		6	2	23
29	Kučin Alexander	Gym. Lipany	3		10		12	0	22
30	Liška Igor	Gym. Jura Hronca BA	2	5			10	6	21
31	Matula Peter	Gym. Školská Turč. Teplice	3	5	8	1	3		17
32	Babej Tomáš	Gymnázium	0	6	10				16
32	Greň Peter	Gym. L. Stöckela Bardejov	1	4	8		4		16
32	Kvasnička Igor	Gym. Jura Hronca BA	3	13			3		16
32	Peťura Oto	SPŠE Michalovce	1	7	3		6		16
32	Šulák Viktor	SPŠE Nové Zámky	2	6			10		16
37	Hojčka Michal	Gym. Partizánske	2	7			7		14
38	Tureková Katarína	Gym. Tajovského B. Bystrica	3	13					13
39	Duchoň Gabriel	Gym. maď. Želiezovce	4	5			6		11
40	Balogh Tomáš	SPŠE Nové Zámky	2	4			6		10
40	Zelenaj Roman	Gym. Golianova Nitra	2	4	6				10
42	Kedrovič Ondrej	Gym. Jura Hronca BA	0	4			5		9
42	Migaš Jozef	Gym. Jura Hronca BA	0	9					9
44	Mamojko Štefan	Gym. Jura Hronca BA	0	8					8
45	Végh Ladislav	Gym. Tornaľa	2	4			3	0	7
46	Buchovecká Simona	Gym. Medzilaborce	4		1		4		5
46	Kohút Matej	Gym. Jura Hronca BA	2	5					5

Výsledková listina po 1. kole kategórie KSP-O

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
1	Okruhlica Adam	Gym. Jura Hronca BA	4	15	11	15	15	15	71
2	Danilák Michal	Gym. Hubeného BA	4	15	11	15	14	13	68
3	Herman Peter	Gym. Jura Hronca BA	4	15	11	15	12	8	61
4	Brezáni Samuel	Gym. Rajec	4	15	14	14	1	13	57
5	Ondruška Peter	SPŠ Dubnica nad Váhom	3	12	12	8	11	9	52
6	Hlavatý Peter	Gym. Jura Hronca BA	3	12	12	8	11	7	50
6	Rampášek Ladislav	Gym. Jura Hronca BA	4	15	11	15	1	8	50
8	Kekely Lukáš	Gym. Varšavská Žilina - Vlčince	3	12	8	8	10	9	47
9	Kočiský Tomáš	Gym. Grösslingová BA	3	15	11	15		4	45
10	Nagy Anton	Gym. maďarské BA	4	12	7	8	8	7	42
11	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	3	15	11	7	2	3	38
12	Boža Vladimír	Gym. Tatarku Poprad	3	13	8	5	11		37
12	Fedáková Dominika	Gym. Stará Ľubovňa	3	12	9	7		9	37
14	Korcsok Peter	Gym. Mládežnícka Šahy	3	12	6	8		10	36
15	Zámečník Dušan	Gym. Trebišovská Košice	4	12	6	8		9	35
16	Hapák Samuel	Gym. Grösslingová BA	3	13	6	15			34
17	Varga András	Gym. H. Selyeho Komárno	3	12	7	7	3	4	33
18	Kostolányi Peter	Gym. Jura Hronca BA	3	15	9	8			32
18	Mikuláš Ondrej	Gym. Haličská Lučenec	4	12	10	3		7	32
18	Nagy Kristián	Gym. Jura Hronca BA	3	13	11	8			32
18	Vojtko Jakub	Gym. Jura Hronca BA	2	12	12	8			32
22	Saleh Adam	Gym. Jura Hronca BA	3	12	9	7			28
23	Tomcsányi György	Gym. H. Selyeho Komárno	4	12		8	4		24
24	Liška Igor	Gym. Jura Hronca BA	2	10	6	7			23
25	Magyar Vladimír	SPŠE Nové Zámky	3	7	8	7			22
26	Chudý Lukáš	Gym. Liptovský Hrádok	2	10	3	8			21
27	Hanes Filip	Gym. Tajovského B. Bystrica	4	12		8			20
27	Herencsár Albert	Gym. maď. Galanta	2	12	8				20
27	Košinárová Alena	Gym. Grösslingová BA	3	12		8			20
27	Kušnier Juraj	Gym. Bánovce nad Bebravou	3	12		8			20
27	Petrucha Michal	Gym. Metodova BA	2	12		8			20
27	Súkeník Ján	Gym. Lettricha Martin	3	12	8				20
33	Mészároš Roman	SPŠ Levice	4	12		7			19
34	Kompan Martin	Gym. Partizánske	4	12		6			18
35	Knotek Tomáš	SPŠ Nitra	4	10		7			17
35	Sucha Martin	Gym. Jura Hronca BA	3	12		5			17
37	Kudláč Matúš	Gym. Bilíkova BA	3	10	5				15
38	Chrzan Martin	Gym. Ľ. Štúra Zvolen	2	10	4				14
39	Hagara Michal	Gym. Jura Hronca BA	1	13					13
39	Simanová Lucia	Gym. Grösslingová BA	3	13					13
41	Bene Daniel	Gym. Haličská Lučenec	3	12					12
41	Boško Lukáš	Gymnázium Považská Bystrica	0	12					12
41	Fecko Stanislav	Gym. Pankúchova Bratislava	4	12					12
41	Fekiač Jozef	Gym. Grösslingová BA	2	12					12
41	Halabuk Milan	Gymnázium Levice	4	12					12
41	Jenis Jakub	Gym. Cyrila a Metoda Nitra	3	12					12
41	Končok Jakub	Gym. Haličská Lučenec	3	12					12
41	Kučin Alexander	Gym. Lipany	3	12	0				12
41	Matejovičová Lenka	Gym. Grösslingová BA	3	12					12
41	Piter Miroslav	Gym. Svidník	0	12					12

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
41	Štefanišin Peter	Gym. Svidník	4	12					12
41	Vlček Tomáš	Gym. Šk. Bratov BA	4	12					12
53	Bachratý Martin	Gym. Veľká Okružná Žilina	2	11					11
53	Kvasnička Igor	Gym. Jura Hronca BA	3	3		8			11
55	Hozza Michal	Gym. Jura Hronca BA	2	10					10
55	Šulák Viktor	SPŠE Nové Zámky	2	10					10
57	Kedrovič Ondrej	Gym. Jura Hronca BA	0	5		4			9
58	Polák Lukáš	Gym. Jura Hronca BA	1	6	2				8
59	Hojčka Michal	Gym. Partizánske	2	7					7
60	Balogh Tomáš	SPŠE Nové Zámky	2	6					6
60	Duchoň Gabriel	Gym. maď. Želiezovce	4	6					6
60	Fico Tomáš	SPŠ Nitra	4	6					6
60	Molčan Dávid	Gym. Slovenská Bardejov	4	6					6
60	Peťura Oto	SPŠE Michalovce	1	6					6
65	Buchovecká Simona	Gym. Medzilaborce	4	4					4
65	Greň Peter	Gym. L. Stöckela Bardejov	1	4					4
67	Matula Peter	Gym. Školská Turč. Teplice	3	3					3
67	Végh Ladislav	Gym. Tornaľa	2	3	0				3

Výsledková listina po 1. kole kategórie KSP-T

	Meno a priezvisko	Škola	Trieda	8	9	10	Σ
1	Danilák Michal	Gym. Hubeného BA	4	13	15	10	38
2	Okruhlica Adam	Gym. Jura Hronca BA	4	15	12	10	37
3	Rampášek Ladislav	Gym. Jura Hronca BA	4	8		10	18
4	Korcsok Peter	Gym. Mládežnícka Šahy	3	10		7	17
5	Ondruška Peter	SPŠ Dubnica nad Váhom	3	9		7	16
6	Brezáni Samuel	Gym. Rajec	4	13			13
7	Fedáková Dominika	Gym. Stará Ľubovňa	3	9			9
7	Kekely Lukáš	Gym. Varšavská Žilina - Vlčince	3	9			9
7	Zámečník Dušan	Gym. Trebišovská Košice	4	9			9
10	Herman Peter	Gym. Jura Hronca BA	4	8			8
11	Hlavatý Peter	Gym. Jura Hronca BA	3	7			7
11	Kompan Martin	Gym. Partizánske	4			7	7
11	Mikuláš Ondrej	Gym. Haličská Lučenec	4	7			7
11	Nagy Anton	Gym. maďarské BA	4	7			7
15	Liška Igor	Gym. Jura Hronca BA	2		6		6
16	Kočiský Tomáš	Gym. Grösslingová BA	3	4			4
16	Varga András	Gym. H. Selyeho Komárno	3	4			4
18	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	3	3			3
18	Kedrovič Ondrej	Gym. Jura Hronca BA	0		3		3