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 letnej časti

Tak ako v zime nesnežilo, tak teraz nemáme bláznivé počasie napriek tomu, že je apríl. Aspoň je vonku pekne. A preto si tesne pred tým, než pôjdete niekam von (alebo večer na dobrú noc), môžete prečítať vzorové riešenia, ktoré držíte v rukách.

Tak neváhaj a kóď

KSPáci

1. Zorro pomstiteľ

opravovala Elfinka
(max. 15 bodov)

Ako si iste všetci všimli, cieľom bolo nájsť najmenší spoločný násobok (nsn) čísel na vstupe. Na úvod si povieme, aké nám prišli riešenia. V zásade sa dali rozdeliť na tieto kategórie:

- Zvyšovali alebo znižovali jednu premennú vždy o 1 a následne skontrolovali, či nie je hľadaným násobkom. Za takéto riešenie bolo 7 bodov.
- Zobrali najväčší prvok a testovali jeho násobky, či nie sú hľadaným číslom – 9 bodov.
- Robili spoločný násobok doterajšieho nsn a ďalšieho čísla tak, že testovali násobky jedného z týchto čísel – 11 bodov.
- Spoločný násobok pomocou najväčšieho spoločného deliteľa Euklidovým algoritmom s odčítaním – 13 bodov.
- Vzorák - 15 bodov

A teraz k hľadaniu násobku. Najmenší spoločný násobok dvoch čísel vieme vypočítať pokiaľ vieme ich najväčšieho spoločného deliteľa (nsd) podľa vzťahu $a \cdot b = \text{nsn}(a,b) \cdot \text{nsd}(a,b)$. Nsd vieme vyrátať pomocou Euklidovho algoritmu, ktorý funguje takto: Majme dve čísla a a b. Predpokladajme, že $a > b$. Potom $\text{nsd}(a,b) = \text{nsd}(a-b,b)$. Označme $d = \text{nsd}(a,b)$, $e = \text{nsd}(a-b,b)$; ukážeme, že $d = e$. Keďže d delí a aj b, musí deliť aj ich rozdiel $a-b$. (Inými slovami, ak a aj b sú násobkami čísla d, potom aj ich rozdiel je zjavne násobkom d – a podobne je na tom súčet a súčin.) Avšak e je najväčšie také číslo, ktoré delí $a-b$ aj b (je to nsd), teda $d \leq e$. Na druhej strane, e delí $a-b$ a b, preto musí deliť aj ich súčet $(a-b)+b = a$. Teda e delí a aj b a keďže $d = \text{nsd}(a,b)$, je $e \leq d$. Preto sa d a e rovnajú.

Výsledok $\text{nsd}(a,b)$ bude ten istý, ako keď od väčšieho čísla odčítame menšie, menšie ponecháme a znovu hľadáme nsd týchto čísel. Takto nám stačí počítať nsd menších a menších čísel, až sa dostaneme na nejaký triviálny prípad, keď $a = 0$ alebo $b = 0$. Vieme, že $\text{nsd}(a,0) = a$. Napríklad:

$$\text{nsd}(225,120) = \text{nsd}(120,105) = \text{nsd}(105,15) =$$

$$\text{nsd}(90,15) = \text{nsd}(75,15) = \cdots = \text{nsd}(30,15) = \text{nsd}(15,15) = \text{nsd}(15,0) = 15.$$

Toto je Euklidov algoritmus (s odčítaním), ktorý však ešte nie je veľmi rýchly (všimnite si tri bodky v uvedenom príklade – číslo 15 sme odčítavali 7-krát; ako by to vyzeralo, keby sme týmto spôsobom počítali $\text{nsd}(100000,1)$?).

Trik je, odpočítať číslo nie raz, ale toľko, koľkokrát môžeme. V k-tom kroku budeme mať nsd v tvare $\text{nsd}(a - kb, b)$. Keď už bude platiť $a - kb < b$, to, čo nám ostalo, je zvyšok po delení väčšieho čísla menším (k je podiel $\lfloor a/b \rfloor$). Náš vylepšený algoritmus bude teda počítať $\text{nsd}(a, b)$ ako $\text{nsd}(b, a \bmod b)$. Ak $b = 0$, potom výsledok je a.

Keď už vieme zistiť nsn dvoch čísel, sme len krok od N čísel. Jediné, čo stačí, je spočítať si vždy nsn všetkých čísel, ktoré sme načítali doteraz, a potom spraviť nsn tohto čísla a nového načítaného ($\text{nsn}(a, b, c) = \text{nsn}(\text{nsn}(a, b), c)$. Stačí nám teda konštantne veľa premenných – pamäťová zložitosť bude $O(1)$. Čas behu Euklidovho algoritmu je v najhoršom prípade $O(\log a)$ kde a je menšie z čísel, a nsd hľadáme N-krát, výsledná časová zložitosť teda bude $O(N \log a)$.

Listing programu:

```
var N, nasobok, novy, i : integer;

function nsd (a, b : integer) : integer;
begin
  if b = 0 then nsd := a
           else nsd := nsd (b, a mod b);
end;

function nsn (a, b : integer) : integer;
begin
  nsn := a * (b div nsd (a, b));
end;

begin
  readln (N);
  read (nasobok);
  for i := 2 to N do begin
    read (novy);
    nasobok := nsn(novy, nasobok);
  end;
  writeln(nasobok-1);
end.
```

opravoval Mišo
(max. 15 bodov)

2. Zákerná kliatba slepého piráta

Zrekapitulujme si trochu zadanie. Piráti sú v lodi a kormidlo im nefunguje, takže sa vedia hýbať len dopredu. Pred nimi (a naštastie len na pravej strane) pláva práve N nepriateľských lodí. O každej lodi vedia, dve čísla a_i, b_i. Ak sa posunú s loďou o x metrov dopredu a vystrelia, tak i-tu nepriateľskú loď trafia práve vtedy, keď $a_i \leq x \leq b_i$ (preto odteraz číslo a_i budeme nazývať začiatok lode, a číslo b_i koniec lode). Keďže však majú iba jeden výstrel, sa musia posunúť tak, aby trafili každú loď.

Aby to mohli spraviť, musia ísť ďalej ako je začiatok lode, ktorá začína najďalej, v opačnom prípade by ju netrafili, a nesmú ísť ďalej ako je koniec lode, ktorá končí najbližšie (tiež by inak nebola trafená). Preto hľadáme maximum zo začiatkov lodí (premenná max) a minimum z koncov lodí (premenná min). Dá sa to celkom jednoducho naprogramovať načítaním si vstupu do poľa a prebehnutím celého poľa. Toto ide v čase $O(N)$. Za také riešenie ste mohli získať maximálne 12 bodov. Totiž na takú jednoduchú úlohu vôbec nie je potrebné pole. Priamo budeme načítavať dve premenné a,b a porovnávame ich s max a min. Nakoniec porovnáme max a min, ak $max > min$, piráti si dospievali pesničky, ak $max \leq min$, stačí im ísť medzi max a min. Toto bolo najviac za 15 bodov. Iste ste si všetci

všimli, že sme takto vyrátali prienik intervalov (každú loď považujeme za interval). Hrôzy typu veľké dvojrozmerné pole pre každú loď a pre každý meter (či sa tam tá loď nachádza) boli za najviac 8 bodov. Za chýbajúcu alebo zle odhadnutú zložitosť som strhával po bode, za zlý popis do polovice bodov, za nefunkčný alebo chýbajúci program do polovice, bodov čo je však viac, ako za prázdny papier, ktorý dostane paušálych 0 bodov.

Listing programu:

```
var max, min, a, b, n, i : longint;
begin
    max := 0; min := 9999999;
    readln (n);

    for i:=1 to n do begin
      readln (a, b);
      if a > max then max := a;
      if b < min then min := b;
    end;
    if max > min then
      writeln ('Pán kapitán, dohára nám knôtik! Vyskočme z našej lode ',
               'a nezabudnime si plávacie rukáviky. Veď žralokom snáď ',
               'postačí pár námorníkov a časť prežije')
    else
      writeln ('Stačí nám posunúť sa o ', max, ' až ', min, ' metrov dopredu ',
               'a prerazíme tie ich škrupinky našou kovovou strelou približne '
               'kalibru sedem palcov!');
end.
```

3. Zaplnený deň

opravoval Luki
(max. 15 bodov)

Riešení prišlo celkom dosť, ale nie každé bolo správne. Na začiatku je potrebné si uvedomiť, že na úradoch strávime vždy čas $\sum_{k=1}^{n} v_k$, keďže sčítanie je komutatívne a vždy musíme navštíviť všetky úrady.

Najlepšie bude, ak úrady navštívime v takom poradí, ako sa otvoria (teda usporiadané podľa času otvorenia vzostupne). Najskôr si dokážeme, že toto poradie je ideálne.

Dôkaz: Nech existuje poradie, kde existuje i, j také, že $o_i < o_j$ a najskôr budeme vybavovať j-ty úrad a tesne za ním i-ty úrad, a zároveň čas, kedy skončíme vybavovanie na úradoch je minimálny možný[1]. Ukážeme, že naše usporiadanie (podľa časov otvorenia) je rovnako dobré. Je zjavné, že vybavovanie i-teho úradu začneme hneď po vybavení úradu j. Zoberme čas – t, kedy sme začali vybavovanie v úrade j. Potom vybavenie v úradoch j a za tým i bude trvať do času $t+o_j+o_i$. V čase t je podľa predpokladov otvorený aj úrad i. Preto kľudne môžeme vtedy vybaviť úrad i a po ňom až úrad j. Návšteva úradov v tomto poradí bude trvať rovnako – $t + o_i + o_j$ a preto môžeme použiť usporiadanie, kde je úrad i pred úradom j a vybavovanie úradov skončíme v rovnakom čase. Tým sme ukázali, že výmenou dvoch po sebe idúcich úradov, tak aby prvý bol ten, čo otvorí skôr, nám nič nepokazí. Takto vieme vymieňať ľubovolné dva susedné úrady a teda ich usporiadať podľa začiatku otvorenia vzostupne.

Obyčajným utriedením dostaneme riešenie, ktoré behá v čase $O(NlogN)$. Keďže ale T môže byť maximálne 10 000 oproti N, ktoré môže byť neobmedzene veľké, tak môžeme použiť iný typ triedenia, a to konkrétne CountSort. Príncíp je jednoduchý. Chceme utriediť

[1] ak také i, j neexistuje, tak máme usporiadané poradie a nemáme čo dokazovať

N prvkov z intervalu $0..T$. Spočítame si počet prvkov, ktoré sa rovnajú 0, počet prvkov, ktoré sa rovnajú 1 a takto budeme rátať až po T (toto vieme spraviť jedným prechodom poľa v čase $O(N)$) a uložíme si ich v poli *sucty*[2]. Teraz máme pole *sucty* naplnené tak, že hodnota *sucty*[k] je počet prvok vo vstupnom poli veľkosti k. Teraz prejdeme raz pole *sucty* a spočítame si pre každú hodnotu k, koľko prvkov je menších alebo rovných, ako naša hodnota (teda $\sum_{i=1}^{k}$ *sucty*[i]). Tieto hodnoty si vieme napočítať v poli *sucty* tak, že ho lineárne prejdeme a urobíme *sucty*[$i+1$] =*sucty*[$i+1$]+*sucty*[i]. [3]

V tomto momente obsahuje pole *sucty*[k] počet prvkov vstupného poľa, ktoré sú menšie alebo rovnaké ako prvok s hodnotou k. Poďme prechádzať vstupné pole, ktoré chceme utriediť. Nech H_i je hodnota i-teho prvku tohto poľa. Potom na pozícií *sucty*[H_i] budeme mať číslo pozície posledného prvku, s hodnotou H_i (na začiatku nám to platí). Budeme postupne prechádzať vstupné pole (cyklus od 1 po N s premennou i). i-ty prvok dáme do výstupného poľa na pozíciu *sucty*[H_i]. Keďže ale *sucty*[H_i] nám už ukazuje na obsadený prvok, tak *sucty*[H_i] znížime o 1. Prečo to robíme? Prvky s hodnotou H_i chceme dať na pozície *sucty*[H_{i-1}]$+1, \cdots$*sucty*[H_i]. Prvý výskyt prvku H_i dáme teda na pozíciu *sucty*[H_i]. Znížením o jednotku dosiahneme to, že ďalší výskyt hodnoty H_i sa dostane na pozíciu *sucty*[H_i] -1. Keď budeme takto pokračovať, tak k-ty výskyt bude uložený na pozíciu *sucty*[H_i] $-k+1$ a posledný výskyt prvku H_i sa dostane na pozíciu *sucty*[H_{i-1}] $+1$. Keď takto prejdeme celé pole, tak sa nám v druhom poli objaví utriedená postupnosť. Všimnite si, že týmto spôsobom sme utriedili čísla v čase $O(N)$.

Hodnotenie:

- riešenia so zložitosťou $O(N^2)$ – 10 bodov
- riešenie so zložitosťou $O(N \log N)$ – 12 bodov
- riešenie so zložitosťou $O(N)$ – 15 bodov
- nesprávne riešenia – max 4 body

Za žiadnu snahu o dôkaz bol -1 bod a potom niekedy sem tam strhnutý bodík alebo 2 za chabý alebo žiaden popis.

Listing programu:

```
{mic to prepise do packalu}
type TUrad=record
       o, v, i : integer;
     end;
var N, T, o, v, i, cas : integer;
    utriedene, urady : array [1..100] of TUrad;
    sucty : array [0..10000] of integer;

procedure CountSort;
var i:integer;
begin
  for i := 0 to T do sucty[i] := 0;
  for i := 1 to N do inc (sucty[urady[i].o]);
  for i := 1 to T do inc (sucty[i],sucty[i-1]);
  for i := 1 to N do begin
    utriedene[sucty[urady[i].o]] := urady[i];
    dec(sucty[urady[i].o]);
  end;
end;

begin
  readln (N, T);
```

[2] hneď v nasledujúcej vete pochopíte, prečo som to pole nazval tak

[3] Rozmyslite sa, že prečo.

```
  for i := 1 to N do begin
    readln(o,v);
    urady[i].o := o;
    urady[i].v := v;
    urady[i].i := i;
  end;
  CountSort;
  cas:=0;
  for i:=1 to N do begin
     if cas < utriedene[i].o then cas := utriedene[i].o;
     writeln (utriedene[i].i, ' ', cas);
     cas := cas + utriedene[i].v;
  end;
end.
```

Opravovala Danka
(max. 15 bodov)

4. Obludný regiment

Prišiel celý kopec riešení tohoto príkladu, večšina bola správna, alebo skoro správna. Pomerne často ste však zabudli, ze konečné usporiadanie regimentu može začínať aj človekom, aj trollom.

Podľa zadania nesmú byť žiadne jednotky ani nuly vedľa seba. Teda na konci by mala vzniknúť postupnosť, ktorá bude vyzerať asi takto ..101010.. Otázka však je, či bude začínať jednotkou alebo nulou. Označme si postupnosť začínajúcu jednotkou ako P1 a postupnosť začínajúcu nulou ako P2. Na chvíľu predpokladajme, že postupnosť vo výstupnom poli bude P1. Takže postpnosť by mala vyzerať takto 101010.... Túto postuposť porovnáme s postupnosťou vo vstupnom poli A. Tieto postupnosti sa na niektorých pozíciach zhodujú. Čísla na týchto pozíciach už nemusíme vymieňať. Veď ako by nám len pomohlo, keby sme zamenili dve správne hodnoty. Pozrime sa na tie ostatné. Nech je tam x jednotiek. Tieto jednotky sú na nesprávnych pozíciach, teda na pozíciach núl. Každú z nich budeme musieť vymeniť za nulu. Na toto budeme potrebovať aspoň x výmien.

Keď ale je x jednotiek na pozíciach núl, potom tieto nuly sa nachádzajú na pozíciach jednotiek. Teda máme aspoň x zle umiestnených núl. Môže ich byť viac? Odpoveď je nie, lebo ak by bola zle umiestnená nejaká nula, obsadila by miesto nejakej jednotke. Ale my sme povedali, že zle umiestnených jednotiek je len x.

Teda máme x zle umiestnených jednotiek a x zle umiestnených núl. Takže stačí vymeniť vždy jednu zle umiestnenú jednotku za zle umiestnenú nulu. Na to treba x výmen.

Takže aby sme dostali z postupnosti v poli A postupnosť P1, potrebujeme x výmen. Ale nie je lepšie, aby postupnosť bola P2? Koľko výmen potrebujeme na toto? Ak sa lepšie pozrieme na postupnosti P1 a P2, je zrejmé, že prvky zle umiestnené vzhľadom na P1 sú správne umiestnené vzhľadom na P2 a naopak. Teda, ak bolo vzhľadom na P1 zle umiestnených x núl a x jednotiek, potom bude vzhľadom na P2 zle umiestnených $N - x$ núl a $N - x$ jednotiek.

Všimneme si ešte poslednú vec, v postupnosti P1 sa striedajú jednotky a nuly, teda jednotky by mali byť na nepárnych pozíciach a nuly na párnych. Jednotky sú zle umiestnené ak sú na párnych pozíciach, kde majú byť nuly. Takže stačí, že spočítame jednotky na párnych pozíciach a výsledkom bude to menšie z čísiel x a $N - x$.

Časová zložitosť bude $O(N)$, pretože nám stačí len raz prejsť pole A a každý druhý prvok porovnať s postupnosťou P1 alebo P2.

Listing programu:

```
var N, a, x: integer;
begin
  readln (N);
  for i := 1 to 2*N do begin
    read (a);
    if i mod 2 = 1 then x := x + a;
  end;
  if  N-x < x then x := N-x;
  writeln ('Je potrebne urobit aspon ', x, ' vymien.');
end.
```

5. O tajnostkárkach a tajnostkároch

opravoval Maják
(max. 15 bodov)

Aby sa nám ľahšie rozmýšľalo, zjednodušme si náš problém. Na vstup sa môžme pozerať ako na graf, pričom vrcholy budú reprezentovať ľudí, a hrany zverejnené pikošky. Pohlavie človeka bude „farba“ vrchola. Ak sa medzi ľudmi nemusí nachádzať žiaden ufón (nevieme to zistiť), tak susedné vrcholy budú mať vždy rôznu farbu. To znamená, že ak sa dá graf ofarbiť dvomi farbami, tak sa tam ufón nemusí nachádzať. Naopak, ak sa daný graf nedá ofarbiť dvomi farbami (nejaké dva susedné vrcholy majú rovnakú farbu), tak sa tam ufón určite nachádza. A ako budeme ofarbovať? Obyčajným prehľadávamím do hĺbky. To funguje nasledovným spôsobom: Predstavme si, že sme sa dostali do nejakého vrcholu grafu. Zistíme si, kam nám z neho vedie hrana, a ak sme v tom-ktorom vrchole ešte neboli, tak ho označíme ako pridaný, a pokračujeme v prehľadávaní z neho. Ak sa z vrcholu, v ktorom práve sme, už nemožme dostať do žiadného ešte nenavštíveného vrcholu, tak sa vratime (tak ako sme prišli) späť a z predchádzajúceho vrcholu znovu skúmame, či sa ešte vieme niekam dostať. Týmto spôsobom postupne prejdeme cez všetky vrcholy grafu.

K tomuto algoritmu pridáme jedno vylepšenie. Keď vrcholy označujeme, tak ich budeme označovať farbou opačnou k vrcholu v ktorom sa práve nachádzame (-1 alebo 1; 0 znamená že sme tam ešte neboli). Ak nastane prípad, že chceme ofarbiť vrchol na jeho farbu, tak sa nič nedeje. Ale ak by sme ho chceli ofarbiť na farbu ktorú nemá, tak sme zistili, že graf sa nedá ofarbiť dvomi farbami, môžme zastaviť prehľadávanie, pretože musí existovať ufón.

Tento príklad obsahoval jeden malý zádrhel, na ktorý ste mnohí zabudli. Zadaný graf nemusí byť súvislý (t.j. môže pozostávať z viacerých komponentov), a preto by sme sa jedným prehľadávaním nedostali do všetkých vrcholov. Preto prehľadávanie spustíme z každého vrcholu, ktorý sme ešte neprehľadali.

Prehľadávaním spracujeme každý vrchol raz a na každú hranu sa pozrieme dva razy, preto časová zložitosť bude $O(N + M)$.

Pamäťová zložitosť sa odvíja od toho, akým spôsobom si pamätáme graf. Optimálne je pre každý vrchol si pamätať hrany ktoré z neho vedú, potom je zložitošt $O(N + M)$. Ak si pamätáme pre každú dvojicu vrcholov či medzi nimi je hrana, tak pamäťová aj časová zložitosť bude $O(N \cdot N)$.

Posledných pár slov sa bude týkať bodovania. Za zlé/chýbajúce odhady ste mohli stratiť dokopy 2 body, za nebratie do úvahy možnú nesúvislosť grafu 3 body, za horšiu zložitosť do 5 bodov a za nefunkčné riešenie viac ako polovicu.

Listing programu:

```
const maxN = 100;
var farba : array [1..maxN] of integer; { pole s farbami jednotlivých vrcholov }
```

```
    hrana : array [1..maxN, 1..maxN] of integer; { pole hovoriace, kam vedie ktorá
hrana }
    hranz : array[1..maxN] of integer; { pole hovoriace, koľko hrán vedie z každého
vrchola }
    ufoni : boolean; { boolean indikujúci prítomnosť ufónov }
    i, n : integer;

procedure nacitaj;
var i, a, b, hran : integer;
begin
  readln(n,hran);
  for i := 1 to n do hranz[i] := 0;
  for i := 1 to hran do begin
    readln (a, b);
    inc(hranz[a]);
    hrana[a,hranz[a]] := b;
    inc (hranz[b]);
    hrana[b,hranz[b]] := a;
  end;
end;

function ofarbi (v : integer) : boolean;
var i : integer;
begin
  if not ufoni then
    for i := 1 to hranz[v] do { ak som ešte nenašiel ufóna, tak sa pozriem kam sa
viem dostať }
      if farba[hrana[v,i]] = 0 then begin { ak som tam ešte nebol }
        farba[hrana[v,i]] := -farba[v]; { tak ho zafarbím opačnou farbou }
        ofarbi := ofarbi(hrana[v,i]); { a pokračujem ďalej ním }
      end
      { ak som tam už bol a má rovnakú farbu ako vrchol v ktorom som, tak som našiel
ufóna }
      else if (farba[hrana[v,i]] = farba[v]) then ofarbi := false
      else ofarbi:=true;
end;

begin
  nacitaj; { načítam vstup }
  ufoni := false; { prítomnosti ufónov ešte nie som presvedčený }
  for i := 1 to N do farba[i]:=0; { na začiatku som ešte nič neofarbil }
  for i := 1 to N do
    if not ufoni and (farba[i] = 0) then begin { ak som ešte neofarbil i-ty vrchol }
      farba[i] := 1; { tak mu dám začiatočnú farbu }
      if not ofarbi(i) then ufoni := true; { a ak sa mi tento komponent nepodarí
ofarbiť, tak tam ufón je }
    end;
  if ufoni then writeln('Ano')
           else writeln('Nie');
end.
```

6. O lakomom a štedrom Maroškovi

opravoval KuKo
(max. 15 bodov)

Hodnotenie: Asi najlepšie možné riešenie bolo vkladanie v $O(1)$ a výber v amortizovanej zložitosti $O(\log n)$ (ale $O(n)$ v najhoršom prípade, ak sme s dačím takým spokojní), alebo – čo sme očakávali – $O(\log n)$ v najhoršom prípade na všetky operácie – za to bolo 15 bodov.

Prišlo viacero pomalších riešení: vkladanie v $O(n)$ a výber v $O(1)$ dostával 10 bodov, vkladanie v $O(1)$ a výber v $O(n)$ bol za 7, horšie ešte menej. (Väčšinou som vám do riešení nakreslil tabuľku, aké rýchle sú ktoré operácie ins/min/max a podľa toho boli body.)

Jednoduché riešenia: Najjednoduchšie riešenie je mať jedno pole a pamätať si, koľko kníh už v ňom máme. Knihu vždy pridáme na koniec (na $(n+1)$-vé políčko) a zvýšime n. Super je, že vieme rýchlo vkladať. Keď však chceme vyberať (minimum alebo maximum), musíme prejsť celé pole a nájsť takú knihu. Potom viacerí z vás túto knihu vymazali a posunuli celý zvyšok poľa o 1 doľava. Stačí pritom takto vzniknutú dieru „zaštupľovať“ posledným prvkom a znížiť n. (Toto je to 7-bodové riešenie $O(1)/O(n)/O(n)$).

Lepšie je udržiavať pole utriedené. Dáme si väčšiu námahu pri vkladaní: nájdeme, kam novú knihu vopcháme (zvyšok poľa musíme posunúť). Takto vieme nájsť (a ak to dobre napíšeme, aj odstrániť) minimum, resp. maximum – je to prvá, resp. posledná kniha v našom zozname. Toto sa dá naprogramovať pomocou tzv. spájaného zoznamu – pre každú knihu si pamätáme, kde je tá nasledujúca a kde je tá predchádzajúca. Druhá možnosť je napísať to podobne ako frontu (až na to, že knihy budú utriedené podľa ceny).[4]

Halda: Na viac-menej vzorové riešenie budeme potrebovať dátovú štruktúru, ktorá sa volá halda. Vysvetlenie haldy tiež nájdete v `http://ksp.sk/ksp/zadania/prilohy/ps243/6/kp_teoria_1.pdf`; alebo na wikipédii: `http://en.wikipedia.org/wiki/Binary_heap`.

Halda je binárny strom – vo vrcholoch máme čísla a každý vrchol má najviac dvoch synov. Vždy platí, že v otcovi je uložené väčšie číslo, ako v jeho synoch. Navyše je halda úplný binárny strom, to znamená, že všetky listy sú na jednej úrovni, prípadne posledná úroveň nemusí byť úplná, ale listy idú zľava; inými slovami, v strome nie je „diera“.

Haldu vieme ľahko implementovať priamo v poli: políčko 1 bude koreň a vždy ľavý syn i-teho vrcholu bude na políčku $2i$ a pravý na políčku $2i+1$. Teda otec je na políčku $\lfloor i/2 \rfloor$. Vďaka usporiadaniu haldy je najväčší prvok v koreni.

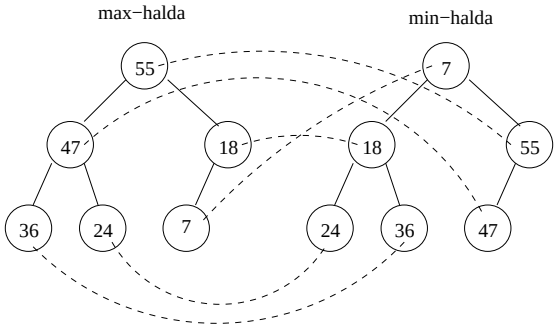
Vkladáme tak, že pridáme prvok na koniec poľa ($(n+1)$-vé políčko, zvýšime n) a „vybubleme“ ho nahor. Môže sa totiž stať, že je väčší ako jeho otec, čím je porušené usporiadanie haldy. Vtedy ho vymeníme s otcom. Ak je väčší ako jeho nový otec, vymeníme ho aj s ním, a tak ďalej, až kým sa nedostaneme do koreňa alebo nenapravíme usporiadanie haldy.

Vymazať nejaký prvok vieme tak, že ho zameníme za posledný prvok, zmenšíme n a „prebubleme“ ho nadol (kým nenapravíme usporiadanie haldy alebo sa nedostaneme k listom).

Takáto halda sa tiež volá max-halda, pretože do nej vieme rýchlo vkladať a rýchlo z nej vyberať maximum. Veľmi podobne vieme spraviť min-haldu, z ktorej vieme rýchlo vyberať minimum (v skutočnosti stačí zmeniť znamienka $>$ na $<$). Ako rýchlo vieme dané operácie robiť? V oboch prípadoch len prebubleme nejaký prvok nahor alebo nadol, čo je v najhoršom prípade úmerné výške haldy a tá je logaritmická.

Čo s haldami: Popísali sme si max-haldu aj min-haldu, no žiadna nám problém celkom nerieši. Z max-haldy vieme rýchlo vybrať maximum; zato pri hľadaní minima sme úplne bezradní – najlepšie, čo vieme spraviť je prezerať iba listy – vieme, že minimum bude v liste. Môže však byť v ktoromkoľvek liste a počet listov je až zhruba polovica počtu všetkých vrcholov! Naopak, v min-halde vieme rýchlo vybrať minimum, ale pri maxime sme stratení. Čo si počneme?

[4] Ak už neviete naprogramovať ani frontu, pozrite si `http://ksp.sk/ksp/zadania/prilohy/ps243/6/kp_teoria_1.pdf`



Použijeme dve haldy – jednu min, jednu max. Pri vkladaní vložíme prvok do jednej aj do druhej. Navyše prvky „prepojíme“, tzn. popri cene knihy si budeme pamätať aj pozíciu, kde sa nachádza rovnaká kniha v druhej halde (pozri obr.). Podľa toho, či budeme vyberať minimum, alebo maximum, ho nájdeme v koreni tej správnej haldy a vďaka prepojeniu vieme príslušnú knihu rýchlo nájsť a vymazať aj z druhej haldy.

Toto riešenie je implementované na konci tohto textu; h je pole, ktoré obsahuje min-haldu aj max-haldu (h[`false`] je min-halda, h[`true`] je max-halda). Procedúra `swap` vymení dva prvky (a upraví príslušné odkazy medzi haldami. Procedúry `up` a `down` bublú i-ty prvok smerom nadol. Parameter m tu určuje, s ktorou haldou sa hráme – či s min alebo s max.

Porovnania som napísal trochu trikovo: napríklad pri prebublávaní smerom nahor bubleme, kým je otec menší ako daný prvok – ak sme v max-halde. Ak sme v min-halde, bubleme, kým je otec nie je menší ako daný prvok. Inými slovami, bubleme, kým je otec menší práve vtedy, keď sme v max-halde – tomu zodpovedá podmienka `(h[m, i div 2].c < h[m, i].c) = m`. Rozmyslite si, že to funguje.

Lepšie: No a ako sa to dalo lepšie? Pre tento problém bola vymyslená špeciálna dátová štruktúra, tzv. min-max halda, z ktorej vieme vyberať aj minimum aj maximum v logaritmickom čase – ušetríme tak dosť pamäte, lebo nemusíme mať prvky uložené dvojmo a nemusíme si pamätať odkazy medzi dvoma haldami. Dostaneme tak o nejakú tú konštantu rýchlejšie riešenie vyžadujúce menej pamäti. Záujemci sa o tejto štruktúre dočítajú napríklad z článku `http://www.cs.otago.ac.nz/staffpriv/mike/Papers/MinMaxHeaps/MinMaxHeaps.pdf`.

Vieme dosiahnuť ešte lepšie riešenie – lepšie je za predpokladu, že viac vkladáme ako vyberáme a že chceme optimalizovať čas *postupnosti viacerých operácií po sebe* a netrápi nás čas jednej konkrétnej operácie). Konkrétne vieme dosiahnuť konštantnú zložitosť na vkladanie a *amortizovanú zložitosť* $O(\log n)$ na výber minima alebo maxima. Znamená to asi toľko, že ak budeme mať n vkladaní a m výberov, spolu to bude trvať $O(n + m \log n)$ – čo je lepšie ako vyššie popísané logaritmické riešenie, ktoré bude trvať $O(n \log n + m \log n) = O((n + m) \log n)$. Dôvod, prečo to nemusí byť vždy lepšie je, že niektoré výbery môžu trvať oveľa dlhšie, dokonca rádovo $\Theta(n)$.

Treba si teda rozmyslieť, čo od dátovej štruktúry chceme. Ak chceme, aby nám, vždy keď sa jej spýtame, dala čo možno najrýchlejšie odpoveď, použijeme 2 haldy alebo min-max haldu, ktoré garantujú pre každú operáciu dobrý čas. Ak vieme, že budeme dávať dátovej štruktúre veľa príkazov a nevadí nám, že raz to bude rýchlo, raz pomaly, ale chceme, aby to dokopy bolo čo najmenej, použijeme riešenie, ktoré si teraz popíšeme.

V zásade na to budeme potrebovať iba dve veci. Jedna je tzv. mergeable heap – čo by sa možno dalo preložiť ako „spájateľnú haldu“. To je taká dátová štruktúra, že keď máme 2 haldy (s prvkami H_1 a H_2), vieme ju rýchlo spojiť (t.j. vytvoriť haldu s prvkami $H_1 \cup H_2$). Napríklad dve „obyčajné“ (napr. max-)haldy, ktoré sme si popísali vyššie, vieme spojiť v čase $O(n)$, ale nie podstatne lepšie. Na druhej strane, existujú také haldy, ktoré to dokážu v logaritmickom čase. Možno niektorí z vás počuli o Fibonacciho halde, alebo binomiálnej halde. V skutočnosti stačí celkom jednoducho implementovateľná halda, tzv. leftist heap (Fibonacciho halda dokáže iné super kúsky, ktoré však my vôbec nepotrebujeme – a bez knižnice by sme

chceli mať implementáciu čo najjednoduchšiu, však). Potrebné popisy aj s implementáciou sa dajú nájsť na `http://www.brpreiss.com/books/opus4/html/page352.html`.

Druhá vec, ktorú budeme potrebovať je lenivosť. Vtip je v tom, že pri vkladaní nebudeme nič robiť, iba hádzať prvky na kopu vedľa haldy. Až príde na lámanie chleba (na výber), začneme upratovať. Z kopy najskôr spravíme haldu, potom ju spojíme s už vybudovanou haldou a nakoniec z nej klasickým spôsobom vyberieme hľadaný prvok.

Toto že je rýchlejšie? Nuž poďme sa na to pozrieť. Predstavme si takúto situáciu: v halde už máme n prvkov; následne vložíme ďalších k a potom jeden vyberieme. Klasickým spôsobom by to bolo v čase $O(k \log(n+k) + \log(n+k))$. Naším lenivým spôsobom vložíme každý prvok v konštantom čase – spolu $O(k)$. Pri výbere najskôr z toho k-prvkového „bordelu“ spravíme haldu – to vieme v $O(k)$, následne 2 haldy spojíme v $O(\log(n+k))$ a výber spravíme tiež v $O(\log(n+k))$ – spolu je to $O(k + \log(n+k))$.

Ako vytvoríme haldu z k prvkov v $O(k)$? Na každý prvok sa môžeme pozerať ako na haldu. Najskôr teda spojíme haldy veľkosti 1, dostaneme $k/2$ háld veľkosti dva. Tie po dvojiciach spojíme a dostaneme $k/4$ háld veľkosti 4, atď. Pre čas dostaneme sumu tvaru $\sum_{i=1}^{\lg k} ik/2^i$, čo je $O(k)$.

Trik teda spočíva v tom, že sme vkladanie odsunuli a vložiť naraz k prvkov vieme rýchlejšie ako vložiť ich po jednom – zvyšný čas sa „skryje“ v zložitosti výberu.

Listing programu:

```
const MAXN = 1000;
      MIN = false;
      MAX = true;

type rec = record
       c, p : integer; { cena a pozicia v druhej halde }
     end;
     minmax = object
       private
         h : array [MIN..MAX, 1..MAXN] of rec; { min a max halda }
         n : integer; { pocet prvkov }
         procedure swap (i, j : integer; m : boolean);
         procedure up (i : integer; m : boolean);
         procedure down (i : integer; m : boolean);
       public
         procedure insert (x : integer);
          function get (m : boolean) : integer;
     end;

var K : minmax;
    op, arg : integer;

procedure minmax.swap (i, j : integer; m : boolean);
var c1, c2, p1, p2 : integer;
begin
  c1 := h[m, i].c; c2 := h[m, j].c; h[m, i].c := c2; h[m, j].c := c1;
  p1 := h[m, i].p; p2 := h[m, j].p; h[m, i].p := p2; h[m, j].p := p1;
  h[not m, p1].p := j; h[not m, p2].p := i;
end;

procedure minmax.up (i : integer; m : boolean);
begin
  while (i > 1) and ((h[m, i div 2].c < h[m, i].c) = m) do begin
    swap (i, i div 2, m);
    i := i div 2;
  end;
```

```
end;

procedure minmax.down (i : integer; m : boolean);
var j : integer;
begin
  while (2*i <= n) do begin
    j := 2*i; if (h[m, j+1].c > h[m, j].c) = m then inc (j);
    { j je ten vaecsi / mensi zo synov i }
    if (h[m, j].c <= h[m, i].c) = m then break;
    swap (i, j, m);
    i := j;
  end;
end;

procedure minmax.insert (x : integer);
begin
  inc (n);
  h[MIN, n].c := x; h[MAX, n].c := x;
  h[MIN, n].p := n; h[MAX, n].p := n;
  up (n, false); up (n, true);
end;

function minmax.get (m : boolean) : integer;
var p : integer;
begin
  get := h[m, 1].c; p := h[m, 1].p;
  swap (1, n, m); swap (p, n, not m);
  dec (n);
  down (1, m); down (p, not m);
end;

begin
  read (op);
  while (op > 0) do begin
    case op of
      1: begin read (arg); K.insert (arg); writeln ('ins ', arg); end;
      2: writeln ('max: ', K.get (MAX));
      3: writeln ('min: ', K.get (MIN));
    end;
    read (op);
  end;
end.
```

Chýba súbor prikl7.tex!!!

7. Obyčajné triedenie?

opravoval MišoF.
(max. tak 25 bodov)

V prvom rade upozornenie. Na tejto úlohe sa dalo robiť aj týždeň. Kto si na to nechal v nedeľu pol hodinky, s veľa bodmi rátať nemohol. Naopak, keby niekto poslal všetko, čo som spísal ako vzorové riešenie, pokojne by aj tých 25 bodov dostal. (To sme ale nečakali. Tá časť vzorového riešenia, ktorá mala byť vo vašich silách, približne zodpovedá 15 bodom.)

V druhom rade reklama. Toto vzorové riešenie, ktoré je uvedené v tlačenej verzii, je nutne osekané a ochudobnené o rôzne šťavnaté kúsky, ako napríklad krásne farebné grafy a dokonca

jedno video. Ak máš poruke internet, prestaň čítať túto verziu a presuň sa na omnoho lepšiu verziu, ktorú nájdeš na `http://www.ksp.sk/ksp/zadania/prilohy/ps243/8/`.

Ak internet po ruke nemáš, nezúfaj, tie najdôležitejšie časti nájdeš aj tu.

Netradične začneme listingami programov, tie boli tentokrát najľahšou časťou riešenia zadanej úlohy.

Listing programu:

```
#include <iostream>
#include <algorithm>
#include <sys/time.h>
#include <time.h>
using namespace std;
#define MAX 1234567

int P[MAX], tmp[MAX];
int N;
char typ;
struct timeval t_start, t_end;

void sort_A() {
  while (1) {
    // vymienanie
    for (int loop=0; loop<N; loop++) {
      int x,y;
      while (1) { x=rand()%N; y=rand()%N; if (x<y) break; }
      if (P[x]>P[y]) swap(P[x],P[y]);
    }
    // kontrola
    int ok=1;
    for (int i=0; i<N-1; i++) ok &= (P[i]<=P[i+1]);
    if (ok) break;
  }
}

void sort_B(int zac, int kon) {
  if (kon-zac <= 1) return;
  if (kon-zac == 2) { if (P[zac]>P[zac+1]) swap(P[zac],P[zac+1]); return; }
  int dlzka = (2*(kon-zac)+2)/3;
  sort_B(zac,zac+dlzka);
  sort_B(kon-dlzka,kon);
  sort_B(zac,zac+dlzka);
}

void sort_C() {
  while (1) {
    // vymienanie
    for (int i=0; i<N-1; i++) tmp[i]=i;
    random_shuffle(tmp,tmp+N-1);
    for (int i=0; i<N-1; i++)
      if ( P[tmp[i]] > P[tmp[i]+1] ) swap( P[tmp[i]], P[tmp[i]+1] );
    // kontrola
    int ok=1;
    for (int i=0; i<N-1; i++) ok &= (P[i]<=P[i+1]);
    if (ok) break;
  }
}

int main() {
```

```
  gettimeofday(&t_start,NULL); srand(time(NULL)+t_start.tv_usec);
  cin >> typ >> N;
  for (int i=0; i<N; i++) P[i]=rand();
  gettimeofday(&t_start,NULL);
  if (typ=='A') sort_A();
  if (typ=='B') sort_B(0,N);
  if (typ=='C') sort_C();
  gettimeofday(&t_end,NULL);
  long long cas = 1000000LL * (t_end.tv_sec - t_start.tv_sec)
                + (t_end.tv_usec - t_start.tv_usec);
  cout << "sort: " << typ << " N: " << N << " cas: " << cas << " us" << endl;
}
```

Poznámky k implementácii

V algoritme A viacerí z vás použili nasledujúci postup:

```
vacsi = 1 + rand() % (N-1);
mensi = rand() % vacsi;
```

Treba si ale uvedomiť, že tento postup nezodpovedá tomu, čo sme mali spraviť. Nie každá dvojica indexov (mensi,vacsi) bude mať rovnakú pravdepodobnosť. Napríklad dvojica $(1, 0)$ bude vychádzať omnoho častejšie ako $(N - 1, 0)$.

Lepšie bolo generovať oba indexy nezávisle na sebe, a v prípade, ak náhodou dostaneme rovnaké, celý proces zopakovať. (Pre veľké N toto bude nastávať zanedbateľne často. Očakávaný počet potrebných pokusov sa s rastúcim N zjavne blíži k 1.)

V algoritme C miešanie kartičiek najľahšie naprogramujeme tak, že do pomocného poľa nahádžeme indexy prvkov (okrem posledného, ktorý už napravo od seba nemá dvojicu), a následne toto pole náhodne prehádžeme. Na to je najlepší spôsob nasledovný:

Zoberieme pole dĺžky N, ktoré chceme premiešať. Vygenerujeme náhodný index prvku, teda náhodné číslo od 0 do $N - 1$. Toľký prvok presunieme na posledné miesto a až do konca ho tam necháme. So zvyšnými prvkami tento postup opakujeme, až kým sa neminú. (V druhom kroku teda budeme generovať index od 0 do $N - 2$, v treťom do $N - 3$, atď.)

Tento postup má zjavne časovú zložitosť lineárnu od dĺžky poľa, a zároveň je evidentné, že každá možná permutácia poľa je rovnako pravdepodobná.

Čo povie testovanie

V online verzii riešenia si môžeš podrobne pozrieť, ako dopadli naše implementácie algoritmov, keď sme ich testovali na rôzne veľkých náhodných vstupoch (pre N do 10 000).

Postup merania bol nasledovný: Pre každú veľkosť vstupu sme robili 15 meraní a ako nameranú hodnotu sme zobrali ich medián, teda prostrednú nameranú hodnota po utriedení. Brať medián je lepšie ako priemer, takto vieme, že nám ani prípadné divoké chyby merania výsledok principiálne neovplyvnia.

Z grafov sa dá odpozorovať nasledovné závislosti: Časová zložitosť algoritmu A je niekde tesne nad N^2, čas algoritmu B sa pohybuje zvláštnymi skokmi a význačným bodom v ňom zhruba zodpovedá funkcia $N^{2.71}/80$, no a čas behu algoritmu C dlho vyzeral dobre, ale pri testovaní na veľkých vstupoch sa ukázalo, že je približne $N^2/25$.

Toto sú teda také návody, že kde hľadať správne dolné a horné odhady časovej zložitosti. A zároveň nám tieto merania môžu slúžiť ako skúška správnosti. Ak niečo spočítame a nebude to sedieť s výsledkami meraní, niekde sme sa asi sekli.

Algoritmus B

Začneme najľahším prípadom, a to je Brankov algoritmus B. Tu nehrá náhoda žiadnu rolu, a takisto skoro žiadnu rolu nehrá obsah triedeného poľa. Možno budeme robiť raz viac,

raz menej výmen, ale výmen bude vždy najviac toľko ako porovnaní, no a počet porovnaní závisí jedine od veľkosti triedeného poľa. Tú budeme označovať N.

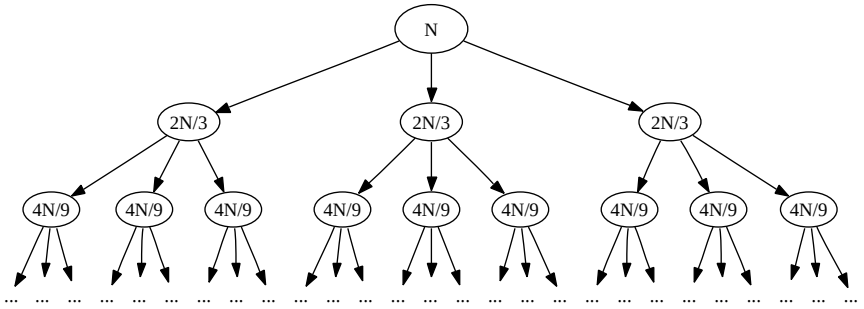
Ako spočítať časovú zložitosť takéhoto algoritmu? Existuje veľa možných postupov. Môžeme si napríklad označiť $T(N)$ čas, ktorý náš algoritmus potrebuje na utriedenie poľa veľkosti N. Potom môžeme túto časovú zložitosť tak približne vyjadriť nasledovne:

$$\forall N > 2; \quad T(N) = 3T(2N/3) + O(1)$$

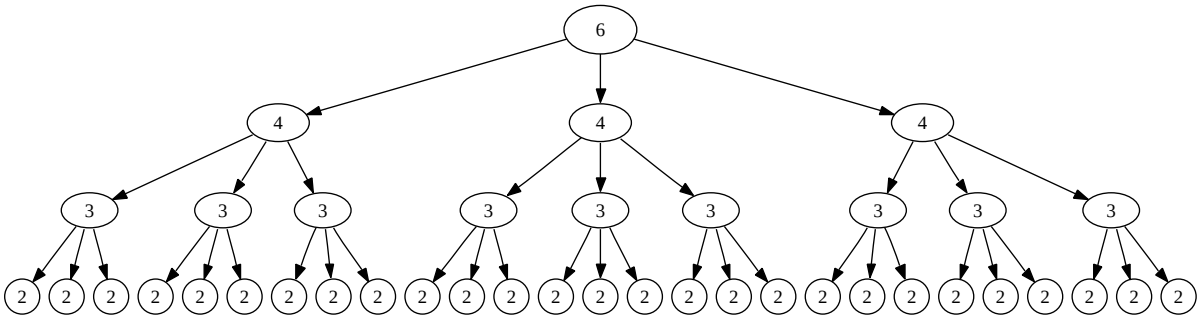
(Slovne: Čas potrebný na utriedenie poľa veľkosti N dostaneme tak, že zoberieme trojnásobok času potrebného na utriedenie menšieho poľa, a navyše pridáme nejakú konštantu, ktorá zodpovedá zvyšnej práci, ktorú v procedúre spravíme.)

Ako sa ale z takéhoto zápisu vysomáriť a zistiť, aká je vlastne tá zložitosť?

Jednou metódou je nakresliť si stromček, ktorý bude znázorňovať jednotlivé rekurzívne volania. Začneme tým, že nakreslíme koreň stromu a do neho napíšeme N – toto bude prvé volanie, „utrieď celé pole“. Toto volanie prebieha tak, že trikrát sa zavoláme na pole veľkosti (približne) $2N/3$. Tak nakreslíme koreňu troch synov a do každého z nich napíšeme $2N/3$, teda aktuálnu veľkosť poľa. Teraz každý z týchto vrcholov dostane pod seba ďalšie tri, v ktorých bude napísané $4N/9$, a tak ďalej.[5]



Keby sme si takýto stromček kreslili pre nejaké konkrétne N, dostali by sme sa časom na úroveň, kedy by už triedené pole malo veľkosť menšiu ako 3. Tam už nerobíme žiadne ďalšie rekurzívne volania, a teda by strom končil.



To, čo je teraz pre nás zaujímavé, je, že tento strom znázorňuje celú prácu, ktorú sme pri algoritme spravili – inými slovami, jeho časovú zložitosť. Ku každému kroku algoritmu vieme nájsť vrchol v strome, ktorý zodpovedá okamihu, kedy sme tento krok spravili.

Chceme teraz celú prácu, ktorú sme spravili, spočítať. V našom prípade je to celkom jednoduché: pri každom volaní rekurzívnej procedúry spravíme okrem samotných rekurzívnych volaní len konštantne veľa iných krokov. Preto celková práca, ktorú spraví náš algoritmus, je priamo úmerná počtu vrcholov stromu.

[5] Všimnime si, že pre jednoduchosť sa netrápime celými časťami. Ak niekoho trápi, prečo si to môžeme dovoliť: S rastúcim N časová zložitosť nášho programu zjavne rastie. Ak zistíme, ako sa správa pre také N, pre ktoré zaokrúhľovať netreba, ostatné hodnoty medzi tým nám to nemajú ako pokaziť. (Aj keď napríklad z grafu vidíme, že sa o to snažia.)

Aby sme vedeli povedať, koľko má náš strom vrcholov, spočítajme najskôr jeho hĺbku h. V hĺbke 0 sedí koreň a v ňom je číslo N. Ak je v nejakom vrchole číslo x, vo vrcholoch o jednu úroveň hlbšie je číslo $2x/3$. V hĺbke y je teda vo vrcholoch číslo $(2/3)^y N$. V listoch sú hodnoty menšie ako 3. Hĺbka stromu bude teda najmenšie také h, pre ktoré už je $(2/3)^h N < 3$. Z toho dostávame, že $h \sim \log_{3/2} N$.

No a teraz už vieme zistiť počet vrcholov. V hĺbke y je ich 3^y. Dokopy ich teda je $1 + 3 + \cdots + 3^h$. Toto je súčet geometrickej postupnosti, a zo známeho vzorca dostávame, že je rovný $(3^{h+1} - 1)/2$.

Časová zložitosť nášho algoritmu je teda $O((3^{h+1}-1)/2) = O(3^h)$. Teraz môžeme dosadiť za h náš odhad hĺbky a počítať ďalej:

$$3^h \sim 3^{\log_{3/2} N} = \left((3/2)^{\log_{3/2} 3}\right)^{\log_{3/2} N} = \left((3/2)^{\log_{3/2} N}\right)^{\log_{3/2} 3} = N^{\log_{3/2} 3}$$

Preto časová zložitosť nášho algoritmu je $O\left(N^{\log_{3/2} 3}\right)$. To môžeme približne vyčísliť ako $O\left(N^{2.71}\right)$.

Pre zaujímavosť, existuje všeobecné tvrdenie známe pod menom *master metóda*, pomocou ktorého sa dá ľahko a rýchlo určiť časová zložitosť mnohých rekurzívnych programov, vrátane toho nášho. Znenie tohoto tvrdenia nájdete napríklad na Wikipédii pod heslom `Master Theorem`.

V online verzii riešenia nájdete ešte nasledovné veci: podrobnejší pohľad na zložitosť algoritmu B (prečo sú v grafe zložitosti také „zuby“?) a veľmi dôležitý je aj bonus: dôkaz, **že algoritmus B naozaj triedi**. (Z jeho popisu to teda jasné nie je.)

Algoritmus A

Obľúbeným tvrdením medzi riešiteľmi bolo, že horný odhad na časovú zložitosť tohoto algoritmu neexistuje, prípadne, že je rovný nekonečnu, pretože (citujem) „náhoda môže byť sviňa a do nekonečna porovnávať tie isté prvky“.

Nuž, ak sa chceme hrať s technickými detailami, v uvedenej podobe citovaný výrok pravdu nemá. Mal by ju v takejto podobe: „náhoda môže byť sviňa a **ľubovoľne dlho** (nie **nekonečne dlho**!) porovnávať tie isté prvky“. Ale ani to nie je všetko. Finta je v tom, že pravdepodobnosť, že povedzme stokrát po sebe vyberieme tú istú dvojicu, je zanedbateľne malá – omnoho omnoho menšia ako napríklad pravdepodobnosť hardvérovej chyby počas výpočtu.

Preto sa u algoritmov, ktoré využívajú nejakým spôsobom náhodu, zvykne v prípade potreby používať pri určovaní časovej zložitosti trochu iný prístup. Napríklad zložitosť v najhoršom prípade sa dá udávať v podobe „s pravdepodobnosťou veľmi blízkou jednej spraví tento algoritmus na vstupe veľkosti N menej ako blabla krokov“.

Takéto niečo budeme chcieť povedať aj o algoritme A. Ale najskôr prebehneme cez nejaké témy, ktoré budeme na to potrebovať.

Základy pravdepodobnosti

Na tomto mieste bez dôkazu uvedieme niekoľko tvrdení, ktoré budeme používať. Ako to už býva zvykom, v online verzii nájdeš aj ich dôkazy a podrobné vysvetlenie.

Základný nástroj, ktorý budeme potrebovať: Majme pokus, ktorý občas dopadne dobre, občas zle. Budeme robiť tento pokus dokola, až kým jeden nedopadne dobre. Ak p je pravdepodobnosť, že pokus dopadne dobre, tak v priemere spravíme $z = 1/p$ pokusov.

Situácia je dokonca ešte lepšia: vieme zaručiť, že skoro určite bude potrebný počet pokusov lineárne úmerný z. Napríklad platí nasledujúci odhad: Pravdepodobnosť, že budeme musieť spraviť viac ako $100z$ pokusov, je menšia ako $1/(2^{50})$. A takáto pravdepodobnosť je už z praktického hľadiska rovná nule.

Inverzie

Cenným nástrojom pri analýze triedení sú inverzie. Inverzia v poli A je taká dvojica indexov (i, j), že $i < j$ a $A[i] > A[j]$. Inými slovami, je to dvojica prvkov A, ktoré sú „vymenené“.

Zjavne pole je utriedené práve vtedy, ak nemá žiadne inverzie. A pri mnohých triediacich algoritmoch platí, že čím je v poli viac inverzií, tým je „menej utriedené“. (Z toho napríklad vyplýva, že vstup $N, N-1, \ldots, 2, 1$ je často dobrým kontrapríkladom.)

Dá sa dokázať, že platí nasledovné tvrdenie: Nech (i, j) je inverzia. Potom ak vymeníme hodnoty na pozíciách i a j, celkový počet inverzií v poli sa určite zmenší.

Odhad zložitosti algoritmu A

Ukážeme, že priemerný čas, ktorý potrebuje algoritmus A na utriedenie ľubovoľného poľa dĺžky N, je nanajvýš priamo úmerný $N^2 \log N$.

K tomu sa dopracujeme nasledujúcou úvahou: Majme pole dĺžky N, v ktorom je M inverzií. Ako dlho bude trvať, kým náš algoritmus trafí jednu z nich? Pravdepodobnosť, že trafíme inverziu, je:

$$p(M) = \frac{M}{\binom{N}{2}} = O\left(M/N^2\right)$$

Preto očakávaný (t.j. priemerný) počet pokusov, kým trafíme inverziu, je $f(M) = 1/p(M) = O\left(N^2/M\right)$.

Vždy, keď sa nášmu algoritmu podarí trafiť a vymeniť inverziu, zmenší sa tým celkový počet inverzií. Na začiatku je inverzií najviac $\binom{N}{2}$. V najhoršom prípade sa ich počet bude zmenšovať vždy o 1, až kým neklesne na 0.

V takomto prípade bude priemerný čas behu nášho algoritmu rovný

$$\sum_{M=1}^{\binom{N}{2}} f(M) = O\left(N^2 \sum_{M=1}^{\binom{N}{2}} \frac{1}{M}\right) = O\left(N^2 \log N\right)$$

(Posledný krok je v online verzii vysvetlený podrobnejšie.)

Ukázali sme teda, že bez ohľadu na to, ako vyzerá vstupné pole (koľko inverzií obsahuje), priemerný čas behu nášho programu bude nanajvýš priamo úmerný $N^2 \log N$. Z nášho úvodu do pravdepodobnosti dokonca vyplýva silnejšie tvrdenie: takmer určite je čas behu nášho programu nanajvýš priamo úmerný $N^2 \log N$.

V online verzii riešenia dokazujeme navyše, že na vstupe: $2, 1, 4, 3, \ldots, N, N-1$ bude algoritmus A naozaj potrebovať čas priamo úmerný $N^2 \log N$, a teda tento odhad zložitosti je tesný.

Odhad zložitosti algoritmu C

Dá sa ukázať, že algoritmus C spraví aj v najhoršom prípade nanajvýš N kôl porovnávania. Z toho bude vyplývať, že bez ohľadu na to, či nám bude generátor náhodných čísel „priateľsky naklonený“ alebo nie, bude časová zložitosť vždy $O(N^2)$.

Všimnite si kontrast oproti algoritmu A. Tam sme vedeli povedať len toľko, že do nejakého času skončí „skoro určite“, tu vieme hornú hranicu povedať úplne naisto. Náhodné čísla sa dajú využívať rôzne. Podobne ako je to v algoritme A, môžeme ich použiť pri hľadaní nejakého objektu, podobne to funguje napríklad v algoritmoch na testovanie prvočíselnosti. A naopak, môžeme si nimi len „tak trochu prilepšiť“, ako napríklad v algoritme C, či randomizovanom QuickSorte. Takéto použitie náhodných čísel má väčšinou za cieľ vyhnúť sa možným zlým vstupom.

Intuitívne, algoritmus C robí niečo podobné ako BubbleSort. Rozdiel je len v tom, že neporovnáva po sebe idúce dvojice zaradom, ale v náhodnom poradí. Podobne ako u BubbleSortu to časom proste nejak do úspešného konca dotlačíme, či už nám bude náhoda priať,

alebo nie. Presný dôkaz uvádzame v online verzii riešenia, je však zložitejší ako by ste možno čakali.

Dolný odhad zložitosti je tentokrát ľahký. Podobne ako pri BubbleSorte bude zlým vstupom opačne utriedené pole: $N, N-1, \ldots, 1$.

Aby sme uvideli, prečo je tento vstup zlý, stačí si všimnúť inverzie. Na začiatku ich tam je $\binom{N}{2}$. No a každá výmena, ktorú náš algoritmus spraví, zmenší počet inverzií o 1. Keďže za jedno kolo spraví najviac $N-1$ výmen, bude potrebovať určite aspoň $N/2$ kôl, aby takýto vstup utriedil.

A keďže už vieme, že každý vstup utriedime v čase $O(N^2)$, sme hotoví. Podobne ako u algoritmu A, aj tu sa nám podarilo ukázať, že náš odhad bol tesný a naozaj existujú vstupy, kde potrebujeme čas kvadratický od dĺžky vstupu.

A to už je v tejto papierovej verzii riešenia naozaj všetko. Dobrú noc!

8. Ťažký život

opravoval Lukáš
(max. 15 bodov)

Tento príklad mal výhodu, že správne riešenie v polynomiálnom čase bolo viditeľné hneď na prvý pohľad. Stačilo vyskúšať všetky súvislé podpostupnosti a v každej zrátať mimimum a súčet. To sa dalo v čase $O(N^3)$ a prefíkanejšie aj v $O(N^2)$. Predstavme si, že sme vyrátali súčet a maximum postupnosti $x_i, x_{i+1}, \ldots, x_j$. Súčet označme s a minimum označme m. Potom súčet postupnosti $x_i, \ldots, x_j, x_{j+1}$ je $s + x_{j+1}$ a minimum je $\min\{m, x_{j+1}\}$. Čiže o jedna dlhšiu postupnosť vieme spracovať v konštantnom čase. V algoritme začneme vždy s postupnosťou dĺžky jedna a tú predlžujeme. Dokopy je postupností zhruba N^2, teda výsledná časová zložitosť je $O(N^2)$. Za takéto riešenie som dával 9 bodov a za pomalšie 7 bodov.

Teraz si ukážeme jednu vec, ktorú neskôr využijeme v dvoch rýchlejších riešeniach. Nech prvok x_i je minimum postupnosti. Keďže prvky v našej postupnosti sú kladné, maximálne ohodnotenie postupnosti dostaneme, keď ľavý a pravý okraj postupnosti obsahujúcej x_i sú najďalej, ako sa dá. To inak znamená, že prvky za ľavým a pravým okrajom sú už menšie ako x_i a všetky prvky vnútri postupnosti nie sú menšie ako x_i. Ku každému prvku x_i teda chceme nájsť najbližší prvok napravo a naľavo taký, že je od neho menší. Označme si $l(i)$ najbližší menší prvok vľavo od prvku x_i a $p(i)$ najbližší menší napravo od neho.

Predpokladajme, že už poznáme hodnoty $p(i)$ a $l(i)$. Potom ľahko nájdeme úsek s maximálnym ohodnotením. Stačí nájsť také i, že $x_i(x_{l(i)+1} + \cdots + x_{p(i)-1})$ je maximálne. Najprv si predpočítame hodnoty $s_i = x_1 + \cdots + x_i$. To vieme v lineárnom čase vďaka vzťahu $s_i = s_{i-1} + x_i$. Vďaka hodnotám s_i platí $x_i(x_{l(i)+1} + \cdots + x_{p(i)-1}) = x_i(s_{p(i)-1} - s_{l(i)})$, teda hodnotu nejakého úseku vieme vyhodnotiť v konštantnom čase. Celková zložitosť tohto vyhodnotenia je preto $O(N)$.

Teraz si ukážeme, ako hľadať hodnoty $l(i)$ a $p(i)$ v čase $O(N \log N)$ (to však ešte nie je vzorák). Asi najjednoduchšie riešenie navrhol Peter Hlavatý. Prvky $l(i)$ a $p(i)$ budeme hľadať pomocou intervalového stromu. Ukážeme si, ako hľadať prvky $l(i)$ – postup pre $p(i)$ je rovnaký, len ideme z opačnej strany. Budeme postupne do intervalového stromu pridávať prvky $x_1, x_2, \ldots, x_n$. V čase, keď vkladáme do stromu prvok x_i, už sme doňho vložili prvky $x_1, \ldots, x_{i-1}$. Náš intervalový strom si bude v každom vrchole pamätať mimimum v danom intervale. Chceme pomocou neho nájsť menší prvok ako x_i, ktorý je najviac vpravo. Začneme od koreňa. Ak pravý syn koreňa obsahuje prvok menší ako x_i, znamená to, že minimum v pravej polovici je menšie ako x_i, teda $l(i)$ je v pravej polovici stromu. Preto sa ďalej rekurzívne zavoláme na pravého syna. V opačnom prípade sa rekurzívne zavoláme na ľavého syna. Podobne postupujeme aj ďalej. Nakoniec prídeme až do listu a nájdeme tak hľadaný menší prvok. Zložitosť tejto operácie závisí od výšky stromu a tá je $\log N$.

Existuje však aj lepšie riešenie využívajúce zásobník. Zásobník si môžeme predstaviť ako vrece. Keď doňho vkladáme prvok, vložíme ho na vrch. Keď prvok vyberáme, zoberieme ten, čo je na vrchu. V našom zásobníku bude platiť invariant, že hodnoty v ňom uložené

budú vždy usporiadané podľa veľkosti od najmenšej po najväčšiu. Keď pridávame prvok x_i, vyhodíme zo zásobníka všetky prvky väčšie ako x_i a tento vložíme na vrch. Všimnime si, že invariant v zásobníku platí po vložení nového prvku. Ale načo nám to je? Prekvapivo predposledný prvok v zásobníku je prvok $l(i)$. Prečo? Všimnime si, aké prvky sme vkladali a vyhadzovali zo zásobníka potom, ako sme tam vložili $l(i)$. Museli to byť prvky väčšie ako $l(i)$, lebo keby sme vložili aj nejaké menšie, $l(i)$ by už v zásobníku nebol. A keby sme doňho vložili prvok rovnako veľký ako $l(i)$, tak by $l(i)$ už nebol určite predposledný. Ešte si dovolím jednu poznámku k implementácii – na začiatku do zásobníka dáme prvok -1. To bude dno zásobníka, ktoré nikdy nevyberieme, lebo na vstupe sú len kladné čísla. Ušetríme si tým kontrolovanie niektorých špeciálnych prípadov.

Teraz sa ešte presvedčíme, že riešenie pomocou zásobníka má lineárnu časovú zložitosť. Do zásobníka vložíme každý prvok práve raz. Pri mazaní sa nám môže stať, že vyhodíme veľa prvkov a teda zložitosť jedného mazania nebude konštantná. Musíme si však uvedomiť, že každý prvok mažeme zo zásobníka iba raz, teda časová zložitosť je naozaj lineárna. Pamäťová zložitosť je takisto lineárna, lebo používame jeden zásobník a nejaké pomocné polia.

Listing programu:

```
#include <vector>
#include <algorithm>
#include <numeric>
#include <utility>
#include <stack>
#include <cstdio>
using namespace std;

#define REP(i,n) for(__typeof(n) i=0; i<(n); ++i)

typedef pair<int, int> PI;
void spracuj(vector<int> a, vector<int> &b) {
    stack<PI> z;
    z.push(PI(-1, -1));
    REP(i,a.size()) {
        while (z.top().first>=a[i])
            z.pop();
        b[i]=z.top().second;
        z.push(PI(a[i], i));
    }
}
int main() {
    int n;
    scanf("%d", &n);
    vector<int> x(n);
    REP(i,n)
        scanf("%lld", &x[i]);
    vector<int> b[2];
    b[0]=vector<int>(n); b[1]=b[0];
    vector<int> s(n+1, 0);
    //zrtame prefixovŠ sumy
    partial_sum(x.begin(), x.end(), s.begin()+1);

    REP(k,2) {
        spracuj(x, b[k]);
        reverse(x.begin(), x.end());
    }
    reverse(b[1].begin(), b[1].end());
    //v b s teraz ÄžavŠ a pravŠ konce sekov s minimom x[i]
```

```
    int res=0;
    REP(i,n) {
        int z=b[0][i]+1, k=n-1-b[1][i];
        res=max(res, (s[k]-s[z])*x[i]);
    }
    printf("%d\n", res);
}
```

opravoval Mic
(max. 15 bodov)

9. Tyče z Toga

No tak milé deti, povieme si, ako bolo treba riešiť posledný, desiaty príklad. Napriek tomu, že príklad bol veľmi pekný, vzorovo ho nevyriešil nik. Dokonca môžem citovať z jedného rešenia „Lepšie ako $\log N$ to určite nepôjde". Ale pozrime sa na to poriadne. Pomocou $\log N$ mesiacov vieme zistiť farbu každej tyče (teda presnejšie rozdeliť tyče na tri kôpky, pričom v každej kôpke sú tyče rovnakej farby). Nám stačí určiť jednu tyč. Toto nám napovedá, že by to mohlo ísť rýchlejšie. A ako rýchlo to pôjde? $\log\log N$?$\sqrt{\log N}$?47?.

Tak sa na to pozrime. Čo je najlepšie spraviť v prvom kroku? Najskôr si tyče rozdelíme do dvojíc (ak je tyčí nepárny počet, tak nám jedna ostane a tú si odložíme) a tie podhodíme lamálkam. Ak nám lamálka povedala, že tyče sú rovnakej farby, tak si tie dve tyče dáme do jedného vrecúška a ak nám nejaká lamálka povedala, že sú rôznej farby, tak ich zahodíme. A môžeme ich zahodiť? Nech počet tyčí svetlotyrkisovej[6] farby je a, a počet tyčí ostatných farieb je b, c. Pred vyhodením platilo $a \geq b+c$ (všimnite si, že je tam aj rovné a to preto, lebo sme si možno odložili práve svetlotyrkisovú). Platí to aj po vyhodení? V najhoršom prípade sme s každou vyhodenou dvojicou vyhodili tyč svetlotyrkisovej farby. Nech sme vyhodili x tyčí svetlotyrkisovej farby. S každou takou tyčou, sme vyhodili jednu tyč inej farby, z čoho dostaneme nerovnosť $a - x \geq b + c - x \Leftrightarrow a \geq b + c$, čiže sme tým nič nepokazili.

Toto bol prvý mesiac.

Teraz už máme niekoľko vrecúšok, pričom v každom vrecúšku sú dve tyče rovnakej farby. Čiže ak zistíme farbu jednej tyče z vrecúška, tak vieme aj farbu druhej tyče. Takže vieme každé vrecúško porovnať dvakrát za mesiac. Postavme si vrecúška do jedného dlhočizného radu. Porovnajme každé dve susedné vrecúška. Ak sú rovnaké, tak vrecúška spojíme dohromady. Dostaneme nový rad, pričom každé vrecúško má aspoň dve tyče a susedné vrecúška majú rôzne farby.

Toto bol druhý mesiac.

Čo spravíme ďalej? Z každého vrecúška (okrem posledného a predposledného) vyberieme jednu tyč. Keď sme vybrali tyč z k-teho vrecúška, tak ju porovnáme s tyčou z $k+2$ vrecúška (samozrejme s inou ako s tou, ktorú sme vybrali z $k+2$ vrecúška). Poďme postupne ofarbovať vrecúška farbami $1, 2$ a 3. Prvé vrecúško ofarbíme farbou 1. Druhé vrecúško ofarbíme farbou 2 (keďže sú susedné, tak nemôžu mať rovnakú farbu). Keď už máme určené farby pre prvé dve vrecúška, tak budeme postupne určovať farby pre každé ďalšie vrecúško. Ako? Zoberme si nejaké vrecúško, napríklad k-te. Ak $k-1$ a $k-2$ vrecúško má určenú farbu, tak viem jednoznačne určiť aj farbu k-teho vrecúška. Určite musí mať inú farbu ako $k-1$ vrecúško. Ale keďže sme ho už porovnávali s $k-2$ vrecúškom v tomto poslednom kole, tak vieme, či má rovnakú farbu ako $k-2$ vrecúško, alebo rôznu. A podľa toho ho patrične zafarbíme. Takto postupne určíme farbu každému vrecúšku.

Jediné čo ešte treba, je spočítať počet tyčí vo vrecúškach s farbou 1, počet tyčí vo vrecúškach s farbou 2 a to isté s vrecúškami s farbou 3. Teraz sa už len stačí pozrieť, ktorej farby je najviac a z nej vybrať nejakú tyč a prehlásiť ju za svetlotyrkisovú. Ale pozor. Ešte sa

[6] svetlotyrkisových je nadpolovičná väčšina

môže stať, že dve farby budú mať rovnaký počet tyčí. Toto ale mohlo nastať iba vtedy, ak bol počet tyčí nepárny a posledná tyč (tá, ktorá nemala tú česť sa dostať do žiadneho vrecúška) bola svetlotyrkisová. Lebo iba vtedy môže nastať, že počet svetlotyrkisových tyčí je rovnaký ako súčet zvyšných tyčí. Skúste si poriadne rozmyslieť prečo `:-)`. No a teda, keď máme dve farby s rovnakým počtom tyčí, tak je jasné, že naša odložená tyč je svetlotyrkisová, lebo ináč by svetlotyrkisových nemohla byť nadpolovičná väčšina.

Tak, a toto bol posledný, teda tretí mesiac.

No včul, máme to hotové. Teraz si ešte povedzme zopár vecí k implementácií. Ľubovoľné výpočty k nejakému mesiacu sa dá triviálne implementovať v čase $O(N)$. Na čo však ešte netreba zabudnúť je ošetrenie okrajových prípadov, kedy sa nám môže podariť povedať ktorá tyč je svetlotyrkisová.

Tak a ako som bodoval. Vzorové riešenia nedostali 15 bodov, lebo žiadne neboli. Za $O(\log N)$ mesiacov som uďeľoval 11 bodov. Lineárne riešenia dostávali 7 bodov. Občas som strhol za chýbajúci kód, zlý popis a podobne.

Domáca úloha. Rozmyslite si, koľko mesiacov nám bude trvať, aby sme vedeli zatriediť tyče do troch kôpok, pričom v každej kôpke sú tyče rovnakej farby:-)

Listing programu:

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

int N;

struct vrecusko{
    int tyce[2];// staci si nam pamatat nejake dve tyce a pocet
    int pocet;
    int farba;
    vrecusko(int a,int b){
        tyce[0]=a;
        tyce[1]=b;
        pocet=2;
    }
};

vector<vrecusko> vrecuska;

void opytaj_sa(int a,int b){
    cout << "("<<a<<" a "<<b<<")";
}

bool test(int poc){//osetrenie specialnych pripadov
    if(poc==0||(poc==2&&vrecuska[0].pocet==vrecuska[1].pocet)){
        cout << "Tyc "<<N<<" je svetlotyrkisova"<<endl;
        return true;
    }
    if(poc==1){
        cout << "Tyc "<<vrecuska[0].tyce[0]<<" je svetlotyrkisova"<<endl;
        return true;
    }
    int maxx=0,suc=0,ktore=0;
    for(int i=0;i<poc;i++){
        maxx=max(maxx,vrecuska[i].pocet);
        if(maxx==vrecuska[i].pocet)ktore=i;
        suc+=vrecuska[i].pocet;
    }
```

```
    suc-=maxx;
    if(maxx>=suc){
        cout << "Tyc "<<vrecuska[ktore].tyce[0]<<" je svetlotyrkisova"<<endl;
        return true;
    }
    return false;
}

int main(){
    cin >>N;
    if(N<3)cout << "Tyc 1 je svetlotyrkisova"<<endl;
    for(int i=0;i<N/2;i++)
        opytaj_sa(i*2+1,i*2+2);
    cout<<endl;
    for(int i=0;i<N/2;i++){
        int res;
        cin >> res;
         if(res)vrecuska.push_back(*(new vrecusko(i*2+1,i*2+2)));
    }
    cout << endl;
    int poc=vrecuska.size();
    if(test(poc))return 0;

    //koniec prveho mesiaca

    for(int i=0;i<poc;i++)
        opytaj_sa(vrecuska[i].tyce[0],vrecuska[(i+1)%poc].tyce[1]);
    int ktory=0;
    for(int i=0;i<poc;i++){
        int res;
        cin >> res;
        if(res){
            if(i!=poc-1){
                vrecuska[ktory].pocet+=vrecuska[i].pocet;
                vrecuska[i].pocet=-1;
            }else{
                vrecuska[0].pocet+=vrecuska[ktory].pocet;
                vrecuska[ktory].pocet=-1;
            }
        }else{
            ktory=i+1;
        }
    }
    ktory=0;
    for(int i=0;i<poc;i++){
        if(vrecuska[i].pocet>0){
            vrecuska[ktory]=vrecuska[i];
            ktory++;
        }
    }
    poc=ktory;
    if(test(poc))return 0;

    //koniec druheho mesiaca

    vrecuska[0].farba=0;
    vrecuska[1].farba=1;
    for(int i=0;i<poc-2;i++)
        opytaj_sa(vrecuska[i].tyce[0],vrecuska[i+2].tyce[1]);
    int pocty[][2]={{0,0},{0,1},{0,0}};
```

```
    for(int i=2;i<poc;i++){
        int res;
        cin >> res;
        if(res)
            vrecuska[i].farba=vrecuska[i-2].farba;
        else
            vrecuska[i].farba=3-vrecuska[i-1].farba-vrecuska[i-2].farba;
        if(vrecuska[i].farba==2)pocty[2][1]=i;
    }
    for(int i=0;i<poc;i++)pocty[vrecuska[i].farba][0]+=vrecuska[i].pocet;
    if(pocty[0][0]<pocty[1][0]){
        swap(pocty[0][0],pocty[1][0]);
        swap(pocty[0][1],pocty[1][1]);
    }
    if(pocty[0][0]<pocty[2][0]){
        swap(pocty[0][0],pocty[2][0]);
        swap(pocty[0][1],pocty[2][1]);
    }
    if(pocty[1][0]<pocty[2][0]){
        swap(pocty[1][0],pocty[2][0]);
        swap(pocty[1][1],pocty[2][1]);
    }
    if(pocty[0][0]==pocty[1][0])cout << "Tyc "<< N << " je svetlotyrkisova" << endl;
        else cout <<"Tyc "<< vrecuska[pocty[0][1]].tyce[0] << "je svetlotyrkisova"
<<endl;
    return 0;
}
```

Výsledková listina po 1. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda	1	2	3	4	5	Σ
1	Fulla Peter	SPŠ Spišská Nová Ves	2	15	15	15	15	15	75
1	Ondrúška Peter	SPŠ Dubnica nad Váhom	3	15	15	15	15	15	75
3	Kostolányi Peter	Gym. Jura Hronca BA	3	15	15	12	15	13	70
4	Fačkovec Boris	Gym. P. de Coubertina Piešťany	4	15	15	11	13	12	66
5	Bačo Ladislav	Gym. Poštová Košice	1	15	15	10	15	9	64
6	Polák Lukáš	Gym. Jura Hronca BA	1	15	15	11	14	8	63
7	Fekiač Jozef	Gym. Grösslingová BA	2	15	15	8	14	6	58
7	Matejovičová Lenka	Gym. Grösslingová BA	3	13	13	11	14	7	58
9	Herencsár Albert	Gym. maď. Galanta	2	13	15	3	14	12	57
10	Kuzma Tomáš	Gym. Alejová Košice	2	15	15		11	15	56
11	Hozza Michal	Gym. Jura Hronca BA	2	15	15	11	14		55
11	Kušnier Juraj	Gym. Bánovce nad Bebravou	3	13	9	9	14	10	55
13	Konečný Jakub	Gym. Grösslingová BA	1	14	14	9	14		51
14	Vaňo Jakub	Gymnázium J.A. Raymana	2	14	12	11	13		50
15	Fecko Stanislav	Gym. Pankúchova Bratislava	4	11	15	11	4	8	49
15	Hojčka Michal	Gym. Partizánske	2	14	12	9	14		49
15	Kudláč Matúš	Gym. Bilíkova BA	3	10	15	9	15		49
18	Halabuk Milan	Gymnázium Levice	4	14	5	9	14	5	47
19	Matula Peter	Gym. Školská Turč. Teplice	3	7	15	10	5	8	45
19	Starovská Mária	Gym. Grösslingová BA	3	15	15		15		45
21	Boško Lukáš	Gymnázium Považská Bystrica	2	10	11	9	14		44
21	Košdy Ivan	Gym. Jura Hronca BA	2	14	15		15		44
21	Tureková Katarína	Gym. Tajovského B. Bystrica	3	8	12	9	15		44
24	Goga Rudolf	Gym. Jura Hronca BA	1	15	15		13		43
25	Balogh Tomáš	SPŠE Nové Zámky	2	10	11	9	9	3	42
26	Jenis Jakub	Gym. Cyrila a Metoda Nitra	3	15	13		13		41
26	Stachová Paula	Gym. Bilíkova BA	3	10	10	9	12		41
26	Štefanišin Peter	Gym. Svidník	4	13	15	7		6	41
29	Bachratý Martin	Gym. Veľká Okružná Žilina	2	6	11	9	14		40
29	Boška Michal	Gym. Jura Hronca BA	3	15	12		13		40
29	Hagara Michal	Gym. Jura Hronca BA	1	14	12	0	14		40
29	Štrbka Dávid	Gym. Grösslingová BA	3	14	15	0,5	3	8	40
33	Kučin Alexander	Gym. Lipany	3	9	12	8	7	1	37
33	Sabo Michal	Gym. Jura Hronca BA	3	14	11	3	5	4	37
35	Meravý Jan	SPŠ Dubnica nad Váhom	3	7	12	4	13		36
36	Dittrich Andrej	Gym. Jura Hronca BA	3	15	15		5		35
36	Hozzánková Hana	Gym. Jura Hronca BA	1	8	12	3	12		35
36	Kikta Michal	Gym. Tatarku Poprad	2	14	10	11			35
36	Uherčík Maroš	Gym. P. de Coubertina Piešťany	2		15	3	13	4	35
40	Behún Marek	Gym. Ľ. Štúra Michalovce	1	6	5	8	11	4	34
40	Mikláš Ján	Gym. P. de Coubertina Piešťany	2	3	11	3	13	4	34
42	Kohút Matej	Gym. Jura Hronca BA	2	11	14		7		32
42	Kvasnička Igor	Gym. Jura Hronca BA	3	11	8		13		32
44	Magdolen Matej	Gym. Golianova Nitra	1	1	12	9	3	6	31
45	Kalugerov Samuel	Gym. Jura Hronca BA	1		11	3	13		27
46	Stančiaková Katarína	Gym. Jura Hronca BA	1	3	6	3	13	1	26
47	Tatara Tomáš	Gym. Jura Hronca BA	3	10	12	2	1		25
48	Truben Martin	Gym. Jura Hronca BA	3	10	3	5	3	2	23
49	Kuchyňár Martin	Gym. Jura Hronca BA	1	11	7			4	22
49	Rigdová Emília	Gym. Popradské nábr. Poprad	1	12	10				22

	Meno a priezvisko	Škola	Trieda	1	2	3	4	5	Σ
51	Chrást Lukáš	Gym. Golianova Nitra	2	7	5	7			19
52	Šutý Karol	Gym. Jura Hronca BA	3	7	11				18
52	Táňa Tóthová	Gym. Jura Hronca BA	1	6			12		18
54	Šebeňová Petra	Gym. Jura Hronca BA	1	3	2		12		17
55	Chudý Lukáš	Gym. Liptovský Hrádok	2	7	6		3		16
55	Molčan Dávid	Gym. Slovenská Bardejov	4	6	10				16
55	Šulák Viktor	SPŠE Nové Zámky	2	7	9		0		16
58	Končok Jakub	Gym. Haličská Lučenec	3		15				15
58	Piter Miroslav	Gym. Svidník	1	15					15
60	Baxová Katarína	Gym. Ľudovíta Štúra Trenčín	2	14					14
61	Jančigová Jarmila	Gym. Jura Hronca BA	3	10			3		13
62	Korenek Roman	Gym. Jura Hronca BA	2	5	6				11
63	Buchovecká Simona	Gym. Medzilaborce	4	0	10				10
63	Husár Milan	Gym. Jura Hronca BA	1	7			3		10
65	Szabadosová Emília	Škola pre mim. nadané deti BA	3		6		3		9
66	Janočko Ján	Gym. Jura Hronca BA	2	4				3	7
66	Poláková ?	Gym. Jura Hronca BA	1			7			7
68	Balogh Imrich	Gym. Jura Hronca BA	1		0		2		2
68	Sevela Richard	Gym. Jura Hronca BA	1		0		2		2
70	Zdechovan Lukáš	Gym. Tajovského B. Bystrica	4				0		0

Výsledková listina po 1. kole kategórie KSP-O

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
1	Hapák Samuel	Gym. Grösslingová BA	3	15	15	15	15	14	74
2	Fulla Peter	SPŠ Spišská Nová Ves	2	15	15	15	15	10	70
3	Hlavatý Peter	Gym. Jura Hronca BA	3	15	15	15	15	8	68
3	Ondrúška Peter	SPŠ Dubnica nad Váhom	3	15	15	15	14	9	68
5	Boža Vladimír	Gym. Tatarku Poprad	3	15	15	15	15		60
6	Košinárová Alena	Gym. Grösslingová BA	3	15	15	15	9	5	59
7	Nagy Kristián	Gym. Jura Hronca BA	3	6	15	15	13	9	58
8	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	3	14	15	15	9	4	57
9	Kostolányi Peter	Gym. Jura Hronca BA	3	15	13	9	9	6	52
10	Varga András	Gym. H. Selyeho Komárno	3	15	12	8	7	7	49
11	Simanová Lucia	Gym. Grösslingová BA	3	15	15	12		2	44
11	Vojtko Jakub	Gym. Jura Hronca BA	2	15	14	15			44
13	Kekely Lukáš	Gym. Varšavská Žilina - Vlčince	3	15	6	14	7		42
14	Fedáková Dominika	Gym. Stará Ľubovňa	3	15	10	7	2		34
15	Kudláč Matúš	Gym. Bilíkova BA	3	15		6	9	3	33
16	Fačkovec Boris	Gym. P. de Coubertina Piešťany	4	13	12	7			32
17	Kočiský Tomáš	Gym. Grösslingová BA	3	15		15			30
18	Súkeník Ján	Gym. Lettricha Martin	3	14	6	5		3	28
19	Petrucha Michal	Gym. Metodova BA	2	15		12			27
19	Sucha Martin	Gym. Jura Hronca BA	3	15	12				27
21	Herencsár Albert	Gym. maď. Galanta	2	14	12				26
21	Kuzma Tomáš	Gym. Alejová Košice	2	11	15				26
23	Kukan Matúš	Gym. Grösslingová BA	3	15	10				25
24	Bačo Ladislav	Gym. Poštová Košice	1	15	9				24
24	Kušnier Juraj	Gym. Bánovce nad Bebravou	3	14	10				24
26	Fecko Stanislav	Gym. Pankúchova Bratislava	4	4	8	11			23
26	Kvasnička Igor	Gym. Jura Hronca BA	3	13		10			23
28	Miňo Igor	Gym. Sobrance	3	13	2	5		2	22
28	Polák Lukáš	Gym. Jura Hronca BA	1	14	8				22
28	Starovská Mária	Gym. Grösslingová BA	3	15		7			22
31	Hojčka Michal	Gym. Partizánske	2	14		7			21
31	Matejovičová Lenka	Gym. Grösslingová BA	3	14	7				21
33	Fekiač Jozef	Gym. Grösslingová BA	2	14	6				20
33	Kalugerov Samuel	Gym. Jura Hronca BA	1	13		7			20
33	Meravý Jan	SPŠ Dubnica nad Váhom	3	13		2		5	20
33	Stančiaková Katarína	Gym. Jura Hronca BA	1	13	1	5	1		20
37	Halabuk Milan	Gymnázium Levice	4	14	5				19
37	Stachová Paula	Gym. Bilíkova BA	3	12		7			19
39	Magyar Vladimír	SPŠE Nové Zámky	3	9	6	3			18
40	Mikláš Ján	Gym. P. de Coubertina Piešťany	2	13	4				17
40	Uherčík Maroš	Gym. P. de Coubertina Piešťany	2	13	4				17
42	Behún Marek	Gym. Ľ. Štúra Michalovce	1	11	4				15
42	Košdy Ivan	Gym. Jura Hronca BA	2	15					15
42	Rampášek Ladislav	Gym. Jura Hronca BA	4	15					15
42	Tureková Katarína	Gym. Tajovského B. Bystrica	3	15					15
46	Bachratý Martin	Gym. Veľká Okružná Žilina	2	14					14
46	Boško Lukáš	Gymnázium Považská Bystrica	2	14					14
46	Hagara Michal	Gym. Jura Hronca BA	1	14					14
46	Hozza Michal	Gym. Jura Hronca BA	2	14					14
46	Konečný Jakub	Gym. Grösslingová BA	1	14					14

	Meno a priezvisko	Škola	Trieda	4	5	6	7	8	Σ
51	Boška Michal	Gym. Jura Hronca BA	3	13					13
51	Goga Rudolf	Gym. Jura Hronca BA	1	13					13
51	Jenis Jakub	Gym. Cyrila a Metoda Nitra	3	13					13
51	Matula Peter	Gym. Školská Turč. Teplice	3	5	8				13
51	Vaňo Jakub	Gymnázium J.A. Raymana	2	13					13
56	Balogh Tomáš	SPŠE Nové Zámky	2	9	3				12
56	Hozzánková Hana	Gym. Jura Hronca BA	1	12					12
56	Šebeňová Petra	Gym. Jura Hronca BA	1	12					12
56	Táňa Tóthová	Gym. Jura Hronca BA	1	12					12
60	Končok Jakub	Gym. Haličská Lučenec	3			11			11
60	Štrbka Dávid	Gym. Grösslingová BA	3	3	8				11
62	Chudý Lukáš	Gym. Liptovský Hrádok	2	3		7			10
62	Jančigová Jarmila	Gym. Jura Hronca BA	3	3		7			10
62	Sabo Michal	Gym. Jura Hronca BA	3	5	4	1			10
65	Magdolen Matej	Gym. Golianova Nitra	1	3	6				9
66	Kučin Alexander	Gym. Lipany	3	7	1				8
67	Kohút Matej	Gym. Jura Hronca BA	2	7					7
68	Štefanišin Peter	Gym. Svidník	4		6				6
69	Dittrich Andrej	Gym. Jura Hronca BA	3	5					5
69	Truben Martin	Gym. Jura Hronca BA	3	3	2				5
71	Kuchyňár Martin	Gym. Jura Hronca BA	1		4				4
72	Husár Milan	Gym. Jura Hronca BA	1	3					3
72	Janočko Ján	Gym. Jura Hronca BA	2		3				3
72	Sevela Richard	Gym. Jura Hronca BA	1	2		1			3
72	Szabadosová Emília	Škola pre mim. nadané deti BA	3	3					3
76	Balogh Imrich	Gym. Jura Hronca BA	1	2					2
77	Tatara Tomáš	Gym. Jura Hronca BA	3	1					1
78	Šulák Viktor	SPŠE Nové Zámky	2	0					0
78	Zdechovan Lukáš	Gym. Tajovského B. Bystrica	4	0					0

Výsledková listina po 1. kole kategórie KSP-T

	Meno a priezvisko	Škola	Trieda	8	9	10	Σ
1	Hapák Samuel	Gym. Grösslingová BA	3	14	9	11	34
2	Hlavatý Peter	Gym. Jura Hronca BA	3	8	13	11	32
3	Ondrúška Peter	SPŠ Dubnica nad Váhom	3	9	9	10	28
4	Kostolányi Peter	Gym. Jura Hronca BA	3	6	11	7	24
5	Boža Vladimír	Gym. Tatarku Poprad	3		12	11	23
6	Kudláč Matúš	Gym. Bilíkova BA	3	3	9	7	19
7	Fačkovec Boris	Gym. P. de Coubertina Piešťany	4		7	10	17
8	Fulla Peter	SPŠ Spišská Nová Ves	2	10			10
9	Kvasnička Igor	Gym. Jura Hronca BA	3		9		9
9	Nagy Kristián	Gym. Jura Hronca BA	3	9			9
11	Šutý Karol	Gym. Jura Hronca BA	3		7		7
11	Varga András	Gym. H. Selyeho Komárno	3	7			7
13	Košinárová Alena	Gym. Grösslingová BA	3	5			5
13	Meravý Jan	SPŠ Dubnica nad Váhom	3	5			5
15	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	3	4			4
16	Súkeník Ján	Gym. Lettricha Martin	3	3			3
17	Miňo Igor	Gym. Sobrance	3	2			2
17	Simanová Lucia	Gym. Grösslingová BA	3	2			2
19	Tatara Tomáš	Gym. Jura Hronca BA	3		1		1
20	Szabadosová Emília	Škola pre mim. nadané deti BA	3			0	0