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 2. kola letnej časti

Práve sa vám dostali do rúk posledné tohtoročné vzorové riešenia KSP. Veríme, že s nimi naložíte tak, ako by ste mali, lebo múdry sa z chýb poučí. Takže si snáď nájdete chvíľku času a popri tých krásnych plážach, čistých horách, očarujúcich ženách a svalnatých chlapoch budete venovať pozornosť aj im. A keď už sme pri tých letných radovánkach, je na mieste vyhlásiť našu letnú skorosúťaž o najexotickejšiu fotografiu so vzorovými riešeniami. Vaše najvydarenejšie zábery posielajte na **mino@foto.ksp.sk**. Zábery ohodnotí nezávislá komisia a najlepší dostane čokoládu.

Emocionálna väzba indivídua na oblasť prírodných vied je vždy
ostro menšia ako vzájomná väzba dvoch indivíduí opačnej polarity.

KSPáci

1. Zradné schody

opravoval Maty
(max. 15 bodov)

Tento príklad bol veľmi ľahký a rýchlo sa dal nájsť všeobecný postup ako hľadanú postupnosť generovať. Body som vám strhával za chýbajúce odhady časovej a pamäťovej zložitosti, prípadne za horšiu pamäťovú zložitosť, teda ak ste použili nejaké pole veľkosti n. A teraz k príkladu. Začnime tým, že musíme skočiť skok vždy rôznej dĺžky. Preto musíme skočiť aj skok dĺžky $n-1$. Ten môžme skočiť iba medzi schodmi $1, n$ a teda určite bude naša postupnosť obsahovať takýto skok. Povedzme si, že spravíme tento skok ako prvý a začneme na schode 1. Zoberme si teraz skok dĺžky $n-2$. Ten nemôžeme spraviť medzi schodmi 1,$n-1$ lebo na schode 1 sme začali a teda už sa tam vrátiť nemôžeme. Teda pre skok dĺžky $n-2$ nám ostáva jediná možnosť – spraviť tento skok medzi schodmi 2 a n. Teda zo schodu n, na ktorom stojíme po prvom skoku, skočíme na schod 2. Pre skok dĺžky $n-3$ dostaneme, že zo schodu 2 musíme skočiť na schod $n-1$.

Týmto postupom získame jednu konkrétnu permutáciu, o ktorej teraz ukážeme, že vyhovuje podmienkam zadania. Hľadaná permutácia vyzerá takto: Začnem na schode 1, skočím na schod n, z neho na schod 2... . Vždy teda skočím na posledný ešte nepoužitý a prvý ešte nepoužitý schod. Zjavne neskočím na žiadny schod dvakrát, lebo vždy skáčem iba na nepoužité schody. A takisto zjavne spravím $n-1$ skokov, a teda skočím na každý schod. Náš program teda vypíše ako výsledok jednu konkrétnu postupnosť $1, n, 2, n-1, 3, n-2, \cdots$ Posledný člen tejto postupnosti bude $\lfloor \frac{n}{2} \rfloor + 1$.

Za domácu úlohu si skúste rozmyslieť, koľko iných postupností vyhovuje zadaniu.

Listing programu:

```
var n,i:longint;
begin
  readln(n);
  for i:=1 to n div 2 do write(i,' ',n+1-i,' ');
```

```
   if n mod 2=1 then write(n div 2+1);
   writeln;
end.
```

2. Zlaté staré časy

opravovala Monika
(max. 15 bodov)

Riešení od vás prišla celá spústa, niektoré neboli vzorové, iné neboli ani len správne, ale aspoň bolo čo opravovať.

Na začiatok si povieme ako som bodovala. Plný počet bodov ste mohli získať za riešenie, ktoré funguje v časovej aj pamäťovej zložitosti $O(N+M)$. 14 bodmi som ohodnotila tých čo mali časovú zložitosť $O(N^2)$ (síce používali rad, tak ako vo vzorovom riešení, ale susedstvá medzi ľudmi si uložili tak nešťastne, že im to zhoršovalo zložitosť). Veľké množstvo riešení malo časovú zložitosť $O(N(N+M))$, čo som ohodnotila 12 bodmi. Horšiu zložitosť mali už len riešenia v $O(N^3)$ s 10 bodmi a $O(N^2M)$ s 9.

Bodíky ste mohli stratiť za chýbajúci alebo nedostatočný popis (1-4 body). Pokiaľ bola použitá v kóde nejaká dátová štruktúra z STL (alebo podobnej knižnice) a nevysvetlili ste dostatočne ako funguje, strhla som 2 body. Riešenia, v ktorých ste zabudli uvažovať prípad, keď sa správa nevie dostať ku všetkým ľuďom, stratili po 2 body. Nefunkčné riešenia mohli získať najviac 5 bodov.

No a teraz poďme na riešenie. Správa sa šíri medzi klebetníkmi od človeka s číslom 1. V ďalšom kole sa ju dozvedia jeho susedia, v ďalšom susedia susedov a tak ďalej. Toto šírenie si môžeme predstaviť ako vlnu, ktorá ide od človeka s číslom 1. Keď povie klebetu svojím susedom, tak sa vlna so správou dostane do vzdialenosti jedna. Potom susedia človeka 1 povedia susedom klebetu a vlna sa posunie do vzdialenosti dva. Takto to pokračuje ďalej. Našou úlohou je zistiť, do akej vzdialenosti sa takáto vlna informácie dostane najďalej, kým sa ju všetci nedozvedia.

Priamo šírenie vlny, tak ako bolo opísané v zadaní, je proces, ktorý prebieha naraz (cudzím slovom tomu hovoríme *paralelne*). V každom kole sa klebeta šíri na veľa miestach naraz. Skúsime toto šírenie správ simulovať postupne (cudzím slovom *sekvenčne*). Cieľ bude, aby všetci ľudia, ktorí práve tvoria vlnu, postupne jeden po druhom povedali klebetu ďalej. A až potom ako posledný z nich doklebetí, možu začať klebetiť ľudia v nasledujúcej vlne.

Toto môžeme realizovať tak, že ľudia, ktorí chcú klebetiť, sa budú stavať do radu. Keď človek príde na rad, tak povie klebetu všetkým susedom, ktorí ju ešte nevedia, a odíde spokojne domov. Každý sused, ktorý sa práve dozvedel klebetu, ju chce vyklebetiť ďalej, a teda sa všetci títo susedia v nejakom ľubovoľnom poradí postavia na koniec radu.

Na začiatku sa pochopiteľne do radu postaví ako jediný človek s číslom 1, pretože on chce začať klebetiť. Všimnite si, že keď človek s číslom 1 doklebetí, budú za ním v rade stáť všetci jeho susedia. Potom sa postupne dostanú títo susedia na rad a klebetia správu všetkým svojim susedom, čo správu ešte nedostali. Tí sa hneď nato postavia na koniec radu. Keď takto odklebetia ľudia z prvej vlny (teda susedia človeka s číslom 1), v rade zostanú práve všetci susedia z druhej vlny (teda susedia susedov človeka s číslom 1, ktorí sa až teraz dozvedeli klebetu). Takto môžeme pokračovať ďalej.

Tento opísaný postup môžeme ľahko implementovať. Budeme používať dátovú štruktúru rad, niekde nazývanú aj fronta alebo front.[1] Rad si môžeme predstaviť ako dlhé pole, ktoré podporuje len dve operácie: vloženie prvku na koniec a výber prvku zo začiatku. Vďaka

[1] Pomenovanie tejto dátovej štruktúry je zdrojom nekonečných bojov podobných bojom o najlepší operačný systém, editor, príchuť Deli tyčiniek či stranu, z ktorej otvárať banán. Najčastejší argument jednej strany je, že slovo fronta je české a v slovenčine je nespisovné. Druhá strana oponuje tým, že v prípade slova „fronta“ nejde o nespisovny pojem ale o zaužívaný technický pojem, tzv. terminus technicus.

týmto dvom operáciám má rad vlastnosť, že prvok, ktorý z neho vyberieme (vždy iba zo začiatku) bol do radu určite vložený skôr ako prvky, ktoré sú ešte v rade. A to je presne to, čo potrebujeme.

Rad sa dá implementovať ako pole s dvoma ukazovateľmi – na začiatok a koniec. Ukazovateľ na začiatok pri výbere prvkov zo začiatku zvyšujeme, ukazovateľ na koniec pri pridávaní prvkov tiež zvyšujeme. (Pri takomto postupe musí byť samozrejme pole dosť veľké. Ak nevieme, koľko prvkov budeme mať, je lepšie použiť spájaný zoznam[2].)

Okrem radu si potrebujeme ešte pamätať, či človek už vie klebetu (a prípadne aj, že kedy sa ju dozvedel, kedže sa to dá implementovať v jedinom poli rozlišovaním kladných hodnôt pre časy a hodnoty -1 ak klebetu nevie). Teraz stačí vložiť na začiatok radu človeka s číslom 1. A potom už len opakujeme dokola to isté, až kým sa rad nevyprázdni: Vyberieme z radu človeka, pozrieme všetkých susedov, vyberieme tých, čo ešte nevedia správu, nastavíme im kolo, kedy sa ju dozvedeli a zaradíme ich na koniec radu.

Problém, na ktorý väčšina z vás zabudla, je, že v zadaní nebolo zaručené, že klebeta sa určite vždy rozšíri do celého spoločenstva ľudí. Môže sa stať, že medzi človekom s číslom 1 a nejakým iným človekom neexistuje žiadne spojenie cez iných ľudí a klebetu sa nedozvie. To treba na záver ošetriť jedným prechodom poľa, kde sú informácie, či človek sa dozvedel klebetu. Pri tomto prechode sa dá nájsť aj maximálne kolo, v ktorom sa človek dozvedel správu. Iná možnosť je si počas výpočtu počítať kolá, resp. vypísať kolo, v ktorom sa dozvedel klebetu posledný človek, ktorý bol v rade.[3]

Čo sa týka formy, v ktorej si udržiavame vzťahy ľudí, tu si treba dávať pozor. Ak ste si nevhodne zvolili tento spôsob, mohlo vám to pokaziť časovú zložitosť. V tomto prípade je najvhodnejšie na ukladanie vzťahov použiť zoznam vzťahov. Ten sa dá prakticky implementovať tak, že máme pole veľkosti N, kde si pamätáme počet vzťahov pre človeka s číslom $1 \leq i \leq N$. Potom si ešte ku každému človeku pamätáme v poli zoznam ľudí, ktorí sú jeho susedia. Takto vieme ľahko ku každému človeku rýchlo prejsť celý zoznam susedov.

Pri načítaní potrebujeme načítať M dvojíc, čo trvá čas $O(M)$. Potom postupne pridávame ľudí z radu a zase ich z neho odoberáme. Každý človek sa do radu dostane najviac raz a každé susedstvo je dvakrát uvažované, keď sa rozhoduje, že či sused už klebetu vie alebo nie (raz v každom smere). Preto celé pridávanie, odoberanie a spracúvanie ľudí bude trvať čas $O(N + M)$. Celkovo je teda časová zložitosť nášho algoritmu $O(N + M)$ – lineárna od veľkosti vstupu.

Takisto pamäťová zložitosť je lineárna od veľkosti vstupu, teda $O(N + M)$.

A teraz už neváhajte a rýchlo si pozrite kód. Predtým ako začnete ešte chcem upozorniť, že v kóde sa alokuje viac pamäte ako je reálne treba (na zoznam vzťahov používam pole rozmerov $N \times N$). Pamäť je síce takto alokovaná ale nikdy sa nepoužije. Je to urobené pre prehľadnosť kódu, aby v ňom zbytočne nezavadzal kód, ktorý sa stará o alokáciu a prácu s pamäťou, ktorú by bolo treba dynamicky vytvárať (či už priamo alokáciou alebo cez spájaný zoznam).

Listing programu:

```
program KSP_2;
const MAX_N = 100;
var ludia: array[1..MAX_N,1..MAX_N] of longint;
    pocet, cas, rad: array[1..MAX_N] of longint;

    zac,kon,i,clov,poc:longint;
    N,M,a,b:longint;
```

[2] http://en.wikipedia.org/wiki/Linked_list

[3] Rozmyslite si, že prečo.

```
begin
  read(N,M);
  for i:=1 to N do begin
    cas[i]:= -1;
    pocet[i]:= 0;
  end;

  for i:=1 to M do begin
    read(a,b);
    inc(pocet[a]);  ludia[a][pocet[a]] := b;
    inc(pocet[b]);  ludia[b][pocet[b]] := a;
  end;

  zac:=1;
  rad[1]:=1;
  cas[1]:=0;
  kon:=2;
  while zac < kon do begin
    clov := rad[zac];
    inc(zac);
    for i:=1 to pocet[clov] do
      if cas[ludia[clov,i]] = -1 then begin
        rad[kon] := ludia[clov,i];
        inc(kon);
        cas[ludia[clov,i]] := cas[clov] + 1;
        poc:=cas[clov]+1;
      end;
  end;

  {overenie, ze ci sme vsetkych ludi pozreli}
  for i:=1 to N do
    if cas[i] = -1 then begin
      writeln('Existuje clovek, ktory nedostane klebetu.');
      exit;
    end;
  writeln('Sprava sa dostane ku vsetkym za ',poc,' kol.');
end.
```

opravoval Pershing
(max. 15 bodov)

3. Zvrhlé pomenovanie

Riešenia, ktoré došli sa dajú rozdeliť do 4 kategórií. Prvou sú vzorové riešenia v lineárnom čase, ktoré dostali 15 bodov. Druhou sú riešenia, ktoré použili triedenie so zložitosťou $O(n \log n)$ a dostali za to 12 bodov. Tretiu kategóriu tvoria riešenia s kvadratickou zložitosťou, za ktorú bolo 9 bodov. A poslednú kategóriu uzatvárajú nefunkčné riešenia, ktoré dostali maximálne 5 bodov.

Hlavnou myšlienkou vzorového riešenia je, že čím viac čísel na prvých (najľavejších) miestach necháme nedotknutých, tým menší bude výsledok. Aby sme teda našli najmenšiu vhodnú permutáciu, chceme nájsť poslednú (najpravejšiu) pozíciu c takú, že napravo od nej je vo vstupnej permutácii nejaké číslo, ktoré je od c-teho čísla väčšie. Matematicky zapísané, musí existovať také d, pre ktoré platí $d > c$ a $a_d > a_c$.

Predstavme si, že sme už c našli a pozrime sa na postupnosť čísel od neho vpravo. Vieme, že pre žiadne z nich už naša vlastnosť neplatí. Inými slovami, ak e je ľubovoľné z týchto čísel, tak všetky čísla napravo od e sú od neho menšie. Z toho dostávame, že postupnosť napravo od pozície c (pozor, nie vrátane c!) musí byť utriedená zostupne.

Toto využijeme na nájdenie čísla c. Začneme na pozícií n a budeme smerom doľava hľadať najdlhší rastúci úsek. Keď sa nám rast pokazí, máme hľadané číslo c. Teda na nájdenie čísla c nám stačí lineárny čas.

Zjavne nevieme zostrojiť väčšiu permutáciu tak, aby sme nechali nedotknutých prvých c miest – zvyšok je už zostupne usporiadaný a teda ho nevieme zväčšiť. Ukážeme, že vieme zostrojiť väčšiu permutáciu tak, aby sme nechali nedotknutých prvých $c-1$ miest.

Pozrime sa na číslo na pozícii c. Aby sme dostali väčšiu permutáciu, musíme ho nutne vymeniť za nejaké väčšie, A tu zlyháva vätčšina chybných riešení, ktoré vymieňajú číslo na pozícii c s číslom na pozícii $c+1$, teda s najväčším z ostatných. Problém je v tom, že permutácia, ktorú takto dostaneme, bude síce väčšia, ale nebude najbližšia.

Číslo, ktoré chceme vymeniť s číslom na pozícii c, musí byť čo najmenšie. A zároveň musí samozrejme byť väčšie ako to, ktoré tam je teraz. Aby sme našli najlepšieho kandidáta na výmenu, lineárne prejdeme všetky čísla vpravo od pozície c a nájdeme index d najmenšieho z nich, ktoré je väčšie od čísla na pozícii c.

Teraz už môžeme vymeniť čísla na pozíciách c a d. Posledná vec, ktorú musíme spraviť, je usporiadať čísla napravo od pozície c do vzostupného poradia. Na to môžeme použiť nejaký triediaci algoritmus, ide to však aj ľahšie. Stačí si uvedomiť, že práve spravená výmena nám nepokazí zostupné usporiadanie čísel napravo od c. Teda aby sme túto postupnosť utriedili vzostupne, stačí ju obrátiť, čo vieme ľahko spraviť v lineárnom čase.

Presnejšie by sa dalo dokonca povedať, že nami nájdené riešenie je lineárne od počtu pozícii, ktoré sa zmenia.

Listing programu:

```
var n,i,j,c,d : longint;
    a : array[0..1000] of longint;

procedure swap(var a,b:longint);
var pom:longint;
begin
    pom:=a; a:=b; b:=pom;
end;

begin
    readln(n);
    a[0]:=-1;
    for i:=1 to n do read(a[i]);
    c:=n;
    while (a[c-1]>a[c]) and (c>0) do dec(c);
    dec(c);
    if c=0 then begin writeln('nejde spravit'); exit; end;
    d:=c+1;
    while (a[d+1]>a[c]) and (d<n) do inc(d);
    swap(a[c],a[d]);
    j:=n; i:=c+1;
    repeat
        swap(a[i],a[j]); inc(i); dec(j);
    until i>=j;
    for i:=1 to n do write(a[i],' '); writeln;
end.
```

4. Zemiaky mám...

opravoval Mišo
(max. 15 bodov)

Trik v tomto príklade spočíval v tom, že riešenie, ktoré možno na prvý pohľad vyzerá optimálne, nemusí vyzerať optimálne na druhý pohľad. Také riešenie v čase $O(N^2)$, ktoré zisťovalo pre každý zemiak najväčší možný zemiak, ktorý s ním môže ísť do hrnca, bolo najviac za 10 bodov. Ten najväčší možný zemiak sa totiž nemusí hľadať prezeraním všetkých zemiakov. Zemiaky sú usporiadané podľa veľkosti, preto hľadaný zemiak vieme nájsť binárnym vyhladávaním v čase $O(\log N)$. Takéto riešenie malo celkovú časovú zložitosť $O(N \log N)$ a mohlo získať najviac 12 bodov.

Pointa vzorového riešenia spočívala v tom, že načo prehľadávať celé pole vždy znovu, ak sme ho už prehľadali pri predchádzajúcom zemiaku? Informáciu, ktorú sme vtedy zistili, môžeme využiť aj pri ďalších zemiakoch. Napríklad majme zemiak a a k nemu sme zistili, že najväčší zemiak, ktorý k nemu môže ísť do hrnca, je b. Ak teraz vezmeme iný zemiak c väčší ako a, tak vieme povedať, že k nemu maximálny zemiak určite nebude menší ako b.

Budeme teda postupne spracúvať zemiaky od najmenšieho po najväčší, a ku každému z nich postupne spočítame najväčší zemiak, ktorý s ním môže ísť do hrnca. Pritom využijeme vyššie uvedené pozorovanie, a teda vždy, keď hľadáme ďalší maximálny zemiak, začíname hľadať od maximálneho zemiaka k predchádzajúcemu spracovanému zemiaku.

Celý postup si môžete predstaviť takto: zemiaky sú body na číselnej osi. Budeme si na ňu ukazovať dvoma prstami: ľavým na zemiak, ktorý práve spracúvame, a pravým na najväčší zemiak, ktorý s ním môže ísť do hrnca. Spracovanie nového zemiaka teda vyzerá nasledovne: Posunieme o 1 doprava ľavý prst, aby ukazoval na nový zemiak, a potom posúvame doprava pravý prst, kým môžeme. V programe si teda stačí pamätať dva indexy, ktoré budú predstavovať aktuálnu pozíciu prstov.

Všimnime si, že každý prst sa posúva zľava doprava a na každý zemiak príde práve raz. Preto celková časová zložitosť je určite lineárna od počtu zemiakov, teda $O(N)$.

V zdrojáku n je počet zemiakov, d je varný index, f je prvý pointer, l druhý pointer, max je maximálny počet zemiakov, ktoré môžu ísť spolu do hrnca, a pre túto situáciu sa uložia pointre f, l do $maxf$, $maxl$.

Listing programu:

```
var i,n,d,f,l,max,maxf,maxl : longint;
    zemiak : array[1..1000] of longint;
begin
    readln(n,d);
    for i:=1 to n do readln(zemiak[i]);
    f:=1; l:=1; max:=0;
    while f<n do begin
        while (zemiak[l+1]-zemiak[f] < d) and (l < n-1) do inc(l);
        if (l-f > max) then begin max:=l-f; maxf:=f; maxl:=l; end;
        inc(f);
    end;
    for i:=maxf to maxl do write(zemiak[i],' ');
    writeln;
end.
```

5. ZO

opravoval kuko
(max. 15 bodov)

Najskôr upozornenie pre súťažiacich: Tí čo chcú body za príklad, musia mať na liahni vyplnené meno. Iná možnosť je poslať spolu s riešeniami série aj papier, na ktorom je napísaný

váš nick v liahni. V opačnom prípade vás nedokážeme identifikovať a nedostanete body. Zároveň sa ospravedlňujeme, že toto upozornenie nebolo už v zadaní.

A teraz už k úlohe. Quicksort vo všeobecnosti funguje tak, že vyberieme jeden prvok a podľa neho rozdelíme všetky prvky na dve kôpky: Na jednu kôpku dáme menšie prvky, na druhú väčšie. Potom stačí rovnakým spôsobom zotriediť obe časti zvlášť. My sme mali danú konkrétnu jednu implementáciu quicksortu (nie práve najlepšiu) a mali sme nájsť čo najhorší vstup – t.j. taký, aby quicksort spravil čo najviac porovnaní.

Náhodná postupnosť: Ako v takomto prípade postupovať? Jedna možnosť bola vygenerovať náhodnú postupnosť, prípadne viacero náhodných postupností a z nich vybrať tú najhoršiu, alebo zobrať náhodnú a nejakým náhodným spôsobom sa ju snažiť vylepšovať. Toto nie je veľmi dobrá cesta, pretože quicksort má veľmi dobrú *priemernú* zložitosť. Na náhodnom vstupe teda quicksort funguje veľmi dobre. Takýmto spôsobom sa dalo dosiahnuť okolo 100 000 porovnaní a dostať do 5 bodov.

Metóda zmrda: Lepší postup je zahrať sa na „zmrda“. Vstup nebudeme mať hotový, ale spustíme quicksort a počas toho, ako triedi, sa budeme pozerať, čo robí a budeme vymýšľať jednotlivé hodnoty tak, aby sme mu pri triedení čo najmenej pomohli. Tento postup si ukážeme na jednoduchšom príklade.

Predstavme si najskôr, že by v zadaní bol quicksort, ktorý ako pivota berie najľavejší prvok. Toto je odstrašujúci príklad, pretože utriedená postupnosť (ktorá by sa mala triediť ľahko) je v skutočnosti najhorší prípad. Keďže najľavejší prvok je najmenší, pole sa nám rozdelí iba na ľavý prvok a zvyšok poľa. Napríklad 5 prvkov sa bude triediť takto:

$$(\underline{1}, 2, 3, 4, 5)$$
$$1, (\underline{2}, 3, 4, 5)$$
$$1, 2, (\underline{3}, 4, 5)$$
$$1, 2, 3, (\underline{4}, 5)$$
$$1, 2, 3, 4, (5)$$

kde podčiarknutý prvok je pivot a uzátvorkované postupnosti sú tie, ktoré triedime.

Preto sa nikdy neberie ako pivot ľavý prvok. Celkom dobrá vec je zobrať ako pivota stredný prvok. Často sa vo vzorákoch KSP objavuje asi takáto implementácia quicksortu:

Listing programu:

```
procedure qsort (l, r : integer);
var i, j, p, t : integer;
begin
  if (l >= r) then exit;
  i := l; j := r;
  p := a[(l+r) div 2];
  repeat
    while a[i] < p do inc (i);
    while a[j] > p do dec (j);
    if i <= j then begin { vymen a[i] a a[j] }
      t := a[i]; a[i] := a[j]; a[j] := t;
      inc (i); dec (j);
    end;
  until i > j;
  qsort (l, j); qsort (i, r);
end;
```

Priemerný prípad takejto implementácie je, rovnako ako pri výbere najľavejšieho prvku(!), $O(n \log n)$. Najhorší prípad je, rovnako ako pri výbere najľavejšieho prvku. Teda kvadratická časová zložitosť,$O(n^2)$. Jediný rozdiel je, že pri takejto implementácií je *ťažšie vymyslieť/zostrojiť najhorší prípad.*

Ako ho však vieme zostrojiť? Ukážme si postup pre $n = 8$. Budeme triediť postupnosť $(a, b, c, \underline{d}, e, f, g, h)$. Ako bude postupovať quicksort uvedený vyššie? Vyberie pivota stredný prvok, to je d. A podľa neho rozdelí pole na menšie a väčšie. Buďme zmrdi a zvoľme $d = 1$ a všetky ostatné premenné väčšie ako 1 (zatiaľ bližšie neurčíme). Takto sa postupnosť rozdelí na dve časti: samotné $d = 1$ a zvyšok: $(b, c, a, \underline{e}, f, g, h)$. Ľavá časť je už utriedená, pravú bude quicksort triediť nasledovne: Pivot bude stredný prvok, t.j. e. Ako dobrí zmrdi zvolíme $e = 2$ a všetky ostatné prvky väčšie ako 2. Takto sa pole rozdelí na e a (c, a, b, f, g, h). Rovnako budeme postupovať ďalej: Nový pivot pravej časti bude b, zvolíme $b = 3$, ostáva utriediť $(a, c, \underline{f}, g, h)$, zvolíme $f = 4$, ostáva $(c, \underline{a}, g, h)$, zvolíme $a = 5$, triedime $(c, \underline{g}, h)$, takže bude $g = 6$, $c = 7$ a $h = 8$.

Výsledok: dostali sme postupnosť $(a, b, c, d, e, f, g, h) = (5, 3, 7, 1, 2, 4, 6, 8)$. Vždy sme zvolili pivota ako najmenšie číslo, ktoré ešte nebolo a simulovali sme, čo bude robiť quicksort. Všimnite si, že sme nepotrebovali vedieť presné hodnoty čísel, ktoré ešte nepoznáme. Vedeli sme, že všetky neurčené hodnoty sú väčšie ako všetky tie, ktoré ešte určené nie sú (to stačí). Poďme si ešte odsimulovať triedenie výslednej postupnosti, aby sme tomu naozaj uverili. Bude to vyzerať nejak takto:

$$(5, 3, 7, \underline{1}, 2, 4, 6, 8)$$
$$1, (3, 7, 5, \underline{2}, 4, 6, 8)$$
$$1, 2, (7, 5, \underline{3}, 4, 6, 8)$$
$$1, 2, 3, (5, 7, \underline{4}, 6, 8)$$
$$1, 2, 3, 4, (7, \underline{5}, 6, 8)$$
$$1, 2, 3, 4, 5, (7, \underline{6}, 8)$$
$$1, 2, 3, 4, 5, 6, (\underline{7}, 8)$$
$$1, 2, 3, 4, 5, 6, 7, (8)$$

Táto metóda sa dá použiť aj na danú implementáciu quicksortu. Vždy zvolíme najľavejší prvok a stredný prvok ako dve najmenšie doteraz nepridelené čísla. T.j. pre $n = 8$ začneme s $(\underline{a}, b, c, \underline{d}, e, f, g, \underline{h})$, z podčiarknutých čísel sa volí pivot. Ak zvolíme $a = 1$, $d = 2$, pole sa rozdelí na (a, d) a $(\underline{c}, b, \underline{e}, f, g, \underline{h})$. Zvolíme $c = 3$ a $e = 4$ a ostane $(\underline{b}, \underline{f}, g, \underline{h})$ a potom $b = 5$, $f = 6$, $g = 7$, $h = 8$. Tento postup nie je úplne najlepší, ale dokáže si vynútiť kvadratický počet výmen!

Koľko porovnaní to bude presne? Dá sa to spočítať. Pre párne $n > 6$ je počet porovnaní presne $n^2/4 + 6n - 19$. Teda pre $n = 4700$ dostaneme 5 550 681 porovnaní. Hodnoty blížiace sa k tomuto dostali 13 bodov.

README!!! Tento odsek čítajte pomaly, pretože je priam nabitý informáciami. Touto metódou vieme vyrobiť vstup, možno nie úplne najhorší, ale taký, ktorý si vynúti kvadratický počet porovnaní, pre prakticky *ľubovoľnú* implementáciu quicksortu (ak predpokladáme, že pivot sa vyberá deterministicky len z konštantného množstva kandidátov (čiže nie náhodne a nie napríklad tak, že sa nájde medián všetkých prvkov a rozdelí sa to podľa neho)).

Kvôli rýchlym čitateľom zopakujeme: Ak sa pivot vyberá iba z malej hŕstky čísel a nenáhodne, vieme vyrobiť dosť zlý vstup (taký, s ktorým by mohol vytúnený bubblesort súťažiť).

Navyše si všimnite, že program, ktorý za nás takúto postupnosť vymyslí dokážeme vyrobiť veľmi rýchlo – iba upravíme samotný quicksort tak, že na začiatku budú v poli iba čísla $n + 1$, ktoré označujú ešte neurčenú hodnotu. Tesne pred výberom pivota niektoré hodnoty

určíme. Tiež si treba v ešte jednom poli udržiavať informáciu, odkiaľ dané čísla pochádzajú, aby sme vedeli príslušnú postupnosť spätne zrekonštruovať.

Lepšie riešenie: Lepšie riešenie získame, ak budeme určovať jednotlivé prvky šikovnejšie. Tu mohlo pomôcť (a mne pomohlo) vyskúšanie všetkých permutácií čísel $1, \ldots, n$ pre malé n. Tieto pokusy ukázali, že ak vezmeme postupnosť tvaru $2, 3, 4, \ldots, n-2, n, n-1, 1$, triedené pole sa nebude skracovať o 2 ale iba o 1, takže dosiahneme asi dvakrát toľko porovnaní. Ako bude vyzerať výpočet pre $n = 8$? Takto:

$$
\begin{aligned}
&(\underline{2}, 3, 4, \underline{5}, 6, 8, 7, \underline{1})\\
&1, (\underline{3}, 4, 5, \underline{6}, 8, 7\underline{2})\\
&1, 2, (\underline{4}, 5, \underline{6}, 8, 7, \underline{3})\\
&1, 2, 3, (\underline{5}, 6, \underline{8}, 7, \underline{4})\\
&1, 2, 3, 4, (\underline{6}, \underline{8}, 7, \underline{5})\\
&1, 2, 3, 4, 5, (\underline{8}, \underline{7}, \underline{6})\\
&1, 2, 3, 4, 5, (\underline{6}, \underline{7}), 8\\
&1, 2, 3, 4, 5, 6, (\underline{7}, \underline{8})
\end{aligned}
$$

Presný počet porovnaní (pre $n \geq 4$) je $n^2/2 + 11n/2 - 4$. A zdá sa, že toto je aj najhorší vstup, ak musia byť všetky čísla rôzne.

Ešte lepšie: Ak povolíme aj rovnaké čísla, napríklad pre $n = 8$ existuje ešte lepšia postupnosť:

$$
\begin{aligned}
&(\underline{3}, 5, 1, \underline{2}, 6, 4, 3, \underline{5})\\
&(\underline{3}, \underline{2}, \underline{1}), 5, 6, 4, 3, 5\\
&(\underline{1}, \underline{2}), 3, 5, 6, 4, 3, 5\\
&1, (\underline{2}, \underline{3}), 5, 6, 4, 3, 5\\
&1, 2, (\underline{3}, 5, \underline{6}, 4, 3, \underline{5})\\
&1, 2, (\underline{3}, \underline{5}, 3, \underline{4}), 6, 5\\
&1, 2, (\underline{3}, \underline{4}, \underline{3}), 5, 6, 5\\
&1, 2, (\underline{3}, \underline{4}), 3, 5, 6, 5\\
&1, 2, 3, (\underline{4}, \underline{3}), 5, 6, 5\\
&1, 2, 3, 3, (\underline{4}, \underline{5}), 6, 5\\
&1, 2, 3, 3, 4, (\underline{5}, \underline{6}, \underline{5})\\
&1, 2, 3, 3, 4, (\underline{5}, \underline{6}), 5\\
&1, 2, 3, 3, 4, 5, (\underline{6}, \underline{5})
\end{aligned}
$$

Táto postupnosť využíva isté blbé vlastnosti zadanej implementácie a potrebuje až 79 porovnaní na utriedenie (predchádzajúci postup dáva 72).

Hoci sme si ukázali, ako nájsť veľmi veľmi zlú postupnosť, hľadanie úplne úplne najsamhoršej postupnosti, resp. výsledok pre $n = 4700$ je stále otvorený problém. Takže sa môžete naďalej snažiť a hľadať horšie a horšie postupnosti. Veľa šťastia!

Mám používať quicksort? Na záver, keď sme tak „dodžubali“ ten quicksort by som sa ešte rád vyjadril k tejto otázke. Určite áno. Quicksort je jeden z najrýchlejších triediacich algoritmov (odtiaľ má aj svoje meno). Treba si však dať pozor na jeho implementáciu. V prvom rade je dobre voliť pivota náhodne, t.j. vo vyššie uvedenej implementácii riadok `p:=a[(l+r) div 2];` nahradíme riadkom `p:=a[l+random(r-l+1)];`. Takto povýšime priemernú zložitosť $O(n \log n)$ na očakávanú zložitosť $O(n \log n)$. Čas už potom nezávisí od vstupu, ale od toho, aké náhodné čísla vygenerujeme. Je to ako keby sme pred triedením pole náhodne premiešali (a na náhodnej postupnosti beží quicksort veľmi dobre).

Aj keď z teoretického hľadiska je lepší heapsort (čas $O(n \log n)$ je garantovaný aj v najhoršom prípade), v praxi je rýchlejší quicksort.

Druhá vec, na ktorú si treba pri quicksorte dávať pozor sú rovnaké čísla. Ideálne je rozdeliť postupnosť na 3 časti: na čísla menšie ako pivot, väčšie ako pivot a na čísla rovnaké ako pivot a potom rekurzívne zotriediť dve časti. Stačí však, keď sa pole rozdelí na $\leq$ a $\geq$, pričom z tých rovnakých bude zhruba polovica v jednej a polovica v druhej časti (to vyššie uvedená implementácia spĺňa.

6. O škole

opravoval MišoF.
(max. 15 bodov)

Dostať 8 bodov bolo triviálne – stačilo pochopiť zadanie a naprogramovať ho nejak. Našli sa hneď tri spôsoby, ako na to:

Prvý postup bol priamočiary. Umiestňujem žiakov pekne za radom od najmenšieho. Keď príde žiak k a povie, že naľavo od neho má byť x menších, dáme ho jednoducho na miesto $x+1$ a všetkých, čo doteraz stáli na mieste x a ďalej, jednoducho posunieme o miesto doprava.

Druhý postup umiestňuje žiakov pekne zľava doprava. Kto môže byť najviac vľavo? Určite to nemôže byť nik, kto má mať naľavo od seba niekoho menšieho. Takže to musí byť niekto, kto má na vstupe nulu. No ale kto z nich? Opäť je to jednoduché – ten najvyšší. Ak by to nebol on, dostal by tým naľavo od seba niekoho menšieho, a už by tú svoju nulu nedosiahol.

Vieme teda umiestniť prvého žiaka. A čo teraz? No, všetkým väčším od neho zmenšíme hodnotu zo vstupu o 1, lebo práve dostali jedného menšieho, ktorý bude naľavo od nich.

A tým sme sa dostali do tej istej situácie, len máme o žiaka menej. Tak opakujeme celý vyššie uvedený postup, až kým sa nám žiaci neminú.

Tretí postup bol založený na nasledujúcej myšlienke: Od najvyššieho nie je väčší nik. Takže počet menších naľavo od neho je rovný počtu všetkých naľavo od neho, a teda ho môžeme rovno umiestniť tam kam chce.

Takže budeme umiestňovať žiakov postupne od najvyššieho. Vo všeobecnosti to bude fungovať takto: nejakí žiaci sú už umiestnení, medzi nimi sú nejaké voľné miesta. Nech teda teraz umiestňujeme žiaka k, ktorý má mať naľavo od seba x menších. To teda znamená, že ho musíme umiestniť presne na $(x+1)$. doteraz voľné miesto zľava. Na každé voľné miesto totiž časom príde niekto menší, a jedine takto dosiahneme, že menších naľavo od neho bude práve x.

Rýchlejšie nie-vzorové riešenie

V prvom riešení vieme rýchlejšie spraviť operáciu „vložím na miesto $x+1$ a posuniem zvyšok“.

Na to „stačí“ zvoliť si nejaký vyvažovaný strom, ktorý sa nám páči. Možností je veľa: AVL-stromy, 2-3- či 2-3-4-stromy, červeno-čierne stromy, splay stromy či treapy. Ak vám to znie ako prechádzka botanickou záhradou, nemýlite sa. Všetky tieto stromy (a mnohé iné) naozaj existujú, ale v súťažnej praxi sa s ich implementáciou príliš nestretnete. Dôvod je jednoduchý – je s tým priveľa práce.

No ale pre úplnosť tu uvedieme aspoň myšlienku takéhoto riešenia. Už umiestnených žiakov si budeme pamätať vo vrcholoch vybraného vyvažovaného stromu. Pozor, **nepôjde** o vyhľadávací strom – žiaci nebudú „zľava doprava“ usporiadaní podľa svojich čísel, ale podľa miesta v rade, na ktorom stoja.

V každom okamihu teda bude platiť, že keby sme si náš strom vypísali v *inorder zápise* (teda rekurzívne vypíšeme ľavý podstrom, vypíšeme koreň a rekurzívne vypíšeme pravý podstrom), dostaneme aktuálny rad vypísaný pekne zľava doprava.

Pri vkladaní nového žiaka teda potrebujeme v strome nájsť miesto, kde ho vložiť tak, aby naľavo od neho bolo práve x iných. Toto vieme efektívne spraviť tak, že si v každom

vrchole budeme pamätať veľkosť podstromu pod týmto vrcholom. Ak teda napríklad chceme vložiť prvok na 16-te miesto a v ľavom podstrome je 9 prvkov, budeme vkladať do pravého podstromu a v rámci neho to bude $16 - 9 - 1 =$ šieste miesto.

Tieto informácie si musíme po vložení prvku samozrejme prerátať (na ceste dĺžky $O(\log N)$ z práve vloženého vrcholu do koreňa) a takisto ich musíme upravovať pri následnom vyvažovaní stromu. Ak si vyberieme taký strom, ktorý sa vyvažuje rotáciami, tak aj to ide spraviť efektívne – vždy po rotácii najskôr prerátame spodný a potom horný vrchol.

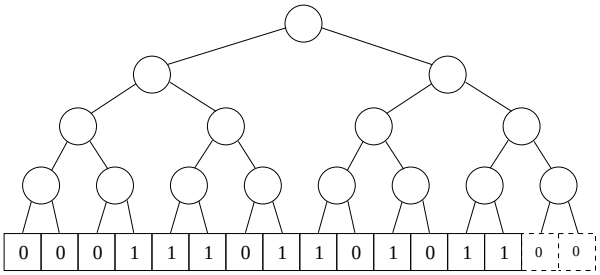
Takéto riešenie by malo časovú zložitosť $O(N \log N)$. Každopádne na jeho implementácii by bolo viac práce ako na kostole. A preto je tu:

Vzorové riešenie

Budeme vychádzať z tretieho riešenia. Ukážeme, ako si šikovne pamätať rozmiestnenie žiakov tak, aby sme vedeli v logaritmickom čase povedať, kam umiestniť práve umiestňovaného žiaka. Inými slovami, budeme chcieť vedieť k ľubovoľnému x nájsť, kde je x-té voľné miesto zľava.

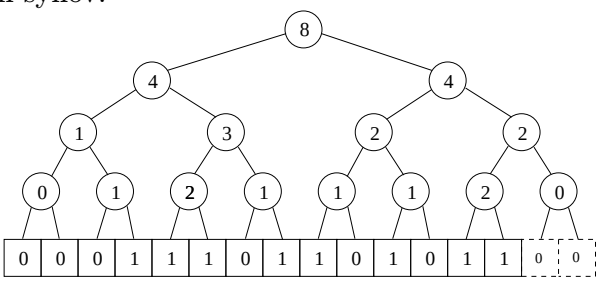
Pre jednoduchosť budeme predpokladať, že N je mocnina dvoch. (Ak by nebolo, zväčšíme ho na najbližšiu mocninu dvoch. Uvedomte si, že tým sa zväčší menej ako na dvojnásobok pôvodnej hodnoty.)

Použijeme dátovú štruktúru známu pod menom *intervalový strom*. Ten bude vyzerať takto:

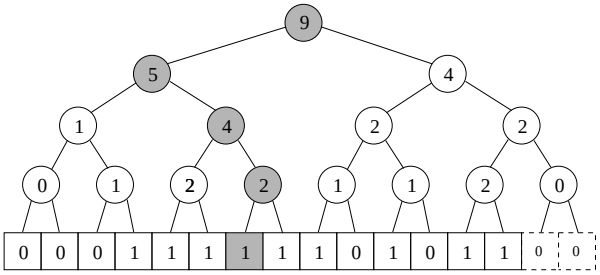


Listy intervalového stromu zodpovedajú jednotlivým miestam vo výslednom poradí. Budeme si v nich značiť, či je dané miesto ešte voľné (0) alebo už obsadené (1).

Všimnite si, že vnútorný vrchol, ktorý je k úrovní nad listami, zodpovedá intervalu obsahujúcemu 2^k po sebe idúcich miest. Tie intervaly, ktoré zodpovedajú vrcholom nášho stromu, budeme volať *jednoduché*. Pre každý jednoduchý interval si budeme v našom strome pamätať, koľko obsadených políčok v ňom je. Toto jednoducho znamená, že v každom vrchole bude súčet z jeho dvoch synov:



Vyplnenie vrcholov na začiatku výpočtu je jasné – jednoducho všade budú nuly. Takisto obsadenie doteraz voľného políčka je ľahké – zvýšime o 1 hodnoty na ceste zo zodpovedajúceho listu do koreňa:



Zostáva posledná operácia: ako k danému x nájsť x-té voľné miesto zľava?

V prvom rade si uvedomme, že ku každému vrcholu vieme ľahko povedať, ako dlhý interval mu zodpovedá. A teda ak vieme počet plných miest v tomto intervale, vieme aj počet prázdnych miest.

Hľadať správne miesto budeme podobne ako pri binárnom vyhľadávaní. Začneme v koreni stromu, ktorý zodpovedá celému poradiu. V každom kroku sa najskôr pozrieme na počet v voľných miest v ľavom podstrome. Ak $v \leq x$, ideme hľadať do ľavého podstromu. A ak $v > x$, ideme do pravého podstromu, no a v tom budeme hľadať $(x - v)$-te voľné miesto.

Aj túto operáciu teda vieme spraviť v čase $O(\log N)$.

Poznámky na záver

V prvom rade: na intervalový strom si dajte fakt že pozor. Táto dátová štruktúra toho vie omnoho viac. Finta je napríklad, že z $O(\log N)$ jednoduchých intervalov vieme „naskladať“ ľubovoľný interval. A u mnohých typov informácií (napríklad minimum alebo súčet hodnôt v danom intervale) vieme teda v $O(\log N)$ naskladať informáciu pre celý interval.

V druhom rade k implementácii. Intervalový strom je skvelý tým, že nám naň stačí obyčajné pole. Bude to vyzerať presne rovnako ako obyčajná halda: Koreň bude na poli s indexom 1, no a vrchol s indexom k bude mať synov na políčkach $2k$ a $2k + 1$.

U takéhoto stromu, kde je v každom vrchole súčet hodnôt zo synov, sa dokonca dá ísť ešte o krok ďalej. Uvedomte si, že teraz je kopa údajov v strome zbytočná. Ak vieme hodnotu vo vrchole a v jeho ľavom synovi, vieme z nich dorátať hodnotu v pravom synovi, nepotrebujeme ju ukladať. Ak si šikovne zvolíme, ktoré hodnoty si pamätať a ktoré nie, vystačíme si dokonca s pôvodným poľom.

Takto získanej dátovej štruktúre sa podľa jej autora hovorí Fenwick tree, u nás je ale medzi pospolitým ľudom známa pod názvom Fínsky stromček. Ak o nej chcete vedieť viac, pozrite si vzorové riešenia IOI 2001, úloha Mobiles, prípadne Fenwickov pôvodný článok `http://cs.ubc.ca/local/reading/proceedings/spe91-95/spe/vol24/issue3/spe884.pdf`

Listing programu:

```
#include <iostream>
using namespace std;
#define MAXN 1000000

// "plne" je intervalovy strom, pamatame kolko plnych policok uz je v kazdom intervale
int plne[2*MAXN + 47];
int vstup[MAXN];
int vysledok[MAXN];
int N, N2;              // N2 je najmensia mocnina 2 co je >= N

void umiestni(int co, int kam, int kde, int dlzka) {
  // umiestni "co" na "kam"-tu volnu poziciu
  // "kde" je vrchol stromu kde prave sme
  // "dlzka" je dlzka intervalu ktoremu aktualny vrchol zodpoveda

  // v tomto podstrome pribudne jeden prvok
  plne[kde]++;
```

```
  if (dlzka == 1) {
    // prisli sme do listu, umiestnime prvok
    vysledok[kde-N2] = co;
  } else {
    // zistime ci pridavame do laveho alebo praveho podstromu
    int volne_vlavo = dlzka/2 - plne[kde*2];
    if (volne_vlavo >= kam)
      umiestni(co,kam,kde*2,dlzka/2);
    else
      umiestni(co,kam-volne_vlavo,kde*2+1,dlzka/2);
  }
}

int main() {
  cin >> N; N2=1; while (N2<N) N2*=2;
  for (int i=0; i<N; i++) cin >> vstup[i];
  for (int i=N-1; i>=0; i--) umiestni(i+1,vstup[i]+1,1,N2);
  for (int i=0; i<N; i++) cout << vysledok[i] << endl;
}
```

7. Odhoď do paže

opravoval Miňo
(max. 15 bodov)

Ako všetci z vás uhádli, pointa úlohy nebola zložitá – stačilo zistiť uhol (alebo niečo ekvivalentné) medzi dvoma susednými kamarátmi, zo všetkých takýchto čísel nájsť maximum a sme za vodou.

Prístupov, ktoré sa dali použiť, je viacero, asi najjednoduchším z nich je zvoliť si jeden vektor (napríklad kladnú Y-ovú os), od ktorej budeme počítať uhly jednotlivých kamarátov. Samotný uhol vieme potom vypočítať pomocou goniometrickej mágie. Tento je však iba od 0 po π. Ten ešte treba napríklad podľa znamienka y-ovej súradnice upraviť tak, aby sme mali uhol z rozsahu 0 až 2π. Pre veľkých lenivcov však existuje minimálne v C funkcia `atan2` (použitá aj vo vzorovom riešení), ktorá sa o toto všetko postará za nás. :-)

Jednoduchšia varianta riešenia vyzerá nasledovne. Kamarátov si najskôr zotriedime podľa ich uhla – tým si zabezpečíme, že medzi dvoma susednými kamarátmi v poli skutočne nikto nie je. A potom už iba spočítame pre každú dvojicu susedných kamarátov veľkosť uhla medzi nimi. Nezabudneme ani na uhol medzi prvým a posledným v poradí (ten treba rátať trošičku ináč) a ani na to, že ak $N = 2$, sú medzi dvoma kamarátmi hneď dva uhly. Časová zložitosť tohoto riešenia je vďaka triedeniu $O(N \log N)$.

Existuje ale aj prefíkanejšie riešenie v lineárnom čase. Celých 2π rad kružnice si rozdelíme na N rovnakých intervalov, do ktorých zaradíme kamarátov. Všimnite si, že z uhlu vieme v konštantnom čase zistiť, do ktorého intervalu patrí.

Trik bude v tom, že na to, aby sme našli najväčší uhol, ich nepotrebujeme všetky utriediť. Hľadaný uhol má určite veľkosť $2\pi/N$ alebo viac. To okrem iného znamená, že kamaráti, ktorí ho určujú, patria do rôznych intervalov.

To isté inými slovami: V rámci jedného intervalu sú vzdialenosti kamarátov rozhodne menšie ako $2\pi/N$, takže nás nezaujímajú. v rámci každého intervalu si teda stačí pamätať iba dvoch krajných kamarátov – ostatní vo vnútri majú medzi sebou príliš malé uhly a s kamarátmi z iných intervalov nie sú „priami“ susedia.

V podstate sa nám podarilo okresať potrebu všeobecného $O(N \log N)$ triedenia na špeciálny prípad, kde si vystačíme s bucket sortom v čase $O(N)$.

Teraz už iba stačí spraviť presne to isté, čo sme robili v predchádzajúcom prípade – skontrolovať priamych susedov z rôznych intervalov a zistiť najväčší uhol. Jediný špeciálny prípad, ktorý tu môže nastať, je že niekotrý interval je prázdny. To nám ale nevadí – prázdny

interval iba znamená, že medzi dvoma kamarátmi je dosť veľká diera. Jednoducho si budeme pamätať posledného kamaráta, ktorého sme videli a keď narazíme na ďalšieho, zistíme uhol. Celá takáto kontrola je lineárna.

Ani jedno vaše riešenie nebolo v lineárnom čase – väčšina z nich mala zložitosť práve $O(N \log N)$ a mala bodový základ 12. $O(N^2)$ riešenia dostali maximálne 9 bodov. Vyskytlo sa aj zopár nefunkčných riešení. Body sa dali veľmi ľahko stratiť za slabý popis (1-2 body), chýbajúce alebo nesprávne odhady zložitostí (po 1 bode) alebo neošetrené okrajové prípady zhruba po 1 bode. Schválne, ruku na srdce, kto neriešil úlohu s tichým predpokladom, že kamaráti budú aspoň traja?

Listing programu:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_KAMARATOV 1047

typedef struct { double x,y,uhol; } kamarat;

kamarat vstup[MAX_KAMARATOV];
int rozdelenie[MAX_KAMARATOV][2];

int main(void) {
    int N, i, p, lastseen=-1, max1=0, max2=0;
    double policko;

    scanf("%d\n",&N); policko=2*M_PI/N;

    if (N == 0) { printf ("Odhod hocikam do paze.\n"); return 0; };

    for (i=0;i<N;i++) {
        rozdelenie[i][0]=-1;rozdelenie[i][1]=-1;
        scanf("%lf %lf\n",&vstup[i].x,&vstup[i].y);
        vstup[i].uhol=atan2(vstup[i].y,vstup[i].x) + M_PI;
    };

    for (i=0;i<N;i++) {
        p=floor(vstup[i].uhol/policko);
        if (rozdelenie[p][0]==-1)                              rozdelenie[p][0]=i;
        if (rozdelenie[p][1]==-1)                              rozdelenie[p][1]=i;
        if (vstup[rozdelenie[p][0]].uhol > vstup[i].uhol) rozdelenie[p][0]=i;
        if (vstup[rozdelenie[p][1]].uhol < vstup[i].uhol) rozdelenie[p][1]=i;
    };

    for (i=0;i<N;i++) {
        if (lastseen!=-1 && rozdelenie[i][0]!=-1) {
            if (vstup[ rozdelenie[i][0] ].uhol - vstup[lastseen].uhol >
                vstup[max2].uhol - vstup[max1].uhol) {
                    max1=lastseen; max2=i;
                };
        };
        if (rozdelenie[i][1]!=-1) {
            lastseen=rozdelenie[i][1];
        };
    };

    /* Osetrime specialny pripad */
    for (i=0;i<N;i++) {
```

```
        if (rozdelenie[i][0]!=0) {
            if (vstup[ rozdelenie[i][0] ].uhol - vstup[lastseen].uhol + 2*M_PI >
                vstup[max2].uhol - vstup[max1].uhol) {
                    max1=lastseen; max2=i;
                };
            break;
        };
    };

    printf ("Najvacsi uhol je medzi [%lf,%lf] a [%lf,%lf]\n",
            vstup[max1].x, vstup[max1].y, vstup[max2].x, vstup[max2].y);
    return 0;
};
```

8. O distribúcii chemikálií

opravoval Zemčo
(max. 15 bodov)

Vitajte v tomto vzoráku. Prišlo nie málo riešení, ktoré boli až na výnimky správne. Mirka ste veľmi potešili a ďakuje Vám za to. Ak niekedy budete potrebovať nejakú chemikáliu, obráťte sa na neho. Sľúbil zľavu.

Počet možných postupov pri riešení tohto problému bol celkom veľký, poďme si ich systematicky spomenúť. Napokon dospejeme k celkom prekvapujúcemu vzoráku.

Našli sa takí, ktorí na tento príklad našili Floyd-Warshallov algoritmus s kubickým časom. Príliš hrubá záplata na takýto tenký problém. Jej hlavný problém spočíva v tom, že nijako nevyužíva fakt, že sa bavíme o strome. Počet bodov 7.

Iné jednoduchšie riešenie pracuje v čase $O(N^2)$. Jedno prehľadávanie v strome nás stojí čas $O(N)$, zároveň prehľadávanie upravíme tak, aby nám zo štartovacieho vrcholu zisťovalo súčet ciest do všetkých ostatných vrcholov. Keď toto spustíme zo všetkých miest, dostaneme riešenie. Stále pomalé. Počet bodov 9.

Ďalšia skupina poslala riešenia bežiace v $O(N \log N)$. Idea spočítala v tom, že v každom okamihu máme v halde všetky listy a v každom kroku sa z haldy dozvieme, ktorý je najvýhodnejší. Tento odstránime. Vrchol, ktorý prežije, je riešenie. Viem, popísal som to až príliš neformálne, ale iniciatívny čitateľ si detaily domyslí iste sám. Napokon, až také zaujímavé to riešenie pre nás momentálne nie je. Počet bodov 12. Podobná myšlienka sa dala realizovať aj v kvadratickom čase, čomu zodpovedali aj body. A myšlienka sa dala realizovať ešte aj nekorektne, aj v tomto prípade za zodpovedajúcu odmenu.

Dostávame sa k lineárnemu času, ktorý je zároveň optimálny. Každý s troškou skúseností s príkladmi so stromami pri prečítaní zadania iste zavetril, že to pôjde takto rýchlo. Avšak postupov ostalo stále viac. Jedna z možností je spraviť dve prehľadávania z jedného vrcholu. V tomto momente je vhodné si strom predstaviť zakorenený – vezmite ľubovoľný vrchol, veľké kladivo a klinec a za tento vrchol úbohý strom pribite na stenu. Vrchol, o ktorý sme pribili nazveme koreň. Teraz pre vrcholy v strome platí, že jednou hranou sú spojené so vrcholom nad sebou, prípadné ďalšie hrany smerujú dole a tvoria podstrom visiaci pod daným vrcholom. Riešenie pracuje tak, že prvým prehľadávaním z koreňa pre každý vrchol zistí súčet ciest a počet vrcholov v podstrome pod ním. Druhým prehľadávaním potom sme schopní túto informáciu doplniť pre každý vrchol o zvyšnú hranu zhora, čím budeme schopní nájsť odpoveď. Za toto riešenie je samozrejme 15 bodov, lebo dosahuje dokázateľne optimálny čas[4].

[4] Ako vraví jeden vyslúžilý KSPák, vstup je dobré načítať...

Náš vzorák predstavuje iný prístup k veci. Bude ľahší na naprogramovanie, ale budeme k nemu potrebovať troška viac teórie. Úvodná poznámka znie: pri všetkých tvrdeniach predpokladám $N > 1$. Iste si viete predstaviť, ako by to bolo v prípade stromu s jedným vrcholom. Ďalej je užitočné spomenúť, že v strome je buď $2\left\lfloor \frac{N}{2} \right\rfloor$ alebo $2\left\lfloor \frac{N}{2} \right\rfloor + 1$ vrcholov,[5] v závislosti od parity čísla N. Takže poďme na to:

Lema 1: Uvažujme ľubovoľný vrchol v. Sklad sa nám oplatí presunúť z v do nejakého jeho suseda u práve vtedy, ked počet vrcholov, do ktorých sa vieme z v dostať cez u je aspoň $\left\lfloor \frac{N}{2} \right\rfloor + 1$.

Dôkaz: Majme suseda u, pre ktorého platí podmienka zo znenia. Uvažujme, ako sa zmení súčet dĺžok ciest, ak presunieme sklad z v do u. Označme c hodnotu hrany medzi týmito dvoma vrcholmi. Vieme, že $c > 0$. Teraz je dobré pripomenúť, že medzi každými dvoma vrcohlmi v strome je práve jedna cesta, takže všetky vrcholy môžeme rozdeliť na dve skupiny – tie, ktoré su bližšie k u a tie, ktorá su bližšie k v. Nech a je počet tých prvých, b počet tých druhých. Zmena, ktorá nastane presunom skladiska do u bude rovná $cb - ca$, lebo sklad sa pribíži k a vrcholom a vzdiali sa od b vrcohlov. Ľahko ale vidno, že vzhľadom k predpokladanej vlasnosti vrcholu u je $a > b$. Takže v u bude súčet menší. Rovnako sa ukáže aj to, že nie je výhodné presunúť sklad do suseda, pre ktorého neplatí podmianka zo znenia.

Všimnite si, že na samotnej hodnote c prekvapujúco vôbec nezáleží, jediné, čo je dôležité, sú počty vrcholov.

Lema 2: Ľubovoľný vrchol v je riešením práve vtedy, keď pre každého jeho suseda u platí, že počet vrcholov, do ktorých sa vieme dostať z v cez u je ostro menší ako $\left\lfloor \frac{N}{2} \right\rfloor + 1$.

Dôkaz: Z predchádzajúcej lemy vieme, že určite nebude výhodnejší žiaden sused vrchola v. Poďme ešte ukázať, že sa nemôže stať, že výhodnejšie bude umiestniť sklad niekam ďalej od nás. Poďme na to sporom, nech existuje vrchol z, pre ktorý platí, že je lepší ako vrchol v zo znenia vety. Medzi týmito vrcholmi určite existuje práve jedna cesta. Tak sa teda po nej vydajme. Po prvom kroku si už ale môžeme všimnúť škaredú vec. Nech sme sa z vrcholu v pohli do nejakého vrcholu y. Potom ale nutne z y smerom do v je vetva, ktoré vedie k počtu vrcholov, ktorý je väčší alebo rovný $\left\lfloor \frac{N}{2} \right\rfloor + 1$. A dá sa všimnúť, že na ďalšej ceste k z sa určite tejto vetvy nezbavíme, takže z bude mať suseda, ktorý bude viesť k nadpolovičnej skupine vrcholov. Ale potom podľa *Lemy 1* z nemôže byť riešením! Tým sme sa dostali k sporu. Ďalšia úvaha nás dovedie k tomu, že sklad vrátime späť do v, kde to bude najvýhodnejšie.

Posledné výsledky chcú ešte troška doplniť. V prvom rade si všimnite, že aj v *Leme 2* platí, že nás ohodnotenie hrán vôbec nezaujíma. Teraz sa zamyslite nad tým, či je teda riešenie vždy jediné. Že áno? Nuž, *Lema 1* hovorí, že ak je splnená uvedená podmienka, tak žiadny sused nie je výhodnejší. Môže ale nastať situácia, kedy je (práve jeden) sused rovnako výhodný. Určite ľahko odhalíte, kedy takáto situácia nastáva.

Teraz už vieme vymyslieť prefíkaný algoritmus, ktorý nájde hľadaný optimálny vrchol.

Začneme tým, že pre každú hranu vychádzajúcu z každého vrchola spočítame, koľko vrcholov je v podstrome „za ňou“. Toto vieme spraviť jedným prehľadávaním do hĺbky, ktoré bude vracať počty vrcholov v podstromoch. (Síce nevieme počet vrcholov z vetvy, z ktorej prehľadávanie prišlo, ale to si ľahko doplníme, keďže vieme, že vrcholov je N a všetky ostatné vetvy poznáme.)

Teraz máme dve možnosti, ako nájsť hľadaný vrchol. Prvá je postupne doň prísť. Začneme trebárs vo vrchole, z ktorého sme prehľadávali. No a teraz keď stojíme v nejakom vrchole, tak sa pozrieme, či sa oplatí presunúť do niektorého suseda. Ak áno, spravíme to a pokračujeme. Po konečnom počte takýchto presunov sa musíme dostať do situácie, kedy už žiaden zo susedov nie je lepší od vrchola, kde práve sme. No a z vyššie dokázaných vecí vieme, že ani nikde inde v grafe lepšieho vrchola niet, a teda sme našli optimálny vrchol.

[5] $\lfloor x \rfloor$ je dolná celá časť čísla x. Teda je to najbližšie menšie alebo rovné celé číslo

Ide to ale aj ináč, možno ešte trochu jednoduchšie. Stačí sa v každom vrchole pozrieť na veľkosti jeho podstromov, nájsť ten správny (kde žiaden podstrom neobsahuje viac ako polovicu vrcholov) a vypísať ho. Takto to robí vzorový program.

Listing programu:

```
const MAXEDGES = 20000;
var vetvy,graf:array[1..MAXEDGES]of array of longint;
    stupen,bol: array[1..MAXEDGES]of longint;
    a,b,i,j,n:longint;
    ok: boolean;

function hladaj(v: longint):longint;
var i,odkial,vratim:longint;
begin
   bol[v]:=1; vratim:=0; odkial:=0;
   for i:=1 to stupen[v] do if bol[graf[v][i]]=0 then
   begin
      vetvy[v][i]:=hladaj(graf[v][i]);
      vratim := vratim + vetvy[v][i];
   end
   else odkial := i;
   vratim := vratim + 1;
   if odkial > 0 then vetvy[v][odkial] := N - vratim;
   hladaj := vratim;
end;

begin
   readln(n);
   for i:=1 to n do
   begin
      setlength(vetvy[i],0); setlength(graf[i],0); bol[i]:=0; stupen[i]:=0;
   end;
   for i:=1 to n-1 do
   begin
      readln(a,b);
      inc(stupen[a]); inc(stupen[b]);
      setlength(graf[a],stupen[a]+1); setlength(graf[b],stupen[b]+1);
      graf[a][stupen[a]]:=b;
      graf[b][stupen[b]]:=a;
   end;
   for i:=1 to n do setlength(vetvy[i],stupen[i]+1);
   hladaj(1);
   for i:=1 to n do
   begin
      ok := true;
      for j:=1 to stupen[i] do if vetvy[i][j]>(n div 2) then ok:=false;
      if ok = true then begin writeln(i); halt; end;
   end;
end.
```

9. Tipujeme výsledky extraligy

opravoval Lukáš
(max. 15 bodov)

Na tomto príklade bolo zákerné to, že v ňom bolo treba použiť prístup z opačného konca. Pár ľuďom sa to podarilo a dokonca mali aj riešenie bežiace v čase $O(N^2)$. Za takéto riešenie

som dával 15 bodov – vlastne iné správne sa ani nevyskytlo. Dvaja z vás použili fakt, že budeme dávať stávky len na Poprad bez uvedenia dôkazu – stŕhal som za to až 2 body.

Poďme si teraz načrtnúť myšlienku riešenia. Namiesto hodnoty P peňazí budeme hovoriť iba o násobku tohto množstva. Čiže ak ide o x peňazí, v skutočnosti je to $x \cdot P$. Označme $f(z,p)$ minimálny zisk Miška, keď celú sériu nakoniec vyhrá Poprad, Žilina potrebuje vyhrať ešte z zápasov a Poprad p (v tomto okamihu je teda stav série $(N-z):(N-p)$). Všimnime si, že pre $z=0$ nebude funkcia definovaná, lebo má zmysel iba vtedy, keď Poprad vyhrá celú sériu. V tomto okamihu Miško staví x peňazí na Poprad alebo na Žilinu. Malá odbočka – situáciu si opäť zjednodušíme – ak bude x záporné, Miško staví na Žilinu, v opačnom prípade na Poprad. Ďalej teda budeme iba hovoriť, že stavíme x peňazí na Poprad, pričom x môže byť aj záporné. Ak Poprad ďalší zápas vyhrá, v najhoršom prípade získame $(1+x)\cdot f(z,p-1)$; ak vyhrá Žilina, v najhoršom prípade máme $(1-x)\cdot f(z-1,p)$ peňazí. Hodnota $f(z,p)$ je minimum týchto dvoch hodnôt, lebo nás zaujíma vždy ten najhorší prípad. No a chceme, aby náš zisk bol v najhoršom prípade čo najvyšší. Hodnota $(1+x)\cdot f(z,p-1)$ s rastúcim x stúpa, hodnota $(1-x)\cdot f(z-1,p)$ s rastúcim x klesá. Minimum z obidvoch hodnôt má byť čo najväčšie, preto musí platiť rovnosť $(1+x)\cdot f(z,p-1) = (1-x)\cdot f(z-1,p)$. Po úprave dostávame

$$x = \frac{f(z-1,p) - f(z,p-1)}{f(z-1,p) + f(z,p-1)}$$

$$f(z,p) = \frac{2\cdot f(z-1,p)\cdot f(z,p-1)}{f(z-1,p) + f(z,p-1)}$$

Ešte potrebujeme doplniť niektoré triviálne hodnoty, aby sme mohli vyrátať všetky hodnoty $f(z,p)$. Napríklad vieme, že $f(z,0)=1$, lebo Poprad už vyhral a nedá sa ďalej stávkovať. Zostane nám preto presne tá suma peňazí (jednonásobok), ktorú máme. Ďalej vieme, že $f(1,p)=2^p$, lebo Poprad musí už všetky ďalšie zápasy vyhrať a môžeme staviť všetko. Náš algoritmus bude fungovať tak, že si predrátame všetkých N^2 hodnôt $f(z,p)$ do tabuľky $N+1 \times N+1$ a po každom zápase odpovieme, koľko treba staviť a na koho. Toto všetko vieme spraviť v kvadratickom čase s kvadratickou pamäťou.

Aby tento vzorák nebol krátky, ukážeme si, prečo vždy stavíme na Poprad (tento fakt niektorí z vás použili, bohužiaľ s nesprávnymi argumentmi). Chceme ukázať, že pre všetky z a p platí:

$$x = \frac{f(z-1,p) - f(z,p-1)}{f(z-1,p) + f(z,p-1)} \geq 0$$

Zjavne $f(z-1,p) \geq 0$ a $f(z,p-1) \geq 0$, lebo pri stávkovaní sa nemáme ako zadlžiť. Nerovnosť je preto ekvivalentná s touto:

$$f(z-1,p) - f(z,p-1) \geq 0 \Leftrightarrow f(z-1,p) \geq f(z,p-1)$$

Tvrdenie budeme dokazovať indukciou vzhľadom na súčet $z+p$. Pre $z+p=2$ platí $2 = f(1,1) \geq f(2,0) = 1$. Nech teda predpoklad platí pre $z+p-2$, dokážeme ho aj pre $z+p-1$:

$$f(z-2,p) \geq f(z-1,p-1) \geq f(z,p-2)$$

$$f(z-2,p) \geq f(z,p-2)$$

$$2f(z-2,p)f(z-1,p-1) \geq 2f(z-1,p-1)f(z,p-2)$$

Označme $a = 2f(z-2,p)f(z-1,p-1)$ a $b = 2f(z-1,p-1)f(z,p-2)$.

$$a\cdot f(z-1,p-1) \geq b\cdot f(z-1,p-1)$$

Platí $a\cdot f(z,p-2) = b\cdot f(z-2,p)$:

$$a\cdot f(z-1,p-1) + a\cdot f(z,p-2) \geq b\cdot f(z-1,p-1) + b\cdot f(z-2,p)$$

$$\frac{a}{f(z-1,p-1)+f(z-2,p)} \geq \frac{b}{f(z-1,p-1)+f(z,p-2)}$$
$$\frac{2f(z-2,p)f(z-1,p-1)}{f(z-1,p-1)+f(z-2,p)} \geq \frac{2f(z-1,p-1)f(z,p-2)}{f(z-1,p-1)+f(z,p-2)}$$
$$f(z-1,p) \geq f(z,p-1)$$

Vyšlo nám to, čo sme chceli. Stavíme teda vždy iba na Poprad.

Listing programu:

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

int main() {
    int n; scanf("%d", &n);
    double f[n+1][n+1];
    for (int i=0; i<=n; i++) f[i][0]=1;
    for (int i=1; i<=n; i++) f[1][i]=f[1][i-1]*2;

    for (int i=2; i<=n; i++)
        for (int j=1; j<=n; j++)
            f[i][j] = 2 * f[i][j-1] * f[i-1][j] / (f[i][j-1] + f[i-1][j]);

    int z=n, p=n;
    double P; scanf("%lf", &P);
    char c[2*n];
    scanf("%s", c);
    while (z>0 && p>0) {
        double x;
        if (z>1) x = (f[z-1][p] - f[z][p-1]) / (f[z-1][p] + f[z][p-1]);
          else x=1; //musíme už hrať vabank a staviť všetko na Poprad
        printf("Stav %.5lf na Poprad\n", x*P);
        if (c[2*n-z-p]=='P') {
            p--; P*=1+x;
        } else {
            z--; P*=1-x;
        }
    }
}
```

opravoval Mic
(max. 15 bodov)

10. Tak si zachráňme svet...

Veľmi ste ma nepotešili, ba skôr naopak. Prišli iba 2 funkčné riešenia a ani jedno nebolo optimálne. Preto písať o tom, ako som hodnotil bude asi zbytočné.

Najskôr si sformulujme našu úlohu do teórie grafov. Dostaneme neorientovaný neohodnotený graf (križovatky sú vrcholy, cesty sú hrany), a našou úlohou je ohodnotiť hrany tak, aby v každom vrchole bol súčet odchádzajúcich hrán 1.

Pomalé riešenie: Predstavme si, že pre každú hranu v našom grafe si spravíme jednu premennú a pre každý vrchol si spravíme rovnicu. Ak máme N vrcholov a M hrán, tak dostaneme N rovníc o M neznámych. Tie stačí už iba vyriešiť. To je však pomalé riešenie. Teraz si ukážeme lepšie riešenie.

Poznámka pred rýchlejším riešením: V nasledujúcich častiach budeme predpokladať, že náš graf je spojitý. Ak nie je, tak to vieme vyriešiť pre každý komponent grafu samostatne. Časovú ani pamäťovú zložitosť nám to nezhorší preto sa tým už ďalej nebudeme zaoberať.

Algoritmus A: Upravme jemne prehľadávanie do hĺbky a to tak, že keď sa budeme vracať z vrchola v (a prišli sme doň hranou u), tak vrátime na koľko treba zväčšiť ohodnotenie na hrane u, aby vo vrchole v bol súčet jedna (to znamená jedna mínus súčet hrán pri vrchole v) a pri vracaní patrične hranu nastavíme. Nie je ťažké nahliadnuť, že keď takéto prehľadávanie spustíme z nejakého vrchola, tak potom vo všetkých ostatných vrcholoch bude súčet ohodnotení hrán jedna.

Princíp vysávača: V niektorých grafoch, existuje takzvaný vysávač. Vysávačom pre nás bude cyklus nepárnej dĺžky. Prečo? Nech by sme chceli vo vrcholoch vysávača[6] ľubovoľný súčet (v rôznych vrcholoch kľudne aj rôzne hodnoty), tak vieme hrany na cykle nastaviť tak, že to bude platiť. Nech počet vrcholov v cykle je k a nech sú vrcholy očíslované od 1 po k. Označme hodnoty, ktoré chceme mať vo vrcholoch v cykle $v_1, v_2, \ldots, v_k$, a hodnoty na hranách $h_1, h_2, \ldots, h_k$ pričom vrchol i je spojený s vrcholom $i+1$ hranou s hodnotou h_i (pre $i < k$), a k-ty vrchol je spojený s vrcholom 1 s hranou s hodnotou h_k. Dostaneme nasledujúcich k rovníc: $h_1 + h_2 = v_2, h_2 + h_3 = v_3, \ldots, h_{k-1} + h_k = v_k, h_k + h_1 = v_1$ Od poslednej rovnice odrátame prvú, potom prirátame druhú, odrátame tretiu,..., až dostaneme rovnicu $2 \cdot h_k = v_1 - v_2 + v_3 - \cdots - v_{k-1} + v_k$, čiže h_k vieme vypočítať[7]. Z toho vypočítame spätným dosadením hodnoty $h_{k-1}, \ldots, h_1$, čím sme zistili potrebné ohodnotenia.

Ako sa používa vysávač? Najskôr by sa oplatilo nájsť vysávač. Potom z grafu odstránime hrany vysávača. Tým sa nám graf možno rozpadol na niekoľko komponentov. V každom z nich spustíme z niektorého vrcholu vysávača algoritmus **A**. Keď skončíme, všetky vrcholy okrem možno niektorých vrcholov vysávača budú mať súčet prislúchajúcich hrán 1. Teraz nám už iba stačí do grafu pridať hrany vysávača a nastaviť im vhodné hodnoty.

A čo ak nemáme vysávač? Ak graf nemá vysávač, tak je potom bipartitný. Graf nazývame bipartitný, ak jeho vrcholy vieme rozdeliť na dve časti tak, že všetky hrany vedú len medzi vrcholmi z rôznych častí (týmto častiam budeme hovoriť partície). Keď zoberieme súčet ohodnotení všetkých hrán, tak ten bude rovný súčtu hodnôt vo vrcholov v nejakej partícií (a samozrejme rovnaký ako aj v druhej), lebo každá hrana je zarátaná raz v jednej partícií a raz v druhej. Ale každý vrchol má mať súčet jedna, takže ak majú obe časti rôzne počty vrcholov, tak nie je možné hrany vhodne ohodnotiť. Naopak, ak majú obe časti rovnaký počet vrcholov, tak by to mohli ísť. Ako? Vyberme si ľubovoľný vrchol (nazvime ho v) a spustime z neho algoritmus **A**. Všetky vrcholy okrem vrchola v budú mať súčet prislúchajúcich hrán 1. Ako to bude v tom našom vrchole v? Tam musí byť tiež jedna, lebo obidve partície bipartitného grafu majú rovnaký počet vrcholov a aj rovnaký súčet.

Ako je to s časovou zložitosťou? Celý algoritmus je iba variácia na prehľadávanie do hĺbky, ktoré má časovú zložitosť $O(N+M)$ za predpokladu, že sme zvolili vhodnú štruktúru na pamätanie grafu a to konkrétne pre každý vrchol zoznam susedov. Z toho tiež vidno, aká asi bude pamäťová zložitosť. Tá je $O(N+M)$.

Implementačný detail: Všimnite si, že celé toto by vedelo fungovať v celých číslach, nebyť jediného delenia dvoma (naozaj si všimnite, že delíme dvoma práve raz). Preto sa nebudeme snažiť a súčet 1 v každom vrchole, ale o súčet 2. Vtedy budeme deliť dvoma párne číslo, takže na celý výpočet nám budú stačiť iba celé čísla. Na konci hodnotu každej hrany vydelíme dvoma.

[6] cyklu nepárnej dĺžky

[7] Skúste si rozmyslieť, čo by sa stalo, ak by k bolo párne

Listing programu:

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

struct Vrchol{
    int bol;
    int hodnota;
    vector<int> susedia;
    vector<int> ch;
    Vrchol(){
        bol=hodnota=0;
    }
};

vector<int> H;
vector<Vrchol> G;

vector<pair<int,int> > cyklus;
int A[3];
bool end;

bool najdi_neparny_cyklus(int v,int f){
    if(G[v].bol!=0){
        if(f==G[v].bol)return false;
        G[v].bol=3;
        return true;
    }
    A[f]++;
    G[v].bol=f;
    for(unsigned int i=0;i<G[v].susedia.size();i++){
        if(najdi_neparny_cyklus(G[v].susedia[i],f%2+1)){
            if(end)return true;
            cyklus.push_back(make_pair(v,i));
            if(G[v].bol==3)end=true;
    G[v].bol=3;
            return true;
        }
    }
    return false;
}

int ohodnot(int v){
    if(G[v].bol==3)return 0;
    if(G[v].bol>3)return 2-G[v].hodnota;
    G[v].bol=4;
    for(unsigned int i=0;i<G[v].susedia.size();i++){
        int ret=ohodnot(G[v].susedia[i]);
        H[G[v].ch[i]]+=ret;
        G[G[v].susedia[i]].hodnota+=ret;
        G[v].hodnota+=ret;
    }
    return 2-G[v].hodnota;
}

bool ide_to(int v){
    A[1]=A[2]=0;
    end=false;
    if(!najdi_neparny_cyklus(v,1)){
```

```
        if(A[1]!=A[2])return false;
        ohodnot(v);
        return true;
    }else{
        for(unsigned int i=0;i<cyklus.size();i++){
            G[cyklus[i].first].bol=2;
            ohodnot(cyklus[i].first);
            G[cyklus[i].first].bol=3;
        }
        vector<int> chcem(cyklus.size());
        for(unsigned int i=0;i<cyklus.size();i++)
            chcem[i]=2-G[cyklus[i].first].hodnota;
        int last=cyklus.size()-1;
        int hodnota=chcem[last];
        int znam=-1;
        for(int i=0;i<last;i++){
            hodnota+=znam*chcem[i];
            znam*=-1;
        }
        hodnota/=2;
        H[G[cyklus[last].first].ch[cyklus[last].second]]=hodnota;
        for(int i=last-1;i>=0;i--){
            hodnota=chcem[i]-hodnota;
            H[G[cyklus[i].first].ch[cyklus[i].second]]=hodnota;
        }
        return true;
    }
}

int main(){
    int N,M;
    cin >>N>>M;
    H.resize(M,0);
    G.resize(N);
    for(int i=0;i<M;i++){
        int a,b;
        cin>>a>>b;  a--;b--;
        G[a].susedia.push_back(b);
        G[a].ch.push_back(i);
        G[b].susedia.push_back(a);
        G[b].ch.push_back(i);
    }
    for(int i=0;i<N;i++){
        if(G[i].bol==0&&!ide_to(i)){
            cout<<"Všetci sa uvaríme!\n";
            return 0;
        }
    }
    for(int i=0;i<M;i++){
        cout<<H[i]/2;
        if(H[i]%2==1)cout<<".5";
        cout<<endl;
    }
    return 0;
}
```

Výsledková listina po 2. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda		1	2	3	4	5	Σ
1	Fulla Peter	SPŠ Spišská Nová Ves	2	75	15	13	15	15	16	149
2	Ondrúška Peter	SPŠ Dubnica nad Váhom	3	75	15	13	15	15	15	148
3	Kostolányi Peter	Gym. Jura Hronca BA	3	70	15	15	15	15	15	145
4	Bačo Ladislav	Gym. Poštová Košice	2011	64	15	10	12	10	15	126
5	Kudláč Matúš	Gym. Bilíkova BA	3	49	15	11	15	15	15	120
6	Fekiač Jozef	Gym. Grösslingová BA	2	58	15	12	15	9	7	116
7	Herencsár Albert	Gym. maď. Galanta	2	57	15	7	12	15	3	109
8	Hagara Michal	Gym. Jura Hronca BA	1	40	15	11	15	12	15	108
9	Hozza Michal	Gym. Jura Hronca BA	2	55	15	2	5	14	15	106
10	Kušnier Juraj	Gym. Bánovce nad Bebravou	3	55	15	11	9	9		99
11	Jenis Jakub	Gym. Cyrila a Metoda Nitra	3	41	14	12	15	14		96
12	Hojčka Michal	Gym. Partizánske	2	49	15	6	15	10		95
13	Matula Peter	Gym. Školská Turč. Teplice	3	45	15	4	5	10	11	90
13	Polák Lukáš	Gym. Jura Hronca BA	1	63	10		5	12		90
15	Balogh Tomáš	SPŠE Nové Zámky	2	42	15	9	8	15		89
15	Stachová Paula	Gym. Bilíkova BA	3	41	15	9	9	10	5	89
17	Tureková Katarína	Gym. Tajovského B. Bystrica	3	44	15	13	14			86
18	Konečný Jakub	Gym. Grösslingová BA	1	51	15	3	3	8	4	84
18	Starovská Mária	Gym. Grösslingová BA	3	45	15		15	9		84
20	Kuzma Tomáš	Gym. Alejová Košice	2	56		8	9	10		83
21	Kučin Alexander	Gym. Lipany	3	37	13		5	10	15	80
22	Halabuk Milan	Gymnázium Levice	4	47	13	3	9	7		79
22	Kikta Michal	Gym. Tatarku Poprad	2	35	15	2	9	15	3	79
24	Boško Lukáš	Gymnázium Považská Bystrica	2	44	13	4	5	10		76
25	Matejovičová Lenka	Gym. Grösslingová BA	3	58	15					73
26	Kohút Matej	Gym. Jura Hronca BA	2	32	15	10		15		72
26	Mikláš Ján	Gym. P. de Coubertina Piešťany	2	34	15	5	9	9		72
28	Kalugerov Samuel	Gym. Jura Hronca BA	2011	27	15	3	15	10		70
29	Boška Michal	Gym. Jura Hronca BA	3	40	14				15	69
30	Fačkovec Boris	Gym. P. de Coubertina Piešťany	4	66						66
31	Uherčík Maroš	Gym. P. de Coubertina Piešťany	2	35	15		3	9	1	63
32	Meravý Ján	SPŠ Dubnica nad Váhom	1	0	15	11	12	15	9	62
33	Kvasnička Igor	Gym. Jura Hronca BA	3	32		4	5		15	56
34	Chrást Lukáš	Gym. Golianova Nitra	2	19	14	12		9		54
35	Šutý Karol	Gym. Jura Hronca BA	3	18	15		9	10		52
36	Vaňo Jakub	Gymnázium J.A. Raymana	2	50						50
37	Fecko Stanislav	Gym. Pankúchova Bratislava	4	49						49
38	Bara Martin	SPŠE Nové Zámky	3	0	15	12	9	10		46
38	Rigdová Emília	Gym. Popradské nábr. Poprad	1	22	13		3	8		46
40	Košdy Ivan	Gym. Jura Hronca BA	2	44						44
41	Goga Rudolf	Gym. Jura Hronca BA	2011	43						43
42	Truben Martin	Gym. Jura Hronca BA	3	23	15	2		2		42
43	Štefanišin Peter	Gym. Svidník	4	41						41
44	Bachratý Martin	Gym. Veľká Okružná Žilina	2	40						40
44	Štrbka Dávid	Gym. Grösslingová BA	3	40						40
46	Janočko Ján	Gym. Jura Hronca BA	2	7	13	10		8		38
47	Sabo Michal	Gym. Jura Hronca BA	3	37						37
48	Meravý Jan	SPŠ Dubnica nad Váhom	3	36						36
49	Dittrich Andrej	Gym. Jura Hronca BA	3	35						35
49	Hozzánková Hana	Gym. Jura Hronca BA	1	35						35

	Meno a priezvisko	Škola	Trieda		1	2	3	4	5	Σ
51	Behún Marek	Gym. Ľ. Štúra Michalovce	1	34						34
52	Magdolen Matej	Gym. Golianova Nitra	1	31						31
53	Stančiaková Katarína	Gym. Jura Hronca BA	1	26		3				29
54	Baxová Katarína	Gym. Ľudovíta Štúra Trenčín	2	14	13					27
55	Tatara Tomáš	Gym. Jura Hronca BA	3	25						25
56	Kuchyňár Martin	Gym. Jura Hronca BA	1	22						22
57	Táňa Tóthová	Gym. Jura Hronca BA	1	18						18
58	Šebeňová Petra	Gym. Jura Hronca BA	1	17						17
59	Chudý Lukáš	Gym. Liptovský Hrádok	2	16						16
59	Molčan Dávid	Gym. Slovenská Bardejov	4	16						16
59	Šulák Viktor	SPŠE Nové Zámky	2	16						16
62	Končok Jakub	Gym. Haličská Lučenec	3	15						15
62	Piter Miroslav	Gym. Svidník	2011	15						15
62	Poláková Veronika	Gym. Jura Hronca BA	1	7			8			15
65	Sucha Tomáš	Gym. Jura Hronca BA	2	0	14					14
66	Dižová Andrea	Gym. Partizánske	0	0	13					13
66	Jančigová Jarmila	Gym. Jura Hronca BA	3	13						13
68	Korenek Roman	Gym. Jura Hronca BA	2	11						11
69	Buchovecká Simona	Gym. Medzilaborce	4	10						10
69	Husár Milan	Gym. Jura Hronca BA	2011	10						10
71	Szabadosová Emília	Škola pre mim. nadané deti BA	3	9						9
72	Sevela Richard	Gym. Jura Hronca BA	1	2		1				3
73	Balogh Imrich	Gym. Jura Hronca BA	1	2						2
74	Zdechovan Lukáš	Gym. Tajovského B. Bystrica	4	0						0

Výsledková listina po 2. kole kategórie KSP-O

	Meno a priezvisko	Škola	Trieda		4	5	6	7	8	Σ
1	Fulla Peter	SPŠ Spišská Nová Ves	2	70	15	16	15	12	15	143
2	Ondrúška Peter	SPŠ Dubnica nad Váhom	3	68	15	15	13	12	12	135
3	Boža Vladimír	Gym. Tatarku Poprad	3	60	15	5	15	12	15	122
4	Nagy Kristián	Gym. Jura Hronca BA	3	58	13	15	8	12	9	115
5	Kostolányi Peter	Gym. Jura Hronca BA	3	52	15	15	5	12	15	114
6	Kekely Lukáš	Gym. Varšavská Žilina - Vlčince	3	51	15	16	8	12	11	113
7	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	3	57	15		11	12	11	106
8	Hapák Samuel	Gym. Grösslingová BA	3	74		13			15	102
9	Varga András	Gym. H. Selyeho Komárno	3	49	15		8	12	9	93
10	Simanová Lucia	Gym. Grösslingová BA	3	44	15	3	8	12	1	83
11	Fedáková Dominika	Gym. Stará Ľubovňa	3	34	12	5	8	11	9	79
11	Kudláč Matúš	Gym. Bilíkova BA	3	33	15	15	8	8		79
13	Súkeník Ján	Gym. Lettricha Martin	3	28	14	15	8	11	1	77
14	Hlavatý Peter	Gym. Jura Hronca BA	3	68						68
15	Kočiský Tomáš	Gym. Grösslingová BA	3	30	10	5		11	7	63
16	Košinárová Alena	Gym. Grösslingová BA	3	59						59
16	Vojtko Jakub	Gym. Jura Hronca BA	2	44	15					59
18	Hozza Michal	Gym. Jura Hronca BA	2	14	14	15		12		55
18	Sucha Martin	Gym. Jura Hronca BA	3	27	15			13		55
20	Bačo Ladislav	Gym. Poštová Košice	2011	24	10	15				49
20	Kvasnička Igor	Gym. Jura Hronca BA	3	23		15		11		49
22	Herencsár Albert	Gym. maď. Galanta	2	26	15	3				44
23	Starovská Mária	Gym. Grösslingová BA	3	22	9			12		43
24	Petrucha Michal	Gym. Metodova BA	2	27	15					42
25	Hagara Michal	Gym. Jura Hronca BA	1	14	12	15			0	41
26	Fačkovec Boris	Gym. P. de Coubertina Piešťany	4	32					5	37
27	Fekiač Jozef	Gym. Grösslingová BA	2	20	9	7				36
27	Kuzma Tomáš	Gym. Alejová Košice	2	26	10					36
27	Meravý Ján	SPŠ Dubnica nad Váhom	1	0	15	9		12	0	36
30	Matula Peter	Gym. Školská Turč. Teplice	3	13	10	11				34
30	Polák Lukáš	Gym. Jura Hronca BA	1	22	12					34
30	Stachová Paula	Gym. Bilíkova BA	3	19	10	5				34
33	Kučin Alexander	Gym. Lipany	3	8	10	15				33
33	Kušnier Juraj	Gym. Bánovce nad Bebravou	3	24	9					33
35	Hojčka Michal	Gym. Partizánske	2	21	10					31
35	Miňo Igor	Gym. Sobrance	3	22	8			1		31
37	Kalugerov Samuel	Gym. Jura Hronca BA	2011	20	10					30
38	Boška Michal	Gym. Jura Hronca BA	3	13		15				28
39	Balogh Tomáš	SPŠE Nové Zámky	2	12	15					27
39	Jenis Jakub	Gym. Cyrila a Metoda Nitra	3	13	14					27
39	Uherčík Maroš	Gym. P. de Coubertina Piešťany	2	17	9	1				27
42	Halabuk Milan	Gymnázium Levice	4	19	7					26
42	Konečný Jakub	Gym. Grösslingová BA	1	14	8	4			0	26
42	Mikláš Ján	Gym. P. de Coubertina Piešťany	2	17	9					26
45	Korcsok Peter	Gym. Mládežnícka Šahy	3	0	10		8		7	25
45	Kukan Matúš	Gym. Grösslingová BA	3	25						25
47	Boško Lukáš	Gymnázium Považská Bystrica	2	14	10					24
48	Fecko Stanislav	Gym. Pankúchova Bratislava	4	23						23
48	Saleh Adam	Gym. Jura Hronca BA	3	0	15			8		23
50	Kohút Matej	Gym. Jura Hronca BA	2	7	15					22

	Meno a priezvisko	Škola	Trieda		4	5	6	7	8	Σ
51	Matejovičová Lenka	Gym. Grösslingová BA	3	21						21
52	Meravý Jan	SPŠ Dubnica nad Váhom	3	20						20
52	Stančiaková Katarína	Gym. Jura Hronca BA	1	20						20
54	Kikta Michal	Gym. Tatarku Poprad	2	0	15	3				18
54	Magyar Vladimír	SPŠE Nové Zámky	3	18						18
56	Behún Marek	Gym. Ľ. Štúra Michalovce	1	15						15
56	Košdy Ivan	Gym. Jura Hronca BA	2	15						15
56	Rampášek Ladislav	Gym. Jura Hronca BA	4	15						15
56	Tureková Katarína	Gym. Tajovského B. Bystrica	3	15						15
60	Bachratý Martin	Gym. Veľká Okružná Žilina	2	14						14
61	Goga Rudolf	Gym. Jura Hronca BA	2011	13						13
61	Vaňo Jakub	Gymnázium J.A. Raymana	2	13						13
63	Hozzánková Hana	Gym. Jura Hronca BA	1	12						12
63	Šebeňová Petra	Gym. Jura Hronca BA	1	12						12
63	Táňa Tóthová	Gym. Jura Hronca BA	1	12						12
66	Janočko Ján	Gym. Jura Hronca BA	2	3	8					11
66	Končok Jakub	Gym. Haličská Lučenec	3	11						11
66	Štrbka Dávid	Gym. Grösslingová BA	3	11						11
69	Bara Martin	SPŠE Nové Zámky	3	0	10					10
69	Chudý Lukáš	Gym. Liptovský Hrádok	2	10						10
69	Jančigová Jarmila	Gym. Jura Hronca BA	3	10						10
69	Sabo Michal	Gym. Jura Hronca BA	3	10						10
69	Šutý Karol	Gym. Jura Hronca BA	3	0	10					10
69	Truben Martin	Gym. Jura Hronca BA	3	5	2		3			10
75	Chrást Lukáš	Gym. Golianova Nitra	2	0	9					9
75	Magdolen Matej	Gym. Golianova Nitra	1	9						9
77	Rigdová Emília	Gym. Popradské nábr. Poprad	1	0	8					8
78	Štefanišin Peter	Gym. Svidník	4	6						6
79	Dittrich Andrej	Gym. Jura Hronca BA	3	5						5
80	Kuchyňár Martin	Gym. Jura Hronca BA	1	4						4
81	Bundala Daniel	Gym. Jura Hronca BA	5	0		3				3
81	Husár Milan	Gym. Jura Hronca BA	2011	3						3
81	Nagy Anton	Gym. maďarské BA	4	0		3				3
81	Sevela Richard	Gym. Jura Hronca BA	1	3						3
81	Szabadosová Emília	Škola pre mim. nadané deti BA	3	3						3
86	Balogh Imrich	Gym. Jura Hronca BA	1	2						2
87	Tatara Tomáš	Gym. Jura Hronca BA	3	1						1
88	Šulák Viktor	SPŠE Nové Zámky	2	0						0
88	Zdechovan Lukáš	Gym. Tajovského B. Bystrica	4	0						0

Výsledková listina po 2. kole kategórie KSP-T

	Meno a priezvisko	Škola	Trieda		8	9	10	Σ
1	Hapák Samuel	Gym. Grösslingová BA	3	34	15	13	11	73
2	Ondrúška Peter	SPŠ Dubnica nad Váhom	3	28	12	14	11	65
3	Boža Vladimír	Gym. Tatarku Poprad	3	23	15	15		53
4	Kostolányi Peter	Gym. Jura Hronca BA	3	24	15	7		46
5	Fulla Peter	SPŠ Spišská Nová Ves	2	10	15	13		38
6	Hlavatý Peter	Gym. Jura Hronca BA	3	32				32
7	Fačkovec Boris	Gym. P. de Coubertina Piešťany	4	17	5	2		24
8	Kudláč Matúš	Gym. Bilíkova BA	3	19			0	19
9	Nagy Kristián	Gym. Jura Hronca BA	3	9	9			18
10	Varga András	Gym. H. Selyeho Komárno	3	7	9			16
11	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	3	4	11			15
12	Kekely Lukáš	Gym. Varšavská Žilina - Vlčince	3	0	11			11
13	Fedáková Dominika	Gym. Stará Ľubovňa	3	0	9			9
13	Kvasnička Igor	Gym. Jura Hronca BA	3	9				9
15	Kočiský Tomáš	Gym. Grösslingová BA	3	0	7			7
15	Korcsok Peter	Gym. Mládežnícka Šahy	3	0	7			7
15	Šutý Karol	Gym. Jura Hronca BA	3	7				7
18	Košinárová Alena	Gym. Grösslingová BA	3	5				5
18	Meravý Jan	SPŠ Dubnica nad Váhom	3	5				5
20	Súkeník Ján	Gym. Lettricha Martin	3	3	1			4
21	Simanová Lucia	Gym. Grösslingová BA	3	2	1			3
22	Meravý Ján	SPŠ Dubnica nad Váhom	1	0	0	2		2
22	Miňo Igor	Gym. Sobrance	3	2				2
24	Tatara Tomáš	Gym. Jura Hronca BA	3	1				1
25	Hagara Michal	Gym. Jura Hronca BA	1	0	0	0	0	0
25	Konečný Jakub	Gym. Grösslingová BA	1	0	0	0	0	0
25	Szabadosová Emília	Škola pre mim. nadané deti BA	3	0				0