Korešpondenčný seminár z programovania
XXI. ročník, 2003/04
Katedra vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporujú: aSc – Applied Software Consultants spol. s r.o.
MICROSTEP-MIS spol. s r.o.

Vzorové riešenia 1. kola letnej časti

Milí riešitelia!
Jarné sústredenie máme úspešne za sebou a Vám sa dostávajú do rúk vzorové riešenia prvého kola. Pozorne si ich prečítajte, aby ste vedeli, ako sa tie úlohy mali riešiť. Ešte nie je nič stratené. Do jesenného sústredka je ešte veľa času a jedno kolo KSP. Poučený týmito riešeniami sa ešte každý z Vás môže na sústredko dostať.

Ráno býva múdrejšie večerať.

KSPáci

1. O matematikoch

opravoval Dávidko
(max. 15 bodov)

Väčšina riešení mala optimálnu časovú zložitosť $O(N^2 \log N)$, ale len Lukáš Poláček mal aj optimálnu pamäťovú zložitosť $O(N)$. Za jeho riešenie, ktoré je rovnaké ako vzorové, dostal zaslúžene plný počet bodov. Za kvadratickú pamäť som strhal 2 body (teda väčšina mala len 13 bodov), za chýbajúci popis a odhady zložitostí po bode. Pomalé riešenia v čase $O(N^3)$ dostali maximálne 10 bodov, $O(N^4)$ max 9 bodov.

Vzorové riešenie

Základná myšlienka je jednoduchá: Rovnosť $a + b + c = d$ platí práve vtedy, keď platí rovnosť $a + b = d - c$. Riešme teda tú druhú rovnicu. Na jej ľavej strane môže stáť súčet ľuvovoľných dvoch prvkov a na pravej strane môže stáť rozdiel ľubovoľných dvoch prvkov. Takže vygenerujeme množinu L všetkých súčtov dvojíc prvkov a množinu P všetkých rozdielov dvojíc prvkov. Rovnica má teda riešenie vtedy a len vtedy, ak L a P majú neprázdny prienik.

Prienik L a P sa hľadá ľahko. Najprv obe množiny utriedime od najmenšieho po najväčší prvok. Ak sa najmenšie prvky množín rovnajú, tak sme našli prienik. Ak nie, tak môžeme menší z najmeších prvkov odobrať z tej ktorej množiny a pokračovať ďalej.

Tento algoritmus je skoro ideálny, ale jeho nedostatok je v tom, že množiny L a P môžu mať až N^2 prvkov a zaberali by nám v pamäti príliš veľa miesta. Tomu sa dá vyhnúť. Stačí vedieť množiny L a P *generovať* v rastúcom poradí. To urobíme pomocu dvoch háld, pre každú množinu jednu.

Najprv si utrieďme všetky prvky a označme si ich $a_1 < a_2 < \ldots < a_N$. Evidetne platia nerovnosti

$$a_1 + a_1 < a_1 + a_2 < a_1 + a_3 < \ldots < a_1 + a_N,$$

$$a_2 + a_1 < a_2 + a_2 < a_2 + a_3 < \ldots < a_2 + a_N,$$

$$a_3 + a_1 < a_3 + a_2 < a_3 + a_3 < \ldots < a_3 + a_N,$$

$$\vdots$$

$$a_N + a_1 < a_N + a_2 < a_N + a_3 < \ldots < a_N + a_N.$$

V týchto nerovnostiach je všetkých N^2 súčtov dvojíc prvkov, t.j. všetky čísla množiny L. V halde si budeme v každej chvíli pamätať z každého riadku nerovností práve jeden súčet. Na začiatku tam dáme prvý súčet z každého riadku. Keď z haldy odoberieme nejaký súčet, povedzme súčet $a_i + a_j$, tak hneď do haldy prihodíme jeho nasledovníka v riadku t.j. súčet $a_i + a_{j+1}$. (Ak $j = N$, tak pochopiteľne už nie je čo prihodiť.)

Pre množinu rozdielov P, to bude obdobne, pričom vychádzame z týchto nerovností:

$$a_1 - a_N < a_1 - a_{N-1} < a_1 - a_{N-2} < \ldots < a_1 - a_1,$$

$$a_2 - a_N < a_2 - a_{N-1} < a_2 - a_{N-2} < \ldots < a_2 - a_1,$$

$$a_3 - a_N < a_3 - a_{N-1} < a_3 - a_{N-2} < \ldots < a_3 - a_1,$$

$$\vdots$$

$$a_N - a_N < a_N - a_{N-1} < a_N - a_{N-2} < \ldots < a_N - a_1.$$

Takže po inicializácii háld L a P, budeme z nich len odoberať menší z ich najmenších prvkov. A namiesto neho prihadzovať vždy ďalší prvok z riadku nerovností. Ak nebodaj nastane rovnosť dvoch najmenších prvkov, tak sme našli prienik a teda riešenie rovnice.

Úvodné triedenie môžeme urobiť hoci aj pomaly v čase $O(N^2)$. Vloženie alebo vybratie z haldy trvá $O(\log N)$, to budeme robiť pre každý zo súčtov/rozdielov maximálne jedno, a preto celková časová zložitosť bude $O(N^2 \log N)$. Pamäťová zložitosť je evidentne len lineárna t.j. $O(N)$, lebo v každej z oboch háld bude vždy nanajvýš N prvkov.

Listing programu:

```
Program O_matematikoch;

type pair = record sum,x,y:integer; end;

var a:array[1..1000] of integer;  { vstupna mnozina cisel }
    L,P:array[1..1000] of pair;   { halda cisel pre lavu a halda pre pravu stranu }
    N,i,j,tmp:integer;
    hp,hl:integer;                { pocty prvkov v haldach }

{ bublanie prvku v halde dole }
procedure heap_down(i,max:integer; var H:array of pair);
var syn:integer; p:pair;
begin
    syn:=2*i;
    while syn<=max do
    begin
        if (syn+1<=max) and (H[syn+1].sum<H[syn].sum) then inc(syn); { mensi zo
synov }

        if H[syn].sum<H[i].sum then { ak je syn mensi ako otec }
        begin
            p:=H[syn];
            H[syn]:=H[i];
            H[i]:=p;

            i:=syn;
            syn:=2*i;
        end
        else break;
    end;
end;
```

```
begin
    assign(input, 'prikl1.in');
    reset(input);

    { nacitame vstup }
    readln(N);
    for i:=1 to N do read(a[i]);

    { utriedime cisla }
    for i:=1 to N do
    for j:=i+1 to N do
    if a[i]>a[j] then
    begin
        tmp:=a[i];
        a[i]:=a[j];
        a[j]:=tmp;
    end;

    { vlozime do haldy prvy sucet/rozdiel z kazdeho riadku }
    for i:=N downto 1 do
    begin
        L[i].x:=i;                  { lava strana }
        L[i].y:=1;
        L[i].sum:=a[i]+a[1];

        P[i].x:=i;                  { prava strana }
        P[i].y:=N;
        P[i].sum:=a[i]-a[N];
    end;

    hp:=N; hl:=N; { pocet prvkov v haldach }

    { pokial je v oboch haldach nieco }
    while (hp>0) and (hl>0) do

        if L[1].sum < P[1].sum then
        begin
            if L[1].y=N then
            begin
                L[1]:=L[hl]; { ak sme uz na konci riadku }
                dec(hl);
            end else
            begin
                inc(L[1].y);
                L[1].sum:=a[L[1].x]+a[L[1].y];
            end;

            heap_down(1,hl,L);
        end

        else if P[1].sum < L[1].sum then
        begin
            if P[1].y=1 then
            begin
                P[1]:=P[hp]; { ak sme uz na konci riadku }
                dec(hp);
            end else
            begin
                dec(P[1].y);
```

```
            P[1].sum:=a[P[1].x]-a[P[1].y];
        end;

        heap_down(1,hp,P);
    end

    else { L[1].sum=P[1].sum }
    begin
        { nasli sme prienik L a P }
        writeln(a[L[1].x], ' + ', a[L[1].y], ' + ', a[P[1].y], ' = ', a[P[1].x]);
        halt;
    end;

end.
```

2. Oooooooo Bisťu Blabla Bum

opravoval Palo
(max. 15 bodov)

Prišlé riešenia sa dali väčšinou zaškatuľkovať do dvoch skupín – riešenia s časovou zložitosťou $O(N)$ a riešenia s časovou zložitosťou $O(N^2)$. Za riešenia z prvej skupiny sa dalo získať maximálne 15 bodov, za riešenia z druhej skupiny maximálne 10 bodov. Body ste hlavne stratili za nedostatočné zdôvodnenie správnosti vášho programu, za nefunkčné alebo chýbajúce programy alebo za zlý odhad časovej zložitosti vášho programu.

Ukážeme si riešenie v čase $O(N)$. Vieme, že križovatky mesta tvoria strom. Pozrime sa na ľubovolný list u - t.j. križovatku majúcu len jedného suseda. Označme jeho suseda v. Ukážeme, že nič nepokazíme ak dáme stánok na ulicu spájajúcu u a v. Pozrime sa na nejaké optimálne riešenie (také, ktoré rozmiestni v meste najviac stánkov). My v ňom poprehadzujeme pár stánkov tak, aby nejaký stánok stál na ulici z u do v a aby sme znovu dostali nejaké (možno) iné optimálne riešenie. V našom optimálnom riešení môžu nastať tieto prípady:

- V optimálnom riešení sa nenachádza žiadny stánok na ulici susediacej s v. V tomto prípade vieme položiť stánok na ulicu z u do v. Neporušíme tým žiadnu podmienku zo zadania, takže vieme rozmiestniť viac stánkov ako v optimálnom riešení, čo sa zjavne nedá. Teda tento prípad nemôže nastať.
- Na nejakej ulici susediacej s v sa stánok nachádza:
  - Stánok sa nachádza na ulici z u do v. V tomto prípade nie je čo riešiť.
  - Stánok sa nachádza na ulici z v do w, kde w je nejaký ďalší sused v. Odstráňme stánok z tejto ulice a postavme ho na ulici z u do v. Týmto zjavne neporušíme žiadnu podmienku zo zadania a dostali sme rozmiestnenie stánkov, ktoré rozmiestni rovnaký počet stánkov ako optimálne riešenie. Teda sme dostali optimálne riešenie, v ktorom stánok stojí na ulici z u do v.

Takže náš algoritmus bude fungovať nasledovne: Nájdeme si nejaký list. Postavíme stánok na ulici spájajúcej tento list so zvyškom stromu (mesta) a odstránime všetky s ňou susediace ulice z mesta. Ostane nám nejaké menšie mesto. Tam musíme zjavne stánky tiež rozmiestniť optimálne. Takže znovu zopakujeme ten istý postup pre toto menšie mesto atď.

Jedna z možných efektívnych implementácií tohoto algoritmu využíva prehľadávanie do hĺbky (DFS). Pre každú križovatku si budeme pamätať zoznam križovatiek, ktoré s ňou susedia a aj ich počet. Zavolajme prehľadávanie do hĺbky z prvej križovatky. Vždy, keď prídeme až do listu, vypíšeme ulicu spájajúcu tento list so zvyškom stromu. Aby sme efektívne implementovali odstránenie susediacich ulíc, vráti funkcia DFS volaná na nejaký vrchol x, či sme umiestnili stánok na nejakú ulicu susediacu s x.

Konkrétne volanie DFS na nejakom vrchole x bude vyzerať nasledovne: Zavoláme sa postupne rekurzívne na všetkých susedov x, okrem toho, z ktorého sme do x prišli. Po

zavolaní sa na y – nejakého suseda križovatky x – sa pozrieme či sme dali stánok na nejakú ulicu susediacu s y.

- Ak nie, pokojne môžme dať stánok na ulicu z x do y a zapamätáme si, že už sme stánok na ulicu susediacu s x dali a ďalší už na ulicu susediacu s x nedáme.
- Ak áno, tak na ulicu z x do y dať stánok nemôžeme a ani ho tam nedáme.

Nakoniec ešte vrátime, či sme vôbec nejaký stánok umiestnili na ulicu susediacu s x.

Časová aj pamäťová zložitosť sú zjavne $O(N + M)$, kde N je počet križovatiek a M je počet ulíc. Vieme, že $M = N - 1$, teda obidve zložitosti sú $O(N)$.

Listing programu:

```
var
   a,b,i: integer;
   n, pocst: integer;
   mes: array[1..100, 1..100] of integer;
   pocsus: array[1..100] of integer;
   bol: array[1..100] of boolean;

function dfs(kde: integer): boolean;
var
   i: integer;
   dalsom: boolean;
begin
   bol[kde] := true;
   dalsom := false;

   for i := 1 to pocsus[kde] do begin
      if not bol[mes[kde][i]] then begin
         if not dfs(mes[kde][i]) then begin
            if not dalsom then begin
               writeln(kde, '-', mes[kde][i]);
               dalsom := true;
               inc(pocst);
            end;
         end;
      end;
   end;

   dfs := dalsom;
end;

begin
   pocst := 0;
   read(n);
   for i := 1 to n do begin
      pocsus[i] := 0;
      bol[i] := false;
   end;
   for i := 1 to n - 1 do begin
      read(a,b);
      inc(pocsus[a]);
      mes[a][pocsus[a]] := b;
      inc(pocsus[b]);
      mes[b][pocsus[b]] := a;
   end;

   writeln('Stanky treba dat na ulice:');
   dfs(1);
   writeln('Dokopy sa da umiestnit ', pocst, ' stankov.');
```

end.

3. O pánoch Hamiltonovi a di Rakovi

opravoval Goober
(max. 15 bodov)

Tento príklad ma dosť potešil, väčšina riešení bola správna, a preto bodovanie záviselo hlavne od časovej zložitosti. Riešenia so zložitosťou $O(N^2)$ si tak získali plných **15 bodov**, o niečo horšia zložitosť si zaslúžila **11 bodov**, za backtrackoidné riešenia bolo **8 bodov** a nefunkčným riešeniam boli udelené nanajvýš **3 body**. No a ako už býva zvykom, nedostatočné popisy, chýbajúce odhady a nedostatočné dôkazy boli patrične ohodnotené nejakým záporným počtom bodíkov.

Ako sa to dalo riešiť? Najprv si úlohu preložíme do reči grafov. Vrcholmi nášho grafu budú ostrovy, hrany budú zodpovedať dvojiciam ostrovov spojených člnovou dopravou. V tomto grafe máme nájsť Hamiltonovskú kružnicu[1]. Zo zadania vieme, že stupeň každého vrchola je väčší ako $\frac{N}{2}$ a ako tvrdí pán di Rak[2], v takomto grafe Hamiltonovská kružnica vždy existuje. Lenže my sa nemôžeme spoliehať na nejakého pána di Raka, a tak to radšej poriadne dokážeme. Keď treba dokázať, že niečo existuje, zvyčajne býva najjednoduchšie ukázať, ako danú vec nájsť.

Najprv budeme hľadať čo najdlhšiu cestu. Začneme cestou pozostávajúcou len z jediného vrcholu a budeme ju postupne predlžovať (skončíme až keď bude obsahovať všetky vrcholy). Túto cestu budeme chápať ako orientovanú – t.j. budeme vedieť povedať, kde má „začiatok“ a „koniec“, a budeme vedieť spraviť „krok v smere cesty“ (aj keď samotný graf nie je orientovaný). Najjednoduchší spôsob na predĺženie cesty je pripojenie ďalšieho vrcholu za aktuálny koniec. Ak sa dostaneme do stavu, že to už nie je možné spraviť (teda všetky vrcholy susediace s koncom cesty už na ceste ležia), skúsime nasledovný fígeľ – cestu vhodne „rozstrihneme“ na dve časti a troška inak ich zlepíme, pričom chceme, aby sme ju potom už vedeli predĺžiť o nový vrchol.

Označme X množinu všetkých vrcholov susediacich s koncom. Ako bolo povedané, všetky vrcholy z X musia ležať na doterajšej ceste. Teraz ich všetky „posunieme“ v smere cesty o jeden krok a vzniknutú množinu nazveme X'. Vezmime si ľubovoľný vrchol y a označme Y množinu všetkých jeho susedov. Podľa zadania platí $|Y| > \frac{N}{2}$ aj $|X'| = |X| > \frac{N}{2}$, čiže $|Y|+|X'| > N$. Preto existuje nejaký vrchol z, ležiaci aj v Y aj v X'. Rozstrihnime teraz cestu tesne pred vrcholom z a časť, v ktorej je vrchol z, prilepme obrátene – t.j. ak pôvodná cesta mala tvar $a \ldots bz \ldots c$, spravíme z nej $a \ldots bc \ldots z$. Touto úpravou sme dosiahli, že koniec novej cesty susedí s y. Ak sme si y zvolili tak, že neležalo na aktuálnej ceste, po použití fígľa môžeme cestu predĺžiť, napríklad pridaním vrchola y.

Takýmto spôsobom teda dokážeme vyrobiť cestu idúcu cez všetky vrcholy. Lenže my potrebujeme kružnicu a nie len cestu. Ak začiatok a koniec našej cesty susedia, skončili sme. Inak použijeme spomínaný fígeľ, akurát si za y vezmeme prvý vrchol našej cesty[3]. Po jeho použití bude už koniec a začiatok cesty susediť, čiže dostaneme kružnicu. A to sme predsa chceli...

No a implementácia? Graf si pamätáme ako maticu susednosti (t.j. štvorcovú maticu $N \times N$, kde políčko na r-tom riadku a v s-tom stĺpci označuje, či vedie hrana medzi vrcholmi r a s), aktuálnu cestu ako zoznam vrcholov. Pamäťová náročnosť je očividne $O(N^2)$, keďže graf má $O(N^2)$ hrán. Časová zložitosť je taktiež $O(N^2)$, lebo jedno predĺženie cesty nám trvá $O(N)$ a predlžujeme $O(N)$-krát.

[1] To je kružnica, ktorá obsahuje všetky vrcholy

[2] Nepliesť s pánom Dirac-om, známym matematikom

[3] Samotný fígeľ nerobí nič s vybratým vrcholom y, takže na ceste sa nám nič nepokazí

Listing programu:

```
const MAX=100;
var g:array[1..MAX,1..MAX] of boolean;
    ces:array[1..MAX] of integer;
    bolo:array[1..MAX] of boolean;
    y, a, b, i, n:integer;

procedure figel(y, i:integer);
var j, k, t:integer;
begin
  for j:=1 to i-1 do if g[ces[j], ces[i]] and g[ces[j+1], y] then break;
  for k:=1 to (i-j) div 2 do begin
    t:=ces[j+k]; ces[j+k]:=ces[i+1-k]; ces[i+1-k]:=t;
  end;
end;

begin
  read(n);
  for a:=1 to n do for b:=1 to n do g[a, b]:=false;
  while not eof(input) do begin
    read(a,b); g[a, b]:=true; g[b, a]:=true;
  end;
  ces[1]:=1; bolo[1]:=true; for i:=2 to n do bolo[i]:=false;

  for i:=1 to n-1 do begin
    y:=1;
    while y<=n do if g[ces[i], y] and not bolo[y] then break else inc(y);
    if (y>n) then begin
      for y:=1 to n do if not bolo[y] then break;
      figel(y, i);
    end;
    ces[i+1]:=y; bolo[y]:=true;
  end;
  if not g[ces[1], ces[n]] then figel(1, n);

  for i:=1 to n do write(ces[i],' '); writeln;
end.
```

4. Objemné teleso

opravoval Žužu
(max. 15 bodov)

Vy ste ale neposlušné deti! Taká krásna úloha, a tak málo riešiteľov. Tí, z hŕstky odvážlivcov, čo si trúfli túto úlohu riešiť, budú odmenení bodíkmi. Vy ostatní môžete len závidieť. A všetci povinne prečítať vzorové riešenie, lebo inak beda!

Vzorové riešenie: Objem telesa budeme počítať postupne po vrstvách. Zo stien na vstupe vyberieme len steny rovnobežné s osami **xy**, teda tie ktorých všetký vrcholy majú rovnakú z-súradnicu. Steny usporiadame vzostupne podľa z-súradnice, čím dostaneme rozdelenie do n vrstiev $z_1, z_2, \ldots, z_n$ také, že vo vrstve z_i sa nachádzajú steny, ktorých z-súradnica je z_i. Steny budeme spracúvať po vrstvách, vždy steny v jednej vrstve naraz.

Jednotlivé vrstvy (hladiny s rovnakou z-súradnicou) nám zodpovedajú akýmsi rezom daného telesa. Teraz sa budeme snažiť určiť, aký prierez má teleso medzi jednotlivými vrstvami. Uvedomme si, že medzi dvoma susednými vrstvami z_i a z_{i+1} sa plocha prierezu S_i nemení, preto ak by sme vedeli určiť túto plochu, vedeli by sme spočítať V objem telesa ako:

$$V = \sum_{i=0}^{n-1} (z_{i+1} - z_i) * S_i$$

Plocha prierezu S_0 je rovná súčtu plôch mnohouholníkov vo vrstve z_0. Pozrime sa teraz na vrstvu z_1. Nejako sa tam prierez zmení, niečo pribudne, niečo ubudne. Skúsme si to všetko predstaviť v jednej rovine. Uvažujme mnohouholníky z vrstvy z_0 a mnohouholníky z vrstvy z_1 preložené cez seba. Na miestach, kde sa mnohouholníky prekrývajú, prierez „končí“ (nebude pokračovať do nasledujúcej vrstvy) a tam, kde sa neprekrývajú buď prierez „začína“ (z z_1) alebo „pokračuje“ (z z_0).

Rovnakú úvahu vieme zopakovať pre ľubovoľné vrstvy z_i a z_{i+1}. Postup výpočtu $S_i + 1$ z S_i spočíva teda v tom, že plochu S_i, ktorú sme vypočítali z vrstiev $z_0, \ldots, z_i$ preložíme mnohouholníkmi z vrstvy z_{i+1} a vzniknuté prekryvy odstránime, čím dostaneme plochu S_{i+1}.

Problémom teda zostáva, vymyslieť postup ako si udržiavať tvar prierezu tak, aby sme vedeli do prierezu mnohouholníky rýchlo pridávať s tým, že prekryvy odstránime.

Pre jednoduchosť zvolíme dátovú štruktúru **Quad-tree**. Quad-tree je štandardná štruktúra pre reprezentáciu plochy, založená na strome, v ktorom každý vnútorný vrchol má štyroch potomkov. Ku každému vrcholu v zodpovedá štvorec Q_v a jeho (štyrom) potomkom zodpovedajú štyri menšie štvorce, ktoré dostaneme rozdelením štvorca Q_v v strede v oboch smeroch (viď. obrázok).

Všimnime si, že všetkých môžných prierezov nemôže byť až tak veľa. Označme k_x počet rôznych x-ových súradníc vrcholov telesa. Podobne definujme k_y. Nech $k = 2^m, k \geq k_x, k_y$ a nech k je najmenšie také možné, potom nad plochou $(0 \ldots k) \times (0 \ldots k)$ vytvoríme úplný *Quad-tree* reprezentujúci prierez telesa (strom bude mať k^2 listov).

Mnohouholníky si znormalizujeme, teda x-ové a y-ové súradnice premietneme do intervalu $0 \ldots k$ tak, aby vzájomne nezmenili usporiadanie. Takto znormalizované mnohouholníky potom šikovne rozdelíme na obdĺžniky, ktoré už spracúvame nasledovnými operáciami *Quad-tree*:

- Pridať obdĺžnik do prierezu tak, že odstráni prekryvy s dovtedy pridanými obdĺžnikmi.
- Zistiť, obsah prierezu.

Týmto je myšlienka dokončená. Zostáva ešte popísať technické detaily.

Normalizáciu súradníc mnohouholníkov realizujeme vložením všetkých týchto súradníc do jedného poľa, utriedením, odstránením duplikátov a hodnotu v znormalizujeme tak, že ju vyhľadáme (binárnym vyhľadávaním) v utriedenom poli a normalizovaná hodnota bude jej index v tomto poli.

Rozdelenie mnohouholníkov na obdĺžniky vyžaduje premysleným spôsobom obiehať po obvode zadaný mnohouholník a šikovným spôsobom „urezávať“ obdĺžniky, až z mnohouholníka nič nezostane. Detaily si premyslite!

3cm

V každom vrchole *Quad-tree* si budeme pamätať tri čísla – $area_v$ - pseudo-pokrytú plochu pod vrcholom v, sum_v - celkovú plochu, ktorú vrchol symbolizuje, $flip_v$ - či je plocha invertovaná. Uvedomme si, že každý vrchol nám reprezentuje nejakú časť nášho prierezu (vlastnosť *Quad-tree*: vrchol v reprezentuje všetky svoje deti). Z týchto troch čísel vieme určiť aká časť plochy pod vrcholom v je naozaj pokrytá:

- Ak $flip_v = 0$, reálne pokrytá plocha má obsah $real_v = area_v$.
- Ak $flip_v = 1$, reálne pokrytá plocha má obsah $real_v = sum_v - area_v$.

Operáciu zistenie obsahu celého prierezu vieme vykonať spočítaním hodnoty $real_1$ pre vrchol 1 (koreň stromu).

Operáciu pridanie obdĺžnika s odstránením prekryvov vykonáme nasledovne. Zoberieme pridávaný obdĺžnik a rekurzívne ho pridáme do koreňa stromu. Ak sa obdĺžnik do vrcholu

v presne vtesná (žiadne voľné miesto), tak vo vrchole zmeníme hodnotu $flip_v = 1 - flip_v$ a vo vrcholoch na ceste smerom ku koreňu prepočítame hodnoty pseudo-plôch; inak obdĺžnik rozsekáme na menšie obdĺžničky a tie rekurzívne pridáme do príslušných detí vrcholu v. Opäť, detaily si rozmyslite!

Aká bude zložitosť celého tohto postupu? Označme N počet vrcholov telesa, Utriedenie stien do vrstiev, normalizácia súradníc vrcholov i sekanie stien na obdĺžniky sa všetko vykonáva v čase najviac $O(N \log N)$. Do nášho *Quad-tree* pridáme najviac $O(N)$ obdĺžnikov, každý v čase najviac $O(N)$, teda výsledná časová zložitosť bude $O(N^2)$. Premyslite si to!. Pamäťová bude rovnaká, pretože si potrebujeme pamätať strom, ktorý má $O(N^2)$ vrcholov.

Bodovanie: Vašich riešení prišlo poskromne. Všetky sa zhodli na tom, že steny na vstupe si najskôr rozdelíme do vrstiev a potom spočítame prierezy. Vaše počítanie ale vo väčšine prípadov nedopadlo dobre, takže sa o tom nebudem veľmi rozpisovať. Zoraďme si to podľa dosiahnutej zložitosti (N je počet vrcholov na vstupe):

- Riešenia lepšie ako vzorové riešenie[4] – **18 bodov**
- Vzorové riešenie v čase $O(N^2)$ – **15 bodov**
- Riešenia v čase $O(N^3)$ – **12 bodov**
- Ostatné funkčné riešenia – **8 bodov** a menej

Ďalšie 1-2 bodíky ste mohli získať alebo stratiť na základe môjho subjektívneho dojmu, tak sa nechajte prekvapiť!

Listing programu:

```
/* 21-3-4, Objemne teleso, Zuzu */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


/* Vrchol (x,y) mnohouholnika */
struct POINT
{
  int x, y;
};

/* Mnohouholnik */
struct POLYGON
{
  /* z-suradnica steny */
  int z;
  /* Vrcholy steny */
  struct POINT *p;
  int n;
};

/* Vrchol Quad-tree */
struct NODE
{
  int area, sum, flip;
};

int z_cmp(const void *va, const void *vb)
{
  return ((struct POLYGON *)va)->z-((struct POLYGON *)vb)->z;
}
```

[4] vyskúšajte ešte vylepšiť počítanie prierezu

```
int int_cmp(const void *va, const void *vb)
{
  return *(int*)va-*(int*)vb;
}

/* Povrch, ktory je pokryty vo vrchole */
int povrch(struct NODE *node)
{
  if(node->flip)
    return node->sum-node->area;
  return node->area;
}

/* Quad-tree */
struct NODE *q;
int nq, dim, *v;

/* Uprav Quad-tree */
void flip(int i, int xmin, int xmax, int ymin, int ymax, int x1, int x2, int y1, int
y2)
{
  int xmid=(xmin+xmax)/2, ymid=(ymin+ymax)/2;

  if(x1==xmin&&x2==xmax&&y1==ymin&&y2==ymax)
  {
    q[i].sum=(v[xmax]-v[xmin])*(v[ymax]-v[ymin]);
    q[i].flip=!q[i].flip;
    return;
  }
   else
  {
    if(x2<=xmid)
    {
      if(y2<=ymid)
        flip(4*i-2, xmin, xmid, ymin, ymid, x1, x2, y1, y2);
       else
      if(y1>=ymid)
        flip(4*i+1, xmin, xmid, ymid, ymax, x1, x2, y1, y2);
       else
      {
        flip(4*i-2, xmin, xmid, ymin, ymid, x1, x2, y1, ymid);
        flip(4*i+1, xmin, xmid, ymid, ymax, x1, x2, ymid, y2);
      }
    }
     else
    if(x1>=xmid)
    {
      if(y2<=ymid)
        flip(4*i-1, xmid, xmax, ymin, ymid, x1, x2, y1, y2);
       else
      if(y1>=ymid)
        flip(4*i+0, xmid, xmax, ymid, ymax, x1, x2, y1, y2);
       else
      {
        flip(4*i-1, xmid, xmax, ymin, ymid, x1, x2, y1, ymid);
        flip(4*i+0, xmid, xmax, ymid, ymax, x1, x2, ymid, y2);
      }
    }
     else
```

```
    {
      if(y2<=ymid)
      {
        flip(4*i-2, xmin, xmid, ymin, ymid, x1, xmid, y1, y2);
        flip(4*i-1, xmid, xmax, ymin, ymid, xmid, x2, y1, y2);
      }
       else
      if(y1>=ymid)
      {
        flip(4*i+1, xmin, xmid, ymid, ymax, x1, xmid, y1, y2);
        flip(4*i+0, xmid, xmax, ymid, ymax, xmid, x2, y1, y2);
      }
       else
      {
        flip(4*i-2, xmin, xmid, ymin, ymid, x1, xmid, y1, ymid);
        flip(4*i-1, xmid, xmax, ymin, ymid, xmid, x2, y1, ymid);
        flip(4*i+0, xmid, xmax, ymid, ymax, xmid, x2, ymid, y2);
        flip(4*i+1, xmin, xmid, ymid, ymax, x1, xmid, ymid, y2);
      }
    }
  }
  if(4*i<nq)
  {
    /* Pri ceste naspat do korena, oprav plochu */
    q[i].area=povrch(&q[4*i-2])+povrch(&q[4*i-1])+povrch(&q[4*i])+povrch(&q[4*i+1]);
  }
}

struct POLYGON *t;

int na_priamke(struct POINT *a, struct POINT *b, struct POINT *c)
{
  if(a->x==b->x&&a->x==c->x)
    return 1;
  if(a->y==b->y&&a->y==c->y)
    return 1;
  return 0;
}

/* Dalsi (cyklicky) vrchol mnohouholnika */
int next(int i, int n)
{
  if(i==n-1)
    return 0;
  return i+1;
}

/* Predchadzajuci (cyklicky) vrchol mnohouholnika */
int prev(int i, int n)
{
  if(!i)
    return n-1;
  return i-1;
}

#define min(a,b) ((a<b)?a:b)
#define max(a,b) ((a>b)?a:b)

/* Obrati celu stenu (mnohouholnik) */
void flip_stena(int index)
```

```
{
  int i=0, j=1, k=2, x, y;
  struct POINT *p=t[index].p;

  while(i!=k)
  {
    while(i!=k&&na_priamke(&p[prev(i,t[index].n)],&p[i],&p[j]))
      i=prev(i,t[index].n);
    while(i!=k&&na_priamke(&p[i],&p[j],&p[k]))
    {
      j=k;
      k=next(j,t[index].n);
    }
    while(i!=k&&na_priamke(&p[j],&p[k],&p[next(k,t[index].n)]))
      k=next(k,t[index].n);
    if(i==k)
      break;
    flip(1, 0, dim, 0, dim, min(p[i].x,min(p[j].x,p[k].x)),
max(p[i].x,max(p[j].x,p[k].x)),
      min(p[i].y,min(p[j].y,p[k].y)), max(p[i].y,max(p[j].y,p[k].y)));
    x=p[i].x^p[j].x^p[k].x;
    y=p[i].y^p[j].y^p[k].y;
    p[k].x=x;
    p[k].y=y;
    j=k;
    k=next(j, t[index].n);
  }
}

int main(void)
{
  int i, j, k, n, z, ok, m=0, objem;

  scanf("%d", &k);
  t=malloc(k*sizeof(struct POLYGON));
  for(n=i=0;i<k;i++)
  {
    scanf("%d", &t[n].n);
    t[n].p=malloc(t[n].n*sizeof(struct POINT));

    ok=1;
    for(j=0;j<t[n].n;j++)
    {
      scanf("%d %d %d", &t[n].p[j].x, &t[n].p[j].y, &z);
      if(!j)
        t[n].z=z;
       else
        if(t[n].z!=z)
          ok=0;
    }
    if(ok)
    {
      m+=t[n].n;
      n++;
    }
  }

  /* Normalizuj suradnice */
  v=malloc(2*m*sizeof(int));
```

```
for(k=i=0;i<n;i++)
  for(j=0;j<t[i].n;j++)
  {
    v[k++]=t[i].p[j].x;
    v[k++]=t[i].p[j].y;
  }

qsort(v, k, sizeof(int), int_cmp);
for(m=i=1;i<k;i++)
  if(v[i]>v[m-1])
    v[m++]=v[i];

for(k=i=0;i<n;i++)
  for(j=0;j<t[i].n;j++)
  {
    t[i].p[j].x=(int*)bsearch(&t[i].p[j].x, v, m, sizeof(int), int_cmp)-v;
    t[i].p[j].y=(int*)bsearch(&t[i].p[j].y, v, m, sizeof(int), int_cmp)-v;
  }

/* Quad-tree */
for(j=i=1;j<m-1;j<<=1);
dim=j;

for(i=m;i<=j;i++)
  v[i]=v[m-1];

for(nq=i=1;i<=j*j;i<<=2)
  nq+=i;

q=malloc(nq*sizeof(struct NODE));
memset(q, 0, nq*sizeof(struct NODE));

/* Zotriedit steny podla Z-suradnice */
qsort(t, n, sizeof(struct POLYGON), z_cmp);

for(i=0;i<n;i++)
{
  if(t[i].z!=t[0].z)
    break;
  flip_stena(i);
}

objem=0;
z=t[0].z;

while(i<n)
{
  objem+=(t[i].z-z)*povrch(&q[1]);
  z=t[i].z;

  /* Zisti dalsi povrch */
  for(j=i;i<n;i++)
  {
    if(t[i].z!=t[j].z)
      break;
    flip_stena(i);
  }
}

printf("Objem: %d\n", objem);
```

```
  return 0;
}
```

opravoval JanoS
(max. 15 bodov)

5. O mravcoch III

Tak ako som sľúbil, tak aj činím, a po dvoch rokoch som sa opäť vrhol na opravovanie vašich výtvorov. Hneď ako som sa do nich začítal som si všimol, že veľa z vás ignorovalo poznámku o tom, že dôležitý je čas, a tvrdohlavo písalo riešenie s prácou $O(N \log N)$, ktoré malo čas horší ako vzorové riešenie.

Vzorové riešenie v čase $O(\log N)$ bolo pritom celkom jednoduché, a bolo oceňované 15 bodmi. Za riešenie v čase $O(\log N. \log N)$, ktoré bolo o dosť ťažšie napísať som rozdával 12 bodov, a za riešenie lineárne ($O(N)$) ste mohli dostať bodov 8. Jedno zblúdilé riešenie, ktoré vôbec nepoužívalo silu paralelných výpočtov a triedilo sekvenčne dostalo 4 body (Keby to aspoň bolo v čase $O(N \log N)$[5] :))

Kvalitu popisu programu som oceňoval tradične 0 až -2 bodmi a drobné chyby boli oceňované po -1 bode. Za úroveň spôsobeného hlavybôľa, prípadne estetického zážitku som pridal -1 až $+1$ bodík.

Vzorové riešenie

Najlepšie riešenie je založené na paralelizovanom mergesorte. Spojiť dve utriedené postupnosti vieme najlepšie v čase $O(\log \log N)$[6], s použitím tohoto spájania by sme dosiahli algoritmus bežiaci v čase $O(\log N \log \log N)$ s celkovou prácou $O(N \log N)$. Takýto algoritmus by bol optimálny, avšak časovo sa dá ešte vylepšiť. Mnohí z vás sa snažili paralelizovať quicksort, ale lepší čas než $O(\log N \log N)$ nedosiahli. Tento mergesert je však lepší, a bol by ocenený 18 bodmi. Zlepšenie tohoto algoritmu vôbec nieje triviálnou záležitosťou, ale dá sa napísať s časovou zložitosťou $O(\log N)$ a optimálnou prácou. Ak by ste náhodou odovzdali takéto riešenie, osobne by som vám prišiel zložiť poklonu :-)

Nám však až tak nezáležalo na optimálnej práci, dostačujúci bol ideálny čas. A riešenie v logaritmickom čase sa dalo celkom napísať poľahky. V prvom kole sme našli algoritmus, ktorý v logaritmickom čase zistil koľko prvkov je menších než X. Hmmm, ak by sme tento algoritmus pustili naraz pre všetky prvky, tak máme v logaritmickom čase zistené pre každý prvok jeho poradie v utriedenom poli. To znie ako dobrý nápad nie? Takýmto prístupom získame časovú zložitosť $O(\log N)$ ale počet počítačov $O(N^2)$. Tento algoritmus teda bude mať celkovú prácu $O(N^2 \log N)$, čo je o jeden rád viac ako optimálny sekvenčný algoritmus.

Ako vzorové riešenie uvádzam algoritmus používajúci $O(N^2)$ počítačov, ak máte záujem o optimálne riešenie, pozrite sa do literatúry.

Listing programu:

```
If CPUID > (Mem[0] * Mem[0]) Then Halt;
N   := Mem[0];
Row := ((CPUID-1) Div N) +1;
Col := ((CPUID-1) Mod N) +1;
Value := Mem[Row];

If (Mem[Row] > Mem[Col]) Then Mem[CPUID]:=1
                         Else Mem[CPUID]:=0;
```

[5] Domáce cvičenie: sadnite si teraz za počítač a napíšte program na triedenie N prvkov v čase $O(N \log N)$ a odmerajte si čas.

[6] Joseph JáJá, An Introduction to Parallel Algorithms - kapitola 4.2.3

```
   Dva_na_K:=1;
7: If (Col > Dva_na_K) Then Mem[CPUID] := Mem[CPUID] + Mem[CPUID-Dva_na_K];
   Dva_na_K := Dva_na_K * 2;
   If Dva_na_K < N Then GoTo 7;

   If (Col = N) Then Mem[Mem[CPUID]] := Value;
```

Výsledková listina po 1. kole kategórie KSP

	Meno a priezvisko	Škola	Ročník	11	12	13	14	15	Σ
1	Cicko Miroslav	Gym. Tajovského B. Bystrica	3	13	15	15	12	15	70
2	Simančík František	Gym. Grösslingová BA	3	13	15	15	8	11	62
3	Imriška Jakub	Gym. Jura Hronca BA	2	13	15		7	11	46
4	Petruľák Matúš	Gym. Grösslingová BA	3	13	8	5	12	7	45
4	Plžík Milan	Gym. Ľ. Štúra Zvolen	3	10	13	3	4	15	45
6	Beleš Lukáš	Gym. Čadca	3	13	2	7	6	15	43
6	Kuštárová Tamara	Bilingválne gym. Sučany	3	9	14	10		10	43
6	Perešíni Peter	Gym. Tajovského B. Bystrica	2	13	15	15			43
9	Schlosáriková Eva	Gym. P. de Coubertina Piešťany	3	13	13	7		9	42
10	Buštor Stano	Gym. Jura Hronca BA	3	13	8	8		12	41
11	Krchniak Matej	Gym. Jura Hronca BA	2	9	14	8		8	39
12	Lenhardt Rastislav	Gym. Jura Hronca BA	4	13	10			15	38
13	Nánási Michal	Gym. Jura Hronca BA	3	5	9	8	4	11	37
13	Poláček Lukáš	Gym. K. Štúra Modra	4	15	14		8		37
15	Bundala Daniel	Gym. Jura Hronca BA	2	13	11			11	35
16	Trnovcová Zuzana	Gym. Jura Hronca BA	3	13	10			9	32
17	Takáč Slavomír	Gymnázium	2	13	11	1			25
18	Beran Jakub	Gym. Alejová Košice	2	9	1	7		7	24
18	Kuchynárová Mariana	Gym. Jura Hronca BA	4	7	10			7	24
20	Zeman Marek	Gym. Jura Hronca BA	3	13		7			20
21	Mindek Peter	Škola pre mim. nadané deti BA	2	9		8			17
21	Urminský Tomáš	SPŠ Nové Mesto nad Váhom	2	9	3	1		4	17
23	Goga Peter	Gym. 17. novembra Topoľčany	4	9	4	1	1	0	15
23	Smažáková Dana	Gym. Jura Hronca BA	4		8			7	15
25	Ďurfina Lukáš	Gym. Golianova Nitra	3	9	2	2			13
26	Bodnár Jozef	Gym. Fiľakovo	3		12				12
27	Glaus Peter	Gym. Jura Hronca BA	4	3				8	11
28	Kováč Michal	Gym. Grösslingová BA	2	7	0	1		1	9
28	Špalek Lukáš	Gym. Čadca	3		2			7	9
28	Vadkerti Miroslav	SPŠE Nové Zámky	4	9					9
31	Morihladko Peter	Gym. Školská Spiš. Nová Ves	3			8			8
32	Vanderka Peter	Gym. Trstená	3	7					7