Korešpondenčný seminár z programovania
XXII. ročník, 2004/05
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.

Vzorové riešenia 2. kola zimnej časti

Tož, už je to tak, je po zimnej časti KSP. Všetko opravené, všetko rozhodnuté, najlepší z vás už môžu netrpezlivo očakávať pozvánku na sústredenie :-) A ostatní sa môžu s chuťou pustiť do riešenia letnej časti, začína sa odznova, nová šanca je pred vami! No a pred tým, ako začnete riešiť nové príklady, prečítajte si tieto vzorové riešenia, nech ste múdrejší.

Iný krajec, iný mravec...

KSPáci

1. Odvážny Marek

opravovala Julka
(max. 15 bodov)

Tento príklad bol dosť ľahký. Väčšina z vás mala vzorové riešenie. Tí, čo nemali vzorové, napísali aspoň s horšou časovou zložitosťou, ale vyskytlo sa aj nejaké to nefunkčné riešenie. No a hodnotenie bolo takéto: riešenia s časovou zložitosťou $O(N \log N)$ dostali maximálne 15 bodov. Za časovú zložitosť $O(N^2)$ bolo možné získať až 11 bodov. Zložitosť $O(N^3)$ dostala 9 bodov. Pomalšie riešenia sa našťastie nevyskytli ;-) Za popis a chýbajúce časové zložitosti som plánovala strhávať veľa bodov, ale nebolo treba, lebo väčšina z vás mala popisy v poriadku.

Priamočiare riešenie je jednoduché. Vyskúšam všetky dvojice prvkov z postupnosti, spočítam súčet prvkov medzi nimi. Toto vedie ku kubickému riešeniu. Dá sa to zlepšiť jednoduchou fintou, ktorú väčšina z vás pozná. Pre každý prvok si predpočítam súčty všetkých predchádzajúcich prvkov. Teda ak tá postupnosť je $\{p_i\}_{i=1}^{n}$, tak si spravíme postupnosť $\{s_i\}_{i=0}^{n}$, pričom $s_0 = 0$ a $s_i = s_{i-1} + p_i$ pre $0 < i \leq n$. Zjavne $s_i = \sum_{j=1}^{i} p_j$, z čoho už ľahko spočítame súčet čísel od i po j, t.j. $\sum_{k=i}^{j} p_i = s_j - s_{i-1}$. Takéto riešenie je už kvadratické. Už to len trošku zlepšiť a bude to vzorové.

Pozrime sa teraz, čo vlastne potrebujeme spraviť. Chceme nájsť také dva indexy i a j, že absolútna hodnota súčtu čísel medzi nimi bude čo najbližšie k zadanému číslu T. Ak by sme mali súčty utriedené, tak sa nám tie indexy budú hľadať ľahšie. Spravíme si pole PORADIE, kde PORADIE[i] bude index na i-ty najmenší čiastočný súčet (s_j). Teraz budeme mať dva pointre i a j znamenajúce, že kontrolujeme súvislú podpostupnosť medzi prvkami $s_{PORADIE[i]}$ nevrátane a $s_{PORADIE[j]}$ vrátane. Tie nám budú ukazovať v poli PORADIE. Na začiatku $i = 0$ a $j = 1$. Keď zvýšime j, tak tým aj zvýšime náš rozdiel (pretože prechádzame akoby po utriedenom poli s čiastočnými súčtami). Naopak keď zvýšime i, tak ten rozdiel zmenšíme. Vždy sa budeme snažiť náš rozdiel dostať čo najbližšie k T. Takto jednoducho nájdeme indexy i a j také, že $s_j - s_i$ je čo najbližšie k T. Takže tá naša správna súvislá podpostupnosť bude $\{p_k\}_{k=i+1}^{j}$.

Čo sa týka výslednej časovej zložitosti, tak vyrátanie čiastočných súčtov trvá $O(N)$, utriedenie $O(N \log N)$ a nájdenie správnych indexov $O(N)$. Takže výsledná časová zložitosť je $O(N \log N)$. Už len škoda že nie je rýchlejšie triedenie... Pamäťová zložitosť je zjavne $O(N)$.

A nasleduje krásne jednoduchý listing programu podľa Mira Cicka.

Listing programu:

```
var N,T:integer;
    postupnost,sucet,poradie:array[0..100] of integer;

procedure qsort(l,p:integer); {quick sort}
var i,j,tmp,x:integer;
begin
    i:=l; j:=p; x:=sucet[poradie[(i+j) div 2]];
    repeat
        while sucet[poradie[i]]<x do inc(i);
        while sucet[poradie[j]]>x do dec(j);
        if i<=j then
            begin
            tmp:=poradie[i]; poradie[i]:=poradie[j]; poradie[j]:=tmp;
            inc(i); dec(j);
            end;
    until i>j;
    if l<j then qsort(l,j);
    if i<p then qsort(i,p);
end;

var i,j,s,tmp,najS,zac,kon:integer;
begin
    readln(N,T);
    {nacitame a zaroven aj vyratame postupne sucty}
    sucet[0]:=0;
    for i:=1 to N do
        begin
            read(postupnost[i]);
            sucet[i]:=sucet[i-1]+postupnost[i];
        end;
    for i:=1 to N do poradie[i]:=i; { v poli PORADIE budu zotriedene ciastocne sucty}
    qsort(1,N);

    i:=0; j:=1;
    NajS:=sucet[poradie[j]]-sucet[poradie[i]];
    zac:=poradie[i]; kon:=poradie[j];
    repeat
        s:=sucet[poradie[j]]-sucet[poradie[i]];
        if abs(T-s)<abs(T-NajS) then {pamatame si, ktore S je k T najblizsie}
            begin NajS:=s; zac:=poradie[i]; kon:=poradie[j]; end;
        {posunieme spravny pointer}
        if (s>T) and (i+1<j) then inc(i) else inc(j);
    until j>N;

    if zac>kon then
        begin tmp:=zac; zac:=kon; kon:=tmp; end;
    write('Vysledna podpostupnost je: ');
    for i:=zac+1 to kon do write(postupnost[i],' ');
end.
```

2. Ojedinelý prípad

opravoval Mirko
(max. 15 bodov)

Sformalizujme si úlohu pomocou teórie grafov: Máme daný orientovaný graf $G = (V, E)$, kde V je vrcholová množina, a E je hranová množina, resp. podmnožina $V \times V$ a potre-

bujeme nájsť nejakú minimálnu podmnožinu vrcholovej množiny (označme ju A) tak, aby neexistovala hrana, ktorá by vychádzala z nejakého vrcholu z A a vchádzala do nejakého vrcholu z $V\backslash A$ (poznámka $V\backslash A$ značí všetky vrcholy grafu ktoré nie sú v A). Takúto množinu nazvime riešením problému.

Množinu vrcholov, v rámci ktorej sa vieme dostať z každého vrcholu do každého iného, voláme silne súvislá. Ak do nej už nevieme pridať žiadne ďalšie vrcholy tak, aby zostala táto vlastnosť zachovaná, voláme ju silne súvislý komponent.

Každý vrchol leží práve v jednom silne súvislom komponente – komponent pre vrchol v obsahuje práve každý vrchol, pre ktorý platí, že sa v našom grafe dá z v dostať doň aj z neho späť. (Táto množina vrcholov je zjavne silne súvislá, odvšadiaľ všade sa vieme dostať cez v. Žiaden iný vrchol do tohto komponentu zjavne patriť nemôže.)

Teraz si dokážeme nasledujúce tvrdenie: Najlepším riešením našej úlohy je najmenší silne súvislý komponent v našom grafe, z ktorého nevychádza hrana. Čitateľ, ktorému je toto jasné, môže nasledujúce riadky preskočiť.

Lema (1). *Ak $x \in A$, a existuje xy-cesta, tak y nutne patrí do A (pod A chápeme nejaké jedno konkrétne riešenie).*

Dôkaz: Stačí dokázať, že ak $x \in A$ a existuje xy-cesta, tak každý vrchol z nej musí patriť do A. Sporom: Predpokladajme, že nejaký vrchol na tejto ceste do A nepatrí. Nájdime posledný vrchol na tejto ceste, ktorý nepatrí do A, vďaka predpokladu je dobre definovaný (označme ho X). K nasledujúcemu vrcholu na tejto ceste (označme ho Y) existuje hrana a Y nepatrí do A. Čiže $Y \in V\backslash A$, dostávame sa ale do sporu s definíciou A (máme teraz hranu z A do $V\backslash A$).

Toto tvrdenie, je postačujúcou podmienkou, aby bol každý kamarát spokojný. Nezaručuje však, aby bol spokojný aj Mirko. Na to je nutné nájsť minimálnu množinu, spĺňajúcu (1). K tomu nám pomôže nasledujúce tvrdenie:

Lema (2). *Ak A je minimálna, tak pre každé dva vrcholy $x, y \in A$ platí: existuje xy-cesta a zároveň existuje yx-cesta.*

Dôkaz urobíme sporom: Predpokladajme, že A je minimálne. Vieme, že pre A platí (1) – ináč nás nezaujíma, či je minimálne. Bez ujmy na všeobecnosti nech neexistuje xy-cesta. Uvažujme množinu B, ktorá je množinou všetkých takých vrcholov z, že existuje xz-cesta. Zjavne pre B platí (1), lebo, ak máme $u \in B$ a existuje uv-cesta, tak potom zreťazením xu-cesty (ktorá existuje, lebo $u \in B$) a uv-cesty vznikne xv-sled, z ktorého možno vybrať xv-cestu, teda existuje xv cesta, a z toho vyplýva $v \in B$. Čo je vlastne (1).

Keďže $x \in A$ a platí (1), vidíme, že B je podmnožinou A (ak by nebolo, musel by existovať taký vrchol, ktorý patri do B a nepatrí do A, čo by bolo v spore s (1)), a z neexistencie xy-cesty vyplýva, že y nepatrí do B. Takže by existovala menšia množina ako A, pre ktorú by tiež platilo (1), takže A nie je minimálne, čo je v spore s predpokladom.

Z (2) vidíme, že najmenšie riešenie bude nejaký silne súvislý komponent.

Navyše z (1) vidíme, že z tohto komponentu nemôže viesť hrana – keby totiž z neho nejaká hrana viedla, museli by sme mať v riešení aj vrchol, kam vedie, ten ale do nášho komponentu nepatrí.

Teraz je vhodné zamyslieť sa, prečo v každom orientovanom grafe existuje aspoň 1 silne súvislý komponent, z ktorého nevedie žiadna hrana.

My si teraz ukážeme algoritmus, ktorý hľadá silne súvislé komponenty.

Náš algoritmus je založený na prehľadávaní do hĺbky (DFS). Pre každý vrchol, si pri prehldávaní budeme pamätať čas, kedy sme doň vošli tzv. $enter[x]$, a čas kedy sme z neho odišli tzv. $exit[x]$ a zavedieme si ešte takú konvenciu pre označovanie vrcholov počas prehľadávania: ak sme vo vrchole ešte neboli bude biely, ak sme z neho ešte nevyšli bude šedý, a ak sme z neho už vyšli bude čierny.

DFS ma pár užitočných vlastností:

1. Pre každé dva vrcholy x, y môže nastať iba jedna z týchto štyroch možností: $enter[x] < enter[y] < exit[y] < exit[x]$, $enter[y] < enter[x] < exit[x] < exit[y]$, $enter[x] < exit[x] < enter[y] < exit[y]$, $enter[y] < exit[y] < enter[x] < exit[x]$.
2. Ak platí $enter[x] < enter[y] < exit[y] < exit[x]$, hovoríme, že y je potomok x, a vieme, že existuje xy-cesta, opačne to platiť nemusí.
3. Ak v čase, keď vstúpime do vrcholu x, existuje xy-cesta, ktorá obsahuje len biele vrcholy (okrem vrcholu ktorý je sedy stava šedý x), potom sa y stane potomkom x.

Algoritmus sa bude skladať z 2 fáz. Najprv pustíme z každého vrcholu (v ľubovoľnom poradí) DFS (v programe dfs1()), aby sme zistili hodnoty $exit[]$, pre každý vrchol. Samozrejme, ak už sme nejaký vrchol navštívili, tak na neho DFS nevoláme, takto navštívime každý vrchol práve raz. Potom zopakujeme podobnú činnosť, s tým rozdielom, že nemudeme menit hodnoty $exit[]$ a $enter[]$ a po hranách budeme chodiť opačne a prehľadávania nebudeme spúšťať v ľubovoľnom poradí, ale v takom poradí, aké nám určia v prvej fáze zistené hodnoty $exit[x]$, konkrétne od najväčšej hodnoty po najmenšiu. Teda najprv nájdeme vrchol s najväčšou hodnotou $exit[x]$, pustíme z neho DFS2, potom nájdeme najväčší z nenavštívených... Samozrejme náš program bude pracovať trochu ináč, aby sa to ľahšie písalo, a rozumnejšie, aby sme mali dobrú zložitosť.

A kde je výstup algoritmu? No výstup je taký skrytý. Pri zavolaní DFS2 na nejaký vrchol x, sa vždy navštívia práve vrcholy zo silne súvislého komponentu, do ktorého patrí x. A zakaždým, keď nájdeme nejaký komponent, tak len skontrolujeme, že či z neho nahodou nevychádza hrana, a že či je menší ako doteraz nájdený. A nakoniec vypíšeme najmenší nájdený komponent.

Poďme si dokázať, že to funguje. Nech sme pustili DFS2 grafe z vrcholu x označili sme množinu vrcholov B. Vieme, že z každého vrcholu z B sa vieme dostať do x, lebo ku každému $y \in B$ sa viem dostať z x, ak pôjdeme po hranách opačne.

Teraz treba ešte dokázať, že sa aj z x vieme dostat dostať do každého vrcholu z B . Vďaka tomu, že postupujeme v špeciálnom poradí, vieme, že $exit[x]$ je väčšie, ak hociktorý z neoznačených vrcholov, teda aj ako každý z B .

Nech y je ľubovoľný vrchol z B. Teraz už len použijeme 1.vlastnosť DFS. Môžu nastať len dva prípady: $enter[x] < enter[y] < exit[y] < exit[x]$ a $enter[y] < exit[y] < enter[x] < exit[x]$. V prvom prípade nám existencia xy-cesty vyplýva priamo z vlastnosti 2. A druhý prípad nemôže nastať, lebo keby nastal, tak máme spor. Ukážeme si prečo.

Vieme, že existuje yx-cesta. Ak v čase, keď sme navštívili y v prvom prehľadávaní, sú všetky vrcholy na tejto ceste biele (okrem y), môžeme použiť vlastnosť 3, z čoho nám vyjde spor s predpokladom konkrétne: muselo by platiť $\ldots exit[x] < exit[y]$, čo nemôže, lebo ako sme sa dohodli $exit[x]$ musi byt väčší. Ak je ale na tejto yx-ceste nejaký šedý vrchol, postupujeme nasledovne. Zoberme posledný takýto a označme ho z. Posledný v takom zmysle, že všetky vrcholy na zx úseku yx cesty sú biele (okrem z, ktorý je šedý). To ale znamená, že môžeme použiť vlastnosť 3 (lebo máme zx-cestu, na ktorú sa vzťahuje 3. vlastnosť). Vieme, že z je určite z B (lebo je na yx-ceste). Z týchto dvoch informácii dostávame: $enter[z] < enter[x] < exit[x] < exit[z]$ a zároveň $z \in B$, ale kedže $z \in B$, implikuje $exit[z] < exit[x]$, takáto situácia nemôže nastať. Čo sme chceli chceli ukázať.

Takže trochu krkolomne, ale dokázali sme, že pre každý vrchol $y \in B$ existuje aj xy-cesta aj yx-cesta. Ako cvičenie prenechávam rozobrať prípad, ak je na yx-ceste nejaký čierny vrchol (čo samozrejme nemôže nastať), a že B je celý komponent.

Na záver ešte poznámka o časovej zložitosti, ktorá je zjavne lineárna, lebo časy nemusíme triediť (dostávame ich v dobrom poradí). A prehľadávanie do hĺbky má lineárnu zložitosť. Bodovanie bolo jednoduché, riešenie ktorých zložitosť je $O(N+M)$ dostali 15 bodov, riešenia ktoré z každého vrcholu púšťali DFS, teda s časovou zložitosťou $O(N(M+N))$ dostali 10 bodov. Iné riešenia neboli a body sa strhávali aj za nedostatočný popis, nejaké chyby, a tak...

Listing programu:

```
#include<cstdio>
#define maxN 100
int jeHrana[maxN][maxN];
int n,m,k;
int vis[maxN],ktoJeKolkaty[maxN],aktualnyKomponent[maxN];
int jeDobry,IchChce,kohoMameNajradsej,precoHoMameNajradsej;

void dfs1(int x,int pis){
    if(vis[x]) return ;
    vis[x]=1;
    if (pis) printf("%d\n",x+1);
    for(int i=0;i<n;i++) if (jeHrana[x][i]) dfs1(i,pis);
    if (!pis) ktoJeKolkaty[--k]=x;
}

void dfs2(int x,int color) {
    if (vis[x]) return;
    vis[x]=color;
    aktualnyKomponent[IchChce++]=x;
    for(int i=0;i<n;i++) if (jeHrana[i][x]) dfs2(i,color)
}

int main(){
    scanf("%d %d",&n,&m);
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) jeHrana[i][j]=0;
    for(int i=0;i<m;i++) {int a,b;
        scanf("%d %d",&a,&b);
        jeHrana[a-1][b-1]=1;
    }
    k=n;
    for(int i=0;i<n;i++) vis[i]=0;
    for(int i=0;i<n;i++) dfs1(i,0);
    for(int i=0;i<n;i++) vis[i]=0;
    kohoMameNajradsej=0;
    for(int i=0;i<n;i++) {
        IchChce=0;
        jeDobry=1;
        dfs2(ktoJeKolkaty[i],i+1);
        for(int j=0;j<IchChce;j++) for(int ii=0;ii<n;ii++)
            if (jeHrana[aktualnyKomponent[j]][ii] &&
                    vis[aktualnyKomponent[j]]!=vis[ii]) jeDobry=0;
        if (IchChce && jeDobry && (IchChce<precoHoMameNajradsej
                    || !kohoMameNajradsej)) {
            precoHoMameNajradsej = IchChce;
            kohoMameNajradsej = ktoJeKolkaty[i];
        }
    }
    printf("pozveme:\n");
    for(int i=0;i<n;i++) vis[i]=0;
    dfs1(kohoMameNajradsej,1);
}
```

3. OEIN GONIATH

opravoval Bee
(max. 15 bodov)

Prišlo veľa rôznych riešení a potešilo ma, že všetky boli viac-menej funkčné a väčšina optimálna. Pozrime sa teda, ako sa tento príklad dal riešiť. Najhoršie riešenia boli v čase $O(N^2 \log N)$ a s kvadratickou pamäťou, tie dostali po 8 bodov. Nasledujú riešenia v kvadratickom čase a lineárnou pamäťou, tie dostali po 12 bodov, no a nakoniec optimálne $O(N \log N)$ za 15 bodov.

Možných prístupov je veľa, najčastejšie prichádzali riešenia založené na zásobníku, nejakej forme rekurzie a iné. Ja som sa rozhodol pre, podľa mňa, najľahšie napísateľné riešenie. Treba si uvedomiť základnú myšlienku, na ktorú väčšina z vás prišla a to, že ak máme dva susedné stromy, tak prežije ten strmší z nich (strmosť chápeme ako absolútnu hodnotu rozdielu uhlu stromu a 90 stupňov). Toto tvrdenie je zrejmé každému, kto už videl trojuholník. Ak sú rovnako strmé, tak buď sú rovnobežné a prežijú oba, alebo sa zrazia korunami a potom prežije ten, ktorý rastie zľava. A teraz myšlienka číslo 2: toto neplatí, len pre susedné stromy, ale pre stromy všeobecne. Toto tvrdenie je zrejmé, pretože to, či strom prežije si už môžeme previesť na to, ako strmo rastie a kedže strmosť je tranzitívna[1], tak tvrdenie platí. Podobne ľahko sa uvedomí, že ak strom rastie doľava, nemôže už naraziť do stromu napravo a naopak. Takisto, ak je strom kolmý, tak ho už nezastaví nič.

Tieto myšlienky nám už stačia na skonštruovanie algoritmu. Najprv si pole utriedime podľa x-ovej súradnice. Teraz nám pole stačí prechádzať zľava, pamätať si doteraz najstrmší strom a kontrolovať, či zabije stromy rastúce doľava. To isté spravíme ešte raz sprava pre stromy, ktoré rastú doprava. V obidvoch prípadoch si pamätáme zabité stromy a nakoniec už len vypíšeme tie, ktoré prežili.

Ešte podotknem, že keby uhly boli celočíselné, tak by sa úloha dala riešiť v lineárnom čase utriedením podľa uhla, napríklad count-sortom. Ale to nie je až také dôležité. Dôležitejšie, aj pre budúcnosť, je: Ak v zadaní nie je nič uvedené, tak počítajte so všetkým!

Takže hor sa na program, lebo hoci väčšina z vás mala optimálne riešenie, tak ten kód nebol zďaleka ideálny a vždy sa je čo učiť. Ešte pripomeniem, že je tam drobná fintička: od uhla každého stromu odpočítam 90, aby sa s tým trochu ľahšie pracovalo.

Listing programu:

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

struct druid{
  double x, an; int id; bool alive;
  bool operator()(druid d1, druid d2){ return d1.x<d2.x; }
};

int main(){
  int n; cin>>n; druid d[n];
  for(int i=0;i<n;i++) {
    cin>>d[i].x>>d[i].an; d[i].an-=90;
    d[i].id=i+1; d[i].alive=1;
  }
  sort(d, d+n, druid());
  double right=90; //najkolmejší sused sprava
  for(int i=n-1;i>=0;i--) {
    if(fabs(right)>fabs(d[i].an)) right=d[i].an;
    if(d[i].an<0)
```

[1] ak $a > b$ a $b > c$, tak $a > c$, v našom prípade je $>$ strmosť

```
      if(fabs(d[i].an)>fabs(right)) d[i].alive=0;
  }
  double left=90; //najkolmejší sused zlava
  for(int i=0;i<n;i++) {
    if(fabs(left)>fabs(d[i].an)) left=d[i].an;
    if(d[i].an>0)
      if(fabs(d[i].an)>fabs(left) ||
          (left<0 && fabs(fabs(left)-fabs(d[i].an))<0.0000001)) d[i].alive=0;
  }
  for(int i=0;i<n;i++) if(d[i].alive) cout<<d[i].id<<' ';
  cout<<endl;
}
```

4. O 47. hyperpriestorovej dimenzii

opravoval Lukáš
(max. 15 bodov)

Ako ste všetci správne pochopili, zadaný graf je vždy strom. Pretože v ňom nie je žiaden cyklus, tak medzi každými dvoma vrcholmi existuje práve jedna cesta a na hľadanie dĺžky cesty nám stačí aj prehľadávanie do hĺbky.

Najjednoduchšie riešenie je pre každú dvojicu vrcholov nájsť dĺžky cesty pomocou prehľadávania, čo však vedie k časovej zložitosti $O(N^3)$ a bodovému zisku 8 bodov. Zlepšiť to môžeme tak, že prehľadáme graf z každého vrchola iba raz a získame vzdialenosti z tohto vrchola do zvyšku grafu. Časová zložitosť tohto algoritmu je $O(N^2)$ a bol ocenený 11 bodmi. My si popíšeme optimálne riešenie pracujúce v čase $O(N)$.

Najprv si zakoreňme graf. Spravíme to prehľadávaním z ľubovoľného vrchola a tento vrchol bude koreňom. Označme $l(V)$ počet hrán na (jedinej) ceste medzi vrcholom V a koreňom. Potom ľubovoľná cesta v strome ide cez vrcholy $V_1, V_2, \ldots, V_k, \ldots, V_n$ tak, že platí $l(V_1) > l(V_2) > \ldots > l(V_k) < \ldots < l(V_n)$. Keby bola časť postupnosti v tvare $l(V_s) < \ldots < l(V_t) > \ldots > l(V_u)$, znamenalo by to, že cesta by sa vzďaľovala od koreňa a potom sa zase k nemu približovala. To však v strome znamená, že by išla späť nutne po tých istých vrcholoch a už by to nebola cesta.

Pre vrcholy najdlhšej cesty teda platí $l(V_1) > \ldots > l(V_k) < \ldots < l(V_n)$. Z tejto nerovnosti vidno, že vrcholy najdlhšej cesty sú hranovo vzdialenejšie od koreňa ako vrchol V_k, takže pri prehľadávaní do hĺbky ich všetky navštívime až po vrchole V_k, ale zároveň predtým, ako ho opustíme. Zároveň sa najdlhšia cesta skladá z dvoch ciest $V_1, \ldots, V_k$ a $V_k, \ldots, V_n$. Keďže chceme maximalizovať celkovú dĺžku, tak aj tieto dve cesty musia byť čo najdlhšie. Predpokladajme, že počas prehľadávania spracovávame vrchol V, navštívili sme už všetkých jeho synov a chceme nájsť dĺžku najdlhšej cesty, ktorá ide cez tento vrchol. Ďalej predpokladajme, že pre každého syna A_i vrchola V, poznáme dĺžku najdlhšej cesty, ktorá sa začína hranou (V, A_i) a hranovo sa vzďaľuje od koreňa. Označme túto dĺžku $d(A_i)$. Potom dĺžka najdlhšej cesty je súčet dvoch najväčších hodnôt $d(A_i), d(A_j)$. Vrchol V je práve ten vrchol cesty, ktorý je hranovo najbližšie ku koreňu.

Už nám zostáva len vyriešiť rátanie hodnôt $d(A_i)$. Využijeme na to rekurzívnu funkciu `prehladaj`, ktorá bude brať ako parameter číslo vrcholu, prehľadá graf z tohto vrcholu a nájde všetky svoje hodnoty $d(A_i)$, pomocou ktorých nájde svoju najdlhšiu cestu. Keď zavoláme funkciu na syna A_i, tá vráti dĺžku $c(A_i)$ najdlhšej cesty, ktorá sa začína vo vrchole A_i a hranovo sa vzďaluje od koreňa. Potom $d(A_i) = c(A_i) + l(V, A_i)$, kde $l(V, A_i)$ je dĺžka hrany (V, A_i). Vrchol V ako svoju hodnotu $c(V)$ vráti maximum z hodnôt $d(A_1), \ldots, d(A_n)$, pretože to je práve dĺžka najdlhšej cesty, ktorá sa začína vo vrchole V a hranovo sa vzďaľuje od koreňa.

Listing programu:

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

struct Hrana {
   Hrana *dal;
   int kam, dlz;
};

bool bol[100]={0};
Hrana *h[100]={0};
int maxx=0;
int prehladaj(int j) {
   bol[j]=true;
   int c[3]={0};
   for (Hrana *g=h[j]; g; g=g->dal) if (!bol[g->kam]) {
      c[2]=prehladaj(g->kam)+g->dlz;
      for (int i=0; i<2; i++) if (c[2]>c[i]) {
         if (i==0) c[1]=c[0];//pamätáme si dve najväčšie hodnoty
         c[i]=c[2]; break;
      }
   }
   maxx>?=c[0]+c[1];// >? je gcc operátor pre maximum
   return c[0];
}

int main() {
   int n; cin>>n;
   Hrana *p; int a, b, c;
   for (int i=0; i<n-1; i++) {
      cin>>a>>b>>c; a--; b--;
      p=new Hrana;
      p->dlz=c; p->kam=b;
      p->dal=h[a]; h[a]=p;
      p=new Hrana;
      p->dlz=c; p->kam=a;
      p->dal=h[b]; h[b]=p;
      //spájaný zoznam sa dá v C++ implementovať aj krajšie
      //v našom prípade list<pair<int, int> >, viac na www.sgi.com/tech/stl
   }

   prehladaj(0);//koreňom bude prvý vrchol
   cout<<maxx<<endl;
}
```

5. Otvor sa, sezam!

opravoval MišoF.
(max. 15 bodov)

Milo ma prekvapil počet a správnosť vašich riešení tohto príkladu, po prvej sérii som nebol až taký optimista. Dúfam, že aj počet čitateľov tohto riešenia by ma milo prekvapil. Tak sa poďme spolu pozrieť, ako sa to celé dalo spraviť.

Na úvod ešte jeden drobný disclaimer. Pozorný a formalizmy obľubujúci čitateľ by iste našiel v našom riešení drobné nepresnosti, nedostatky a nekonzistentnosti.[2] Vieme o nich, ale

[2] Čo presne znamená „nerozlíšiteľný“? Ako je to vlastne presne s Paľovou výpočtovou silou? Ak sa používa šifrovanie, nemôže Paľo podvádzať voľbou vhodnej šifry? A tak ďalej...

takto by to celé malo byť zrozumiteľnejšie. Koniec koncov, hlavným cieľom tohto príkladu malo byť ukázať vám, že takéto niečo existuje a na akých princípoch to funguje – nie nútiť vás rozumieť technickým detailom, ktoré to pod povrchom skrýva. Samozrejme, záujmu sa medze nekladú :-)

Podúloha a.

Máme graf G, o ktorom nemáme blademodrý šajn, či obsahuje hamiltonovskú kružnicu. Radi by sme vygenerovali postupnosť trojíc, ktorá by vyzerala na nerozoznanie podobne ako postupnosť trojíc (Paľom zverejnené súbory, Vladova voľba, Paľova odpoveď), ktorá by vznikla pri dôkaze našim protokolom.

Jedna možnosť: Nezačneme generovaním zverejnených súborov, ale vygenerujeme najskôr všetky Vladove voľby. Pozrime sa teraz na i-tu z nich. Ak chcel Vlado vidieť, či Paľo zverejnil graf G, vyrobíme súbory s náhodným prečíslovaním grafu G. V opačnom prípade vyrobíme súbory s kružnicou dĺžky N a náhodne doplníme ďalšie hrany tak, aby sedel ich počet. V každom prípade vieme doplniť tretiu zložku trojice tak, aby to vyzeralo, že komunikácia prebehla úspešne.

Druhou možnosťou je použiť fintu reportérov z príbehu: Skúsime spraviť veľa kôl protokolu medzi Vladom a Petrom (ktorý hamiltonovskú kružnicu nepozná). Peter sa snaží uhádnuť, ktorú možnosť si Vlado vyberie a také údaje zverejní. V približne polovici prípadov ho Vlado odhalí. Tieto odignorujeme a zoberieme zápisy komunikácie len pre tie prípady, kedy sa Petrovi Vlada podarilo presvedčiť.

Bez ohľadu na konkrétny spôsob generovania morálne poučenie zostáva to isté: keďže bez toho, aby sme poznali čokoľvek o hamiltonovskej kružnici v G, vieme generovať zápis komunikácie, **znamená to, že táto komunikácia žiadne informácie o dotyčnej kružnici neobsahuje**.

Podúloha b.

Zamyslime sa nad hintom: *Ako by ste dokázali farboslepému človeku, že dve kartičky sú rôznych farieb?* Nuž, povieme mu, ktorá je ktorá a necháme ho, nech si to na ne z jednej strany napíše. Teraz nám bude dokola ukazovať vždy jednu z kartičiek a kontrolovať, či mu na ňu povieme to isté, čo na začiatku.

A to isté budeme robiť aj s grafmi. Majme teda dva grafy G_1, G_2, Paľo chce Vladovi dokázať, že nie sú izomorfné. Opäť budeme dokola opakovať jeden protokol až kým už Vlado nebude mať dostatočnú istotu.

Priebeh jedného kola: Vlado si vyberie G_1 alebo G_2. Náhodne mu prečísluje vrcholy a výsledok pošle Paľovi. Paľo pomocou svojej výpočtovej sily (alebo ekvivalentne vševedúcosti) spozná, ktorý graf to je a odpovie Vladovi číslom 1 alebo 2. Vlado si overí, či Paľovo číslo je naozaj číslo grafu, ktorý si on vybral.

Funkčnosť je zjavná. Ak G_1 a G_2 nie sú izomorfné, Paľo vie Vladovi vždy jednoznačne odpovedať a presvedčí ho.

Čo **správnosť**? Nech teda Peter chce presvedčiť Vlada, že dva grafy G_1, G_2 nie sú izomorfné, pričom v skutočnosti izomorfné sú. Čo môže Peter robiť? Vidí graf, ktorý mu poslal Vlado. Ten je ale izomorfný ako s G_1, tak aj s G_2. Peter nemá ako vedieť, z ktorého z nich ho Vlado vytvoril, preto nemá žiadnu lepšiu možnosť ako tipnúť si – a s 50% pravdepodobnosťou byť odhalený.

Na záver **bezznalostnosť**. Falošné prepisy komunikácie vygenerujeme hravo. Použiť opäť môžeme oba postupy z predchádzajúcej časti. Najjednoduchšie je tentokrát robiť Vladov krok správne a ako Paľovu odpoveď vždy napísať to číslo, ktoré si Vlado na začiatku zvolil.

Podúloha c.

Tentokrát máme dva grafy G_1, G_2, Paľo chce Vladovi pre zmenu dokázať, že sú izomorfné. Samozrejme, opäť to musí zvládnuť bez toho, aby Vlada čokoľvek iné „naučil“. Opäť budeme dokola opakovať jeden protokol až kým už Vlado nebude mať dostatočnú istotu.

Priebeh jedného kola: Začína Paľo. Náhodne prečísluje vrcholy jedného z daných dvoch grafov (keďže sú izomorfné, je jedno, ktorého) a výsledok zverejní. Vlado si vyberie „challenge“ $x = 1$ alebo $x = 2$ – teda číslo grafu. Paľo mu ukáže izomorfizmus (teda vzájomné priradenie vrcholov) zverejneného grafu a G_x, Vlado si to skontroluje.

Medzi rečou, urobme drobný experiment. Prví traja riešitelia, ktorí sa dočítajú až sem a ozvú sa mi mailom na mišof zavináč ksp bodka sk (píšem to slovom, aby to nebilo do očí), majú u mňa nejakú vhodnú sladkosť. Ak to bude možné, odovzdávanie prebehne na sústredku. No ale späť k riešeniu.

Funkčnosť je opäť zjavná. Ak sú G_1 a G_2 izomorfné, Paľo nemá najmenší problém odpovedať. Uvedomte si, že ak pozná izomorfizmus medzi G_1 a G_2 (a zapamätá si, ako prečísloval vrcholy G_1 pred zverejnením), dokáže na Vladove otázky odpovedať dokonca efektívne – v lineárnom čase.

Správnosť je takisto evidentná. Nech G_1 a G_2 nie sú izomorfné. Čo môže Peter robiť, aby presvedčil Vlada, že izomorfné sú? Nech počíta, čo chce, musí časom zverejniť nejaký graf. Ten bude izomorfný s najviac jedným spomedzi G_1, G_2. Vlado má teda aspoň 50% šancu, že dá Petrovi výzvu, ktorú ten nezvládne splniť.

No a klasické triky na dokazovanie **bezznalostnosti** nás nesklamú ani teraz. Ako prvé vygenerujeme Vladove výzvy. Ako druhé vygenerujeme náhodné permutácie, ktoré akože Paľo zverejnil. No a na záver pre každú výzvu x z grafu G_x príslušnou permutáciou vyrobíme graf, ktorý akože Paľo v prvom kroku zverejnil.

Bodovanie.

Tri body za prvú časť, po šesť za zvyšné dve. Prípadne menej, podľa toho, nakoľko poriadne to bolo napísané, prípadne ako ďaleko do správnosti to malo.

To by bolo na dnes všetko, v nasledujúcej sérii sa pozrieme na kryptologickú klasiku najsamväčšiu, a to šifrovanie. Ale neznamená to, že bude ľahšia… ani že bude menej zaujímavá :-)

Výsledková listina po 2. kole kategórie KSP

	Meno a priezvisko	Škola	Trieda		21	22	23	24	25	Σ
1	Simančík František	Gym. Grösslingová BA	4	75	15	15	15	15	15	150
2	Perešíni Peter	Gym. Tajovského B. Bystrica	3	69	15	15	15	15	15	144
3	Takáč Slavomír	Gym. Nové Zámky	3	68	15	15	15	15	9	137
4	Nánási Michal	Gym. Jura Hronca BA	4	69	15	14	15	15	8	136
5	Bílka Ondřej	Gymnázium	3	52	15	12	11	15	10	115
6	Imriška Jakub	Gym. Jura Hronca BA	3	53	15	14	15	15		112
7	Bundala Daniel	Gym. Jura Hronca BA	3	56	15	10	15	15		111
8	Cicko Miroslav	Gym. Tajovského B. Bystrica	4	53	15	11	8	10	13	110
9	Petruľák Matúš	Gym. Grösslingová BA	4	59	11	10	14	11	4	109
10	Mikuláš Ján	Gym. Haličská Lučenec	3	41	15	15	15	15		101
10	Sudolský Michal	Gym. Tajovského B. Bystrica	2	42	10	10	15	11	13	101
12	Fedák Matúš	Gym. Stará Ľubovňa	3	54	11	10	15	3	6	99
12	Špalek Lukáš	Gym. Čadca	4	43	15	8	15	10	8	99
14	Bulánek Jan	Gymnázium Klatovy	4	59	11	13		15		98
15	Plžík Milan	Gym. Ľ. Štúra Zvolen	4	49	15	10	15	8		97
16	Beleš Lukáš	Gym. Čadca	4	38	11	10	15	14	6	94
17	Buštor Stano	Gym. Jura Hronca BA	4	38	10	8	12	14		82
18	Ďuďák Juraj	Gym. Golianova Nitra	3	47	11		11		10	79
19	Zeman Marek	Gym. Jura Hronca BA	4	44	15		8	11		78
20	Trnovcová Zuzana	Gym. Jura Hronca BA	4	21	15	10	14	11		71
21	Bodnár Jozef	Gym. Fiľakovo	4	15	15			15	15	60
21	Krchniak Matej	Gym. Jura Hronca BA	3	26	11		12	11		60
23	Ďurfina Lukáš	Gym. Golianova Nitra	4	22	11	9	11			53
24	Mindek Peter	Škola pre mim. nadané deti BA	3	30	9	10		3		52
25	Paulis Peter	Gym. Cyrila a Metoda Nitra	4	28	11		10			49
26	Ambrošová Lucia	Gym. Jura Hronca BA	3	27	2		15			44
27	Palenčár Ján	Gym. Malá Hora Martin	4	43						43
28	Dzurenko Miroslav	Gym. Ľ. Štúra Zvolen	4	37						37
29	Kundrák Ľubomír	Ev. lýceum BA	2	34						34
30	Dzetkulič Michal	Gym. P. Horova Michalovce	4	32						32
31	Krištof Peter	Gym. Haličská Lučenec	4	28						28
32	Beran Jakub	Gym. Alejová Košice	3	24						24
32	Domány Dušan	Gym. P. Horova Michalovce	4	24						24
34	Janík Oliver	Gym. L. Stöckela Bardejov	5	23						23
35	Okruhlica Adam	Gym. Jura Hronca BA	2	22						22
36	Kuštárová Tamara	Bilingválne gym. Sučany	4	21						21
37	Schuster Vladimír	Gym. Jura Hronca BA		10	7					17