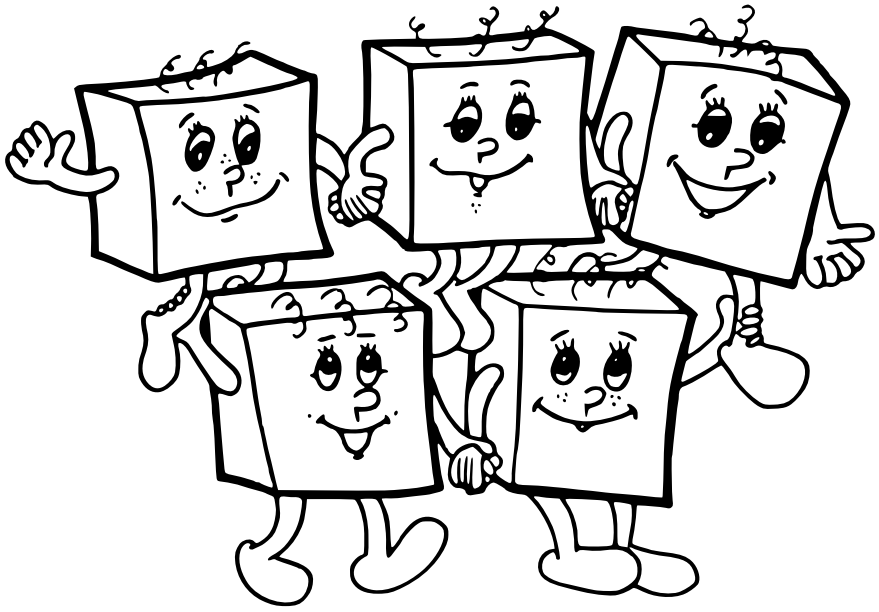
OLYMPIÁDA V INFORMATIKE
NA STREDNÝCH ŠKOLÁCH



dvadsiaty druhý ročník
školský rok 2006/07

riešenia celoštátneho kola
kategória A

2. súťažný deň

- **Olympiáda v informatike** je od školského roku 2006/07 samostatnou súťažou. Predchádzajúcich 21 ročníkov tejto súťaže prebiehalo pod názvom **Matematická olympiáda, kategória P** (programovanie).
- Oficiálnu **webstránku** súťaže nájdete na `http://www.ksp.sk/oi/`.

Riešenia druhého súťažného dňa

A-III-4 Polícia zasahuje

Na začiatku si všimnime, že mapa kanalizácie predstavuje strom (čiže súvislý graf bez kružníc). Označme tento strom T. Môžeme si zvoliť ľubovoľný vrchol u a v ňom tento strom zakoreniť – odteraz bude pre nás u koreňom daného stromu T. Môžete si to predstaviť tak, že strom chytíme za vrchol u a zavesíme ho zaň na stenu.

V ďalšom texte budeme označovať T_v podstrom s koreňom v. To je tá časť stromu, ktorá by odpadla, keby sme vrchol v odstránili.

Vrchol v budeme nazývať *bezpečný*, ak sa žiaden mafián, ktorý býva v podstrome T_v, nevie dostať nestráženou cestou do vrcholu v.

Náš algoritmus bude postupovať smerom od listov stromu T k jeho koreňu u. Keď budeme spracúvať nejaký vrchol v, budeme pre neho chcieť spočítať dve veci:

- Aký je minimálny počet strážcov $S(v)$ potrebný na pokrytie podstromu T_v tak, aby sa žiadni dvaja mafiáni z neho nemohli stretnúť,
- a tiež, či vieme týmto počtom strážcov navyše dosiahnuť, aby bol vrchol v bezpečný.

Pre listy je situácia jednoduchá: V liste sa môže nachádzať najviac jeden mafián, a teda nie je potrebné ho oddeľovať. Počet potrebných strážcov je teda 0. Tento vrchol je bezpečný práve vtedy, ak v ňom nie je mafián.

Nech teraz náš algoritmus ide spracovať vrchol v. Označme si jeho priamych potomkov $v_1, \ldots, v_k$. Pre každý z podstromov $T_{v_1}, \ldots, T_{v_k}$ máme už spočítaný minimálny potrebný počet strážcov a aj to, či je pri ich použití vrchol v_i bezpečný.

Označme $S = S(v_1) + \cdots + S(v_k)$. Optimálne riešenie pre T_v je určite väčšie alebo rovné ako S (teda súčet optimálnych riešení pre jednotlivé podstromy), lebo v každom z týchto podstromov musíme od seba oddeliť všetkých mafiánov.

My ale navyše musíme oddeliť od seba aj mafiánov v rôznych podstromoch, a možno ešte aj mafiána vo vrchole v. Ukážeme si, ako na to. Rozoberieme dva prípady.

Vo vrchole v je mafián. Ak majú všetky podstromy bezpečné optimálne riešenie, stačí nám na celý strom T_v použiť S strážcov. Lepšie to zjavne nejde. Výsledné rozmiestnenie strážcov určite nebude bezpečné, lebo mafián bývajúci vo v sa do v dostať vie.

V opačnom prípade musíme od seba oddeliť mafiána vo vrchole v a mafiánov v nebezpečných podstromoch. Musíme teda pridať aspoň jedného strážcu. Zjavne stačí pridať práve jedného strážcu, a to do vrcholu v. Takto teda dostaneme riešenie s $S + 1$ strážcami. Toto riešenie je určite bezpečné.

Vo vrchole v nie je mafián. Ak je najviac jeden podstrom, ktorý nemá bezpečné optimálne riešenie, stačí nám na celý strom T_v použiť S strážcov. Lepšie to zjavne nejde. Ak boli všetky podstromy bezpečné, bude bezpečné aj celkové rozmiestnenie, inak nie. (Ten istý mafián, ktorý to kazil v nebezpečnom podstrome, to kazí naďalej.)

V opačnom prípade musíme od seba oddeliť (aspoň dvoch) mafiánov v nebezpečných podstromoch. Musíme teda pridať aspoň jedného strážcu. Zjavne stačí pridať práve jedného strážcu, a to do vrcholu v. Takto teda dostaneme riešenie s $S + 1$ strážcami. Toto riešenie je určite bezpečné.

Keď takto nakoniec spracujeme vrchol, ktorý sme si zvolili ako koreň stromu, dostaneme optimálny počet strážcov. Ten vypíšeme na výstup a sme hotoví.

Spracovať jeden konkrétny vrchol vieme v čase priamo úmernom počtu jeho podstromov, čiže počtu hrán, ktoré z neho vychádzajú. Celkovo má teda tento algoritmus časovú zložitosť priamo úmernú súčtu stupňov vrcholov. Ale súčet stupňov vrcholov nie je nič iné ako dvojnásobok počtu hrán. (Každá hrana je zarátaná v dvoch stupňoch vrcholov.)

No a keďže zadaný graf je strom a má $N - 1$ hrán, je časová zložitosť tohto algoritmu $O(N)$.

Iné riešenie

Táto úloha má aj iné lineárne riešenia, sú však náročnejšie na implementáciu ako vzorové riešenie. Uvedieme myšlienku jedného z nich.

Ak máme list stromu, v ktorom nebýva mafián, môžeme tento list a hranu, ktorá doň vedie, zahodiť, hodnotu optimálneho riešenia to nezmení. Podobne ak máme vrchol, z ktorého vedú dve hrany a v ktorom nebýva mafián, môžeme ho z grafu odstrániť a jeho dvoch susedov spojiť priamou hranou.

Ak už sa v nejakom okamihu žiadna z týchto operácií nedá spraviť, musí existovať vrchol, ktorého najviac jeden sused nie je list. (Dôkaz a návod na hľadanie: Ak by sme si strom zakorenili, stačí zobrať otca niektorého najhlbšieho listu.)

Rozmyslite si, že do tohto vrcholu potom musíme umiestniť strážnika. Tým sme sa ale zbavili tohto vrcholu a všetkých listov, ktoré s ním susedia. Vyhodíme ich z grafu a pokračujeme vyššie popísaným postupom ďalej, až kým už nie je čo riešiť.

Listing programu:

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

#define MAXN 100000

FILE *in, *out;         /* Vstupní a výstupní soubor */
int n, p;               /* Počet větvení a počet mafiánů */
int deg[MAXN];          /* Počet stok vedoucích z větvení */
int e[MAXN][2];         /* Seznam stok */
int edge_ptr[MAXN];     /* Index v neib, kde začínají sousedé každého větvení */
int neib[2*MAXN];       /* Seznamy sousedů větvení */
char podezrely[MAXN];   /* Je do daného větvení připojen dům mafiána? */
int hlidek;             /* Potřebný počet hlídek */

void nacti_vstup(void) {
  int i, a, b;
  int tmp_ptr[MAXN];

  /* Načteme stoky */
  fscanf(in, "%d %d", &n, &p);
  memset(deg, 0, n*sizeof(int));
  for (i = 0; i < n-1; i++) {
    fscanf(in, "%d %d", &a, &b);
    a--; b--;
    e[i][0] = a;
```

```
    e[i][1] = b;
    deg[a]++;
    deg[b]++;
  }

  /* Vytvoříme seznamy sousedů a nastavíme správně jejich počátky */
  edge_ptr[0] = 0;
  for (i = 1; i < n; i++)
    edge_ptr[i] = edge_ptr[i-1]+deg[i-1];
  memset(tmp_ptr, 0, n*sizeof(int));
  for (i = 0; i < n-1; i++) {
    neib[edge_ptr[e[i][0]]+tmp_ptr[e[i][0]]++] = e[i][1];
    neib[edge_ptr[e[i][1]]+tmp_ptr[e[i][1]]++] = e[i][0];
  }

  /* Načteme seznam mafiánů */
  for (i = 0; i < p; i++) {
    fscanf(in, "%d", &a);
    a--;
    podezrely[a] = 1;
  }
}

/* Spočte počet hlídek pro T_v, vrátí 1, pokud je T_v nehlídaná. */
int hledej(int v, int rodic) {
  int i, nehlidanych = 0;

  /* Projdeme všechny sousedy, ze kterých k nám vede stoka, a zjistíme situaci */
  for (i = 0; i < deg[v]; i++)
    if (neib[edge_ptr[v]+i] != rodic)
      nehlidanych += hledej(neib[edge_ptr[v]+i], v);
  /* Musíme větvení v hlídat? */
  /* To je třeba buď pokud máme alespoň 2 nehlídané sousedy (pak je třeba
   * izolovat mafiány z nich), nebo pokud ve 'v' sídlí mafián a máme alespoň
   * jednoho nehlídaného souseda. */
  if (nehlidanych > 1 || (podezrely[v] && nehlidanych > 0)) {
      hlidek++;
      return 0;
  }
  /* Jinak nemusíme hlídat, jen vrátíme, jestli je T_v nehlídaný */
  return nehlidanych || podezrely[v];
}

int main(void) {
  in = fopen("policie.in", "r");
  out = fopen("policie.out", "w");
  nacti_vstup();
  hledej(0,0);
  fprintf(out, "%d\n", hlidek);
  return 0;
}
```

A-III-5 Rybka Julka

Najjednoduchším správnym riešením by bolo postupne pre každý čas $t = 1, 2, 3, \ldots$ spočítať množinu všetkých políčok, kde sa Julka môže v danom čase nachádzať.

Ak M_t je množina tých políčok, kde mohla byť Julka v čase t, množinu M_{t+1} ľahko zostrojíme z množiny M_t. Stačí pridať tie políčka, ktoré susedia s nejakým políčkom z M_t a majú teplotu takú, že tam Julka smie v $(t+1)$-ej sekunde preplávať. A naopak, vyhodíme tie políčka, ktoré už sú pre Julku príliš horúce.

Časom buď narazíme na také T, že M_T bude obsahovať cieľové políčko, alebo zistíme, že aktuálna množina M_T je prázdna. V prvom prípade sme práve našli najrýchlejšiu cestu do cieľa, v druhom sa nám práve Julka uvarila.

Takéto riešenie vieme ľahko implementovať tak, aby malo časovú zložitosť $O(RST)$, kde R, S sú rozmery rybníka a T je čas, kedy prestaneme hľadať.

Myšlienka efektívnejšieho riešenia

Lepšie riešenie bude založené na nasledujúcom pozorovaní: Pre každé políčko nám stačí vedieť, ako najrýchlejšie (a kadiaľ) sa naň vieme dostať. Totiž ak sa rozhodneme, že naša trasa povedie cez nejaké konkrétne políčko, nemá zmysel prísť tam inou ako najrýchlejšou cestou.

Pre každé políčko $[r, s]$ budeme teda chcieť spočítať hodnotu $c(r, s)$ – najmenší počet sekúnd, po ktorom vieme byť na tomto políčku. (Navyše si budeme pamätať aj smer, odkiaľ sem vedie jedna možná najrýchlejšia cesta.)

Na počítanie týchto hodnôt použijeme Dijkstrov algoritmus na hľadanie najkratšej cesty v grafe.

Myšlienka je nasledovná: Všetky možné políčka budeme mať rozdelené do dvoch množín: $\mathcal{H}$ budú hotové políčka, pre ktoré už vieme správnu hodnotu $c(r, s)$, a $\mathcal{S}$ budú ešte spracúvané políčka.

Pre každé políčko $[r, s]$ v $\mathcal{S}$ bude aktuálna hodnota $c(r, s)$ predstavovať najmenší čas, v akom sa vieme na políčko $[r, s]$ dostať, *ak ideme len cez políčka v* $\mathcal{H}$. (Prípadne $c(r, s) = \infty$, ak žiadna takáto cesta neexistuje).

Na začiatku nie je hotové žiadne políčko, teda $\mathcal{H} = \emptyset$. Zjavne $c(J_r, J_s) = 0$ a pre ostatné políčka je $c(r, s) = \infty$.

Zvyšok algoritmu bude prebiehať v kolách. V každom kole nájdeme v množine $\mathcal{S}$ jedno políčko, o ktorom vieme dokázať, že čas najkratšej cesty doň je už spočítaný správne. Toto políčko zakaždým presunieme do $\mathcal{H}$ a upravíme najlepšie časy pre políčka, ktoré zostali v $\mathcal{S}$. Akonáhle sa cieľové políčko dostane medzi hotové, môžeme skončiť.

Ako teda nájsť v množine $\mathcal{S}$ políčko, ktoré už môžeme presunúť medzi hotové? Jednoducho vyberieme z množiny $\mathcal{S}$ to políčko $[r, s]$, ktoré má najmenšiu hodnotu $c(r, s)$. (Ak by takých bolo viac, tak ľubovoľné z nich.) Tvrdíme, že hodnota $c(r, s)$ pre toto políčko je už správna.

Prečo? Všimnime si všetky možné spôsoby, ako sa dostať na políčko $[r, s]$. Už sme zobrali do úvahy všetky cesty, ktoré používajú len políčka z $\mathcal{H}$. Každá zo zvyšných ciest teda musí ísť cez aspoň jedno iné políčko $[r', s'] \in \mathcal{S}$. Lenže vieme, že už $c(r', s') \geq c(r, s)$, a teda žiadna takáto cesta nebude rýchlejšia ako $c(r, s)$.

Políčko $[r, s]$ teda prehlásime za hotové a presunieme ho z $\mathcal{S}$ do $\mathcal{H}$.

Teraz ešte zostáva upraviť hodnoty $c(r, s)$ pre políčka, ktoré zostali v $\mathcal{S}$. To je ale jednoduché: Jediné políčka, kde sa niečo mohlo zmeniť, sú susedia políčka $[r, s]$. Tam pribudla možnosť, že najrýchlejší spôsob, ako sa dostať na takéto políčko, je práve cez $[r, s]$.

Jednoducho teda prejdeme všetkých (nanajvýš štyroch) susedov políčka $[r, s]$. Pre každého suseda $p = [r_p, s_p]$: Ak p patrí do $\mathcal{H}$, nič sa nedeje. Ak patrí do $\mathcal{S}$, nová hodnota $c(r_p, s_p)$ bude minimum z doterajšej hodnoty a z hodnoty $c(r, s) + t$, kde t je čas potrebný na presun z $[r, s]$ na $[r_p, s_p]$.

(Čas t potrebný na presun závisí od $c(r, s)$ a od teplôt oboch políčok. Z týchto údajov ho vieme ľahko vypočítať v konštantnom čase.)

Implementácia efektívnejšieho riešenia

Sú v princípe dva rôzne spôsoby, ako Dijkstrov algoritmus implementovať.

Ten jednoduchší a priamočiarejší vyzerá nasledovne: použijeme dve polia rozmerov rovnakých ako má rybník. V jednom si pre každé políčko budeme

pamätať, či už je hotové, v druhom aktuálne hodnoty $c(r, s)$.

Nájsť nové políčko, ktoré treba presunúť do $\mathcal{H}$, vieme v čase $O(RS)$ tak, že prezrieme všetky políčka a nájdeme najlepšie z nich. Zvyšok kola, teda upraviť hodnoty c pre susedov, už vieme v konštantnom čase.

Keďže v každom kole presunieme do $\mathcal{H}$ jedno políčko, kôl bude rádovo toľko ako políčok. Celková časová zložitosť takéhoto riešenia je teda $O(R^2S^2)$.

Je zjavné, že najpomalšou časťou predchádzajúcej implementácie je výber nového hotového políčka. Zbytočne znova a znova prechádzame všetky políčka. Nešlo by to lepšie?

Čo by to bola za rečnícka otázka, keby odpoveď nebola „áno“. Použijeme dátovú štruktúru nazývanú halda. (Podrobný popis haldy nebudeme uvádzať, v prípade záujmu ho nájdete napr. v Programátorskej liahni, rovnako ako aj podrobnejší popis tejto implementácie Dijkstrovho algoritmu.)

V halde si budeme pamätať políčka množiny $\mathcal{S}$, zoradené podľa doteraz zisteného času cesty do nich. Takto vieme nové hotové políčko vybrať v čase $O(\log(RS))$.

Keď teraz získame pre nejakého suseda nový, lepší čas, jednoducho vložíme do haldy pre tohto suseda nový záznam s týmto časom. (Netrápi nás, že ten starý tam niekde zostane, lebo ten nový vyberieme skôr.) Každú takúto úpravu vieme teda spraviť v čase $O(\log(RS))$.

Podarilo sa nám teda vylepšiť trvanie jedného kola z $O(RS)$ na $O(\log(RS))$, a tým časovú zložitosť celého algoritmu na $O(RS \log(RS))$.

Program uvedený v listingu namiesto nami uvedeného triku na upravovanie vzdialeností susedov používa inú metódu: pamätá si, kde v halde sa nachádza záznam pre ktoré políčko, a keď sa nejakému políčku zmenší vzdialenosť, „prebuble“ týmto políčkom v halde dohora.

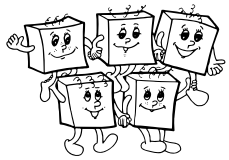
Listing programu:

```
program rybka;

const   MAXM=1000;
        MAXN=1000;
        INFTY=3000000;

type    PPole=^TPole;
        TPole=record
                x,y:longint;
                halda:longint;
                t,c:longint;
                zpole:PPole;
        end;

var     m,n,x1,y1,x2,y2,t1,t2:longint;          { zadané parametry }
        rybnik:array[1..MAXM,1..MAXN] of TPole; { pole rybníka }
        halda:array[1..MAXM*MAXN] of PPole;     { halda v poli }
        velhaldy:longint;

procedure prehod(pos1:longint; pos2:longint);   { přehoď v haldě dvě pole }
var tmp:PPole;
begin
        halda[pos1]^.halda:=pos2;
        halda[pos2]^.halda:=pos1;
        tmp:=halda[pos1];
        halda[pos1]:=halda[pos2];
        halda[pos2]:=tmp;
end;

procedure upravhaldu(var pole:TPole);    { sniž hodnotu c(x,y) pole v haldě }
var     hpos:longint;
begin
        hpos:=pole.halda;
        while (hpos>1) and (halda[hpos]^.c < halda[hpos div 2]^.c) do begin
                prehod(hpos,hpos div 2);
                hpos:=hpos div 2;
        end;
end;

function vyberzhaldy:PPole;         { vyber z haldy pole s nejnižším c(x,y) }
var     hpos,minpos:longint;
        hotovo:boolean;
begin
        if velhaldy=0 then
                vyberzhaldy:=nil
        else begin
                vyberzhaldy:=halda[1];
                halda[1]:=halda[velhaldy];
                halda[1]^.halda:=1;
                dec(velhaldy);
                hotovo:=false;
                hpos:=1;
                repeat
                        { buď už jsme na spodku haldy }
```

```
                        if (hpos*2>velhaldy) then hotovo:=true
                        { nebo máme jen jednoho potomka }
                        else if (hpos*2=velhaldy) then begin
                                hotovo:=true;
                                if (halda[hpos]^.c>halda[hpos*2]^.c) then
                                        prehod(hpos,hpos*2);
                        end
                        { nebo máme dva potomky }
                        else begin
                                minpos:=hpos;
                                if (halda[minpos]^.c>halda[hpos*2]^.c) then
                                        minpos:=hpos*2;
                                if (halda[minpos]^.c>halda[hpos*2+1]^.c) then
                                        minpos:=hpos*2+1;
                                if (minpos=hpos) then
                                        hotovo:=true
                                else begin
                                        prehod(hpos,minpos);
                                        hpos:=minpos;
                                end;
                        end;
                until hotovo;
        end;
end;

procedure vypis(var f:text; var pole:TPole);
begin
        if pole.zpole<>nil then begin
                vypis(f,pole.zpole^);
                if pole.c-pole.zpole^.c>1 then
                        write(f,pole.c-pole.zpole^.c-1,' ');
                if pole.x>pole.zpole^.x then write(f,'V ');
                if pole.x<pole.zpole^.x then write(f,'Z ');
                if pole.y>pole.zpole^.y then write(f,'J ');
                if pole.y<pole.zpole^.y then write(f,'S ');
        end;
end;

procedure vylepsipole(var ktere:TPole; var odkud:TPole);
var cek:longint;
begin
        { jak dlouho musím počkat, než se pole dost ohřeje? }
        cek:=t1-(odkud.c+ktere.t);
        if cek<0 then cek:=0;
        { jedná se o cílové pole? pokud ano, vůbec nečekej }
        if (ktere.x=x2) and (ktere.y=y2) then begin
                ktere.c:=odkud.c+1;
                ktere.zpole:=@odkud;
                upravhaldu(ktere);
        end
        { když počkám tolik, bude to ok na tomto i cílovém poli?
          je to lepší cesta? }
        else if (odkud.c+cek+odkud.t<=t2) and (odkud.c+cek+ktere.t+1<=t2)
             and (ktere.c>odkud.c+cek+1) then begin
                ktere.c:=odkud.c+cek+1;
                ktere.zpole:=@odkud;
```

```
                upravhaldu(ktere);
        end;
end;

var     min:PPole;
        hotovo:boolean;
        f1,f2:text;
        i,j:longint;
begin
        assign(f1,'rybka.in');    reset(f1);
        assign(f2,'rybka.out');   rewrite(f2);
        readln(f1,n,m,t1,t2);
        readln(f1,y1,x1,y2,x2);
        for j:=1 to n do
                for i:=1 to m do begin
                        read(f1,rybnik[i,j].t);
                        rybnik[i,j].x:=i;
                        rybnik[i,j].y:=j;
                        rybnik[i,j].c:=INFTY;
                        rybnik[i,j].zpole:=nil;
                        rybnik[i,j].halda:=velhaldy+1;
                        halda[velhaldy+1]:=@rybnik[i,j];
                        inc(velhaldy);
                end;
        rybnik[x1,y1].c:=0;
        upravhaldu(rybnik[x1,y1]);

        { hlavní smyčka výběru z haldy }
        hotovo:=false;
        repeat
                min:=vyberzhaldy;
                { buď už v haldě nic k přidání není }
                if (min=nil) or (min^.c=INFTY) then begin
                        writeln(f2,'Chudinka Julka!');
                        hotovo:=true;
                end
                { nebo vybírám cílové pole }
                else if (min^.x=x2) and (min^.y=y2) then begin
                        hotovo:=true;
                        vypis(f2,rybnik[x2,y2]);       { Ať žije Julka! }
                end
                { nebo je to nějaké obecné pole }
                else begin
                        if min^.x>1 then
                                vylepsipole(rybnik[min^.x-1,min^.y],min^);
                        if min^.y>1 then
                                vylepsipole(rybnik[min^.x,min^.y-1],min^);
                        if min^.x<M then
                                vylepsipole(rybnik[min^.x+1,min^.y],min^);
                        if min^.y<N then
                                vylepsipole(rybnik[min^.x,min^.y+1],min^);
                end;
        until hotovo;
        close(f1); close(f2);
end.
```

Olympiáda
v Informatike



2006/07
celoštátne kolo, deň 2
riešenia kategórie A

SLOVENSKÁ KOMISIA OLYMPIÁDY V INFORMATIKE

DVADSIATY DRUHÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Vydala IUVENTA s finančnou podporou Ministerstva školstva SR
Náklad: 40 výtlačkov
Zodpovedný redaktor: Michal Forišek
Sadzba programom LaTeX