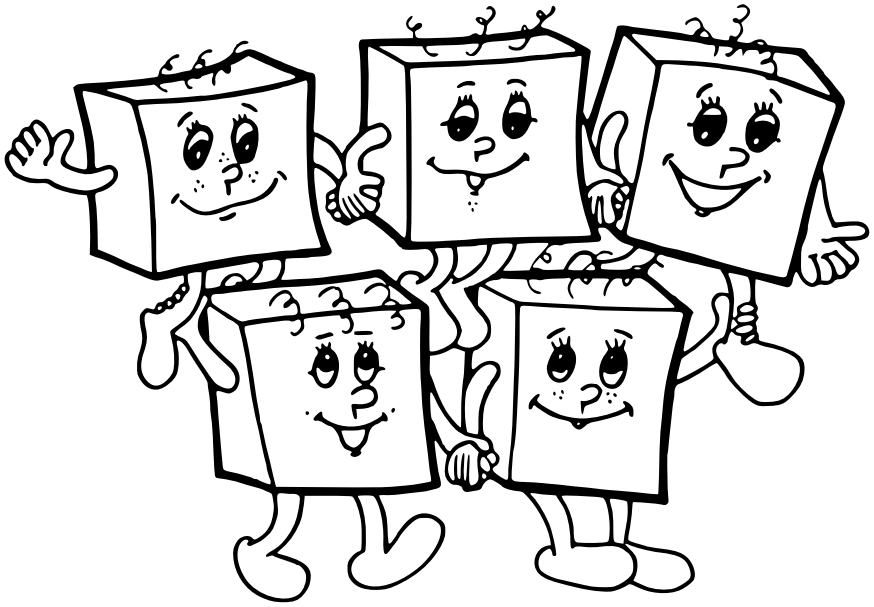
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH



dvadsiaty piaty ročník

školský rok 2009/10

riešenia domáceho kola
kategória B

- **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://oi.sk/`.

Riešenia kategórie B

B-I-1 Majstrovstvá na rybníku

Pre každého súťažiaceho vieme jeho výsledné skóre zistiť rovno pri načítaní vstupu. Najjednoduchšie je spraviť si pomocné pole veľkosti R, do ktorého načítame všetky známky daného súťažiaceho. Potom pre toto pole postupne zistíme súčet, maximum aj minimum a z nich už ľahko určíme výsledné skóre.

(Za zmienku stojí, že sa zaobídeme aj bez takéhoto pomocného poľa. Vystačíme si s tromi premennými. V dvoch z nich si budeme pamätať doteraz najmenšiu a doteraz najväčšiu známku, ktorú aktuálny súťažiaci dostal, a v tretej bude súčet ostatných už spracovaných známok.)

Po dočítaní vstupu teda máme S usporiadaných dvojíc (skóre, meno), ktoré máme utriediť.

V takejto situácii je väčšinou dobré (kvôli prehľadnosti programu) „zabaliť“ si údaje, ktoré ideme triediť, do vhodného dátového typu – napr. `record` v Pascale alebo `struct` v C/C++.

Následne na utriedenie týchto záznamov použijeme nejaký štandardný triediaci algoritmus. V C máme k dispozícii funkciu `qsort`, v C++ funkciu `sort`. Obe tieto funkcie implementujú dobré algoritmy na triedenie – utriedia N prvkov v čase $O(N \log N)$. V Pascale bolo potrebné naprogramovať si vlastné triedenie. V tomto riešení si popíšeme HeapSort – triedenie pomocou haldy.

Listing programu:

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

struct sutaziaci { string meno; int body; };

bool operator< (const sutaziaci &A, const sutaziaci &B) {
  if (A.body != B.body) return A.body > B.body; else return A.meno < B.meno;
}

int S, R, lo, hi, cur;

int main() {
  cin >> S >> R;
  vector<sutaziaci> vysledky(S);
  for (int s=0; s<S; ++s) {
    vysledky[s].body = 0;
    cin >> vysledky[s].meno >> lo >> hi;
    if (lo > hi) swap(lo,hi);
    for (int r=0; r<R-2; ++r) {
      cin >> cur;
      if (cur < lo) swap(lo,cur);
      if (cur > hi) swap(hi,cur);
      vysledky[s].body += cur;
    }
  }
  sort( vysledky.begin(), vysledky.end() );
  for (int s=0; s<S; ++s) cout << vysledky[s].body << ": " << vysledky[s].meno << endl;
}
```

Halda

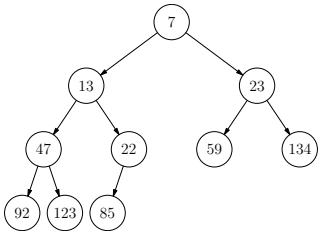
Predstavme si, že máme čiernu krabičku s tlačidlom, do ktorej vieme vhadzovať čísla a z ktorej po každom stlačení tlačidla vypadne najmenšie z čísel, ktoré sú práve v nej.

Takáto krabička vie byť celkom užitočná. Pomocou nej by sa nám napríklad ľahko triedilo – nahádžeme všetky čísla do nej, no a potom už len stláčame tlačidlo, kým postupne nevypadnú všetky v utriedenom poradí.

Jednou šikovnou implementáciou takejto krabičky je halda. Je to vlastne binárny strom, ktorý má každé poschodie úplne plné, možno okrem toho posledného, najspodnejšieho. Každý prvok, okrem tých najspodnejších,

má teda práve dvoch synov. Prvky musia byť usporiadané tak, aby platilo, že hodnota uložená v ľubovoľnom vrchole je menšia alebo rovná ako každá z hodnôt uložená v jeho synoch (ak nejakých má).

Takto môže vyzerať halda:



Všimnime si zaujímavú vlastnosť haldy: najmenšie číslo je určite v jej koreni. O ostatných číslach už toho tak veľa nevieme povedať – všimni si, že napríklad 22 je hlbšie ako 23.

Na to, že halda je strom, môžeme zase šťastne zabudnúť. Haldu si totiž vieme úplne jednoducho pamätať v poli. Stačí si vrcholy očíslovať po vrstvách. Na políčku s číslom x teda bude prvok, ktorý by sme prečítali ako x-tý, keby sme haldu „čítali po riadkoch“.

Všimnime si niektoré vlastnosti takto uloženej haldy:

- Ak je v halde N prvkov, v poli budú na políčkach 1 až N.
- Koreň haldy, teda najmenší prvok v nej, je na políčku s čísom 1.
- K-ta vrstva haldy sa v poli začína na políčku s indexom 2^{K-1}.
- Synovia vrcholu s číslom x majú vždy čísla $2x$ a $2x + 1$.
- A opačne, otec vrcholu s číslom x má číslo $\lfloor x/2 \rfloor$.

Halda z obrázku by vyzerala v poli takto:

1	2	3	4	5	6	7	8	9	10	11	12	
7	13	23	47	22	59	134	92	123	85			

Ukážeme teraz, ako do haldy efektívne pridávať nové čísla a ako z nej efektívne vyberať najmenšie.

Vloženie prvku

Ako teda vložíme nejaký prvok do haldy? Zaradíme ho do poľa na prvé voľné miesto. Jediné, čo nám teraz môže kaziť „haldovitosť“, je, že tento prvok môže byť menší od prvku nad ním. V tom prípade túto dvojicu prvkov vymeníme. Rozmyslite si, že opäť môže nastať jediný problém: nový prvok môže byť menší aj od svojho nového otca. V takomto prípade postup opakujeme. (Tomuto sa hovorí „bublanie prvku dohora“.)

Tento postup určite skončí, prinajhoršom vtedy, ked sa nový prvok dostane na úplný vrch haldy – do koreňa. Po jeho skončení je opäť celá halda v poriadku, úspešne sme teda vložili nový prvok.

Výber najmenšieho prvku

A čo s vybratím najmenšieho prvku? To bude fungovať podobne. Vieme, že najmenší prvok je ten na vrchu haldy. Tentokrát ho ale nemôžeme len tak odstrániť, vznikla by nám tam totiž diera. No a tú treba niečím zaplniť. Najjednoduchšie riešenie: Zoberieme posledný prvok v poli (t. j. najpravejší list v poslednej vrstve) a presunieme ten na začiatok poľa.

Touto zmenou sme opäť dosiahli, že čísla máme uložené na prvých niekoľkých políčkach poľa. Nemusí to ale ešte byť korektná halda. To, čo nám ju môže kaziť, je práve presunutý prvok. Preto zopakujeme niečo podobné, ako pri vkladaní. Tentokrát ale presunutý prvok môže byť len priveľký, preto ho budeme musieť „prebublať“ dodola. Toto bublanie treba robiť trochu šikovnejšie. Tentokrát totiž náš „zlý“ prvok môže mať až dvoch synov a byť väčší od každého z nich. Hravo ale zistíme, že riešenie je jednoduché: stačí ho vymeniť s menším z oboch synov.

Opäť, tento postup je konečný. Skončíme, ak už sú obidvaja aktuálni synovia väčší alebo rovní presunutému prvku, prípadne ak sa náš prvok prebublal až do najspodnejšej vrstvy. V každom prípade máme opäť korektnú haldu.

Odhad zložitosti

Všimnime si, čo sa deje pri jednom prebublaní prvku. Pri vkladaní prebubleme v najhoršom z poslednej vrstvy až do koreňa, pri vyberaní minima naopak, z koreňa až po najhlbšiu vrstvu. V obidvoch prípadoch je počet operácií úmerný hĺbke stromu, teda počtu vrstiev. A aká je tá hĺbka? Na zaplnenie K vrstiev haldy potrebujeme $2^K - 1$ prvkov, preto halda s N prvkami má približne $\log_2 N$ vrstiev. Každá operácia s haldou, v ktorej je N prvkov, má teda časovú zložitosť $O(\log N)$.

Triedenie pomocou haldy

Ako sme už spomínali na začiatku, pomocou haldy vieme ľahko napísať triedenie, známe pod menom HeapSort. (Heap je halda po anglicky.) Pri triedení N prvkov najskôr N-krát vložíme prvok do haldy, potom odtiaľ N-krát vyberieme najmenší. Počas celého tohto procesu nie je nikdy v halde viac ako N prvkov, preto časová zložitosť každej operácie s ňou je $O(\log N)$. Týchto operácií je $2N$, preto je výsledná časová zložitosť HeapSortu $O(N \log N)$. Pamäťová zložitosť haldy aj triedenia pomocou nej je samozrejme $O(N)$.

Listing programu:

```
type sutaziaci = record
                    meno: string[12];
                    body: longint;
                 end;

var S, R : longint;
    vysledky : array[1..1000047] of sutaziaci;
    i, j, max, min, cur : longint;

procedure swap(var x, y : longint);
var z : longint;
begin z:=x; x:=y; y:=z; end;

procedure swap(var x, y : sutaziaci);
var z : sutaziaci;
begin z:=x; x:=y; y:=z; end;

function horsi(var x, y : sutaziaci) : boolean;
begin
   if x.body <> y.body then horsi:=(x.body < y.body) else horsi:=(x.meno > y.meno);
end;

procedure insert(i : longint);
begin
   while (i>1) and (horsi(vysledky[i],vysledky[i div 2])) do begin
      swap( vysledky[i div 2], vysledky[i] );
      i := i div 2;
   end;
end;

procedure extract(pocet : longint);
var i, j : longint;
begin
   swap( vysledky[1], vysledky[pocet] );
   dec(pocet);
   i := 1;
   while true do begin
      j := i;
      if (2*i <= pocet) and (horsi(vysledky[2*i],vysledky[j])) then j:=2*i;
      if (2*i+1 <= pocet) and (horsi(vysledky[2*i+1],vysledky[j])) then j:=2*i+1;
      if j=i then break;
      swap( vysledky[i], vysledky[j] );
      i := j;
   end;
end;

begin
   { nacitame pocet sutaziacich a pocet rozhodcov }
```

```
   readln(S,R);

   { pre kazdeho sutaziaceho:
     * nacitame znamky ktore dostal
     * pocas nacitavania si pamatame doteraz najmensiu, najvacsiu a sucet ostatnych }
   for i:=1 to S do begin
      readln(vysledky[i].meno);
      vysledky[i].body := 0;
      read(max);
      read(min);
      if (max < min) then swap(max,min);
      for j:=3 to R do begin
         read(cur);
         if (cur > max) then swap(max,cur);
         if (cur < min) then swap(min,cur);
         vysledky[i].body := vysledky[i].body + cur;
      end;
      readln;
   end;

   { utriedime a vypiseme vysledky }
   for i:=2 to S do insert(i);
   for i:=S downto 2 do extract(i);
   for i:=1 to S do writeln(vysledky[i].body,': ',vysledky[i].meno);
end.
```

B-I-2 Cesta

Najjednoduchším riešením tejto úlohy bolo odpovedať na každú otázku simuláciou: Prejdeme po celej ceste od začiatku po koniec, úsek po úseku. Keď nájdeme úsek, kde leží x, začať merať čas a pri y zase prestaneme a vypíšeme odpoveď. Takéto riešenie má časovú zložitosť $O(NQ)$ – pre každú z Q otázok prejdeme v najhoršom prípade všetkých N úsekov cesty.

Lepšie riešenie je založené na nasledujúcom pozorovaní: čas cesty z bodu x do bodu y vieme vypočítať ako rozdiel dvoch časov: času cesty z 0 do y a času cesty z 0 do x.

Ak je celková dĺžka celej cesty D malá, môžeme spraviť jednoduchý trik. Rozdelíme si cestu na D úsekov dĺžky 1 a následne postupne pre každé q od 0 po D spočítame hodnotu c_q: čas, za ktorý sa vieme dostať z bodu 0 do bodu q. Tieto hodnoty spočítame ľahko – zjavne $c_0 = 0$, no a ak poznáme c_q, tak vieme vypočítať c_{q+1} ako $c_q + 1/v_q$, kde v_q je maximálna povolená rýchlosť na úseku medzi bodmi q a $q + 1$.

Keď máme spočítané hodnoty c_q, vieme na každú otázku odpovedať v konštantnom čase – pre ľubovoľné x, y vieme čas cesty z x do y vyjadriť ako $c_y - c_x$.

Takéto riešenie má časovú zložitosť $O(N + D + Q)$.

Toto riešenie vieme pre obmedzenia dané v zadaní „hackersky“ vylepšiť do podoby, v ktorej získa 9 alebo 10 bodov. Vieme si napríklad cestu namiesto úsekov dĺžky 1 rozdeliť na úseky dĺžky 1000, pre každý úsek si zapamätať zmeny rýchlosti, ktoré sa na ňom udejú, a hodnoty c_q si pamätať len pre násobky 1 000. V takto upravenom riešení nám na zodpovedanie ľubovoľnej otázky vystačí nanajvýš pár tisíc operácií.

Vzorové riešenie, ktoré teraz popíšeme, je pravdepodobne o niečo ľahšie na implementáciu. Opäť budeme používať myšlienku o rozdiele dvoch časov. Tentokrát však nebudeme cestu nijak deliť, a jednoducho spočítame hodnoty $T[i]$: čas potrebný na prechod od začiatku cesty po koniec i-teho úseku.

Čo robiť, ak miesto, v ktorom chceme skončiť, neleží presne na hranici úseku? Stačí nám vedieť, v ktorom úseku sa koniec našej cesty nachádza a aká je vzdialenosť medzi začiatkom tohto úseku a koncom našej cesty.

Ešte raz a poriadnejšie. Položme $p_0 = t_0 = 0$ a pre všetky i nech je $p_{i+1} = p_i + d_i$ a $t_{i+1} = t_i + d_i/v_i$. Teda p_i je vzdialenosť od začiatku cesty po koniec i-teho úseku a t_i je čas, za ktorý na toto miesto vieme doraziť. Hodnoty p_i a t_i vieme spočítať v čase $O(N)$ použitím vyššie uvedeného vzťahu.

Teraz ukážeme, ako pre daný bod x zistiť čas, za ktorý vieme prísť od začiatku cesty k nemu. Toto spravíme v dvoch krokoch. V prvom kroku nájdeme úsek, v ktorom x leží – teda nájdeme i, pre ktoré $p_{i-1} < x \leq p_i$. V druhom kroku spočítame samotný čas cesty ako $t_{i-1} + (x - p_{i-1})/v_i$.

Ako efektívne realizovať prvý krok? Použijeme *binárne vyhľadávanie*. Vieme, že hľadané i je z množiny $\{1, \ldots, n\}$. Túto množinu budeme dokola deliť na polovicu, až kým nezostane len jedna hodnota – tá správna.

Predpokladajme, že vieme, že pre nejaké a a b platí $p_a < x \leq p_b$, pričom $b - a > 1$. V takejto situácii zoberieme $c = \lfloor (a + b)/2 \rfloor$ a porovnáme p_c a x. Ak zistíme, že $p_c < x$, dostávame nový vzťah $p_c < x \leq p_b$, v opačnom prípade dostávame $p_a < x \leq p_c$. V oboch prípadoch sme zmenšili interval možností približne na polovicu.

Keďže v našom prípade má cesta N úsekov, vieme ten správny binárnym vyhľadávaním nájsť v čase $O(\log N)$. Zvyšok výpočtov už vieme realizovať v konštantnom čase. Preto vieme na každú otázku odpovedať v čase $O(\log N)$, a celková časová zložitosť nášho vzorového riešenia je $O(N + Q \log N)$.

Listing programu:

```
var N, Q, i, x, y : longint;
    d, v, p : array[0..200047] of longint;
    t : array[0..200047] of double;

function cas(x : longint) : double;
var i, j, k : longint;
begin
   if x=0 then cas:=0 else begin
      i := 0; j := N;
      while (j-i > 1) do begin
         k := (i+j) div 2;
         if p[k]<x then i:=k else j:=k;
      end;
      cas := t[i] + (x-p[i]) / v[i];
   end;
end;

begin
   read(N);
   for i:=0 to N-1 do read(d[i]);
   for i:=0 to N-1 do read(v[i]);
   t[0] := 0;
   for i:=0 to N-1 do t[i+1] := t[i] + (d[i] / v[i]);
   p[0] := 0;
   for i:=0 to N-1 do p[i+1] := p[i] + d[i];
   read(Q);
   while Q>0 do begin
      dec(Q);
      read(x,y);
      writeln((cas(y)-cas(x)):0:10);
   end;
end.
```

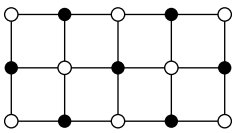
B-I-3 Squaplex

Pre $R = S = 1$ má úloha triviálne riešenie: položíme ceruzku na papier a hneď ju aj zdvihneme.

Ak je práve jedno z R a S rovné 1, úloha zjavne riešenie nemá – akonáhle opustíme začiatočný mrežový bod, použijeme na to jedinú hranu, ktorá z neho vychádza, a teda sa už nemáme ako vrátiť doň späť.

Predpokladajme teda, že $R, S > 1$.

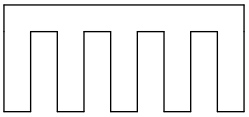
Najprv si ukážeme, že úloha nemôže mať riešenie, ak sú oba čísla R a S nepárne. Uvažujme ofarbenie mrežových bodov dvoma farbami „šachovnicovým“ spôsobom znázorneným na nasledujúcom obrázku:



Každý ťah spĺňajúci podmienky zadania teraz musí striedavo prechádzať čiernymi a bielymi bodmi. Presnejšie, ak si postupne zaznačíme farby navštívených bodov (Č=čierna, B=biela), tie musia tvoriť postupnosť BČBČB…BČB. Táto postupnosť musí začínať aj končiť bielou, keďže začíname aj končíme v ľavom hornom rohu. To ale znamená, že celková dĺžka postupnosti je určite nepárna.

Lenže ak sú oba rozmery šachovnice nepárne, je celkový počet bodov nepárny. Ak navštívime každý z nich práve raz a následne sa vrátime do bodu, kde sme začínali, nutne dostaneme postupnosť párnej dĺžky.

V prípade, že aspoň jedno z čísel R a S je párne, úloha má riešenie. Ťah možno zostrojiť napríklad spôsobom znázorneným na nasledujúcom obrázku:



Je zrejmé, že ak hodnota S je párna, riešenie z predchádzajúceho obrázku funguje pre akúkoľvek výšku, t.j. hodnotu R. V prípade, že S je nepárne a R je párne, stačí predchádzajúci obrázok preklopiť, t. j. vymeniť x-ovú a y-ovú os.

Listing programu:

```
var R, S, i : longint;

procedure chod(kolko : longint; ako : char);
var j : longint;
begin
   for j:=1 to kolko do write(ako);
end;

begin
   readln(R,S);
   if (R=1) and (S=1) then begin writeln; halt; end;
   if (R=1) or (S=1) then begin writeln('Nema riesenie.'); halt; end;
   if (R mod 2=1) and (S mod 2=1) then begin writeln('Nema riesenie.'); halt; end;
   if (S mod 2=0) then begin
      chod(R-1,'D');
      for i:=1 to S-1 do begin
         chod(1,'P');
         if (i mod 2=1) then chod(R-2,'H') else chod(R-2,'D');
      end;
      chod(1,'H');
      chod(S-1,'L');
   end else begin
      chod(S-1,'P');
      for i:=1 to R-1 do begin
         chod(1,'D');
         if (i mod 2=1) then chod(S-2,'L') else chod(S-2,'P');
      end;
      chod(1,'L');
      chod(R-1,'H');
   end;
   writeln;
end.
```

B-I-4 Gaštanová žirafa

Začnime tým, že si skúsime spísať niekoľko jednoduchých pozorovaní:

Najľahšie vieme gaštany priradiť častiam tela žirafy podľa toho, koľko z nich vedie zápaliek. Z hlavy aj z každého kopyta je to práve jedna, z krku sú dve, zo zadku aspoň tri, a z trupu aspoň 4.

Trup od zadku vieme odlíšiť podľa toho, že z neho vedie zápalka do krku.

Ak poznáme trup aj zadok, vieme nájsť hlavu – je to jediný gaštan, ktorý nesusedí ani s jedným z nich.

Tieto pozorovania nám už stačia na návrh efektívnej stratégie:

1. ak ešte nevieme nič

- Vyberieme si niektoré 4 gaštany (napr. tie s číslami 1, 2, 3 a 4).
- Pre každý z nich sa opýtame na všetky ostatné gaštany, aby sme vedeli, s ktorými je spojený a s ktorými nie.
- Keďže v celej žirafe sú len 3 gaštany, z ktorých ide viac ako jedna zápalka, aspoň jeden z našich gaštanov má len jednu zápalku. Jeden taký si vyberme a označme ho a.

- Už poznáme aj gaštan, ktorý je spojený s gaštanom a, označme ho b.
- Opýtame sa na všetky dvojice (b, iný gaštan). Ak zistíme, že z b vedú dve zápalky, je a hlava a b krk. V opačnom prípade je b buď trup alebo zadok.

2. ak vieme, že gaštan b je trup alebo zadok

- Určite existujú aspoň 3 gaštany, s ktorými b nie je spojený. Vyberieme si niektoré 3 takéto gaštany.
- Pre každý z nich zistíme, koľko zápaliek z neho vedie.
- V celej žirafe sú len 3 gaštany, z ktorých ide viac ako jedna zápalka, a jeden z týchto troch je b. Preto aspoň jeden z našich troch gaštanov má len jednu zápalku. Jeden taký si vyberme a označme ho c.
- Gaštan spojený s c označme d.
- Opýtame sa na všetky dvojice (d, iný gaštan). Ak zistíme, že z d vedú dve zápalky, je c hlava a d krk. V opačnom prípade je d buď trup alebo zadok.

3. ak vieme, že gaštany b a d sú trup a zadok

- Jediný gaštan, ktorý nie je spojený ani s trupom, ani so zadkom, je hlava. Keďže už poznáme čísla gaštanov spojených s b aj s d, to jediné, ktoré medzi nimi nie je, je číslo hlavy.

4. ak už vieme, ktorý gaštan je hlava

- Jediný gaštan, ktorý je spojený s hlavou, je krk.
- Jediný ďalší gaštan, ktorý je spojený s krkom, je trup.
- Všetky gaštany, ktoré nie sú spojené s hlavou, krkom ani trupom, sú zadné kopytá.
- Ak ešte nevieme, ktorý gaštan je zadok, nájdeme ho tak, že zoberieme ľubovoľné zadné kopyto a nájdeme gaštan, s ktorým susedí.
- Všetky ostatné gaštany sú predné kopytá.

Ľahko spočítame, že ak sa budeme držať tejto stratégie, určite budeme potrebovať menej ako $10N$ otázok.

SLOVENSKÁ KOMISIA OLYMPIÁDY V INFORMATIKE

DVADSIATY PIATY ROČNÍK OLYMPIÁDY V INFORMATIKE

Vydala IUVENTA s finančnou podporou Ministerstva školstva SR
Náklad: 400 výtlačkov
Zodpovedný redaktor: Michal Forišek
Sadzba programom LaTeX