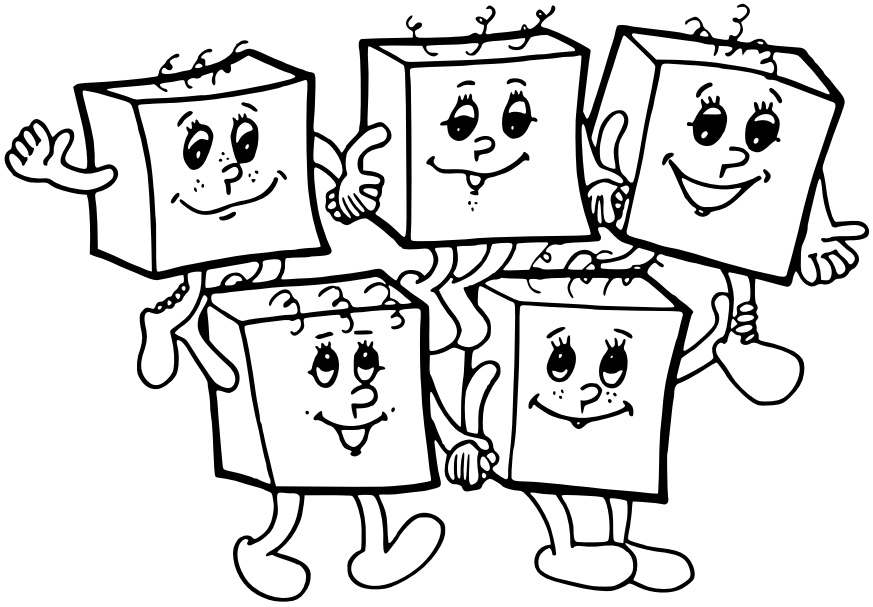
OLYMPIÁDA V INFORMATIKE
NA STREDNÝCH ŠKOLÁCH



dvadsiaty tretí ročník
školský rok 2007/08

riešenia celoštátneho kola
kategória A

1. 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 prvého súťažného dňa

A-III-1 Najväčšia horúčava

Označme si roky zo vstupu $r_1, \ldots, r_n$ a zodpovedajúce teploty $t_1, \ldots, t_n$. Platí $r_1 < \cdots < r_n$.

Začneme tým, že si uvedomíme nasledovnú vec: Ak pre nejaké i a j platí $j < i$ a $t_j \geq t_i$, tak určite rok r_i mohol byť nanajvýš najteplejší od roku r_j, určite nie dlhšie.

Zišlo by sa nám teda pre každý zo zadaných rokov zistiť, ktorý je najbližší skorší rok, v ktorom bolo aspoň tak teplo. Presnejšie, chceme hodnoty $prev_1, \ldots, prev_n$, kde $prev_i$ je najväčšie j, pre ktoré platí $j < i$ a $t_j \geq t_i$ (alebo 0, ak také j neexistuje).

Podobne sa nám budú hodiť hodnoty $next_1, \ldots, next_n$, ktoré hovoria, kde je najbližší neskorší rok, v ktorom bolo aspoň tak teplo.

Tieto hodnoty vieme jednoducho spočítať v čase $\Theta(n^2)$, jednoducho pre každé i v cykle vyskúšame všetky j. Neskôr ukážeme, ako hodnoty $prev_i$ šikovnejšie spočítať v čase $\Theta(n)$.

Majme teraz výrok „v roku q_1 bolo najväčšie teplo od roku q_2“. Ako zistiť jeho pravdivostnú hodnotu?

V prvom rade si zistíme, či sú roky q_1 a q_2 medzi zadanými n rokmi, a ak áno, nájdeme indexy a a b také, že $q_1 = r_a$ a $q_2 = r_b$. Keďže roky na vstupe boli utriedené, vieme toto spraviť v čase $\Theta(\log n)$, napríklad pomocou binárneho vyhľadávania. (V našej implementácii namiesto toho používame `map` z STL.)

Na základe toho, či sa nám podarilo roky q_1 a q_2 nájsť medzi údajmi zo vstupu, vieme pravdivosť daného výroku rozhodnúť nasledovne:

1. **Ani a ani b neexistuje.**
   Výrok je vždy otázny.

2. **Aj a aj b existuje.**
   Ak $prev_a = b$, výrok je pravdivý alebo otázny (podľa toho, či poznáme všetky teploty medzi danými rokmi). Inak je výrok nepravdivý.

3. **a existuje, b nie.**
   Nech $prev_a = x$. Ak $r_x > q_2$, tak je výrok nepravdivý, inak je otázny.

4. **b existuje, a nie.**
   Nech $next_b = x$. Ak $r_x < q_1$, tak je výrok nepravdivý, inak je otázny.

Zdôvodníme teraz jednotlivé prípady.

- V prípade 1 výrok zjavne nie je pravdivý (lebo nepoznáme niektoré teploty). Je však vždy otázny – aby bol pravdivý, stačí doplniť v rokoch q_1 a q_2 tú istú teplotu, vyššiu od všetkých teplôt medzi nimi.

- V prípade 2 vieme teplotu aj v roku q_1, aj v roku q_2. Aby výrok mohol byť pravdivý, musí nutne byť teplota v q_2 aspoň taká ako v q_1, a navyše q_2 musí byť najbližší takýto rok. Inými slovami, musí platiť $prev_a = b$. Ak teda neplatí $prev_a = b$, výrok je nepravdivý.

  Ak platí $prev_a = b$, výrok je buď pravdivý (ak poznáme všetky teploty medzi q_1 a q_2), alebo otázny (inak).

  To, či poznáme všetky teploty v danom úseku, ľahko zistíme v konštantnom čase. Stačí porovnať hodnoty $q_1 - q_2$ a $a - b$. Ak si nie sú rovné (a teda $q_1 - q_2 > a - b$), niektoré roky nám chýbajú.

- Prípad 3 vyzerá nasledovne: Nech $prev_a = x$. Inými slovami, rok r_x je najbližší rok pred q_1, o ktorom vieme, že v ňom bolo teplejšie. Ak $r_x > q_2$, leží tento rok medzi rokmi q_1 a q_2, a teda daný výrok je nutne nepravdivý. Naopak, ak $r_x < q_2$, znamená to, že o žiadnom takom roku medzi q_1 a q_2 nevieme, a ľahko doplníme chýbajúce teploty tak, aby výrok pravdivý bol.

- Prípad 4 funguje analogicky ako prípad 3.

Predpočítanie

Zostáva vysvetliť, ako v lineárnom čase spočítať hodnoty *prev* a *next*. Vysvetlíme si spočítanie hodnôt *prev*, pre *next* to vyzerá analogicky.

Budeme postupne spracúvať hodnoty zo vstupu. V každom okamihu si budeme pamätať množinu tých rokov, ktoré ešte môžu byť „najbližším predchodcom“ nejakého roku.

Kľúčové pozorovanie je nasledovné: Akonáhle spracujeme rok r, v ktorom bola teplota t, môžeme zabudnúť na všetky predchádzajúce roky, ktoré mali teplotu nižšiu ako t.

(Napríklad keby boli na vstupe postupne roky s teplotami: 100, 14, 74, 39, 40, 27 a 12, stačilo by si pamätať roky s teplotami 100, 74, 40, 27 a 12.

Keby teraz prišiel ďalší rok s teplotou 47, mohli by sme po jeho spracovaní zabudnúť na ďalšie tri roky, a pamätať si len roky s teplotami 100, 74 a 47.)

Spracovanie jedného roku r teda bude vyzerať nasledovne: Zabudneme na všetky roky, ktoré majú nižšiu teplotu ako práve spracúvaný. Následne zoberieme posledný rok p s vyššou alebo rovnou teplotou ako r, a zapíšeme si $prev_r = p$. Na záver pridáme rok r medzi pamätané roky.

Keďže roky, ktoré si v ľubovoľnom okamihu pamätáme, sú utriedené podľa teploty, môžeme na implementáciu vyššie popísaného postupu jednoducho použiť zásobník.

Odhad časovej zložitosti potom spravíme nasledovne: Každý rok raz spracujeme a vložíme na zásobník, a každý rok najviac raz zo zásobníka vyhodíme. Preto dokopy spravíme pri predpočítaní $O(N)$ operácií.

Záver

Ukázali sme riešenie, ktoré si v čase $O(N)$ predpočíta hodnoty, ktoré následne používa na to, aby ľubovoľnú otázku zodpovedalo v čase $O(\log N)$. Celková časová zložitosť nášho riešenia je teda $O(N + M \log N)$. Pamäťová zložitosť je $O(N)$.

Iné riešenia

Existuje viacero iných prístupov, ktorými sa dá dosiahnuť rovnaká časová zložitosť ako vo vzorovom riešení. Môžeme si napríklad nad vstupnými dátami postaviť dva intervalové stromy. Pomocou prvého budeme vedieť v ľubovoľnom intervale rokov v čase $O(\log N)$ nájsť maximálnu teplotu, pomocou druhého budeme o ľubovoľnom intervale vedieť povedať, či máme zadané teploty pre všetky roky v ňom. Detaily implementácie maximového intervalového stromu nájdete napríklad vo vzorových riešeniach domáceho kola.

Listing programu:

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

#define MAXN 101000
#define NEKONECNO 1987654321

int N, M;
int R[MAXN], T[MAXN];
map<int,int> kde;
int prev[MAXN], next[MAXN];

int main() {
  // nacitame vstup
  cin >> N;
  for (int i=1; i<=N; i++) cin >> R[i] >> T[i];
  vector< pair<int,int> > napln_kde(N);
  for (int i=1; i<=N; i++) napln_kde[i-1] = make_pair(R[i],i);
  kde.insert( napln_kde.begin(), napln_kde.end() );
  R[0] = -NEKONECNO;
  R[N+1] = NEKONECNO;

  // spocitame polia prev[] a next[]
  stack<int> teplo;
  stack<int> cislo;
  teplo.push(NEKONECNO); cislo.push(0);
  for (int i=1; i<=N; i++) {
    while (teplo.top() < T[i]) { cislo.pop(); teplo.pop(); }
```

```
    prev[i] = cislo.top();
    teplo.push(T[i]); cislo.push(i);
  }
  while (!teplo.empty()) { teplo.pop(); cislo.pop(); }
  teplo.push(NEKONECNO); cislo.push(N+1);
  for (int i=N; i>=1; i--) {
    while (teplo.top() < T[i]) { cislo.pop(); teplo.pop(); }
    next[i] = cislo.top();
    teplo.push(T[i]); cislo.push(i);
  }

  // odpovedame na otazky
  cin >> M;
  while (M--) {
    int q1, q2;
    cin >> q1 >> q2;
    int a = kde[q1], b = kde[q2];
    if (a==0 && b==0) { cout << "otazny" << endl; continue; }
    if (a!=0 && b!=0 && b!=prev[a]) { cout << "nepravdivy" << endl; continue; }
    if (a!=0 && b!=0 && a-b==q1-q2) { cout << "pravdivy" << endl; continue; }
    if (a!=0 && b!=0) { cout << "otazny" << endl; continue; }
    if (a!=0 && R[prev[a]]>q2) { cout << "nepravdivy" << endl; continue; }
    if (a!=0) { cout << "otazny" << endl; continue; }
    if (R[next[b]]<q1) { cout << "nepravdivy" << endl; continue; }
    { cout << "otazny" << endl; continue; }
  }
  return 0;
}
```

A-III-2 Tobogany

V riešení budeme používať grafovú terminológiu, bazény budeme volať vrcholmi a tobogany hranami.

Definujme *výšku* vrcholu nasledovne: Cieľový vrchol (spodný bazén) má výšku 0. Pre každý iný vrchol výšku spočítame ako jedna plus maximum výšok vrcholov, do ktorých sa z neho vieme dostať toboganom.

Množinu vrcholov, ktoré majú rovnakú výšku, budeme volať *vrstva*.

Jazdou z vrcholu v nazveme postupnosť, v ktorej sa striedajú typy bazénov, ktoré stretáme a čísla toboganov, ktoré si vyberáme. Napríklad pre druhý príklad v zadaní jedna možná jazda z vrcholu 2 vyzerá nasledovne: 6,1,4,3,6,3. (Začneme v šesťuholníku, toboganom 1 prejdeme do štvorca, odtiaľ toboganom 3 do šesťuholníka, a odtiaľ toboganom 3 do cieľa.)

Pozorovanie 1. Výška vrcholu zodpovedá počtu toboganov v najdlhšej jazde z neho.

Pozorovanie 2. Ak majú dva vrcholy rovnakú množinu jázd, tak ležia v tej istej vrstve.

Pravdivosť pozorovania 2 vyplýva jednoducho z toho, že množinou jázd je jednoznačne určená výška vrcholu.

Dva vrcholy, ktoré majú rovnakú množinu jázd, budeme volať *ekvivalentné*.

Pozorovanie 3. V optimálnom grafe žiadne dva vrcholy nebudú ekvivalentné.

Pravdivosť tohoto tvrdenia by mala byť intuitívne jasná. Formálne ho môžeme dokázať napríklad nasledovne. Sporom, nech tvrdenie neplatí. Nájdime teda v optimálnom grafe najnižšiu vrstvu, v ktorej sú dva ekvivalentné vrcholy. Potom ale môžeme zostrojiť nový graf, v ktorom jeden z týchto dvoch vrcholov zahodíme, a všetky tobogany, ktoré šli doň, presmerujeme do toho druhého. To je ale spor s tým, že pôvodný graf už bol optimálny.

Pozorovanie 4. Optimálny graf má toľko vrcholov, koľko rôznych množín jázd majú vrcholy pôvodného grafu – čiže koľko rôznych navzájom neekvivalentných vrcholov pôvodný graf obsahuje.

Všimnime si ľubovoľný vrchol v pôvodného grafu. V pôvodnom grafe existuje cesta, ktorou sa doň z horného vrcholu dostaneme. Táto istá cesta musí v optimálnom grafe tiež viesť do nejakého vrcholu v', a keďže má optimálny graf byť nerozlíšiteľný od pôvodného, musí mať v' rovnakú množinu jázd ako v. Preto má optimálny graf aspoň toľko vrcholov, koľko rôznych množín jázd majú vrcholy pôvodného grafu. Opačná nerovnosť vyplýva z pozorovania 3.

Pozorovanie 5. Optimálny graf vieme vyrobiť z pôvodného tak, že postupne oddola nahor pospájame v každej vrstve všetky skupiny navzájom ekvivalentných vrcholov.

Ako vieme efektívne zistiť, či sú nejaké dva vrcholy ekvivalentné? Najjednoduchšia (a najmenej efektívna) metóda je samozrejme vyskúšať všetky možné jazdy z jedného a skontrolovať, že idú spraviť aj z druhého a naopak. Našťastie pre nás, ak budeme postupovať pekne systematicky po vrstvách, vieme si skoro všetku túto prácu ušetriť.

Pozorovanie 6. Nech u a v sú dva vrcholy v tej istej vrstve, a nech už v nižších vrstvách žiadne dva vrcholy nie sú ekvivalentné. Potom u a v sú ekvivalentné vtedy a len vtedy, ak majú rovnaký tvar a pre každé i platí,

že i-tou hranou sa z u dostanem do toho istého vrcholu ako z v.

Pozorovanie 6 nám dáva veľmi efektívny návod, ako zistiť, či je práve spracúvaná dvojica vrcholov ekvivalentná.

Naše riešenie sa bude skladať z dvoch krokov. V prvom prehľadávaním do hĺbky zistíme výšky jednotlivých vrcholov, a roztriedime ich do vrstiev. Následne v druhom kroku budeme vrcholy postupne po vrstvách spracúvať.

Popíšeme si teraz podrobnejšie, ako bude spracovanie jednej vrstvy prebiehať. Najskôr utriedime vrcholy podľa štvoríc (tvar,ciele všetkých troch vychádzajúcich hrán). Takto sa dostanú ekvivalentné vrcholy k sebe. Teraz z každej skupiny ekvivalentných vrcholov necháme len jeden, a presmerujeme do neho hrany, ktoré doteraz viedli do ostatných vrcholov.

Odstránenie vrcholu vieme ľahko spraviť v čase lineárnom od počtu vchádzajúcich hrán. (Potrebujeme na to pamätať si pre každý vrchol zoznam hrán, ktoré doň vchádzajú.) Keďže každý vrchol odstránime najviac raz, vieme dokopy všetko odstraňovanie spraviť v čase $O(N)$.

Ak by sme na triedenie vrcholov použili všeobecný triediaci algoritmus, dostali by sme tak riešenie s časovou zložitosťou $O(N \log N)$.

V našom prípade však vieme vrcholy utriediť v lineárnom čase, pomocou triedenia CountSort. (Tvary majú len 3 rôzne hodnoty. Vrcholov, do ktorých smerujú hrany, je len lineárne veľa, a v čase lineárnom od ich počtu si ich vieme prečíslovať číslami od 1 po nanajvýš ich počet.)

Takto dostávame riešenie s časovou zložitosťou $O(N)$.

Listing programu:

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

#define SIZE(t) ((int)((t).size()))
#define MAXN 100047

int N;
int tvar[MAXN], tobogany[MAXN][3];
int hlbka[MAXN];

int pocitaj_hlbku(int v) {
  int &res = hlbka[v];
  if (res >= 0) return res;
  if (v==N-1) return res=0;
  for (int i=0; i<3; i++) res = max(res,1+pocitaj_hlbku(tobogany[v][i]));
  return res;
}

vector<int> count_sort(vector<int> prvky, vector<int> hodnoty) {
  int najvacsia = *max_element( hodnoty.begin(), hodnoty.end() );

  vector<int> pocty(najvacsia+1), offset(najvacsia+1);
  for (int i=0; i<SIZE(hodnoty); i++) pocty[ hodnoty[i] ]++;
  for (int i=1; i<=najvacsia; i++) offset[i] = offset[i-1] + pocty[i-1];

  vector<int> vystup( SIZE(prvky) );
  for (int i=0; i<SIZE(prvky); i++) vystup[ offset[ hodnoty[i] ]++ ] = prvky[i];
  return vystup;
}

bool rovnake(int x, int y) {
  if (tvar[x] != tvar[y]) return false;
  for (int q=0; q<3; q++) if (tobogany[x][q] != tobogany[y][q]) return false;
  return true;
}

int main() {
  // nacitame vstup
  cin >> N;
  for (int i=0; i<N-1; i++)
    scanf("%d %d %d %d ",&tvar[i],&tobogany[i][0],&tobogany[i][1],&tobogany[i][2]);
  for (int i=0; i<N-1; i++) for (int q=0; q<3; q++) tobogany[i][q]--;

  // spocitame hlbky vrcholov a roztriedime vrcholy do vrstiev
  memset(hlbka,-1,sizeof(hlbka));
  for (int i=0; i<N; i++) hlbka[i] = pocitaj_hlbku(i);
  int H = hlbka[0]+1;
  vector< vector<int> > vrstvy(H);
  for (int i=0; i<N; i++) vrstvy[hlbka[i]].push_back(i);
```

```
  // pre kazdy vrchol si najdeme vsetky, ktore nan ukazuju
  vector< vector<int> > opacne(N);
  for (int i=0; i<N-1; i++) for (int j=0; j<3; j++) opacne[ tobogany[i][j] ].push_back(i);

  vector<int> cislo(N);

  // postupne po vrstvach hladame a odstranujeme ekvivalentne vrcholy
  for (int i=0; i<H; i++) {
    // utriedime vrcholy vo vrstve
    vector<int> hodnoty( SIZE(vrstvy[i]) );
    for (int j=0; j<SIZE(vrstvy[i]); j++) hodnoty[j] = tvar[ vrstvy[i][j] ];
    vrstvy[i] = count_sort( vrstvy[i], hodnoty );
    for (int q=0; q<3; q++) {
      // precislujeme vrcholy
      int last = 0;
      for (int j=0; j<SIZE(vrstvy[i]); j++)
        if (!cislo[ tobogany[ vrstvy[i][j] ][q] ])
          cislo[ tobogany[ vrstvy[i][j] ][q] ] = ++last;
      // vyrobime pomocne pole
      for (int j=0; j<SIZE(vrstvy[i]); j++) hodnoty[j] = cislo[ tobogany[ vrstvy[i][j] ][q] ];
      // utriedime podla neho
      vrstvy[i] = count_sort( vrstvy[i], hodnoty );
      // upraceme po sebe cisla
      for (int j=0; j<SIZE(vrstvy[i]); j++) cislo[ tobogany[ vrstvy[i][j] ][q] ] = 0;
    }
    // najdeme a odstranime duplikaty
    vector<int> ostava(1,vrstvy[i][0]);
    for (int j=1; j<SIZE(vrstvy[i]); j++) {
      if (rovnake(vrstvy[i][j],vrstvy[i][j-1])) {
        int stare = vrstvy[i][j], nove = ostava.back();
        for (int k=0; k<SIZE( opacne[stare] ); k++)
          for (int q=0; q<3; q++)
            if (tobogany[ opacne[stare][k] ][q] == stare)
              tobogany[ opacne[stare][k] ][q] = nove;
      } else ostava.push_back( vrstvy[i][j] );
    }
    vrstvy[i] = ostava;
  }

  int res = 0;
  for (int i=0; i<H; i++) res += SIZE( vrstvy[i] );
  cout << res << endl;

  return 0;
}
```

A-III-3 Prekladacie stroje

Podúloha a)

Všetky stroje, ktoré v tejto úlohe zostrojíme, budú samozrejme fungovať tak, že kopírujú vstupný reťazec na výstup a popri tom kontrolujú, či má požadovaný tvar.

Najjednoduchšie riešenie je stroj s 105 stavmi, ktorý si bude v stave pamätať zvyšok, ktorý dáva doteraz prečítaný reťazec po delení 105.

$$\begin{aligned} A_1 &= (K_1, \Sigma, P_1, 0, F_1) \\ \Sigma &= \{0, \ldots, 9\} \\ K_1 &= \{0, \ldots, 104\} \\ F_1 &= K_1 - \{0\} \\ P_1 &= \{(x, y, y, (10x + y) \bmod 105) \mid \forall x \in K,\ \forall y \in \Sigma\} \end{aligned}$$

Existuje však ešte veľa priestoru na zlepšenie. V prvom rade si môžeme 105 zapísať ako 5×21. Aby sme zistili deliteľnosť 105, stačí zistiť deliteľnosť 21 a deliteľnosť 5. Lenže na kontrolu deliteľnosti 5 si nepotrebujeme priebežne počítať zvyšok. Stačí sa pozrieť na poslednú cifru, či je to 5 alebo 0.

Ukončovacie stavy takéhoto stroja budú teda zodpovedať situácii „doteraz prečítané číslo nie je deliteľné 21, alebo posledná cifra nebola 0 alebo 5“.

Takýto stroj vieme ľahko zostrojiť tak, že bude mať 42 stavov – budeme si v stave pamätať zvyšok po delení 21 a jeden bit hovoriaci, či bola posledná cifra 0 alebo 5.

Riešenie vieme ešte zlepšiť, ak si uvedomíme, že posledná cifra nás zaujíma len ak doteraz prečítané číslo je deliteľné 21. Takto upravený stroj už má len 22 stavov:

$$\begin{aligned} A_2 &= (K_2, \Sigma, P_2, 0_{zle}, F_2) \\ \Sigma &= \{0, \ldots, 9\} \\ K_2 &= \{0_{zle}, 0_{dobre}, 1, \ldots, 20\} \\ F_2 &= K_2 - \{0_{zle}\} \\ P_2 &= \{(x, y, y, z(x,y) \mid \forall x \in \{1, \ldots, 20\},\ \forall y \in \Sigma\} \end{aligned}$$

Nový stav $z(x, y)$ je jednoznačne určený pôvodným stavom x a prečítanou číslicou y nasledovne: Nech x' je číselná hodnota x. (Teda pre $x \in \{0_{zle}, 0_{dobre}\}$ je $x' = 0$, inak $x' = x$.) Hodnota x' bol zvyšok, ktorý dávala doteraz prečítaná časť čísla po delení 21. Nový zvyšok po delení 21 je $z' = (10x' + y) \bmod 21$. Ak $z' \neq 0$, je nový stav $z(x, y) = z'$. V opačnom prípade nový stav závisí od y. Ak $y \in \{0, 5\}$, tak $z(x, y) = 0_{zle}$, inak $z(x, y) = 0_{dobre}$.

Aj toto riešenie však ešte môžeme výrazne zlepšiť.

Doteraz boli všetky prekladacie stroje, ktoré sme zostrojili, *deterministické*. (T. j. na každom vstupe bol preklad jediný, jednoznačne určený.)

Môžeme však ešte využiť to, že prekladací stroj môže mať na jednom slove prekladov viac, a výstupná množina je tvorená výstupmi všetkých platných prekladov.

Namiesto toho, aby sme na každom slove mali jeden výpočet „over, že nie je deliteľné 21“, budeme mať výpočty dva: „over, že nie je deliteľné 7“ a „over, že nie je deliteľné 3“.

Uvedieme konštrukciu prekladacieho stroja, ktorý si vystačí s 12 stavmi: začiatočný stav Z, sedem stavov na kontrolu deliteľnosti siedmimi $s_0, \ldots, s_6$, tri stavy na kontrolu deliteľnosti troma t_0, t_1, t_2 a ukončovací stav U, do ktorého sa budeme vedieť dostať len vtedy, ak prekladané slovo nie je deliteľné 105.

$$\begin{aligned} A_3 &= (K_3, \Sigma, P_3, Z, F_3) \\ \Sigma &= \{0, \ldots, 9\} \\ K_3 &= \{Z, U, s_0, \ldots, s_6, t_0, t_1, t_2\} \\ F_3 &= \{U\} \\ P_3 &= \{(Z, \varepsilon, \varepsilon, s_0), (Z, \varepsilon, \varepsilon, t_0)\} \\ &\cup \{(s_x, y, y, s_{(10x+y) \bmod 7}) \mid \forall x \in \{0, \ldots, 6\},\ \forall y \in \Sigma\} \\ &\cup \{(t_x, y, y, t_{(10x+y) \bmod 3}) \mid \forall x \in \{0, \ldots, 2\},\ \forall y \in \Sigma\} \\ &\cup \{(s_x, \varepsilon, \varepsilon, U) \mid \forall x \in \{1, 2, 3, 4, 5, 6\}\} \\ &\cup \{(t_x, \varepsilon, \varepsilon, U) \mid \forall x \in \{1, 2\}\} \\ &\cup \{(s_x, y, y, U) \mid \forall x \in \{0, \ldots, 6\},\ \forall y \in \{1, 2, 3, 4, 6, 7, 8, 9\}\} \\ &\cup \{(t_x, y, y, U) \mid \forall x \in \{0, \ldots, 2\},\ \forall y \in \{1, 2, 3, 4, 6, 7, 8, 9\}\} \end{aligned}$$

Do ukončovacieho stavu U môže takýto stroj prejsť len v nasledujúcich prípadoch:

- Ak sa rozhodol overovať, že číslo nie je deliteľné 7, a doteraz prečítané číslo deliteľné 7 nebolo.
- Ak sa rozhodol overovať, že číslo nie je deliteľné 3, a doteraz prečítané číslo deliteľné 3 nebolo.
- Ak sa rozhodol, že práve číta poslednú cifru, a tá nie je deliteľná 5.

Zjavne pre každé číslo, ktoré nie je deliteľné 105, vieme nájsť nejaký platný preklad. A naopak, ak máme platný preklad, musel do U prísť až po prečítaní celého vstupu, a teda vstup musel byť jedného z vyššie popísaných tvarov.

Na záver riešenie podľa Samuela Hapáka: Na ušetrenie ešte jedného stavu môžeme použiť to, že pomocou špeciálneho znaku \$ vieme zistiť, kedy nám skončil vstup.

Oproti predchádzajúcej konštrukcii zrušíme ukončovací stav U a namiesto neho prehlásime za ukončovacie stavy s_x a t_y pre všetky $x, y \neq 0$. Takto už máme zabezpečené, že vypíšeme čísla, ktoré nie sú deliteľné 3 a 7. Kvôli číslam nedeliteľným 5 pridáme pravidlá tvaru „ak si v nejakom stave t_x a čítaš reťazec y\$ (kde y je cifra iná od 0 a 5), vypíš y a prejdi do niektorého ukončovacieho stavu.

Dostávame teda prekladací stroj, ktorý rieši zadanú úlohu a má len 11 stavov.

Pomocné tvrdenie

Ku každému prekladaciemu stroju A existuje prekladací stroj B, ktorý vyrába presne rovnaké preklady, ale v každom kroku prekladu vypíše najviac jedno písmeno.

Dôkaz tohto tvrdenia je triviálny. Budeme B postupne vyrábať z A, pričom len upravíme prekladové pravidlá, pri ktorých A zapisoval viac ako jedno písmeno.

Nech napríklad A obsahuje pravidlo (q, def, aba, p) ("ak si v stave q, a neprečítaná časť vstupu začína na def, môžeš prečítať def, zapísať aba a zmeniť stav na p„).

Toto pravidlo je zlé, lebo ak by sme ho v niektorom kroku použili, zapíšeme až tri písmená. Nahradíme ho teda v B tromi pravidlami, ktoré tieto tri písmená zapíšu postupne.

Presnejšie, do B pridáme dva nové stavy (nazvime ich tu trebárs ♡ a ♠), a namiesto pôvodného pravidla z A budeme v B mať pravidlá: $(q, def, a, \heartsuit)$, $(\heartsuit, \varepsilon, b, \spadesuit)$, $(\spadesuit, \varepsilon, a, p)$.

Ľahko nahliadneme, že akonáhle pri preklade v B použijeme prvé z trojice pravidiel, musíme v nasledujúcich dvoch krokoch použiť zvyšné dve pravidlá.

Uvedeným spôsobom upravíme postupne všetky pravidlá z A. Dostaneme tým B, ktorý má síce viac stavov ako A, ale vyrába presne tie isté preklady, a navyše v každom kroku prekladu vypíše najviac jedno písmeno. A to je presne to, čo sme chceli.

Podúloha b)

Tento prekladací stroj neexistuje. Intuitívne môžeme povedať, že je to preto, že máme len konečne veľa stavov, a teda si nevieme pomocou stavu pamätať, koľko a sme už zapísali.

Dokážeme to sporom, a pre jednoduchší dôkaz použijeme pomocné tvrdenie, ktoré sme si práve dokázali.

Predpokladajme teda, že hľadaný prekladací stroj existuje. Potom podľa pomocného tvrdenia existuje prekladací stroj, ktorý robí to isté, a navyše v každom kroku prekladu vypíše najviac jedno písmeno.

Zoberme si jeden takýto prekladací stroj A. Označme jeho počet stavov n.

Všimnime si teraz reťazce ε, ab, $aabb$, $\ldots$, až $\underbrace{a\ldots a}_{n}\underbrace{b\ldots b}_{n}$.

Toto je $n+1$ rôznych reťazcov, ktoré náš prekladací stroj musí vypísať na výstup. Ku každému z nich musí existovať v A aspoň jeden platný preklad. Vyberme si teda ku každému reťazcu jeden platný preklad.

Teraz využijeme, že náš stroj vypisuje písmená na výstup po jednom. Všimnime si v každom z našich $n+1$ prekladov okamih, kedy sme práve vypísali na výstup posledné a. Vypíšme si, v akom stave sa práve náš stroj nachádzal.

Takto dostaneme zoznam, v ktorom bude $n+1$ stavov. Náš stroj A však má len n rôznych stavov, a teda je v našom zozname nejaký stav aspoň dvakrát.

Čo toto znamená?

Znamená to, že existujú dva rôzne platné preklady, ktoré vyzerajú nasledovne:

Prvý: Začneme v začiatočnom stave. Vypíšeme x písmen a. V tomto okamihu sme v nejakom stave p. Vypíšeme x písmen b, a tým sa dostaneme do ukončovacieho stavu.

Druhý: Začneme v začiatočnom stave. Vypíšeme y písmen a (kde $y \neq x$). V tomto okamihu sme v tom istom stave p ako prvý preklad. Vypíšeme y písmen b, a tým sa dostaneme do ukončovacieho stavu.

Aby sme dostali spor, stačí si už len uvedomiť, že môžeme oba preklady "rozstrihnúť„ a následne "zlepiť„ prvú polovicu prvého a druhú polovicu druhého prekladu.

Inými slovami, aj toto musí byť platný preklad:

Tretí: Začneme v začiatočnom stave. Vypíšeme x písmen a. V tomto okamihu sme v stave p. Vypíšeme y písmen b, a tým sa dostaneme do ukončovacieho stavu.

Vo výslednej množine $A(X)$ sa teda musí nachádzať aj slovo $\underbrace{a \ldots a}_{x}\underbrace{b \ldots b}_{y}$. Takéto slovo tam ale byť nesmie, a to je práve hľadaný spor.

SLOVENSKÁ KOMISIA OLYMPIÁDY V INFORMATIKE
DVADSIATY TRETÍ ROČNÍK OLYMPIÁDY V INFORMATIKE

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