Korešpondenčný seminár z programovania
XX. ročník, 2002/03
Katedra vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporuje MICROSTEP-MIS spol. s r.o.

Vzorové riešenia 2. kola letnej časti

Milí riešitelia!
My KSPáci sme už starí a leniví, navyše sme rozlezeni po celom Slovensku (a okolí) a prázdninujeme. Preto to tak dlho trvalo, kým sa tieto vzorové riešenia dostali až do vašich rúk. Čo dodať na záver jubilejného 20. ročníka? S najlepšími sa stretneme na jesennom sústredení na Dobrej Vode, s najstaršími na matfyze a s ostatnými pri riešení ďalšieho ročníka KSP.

Kto nepracuje, nech sa aspoň naje.

KSPáci

1. O doplnení postupnosti

opravoval Goober
(max. 15 bodov)

Vaše riešenia ma neveľmi potešili, lebo aj keď väčšina bola založená na správnej myšlienke[1], väčšinou ste v jej implementácii spravili jednu či dve malé, no závažné chyby.

Celkove sa vaše riešenia dajú rozdeliť do štyroch kategórií – úplne zlé (ohodnotené nanajvýš **3 bodmi**), kubické (**7 bodov**), kvadratické – zväčša založené na Newtonovej interpolácii[2] (**11 bodov**) a lineárne – využívajúce zväčša Lagrangeovu interpoláciu **15 bodov**. Ako už býva zvykom, pár bodíkov mohlo ísť dole za nedostatočný popis, chýbajúce odhady zložitosti, prehnané pamäťové nároky, či chýbajúci dôkaz správnosti. Elegantné riešenia naopak mohli získať bodík naviac. Okrem toho sa vo viacerých kvadratických riešeniach vyskytovala podobná chyba – zrejme kvôli optimalizácii ste pri počítaní rozdielov zastavili akonáhle ste narazili na jednu nulu (alebo niečo podobné). Pritom v popise programu ste takéto „skoré zastavenie“ ani slovkom nespomenuli. Normálne by takéto riešenie bolo ohodnotené ako zlé, no keďže som mal veľmi dobrú náladu, rozhodol som sa za takúto chybu stŕhať len 2 body.

A ako sa to dalo robiť lineárne? Najprv si dohodneme terminológiu: $P(x)$ bude polynóm *stupňa* $n \geq 1$ práve vtedy, ak x^n je najvyššia mocnina x vyskytujúca sa v $P(x)$. V prípade, že $P(x)$ neobsahuje žiadnu kladnú mocninu x, definujeme jeho stupeň ako 0, okrem prípadu, keď $P(x)$ je nulový polynóm (t.j. $P(x) = 0$), kedy bude jeho stupeň $-\infty$. Okrem toho, budeme hovoriť, že $P(x)$ je *rádu* n, ak je jeho stupeň nanajvýš n.

Keď už máme tieto pojmy objasnené, dokážeme si niekoľko užitočných tvrdení:

Lema 1: Nenulový polynóm stupňa k nemá viac ako k rôznych koreňov.

Dôkaz indukciou: Pre $k = 0$ tvrdenie očividne platí – keďže $P(x)$ má stupeň 0, je konštantný a nenulový, a preto nemá ani jeden koreň.

Inak predpokladajme, že existuje polynóm stupňa k, ktorý má aspoň $k + 1$ rôznych koreňov (označme ich $x_1 < x_2 < \ldots < x_k < z$). Nech A je koeficient pri najvyššej mocnine x v polynóme $P(x)$ (teda má tvar $P(x) = Ax^k + S(x)$, kde $S(x)$ je rádu $k - 1$) a nech $Q(x) = (x - x_1)(x - x_2)\ldots(x - x_k)$. Všimnime si teraz polynóm $R(x) = P(x) - AQ(x)$. Zjavne, čísla $x_1, x_2, \ldots x_k$ sú koreňmi $R(x)$, zatiaľčo číslo z nie je (lebo $P(z) = 0$ a $Q(z) > 0$). Okrem toho, $R(x)$ je rádu $k - 1$, lebo člen Ax^k z $P(x)$ sa odčíta s takým istým členom z

[1] a zvyšné boli zlé iba kvôli nepochopeniu zadania...

[2] i keď ste tak svoj postup nenazývali :-)

$AQ(x)$. Teda $R(x)$ je nenulový polynóm rádu $k-1$ a vzťahuje sa naň indukčný predpoklad, ktorý hovorí, že nemá viac ako $k-1$ koreňov. To je však v spore s tým, že $x_1, x_2, \ldots, x_k$ sú jeho korene.

Lema 2: Nech $f(x)$, $g(x)$ sú polynómy rádu $k \geq 0$ a platí $f(x) = g(x)$ pre $k+1$ rôznych hodnôt x (označme ich $x_0, x_1, \ldots, x_k$). Potom $f(x) = g(x)$ (pre všetky x).

Dôkaz: Všimnime si polynóm $h(x) := f(x) - g(x)$. Ak je nulový, niet čo riešiť. Inak je nenulový a má stupeň nanajvýš k. Potom podľa lemy 1 má nanajvýš k rôznych koreňov. Na druhej strane, $h(x_i) = f(x_i) - g(x_i) = 0$ pre $0 \leq i \leq k$, a teda sme našli $k+1$ rôznych koreňov. Spor.

Vráťme sa teraz k zadaniu úlohy. To hovorí, že máme akýchsi n hodnôt a hľadáme polynóm, ktorý tieto hodnoty nadobúda v určitých určitých, pevne zvolených, bodoch. Zakrátko ukážeme, že vždy vieme nájsť polynóm $P(x)$ rádu $(n-1)$ s touto vlastnosťou. Zadanie však ďalej požaduje, aby stupeň tohoto polynómu bol čo najnižší. No a tu nám pomôže lema 2 – hovorí totiž, že pre daných n hodnôt neexistujú dva rôzne polynóm rádu $(n-1)$. Každý iný kandidujúci polynóm má teda stupeň väčší ako n, a teda $P(x)$ je ten polynóm, ktorý hľadáme.

Zostáva teda ukázať, ako ho nájdeme. Na to nám poslúži vzorček, pomenovaný po pánovi Lagrange-ovi – *Lagrangeov interpolačný polynóm.* Ten hovorí, ako nájsť polynóm rádu[3] $(n-1)$, ktorý nadobúda v bodoch x_i hodnoty y_i, kde $1 \leq i \leq n$ a vyzerá nasledovne:

$$P(x) = \sum_{i=1}^{n} y_i b_i(x), \qquad \text{kde } b_i(x) = \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

Najprv si všimneme peknú vlastnosť $b_i(x)$ – sú to všetko polynómy stupňa $(n-1)$ a okrem toho pre $1 \leq i, j \leq n$ platí, že $b_i(x_j) = 1$ ak $i = j$, a $b_i(x_j) = 0$ inak. Stadiaľ je zrejmé, že $P(x_i) = y_i$ (okrem jedného budú všetky členy v sume nulové).

V našom prípade je x_i pevne určené ako $x_i := i$ a navyše nás ani nezaujíma to, ako vyzerá polynóm $P(x)$ samotný, skôr nás zaujíma jeho hodnota pre $x = (n+1)$. Z uvedeného vzorčeka dostávame:

$$P(n+1) = \sum_{i=1}^{n} y_i c_i \quad \text{a} \quad c_i := b_i(n+1) = \prod_{j=1, j\neq i}^{n} \frac{n+1-j}{i-j}$$

Malými úpravami môžeme c_i zjednodušiť na tvar

$$c_i = \frac{n!}{n+1-i} \prod_{j=1, j\neq i}^{n} \frac{1}{i-j} = (-1)^{n-i} \frac{n!}{(i-1)!(n+1-i)!} = (-1)^{n-i} \binom{n}{i-1}$$

Už sme skoro na konci, stačí už len prísť na rýchly spôsob výpočtu c_i. Na to nám poslúži známy vzťah pre kombinačné čísla: $\binom{n}{i} = \binom{n}{i-1} \frac{n+1-i}{i}$. Potom $c_{i+1} = c_i(-1)\frac{n+1-i}{i} = c_i \frac{i-1-n}{i} = c_i - c_i \frac{n+1}{i}$. No a na začiatku to treba celé „naštartovať“ pomocou $c_1 = (-1)^{n-1}$.

Samotný algoritmus bude teda veľmi jednoduchý – popísanú sumu budeme počítať člen po člene s využitím vzorčeka na výpočet c_{i+1} z c_i, vždy si prečítame jedno číslo, prenásobíme ho príslušným c_i a pričítame k doterajšiemu výsledku.

Časová zložitosť bude teda $O(n)$ a pamäťová $O(1)$.

Poznámka: Všimnite si, že sme našli asymptoticky *najlepší* algoritmus na riešenie tohto problému – časovú ani pamäťovú náročnosť nemožno asymptoticky zlepšiť, lebo musíme prečítať všetky čísla zo vstupu a potrebujeme mať pamäť aspoň na výsledok.

[3] Pozor! Nemusí byť stupňa $(n-1)$!

A predsa je to trošku ináč. V našom odhade pamäťovej (a aj časovej) zložitosti sme boli máličko benevolentní. Zamyslite sa nad tým, ako môže výsledok narásť napriek tomu, že vstupné čísla boli malé[4] a koľko pamäte naň treba.

Listing programu:

```
#include <stdio.h>
int i,n,c,a,s=0;

int main() {
  scanf("%d",&n);
  c=(n&1)?1:-1;
  for(i=1;i<=n;i++) {
    scanf("%d",&a);
    s+=c*a;
    c-=(c*(n+1))/i;
  }
  printf("Dalsie cislo: %d\n",s);
  return 0;
}
```

2. Opäť o metre

opravoval Braňo
(max. 15 bodov)

Skoro všetky vaše riešenia boli založené na Dijkstrovom algoritme pre hľadanie najkratšej cesty medzi dvoma vrcholmi. Výnimkou nebude ani vzorák. Tých, čo Dijkstrov algoritmus nepoznajú, nasleduje jeho stručný opis. Záujemcov o podrobnejší popis spolu s dôkazom správnosti odkazujem na vzorák 3. príkladu 2. kola zimnej časti KSP.

Dijkstrov algoritmus: Máme daný graf s nezápornými hranami a začiatočný vrchol. Výstupom algoritmu budú najmenšie vzdialenosti zo začiatočného vrcholu do všetkých ostatných vrcholov a cesta, akou sa do nich dá oprimálne dostať.

Pre každý vrchol si budeme udržiavať jeho doteraz nejlepšiu vzdialenosť od začiatočného vrchola do všetkých ostatných. Niektoré z týchto vzdialeností sú trvalé, iné sa ešte možno môžu zmenšiť (označme si ich dočasné). Na začiatku sú všetky vzdialenosti dočasné s hodnotou nekonečno, iba začiatočný vrchol má vzdialenosť 0.

Algoritmus vždy vyberie s pomedzi dočasných vrcholov ten, ktorý má najmenšiu vzdialenosť. Tento vrchol prehlási za trvalý a preverí všetkých jeho susedov, či sa nedá zlepšiť ich vzdialenosť tým, že do nich pôjdeme z tohto vrcholu.

Pre každý vrchol si budeme navyše pamätať, z kade sa do neho vchádza na optimálnej ceste zo začiatočného vrcholu. Táto informácia sa aktualizuje vždy, keď skrátime dočasnú vzdialenosť cez niektorý vrchol. Najkratšiu cestu z ľubovolného vrcholu zrekonštruujeme tak, že začneme v žiadanom vrchole a pokračujeme vždy vrcholom, z ktorého sa do neho vchádza na optimálnej ceste. Takto nájdeme optimálnu cestu odzadu. Keď ju chceme odpredu, ľahko si ju otočíme. Iná možnosť je nedať hľadať najkratšiu cestu z vrcholu 1 do N, ale z vrcholu N do 1 – ak je graf symetrický.

Dijkstrov algoritmus stačilo zmodifikovať tak, že si o vrcholoch neudržiaval iba ich doteraz (resp. trvalo) najmenšie vzdialenosti od počiatočného vrcholu, ale aj najmenší počet prestupov spolu so spoločnosťami, ktorými sa na tento počet prestupov dá v danom vrhole skončiť.

Vzorové riešenie: Je založené na inej myšlienke. Zmodifikujme graf na vstupe nasledovne:

- každý vrchol rozdvojíme na vrchol pre spoločnost KSP a vrchol pre spoločnosť DPMB t.j. vrchol k na vrcholy $2k$, $2k+1$

[4] Hint: skúste postupnosti tvaru 1,-1,1,-1,1,...

- každá hrana spoločnosti KSP z vrcholu k do l s cenou c sa zmení na hranu z vrcholu $2k$ do $2l$ s cenou $(c, 0)$
- každá hrana spoločnosti DPMB z vrcholu k do l s cenou c sa zmení na hranu z vrcholu $2k + 1$ do $2l + 1$ s cenou $(c, 0)$
- jednotlivé časti každého rozdvojeného vrchola $2k$, $2k + 1$ spojíme obojsmernou hranou s cenou $(0, 1)$. Výnimku tvoria dvojice 2, 3 a $2N$, $2N + 1$ (t.j. pôvodne vrcholy 1 a N). Tie spojíme hranou s cenou $(0, 0)$

Cena hrany v novom grafe nie je normálne číslo, ale vektor dvoch čísel. Aby Dijkstrov algoritmus vedel pracovať s našimi vektormi, musíme si na nich zadefinovať sčítanie a porovnanie. Sčitovať budeme po zložkách t.j. $(a, b) + (c, d) = (a + c, b + d)$. Pri porovnávaní sa rozhodujeme najprv podľa prvej zložky a keď sa prvé zložky rovnajú, tak podľa druhej zložky.

Keď na takto pozmenenom grafe pustíme Dijkstrov algoritmus, tak ten bude hľadať najlacnejšiu cestu, pričom ako hlavné kritérium volí dĺžku cesty. Keď je rovnako dlhých ciest viac, vyberá tie, ktoré majú menší počet prestupov.

To je ale predsa presne to, čo po nás chcelo zadanie. Akurát potrebujeme z najlacnejšejšej cesty v novom grafe zostrojiť cestu v starom. Povyhadzujeme z cesty hrany medzi rozdelenými vrcholmi a prečísluje vrcholy späť.

Zložitosť: Keď sme mali na vstupe graf s N vrcholmi a M hranami, tak po úprave bude mať graf $2N$ vrcholov a $M + N$ hrán. Časovú ani pamäťovú zložitosť nám to nepokazí, lebo $O((2N)^2) = O(4N^2) = O(N^2)$.

Bodovanie: Riešenia v čase $O(N^2)$ dostali 15 bodov, backtracky dostali 10 bodov, niečo medzi tým dostalo niečo medzi tým.

Listing programu:

```
program Opat_o_metre;
const MAX_N = 1000;
      INF   = 10000;
type TVec2 = record c, p : integer; end;  {cena, prestupy}

var A : array [1..2*MAX_N+2, 1..2*MAX_N+2] of TVec2;  {matica susednosti}
    N,M : integer;
    naj_cesta : array [1..2*MAX_N+2] of integer; {najkratsia cesta z vrchola 2 do 2*N}
    naj_cena : TVec2;         {cena najlacnejsej cesty}

procedure citaj;
var i,j,k,l,c : integer;
begin
  readln(N,M);

  for i:=1 to 2*N+2 do    {prazdny graf}
    for j:=1 to 2*N+2 do begin
      A[i][j].c:=INF;    A[i][j].p:=INF;
    end;

  for i:=1 to N do begin  {obojsmerne hrany medzi castami zastavky}
    A[2*i,2*i+1].c:=0;   A[2*i,2*i+1].p:=1;
    A[2*i+1,2*i].c:=0;   A[2*i+1,2*i].p:=1;
  end;
  A[2,3].p:=0;           A[3,2].p:=0;
  A[2*N,2*N+1].p:=0;     A[2*N+1,2*N].p:=0;

  for i:=1 to M do begin
    readln(k,l,c);
    A[2*k,2*l].c:=c;     A[2*k,2*l].p:=0;     {hrana KSP}
```

```
    A[2*l+1,2*k+1].c:=c; A[2*l+1,2*k+1].p:=0; {hrana DPMB}
  end;
end;

function tvec2_plus( const a, b : TVec2 ) : TVec2;       {a+b}
var x:TVec2;
begin
  x.c:= a.c+b.c;
  x.p:= a.p+b.p;
  tvec2_plus:=x;
end;

function tvec2_je_mensie( const a, b : TVec2 ) :boolean; {je a mensie ako b?}
begin
  tvec2_je_mensie:= (a.c<b.c) or ( (a.c=b.c) and (a.p<b.p ));
end;


procedure dijkstra;       {najde najkratsiu cestu z vrchola 2*N do 2}
var ceny : array [ 0..2*MAX_N+2 ] of TVec2;    {cena vrchola}
    def  : array [ 0..2*MAX_N+2 ] of boolean;  {je vrchol definitivny?}
    spat : array [ 0..2*MAX_N+2 ] of integer;  {z kade sme isli do daneho vrchola}
    i, naj : integer;
begin
  for i:=0 to 2*N+2 do begin
    ceny[i].c:=INF; ceny[i].p:=INF; def[i]:=false;{nevieme sa dostat do ziadneho vrchola}
  end;
  ceny[2*N].c:=0; ceny[2*N].p:=0;             {zaciatocny vrchol mame zadarmo}

  while (true) do begin
    naj:=0;
    for i:=2 to 2*N+2 do                      {najdeme najblizsi nedefinitivny vrchol}
      if (not def[i]) and tvec2_je_mensie(ceny[i], ceny[naj]) then
        naj:=i;
    def[naj]:=true;

    if (naj=0) then begin writeln('Cesta neexistuje.'); halt(1); end;
    if (naj=2) then break;                    {sme na konci cesty}

    for i:=2 to 2*N+2 do                      {skracujeme cestu cez vsetky hrany z vrcholu naj}
      if tvec2_je_mensie( tvec2_plus( ceny[naj], A[naj,i] ), ceny[i] ) then begin
        ceny[i]:=tvec2_plus( ceny[naj], A[naj,i] );
        spat[i]:=naj;                         {do vrchola i sme dosli z naj}
      end;
  end;

  i:=1; naj_cesta[i]:=2;                      {hladame cestu spat}
  while (naj_cesta[i]<>2*N) do begin          {hladali sme odzadu, netreba otacat}
    inc(i);
    naj_cesta[i]:= spat[ naj_cesta[i-1] ];
  end;
  naj_cena:=ceny[2];
end;

procedure vypis;
var i,x:integer;
begin
  write('Najkratsia cesta: ');
  i:=1; x:=N+4;
  repeat
```

```
    if (naj_cesta[i] div 2 <> x) then begin
      x:=naj_cesta[i] div 2;
      write(x);
      if (x<>N) then write(' --> ');
    end;
    inc(i);
  until x=N;
  writeln;
  writeln('Vzdialenost: ', naj_cena.c, '   Prestupov:', naj_cena.p);
end;

begin
  citaj;
  if (N<1) then exit;
  dijkstra;
  vypis;
end.
```

opravoval MišoF.
(max. 15 bodov)

3. O vyberaní mýta

Na úvod pár slov pre tých, ktorým slová „teória grafov“ príliš nehovoria. V našom chápaní **graf** je niekoľko bodov (ktoré budeme volať **vrcholy**), niektoré dvojice bodov sú pospájané čiarami (ktoré budeme volať **hrany**). Alebo ešte raz a formálnejšie, (neorientovaný) graf je dvojica $G = (V, E)$, kde V je množina vrcholov a $E \subseteq \{\{x, y\} \mid x, y \in V\}$ je množina neusporiadaných dvojíc vrcholov, t.j. hrán. Práve takýto graf máme v našej úlohe na vstupe – križovatky sú vrcholy, cesty medzi nimi sú hrany.

Hovoríme, že graf je **súvislý**, ak sa po hranách dá prejsť z ľubovoľného vrcholu do ľubovoľného iného. Každý graf vieme jednoznačne rozdeliť na niekoľko súvislých častí, medzi ktorými nevedú hrany. Tieto časti budeme volať **komponenty súvislosti**.

Zadanie úlohy nedáva príliš zmysel, ak by pôvodný graf nebol súvislý. Na druhej strane, v zadaní to explicitne uvedené nebolo. Ak ste si túto skutočnosť uvedomili, ocenil som ju jedným bodom. V ďalšom texte riešenia budeme predpokladať, že graf na vstupe (označme ho G) je súvislý.

Nech k je nejaká križovatka, na ktorej sa oplatí postaviť mýtnicu, nech a, b sú nejaké dve križovatky také, že každá cesta medzi nimi prechádza cez k. To ale znamená, že keby sme z G odstránili vrchol k, z a do b sa nebude dať prejsť, inými slovami dostaneme nesúvislý graf. Na druhej strane, nech v je ľubovoľný vrchol G, po ktorého odstránení dostaneme nesúvislý graf. Zoberme vrcholy a, b v rôznych komponentoch výsledného grafu. Potom ale všetky cesty z a do b viedli cez v, a teda sa v ňom oplatí postaviť mýtnicu.

Našou úlohou je teda nájsť všetky vrcholy, ktorých odstránenie poruší súvislosť grafu G. Takéto vrcholy voláme **artikulácie**.

Jednoduché, ale pomalšie riešenie.

Ako zistiť, či je graf súvislý? Existuje viacero algoritmov, sú známe pod spoločným názvom **ofarbovanie vrcholov** alebo tiež **prehľadávanie**. Základná myšlienka: začneme v nejakom vrchole grafu a postupne systematicky ofarbujeme všetky vrcholy, kam sa vieme dostať. Keď už nevieme nič ofarbiť, skončili sme. Teraz sa už len stačí pozrieť, či sú ofarbené všetky vrcholy. Samozrejme, treba vrcholy ofarbovať tak, aby sme určite žiaden nevynechali. Teraz si vysvetlíme jeden vhodný postup, nazývaný prehľadávanie do hĺbky.

Prehľadávanie do hĺbky je podobné postupu, akým človek skúma neznáme mesto. Začneme tým, že sa postavíme do nejakého vrcholu a ofarbíme ho. Odteraz budeme farbiť vrcholy aj hrany grafu, kade chodíme. Ak z vrcholu, kde sme, vedie ešte nepoužitá hrana, vyberieme sa ňou ďalej. Ak sme prišli do ešte nenavštíveného vrcholu, ofarbíme ho a rekurzívne zavoláme prehľadávanie z neho. (Teda opäť sa snažíme nájsť nepoužitú hranu, atď.)

Ak sme prišli do skôr navštíveného vrcholu (t.j. ofarbeného), okamžite sa po hrane, ktorou sme prišli, vrátime späť. No a posledný prípad je keď sme vo vrchole, z ktorého vedú samé ofarbené hrany. V takomto prípade sa len vrátime tou hranou, ktorou sme do neho prvýkrát prišli. Keď sa takto chceme vrátiť z vrcholu, kde sme začínali, prehľadávanie končí.

Zjavne takto prejdeme práve dvakrát (tam a späť) po každej z hrán, ku ktorým sa vieme dostať a navštívime všetky vrcholy, ku ktorým sa vieme dostať zo začiatočného vrcholu. Algoritmus je teda korektný a jeho časová zložitosť je $O(M + N)$. Dá sa ľahko rekurzívne implementovať, viď program na konci riešenia.

Najjednoduchším riešením pôvodnej úlohy by teda bolo postupne vyskúšať každý vrchol odstrániť a pozrieť sa, či je výsledný graf ešte stále súvislý. Takéto riešenie by malo časovú zložitosť $O(N(M + N))$ – pre každý vrchol potrebujeme spustiť jedno prehľadávanie.

Dá sa spraviť drobný trik, aby sme si ušetrili robotu s postupným odstraňovaním vrcholov z grafu. Nech v je vrchol, o ktorom chceme zistiť, či je artikulácia. Jednoducho z neho spustíme prehľadávanie pôvodného grafu a prestaneme v okamihu, keď sa doň prvýkrát vrátime. Potom v je artikulácia práve vtedy, ak ešte nemáme ofarbený celý graf. (Prečo?)

Pár slov o prehľadávaní do hĺbky.

Lepšie riešenie bude modifikáciou algoritmu prehľadávania do hĺbky. Predtým, než vysvetlíme samotné riešenie, potrebujeme si ukázať niekoľko vlastností prehľadávania do hĺbky. Spustime ho (začínajúc v ľubovoľnom vrchole) na našom súvislom grafe zo vstupu. Všimnime si tie hrany grafu, ktorými sme počas prehľadávania prišli do dovtedy nenavštíveného vrcholu. Týchto hrán je zjavne $N-1$ (jedna pre každý vrchol okrem toho, v ktorom sme začínali). Graf nimi tvorený je strom, lebo je súvislý a neobsahuje kružnice. Tento strom budeme volať DFS strom (DFS = depth-first search = prehľadávanie do hĺbky). Vrchol, z ktorého sme začínali prehľadávať, budeme volať koreň. Z každého iného vrcholu x vedie po stromových hranách (hranách DFS stromu) do koreňa práve jedna cesta. Vrcholy na tejto ceste budeme volať predkami vrcholu x, vrchol x budeme volať ich potomkom. Špeciálne každý vrchol je sám sebe aj predkom, aj potomkom. Všetci potomkovia vrcholu x a stromové hrany medzi nimi tvoria podstrom s koreňom x.

Ostatné hrany teoreticky môžu byť dvoch typov. Ak hrana spája vrchol s nejakým jeho predkom alebo potomkom, budeme ju volať spätná, ostatné hrany budeme volať priečne. (Nech uv je hrana, ktorá nie je stromová. Všimnime si podstromy s koreňmi u, v. Sú dve možnosti – ak je jeden z nich podgrafom druhého, hrana uv je spätná, inak musia tieto podstromy byť disjunktné (prečo?) a hrana uv je priečna.)

V DFS strome však žiadne priečne hrany nemôžu byť. Prečo? Sporom. Nech uv je priečna hrana. Bez ujmy na všeobecnosti nech sme počas prehľadávania do u prišli skôr. Všimnime si teraz okamih, keď sa počas prehľadávania ideme vrátiť späť z u. Aby uv bola priečna hrana, nesmeli sme doteraz v navštíviť (ináč by v bol potomok u a hrana uv by bola spätná). Ale v je sused u, preto by sme sa z u ešte nemali vracať späť ale mali by sme sa vybrať do v, spor.

3 cm

Takže hrany grafu môžeme rozdeliť na stromové a spätné. Každá spätná hrana uv leží na kružnici (tvorenej hranou uv a cestou z u do v po DFS strome).

Rýchlejšie, ale zložitejšie riešenie.

Teraz ukážeme algoritmus, ktorý bude bežať v optimálnom čase $O(M + N)$.

Predstavme si, že sme už spustili na G prehľadávanie do hĺbky a máme zostrojený DFS strom. Teraz nás zaujíma, či sa po odobratí konkrétneho vrcholu v graf G rozpadne alebo nie.

Ošetrime najskôr špeciálne koreň v DFS stromu, teda vrchol, odkiaľ sme začali prehľadávať. Podobne ako v pomalšom riešení platí, že je to artikulácia, ak sme sa doň vrátili pred skončením prehľadávania. To totiž znamená, že ku dovtedy neofarbeným vrcholom sa

nevieme z už ofarbených vrcholov dostať inak ako cez v. Teda stačí pozrieť, či má v v DFS strome viac ako jedného syna.

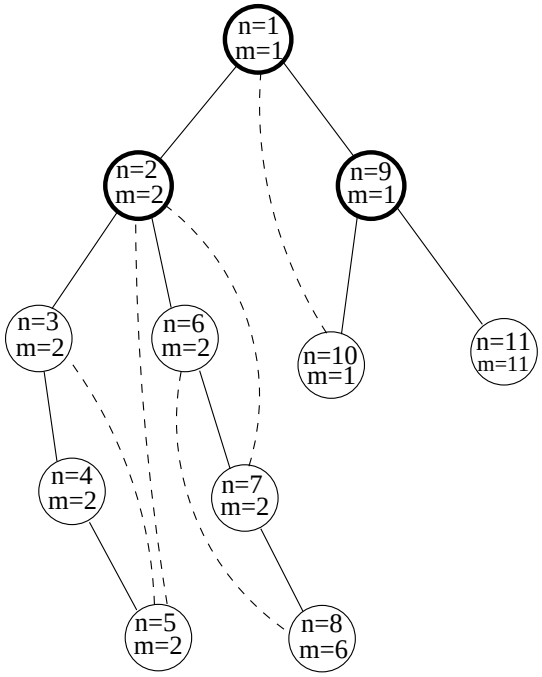
Nech teraz v je ľubovoľný vrchol rôzny od koreňa. Ako sa môže rozpadnúť G po odstránení v? Zjavne jediné, čo sa môže stať, je že od „hlavnej“ časti (tej obsahujúcej koreň) odpadne niekoľko podstromov visiacich pod v.

Nech teda u je ľubovoľný syn v. Ako spoznať, že podstrom s koreňom u po odstránení v upadne? Upadne práve vtedy, ak sa z podstromu s koreňom u nevieme dostať von inak ako cez v.

Ak takáto cesta von existuje, určite existuje aj cesta, ktorá použije práve jednu spätnú hranu, a to ako poslednú. Dôkaz: Zoberme ľubovoľnú cestu von a skončime ju v okamihu, keď sa dostaneme z podstromu s koreňom u. Posledná použitá hrana je nutne spätná. Jej počiatočný vrchol leží v podstrome s koreňom u, a teda sa doň vieme dostať z u po stromových hranách.

Keby sme pre každý vrchol vedeli, či z jeho podstromu existuje cesta von, máme vyhraté. Totiž konkrétny vrchol v (rôzny od koreňa) je zjavne artikulácia práve vtedy, ak z podstromu *aspoň jedného* syna v nevedie cesta von.

Počas prehľadávania číslujme vrcholy v poradí, v akom do nich prichádzame. Číslo vrcholu x budeme značiť $n(x)$. Zjavne všetky vrcholy v podstrome s koreňom v majú číslo $\geq n(v)$. Na druhej strane všetci predkovia v (okrem v) majú číslo menšie ako $n(v)$. Pre každý vrchol x budeme chcieť spočítať $m(x)$ – najmenšie číslo vrcholu, do ktorého sa z neho vieme dostať vyššie popísanou cestou (takou, čo ide najskôr niekoľkými stromovými hranami „dodola“ a potom jednou spätnou „dohora“).



0 cm

Tým už máme výsledný algoritmus takmer hotový, zostáva si to už len celé zhrnúť. Prehľadávame náš graf do hĺbky a zároveň si pre každý vrchol x počítame dve čísla: $n(x)$ (koľký objavený vrchol to je) a $m(x) = \min\{n(y) \mid$ do y vedie z x cesta vyššie uvedeného tvaru$\}$. Ako počítať hodnotu $n(x)$ je zjavné. Hodnota $m(x)$ je minimum z $n(x)$, zo všetkých hodnôt $m(x_i)$ pre synov vrcholu x a zo všetkých hodnôt $n(y_i)$ vrcholov, do ktorých vedie z x spätná hrana. Hodnotu $m(x)$ teda vieme spočítať v okamihu, keď sa počas prehľadávania vraciame z vrcholu x. V tomto okamihu vieme aj rozhodnúť o x, či je to artikulácia – stačí porovnať hodnotu $n(x)$ s hodnotami $m(x_i)$ pre jeho synov.

Pri každom kroku robíme oproti klasickému prehľadávaniu do hĺbky len konštantne veľa operácií navyše, preto zjavne časová zložitosť ostane nezmenená – $O(M + N)$.

Listing programu:

```
program Artikulacie;

var G : array[1..100,1..100] of integer;   { graf }
    deg,num,up : array[1..100] of integer; { stupne vrcholov a obe cisla pre ne }
    visited : array[1..100] of boolean;    { bol som uz v tomto vrchole? }
    N,M,C : integer;            { pocet vrcholov, hran, navstivenych vrcholov }

procedure Load;
var i,x,y : integer;
begin
   read(N,M); fillchar(deg,sizeof(deg),0);
   for i:=1 to M do begin
      read(x,y);
      inc(deg[x]); G[x][deg[x]]:=y;
      inc(deg[y]); G[y][deg[y]]:=x;
   end;
end;

procedure DFS(v,parent : integer);
var i,deti : integer;
    je_artik : boolean;
begin
   visited[v]:=true; deti:=0; je_artik:=false;
   num[v]:=C; up[v]:=C; inc(C); { nastavime obe cisla vo vrchole }
   for i:=1 to deg[v] do if not visited[G[v][i]] then begin
      inc(deti); if (deti>1) and (v=parent) then je_artik:=true; { koren }
      DFS(G[v][i],v);
      if up[G[v][i]]<up[v] then up[v]:=up[G[v][i]];
      if up[G[v][i]]>=num[v] then je_artik:=true;
   end else begin { spaetna hrana }
      if G[v][i]<>parent then
        if num[G[v][i]]<up[v] then up[v]:=num[G[v][i]];
   end;
   if (je_artik) then writeln(v);
end;

begin
   Load;
   fillchar(visited,sizeof(visited),0); C:=1;
   DFS(1,1);
end.
```

4. Ospalý čarodej

opravoval Paľo
(max. 15 bodov)

V tomto príklade veľká časť z vás prišla na vzorové riešenie alebo jemu podobné, čím ste ma milo prekvapili. Za riešenia v čase $O(N)$ sa dalo získať 15 bodov, za riešenia v čase $O(NM)$ sa dalo získať 10 bodov. Body ste väčšinou stratili za zhoršenú pamäťovú zložitosť alebo odhad zložitosti.

Celý text, ktorý povedal počas spánku čarodejník označme jednoducho text. Predpokladajme, že dĺžka kúzla je M a dĺžka textu je N. Jednoduché ale pomalé riešenie spočíva v tom, že vyskúšame všetky možné pozície kúzla v texte. Tých pozícii je zjavne $N - M + 1$ a pre každú pozíciu musíme v najhoršom prípade porovnať všetky písmená v kúzle, ktorých

je M, teda výsledná časová zložitosť je $O(NM - M^2)$, čo pre relatívne malé M je $O(NM)$. Toto sa dá ale ešte zlepšiť. Existuje viacero lineárnych algoritmov, ktorých časová zložitosť je $O(N + M)$. My si z nich vyberieme Knuth-Morris-Prattov (KMP) algoritmus:

Všimnime si podrobnejšie horeuvedený algoritmus. Pri skúmaní, či sa na i-tom mieste nachádza kúzlo, sa môže stať, že prvých j písmeniek v kúzle sa zhoduje s textom, ale $j + 1$ písmenko sa už nezhoduje. Teraz by sme sa vrátili a skúšali, či sa kúzlo nenachádza na $i + 1$ písmenku. Predstavme si však, že kúzlo je napríklad „baaaa“ a zistíme, že prvých j písmenok sa zhoduje s textom a $j + 1$-vé už nie. V tomto špeciálnom prípade sa teraz nemusíme vracať až na $i + 1$ pozíciu, pretože žiadne z predchádzajúcich $j - 1$ písmen nemôže byť začiatočné písmeno kúzla, a teda môžme ďalej pokračovať až na $i + j$-tom písmenku textu. Toto ale nemôžeme urobiť pre iné kúzla. KMP algoritmus je vlastne zovšeobecnením tejto myšlienky pre ľubovoľné kúzlo.

Nemôžeme ale vždy preskočiť hneď všetkých j písmeniek. Napríklad ak by sme hľadali v texte „bababaabbb“ kúzlo „babaabbb“, tak zistíme na piatom mieste, že sa písmenká nezhodujú, ale musíme znovu začať hľadať kúzlo od tretieho miesta, aby sme ho našli. Teda stačí nám pre každú pozíciu j v kúzle vedieť miesto `next[j]`, kde musíme pokračovať hľadanie, ak sa na nej nachádza prvá nezhoda písmeniek kúzla a textu. Toto ale závisí len od kúzla, ktoré hľadáme a možeme si to predpočítať ešte pred samotným hľadaním kúzla v texte. Všimnime si, že teraz už nepotrebujeme znovu porovnávať písmenká na menších pozíciach v kúzle ako `next[j]`, lebo tie písmenká sa už určite s textom zhodujú.

V príklade vyššie: Prečítali sme prvé 4 písmená textu, zhodujú sa s prvými štyrmi písmenami kúzla. Piate písmeno textu a kúzla sú rôzne. Ale z informácie, že sme mali prečítané 4 písmená kúzla (bez toho, aby sme sa na text museli opäť pozerať) vieme, že predchádzajúce dve písmená boli ba, preto môžeme teraz mať prečítané prvé 3 písmená kúzla.

Ak sa nezhoduje j-te písmenko kúzla s textom a prvých $j - 1$ písmeniek sa zhoduje, tak si v poli `next[j]` zapamätáme, od ktorého písmenka v kúzle musíme znovu začať porovnávať kúzlo s textom. Hľadajme teda `next[j]`. Zoberme si kópiu prvých $j - 1$ písmeniek a položme ich pod kúzlo tak, že prvé písmenko kópie leží pod druhým písmenkom v kúzle. Posúvajme tú kópiu doprava pokiaľ sa už žiadne písmenká neprekrývajú alebo pokiaľ prekrývajúce sa písmenká sú navzájom rovnaké. Prekrývajúce sa písmenka určujú pozíciu, kde musíme znovu začať hľadať kúzlo ak sa písmenka nezhodujú na j-tom mieste – `next[j]` vypočítame ako počet prekrývajúcich sa písmenok plus jedna, pretože ich počet určuje počet zhodujúcich sa písmenok textu a kúzla ak začneme na tejto pozícii a nám stačí začať porovnávať písmenká textu a kúzla až za nimi.

Presnejšie: pre $j > 1$ zoberme reťazec R tvorený prvými $j - 1$ písmenami kúzla. Potom `next[j]` je rovné najväčšiemu takému $k < j$, že prefix R dĺžky $k - 1$ je aj sufixom R (t.j. prvých $k - 1$ písmen R je rovnakých ako posledných $k - 1$).

Takže samotný algoritmus hľadania kúzla v texte bude vyzerať nasledovne: Porovnávajme i-te písmenko textu s j-tym písmenkom kúzla. Ak sa zhodujú, tak aj i aj j zvýšime o jedna a teda sa posunieme na ďalšie písmenko v texte aj v kúzle. Ak sa nezhodujú, tak ďalšia možná pozícia v texte, od ktorej možeme začneme hľadať kúzlo je `i-next[j]+1`, ale prvých `next[j]-1` písmeniek v kúzle sa zhoduje s prvými `next[j]-1` písmenkami v texte, a teda i nemusíme zmeniť a stačí nám zmeniť j na `next[j]`. A aká je časová zložitosť? Pri každom porovnaní písmeniek zvýšime buď naraz i aj j, alebo znížime j. Teda tých porovnaní, kde zvýšime j je N a tých porovnaní, kde ho znížime môže byť znovu len N, pretože j nie je nikdy záporné. Teda dokopy máme iba $2N$ porovnaní písmeniek. Takže časová zložitosť tejto časti programu je $O(N)$. Všimnime si, že pre ľubovoľné kúzlo a text časová zložitosť tejto časti nezávisí vôbec od dĺžky kúzla.

Teraz nám už stačí vypočítať iba pole `next`. To môžeme urobiť nasledovnou fintou: `next[1]` môžeme dať definitoricky 0, čo nám vyhovuje pri implementácii. Všimnime si, že `next[2]` je vždy 1. Hľadajme vyššie uvedeným algoritmom kúzlo v sebe samom, ale začnime

hľadať až od druhého písmenka. Pozrime sa lepšie na moment po inkrementovaní i aj j, teda po posunutí o jedno písmenko v texte doprava. V tomto okamihu sa prvých $j-1$ písmeniek kúzla zhoduje s písmenkami na pozíciach $i-j-1, i-j, \ldots, i-1$, čo je posledných $j-1$ písmeniek z prvých $i-1$ písmeniek kúzla. A toto je zjavne najväčšie j s touto vlastnosťou. Takže môžeme do `next[i]` priradiť j. Tento algoritmus by mohol zhavarovať, ak by sme v ňom potrebovali vedieť `next[j]` skôr než ho vypočítame. Toto sa ale nemôže stať, lebo to by znamenalo, že $i-1$ písmeniek sa zhoduje s $j-1$ písmenkami, teda by sa i rovnalo j, teda by sme museli začať hľadať kúzlo od prvého písmenka, čo je v spore s tým, že hľadáme od druhého písmenka. Takže táto časť má zložitosť $O(M)$, lebo robíme presne to isté ako v normálnom vyhľadávaní a navyše vypočítame pole `next`, ktorého veľkosť je M. Takže výsledná časová zložitosť je $O(N+M)$.

Všimnime si, že si nemusíme pamätať celý text vyrieknutý čarodejníkom, ale stačí si pamätať iba posledné písmenko, lebo sa v ňom nikdy nemusíme vracať. Takže pamäťová zložitosť algoritmu je $O(M)$.

Ako ale nájsť počet kúziel v texte? Kúzlo nájdeme na nejakej pozícii v texte ak $j = M+1$. Keďže sa kúzla môžu prekrývať, my by sme potrebovali vedieť `next[M+1]`, čo ale môžeme vypočítať tým istým spôsobom ako ostatné prvky poľa `next`. Keď nájdeme výskyt kúzla, len si zvýšime počítadlo a nikým nerušení pokračujeme ďalej.

Listing programu:

```
const
  MAXM=200;

var
  zak: string; {zaklinadlo}
    m: integer;{dlzka zaklinadla}
    next: array[1..MAXM] of integer;
    i,j: integer;
    c: char;
    poc: integer; {pocet vyskytov zaklinadla}
begin
    {nacitanie zaklinadla}
    readln(zak);
    m := length(zak);

    next[1] := 0;
    next[2] := 1;
    j := 1;
    for i := 2 to m do begin
        while (j <> 0) and (zak[i]<>zak[j]) do begin
          j := next[j];
        end;
        inc(j);
        next[i+1] := j;
    end;

    j := 1;
    while not seekeoln(input) do begin
        read(c);
        while (j <> 0) and (c <> zak[j]) do begin
            j := next[j];
        end;
        inc(j);
        if (j > m) then begin
            inc(poc);
            j := next[j];
```

```
        end;
    end;

    writeln(poc);
end.
```

5. O celulárnych automatoch IV

opravoval tradične Martin
(max. 15 bodov)

Hoci tento príklad poslalo veľmi málo riešiteľov, takmer všetci prišli na správne riešenie. Horšie to už bolo s implementáciou a napísaním poriadného popisu. Len zopár z vás poriadne a výstižne vysvetlilo, ako jeho automat funguje a (čo je naozaj smutné) nikto sa ani len nepokúsil dokázať, že ním navrhnutý automat robí naozaj to čo má.

Za správne riešenie ste mohli získať maximálne **15 bodov**, pričom za odfláknutý, alebo dokonca chýbajúci popis, ste mohli získať prémiu až **-4 body**. Ďalej ste mohli stratiť zopár bodíkov za rôzne chyby v implementácii automatu, alebo za lenivosť čo i len sa pokúsiť o dôkaz správnosti riešenia.

V minulom kole sme vyriešili úlohu, ako nájsť prostredný, respektíve prostredné dva automaty v rade automatov. Teraz nám riešenie tohto príkladu príde vhod. Predstavme si, že máme vedľa seba niekoľko automatov a chceme, aby v niektorej sekunde všetky naraz prešli do stavu S, pričom žiaden z nich by nebol v tom stave skôr. Pomocou riešenia z minulého kola prevedieme prostredné automaty do stavu ¢, čím dostaneme dva rovnako dlhé úseky automatov oddelelé jedným alebo dvoma automatmi v stave ¢. Tieto dva úseky budeme ďalej rekurzívne deliť na menšie a menšie, až kým nedostaneme úseky s len jedným alebo dvoma automatmi. Počas tohto delenia budeme stav ¢[5] považovať za stav totožný so stavom \$. Všimnite si, že počas celého delenia, ak je nejaký automat v stave ¢, tak potom aspoň jeden z jeho susedov nie je ani v stave ¢ ani v stave \$. Dôležité je si tiež uvedomiť že všetky delené úseky sú stále rovnako dlhé a preto sa všetky rozdelia za rovnako dlhý čas na nové tiež rovnako dlhé úseky. Po skončení delenia budeme mať úseky jedného alebo dvoch automatov oddelené jedným alebo dvoma automatmi v stave ¢. Ak sa teraz opäť pokúsime rozdeliť tieto úseky na polovicu, aj zvyšné automaty prejdu do stavu ¢. Až nakoniec bude každý automat v stave ¢ a teda aj obaja susedia každého automatu budú v stave ¢ alebo \$. Potom už len všetky tieto automaty v stave ¢ prevedieme do vytúženého stavu S. Hoci to na prvý pohľad nieje až také zrejmé, časová zložitosť tohto riešenia je lineárna.

Zapíšme to teraz formálne: Automaty budú nadobúdať stavy z množiny
$K = \{0, 1, A_0, A_1, A_2, L, R, 1', A_0', A_1', A_2', L', R', P, ¢\}$ a ich prechodová funkcia bude:

Nájdenie stredu úseku automatov (riešenie úlohy z minulého kola):

$$
\begin{array}{llcl@{\qquad}llcl}
(1) & \xi 1 \xi & \to & P & (8) & \gamma R 0 & \to & 0 \\
 & & & & (9) & R 0 \alpha & \to & R \\
(2) & \xi 10 & \to & A_1 & (10) & \gamma R \xi & \to & L \\
(3) & 10\alpha & \to & R & (11) & \delta L \alpha & \to & 0 \\
 & & & & (12) & \gamma 0 L & \to & L \\
(4) & \alpha A_0 \beta & \to & A_1 & & & & \\
(5) & \alpha A_1 \beta & \to & A_2 & (13) & \alpha A_2 L & \to & P \\
(6) & \alpha A_2 \beta & \to & 0 & (14) & A_2 L \alpha & \to & P \\
(7) & A_2 0 \beta & \to & A_0 & (15) & \alpha A_0 L & \to & P
\end{array}
\qquad
\begin{array}{lcl}
\xi & \in & \{\$, ¢\} \\
\alpha & \in & \{0, \$, ¢\} \\
\beta & \in & \{0, R\} \\
\gamma & \in & \{0, A_1\} \\
\delta & \in & \{0, A_0\}
\end{array}
$$

[5] Takzvaný falošný \$.

Nájdenie stredu úseku automatov, pričom automatom signalizujúcim začiatok výpočtu je pravý krajný automat:

$$\left.\begin{array}{llcl|llcl}
(1') & \xi 1'\xi & \to & P & (8') & 0R'\gamma & \to & 0 \\
 & & & & (9') & \alpha 0R' & \to & R' \\
(2') & 01'\xi & \to & A'_1 & (10') & \xi R'\gamma & \to & L' \\
(3') & \alpha 01' & \to & R' & (11') & \alpha L'\delta & \to & 0 \\
 & & & & (12') & L'0\gamma & \to & L' \\
(4') & \beta A'_0\alpha & \to & A'_1 & & & & \\
(5') & \beta A'_1\alpha & \to & A'_2 & (13') & L'A'_2\alpha & \to & P \\
(6') & \beta A'_2\alpha & \to & 0 & (14') & \alpha L'A'_2 & \to & P \\
(7') & \beta 0A'_2 & \to & A'_0 & (15') & L'A'_0\alpha & \to & P
\end{array}\right\} \begin{array}{lcl} \xi & \in & \{\$, \text{¢}\} \\ \alpha & \in & \{0, \$, \text{¢}\} \\ \beta & \in & \{0, R'\} \\ \gamma & \in & \{0, A'_1\} \\ \delta & \in & \{0, A'_0\} \end{array}$$

Dodefinovanie prechodovej funkcie pre prípady, keď automat je v stave 0 a nemení svoj stav:

$$(16, 16') \quad \zeta 0\eta \quad \to \quad 0; \quad \begin{array}{lcl} \zeta & \in & \{0, A_0, A_1, L, A'_0, A'_1, A'_2, R', \$, \text{¢}\}, \\ \eta & \in & \{0, A_0, A_1, A_2, R, A'_0, A'_1, L', \$, \text{¢}\} \end{array}$$

Samotné riešenie dnešnej úlohy:

$$\left.\begin{array}{llcl|llcl}
(i) & \psi P\psi & \to & \text{¢} & (iv) & \psi\text{¢}\phi & \to & \text{¢} \\
(ii) & P0\psi & \to & 1 & (v) & \phi\text{¢}\psi & \to & \text{¢} \\
(iii) & \psi 0P & \to & 1' & (vi) & \xi\text{¢}\xi & \to & S
\end{array}\right\} \begin{array}{lcl} \xi & \in & \{\$, \text{¢}\} \\ \psi & \in & \{0, P, \$, \text{¢}\} \\ \phi & \in & \{0, P, \$\} \end{array}$$

Na ozrejmenie činnosti automatu si pozrite tento príklad výpočtu pre dvanásť automatov:

$0 : \$1000000000000\$$	$11 : \$000A_2 0000000R\$$	$22 : \$0R'0A'_0 0\text{¢}\text{¢}0A_0 0R0\$$
$1 : \$A_1 R0000000000\$$	$12 : \$0000A_0 000000L\$$	$23 : \$R'00A'_1 0\text{¢}\text{¢}0A_1 00R\$$
$2 : \$A_2 0R000000000\$$	$13 : \$0000A_1 00000L0\$$	$24 : \$L'00A'_2 0\text{¢}\text{¢}0A_2 00L\$$
$3 : \$0A_0 0R00000000\$$	$14 : \$0000A_2 0000L00\$$	$25 : \$0L'A'_0 00\text{¢}\text{¢}00A_0 L0\$$
$4 : \$0A_1 00R0000000\$$	$15 : \$00000A_0 00L000\$$	$26 : \$00P00\text{¢}\text{¢}00P00\$$
$5 : \$0A_2 000R000000\$$	$16 : \$00000A_1 0L0000\$$	$27 : \$01'\text{¢}10\text{¢}\text{¢}01'\text{¢}10\$$
$6 : \$00A_0 000R00000\$$	$17 : \$00000A_2 L00000\$$	$28 : \$R'A'_1\text{¢}A_1 R\text{¢}\text{¢}R'A'_1\text{¢}A_1 R\$$
$7 : \$00A_1 0000R0000\$$	$18 : \$00000PP00000\$$	$29 : \$L'A'_2\text{¢}A_2 L\text{¢}\text{¢}L'A'_2\text{¢}A_2 L\$$
$8 : \$00A_2 00000R000\$$	$19 : \$00001'\text{¢}\text{¢}10000\$$	$30 : \$PP\text{¢}PP\text{¢}\text{¢}PP\text{¢}PP\$$
$9 : \$000A_0 00000R00\$$	$10 : \$000R'A'_1\text{¢}\text{¢}A_1 R000\$$	$31 : \$\text{¢}\text{¢}\text{¢}\text{¢}\text{¢}\text{¢}\text{¢}\text{¢}\text{¢}\text{¢}\text{¢}\$$
$10 : \$000A_1 000000R0\$$	$21 : \$00R'0A'_2\text{¢}\text{¢}A_2 0R00\$$	$32 : \$SSSSSSSSSSSS\$$

Dôkaz. Najprv definujme niekoľko užitočných označení. Ak q je stav a k je celé nezáporné číslo, zápisom q^k budeme označovať k za sebou idúcich automatov v stave q. Automaty budeme číslovaž číslami od 1 po poradí začínajúc ľavým krajným a končiac pravým krajným automatom. Konfiguráciou úseku automatov začinajúceho v i-tom automate o dĺžke k automatov v t-tej sekunde ($t \geq 0$) je postupnosť stavov týchto automatov a zapisujeme ju: $t :_i^k q_i q_{i+1} \dots q_{i+k-1}$, kde q_j ($i \leq j \leq i + k - 1$) je stav j-tého automatu v t-tej sekunde. Ak j je 0 alebo je väčšie ako počet automatov, potom $q_j = \$$.

Matematickou indukciou od dĺžky k ($k \geq 3$) úseku automatov ukážeme, že ak $t :_i^k \xi_1 10^{k-3}\xi_2$ alebo $t :_i^k \xi_1 0^{k-3}1'\xi_2$, pričom $\xi_1, \xi_2 \in \{\$, \text{¢}\}$, potom existuje $0 < f(k) \leq 3k$ take, že $t + f(k) :_i^k \xi_1 \text{¢}^{k-2}\xi_2$, a každý automat z tohto úseku v čase od t-tej do $(t + f(k) - 1)$-tej sekundy, ktorý bude v stave \$ alebo ¢, bude mať aspoň jedného suseda, ktorý nebude ani v stave \$ ani ¢.

- Ak $k = 3$. Predpokladajme, že $t :_i^3 \xi_1 1\xi_2$ a $\xi_1, \xi_2 \in \{\$, \text{¢}\}$, potom v $(t + 1)$-vej sekunde podľa (1), (iv) a (v) bude $t + 1 :_i^3 \xi_1 P\xi_2$. V $(t + 2)$-hej sekunde podľa (i), (iv) a (v)

bude $t+2 :_i^3 \xi_1 ¢ \xi_2$, preto $f(k) = f(3) = 2 < 3 \cdot 3 = 3k$. Ak $t :_i^3 \xi_1 1' \xi_2$, postupujeme podobne.

- Ak $k = 4$. Opäť predpokladajme, že $t :_i^4 \xi_1 10 \xi_2$ a $\xi_1, \xi_2 \in \{\$, ¢\}$, potom podľa (2), (3), (iv) a (v) je $t+1 :_i^3 \xi_1 A_1 R \xi_2$, podľa (5), (10), (iv) a (v) je $t+2 :_i^3 \xi_1 A_2 L \xi_2$, podľa (13), (14), (iv) a (v) je $t+3 :_i^3 \xi_1 PP \xi_2$ a podľa (i), (iv) a (v) je $t+4 :_i^3 \xi_1 ¢¢ \xi_2$, preto $f(k) = f(4) = 4 < 3 \cdot 4 = 3k$. Ak $t :_i^4 \xi_1 01' \xi_2$, postupujeme podobne.
- Ak $k \geq 5$ a je párne, potom existuje také číslo l ($l \geq 3$), pre ktoré $k = 2l$. Predpokladajme, že dokazované tvrdenie platí pre všetky $3 \leq k' < k$ a že $t :_i^{2l} \xi_1 10^{2l-3} \xi_2$. Platnosť tvrdenia dokážeme aj pre k. Z dôkazu z minulého kola vyplýva, že podľa (1), (2), ... (16) je $t+3l-3 :_i^{2l} \xi_1 0^{l-2} PP 0^{l-2} \xi_2$, pričom žiaden automat počas tejto časti výpočtu ani raz nenadobudol stav ¢. Podľa (16), (i), (ii), (iii), (iv) a (v) plati $t+3l-2 :_i^{2l} \xi_1 0^{l-3} 1' ¢ 10^{l-3} \xi_2$, preto plati aj $t+3l-2 :_i^l \xi_1 0^{l-3} 1' ¢$ a $t+3l-2 :_{i+l}^l ¢ 10^{l-3} \xi_2$. Z indukčného predpokladu potom vyplýva, že $t+3l-2+f(l) :_i^l \xi_1 ¢^{l-2} ¢$ a $t+3l-2+f(l) :_{i+l}^l ¢ ¢^{l-2} \xi_2$, pričom $f(l) \leq 3l$ a od $(t+3l-2)$-hej do $(t+3l-2+f(l)-1)$-vej sekundy každy z automatov v stave \$ alebo ¢ mal aspoň jedného suseda, ktorý nebol ani v stave \$ ani v stave ¢. Preto platí $t+3l-2+f(l) :_i^{2l} \xi_1 ¢^{l-2} ¢ ¢ ¢^{l-2} \xi_2 \equiv t+f(2l) :_i^{2l} \xi_1 ¢^{2l-2} \xi_2$ a od t-tej do $(t+f(2l)-1)$-vej sekundy každy z automatov v stave \$ alebo ¢ mal aspoň jedného suseda, ktorý nebol ani v stave \$ ani v stave ¢. Keďže platí aj nerovnosť $f(k) = f(2l) = 3l-2+f(l) \leq 3l-2+3l < 6l = 3k$, tak sme tvrdenie dokázali aj pre k. Ak $t :_i^{2l} \xi_1 0^{2l-3} 1' \xi_2$, namiesto tvrdenia dokázaného v minulom kole použijeme obodobné tvrdenie o pravidlách $(1')$, $(2')$, až $(16')$.
- Ak $k \geq 5$ a je nepárne, postupujeme velmi podobne. Existuje také číslo l ($l \geq 3$), pre ktoré $k = 2l-1$. Predpokladajme platnosť tvrdenia pre všetky $3 \leq k' < k$ a tiež predpokladajme, že $t :_i^{2l-1} \xi_1 10^{2l-4} \xi_2$. Platnosť tvrdenia dokážeme aj pre k. Z dôkazu z minulého kola vyplýva, že podľa (1), (2), ... (16) je $t+3l-5 :_i^{2l-1} \xi_1 0^{l-2} P 0^{l-2} \xi_2$, pričom žiaden automat počas tejto časti výpočtu ani raz nenadobudol stav ¢. Podľa (16), (i), (ii), (iii), (iv) a (v) plati $t+3l-4 :_i^{2l-1} \xi_1 0^{l-3} 1' ¢ 10^{l-3} \xi_2$, preto plati aj $t+3l-4 :_i^l \xi_1 0^{l-3} 1' ¢$ a $t+3l-2 :_{i+l-1}^l ¢ 10^{l-3} \xi_2$. Z indukčného predpokladu potom vyplýva, že $t+3l-4+f(l) :_i^l \xi_1 ¢^{l-2} ¢$ a $t+3l-4+f(l) :_{i+l-1}^l ¢ ¢^{l-2} \xi_2$, pričom $f(l) \leq 3l$ a od $(t+3l-4)$-tej do $(t+3l-4+f(l)-1)$-vej sekundy každy z automatov v stave \$ alebo ¢ mal aspoň jedného suseda, ktorý nebol ani v stave \$ ani v stave ¢. Preto platí $t+3l-4+f(l) :_i^{2l-1} \xi_1 ¢^{l-2} ¢ ¢^{l-2} \xi_2 \equiv t+f(2l-1) :_i^{2l-1} \xi_1 ¢^{2l-3} \xi_2$ a od t-tej do $(t+f(2l-1)-1)$-vej sekundy každý z automatov v stave \$ alebo ¢ mal aspoň jedného suseda, ktorý nebol ani v stave \$ ani v stave ¢. Rovnako aj teraz platí nerovnosť $f(k) = f(2l-1) = 3l-4+f(l) \leq 3l-4+3l < 6l-3 = 3k$, tak sme tvrdenie dokázali aj pre k. Ak $t :_i^{2l-1} \xi_1 0^{2l-4} 1' \xi_2$, postupujeme podobne ako pri párnom počte automatov.

Zoberme n ($n \geq 1$) automatov a nech ich konfigurácia na začiatku výpočtu (teda v nultej sekunde) je $0 :_0^{n+2} \$10^{n-1}\$$. Podľa predošlého tvrdenia existuje číslo $f(n+2) \leq 3n+6$ také, že konfigurácia $f(n+2) :_0^{n+2} \$¢^n\$$, pričom od začiatku výpočtu do $(f(n+2)-1)$-vej sekundy každý z automatov v stave \$ alebo ¢ mal aspoň jedného suseda, ktorý nebol ani v stave \$ ani v stave ¢. Keďže automat môže nadobudnúť stav S len pomocou pravidla (vi), tak žiaden z automatov sa nemohol počas doterajšieho výpočtu dostať do stavu S, ale práve podľa tohto pravidla bude v $(f(n+2)+1)$-vej sekunde platiť $f(n+2)+1 :_0^{n+2} \$S^n\$$.

Ukázali, že po $f(n+2)+1 \leq 3n+7$ sekundách sa poprvýkrat dostane niektorý automat do stavu S, a síce sa doň vtedy dostanú naraz všetky automaty. Dokázali sme, že výpočet sa zastaví a že konečná konfigurácia zodpovedá zadaniu, preto skonštruovaný automat je korektný a rieší daný problém.

Z dôkazu tiež vyplýva, že výpočet trvá $f(n+2)+1 \leq 3n+7$ sekúnd, kde n je počet automatov. Takže časová zložitosť výpočtu je $O(n)$.

Výsledková listina po 2. kole kategórie KSP

	Meno a priezvisko	Škola	Ročník		21	22	23	24	25	Σ
1	Jančuška Marek	Gym. Párovská Nitra	3	72	15	15	14	15	14	145
2	Repovský Michal	Gym. Komenského Trebišov	3	65	5	15	13	15	13	126
3	Nánási Michal	Gym. Jura Hronca BA	2	62	11	15	10	14	10	122
3	Poláček Lukáš	Gym. K. Štúra Modra	3	56	10	15	13	15	13	122
5	Perešíni Peter	Gym. Tajovského B. Bystrica	1	48	12	15	14	14	9	112
6	Lenhardt Rastislav	Gym. Jura Hronca BA	3	56	10	15	10	14		105
7	Kováč Jakub	Gym. Jura Hronca BA	3	49	12	15	13	15		104
7	Kuchynárová Mariana	Gym. Jura Hronca BA	3	50	8	15	9	9	13	104
7	Simančík František	Gym. Grösslingová BA	2	59	15	15	15			104
10	Bernát Marek	Gym. K. Štúra Modra	3	38	12	15	14	14		93
10	Štefanek Anton	Gym. Jura Hronca BA	3	40	9	15	14	15		93
12	Ludha Marek	Gym. Tajovského B. Bystrica	3	35	8	15	13	10	11	92
13	Glaus Peter	Gym. Jura Hronca BA	4	45	9	15	10	10	2	91
14	Baláž Miroslav	Gym. Jura Hronca BA	3	53	10		11	13		87
14	Smažáková Dana	Gym. Jura Hronca BA	3	35	10	14	9	9	10	87
16	Rejda Martin	Gym. Grösslingová BA	3	39		15	10	14		78
17	Imriška Jakub	Gym. Jura Hronca BA	1	23	3	15	10	10	13	74
18	Trejbal Ivan	Gym. Jura Hronca BA	3	27		15	7	10	9	68
19	Tekeľ Jakub	Gym. Jura Hronca BA	3	67						67
20	Buštor Stano	Gym. Jura Hronca BA	2	35	8		9	10	0	62
21	Zeman Marek	Gym. Jura Hronca BA	2	20	9	10	7	9		55
21	Ďuriš Michal	Gym. Grösslingová BA	2	55						55
23	Petruľák Matúš	Gym. Grösslingová BA	2	45						45
24	Takáč Slavomír	Neznama skola	1	24		10		10		44
25	Šmitala František	Gym. Cyrila a Metoda Nitra	3	36						36
26	Baník Dušan	Gym. Popradské nábr. Poprad	3	30						30
26	Štolc Miroslav	Gym. Párovská Nitra	3	30						30
28	Hruda Juraj	SPŠE Stará Turá	2	29						29
29	Hanulová Anna	Gym. Jura Hronca BA	3	0	5		11	12		28
30	Blénessy Tibor	Gym. Poštová Košice	3	27						27
31	Kotrbčík Michal	Gym. Jura Hronca BA	???	0			11	15		26
31	Plžík Milan	Gym. Ľ. Štúra Zvolen	2	16				10		26
33	Kadák Michal	Gym. Ľudovíta Štúra Trenčín	3	15	1			9		25
34	Kollár Juraj	Gym. Jura Hronca BA	2	23						23
35	Sedlák Milan	Gym. Liptovský Hrádok	3	12			10			22
36	Vadkerti Miroslav	SPŠE Nové Zámky	3	0	2	10		9		21
37	Ružička Peter	Gym. Jura Hronca BA	2	20						20
37	Šufliarsky Peter	Gym. Nové Zámky	3	20						20
39	Šuška Martin	Gymnázium Levice	4	17						17
40	Bajtoš Maroš	Gym. Jura Hronca BA	2	13						13
40	Janík Oliver	Gym. L. Stöckela Bardejov	3	13						13