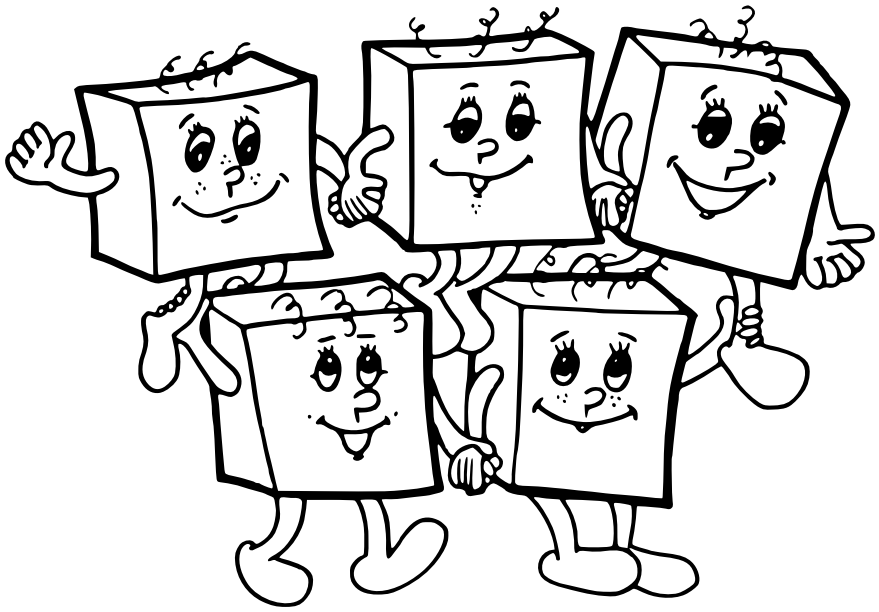
OLYMPIÁDA V INFORMATIKE
NA STREDNÝCH ŠKOLÁCH



dvadsiaty druhý (a zároveň prvý) **ročník**
školský rok 2006/07

riešenia krajského kola
kategória A

- **Olympiáda v informatike** je od tohto školského roku 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/`.
- Novinkou je zavedenie **dvoch kategórií**. Nová kategória B je určená pre mladších riešiteľov.

Riešenia kategórie A

A-II-1 Stále ešte zasypané mesto

Očíslujme si možné zamerania: stredovek bude 1, starovek 2 a archeológia 3. Nech a_1, a_2 a a_3 sú počty prvákov s jednotlivými zameraniami, podobne nech b_1, b_2, b_3 sú počty druhákov. Tieto počty si vieme spočítať pri načítavaní vstupu.

Začnime tým, že sa pozrieme na nejaké situácie, kedy riešenie nebude existovať. Všimnime si napríklad prvákov so zameraním 1. Každý z nich musí mať partnera druháka so zameraním 2 alebo 3. Ak možných partnerov nemáme dosť (teda platí $a_1 > b_2+b_3$), riešenie určite neexistuje. Ďalšie podmienky dostaneme cyklickou obmenou tejto úvahy.

Keďže platí $a_1 + a_2 + a_3 = b_1 + b_2 + b_3 = N$, vieme získané podmienky upraviť do ekvivalentného tvaru: Aby existovalo riešenie, musí nutne platiť ($\heartsuit$) $a_1 + b_1 \leq N$, $a_2 + b_2 \leq N$ a $a_3 + b_3 \leq N$. V ďalšom texte ukážeme, že ak sú tieto podmienky splnené, riešenie vždy existuje.

Bez ujmy na všeobecnosti predpokladajme ($\star$), že $a_1 \geq a_2 \geq a_3$ a $a_3 \leq \min\{b_1, b_2, b_3\}$. (Inými slovami, vždy vieme dosiahnuť, aby tieto nerovnosti platili, stačí vhodne očíslovať zamerania a prípadne vymeniť prvákov s druhákmi.) Rozoberieme teraz dva prípady.

Nech platí $a_2 \leq b_1$. Potom vytvoríme dvojice nasledovne: Bude a_2 dvojíc (prvák so zameraním 2)+(druhák so zameraním 1). Zostalo nám $b_1 - a_2$ druhákov so zameraním 1. Keďže vieme zo $\heartsuit$, že $b_1 \leq a_2 + a_3$, je $b_1 - a_2 \leq a_3$. Zvyšných druhákov so zameraním 1 teda popárujeme s niektorými prvákmi so zameraním 3. Zostalo nám niekoľko (presnejšie $a_3 - b_1 + a_2$) prvákov so zameraním 3. Keďže podľa našich predpokladov $\star$ je $a_3 \leq b_2$, zjavne môžeme všetkých týchto prvákov popárovať s druhákmi so zameraním 2. A už sme vyhrali – ostali nám len prváci so zameraním 1 a druháci so zameraniami 2 a 3.

Zostáva nám vyriešiť prípad keď $a_2 > b_1$. Opäť začneme dvojicami (prvák so zameraním 2)+(druhák so zameraním 1), tentokrát ich bude b_1 a zostanú nám nejakí nepriradení prváci so zameraním 2. Tých priradíme druhákom so

zameraním 3 (podľa ♡ ich máme dosť). Podľa ⋆ máme aspoň toľko druhákov so zameraním 2 ako prvákov so zameraním 3, popárujeme ich, koľko ide. A opäť sme už vyhrali – ostali nám len prváci so zameraním 1 a druháci so zameraniami 2 a 3.

Takto sme dostali algoritmus, ktorého časová zložitosť (až na načítanie vstupu a výpis výstupu) aj pamäťová zložitosť je konštantná.

Listing programu:

```
program mesto;
var N:longint;
    a,b:array[1..3] of longint;
const spec:array[1..3] of string = ('starovek','stredovek','archeologia');
var rocnik_vymeneny:boolean; { pokiaľ a[3]<b[3], vymeníme ročníky }
    subor:text;

procedure nacitaj;
var subor:text;
    i,rocnik:longint;
    s:string;
begin
  assign(subor,'mesto.in');
  reset(subor);
  readln(subor,N);
  for i:=1 to 3 do begin a[i]:=0; b[i]:=0; end;
  for i:=1 to 2*N do begin
    readln(subor,rocnik,s);
    if rocnik=1 then for i:=1 to 3 do if (s=' '+spec[i]) then inc(a[i]);
    if rocnik=2 then for i:=1 to 3 do if (s=' '+spec[i]) then inc(b[i]);
  end;
  rocnik_vymeneny:=false;
end;

procedure zorad;
var x,i,j:longint;
    s:string;
begin
  for j:=1 to 3 do for i:=1 to j-1 do if a[i]<a[j] then begin
     x:=a[i]; a[i]:=a[j]; a[j]:=x;
     x:=b[i]; b[i]:=b[j]; b[j]:=x;
     s:=spec[i]; spec[i]:=spec[j]; spec[j]:=s;
  end;
  if b[3]<a[3] then begin
    for i:=1 to 3 do begin x:=a[i]; a[i]:=b[i]; b[i]:=x; end;
    rocnik_vymeneny:=true;
    zorad;
  end;
end;

procedure vypis(pocet: longint; spec1,spec2: integer);
  { vypíše "pocet" dvojíc študentov so špecializáciami spec1 a spec2 }
```

```
var x:integer;
begin
  if rocnik_vymeneny then begin x:=spec1; spec1:=spec2; spec2:=x end;
  for x:=1 to pocet do writeln(subor,spec[spec1],' ',spec[spec2]);
end;

begin
  nacitaj;
  zorad;
  assign(subor,'mesto.out');
  rewrite(subor);
  if (a[1]+b[1]>N) or (a[2]+b[2]>N) or (a[3]+b[3]>N) then begin
    writeln(subor,'Študenti sa nedajú rozdeliť do dvojíc.');
  end else
    if (a[2]<=b[1]) then begin
      vypis(a[2],2,1);
      vypis(b[1]-a[2],3,1);
      vypis(a[3]-(b[1]-a[2]),3,2);
      vypis(a[1]-b[3],1,2);
      vypis(b[3],1,3);
    end else begin
      vypis(b[1],2,1);
      vypis(a[2]-b[1],2,3);
      vypis(a[3],3,2);
      vypis(b[2]-a[3],1,2);
      vypis(b[3]-(a[2]-b[1]),1,3);
    end;
  close(subor);
end.
```

A-II-2 Nová okružná jazda

Na úvod si zopakujme terminológiu z riešenia úlohy domáceho kola. Mapu Blatislavy budeme volať graf, križovatky sú jeho vrcholy a ulice tvoria jeho hrany. Počet vrcholov je N, počet hrán je M.

Hranu medzi vrcholmi u a v značíme uv. Stupeň vrcholu je počet hrán, ktoré z neho vychádzajú. Podľa zadania má náš graf všetky stupne vrcholov párne a väčšie ako 2. Hovoríme, že hrany e a f na seba nadväzujú, ak majú spoločný vrchol.

Ťah je postupnosť vrcholov taká, že každé dva po sebe nasledujúce vrcholy sú spojené hranou, pričom žiadna hrana nie je použitá viac ako jedenkrát. Uzavretý ťah je ťah, ktorý začína a končí v tom istom vrchole. Eulerovský ťah je ťah, v ktorom je každá hrana grafu použitá práve raz. Našou úlohou je teda nájsť v danom grafe vhodný uzavretý eulerovský ťah.

Pre jednoduchšie vysvetlenie algoritmu špeciálne ošetríme vrcholy stupňa 4. Ľahko overíme, že keď zakážeme 3 prechody cez takýto vrchol, buď sa cez neho nebude dať dvakrát prejsť, alebo bude jednoznačne určené, ako tieto prechody vyzerajú. V prvom prípade hľadaný ťah neexistuje, v druhom môžeme prechody cez tento vrchol nahradiť dvoma hranami.

Takto sa zbavíme všetkých vrcholov stupňa 4. Môžeme teda predpokladať, že každý vrchol v zadanom grafe má stupeň aspoň 6. Ukážeme, že ak je takýto graf súvislý, tak v ňom hľadaný ťah existuje.

Začneme tým, že graf rozložíme na niekoľko uzavretých ťahov, z ktorých žiaden nebude obsahovať zakázaný prechod: V každom vrchole popárujeme hrany do dvojíc tak, aby žiadna dvojica netvorila zakázaný prechod. Keďže vrcholy sú stupňa 6 a viac, toto zjavne vždy vieme spraviť. Tým už sme jednoznačne určili nejakú množinu uzavretých ťahov, ktoré dokopy obsahujú každú hranu grafu práve raz. (Začneme ľubovoľnou hranou, a vždy, keď prídeme do vrcholu, odídeme hranou, ktorá s ňou tam bola v páre, až kým sa nevrátime ku hrane, kde sme začali.)

Ak sa nám takto graf rozpadol na viac ako jeden ťah, musíme teraz ťahy pospájať do jedného. Zvolíme si ľubovoľný jeden ťah, nazveme ho hlavným ťahom a ostatné k nemu budeme pripájať.

Ak sa niekedy ocitneme v situácii, že ešte máme nepripojené ťahy, ale žiadny z nich nemá spoločný vrchol s hlavným ťahom, znamená to, že náš graf nie je súvislý, a teda v ňom eulerovský ťah neexistuje. Nech teraz v je vrchol, ktorým okrem hlavného ťahu prechádza ešte nejaký iný ťah. Ukážeme, ako tieto dva ťahy spojiť do jedného ťahu bez zakázaných prechodov.

Nech vo v na seba (okrem iného) nadväzujú hrany e_1 a e_2 v hlavnom ťahu a hrany e_3 a e_4 v nejakom inom ťahu. Ak sú prechody e_1e_3 a e_2e_4 povolené, môžeme tieto dva ťahy spojiť do jedného – nebude nadväzovať (e_1 na e_2) a (e_3 na e_4), ale (e_1 na e_3) a (e_2 na e_4). Ukážeme teraz, že takéto dve dvojice hrán určite vieme nájsť. Rozoberieme dva prípady:

- Nech hlavný ťah prechádza cez v len raz, cez hrany e_1 a e_2. Potom vo v sú aspoň dva páry hrán, ktoré do hlavného ťahu nepatria. Označme dva z nich e_3e_4 a e_5e_6. Všimnime si teraz nasledovné 4 dvojice prechodov: (e_1e_3

a e_2e_4), (e_1e_4 a e_2e_3), (e_1e_5 a e_2e_6) a (e_1e_6 a e_2e_5). Keďže zakázané sú len 3 prechody cez v, v niektorej z týchto 4 možností musia byť oba prechody povolené. Táto možnosť teda hovorí, ako môžeme nejaký ďalší ťah pripojiť ku hlavnému.

- Nech teda hlavný ťah prechádza cez v aspoň dvakrát. Zoberme dva takéto prechody a označme ich e_3e_4 a e_5e_6. Zoberme jeden prechod iného ťahu cez v a označme ho e_1e_2. Rovnakou úvahou ako v predchádzajúcom prípade zistíme, že niektorý zo štyroch spôsobov ako tieto dva ťahy spojiť nie je zakázaný.

Posledná otázka: S akou časovou zložitosťou vieme tento algoritmus implementovať? V každom vrchole si budeme pamätať, ktoré hrany na seba nadväzujú a ktoré patria do hlavného ťahu. Ďalej budeme mať v každom vrchole zoznam hrán, ktoré do hlavného ťahu nepatria a nanajvýš dva prechody, ktoré do hlavného ťahu patria. Okrem toho budeme mať jeden globálny zoznam W vrcholov, cez ktoré ide hlavný ťah, ale ktoré ním ešte nie sú celé pokryté.

Náš program bude fungovať nasledovne: Ak je W prázdny, skončili sme. V opačnom prípade nech v je ľubovoľný vrchol z W. Pomocou pamätaných údajov vieme v konštantnom čase predĺžiť hlavný ťah vo vrchole v, a následne prejsť po pridanej časti ťahu a upraviť informácie vo vrcholoch, cez ktoré prechádza. Tento prechod má časovú zložitosť lineárnu od veľkosti pridanej časti. Celý algoritmus má teda časovú aj pamäťovú zložitosť lineárnu od veľkosti zadaného grafu.

(Listing programu z priestorových dôvodov neuvádzame, v prípade záujmu ho nájdete vo verzii, ktorá bude po skončení súťaže uverejnená na webe OI.)

A-II-3 Rozvoz pizze vo väčšom

Úlohu budeme riešiť metódou dynamického programovania. Postupne pre každé i spočítame nasledujúce údaje:

- Počet pizzérií P_i, ktoré treba na pokrytie úsekov i až N.

- Najväčší index Z_i taký, že platí: Ak máme P_i pizzérií, tak nimi okrem úsekov i až N vieme ešte pokryť aj úseky 1 až Z_i.
- Ako pomocný údaj si ešte budeme pamätať počet úsekov U_i a súčet počtov zákazníkov S_i, ktorých obsluhuje pri vyššie popísanom riešení „prvá“ pizzéria, teda pizzéria obsluhujúca úseky od i ďalej.

Na začiatku vieme povedať, že $P_N = 1$. V lineárnom čase nájdeme hodnotu Z_N (pridávame úseky, kým by súčet počtov zákazníkov nemal prekročiť K). U_N zatiaľ nebude zaujímavé, ale pre poriadok ho môžeme nastaviť na $Z_N + 1$, S_N bude práve zistený súčet počtov zákazníkov.

Teraz ideme nájsť všetky i, pre ktoré ešte platí, že $P_i = 1$. Nech teda už poznáme hodnoty $P_{i+1} = 1$, Z_{i+1} a S_{i+1}. Ak $A_i + S_{i+1} \leq K$, môžeme našej zatiaľ jedinej pizzérii jednoducho pridať nový úsek a nič sa nemení. Bude teda $P_i = 1$, $Z_i = Z_{i+1}$, $U_i = (N + 1 - i) + Z_i$ a $S_i = A_i + S_{i+1}$. V opačnom prípade už nebudeme vedieť pokryť toľko úsekov „na začiatku“, teda musíme postupne zmenšovať Z_i (a adekvátne upravovať U_i a S_i), až kým nebude $S_i \leq K$.

Takto pokračujeme, kým $Z_i \geq 0$. Akonáhle sa dostaneme do situácie, že je $Z_i = 0$, a napriek tomu už je S_i priveľké, znamená to, že budeme potrebovať druhú pizzériu. Od tohoto okamihu budeme nové hodnoty počítať nasledovne:

Položíme $U_i = U_{i+1} + 1$ a $S_i = A_i + S_{i+1}$, teda pridáme „prvej“ pizzérii nový úsek. Kým $S_i > K$, zmenšujeme U_i o 1 a S_i o A_{i+U_i}. Takto vlastne „prvej“ pizzérii priradíme najväčší možný počet úsekov začínajúci i-tym. A čo ďalej? Zostáva nám pokryť úseky od $i + U_i$ po N. Toto sme už ale riešili a optimálne riešenie máme zapísané. Bude teda $P_i = 1 + P_{i+U_i}$ a $Z_i = Z_{i+U_i}$.

Spočítali sme teda hodnoty P_i a Z_i. Potrebujeme ešte ukázať dve veci: akú má ich výpočet časovú zložitosť a ako teraz zistíme optimálne riešenie pre celý kruh.

Tvrdíme, že náš výpočet má časovú zložitosť $O(N)$, teda lineárnu od počtu úsekov. Prečo je tomu tak? Môže sa síce stať, že pri výpočte niektorej konkrétnej hodnoty U_i budeme musieť prvej pizzérii postupne zobrať veľa úsekov, všimnime si ale, že dokopy za celý výpočet každý úsek prvej pizzérii zoberieme najviac jedenkrát. Preto na výpočet všetkých hodnôt U_i a S_i dokopy nám stačí čas $O(N)$. Výpočet hodnôt P_i a Z_i nám zjavne zložitosť nepokazí.

A ako teda nájsť optimálne riešenie pôvodnej úlohy? Predstavme si takéto optimálne riešenie. Ak nejaká pizzéria obsluhuje úseky 1 aj N, na chvíľu ju zrušíme. Teraz nech x je číslo najmenšieho obsluhovaného úseku. Potom zjavne P_x je počet pizzérií v optimálnom riešení. A môžeme si všimnúť, že $Z_x \geq x - 1$, lebo posledná použitá pizzéria musí pokryť aj prvých $x - 1$ úsekov.

My ale hodnotu x nepoznáme. Nič nám však nebráni v čase $O(N)$ vyskúšať všetky možné x, pre ktoré $Z_x \geq x - 1$, a zobrať minimum z hodnôt P_x.

Listing programu:

```
program pizza;
const maxN=100000;
var A: array[1..maxN] of longint;
    N, K: longint;
    Pocet, Kam, Dalsi: array[1..maxN] of longint;
    { Pocet[i] je minimální počet intervalů nutných k pokrytí úseků od i do N;
      tyto intervaly navíc pokryjí všechny úseky od i do Kam[i];
      v optimálním pokrytí začíná po intervalu se začátkem i
      další interval na Dalsi[i] }
    soubor: text;
    i, j: longint;
    soucet, optimum: longint;

begin
  assign(soubor,'pizza.in');
  reset(soubor);
  readln(soubor,N,K);
  for i:=1 to N do readln(soubor,A[i]);
  close(soubor);
  { nejdříve inicializujeme konec pole }
  { j bude největší číslo takové, že součet A[1] až A[j] je menší nebo roven K }
  j:=1; soucet:=A[1];
  while (j<=N) and (soucet+A[j]<=K) do begin
    soucet:=soucet+A[j];
    inc(j);
  end;
  if j=N+1 then begin
    { stačí jedna pizzerie }
    assign(soubor,'pizza.out');
    rewrite(soubor);
    writeln(soubor,1);
    writeln(soubor,1,' ',N);
    close(soubor);
    exit
  end;
  { nyní spočítáme hodnoty na konci pole }
  i:=N;
  { i bude index prvku, jehož hodnoty nyní počítáme }
  while j>0 do begin
    if soucet+A[i]>K then begin
      soucet:=soucet-A[j];
```

```
      dec(j);
      continue;
    end;
    Pocet[i]:=1;
    Kam[i]:=j;
    Dalsi[i]:=j+1;
    soucet:=soucet+A[i];
    dec(i);
  end;
  while soucet+A[i]<=K do begin
    Pocet[i]:=1;
    Kam[i]:=0;
    Dalsi[i]:=N+1;
    soucet:=soucet+A[i];
    dec(i);
  end;
  { i je stále index počítaného prvku a j je maximální
    index takový, že součet A[i+1] až A[j] je menší nebo roven K }
  j:=N;
  while i>0 do begin
    if soucet+A[i]>K then begin
      soucet:=soucet-A[j];
      dec(j);
      continue;
    end;
    Pocet[i]:=Pocet[j+1]+1;
    Kam[i]:=Kam[j+1];
    Dalsi[i]:=j+1;
    soucet:=soucet+A[i];
    dec(i);
  end;
  { zbývá nalézt optimum }
  optimum:=Pocet[1]; j:=1;
  for i:=1 to N do
    if (Kam[i]+1>=i) and (optimum>Pocet[i]) then begin
      optimum:=Pocet[i];
      j:=i;
    end;
  assign(soubor,'pizza.out');
  rewrite(soubor);
  writeln(soubor,optimum);
  i:=j;
  while Pocet[j]>1 do begin
    writeln(soubor,j,' ',Dalsi[j]-1);
    j:=Dalsi[j];
  end;
  if i=1 then
    writeln(soubor,j,' ',N)
  else
    writeln(soubor,j,' ',i-1);
  close(soubor);
end.
```

A-II-4 Grafomat loví zver

Zaoberajme sa najskôr jednoduchším problémom. Majme v grafe dva vrcholy l a p, ktoré sú spojené cestou. Chceme ich zosynchronizovať, teda spôsobiť, aby nejaká udalosť („zazretie zvieraťa“) v p spôsobila, že časom tá istá udalosť („výstrel“) nastane v oboch naraz.

Toto vieme dosiahnuť napr. trikom s rôzne rýchlymi signálmi, ktorý sme si ukázali v riešeniach domáceho kola. Vyšleme z p do l dva signály, z ktorých jeden ide polovičnou rýchlosťou ako druhý. Keď rýchlejší príde do l, odrazí sa a putuje naspäť. V okamihu, keď sa rýchlejší signál vráti do p, príde do l pomalší signál. V programe to vyzerá takto:

Listing programu:

```
var x: 0..3;               { 1=l, 2=p, 3=p během výpočtu, 0=ostatní }
    y: 0..1 = 0;           { výstup }
    a, aa, b: 0..3 = 0;    { signály a kolik taktů zbývá do jejich předání dál }
begin
   { Na počátku vyšleme signály A a B. }
   if x=2 then begin x:=3; a:=2; b:=3; end;

   { Signál A putuje vlevo rychlostí 1, vlevo se odrazí a je z něj A'. }
   if a>0 then dec(a);
   if S[1].a=1 then a:=1;
   if (x=1) and (a>0) then begin a:=0; aa:=2; end;

   { Signál A' putuje vpravo rychlostí 1. }
   if aa>0 then dec(aa);
   if S[2].aa=1 then aa:=1;

   { Signál B putuje vlevo rychlostí 1/2. }
   if b>0 then dec(b);
   if S[1].b=1 then b:=2;

   { Když A' a B doputují na protilehlý konec, vypálíme. }
   if (x=1) and (b>0) then begin b:=0; y:=1; end;
   if (x=3) and (aa>0) then begin aa:=0; y:=1; end;

   { Není-li co dělat, končíme. }
   if (a=0) and (aa=0) and (b=0) then stop;
end.
```

Keď vieme synchronizovať dvojice vrcholov, môžeme väčší počet vrcholov zosynchronizovať použitím metódy rozdeľ a panuj. Predstavme si, že máme $2^k + 1$ vrcholov v rade, pričom najľavejší z nich je na začiatku označený ako začiatok intervalu a najpravejší ako zarážka za koncom intervalu. Ako zosynchronizovať

všetky vrcholy okrem najpravejšieho?

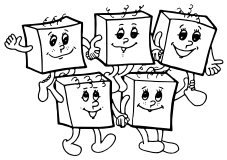
Pre $k = 1$ máme zosynchronizovať dva vrcholy, a to už vieme spraviť. Pre väčšie k rozdelíme náš interval na dve rovnako veľké polovice, zosynchronizujeme ich najľavejšie vrcholy, a potom paralelne zosynchronizujeme každú polovicu zvlášť. Ako rozdeliť interval na polovicu, keď nepoznáme jeho dĺžku? Opäť použijeme dva signály, tentokrát pomalší pôjde tretinovou rýchlosťou. Kým rýchlejší signál prejde na koniec, odrazí sa a vráti sa do polovice, pomalší práve dorazí do polovice intervalu. Keď sa teda signály stretnú, našli sme stred intervalu.

Majme teraz $N = 2^k$ vrcholov na kružnici. Tam to bude fungovať úplne rovnako, len vrchol, ktorý „zazrel zver“, bude v prvom kole slúžiť aj ako ľavý okraj intervalu, aj ako zarážka za pravým okrajom.

Každá iterácia tohoto algoritmu trvá lineárne dlho od dĺžky intervalov, ktoré práve synchronizujeme. Celková časová zložitosť teda je $O(N + N/2 + N/4 + \cdots + 1) = O(N)$.

Listing programu:

```
var x: 0..3;                        { 1=počátek, 2=okraj, 3=střed intervalu }
    y: 0..1 = 0;                    { výstřel }
    a, aa, b, c, cc, d: 0..3 = 0; { signály a kolik taktů zbývá do předání dál }

begin
   { Triviální vstup => vypálíme hned. }
   if (x=1) and (S[1].x=1) then begin y:=1; stop; end;

   { Na počátku vyšleme signály C a D. }
   if x=1 then begin x:=2; d:=3; end;
   if S[2].x=1 then c:=2;

   { Signál C putuje vpravo rychlostí 1, vpravo se odrazí jako C'. }
   if c>0 then dec(c);
   if S[2].c=1 then c:=1;
   if (x=2) and (c>0) then begin c:=0; cc:=2; end;

   { Signál C' putuje vlevo rychlostí 1. }
   if cc>0 then dec(cc);
   if S[1].cc=1 then cc:=1;

   { Signál D putuje vpravo rychlostí 1/3. }
   if d>0 then dec(d);
   if S[2].d=1 then d:=3;

   { Když se C' a D potkají, máme střed. Vyšleme z něj A a B. }
   if (x=0) and (cc>0) and (d>0) then begin cc:=0; d:=0; x:=3; a:=2; b:=3; end;
```




```
   { Signál A putuje vlevo rychlostí 1, vlevo se odrazí a je z něj A'. }
   if a>0 then dec(a);
   if S[1].a=1 then a:=1;
   if (x=2) and (a>0) then begin a:=0; aa:=2; end;

   { Signál A' putuje vpravo rychlostí 1. }
   if aa>0 then dec(aa);
   if S[2].aa=1 then aa:=1;

   { Signál B putuje vlevo rychlostí 1/2. }
   if b>0 then dec(b);
   if S[1].b=1 then b:=2;

   { Když A' doputuje zpět do středu a B doleva, spustíme se v obou polovinách. }
   if (x=2) and (b>0) then begin b:=0; x:=1; end;
   if (x=3) and (aa>0) then begin aa:=0; x:=1; end;

   { Není-li co dělat, končíme. }
   if (a=0) and (aa=0) and (b=0) and (c=0) and (cc=0) and (d=0) and (x<>1)
      then stop;
end.
```

Poznámka: Riešenie pre všeobecnú kružnicu nie je o toľko ťažšie. Keď delíme na polovicu interval nepárnej dĺžky, to, že je nepárnej dĺžky, zistíme tak, že sa nám signály v strede nestretnú presne. V takomto prípade stredný vrchol vynecháme a nebudeme ho synchronizovať. Skončíme s tým, že zosynchronizujeme nejakú množinu vrcholov, pričom žiadne dva nezosynchronizované spolu nesusedia. Stačí teda, aby si namiesto „strieľaj“ zosynchronizované vrcholy nastavili novú značku „zamier“. A v nasledujúcom takte vystrelí každý, kto vidí značku „zamier“ u seba alebo niektorého suseda.

SLOVENSKÁ KOMISIA OLYMPIÁDY V INFORMATIKE

DVADSIATY DRUHÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Zodpovedný redaktor: Michal Forišek
Sadzba programom LaTeX