MATEMATICKÁ OLYMPIÁDA NA STREDNÝCH ŠKOLÁCH

55. ročník, školský rok 2005/2006

Zadania úloh 1. kola kategórie P

Matematická olympiáda je súťaž pre študentov stredných škôl našej republiky. **Kategória P** je zameraná na programovanie a je určená študentom všetkých ročníkov.

Organizácia súťaže v kategórii P

Kategória P matematickej olympiády má tri postupové kolá – domáce, krajské a celoštátne.

V **I. kole** účastníci riešia štyri úlohy, uvedené v tomto letáku. Riešenia odovzdajú svojmu učiteľovi informatiky do **15. novembra 2005**. Učitelia informatiky odošlú riešenia v tomto termíne zo školy na príslušnú adresu podľa krajov takto:

Košický, Prešovský:
doc. RNDr. G. Andrejková, CSc., KMI PF UPJŠ, Jesenná 5, 041 54 Košice
Žilinský:
RNDr. P. Varša, Ph.D., KI FRI Žilinská univerzita, Moyzesova 20, 010 26 Žilina
Banskobystrický:
PaedDr. L. Huraj, KI FPV UMB, Tajovského 40, 974 01 Banská Bystrica
Trnavský, Trenčiansky, Nitriansky:
prof. Ing. V. Stoffová, CSc., KI FPV UKF, tr. A. Hlinku 1, 949 74 Nitra
Bratislavský:
RNDr. A. Blaho, KVI FMFI UK, Mlynská dolina, 842 48 Bratislava 4

Najúspešnejší riešitelia domáceho kola sú pozvaní do **II. kola** (krajského), kde riešia štyri teoretické úlohy. Do **III. kola** (celoštátneho) sú pozývaní najúspešnejší riešitelia všetkých krajských kôl, pričom riešia tri teoretické úlohy a dve praktické úlohy pri počítači. Z najlepších riešiteľov tohto kola SK MO vyberie družstvá pre Medzinárodnú informatickú olympiádu a Stredoeurópsku informatickú olympiádu.

Predbežné termíny 55. ročníka MO, kategória P

I.	Domáce kolo	15. novembra 2005
II.	Krajské kolo	17. januára 2006
III.	Celoštátne kolo	5.-8. apríla 2006

Usporiadateľ súťaže

Matematickú olympiádu vyhlasuje *Ministerstvo školstva SR* v spolupráci s *Jednotou slovenských matematikov a fyzikov* a *Slovenskou komisiou Matematickej olympiády*. Súťaž organizuje *Slovenská komisia MO* a v jednotlivých krajoch ju riadia *krajské komisie MO* pri pobočkách JSMF. Na jednotlivých školách ju zaisťujú učitelia matematiky a informatiky. Celoštátne kolo MO, tlač materiálov a ich distribúciu po organizačnej stránke zabezpečuje IUVENTA v tesnej súčinnosti so Slovenskou komisiou MO.

Formálna úprava riešenia

Riešenia súťažných úloh domáceho kola kategórie P pozostávajú z dvoch častí:

Popis riešenia. Riešenia musia obsahovať podrobný popis použitého algoritmu, **zdôvodnenie jeho správnosti** a diskusiu o efektivite zvoleného riešenia (t.j. posúdenie časových a pamäťových nárokov programu). Algoritmus by mal byť jasný už z popisu riešenia bez toho, aby bolo potrebné nahliadnuť do programu.

Program. V úlohách **P–I–1**, **P–I–2** a **P–I–3** je potrebné k riešeniu pripojiť odladený program – najlepšie je, ak je napísaný v jazyku Pascal, C alebo C++. Program sa odovzdáva v písomnej forme (jeho výpis je teda súčasťou riešenia) i na CD (resp. diskete), aby bolo možné otestovať jeho funkčnosť.

Súbory na diskete pomenujte `p1`x`.pas/.c/.cpp`, kde x je číslo súťažnej úlohy. CD označte menom riešiteľa. Z jednej školy možno poslať všetky riešenia na jednom CD. V tomto prípade pre každého riešiteľa vytvorte podadresár označený jeho priezviskom a CD označte adresou školy.

V úlohe **P–I–4** je potrebné uviesť program pre paralelizátor.

Písomnú časť riešenia vypracujte čitateľne na listy formátu A4. **Každú úlohu začnite na novom liste** a v záhlaví uveďte vaše meno, triedu, adresu školy a označenie príkladu podľa tohto letáku. Zadania úloh nemusíte opisovať. Ak sa vám riešenie úlohy nezmestí na jeden list, uveďte na ďalších listoch vľavo hore svoje meno a označenie úlohy, listy očíslujte a zopnite.

P–I–1

Na monitore sa práve schyľuje k veľkej bitke medzi armádou hráča a armádou jeho počítača. Sily sú vyrovnané, obe armády obsahujú rovnaký počet plukov, pričom ale jednotlivé pluky môžu obsahovať rôzne počty vojakov. Na začiatku bitky sa pluky oboch armád zoradia do dvoch radov tak, že oproti každému hráčovmu pluku stojí práve jeden pluk patriaci počítaču. Potom sa začne samotný boj. Pluky stojace oproti sebe na seba zaútočia. A keďže v počte je sila, zvíťazí ten z nich, ktorý má viac vojakov. Ak náhodou majú súperiace pluky rovnaký počet vojakov, vyhráva pluk patriaci počítaču.

Hráčova armáda má veľmi schopných špiónov, ktorí pred bitkou zistia, koľko vojakov má nepriateľ v ktorom pluku a ako sú jeho pluky rozmiestnené. Vašou úlohou je rozmiestniť na základe týchto informácií hráčove pluky tak, aby čo najviac z nich svoj súboj vyhralo.

Úloha: Napíšte program, ktorý vám poradí, ako najlepšie rozmiestniť pluky, ktoré máte k dispozícii. Na vstupe dostanete počet plukov N v každej armáde a počty vojakov v každom z $2N$ plukov na bojisku. Ako výstup programu nám stačí jediné celé číslo – najväčší počet hráčovych plukov, ktoré vyhrajú svoj súboj pri nejakom rozostavení.

Formát vstupu: Prvý riadok vstupného súboru `pluky.in` obsahuje jedno celé číslo N ($1 \leq N \leq 10\,000$) – počet plukov v každej z armád. V druhom riadku je medzerami oddelených N celých čísel $A_1, \ldots, A_N$ ($1 \leq A_i \leq 100\,000\,000$) – počty vojakov v hráčovych plukoch. V treťom riadku je medzerami oddelených N celých čísel $B_1, \ldots, B_N$ ($1 \leq B_i \leq 100\,000\,000$) – počty vojakov v plukoch patriacich počítaču.

Formát výstupu: Jediný riadok výstupného súboru `pluky.out` má obsahovať jediné celé číslo – najväčší počet hráčovych plukov, ktoré môžu naraz vyhrať svoj súboj.

Príklad:

pluky.in	**pluky.out**
`5`	`3`
`7 12 1 7 47`	*(Ak to hráč spraví dobre, vyhrajú jeho pluky*
`7 12 1 7 47`	*veľkosti 47, 12 a jeden z plukov veľkosti 7.)*

pluky.in	**pluky.out**
`4`	`0`
`10 10 10 10`	*(Pri každom rozostavení všetky hráčove*
`10 10 10 10`	*pluky prehrajú.)*

pluky.in	**pluky.out**
`5`	`4`
`1 3 5 7 9`	*(Obetujeme hráčov najmenší pluk, pošleme*
`2 4 6 8 10`	*ho proti pluku veľkosti 10. Ostatné pluky vieme potom rozmiestniť tak, aby vyhrali.)*

P–I–2

Vedcom sa konečne podarilo vymyslieť efektívny spôsob cestovania v časopriestore. Ich testovacie stredisko sa skladá z niekoľkých lokalít. V každej lokalite je umiestnených niekoľko teleportov. Keď vstúpime do teleportu, presunie nás na vopred zadanú lokalitu (čo by sme od teleportu očakávali), ale navyše nás presunie aj v čase o zadaný počet minút (buď dopredu alebo dozadu). Vedci by chceli zistiť, či je cestovanie teleportami výhodné. Práve sa nachádzajú pri centrálnom počítači a chceli by sa ísť najesť do bufetu. A keďže čas sú peniaze, chceli by byť v bufete čo najskôr. Pohybovať v čase a priestore sa samozrejme chcú len pomocou už postavených teleportov.

Úloha: Program dostane na vstupe počet lokalít N, ktoré budeme označovať číslami $1, \ldots, N$. Centrálny počítač je v lokalite číslo 1, bufet má číslo N. Nasleduje počet teleportov M a zoznam týchto teleportov. Pre každý teleport je určená začiatočná lokalita, koncová lokalita a zmena času v minútach, ktorá nastane pri prechode týmto teleportom (kladné číslo znamená posun do budúcnosti, záporné do minulosti a 0 znamená, že sa na koncovej lokalite objavíme v tom istom čase, v akom sme nastúpili do teleportu).

Každý teleport sa dá použiť len tým smerom, ktorý je uvedený na vstupe. Medzi dvoma lokalitami môže byť viacero teleportov. Dokonca môže existovať teleport, ktorý nás presunie len v čase (teda koncová a začiatočná lokalita je tá istá).

Program má vypočítať čas, kedy najskôr možeme byť v lokalite N, ak sa v lokalite 1 nachádzame v čase 0. Ak tam vieme byť lubovoľne skoro (teda vieme cestovať do nekonečna do minulosti), alebo ak sa tam nevieme dostať vôbec, program by o tom mal podať príslušnú správu.

Formát vstupu: Prvý riadok vstupného súboru `teleport.in` obsahuje dve čísla N a M ($2 \leq N \leq 1\,000$, $0 \leq M \leq 50\,000$) oddelené medzerou. Nasleduje M riadkov, na každom sú tri čísla A_i, B_i, T_i ($1 \leq A_i, B_i \leq N$, $|T_i| \leq 10\,000$) popisujúce teleport z lokality A_i do lokality B_i so zmenou času T_i minút.

Formát výstupu: Jediný riadok výstupného súboru `teleport.out` má obsahovať správu „`Vedci umru od hladu`“, ak sa od centrálneho počí-

tača nedá dostať do bufetu, resp. správu „`Vedci spoznaju zaciatok vesmiru`“, ak vieme cestovať do nekonečna do minulosti. Inak má obsahovať najskorší čas v minútach, kedy sa vedia vedci dostať do bufetu.

Príklad:

teleport.in	**teleport.out**

`3 4`
`1 2 5`
`2 3 -7`
`1 3 -1`
`1 3 16`
	`-2`
	(Prvým teleportom sa dostanú do lokality 2 v čase 5, odtiaľ druhým do lokality 3 v čase $5+(-7)=-2$*. Ostatné možnosti sú horšie.)*

teleport.in	**teleport.out**

`2 2`
`1 1 -1`
`1 2 0`
	`Vedci spoznaju zaciatok vesmiru`
	(Skôr ako sa druhým teleportom presunú do bufetu, môžu prvým odísť ľubovoľne ďaleko do minulosti.)

teleport.in	**teleport.out**

`4 2`
`1 2 -1`
`2 3 0`
`4 3 10`
	`Vedci umru od hladu`
	(Posledný teleport nemôžu použiť na presun z lokality 3 do lokality 4, iba naopak.)

P–I–3

Kde bolelo, tam bolelo, žil raz v istom kráľovstve starý kráľ. Kráľovstvo tvorilo N miest a kde-tu nejaká cesta. Keďže králi sú od prírody lakomí, v kráľovstve veru nebolo udržiavaných ciest nazvyš. Presnejšie, ciest bolo práve $N-1$ a boli zvolené tak, aby sa medzi každými dvoma mestami dalo prejsť po cestách (možno idúc cez iné mestá). V reči teórie grafov takejto cestnej sieti hovoríme strom.

Na staré kolená však kráľa navštívila teta Paranoja a našepkala mu, že susedia chcú napadnúť jeho kráľovstvo. Preto sa rozhodol, že lakomosť musí nabok, postaví v mestách vojenské posádky. Paranoja však šepkala ďalej: „Zbláznil si sa? Ak sú dve posádky v susediacich mestách, budú si medzi sebou posielať odkazy. A vieš, ako to dopadne.... Nechaj veľa vojakov pokope, vzbúria sa proti tebe!“

Tri dni a tri noci kráľ nespal, až vyhútal nasledujúci kompromis: Vyberie niekoľko miest, v ktorých postaví vojenské posádky. Aby mu nehrozila vzbura, rozhodol sa, že nikdy nesmú byť pri sebe viac ako 3 posádky. Teraz sedí nad mapou a húta, ako ich len rozmiestniť, aby kráľovstvo bolo čo najbezpečnejšie.

Úloha: Ešte raz si formálnejšie zopakujme, o čo kráľovi ide.

Na vstupe máte počet miest N a popis ciest medzi nimi. Ciest je práve $N - 1$, nikde sa nekrižujú, nimi tvorená cestná sieť je súvislá a spája všetky mestá. Pre každé mesto i vieme číslo b_i – toto číslo hovorí, koľko pridá posádka v i-tom meste k bezpečnosti kráľovstva. Kráľovou (a vašou) úlohou je vybrať množinu miest, v ktorých postaviť posádky. Táto množina musí spĺňať nasledujúce podmienky:

- Každá jej súvislá podmnožina má veľkosť najviac 3.
- Spomedzi všetkých takýchto množín má najväčší možný súčet hodnôt b_i – bezpečnosť kráľovstva.

(Množinu miest voláme súvislá, ak sa medzi každými dvomi mestami z nej dá prejsť po cestách bez toho, aby sme navštívili mesto, ktoré do tejto množiny nepatrí.)

Formát vstupu: Prvý riadok vstupného súboru `posadky.in` obsahuje jedno číslo N ($1 \leq N \leq 100\,000$) – počet miest v kráľovstve. Mestá sú očíslované od 1 do N. Každý z nasledujúcich $N - 1$ riadkov obsahuje dve čísla miest, ktoré sú spojené cestou. Môžete predpokladať, že cestná sieť je súvislá.

Posledný riadok vstupného súboru obsahuje N celých čísel $b_1, \ldots, b_N$ ($0 \leq b_i \leq 10\,000$), ktoré udávajú, kde je ako výhodné mať vojenskú posádku.

Formát výstupu: Prvý riadok výstupného súboru `posadky.out` má obsahovať jediné celé číslo – najlepšiu dosiahnuteľnú bezpečnosť kráľovstva. Druhý riadok má obsahovať niekoľko čísel oddelených medzerami – jednu vhodnú množinu miest, pre ktorú sa uvedená bezpečnosť dosiahne.

Príklad:

posadky.in

```
7
1 2
2 3
3 4
4 5
5 6
6 7
1 1 1 1 1 1 1
```

posadky.out

```
6
1 2 3 5 6 7
```

(Všade je zisk z posádky rovnaký, chceme ich čo najviac.)

posadky.in

```
5
1 5
2 5
3 5
4 5
1 6 5 2 1000
```

posadky.out

```
1011
2 3 5
```

(Zjavne chceme posádku v meste 5. Potom už ale môžeme vybrať len 2 spomedzi ostatných miest.)

posadky.in

```
5
1 5
2 5
3 5
4 5
4 4 4 4 5
```

posadky.out

```
16
1 2 3 4
```

(Nie vždy sa oplatí vybrať mesto s najväčším b_i.)

P–I–4

Za siedmimi horami a šiestimi dolinami vynašiel vynálezca Kleofáš podivnú to mašinu, ktorú nazval *paralelizátor*. Na prvý pohľad vyzeral paralelizátor ako obyčajný počítač.... Bol tu však jeden malý, ale o to dôležitejší rozdiel. Za určitých okolností vedel paralelizátor paralelne (t.j. súčasne) spustiť viacero vetiev programu bez toho, aby ho to akokoľvek spomalilo. Kleofáš rýchlo pochopil, že zo slovného popisu tohto zázraku by nik nezmúdrel, a tak vymyslel aj programovací jazyk, v ktorom sa dali písať programy pre jeho paralelizátor.

Tento programovací jazyk je kópiou jazyka Pascal. Oproti klasickému Pascalu nemáme k dispozícii generátor náhodných čísel (a teda napríklad príkaz **Random**), takže je dopredu určené, ako bude beh každého programu vyzerať.

Programy pre paralelizátor sa budú od klasických jemne líšiť tým, že nebudú mať výstup. Budeme iba rozlišovať, či program skončil *úspešne* alebo nie. U klasických programov by to znamenalo, že nás zaujíma jedine tzv. exit code (po slovensky „návratová hodnota“) programu.

Oproti štandardnému Pascalu nám pribudli štyri príkazy: **Accept**, **Reject**, **Both**(x) a **Some**(x) (kde x je premenná typu integer). Čo každý z týchto príkazov robí?

Príkaz **Accept** *úspešne* ukončí bežiaci program.

Príkaz **Reject** ukončí bežiaci program, avšak *nie úspešne*. (To isté spraví aj vykonanie štandardných Pascalovských príkazov **Halt** a **End**., príkaz **Reject** definujeme len kvôli názornosti.)

V nasledujúcom texte pod *vytvorením kópie programu* rozumieme, že sa v operačnej pamäti vytvorí úplne presná kópia celého programu vrátane obsahu jeho premenných – výsledok bude rovnaký, ako keby sme už na začiatku daný program spustili nie raz, ale dvakrát.

Príkaz **Both**(x) zastaví aktuálne bežiaci program. Vytvoria sa dve jeho identické kópie. V prvej z nich je hodnota premennej x nastavená na 0, v druhej na 1. Obe kópie programu sú paralelne spustené, pričom ich výpočet pokračuje príkazom nasledujúcim za príslušným príkazom **Both**

Sú tri možnosti, ako môže výpočet dopadnúť:

- Ak obe kópie úspešne skončia, v nasledujúcom takte procesora úspešne skončí aj pôvodný program.
- Ak aspoň jedna kópia skončí, ale nie úspešne, v v nasledujúcom takte procesora skončí aj pôvodný program, tiež nie úspešne.
- Vo všetkých ostatných prípadoch pôvodný program nikdy neskončí.

Príkaz **Some**(x) funguje podobne. Taktiež zastaví aktuálne bežiaci program. Opäť sa vytvoria dve jeho identické kópie, v prvej z nich je hodnota premennej x nastavená na 0, v druhej na 1, atď.

Opäť sú tri možnosti, ako môže výpočet dopadnúť:

- Ak obe kópie skončia, pričom ani jedna z nich úspešne, v nasledujúcom takte procesora skončí aj pôvodný program, tiež nie úspešne.
- Ak aspoň jedna kópia úspešne skončí, v v nasledujúcom takte procesora úspešne skončí aj pôvodný program.
- Vo všetkých ostatných prípadoch pôvodný program nikdy neskončí.

Slovne môžeme tieto operácie popísať nasledovne: Príkaz **Both** robí „paralelný and“ – overí, či obe vetvy úspešne skončia. Príkaz **Some** robí „paralelný or“ – overí, či aspoň jedna z vetiev úspešne skončí.

Netrvalo dlho a Kleofáš si uvedomil, že na takomto zázračnom zariadení dokáže niektoré problémy riešiť priam až neuveriteľne rýchlo. Napríklad testovanie prvočíselnosti je skutočne ľahké.

Príklad 1

V premennej N je prirodzené číslo. Napíšte program pre paralelizátor, ktorý pre každé N skončí, pričom *úspešne* skončí práve vtedy, keď N je prvočíslo.

Riešenie. Pomocou volaní **Both** paralelne vygenerujeme všetky čísla od 2 do $N-1$ a naraz pre každé z nich overíme, či delí N. Každá vetva úspešne skončí, ak „jej“ číslo nedelí N. Aby pôvodný program úspešne skončil, musia úspešne skončiť všetky vetvy, teda žiadne z vygenerovaných čísel nesmie deliť N. Časová zložitosť takéhoto programu je $O(\log N)$.

```
{ VSTUP:  N : integer; }

var moc2, pocet_cifier : integer;
    cislo : integer;
    i,x : integer;

begin
   { osetrime okrajovy pripad }
   if ( N = 1 ) then Reject;

   { zistime, kolko ma N cifier v dvojkovej sustave }
   moc2 := 1;
   pocet_cifier := 0;
   while (moc2 <= N) do begin
      moc2 := moc2 * 2;
      inc( pocet_cifier );
   end;

   { vygenerujeme cisla od 0 do 2^pocet_cifier - 1 }
   cislo := 0;
   for i:=1 to pocet_cifier do begin
      Both(x);
      cislo := 2*cislo + x;
   end;

   { primale delitele skusat nebudeme, prehlasime za dobre }
   if (cislo <= 1) then Accept;
   { ani privelke delitele skusat nebudeme }
   if (cislo >= N) then Accept;
   { inak skusime, ci vygenerovane cislo deli N }
   if (N mod cislo <> 0) then Accept;
   Reject;
end.
```

Názorne si ukážeme, ako vyzerá výpočet paralelizátora na tomto programe pre $N = 3$ a pre $N = 6$. Kópie programu, ktoré vznikajú počas výpočtu, budeme číslovať v poradí, v akom vznikajú.

Pre $N = 3$ bude výpočet prebiehať nasledovne:

- Spustí sa kópia #1 (teda vlastne originál).
- Spočíta, že pocet_cifier = 2.
- Spustí sa for-cyklus pre i = 1.
- Kópia #1 sa zastaví, vzniknú kópie #2 a #3.
- V kópii #2 je cislo = 0, v kópii #3 je cislo = 1.
- V oboch bežiacich kópiách sa spustí for-cyklus pre i = 2.
- Kópie #2 a #3 sa zastavia, z #2 vzniknú #4 a #5, z #3 vzniknú #6 a #7.
- V kópiách #4 až #7 nadobudne premenná cislo hodnoty 0 až 3.
- Kópie #4 a #5 úspešne skončia, lebo čísla 0 a 1 nechceme testovať ako delitele.
- Kópia #2 úspešne skončí, lebo už úspešne skončili obe kópie, ktoré z nej vznikli.
- Kópia #7 úspešne skončí, lebo ani číslo 3 nechceme testovať.
- Kópia #6 úspešne skončí, lebo 2 nedelí 3.
- Kópia #3 úspešne skončí, lebo už úspešne skončili obe kópie, ktoré z nej vznikli.
- Kópia #1 (teda pôvodný program) úspešne skončí, lebo už úspešne skončili obe kópie, ktoré z nej vznikli.

Pre $N = 6$ bude výpočet prebiehať nasledovne:

- Podobne ako pri $N = 3$ sa dostaneme do situácie, kedy bežia kópie #8 až #15, premenná cislo v nich má hodoty postupne od 0 do 7.
- Kópie #8 a #9 (s primalým číslom) úspešne skončia.
- Kópia #4 (z ktorej vznikli #8 a #9) úspešne skončí.
- Kópie #14 a #15 (s priveľkým číslom) úspešne skončia.
- Kópia #7 (z ktorej vznikli #14 a #15) úspešne skončí.
- Kópie #10 až #13 skončia – a to: #12 a #13 úspešne (4 ani 5 nedelí 6), #10 a #11 neúspešne (2 aj 3 delí 6).
- Kópia #5 skončí neúspešne (obe jej „deti“ skončili neúspešne), kópia #6 skončí úspešne.
- Kópia #2 skončí neúspešne (lebo kópia #5 skončila neúspešne), kópia #3 skončí úspešne.
- Kópia #1 (teda pôvodný program) skončí neúspešne.

Príklad 2

V premenných N a K sú prirodzené čísla. Napíšte program pre paralelizátor, ktorý pre každé N skončí, pričom *úspešne* skončí práve vtedy, keď N má nejakého deliteľa z množiny $M = \{2, 3, \ldots, 2^K - 1\}$.

Riešenie. Pomocou volaní **Some** paralelne prezrieme všetky $m \in M$, stačí nám, ak ľubovoľné jedno z nich delí N.

(Iný pohľad na to isté riešenie: Pomocou volaní **Some** „uhádneme“ deliteľa $m \in M$ a overíme, či sme ho uhádli správne. Na náš program sa môžeme pozerať tak, že sa nevetví, ale každé volanie **Some** „uhádne“ a do x dosadí tú „správnu“ hodnotu. Ak teda N má v množine M deliteľa, nájdeme ho, inak skončíme s nejakým číslom, ktoré N nedelí.)

Časová zložitosť takéhoto programu je $O(K)$.

```
{ VSTUP:  N, K : integer; }

var cislo : integer;
    i,x : integer;

begin
   { paralelne skusame cisla od 0 do 2^K - 1 }
   cislo := 0;
   for i:=1 to K do begin
      Some(x);
      cislo := 2*cislo + x;
   end;

   { 0 a 1 do mnoziny M nepatria }
   if (cislo <= 1) then Reject;
   { skusime, ci vygenerovane cislo deli N }
   if (N mod cislo = 0) then Accept;
   Reject;
end.
```

Súťažná úloha

a) V premenných *ihla* a *seno* sú dva reťazce. Napíšte čo najrýchlejší program pre paralelizátor, ktorý pre každý vstup skončí, pričom *úspešne* skončí práve vtedy, ke sa reťazec *ihla* nachádza v reťazci *seno* ako (súvislý) podreťazec.

Váš program by teda mal úspešne skončiť ak napríklad:
ihla = *abcd*, *seno* = *aaabcdddaa*
ihla = *ddda*, *seno* = *aaabcdddaa*

ale nie v prípadoch:
$ihla = abcd$, $seno = aaabcEdddaa$
$ihla = jasomihla$, $seno = vtejtokopesenaihlyniet$

b) Nad poľom nezáporných celých čísel môžeme postaviť „pyramídu“. Spodný riadok pyramídy bude tvoriť samotné pole. Každý vyšší riadok bude o 1 kratší ako predchádzajúci, pričom i-ty prvok v novom riadku je rovný súčtu i-teho a $(i+1)$-ého prvku z riadku pod ním, modulo 10 000. (Čiže ak by bol súčet väčší ako 9999, necháme z neho len posledné 4 cifry.) Vrchný riadok je tvorený jedným číslom.

V premennej N máme prirodzené číslo. V poli A na pozíciách 1 až N máme N nezáporných celých čísel menších ako 10 000. V premennej V je nezáporné celé číslo menšie ako 10 000.

Napíšte čo najrýchlejší program pre paralelizátor, ktorý pre každý vstup skončí, pričom *úspešne* skončí práve vtedy, ak V je na vrchu pyramídy, utvorenej nad poľom A.

Príklad:

Vstup
$N = 4$
$A = \{6, 3, 9, 3\}$
$V = 17$

Výstup
neskončí úspešne

Vstup
$N = 4$
$A = \{1, 2, 3, 4\}$
$V = 20$

Výstup
skončí úspešne
Pyramída vyzerá nasledovne:

```
          20
       8     12
    3     5     7
 1     2     3     4
```

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY
55. ROČNÍK MATEMATICKEJ OLYMPIÁDY
Zadania 1. kola kategórie P

Vydala IUVENTA pre vnútornú potrebu Ministerstva školstva SR
Zodpovedný redaktor: M. Forišek
Sadzba programom LaTeX