MATEMATICKÁ OLYMPIÁDA NA STREDNÝCH ŠKOLÁCH

55. ročník, školský rok 2005/2006

Zadania úloh 2. kola kategórie P

Druhé kolo 55. ročníka MO kategórie P sa koná 17. januára 2006 v dopoludňajších hodinách. Na riešenie úloh máte 4 hodiny čistého času. Riešenie každého príkladu musí obsahovať (pokiaľ nie je v zadaní uvedené ináč):

- **Popis riešenia,** to znamená slovný popis použitého algoritmu, argumenty zdôvodňujúce jeho správnosť (prípadne dôkaz správnosti algoritmu), diskusiu o efektivite vášho riešenia (časová a pamäťová zložitosť). Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného zápisu algoritmu (do programu).

- **Program.** V úlohách **P-II-1**, **P-II-2** a **P-II-3** treba uviesť dostatočne podrobný zápis algoritmu, najlepšie v tvare zdrojového textu najdôležitejších častí programu v jazyku Pascal alebo C/C++. Zo zápisu môžete vynechať jednoduché operácie ako vstupy, výstupy, implementáciu jednoduchých matematických vzťahov a pod.

  Triedenie už **nie je** jednoduchá operácia, preto ak píšete program v Pascale a používate triedenie, nezabudnite stručne uviesť aj jeho algoritmus. V jazyku C/C++ môžete namiesto toho použiť knižničné funkcie na triedenie, v takom prípade stačí uviesť porovnávaciu funkciu.

  Ak píšete program v C++ a používate STL, súčasťou popisu vášho algoritmu by mal byť dostatočne podrobný popis implementácie dátových štruktúr, ktoré používate.

  V úlohe **P-II-4** je potrebné uviesť program pre paralelizátor.

Hodnotí se nielen správnosť programu, ale tiež kvalita popisu riešenia a efektivita zvoleného algoritmu.

P–II–1

V Absurdistane práve začína nový ročník futbalovej súťaže. Tento rok ju bude hrať aj slávny tím Dynamo Zbicykla. Jeho tréner už tri noci poriadne nespal, no napriek tomu ešte nemá hotový plán na túto sezónu. Dobre vie, že žiadne mužstvo nedokáže vyhrávať všetky zápasy, lebo každá výhra stojí hráčov veľa síl. Vymyslel si preto nasledujúce zjednodušenie:

Aktuálny stav jeho mužstva bude popisovať jedno celé číslo S, ktoré udáva, koľko majú hráči sily. Na začiatku sezóny (v deň číslo 0) je sila tímu nulová. Každú noc si hráči oddýchnu, a preto sa ich sila zväčší o 1. Ak chcú nejaký zápas vyhrať, musia sa snažiť – každá výhra ich stojí V sily. Môžu samozrejme aj „hrať na remízu“, čo ich stojí len R sily, prípadne zápas úplne vypustiť a prehrať ho, čo ich nestojí nič. (Ak chcú nejaký zápas vyhrať alebo remizovať, musia mať na to dosť síl, sila tímu nesmie nikdy klesnúť pod nulu.)

Tréner už pozná presný rozpis ligy, vie teda, v ktoré dni má jeho mužstvo voľno a kedy hrá nejaký zápas. Napíšte program, ktorý vypočíta, koľko najviac bodov môže jeho mužstvo v tomto ročníku ligy získať. (Ako už vo futbale istú dobu platí, za výhru sú tri body a za remízu jeden.)

Formát vstupu: Na vstupe sú celé čísla V, R (vysvetlené vyššie) a počet zápasov N. Nasleduje N celých čísel – čísla dní, v ktoré hrá naše futbalové mužstvo zápas.

Môžete predpokladať, že $N \leq 10\,000$. Čísla V, R aj všetky čísla dní sa zmestia do bežnej 32-bitovej celočíselnej premennej. Čísla dní zápasov sú uvedené v rastúcom poradí.

Formát výstupu: Vypíšte jediné celé číslo – maximálny počet bodov, ktoré vie mužstvo získať.

Príklad:

vstup	**výstup**
$V = 10$, $R = 3$, $N = 2$ dni zápasov: 3, 13	4 *Hráči stihnú nazbierať presne toľko síl, aby zvládli prvý zápas remizovať a druhý vyhrať.*

vstup	**výstup**
$V = 20$, $R = 15$, $N = 4$ dni zápasov: 23, 24, 25, 26	3 *Mužstvo dokáže vyhrať ľubovoľný jeden z týchto štyroch zápasov.*

vstup	**výstup**
$V = 30$, $R = 9$, $N = 4$ dni zápasov: 30, 32, 34, 36	4 *Nie vždy sa oplatí vyhrať, v tomto prípade je optimálne všetky 4 zápasy remizovať.*

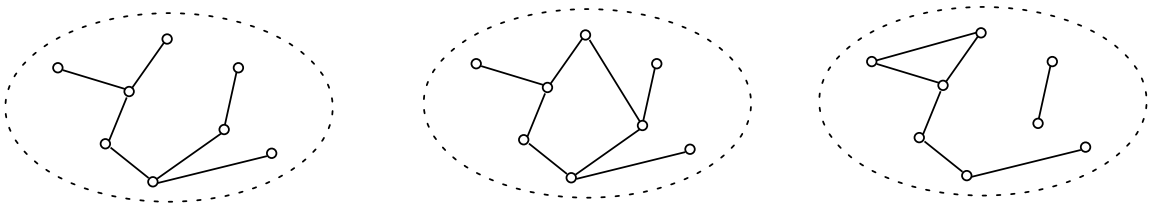
P–II–2

Strom je objekt, ktorý má nasledujúce vlastnosti:

- Obsahuje konečný počet význačných miest (tie voláme *vrcholy*, ich počet značíme N), niektoré dvojice vrcholov sú prepojené „konármi“ (tie voláme *hrany*).
- Je súvislý, teda z každého vrcholu sa viem dostať do každého iného tak, že postupne prejdem po niekoľkých hranách.
- Obsahuje práve $N - 1$ hrán.

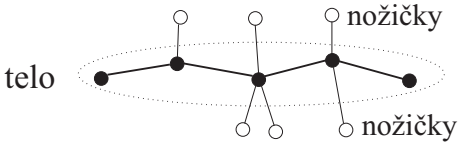
Strom si môžeme predstaviť napríklad ako súvislú cestnú sieť, ktorú tvorí N miest a práve $N - 1$ ciest medzi nimi.

Na nasledujúcom obrázku ľavý graf je strom, zvyšné dva nie sú – druhý obsahuje priveľa hrán a tretí má síce správny počet hrán, ale nie je súvislý.

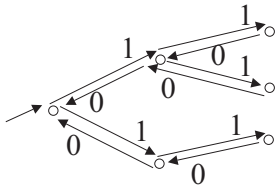


Postupnosť na seba nadväzujúcich hrán, v ktorej sa žiadna hrana neopakuje, nazveme *cesta*. Všimnite si, že v strome medzi každými dvoma vrcholmi vedie práve jedna cesta.

Húsenica je taký strom, v ktorom existuje cesta (túto cestu voláme *telo*) taká, že každý vrchol stromu je buď na tejto ceste, alebo susedí s nejakým vrcholom cesty (takýto vrchol voláme *nožička*). Príklad húsenice nájdete na obrázku vľavo.



Húsenica.



1 1 0 1 0 0 1 1 0 0

Popis stromu postupnosťou.

Spôsobov, ako zadať strom, je niekoľko. My strom popíšeme postupnosťou núl a jednotiek: Vyberieme si jeden vrchol a začneme sa z neho po strome prechádzať, pričom chceme navštíviť každý vrchol a každou hranou prejsť práve dvakrát (raz v každom smere). Počas tejto prechádzky si budeme zapisovať nuly a jednotky nasledovne: Vždy, keď prídeme do vrcholu, v ktorom sme ešte neboli, napíšeme jednotku, vždy, keď sa z vrcholu vraciame späť (hranou, ktorou sme doň prvýkrát prišli), napíšeme nulu. Rozmyslite si, že takto vieme (aspoň jedným spôsobom) popísať ľubovoľný strom a že z takéhoto popisu vieme strom jednoznačne zostrojiť.

Súťažná úloha

Zo zadanej postupnosti núl a jednotiek zostrojte strom a zistite, koľko najmenej vrcholov z neho treba odstrániť, aby sme dostali húsenicu.

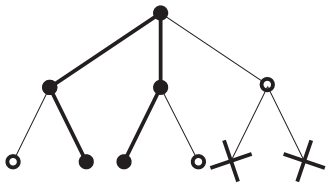
Inými slovami, nájdite v zadanom strome takú cestu, pre ktorú je množina vrcholov, ktoré na nej neležia, ani s ňou nesusedia, najmenšia možná.

Formát vstupu: Na vstupe je postupnosť núl a jednotiek reprezentujúca strom tak, ako je to popísané vyššie.

Formát výstupu: Vypíšte jediné celé číslo – minimálny počet vrcholov, ktoré je treba odstrániť z pôvodného stromu, aby sme dostali húsenicu.

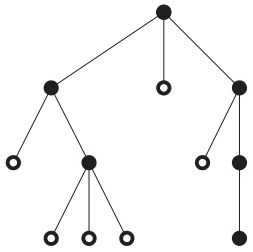
Príklad:
vstup
`110100110100110100`

výstup
`2` *(Treba odstrániť 2 vrcholy.)*



vstup
`11011010100010110110000`

výstup
`0` *(Zadaný strom už je húsenica.)*



Poznámka. V oboch príkladoch vstupu prechádzku začíname v „hornom“ vrchole stromu a ostatné vrcholy navštevujeme v poradí „zľava doprava“.

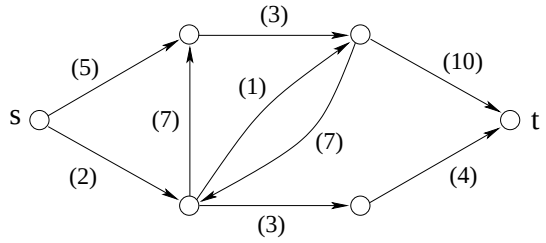
P-II-3

Predstavte si sieť na seba ponapájaných potrubí. Miesta, kde sa stretajú konce a začiatky potrubí, budeme volať uzly. Z každého uzlu môže viesť ľubovoľne veľa potrubí, podobne do každého uzlu môže ľubovoľne veľa potrubí prichádzať. Každé potrubie je upravené tak, že ním voda môže tiecť len jedným smerom. Rôzne potrubia môžu mať rôznu hrúbku, a teda nimi za sekundu môže pretiecť rôzne množstvo vody. (Maximálne množstvo vody, ktoré za sekundu môže pretiecť potrubím, voláme jeho *kapacitou*.)

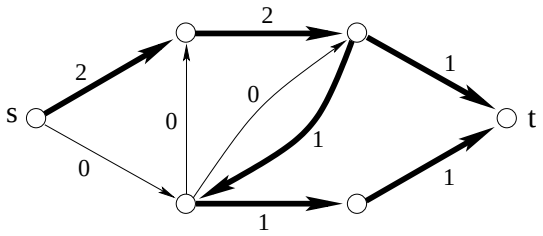
Dva uzly budú mať špeciálny význam. Jeden z nich voláme zdroj (a značíme s), druhý voláme ústie (a značíme t). Zdroj je miesto, kde do

potrubia môže pritekať voda, ústie je miesto, kde naopak voda môže odtekať. Pre jednoduchosť budeme predpokladať, že do zdroja ani z ústia žiadne potrubia nevedú.

Teraz si predstavte, že cez takúto sieť potrubí nejako pustíme vodu a pre každé potrubie si zapíšeme množstvo vody, ktoré ním za sekundu pretečie. Tomuto zoznamu čísel hovoríme *tok*. *Veľkosť* tohto toku je množstvo vody, ktoré za sekundu vytečie von ústím (alebo ekvivalentne, ktoré za sekundu pritečie zo zdroja).



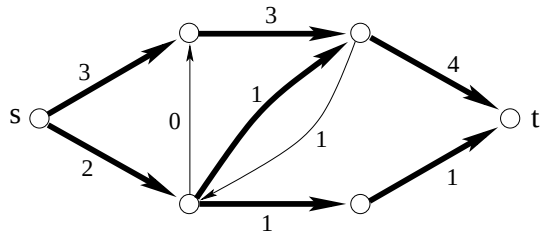
a) príklad siete potrubí



b) príklad toku veľkosti 2

Na obrázku a) je príklad siete potrubí, čísla v zátvorkách predstavujú kapacity jednotlivých potrubí. Na obrázku b) je jeden možný tok pre našu ukážkovú sieť potrubí. Hrubou čiarou sú znázornené potrubia, cez ktoré tečie nejaká voda, čísla pri potrubiach udávajú množstvo vody, ktoré za sekundu cez dané potrubie pretečie.

Maximálny tok je tok, ktorý má pre danú sieť potrubí najväčšiu možnú veľkosť. Inými slovami, maximálny tok popisuje, ako cez našu sieť potrubí „pretlačiť“ čo najväčšie množstvo vody za jednu sekundu.



c) príklad maximálneho toku

Súťažná úloha

Predstavte si, že máte k dispozícii čiernu krabičku, ktorej zadáte popis nejakej siete potrubí (spolu s ich kapacitami) a ona vám nájde hodnotu maximálneho toku. Pomocou nej vyriešte nasledovný problém:

Na vstupe máte zadané bludisko – neorientovaný graf s N lokalitami a M chodbičkami medzi nimi. Myška sa nachádza v lokalite 1. V lokalite N sa nachádza syr. Myška vie naraz uniesť maximálne 1 kúsok syra a chce preniesť čo najviac syra z lokality N do lokality 1. Nechce ale prísť dvakrát do tej istej lokality (samozrejme okrem lokalít 1 a N), lebo nechce riskovať, že si ju tam počká kocúr, ktorý ju tam po prvom prechode mohol

zaňuchať. Koľko kúskov syra vie maximálne preniesť? (Predpokladajte, že lokality 1 a N nie sú spojené chodbičkou, vtedy by samozrejme myška mohla postupne nanosiť všetok syr.)

Riešením tejto úlohy je teda program, ktorý ju rieši, pričom v ňom môžete volať funkciu `NajdiMaximalnyTok(...)`, ktorej zadáte ako parametre popis siete potrubí (počet uzlov, počet potrubí, pre každé potrubie jeho začiatok, koniec a kapacitu, a navyše informáciu, ktorý uzol je zdroj a ktorý je ústie) a ona vám vráti hodnotu maximálneho toku v ňom. Túto funkciu nemusíte implementovať, presný formát parametrov si zvoľte tak, ako vám bude vyhovovať.

Príklad:

vstup	**výstup**
$N = 8$, $M = 9$	1
$1-2$, $1-5$, $1-7$, $2-3$, $2-4$, $3-8$, $4-8$, $5-6$, $6-8$, $7-8$	*(Myška môže ísť* $1-2-3-8$ *po syr,* $8-7-1$ *späť. Mohla by ešte ísť* $1-5-6-8$ *po syr, ale naspäť sa už nedostane.)*

Študijný text – toky v grafoch

V tejto časti zadania uvádzame formálnejšie definície vyššie spomenutých pojmov. Pokiaľ ti je všetko jasné, tento text čítať nemusíš, použi ho len v prípade nejasností v neformálnom popise.

Začneme niekoľkými definíciami: Graf je usporiadaná dvojica (V, E), kde V je konečná množina jeho vrcholov a E je konečná množina jeho hrán. Počet vrcholov označme N a počet hrán M, hrany označme e_1 až e_M. Každá hrana e_i spája práve dva vrcholy grafu a_i a b_i. Ak sa bude dať prechádzať po hrane e_i iba jedným smerom (t.j. vieme po nej prejsť z vrcholu a_i do b_i, ale nie opačne), budeme ju volať orientovaná hrana, inak ju budeme volať neorientovaná hrana. Graf nazveme orientovaný, resp. neorientovaný, ak sú všetky hrany v ňom orientované, resp. neorientované.

V grafe budeme mať dva špeciálne vrcholy. Jeden z nich nazveme *zdroj* (značíme s) a ten druhý *ústie* (značíme t). Ďalej budeme predpokladať, že každá hrana má svoju kapacitu $c_i \geq 0$. V orientovanom grafe hrana z a_i do b_i môže mať inú kapacitu ako hrana z b_i do a_i, resp. niektorá z nich ani nemusí existovať.

Funkciu f, ktorá priraďuje každej hrane množstvo vody, ktoré ňou preteká, nazveme tok, ak spĺňa nasledovné podmienky:

- $f(e_i) \in \mathbb{Z}$ pre $1 \leq i \leq M$. Teda cez každú hranu môže tiecť len celočíselné množstvo vody.

- $0 \leq f(e_i) \leq c_i$ pre $1 \leq i \leq M$. Teda cez každú hranu nemôže tiecť viac vody ako je jej kapacita, ani menej ako 0.
- Nech a je vrchol rôzny od zdroja a ústia. Nech hrany vedúce z vrcholu a majú čísla $v_1, \ldots, v_k$. Podobne, nech hrany vedúce do vrcholu a majú čísla $p_1, \ldots, p_l$. Potom platí: $\sum f(e_{v_i}) = \sum f(e_{p_i})$. T.j. vo „vnútorných“ vrcholoch sa voda nemôže hromadiť.

Teda tok je taká funkcia f, ktorá nám určuje, ktorým potrubím tečie koľko vody. Predpokladajme, že zdroj nemá žiadne prichádzajúce hrany a ústie nemá žiadne odchádzajúce hrany. Potom môžeme hodnotu toku f definovať ako množstvo vody, ktoré odteká zo zdroja. *Maximálny tok* je tok s najväčšou možnou hodnotou pre daný graf.

P–II–4

V krajine je niekoľko miest, každé z nich má priradené nejaké prirodzené číslo (ktoré sa zmestí do bežnej celočíselnej premennej). Rôzne mestá majú rôzne čísla, ale inak číslovanie miest nie je nijako systematické.

Medzi niektorými dvojicami miest sú postavené cesty. Dokopy je M takýchto ciest. Všetky cesty sú jednosmerné. Všetky križovatky sú mimoúrovňové, t.j. keď sa vyberieme nejakou cestou, musíme po nej zájsť až do mesta, kde končí. Môžete predpokladať, že v každom meste aspoň jedna cesta začína alebo končí.

V poli `C[0..M-1][0..1]` sú zadané jednotlivé cesty (i-ta cesta spája mestá s číslami `C[i-1][0]` a `C[i-1][1]`).

Súťažná úloha

Napíšte čo najrýchlejší program pre paralelizátor, ktorý pre každý prípustný vstup skončí, pričom úspešne skončí práve vtedy, ak je cestná sieť silne súvislá – teda ak sa z ľubovoľného mesta vieme dostať po cestách do ľubovoľného iného mesta.

Poznámka. Ide to v lepšom ako lineárnom čase. Samozrejme, ak sa vám takéto riešenie nájsť nepodarí, môžete niekoľko bodov získať aj za pomalšie riešenia.

Príklad:

Vstup	**Výstup**
$M = 5$ cesty: $1 \to 2$, $2 \to 3$, $3 \to 1$, $47 \to 1$, $2 \to 47$	skončí úspešne *(Po prvých troch cestách vieme voľne prechádzať medzi mestami 1, 2 a 3, vďaka zvyšným dvom sa vieme dostať aj do mesta 47 a z neho zase preč.)*

Vstup	**Výstup**
$M = 3$ cesty: $123456 \to 234567$, $23 \to 47$, $345678 \to 234567$	neskončí úspešne *(Nevieme sa dostať napr. z mesta 47 do mesta 123456.)*

Študijný text – paralelizáror

(Tento študijný text je identický s textom zverejneným v zadaniach domáceho kola.)

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 ani privelke delitele skusat nebudeme }
   if (cislo <= 1) then Accept;
   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.
```

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY

55. ROČNÍK MATEMATICKEJ OLYMPIÁDY

Zadania 2. kola kategórie P

Vydala IUVENTA s finančnou podporou Ministerstva školstva SR
Náklad: 320 výtlačkov
Zodpovedný redaktor: M. Forišek
Sadzba programom LaTeX