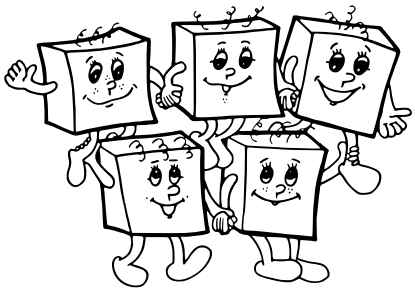
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH

http://oi.sk/



dvadsiaty ôsmy ročník
školský rok 2012/2013

zadania krajského kola
kategória A

Priebeh krajského kola

Krajské kolo 28. ročníka Olympiády v informatike, kategória A, sa koná 22. januára 2013 v dopoludňajších hodinách. Na riešenie úloh majú súťažiaci **4 hodiny čistého času**. Rôzne úlohy riešia súťažiaci na samostatné listy papiera. Akékoľvek pomôcky okrem písacích potrieb (napr. knihy, výpisy programov, kalkulačky) sú zakázané.

Čo má obsahovať riešenie úlohy?

- Slovne popíšte algoritmus.
  Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného algoritmu/programu.
- Zdôvodnite správnosť vášho algoritmu.
- Uveďte a zdôvodnite jeho časovú a pamäťovú zložitosť.
- Podrobne uveďte dôležité časti algoritmu, ideálne vo forme programu v Pascale alebo C/C++.
- V prípade, že používate vo svojom programovacom jazyku knižnice, ktoré obsahujú implementované dátové štruktúry a algoritmy (napr. STL pre C++), v popise algoritmu stručne vysvetlite, ako by ste napísali program s rovnakou časovou zložitosťou bez použitia knižnice.

Hodnotenie riešení

Za každú úlohu môžete získať od 0 do 10 bodov.

Pokiaľ nie je v zadaní povedané ináč, najdôležitejšie dve kritériá hodnotenia sú v prvom rade **správnosť** a v druhom rade **efektívnosť** navrhnutého algoritmu. Na výslednom počte bodov sa môže prejaviť aj kvalita popisu riešenia a zdôvodnenie tvrdení o jeho správnosti a efektívnosti.

Efektívnosť algoritmu posudzujeme vypočítaním jeho časovej zložitosti – funkcie, ktorá hovorí, ako dlho vykonanie algoritmu trvá v závislosti od veľkosti vstupných parametrov. Nezávisí pri tom na konštantných faktoroch, len na rádovej rýchlosti rastu tejto funkcie.

V zadaní úloh uvádzame časť „Hodnotenie“, v ktorej nájdete približné limity na veľkosť vstupných údajov. Pod pojmom „efektívne vyriešiť“ chápeme to, že váš program spustený na modernom počítači by mal dať odpoveď nanajvýš do niekoľkých sekúnd.

Údaje z tejto časti zadania by mali slúžiť hlavne na to, aby ste o riešení, ktoré vymyslíte, vedeli približne povedať, koľko bodov zaň dostanete.

A-II-1 Výkrm

Kráľ Hurim a kráľovná Totňa sa vydali na cestu Lineárnym kráľovstvom.

Verné svojmu menu, Lineárne kráľovstvo je tvorené jedinou cestou. Na tej je n miest, ktoré sú postupne (v poradí, v akom na ceste ležia) očíslované od 1 po n. Hurim a Totňa pôjdu krajinou v tom smere, v ktorom sú číslované mestá, pričom sa postupne zastavia v práve ℓ z nich.

V každom meste varia práve jednu miestnu špecialitu. Môže sa stať, že viaceré mestá majú tú istú špecialitu. V celom kráľovstve je rôznych miestnych špecialít presne m a sú očíslované od 1 po m.

Totňa vie, že Hurim má mlsný jazýček, chcela by teda vybrať zastávky tak, aby sa mu podávané pokrmy páčili. Kráľovskí radcovia pracovali dňom i nocou, až nakoniec prišli s k návrhmi Hurimovho stravovania. Každý návrh je postupnosť ℓ špecialít, ktoré keď Hurim v danom poradí zje, bude na vrchole blaha.

Súťažná úloha

Totňa sa síce potešila, že má na výber hneď k rôznych návrhov, no optimizmus ju rýchlo prešiel. Čo ak sa niektoré (alebo nebodaj všetky) z nich nedajú v jej kráľovstve realizovať? A keďže miest aj plánov je strašne veľa, potrebuje Totňa vašu pomoc.

Napíšte program, ktorý o každom pláne zistí, či existuje taká postupnosť ℓ zastávok v mestách, pri ktorej Hurim zje presne predpísanú postupnosť jedál.

Upozorňujeme, že sa kráľovský pár nesmie pri svojej ceste vracať späť (do už prejdeného mesta) ani sa dvakrát po sebe zastaviť v tom istom meste.

Formát vstupu

V prvom riadku vstupu je počet miest n, počet špecialít m (pričom $m \leq n$), počet návrhov stravovania k a počet zastávok ℓ.

V druhom riadku je n celých čísel $s_1, \ldots, s_n$. Číslo s_i ($1 \leq s_i \leq m$) je číslo špeciality podávanej v meste i.

Nasleduje k riadkov, každý z nich popisuje jeden návrh Hurimovho stravovania. Presnejšie, každý z týchto riadkov obsahuje postupnosť presne ℓ celých čísel: čísla špecialít, ktoré by mal Hurim počas výletu v danom poradí jesť.

Formát výstupu

Pre každý návrh vypíšte jeden riadok s textom „`ano`“ ak sa daný návrh dá realizovať, resp. „`nie`“ ak sa realizovať nedá.

Hodnotenie

Aspoň 9 bodov môže získať každé riešenie, ktoré efektívne vyrieši každý vstup s $n \leq 100\,000$ a $k \cdot \ell \leq 100\,000$.
(O udelení 10 alebo 9 bodov rozhoduje optimálnosť asymptotickej časovej zložitosti riešenia.)
Až 7 bodov môžete získať za riešenie, ktoré navyše predpokladá aj $m \leq 10$.
Až 5 bodov môžete získať za riešenie, ktoré efektívne vyrieši ľubovoľný vstup v ktorom $k \cdot (n+\ell) \leq 1\,000\,000$.
Za ľubovoľné korektné riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 20$, môžete získať 3 body.

Príklad

vstup

```
10 4 4 3
1 2 1 3 4 1 2 1 3 4
1 4 2
4 3 2
1 1 1
4 4 4
```

výstup

```
ano
nie
ano
nie
```

Prvý návrh sa dá realizovať napr. zastávkou v treťom, piatom a následne siedmom meste.
Tretí návrh sa dá realizovať napr. zastávkou v prvom, šiestom a ôsmom meste.

A-II-2 Vyvážené jablone

Na okraji mestečka Ponyville (viď seriál My Little Pony: Friendship is Magic) žije rodina poníkov, ktorá sa živí pestovaním jabĺk. Darí sa im, stromy v ich sade sú jablkami úplne obsypané. Applejack si ale nedávno všimla, že sa im niektoré stromy ohýbajú. Totiž hen rastie jabĺk veľa, tam zase málo, a tak jabloň nie je vyvážená.

Všetky jablone v sade sú *binárne*. Binárna jabloň je strom veľmi špeciálneho tvaru. Skladá sa z uzlov, vetiev, listov a jabĺk. Najspodnejší z uzlov nazývame *koreň*. Z každého uzlu vedú dohora práve dve vetvy. Na hornom konci každej vetvy je buď ďalší uzol, alebo list. Celý strom drží pokope (od koreňa sa po vetvách vieme dostať ku každému inému uzlu) a jeho vetvy sa nikde nespájajú (takže sa po nich nikde nedá chodiť do kolečka).

Jablká rastú len pri listoch stromu. Pri každom liste môže rásť ľubovoľne veľa jabĺk (aj nula).

Pre ľubovoľnú vetvu v strome si môžeme všimnúť, že nad touto vetvou leží nejaká ucelená časť stromu. V tejto časti stromu sa môžu nachádzať aj nejaké jablká. Ich celkový počet nazveme *záťažou* dotyčnej vetvy.

Uzol stromu je *vyvážený*, ak platí, že obe vetvy, ktoré z neho vedú dohora, majú presne rovnakú záťaž.

Celý strom je *vyvážený*, ak sú vyvážené úplne všetky jeho uzly. Ak náhodou strom nie je vyvážený, jediný spôsob, ktorým to smieme zmeniť, je ten, že odtrhneme niektoré vhodne zvolené jablká.

Súťažná úloha

Napíšte program, ktorý zistí, koľko *najmenej* jabĺk treba z binárnej jablone otrhať, aby bola vyvážená.

Formát vstupu a výstupu

Môžete predpokladať, že vstup je korektný, teda že naozaj popisuje binárnu jabloň. Pre jednoduchosť budeme uzly a listy jablone označovať spoločným názvom *vrcholy*. V prvom riadku vstupu je číslo n, udávajúce celkový počet vrcholov na našej jabloni. Vrcholy si očíslujeme od 1 po n tak, aby vrchol s číslom 1 bol koreň.

Nasledujúcich n riadkov popisuje jednotlivé vrcholy. Ak je vrchol i uzol, tak je v i-tom z týchto riadkov text „`U` x_i y_i“, kde x_i a y_i sú čísla vrcholov, ktorými končia vetve idúce z uzlu i dohora. Ak je vrchol i list, tak je v i-tom z týchto riadkov text „`L` j_i“, kde j_i je počet jabĺk rastúcich pri tomto liste. Pri písaní programu môžete predpokladať, že sa vám celkový počet jabĺk na strome zmestí do bežnej celočíselnej premennej.

Vypíšte jeden riadok a v ňom jedno celé číslo: najmenší celkový počet jabĺk, ktoré keď z vhodných listov odtrhneme, dostaneme vyvážený strom. Všimnite si, že odpoveď vždy existuje – v najhoršom vždy môžeme otrhať úplne všetky jablká, lebo strom bez jabĺk je zjavne vyvážený.

Hodnotenie

Plných 10 bodov dostanete za riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 1\,000\,000$.

Až 6 bodov môžete získať za riešenie, ktoré je efektívne pre $n \leq 1000$ a najviac 1000 jabĺk na strome.

Aspoň 3 body budú za ľubovoľné riešenie, ktoré je efektívne pre $n \leq 1000$ a najviac 20 jabĺk na strome.

Príklady

vstup

```
3
U 2 3
L 14
L 17
```

výstup

```
3
```

Odtrhneme tri jablká vo vrchole číslo 3. Tým bude vo vrcholoch 2 a 3 po 14 jabĺk a jabloň bude vyvážená.

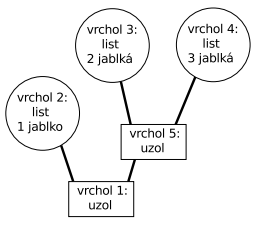
vstup

```
5
U 2 5
L 1
L 2
L 3
U 3 4
```

V tomto prípade musíme otrhať všetky jablká.

output

```
6
```



A-II-3 Zaokrúhľovanie

Janko chcel Peťku prekvapiť, a tak jej na Vianoce daroval krásnu tabuľku plnú reálnych čísel. Lenže čo čert nechcel, Peťke sa páčia len tabuľky plné celých čísel, a také veru v tabuľke nebolo ani jedno. Teraz si to teda musí Janko u nej vyžehliť. Teda pardón, musí si to zaokrúhliť.

Našťastie si Janko všimol, že v Peťkinej tabuľke je súčtom každého riadku aj každého stĺpca celé číslo. Tie sa samozrejme Peťke páčia, a tak ich Janko nesmie zaokrúhľovaním zmeniť.

Každé číslo môže pritom zaokrúhliť buď nadol alebo nahor na najbližšie celé, a to bez ohľadu na hodnotu jeho desatinnej časti. Teda napr. z čísla 6.017 môže vyrobiť buď číslo 6, alebo číslo 7.

Súťažná úloha

Na vstupe je obdĺžniková tabuľka tvorená $r \times s$ kladnými reálnymi necelými číslami. Súčet čísel v každom riadku aj stĺpci tejto matice je celé číslo.

Váš program má každé číslo zaokrúhliť nahor alebo nadol tak, aby všetky súčty riadkov a stĺpcov zostali nezmenené, prípadne zistiť, že takýto spôsob zaokrúhlenia neexistuje.

Pri písaní programu môžete predpokladať, že váš programovací jazyk vie dokonale presne ukladať, sčítať, odčítať a porovnávať spracúvané reálne čísla.

Formát vstupu

V prvom riadku je počet riadkov tabuľky r a počet stĺpcov tabuľky s. Nasleduje r riadkov a v každom s kladných reálnych čísel, z ktorých žiadne nie je celé.

Formát výstupu

Ak úloha nemá riešenie, vypíšte jeden riadok s textom „`NEDA SA`“.

V opačnom prípade vypíšte r riadkov a v každom s znakov. Znak „`H`“ znamená, že príslušné číslo máme zaokrúhliť nahor, znak „`D`“ zase zaokrúhlenie nadol.

Hodnotenie

Plných 10 bodov dostanete za riešenie, ktoré dokáže efektívne vyriešiť vstupy, v ktorých $r \cdot s \leq 10^6$. Až 6 bodov môžete získať za riešenie, ktoré dokáže efektívne vyriešiť vstupy, v ktorých $r \cdot s \leq 1000$. Nanajvýš 3 body sú za riešenie, ktoré zvládne vyriešiť vstupy, v ktorých $r \cdot s \leq 20$.

Príklad

vstup

```
3 4
5.31 2.39 7.13 0.17
3.33 4.43 1.92 2.32
9.36 1.18 6.95 4.51
```

Súčty riadkov sú 15, 12 a 22.
Súčty stĺpcov sú 18, 8, 16 a 7.

output

```
DDHD
HHDD
DDHH
```

Tabuľka po zaokrúhlení:

```
5 2 8 0
4 5 1 2
9 1 7 5
```

A-II-4 Log-space výpočty

Študijný text k tejto úlohe je uvedený nižšie. Je identický so študijným textom z domáceho kola.
*Pripomíname, že pri riešení súťažných úloh nám **nezáleží** na časovej, ale iba na pamäťovej zložitosti programov.*

Súťažná úloha 4a (4 body)

Napíšte log-space program, ktorý vynásobí dve veľké čísla zadané ako postupnosti cifier.

Presnejšie, vstupom je číslo n a dve n-ciferné čísla α a β zadané ako polia $A[0..n-1]$ a $B[0..n-1]$. Číslo $A[i]$ je cifra rádu 10^i v zápise čísla α v desiatkovej sústave. Podobne sú $B[i]$ jednotlivé cifry čísla β. Platí teda:

$$\alpha = A[0] \cdot 10^0 \ + \ A[1] \cdot 10^1 \ + \ \cdots \ + \ A[n-1] \cdot 10^{n-1}$$
$$\beta = B[0] \cdot 10^0 \ + \ B[1] \cdot 10^1 \ + \ \cdots \ + \ B[n-1] \cdot 10^{n-1}$$

Pre všetky prvky polí A a B platí $0 \leq A[i], B[i] \leq 9$, pričom $A[n-1] \neq 0$ a $B[n-1] \neq 0$.

Výstup algoritmu, teda číslo $\gamma = \alpha \cdot \beta$, uložte rovnakým spôsobom po cifrách do poľa $C[0..2n-1]$. Ak je výsledok násobenia menší ako 10^{2n-1}, musí byť $C[2n-1] = 0$.

Príklad: Nech $A = (7, 3, 1)$, teda $A[0] = 7$, $A[1] = 3$ a $A[2] = 1$. Nech $B = (3, 2, 1)$. Tieto dve polia reprezentujú čísla $\alpha = 137$ a $\beta = 123$. Ich súčinom je číslo $\gamma = \alpha \cdot \beta = 16851$. Po skončení programu musí teda platiť $C = (1, 5, 8, 6, 1, 0)$.

Súťažná úloha 4b (6 bodov)

Daný je neorientovaný graf bez cyklov, tzv. *les*. Komponenty súvislosti tohto grafu sú *stromy*. Napíšte log-space program, ktorý na vstupe dostane popis lesa a dva jeho vrcholy a zistí, či dotyčné dva vrcholy ležia na tom istom strome.

Na vstupe dostanete štyri celé čísla. Dve z nich sú počet vrcholov grafu n a počet hrán grafu m. Vrcholy grafu sú označené číslami od 1 do n. Ďalšie dve čísla na vstupe sú čísla u a v dvoch rôznych vrcholov ($1 \leq u < v \leq n$).

Na vstupe tiež dostanete polia $A[1..m]$ a $B[1..m]$. Pre každé i platí, že vrcholy s číslami $A[i]$ a $B[i]$ sú v našom grafe spojené hranou. Môžete predpokladať, že graf na vstupe je les. Platí teda $0 \leq m \leq n-1$ a navyše hrany nášho grafu netvoria žiadny cyklus.

Výstupom programu je celočíselná premenná *spolu*. Do nej priraďte 1, ak vrcholy u a v ležia v tom istom komponente súvislosti, resp. 0, ak nie. Inými slovami, *spolu* $= 1$ znamená, že sa z vrcholu u dá dostať do vrcholu v tak, že postupne prejdeme po niekoľkých hranách.

Príklad: Nech $n = 5$, $m = 3$, $A = (1, 4, 4)$ a $B = (2, 3, 5)$. Máme teda graf s 5 vrcholmi a 3 hranami: $1-2$, $4-3$ a $4-5$.

Ak by sme vzali $u = 3$ a $v = 5$, správnou odpoveďou je *spolu* $= 1$, keďže z vrcholu 3 sa dá dostať do vrcholu 5 cez vrchol 4. Naopak, pre $u = 3$ a $v = 1$ má byť *spolu* $= 0$.

Študijný text

Ak poznáme viac algoritmov, ktoré riešia tú istú úlohu, väčšinou za lepší považujeme ten, ktorý má menšiu časovú zložitosť. Pamäťovú zložitosť zväčša používame len ako dodatočné kritérium (s výnimkou situácií, kedy sú pamäťové nároky algoritmu absurdne vysoké). V úlohách, ku ktorým patrí tento študijný text, bude situácia presne opačná: zaujímať nás bude takmer výlučne pamäťová zložitosť programu. Tá bude musieť byť veľmi malá.

Presnejšie, budeme písať *log-space programy*. Pôjde o obyčajné programy vo vašom obľúbenom bežnom programovacom jazyku, len budú navyše musieť spĺňať nasledujúce obmedzenia:

- Každá použitá premenná musí byť jednoduchého *celočíselného* typu (napr. `int` v C++ či `longint` v Pascale).
- U celočíselných premenných nám nebude záležať na presnom rozsahu, nemusí vás teda napr. trápiť, či sú 32-bitové alebo 64-bitové. Namiesto toho si povolený rozsah hodnôt, ktoré sa do premenných zmestia, definujeme nasledovne: Nech n je veľkosť vstupu (teda napr. počet čísel na vstupe). Potom do premenných môžeme ukladať len hodnoty ktoré sú *polynomiálne veľké* v závislosti od n.

  Môžeme mať teda napr. premennú, ktorá bude nadobúdať hodnoty $-n \dots n$, hodnoty $-3n^5 \dots 3n^5$, čí len hodnoty $-4 \dots 7$. Nesmieme však už mať premennú nadobúdajúcu napr. hodnoty $0 \dots 2^n$.
- Pre pohodlie budeme aj typy `char` a `boolean` (resp. `bool` v C++) považovať za celočíselné, a teda povolené.
- Žiadne iné typy premenných (teda napr. ani polia alebo ukazovatele) nie sú povolené.
- Výnimku z predchádzajúcich pravidiel tvoria vstup a výstup. Vstup programu bude dostupný v nejakých špeciálnych premenných (zväčša to budú polia), ktoré môže váš program *len čítať*. A podobne výstup bude váš program ukladať do iných špeciálnych premenných, do ktorých smie *len zapisovať*.

  (Špeciálne upozorňujeme, že nesmiete výstupnú premennú zväčšiť o danú hodnotu. Teda na ňu nesmiete napr. v Pascale použiť príkaz `inc`, v C++ operátor `++` či `+=`. Všetky tieto zmeny totiž vyžadujú nie len zápis novej hodnoty, ale najskôr aj prečítanie starej – lenže výstupnú premennú čítať nesmiete.)

  Čísla na vstupe vždy budú mať veľkosť polynomiálnu od veľkosti vstupu, a teda každé z nich sa zmestí do pracovnej premennej.
- Vaše programy nesmú používať rekurziu.

Príklad 1. Ukážeme si log-space program, ktorý nájde maximum v poli čísel.

Listing programu (Pascal)

```
var n: integer;                  { vstupná premenná: počet čísel }
    A: array [1..n] of integer;  { vstupná premenná: pole čísel }
    m: integer;                  { výstupná premenná: pozícia maxima }
    i, j: integer;               { pracovné premenné }
begin
    j := 1;
    for i := 2 to n do
        if A[i] > A[j] then j := i;
    m := j;
end;
```

Jediné dve premenné sú `i` a `j` a zjavne nadobúdajú len hodnoty z rozsahu $1 \dots n$. Tým sú teda splnené všetky potrebné podmienky a teda skutočne ide o log-space program.

Príklad 2. Nasledujúci log-space program nájde v poli čísel to, ktoré sa tam vyskytuje najčastejšie.

Listing programu (Pascal)

```
var n: integer;                  { vstupná premenná: počet čísel }
    A: array [1..n] of integer;  { vstupná premenná: pole čísel }
    m: integer;                  { výstupná premenná: jedna pozícia najčastejšieho }
    i, j, c, cmax: integer;      { pracovné premenné }
begin
    cmax := 0;
    for i := 1 to n do begin
        { spočítame, koľkokrát sa vyskytuje A[i] }
        c := 0;
        for j := 1 to n do
            if A[j] = A[i] then c := c+1;
        { je to viac ako doterajšie maximum? }
        if c > cmax then begin
            cmax := c;
            m := c;
        end;
    end;
end;
```

Opäť si ľahko rozmyslíme, že hodnoty pracovných premenných nikdy neprekročia n. Taktiež všetky ostatné požiadavky na log-space program sú dodržané.

Prečo práve takéto programy?

Isto ste si už položili otázku, prečo takéto programy nazývame log-space a načo je vlastne dobré sa nimi zaoberať.

Najpresnejší spôsob merania pamäťovej zložitosti daného programu dostaneme vtedy, keď zistíme počet *bitov pamäte*, ktorú potrebuje v závislosti od veľkosti vstupu.

Existuje síce pár veľmi jednoduchých problémov, ktoré vieme riešiť napríklad len s troma bitmi pamäte, veľa ich ale nie je a nebudú nás príliš zaujímať. My sa sústredíme na programy, ktoré vstupné dáta dostanú v nejakom n-prvkovom poli. Na to, aby sme vôbec mohli k prvkom takéhoto poľa rozumne pristupovať, potrebujeme do neho vedieť *indexovať*. A na to treba mať premennú, ktorá môže nadobudnúť n rôznych hodnôt.

Jeden bit pamäte má dva rôzne stavy: 0 alebo 1. Ak máme k-bitovú premennú, tá môže nadobúdať 2^k rôznych stavov – nezávisle pre každý z k bitov máme dve možnosti. Tieto stavy si my potom rôzne interpretujeme: raz ako znaky, inokedy ako čísla, a podobne. Napríklad taká 8-bitová premenná môže mať 256 rôznych stavov. Niekedy tieto stavy môžeme považovať za čísla od 0 do 255, inokedy za čísla od -100 do 155, a ešte inokedy za znaky v 8-bitovom ASCII kódovaní.

A tú istú úvahu môžeme spraviť aj opačne. Ak potrebujeme premennú, ktorá vie mať n rôznych hodnôt, potrebujeme, aby pre jej počet bitov k platila nerovnosť $2^k \geq n$. Inými slovami, najmenšie vyhovujúce k je $\lceil \log_2 n \rceil$ (t.j. horná celá časť dvojkového logaritmu čísla n).

Dostávame teda nasledovný záver: na prácu s n-prvkovým poľom určite treba aspoň rádovo $\log_2 n$ bitov pamäte. Toto je teda v istom zmysle najmenšia prakticky zaujímavá pamäťová zložitosť programov.

No a práve takúto (asymptotickú) pamäťovú zložitosť budú mať aj všetky programy, ktoré budete písať vo svojich riešeniach. Totiž naše obmedzenie na počet a veľkosť premenných, ktoré smiete používať, zaručuje, že celkový počet bitov pamäte, ktoré použijete, bude nanajvýš rádovo logaritmický od n. Odtiaľ teda pochádza názov „log-space“ – sú to programy, ktoré používajú $O(\log n)$ bitov pamäte.

(Príklad: Ak použijete 3 premenné, z ktorých každá môže nadobúdať hodnoty od 1 po n^5, bude váš program používať $3\lceil \log_2 n^5 \rceil \approx 15 \log_2 n$ bitov pamäte.)

O zákaze používania polí a rekurzie

Teoreticky by sa do logaritmického počtu bitov pamäte nejaké maličké polia zmestili – ale chcelo by to, aby súčet veľkostí všetkých prvkov bol nanajvýš logaritmický. Teda napríklad by sme mohli mať pole obsahujúce $2 \log n$ čísel z rozsahu 0 až 7. Alebo pole obsahujúce 3 čísla z rozsahu 1 až n^2. Nič z toho vám zrejme pri súťažných úlohách nepomôže, preto sme pre jednoduchosť polia zakázali úplne. (Namiesto poľa obsahujúceho 3 čísla predsa vždy môžete použiť 3 vhodne pomenované premenné.)

Druhou konštrukciou, ktorú sme museli zakázať, je rekurzia. Treba si totiž uvedomiť, že pamäť počas behu programu nepoužívame len na premenné. Vždy, keď v programe zavoláme nejakú funkciu, musí sa vyrobiť (na tzv. zásobníku) záznam, kde si okrem iného program zapamätá správnu návratovú adresu a tiež hodnoty parametrov, s ktorými sme danú funkciu zavolali. Taktiež lokálne premenné pre danú funkciu niekam treba uložiť. A tu práve rekurzia začína robiť problémy.

Totiž ak máme program pozostávajúci z viacerých funkcií, ale bez použitia rekurzie (t.j. každá funkcia volá len tie, ktoré boli v programe uvedené pred ňou), vždy ho vieme prerobiť („dosadením“ celého tela funkcie namiesto každého jej volania) na program, ktorý počíta to isté, ale už nepoužíva funkčné volania. Môžete si rozmyslieť, že sa pri tejto zmene nijak výrazne nezmenia pamäťové nároky programu.

Akonáhle však povolíme rekurziu (funkciu, ktorá volá sama seba, prípadne viacero funkcií, ktoré sa vedia volať navzájom), mohlo by sa stať, že pri behu programu nám bude postupne vznikať viac a viac lokálnych premenných a celková pamäť ľahko narastie nad logaritmickú. Preto radšej rekurziu úplne zakazujeme. Rovnako ako pri poliach, aj tu by malo platiť, že vám toto obmedzenie nijak zásadne nesťaží riešenie súťažných úloh.

DVADSIATY ÔSMY ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Ján Hozza
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2012