MATEMATICKÁ OLYMPIÁDA NA STREDNÝCH ŠKOLÁCH

52. ROČNÍK, školský rok 2002/2003

Zadania úloh 2. kola kategórie P

Druhé kolo 52. ročníka MO kategórie P sa koná 7. januára 2003 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. Zo zápisu môžete vynechať jednoduché operácie ako vstupy, výstupy, implementáciu jednoduchých matematických vzťahov a pod. Na druhej strane triedenie **nie je** jednoduchá operácia, preto ak používate triedenie, nezabudnite uviesť aj jeho algoritmus. (V jazyku C máte k dispozícii funkciu `qsort`, ak ju chcete použiť, stačí uviesť porovnávaciu funkciu.) V úlohe **P-II-4** napíšte príslušnú reverzibilnú procedúru.

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

P-II-1

Spoločnosť pre výskum mimozemského života zistila, že náš vesmír v minulosti obývala mimoriadne vyspelá civilizácia, ktorú naši vedci nazvali Tvorcovia hviezd. Ich technická vyspelosť im totiž umožňovala tvoriť nové hviezdy. Tvorcovia hviezd mali silne rozvinutý zmysel pre symetriu, preto nové hviezdy stavali stredovo súmerne okolo bodu, kde kedysi stála ich rodná planéta, kým sa nezrazila s kométou. Vedci, ktorí s touto teóriou prišli, by jej radi dodali trochu vierohodnosti. Zistili, ktoré hviezdy podľa nich stvorila táto mimozemská civilizácia. Teraz by potrebovali overiť, či tieto hviezdy naozaj ležia stredovo súmerne okolo nejakého bodu. (Skutočnú pozíciu, kde sa kedysi nachádzala ich domáca planéta, nepoznajú.) Keďže hviezd je vcelku dosť, rozhodli sa vedci použiť výpočtovú techniku.

Súťažná úloha

Vašou úlohou je napísať pre nich program, ktorý dostane na vstupe počet hviezd N a súradnice týchto N hviezd. Súradnice hviezdy i sú tri celé čísla x_i, y_i, z_i. Ak sú hviezdy rozmiestnené stredovo súmerne okolo nejakého bodu, mal by váš program vypísať súradnice tohto bodu. Ak takýto bod neexistuje, váš program by mal o tom podať správu.

Hovoríme, že hviezdy sú rozmiestnené stredovo súmerne okolo bodu S, ak pre každú hviezdu H existuje hviezda H' taká, že S je stredom úsečky HH'. Niektorá hviezda môže ležať aj priamo v bode S.

Pre účely tejto úlohy na chvíľu zabudnite na fyziku a predpokladajte, že hviezdy sa nepohybujú `:-)`.

Príklad:

Vstup
$N = 6$
$[0, 1, -1], [2, 0, 1], [4, 0, 3],$
$[0, 4, 1], [4, 3, 5], [2, 4, 3]$

Výstup
$[2, 2, 2]$
(Zodpovedajú si dvojice hviezd 1-5, 2-6, 3-4.)

Vstup
$N = 4$
$[0, 0, 0], [5, 0, 0], [2, 1, 0],$
$[2, -1, 0]$

Výstup
`STRED NEEXISTUJE`

P-II-2

Knihovníčka Milka potrebuje objednať ďalšiu skriňu s poličkami do svojej knižnice, nevie však sama spočítať jej optimálne rozmery. Keďže ste jej minule pomohli, rovno sa obrátila na vás. Do novej skrine by chcela

umiestniť N kníh. Tentokrát jej však nezáleží na tom, v akom poradí ich do skrine umiestni. Samozrejme aj táto skriňa musí vôjsť do 250 cm vysokej miestnosti a Milka chce, aby bola najužšia možná.

Súťažná úloha

Váš program dostane na vstupe počet kníh N a ich výšky $v_1, \ldots, v_N$ v centimetroch. Každá z kníh má hrúbku 1 cm. Z týchto údajov by váš program mal spočítať najmenšiu možnú šírku skrine s a popísať jednu takto širokú skriňu, presnejšie:

- počet jej poličiek p
- výšky jednotlivých poličiek $w_1, \ldots, w_p$
- rozmiestnenie kníh na poličky tejto skrine

Samozrejme musia platiť nasledovné veci:

- výška každej z kníh v i-tej poličke je najviac w_i
- na každej poličke je najviac s kníh
- výška skrine $p + 1 + \sum_{i=1}^{p} w_i$ je najviac 250 (hrúbka dosky medzi poličkami je 1 cm)

Príklad:

Predpokladajme, že Milka má 13 kníh, z ktorých 9 má výšku 50 cm, zvyšné 4 majú výšku 40 cm. Najužšia možná skriňa má šírku 3. Jedna takáto skriňa vyzerá nasledovne: Má 5 poličiek, štyri z nich majú výšku 50 cm a jedna 41 cm. (Výška tejto skrine je 247 cm.) Na prvých dvoch poličkách sú po tri 50 cm vysoké knihy, na tretej sú dve 50 cm vysoké knihy a jedna 40 cm, na štvrtej jedna 50 cm a jedna 40 cm vysoká a na poslednej piatej sú posledné dve 40 cm vysoké knihy. Všimnite si, že existujú aj nižšie vhodné skrine s rovnakou šírkou.

P-II-3

Jedna z metód spracovania textu používa nasledujúci transformačný algoritmus: Na vstupe je n-znakový reťazec $C = c_1c_2...c_n$, ktorého všetky znaky sú navzájom rôzne. Keď presunieme prvé jeho písmeno na koniec, dostaneme zrotovaný reťazec. Reťazec $c_{k+1}c_{k+2}...c_nc_1...c_k$ budeme nazývať k-krát zrotovaný reťazec a budeme ho značiť C_k. Napríklad reťazec `eldat` je 3-krát zrotovaný reťazec `datel`.

Napíšme teraz do riadkov pod seba reťazce $C = C_0$, C_1, ..., C_{n-1}. Takto dostaneme tabuľku n reťazcov. Riadky tejto tabuľky zoradíme lexikograficky (t.j. podľa abecedy). Z výslednej tabuľky si zapamätáme posledný stĺpec S a číslo riadku r, v ktorom sa po zoradení nachádzal pô-

vodný reťazec. Aj keď to vyzerá dosť magicky, dvojica (S, r) nám stačí na to, aby sme vedeli jednoznačne určiť pôvodný reťazec.

Príklad transformácie

Na vstupe máme reťazec `abraka`.

Pôvodná tabuľka:	Zotriedená tabuľka:
`abraka`	`aabrak`
`brakaa`	`abraka` ←
`rakaab`	`akaabr`
`akaabr`	`brakaa`
`kaabra`	`kaabra`
`aabrak`	`rakaab`

Výsledkom transformácie je teda reťazec `karaab` a číslo 2, lebo slovo `abraka` je v druhom riadku zoradenej tabuľky.

Súťažná úloha

Váš program dostane na vstupe reťazec S dĺžky n ($1 \leq n \leq 10\,000$) a číslo r ($1 \leq r \leq n$). Tento reťazec bude tvorený malými písmenkami anglickej abecedy. **Na rozdiel od domáceho kola sa v ňom to isté písmenko môže vyskytnúť aj viackrát.** Ak programujete v Pascale, môžete predpokladať, že sa vám celý reťazec vôjde do premennej typu `string`. Úlohou vášho programu je nájsť taký reťazec C, aby výsledkom vyššie popísanej transformácie reťazca C bola práve dvojica (S, r). Môžete predpokladať, že taký reťazec C existuje. Bolo by dobré, keby časové aj pamäťové nároky vášho algoritmu boli lepšie ako kvadratické od n.

Príklad:

Vstup	**Výstup**
`karaab` `2`	`abraka`

P-II-4

Táto úloha sa týka reverzibilných algoritmov. Ich definícia je rovnaká ako v domácom kole. Študijný text z domáceho kola nájdete za zadaním súťažnej úlohy.

V hlavnom meste nemenovaného štátu S sídli inštitúcia I. Za dlhé roky svojej existencie sa inštitúcia I tak rozrástla, že dnes má N ($N > 1$) miestností (očíslovaných od 1 do N) plných byrokratov. A ako to už býva, byrokrati si medzi sebou radi posielajú listy, tlačivá, doklady, obežníky, potvrdenia, overené kópie, lístky na obed a kopu iných dôležitých materiálov. Preto si na štátne náklady dali postaviť potrubnú poštu.

Potrubná pošta je tvorená sieťou rúr. Každá rúra vedie z jednej kancelárie do druhej a za pomoci stlačeného vzduchu sa ňou (len jedným smerom, aby nedošlo ku kolíziám) dajú posielať zásielky. V inštitúcii I v miestnosti N sedí riaditeľka R a v miestnosti 1 sedí lenivec L. Lenivec L mal už dávno riaditeľke zaniesť dôležitú faktúru, ale bol lenivý, tak si radšej počkal, kým postavia potrubnú poštu. Teraz by chcel vedieť, či už ňou vie poslať riaditeľke spomínanú faktúru. (Nemusí ju posielať priamo, faktúra môže počas svojej cesty prejsť ľubovoľne veľa rúrami.) Je však lenivý zistiť si to, preto táto milá úloha pripadne vám.

Súťažná úloha

Napíšte *reverzibilnú* procedúru `Skumaj(var N:word; var A:array[1..N] of array[1..N] of bit; var D:word)`, ktorá dostane ako vstup počet budov N a maticu A, popisujúcu sieť rúr. (Ak z miestnosti i vedie rúra do miestnosti j, je $A[i][j] = 1$, inak $A[i][j] = 0$.) Ak sa dá poslať pošta z miestnosti 1 do miestnosti N, mala by táto procedúra zistiť, najmenej koľkými rúrami pošta po ceste prejde a tento počet pripočítať k premennej D. Ak sa pošta z miestnosti 1 do miestnosti N nedá poslať, procedúra by nemala zmeniť obsah premennej D.

Váš program by mal mať čo najmenšiu **pamäťovú** zložitosť, pričom ale jeho časová zložitosť musí byť polynomiálna od N.

Príklad:

Vstup	**Výstup**
$N = 4$	$D = 2$
rúry: $1 \to 3$, $2 \to 4$, $3 \to 2$, $3 \to 4$	*(Najkratší spôsob poslania pošty*
$D = 0$	*je cez 2 rúry:* $1 \to 3 \to 4$*.)*

Študijný text

Pri hľadaní úspornejších polovodičových technológii sa zistilo, že najviac energie sa spotrebúva pri mazaní informácií. Teda optimálne sú tie výpočty, pri ktorých sa žiadne informácie nestrácajú. Takýmto výpočtom sa hovorí *reverzibilné*, lebo vďaka tejto vlastnosti môžu prebiehať oboma smermi – dá sa určiť nielen zo vstupu výstup, ale aj z výstupu vstup. Aj my sa teraz vydáme do tohoto zvláštneho symetrického sveta a preskúmame, ako sa programuje „ekologicky“.

Začneme tým najjednoduchším, čo v klasických programovacích jazykoch máme, a to priraďovacím príkazom. Tu si nič také bohužiaľ nemôžeme dovoliť, lebo by sme stratili pôvodný obsah premennej, do ktorej priraďujeme! Namiesto toho zavedieme niekoľko príkazov, ktoré budú premennú

modifikovať vratne:

- `premenná += hodnota` – pripočíta hodnotu k premennej
- `premenná -= hodnota` – odpočíta hodnotu od premennej
- `premenná ^= hodnota` – prixoruje hodnotu k premennej
- `premenná =:= premenná` – vymení obsah dvoch premenných

Operácia **xor** je bitová operácia, ktorá má pre dve jednobitové čísla výsledok 1 práve vtedy, keď sú rôzne. Teda (0 xor 0) = (1 xor 1) = 0 a (0 xor 1) = (1 xor 0) = 1. Viacbitové čísla sa xorujú po bitoch – i-tý bit výsledku je i-tý bit prvého čísla **xor** i-tý bit druhého čísla. Napr. 5 xor 14 = $(0101)_2$ xor $(1110)_2$ = $(1011)_2$ = 11. Operácia **xor** má veľa užitočných vlastností, okrem iného (x xor y) = (y xor x), (x xor x) = 0, (x xor 0) = x a ((x xor y) xor z) = (x xor (y xor z)). Podobne sa dajú zaviesť aj bitový **and** a **or**: je (0 and 0) = (0 and 1) = (1 and 0) = 0, (1 and 1) = 1, (0 or 0) = 0, (0 or 1) = (1 or 0) = (1 or 1) = 1. Tieto operácie nás až tak nebudú zaujímať, lebo nie sú reverzibilné.

Aby sme sa vyhli problémom s pretečením (čo je potom inverzná operácia?), dohodneme sa, že budeme počítať len s nezápornými celými číslami od 0 do `maxword`. Takýmto číslam budeme odteraz hovoriť *prirodzené*. Všetky operácie budeme robiť modulo (`maxword`+1), čiže výsledkom každej z operácií na prirodzených číslach bude opäť prirodzené číslo. Navyše príkaz `-=` bude naozaj inverzný k `+=` a naopak. Príkazy `^=` a `=:=` sú zjavne inverzné samy k sebe.

Čo všetko ale môže byť `hodnota`? Iste to môže byť ľubovoľná konštanta. Takisto to môže byť ľubovoľná premenná, samozrejme okrem tej, do ktorej priraďujeme. Inak by sme mohli napísať napr. „`a -= a`“, čo zjavne nie je reverzibilné. Ešte by sme mali povoliť základné aritmetické operácie – tie samy nemusia byť reverzibilné, stačí, keď reverzibilne spracujeme ich výsledok. Každý zložitejší výraz potom môžeme prepísať na výrazy s jednou operáciou, napr. „`x ^= (a*b)+(c*d)`“rozpíšeme takto:

```
t1 += a*b;
t2 += c*d;
x ^= t1+t2;
t2 -= c*d;
t1 -= a*b;
```

Pritom `t1` a `t2` sú pomocné premenné, ktoré sú na počiatku výpočtu nulové a po dopočítaní výrazu sa opäť k nulovým vrátia, takže ich môžeme znovu použiť. Podobne sa dá rozpísať do reverzibilného tvaru výpočet ľubovoľného výrazu, takže odteraz môžeme používať aj zložené výrazy (bez toho, aby sme ich museli rozpisovať).

Trik s odpočítavaním medzivýsledkov a spúšťaním častí programu odzadu sa nám ešte môže hodiť. Zadefinujme si teda, že `undo prikaz` zna-

mená spustiť príkaz odzadu a že `wrap prikaz1 on prikaz2` najskôr vykoná `prikaz1`, potom `prikaz2` a nakoniec `undo prikaz1`. Náš príklad s postupným výpočtom výrazu by sme teda mohli prepísať nasledovne:

```
wrap
  begin
    t1 += a*b;
    t2 += c*d;
  end
on x ^= t1+t2;
```

Podmienené príkazy `if-then-else` môžeme používať, ak zaručíme, že po vykonaní podmieneného príkazu bude pravdivosť podmienky rovnaká ako pred jeho vykonaním (napríklad preto, že žiadnu z premenných, ktoré sú v podmienke, v podmienenej časti programu nemeníme). Potom totiž vieme aj pri vykonávaní výpočtu odzadu rozhodnúť, ktorou vetvou sa má výpočet vydať.

Ťažšie to bude u cyklov. Tam si nevystačíme s nemeniacou sa podmienkou – to by cyklus buď neprebehol ani raz, alebo sa opakoval do nekonečna. My budeme používať cykly typu `for`. Tie budú reverzibilné, ak vnútri cyklu nikde nemeníme riadiacu premennú cyklu, ani jej hranice. Toto nie je až také veľké obmedzenie – nesmie sa to robiť ani v niektorých obyčajných programovacích jazykoch. Navyše aby nás netrápilo, čo bolo v riadiacej premennej pred začiatkom cyklu a čo je v nej po jeho skončení, dohodneme sa, že príkaz `for` si túto premennú sám vytvorí a na konci ju zase zruší.

Príkaz `goto` pre istotu zakážeme úplne.

Procedúry môžu fungovať reverzibilne, ale musíme sa vyhnúť kopírovaniu parametrov a výsledkov. Všetky premenné preto budeme procedúre odovzdávať odkazom (v Pascale `var`, v C `*`). Lokálne premenné procedúry budú pri jej spustení nulové a procedúra ich musí pred skončením opäť uviesť do tohoto stavu. Rekurzia funguje bez problémov.

Teraz už máme všetko pripravené na to, aby sme vybudovali reverzibilný programovací jazyk. Ten náš bude príbuzný Pascalu a bude vyzerať takto:

Dátové typy. K dispozícii máme typy `word` (nezáporné celé číslo), `bit` (jednobitové číslo, t.j. 0 alebo 1, používa sa aj pre pravdivostné hodnoty ako pascalovský `boolean`) a pole `array[x..y] of typ` (`x` a `y` udávajú rozmedzie indexov a okrem čísel to môžu byť aj výrazy, ktorých hodnota sa počas existencie poľa nezmení). Prvky polí môžu byť aj polia, takto získame viacrozmerné polia. Svoj vlastný typ si môžete zaviesť deklaráciou `type identifikator = typ`, napr.:

```
type boolean = bit;
```

`type screen = array[0..199] of array[0..319] of bit;`

Identifikátory slúžia na pomenovanie typov, premenných a procedúr a sú to ľubovoľné reťazce písmen, číslic a znakov '_', ktoré nezačínajú číslicou a nezhodujú sa s niektorým z kľúčových slov jazyka. Malé a veľké písmená sa nerozlišujú.

Procedúry sa deklarujú konštrukciou:

`procedure identifikator ( parametre );`

(deklarácie lokálnych typov, premenných a procedúr)

`begin`

(príkazy oddelené bodkočiarkami)

`end;`

Parametre procedúry majú syntax `var meno : typ`, kde `meno` je identifikátor, ktorým sa na parameter odkazujeme vnútri procedúry. Ak je parametrov viac, oddeľujú sa pri deklarácii procedúry bodkočiarkami. Ak sú parametre rovnakého typu, môžeme zápis skracovať, napr. `procedure X(var m,n : word; var A:array[1..n] of bit);` Všetky objekty deklarované vnútri procedúry (parametre, typy, premenné aj vnorené procedúry) existujú len počas behu procedúry. Každá procedúra vidí svoje lokálne premenné a navyše aj lokálne premenné všetkých procedúr, vnútri ktorých je deklarovaná (ak sa ich názvy líšia od názvov jej lokálnych premenných). Presne rovnako to funguje v Pascale.

Premenné sú pomenované identifikátormi, musia sa vytvoriť deklaráciou `var identifikator : typ;`. Pri vstupe do procedúry, v ktorej sú deklarované, majú nulovú hodnotu (v prípade poľa: všetky jeho prvky majú nulovú hodnotu) a predtým, ako premenná na konci procedúry zanikne, musí byť jej hodnota opäť nulová. Deklaráciu viac premenných toho istého typu môžeme skrátene zapísať `var i1, i2, ..., in : typ`.

Výrazy môžu obsahovať:

- konštanty (prirodzené čísla a konštanta `maxword` – najväčšie prirodzené číslo)
- premenné
- prvky polí (`pole[vyraz]`)
- aritmetické operácie, ktorých vstupom aj výstupom sú prirodzené čísla: $+$, $-$, $*$, **div** (celá časť podielu), **mod** (zvyšok po delení), **and**, **or**, **xor** (bitové operácie, definície viď vyššie) a **not** (prehodenie nulových a jednotkových bitov) – výsledky operácií sa automaticky berú modulo (`maxword`+1)
- relačné operácie (vstupom sú dve prirodzené čísla, výstupom bitová hodnota 1, ak relácia platí a 0, ak neplatí): $<$, $>$, $=$, $<=$, $>=$, $<>$
- zátvorky

Príkazy môžu byť nasledovných druhov:

- Blok: `begin` (príkazy oddelené bodkočiarkami) `end` – spôsobí postupné vykonanie príkazov, ktoré obsahuje, v danom poradí.
- Modifikačné príkazy: `premenna += vyraz` – spôsobí vyhodnotenie výrazu a jeho pripočítanie k premennej. Pritom `premenna` môže byť aj prvok poľa indexovaný nejakým výrazom. Premenná (resp. prvok poľa), ktorú príkaz modifikuje, sa už nikde v tomto príkaze nesmie vyskytnúť. Analogicky príkazy `-=` a `^=`.
- Vymieňací príkaz: `premenna =:= premenna` – vymení obsah dvoch premenných rovnakého typu. Ak sa jedná o prvky polí, nesmie sa žiadne z týchto polí používať vo výrazoch určujúcich indexy.
- Podmienený príkaz: `if podmienka then prikaz1 else prikaz2` – vyhodnotí sa podmienka (výraz s bitovým výsledkom), ak je výsledok 1, vykoná sa `prikaz1`, inak sa vykoná `prikaz2`. Platnosť podmienky sa vykonaním príslušného príkazu nesmie zmeniť. Časť `else` sa môže vypustiť, prípadné `else` sa vzťahuje k najbližšiemu neukončenému `if`.
- Príkaz cyklu: `for var identifikator = d to h do prikaz` – založí novú premennú daného mena, daný príkaz postupne vykonáva pre túto premennú nadobúdajúcu hodnoty `d`, `d`+1 až `h` a nakoniec riadiacu premennú opäť zruší. Hranice `d` a `h` sú celočíselné výrazy, ak `d`>`h`, cyklus sa ani raz nevykoná. Príkaz `prikaz` musí zachovávať hodnotu riadiacej premennej aj oboch hraníc cyklu (presnejšie: môže ich modifikovať, ale na konci prechodu cyklom musia mať tú istú hodnotu ako mali na začiatku príslušného prechodu). Takisto sa dá použiť konštrukcia `h downto d` namiesto `d to h`, potom bude riadiaca premenná nadobúdať hodnoty v opačnom poradí, t.j. `h`, `h`-1, až `d`.
- Volanie procedúry: `nazov_procedury ( par1, ..., parN )` – zavolá procedúru so zadanými parametrami. Parametre môžu byť premenné alebo prvky poľa (potom ale výraz, určujúci index prvku, musí mať po návrate z procedúry rovnakú hodnotu ako pred vstupom do nej). Počet parametrov aj ich typy musia zodpovedať deklarácii procedúry.
- Príkaz obrátenia výpočtu: `undo príkaz` – vykoná daný príkaz „odzadu“ podľa nasledujúcich pravidiel:

`undo begin`
`  p1; ...; pn`
`end`
	`begin`
	`  undo pn; ...; undo p1`
	`end`

`undo x += y`	`x -= y`
`undo x -= y`	`x += y`
`undo x ^= y`	`x ^= y`
`undo x =:= y`	`x =:= y`
`undo if x then y else z`	`if x then undo y` `  else undo z`
`undo for x = d to h do p`	`for x = h downto d do` `  undo p`
`undo P(x1; ...; xn)`	`undo` tela procedúry (`begin ... end`)
`undo undo p`	`p`

Konštrukcia `begin p; undo p; end` teda nevykoná nič. (Aj keď počítať môže pomerne dlho.)

- Príkaz lokálneho výpočtu: `wrap prikaz1 on prikaz2` – skrátený zápis konštrukcie `begin prikaz1; prikaz2; undo prikaz1; end`.

Komentáre. Čokoľvek uzavreté v zložených zátvorkách (‘`{`’ a ‘`}`’) je komentár, ktorý počítač ignoruje (ako keby namiesto neho boli medzery). Komentár nesmie obsahovať zložené zátvorky.

Hlavný program nebudeme zavádzať. Aby sme sa vyhli problémom so vstupom a výstupom, budeme všetko programovať ako procedúry. Tie dostanú ako svoje parametre premenné, obsahujúce vstupné dáta a premenné, ktoré majú byť predpísaným spôsobom modifikované podľa výstupu.

Časová a pamäťová zložitosť sú definované podobne ako v klasickom programovaní: Časová zložitosť je počet vykonaných príkazov. Veľkosť pamäti využitej v danom okamihu spočítame ako súčet veľkostí všetkých lokálnych premenných (typy bit a word majú jednotkovú veľkosť, veľkosť poľa je súčet veľkostí jeho prvkov), počtu všetkých parametrov práve zavolaných procedúr a počtu práve zavolaných procedúr. Parametre sa bez ohľadu na ich typ počítajú ako jednotka, lebo sa odovzdávajú odkazom. Potom pamäťová zložitosť je maximum množstva využitej pamäti počas behu programu. Pozor! Keďže aj náš program je procedúra, jeho vstupy a výstupy sa do pamäťovej zložitosti započítavajú len ako konštanty, aj keď to môžu byť veľké polia.

Príklad 1

Procedúra na prehodenie obsahu dvoch premenných (ktorá vlastne ukazuje, že príkaz `=:=` vieme odvodiť pomocou ostatných príkazov). Časová aj pamäťová zložitosť sú konštantné, teda $O(1)$.

```
procedure Vymen(var x,y : word);
begin         { x = X, y = Y (X,Y sú pôv. hodnoty) }
  x ^= y;     { x = X xor Y, y = Y }
  y ^= x;     { x = X xor Y, y = Y xor (X xor Y) = X }
  x ^= y      { x = (X xor Y) xor X = Y, y = X }
end;
```

Príklad 2

Procedúra na výpočet maxima zo zadaných `n` čísel. Dané je pole `X` celých čísel a premenná `max`, ku ktorej máme hľadané maximum pripočítať. To dokážeme takto: najskôr predpočítame pole `M`, kde `M[i]` bude maximum spomedzi čísel `X[1]` až `X[i]`. Potom pripočítame `M[n]` ku `max` a nakoniec `M[i]` opäť vyprázdnime. Časová aj pamäťová zložitosť sú lineárne, teda $O(n)$.

```
procedure Maximum(var n:word; var X:array [1..n] of word;
                  var max:word);
var M:array [0..n] of word;
begin
  wrap for var i=1 to n do
         if X[i]>M[i-1] then
           M[i] += X[i]
         else
           M[i] += M[i-1]
  on max += M[n];
end;
```

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY

52. ROČNÍK MATEMATICKEJ OLYMPIÁDY

Zadania 2. kola kategórie P

Vydala IUVENTA
pre vnútornú potrebu Ministerstva školstva SR
Autori príkladov P. Töpfer, D. Kráľ, M. Mareš
Zodpovedný redaktor M. Forišek
Sadzba programom LaTeX