Priebeh krajského kola

Krajské kolo 36. ročníka Olympiády v informatike, kategória B, sa koná 19. januára 2021 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 nejakom bežnom programovacom jazyku (napr. C++, Python, Java, Pascal).
- 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.

B-II-1 Výstavba v Egypte pokračuje

Aj vďaka vašej pomoci s navigáciou lodí mestá pozdĺž Nílu utešene rastú. Kontrolné stanice pozdĺž rieky boli zrušené a úradníci sa z nich presunuli na staveniská, kde zisťujú, koľko kameňa je v skutočnosti potrebného na dokončenie výstavby. Výsledok? Požadované množstvá kameňa sa naprieč mestami diametrálne líšia.
Všetky svoje požiadavky poslali správcovi kameňolomu, ktorého úlohou je naplánovať rozvoz kameňa. Správca vymyslel veľmi jednoduchý postup – lode s nákladom bude posielať jednu po druhej a to zakaždým do toho mesta, ktoré aktuálne potrebuje najviac kameňa.
Presnejšie, jeho spôsob vysielania lodí funguje nasledovne. V Egypte je n miest, ktoré sú očíslované od 1 po n. Správca dostal pre každé mesto hodnotu x_i, ktorá určuje, že mesto i potrebuje dostať x_i zásielok kameňa. Keď správca vysiela ďalšiu loď, jednoducho vyberie mesto s najväčšou hodnotou x_i a do tohto mesta túto loď vyšle, pričom si zároveň hodnotu x_i zmenší o 1. V prípade, že viacero miest potrebuje rovnako veľa dodávok kameňa, vyberie z nich to, ktorého **počiatočná hodnota x_i bola najväčšia**, a ak aj tomuto kritériu vyhovuje viacero miest, vyberie z nich to s najnižšou hodnotou i.
Kapitán Peter vie, že jeho loď bude vyslaná k-ta v poradí. A keďže správcove rozhodovanie nie je založené na náhode, mal by byť schopný zistiť, do ktorého mesta sa poplaví. Nevie si s tým však rady, preto sa obrátil na vás. A povedal o tom aj svojim kamarátom, je preto možné, že tých otázok budete musieť zodpovedať viac ...

Odporúčanie: skôr ako začnete túto úlohu riešiť, pozorne si prečítajte časť *Hodnotenie*, keďže úloha má dva varianty na základe počtu hodnôt k, na ktoré je potrebné odpovedať.

Formát vstupu a výstupu

V prvom riadku vstupu je číslo n označujúce počet miest v Egypte.
V druhom riadku je n čísel x_1 až x_n – počet zásielok kameňa, ktoré je potrebné dopraviť do jednotlivých miest. Hodnota x_i prislúcha mestu s číslom i.
Nasleduje riadok s číslom q, ktoré určuje počet otázok, na ktoré máte odpovedať.
Na ďalšom riadku je potom q čísel k_1 až k_q – jednotlivé otázky. Môžete predpokladať, že zadané lode budú naozaj vypravené, teda všetky hodnoty k_j sú nanajvýš rovné súčtu hodnôt x_i.

Na výstup vypíšte q riadkov, na každom bude odpoveď na jednu otázku. Na riadok i vypíšte číslo mesta, do ktorého bude poslaná k_i-ta loď v poradí.

Obmedzenia a hodnotenie

- Nanajvýš **10 bodov** dostanete za riešenie, ktoré efektívne vyrieši ľubovoľný vstup, v ktorom platí $n \leq 1\,000\,000$, $0 \leq x_i, k_j \leq 10^{12}$ a $q \leq 1\,000\,000$.
- Nanajvýš **7 bodov** dostanete za riešenie, ktoré efektívne vyrieši ľubovoľný vstup, v ktorom platí $n \leq 1\,000\,000$, $0 \leq x_i, k_j \leq 10^{12}$ a $q = 1$. To znamená, že viete **efektívne zistiť odpoveď na jednu otázku**.
- Nanajvýš **5 bodov** dostanete za riešenie, ktoré efektívne vyrieši ľubovoľný vstup, v ktorom platí $n \leq 1000$, $0 \leq x_i, k_j \leq 10^{12}$ a $q = 1$.
- Nanajvýš **3 body** dostanete za ľubovoľné iné správne riešenie, napríklad aj také, v ktorom predpokladáte, že $0 \leq n, x_i, k_j \leq 1000$ a $q = 1$.

(Zadanie pokračuje príkladom na nasledujúcej strane.)

Príklad

vstup	výstup
5 3 4 8 6 4 4 8 3 18 25	4 3 2 1

Prvá loď ide do mesta 3, ktoré potrebuje až 8 dodávok kameňa. Po nej sa stav zmení na $(3, 4, 7, 6, 4)$*. Aj druhá loď preto pôjde do mesta 3, aktuálny stav je* $(3, 4, 6, 6, 4)$*.*

V tomto momente sú až dve mestá, v ktorých je potrebných 6 dodávok (čo je aktuálne najviac), uprednostní sa preto mesto s väčšou počiatočnou hodnotou, teda opäť mesto 3 (lebo $8 > 6$*). Ďalšie lode pôjdu postupne do miest: 4, 3, 4, 3, 4, 2.*

Po vyslaní sedemnástej lode bude stav potrieb $(2, 2, 1, 1, 2)$*. Ďalšia loď preto musí ísť do niektorého z miest 1, 2 alebo 5. Z nich mali najvyššiu pôvodnú hodnotu mestá 2 a 5 (táto hodnota bola 4). Musí sa preto použiť posledné kritérium a z týchto dvoch miest sa vyberie to s menším číslom, teda mesto 2.*

Dvadsiata piata loď v poradí je posledná, ktorá bude vyslaná, po nej budú potreby všetkých miest naplnené. Táto loď bude smerovať do mesta číslo 1.

B-II-2 Generátor zátvoriek

Ako to už s deťmi býva, hranie sa v parku s trojuholníkmi Tima rýchlo prestalo baviť. Jeho novou záľubou je počítanie. Rodičia sú už unavení z jeho neustáleho sčítavania a odčítavania všetkého čo mu príde pod ruku. Rozhodli sa, že mu budú dávať o niečo ťažšie úlohy obsahujúce aj zátvorky. Nakoľko je Tim šikovný, rodičia veria, že zvládne rovno dva typy zátvoriek. Rozhodli sa, že ho otestujú – budú mu postupne dávať rôzne uzátvorkované úlohy a budú sledovať, ako sa s nimi popasuje. Radi by sa však vyhli tomu, že Tim bude riešiť tak isto uzátvorkovaný výraz viackrát. Chceli by mať nejaký dômyselný systém na generovanie neopakujúcich sa uzátvorkovaní. V tejto úlohe im s tým pomôžete.

Súťažná úloha

Vašou úlohou bude vymyslieť program, ktorý dostane na vstupe dobre uzátvorkovaný reťazec a vráti nasledujúci v tzv. lexikografickom poradí. (Nižšie si vysvetlíme, čo to presne znamená.)

V tejto úlohe budeme uvažovať iba dva typy zátvoriek a to guľaté - `()` a hranaté - `[]`. Dobre uzátvorkované reťazce zadefinujeme rekurzívne:

1. `()` a `[]` sú najmenšie dobre uzátvorkované reťazce.
2. Ak x a y predstavujú dobre uzátvorkované reťazce tak aj `(`x`)`, `[`y`]` a xy sú dobre uzátvorkované reťazce.

Uvedomte si, že táto možno zložitá definícia vlastne len formálne popisuje dobré uzátvorkovania s akými ste sa zrejme už stretli. Príkladom dobre uzátvorkovaného výrazu je `[()()]`. Ten vznikol tak, že sme vzali dva najmenšie dobre uzátvorkované reťazce $x =$ `()` a $y =$ `()`, spojili sme ich druhým pravidlom do $z = xy =$ `()()` a potom sme tento nový výraz z (nakoľko je tiež dobre uzátvorkovaným) ešte obalili zátvorkami (teda použili zas druhé pravidlo `[`z`]`) aby vznikol finálny reťazec `[()()]`. Príkladom zle uzátvorkovaného výrazu je napríklad `())(`, `)(` alebo `[(]]`. V prvom prípade je vo výraze o jednu zatváraciu zátvorku naviac, v druhom je zatváracia zátvorka ktorá nemá pred sebou otváraciu, no a v treťom prípade sedí počet aj poradie otváracích a zatváracích zátvoriek, nesedia však ich typy.

Lexikografické usporiadanie je usporiadanie na základe poradia, ktoré by dané reťazce mali v slovníku. V prípade slov je to poradie na základe písmen, ktoré dané slová obsahujú. Slovo `aaaa` je v slovníku skôr ako slovo `aaba`. To z dôvodu, že prvá pozícia kde sa líšia je predposledné miesto, pričom prvé slovo má na ňom `a`, ktoré je menšie ako `b` v druhom slove.
Pre naše uzátvorkované reťazce to bude fungovať podobne, len sa musíme dohodnúť na poradí jednotlivých symbolov v abecede. My použijeme poradie `()[]`.
Ak teraz chceme porovnať dva rôzne reťazce rovnakej dĺžky, ideme zľava doprava, až kým nenájdeme prvú pozíciu, na ktorej sa líšia. Ten reťazec, ktorý tam má skorší symbol, je v lexikograficky usporiadanom poradí skôr.
Určime napríklad, či je lexikograficky menší reťazec `()()[]` alebo `()[]()`. Nájdeme prvé miesto, na ktorom sa nezhodujú a to je na pozícii 3 (číslované od 1). Na tomto mieste sa nachádza v prvom reťazci zátvorka `(` a v druhom `[`. Kedže je `(` v našom zozname skôr ako `[`, hovoríme, že prvý reťazec – `()()[]` – je lexikograficky menší ako druhý – `()[]()`.

Formát vstupu a výstupu

Na vstupe je jediný reťazec zátvoriek s. Jeho dĺžku označíme n. Je zaručené, že tento reťazec je **korektne uzátvorkovaný**.
Predstavme si, že sme úplne všetky možné korektne uzátvorkované reťazce dĺžky n lexikograficky usporiadali. Na výstup vypíšte ten, ktorý by nasledoval bezprostredne po reťazci s. Ak taký reťazec neexistuje, podajte o tom správu.
(Zadanie pokračuje na nasledujúcej strane popisom hodnotenia úlohy a príkladmi.

Hodnotenie

Plný počet 10 bodov môžete získať za riešenie, ktoré efektívne vyrieši ľubovoľný vstup, v ktorom pre dĺžku reťazca s platí $n \leq 1\,000\,000$.
Nanajvýš 6 bodov môžete získať za riešenie, ktoré efektívne vyrieši ľubovoľný vstup, v ktorom platí, že $n \leq 1\,000\,000$ a navyše predpokladáme, že dobre uzátvorkované výrazy obsahujú **iba guľaté zátvorky**. Neuvažujeme teda výrazy, ktoré obsahujú hranaté zátvorky a teda platí, že ako na vstupe tak aj v lexikografickom poradí sú iba výrazy zložené z guľatých zátvoriek.
Ak budú vaše riešenia efektívne iba po zhruba $n \leq 5000$, dostanete nanajvýš 8 bodov ak zvládate oba typy zátvoriek, resp. nanajvýš 5 bodov ak pracujete len s guľatými zátvorkami.
Nanajvýš 4 body dostanete za ľubovoľné správne riešenie, v ktorom predpokladáte, že $n \leq 10$.
Nanajvýš 3 body dostanete za ľubovoľné správne riešenie, v ktorom predpokladáte, že $n \leq 10$ a navyše tiež, že dobre uzátvorkované výrazy obsahujú **iba guľaté zátvorky**.

Príklady

vstup	výstup
`((()))`	`(()())`

Reťazec na vstupe je lexikograficky prvým dobre uzátvorkovaným reťazcom dĺžky 6.

vstup	výstup
`()[]([])`	`()[]([])()`

vstup	výstup
`()([])`	`()[()]`

vstup	výstup
`[][][]`	`neexistuje`

Ide o lexikograficky posledný reťazec dĺžky 6. Akýkoľvek iný je v lexikografickom usporiadaní skôr.

B-II-3 Sociálne dištančné preteky

V hlavnom meste Absurdistanu sa každoročne organizujú hromadné bežecké preteky. Tento ročník je však zo zrejmých dôvodov ohrozený. Hlavný organizátor Andrej by ho však napriek tomu rád zorganizoval, samozrejme držiac sa platných epidemiologických opatrení. Predsa len, treba posilniť imunitu športom na čerstvom vzduchu. Bežne vedú preteky z Námestia nezmyselnosti niekoľkými uličkami mesta až na Námestie nonsensu. Andrejovi napadlo, že tento rok by mohol mať každý účastník vlastnú trasu uličkami mesta tak, aby sa tieto trasy navzájom nekrižovali, čím by zabránil nebezpečnému kontaktu. Teraz však sedí nad mapou mesta a rozmýšľa, koľko najviac účastníkov sa môže pretekov zúčastniť. Vedeli by ste mu pomôcť?

Súťažná úloha

Andrejovým najväčším problémom je špecifická topológia hlavného mesta Absurdistanu. Skladá sa totiž z n námestí, ktoré sú navzájom pospájané nekrižujúcimi sa uličkami. Každá ulička teda spája práve dve námestia a nedá sa na nej nikam inam odbočiť.
Takýchto uličiek je v meste **presne** $n - 1$ a platí, že medzi každými dvoma námestiami existuje práve jedna cesta (postupnosť rôznych uličiek), ktorá ich spája. V meste totiž neexistujú žiadne cykly – ak vyrazíte na prechádzku z nejakého námestia, nie je možné sa naň vrátiť bez toho, aby ste nemuseli po nejakej uličke prejsť viackrát. (Matematik by tomu povedal, že mesto má stromovú topológiu.)

Bežecká trasa začína na nejakom námestí, následne pozostáva z niekoľkých rôznych na seba nadväzujúcich uličiek a končí na inom námestí. Dve trasy sa nekrižujú, ak nemajú žiaden spoločný bod. (Teda nie len že nesmú ísť tou istou uličkou, ale dokonca ani nesmú mať na trase to isté námestie.)

Posledným logistickým problémom je, že nie všetky námestia sú vhodné na umiestnenie štartu alebo cieľu trasy. Tie vyhovujúce sú na mape zaznačené. Andrejove trasy musia všetky začínať aj končiť na vyhovujúcich námestiach.

Koľko najviac rôznych nekrižujúcich sa trás vie Andrej v meste vytvoriť?

Formát vstupu a výstupu

Na prvom riadku je číslo n $(2 \leq n)$ udávajúce počet námestí v hlavnom meste. Námestia sú označené číslami od 1 po n.
Nasleduje $n - 1$ riadkov, ktoré popisujú uličky mesta. Na každom riadku sú dve čísla x_i a y_i udávajúce uličku medzi námestiami x_i a y_i. Táto ulička je obojsmerná a dá sa ňou prechádzať v ľubovoľnom smere. Platí, že $1 \leq x_i \neq y_i \leq n$ a mapa vytvorená zo zadaných uličiek spĺňa podmienky uvedené v časti „Súťažná úloha“.
Ďalší riadok obsahuje číslo k $(0 \leq k \leq n)$ – počet námestí, do ktorých môže byť umiestnený štart alebo koniec trasy. Za ním je riadok s k rôznymi, medzerami oddelenými, číslami p_1 až p_k – čísla vyhovujúcich námestí. Platí, že $1 \leq p_i \leq n$.

Na výstup vypíšte jediné číslo – maximálny počet nekrižujúcich sa trás začínajúcich a končiacich na vyznačených námestiach, ktoré je možné na mape vyznačiť.

Hodnotenie

Plný počet bodov môžete získať za riešenie, ktoré efektívne vyrieši ľubovoľný vstup, v ktorom platí $n \leq 1\,000\,000$.
Až 6 bodov môžete získať za riešenie, ktoré efektívne vyrieši ľubovoľný vstup, v ktorom platí $n \leq 1\,000\,000$ **a zároveň predpokladáte, že** $k = n$ (čiže na každom námestí sa dá začínať aj končiť trasa).
Za ľubovoľné správne riešenie riešenie získate aspoň 3 body.

Upozornenie: pri opravovaní tejto úlohy budeme obzvlášť prihliadať na odôvodnenia správnosti vašich riešení. Pokúste sa preto čo najlepšie zdôvodniť, že vami nájdený výsledok je naozaj maximálny možný a žiaden iný návrh trás nevedie k väčšiemu počtu trás.

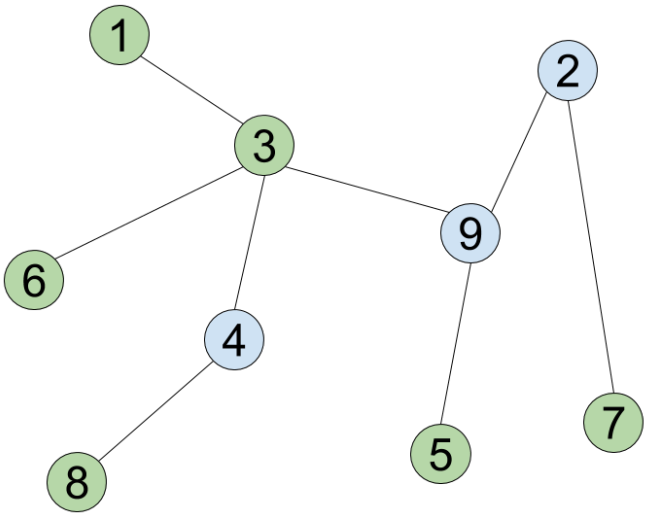
Príklad

vstup

```
9
1 3
3 6
4 3
3 9
4 8
5 9
9 2
2 7
6
1 3 6 8 5 7
```

výstup

```
2
```



Zadané mesto zodpovedá mape zobrazenej na obrázku vyššie. Krúžky sú námestia, pričom zelenou sú označené tie, kde môžu začínať a končiť trasy.
Napriek tomu, že máme až 6 takýchto námestí, nevieme navrhnúť viac ako dve nekrižujúce sa trasy. Existuje viacero možností na vytvorenie týchto dvoch trás. Jednou z nich sú trasy $5 - 9 - 2 - 7$ *a* $1 - 3 - 6$. *(Všimnite si, že trasa môže prechádzať aj cez iné „zelené“ námestia, a tiež že rôzne trasy môžu byť rôzne dlhé.)*
Rovnako dobre by sme však mohli napríklad namiesto trasy $1 - 3 - 6$ *použiť trasu* $1 - 3$ *alebo* $1 - 3 - 4 - 8$.

B-II-4 Bitové operácie

K tejto úlohe patrí aj študijný text uvedený na nasledujúcich stranách. Je totožný so študijným textom z domáceho kola.

Podúloha A (1 bod). V registri x je vstupné číslo. Napíšte program, ktorý *v konštantnom čase* zrotuje bity tohto čísla o tri pozície doprava.
Napríklad ak máme registre s $B = 8$ bitmi, pre vstup $x = 10111111$ je správnym výstupom 11110111.

Podúloha B (5 bodov). V registri x je vstupné číslo. Napíšte program, ktorý zistí, či sa toto číslo dá zapísať ako rozdiel dvoch rôznych mocnín dvojky. (Plný počet bodov je za riešenie v konštantnom čase.)

Podúloha C (4 body). V registri x je vstupné číslo. Je zaručené, že x je rovné $2^y - 2^z$ pre nejaké y, z také, že $0 \leq z < y < B$.
Napíšte program, ktorý z čísla x určí hodnoty y a z. Na plný počet bodov je potrebné, aby časová zložitosť vášho programu bola lepšia ako lineárna od počtu bitov registra.

Študijný text: Registre

Jednou zo základných operácií, ktoré vedia moderné procesory robiť, je aritmetika na prirodzených číslach. Presnejšie, nejde o výpočty so skutočnými prirodzenými číslami (ktoré môžu byť ľubovoľne veľké), ale o výpočty na celočíselných *registroch*. Na register sa môžeme dívať ako na celočíselnú premennú s pevne zvoleným počtom bitov. Tento počet bitov budeme označovať B.
Veľká väčšina súčasných procesorov má tzv. 64-bitovú architektúru, čo okrem iného znamená, že celočíselné registre, s ktorými pracujú, majú 64 bitov (teda $B = 64$).
Na číslo, uložené v B-bitovom registri, sa môžeme pozerať viacerými spôsobmi. Niekedy sa nám oplatí tieto hodnoty interpretovať ako celé čísla so znamienkom (signed int), v našom študijnom texte aj v súťažných úlohách si však vyberieme iba najjednoduchšiu možnosť (unsigned int). Na jednotlivé bity registra sa budeme dívať ako na cifry nezáporného celého čísla zapísaného v dvojkovej sústave. V B-bitovom registri teda vieme uložiť hodnoty od 0 až po $2^B - 1$.
V tomto študijnom texte si budeme ukazovať príklady hodnôt v registroch a operácie s nimi. V týchto príkladoch budeme kvôli lepšej čitateľnosti používať len hodnotu $B = 8$ (teda osembitové registre). Najmenej významný bit píšeme najviac vpravo. Teda napr. hodnota `00000000` uložená v osembitovom registri predstavuje číslo 0, hodnota `00000101` číslo 5 a hodnota `10000000` číslo 128.
V hardvéri procesore sú časti, pomocou ktorých vie procesor robiť rôzne jednoduché operácie s registrami. V nasledujúcom texte sa zoznámime s niektorými z nich.

Aritmetické operácie

Základné aritmetické operácie sú sčítanie, odčítanie, násobenie, celočíselné delenie a zvyšok po delení. Značiť ich budeme `+ - * div mod`.
Sčítanie a násobenie fungujú presne ako v dvojkovej sústave, s jedným rozdielom: pri výpočtoch môže dôjsť k pretečeniu, teda k situácii, kedy by sme na uloženie výsledku potrebovali viac bitov ako máme k dispozícii. V takejto situácii najvyššie bity bity, ktoré sa do registra nezmestia, jednoducho odignorujeme. Napr. ak sčítame `10000000 + 10000000`, dostaneme `00000000`.
Rozmyslite si, že takto uložená hodnota zodpovedá zvyšku, ktorý dáva skutočný výsledok po delení 2^B. (Bity, ktoré sa do registra nezmestili, totiž predstavujú túto a vyššie mocniny dvoch.) Preto niekedy hovoríme, že pri týchto výpočtoch používame *modulárnu aritmetiku*, respektíve že všetky tieto výpočty robíme „modulo 2^B“.

Odčítanie sa tiež správa rovnako, len tentokrát môže prísť k podtečeniu – ak od menšieho čísla odčítame väčšie, mali by sme dostať záporný výsledok. Aj vtedy sa jednoducho budeme správať tak, ako keby sme počítali modulo 2^B. Teda napr. -1 je to isté, ako $2^B - 1$.
Predstavte si všetkých 2^B možných hodnôt napísaných za radom po obvode kruhu, v smere ručičiek. Operácia `+7` robí posun o 7 pozícií v smere ručičiek – zväčša na hodnotu o 7 väčšiu, len v prípade pretečenia začneme znova od nuly. Presne rovnako robí operácia `-7` posun o 7 pozícií proti smeru ručičiek.

Delenie a zvyšok nemajú problémy s pretečením ani podtečením, len si musíme dať pozor na to, že deliť nulou je zakázané (a spôsobí chybu, keď sa o to pokúsime).

Bitové operácie

Ďalšiu triedu operácií predstavujú operácie, ktoré priamo manipulujú s bitmi v registri.
Prvou je unárna operácia `not`, ktorá všetky bity zmení na opačné. Napr. z hodnoty `00010110` dostaneme operáciou `not` hodnotu `11101001`.
Ďalej tu máme binárne operácie `and` („a súčasne“), `or` („alebo“) a `xor` (exclusive or, teda „buď alebo“).
Všetky tieto operácie vyrobia z dvoch hodnôt jednu novú, ale každá ináč. Pri operácii `and` majú vo výstupnej hodnote hodnotu 1 tie bity, ktoré mali hodnotu 1 v jednom *a súčasne* v druhom vstupe. Pri operácii `or` sú to tie bity, ktoré mali hodnotu 1 v jednom *alebo* v druhom vstupe. No a pri operácii `xor` majú vo výstupnej hodnote hodnotu 1 tie bity, ktoré mali hodnotu *buď* v jednom, *alebo* v druhom vstupe, *ale nie v oboch naraz*.
Pri počítaní týchto operácií je praktické napísať si vstupy aj výstup pod seba. Potom každý bit výstupu závisí len od bitov vstupu, ktoré sú v jeho stĺpci. Príklad:

```
      x = 00101101
      y = 01100110
------------------
x and y = 00100100
x  or y = 01101111
x xor y = 01001011
```

No a na záver tu máme ešte dve operácie: bitové posuny `shl` a `shr` (shift left, shift right).
Operácia `x shl y` zoberie bity hodnoty `x` a posunie ich o `y` pozícií doľava. Na ľavom konci registra sa takto najvýznamnejších `y` bitov dostane na pozície, ktoré sa už do registra nezmestia. Tieto bity stratíme. Naopak, na pravom konci registra vznikne `y` voľných pozícií. Tieto bity nastavíme na nulu. Teda napríklad `01101001 shl 2 = 10100100`.
Posun doprava funguje analogicky, napr. `10111111 shr 3 = 00010111`.
Rozmyslite si, že každý posun doľava hodnotu v registri vynásobí dvoma, zatiaľ čo každý posun doprava ju vydelí dvoma (a zahodí zvyšok).

Programovanie pomocou registrov

Programy, ktoré budeme písať, môžete písať v ľubovoľnom bežnom programovacom jazyku, len budeme mať nasledujúce úpravy:

- Neexistujú žiadne funkcie, podprogramy ani nič podobné.
- Jediný dátový typ je „register“, teda každá premenná použitá v programe predstavuje nejaký register. Neexistujú polia ani nič zložitejšie.
- Premenné netreba deklarovať. Ak nie je povedané ináč, každá premenná obsahuje hodnotu 0, keď je prvýkrát použitá.
- Pri písaní programov budeme pre jednoduchosť explicitne predpokladať, že $B = 64$. (Pri analýze ich časovej zložitosti však budeme B brať ako parameter – teda bude nás zaujímať, ako závisí počet krokov výpočtu od počtu bitov v registri.)

Príklad

V registri x je vstupná hodnota. Zistite, či je počet bitov nastavených na hodnotu 1 párny.

Riešenie 1.
V cykle prejdeme všetky bity registra a spočítame, koľko ich je nastavených.
Kontrolu, či je bit číslo i nastavený, vieme spraviť napríklad nasledovne. Hodnota `1` má nastavený na 1 len bit číslo 0. Hodnota `1 shl i` má nastavený na 1 len bit číslo i. Ak teraz vypočítame `x and (1 shl i)`, sú len dve

možnosti: ak mal aj v x bit číslo i hodnotu 1, bude tento bit nastavený na 1 aj vo výsledku. V opačnom prípade bude výsledok zjavne nulový.
Výsledný program v pseudokóde:

```
pocet_jednotiek = 0
for i = 0 to 63:
    if (x and (1 shl i)) != 0:
        pocet_jednotiek += 1

if pocet_jednotiek mod 2 == 0 then print('parny') else print ('neparny')
```

Iná možnosť ako spraviť tú istú kontrolu je naopak posunúť hodnotu x o i pozícií doprava a potom sa pozrieť na jej najmenej významný bit. Namiesto `(x and (1 shl i)) != 0` sme teda mohli písať aj napr. `(x shr i) mod 2 == 1`.

Riešenie 2.
Pozrime sa presnejšie, čo robí operácia `-1` v dvojkovej sústave. Ak odčítame 1 od nepárneho čísla, len zmeníme najmenej významný bit z 1 na 0. Ak odčítame 1 od párneho čísla, zmení sa najmenej významný bit z 0 a 1 a nastane prenos – čiže sa posunieme o bit doľava a chceme odčítať 1 od neho. Takto pokračujeme ďalej, až kým nenarazíme na prvý bit s hodnotou 1. Napr. ak odčítame 1 od `01101000`, dostaneme `01100111`.
Vždy sa teda stane to, že napravejšia 1 sa zmení na 0 a všetky 0 za ňou sa zmenia na 1.
Všimnime si teraz, čo dostaneme, keď zoberieme nejaké nenulové x a vypočítame hodnotu `x and (x-1)`. Na najvýznamnejších bitoch sa x a $x-1$ zhodujú, tieto teda ostanú zachované. Až na konci je rozdiel: kým x končí `100...0`, $x-1$ končí `011...1`. No a `and` týchto dvoch častí zjavne obsahuje samé nuly.
Hodnota `x and (x-1)` má teda presne tie isté jednotkové bity ako x, s jediným rozdielom: najmenej významná jednotka zmizla. A toto nám ponúka novú možnosť, ako spočítať jednotkové bity v danom registri:

```
pocet_jednotiek = 0
while x != 0:
    x = x and (x-1)
    pocet_jednotiek += 1
```

Riešenie 3.
V tomto riešení využijeme silu paralelizmu. Predstavte si, že sme náš 64-bitový register rozdelili na dva 32-bitové. Čo sa stane, ak vypočítame ich `xor`? Ak sa stretnú dva bity s hodnotami 0 a 1, bude vo výsledku 1. Ak sa stretnú dve 0 alebo dve 1, bude vo výsledku 0. V oboch prípadoch teda platí, že výsledná 32-bitová hodnota má rovnakú paritu počtu jednotiek ako pôvodná 64-bitová.
No a teraz môžeme pokračovať rovnako ďalej: rozdelíme 32-bitový register na dva 16-bitové, atď. Po niekoľkých kolách nám už ostane len jeden jediný bit a toho paritu už skontrolujeme ľahko.
Ako ale rozdeliť 64-bitový register na polovice? Hornú polovicu bitov vieme dostať tak, že posunieme bity registra doprava: `x shr 32` má na 32 najvýznamnejších pozíciách nuly a na 32 najmenej významných pozíciách má 32 najvyšších bitov hodnoty uloženej v x. Hodnota `x xor (x shr 32)` má preto na najnižších 32 pozíciách tých 32 bitov, ktoré sme chceli dostať. A rovnako môžeme pokračovať aj ďalej. Program teda môže vyzerať nasledovne:

```
y = x xor (x shr 32)
y = y xor (y shr 16)
y = y xor (y shr 8)
y = y xor (y shr 4)
y = y xor (y shr 2)
y = y xor (y shr 1)
if y mod 2 == 0 then print('parny') else print('neparny')
```

Toto riešenie funguje, ale vo výslednom y nám na pozíciách iných ako poslednej vznikol neporiadok. Pre názornosť si ukážeme, ako sa ho zbaviť. Už vieme, že `x shr 32` má v ľavej polovici nuly. My by sme teraz chceli spraviť to isté aj s druhou polovicou registra – teda spodných 32 bitov nechať na mieste a horných 32 nastaviť na nuly. Toto vieme spraviť viacerými spôsobmi. Jedna možnosť je posunúť obsah tohto registra najskôr o 32 pozícií doľava (čím zahodíme ľavých 32 bitov pôvodnej hodnoty) a potom naspäť o 32 pozícií doprava (čím ich nahradíme nulami). Univerzálnejšou technikou na realizáciu operácie „tieto bity si chcem nechať a tamtie chcem zahodiť“ je použitie `and`. Vyrobíme si hodnotu, ktorá má jednotky na pozíciách, ktoré chceme, a nuly na tých, ktoré nechceme. Keď spravíme `and` hodnoty x a takto pripravenej hodnoty (ktorej zvykneme hovoriť „bitová maska“), dostaneme presne to, čo sme chceli.
Hodnotu by sme do programu mohli zadať ako konštantu (pričom zväčša takéto konštanty píšeme priamo v dvojkovej alebo šestnástkovej sústave), vieme si ju ale aj ľahko vypočítať: je to hodnota $2^{32} - 1$, ktorú vieme vyjadriť napríklad ako `(1 shl 32) - 1`.
Iná verzia vyššie uvedeného programu by teda mohla vyzerať nasledovne:

```
lava = x shr 32 ;     prava = x and ((1 shl 32) - 1) ;     y = lava xor prava
lava = y shr 16 ;     prava = x and ((1 shl 16) - 1) ;     y = lava xor prava
lava = y shr  8 ;     prava = x and ((1 shl  8) - 1) ;     y = lava xor prava
lava = y shr  4 ;     prava = x and ((1 shl  4) - 1) ;     y = lava xor prava
lava = y shr  2 ;     prava = x and ((1 shl  2) - 1) ;     y = lava xor prava
lava = y shr  1 ;     prava = x and ((1 shl  1) - 1) ;     y = lava xor prava
if y == 0 then print('parny') else print('neparny')
```

Riešenie 4.
Vyššie uvedené riešenie vlastne na záver vypočítalo xor všetkých bitov registra x. Vedelo to spraviť rýchlo, lebo každou operáciou `xor` sme naraz prexorovali veľa dvojíc bitov.
Inou možnosťou, ako dosiahnuť to isté, je začať od susedných dvojíc bitov: bity si rozdelíme na 32 po sebe idúcich dvojíc, každú dvojicu prexorujeme, výsledky si rozdelíme na 16 po sebe idúcich dvojíc, každú dvojicu prexorujeme, a tak ďalej, až kým prexorovaním posledných dvoch výsledkov nedostaneme hľadaný výstup. V programe by tento postup vyzeral nasledovne:

```
y = x xor (x shr 1)
y = y xor (y shr 2)
y = y xor (y shr 4)
y = y xor (y shr 8)
y = y xor (y shr 16)
y = y xor (y shr 32)
if y mod 2 == 0 then print('parny') else print('neparny')
```

Zhrnutie.
Riešenie 1 potrebovalo na výpočet parity počtu jednotiek počet krokov priamo úmerný hodnote B. Počet krokov riešenia 2 bol priamo úmerný skutočnému počtu jednotiek – v najhoršom prípade bol teda stále priamo úmerný B, ale častokrát mohlo toto riešenie byť rýchlejšie.
Riešenie 3 aj riešenie 4 mali počet krokov priamo úmerný dvojkovému logaritmu čísla B. (Ak by sme napríklad B zväčšili zo 64 na 128, prvému riešeniu by pribudlo 64 iterácií, zatiaľ čo tretiemu aj štvrtému riešeniu by pribudol len jeden krok.)

TRIDSIATY ŠIESTY ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek, Andrej Korman
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2021