Informácie a pravidlá

Pre koho je súťaž určená?

Do **kategórie B** sa smú zapojiť len tí žiaci základných a stredných škôl, ktorí ešte ani v tomto, ani v nasledujúcom školskom roku nebudú končiť strednú školu.
Do **kategórie A** sa môžu zapojiť všetci žiaci (základných aj) stredných škôl.

Odovzdávanie riešení domáceho kola

Riešitelia domáceho kola odovzdávajú riešenia sami, v elektronickej podobe, a to priamo na stránke olympiády: http://oi.sk/. Odovzdávanie riešení bude spustené niekedy v septembri.

Riešenia kategórie A je potrebné odovzdať najneskôr **15. novembra 2020**.
Riešenia kategórie B je potrebné odovzdať najneskôr **30. novembra 2020**.

Priebeh súťaže

Za každú úlohu domáceho kola sa dá získať od 0 do 10 bodov. Na základe bodov domáceho kola stanoví Slovenská komisia OI (SK OI) pre každú kategóriu bodovú hranicu potrebnú na postup do **krajského kola**. Očakávame, že táto hranica bude približne rovná **tretine maximálneho počtu bodov**.

V krajskom kole riešitelia riešia štyri teoretické úlohy, ktoré môžu tematicky nadväzovať na úlohy domáceho kola. V kategórii B súťaž týmto kolom končí.

V kategórii A je približne najlepších 30 riešiteľov krajského kola (podľa počtu bodov, bez ohľadu na kraj, v ktorom súťažili) pozvaných do **celoštátneho kola**. V celoštátnom kole účastníci prvý deň riešia teoretické a druhý deň praktické úlohy. Najlepší riešitelia sú vyhlásení za víťazov. Približne desať najlepších riešiteľov následne SK OI pozve na týždňové výberové sústredenie. Podľa jeho výsledkov SK OI vyberie družstvá pre Medzinárodnú olympiádu v informatike (IOI) a Stredoeurópsku olympiádu v informatike (CEOI).

Ako majú vyzerať riešenia úloh?

V praktických úlohách je vašou úlohou vytvoriť program, ktorý bude riešiť zadanú úlohu. Program musí byť v prvom rade korektný a funkčný, v druhom rade sa snažte aby bol čo najefektívnejší.

V kategórii B môžete použiť ľubovoľný programovací jazyk.

V kategórii A musíte riešenia praktických úloh písať v jednom z podporovaných jazykov (napr. C++, Pascal alebo Java). Odovzdaný program bude automaticky otestovaný na viacerých vopred pripravených testovacích vstupoch. Podľa toho, na koľko z nich dá správnu odpoveď, vám budú pridelené body. Výsledok testovania sa dozviete krátko po odovzdaní. Ak váš program nezíska plný počet bodov, budete ho môcť vylepšiť a odovzdať znova, až do uplynutia termínu na odovzdávanie.

Presný popis, ako majú vyzerať riešenia praktických úloh (napr. realizáciu vstupu a výstupu), nájdete na webstránke, kde ich budete odovzdávať.

Ak nie je v zadaní povedané ináč, riešenia teoretických úloh musia v prvom rade obsahovať **podrobný slovný popis použitého algoritmu, zdôvodnenie jeho správnosti** a diskusiu o efektivite zvoleného riešenia (t. j. posúdenie časových a pamäťových nárokov programu). Na záver riešenia uveďte program. Ak používate v programe netriviálne algoritmy alebo dátové štruktúry (napr. rôzne súčasti STL v C++), súčasťou popisu algoritmu musí byť dostatočný popis ich implementácie.

Usporiadateľ súťaže

Olympiádu v informatike (OI) vyhlasuje *Ministerstvo školstva SR* v spolupráci so *Slovenskou informatickou spoločnosťou* (odborným garantom súťaže) a *Slovenskou komisiou Olympiády v informatike*. Súťaž organizuje *Slovenská komisia OI* a v jednotlivých krajoch ju riadia *krajské komisie OI*. Na jednotlivých školách ju zaisťujú učitelia informatiky. Celoštátne kolo OI, tlač materiálov a ich distribúciu po organizačnej stránke zabezpečuje IUVENTA v tesnej súčinnosti so Slovenskou komisiou OI.

Covid-19

Krajské kolo súťaže je plánované na 19. januára 2021 a celoštátne kolo na 24.-27. marca 2021. V súčasnosti ešte nevieme zaručiť, či tieto kolá prebehnú tradičnou prezenčnou formou.

Slovenská komisia OI dôsledne sleduje aktuálny vývoj epidemiologickej situácie a v prípade potreby mu prispôsobí organizáciu súťaže. Domáce kolo súťaže určite prebehne v tradičnej podobe, keďže naň v súčasnosti platné opatrenia nemajú žiaden vplyv. O presnej forme a spôsobe organizácie ďalších kôl súťaže budú postupujúci súťažiaci informovaní priebežne.

B-I-1 Korytnačie preteky

Medzi matfyzákmi sú Korytnačie preteky veľmi obľúbenou spoločenskou hrou a to napriek tomu, že táto hra je primárne určená pre päťročné deti. Pri prvom opise je totiž naozaj veľmi jednoduchá. Hrací plán pozostáva z niekoľkých za sebou idúcich políčok a cieľom hráčov je dostať korytnačku ich farby čo najrýchlejšie do cieľa. Korytnačky sa však môžu hýbať iba pomocou kariet, ktoré hráči na striedačku vykladajú. Každá karta určuje farbu korytnačky a počet políčok, o ktoré sa má korytnačka posunúť (tento počet môže byť aj záporný, vtedy ide korytnačka dozadu).

Toto znie pomerne priamočiaro. Akurát, hráči poznajú iba farbu svojej korytnačky, nevedia za ktorú korytnačku hrajú súperi a na ruke môžu mať karty ľubovoľnej farby. Navyše, korytnačky majú vskutku špecifický spôsob pohybu. Ak je totiž korytnačka presunutá na políčko, na ktorom sú iné korytnačky, nepostaví sa vedľa nich, ale ich kladieme na seba. Všetky korytnačky stojace na tom istom políčku teda vždy tvoria jeden „stĺpik“. Novú korytnačku, prichádzajúcu na políčko, vždy položíme **na vrch stĺpika**. A keď sa niektorá z korytnačiek tvoriacich stĺpik pohne, **zoberie so sebou (presnejšie, na sebe) všetky na nej stojace korytnačky**.

Pri hre naozaj často vznikajú takéto stĺpiky korytnačiek, ktoré sa spoločne presúvajú. Hráči sa snažia takticky držať svoje korytnačky na vrchu týchto kôpok (aby im ich posúvali aj ostatní hráči), a potom vo vhodnej chvíli rýchlo utiecť do cieľa.

Jano a Žaba sú vášniví hráči. Vytvorili si dokonca viacero hracích plánov, v ktorých je čoraz viac políčok aj korytnačiek. Nedávno ich pri hraní prišiel navštíviť Syseľ, drgol však do stola a rozsypal im všetky korytnačky. Jano a Žaba by chceli túto partiu dohrať. Našťastie si pamätajú všetky karty, ktoré postupne zahrali. Teraz by potrebovali rýchlo zistiť, aké bolo rozloženie korytnačiek v momente, keď prišiel Syseľ. Pomôžete im?

Súťažná úloha

Dostanete popis hry, ktorá sa odohrávala na p políčkach s n korytnačkami. Políčka sú očíslované od 1 po p, pričom políčko 1 je začiatok a políčko p je cieľ. Korytnačky sú očíslované od 1 po n. Na začiatku všetky korytnačky tvoria jeden stĺpik na políčku 1. Poradie korytnačiek v štartovacom stĺpiku dostanete zadané.

Následne sa odohralo q kariet. Každá karta určuje korytnačku, ktorá sa pohla, a počet políčok, o ktoré sa posunula dopredu. Toto číslo môže byť kladné aj záporné.

Vašou úlohou je zistiť, ako vyzerá hrací plán po odohratí všetkých zadaných kariet.

Formát vstupu a výstupu

Od nás dostanete päť vstupných súborov označených `1.txt` až `5.txt`. Každý z nich má nasledujúci formát.

V prvom riadku vstupu sú tri čísla n, p a q – počet korytnačiek, počet políčok a počet zahraných kariet.

V druhom riadku je n medzerou oddelených čísel udávajúcich začiatočné postavenie korytnačiek na políčku 1. Prvé číslo určuje číslo korytnačky na samom spodku stĺpika, druhé je číslo korytnačky priamo nad ňou a tak

ďalej až po posledné číslo označujúce najvyššie položenú korytnačku. (Všetky tieto čísla sú od 1 po n a sú navzájom rôzne.)
Nasleduje q riadkov popisujúce karty v poradí, v akom boli zahrané. Každý z týchto riadkov sa skladá z troch čísel k_i, p_i a x_i. Tento riadok udáva, že korytnačka k_i pri zahratí tejto karty stála na políčku p_i a posunula sa o x_i políčok dopredu. (Záporné x_i teda v skutočnosti označuje pohyb dozadu.)
Zadaný vstup je korektný. Môžete teda predpokladať, že korytnačky skončia po každom pohybe na niektorom z políčok 1 až p a korytnačka k_i pri zahratí i-tej karty naozaj stojí na políčku p_i.

Na výstup vypíšte p riadkov. V i-tom z týchto riadkov vypíšte popis korytnačiek na i-tom políčku. Prvé číslo riadku (označme ho x_i) udáva počet korytnačiek na tomto políčku. Zvyšok riadku má tvoriť ďalších x_i čísel: čísla korytnačiek na tomto políčku v poradí od spodku stĺpika po jeho vrch.

Obmedzenia veľkosti vstupov

V jednotlivých vstupoch sa čísla n, p a q rovnajú hodnotám uvedeným v tabuľke. Okrem toho platí, že **vo vstupe 3.txt** každý presun korytnačky smeruje na prázdne políčko. V tomto vstupe sa teda nestane, že by sa korytnačka postavila na chrbát inej (stále však môžu cestovať v kôpke určenej z počiatočného rozostavenia).

	1.txt	**2.txt**	**3.txt**	**4.txt**	**5.txt**
n	5	100	10 000	100 000	300 000
p	20	1000	10 000	1000	10 000
q	25	1000	5 000 000	2 000 000	5 000 000

Odovzdávanie riešení

Toto je praktická úloha. Napíšte **v ľubovoľnom programovacom jazyku** program, ktorý ju rieši.
Zo stránky `http://oi.sk/` stiahnite ZIP archív obsahujúci 5 testovacích vstupov, nazvaných `1.txt` až `5.txt`.
Vyrobte k čo najviac vstupom správne výstupy a uložte ich do súborov `sol1.txt` až `sol5.txt`.
Odovzdajte ZIP archív obsahujúci **zdrojový kód vášho programu** a tieto výstupné súbory.
Za každý správny výstupný súbor získate 2 body.

Príklad

vstup

```
5 7 4
2 4 1 3 5
1 1 3
3 4 -3
4 1 5
1 4 2
```

výstup

```
1 2
0
0
0
0
4 4 3 5 1
0
```

Máme päť korytnačiek, sedem políčok a štyri zahrané karty. Na začiatku sú na políčku 1 korytnačky v poradí 2, 4, 1, 3, 5 zdola hore. Potom sa postupne udeje nasledovné:

- *Korytnačka 1 sa z políčka 1 pohne o 3 dopredu, na políčko 4. Na chrbte so sebou odnesie aj korytnačky 3 a 5.*
- *Korytnačka 3 sa z políčka 4 pohne o −3 dopredu, teda naspäť na políčko 1. Na chrbte so sebou odnesie aj korytnačku 5. Na políčku 1 je teraz stĺpik 2, 4, 3, 5, zatiaľ čo na políčku 4 je samotná korytnačka 1.*
- *Korytnačka 4 sa z políčka 1 pohne o 5 dopredu. Na políčku 1 zostane len korytnačka 2, ostatné korytnačky odnesie korytnačka 4 so sebou na políčko 6.*
- *Korytnačka 1 sa zo svojho aktuálneho políčka 4 pohne o 2 dopredu a tam naskočí na vrch stĺpika.*

B-I-2 Trojuholníky

Deti odjakživa obdivovali pravidelné geometrické tvary. Výnimkou nie je ani Tim. Malý Tim rýchlo rastie a momentálne miluje čokolvek trojuholníkové. Najmä síce čokoládu ale okrem toho aj všakovaké skladačky. Keď je s rodičmi v parku, nazbiera všetky možné drievka a halúzky v okolí a potom sa z nich snaží postaviť čo najviac trojuholníkov. Jedného dňa mu už ostali iba drievka dĺžiek 4, 5 a 10 centimetrov. Neznalý trojuholníkového pravidla sa snažil z paličiek poskladať trojuholník. Keď mu to chvíľu nešlo tak sa bezradne rozplakal a rodičia ho dokázali upokojiť až teplým kakaom. Jeho rodičia by sa najbližšie radi vyhli tomuto problému a malému Timovi nechali na kôpke iba paličky z ktorých vie poskladať trojuholník. Viete im pomôcť takéto paličky vybrať?

Súťažná úloha

Na kope sa nachádza $n \geq 3$ drievok, ktorých dĺžky sú $l_0, l_1, \ldots, l_{n-1}$. Zaujíma nás veľkosť najväčšej podmnožiny drievok, v ktorej každé tri drievka tvoria trojuholník.

Formát vstupu a výstupu

Dostanete od nás 5 vstupných súborov, označených `1.txt` až `5.txt`. Každý z nich obsahuje práve 2 riadky a má nasledujúci formát: V prvom riadku je číslo n udávajúce počet paličiek. V druhom riadku sa nachádzajú čísla $l_0, l_1, \ldots, l_{n-1}$.
Na výstup vypíšte jeden riadok a v ňom jediné číslo. Vyjadrovať má maximálny počet drievok zo vstupu, ktoré medzi sebou v ľubovolnej trojici tvoria trojuholník.
(Môžete predpokladať, že v každom z použitých vstupov existuje aspoň jedna trojica paličiek, z ktorých sa dá postaviť trojuholník.)

Odovzdávanie riešení

Toto je praktická úloha. Napíšte **v ľubovoľnom programovacom jazyku** program, ktorý ju rieši.
Zo stránky `http://oi.sk/` stiahnite ZIP archív obsahujúci 5 testovacích vstupov, nazvaných `1.txt` až `5.txt`.
Vyrobte k čo najviac vstupom správne výstupy a uložte ich do súborov `sol1.txt` až `sol5.txt`.
Odovzdajte ZIP archív obsahujúci **zdrojový kód vášho programu** a tieto výstupné súbory.
Za každý správny výstupný súbor získate 2 body.

Veľkosti vstupov

V jednotlivých vstupoch má hodnota n postupne hodnoty 15, 30, 10 000, 1 000 000 a 1 000 000. Hodnoty $l_0, l_1, \ldots, l_{n-1}$ sú v rozsahu bežnej celočíselnej premennej `int` a sú kladné.

Príklad

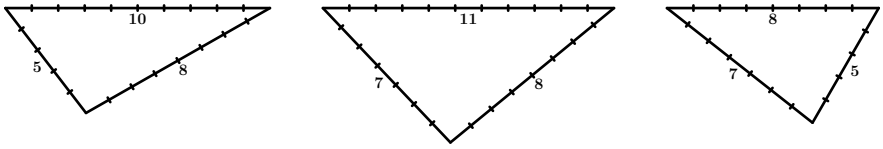
vstup

```
6
4 5 10 8 7 11
```

výstup

```
5
```

Všetky drievka nemôžeme vybrať – napr. z trojice drievok s dĺžkami 4, 7, 11 sa trojuholník (tesne) spraviť nedá. Najlepšie riešenie má teda nanajvýš päť drievok. No a ľahko overíme, že pre päticu drievok dĺžiek 5, 10, 8, 7 a 11 už platí, že akákoľvek trojica tvorí trojuholník. Tri z nich sú na obrázku nižšie.



B-I-3 Delta Nílu

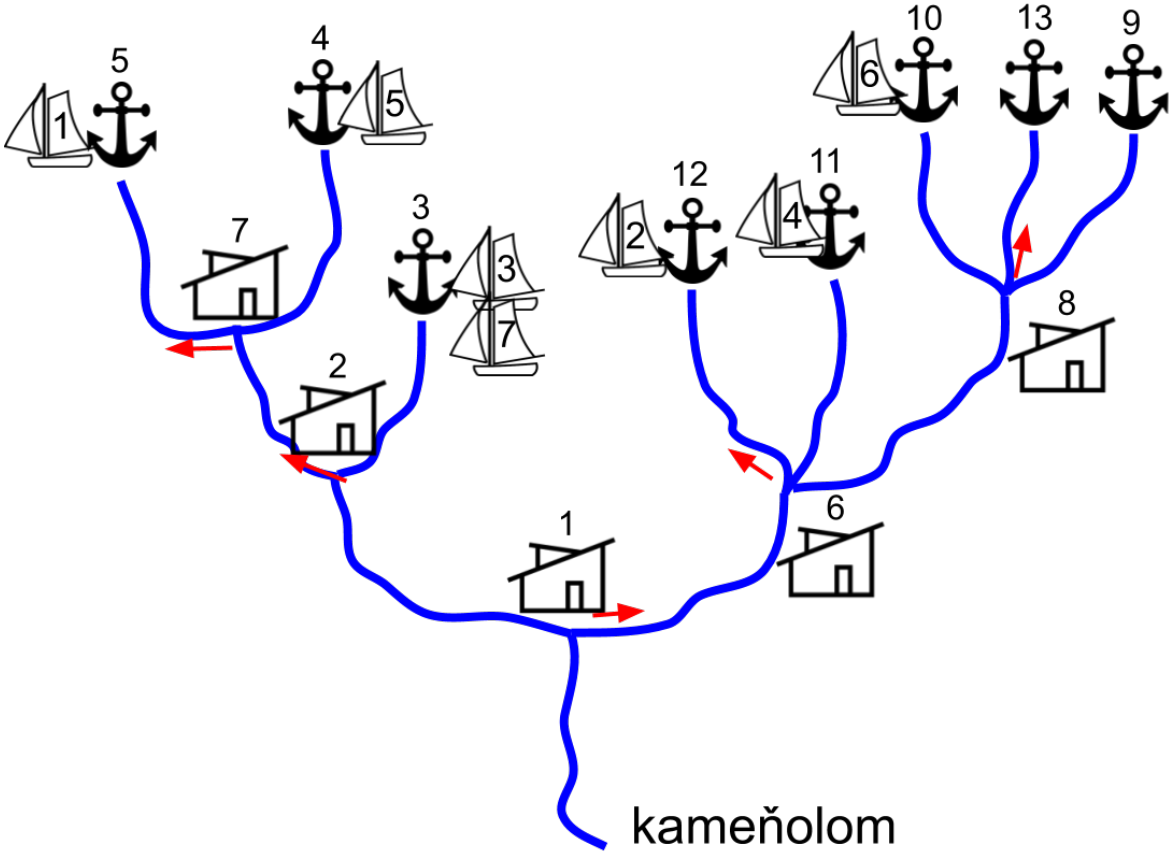
Toto je teoretická úloha. Pomocou webového rozhrania odovzdajte súbor vo formáte PDF, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách.

Na stavbu egyptských miest v okolí delty Nílu bolo potrebné veľké množstvo kameňa. Ten sa však ťažil vo vzdialených lomoch, na jeho prepravu sa preto využívala práve najdlhšia rieka sveta. Pri lome naložili loďku kameňom a tú poslali dole prúdom.

Egypťania však museli vyriešiť niekoľko logistických problémov, ktoré spôsobovalo vetvenie rieky. Na Níle je totiž pomerne bežné, že rieka sa rozdelí na viacero samostatných tokov. Tieto toky sa však už nikdy opäť nespoja, postupné vetvenie teda vytvára ďalšie a ďalšie toky, ktoré dodávajú delte jej známy vzhľad.

Na konci každého toku je prístav, do ktorého treba dostať kameň. Ak by však plavba lodiek nemala žiadne pravidlá a vetvy by si vyberala náhodne, mohlo by sa stať, že do niektorého prístavu by sa kameň nikdy nedostal a v inom by ho zase bolo priveľa. Pri každom vetvení preto egypťania postavili kontrolnú stanicu, ktorá príchodzím loďkám určuje, ktorým tokom vetvenia sa majú vybrať. Aby sa zaručila rovnomernosť, kontrolná stanica cyklicky strieda smery, ktorými loďku posiela. Napríklad, ak sa pri stanici rieka delí na tri samostatné toky, prvá loďka, ktorá k nej príde ide prvým tokom, druhá druhým, tretia tretím, štvrtá opäť prvým atď. Všetky stanice fungujú rovnakým spôsobom a nezávisle od seba.

Loďky s kameňom si môžeme očíslovať v poradí, v ktorom boli vypustené z kameňolomu a následne sledovať, do ktorého prístavu dorazili. Ich plavba je totiž jednoznačne určená. Na obrázku nižšie je uvedený príklad rieky a niekoľkých lodiek, ktoré ňou preplávali. Kotva označuje prístav, domček označuje kontrolnú stanicu. Loďka s číslom i vyplávala z kameňolomu i-ta v poradí a skončila v príslušnom prístave. Červené šípky označujú tok, ktorým kontrolná stanica pošle ďalšiu loďku. Na začiatku ukazovali všetky červené šípky na najľavejší tok a postupovali zľava doprava.



Súťažná úloha

Kapitán Peter velí loďke, ktorá z kameňolomu vyrazí ako k-tá v poradí. Síce dobre pozná každú kontrolnú stanicu a prístav na delte Nílu, nie je si ale istý, v ktorom prístave skončí. Pomôžte mu a on sa vám odvďačí dvoma zlatými talentami.

Formát vstupu a výstupu

Pre jednoduchosť bude *objekt* označovať aj kontrolnú stanicu aj prístav. Na Níle je dokopy n objektov, každý z nich má priradené unikátne číslo od 1 po n.
V prvom riadku vstupu sú zadané dve celé čísla n a k – počet objektov na rieke a poradie lode kapitána Petra. Nasleduje n riadkov, i-ty z nich popisuje objekt označený číslom i. Ak je objekt i prístav, v riadku je zadaná hodnota -1. Ak je objekt i kontrolná stanica, v riadku je zadaných niekoľko čísel. Prvé číslo p_i ($p_i \geq 2$) udáva počet vetiev, ktoré vznikajú pri tejto stanici. Za týmto číslom je zadaných p_i hodnôt $o_1, o_2, \ldots, o_{p_i}$, ktoré udávajú číslo najbližšieho objektu, na ktorý loďka narazí, keď sa vydá príslušnou vetvou. Toto poradie zároveň určuje, akým spôsobom kontrolná stanica posiela lode ďalej: prvú pošle smerom k objektu o_1, druhú smerom k o_2 a tak ďalej. Po objekte o_{p_i} a následne opäť nasleduje objekt o_1.
Číslo 1 má objekt, ktorý je najbližšie ku kameňolomu. Sem teda postupne priplávajú všetky lode.
Keďže sa jednotlivé vetvy Nílu v delte nikdy nespájajú, platí, že všetky čísla 2 až n (teda s výnimkou 1) sa na vstupe objavia práve raz ako niektorá z hodnôt o_i.

Na výstup vypíšte číslo prístavu, do ktorého pripláva k-ta loď poslaná z kameňolomu.

Obmedzenia a hodnotenie

Plných 10 bodov môžu získať riešenia, ktoré efektívne vyriešia ľubovoľný vstup, v ktorom je $n \leq 10^6$ a $k \leq 10^9$.
Za riešenia, ktoré efektívne vyriešia ľubovoľný vstup s $n \leq 1000$ a $k \leq 1000$ môžete získať najviac 5 bodov.
Za ľubovoľné správne riešenie je možné získať aspoň 3 body.

Príklady

vstup

```
13 7
2 2 6
2 7 3
-1
-1
-1
3 12 11 8
2 5 4
3 10 13 9
-1
-1
-1
-1
-1
```

výstup

```
3
```

Vstup zodpovedá riekam z obrázku vyššie. Loďka číslo 7 skončí v prístave číslo 3.

B-I-4 Bitové operácie

Toto je teoretická úloha. Pomocou webového rozhrania odovzdajte súbor vo formáte PDF, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách. K tejto úlohe patrí študijný text uvedený na nasledujúcich stranách. Odporúčame najskôr prečítať ten a až potom sa vrátiť k samotným súťažným úlohám.

Podúloha A (3 body). V registri x je vstupné číslo. Napíšte program, ktorý *v konštantnom čase* zistí, či je x mocninou dvojky. (Pozor na špeciálny prípad: nula nie je mocninou dvojky.)
Podúloha B (3 body). V registri x je vstupné číslo. Napíšte program, ktorý *v konštantnom čase* zistí, či sa v registri x niekde nachádzajú tri jednotkové bity vedľa seba.
Podúloha C (4 body). V registri x je vstupné číslo, pričom je zaručené, že je nenulové. Napíšte program, ktorý vypočíta, akou najväčšou mocninou dvojky je toto číslo deliteľné. 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 2019