Brojni sistemi

Kako se u ovoj lekciji (globalno) bavimo matematičkim algoritmima sa naglaskom na osnovnu teoriju brojeva, jako je korisno za početak pomenuti najčešće korišćene tipove podataka i aritmetičke operacije. Uvek je pre izrade zadatka potrebno analizirati ograničenja u zadatku i na osnovu tih analiza odrediti koje ćemo tipove podataka koristiti.

1. Kada radimo sa celim brojevima čija apsolutna vrednost ne prelazi $2^{31} \approx 2 \cdot 10^9$, treba koristiti 32-bitni tip podataka - $\boldsymbol{longint}$ (Pascal) odnosno tip $\boldsymbol{int}$ (C/C++).
2. Kada radimo sa većim celim brojevima, čija aposolutna vrednost ne prelazi $2^{63} \approx 8 \cdot 10^{18}$ treba koristiti 64-bitni tip podataka $\boldsymbol{int64}$ (Pacal) odnosno tip $\boldsymbol{long\ long}$ (C/C++).
3. Kada radimo sa realnim brojevima, najbolje je uvek koristiti 64-bitni tip podataka $\boldsymbol{double}$ (i Pascal i C/C++) jer je on mnogo precizniji od tipa $real$ (Pascal) i tipa $float$ (C/C++).

Osim brojeva koristimo standardne aritmetičke operacije. Kako se bavimo teorijom brojeva, od najvećeg interesa su nam celobrojno deljenje (operacija $\boldsymbol{div}$ u Pascal-u, odnosno operacija / u C/C++-u) i ostatak pri deljenju (operacija $\boldsymbol{mod}$ u Pascal-u, odnosno operacija **%** u C/C++-u). Ove operacije su "inspirisane" sledećom dobro poznatom teoremom:

Teorema 1 (Teorema o deljenju sa ostatkom). Svaki ceo broj a se uz pomoć proizvoljnog prirodnog broja b može na **jedinstven način** prikazati u obliku

$$a = bq + r, \qquad 0 \le r < b,$$

gde su q i r celi brojevi .

Ako su $a \ge 0$ i $b > 0$ celi brojevi, tada je, koristeći oznake iz prethodne teoreme, $q = a\ div\ b$ i $r = a\ mod\ b$. Broj r se naziva i **ostatak broja $\boldsymbol{a}$ po modulu $\boldsymbol{b}$**. Proveru da li je broj a deljiv brojem b vršimo jednostavnim ispitivanjem uslova $a\ mod\ b = 0$. Za pozitivne cele brojeve ove lepo radi, ali koliko je npr. $-17\ div\ 5$ ili $17\ mod - 5$? Ispostavlja se da za negativne vrednosti a i/ili b, rezultat zavisi od kompajlera i **treba izbegavati primenu ovih operacija na negativne brojeve**. Ipak, bez obzira na implementaciju kompajlera, uvek se garantuje da "važi teorema", tj. da je uvek $a = b \cdot (a\ div\ b) + (a\ mod\ b)$.

Operacija "ostatak po modulu" se često koristi u mnogim zadacima. Naime, čest je slučaj da suština zadatka glasi "**Izračunati neki izraz po modulu $\mathbf{M}$**" gde je M dati broj ili jedan od ulaznih podataka (uglavnom važi $M \le 10^9$). Glavni razlog je što tačna vrednost datog izraza može biti jako veliki broj koji ne može stati ni u jedan (prost) tip podataka. Računanje po modulu obezbeđuje da će vrednost izraza biti iz segmenta $[0, M-1]$.

Komentar . U svim lekcijama je bitno shvatiti kada je neki algoritam efikasan (brz, izvršava "dovoljno malo" operacija). Sa složenošću algoritma ćemo se formalno upoznati u nekim od narednih lekcija; za sada je dovoljno znati da izraz "složenost algoritma je $O(nešto)$" znači da taj algoritam izvrši najviše $c \cdot nešto$ operacija gde je c neka konstanta koja ne zavisi od ulaznih podataka.

Vratimo se sada na temu ovog dela lekcije – brojne sisteme.

Teorema o deljenju sa ostatkom se može malo "proširiti":

Teorema 2 . Neka je $A > 1$ prirodan broj. Tada se svaki nenegativan broj X može na **jedinstven način** predstaviti u obliku

$$X = a_0 + a_1 \cdot A + a_2 \cdot A^2 + \cdots + a_n A^n,$$

gde je $n \in N$ i $0 \leq a_i < A$ za svako $i = \overline{0, n}$.

Ova teorema zapravo tvrdi da za sve cele brojeve $A > 1$ i $X \geq 0$ postoji **jedinstven** polinom P (koji zavisi od X) sa celobrojnim koeficijentima iz segmenta $[0, A-1]$ takav da je $P(A) = X$. Npr. neka je $A = 8$. Tada je $534 = 6 + 2 \cdot 8 + 0 \cdot 8^2 + 1 \cdot 8^3$, $100 = 4 + 4 \cdot 8 + 1 \cdot 8^2$, $10 = 2 + 1 \cdot 8$, $1 = 1$ itd. pri čemu su ovo jedinstvene odgovarajuće reprezentacije ovih brojeva.

Zbog jedinstvenosti zapisa, možemo govoriti o **brojnom sistemu sa osnovom (bazom)** $\boldsymbol{A}$: to je sistem u kome korisimo samo cifre $0,1,2 \ldots, A-1$ za zapis prirodnih brojeva i 0, pri čemu je vrednost broja

$$(a_n a_{n-1} \ldots a_1 a_0)_A$$

jednaka

$$a_0 + a_1 \cdot A + a_2 \cdot A^2 + \cdots + a_n A^n.$$

U zapisu $(a_n a_{n-1} \ldots a_1 a_0)_A$, broj A u indeksu označava o kojoj se osnovi (bazi) radi. Koristeći prethodne primere, imamo $534 = (1026)_8$, $100 = (144)_8$, $10 = (12)_8$ i $1 = (1)_8$. Primetimo da smo u zapisu obrnuli redosled koeficijenata i da je koeficijent uz najveći stepen broja A – prvi.

Šta se dešava za $A = 10$? Npr. važi $534 = 4 + 3 \cdot 10 + 5 \cdot 10^2$, tj. $534 = (534)_{10}$. Ovo je bilo očekivano: mi koristimo **dekadni brojni sistem** tj. sistem sa osnovom 10 i ciframa $0,1,2, \ldots ,9$. Svaki broj koji zapisujemo je formalno broj oblika $(a_n a_{n-1} \ldots a_1 a_0)_{10}$ samo što izostavljamo zagrade i indeks zbog jednostavnosti zapisa. Ipak, kada radimo sa sistemima različitim od dekadnog, treba pisati oznaku osnove jer isti zapis (niz cifara) ima različite vrednosti u različitim brojnim sistemima:

$$(34)_5 = 19, \qquad (34)_6 = 22, \qquad (34)_8 = 28, \qquad (34)_{10} = 34 \ldots$$

Napomenimo da je **vrednost** broja ista (nepromenljiva) bez obzira koji brojni sistem koristimo za njegov **zapis** ili **izgovaranje.** Npr. $(100010)_2 = (114)_5 = (42)_8$ su različiti zapisi istog broja 34 (koji smo sada zapisali u dekadnom brojnom sistemu).

Pozabavimo se sada sledećim problemom:

Problem 1. Dat je nenegativan ceo broj u sistemu sa osnovom A. Prebaciti ga u sistem sa osnovom B. Garantuje se da vrednost datog broja nije veća od 10^9.

Dakle, dati su brojevi A i B, kao i niz $a_0, a_1, \ldots, a_n$ – zapis nekog broja u sistemu sa osnovom A. Potrebno je zapisati taj broj u sistemu sa osnovom B, tj. izračunati m i odgovarajući niz $b_0, b_1 \ldots b_m$. Npr. $A = 2, B = 5$ i $a_0 = 0, a_1 = 1, a_2 = 0, a_3 = 0, a_4 = 0, a_5 = 1$. Ovaj problem ćemo podeliti na dva dela: prvo ćemo izračunati vrednost datog broja (u našem poznatom dekadnom sistemu) a zatim ćemo tu vrednost zapisati u osnovi B.

Prvi deo je jednostavan – samo je potrebno izračunati vrednost $a_0 + a_1 \cdot A + a_2 \cdot A^2 + \cdots + a_n A^n$. Umesto da računamo svaki član posebno, ovo ćemo raditi iterativno, koristeći izraz

$$a_n A^n + a_{n-1} A^{n-1} + \cdots + a_1 x + a_0 = \left(\dots \left((a_n A + a_{n-1}) + a_{n-2} \right) A + \cdots + a_1 \right) A + a_0.$$

Drugim rečima, krećemo od nule i trenutni izraz množimo sa A a zatim mu dodajemo a_i, gde idemo od najvećeg do najmanjeg koeficijenta. Ovo je ilustrovano sledećim pseudokodom

```
================================================================
function Value(int a[], int A) : int

        val ← 0;
        for i ← n downto 0 do
                val ← val * A + a[i];
        end for
        return val;

end function
================================================================
```

Po uslovu problema, vrednost val neće preći 10^9 pa možemo koristiti 32-bitni tip podataka. Za konkretan primer, $val = (100010)_2 = 2 + 2^5 = 34$. Drugi deo nije ništa teži ali nije tako očigledan. Za dato val i B, treba odrediti niz b_i tako da je $val = b_0 + b_1 \cdot B + b_2 \cdot B^2 + \cdots + b_m \cdot B^m$. Primetimo da zapravo važi

$$val = B(b_1 + b_2 B + \cdots b_m B^{m-1}) + b_0.$$

Kako je $0 \le b_0 < B$, b_0 je zapravo ostatak pri deljenju broja val brojem B. Sa druge strane, celobrojni količnik je jednak

$$val\ \boldsymbol{div}\ B = b_1 + b_2 \cdot B + b_3 \cdot B^2 + \cdots + b_m \cdot B^{m-1}.$$

Sada je b_1 ostatak pri deljenju novog broja val brojem B. Ponavljanjem postupka "izračunaj ostatak a zatim podeli sa B" određujemo niz b član po član sve dok val ne postane nula. Ovaj postupak je prikazan u sledećem pseudokodu.

```
================================================================
function Digits(int value, int B) : int[]

        i ← -1;
        while (value > 0) do
                i ← i + 1;
                b[i] ← value mod B;
                value ← value div B;
        end while
        return b[];

end function
================================================================
```

Na konkretnom primeru dobijamo $b_0 = 34\ mod\ 5 = 4$. Kako je $34\ div\ 5 = 6$, $b_1 = 6\ mod\ 5 = 1$. Kako je $6\ div\ 5 = 1$, važi $b_2 = 1\ mod\ 5 = 1$. Zbog $1\ div\ 5 = 0$, ovde algoritam prestaje. Složenost funkcije $Value$ je $O(n)$ dok je složenost funkcije $Digits$ $O(m)$ gde su n i m, redom, broj cifara datog broja u sustemu sa osnovama A i B. Ove funkcije se mogu koristiti potpuno nezavisno, npr. kada je potrebno prikazati broj x u binarnom brojnom sistemu, dovoljno je pozvati funkciju $Digits(x, 2)$.

Problem 2. Dat je broj x nizom cifara dužine $n \leq 10^6$. Odrediti ostatak pri deljenju ovog broja brojem $M \leq 10^9$.

Jasno, eksplicitno računanje vrednosti broja x ne dolazi u obzir jer je previše veliki. Međutim, to nije ni potrebno. Ukoliko je a niz cifara broja x, tada možemo pozvati funkciju $Value(a, 10)$ pri čemu ćemo liniju 04 ove funkcije zameniti sa "$val \leftarrow (val * 10 + a[i]) \; mod \; M$". U tom slučaju mi jednostavno računamo vrednost polinoma $a_0 + a_1 10 + \cdots + a_{n-1} 10^{n-1}$ po modulu M, pri čemu "modujemo" **u svakom koraku**. Složenost ovog algoritma je $O(n)$. Međutim, ovde treba biti oprezan! Iako je $M \leq 10^9$ a samim tim i krajnje rešenje $val < M \leq 10^9$, za promenljivu val je **potrebno koristiti 64-bitni tip podataka** jer vrednost $val * 10 + a[i]$ (pre primene operacije mod) može ispasti iz opsega 32-bitnih brojeva.

Napomena 1. Treba napomenuti da je jedan od najčešćih uzroka grešaka prilikom korišćenja tipova podataka činjenica da učenici/takmičari prilikom prvog susreta sa Pascal-om "nauče" da se za cele brojeve koristi tip $integer$ a za realne tip $real$ ne razumevajući šta znači opseg promenljivih tj. da se sa tipom $integer$ može raditi samo sa brojevima reda veličine 30.000 kao i da tip $real$ nije dovoljno precizan. Sa druge strane, učenici koji rade u C/C++-u ponekad koriste tip promenljivih $long$ za koji očekuju da je 64-bitni dok to zavisi od kompajlera/operativnog sistema. Dobra preporuka je da se, prilikom takmičarskog programiranja, **treba držati boldovanih tipova podataka sa početka teksta kada se radi sa celim ili realnim brojevima**. Takođe, pogledati ovaj tekst za detaljniju analizu klasičnih takmičarskih grešaka.

Napomena 2. Prilikom analize složenosti, dobro je znati da su "najbrže" aritmetičke operacije – sabiranje, oduzimanje i poređenje. Računanje apsolutne vrednosti je oko 2 puta sporije dok je množenje 4 puta sporije. Operacije div i mod su oko 10 puta sporije od sabiranja dok su korenovanje i trigonometrijske funkcije oko $30 - 80$ puta sporije. Takođe, rad sa realnim brojevima je mnogo sporiji nego rad sa celim brojevima; sa druge strane, operacije nad 64-bitnim brojevima su sporije nego operacija nad 32-bitnim, što između ostalog znači da nije pametno (i zbog vremena i zbog memorije) uvek koristiti 64-bitne tipove.

Napomena 3. Osnove brojnih sistema mogu (naravno) biti i brojevi veći od 10. U tom slučaju je potrebno ili koristiti dodatne simbole ili odvajati brojeve da bi zapis bio pregledan jer tada više nemamo "cifre" nego "brojeve". Npr. prilikom korišćenja heksadecimalnog brojnog sistema $(A = 16)$ umesto "cifara" $10, 11, 12, 13, 14, 15$ se koriste slova A, B, C, D, E, F. Na ovaj način se izbegava zapis $180 = (11{,}4)_{16}$ (potreban je zarez da ne bismo pomešali 11 i 4 sa 114) već se koristi $180 = (B4)_{16}$.