Matematička formalizacija algoritama

Marko Horvat

Jeste li ikad poželjeli zabilježiti neki zanimljiv algoritam na način koji ga ne bi ograničavao na neki određeni programski jezik, nego biste ga mogli svuda jednako lako implementirati? Ako jeste, dobra vijest: *svaki* algoritam moguće je zapisati pomoću postojećih ili u trenutku pisanja definiranih matematičkih simbola i operacija. Ali prije nego prijeđemo na stvar, vratimo se malo u povijest kako bismo mogli utvrditi podrijetlo i značenje te riječi.

Naziv *algoritam* dolazi od imena poznatog perzijskog matematičara zvanog Abū'Abd Allāh Muhammad ibn Mūsā al-Khwārizmī, autora proslavljenog arapskog teksta *Kitāb al-jabr wa'l-muqābala* (*Pravila popunjavanja i izjednačavanja*), iz čijeg naslova slijedi riječ *algebra*. Riječ *algoritam* je kroz stoljeća poprimala mnoga različita značenja, a sredinom 20. stoljeća se najčešće povezivala s Euklidovim algoritmom za pronalaženje najveće zajedničke mjere dvaju brojeva iz njegove knjige *Elementi*, svezak VII. Navedeni algoritam glasi ovako:

Euklidov algoritam (1)**.** Za dva pozitivna cijela broja m i n nađi njihovu najveću zajedničku mjeru, najveći pozitivni cijeli broj koji dijeli i m i n.

Korak 1. [Nađi ostatak.] Podijeli m sa n i neka r bude ostatak. (Dobit ćemo $0 \leq r < n$.)

Korak 2. Ako je $r = 0$, algoritam je završen; n je odgovor.

Korak 3. Postavi $m \leftarrow n$, $n \leftarrow r$ i vrati se na prvi korak. □

Pokušajmo sada izvesti ovaj algoritam. Uzmimo, na primjer, brojeve $m = 119$ i $n = 544$. Počinjemo s prvim korakom. Podijeliti m i n sad će biti možda prejednostavno jer je količnik 0, a ostatak 119. Dakle, $r \leftarrow 119$. U drugom koraku ništa se ne događa jer je $r \neq 0$. U trećem koraku postavljamo $m \leftarrow 544$, $n \leftarrow 119$. Jasno je da ako je $m < n$, onda će količnik u prvom koraku uvijek biti 0 i algoritam će zamijeniti m i n u tri koraka. Učinimo to u jednom koraku koji možemo dodati prije prvog:

Korak 0. [Osiguraj $m \geq n$.] Ako je $m < n$, zamijeni $m \leftrightarrow n$.

Ovo ne uzrokuje nikakvu veliku razliku u algoritmu, osim što ga malo produljuje, te povećava brzinu njegova izvođenja za otprilike polovicu slučajeva.

Natrag na prvom koraku, vidimo da je $\frac{544}{119} = 4\frac{68}{119}$, pa je $r \leftarrow 68$. Drugi korak je ponovno nepotreban, a u trećem postavimo $m \leftarrow 119$ i $n \leftarrow 68$. Sljedeći prolaz postavlja $r \leftarrow 51$, te $m \leftarrow 68$ i $n \leftarrow 51$. Dalje je $r \leftarrow 17$, $m \leftarrow 51$ i $n \leftarrow 17$. Konačno, kad 51 podijelimo sa 17, dobivamo $r \leftarrow 0$, pa u drugom koraku algoritam završava. Najveći zajednički djelitelj brojeva 119 i 544 je 17.

Osim što algoritam sadrži konačan broj pravila koja daju redoslijed operacija za rješavanje određenog problema, on mora imati pet važnih svojstava:

1. *Konačnost.* Algoritam uvijek mora biti konačan, tj. završiti nakon nekog broja koraka. Euklidov algoritam uvijek zadovoljava ovo pravilo, jer je nakon prvog koraka uvijek $r < n$, pa ako je $r \neq 0$, vrijednost od n se *smanjuje* sljedeći put u prvom koraku. Iako broj koraka može biti i nekoliko milijuna za vrlo velike pozitivne cijele brojeve, on će uvijek biti konačan. (Postupak koji ima sve karakteristike algoritma osim konačnosti naziva se *računska metoda*, koja prevedena u eksplicitne instrukcije nekog programskog jezika čini *program*.)

2. *Određenost.* Svaki korak algoritma mora biti precizno definiran; postupci koji će se izvesti moraju biti rigorozno i jednoznačno određeni za svaki mogući slučaj. Tako je u Euklidovom algoritmu potrebno naglasiti da su m i n obavezno *pozitivni cijeli brojevi*. Kad to ne bismo istakli, ne bi bilo jasno kako izračunati ostatak dijeljenja -8 i $-\pi$ ili $\frac{59}{13}$ i 0, itd.

3. *Ulaz.* Algoritam ima 0 ili više *ulaza*: vrijednosti koje se unose prije nego izvršavanje počne, ili dinamički za vrijeme izvođenja algoritma. U Euklidovom algoritmu susrećemo dva ulaza, m i n, uzeta iz skupa pozitivnih cijelih brojeva.

4. *Izlaz.* Algoritam ima jedan ili više *izlaza*: vrijednosti koje su u određenoj relaciji s ulazima. Euklidov algoritam ima jedan izlaz, n u drugom koraku, koji je ujedno i najveća zajednička mjera dvaju ulaza.

5. *Učinkovitost.* Od algoritma se u globalu očekuje učinkovitost, u smislu da njegove operacije moraju biti dovoljno elementarne da se njihovo izvršavanje u principu može prikazati u konačnom vremenu uz pomoć papira i olovke. Euklidov algoritam koristi samo operacije dijeljenja jednog pozitivnog cijelog broja drugim, testiranja je li neki broj jednak nuli, te postavljanja vrijednosti jedne varijable jednakoj vrijednosti druge. Ove operacije su učinkovite jer se cijeli brojevi mogu prikazati na papiru u konačnom vremenu i zato što postoji najmanje jedna metoda za dijeljenje jednog broja drugim.

Pokušajmo usporediti koncept algoritma s onim kulinarskog recepta. Recept uvijek ima svojstva konačnosti (1 sat, 15 minuta), ulaze (jaja, brašno, itd.), te izlaze (splačina), ali mu najčešće nedostaje određenost. Česti su slučajevi u kojima su kuharove upute neodređene: "Dodajte dašak soli." *Dašak* se definira kao *manje od 1/8 čajne žličice* i sol je možda dovoljno precizno definirana, ali gdje bi sol trebalo dodati? Na vrh? Sa strane? Upute poput "kuhajte na laganoj vatri dok se smjesa ne zgusne" ili "zagrijte ulje u maloj tavi" adekvatne su kao objašnjenja izučenom kuharu, ali algoritam mora biti definiran do stupnja na kojem ga čak i računalo može "razumjeti".

U praksi mi želimo algoritme koji su *dobri* u nekom subjektivno-estetskom smislu. Jedan od kriterija će biti *vrijeme algoritma* (vrijeme potrebno da se algoritam izvrši), *portabilnost* (prilagodba algoritma velikom broju različitih programskih jezika i računalnih platformi), jednostavnost i elegancija *implementacije* itd.

Sad kad znamo što je algoritam i kako je definiran, pokušajmo ga strogo matematički formalizirati preko teorije skupova. Definirajmo *računsku metodu* uz pomoć četveročlanog vektora (Q, I, Ω, f) u kojem je Q skup koji sadrži podskupove I i Ω, a f je funkcija kojoj je domena i kodomena skup Q. Četiri vrijednosti Q,I,Ω,f predstavljaju redom : računska stanja, ulaze, izlaze i pravilo pridruživanja. Svaki ulaz x u skupu I definira *računski niz*, $x_0, x_1, x_2, \ldots$, na sljedeći način:

$$x_0 = x \quad i \quad x_{k+1} = f(x_k) \quad za \quad k \geq 0$$

Računski niz *završava u k koraka* ako je k najmanji cijeli broj za koji je x_k u Ω i u tom slučaju kažemo da izlaz x_k slijedi iz x. Vrijedi ponoviti da neke računske metode neće nikad završiti; računska metoda koja završava u konačnom broju koraka za sve $x \in I$ naziva se *algoritam*.

Euklidov algoritam mogli bismo formalizirati na sljedeći način: neka Q bude skup svih monoma (n), svih uređenih parova (m, n) i svih vektora $(m, n, r, 1)$, $(m, n, r, 2)$ i $(m, n, r, 3)$ gdje su m i n pozitivni cijeli brojevi i r je nenegativni cijeli broj, a četvrti broj shvatimo kao broj linije programskog koda, tj. implementacije algoritma u nekom programskom jeziku; neka I bude podskup svih uređenih parova (m, n) i Ω podskup svih monoma (n); neka f bude definiran na sljedeći način:

$$\begin{aligned}
f\big((m,n)\big) &= (m,n,0,1); \quad f\big((n)\big) = (n);\\
f\big((m,n,r,1)\big) &= (m,n,\text{ostatak dijeljenja } m/n, 2);\\
f\big((m,n,r,2)\big) &= \begin{cases} r=0: & (n)\\ r\neq 0: & (m,n,r,3)\end{cases}\\
f\big((m,n,r,3)\big) &= (n,r,r,1)
\end{aligned}$$

Pročimo sada kroz ovako zapisan algoritam koristeći iste one brojeve, 544 i 119:

$$\begin{aligned}
f\big((544,119)\big) &= (544,119,0,1);\\
f\big((544,119,0,1)\big) &= (544,119,68,2);\\
f\big((544,119,68,2)\big) &= (544,119,68,3);\\
f\big((544,119,68,3)\big) &= (119,68,68,1);\\
f\big((119,68,68,1)\big) &= (119,68,51,2);\\
f\big((119,68,51,2)\big) &= (119,68,51,3);\\
f\big((119,68,51,3)\big) &= (68,51,51,1);\\
f\big((68,51,51,1)\big) &= (68,51,17,2);\\
f\big((68,51,17,2)\big) &= (68,51,17,3);\\
f\big((68,51,17,3)\big) &= (51,17,17,1);\\
f\big((51,17,17,1)\big) &= (51,17,0,2);\\
f\big((51,17,0,2)\big) &= (17);\\
f\big((17)\big) &= (17);
\end{aligned}$$

Na kraju smo došli do takozvane *beskonačne petlje*: ako je $f(x) = x$, x je izlaz algoritma.

Sada ćemo pokušati primijeniti ovu metodu formalizacije na još jednom algoritmu kojeg sam se dosjetio eksperimentirajući s petljama u programskom jeziku C. Htio sam napraviti što jednostavniji algoritam sa što zanimljivijim izlazom i tako dobio novi način izračunavanja članova Fibonaccijevog niza koristeći se samo *dvjema* varijablama (ne računajući, naravno, varijablu u kojoj je pohranjen broj do kojeg će se algoritam izvoditi). Navedeni niz se sastoji od brojeva koji su dobiveni zbrajanjem prethodna dva, ako uzmemo da su prva dva člana 0 i 1. Dakle:

$$F_0 = 0,\ F_1 = 1,\ F_k = F_{k-2} + F_{k-1}$$

Dva najčešća algoritma su rekurzivni (izvodi se pozivajući samog sebe, pa je vrijeme algoritma eksponencijalno, te je stoga neupotrebljiv za veće brojeve) i onaj koji točno slijedi definiciju niza (koristi tri varijable). Moju verziju algoritma možemo zapisati na sljedeći način:

$$\begin{aligned}
f\big((n)\big) &= (0,1,n,1,\phi);\\
f\big((a,i,n,1,\varphi)\big) &= \begin{cases} i \le n: & (i-a,i,n,2,\varphi+i)\\ i > n: & (\varphi)\end{cases}\\
f\big((a,i,n,2,\varphi)\big) &= (a,i+a,n,1,\varphi);\\
f\big((\varphi)\big) &= (\varphi)
\end{aligned}$$

Dakle, vidimo da su prva dva argumenta dvije izvršne varijable a i i; treći, n, određuje broj iznad kojeg se Fibonaccijevi brojevi više neće ispisivati; četvrti je broj reda, kao i u prethodnom primjeru; peti je nešto novo: niz u koji se pohranjuju elementi zadanog niza (jer ispis se vrši samo kad algoritam dođe do beskonačne petlje, ali onda je isto tako gotovo njegovo izvršavanje: jasno je da moramo

pohraniti izlazne elemente u niz i ispisati ih sve odjednom). Isprobajmo algoritam do, recimo, 30:

$$\begin{aligned}
f\big((30)\big) &= (0,1,30,1,\phi);\\
f\big((0,1,30,1,\phi)\big) &= (1,1,30,2,[1]);\\
f\big((1,1,30,2,[1])\big) &= (1,2,30,1,[1]);\\
f\big((1,2,30,1,[1])\big) &= (1,2,30,2,[1,2]);\\
f\big((1,2,30,2,[1,2])\big) &= (1,3,30,1,[1,2]);\\
f\big((1,3,30,1,[1,2])\big) &= (2,3,30,2,[1,2,3]);\\
f\big((2,3,30,2,[1,2,3])\big) &= (2,5,30,1,[1,2,3]);\\
f\big((2,5,30,1,[1,2])\big) &= (3,5,30,2,[1,2,3,5]);\\
f\big((3,5,30,2,[1,2,3,5])\big) &= (3,8,30,1,[1,2,3,5]);\\
f\big((3,8,30,1,[1,2,3,5])\big) &= (5,8,30,2,[1,2,3,5,8]);\\
f\big((5,8,30,2,[1,2,3,5,8])\big) &= (5,13,30,1,[1,2,3,5,8]);\\
f\big((5,13,30,1,[1,2,3,5,8])\big) &= (8,13,30,2,[1,2,3,5,8,13]);\\
f\big((8,13,30,2,[1,2,3,5,8,13])\big) &= (8,21,30,1,[1,2,3,5,8,13]);\\
f\big((8,21,30,1,[1,2,3,5,8,13])\big) &= (13,21,30,2,[1,2,3,5,8,13,21]);\\
f\big((13,21,30,2,[1,2,3,5,8,13,21])\big) &= (13,34,30,1,[1,2,3,5,8,13,21]);\\
f\big((13,34,30,1,[1,2,3,5,8,13,21])\big) &= ([1,2,3,5,8,13,21]);\\
f\big(([1,2,3,5,8,13,21])\big) &= ([1,2,3,5,8,13,21]);
\end{aligned}$$

Izgleda da algoritam radi, ali možete li naći njegov opći dokaz? Pošaljite nam ga i nagradit ćemo vas malim iznenađenjem. Isto vrijedi i za svaki algoritam koji formalizirate na opisani način, te implementaciju navedenog algoritma za Fibonaccijev niz. Neka rješenja objavit ćemo u sljedećem broju!