Elegancija i složenost algoritma

Dušan T. Malbaški, *Member IEEE* i Aleksandar D. Kupusinac

***Sadržaj* — U literaturi se često pominje termin *elegancija* algoritma, iako nije jasno objašnjeno šta se pod time podrazumeva. U matematici se može govoriti o eleganciji matematičkog rešenja, pri čemu se pod time najčešće podrazumeva kratkoća. Činjenica je da se rešivi problem može algoritamski rešavati na više načina, a da se dobijena rešenja razlikuju po kratkoći. U ovom radu ćemo pokazati da u opštem slučaju ne postoji veza između elegancije i složenosti algoritma, već naprotiv, može se desiti da fizički kraće rešenje predstavlja uzrok još većoj složenosti algoritma.**

***Ključne reči* — algoritam, elegantnost, složenost, teorija algoritama.**

I. Uvod

SLOŽENOST predstavlja osobinu algoritama koja nije definiciona i nije inherentna, ali je u programerskoj praksi generalno jedna od najvažnijih [1]. Poznato je da se rešivi problemi [2], [3] mogu algoritamski rešavati na više načina, odnosno da se može formulisati više algoritama koji rešavaju dati problem. Svi oni za isti ulaz daju isti izlaz, ali uz različit utrošak resursa izvršioca algoritma. Polazeći od činjenice da današnji izvršioci algoritama su digitalni računari zaključujemo da se algoritmi koji rešavaju isti problem razlikuju po utrošku vremena i memorije računara. Međutim, utrošak vremena i memorije računara nije jednostavno definisati, jer zavise od konkretnog računara. Zbog toga, razmatranje složenosti algoritma se vezuje za apstraktni računar, tačnije za Tjuringovu mašinu [4]. Sada se vreme izvršavanja definiše kao broj koraka Tjuringove mašine od početka do završetka rada, a utrošak memorije kao maksimalni broj zauzetih ćelija memorijske trake u toku izvršenja. Međutim, treba istaći i to da čak i ove apstraktne mere nisu sasvim idealne [5].

Poznato je pravilo o recipročnosti utroška memorije i vremena izvršenja, koje kaže da se u cilju ubrzavanja programa mora žrtvovati memorijski prostor i obrnuto [5]. Ipak u praksi se vremenu pridaje veća važnost, pa ćemo se fokusirati na analizu vremenske složenosti algoritama. Osnova za procenu vremenske složenosti jeste tzv. dimenzija problema, definisana kao broj polaznih podataka [6]. Npr. za algoritam koji obrađuje niz od n elemenata kažemo da ima dimenziju n. Vremenska složenost jeste funkcija $T(n)$ koja preslikava dimenziju u vreme. Tačnu vrednost funkcije $T(n)$ moguće je odrediti samo za jednostavnije algoritme, dok u najvećem broju realnih slučaja dovoljno je utvrditi asimptotsko ponašanje $T(n)$ za velike vrednosti n imajući u vidu da je $T(n)$ monotono neopadajuća funkcija.

Termin *elegancija* algoritma [7], [8] se vrlo često može naći u literaturi, iako objašnjenje ovog termina ne postoji. Termin potiče iz matematike, gde elegancija nekog rešenja označava njegovu kratkoću. Ovo ima smisla u matematici, jer jedan isti problem može se rešiti na više načina, ali način u kojem se koristi neka matematička dosetka može pojednostaviti i/ili skratiti rešenje. Takva rešenja se posebno cene u matematici i za njih se kaže da su elegantna. Međutim, situacija u programiranju nije ista kao u matematici. Naime, fizički kraće rešenje, koje čak i programerovom oku deluje primamljivije, jer u sebi krije neku programersku dosetku, u opštem slučaju, ne garantuje i manju algoritamsku složenost, već naprotiv, može je čak i povećati. Cilj ovoga rada jeste da argumentuje da ne postoji analogija između elegancije rešenja u matematici i u teoriji algoritama. U prilog tome, mi ćemo u ovom radu analizirati jedan jednostavan primer (algoritam koji određuje Fibonačijeve brojeve) iz kojeg sledi da elegantnije algoritamsko rešenje predstavlja uzrok većoj složenosti algoritma.

II. Vremenska složenost

Asimptotska procena vremenske složenosti može se izvršiti na dva načina: određivanjem reda veličine i određivanjem gornje granice. Prvi način, ako je moguć, je precizniji i podrazumeva da se složenost odredi tako što se pronađe funkcija $f(n)$ za koju važi:

$$\lim_{n\to\infty}\frac{T(n)}{f(n)} = const \neq 0,$$

što znači da za velike vrednosti n složenost $T(n)$ i funkcija $f(n)$ podjednako brzo rastu, pa kažemo da je složenost $T(n)$ reda $f(n)$ i za red veličine koristimo oznaku „veliko O“ i pišemo:

$$T(n) = O\left[f(n)\right].$$

Kada je nemoguće složenost odrediti pomoću reda veličine, onda se koristi gornja granica koja se označava „malim o“. Ako za neku funkciju $g(n)$ važi:

$$\lim_{n\to\infty}\frac{T(n)}{g(n)} = 0$$

tada za velike vrednosti n složenost $T(n)$ sporije raste nego $g(n)$ i pišemo:

$$T(n) = o\left[g(n)\right].$$

Funkciju g je uvek lako pronaći, s tim da je ona manje informativna, pa se zato i kaže da predstavlja slabiju meru od funkcije f.

Dušan T. Malbaški, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (phone: 381-21-485-2421; e-mail: malbaski@uns.ac.rs).

Aleksandar D. Kupusinac (autor za kontakte), Fakultet tehničkih nauka, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (phone: 381-21-485-2441, e-mail: sasak@uns.ac.rs).

Postoji nekoliko karakterističnih oblika funkcije $f(n)$ prema kojima se vrši klasifikacija algoritama. Specifični oblici funkcije $f(n)$ sa nazivima odgovarajućih klasa algoritama dati su u Tabeli 1.

TABELA 1: KLASE ALGORITAMA.

f(n)	*Klase algoritama*
const	konstantan
$\log_2 n$	logaritamski
n	linearan
$n \cdot \log_2 n$	linearno-logaritamski
n^2	kvadratni
n^k (k>2)	stepeni
k^n (k>1)	eksponencijalni
$n!$	faktorijelni

Klase algoritama su poređane po rastućem redosledu složenosti u Tabeli 1.

III. ELEGANCIJA I SLOŽENOST

Sada ćemo razmotriti algoritam koji određuje Fibonačijeve brojeve. U matematici Fibonačijevi brojevi se definišu na sledeći način:

$$F(0) = 0,$$
$$F(1) = 1$$
$$F(n) = F(n-1) + F(n-2), \text{ za } n = 2,3,\ldots.$$

Rekurzivnu funkciju za određivanje Fibonačijevih brojeva je izuzetno lako napisati:

```
function Fr(n: longint): longint;
begin
  if n<2 then Fr:=n
  else Fr:=Fr(n-1)+Fr(n-2)
end;
```

Iterativna funkcija za određivanje Fibonačijevih brojeva izgleda ovako:

```
function Fi(n: longint): longint;
var
  i, a, b, c: longint;
begin
  a:=0; b:=1; c:=0;
  for i:=2 to n do
  begin
    c:=b+a;
    a:=b;
    b:=c
  end;
  Fi:=c;
end;
```

Možemo reći da rekurzivna funkcija deluje kratko, jednostavno i razumljivo, dok iterativna funkcija deluje početnički. Možemo reći da rekurzivna funkcija predstavlja elegantno algoritamsko rešenje problema određivanja Fibonačijevih brojeva. Međutim, ako se izvrše merenja vremena izvršavanja ove dve funkcije za različite vrednosti parametra n dobijaju se iznenađujući rezultati. Izmerene vrednosti vremena u sekundama izvršavanja iterativne i rekurzivne funkcije na računaru PC 386/40MHz su date u Tabeli 2.

TABELA 2: TRAJANJE ITERATIVNE I REKURZIVNE FUNKCIJE.

n	*Trajanje Fi(n)*	*Trajanje Fr(n)*
0	0.0	0.0
5	0.0	0.0
10	0.0	0.0
15	0.0	0.0
20	0.0	0.1
25	0.0	1.2
30	0.0	13.0
32	0.0	33.7
35	0.0	143.4

Očigledno, razlike između vremena izvršavanja iterativne i rekurzivne funkcije su upadljive. Da bi se utvrdili razlozi ovakvog ponašanja funkcija *Fi* i *Fr* izvešćemo njihovu analizu složenosti. Za iterativnu funkciju *Fi* se lako dobija da je:

$$T_{Fi}(n) = C_1 + C_2(n-1) = O[n],$$

odnosno da funkcija *Fi* spada u klasu linernih algoritama.

Analiza rekurzivne funkcije *Fr* je nešto složenija. Iz programskog teksta funkcije se dobija:

$$T_{Fr}(n) = C + T_{Fr}(n-1) + T_{Fr}(n-2), \quad n \geq 2,$$

odakle dobijamo:

$$T_{Fr}(2) = C + T_{Fr}(1) + T_{Fr}(0)$$
$$T_{Fr}(3) = C + T_{Fr}(2) + T_{Fr}(1) = 2C + 2T_{Fr}(1) + T_{Fr}(0)$$
$$T_{Fr}(4) = 4C + 3T_{Fr}(1) + 2T_{Fr}(0)$$
$$T_{Fr}(3) = 7C + 5T_{Fr}(1) + 3T_{Fr}(0) \text{ itd.}$$

, odnosno

$$T_{Fr}(n) = (F(n+1) - 1)C + F(n)T_{Fr}(1) + F(n-1)T_{Fr}(0)$$

, gde $F(n)$ označava n-ti Fibonačijev broj, a kako je $T_{Fr}(1) = T_{Fr}(0) = a = const$ dobijamo:

$$T_{Fr}(n) = (F(n+1) - 1)C + (F(n) + F(n-1))a,$$
$$T_{Fr}(n) = (F(n+1) - 1)C + (F(n+1))a,$$
$$T_{Fr}(n) = F(n+1)(C + a) - C.$$

Prema tome, vrednost $T_{Fr}(n)$ je proporcionalna vrednosti $F(n+1)$. S obzirom da je [9], [10]

$$F(k) = \frac{x^k - (1-x)^k}{\sqrt{5}}, \text{ gde je } x = \frac{1+\sqrt{5}}{2} \approx 1.618.$$

Za velike vrednosti k biće $F(k) \sim \frac{x^k}{\sqrt{5}}$. Odavde se lako vidi da je:

$$\lim_{n\to\infty} T_{Fr}(n) = \frac{x(C+a)}{\sqrt{5}} \lim_{n\to\infty} x^n$$

, gde su x, C i a konstante.

Na osnovu toga zaključujemo da je

$$T_{Fr}(n) = O\left[x^n\right], \text{ gde je } x \approx 1.618 > 1,$$

iz čega sledi zaključak da rekurzivna funkcija *Fr* pripada klasi eksponencijalnih algoritama.

Dakle, dokazali smo da iterativna funkcija *Fi* ima linearnu složenost, a rekurzivna funkcija *Fr* ima eksponencijalnu složenost. Iako rešavaju isti problem i iako je rekurzivno rešenje elegantnije, njihova složenost nije ista, odnosno u tom smislu rekurzivno rešenje je mnogo lošije od iterativnog. Time su u potpunosti objašnjeni iznenađujući rezultati dobijeni u našem eksperimentu, a koji su prikazani u Tabeli 2. Najzad, na osnovu teorijske analize možemo zaključiti i to da bi računar bez problema izračunao vrednost iterativne funkcije *Fi*(100), dok za rekurzivnu funkciju *Fr*(100) bi mu bilo potrebno oko 1.7 milijardi godina.

IV. ZAKLJUČAK

Analizom algoritma za određivanje Fibonačijevih brojeva, u ovom radu prikazali smo da elegancija algoritamskog rešenja ne znači i manja složenost. Naprotiv, pokazali smo da rekurzivna funkcija koja određuje Fibonačijeve brojeve predstavlja elegantno rešenje, ali ima eksponencijalnu složenost, dok iterativna funkcija koja ne predstavlja elegantno rešenje ima linearnu složenost. Time smo pokazali da elegantno rešenje može dovesti čak i do povećanja složenosti i time smo argumentovali da ne postoji analogija između elegancije rešenja u matematici i u teoriji algoritama.

LITERATURA

[1] P. Hotomski i D. Malbaški, *Matematička logika i principi programiranja*. Tehnički fakultet "Mihajlo Pupin", Zrenjenin, 2006.

[2] Z. V. Alferova, *Teorija algoritmov*. Statistika, Moskva, 1973.

[3] A. N. Kolmogorov, *O ponjatii algoritmov*. Uspehi matematičeskih nauk, T8, vypusk 4(56), 1953.

[4] Z. Ognjanović i N. Krdžavac, *Uvod u teorijsko računarstvo.* 2004. http://www.mi.sanu.ac.rs/~zorano/taja/TeorijskoRacunarstvo.pdf

[5] V. A. Uspenskij i A. L. Semjonov, *Teorija algoritmov: osnovnye otkrytija i priloženija.* Nauka, Moskva, 1987.

[6] S. Lipschutz, *Data Structures*. Schaum Outline Series, McGraw Hill, 1986.

[7] A. M. Chan and F. R. Kschischang, "A Simple Taboo-Based Soft-Decision Decoding Algorithm for Expander Codes", *IEEE Communications Letters*, vol. 2, no. 7 (July), 1998.

[8] G. Taubenfeld, "The Black-White Bakery Algorithm and Related Bounded-Space, Adaptive, Local-Spinning and FIFO Algorithms", *DISC 2004: The 18th Annual Conference on Distributed Computing*, pp. 56-70, (October) 2004.

[9] J. J. Dujmović, *Programski jezici i metode programiranja*. Naučna knjiga, Beograd, 1990.

[10] R. Tošić, *Kombinatorika*. PMF, Novi Sad, 1999

ABSTRACT

Term algorithm *elegance* is frequently used in literature, but its meaning is unclear. We can consider elegance of mathematical solutions, where we assume that they solve a mathematical problem on the short way, but the situation in theory of algorithms is quite different than in mathematics. In this paper we will show that algorithm elegance can be a factor which can increase algorithm complexity.

ALGORITHM ELEGANCE AND COMPLEXITY

Dušan T. Malbaški and Aleksandar D. Kupusinac