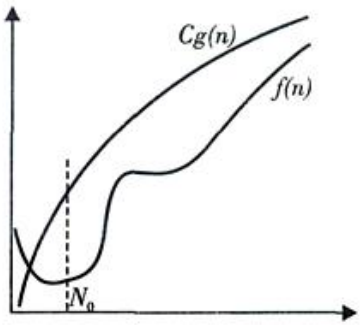
О АЛГОРИТМИМА

***мр Ана Капларевић–Малишић,* Крагујевац**

КЛАСЕ КОМПЛЕКСНОСТИ

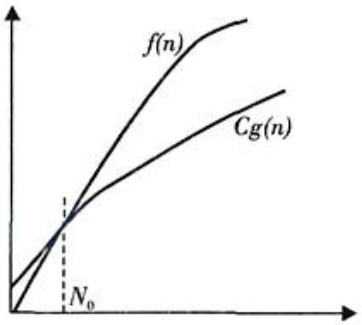
У прошлом броју смо увели појмове асиптотских симбола, тј. дефинисали шта се крије иза симбола o, O, Ω, Θ, $\sim$. Да би вам наредна прича била јаснија вратимо се још једном на дефиниције.

Као што смо већ рекли, за неку функцију f кажемо да припада класи функција $O(g)$ ако расте брзином која је једнака или мања брзини пропорционалној брзини раста фукције g. Поштујући дефиницију графички представљене функције f и Cg (C је константа која према дефиницији постоји) би могле изгледати овако

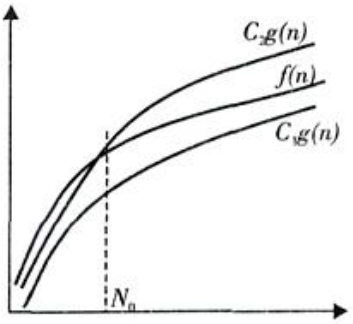


$f \in O(g)$

Као што можете видети, „ограничење“ функције f функцијом Cg мора да важи само за $n \geqslant N_0$. Исто важи и за остале класе функција (видети графике)



$f \in \Omega(g)$

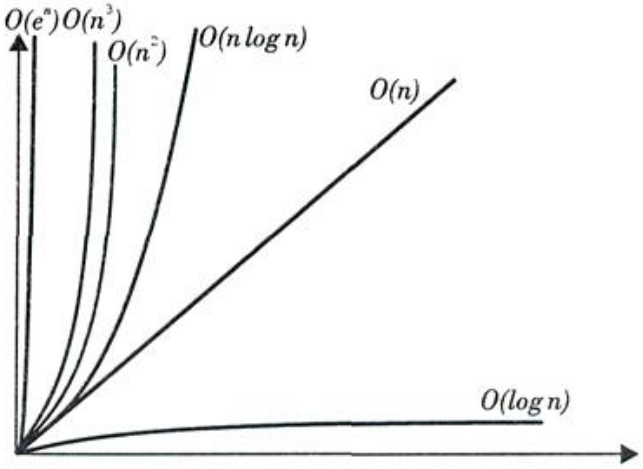


$f \in \Theta(g)$

Ако се сада вратимо на употребу ових симбола при поређењу комплексности два алгоритма, морамо закључити да асимптотско поређење комплексности два алгоритма има смисла само за „велики“ улаз. За „мале“ величине улаза један алгоритам може бити бржи

од другог, док је код „великог“ улаза ситуација обрнута. На пример, $n \log_2 n$ расте брже од $32n$, али је бољи за „мале“ вредности n ($n < 4294967296$). Битно је имати у виду да асимптотско поређење представља „груб алат“, па тако ако поредите алгоритме код којих је $T(n)$ линеарна функција нећете добити довољно јасан одговор.

Ради потпуније представе упоредимо још O класе [§] неких функција (погледати график).



$f \in O(g)$

Ако превидимо „фине“ прелазе између „суседних“ класа (нпр. $O(n^2)$ и $O(n^3)$), алгоритме према комплексности можемо грубо поделити у три групе:

– алгоритми експоненцијалне коплексности као најспорији

– алгоритми са „полиномним временом“ (код сложених проблема овакве брзине су пожељне и најчешће тешко достижне)

– алгоритми чија комплексност „логаритамски“ зависи од величине улаза као најбржи.

Закључимо на крају да при поређењу комплексности два алгоритма асимптотском анализом тачан број операција није битан, већ само којим класама комплексности алгоритми припадају. Описивањем класама комплексности занемарујемо „мале“ улазе и коефицијенте пропорционалности ($3n^2 \in O(n^2), 1000n^2 \in O(n^2), \ldots$).

Како одредити комплексност

Сада се поставља питање како доћи до процене комлексности једног алгоритма. За почетак морате да се одлучите број којих операција (у зависности од величине улаза) одређујете. За које ћете се одлучити зависи од тога које су „проблемати чне “, тј. чији број зависи од улаза и ако алгоритам поредите са неким дугим у броју којих операција се разликују када се величина улаза мења. Након тога их треба „пребројати“. Дајемо пример два алгоритма за множење матрица.

[§] на слици су дати графици по једне произвољно изабране функције из сваке класе

Множење матрица 'по дефиницији'

Нека су A и B квадратне матрице димензија $N \times N$. Елемент матрице производа A и B се израчунава по следећем обрасцу

$$c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj} \qquad (i, j = 1, N).$$

Део кода који би одредио све елементе би у Pascal-у изгледао овако

```
FOR I:=1 TO N DO
    FOR J:=1 TO N DO
        BEGIN
        C[I,J]:=0;
        FOR K:=1 TO N DO C[I,J]:=C[I,J]+A[I,K]*B[K,J];
        END;
```

За рачунање сваког c_{ij} појединачно поребно је N множења и N сабирања. А како елемената матрице C има укупно $N \cdot N$, то значи да је укупан број множења потребних за одређивање целе матрице $N \cdot N \cdot N$, тј. N^3. Исто важи и за укупан број сабирања.

Strassen-ов алгоритам

1969. године В. Strassen је описао другачији поступак множења матрица у коме је број операција смањен. Наиме, он је за почетак дао нешто другачији поступак долажења до матрице производа две матрице 2×2 који се одвија на следећи начин:

$$\begin{aligned}
I &= (a_{12} - a_{22}) \times (b_{21} + b_{22}) \\
II &= (a_{11} + a_{22}) \times (b_{11} + b_{22}) \\
III &= (a_{11} - a_{21}) \times (b_{11} + b_{12}) \\
IV &= (a_{11} + a_{12}) \times b_{22} \\
V &= a_{11} \times (b_{12} - b_{22}) \\
VI &= a_{22} \times (b_{21} - b_{12}) \\
VII &= (a_{21} + a_{22}) \times b_{11} \\
C_{11} &= I + II - IV + VI \\
C_{12} &= IV - V \\
C_{21} &= VI + VII \\
C_{22} &= II - III + V - VII
\end{aligned}$$

Дакле, за матрицу 2×2 потребно је 7 множења и 18 $' + -'$ операција. Ова измена за матрицу 2×2 не прави неку разлику, али доноси битно побољшање за матрице $N \times N$. Наиме, идеја је да се множење матрица $N \times N$ рекурзивно сведе на множење матрица 2×2. Како?

Нека је $N = 2^n$ и нека су дате две $n \times n$ матрице A и B. Изделимо матрице A и B на четири $2^{n-1} \times 2^{n-1}$ матрице. Тада се тражена матрица може добити на следећи начин

$$(*) \qquad \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

где су A_{ij}, B_{ij} матрице добијене дељењем матрица A и B, а C_{ij} одговарајући делови резултујуће матрице. Да би се одредили C_{ij} елементи користе се наведене формуле за 2×2 матрице. Ако је $n > 2$ онда се сваки појединачни елемент C_{ij} рачуна даљим дељењима матрица A_{ij} и B_{ij} и поновном применом $(*)$ све док се димензије подматрица не сведу на 1×1. За реализацију овог алгоритма потребно је написати рекурзивну функцију која би могла, скраћено, бити описана на следећи начин:

```
FUNCTION MatrProd (A,B: matrix; N: INTEGER):matrix;
BEGIN
   IF N nije stepen od 2 THEN
       Prosiruj A i B vrstama i kolonama nula dok im nova dimenzija ne postane
       stepen od 2 i dodeli N novu vrednost;
   IF N=1 THEN MatrProd:=A*B
        ELSE
        BEGIN
         Izdeli A i B prema (*);
         I:=MatrProd(A11-A12,B21+B22,N/2);
         II:=MatrProd(A12+A22,B11+B22,N/2);
           ... i ostale formule
         C22:=II-III+V-VII;
        END;
END;
```

Овде се подразумева да се не може писати $A_{11} - A_{12}$, односно $B_{21} + B_{22}$, већ је за израчунавање потребно написати функције/процедуре за одузимање/сабирање матрица.

Процедура рекурзивно позива саму себе 7 пута и има 18 позива сабирања/одузимања матрица, за сваку вредност n (где је n број из $N = 2^n$).

Одредимо комплексност овог алгоритма, као и код претходног, по два критеријума :

1. број множења бројева потребних за множење две $2^n \times 2^n$ матрице – означимо га функцијом $f(N)$;

2. број сабирања бројева потребних за множење две $2^n \times 2^n$ матрице – означимо га функцијом $g(N)$.

1. Број множења

MatrProd позива саму себе 7 пута и у сваком позиву има $f(n-1)$ множења, при чему је $f(0) = 1$. Дакле, како је $f(n) = 7f(n-1)$, $f(1) = 1$ следи да је $f(n) = 7^n, n \geqslant 0$. Значи, за две матрице $2^n \times 2^n$ је потребно 7^n множења. Ако n изразимо преко N тада, онда је

$$7^n = 7^{\frac{\log N}{\log 2}} = N^{\frac{\log 7}{\log 2}} < N^{2.81}.$$

Ако се подсетите дефиниције O, видећете да важи да је $f(n) = O\left(N^{2.81}\right)$, тј. да функција $f(n)$ расте брзином пропорционалном брзини раста $N^{2.81}$ или спорије.

2. Број сабирања

У сваком позиву има 18 сабирања/одузимања и 7 рекурзивних позива са $g(n-1)$ сабирања/одузимања бројева, па је

$$g(n) = 7g(n-1) + 18 \cdot 4^{n-1} = 7g(n-1) + \frac{9}{2}4^n.$$

Нека је $g(n) = 7^n y_n$ $(n \geqslant 0)$. Тада је $y_0 = 0$ и

$$y_n = y_{n-1} + \frac{9}{2}\left(\frac{4}{7}\right)^n = 0 + \frac{9}{2}\cdot\frac{4}{7} + \frac{9}{2}\left(\frac{4}{7}\right)^2 + \cdots + \frac{9}{2}\left(\frac{4}{7}\right)^n$$
$$= \frac{9}{2}\sum_{j=1}^{n}\left(\frac{4}{7}\right)^j \leqslant \frac{9}{2}\sum_{j=1}^{\infty}\left(\frac{4}{7}\right)^j = \frac{21}{2}.$$

Дакле, $g(n) = 7^n y_n \leqslant (10.5)7^n = O\left(7^n\right)$, па је $g(n) = O\left(N^{2.81}\right)$.

Погледајмо сада оба алгоритма. Неко прецизније одређење функције $f(n)$, односно $g(n)$ за Strassen–ов алгоритам није потребно, јер је, као и у највећем броју случајева, за поређење брзина два алгоритма довољно „ограничити“ брзину раста са горње стране. Што се броја множења, односно сабирања „стандардног“ алгоритма тиче, очигледно је да припадају класи $O\left(N^3\right)$.

Можда не изгледа да између N^3 и $N^{2.81}$ (самим тим и $O\left(N^3\right)$ и $O\left(N^{2.81}\right)$) постоји велика разлика, али што је N веће то је разлика у брзини раста осетнија.

На крајњу процену комплексности Strassen-овог алгоритма неће утицати ако се узме у обзир случај када се матрица проширује $0-$врстама и колонама до одговарајућих димензија, под условом да се алгоритам преправи тако да буду избегнуте све операције у којима учествују $0-$матрице.

Обратите пажњу на то да на комплексност, односно брзину ова два алгоритма не утиче садржај улаза, већ само величина (како год организовали множење матрица „пролаз“ кроз целе улазне матрице нећете моћи да избегнете). То није случај код свих алгоритама.

Узмимо пример претраге низа или листе по кључу (нпр. са садржајем телефонског именика), тј. утврђивања да ли у низу постоји елемент са вредношћу кључа једнаком траженој вредности (алгоритми за претрагу обично подразумевају да је улазна листа сортирана). На брзину рада алгоритама са оваквим задатком утиче број поређења потребних да се изврше да би се дошло до одговора. Како год да је замишљен овај алгоритам може дати одговор после 0 (или једног поређења), ако је унети кључ једнак кључу елемента листе од којег се „креће“ са поређењем. Са друге стране могуће је да се елемент са унетим кључем не налази у листи или се налази на „последњем месту“ на којем ће га алгоритам тражити (што је по Марфијевом закону врло могуће).

Очигледно је да се комплексност алгоритама који зависе од „квалитета“ улаза не може одређивати једнозначно, па се тако за њих одређују брзине у најбољем, просечном и најгорем случају (best-case, average-case, worst-case) задавања улаза. Поређење се међу њима може вршити у сваком од ова три случаја.

Статијата прв пат е објавена во списанието Тангента на ДМ на Србија во 2004/05 година