The Third International Conference "Problems of Cybernetics and Informatics", September 6-8, 2010, Baku, Azerbaijan. Section #7 "Numerical Methods and Computational Technology"
www.pci2010.science.az/7/26.pdf

ASYMPTOTIC PROPERTIES OF SOLUTIONS OF A SPECIAL TYPE OF RECURRENCE RELATIONS

Valentina Bykova

Institute of Mathematics Siberian Federal University, Russia, Krasnoyarsk
bykvalen@mail.ru

The modern software industry requires development of special methods of analysis algorithms. Analysis algorithms in the theory of complexity are traditionally performed in terms of length of computational processes that generate algorithms. Computational complexity of the algorithm is formally described by t (n), which is a function of time complexity. This function represents the maximum number of elementary steps which requires the algorithm to achieve results depending on n (n is length of input of algorithm). Typically the length of the entrance of the algorithm is the size of the problem being solved [1]. In the analysis algorithms have traditionally study the behavior of functions complexity as $n\to\infty$ [2 – 4].

There are two broad classes of algorithms: classes of iterative algorithms and of recursive algorithms. Iterative algorithms are repeated many time the same actions. The structure of the iterative algorithm is well described by algorithmic elements: "to follow the order", "branching", "cycle". Analysis of complexity iterative algorithms is reduced to determining the complexity these elements and the formation of the integral of the asymptotic evaluation [3].

Recursive algorithm is copying a whole in parts. For example transition from the problem of size n to the same problem of smaller size. There are several important reasons that hinder the wide application of recursion. First recursive algorithms are often more costly in terms of time and memory than the iterative algorithms that solve the same problem. The complexity of recursive algorithm is greatly influenced by the organization of recursion. Secondly the analysis of the complexity of recursive algorithms is one of the most difficult problems of the theory of computational complexity.

The main tool for investigating the complexity of recursive algorithms is the method of recurrence relations. The idea of the method consists in constructing and solving the recurrence equation for $t(n)$ function of complexity algorithm. The solution found allows you to get the O-estimates for $t(n)$.

However it can be difficult since not are there general methods for solving recurrences relations. We know the only methods for solving some classes of recurrences relations. Nevertheless in many practical situations we find the way. For example relatively well resolved recurrence relations which are characteristic for recursive algorithms of type "divide and conquer" and additive reduction of the size of the problem by some constant. There are two main theorems of recurrence relations. These theorems are a convenient mathematical tool for analyzing the complexity of the two most common principles of organization of recursion. Their use avoids tedious calculations; choose the least labor-intensive organization scheme of recursion. In this paper we study the second of these theorems. This theorem was proved by the author in [5]. We got new studies of this theorem and practical recommendations on the organization of recursion additive to reduce the size of the problem. These results we intend to present.

Initially we point out the characteristic features of recurrence relations for the two main principles of organization of recursion. When recording a recurrence relation for $t(n)$ function should take into account the complexity of no recursive and recursive branch of the algorithm. No recursive branch defines sets the initial conditions. Recursive branch defines itself recurrence relation. The values of $t(n)$ function to recursive algorithm are calculated by the formula:

$$t(n)=\begin{cases} c, & 0\le n\le n_0, \\ t_r+t_s+t_u, & n>n_0\ge 0, \end{cases} \tag{1}$$

where n is the parameter recursion, n_0 is the size of the problem, in which the algorithm does not depend on n, $c \geq 0$ is the complexity of no recursive branch (a constant value), t_s is have time to go to the subproblems, t_r is the complexity of recursive branches (time for computing subproblems), t_u is the time integration of solutions which were obtained of subproblems.

In principle "divide and conquer" problem which has size n divided into subproblems size n/k, where $k > 1$ is the whole constant. In this case the relation (1) takes the form:

$$t(n) = \begin{cases} c, & n = n_0, \\ A(n)\ t(n/k) + B(n), & n > n_0 \geq 0\ . \end{cases} \tag{2}$$

Here $A(n)$, $B(n)$ are nonnegative, monotonically increasing, real-valued functions of $n \in \mathbf{Z}_+$ ($\mathbf{Z}_+$ is the set of positive integers). These functions characterize the cost of a recursive transition. Since k is a constant then the transformation $g(n) = n/k$ the size of original problem in the size of subproblems is linear in n. Obviously the relation (2) uniquely defines the function $t(n)$ only for $n = 0$ and $n = k^m$, $m \in \mathbf{Z}_+$.

For recursive algorithms, organized by the additive reduce of the original problem size n by some constant $k \geq 1$ equation (1) reduces to:

$$t(n) = \begin{cases} c, & 0 \leq n \leq n_0, \\ A(n)\ t(n-k) + B(n), & n > n_0 \geq 0, \end{cases} \tag{3}$$

where $A(n)$, $B(n)$ have the same meaning as in (2). In this case the function $g(n) = n - k$ also determines the linear transformation of the size of the original problem in size subproblems and recurrence relation (3) uniquely defines a function $t(n)$ only if $n = km$, $m \in \mathbf{Z}_+$.

Solve the equation (2), (3) in general is not possible, because they are too general appearance. Meanwhile in the particular case when $A(n) = a$ and $B(n) = bn^\tau$, where a, b, τ are positive constants the type of solutions of (2), (3) define the two main theorems. The constant a is interpreted as a number of subproblems generated by the recursive branch of the algorithm and a power function bn^τ defines complexity recursive of the transition.

The first fundamental theorem to recurrence relations proved J. Bentley, D. Haken and J. Saxe in 1980 [2, 3]. This theorem gives the solution of recurrence relation which is characteristic of recursion such as "divide and conquer".

Theorem 1 Consider the recurrence relation

$$t(n) = \begin{cases} c, & n = 1, \\ at(n/k) + bn^\tau, & n > 1, \end{cases}$$

where $a > 0$, $k > 1$ are integer constants and $b \geq 0$, $c \geq 0$, $\tau \geq 0$ are real constants. Then for $n = k^m$, $m \in \mathbf{Z}_+$ solution of the given ratio is the function:

$$t(k^m) = \begin{cases} a^m c + b k^{m\tau} m, & a = k^\tau, \\ a^m c + b k^{m\tau} \dfrac{(a/k^\tau)^m - 1}{(a/k^\tau) - 1}, & a \neq k^\tau. \end{cases}$$

Under the assumptions of Theorem 1 when $n \to \infty$ and any $b > 0$ and $c \geq 0$ are correct estimates:

$$t(n) = \begin{cases} O(n^\tau \log_k n), & a = k^\tau, \\ O(n^\tau), & a < k^\tau, \\ O(n^{\log_k a}), & a > k^\tau. \end{cases} \tag{4}$$

When $b = 0$ and $c > 0$, is always correct estimate:

$$t(n) = O(a^{\log_k n}) = O(n^{\log_k a}). \tag{5}$$

All estimates (4), (5) show a polynomial order of increasing function $t(n)$ for any value of $t(1) = c \geq 0$. Consequently the recursion type of "divide and conquer" always leads to polynomial algorithms. In addition, the computational complexity is clearly dependent on the number of subproblems and the size of the subproblems: if a more balanced split the task into subtasks, then the recursive algorithm will have a better (lower) estimate of complexity.

The second fundamental theorem of recurrence relations shows estimates for the solution of recurrent relations, which is characteristic for recursion with an additive reduction of the size of the problem by some constant $k \geq 1$. For example such relationships arise in recursive implementation of the method of dynamic programming.

Theorem 2 [5] Consider the recurrence relation

$$t(n) = \begin{cases} c, & 0 \leq n \leq k-1, \\ a\,t(n-k) + b\,n^{\tau}, & n \geq k, \end{cases} \tag{6}$$

where $a > 0$, $k \geq 1$ are integer constants and $b \geq 0$, $c \geq 0$, $\tau \geq 0$ are real constants. Then for $n = km$, $m \in \mathbf{Z}_+$ are correct inequalities:

$$c + bk^{\tau-1}n \leq t(n) \leq c + \frac{b}{k}n^{\tau+1}, \quad a = 1, \tag{7}$$

$$a^{n/k}c + bk^{\tau}\,\frac{a^{n/k}-1}{a-1} \leq t(n) \leq a^{n/k}c + bn^{\tau}\,\frac{a^{n/k}-1}{a-1}, \quad a \neq 1. \tag{8}$$

Formula (7) (8) do not give an exact solution to the recurrence relation (6) and hence the exact asymptotic estimates. They only define the upper and lower boundaries. In this sense Theorem 2 is weaker than the first main theorem. Nevertheless in some special cases you can get exact solutions and evaluation. This is evidenced by the following investigation.

Consequence 1 Under the assumptions of Theorem 2 for $\tau = 0$ the solution of recurrence relation (6) is a function:

$$t(n) = \begin{cases} c + \frac{b}{k}n, & a = 1, \\ a^{n/k}c + b\,\frac{a^{n/k}-1}{a-1}, & a \neq 1. \end{cases} \tag{9}$$

Consequence 2 For $\tau = 0$, $c \geq 0$ and $n \to \infty$ for the recurrence relation (6) are correct asymptotic estimates:

$$t(n) = \begin{cases} O(n), & a = 1, \ b > 0, \\ O(a^{n/k}), & a \neq 1, \ b > 0. \end{cases} \tag{10}$$

In particular, when $\tau = 0$, $a = 1$ and $b = 0$ is always $t(n) = O(1)$.

Consequence 3 For $\tau \geq 0$, $a = 1$, $c \geq 0$, $b > 0$ and $n \to \infty$ for the recurrence relation (6) is correct asymptotic estimate:

$$t(n) = O(n^{\tau+1}). \tag{11}$$

Consequence 4 For $a > 1$, $c \geq 0$, $b > 0$ and positive integers τ for the recurrence relation (6) is correct asymptotic estimate:

$$t(n) = O(n^{\tau}a^{n/k}). \tag{12}$$

From (9) – (12) can make an important practical conclusion: recursive algorithm which creates an additive reduction in the size of the problem to some constant can be a polynomial and exponential complexity.

If $B(n) = bn^{\tau}$ and $a \neq 1$, $b > 0$ and $c \geq 0$ (or $b \geq 0$ and $c > 0$) then algorithm will be exponential complexity always. What do these requirements? How should organize a recursive algorithm to achieve polynomial complexity? For answers to these questions is necessary to recall the meaning of the constants of (6). Constant a determines the number of subproblems of size $n - k$. Integer values a (in Theorem 2) is only needed for this interpretation. Constant b and τ describe the complexity of recursive transition from n to $n - k$. The constant c characterize the time costs required to directly address the problem, when its size is extremely small.

For real algorithms, there are always some costs for the organization of recursion i.e. $b > 0$ and $c > 0$. If these costs do not depend on n then $\tau = 0$. At $\tau = 0$, $a = 1$ by Consequence 2 recursive algorithm has linear complexity. When $\tau \geq 0$, $a = 1$ and $n \to \infty$ by Consequence 3 for $t(n)$ true the estimate $t(n) = O(n^{\tau+1})$. So it is always at $a = 1$ a recursive algorithm organized by reducing the size of the problem to some constant is a polynomial. When $a \neq 1$ and

the positive integral values of the constants τ (Consequence 4) is correct asymptotic estimate $t(n) = O(a^{n/k})$. In general, when $a \neq 1$ and any $\tau \geq 0$ and a fixed value of $1 \leq k < n$ inequality (8) indicates that the function $t(n)$ grows no faster than $O(n^{\tau}a^{n/k})$.

Now we investigate the lower boundary of the growth rate of the function $t(n)$. We put in the ratio (6) $b = 0$. Then the recurrence relation (6) becomes uniform

$$t(n) = \begin{cases} c, & 0 \leq n \leq k-1, \\ a\,t(n-k), & n \geq k. \end{cases} \tag{13}$$

Inequality (7), (8) becomes an equality $t^0(n) = a^{n/k}c$. This equation can be interpreted as a general solution of homogeneous equation (13). If $a \neq 1$ then solution $t^0(n) = a^{n/k}c$ always has an exponential order of growth. According to (8) when $a \neq 1$ the particular solution $t^*(n)$ for recurrence relation (6) is a function that for large n grows no faster than

$$bn^{\tau}\frac{a^{n/k}-1}{a-1} = O(n^{\tau}a^{n/k}). \tag{14}$$

According to (14) for $a \neq 1$ the function $t(n) = t^0(n) + t^*(n)$ grows no slower than $t^0(n) = a^{n/k}c$ and no faster than $O(n^{\tau}a^{n/k})$.

In practical terms, this means if the number of subproblems at each recursive step greater than unity ($a \neq 1$), then recursive algorithm organized by additive reduction the size of the problem always has exponential complexity. In this case, improvement of procedures for splitting and combining subproblems cannot change the class of complexity. Cause an exponential complexity when $a \neq 1$ lies in the appearance at each step of recursion overlapping subproblems.

All results are obtained in this work are extremely important for software developers and experts in the field of theoretical computer science. These results allow us to correctly use a recursive approach in programming. The general practical advice is this: to get the recursive algorithms of polynomial complexity is necessary, first of all, take care of the balance of subproblems, at each step of recursion create new independent subproblems, effectively carry out the procedure for partitioning and consolidation problems.

References

1. D.B. Yudin, A.D. Yudin, Mathematic measures the Complexity (Moscow, Russia, 2009).
2. R. Graham, D. Knuth, O. Patashnik, Concrete Mathematics. A Foundation for Computer Science (Addison-Wesley Publishing Company, 1994).
3. S. Baase, A. Gelder, Computer Algorithms: introduction to Design and Analysis (Addison-Wesley Publishing Company, 2000).
4. V.V. Bykova, Recognition Method of Algorithms Classes on the Basis of Asymptotics for Elasticity Functions Complexity, Journal of Siberian Federal University, Mathematics & Physics. 2(1), 2009, 48-62 (in Russian).
5. V.V. Bykova, Mathematical methods of analysis of recursive algorithms, Journal of Siberian Federal University, Mathematics & Physics. 1(3), 2008, 236-246 (in Russian).