Reduction of programs execution time for tasks related to sequences or matrices

Oleksandr Mitsa[1], *Yurii* Horoshko[2,*], and *Serhii* Vapnichnyi[1]

[1]Uzhhorod National University, Uzhhorod, Ukraine
[2]T.H. Shevchenko National University "Chernihiv Colehium", Chernihiv, Ukraine

Abstract. The article discusses three approaches to reducing runtime of the programs, which are solutions of Olympiad tasks on computer science, related to sequences or matrices. The first approach is based on the representation of some sequences in matrix form and then the program of calculating the members of the sequence will have asymptotics equal to the time complexity of the exponentiation algorithm and will be O(log (n)). The second approach is to upgrade the known code to obtain significant reduction of the program runtime. This approach is very important to know for scientists who write code for scientific researches and are faced with matrix multiplication operations. The third approach is based on reducing time complexity by search for regularities; the author's task is presented and this approach is used to solve it.

Introduction

Sports programming has now become a promising intellectual sport. Every year, the number of pupils and students interested in Olympiads in computer science, as perhaps the most common type of sports programming, is growing. There are many Olympiads and other competitions held by the largest IT companies. Relevant HR specialists from these companies have been monitoring the results of various competitions and specific participants for many years. The most promising and successful participants are offered internships, combined with university study and the opportunity to gain full-time employment at the company after training. Also, many former Olympians are organizing successful projects related not only to programming and IT. Due to their participation in the Olympics, they were able to develop resistance to complex psychological stress. After spending so much time training, they have learned how to evaluate the likelihood of victory and defeat, have mastered existing and developed their own methods of dealing with stressful situations, the doubts and anxieties experienced by Olympic athletes in varying degrees.

Participation in Olympiads, tournaments and other competitions help students to improve their skills [1]. At first glance, it seems that to achieve solid results at the Olympics, it is enough to study a certain number of existing algorithms and theoretical material, and then only to successfully use them in competitions, leaving others no chance of winning. But it is not. Tasks in competitions are usually formulated in such a way that it is not enough to guess which algorithm to use to solve it. Almost always, in order to obtain a complete solution, it is necessary to upgrade the known algorithm, to supplement it, to combine several algorithms in one program, and to take some steps to reduce the time complexity of the solution [2].

This paper proposes three ways to reduce the runtime for computer science tasks that require the use of sequences and / or arrays:
- performing calculations using a matrix representation of sequences;
- reducing program execution time by using the features of the programming language;
- reducing time complexity by looking for regularities.

The first two techniques need to be learned to show the best results in standard situations. The third approach already needs a creative approach, has no general recommendations and is often used with the first two approaches.

1 Performing calculations using a matrix representation of sequences

Matrix data representation allows to use such algorithm as rapid exponentiation, that will significantly accelerate the program's work to find the desired element. One of these sequences that can be written in matrix form is the second-order linear recurrent sequences named after Edward Luke. These are pairs of sequences $\{U_n(P,Q)\}$ and $\{V_n(P,Q)\}$, whose recurrence relationship is written as follows:

$$\begin{gathered} U_0(P,Q)=0, \quad U_1(P,Q)=1, \\ U_{n+2}(P,Q)=P*U_{n+1}(P,Q)-Q*U_n(P,Q), n\geq 0 \\ V_0(P,Q)=2, \quad V_1(P,Q)=P, \\ V_{n+2}(P,Q)=P*V_{n+1}(P,Q)-Q*V_n(P,Q), n\geq 0 \end{gathered} \quad (1)$$

* Corresponding author: horoshko_y@ukr.net

Partial variants of Luke's sequences are well studied and have their own names. In particular, the sequence $\{U_n(1,-1)\}$ is better known as the Fibonacci sequence, and the sequence $\{U_n(2,-1)\}$ – as the Pell sequence. The Pellet sequence is used to quickly find $\sqrt{2}$, Pythagorean triples, etc. The Pellet sequence numbers themselves in the ratio approach the silver intersection, similar to the Fibonacci sequence numbers in the ratio approach the gold intersection. Another known sequence is the sequence $\{U_n(3,2)\}$, which is called the Mersenne sequence. It is the numbers of this sequence that are the largest known prime numbers. The numbers of this sequence can be easily verified using the Luke-Lemmer test. They are also used to effectively construct long-period pseudorandom number generators called the Mersenne vortex.

A slightly less well-known practical application, compared to the sequences discussed above, is the sequence $\{U_n(1,-1)\}$, which is called the Jacobsthal sequence. Elements of this sequence are easy to find by different schemes. The most known is the recurrence ratio:

$$J_n = \begin{cases} 0, & n = 0; \\ 1, & n = 1; \\ J_{n-1} + 2J_{n-2,} & n > 1. \end{cases} \quad (2)$$

One can also use the following recursive records

- $J_{n+1} = 2J_n + (-1)^n$; (3)
- $J_{n+1} = 2^n - J_n$. (4)

There is a known relation of the Jacobsthal sequence with the Pascal triangle [3]. It consists in the rule of choosing in the line of the Pascal triangle certain numbers, the sum of which will be the number of the Jacobsthal sequence (Fig. 1).

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1

-
- 1
1 - -
- - 3 -
- 4 - - 1
1 - - 10 - -
- - 15 - - 6 -
- 7 - - 35 - - 1
1 - - 56 - - 28 - -

Fig. 1. Relation of the Jacobsthal sequence with the Pascal triangle.

This relation can be represented as a formula as follows:

$$J(n) = \sum_{(n+k)\ mod\ 3=1} C(n,k) = \sum_{(n+k)\ mod\ 3=2} C(n,k) \quad (5)$$

The Jacobsthal sequence is also used in the problem of convergence of certain centers of a triangle on the Eulerian line of an arbitrary triangle [3]. The various relations that arise between Jacobsthal numbers are well explored in [4]. Our work below discusses a problem which has an effective solution that is based on the use of elements of the Jacobsthal sequence.

Any sequence from the Luke family of sequences is easily represented in matrix form. For example, the Fibonacci sequence has a known matrix representation [5]:

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n, \quad (6)$$

which can be overwritten as

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} F_{n-2} \\ F_{n-1} \end{pmatrix}, \quad (7)$$

or

$$\begin{pmatrix} F_{2n} \\ F_{2n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}^n \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \quad (8)$$

Other sequences under consideration are specified similarly, and the time complexity of the program of finding members of the sequence will be equal to the time complexity of the exponentiation algorithm and will be O(log(*n*)).

The ability to find directly the value of an element of the Fibonacci sequence by the formula

$$F_n = \left[\frac{\varphi^n}{\sqrt{5}}\right] \text{ or } F_n = \left\lfloor \frac{\varphi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor, \text{ where } \varphi = \frac{1+\sqrt{5}}{2}, \quad (9)$$

faces the problem of cumulative computational error and is of little use. On the other hand, some other sequences, such as the Jacobsthal sequence, have a convenient formula

$$J_n = \frac{2^n - (-1)^n}{3}. \quad (10)$$

Also is known the formula writing to find elements of a Fibonacci sequence across a continuum of size *n* x *n*:

$$F_{n+1} = det \begin{vmatrix} 1 & 1 & 0 & \cdots & 0 \\ -1 & 1 & 1 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 1 \end{vmatrix}. \quad (11)$$

If the *n*-th element of the sequence equals to the sum of *k* previous elements

$$A_n = A_{n-1} + A_{n-2} + \cdots + A_{n-k} \qquad (12),$$

then such a sequence is written in the following matrix form

$$\begin{pmatrix} A_n \\ A_{n-1} \\ A_{n-2} \\ \vdots \\ A_{n-k+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}^{n-k} \cdot \begin{pmatrix} A_k \\ A_{k-1} \\ A_{k-2} \\ \vdots \\ A_1 \end{pmatrix}. \quad (13)$$

The matrix with the help of which the calculations will be made will be of dimension *k* x *k*.

Therefore, performing calculations to find members of sequences using the matrix form of their representation significantly reduces the time complexity of the corresponding algorithms.

2 Reducing program execution time by using the features of the programming language

When solving a problem, it is very important to use the features of the programming languages in which the solution is implemented. In particular, let us take the well-known problem of multiplying two matrices. Consider two implementations of this operation.

Table 1. Two implementations of multiplication of two matrices.

Well-known variant

```
for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
    for (int k = 0; k < n; k++)
      c[i][j] += a[i][k] * b[k][j];
```

Accelerated variant

```
for (int i = 0; i < n; i++)
  for (int k = 0; k < n; k++) {
    long long x = a[i][k];
    for (int j = 0; j < n; j++)
      c[i][j] += x * b[k][j];
  }
```

Table 1 describes the finding of the product of matrices **C** = **A** · **B**, all matrices of dimensions *n* x *n*. If in the well-known variant the second and third cycles are swapped and the element of the first matrix is fixed in the usual variable, then the multiplication operation rate for matrices of dimension 1000 x 1000 will increase more than 15 (!) times. As the dimension increases, the advantage of the accelerated version will increase further. Further improvement steps are possible [6], but they will not give such a tangible advantage.

Scientific problems [7] also require the use of a matrix recording form for a particular model. Another recommendation is to keep the values that are constantly repeated in the calculation in a regular array. In particular, the values of trigonometric functions, if there is enough memory to store them, should be stored in the array, not re-calculated every time. This will significantly reduce the running time of the program.

3 Reducing time complexity by looking for regularities.

Solving regularity search tasks often leads to the identification of known sequences. Consider the problem proposed by Oleksandr Mitsa at the 13th Open Student International Programming Olympiad named after S. O. Lebedev and V. M. Glushkov "KPI-OPEN 2018" [8]. The title of this task is "Counter Racing"" This task combines, under a completely new perspective, two tasks that are well known to the general public – the Joseph Flavius problem [9] and the task No 2808 taken from the well-known E-Olymp site [10], which is described as the Choriv counter.

3.1 The task

The legendary Shchek and Choriv decided to arrange a competition for their counters.

Shchek counter was created on the base of the story of Josephus Flavius, when *N* people are in a circle and every second person is taken out of the circle. The remaining person number will be the result of the counter. For example, when there are 5 people in a circle, people will be taken out in the order of their numbers – 2, 4, 1, 5 and the result will be number 3.

Choriv's counter was based on a completely different principle. He took the number *N* and wrote out in a row all the numbers from 1 to *N*. Then he cross out the numbers that are in odd positions. Further, he lined them up anew, but then crossed out those that are in even positions. These actions were repeated until one number remained, which would be the result. For example, for *N*=5, the numbers with odd numbers – 1, 3, 5 are first crossed out, then from remaining numbers – 2, 4 – the number, which is in the even position, is crossed out, that is, 4. Therefore, the result will be 2.

For the full objectivity of determining the winner, it was decided to compete counts for each natural value from 1 to *N*. If, as a result, for some value the result of the Shchek counter is greater than the result of the Choriv counter, the Shchek will receive one point, if less, one point will receive Choriv, in case of a draw – the current account will not change.

It is need to determine the game score for a given number ***N***.

Input format

Enter the number *N* ($1<N<10^{18}$).

Output *format*

Display the score of the competition.

Table 2. Example to the task.

Standard input	*Standard output*
10	3 6
100	48 51

Note. In the first example, the Shchek counter will win only at values 3, 5 and 7, at value 1 it will be a draw and in other cases the Choriv counter will win.

3.2 Solution of the problem

We first examine the regularities in the first counter. To do this, we use the scheme proposed in [9] and refine it. First, let's consider how one can reduce the dimension of the problem twice with an even value of *N*.

Table 3. Simulation of Joseph Flavius problem with an even value of *N*.

1	~~2~~	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
1		2		3		4		5	

Table 3 shows that the dimension of the problem has decreased by 2 times and the formula for the transition from old to new values will look like

$$T(N) = 2 * T\left(\frac{N}{2}\right) - 1. \quad (14)$$

For an odd value of N we use the same scheme and note that the value 1 in this case will never be a solution (Table 4).

Table 4. Simulation of Joseph Flavius problem with an odd value of *N*.

~~1~~	~~2~~	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11
		1		2		3		4		5

Again we see that the dimension of the problem has decreased by 2 times and it is easy to deduce the formula of transition from old to new values, which will look like

$$T(N) = 2 * T\left(\frac{N}{2}\right) + 1. \quad (15)$$

To summarize, we give a complete scheme of recalculation

$$T(N) = \begin{cases} 1, & if\ N = 1; \\ 2 * T\left(\frac{N}{2}\right) - 1, & if\ N - even; \\ 2 * T\left(\frac{N}{2}\right) + 1, & if\ N - odd. \end{cases} \quad (16)$$

But even this formula is not enough to solve the problem as a whole. Therefore, we write the solutions for both counters for values *N* from 1 to 30 (Table 5).

Table 5. Tables for preliminary research.

N	1	2	3	4
Counter 1	1	1	3	1
Counter 2	1	2	2	2
Winner	Draw	Choriv	Shchek	Choriv
N	5	6	7	8
Counter 1	3	5	7	1
Counter 2	2	6	6	6
Winner	Shchek	Choriv	Shchek	Choriv
N	9	10	11	12
Counter 1	3	5	7	9
Counter 2	6	6	6	6
Winner	Choriv	Choriv	Shchek	Shchek
N	13	14	15	16
Counter 1	11	13	15	1
Counter 2	6	6	6	6
Winner	Shchek	Shchek	Shchek	Choriv
N	17	18	19	20
Counter 1	3	5	7	9
Counter 2	6	6	6	6
Winner	Choriv	Choriv	Shchek	Shchek
N	21	22	23	24
Counter 1	11	13	15	17
Counter 2	6	22	22	22
Winner	Shchek	Choriv	Choriv	Choriv
N	25	26	27	28
Counter 1	19	21	23	25
Counter 2	22	22	22	22
Winner	Choriv	Choriv	Shchek	Shchek
N	29	30	31	32
Counter 1	27	29	31	1
Counter 2	22	22	22	22
Winner	Shchek	Shchek	Shchek	Choriv

From Table 5, one can make the following observation: in the first counter, for N, which is a degree of two, always the answer is 1, and for the following *N* the answer is incremented by 2. That is, if the closest to *N* degree of two is equal 2^k, then the answer is easily determined by formula

$$T(N) = 1 + 2(N - 2^k) \quad (17)$$

Also note that the number of values of the degree of two for the input values from 1 to 10^{18} is only 60.

Let us proceed to the analysis of the second counter. Table 5 shows that the number of answers is negligible. Moreover, it can be seen that for 1 the answer will be 1, then from 2 to 5 the answer will be 2, and from 22 and to the next value to be investigated - the answer will be 22.

Let's simulate this task and write down the values of the answers that occur in it. These will be the following values

1, 2, 6, 22, 86, 342, 1366, 5462, 21846, 87382, 349526, 1398102, 5592406, 22369622,…

With these values in front of you, it is easy to determine the scheme of their calculation

$$P(k) = 4*P(k\text{-}1) - 2, \text{ where } P(1) = 1. \quad (18)$$

It is also understood that in the interval $[P(k), P(k+1)-1]$ the answer will be $P(k)$. Moreover, there will be very few such values. So, in the interval from 1 to 1018 there will be only 31 of them.

Thus, one of the schemes of the solution could be the following. At each interval from the number $N = 2^k+1$ to $N = 2^{k+1}$ we see how many numbers of the sequence 2 it contains, and accordingly we consider this when forming the account together with the sequence 1. Of course, the last interval will go only to the number *N*.

The described option will have the following solution in C++ programming language:

```
#include <iostream>

using namespace std;
```

```
int main() {
  long long N;
  cin >> N;
  long long p = 1, q = 2, Choriv = 0;
  while (2 * p + 1 <= N) {
    p = 2 * p + 1;
    if (p > 4 * q - 2) {
      long long pp = p;
      while (pp > 4 * q - 3)
        pp = (pp - 1) / 2;
      pp = 2 * (4 * q - 3 - pp) - 1;
      if (pp > q)
        Choriv += (pp - q) / 2 + 1;
      q = 4 * q - 2;
    }
    Choriv += (p - q) / 2 + 1;
  }
  if (N >= 4 * q - 2) {
    long long pp = p;
    while (pp > 4 * q - 3)
      pp = (pp - 1) / 2;
    pp = 2 * (4 * q - 3 - pp) - 1;
    if (pp > q)
      Choriv += (pp - q) / 2 + 1;
    q = 4 * q - 2;
  }
  p = 2 * (N - p) - 1;
  if (p > q) Choriv += (p - q) / 2 + 1;
  cout << Choriv << " " <<
          N - Choriv - 1 << endl;
  return 0;
}
```

But if to continue the research, one can get a simpler way of solving the problem under consideration.

Note that in this game only when $N = 1$, both players will draw. With all other values, N wins either the first or the second. So let's translate the game results to 0-1 form. Write the sequence in which the i-th element is 1 if the second player wins and 0 - otherwise, starting with the game for $N = 2$:

1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

Then we write down the quantities of consecutive numbers:

1, 1, 1, 1, 1, 1, 3, 5, 3, 3, 5, 5, 11, 21, 11, 11, 21, 21, ...

The next step is to divide this sequence into blocks of 6 elements and notice that each block consists of two sequential Jacobshtal numbers

1, 1, 3, 5, 11, 21, 43, 86, 171, ...

That is, the new scheme for solving this problem will be as follows. Each time, by choosing two values of the Jacobshtal sequence, we will form a current account. Of course, in forming the final account we will use only those values in the last six, which are limited by the number of lots N.

For example, we calculate the score for the number of lots $N = 40$. Note that this will require the use of the first two sixths and part of the first value of the third sixth. The score after 31 games will be described by summing the elements of the first two sixes. To count the number of games won by the first player, we count the items in even positions $1 + 1 + 1 + 5 + 3 + 5 = 16$, and to count the number of games won by the second player, we count the items in odd positions $1 + 1 + 1 + 3 + 3 + 5 = 14$. Then we consider that the third six starts with 11 second player victories, of which we need to count 9. That is, the final score will be 16:23 in favor of the second player.

The described solution in C++ programming language will be quite simple and compact:

```
#include <iostream>

using namespace std;

long long J[61], N, Choriv, Shchek, rem;

void Score (long long &X, long long Y) {
  if (rem > Y) {
    X += Y; rem -= Y;
  }
  else {
    X += rem; rem = 0;
  }
}

int main() {
  cin >> N;
  int i = 1;
  J[1] = 1; J[2] = 1;
  while(Shchek+Choriv+3*(J[i]+J[i+1])<N) {
    Shchek += 2 * J[i] + J[i + 1];
    Choriv += J[i] + 2 * J[i + 1];
    i += 2;
    J[i] = J[i - 1] + 2 * J[i - 2];
    J[i + 1] = J[i] + 2 * J[i - 1];
  }
  rem = N - 1 - Shchek - Choriv;
  Score(Shchek, J[i]);
  Score(Choriv, J[i + 1]);
  Score(Shchek, J[i]);
  Score(Choriv, J[i]);
  Score(Shchek, J[i + 1]);
  Score(Choriv, J[i + 1]);
  cout << Choriv << " " << Shchek;
  return 0;
}
```

It should be noted that in this task the number of involved Jakobshtal numbers does not exceed 60. The Score procedure allows to realize an ending of the task when the last block is not fully used.

Thus, the considered problem (of increased complexity) used in the international competition is a combination of two well-known tasks under a completely new perspective that has not been used before. Two variants of its solution are presented. In a more flexible and efficient way, it was enough to find the regularity given by the elements of the Jacobsthal sequence.

Conclusions

This paper discusses three approaches to reducing solution execution times for computer science tasks that require some knowledge of sequences and / or arrays to solve them. The first approach is to write the sequence in the matrix form and then use the rapid matrix exponentiation. This allows to quickly identify a particular element of a sequence. The second approach is essentially to improve the code of the program, which is considered traditional, and can significantly speed up the program, significantly (for matrices 1000 by 1000 more than 15 times) reducing the time of finding matrices using a fairly simple method. This approach is tested for C++, the most popular sports programming language. It is effective in solving sports programming tasks, because in this area, rapid methods of matrix multiplication are rarely used due to the excessive size of their code. It is also very important to know for scientists who write code for scientific researches and are faced with matrix multiplication operations. To demonstrate the third approach, we present a rather complex authorial task and show that its solution can be based on finding members of the well-known Jacobsthal sequence. The approaches presented in the paper can be used both individually and in combination.

The work will be interesting to pupils, students and teachers interested in programming, especially sports, and for scientists who write code for scientific researches.

References

1. S.S. Zhukovsky, Methodical Aspects of Preparing Gifted Schoolchildren for Informatics Olympiads. Problems of modern pedagogical education **47**(V), 62–69 (2015)
2. Y.V. Horoshko, O.V. Mitsa, V.I. Melnyk, Information Technologies and Learning Tools **71**(3), 40–52 (2019)
3. P. Barry, Irish Mathematical Society Bulletin **51**, 45–57 (2003)
4. Z. Čerin, Journal of Integer Sequences **10**(07.2), 5 (2003)
5. D.E. Knuth, *The Art of Computer Programming, Volumes 1-4A Boxed Set*, 3rd edn (Addison-Wesley, Reading, 2011)
6. I. Ermolaev, Umnozhenie matritc: effektivnaia realizatciia shag za shagom (Matrix multiplication: effective implementation step by step) (2019), https://habr.com/ru/post/359272/. Accessed 21 Mar 2019
7. P.I. Stetsyuk, Ellipsoid methods and r-algorithms (Evrika, Chisinau, 2014).
8. 13th Open International Student Programming Olympiad “KPI-OPEN 2018” named after S.O. Lebediev and V.M. Glushkov “KPI-OPEN 2018” (2018), http://kpi-open.org/. Accessed 28 Nov 2019
9. R.L. Graham, D.E. Knuth, O. Patashnik, *Concrete Mathematics*, 2nd edn (Addison-Wesley, Reading, 1994)
10. Internet portal of organizational and methodological support of distance Olympiads on programming for gifted youth of educational institutions of Ukraine E-Olimp (2019), https://www.e-olymp.com/uk/problems/2808. Accessed 28 Nov 2019