# Membership Testing on Monotonic Linear Recurrence Relation Using Matrix Exponentiation and Decrease and Conquer Algorithm

Farizki Kurniawan - 13521082
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521082@std.stei.itb.ac.id

***Abstract*— Linear recurrence relations are mathematical objects that describe a sequence of numbers in terms of its previous terms. In this paper, an algorithm based on matrix exponentiation and decrease and conquer approach is proposed to solve one of the problem regarding these relations, namely membership testing, where there is a need to determine whether a given number belongs to the sequence generated by a particular recurrence relation. The algorithm works specifically on monotonic linear recurrence relation, where the relation should be either increasing or decreasing constantly, to ensure that the decrease and conquer part of the algorithm works as intended.**

***Keywords—membership testing; linear recurrence; matrix exponentiation; decrease and conquer***

## I. INTRODUCTION

Linear recurrence relations are mathematical objects that describe a sequence of numbers in terms of its previous terms. They have wide applications in various areas, such as computer science, cryptography, and physics. A problem in linear recurrence relations is to determine whether a given number belongs to the sequence generated by a particular recurrence relation, called membership testing.

In this paper, an algorithm to test whether or not a given number is a member of a recurrence relation is proposed. This algorithm combines two well-known algorithms, namely matrix exponentiation and decrease and conquer (DnC). Matrix exponentiation is used to compute the nth term of the sequence generated by the recurrence relation, while decrease and conquer is used to reduce the computation time by recursively dividing the exponent into smaller subproblems. However, due to the nature of the decrease and conquer algorithm, the linear recurrence relation needs to be monotonic, i.e. every elements of the recurrence relation needs to be strictly larger or smaller than the element before it.

Later on, an analysis of the proposed algorithm is detailed, including its time complexity and space requirements. An example of the algorithm being used is also given, both for the searching the n-th term of the monotonic linear recurrence relation and the algorithm of membership testing of a given number in the linear recurrence relation.

## II. THEORETICAL FRAMEWORK

### A. Linear Recurrence Relation

A reccurence relation is a mathematical relationship expressing $f_n$ as some combination of $f_i$ with $i < n$. When formulated as an equation to be solved, recurrence relations are known as recurrence equations, or sometimes difference equations [1].

A linear recurrence equation is a recurrence equation on a sequence of numbers $x_n$ expressing $x_n$ as a first-degree polynomial in $x_k$ with $k < n$ [2]. An example of such equation is as follows.

$$x_n = Ax_{n-1} + Bx_{n-2} + Cx_{n-3} + \dots$$

A monotonic recurrence relation is a type of recurrence relation where the terms of the sequence either consistently increase or consistently decrease as the index of the sequence increases. In a monotonic increasing recurrence relation, each term of the sequence is greater than or equal to the previous term.

Mathematically, if we have a recurrence relation where $a_n$ represents the nth term of the sequence, the sequence is called monotonic increasing if

$$a_n \geq a_{n-1}$$

for all n.

On the other hand, the sequence is called monotonic decreasing if

$$a_n \leq a_{n-1}$$

for all n.

### B. Membership Testing

Membership testing is the problem of testing whether or not an element is in a set of elements [3]. This set of elements may be stated may be either finite or infinite. Finite set of elements are usually elements with clearly stated boundaries and explanation on what the members are.

On the other hand, an infinite set is usually generated with some kind of functions or languages with predetermined rules on determining the members of the set.

## C. *Matrix Exponentiation*

The concept of matrix exponentiation in its most general form is very useful in solving questions that involve calculating the term of a linear recurrence relation in time of the order of $\log_2 n$ [4]. Matrix exponentiation is useful in solving these linear recurrence relation due to the fact that multiple variables can be calculated at once in a matrix, and also the fact that exponentiating a matrix will cost less in terms of multiplication rather than iterative solving makes it an efficient choice for solving linear recurrence relation problems.

Matrix exponentiation uses the concept of divide and conquer to make the amount of matrix multiplication operations as minimum as possible. For example, to calculate the value of a matrix exponentiated to the power of 16, instead of multiplying the matrix by itself 16 times, it calculates power of 16 as power of 8 times power of 8, then power of 8 as multiplication of power of 4s, all the way to multiplication of the original matrices, resulting in about $\log_2 n$ number of matrix multiplication.

## D. *Decrease And Conquer Algorithm*

Decrease and conquer algorithms are algorithms which attempt to reduce a problem into smaller sub-problems to finally compute only one sub-problem. This approach can be seen as a modification to the more popular divide and conquer algorithms, which divide a problem into smaller subproblems, processing both, and combining the solutions of each sub-problem [5].

Decrease and conquer algorithms have two steps, which are decrease and conquer. The first step of the algorithm, decrease, is about reducing a problem into smaller sub-problems. On the other hand, the conquer step is where the algorithm processes only one of the divided subproblem. There is no “combine” step in the decrease and conquer algorithm, as there is only one processed sub-problem [6].

There are three variants of decrease and conquer:

1. Decrease by a constant: the problem is reduced by a constant value in each iteration. Some examples include insertion sort and selection sort.
2. Decrease by a constant factor: the problem is reduced by a constant factor in each iteration. Some examples include binary search and fake-coin problems.
3. Decrease by a variable size: the problem is reduced by different amounts in each iteration. Some examples include Euclid’s algorithm and selection by partition.

The algorithm designed in this paper will take the second variant, decrease by a constant factor.

## III. Algorithm to Calculate the N-th Term of a Linear Recurrence Relation

In this paper, the linear recurrence solved is going to have the form of

$$f_n = \sum_{i=1}^{k} c_i * f_{n-i}$$

We can construct a k*k matrix T:

$$T = \begin{pmatrix} 0 & 1 & 0 & 0 & . & . \\ 0 & 0 & 1 & 0 & . & . \\ 0 & 0 & 0 & 1 & . & . \\ . & . & . & . & . & . \\ c_k & c_{k-1} & c_{k-2} & . & . & c_1 \end{pmatrix}$$

And k*1 matrix F

$$F = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ . \\ . \\ . \\ f_{k-1} \end{pmatrix}$$

Where T*F equals to

$$T * F = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ . \\ . \\ . \\ f_k \end{pmatrix}$$

To calculate the n-th term of the recurrence relation using this method, we can calculate the matrix

$$C_n = T^n * F$$

where the n-th term is the first term of the matrix $C_n$ [4]. Then, using the divide and conquer method, we can easily get $T^n$ using the fact that

$$T^n = \begin{cases} T^{\frac{n}{2}} * T^{\frac{n}{2}}, & \text{if } n \text{ is even} \\ T^{n-1} * T, & \text{if } n \text{ is odd} \end{cases}$$

We can find the matrix $T^n$ in around $\log_2 n$ matrix multiplications. Given that each matrix multiplication requires around $k^3$ operations, the time complexity of calculating the n-th term of a linear recurrence relation is about $O(k^3 \log_2 n)$.

As an example, suppose we are searching for the 10-th term of the Pell sequence where:

$$P_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ 2P_{n-1} + P_{n-2} & \text{otherwise.} \end{cases}$$

The steps to calculate the 10-th term is as follows:

1. Construct the 2 * 2 matrix T as follows

$$T = \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix}$$

2. Calculate $T^{10}$, notice that $T^{10} = T^5 * T^5$ , where $T^5 = T^4 * T$, $T^4 = T^2 * T^2$ and $T^2 = T * T$. Then, the value of $T^{10}$ is obtained as

$$T^{10} = \begin{pmatrix} 985 & 2378 \\ 2378 & 5741 \end{pmatrix}$$

3. Construct the matrix F as follows

$$F = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

4. Calculate C10 as follows

$$C_{10} = \begin{pmatrix} 985 & 2378 \\ 2378 & 5741 \end{pmatrix} * \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

where we got the value of C10 as

$$C_{10} = \begin{pmatrix} 2378 \\ 5741 \end{pmatrix}$$

5. The 10-th term of the recurrence relation is then obtained as the first value of C10. Thus, the following value is obtained.

$$P_{10} = 2378$$

From the explanation before, we can see that the algorithm to get th n-th term of the linear recurrence relation works by using the concept of matrix exponentiation, where we find the n-th power of a matrix using a divide and conquer approach. The value of the matrix is then used to be multiplied with a starting state values of the recurrence relation to finally get the correct term of the linear recurrence.

## IV. Algorithm to Test Membership of a Given Number in a Monotonic Linear Recurrence Relation

The algorithm starts with a state of testing whether or not the given number is the first element of the linear recurrence relation. If the number is not the first element of the linear recurrence relation, then we can check the second element of the linear recurrence relation to see if the linear recurrence is increasing or decreasing.

If the number is still not found as a member in the linear recurrence relation at this point, we should check whether the given number may appear in the later elements of the sequence. We can check from the first element and rule out the possibilities of the number appearing if one of the followings is true:

1. The monotonic linear recurrence relation is increasing and the given number is smaller than the first element, or
2. The monotonic linear recurrence relation is decreasing and the given number is larger than the first element.

If it is shown that the given number may appear in the later indices of the linear reccurence relation, then we may proceed to make the set to be tested larger by increasing the checked number of the linear recurrence relation by a constant multiplier until either one of the three conditions is met:

1. The monotonic linear recurrence relation is increasing and the current checked index of the recurrence relation is larger than the given number, or
2. The monotonic linear recurrence relation is decreasing and the current checked index of the recurrence relation is smaller than the given number, or
3. The current checked index of the recurrence relation is same as the given number.

At this point, if we found a number in the recurrence relation to be the exact same as the given number, we can proceed to declare that the given number is, in fact, a member of the recurrence relation. If not, then we can proceed with the next part of the algorithm.

When the testing state is finalized, i.e. we know that the given number may only be a member in that state, then we can proceed to implement a decrease and conquer technique to find whether the given number is a member in the monotonic linear recurrence relation. The algorithm used is as follows:

1. Establish three variables, namely left, middle, and right, with left being the number one and right being the number of the largest index of the testing state.
2. Calculate middle as the midpoint between left and right.
3. Check whether the middle's element of the linear recurrence relation is the given number. If not, check whether the given number should be in the leftside of the midpoint or the rightside of the midpoint. We can

know whether a given number may appear in the leftside or the rightside of the midpoint with rules as follows:

a. If the monotonic linear recurring relation is increasing, the given number may appear on the leftside of the midpoint if the given number's value is smaller than the midpoint's value. Otherwise, it may lie on the rightside of the midpoint.

b. If the monotonic linear recurring relation is decreasing, the given number may appear on the leftside of the midpoint if the given number's value is larger than the midpoint's value. Otherwise, it may lie on the rightside of the midpoint.

4. If the given number may lie on the leftside of the middle point, change the value of right to middle-1. Otherwise, change the value of left to middle+1.

5. If the given number is already found in the linear recurrence relation declare the given number as a member of linear recurrence relation and stop the algorithm. On the other hand, if the size of the testing state is one and the given number is not found, it is declared as not a member of the linear recurrence relation. Otherwise, go to step 2 of the algorithm.

As an example, suppose the algorithm attempts to check the membership of the number 6765 in the linear recurrence with the formula of

$$F_n = \begin{cases} 0, & \text{for } n = 0 \\ 1 & \text{for } n = 1 \\ F_{n-1} + F_{n-2} & \text{for } n > 1 \end{cases}$$

The following are the steps taken to test the membership of the given number:

1. Check the first number of the recurrence relation. $F_1 = 1$ and the given number 6765 may occur in the later index of the sequence since the relation is increasing.
2. Expand the testing size with constant multiplier of 2.

   $F_2 = 2 < 6765$, expand space by a factor of 2.
3. $F_4 = 3 < 6765$, expand space by a factor of 2.
4. $F_8 = 21 < 6765$, expand space by a factor of 2.
5. $F_{16} = 987 < 6765$, expand space by a factor of 2.
6. $F_{32} = 2178309 > 6765$, the testing state is finalized.
7. Establish left = 1, right = 32, and mid = 16.
8. Check $F_{mid}. F_{16} = 987 < 6765$, the given number may appear on the rightside of the midpoint.
9. Change left's value to mid+1. left <- 17.
10. Calculate new mid index = 24.
11. Check $F_{mid}. F_{24} = 46368 > 6765$, the given number may appear on the leftside of the midpoint.
12. Change right's value to mid-1. right <- 23.
13. Calculate new mid index = 20.
14. Check $F_{mid}. F_{20} = 6765 = 6765$.
15. It is concluded that 6765 is a member of the linear recurrence given $F_n = F_{n-1} + F_{n-2}$ *with* $F_0 = 0$ *and* $F_1 = 1$.

Another example is the case where the given number is not a member of the given linear recurrence relation. For example, suppose the algorithm attempts to check the membership of the number 600 in the linear recurrence with the formula of

$$L_n = \begin{cases} 2, & \text{for } n = 0 \\ 1 & \text{for } n = 1 \\ L_{n-1} + L_{n-2} & \text{for } n > 1 \end{cases}$$

The following are the steps taken to test the membership of the given number:

1. Check the first number of the recurrence relation. f1=1 and the given number 600 may occur in the later index of the sequence since the relation is increasing.
2. Expand the testing size with constant multiplier of 2.

   $L_2 = 3 < 600$, expand space by a factor of 2.
3. $L_4 = 7 < 600$, expand space by a factor of 2.
4. $L_8 = 47 < 600$, expand space by a factor of 2.
5. $L_{16} = 2207 > 600$, the testing space is finalized.
6. Establish left = 1, right = 16, and mid = 8.
7. Check $L_{mid}. L_8 = 47 < 600$, the given number may appear on the rightside of the midpoint.
8. Change left's value to mid+1. left <- 9.
9. Calculate new mid index = 12.
10. Check $L_{mid}. L_{12} = 322 < 600$, the given number may appear on the rightside of the midpoint.
11. Change left's value to mid+1. left <- 13.
12. Calculate new mid index = 14.
13. Check $L_{mid}. L_{14}$ = 843 > 600, the given number may appear on the leftside of the midpoint.
14. Change right's value to mid-1. right <- 13.
15. Check $L_{mid}. L_{13}$ = 233 != 600 and the testing state size is <2.
16. It is concluded that 600 is not a member of the linear recurrence given $Ln = Ln - 1 + Ln - 2$ with $L_0 = 2$ *and* $L_1 = 1$.

From the examples, we can see that the algorithm starts with expanding the testing state first, until we can be sure that the given number may only lie inside the established testing state. This is done so that the decrease and conquer algorithm used afterwards can guarantee whether or not the given number is a member of the monotonic linear recurrence relation.

## V. Complexity Analysis

### A. Time Complexity

The time complexity analysis can be divided into the two parts of the algorithm, namely matrix exponentiation algorithm and decrease and conquer algorithm. The relation between the two algorithm is that the decrease and conquer algorithm uses the matrix exponentiation in its part to calculate the n-th term of the linear recurrence relation.

The time complexity of the matrix exponetiation first comes from the multiplication of the matrices with complexity of $O(k^3)$ where k is the number of row of the matrix being multiplied. This matrix multiplication will then be done with the complexity of $O(\log_2 n)$ due to divide and conquer, where n equals to the term that is currently being searched. Thus, the final time complexity of the matrix exponentiation is $O(k^3 \log_2 n)$.

The time complexity of the decrease and conquer first comes from the expanding the state size to accommodate the given number in the linear reccurence. This takes the complexity of $O(k^3 \log_2 n)$. * $O(\log_2 n)$ or equals to $O(k^3 \log_2^2 n)$ where $O(k^3 \log_2 n)$ is obtained from the complexity of the matrix exponentiation used for every term checked. The variable n refers to the minimum number needed to accommodate the given number and k refers to the highest value in the recurrence relation formula $x_n$.

The decrease and conquer then takes another time complexity of $O(k^3 \log_2 n)$* $O(\log_2 n))$ or equals to $O(k^3 \log_2^2 n)$ to search for the given number in the recurrence relation. This is due to the algorithm using matrix exponentiation and the narrowing-down of the decrease and conquer in the later parts.

Thus, the total time complexity of the algorithm is $O(k^3 \log_2^2 n)$, where n refers to the minimum number needed to accommodate the given number and k refers to the highest value in the recurrence relation formula $x_n$.

However, it needs to be noted that the time complexity may get worse for larger number higher than the 64-bit maximum value due to the operations needed for such number gets more and more time consuming as it gets further from the value. Thus, in reality, one can not expect the algorithm to run perfectly in $O(k^3 \log_2^2 n)$ as the constant gets larger for higher numbered values.

### B. Space Complexity

The space complexity of this algorithm comes from storing the data of the matrix being used for matrix exponentiation. Approximately, about $\log_2 n$ of k*k matrices and two k*1 matrix are needed for this algorithm to work.

Around $\log_2 n$ k*k matrices are needed for storing the value of the matrices for every power of two if needed, and two k*1 matrix is needed for storing the starting values of the linear recurrence term and the needed values of the linear recurrence term.

Thus, the total space complexity needed for this algorithm is $O(k^2 \log_2 n)$. An important thing to note is for number larger than the maximum 64-bit value, the constant of the space complexity gets larger and the program can not be expected to run in aforementioned space complexity.

## VI. Testing

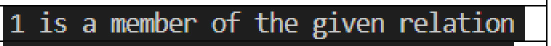
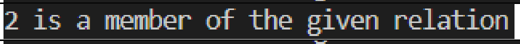
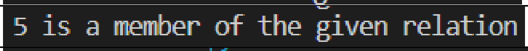
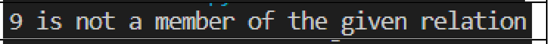
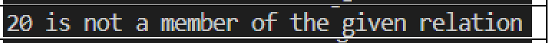
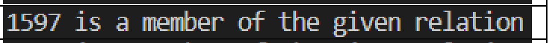
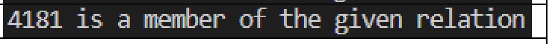
The following shows the results of the program for some known monotonic linear recurrence relation.

### A. Fibonacci Sequence

A fibonacci sequence is defined as follows.

$$F_n = \begin{cases} 0, & \text{for } n = 0 \\ 1 & \text{for } n = 1 \\ F_{n-1} + F_{n-2} & \text{for } n > 1 \end{cases}$$

The result of testing a number's membership in the fibonacci sequence is as follows.

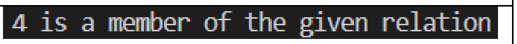
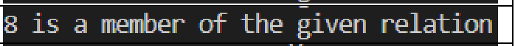
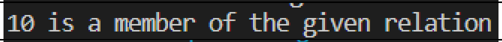
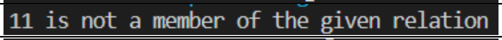
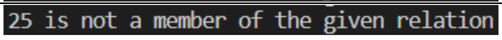
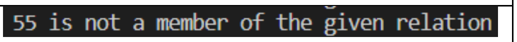
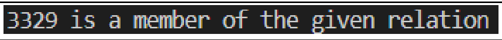
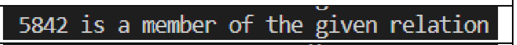
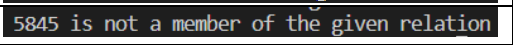
| Given Num ber | Is A Mem ber | Resullt |
|---|---|---|
| 1 | Yes |  1 is a member of the given relation |
| 2 | Yes |  2 is a member of the given relation |
| 3 | Yes | 3 is a member of the given relation |
| 5 | Yes |  5 is a member of the given relation |
| 9 | No |  9 is not a member of the given relation |
| 20 | No |  20 is not a member of the given relation |
| 100 | No | 100 is not a member of the given relation |
| 1597 | Yes |  1597 is a member of the given relation |
| 4181 | Yes |  4181 is a member of the given relation |
| 4182 | No | 4182 is not a member of the given relation |

### B. Padovan Sequence

A padovan sequence is defined as follows.

$$P_n = \begin{cases} 1, & \text{if } 0 \leq n \leq 2 \\ P_{n-2} + P_{n-3}, & \text{if } n \geq 3 \end{cases}$$

The result of testing a number's membership in the padovan sequence is as follows.

| Given Number | Is A Member | Resullt |
|---|---|---|
| 4 | Yes |  4 is a member of the given relation |
| 6 | Yes | 6 is a member of the given relation |
| 8 | Yes |  8 is a member of the given relation |
| 10 | Yes |  10 is a member of the given relation |
| 11 | No |  11 is not a member of the given relation |
| 25 | No |  25 is not a member of the given relation |
| 55 | No |  55 is not a member of the given relation |
| 3329 | Yes |  3329 is a member of the given relation |
| 5842 | Yes |  5842 is a member of the given relation |
| 5845 | No |  5845 is not a member of the given relation |

## VII. Conclusion

From the explanations and testing carried out, it can be seen that the algorithm proposed in this paper can solve the membership testing problem of a given number in a monotonic linear recurrence relation. This algorithm uses the concept of matrix exponentiation and decrease and conquer algorithm and have the time complexity of $O(k^3 \log_2^2 n)$ and space complexity of $O(k^2 \log_2 n)$, where n refers to the minimum index whose value is enough to accommodate the given number and k refers to the highest value in the recurrence relation formula $x_n$.

## Acknowledgement

The author would like to express gratitude to Dr. Nur Ulfa Maulidevi, S.T., M.Sc. for her guidance and lessons throughout the semester as class lecturer for IF2211 Algorithm Strategies course which insights have been invaluable to the completion of this project.

## References

[1] Weisstein, Eric W. "Recurrence Relation." From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/RecurrenceRelation.html

[2] Weisstein, Eric W. "Linear Recurrence Equation." From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/LinearRecurrenceEquation.html

[3] Yanqing Peng, Jinwei Guo, Feifei Li, Weining Qian, Aoying Zhou. ”Persistent Bloom Filter: Membership Testing for the Entire History” University of Utah, East China Normal University

[4] Rohan Bhandari, “A Complete Guide on Matrix Exponentiation” https://codeforces.com/blog/entry/67776.

[5] R. Munir, “Algoritma Decrease and Conquer.” Available at: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Decrease-and-Conquer-2021-Bagian1.pdf (Accessed May 21, 2023)

[6] “Decrease and Conquer - GeeksforGeeks.” https://www.geeksforgeeks.org/decrease-and-conquer/ (Accessed May 22, 2023).

## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

[illegible]

Farizki Kurniawan 13521082