Synthetic Division for Evaluating Polynomials

Wesly Giovano - 13520071
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13520071@std.stei.itb.ac.id

***Abstract*—Polynomial function is used a lot in many fields, and most of the time it is needed to be evaluated. As the degree of polynomial increases, we need a more efficient way to evaluate high-degree polynomial. Polynomial evaluation can be approached by brute-force method which is an intuitive method, however it gives time complexity of $O(n^2)$. Another brute-force method but enhanced one gives a better time complexity of O(n log n). In this paper, we use divide-and-conquer approach to evaluate polynomial synthetic division as a method to evaluate polynomial, and analysis shows it has time complexity of O(n) which is more efficient than both former methods. Synthetic division method to evaluate polynomial can be implemented in either recursive or iterative approach with the idea of alternating between multiplying the value of indeterminate and adding the next coefficient.**

Keywords—polynomial evaluation; divide and conquer algorithm; synthetic division

I. Introduction

Polynomial is one of the most widely-used expression in solving problems from many fields, ranging from engineering to finance. For example, aerospace engineers have to determine the acceleration of a rocket based on variables such as gravitation and mass, while financiers would use polynomial as a mean to forecast market trends and make decision based on it.

Sometimes, polynomials are needed to be evaluated for a large number of variable/indeterminate values. With the method of solving the arithmetic operations sequentially—also named as brute-force method—it would be less time-efficient as we need to calculate the power of the indeterminate repeatedly. One method is to "memorize" the values for each power of the indeterminate. However, the method would require memory to store those values for computational calculation and writing those values elsewhere before proceeding to the polynomial evaluation for manual calculation. In this paper, we would discuss about synthetic division as a mean to evaluating polynomial in a more efficient way, usable for both manual calculation and computational calculation.

II. Theoretical Framework

A. Divide-and-Conquer Algorithm

Divide and conquer is an algorithm design paradigm of solving a large problem by solving subproblems from the breakdowns of the initial problem [1]. The approach has three parts: (1) divide, that is dividing the problem into subproblems, (2) conquer, that is solving/conquering each of the subproblems by either solving it directly if it is simple enough to be solved or dividing it again into smaller subproblems, and (3) combine, that is combining solutions of all the subproblems into one solution to the initial problem. Below is the illustration to the approach.

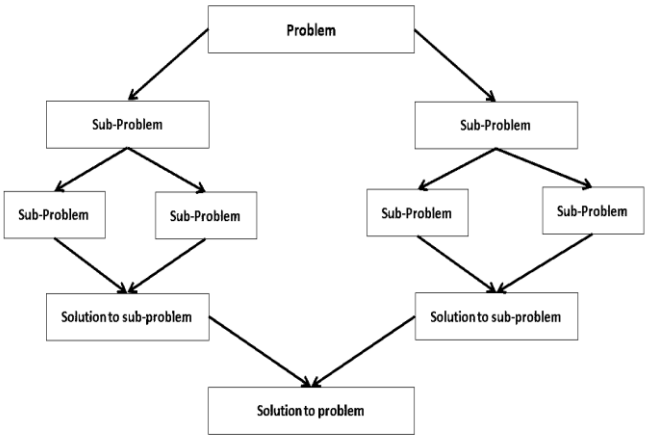


Fig. 1. Illustration to divide-and-conquer algorithm.

Time complexity to the divide-and-conquer algorithm is generalized as [2]

$$T(n) = \begin{cases} g(n), & n \le n_0 \\ T(n_1) + T(n_2) + \cdots + T(n_r) + f(n), & n > n_0 \end{cases} \quad (1)$$

where:

- $T(n)$ is the time complexity to solve problem of size n,
- $T(n_i)$ is the time complexity to solve each of the subproblems,
- $g(n)$ is the time complexity to solve a small-enough subproblem, and
- $f(n)$ is the time complexity to combine solution of the subproblems.

Recursive time complexity in the divide-and-conquer algorithm can be solved with Master Theorem as a shortcut, which generally is as follows [2].

For a monotonically increasing function $T(n) = a\,T\left(\frac{n}{b}\right) + cn^d$ that satisfies the constraints $a \geq 1$, $b \geq 2$, and $c, d \geq 0$, then the non-recursive form of $T(n)$ in Big-O notation is

$$T(n) = \begin{cases} O(n^d), & a < b^d \\ O(n^d \log n), & a = b^d \\ O\left(n^{\log_b a}\right), & a > b^d \end{cases} \quad (2)$$

B. Polynomial and Division Algorithm

A function p is a polynomial if

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \quad (3)$$

or more compactly,

$$p(x) = \sum_{i=0}^{n} a_i x^i \quad (4)$$

where n is a nonnegative integer and a_i are the coefficients of the polynomial [3]. The polynomial is said to have a degree of n.

Arithmetic operations can be applied to polynomials, such as addition, subtraction, multiplication, and division between two polynomials. The division algorithm in polynomial operation is

$$f(x) = d(x)q(x) + r(x) \quad (5)$$

where $f(x)$ is the dividend, $d(x)$ is the divisor, $q(x)$ is the quotient, and $r(x)$ is the remainder [3]. If $r(x) = 0$ then the divisor $d(x)$ is said to divide evenly into the dividend $f(x)$. The remainder $r(x)$ also has the property of having lower degree than $d(x)$. Hence, if $d(x)$ has a degree of 1, $r(x)$ is a constant.

Two methods exist to solve division problem manually: long division and synthetic division. The latter is a shortcut for the former by divisors in form of $x - k$. Let $f(x) = ax^3 + bx^2 + cx + d$ and $d(x) = x - k$, then the method of synthetic division is as follows. Note that we are adding terms in the vertical pattern and multiplying by k in the diagonal pattern.

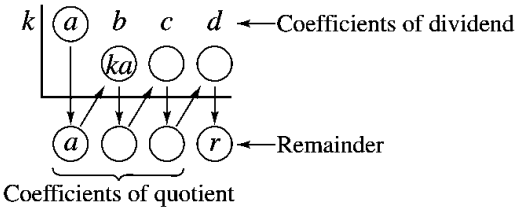


Fig. 2. Method of synthetic division.

III. Polynomial Evaluation Algorithm

In this paper, we are interested in number of multiplication but not addition and subtraction since the calculation speed of multiplication is comparatively slow. Hence, time complexity of the algorithms only represents multiplication time and does not include addition and subtraction time.

A. Brute-Force Method

The brute-force method of polynomial evaluation is evaluating polynomial according to the order of operations such as PEMDAS rule (parentheses, exponents, multiplications and divisions, additions and subtractions). Let us the example of the polynomial $p(x) = 2x^3 - 7x^2 + 6x + 2$, then the value of $p(3)$ is as follows with brute-force method.

$$\begin{aligned} p(3) &= 2(3)^3 - 7(3)^2 + 6(3) + 2 \\ &= 2(3 \times 3 \times 3) - 7(3 \times 3) + 6(3) + 2 \\ &= 2(27) - 7(9) + 6(3) + 2 \\ &= 54 - 63 + 18 + 2 \\ &= 11 \end{aligned}$$

The time complexity for this method for a polynomial in form of one in equation (4) is shown as follows.

(1) Time complexity for calculating power of a number.

$$T_{pow}(n) = n \quad (6)$$

(2) Time complexity for evaluating polynomial.

$$T(n) = \sum_{i=0}^{n} (1 + T_{pow}(i)) = \sum_{i=0}^{n} (1 + i)$$

$$T(n) = \tfrac{1}{2}(n+1)(n+2) \quad (7)$$

From equation (7), we can conclude that the brute-force method has time complexity of $O(n^2)$.

B. Enhanced Brute-Force Method

The enhanced brute-force method takes the divide-and-conquer approach on calculating power of a number. Let us calculate a^n with mentioned approach [2].

(1) For $n = 0$, $a^n = 1$.

(2) For $n > 0$,

a. If n is even, then $a^n = a^{\frac{n}{2}} \times a^{\frac{n}{2}}$.

b. If n is odd, then $a^n = a^{\lfloor \frac{n}{2} \rfloor} \times a^{\lfloor \frac{n}{2} \rfloor} \times a$.

Time complexity for this method is shown as follows.

(1) Time complexity for calculating power of a number.

$$T_{pow}(n) = \begin{cases} 0, & n = 0 \\ T_{pow}\left(\frac{n}{2}\right) + 1, & n > 0 \text{ and } n \text{ is even} \\ T_{pow}\left(\lfloor \frac{n}{2} \rfloor\right) + 2, & n > 0 \text{ and } n \text{ is odd} \end{cases} \quad (8)$$

With the Master Theorem in equation (2), we can transform equation (8) into

$$T_{pow}(n) = O(\log n) \tag{9}$$

(2) Time complexity for evaluating polynomial.

$$T(n) = \sum_{i=0}^{n} (1 + T_{pow}(i)) = \sum_{i=0}^{n} (1 + O(\log n))$$

$$T(n) = n \times O(\log n) = O(n \log n) \tag{10}$$

The enhanced brute-force method is proven better than the original brute-force method with time complexity of $O(n \log n)$.

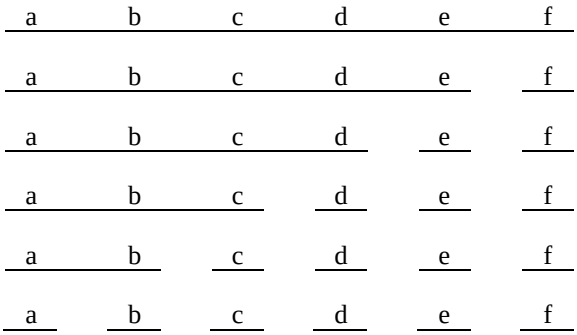
C. Synthetic Division Method

The synthetic division method uses the remainder of the polynomial $p(x)$ divided by $x - k$ to calculate $p(k)$ in a recursive way. This method has also been supported by polynomial remainder theorem. For example, in Fig. 2, we can show that $r = d + k\big(c + k(b + ka)\big) = ak^3 + bk^2 + ck + d$.

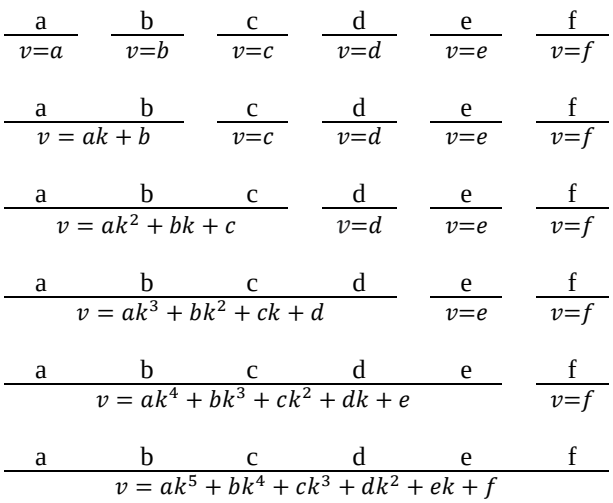
The divide-and-conquer approach to polynomial evaluation, i.e., synthetic division method, is illustrated as follows.

Let the polynomial be $p(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$ and we are to calculate the value of $p(k)$.

(1) Divide and conquer.



(2) Solve and combine.



Generally, the method uses the approach of

$$p_i(k) = \begin{cases} a_i, & i = 0 \\ a_i k + p_{i-1}(k), & i > 0 \end{cases} \tag{11}$$

with the sub-polynomial $p_i(x)$ is defined as

$$p_i(x) = \sum_{j=0}^{i} a_j x^j \tag{12}$$

and the value of $p(k)$ is $p_n(k)$.

From equation (11), we can write the time complexity of this method as

$$T(n) = \begin{cases} 0, & n = 0 \\ 1 + T(n-1), & n > 0 \end{cases} \tag{13}$$

We can calculate $T(n)$ in a non-recursive model as follows.

$$\begin{aligned} T(n) &= 1 + T(n-1) \\ &= 1 + \big(1 + T(n-2)\big) \\ &= 2 + T(n-2) \\ &= k + T(n-k) \\ &= n \end{aligned}$$

Hence, the asymptotic time complexity of the method is

$$T(n) = O(n) \tag{14}$$

We can infer that the synthetic division method with its divide-and-conquer approach outperforms the two former methods.

IV. Implementation of Synthethic Division Method for Evaluating Polynomial

Implementation of the method uses the property that divide-and-conquer approach is recursive. Let a polynomial $p(x)$ having degree of n be stored in an array `arr` of size $n + 1$ and coefficient of i-powered indeterminate is stored in the i-th element of the array. The implementation follows the piecewise equation (11).

```
function subPolyValue (i: integer, arr:
array of integer, k: integer) → integer
  if i = 0 then
    → arr[i]
  else
    → arr[i] * k + subPolyValue(i-1, arr, k)


function evalPoly(arr: array of integer, k:
integer) → integer
  n ← len(arr) - 1
  → subPolyValue(n, arr, k)
```

Alternative to the recursive approach is the iterative one. The idea is to iterate the array from the last index to first index while calculating the value of the sub-polynomial $p_i(k)$. The implementation for the iterative approach is as follows.

```
function evalPoly (arr: array of integer, k: integer) → integer
  n ← len(arr) - 1
  value ← 0
  i traversal [n..0]
    value ← value * k + arr[i]
  → value
```

V. Examples of Synthetic Division Method for Evaluating Polynomial

A. Manual Evaluation

As synthetic division is originally used in manual fashion, we try to evaluate two polynomials, one with low degree and another with higher degree.

1) Polynomial with degree of 3

Let the polynomial be $p(x) = 2x^3 - 7x^2 + 6x + 2$ and we want to evaluate the value of $p(3)$. The synthetic division method is as follows.

3	2	-7	6	2
		6	-3	9
	2	-1	3	11

From the calculation above, we evaluated the value of $p(3)$ to be 11. The synthetic division method only needs 3 multiplications and 3 additions/subtractions, while brute-force method would need 6 multiplications and 3 additions/subtractions.

2) Polynomial with degree of 7

Let the polynomial be $p(x) = -x^7 + 12x^5 - 4x^4 + 27x^2 + 12x$ and we want to evaluate the value of $p(4)$. The synthetic division method is as follows.

4	-1	0	12	-4	0	27	12	0
		-4	-16	-16	-80	-320	-1172	-1160
	-1	-4	-4	-20	-80	-293	-1160	-4640

From the calculation above, we evaluated the value of $p(4)$ to be -4640. The synthetic division method only needs 7 multiplications and 7 additions/subtractions, while brute-force method would need 18 multiplications and 4 additions/subtractions.

B. Computational Evaluation

To test the claim that synthetic division method is more efficient than both brute-force methods computationally, we implemented three of the methods in Python 3.10 and then tested the performance of each method on six cases with random coefficients and random k value in calculating $p(k)$.

The original brute-force method is implemented as follows.

```
def evalPoly1(arr, k):
    val = 0
    n = len(arr)-1
    for i in range(len(arr)):
        temp = arr[i]
        for j in range(n-i):
            temp = temp * k
        val += temp
    return val
```

The enhanced brute-force method is implemented as follows.

```
def power(a, n):
    if n==0:
        return 1
    else:
        if n%2==0:
            x = power(a, n//2)
            return x*x
        else:
            x = power(a, n//2)
            return x*x*a


def evalPoly2(arr, k):
    val = 0
    n = len(arr)-1
    for i in range(len(arr)):
        val += arr[i] * power(k, n-i)
    return val
```

The synthetic division method is implemented using the iterative approach as follows.

```
def evalPoly3(arr, k):
    val = 0
    n = len(arr)-1
    for i in range(len(arr)):
```

```
        val = val * k + arr[i]
    return val
```

1) Polynomial with degree of 0

First, we tested the performance of the methods given that the polynomial has a degree of 0, in other word, a constant.

```
Number of iterations: 100000
Polynomial with degree of 0
1. Brute-force method:          44.02 ms
2. Enhanced brute-force method: 44.02 ms
3. Synthetic division method:   33.02 ms
```

The result shows that synthetic division method is slightly faster than the other methods by approximately only 25%.

2) Polynomial with degree of 5

Next, we tested the performance of the methods by using a polynomial with degree of 5, which is a standard polynomial used in everyday life.

```
Number of iterations: 100000
Polynomial with degree of 5
1. Brute-force method:          186.05 ms
2. Enhanced brute-force method: 287.07 ms
3. Synthetic division method:   64.01 ms
```

The result shows that the synthetic division method is about 200% faster than the original brute-force-method, and an anomaly of enhanced brute-force method being the slowest happens.

3) Polynomial with degree of 20

We also tested the performance of the methods by using a polynomial with a degree of 20 which apparently yielded similar result to the test with degree of 5: synthetic division method is six times as fast as the original brute-force method, and enhanced brute-force method being the slowest.

```
Number of iterations: 5000
Polynomial with degree of 20
1. Brute-force method:          59.01 ms
2. Enhanced brute-force method: 79.02 ms
3. Synthetic division method:   10.0 ms
```

4) Polynomial with degree of 100

Next, we tested the performances on a polynomial with degree of 100.

```
Number of iterations: 1000
Polynomial with degree of 100
1. Brute-force method:          272.07 ms
2. Enhanced brute-force method: 116.99 ms
3. Synthetic division method:   11.01 ms
```

The difference between both brute-force methods and synthetic division method is becoming apparent. Synthetic division method is about 25 times as fast as original brute-force method and 10 times as fast as enhanced one.

5) Polynomial with degree of 1000

We jumped to test evaluating a polynomial with a degree of 1000.

```
Number of iterations: 100
Polynomial with degree of 1000
1. Brute-force method:          5984.02 ms
2. Enhanced brute-force method: 401.51 ms
3. Synthetic division method:   28.99 ms
```

The result shows that the original brute-force method is significantly slower, taking up to six seconds while enhanced brute-force method took 1/15 of it and synthetic division method only took about 1/200 of it.

6) Polynomial with degree of 5000

Lastly, we tested the performances by using a polynomial with degree of 5000.

```
Number of iterations: 20
Polynomial with degree of 5000
1. Brute-force method:          105118.09 ms
2. Enhanced brute-force method: 4337.78 ms
3. Synthetic division method:   103.04 ms
```

The result shows the synthetic method is indeed the fastest among others, being approximately 43 times as fast as enhanced brute-force method and 1000 times as fast as original brute-force method.

VI. Conclusion

From the study, it was affirmed that divide-and-conquer algorithm can be applied to evaluating polynomial in form of synthetic division method. The synthetic division method is significantly more efficient that the two other methods, and can be applied manually or computationally.

Video Link at Youtube

The explanation video of this paper can be found on the following link: https://youtu.be/7KDm5lxdINQ.

Acknowledgment

The author would like to express gratitude to friends who provided help and supports during the writing of this paper. Author would also like to thank Dr. Nur Ulfa Maulidevi, S.T., M.Sc. as class lecturer for IF2211 Algorithm Strategies course for the spirit and determination to educate students with enthusiasm.

References

[1] A. Levitin, Introduction to the Design and Analysis of Algorithms, 3rd ed. New Jersey: Addison-Wesley, 2012, ch. 5.

[2] R. Munir, School of Electrical Engineering and Informatics, Bandung Institute of Technology, lecture slide: "Algoritma divide and conquer", 2021.

[3] R. Larson, Algebra and Trigonometry, 8th ed. California: Brooks/Cole, 2011, ch. 3.

Statement

Hereby I state that this paper is my own writing and not a copy, translation, nor plagiarism of others' works.

Bandung, 20 May 2022

Wesly Giovano (13520071)