APPLICATIONS OF POLYNOMIAL INTERPOLATION

NAME – AYUSHI PANDEY

COURSE TITLE - LINEAR ALGEBRA

PROJECT GROUP – 8

INSTITUTE - INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY, HYDERABAD

DATE – 21st November 2020

ABSTRACT

In this work, one of the applications of Polynomial Interpolation - Toom Cook Multiplication is discussed. Also, the role and usage of Linear Algebra in Toom-Cook Multiplication is explored. In this paper we work on the classical Toom-k way of dealing with the multiplication of large integers.

INTRODUCTION

Polynomial Interpolation

Polynomial interpolation is a method of estimating values between known data points. When graphical data contains a gap, but data is available on either side of the gap or at a few specific points within the gap, an estimate of values within the gap can be made by interpolation. The simplest method of interpolation is to draw straight lines between the known data points and consider the function as the combination of those straight lines. This method, called linear interpolation, usually introduces considerable error. A more precise approach uses a polynomial function to connect the points. A polynomial is a mathematical expression comprising a sum of terms, each term including a variable or variables raised to a power and multiplied by a coefficient. The simplest polynomials have one variable. Polynomials can exist in factored form or written out in full.

The value of the largest exponent is called the degree of the polynomial. If a set of data contains n known points, then there exists exactly one polynomial of degree n1 or smaller that passes through all of those points. The polynomial's graph can be thought of as "filling in the curve" to account for data between the known points. This methodology, known as polynomial interpolation, often (but not always) provides more accurate results than linear interpolation.

Applications of Polynomial Interpolation

Polynomials can be used to approximate complicated curves, for example, the shapes of letters in typography (typography is the art of arranging letters and test in a way that makes the copy legible, clear, and visually appealing to the reader), given a few points. A relevant application is the evaluation of the natural logarithm and trigonometric functions: pick a few known data points, create a lookup table, and interpolate between those data points. This results in significantly faster computations. Polynomial interpolation also forms the basis for algorithms in numerical quadrature and numerical ordinary differential equations and Secure Multi Party Computation, Secret Sharing schemes. Polynomial interpolation is also essential to perform sub-quadratic multiplication and squaring such as Karatsuba multiplication and Toom–Cook multiplication, where an interpolation through points on a polynomial which defines the product yields the product itself.

TOOM-COOK MULTIPLICATION

The Toom-Cook algorithm follows the divide and conquer method for multiplying large integers. Just like Karatsuba it splits the given integer into n limbs of some fixed size. The division is then applied recursively with Toom-3 algorithm. This goes on until we can apply another algorithm on it for the last stage of recursion, or until the desired multiplier is reached. The method works on the principles of polynomial multiplication. The input numbers are divided into limbs of a given size, and each in the form of polynomial, and the limb size is used as radix. Instead of multiplying the obtained polynomials directly, they are evaluated at a set of points, and the values multiplied together at those points. Based on the products obtained at those points the product polynomial is obtained. The result is then obtained by substituting the radix.

Given two large integers, *a* and *b*, Toom–Cook splits up *a* and *b* into *k* smaller parts each of length *l*, and performs operations on the parts. As *k* grows, one may combine many of the multiplication sub-operations, thus reducing the overall complexity of the algorithm. The multiplication sub-operations can then be computed recursively using Toom–Cook multiplication again, and so on. Although the terms "Toom-3" and "Toom–Cook" are sometimes incorrectly used interchangeably, Toom-3 is only a single instance of the Toom–Cook algorithm, where *k* = 3.

The First algorithm that was developed for the univariate polynomials is Karatsuba. It was a steppingstone for other algorithms for multiplication, including the Toom-Cook algorithm. The Toom-Cook method is actually 2 based on the Karatsuba method by splitting each number to be multiplied into multiple parts. Toom -3 way reduces number of multiplications greatly depending on the number of splits done.

The algorithm has five main steps:

1. Splitting
2. Evaluation
3. Pointwise multiplication
4. Interpolation
5. Recomposition

In a typical large integer implementation, each integer is represented as a sequence of digits in positional notation, with the base or radix set to some (typically large) value *b*; for this example we use *b* = 10000, so that each digit corresponds to a group of four decimal digits.

1. SPLITTING

The first step is to select the base $B = b^i$, such that the number of digits of both *m* and *n* in base *B* is at most *k* (e.g., 3 in Toom-3). A typical choice for *i* is given by:

$$i = max\left\{\left\lfloor\frac{\lfloor\log_b m\rfloor}{k}\right\rfloor, \left\lfloor\frac{\lfloor\log_b n\rfloor}{k}\right\rfloor\right\} + 1$$

In our example we'll be doing Toom-3, so we choose $B = b^2 = 10^8$. We then separate *m* and *n* into their base *B* digits m_i, n_i:

$$\begin{aligned} m_2 &= 123456 \\ m_1 &= 78901234 \\ m_0 &= 56789012 \\ n_2 &= 98765 \\ n_1 &= 43219876 \\ n_0 &= 54321098 \end{aligned}$$

We then use these digits as coefficients in degree-($k - 1$) polynomials *p* and *q*, with the property that $p(B) = m$ and $q(B) = n$:

$$p(x) = m_2x^2 + m_1x + m_0 = 123456x^2 + 78901234x + 56789012$$
$$q(x) = n_2x^2 + n_1x + n_0 = 98765x^2 + 43219876x + 54321098$$

The purpose of defining these polynomials is that if we can compute their product $r(x) = p(x)q(x)$, our answer will be $r(B) = m \times n$.

In the case where the numbers being multiplied are of different sizes, it's useful to use different values of k for m and n, which we'll call k_m and k_n. For example, the algorithm "Toom-2.5" refers to Toom–Cook with $k_m = 3$ and $k_n = 2$. In this case the i in $B = b^i$ is typically chosen by:

$$i = max\left\{\left\lfloor\frac{\lceil\log_b m\rceil}{k_m}\right\rfloor, \left\lfloor\frac{\lceil\log_b n\rceil}{k_n}\right\rfloor\right\}$$

2. EVALUATION

The Toom–Cook approach to computing the polynomial product $p(x)q(x)$ is a commonly used one. Note that a polynomial of degree d is uniquely determined by $d + 1$ points (for example, a line - polynomial of degree one is specified by two points). The idea is to evaluate $p(\cdot)$ and $q(\cdot)$ at various points. Then multiply their values at these points to get points on the product polynomial. Finally interpolate to find its coefficients.

Since $\deg(pq) = \deg(p) + \deg(q)$, we will need $\deg(p) + \deg(q) + 1 = k_m + k_n - 1$ points to determine the final result. Call this d. In the case of Toom-3, $d = 5$. The algorithm will work no matter what points are chosen (with a few small exceptions, see matrix invertibility requirement in Interpolation), but in the interest of simplifying the algorithm it's better to choose small integer values like 0, 1, −1, and −2.

One unusual point value that is frequently used is infinity, written ∞ or 1/0. To "evaluate" a polynomial p at infinity actually means to take the limit of $p(x)/x^{\deg p}$ as x goes to infinity. Consequently, $p(\infty)$ is always the value of its highest-degree coefficient (in the example above coefficient m_2).

In our Toom-3 example, we will use the points 0, 1, −1, −2, and ∞. These choices simplify evaluation, producing the formulas:

$$\begin{array}{lllll} p(0) & = & m_0 + m_1(0) + m_2(0)^2 & = & m_0 \\ p(1) & = & m_0 + m_1(1) + m_2(1)^2 & = & m_0 + m_1 + m_2 \\ p(-1) & = & m_0 + m_1(-1) + m_2(-1)^2 & = & m_0 - m_1 + m_2 \\ p(-2) & = & m_0 + m_1(-2) + m_2(-2)^2 & = & m_0 - 2m_1 + 4m_2 \\ p(\infty) & = & m_2 & & \end{array}$$

and analogously for q. In our example, the values we get are:

$$
\begin{aligned}
p(0) &= m_0 &&= 56789012 &&= 56789012\\
p(1) &= m_0 + m_1 + m_2 &&= 56789012 + 78901234 + 123456 &&= 135813702\\
p(-1) &= m_0 - m_1 + m_2 &&= 56789012 - 78901234 + 123456 &&= -21988766\\
p(-2) &= m_0 - 2m_1 + 4m_2 &&= 56789012 - 2 \times 78901234 + 4 \times 123456 &&= -100519632\\
p(\infty) &= m_2 &&= 123456 &&= 123456\\
q(0) &= n_0 &&= 54321098 &&= 54321098\\
q(1) &= n_0 + n_1 + n_2 &&= 54321098 + 43219876 + 98765 &&= 97639739\\
q(-1) &= n_0 - n_1 + n_2 &&= 54321098 - 43219876 + 98765 &&= 11199987\\
q(-2) &= n_0 - 2n_1 + 4n_2 &&= 54321098 - 2 \times 43219876 + 4 \times 98765 &&= -31723594\\
q(\infty) &= n_2 &&= 98765 &&= 98765.
\end{aligned}
$$

As shown, these values may be negative.

For the purpose of later explanation, it will be useful to view this evaluation process as a matrix-vector multiplication, where each row of the matrix contains powers of one of the evaluation points, and the vector contains the coefficients of the polynomial:

$$
\begin{pmatrix} p(0) \\ p(1) \\ p(-1) \\ p(-2) \\ p(\infty) \end{pmatrix} = \begin{pmatrix} 0^0 & 0^1 & 0^2 \\ 1^0 & 1^1 & 1^2 \\ (-1)^0 & (-1)^1 & (-1)^2 \\ (-2)^0 & (-2)^1 & (-2)^2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ m_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & -2 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ m_2 \end{pmatrix}
$$

3. POINTWISE MULTIPLICATION

Unlike multiplying the polynomials $p(\cdot)$ and $q(\cdot)$, multiplying the evaluated values $p(a)$ and $q(a)$ just involves multiplying integers — a smaller instance of the original problem. We recursively invoke our multiplication procedure to multiply each pair of evaluated points. In practical implementations, as the operands become smaller, the algorithm will switch to schoolbook long multiplication. Letting r be the product polynomial, in our example we have:

$$
\begin{aligned}
r(0) &= p(0)q(0) &&= 56789012 \times 54321098 &&= 3084841486175176\\
r(1) &= p(1)q(1) &&= 135813702 \times 97639739 &&= 13260814415903778\\
r(-1) &= p(-1)q(-1) &&= -21988766 \times 11199987 &&= -246273893346042\\
r(-2) &= p(-2)q(-2) &&= -100519632 \times -31723594 &&= 3188843994597408\\
r(\infty) &= p(\infty)q(\infty) &&= 123456 \times 98765 &&= 12193131840.
\end{aligned}
$$

As shown, these can also be negative. For large enough numbers, this is the most expensive step, the only step that is not linear in the sizes of m and n.

4. INTERPOLATION

This is the most complex step, the reverse of the evaluation step: given our *d* points on the product polynomial *r*(·), we need to determine its coefficients. In other words, we want to solve this matrix equation for the vector on the right-hand side:

$$\begin{pmatrix} r(0) \\ r(1) \\ r(-1) \\ r(-2) \\ r(\infty) \end{pmatrix} = \begin{pmatrix} 0^0 & 0^1 & 0^2 & 0^3 & 0^4 \\ 1^0 & 1^1 & 1^2 & 1^3 & 1^4 \\ (-1)^0 & (-1)^1 & (-1)^2 & (-1)^3 & (-1)^4 \\ (-2)^0 & (-2)^1 & (-2)^2 & (-2)^3 & (-2)^4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -2 & 4 & -8 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix}$$

This matrix is constructed the same way as the one in the evaluation step, except that it's *d* × *d*. We could solve this equation with a technique like Gaussian elimination, but this is too expensive. Instead, we use the fact that, provided the evaluation points were chosen suitably, this matrix is invertible (see also Vandermonde matrix), and so:

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -2 & 4 & -8 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} r(0) \\ r(1) \\ r(-1) \\ r(-2) \\ r(\infty) \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{3} & -1 & \frac{1}{6} & -2 \\ -1 & \frac{1}{2} & \frac{1}{2} & 0 & -1 \\ -\frac{1}{2} & \frac{1}{6} & \frac{1}{2} & -\frac{1}{6} & 2 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r(0) \\ r(1) \\ r(-1) \\ r(-2) \\ r(\infty) \end{pmatrix}$$

All that remains is to compute this matrix-vector product. Although the matrix contains fractions, the resulting coefficients will be integers — so this can all be done with integer arithmetic, just additions, subtractions, and multiplication/division by small constants. A difficult design challenge in Toom–Cook is to find an efficient sequence of operations to compute this product; one sequence given by Bodrato[6] for Toom-3 is the following, executed here over the running example:

$$\begin{aligned}
r_0 &\leftarrow r(0) && = 3084841486175176\\
r_4 &\leftarrow r(\infty) && = 12193131840\\
r_3 &\leftarrow (r(-2) - r(1))/3 && = (3188843994597408 - 13260814415903778)/3\\
& && = -3357323473768790\\
r_1 &\leftarrow (r(1) - r(-1))/2 && = (13260814415903778 - (-246273893346042))/2\\
& && = 6753544154624910\\
r_2 &\leftarrow r(-1) - r(0) && = -246273893346042 - 3084841486175176\\
& && = -3331115379521218\\
r_3 &\leftarrow (r_2 - r_3)/2 + 2r(\infty) && = (-3331115379521218 - (-3357323473768790))/2 + 2 \times 12193131840\\
& && = 13128433387466\\
r_2 &\leftarrow r_2 + r_1 - r_4 && = -3331115379521218 + 6753544154624910 - 12193131840\\
& && = 3422416581971852\\
r_1 &\leftarrow r_1 - r_3 && = 6753544154624910 - 13128433387466\\
& && = 6740415721237444.
\end{aligned}$$

We now know our product polynomial *r*:

$$\begin{array}{ll}
r(x) = & 3084841486175176\\
+ & 6740415721237444x\\
+ & 3422416581971852x^2\\
+ & 13128433387466x^3\\
+ & 12193131840x^4
\end{array}$$

If we were using different k_m, k_n, or evaluation points, the matrix and so our interpolation strategy would change; but it does not depend on the inputs and so can be hard-coded for any given set of parameters.

5. RECOMPOSITION

Finally, we evaluate r(B) to obtain our final answer. This is straightforward since B is a power of *b* and so the multiplications by powers of B are all shifts by a whole number of digits in base *b*. In the running example b = 10^4 and B = b^2 = 10^8.

```
                                    3084 8414 8617 5176
                          6740 4157 2123 7444
                3422 4165 8197 1852
        13 1284 3338 7466
+ 121 9313 1840
-------------------------------------------------------
  121 9326 3124 6761 1632 4937 6009 5208 5858 8617 5176
```

And this is in fact the product of 1234567890123456789012 and 987654321987654321098.

Interpolation matrices for various k –

Here we give common interpolation matrices for a few different common small values of k_m and k_n.

TOOM-1 :

Toom-1 ($k_m = k_n = 1$) requires 1 evaluation point, here chosen to be 0. It degenerates to long multiplication, with an interpolation matrix of the identity matrix:

$$(1)^{-1} = (1)$$

TOOM-1.5 :

Toom-1.5 ($k_m = 2$, $k_n = 1$) requires 2 evaluation points, here chosen to be 0 and ∞. Its interpolation matrix is then the identity matrix:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

This also degenerates to long multiplication: both coefficients of one factor are multiplied by the sole coefficient of the other factor.

TOOM-2 :

Toom-2 ($k_m = 2$, $k_n = 2$) requires 3 evaluation points, here chosen to be 0, 1, and ∞. It is the same as Karatsuba multiplication, with an interpolation matrix of:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

TOOM-2.5 :

Toom-2.5 ($k_m = 3$, $k_n = 2$) requires 4 evaluation points, here chosen to be 0, 1, −1, and ∞. It then has an interpolation matrix of:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} & -1 \\ -1 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

REFERENCES

1. Bodrato, M., Zanoni, A.: Integer and Polynomial Multiplication: Towards Optimal Toom-Cook Matrices.

2. D. Knuth. The Art of Computer Programming, Volume 2. Third Edition, Addison-Wesley, 1997. Section 4.3.3.A: Digital methods

3. Cook, Stephen A., On the Minimum Computation Time of Functions, 1966 PhD thesis, Harvard University Department of Mathematics.

4. https://en.wikipedia.org/wiki/Toom%E2%80%93Cook_multiplication