CS487 - Symbolic Computation

University of Waterloo
Nicholas Pun
Winter 2020

Contents

Lecture 1: Introduction

1.1 Course Preview

Example 1.1 (Simplyfying Rational Expressions)**.** Suppose we have the two following expressions:

$$f := \frac{x+1}{x-1} - \frac{x^3-2x+x^2+2}{x^3+2x-x^2-2} + \frac{x^2+3}{x-1} \tag{1.1}$$

$$g := \frac{(x-1)^2 - x^2 - x + 2x}{(x+y+2)^{100}} \tag{1.2}$$

Question: How do we simplify these expressions to a single $\frac{poly}{poly}$ or return that it is 0?

One idea: Define a "normal" function:

1. If expression is 0, the normal function will be 0
2. If not, the normal function will be the simplest form

(More) Questions: What else do we need to consider?

- How do we represent polynomials (i.e. What data structure do we use?)
- How do we perform polynomial operations computationally?
- Do we need to consider the size of the integers in our computations?

Example 1.2 (Solving Recurrences)**.** Suppose we have the recurrence:

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + \frac{n}{2} & n > 1 \\ 1 & n = 1 \end{cases} \tag{1.3}$$

We can solve this by hand (using Master theorem or other techniques) to obtain the answer:

$$T(n) = n(1 + \log_2(n)) \tag{1.4}$$

Question: How do we do this computationally?

Example 1.3. Consider the following identities:

$$\sum_{k=0}^{n} k = \frac{n(n-1)}{2} \tag{1.5}$$

$$\sum_{k=0}^{n} k^4 = \frac{n(n-1)(2n-1)(3n^3-3n-1)}{30} \tag{1.6}$$

Question: Can we return a closed form (without involving the index k) for any general expression or report that one doesn't exist?

1.2 Representation of Integers

Current computers are based on architecture with 64 bits (We will call this number of bits the word size)

Example 1.4. The unsigned long in C represents integers in exactly the range $[0, 2^{64} - 1]$

Question: How do we represent larger numbers?

Idea: Use an array of word size numbers.

Any integer a can be expressed as the following summation:

$$a = (-1)^s \sum_{i=0}^{n} a_i 2^{64i} \tag{1.7}$$

where $s \in \{0, 1\}$ represents the sign of a and $0 \leq a_i \leq 2^{64} - 1$ are the individual elements in the array.

If we assume $0 \leq n + 1 \leq 2^{63}$, then we can encode a as an array:

$$[s \cdot 2^{63} + n + 1, a_0, a_1, \ldots, a_n] \tag{1.8}$$

This is sufficient for all practical purposes.

Note. The length of a is given by: $\lfloor \log_{2^{64}} |a| \rfloor + 1 \in \mathcal{O}(\log |a|)$ words

1.3 Addition of Integers

Suppose our input is $a : a_0 + a_1\beta + a_2\beta^2 + \ldots a_n\beta^n$ and $b : b_0 + b_1\beta + b_2\beta^2 + \ldots b_m\beta^m$ (where $m \leq n$). Let $c = a + b = c_0 + c_1\beta + c_2\beta^2 + \ldots c_n\beta^n$, each $c_i = a_i + b_i$ if $i \leq m$ and $c_i = a_i$ otherwise.

$a_i + b_i$ may be greater than β. In this case, the addition creates a *carry* to the $(i + 1)$-th term.

Question: How large can c get?

In particular, will our array drastically change in size?

We can begin with the case of $\beta = 2$. This gives us binary strings, a case we may be familiar with. We can simply every bit equal to 1 to obtain:

$$1 + 1 \cdot 2 + 1 \cdot 2^2 + \ldots + 1 \cdot 2^m = 2^{m+1} - 1$$

For general β this suggests the following:

$$\sum_{i=0}^{m} = (\beta - 1)\beta^i = \beta^{m+1} - 1 \tag{1.9}$$

So, given two equal length (array-wise) integers a, b:

$$\begin{aligned}(a_0 + a_1\beta + \ldots + a_m\beta^m) + (b_0 + b_1\beta + \ldots + b_m\beta^m) &\leq 2(\beta^{m+1} - 1) \\ &= (\beta^{m+1} - 2) + \beta^{m+1}\end{aligned} \tag{1.10}$$

This implies that the largest the carry bit can be is 1.

Lecture 2: Complexity of Arithmetic Operations

We want to talk about basic operations (i.e. $\{+, -, \times, \div\}$) over a ring. (Note: Division may not always be possible)

Example 2.1. The following rings will come up:

1. Integers ($\mathbb{Z}$)
2. Rationals ($\mathbb{Q}$)
3. Fields (E.g. $\mathbb{Z}_7$)
4. Polynomial Rings ($R[x]$), where R is any commutative ring. E.g. $\mathbb{Z}[x], \mathbb{Q}[x], \mathbb{Z}_p[x]$
5. Field of rational functions ($R(x)$). E.g. $\mathbb{Q}(x)$

2.1 Naive upper bounds on costs

For polynomials, we are interested in $a, b \in R[x]$. We will let $n = deg(a),\ m = deg(b)$ and we will count ring operations from R.

For integers, we will count bit operations.

We'll also define the following operation: for $a \in \mathbb{Z}$, $\lg a = \begin{cases} 1 & \text{if } a = 0 \\ 1 + \lfloor \log_2 |a| \rfloor & \text{if } a \neq 0 \end{cases}$

The following table summarizes the upper bounds:

Operation	Polynomials	Integers
$a + b$	$n + m + 1$	$\lg a + \lg b$
$a - b$	$n + m + 1$	$\lg a + \lg b$
$a \times b$	$(n+1)(m+1)$	$(\lg a)(\lg b)$
$a = qb + r$	$(n-m+1)(m+1)$	$(\lg \frac{a}{b})(\lg b)$

2.1.1 Addition

$$\begin{array}{ll} a_0 + a_1 x + \ldots + a_m x^m + & a_{m+1}x^{m+1} + \ldots + a_n x^n \\ b_0 + b_1 x + \ldots + b_m x^m & \\ \hline c_0 + c_1 x + \ldots + c_m x^m + & c_{m+1}x^{m+1} + \ldots + c_n x^n \end{array}$$

While we really only add the first $m + 1$ terms, the add operation returns a new polynomial c. As such, we really perform $\max\{m, n\} + 1 \in \Theta(n + m) + 1$ operations.

The same analysis can be used for the add operation on integers.

2.1.2 Multiplication

Consider $a = \sum^n a_i x^i, b = \sum^m b_i x^i$, and $c = a \times b = \sum^{n+m} c_k x^k$, where $c_k = \sum a_i b_j$.

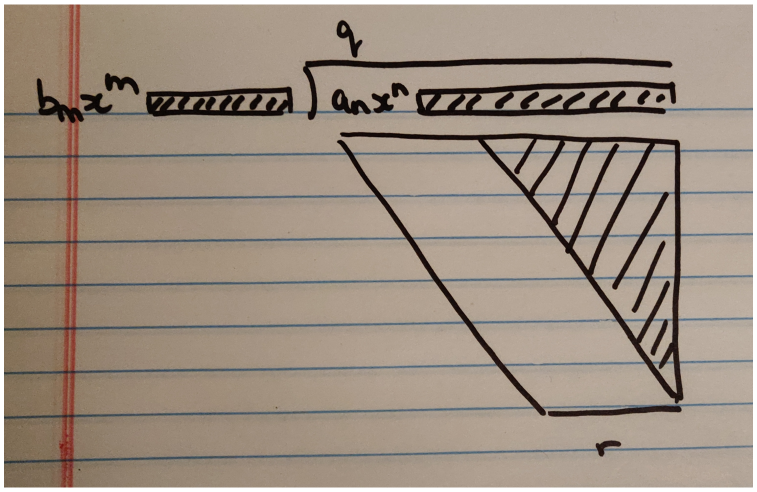
Classical "school" method: Cost is $(n+1)(m+1)$ multiplications and nm additions (exactly).

2.1.3 Division with Remainder

Given $a, b \in \mathbb{Z}$ (or $R[x]$), we want to find $q, r \in \mathbb{Z}$ with $size(r) < size(b)$ so that $a = bq + r$. Note that: $size(\cdot)$ for integers is just the magnitude, and for polynomials, $size(r) = \deg(r)$

We will require that for polynomials a, b, in $a \div b$, the constant term of b is a unit (and so an inverse exists)

Doing long division results in something that will look like the drawing below:



Within the shaded region of the trapezoid, no changes are made to the polynomial. In each step of the long division, we only perform changes to m terms within the unshaded band in the trapezoid. There are a total of $n - m$ steps (This is the resulting degree of q).

So, in total, long division of polynomials can be done in $\mathcal{O}((m+1)(n-m+1))$ operations.

Note. Why do we not just reuse our subtraction operation? We want this division operation to be primitive. The operation only performs ring operations as needed.

The same analysis can be performed on long division of integers.

2.2 Multimodular Reduction

Suppose $a \in \mathbb{Z}$, $p_1, \ldots, p_k \in \mathbb{Z}_{>1}$, with $a < p := p_1 \ldots p_k$. What is cost of computing $a \mod p_1$, $a \mod p_2$, ..., $a \mod p_k$? (i.e. Obtaining the remainders)

Rough Bound: We can use the division with remainder operation. Both a and the p_i's are bounded by p. Since there are k p_i's, we will perform the operation at most k times. This gives the bound $\mathcal{O}(k(\lg p)^2)$

But, of course we can be more accurate with this bound. In total, the k division with remainders require $\sum_{i=1}^{k} C\left(\lg \frac{a}{p_i}\right)(\lg p)$ operations (The C comes from the big-$\mathcal{O}$ of the division with remainder operation). We get:

$$\begin{aligned}
&\sum_{i=1}^{k} C\left(\lg \frac{a}{p_i}\right)(\lg p) \\
&= C\sum_{i=1}^{k}\left(\lg \frac{a}{p_i}\right)(\lg p) \\
&\leq C(\lg p)\sum_{i=1}^{k}(\lg p) && (\lg \frac{a}{p_i} \leq \lg a \leq \lg p) \\
&\leq C(1+\log p)\sum_{i=1}^{k}(1+\log p) && (\text{Get rid of the } \lg) \\
&\leq C(2\log p)\sum_{i=1}^{k}(2\log p) && (\text{If } x > 1, 1+\log x \leq 2\log x) \\
&= 4C(\log p)^2
\end{aligned} \tag{2.1}$$