Title

Your Name

February 28, 2018

Contents

Todo list

1 Definitions

In this section we define Kurepa's conjecture and Landau Notation.

1.1 Kurepa's Conjecture

Đuro Kurepa defined the left factorial of n as $\sum_{k=0}^{n-1} k!$ for all positive integers n and $!0 := 1$ in [Kur71]. For example $!7 = 6! + 5! + 4! + 3! + 2! + 1! + 0! = 874$.

We denote the greatest common divisor of $!n$ and $n!$ as M_n; that is,

$$M_n = (!n, n!).$$

We define r_n as the smallest non-negative integer such that $r_n \equiv !n \bmod n$. Thus if we write $M_n = nq + r$ using the division algorithm, then $r = r_n$. Furthermore, Kurepa hypothesized that $M_n = 2$ for all $n > 1$ and showed that this was in fact equivalent to the statement that $r_n \not\equiv 0 \bmod n$ for all $n > 2$; specifically $r_p \not\equiv 0 \bmod p$ for all odd primes p.

For example let us consider the case where $n = 7$. Then $M_7 = (!7, 7!) = (874, 5040) = 2$ which satifies the hypothesis. Also $M_7 = 874 = 7 \times 124 + 6$ so $r_7 = 6$. Thus $r_7 \not\equiv 0 \bmod 7$.

1.2 Landau Notation

In this subsection we remind the reader of Landau

landau-bachmann?

notation.

Let f be a function that maps some subset of $\mathbb{R}$ to $\mathbb{R}$. We say that $f(x) = (g(x))$ if there exists a positive real number C and a real number x_0 such that $|f(x)| \leq C|g(x)|$ for all $x \geq x_0$. Intuitively, big notation provides a way of describing an upper bound to the asymptotic behaviour of a given function.

For example, define $f(x) := x^3 + x^2 \sin(x) + x \log(x)$. Then

$$\begin{aligned}
|f(x)| &= |x^3 + x^2 \sin(x) + x \log(x)| \\
&\leq |x^3| + |x^2||\sin(x)| + |x||\log(x)| \text{ (By the Triangle Inequality)} \\
&\leq |x^3| + |x^3| + |x||\log(x)| \text{ ($\sin(x) \leq x$ whenever $x > 0$)} \\
&\leq 2|x^3| + |x^2| \text{ ($\log(x) \leq x$ whenever $x \geq 1$)} \\
&\leq 3|x^3| \text{ ($x^2 \leq x^3$ whenever $x \geq 1$)}
\end{aligned}$$

so by defining C as 3 and x_0 as 1, it follows that $f(x) = (x^3)$.

Do I need to discuss = in big O isnt a symmetric operator and should be thought of as $\in$

introduce little o or epsilon notation

Now we introduce little o notation. We say that $f(x) = o(g(x))$ if

$$\lim_{x \to \infty} f(x)/g(x) = 0.$$

An alternative characterisation of little o notation is that for all $C > 0$ there exists a real number x_0 such that $|f(x)| < C|g(x)|$ for all $x \geq x_0$. This is similar to the definition of big notation except that it is for all positive constants C not a particular value of C. We denote $f(x)$ as (x^ϵ) if there exists a non negative function $g(x)$ such that $g(x) = o(1)$ and $|f(x)| \leq C|x^{g(x)}|$ for all $x \geq x_0$ and a positive constant C.

1.3 Time and Space Complexity of arithmetic operations over the integers

find harveys paper where he has results and use them here

Throughout this section, assume that m and n are both B-bit integers. Then the space complexity of adding m and n is (B)

what is this?

and time complexity of adding m and n is (B).

is subtraction the same? should be

Furthermore, the space complexity of multiplying m and n is (B) and the time complexity is $(B \log(B) \log\log(B))$. Similarly, the space complexity of diving with remainder m and n is (B) and the time complexity is $(B \log(B) \log\log(B)) = (B \log^{1+\epsilon}(B)$. We note that although multiplication and division with remainder have the same asymptotic growth, the constant in the division algorithm is larger.

needs a reference

For further details, we refer the reader to [vZGG99, Ch.8,9].

1.4

2 previous results and methods

Give previous methods and complexities and results. 2-3 pages

3 Background of accumulating remainder tree and other applications

give history (who created it and when), give applications focusing on wilson primes one, the hyperelliptic curves. give examples of one. 6 pages

3.1 Solving a certain class of recurrences

First we recall for a matrix $A = (a_{ij})_{ij}$ with integer entries and integer modulus m, we define

$$A \bmod m := (a_{ij} \bmod m)_{ij}.$$

Given a sequence of $d \times d$ matrices $A_1, \ldots, A_n$ over the integers, a $d \times k$ matrix V over the integers and a sequence of integers $m_1, \ldots, , m_n$ our goal is to calculate

$$V \bmod m_1, A_1 V \bmod m_2, \ldots, A_{n-1} \cdots A_1 V \bmod m_n.$$

Henceforth we denote $A_{k-1} \cdots A_1 V \bmod m_k$ as C_k. We shall now give some examples of how to transform certain problems using this notation.

Our first example is to compute the set of Wilson primes up to some upper bound N. We remind the reader that Wilson's theorem states that $(p-1)!+1 \equiv 0 \bmod p$ for all prime p and a Wilson prime p is defined to be a prime which satisfies the equation $(p-1)! + 1 \equiv 0 \bmod p^2$. The only known Wilson primes are $5, 13$ and 563 [CGH14] but it is conjectured that there are infinitely many of them [CGH14]. Now $C_n = (n-1)! \bmod n^2$ so

$$A_n = n, V = 1, m_n = n^2.$$

A similar example to the search for Wilson primes is to compute the set of Wolstenholme primes up to some upper bound N. Wolstenholme's theorem states that $\binom{2p-1}{p-1} \equiv 1 \bmod p^3$ for all primes $p > 3$[Wol62]. A Wolstenholme prime satisfies the stronger condition that $\binom{2p-1}{p-1} \equiv 1 \bmod p^4$ for $p > 7$. The only known Wolstenholme primes are 16843 and 2124679

reference

but it is conjectured that there are infinitely many of them

reference

. Hence our goal is to compute $\binom{2p-1}{p-1} \bmod p^4$. Now $\binom{2n-1}{n-1} = 2(2n-1)/n\binom{2n-3}{n-2}$ so we let $A_n = 2(2n+1)/(n+1), V = 1$ and $m_n = n^4$. Then

$$A_{n-1} A_{n-2} \ldots A_1 V \bmod m_n = \binom{2n-1}{n-1} \bmod n^4$$

as required.

A less trivial example is the verification of Kurepa's conjecture up to some upper bound N. As shown earlier

reference earlier chapter

an equivalent formulation of Kurepa's conjecture is that $!p \not\equiv 0 \bmod p$ for all odd primes p. Thus our object of interest is $!n \bmod n$ but this can not be written as a multiplicative recurrence using integers and instead requires higher dimensional matrices. For this recurrence relation, $A_n = \begin{pmatrix} n & 0 \\ n & 1 \end{pmatrix}, V = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $m_n = n$. Thus

$$A_{n-1}A_{n-2}\cdots A_1 V \equiv \begin{pmatrix} !n \\ n! \end{pmatrix} \bmod n$$

and from this, we can read off $!n \bmod n$.

Another example is the verification of the central trinomial coefficient conjecture up to some upper bound N. We define $T(n)$ as the coefficient of x^n in the expansion of $(1 + x + x^2)^n$. Zhi-Wei Sun conjectured that an integer $n > 3$ is prime if and only if $T(n) \equiv 1 \bmod n^2$ and therefore our object of interest is $T(n) \bmod n^2$.

insert reference

It is known that $T(n)$ satisfies the recurrence

$$T(n) = \frac{(2n-1)T(n-1) + 3T(n-2)}{n}$$

insert reference or prove

. This can be written using matrices as

$$\begin{pmatrix} T(n) \\ T(n-1) \end{pmatrix} = \frac{1}{n} \begin{pmatrix} 2n-1 & 3 \\ n & 0 \end{pmatrix} \begin{pmatrix} T(n-1) \\ T(n-2) \end{pmatrix}$$

so it follows that if $A_n = 1/(n+1) \begin{pmatrix} 2n+1 & 3 \\ n+1 & 0 \end{pmatrix}, V = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $m_n = n^2$, then

$$A_{n-1}\cdots A_1 V \bmod m_n = \begin{pmatrix} T(n) \bmod n^2 \\ T(n-1) \bmod n^2 \end{pmatrix}$$

so we can read off $T(n) \bmod n^2$ as required.

i dont understand maths of zeta function

3.2 Naive algorithm and its complexity

We also give basic results on the complexity of binary operations using integers. We assume m and n are both B-bit integers. Then the space complexity of adding m and n is $O(B)$ and time complexity of adding m and n is $O(B)$. Furthermore, the space complexity of multiplying m and n is $O(B)$ and the time complexity is $O(B\log(B)\log\log(B)) = O(B\log^{1+\epsilon}(B))$ where $\epsilon = o(1)$. Similarly, the space complexity of dividing with remainder m and n is $O(B)$ and the time complexity is $O(B\log^{1+\epsilon}(B))$. We note that although multiplication

and division with remainder have the same asymptotic growth, the constant in the division algorithm is larger.

needs a reference

For further details, we refer the reader to [vZGG99, Ch.8,9].

In this section, we approximate the number of bits required to store an object X as $\beta(X)$. This is a purely theoretical measure and is independent of the computer and the language. For example, $\beta(n) = \lg(n)$ for a positive integer n whereas the space required is $\lceil \lg(n) \rceil + 1 + C$ where C is the overhead. Furthermore, we assume that β has some nice properties such as $\beta(XY) \leq \beta(X) + \beta(Y)$. Inductively, this means that $\beta(kX) \leq k\beta(X)$ which provides a very useful bound which will be utilised throughout this chapter. For a $d \times d$ matrix X with positive integer coefficients, we define $\beta(X)$ as $\lg(\|X\|_1)$ where $\|\cdot\|_1$ is the operator 1 norm; this is equivalent to the maximum column sum of X. This is a generalisation of $\beta(n) = \lg(n)$ for higher dimension matrices. Then let X and Y be $d \times d$ matrices with positive integer coefficients. It follows that $\beta(XY) = \lg(\|XY\|_1) \leq \lg(\|X\|_1\|Y\|_1) = \lg(\|X\|_1) + \lg(\|Y\|_1) = \beta(X) + \beta(Y)$ so $\beta(XY) \leq \beta(X) + \beta(Y)$ as required since the $\|\cdot\|_1$ norm is submultiplicative.

Theorem 3.1 *Suppose X and Y are $d \times d$ matrices with positive integer coefficents such that $\beta(X), \beta(Y) \leq n$. Then the space complexity of computing XY is $O(n)$ and the time complexity is $O(n \log^{1+\varepsilon}(n))$.*

Proof 3.2 *Let $X = (x_{ij})_{ij}$ and $Y = (y_{ij})_{ij}$. Denote XY as $Z = (z_{ij})_{ij}$. Then it follows that $\lg(x_{ij}), \lg(y_{ij}) \leq n$ since $\beta(X), \beta(Y) \leq n$ so x_{ij}, y_{ij} are n-digit numbers when written in binary. Then $z_{ij} = \sum_{k=1}^{d} x_{ik}y_{kj}$. We note that $x_{ik}y_{kj}$ requires $O(n)$ bits and requires $O(n \log^{1+\varepsilon}(n))$ time. There are d additions so computing each z_{ij} requires $O(dn)$ memory and $O(dn \log^{1+\varepsilon}(n))$ time. As there are d^2 entries in Z so the total memory required is $O(d^3n)$ and $O(d^3n \log^{1+\varepsilon}(n))$ time. Hence, for fixed d, the space complexity of computing XY is $O(n)$ and the time complexity is $O(n \log^{1+\varepsilon}(n))$ as claimed.*

This theorem is extremely powerful in that it allows us to work with $\beta(X)$ for computing complexity bounds which is much easier to deal with.

Suppose we want to verify any of the conjectures or search for the class of primes in subsection

reference subsection above

up to some upper bound N. The naive algorithm involves calculating V mod m_1 then A_1V and then reducing modm_2 and so forth until $A_{N-1} \cdots A_1V$ mod m_N.

do i assume β is a monotonic function

First we introduce $\beta(X)$

introduce lemma about complexity of multiplying k n bit integers is $O(\log(k)M(k \cdot n))$

For the Wilson prime case, we recall that $A_n = n, V = 1$ and $m_n = n^2$ if n is prime and 1 otherwise. Now using the method found in

cite Borwein's paper

, it follows that the time complexity of calculating $A_{p-1}\cdots A_1V$ is $O(p\log^{2+\varepsilon}(p))$. The total time complexity is $\sum_{\substack{p\ prime\\ p\leq N}} O(p\log^{2+\varepsilon}(p)(p)) = O(N^2\log^{2+\varepsilon}(N))$.

use new bounds for the rest of the subsection

We remind the reader that for the Wolstenholme prime case that $A_n = 2(2n+1)/(n+1), V = 1$ and $m_n = n^4$. Then $\beta(A_n) = \lg(2(2n+1))+\lg(n+1) \leq 2\lg(2N)$ and $\beta(m_n) = \lg(n^4) \leq 4\lg(N)$. Then the time and space complexity of computing $A_{n-1}A_{n-2}\cdots A_1V$ is $O(2n\lg(2N))$ and $O(2n\lg(2N)\lg^{1+\epsilon}(2n\lg(2N))$ respectively. To reduce $A_{n-1}A_{n-2}\cdots A_1V$ modulo m_n has time complexity $O(2n\lg(2N))$ and space complexity $O(2n\lg(2N)\lg^{1+\epsilon}(2n\lg(2N))$. Thus, summing over all n, the time complexity is

$$\begin{aligned}\sum_{n=1}^{N} O(2n\lg(2N)) &= O(2\lg(2N)\sum_{n=1}^{N} n)\\ &= O(N^2\lg(2N))\end{aligned}$$

and the space complexity is

$$\begin{aligned}\sum_{n=1}^{N} O(2n\lg(N)\log^{1+\epsilon}(2n\lg(N))) &= O(\lg(2N)\sum_{n=1}^{N}\log^{1+\epsilon}(n\lg(N)))\\ &= O(\lg(N)\log^{1+\epsilon}(N\lg(N))\sum_{k=1}^{N} n)\\ &= O(N^2\lg(2N)\log^{1+\epsilon}(2N\lg(2N)))\end{aligned}$$

respectively.

For Kurepa's conjecture we recall that $A_n = \begin{pmatrix} n & 0\\ n & 1\end{pmatrix}, V = \begin{pmatrix}1\\1\end{pmatrix}$ and $m_n = n$. We note that the set of matrices of the form $\begin{pmatrix} a & 0\\ b & 1\end{pmatrix}$ with positive integers a, b is closed under multiplication. Let $M = \begin{pmatrix} a & 0\\ b & 1\end{pmatrix}$ then

$$\beta(M) = \lg(\|M\|_1)) = \lg(a+b)$$

where $\|\cdot\|_1$ is the 1-norm for matrices since $a+b \geq 1$. Then $\beta(A_n) = 2\lg(n) \leq 2\lg(N)$ and $\beta(m_n) = \lg(n) \leq \lg(n)$. Then the time and space complexity of computing $A_{n-1}A_{n-2}\cdots A_1V$ is $O(n\lg(N))$ and $O(n\lg(N)\lg^{1+\epsilon}(n\lg(N))$ respectively. Summing over all n, the total time complexity is

$$\begin{aligned}\sum_{n=1}^{N} O(n\lg(N)) &= O(\lg(N)\sum_{n=1}^{N} n)\\ &= O(N^2\lg(N))\end{aligned}$$

and total space complexity is

$$\begin{aligned}\sum_{n=1}^{N} O(n\lg(N)\log^{1+\epsilon}(n\lg(N))) &= O(\lg(N)\sum_{n=1}^{N}\log^{1+\epsilon}(n\lg(N)))\\ &= O(\lg(N)\log^{1+\epsilon}(N\lg(N))\sum_{k=1}^{N} n)\\ &= O(N^2\lg(N)\log^{1+\epsilon}(N\lg(N)))\\ &= O(N^2\log^{2+\epsilon}(N))\end{aligned}$$

respectively.

For the central trinomial conjecture, $A_n = 1/(n+1)\begin{pmatrix}2n+1 & 3\\ n+1 & 0\end{pmatrix}$ and $m_n = n^2$. Then $\beta(A_n) = \lg(n+1) + \lg(2n+1+n+1) = \lg(n+1) + \lg(3n+2) \leq N\lg(3)$ and $\beta(m_n) = 2\lg(n) \leq 2\lg(N)$. Then the time and space complexity of computing $A_{n-1}A_{n-2}\cdots A_1 V$ is $O(3n\lg(N))$ and $O(3n\lg(N)\lg^{1+\epsilon}(3n\lg(N))$ respectively. Thus, summing over all n, the time complexity is

$$\begin{aligned}\sum_{n=1}^{N} O(3n\lg(N)) &= O(3\lg(N)\sum_{n=1}^{N} n)\\ &= O(3N^2\lg(N))\end{aligned}$$

and the space complexity is

$$\begin{aligned}\sum_{n=1}^{N} O(3n\lg(N)\log^{1+\epsilon}(3n\lg(N))) &= O(3\lg(N)\sum_{n=1}^{N} n\log^{1+\epsilon}(n\lg(N)))\\ &= O(\lg(N)\log^{1+\epsilon}(N\lg(N))\sum_{k=1}^{N} n)\\ &= O(N^2\lg(N)\log^{1+\epsilon}(3N\lg(N)))\end{aligned}$$

respectively.

The time complexity of these algorithms are quadratic which means they are quasilinear on average. Therefore they are exponential in $\log(N)$ so they are too slow for large N to be useful for practical purposes. Hence we need a more efficient algorithm which will be introduced in the next section.

3.3 Definition of the accumulating remainder tree technique

Fix a positive integer d and let $\{A_1, A_2, \ldots, A_n\}$ be a sequence of $d \times d$ matrices with rational coefficents. Also let V be a vector of length d with rational coefficients. Let $\{m_1, m_2, \ldots, m_n\}$ We define $C : \mathbb{Z} \to M$ as $C_n :=$

$A_{n-1}A_{n-2}\cdots A_1 V \bmod m_n$ assuming that none of the denominators of the entries of C_n are divisible by m_n and the object of interest is C_n for all n up to some predetermined upper bound N.

In order to calculate C_n for all n we introduce the product tree and remainder tree algorithms.

The product tree has inputs of two positive integers M and N where $M \leq N$ and a sequence of objects $X_M, X_{M+1}, \ldots, X_N$ which has an associative multiplicative structure. The function recursively calculates $X_M X_{M+1} \cdots X_N$ and stores any subproducts calculated. This can be written algorithmically as:

Algorithm 1 Product tree algorithm

procedure PRODUCT(M, N)
  if N=M+1 **then**
    $X_{M,N} \leftarrow X_M$
    return $X_{M,N}$
  else
    $K = \lfloor (M+N)/2 \rfloor$
    $X_{leftprod} \leftarrow Product(M, K)$
    $X_{rightprod} \leftarrow Product(K, N)$
    $X_{M,N} \leftarrow X_{leftprod} X_{rightprod}$
    return $X_{M,N}$
  end if
end procedure

The remainder tree has inputs of two positive integers M and N where $M < N$, a sequence of objects $m_M, m_{M+1}, \ldots, m_N$ and an object V. The output is the sequence $V \bmod m_M, V \bmod m_{M+1}, \ldots, V \bmod m_{N-1}$. Then $V_{k,k+1} = V \bmod m_k$.

fix this algorithm

Algorithm 2 Remainder tree algorithm

procedure REMAINDER(M, N, V)
  $T \leftarrow$ Product Tree of $m_M, \ldots, m_N$.
  $V_{M,N} = V \bmod m_M \cdots m_{N-1}$
  if N=M+1 **then**

    return $V_{M,M+1}$
  else
    $K = \lfloor (M+N)/2 \rfloor$
    $V_{M,K} \leftarrow Remainder(M, K, V_{M,N})$
    $V_{K,N} \leftarrow Remainder(K, N, V_{M,N})$
    return
  end if
end procedure

Then $V_{k,k+1} = V \bmod m_k$ as required.

We now have the tools to introduce the accumulating remainder tree algorithm. In the remainder tree algorithm, we are changing the moduli but are keeping the object we are reducing modulo constant so we calculate $V \bmod m_M, V \bmod m_{M+1}, \ldots, V \bmod m_{N-1}$. However, for the examples in the section above,

reference

, we would like to calculate $A_{n-1}A_{n-2}\cdots A_1 V \bmod m_n$ for all n up to N so a remainder tree is not applicable.

Algorithm 3 Accumulating Remainder tree algorithm

procedure ACC REMAINDER(M, N, V)
  $T_1 \leftarrow Product(M, N)$ where $X_k = m_k$.
  $T_2 \leftarrow Product(M, N)$ where $X_k = A_k$.
  $V_{M,N} = V \bmod m_{M,N}$
  if N=M+1 **then**

    return $V_{M,M+1}$
  else
    $K = \lfloor (M+N)/2 \rfloor$
    $V_{M,K} \leftarrow V_{M,N} \bmod m_{M,K}$
    $V_{K,N} \leftarrow A_{M,K} V_{M,N} \bmod m_{K,N}$
    return
  end if
end procedure

Then $V_{n,n+1} = A_{n-1}\cdots A_1 V \bmod m_n$ as required.

3.4 Complexity Analysis of the accumulating remainder tree algorithm

We now present space and time complexity bounds for the accumulating remainder tree for general A_n under the assumption that $\beta(m_{n+k}\cdots m_n) \leq k\beta(m_n)$ $\beta(A_{n+k-1}\cdots A_n) \leq k\beta(A_n)$ for all k and n and that $\beta(A_k)$ and $\beta(m_k)$ are nondecreasing functions.

First we calculate the space and time complexity for each level of the product tree of the moduli. We shall assume that $M = 1$. Then $\beta(m_n) \leq \beta(m_N)$ for all $n \leq N$ since $\beta(m_n)$ is an increasing function. First we note that the space complexity of storing all the m_n is $\beta(m_n)$ so the complexity of storing the bottom layer is bounded by $N\beta(m_N)$. Now the space complexity of computing $m_{2n+1}m_{2n}$ is $\beta(N)$ and the time complexity is $\beta(N)\log(\beta(N))$. Then the space and time complexity of computing the second level is

$$\sum_{n=1}^{N/2} \beta(N) = N\beta(m_N)/2$$

and

$$\sum_{n=1}^{N/2} \beta(m_N)\log(\beta(N)) = N\beta(m_N)\log(\beta(m_N))/2$$

respectively. To compute the space and time complexity of the k-th level, we first need to calculate $m_{2^{k-1}n+2^{k-1}-1}m_{2^{k-1}n+2^{k-1}-2}\cdots m_{2^{k-1}n}$. We bound $\beta(m_{2^{k-1}n+2^{k-1}-1}m_{2^{k-1}n+2^{k-1}-2}\cdots m_{2^{k-1}n+2^{k-2}})$ and $\beta(m_{2^{k-1}n+2^{k-2}-1}m_{2^{k-1}n+2^{k-1}-2}\cdots m_{2^{k-1}n})$ above by $2^{k-2}\beta(m_N)$ since each product contains 2^{k-2} terms and *beta* is an increasing function. Hence the time complexity of computing the k-th level is bounded above by

$$\sum_{n=1}^{\lfloor N/2^{k-1}\rfloor} 2^{k-2}\beta(m_N) = N\beta(m_N)/2$$

and the space complexity is bounded by

$$\sum_{n=1}^{\lfloor N/2^{k-1}\rfloor} 2^{k-2}\beta(m_N)\log(2^{k-2}\beta(m_N)) = N\beta(m_N)\log(2^{k-2}\beta(m_N))/2$$

and these bounds are valid for $k \geq 1$.
Now k ranges from 0 to $\lceil \lg(N)\rceil$ so the total space complexity is

$$N\beta(m_N) + \sum_{k=1}^{\lceil \lg(N)\rceil} N\beta(m_N)/2 = N\beta(m_N)(1+\lceil \lg(N)\rceil/2)$$

and the total time complexity is

$$\begin{aligned}
\sum_{k=1}^{\lceil \lg(N)\rceil} N\beta(m_N)\log(2^{k-2}\beta(m_N))/2 &= N\beta(m_N)/2 \sum_{k=1}^{\lceil \lg(N)\rceil} \left(\log(2^{k-2}) + \log(\beta(m_N))\right) \\
&= N\beta(m_N)/2 \left(\lceil \lg(N)\rceil \log(\beta(m_N)) + \log(2) \sum_{k=1}^{\lceil \lg(N)\rceil} (k-2)\right) \\
&= N\beta(m_N)/2 \left(\lceil \lg(N)\rceil \log(\beta(m_N)) + \log(2)\lceil \lg(N)\rceil \left(\lceil \lg(N)\rceil - 1\right)\right) \\
&= N\beta(m_N)\lceil \lg(N)\rceil / \left(\log(\beta(m_N)) + \log(2)\left(\lceil \lg(N)\rceil - 1\right)\right).
\end{aligned}$$

For the product tree of the objects A_n, it can be shown similarly that the space and time complexity are

$$N\beta(A_N)(1+\lceil \lg(N)\rceil/2)$$

and

$$N\beta(m_N)\lceil \lg(N)\rceil / \left(\log(\beta(A_N)) + \log(2)\left(\lceil \lg(N)\rceil - 1\right)\right)$$

respectively.

To compute the complexity of the remainder tree, this requires knowing the relationship between $\beta(A_N)$ and $\beta(m_N)$ so one can determine when $\beta(A_{k+n}\cdots A_n) \leq \beta(m_n)$. For now, we shall consider the specific example of Kurepa's conjecture. We remind the reader that $A_n = \begin{pmatrix} n & 0 \\ n & 1 \end{pmatrix}, V = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $m_n = n$. Then $\beta(A_N) = \lg(2N)$ and $\beta(m_N) = \lg(N)$.

We also note that only two levels of the tree must be stored at any one time. We will bound the space complexity first. We note that $\beta(V) \ll \beta(m_1 \cdots m_N) \leq N\beta(m_N) = N\lg(N)$. Thus the space complexity to compute $V \bmod m_1 \cdots m_N$ is $O(N\lg(N))$. To compute the second level, we need to calculate $(V \bmod m_1 \cdots m_N) \bmod m_1 \cdots m_{\lceil N/2 \rceil}$ and $A_{\lceil N/2 \rceil} \cdots A_1 (V \bmod m_1 \cdots m_N) \bmod m_{\lceil N/2 \rceil} \cdots m_N$. The space complexity of computing $(V \bmod m_1 \cdots m_N) \bmod m_1 \cdots m_{\lceil N/2 \rceil}$ is $O(N\lg(N)/2)$. To compute $A_{\lceil N/2 \rceil} \cdots A_1 (V \bmod m_1 \cdots m_N) \bmod m_{\lceil N/2 \rceil} \cdots m_N$, we calculate $A_{\lceil N/2 \rceil} \cdots A_1 (V \bmod m_1 \cdots m_N)$ first and then reduce it modulo $m_{\lceil N/2 \rceil} \cdots m_N$. We note that

$$\beta(A_{\lceil N/2 \rceil} \cdots A_1) \approx N\beta(A_N)/2 = N/2 \lg(2N)$$

and

$$\beta\left(V \bmod m_1 \cdots m_N\right) \approx N\lg(N)$$

so

$$\beta\left(V \bmod m_1 \cdots m_N\right) \geq \beta(A_{\lceil N/2 \rceil} \cdots A_1)$$

for all $N > 2$. Thus the space complexity of computing $A_{\lceil N/2 \rceil} \cdots A_1 (V \bmod m_1 \cdots m_N)$ is $O(N\lg(N))$. To calculte the space complexity of reducing $A_{\lceil N/2 \rceil} \cdots A_1 (V \bmod m_1 \cdots m_N)$ modulo $m_{\lceil N/2 \rceil} \cdots m_N$, we note that $\beta(A_{\lceil N/2 \rceil} \cdots A_1 (V \bmod m_1 \cdots m_N)) \leq \beta(A_{\lceil N/2 \rceil} \cdots A_1) + \beta(V \bmod m_1 \cdots m_N) \leq 2\beta(V \bmod m_1 \cdots m_N) \leq 2\beta N\lg(N)$. Now $\beta(m_{\lceil N/2 \rceil} \cdots m_N) \approx N\beta(m_N)/2 = N\lg(N)/2$ and $N\lg(N)/2 \leq \beta 2N\lg(N)$ so the space complexity is $O(2N\lg(N))$. Hence the space complexity of computing the second level is $O(N\lg(N))$. Now we consider the space complexity of computing the k-th level. First we must compute the first branch; that is

$$\left(V \bmod m_1 \cdots m_{\lceil N/2^{k-1} \rceil}\right) \bmod m_1 \cdots m_{\lceil N/2^k \rceil}$$

and

$$\left(\left(A_{\lceil N/2^k \rceil - 1} \cdots A_1\right)\left(V \bmod m_1 \cdots m_{\lceil N/2^{k-1} \rceil}\right)\right) \bmod m_{\lceil N/2^k \rceil + 1} \cdots m_{\lceil N/2^{k-1} \rceil}$$

given $V \bmod m_1 \cdots m_{\lceil N/2^{k-1} \rceil}$. Now $\beta(m_1 \cdots m_{\lceil N/2^{k-1} \rceil}) \geq \beta(m_1 \cdots m_{\lceil N/2^k \rceil})$ and $\beta(m_1 \cdots m_{\lceil N/2^{k-1} \rceil}) \approx N/2^{k-1}\beta(m_N) = N/2^{k-1}\lg(N)$ so it follows that the space complexity of computing

$$\left(V \bmod m_1 \cdots m_{\lceil N/2^{k-1} \rceil}\right) \bmod m_1 \cdots m_{\lceil N/2^k \rceil}$$

is $O(N/2^{k-1}\lg(N))$. We note that $\beta(A_{\lceil N/2^k \rceil - 1} \cdots A_1) \approx N/2^k \lg(2N)$ and $\beta(m_1 \cdots m_{\lceil N/2^{k-1} \rceil}) \approx N/2^{k-1}\lg(N)$. Furthermore $N/2^k \lg(2N) \leq N/2^{k-1}\lg(N)$ if $N > 2$; therefore the space complexity is $O(N/2^{k-1}\lg(N))$. Now we need to

calculate the space complexity of reducing this modulo $\bmod m_{\lceil N/2^k \rceil + 1} \cdots m_{\lceil N/2^{k-1} \rceil}$. We see that

$$\beta(\left(A_{\lceil N/2^k \rceil - 1} \cdots A_1\right) \left(V \bmod m_1 \cdots m_{\lceil N/2^{k-1} \rceil}\right)) \leq 2\beta(m_1 \cdots m_{\lceil N/2^{k-1} \rceil}) \approx N/2^{k-2} \lg(N)$$

so the space complexity is $O(N/2^{k-2} \lg(N)$. Hence the total space complexity of computing both branches is $O(N/2^k \lg(N)$. These bounds are valid for each pair of branches and there are 2^{k-1} branches so the total space complexity of calculating the k-th level is $O(N \lg(N)$. Now only 2 levels are needed at any time so the total space complexity is $O(N \lg(N)$.

3.5 Background of accumulating remainder tree

give complexity bounds of arithmetic here instead

The product tree has been invented independently by many authors; most of the early uses of product trees only considered specific applications and not in full generality

reference

. Some of these applications include calculating the sum of two non-negative integers in 1958

reference

and single variable polynomial evaluation in 1960

reference

. Kogge and Stone recognised in 1973 that H.R. Downs, H Lomax and Trout had independently discovered the product tree algorithm in full generality

reference

.

The remainder tree was first introduced by Moenck and Borodin in 1972 with regards to polynomial division and the Chinese remainder theorem for integers. The algorithm is defined for elements of a Euclidean Domain [MB72].

add more details

.

The accumulating remainder tree algorithm was first used by Gerbicz in 2011

reference it?

and was first published in 2014 to search for Wilson primes in [CGH14]. It was subsequently used in [Har14] to compute the zeta function of a hyperelliptic curve over $\mathbb{F}_p$ for all $p < N$ and generalised to all arithmetic schemes in [Har15]. It was also applied in [HS14] to compute the Hasse-Witt matrix of an arbitrary hyperelliptic curve for all primes of good reduction up to some upper bound N. This algorithm was subsequently improved in [HS16]. Another use was in [HMS16] to calculate the local zeta functions of a given curve of genus three. These papers at some level all do the same thing; they want to calculate objects modulo p for many primes p and they use accumulating remainder trees to

calculate the product of these objects modulo the product of all the primes and then reduce this product modulo each prime individually.

fix this sentence

.

3.6 Other applications

4 Theorems on bounds of time and space usage

give bounds of time and space usage, one for interval 1 up to N, the other one from M to N. 10+ pages

4.1 bounds of time with no space usage

given an input N, calculate the average time for r_p where $p < N$ in $O(blah)$ bit operations (time??) and $O(blah)$ space. 2+ pages

4.2 bounds of time with space constraints

given an input M, N, calculate the average time for r_p where $M < p < N$ in $O(blah)$ bit operations (time???) and $O(blah)$ space. Use Stage 1 and 2 ideas. 3+ pages

4.3 why the bounds are useful

Explain how if you have limited memory, can use the stage 1 and 2 ideas to calculate it. prove the bounds for the width of each block in stage 1 and width for each tree of the forest in stage 2. 5+pages

5 Give results

time taken to get up to different values of N in a table. Compare results to previous results. 3+ pages

6 Upscaling

Discuss further research. 2+ pages.

References

[CGH14] Edgar Costa, Robert Gerbicz, and David Harvey, *A search for wilson primes*, Mathematics of Computation **83** (2014), 3071–3091.

[Har14] David Harvey, *Counting points on hyperelliptic curves in average polynomial time*, Annals of Mathematics **179** (2014), 783–803.

[Har15] ———, *Computing zeta functions of arithmetic schemes*, Proceedings of the London Mathematical Society **111** (2015), no. 6, 1379–1401.

[HMS16] David Harvey, Maike Massierer, and Andrew V. Sutherland, *Ants xii*, Computing L-series of geometrically hyperelliptic curves of genus three, 2016.

[HS14] David Harvey and Andrew V. Sutherland, *Ants xi*, Computing HasseâĂŞWitt matrices of hyperelliptic curves in average polynomial time, 2014.

[HS16] ———, *Frobenius distributions: Lang âĂŞ trotter and sato âĂŞ tate conjectures*, Computing HasseâĂŞWitt matrices of hyperelliptic curves in average polynomial time, II, 2016.

[Kur71] Đuro Kurepa, *On the left factorial !n*, Mathematica Balkanica **1** (1971), 147–153.

[MB72] R. Moenck and A. Borodin, *Fast modular transforms via division*, Proceedings of the 13th Annual Symposium on Switching and Automata Theory (Swat 1972) (Washington, DC, USA), SWAT '72, IEEE Computer Society, 1972, pp. 90–96.

[vZGG99] Joachim von Zur Gathen and Jurgen Gerhard, *Modern computer algebra*, 1 ed., Cambridge University Press, 1999.

[Wol62] Joseph Wolstenholme, *On certain properties of prime numbers*, The Quarterly journal of pure and applied mathematics (1862), 35–39.