# Calculating Frobenius Numbers with Boolean Toeplitz Matrix Multiplication

For Dr. Cull, CS 523, March 17, 2009

Christopher Bogart
bogart@eecs.oregonstate.edu

## ABSTRACT

I consider a class of algorithms that solve the Frobenius problem in terms of matrix index of primitivity. I discuss space tradeoffs in representation of the input numbers to the problem, which can be $O(a_n)$ or $O(n \log a_n)$; the $O(a_n)$ encoding is more efficient and leads to a lower complexity solution for high $n$ and low $a_n$. I argue complexity bounds for index of primitivity based on its relationship with Frobenius. I conjecture that powers of Boolean Minimal Frobenius matrices are always Toeplitz matrices, and give an $O(n^2 \log n)$ index of primitivity algorithm that depends on that assumption. I give empirical evidence for the conjecture, but no proof. Finally, I discuss another matrix representation that I considered, and rejected, for faster index of primitivity calculations.

## 1. INTRODUCTION

In his lectures, the German mathematician Ferdinand Georg Frobenius (1849-1917) used to raise the following problem, which is named after him, although he never published anything on it [1]:

Given a list of distinct positive integers, $a_1 \ldots a_n$, such that $\gcd(a_1 \ldots a_n) = 1$, what is the highest integer that *cannot* be represented as a sum of integer multiples of these numbers?

The Frobenius problem is also known as the "coin problem".[1] For example, if a monetary system only had a nickel and a "trime" (a three-cent piece), it would be impossible to make change of 1, 2, 4, or 7 cents. Above that, all combinations would be possible. So we call 7 the Frobenius number of the sequence (3,5).

The Frobenius problem is related to the Index of Primitivity of a matrix: given a square matrix $A$ with nonnegative entries, what is the lowest number $k$ such that $A^k >> 0$, i.e. where all entries in $A^k$ are positive?

Alfonsín [6] proved that the Frobenius problem was NP-complete, by reducing it to the Integer Knapsack Problem.

Given that the Frobenius problem is known to be NP-hard, and that it can be solved by way of the index of primitivity, my goal was to see what complexity bounds that implied for the index of primitivity problem.

[1]The "postage stamp problem" is like the coin problem, but adds an extra constraint of a maximum number of stamps that will fit on an envelope.

## 2. PREVIOUS WORK

Heap and Lynn [4] described an algorithm for the index of primitivity, shown in Figure 1.

```
INDEX-OF-PRIMITIVITY(m: matrix of size n x n)
  Create an array A of matrices
  k = 1
  A(1) = m
  for j = 2 to (n-1)^2+1
     A(j) = A(j-1)*m
     if (A(j) >> 0) then exit loop

  k = j-1
  B = A(k)
  for j = j-1 downto 1
     if not(answer*A(j) >> 0)
         answer = answer * A(j)
         k = k + j

  return k
```

**Figure 1: Index of Primitivity Algorithm takes a matrix m of size as an argument.**

Alfonsín [1] is a good starting point for anything having to do with the Frobenius problem. He explains Heap and Lynn's proof [5] of the relationship between the Frobenius number and the index of primitivity:

$$g(a_1, a_2, \ldots, a_n) = \gamma(\overline{B}) - a_n$$

where $a_1$ through $a_n$ are the coin sizes in the Frobenius problem, and $\bar{\text{B}}$ is graph specially constructed from them. $\gamma(\overline{B})$ is the index of primitivity, and $g(a_1 \ldots a_n)$ is the Frobenius number. The graph $\bar{\text{B}}$ they call the Minimal Frobenius graph, and it is formally defined in the Definitions section below.

Figure 2 shows Heap and Lynn's algorithm for calculating the Frobenius number, which comes immediately from that equation.

```
CALC-FROBENIUS(A1,A2,...AN):
   CONSTRUCT A MINIMAL FROBENIUS MATRIX B from A1...AN
   GAMMA = INDEX-OF-PRIMITIVITY(B)
   RETURN GAMMA-AN
```

**Figure 2: Heap and Lynn's Frobenius number algorithm**

## 3. DEFINITIONS

I am using Alfonsín's notation [1] of $g(a_1, a_2 \ldots a_n)$ as the Frobenius number of a list of integers, and $\gamma(A)$ as the index of primitivity of a matrix $A$.

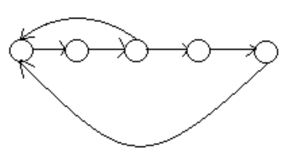
Given a list of integers $a_1$ through $a_n$, I will use the term a *Coin Matrix* to be one whose entries are defined as:

$$c_{i,j} = \begin{cases} 1 & \text{if } j - i = 1 \\ 1 & \text{if } j = 0 \text{ and } i = a_k \text{ for any } k, \\ 0 & \text{otherwise} \end{cases}$$

For example, for the nickel-trime system ($n = 2$, $a_1 = 3$, $a_2 = 5$), the coin matrix is:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Notice that this is a special case of a transposed Leslie matrix [2]. The corresponding *coin graph* is:



I will use the term *minimal Frobenius matrix* to be one as described in [5] where:

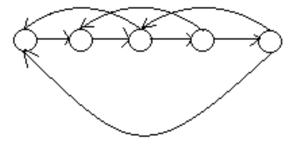
$$c_{i,j} = \begin{cases} 1 & \text{if } j - i = 1 \\ 1 & i - j = a_k - 1 \text{ for all } a_k \\ 0 & \text{otherwise} \end{cases}$$

The minimal Frobenius matrix for (3,5) is:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Notice that the matrix is a Boolean Toeplitz matrix, and it has all zeroes above the superdiagonal.

and the *minimal Frobenius graph* is:



## 4. ENCODING THE PROBLEM

Consider the most space-efficient data structure for representing a list of distinct positive integers. Assume the numbers $a_1$ through $a_n$ are sorted, so that $a_n$ is the largest. We can represent this list in exactly $a_n$ bits, by representing the whole list as a single string of $a_n$ bits, where the $i^{th}$ bit indicates whether $i$ is in the list. So the (3,5) system would be represented as five bits: "00101". I will refer to this henceforth as *bit-list encoding.*

A more efficient representation might be to use $k = \lceil \log_2 a_n \rceil$, bits to represent each number. I will call this *number-list encoding.* But this is only more space efficient when the numbers are fairly sparse, specifically when

$$k * n < a_n$$

$$n < \frac{a_n}{\lceil \log_2 a_n \rceil}$$

Our (3,5) problem would be represented as six bits, "011101", assuming we knew that each number would take exactly three bits.

So, for example, if $a_n = 100$, we need 7 bits to represent each number, and our whole list exceeds 100 bits if there are 15 or more numbers.

## 5. TIME COMPLEXITY BOUNDS

We would like to show that if the index of primitivity calculation is $O(E)$, that the Frobenius problem is subexponential in the number of bits used to express it. But what would that entail?

Algorithm 2 requires building an $a_n \times a_n$ matrix. We have two options to consider: bit-list encoding and number-list encoding.

For number-list encoding, it takes $b = \lceil \log_2 a_n \rceil$) bits to represent $a_n$. (The problem itself will take $n\lceil \log_2 a_n \rceil$ to represent, but only the bits for $a_n$ figure into the algorithm's space and time complexity). Creating a Boolean matrix from this number requires $a_n \times a_n$ entries in the matrix, with one bit each. Since $a_n = 2^b$, it will take $2^{b^2}$ or $4^b$ bits to represent the matrix, and therefore at least $O(4^b)$ time.

For large $n$ relative to $a_n$, as mentioned before, it is more efficient to use bit-list encoding. If we do things this way, there will be one graph node per bit, so the space and time will be $O(b^2)$.

Now suppose we had an $O(E)$ algorithm for the index of primitivity of a Coin matrix. We know the number of edges in this Coin graph must be at least $n$, and less than $2n$, so $O(E) = O(n)$.

Of course with the number-list encoding, this speedy algorithm would not help: solving the problem would still take $O(4^b + cb)$ time, no better asymptotically than $O(4^b)$. No matter how fast the index of primitivity algorithm is, it cannot get past the exponential hurdle in coding up the matrix. Because of this, the known NP-hard status of the Frobenius problem places no constraints on the complexity of index of primitivity.

However with the second coding scheme, the complexity of the index of primitivity the algorithm starts to matter. If it takes $O(b^2)$ time to create the matrix, and $O(b)$ time to calculate the index of primitivity, that puts us at $O(b^2)$.

Lamentably, we do not know of an $O(E)$ algorithm for the index of primitivity. Heap and Lynn's algorithm in Figure 1 takes $O(n^3 \log n)$. Since the $n$ here is the same as $b$ in bit-list encoding, this part of the calculation dominates the complexity, and we have $O(b^3 \log b)$. Thus the Frobenius problem is not NP-hard under bit-list encoding.

## 6. USING TOEPLITZ MATRICES

A minimal Frobenius matrix is a Toeplitz matrix. Since a Toeplitz matrix has fewer degrees of freedom than an arbitrary matrix, there are algorithmic speedups available for operations on them.

Unfortunately the product of two Toeplitz matrices is not necessarily another Toeplitz matrix, and the square or power of a Toeplitz matrix may not be a Toeplitz matrix.

However, I conjecture that all integer powers of minimal Frobenius matrices are Toeplitz matrices. I have not been able to prove this, but by exhaustive software search, I have ruled out coin problems under $n \leq 10$ and $a_n \leq 10$ and exponents $k \leq 30$.

I have found some facts that would seem to bear on this issue, but they do not yet add up to a proof:

- Powers of Minimal Frobenius matrices are only "Toeplitz" in the Boolean sense: their zeroes all line up diagonally. The positive numbers are not the same along diagonals, but we do not care about the particular non-zero values for the purposes of index of primitivity calculations, and therefore for Frobenius calculation.
- Minimal Frobenius matrices have all zeroes above the upper superdiagonal, and from my experiments with various Toeplitz matrices, it appears to be the case that the Toeplitz matrices which are *not* closed under exponentiation, seem to be the ones that do not share this property.
- Circulant matrices are a sublass of Toeplitz matrix that are known to be closed under multiplication, unlike Toeplitz matrices [3]. Minimal Frobenius matrices are not circulant, however. My intuition was that they would work just as well for the representation of the Frobenius problem, but it turned out not to be the case, at least without making any other modifications to the algorithm.
- Multiplying two Minimal Frobenius matrices does not necessarily result in a Toeplitz matrix. Multiplying two powers of a particular minimal Frobenius matrix, however, does seem to.

It is risky to build an algorithm based on an unproven conjecture, but using this technique as part of the algorithm to calculate Frobenius numbers for the large problems cited by Heap and Lynn [5] gives the same results. So the conjecture seems sound, if unproven.

The time benefits are clear from the algorithm in the figure below: two Minimal Frobenius matrices can be multiplied in $O(n^2)$ time. Getting the index of primitivity takes $O(\log n)$ matrix multiplications for an $n \times n$ matrix, which means we can reduce the index of primitivity calculation down from $O(n^3 \log n)$ to $O(n^2 \log n)$. Here is an algorithm for multiplying two minimal Frobenius matrices. It assumes they are stored as one-dimensional arrays indexed from $-n$ to $n$:

```
MIN-FROB-MAT-MULT(A, B):
   Create a new matrix C, with all entries=0

   For R from 0 to N
     For I from 0 to N-1
       C(-R) = A(-I)*B(I-R)

   For R from 1 to N
     For I from 0 to N-1
       C(R) = A(R-I)*B(I)

   Return C
```

**Figure 3: $O(n^2)$ Matrix multiplication algorithm for use in exponentiation of minimal Frobenius matrices**

Turning this matrix multiplication algorithm into an index of primitivity algorithm follows by simply replacing the $O(n^3)$ matrix multiply in Figure 1

Both loops execute in $O(\log((n-1)^2+1)) = O(\log n^2) = O(\log n)$. If the matrix multiply takes $O(n^3)$, then the algorithm overall is $O(n^3 \log n)$.

However for the particular case of the Frobenius problem where we can use the more specialized matrix multiplication, the index of primitivity algorithm becomes $O(n^2 \log n)$. Put together with the construction of the matrix, we have either $O(4^b \log 2^b) = O(b\ 4^b)$ for number-list encoding, or $O(b^2 \log b)$ for bit-list encoding.

Another speedup is possible as well. Because these are boolean matrices, with no subtraction, we can short-circuit the dot product calculation in the center of the double loop, as soon as we encounter a non-zero product. In the best case, this makes Toeplitz multiplication $O(n)$; in the worst case, it does not save multiplications, but in fact adds comparisons. Assuming a boolean comparison is about the same cost as a boolean multiply, this only really doubles the cost, so there is no asymptotic harm. As I will show below, however, this speedup did not turn out to be of practical importance.

## 7. EMPIRICAL RESULTS

Table 1 is a speed comparison with the problems Heap and Lynn [5] tried their algorithm on. Because I have run this algorithm on faster hardware (the Heap and Lynn article is dated 1965), the column of interest is the Ratio, representing how much faster this calculation ran on my 2Ghz Intel MacBook than on their "English Electric-Leo KDF 9 computer". What the ratio column demonstrates is that the problems with larger maximum denominations also show a faster speedup, indicating that I have improved somewhat on their algorithm in terms of complexity.

| n | Coins | g | Heap/ Lynn [5] | Bogart | Ratio |
|---|---|---|---|---|---|
| 4 | 140, 141, 144, 145 | 3919 | 102 | 8.93 | 11.4 |
| 6 | 130, 135, 140, 141, 144, 145 | 1452 | 84 | 7.51 | 11.2 |
| 8 | 120, 125, 130, 135, 140, 142, 144, 145 | 883 | 78 | 6.05 | 12.9 |
| 3 | 137, 251, 256 | 4948 | 348 | 22.11 | 15.7 |
| 10 | 239, 241, 251, 257, 263, 269, 271, 277, 281, 283 | 2866 | 390 | 27.43 | 14.2 |
| 4 | 271, 277, 281, 283 | 13022 | 510 | 36.54 | 14.0 |

**Table 1: Runtimes (seconds) for various Frobenius problems: Common Lisp on a 2GHz MacBook vs. the "English Electric-Leo KDF 9". Column g is the Frobenius number**

It is hard to compare the time complexity of index of primitivity for different problem sizes, since the number of calculations is highly problem-dependent. So in addition the table above, I decided to verify the time complexity of divide-and-conquer matrix exponentiation using three tecnhiques below.

While taking advantage of the Toeplitz property of the matrices was helpful, the short-circuited dot products did not help dramatically. Figure 2 shows the results of running matrix exponentiations with each of the three algorithms. Since I was not doing anything to improve the number of matrix multiplications involved, the comparison is between the same power, but comparing between different sized matrices. I chose to raise them to the 511th power since this involves 16 matrix multiplications. The matrices were fairly sparse: they were constructed as minimal Frobenius graphs for two coins: (3,n) (which isn't a valid problem for 30, 60, and 90, but that should not affect the complexity of the exponentiation step).

| n | power | Full | Toeplitz Short-cut | Toeplitz |
|---|---|---|---|---|
| 10 | 511 | .10 | .03 | .04 |
| 20 | 511 | .67 | .10 | .15 |
| 30 | 511 | 2.09 | .32 | .26 |
| 40 | 511 | 5.13 | .40 | .52 |
| 50 | 511 | 9.90 | .66 | .81 |
| 60 | 511 | 16.27 | 1.32 | .99 |
| 70 | 511 | - | 1.36 | 1.49 |
| 80 | 511 | - | 1.82 | 1.91 |
| 90 | 511 | - | 3.00 | 2.17 |
| 100 | 511 | - | 2.98 | 2.89 |

**Table 2: Timing for matrix exponentiation with various algorithms. All require 16 matrix multiplications. "Full" uses a standard $O(n^2)$ representation of a matrix; "Toeplitz" uses a 1-d matrix of size $2n+1$ to represent them, and "Toeplitz shortcut" cuts off dot product calculations as soon as a non-zero result is acheived.**

Table 2 and Figure 4 show the results, and demonstrate that the short-cut technique hurts as often as it helps, at least in the cases I ran.

The linearity of the graph on a log-log scale demonstrates that both algorithms are polynomial in $n$.

The Full line has a slope here of $(\log 16.27 - \log .1)/(\log 60 - \log 10) = 2.8$, implying a complexity of $O(n^{2.8})$; a little less than the expected comlexity of $O(n^3)$ for matrix multiplication.

The Toeplitz line has a slope of $(\log 2.89 - \log .04)/(\log 100 - \log 10) = 1.86$ implying a complexity of $O(n^{1.86})$; a little less than the expected complexity of $O(n^2)$ for this algorithm.

The shortcut line is too irregular to extract a meaningful slope out of it, at least compared with the non-shortcut Toeplitz line.

## 8. RELATIONSHIP BETWEEN $N$ AND $G$

One factor that weighs in the choice of encoding scheme is the fact that perhaps the Frobenius problem is more difficult for smaller n. Given a highest coin denomination, the Frobenius number, $g$, *generally* varies inversely with the total number of denominations available, $n$.

One would hope that g strictly decreased as a function of n. This would give us some hope that by limiting the number of edges in the Frobenius graph, we could limit the size of g, and perhaps be able to find it faster. However this turns out not to be the case. I found counterexamples even for quite small problems. For example the $n = 3$ problem of (7,6,3) has a Frobenius number of 11, but a smaller $n = 2$ problem of (7,2), with the same highest coin, has a Frobenius number of only 5.

However, on the whole we can still say that small values of $n$ make for more difficult Frobenius problems, and that therefore the number-list encoding (for which the Frobenius

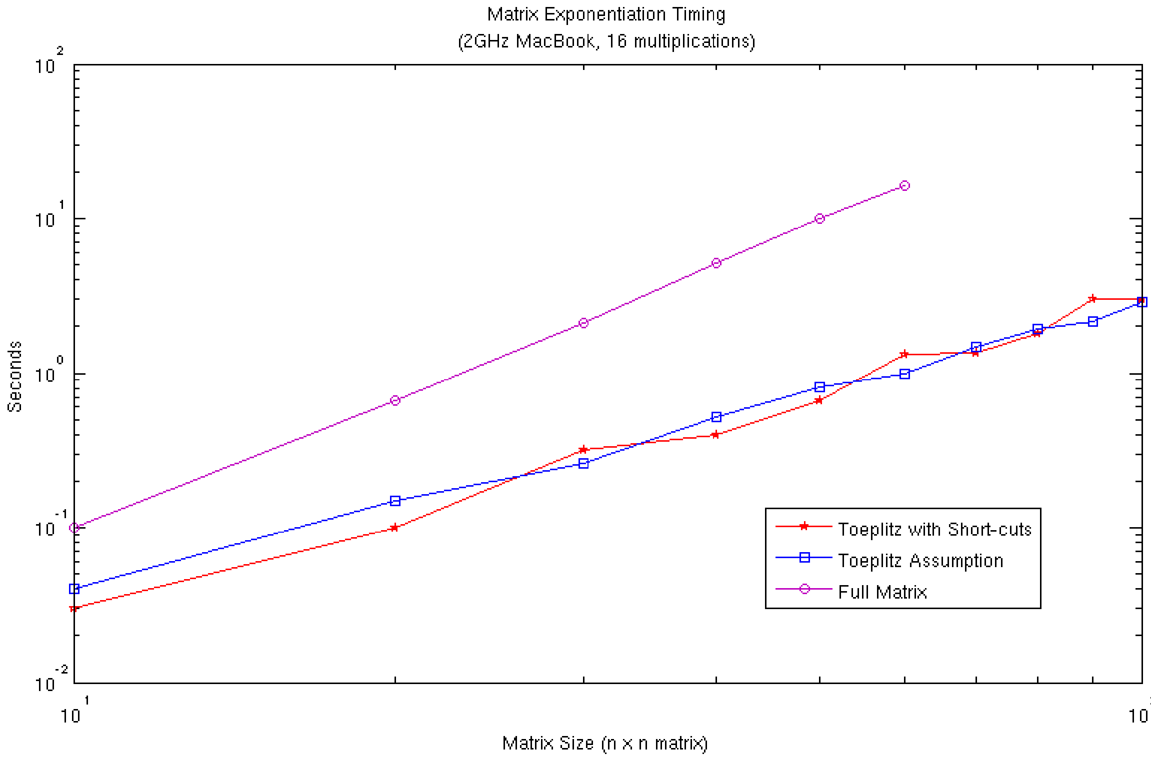



**Figure 4: Matrix Exponentiation times, log-log scale**

problem is NP-hard) is the more interesting one to consider.

## 9. LAZY MATRIX SQUARING

In this section I describe an earlier attempt to speed up the Index of Primitivity algorithm.

I originally started working with Coin matrices, and it seemed like a lot of extra work, taking $O(n^3)$ time to square these matrices which, after all, are very sparse. That suggested an opportunity for speedup.

Is there a faster way to square a matrix, if it is sparse? Suppose we represent an $n \times n$ boolean matrix as a kind of double adjacency list: for each of the $n$ rows we store a sorted list of columns that contain a 1, and for each of the $n$ columns we store a list of the rows that contain a 1.

```
Given:
  M.COLS[i] = {cols with 1 in row i}
  M.ROWS[i] = {rows with 1 in col i}
And starting with empty:
  SQ.COLS[]
  SQ.ROWS[]

For i = 1 to N
    For every possible pair (M.COLS[i][j],M.ROWS[i][k])
       Add j to SQ.ROWS[i]
       Add k to SQ.COLS[i]
```

**Figure 5: Sparse Boolean Matrix Multiplication**

Suppose we call a matrix with no more than q nonzero entries in any row or column a "degree-q matrix". Then for an $n \times n$ degree-q matrix, this algorithm should take $O(n) = nq^2$ operations to square the matrix.

A Leslie or a Coin matrix are degree 2, so they can be squared in $O(n)$ operations.

But to calculate the index of primitivity, the squaring has to continue up until the point where there are no zeroes. If there are no zeroes, $q = n$, and the algorithm is $O(n^3)$.

Does it save any time in the aggregate? Squaring a matrix will at worst results in a matrix of degree $q^2$, and at best a matrix of degree $q$. If we supposed that squaring repeatedly over the course of finding the index of primitivity increased the degree linearly with each squaring, then on average $q = n/2$, resulting in an average $O(n^3/4) = O(n^3)$. This is no improvement.

## 10. CONCLUSIONS

I have discussed some tradeoffs in the coding of the Frobenius problem. The tradeoffs are a bit academic, in that two representations of similar length can lead to exponential or low polynomial complexity for the same problem. To come up with a useful answer to the question of the complexity of these problems, one would have to ask to what types of large numbers would we like to apply them for some application: large numbers of coins, or large denominations of coins.

I have shown that Heap and Lynn's algorithm can be sped up by the observation that the matrices involved in their index of primitivity calculations are of a variety that can be multiplied in $O(n^2)$ instead of $O(n^3)$ time.

While I improved on the speed of Heap and Lynn's Frobenius algorithm, I did not actually prove it correct. One route to doing this may simply be to find a reformulation in terms of a circulant matrix; but a reformulation, if it exists is not obvious. The other route would be to prove my conjecture about powers of minimal Frobenius matrices.

## 11. REFERENCES


[1] Jorge L. Ramírez Alfonsín. *The Diophantine Frobenius Problem*. 2005.

[2] Paul Cull, Mary E. Flahive, Robby Robson, and Robert O. Robson. *Difference Equations*. 2005.

[3] R. M. Gray. *Toeplitz and circulant matrices: A review*, volume 2, Issue 3, pages 155–239. Now Publishers Inc, 2006. [Online] http://ee.stanford.edu/~gray/toeplitz.pdf [Accessed: March 16, 2009].

[4] B. R. Heap and M. S. Lynn. A graph-theoretic algorithm for the solution of a linear diophantine problem of frobenius. *Numerische Mathematik*, 6(1):346–354, December 1964.

[5] B. R. Heap and M. S. Lynn. On a linear diophantine problem of frobenius: an improved algorithm. *Numerische Mathematik*, 7(3):226–231, June 1965.

[6] J. L. Ramírez-Alfonsín. Complexity of the frobenius problem. *Combinatorica*, 16(1):143–147, March 1996.


## 12. APPENDIX

The code used to explore these questions and to test the algorithm is available online at:

http://engr.oregonstate.edu/~bogart/frobenius.html