**A Mini Project Report on**

**‘Efficient exponentiation’**

**Submitted By**

Mehul Dharmraj Patil (S174081)

Niranjan Vinod Patil (S174083)

Pranay Avinash Patil (S174084)

Nikhil Potale(S1740)

A Project report submitted as a partial fulfillment towards Practical of Applied Mathematics for semester IV of S. Y. B.Tech. (Dept. of Computer Science & Engg.)

2017-18

Under the guidance of

**Dr. Suresh Hiremath**

Department of Computer Science & Engineering

MIT Academy of Engineering, Alandi (D), Pune-412 105

**C E R T I F I C A T E**

*This is to certify that* ***Mehul Dharmraj Patil (S174081), Niranjan Vinod Patil (S174083), Pranay Avinash Patil (S174084), Nikhil Potale(S1740)*** *of MIT Academy of Engineering, Alandi (D), Pune have submitted MATLAB Project report on “****Efficient exponentiation****” as a partial fulfillment of Semester-IV S. Y. B.Tech. for completion of Applied Mathematics Practical work during the academic year 2017-18.*

**Dr. Suresh Hiremath Prof. Satish Gajbhiv**

**(Guide) (Head, Applied Sciences)**

**External Examiner**

**ACKNOWLEDGEMENT**

I take this opportunity to record my profound gratitude and indebtedness to Dr. Suresh Hiremath, Assistant Professor, Department of Applied Sciences for their inspiring guidance, valuable advices, constant encouragement and untiring supervision throughout my project work.

I express my deep sense of gratitude to Prof. Satish Gajbhiv Head, Dept. of Applied Sciences, for his continuous inspiration and encouragement.

Finally, I would like to acknowledge and express my special thanks to my family, friends and classmates for their patience, encouragement, support they have made during the period of this work.

**Name of Students**

Mehul Dharmraj Patil (S174081)

Niranjan Vinod Patil (S174083)

Pranay Avinash Patil (S174084)

Nikhil Potale(S1740)

**INDEX**

List of Figures …………………………………………………………………….. *5*

Abstract …………………………………………………………………………… 6

CHAPTER 1 INTRODUCTION ……………………………………………. 7

### Problem Definition

* 1. Theoretical Background

CHAPTER 2 METHODOLOGY …………………………………………… 9

CHAPTER 3 ALGORITHM ……………………………………………….. 14

CHAPTER 4 IMPLEMENTATION IN MATLAB ………………………… 15

CHAPTER 5 RESULTS ……………………………………………………. 16

CHAPTER 6 CONCLUSION ………………………………………………. 17

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title of Figure** | **Page No.** |
| 1 | Chain stage 1 | 11. |
| 2 | Chain stage 2 | 12. |
| 3 | Chain stage 1 | 13. |
| 4 | Chain stage 4 | 14. |

**ABSTRACT**

We have generally seen that the shortest addition-chain algorithm requires no more multiplications than binary exponentiation and usually less. The first example of where it does better is for *n*15, where the binary method needs six multiplications but a shortest addition chain requires only five.

But, if we perform a calculation of the minimum number of multiplication to find out nk where the exponent is very high, it takes a lot of time to calculate the minimum number of multiplication to find out nk. As if there a variable n having exponent term 116 then it takes 9 multiplications to get the result without any algorithm. Perhaps if an algorithm is designed which will reduce the time complexity of calculations and will perform it faster, it will be a better way.

Hence we have designed an algorithm and program to solve such kind of problem.

Our aim is to reduce the time taken by a computer for calculate the minimum number of multiplication to find out nk..

**CHAPTER 1: INTRODUCTION**

* 1. **Problem Definition**

The most naive way of computing n 15 requires fourteen multiplications:

n × n × ... × n = n15

But using a ‘binary’ method you can compute it in six multiplications:

n × n = n2

n2 × n2 = n4

n4 × n4 = n8

n8 × n4 = n12

n12 × n2 = n14

n14 × n = n15

However it is yet possible to compute it in only five multiplications:

n × n = n2

n2 × n = n3

n3 × n3 = n6

n6 × n6 = n12

n12 × n3 = n15

We shall define m (k) to be the minimum number of multiplications to compute nk; for example m(15) = 5.

For 1 ≤ k ≤ 200, find ∑ m (k).

**1.2 Theoretical Background**

A method of exponentiation by positive integer powers that requires a minimal number of multiplications called as addition chain exponentiation. It works by creating the shortest addition chain that generates the desired exponent. Each exponentiation in the chain can be evaluated by multiplying two of the earlier exponentiation results.

The shortest addition-chain algorithm requires no more multiplications than binary exponentiation and usually less. The first example of where it does better is for *n*15, where the binary method needs six multiplications but a shortest addition chain requires only five:

So, from the project statements we have to develop a function m (k) for calculating the minimum number of multiplication to find out nk and k is lies between the greater than or equal to 1 and less than or equal to 200 and also calculate sum of all possible solution or return values of that function m(k) in the range values if k.

**CHAPTER 2: METHODOLOGY**

We selected square and multiply method as solution to given MATLAB project statement.

**1) Square and multiply** **method:**

Square and multiply method works:

1. Convert the exponent (k) to Binary.
2. For the first 1, simply list the number.
3. For each ensuing 0, do square operation.
4. For each ensuing 1, do Square and multiply operations.

As an example, let’s run n5 and n37 through the process:

5=101 in binary.

1 First one lists number n

0 Zero calls for square (n)2

1 one calls for square + multiply ((n)2)2\*n

We went from requiring 5 steps to only 3.

37 = 100101 in Binary

1 First one lists number n

0 Zero calls for Square (n)2

0 Zero calls for Square ((n)2)2

1 One calls for Square + Multiply (((n)2)2)2\*n

0 Zero calls for Square ((((n)2)2)2\*n)2

1 One calls for Square + Multiply (((((n)2)2)2\*n)2)2\*n

We went from requiring 37 steps to only 7.

You can determine the total number of calculations required when using the Square and Multiple methods. Since each step requires at least a *square* operation, each binary digit (after the first) results in at least one operation. Then, since each binary 1 also calls for an additional *and multiply* operation, you count the binary 1’s a second time.

Or, to put it another way, ignoring the left-most binary 1, count each binary 0 as one operation, and each binary 1 as two operations. So we can have formula for calculating minimum number of multiplications to compute nk:

1) For odd numbers or first bit of binary number of k contains 1:

Minimum number of multiplication=

( 2 \* ( ( Number of ones in binary number of k ) - 1 ) ) + (Number of Zeroes in binary number of k ).

2) For even numbers or first bit of binary number of k contains 0:

Minimum number of multiplication=( 2\* ( Number of ones in binary number of k ) ) + ( Number of Zeroes in binary number of k ) – 2.

For using above technique Total sum of minimum number of multiplications to compute n^k if k range is 1 to 200 is 1688.

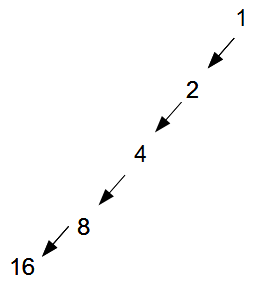
**Other method to calculate the minimal number of multiplications is Addition chain exponentiation using back tracing method.**

**2) Addition chain exponentiation:**

A method of exponentiation by positive integer powers that requires a minimal number of multiplications called as addition chain exponentiation. It works by creating the shortest addition chain that generates the desired exponent. Each exponentiation in the chain can be evaluated by multiplying two of the earlier exponentiation results. The shortest addition-chain algorithm requires no more multiplications than binary exponentiation and usually less.

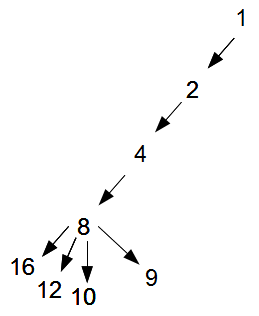
For that we generate a tree of solutions by combining all previously built solutions. The easiest way to approach this was through a recursive function where we keep track of the number of steps already taken as well as the path we are currently in. Then we explore the path until we reach the limit of 200 and then starting to backtrack and branch out.

So lets go through the algorithm (or part of it, for k ≤ 10). So We start at 1 and each time we do a binary exponentiation such that we take the number we just got and double it. That means we will go on till we hit 16, which is the first number which is larger than out limit. For each number we generate we keep track of the path length to reach it in an array.



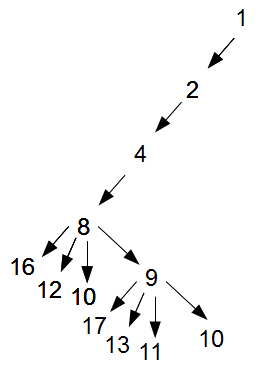
**Fig 1. Chain stage 1**

Once we hit 16 we start backtracking. So we go back to 8 and add the number prior to that, which is four and check the result. We do that until we reach a number which is smaller than our limit.



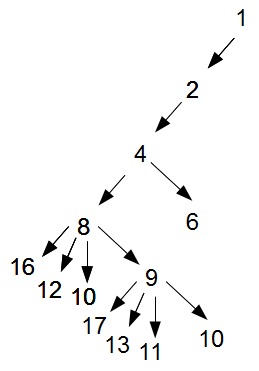
**Fig 2. Chain stage 2.**

So when we hit 8+1 we are finally smaller than the limit, and we will continue out of that path. So our path right now is 1, 2,4,8,9. Once again we will hit an awful lot of number larger than the limit.



**Fig 3. Chain stage 3**

Among them one interesting, the number 10. We have already seen that before. However the path to get to 10 is longer than last time we reached it, so we won’t update the path length. We have now used all options to generate numbers from the path going to 8, so we will backtrack to 4 and start expanding the tree there. That will include the child node 6 which can then be expanded.



**Fig 4. Chain stage 4**

Since the tree contains 1, we can prove that we will generate all numbers less than the limit at some point. However, we don’t want to do that right here. Once we have generated the whole tree all we need is to sum up the array of path lengths. And we get different combination for given exponentiation but we have to find minimum number of multiplication to find out nk hence we search path for minimum length of array and length of that array path is minimum number of multiplication to find out nk.

**CHAPTER 3: ALGORITHM**

1. START.
2. Set f\_count as variable equal to zero for calculate Total sum of minimum number of multiplications to compute nk if k range is 1 to 200.
3. Using for loop we considered following steps from 1 to 200:
4. We take number k (exponent) one by one from range 1 to 200 and convert that number in binary number.
5. Calculate length of bits of binary number.
6. Set ones and zeroes as variable equal to zero for calculating number of ones and zeroes in binary number.
7. Calculate number of ones and zeroes in binary number by for loop range from 1 to length of binary number.
8. Check given number (exponent) is even or odd by comparing first bit of binary number.
9. If odd numbers or first bit of binary number of k contains 1

Then

Minimum number of multiplication=

( 2 \* ( ( Number of ones in binary number of k ) - 1 ) ) + (Number of Zeroes in binary number of k ).

Else

if even numbers or first bit of binary number of k contains 0

then

Minimum number of multiplication=( 2\* ( Number of ones in binary number of k ) ) + ( Number of Zeroes in binary number of k ) – 2.

1. After that we add minimum number of multiplication to f\_count variable.
2. End of for loop of step 3.
3. Display Total sum of minimum number of multiplications to compute nk if k range is 1 to 200.
4. STOP.

**CHAPTER 4: IMPLEMENTATION IN MATLAB**

clear all;  
close all;  
clc;  
  
f\_count=0;  
for i=1:200  
   
d=de2bi(i);  
l=length(d);  
ones=0;  
zeroes=0;  
  
for j=1:l  
   
 if(d(j)==1)  
 ones=ones+1;  
 else  
 zeroes=zeroes+1;  
 end  
end  
if(d(1)==1)  
 count=(ones-1)\*2 +(zeroes);  
 else  
 count=(ones\*2)+(zeroes)-2;  
 end  
f\_count=f\_count+count;  
  
  
% disp(d);  
end  
fprintf('Sum of minimum value is :')  
disp(f\_count);

**CHAPTER 5: RESULTS**

Sum of minimum number of multiplication to find out nk if k is lies between 1 to 200 using square and multiply method is 1688.

**CHAPTER 6: CONCLUSION**

We develop code with a function m (k) for calculating the minimum number of multiplication to find out nk and k is lies between the greater than or equal to 1 and less than or equal to 200 and also calculate sum of all possible solution or return values of that function m (k) in the range values if k using square and multiply method and we get c but actual sum of minimum number of multiplication to find out nk if k is lies between 1 to 200 is 1582.

There is another method is addition chain exponentiation using back tracing But, unfortunately we have not make a code for efficient exponentiation using back tracing method. We require more time or find out any other logic to calculate efficient exponentiation using back tracing method.

**REFERENCES**

1. **Books:**

1. ‘Higher Engineering Mathematics’ - B .V. Ramana.

2. MATLAB for Engineers.

3. Introduction to MATLAB with Application.

1. **URL:**

1. <https://www.mathblog.dk/project-euler-122-efficient-exponentiation/>

2. <https://en.wikipedia.org/wiki/Addition-chain_exponentiation>

3. <http://www.practicalnetworking.net/stand-alone/square-and-multiply/>