Pow

XXX team

2019-09-21

CONTENTS

1 Introduction

1.1 Background

In the process of solving practical problems, it is often necessary to calculate $X^N(N \geq 0)$. The most simple algorithm is to multiply X by $N-1$ times, but in practical applications, the time complexity of this algorithm is too high to meet the requirements of projects. Therefore, a faster algorithm is needed.

1.2 Description

It is easy to find that in the algorithm of $N-1$ multiplication, many processes are repeatedly executed. To simplify the process, we can use such a property of the power of X :

$$X^N = \begin{cases} X^{\lfloor \frac{N}{2} \rfloor} \times X^{\lfloor \frac{N}{2} \rfloor} \times X & N\ is\ odd \\ X^{\lfloor \frac{N}{2} \rfloor} \times X^{\lfloor \frac{N}{2} \rfloor} & N\ is\ even \end{cases}$$

So X^N can be calculated by calculating $X^{\lfloor \frac{N}{2} \rfloor}$. Using the idea of recursion, to calculate $X^{\lfloor \frac{N}{2} \rfloor}$, just calculate $X^{\lfloor \frac{\lfloor \frac{N}{2} \rfloor}{2} \rfloor}$, until the index number becomes 1. In this recursive execution, each time N is reduced by half. Then we can use only $logN$ times to reduce N to 1. Therefore, the algorithm is able to calculate the value of X^N at the time complexity of $O(logN)$. The algorithm compared to the $N-1$ multiplication greatly improves the efficiency of the program.

2 Algorithm Specification

2.1 Algorithm 1:N-1 multiplications

Do $N-1$ multiplications.

Algorithm 1 $N-1$ multiplication

```
Input: $X$:base number; $N$:index number;
Output: $ans$:The value of $X^N$
ans ← X
for $i = 1$ to $N-1$ do
    $ans \leftarrow ans * X$
end for
```

2.2 Algorithm 2: iterative version

Record the binary information of N with an array. Enumerate each binary bit of N from the high position to the low,and multiply the answer by itself. When the bit is 1, multiply the answer by X.

Algorithm 2 Iterative version

```
Input: $X$:base number; $N$:index number;
Output: $ans$:The value of $X^N$
$cnt \leftarrow 0$
while $N \neq 0$ do
    $bit[cnt++] \leftarrow N\ Mod\ 2$
    $N \leftarrow N/2$
end while
$ans \leftarrow X$
for $i = cnt-1$ to 0 do
    $ans \leftarrow ans * ans$
    if $bit[i] = 1$ then $ans \leftarrow ans * X$
    end if
end for
```

2.3 Algorithm 3: recursive version

When $N = 1$,the function returns X. When $N \neq 1$, recursively call function to calculate $x^{\lfloor \frac{N}{2} \rfloor}$, then return $X^{\lfloor \frac{N}{2} \rfloor} \times X^{\lfloor \frac{N}{2} \rfloor} \times X$ if N *is odd*, otherwise return $X^{\lfloor \frac{N}{2} \rfloor} \times X^{\lfloor \frac{N}{2} \rfloor}$.

Algorithm 3 Recursive version

```
Input: $X$:base number; $N$:index number;
Output: $ans$:The value of $X^N$
function Pow($X, N$)
    $tmp \leftarrow$ Pow($X, N/2$)
    if $X$ Mod $2 = 1$ then return $tmp * tmp * x$
    else return $tmp * tmp$
    end if
end function
$ans \leftarrow$ Pow($X, N$)
```

3 Testing Results

3.1 Testing Results in Table

- Algorithm 1

N	1000	5000	10000	20000
Iterations(K)	10^7	10^6	10^5	10^5
Ticks	33426	16942	3295	6554
Total Time(sec)	33.4	16.9	3.295	6.554
Duration(s)	$3.34*10^{-6}$	$1.69*10^{-5}$	$3.30*10^{-5}$	$6.56*10^{-5}$
N	**40000**	**60000**	**80000**	**100000**
Iterations(K)	10^5	10^5	10^5	10^5
Ticks	13433	20344	25917	33167
Total Time(sec)	13.43	20.344	25.917	33.167
Duration(s)	$1.34*10^{-4}$	$2.03*10^{-4}$	$2.59*10^{-4}$	$3.32*10^{-4}$

- Algorithm 2 (iterative version)

N	1000	5000	10000	20000
Iterations(K)	10^8	10^8	10^8	10^8
Ticks	4862	6251	6733	7178
Total Time(sec)	4.862	6.25	6.733	7.178
Duration(s)	$4.86*10^{-8}$	$6.25*10^{-8}$	$6.73*10^{-8}$	$7.18*10^{-8}$
N	**40000**	**60000**	**80000**	**100000**
Iterations(K)	10^8	10^8	10^8	10^8
Ticks	7622	7873	8187	8381
Total Time(sec)	7.622	7.873	8.187	8.381
Duration(s)	$7.62*10^{-8}$	$7.87*10^{-8}$	$8.19*10^{-8}$	$8.38*10^{-8}$

- Algorithm 3 (recursive version)

N	1000	5000	10000	20000
Iterations(K)	10^8	10^8	10^8	10^8
Ticks	20687	26257	27151	29617
Total Time(sec)	20.687	26.257	27.151	29.617
Duration(s)	$2.07*10^{-7}$	$2.62*10^{-7}$	$2.71*10^{-7}$	$2.96*10^{-7}$
N	**40000**	**60000**	**80000**	**100000**
Iterations(K)	10^8	10^8	10^8	10^8
Ticks	31282	31784	33627	34333
Total Time(sec)	31.282	31.784	33.627	34.33
Duration(s)	$3.12*10^{-7}$	$3.18*10^{-7}$	$3.36*10^{-7}$	$3.43*10^{-7}$

3.2 Testing Results in Graph

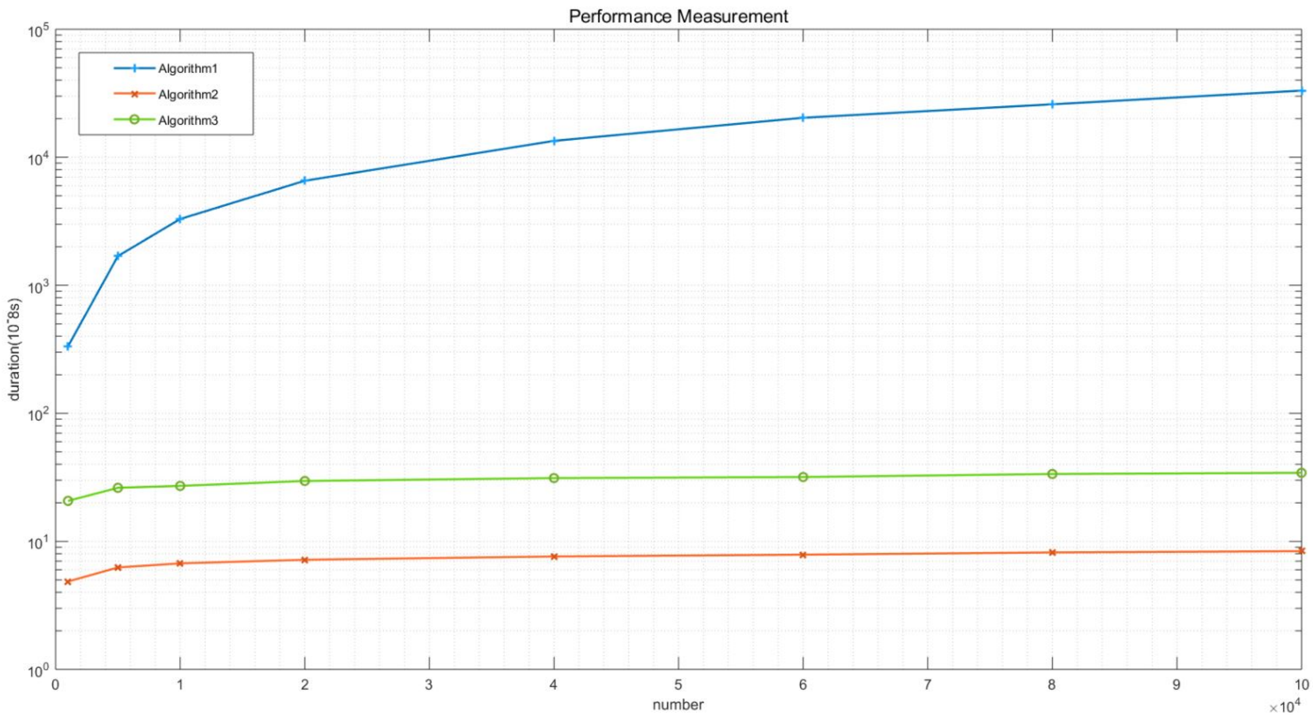

Figure 1:

Special note: Due to unstable running time, we have repeatedly tested the same data size for the same algorithm and calculate the average running time. Also, since the difference in running time of different algorithms is significant, I adjust the y-axis of the image to be unevenly distributed by the log function. At the same time, the unit of the y-axis is $10-8$ seconds, and the unit of the x-axis is 10^4.

4 Testing Analysis and Comments

4.1 Time Complexity Analysis

As we can see, different algorithms correspond to different time complexity, and the difference in running time is very significant. For Algorithm 1, its time complexity is $O(n)$. When the data size reaches 100,000, the operating efficiency is dramatically reduced. Although this algorithm looks straightforward and clear-cut, its runtime is linear with the size of the data. When the data size becomes larger, it takes too much time to reach the requirements of this project.

In Algorithm 2 and Algorithm 3, each time N is reduced by half. So we can use only $logN$ times to reduce N to 1. The optimization of this step algorithm greatly reduces the running time. The time complexity of Algorithm 2 and Algorithm 3 are both $O(logN)$. Therefore, with respect to Algorithm 1, as the data size continues to increase, the running times of Algorithms 2 and 3 increase steadily, while the running time of Algorithm 1 increases linearly.

However, although the time complexity of both Algorithm 2 and Algorithm 3 is $O(logN)$, Algorithm 2 is better than Algorithm 3 in terms of operational efficiency. On average, the running time of the iterative algorithm is one order of magnitude different from the running time of the recursive algorithm. This is because recursion calls the function repeatedly when dealing with problems, which increases its space and time overhead. For this problem, the recursive algorithm needs to perform $logN$ function calls, while the iterative algorithm only needs $logN$ iterations. The difference in efficiency is significant.

4.2 Space Complexity Analysis

Thinking from the perspective of spatial complexity, the spatial complexity of the Algorithm 1 is $O(1)$, but the spatial complexity of Algorithm 2 and Algorithm 3 is $O(logN)$.Although Algorithm 2 and Algorithm 3 are not as spatially complex as Algorithm 1, the operational efficiency is greatly improved. This means that sometimes it makes sense to sacrifice space for time. In the comparison of Algorithm 2 and Algorithm 3, we find that the efficiency and cost problem is the biggest drawback of recursion. Of the problems that can be easily solved using iterations, the use of recursion can simplify the thinking process, but it is not cost effective.

5 Appendix

Code:Please refer to the directory ../code

Raw data: Please refer to the file ./raw_data

6 Reference

[1]Performance Measurement (POW) https://pintia.cn/problem-sets/1173630572388691968/problems/1173631529453367296

7 Declaration

We hereby declare that all the work done in this project titled "Pow" is of our independent effort as a group.

Duty Assignment:

Programmer: XXX

Tester: XXX

Report Writer:XXX