浙江大學

fundamental data structure report1

Experiment name	Performance Measurement (POW)
Author's name	Tao hong yu 3200103929
Experiment date	2021/10/4

Contents

1 Chapter 1:Introduction

There are at least two different algorithms can calculate X^N. For a positive integer N, algorithm uses N-1 multiplication. Algorithm 2 works as follows:

if N is an even number :$X^N = X^{N/2} * X^{N/2}$

if N is an odd number :$X^N = X^{(N-1)/2} * X^{(N-1)/2} * X$

1.1 algorithm 1

To implement algorithm 1, we only need to multiply n numbers.

The pseudo code is as follows:

Algorithm 1 Calculate X^N

Require: $N \geq 0 \vee x \geq 0$
  $i \leftarrow 0$
  if $i \ < N$ **then**
    $res \leftarrow res * X$
    $i \leftarrow i + 1$
  end if
  return res

1.2 algorithm 2

And there are at least ways to implement algorithm 2, one is iteration and the other is recursion.

Iterative version: The pseudo code is as follows

Algorithm 2 Calculate X^N, Iterative version

Require: $N \geq 0 \vee x \geq 0$
  $temp = X$
  $i = 1$
  if $N\ mod\ 2 == 0$ **then**
    if $i\ < N$ **then**
      $temp1 \leftarrow temp * temp$
      $temp \leftarrow temp1$
      $i \leftarrow i * 2$
    end if
  else
    if $N\ mod\ 2 == 0$ **then**
      $temp1 \leftarrow temp * temp$
      $temp \leftarrow temp1$
      $i \leftarrow i * 2$
    end if
  end if
  return temp1

2 Chapter 2:Algorithm Specification

2.1 algorithm 1

The code of function function is as follows:

```
double function(double num,int N){
    double res = 1;
    if(N == 0)
        return 1;
    else{
        for(int i = 0;i<N;i++){
            res *= num;
        }
        return res;
    }
}
```

The time complexity of this algorithm is O(N), It uses a for loop that loops n times to get the nth power of X.

The spatial complexity of this algorithm is O(1).

2.2 algorithm 2

The function code of iterative version is as follows:

```
double function(double num,int N){
    double res;
    double temp = num;
    double temp1;//Return temp1 on return
    int i = 1;
    if(N == 0)
        return 1;
    else if(N == 1)
        return num;
    else if(N % 2 == 0){
        for(;i<N;){
            temp1 = temp * temp;
            temp = temp1;
            i *= 2;
        }
    }
    else{
        for(;i<N-1;){
            temp1 = temp * temp;
            temp = temp1;
            i *= 2;
        }
        temp1 *= num;
    }
    res = temp1;
    return res;
}
```

The time complexity of the code is O(log N). In this version of the function, n is continuously divided into two parts, and the temp variable is set to store the previous product. Each iteration, I is multiplied by 2, and temp is equivalent to square once until I is equal to n.

The spatial complexity of this algorithm is O(1).

The function code of recursive version is as follows:

```
double function(double num,int N){
    if(N == 0)
        return 1;
    if(N == 1)
        return num;
    if(N % 2 == 0)
        return function(num*num,N/2);
    else
        return function(num * num,N/2) * num;
}
```

The time complexity of the code is O(log N), for example to caculate $X^{6}2$, We only used nine multiplication. Obviously, the maximum number of multiplications required is 2 log N. Because it takes up to two multiplications to divide the problem in half.

The spatial complexity of this algorithm is O(log N). Each recursive call to the function will declare a new variable, so the spatial complexity of the algorithm is O(log N).

3 Chapter 3: Testing Results

We take x = 1.0001 and N as 1000, 5000, 10000, 20000, 40000, 60000, 80000 and 100000 respectively. Because the time of one test is too short, I repeat 1000000000 cycles to obtain large enough ticks. The test data results of the three algorithms are as follows:

	N	1000	5000	10000	20000	40000	60000	80000	100000
Algorithm1	iterations(K)	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000
	ticks	2657	13270	26423	53469	107586	156000	214099	267605
	total time(sec)	2.6	13	26	53	107	156	210	267
	Algorithm1 duration(sec)	0.000002	0.000013	0.000026	0.000053	0.000107	0.000156	0.00021	0.000267
Algorithm2 (iterative version)	iterations(K)	1000000000	1000000000	1000000000	1000000000	1000000000	1000000000	1000000000	1000000000
	ticks	22430	29658	32016	34199	38103	38069	41162	40000
	total time(sec)	22	29	32	34	38	38	41	40
	terative version duration(sec	0.000000022	0.000000029	0.000000032	0.000000034	0.000000038	3.8E-08	0.000000041	0.00000004
Algorithm2 (recursive version)	iterations(K)	1000000000	1000000000	1000000000	1000000000	1000000000	1000000000	1000000000	1000000000
	ticks	28752	36694	40308	44313	47784	47096	49431	49671
	total time(sec)	28	36	40	44	47	47	49	49
	ecursive version duration(sec	0.000000028	0.000000036	0.00000004	0.000000044	0.000000047	4.7E-08	0.000000049	0.000000049

Figure 1: result

Since the running time of algorithm 1 is too large compared with that of algorithm 2, the n-runtime line graph of iterative version and recursive version of algorithm 2 is shown below

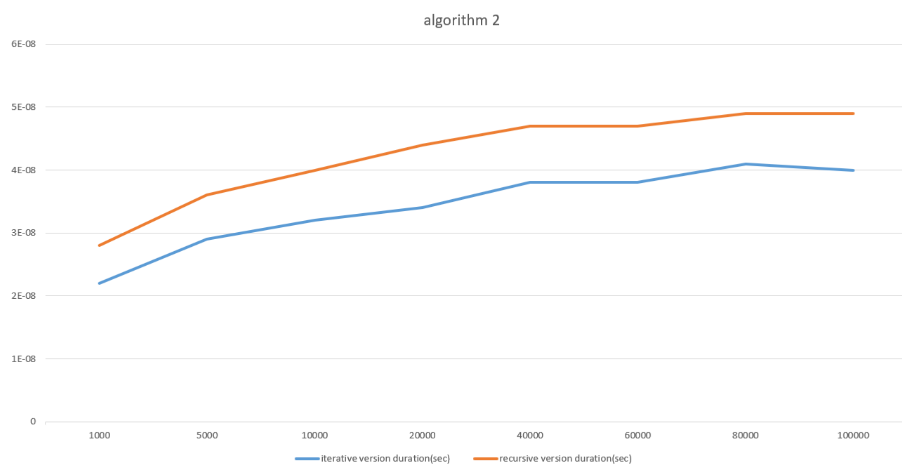

Figure 2: algorithm2 result

The following figure is the n-runtime diagram obtained by drawing the running time of two versions of algorithm 1 and algorithm 2 in the same line graph

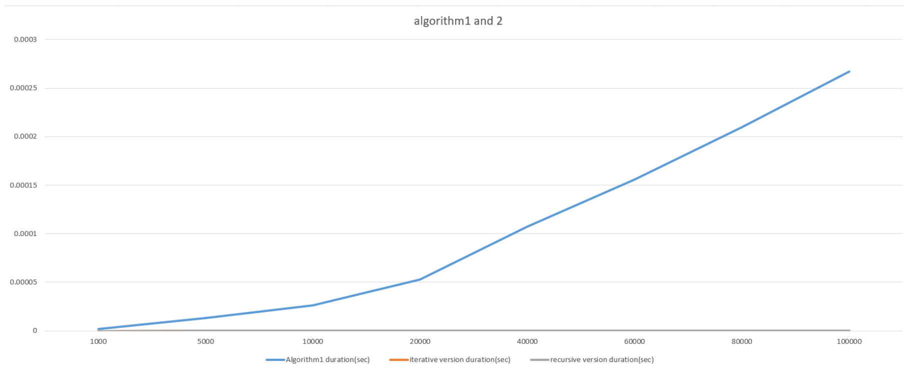

Figure 3: algorithm1 and 2 result

From the line graph, we can see that the running time of algorithm 1 is much longer than that of algorithm 2, and the running time of the two versions of algorithm 2 is almost the same

4 Declaration

I hereby declare that all the work done in this project titled ”POW” is of my independent effort.