Fundamentals of Data Structures

Laboratory Project 1

Performance Measurement(POW)

Date: 2023-10-07

Author:陈硕
Number:3220106406

Chapter 1: Introduction

This project uses two different algorithms to compute X^N for some positive integer N.. Then, we analyze the complexities of the two algorithms by measuring and comparing the performances of Algorithm 1 and the iterative and recursive implementations of Algorithm 2 for X=1.0001 and N = 1000, 5000, 10000, 20000, 40000, 60000, 80000, 100000.

Chapter 2: Algorithm Specification

Algorithm 1 : use N−1 multiplications.(Let's call it iterativepow1)

Here I use a loop to calculate.

```
//This function uses the Algorithm1 to conpute x^m
double iterativepow1(double x,long m){
    long i;
    double result=1;
    for(i=1;i<=m;i++){   //getting the result by using N-1 multiplications
        result*=x;
    }
    return result;
}
```

Algorithm 2 works in the following way: if N is even, X^N=X^(N/2)×X ^(N/2); and if N is odd, X ^N=X ^(N−1)/2×X ^(N−1)/2×X.

I implement algorithm 2 in a recursive version and an iterative version.

1、 **The recursive version of algorithm 2(Let's call it recursivepow)**

When exponent=0, return 1 as the exit of recursion. Otherwise, make the latter base square of the previous base to halve the times that we need to multiply.

```
//This function uses the recursive version of Algorithm2 to compute x^m
double recursivepow(double x,long m){
    if(m==0){      //the exit of recursion
      return 1;
    }else if(m%2==0){   //if m is even,x^m=(x^x)^(m/2);
        return recursivepow(x*x,m/2);
    }else{
        return recursivepow(x*x,m/2)*x;  //if m is odd,x^m=x*(x^x)^((m-1)/2)
    }
}
```

2、 **The iterative version of algorithm 2(Let's call it iterativepow2)**

I use a loop. The loop number i means the times left to multiply, so i starts from the exponent m. Every time we finish the operarion in a loop,i=i/2.

```
//This function uses the iterative version of Algorithm2 to compute x^m
double iterativepow2(double x,long m)
    {
    long i;
    double result=1;
    for(i=m;i!=0;i/=2){
        if(i%2==1)   //if i is odd,multiply x;Then, is even.
          result *= x;
        x=x*x;     //This operation halves the the number of x left to multiply
    }
    return result;
    }
```

Chapter 3: Testing Results

	N	1000	5000	10000	20000	40000	60000	80000	100000
Algorithm1	Iterations(K)	2086	1707	1200	493	4	149	85	40
	Ticks	16	15	15	15	15	16	15	16

	Total Time(sec)	0.016	0.015	0.015	0.015	0.015	0.016	0.015	0.016
	Duration(sec)	0.0000076702	0.0000087873	0.0000125000	0.0000304280	0.00375	0.0001073826	0.0001764706	0.0004
Algorithm2 (iterative Version)	Iterations(K)	168014	2626974	164597	188258	1669371	200323	348212	21124
	Ticks	14	15	15	15	16	15	16	15
	Total Time(sec)	0.014	0.015	0.015	0.015	0.016	0.015	0.016	0.015
	Duration(sec)	0.0000000833	0.0000000057	0.0000000911	0.0000000797	0.0000000096	0.0000000749	0.0000000459	0.0000007101
Algorithm2 (recursive Version)	Iterations(K)	127349	50207	333404	25496	10528	495370	148485	78183
	Ticks	16	10	15	16	14	25	16	14
	Total Time(sec)	0.016	0.01	0.015	0.016	0.014	0.025	0.016	0.014
	Duration(sec)	0.00001256	0.00001992	0.0000000450	0.0000006275	0.0000013298	0.0000000505	0.0000001078	0.0000001791

Chapter 4: Analysis and Comments

1、 Theoretically analyze the time and space complexity of the three methods Compute X^N:

1) Algorithm 1
   the loop will always circulate for n times, so the time complexity is O(n).
   the space complexity is O(1)
2) Algorithm 2(recursive version)
   Every time we make m/2 until m=0,then we get out of the recursion.
   So the time complexity is log(n).
   The space complexity is O(n)
3) Algorithm 2(iterative version)
   O(logn), just similar with 2)
   the space complexity is O(1)

2、 Some conclusions we practically get from our test data in Chapter 3

1) We can see that As a whole, algorithm 2 runs more quickly than algorithm 1,no matter the recursive version or the iterative version. What's more, as N grows larger, the gap between the two algorithms becomes larger.
2) In terms of algorithm 2, Iterative version runs more quickly than recursive version. I think maybe it's because recursion version has a process of backtracking, consuming some time. But the gap isn't very big, and when N = 100000,I even see that recursive version runs more quickly.

3、 Some comments and confusion

I think as a whole I finish the project perfectly and try my best to use as little code as possible to reach the goal.

But I find that the time don't grow very regularly as the time complexity shows, for example, time complexity of algorithm 1 is O(n),but the time it consumes isn't linear. This confuses me a lot.

Chapter 5: Appendix: Here are part of my Source Code (in C)

```
    do{
    start=clock();

    //run the testing function in the following line please
    res = function(x,m);    //for example,change the line to "res = recursivepow(x,m);"

    stop=clock();
    ticks+=(stop-start);
    iterations++;         //iterations represents the times the functions have been operated
    }while(ticks<10);    // ticks=10 is large enough if we want an accuracy of at least 10%.
    overallduration=(double)(ticks)/CLK_TCK;
    duration=overallduration/iterations;//computing the average operating time

    printf("Ans = %lf\n",res); //part of printing results
    printf("Iterations = %ld\n",iterations);
    printf("Ticks = %ld\n",ticks);
    printf("Totaltime = %lf\n",overallduration);
    printf("Duration = %.10lf\n",duration);
}
```

Chapter 6:Declaration

I hereby declare that all the work done in this project titled"陈硕" is of my independent effort