**Chapter 3: Testing Results**

1. **Testing Environment**

Intel Core i7-4700M CPU @3.22GHz

RAM 15.90GB

Dev-C++ 5.6.3 on Windows 8.1 (64bits)

1. **Some Modification**

Changing the preprocessing order, replace 1000 with a sufficient large number to avoid array bound

|  |
| --- |
| #define max 63000  #define MAX 63000 |

1. **Testing Results**

In this project, we tested the two different implementation of the project. And we compare the performance of the two algorithm.

It is obvious to see that in this project the time complexity depends on two aspect:

1. The building time of a Haffman tree

Obviously the first part is closely connected to the number of distinct characters, that is N;

1. The test time of all the students’ submission

The second part is closely linked with how many student submitted their homework that is M, and N as well. We define the average length of the code of a single character (example: for A 10010, the length of the code is 5) is E. Thus the time complexity is M\*(N\*E);

For the first algorithm, the theoretical time complexity of the queue implement algorithm is N^2+M\*(N\*E); The second algorithm’s theoretical time complexity is NlogN+M\*(N\*E)

**Case1: The number of student M is set to a constant: 1**

**Change the number of N.**

**Input:** N7.in; N17.in; N27.in; N37.in; N47.in; N60.in

**Comments**: In this case there are 6 test files, the number of the file indicates how many distinct characters are in the test file. For each input file we record the time of input Ti, and record the time of the whole program Tw, the actual running time of the core function is Tw-Ti

**Time Table and Graph**:

In the test the running time of a single run is too small to record, so we run the program for K times and record the time Tf, the time of a single run is Tf/K, in the test: Total iteration K=100000

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Num of distinct characters/N** | **Time of the whole program Tw/s** | **Time of input Ti/s** | **Time of the function Tf=Tw-Ti/s** | **Average time Tf/s** | **Avg time per run t/us** |
| N=7 | 2.625 | 2.453 | 0.172 | 0.144 | 1.44 |
| 2.593 | 2.484 | 0.109 |
| 2.578 | 2.437 | 0.141 |
| 2.594 | 2.421 | 0.173 |
| 2.546 | 2.421 | 0.125 |
| N=17 | 2.996 | 2.531 | 0.465 | 0.4884 | 4.884 |
| 3.052 | 2.525 | 0.527 |
| 2.969 | 2.531 | 0.438 |
| 3.043 | 2.531 | 0.512 |
| 3.015 | 2.515 | 0.5 |
| N=27 | 3.609 | 2.718 | 0.891 | 0.802 | 8.02 |
| 3.491 | 2.734 | 0.757 |
| 3.5 | 2.721 | 0.779 |
| 3.504 | 2.703 | 0.801 |
| 3.48 | 2.698 | 0.782 |
| N=37 | 4.046 | 2.964 | 1.082 | 1.0788 | 10.788 |
| 4.046 | 2.963 | 1.083 |
| 4.026 | 2.94 | 1.086 |
| 3.995 | 2.941 | 1.054 |
| 4.026 | 2.937 | 1.089 |
| N=47 | 4.698 | 3.125 | 1.573 | 1.4074 | 14.074 |
| 4.559 | 3.151 | 1.408 |
| 4.505 | 3.219 | 1.286 |
| 4.479 | 3.109 | 1.37 |
| 4.479 | 3.079 | 1.4 |
| N=60 | 5.526 | 3.359 | 2.167 | 2.1808 | 21.808 |
| 5.627 | 3.354 | 2.273 |
| 5.433 | 3.339 | 2.094 |
| 5.511 | 3.339 | 2.172 |
| 5.526 | 3.328 | 2.198 |

**Comments**: In case 1, M is set to a constant 1. Theoretically the Time complexity is O(N^2+N\*E)which is O(N^2). According to the test, the result conforms to the analysis.

**Case2: The number of distinct characters N is set to a constant:60**

**Change the number of submission by students: M**

**Input: M1.in; M200.in; M400.in; M600.in; M800.in; M1000.in**

**Comments**: In this case we set N as a constant and change M, the number of the input file indicates how many student submitted their homework, Theoretically the time complexity is linear O(M), because N is a constant

We just test the running time of the whole program because N is the same and the time of input is thus the same.

K=1000;

**Time Table and Graph**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Num of students /M** | **Time of the whole program Tw/s** | **Average time Tf/s** | **Avg time per run t/us** |
| M=1 | 0.062 | 0.053 | 53.0 |
| 0.047 |
| 0.062 |
| 0.047 |
| 0.047 |
| M=200 | 0.500 | 0.4876 | 487.6.0 |
| 0.516 |
| 0.469 |
| 0.484 |
| 0.469 |
| M=400 | 0.671 | 0.675 | 675.0 |
| 0.688 |
| 0.656 |
| 0.641 |
| 0.719 |
| M=600 | 1.062 | 1.028 | 1028.0 |
| 1.078 |
| 1.016 |
| 1.000 |
| 0.984 |
| M=800 | 1.421 | 1.4186 | 1418.6 |
| 1.422 |
| 1.406 |
| 1.422 |
| 1.422 |
| M=1000 | 2.031 | 2.0406 | 2040.6 |
| 2.031 |
| 2.016 |
| 2.078 |
| 2.047 |

**Comments**: In case2, N is set to a constant 60. Theoretically the Time complexity is O(M). According to the test, the result conforms to the analysis.

**Case3: Extreme case 1, all the frequency is the same**

**Input:** Ex1.in

**Comments**: In extreme case 1, all the frequency is set to the same constant, which is 1000

**Time Graph**:

**Comments**: In Extreme case 1,

**Case 4: Extreme case 2: the frequency of the character is in increasing/ decreasing order**

**Input: Ex2incre.in; Ex2decre.in; Ex2random.in**

**Comments**: In extreme case 2, the frequency is in increasing order from 2-1000,and the frequency in decreasing order from 1000-2 and a random order input. I design it to test whether the ordering affect the build haffman tree function, as the same M is set to a constant.

**Time Graph**:

**Comments**: According to the test result, in queue implementation, the order of frequency does not affect running time.

**Chapter4 Analysis**

The comment in each test case has analyzed thoroughly.

To summarize, in our first implementation (queue implementation), the time complexity is O(N^2+M\*N),

Building a haffman tree is O(N^2) and testing the student answer is O(M\*N) both M and N affect the time complexity greatly, the test result of case 1 and 2 conforms to our hypothesis.

And in our second implementation, the time complexity is O(NlogN+M\*N) according to our calculation and test results. O(NlogN) for building haffman tree and O(M\*N) for testing student results.