Sort Chinese

**INFO6205**

Haoyang Hu 001005743

Kaifeng Ruan 002133899

# Abstract

Your task is to implement MSD radix sort for a natural language which uses Unicode characters.

You may choose your own language or (Simplified) Chinese. Additionally, you will complete a

literature survey of relevant papers and you will compare your method with Timsort, Dual-pivot

Quicksort, Huskysort, and LSD radix sort.

# Introduction

The project is to analyze the efficiency of different sorting algorithms including MSD radix sort, Timsort, Dual-pivot Quicksort, Huskysort and LSD radix sort sorting on Unicode(Chinese). Chinese words are different from any other words. It is ordered by pinyin. As Chinese words can’t be sort simply by their Unicode order, we should use other method or library to sort Chinese.

# Discussion

At first, it turned out that there is a *Collator* class which can help us compare different Chinese words by setting *new Collator(). getInstance(Locale.CHINA)* . However, when we set up a unit test for *Collator.compare(),* it turned out that the *Collator* class couldn’t sort Chinese words correctly. At first, we thought that maybe *Collator* class follows English names principle, which it treats people’s first name more significantly rather than last name. Then we tested comparison between single Chinese words. It turned that it still couldn’t compare Chinese words properly. So, we searched for another method to sort Chinese words, which is*, pinyin4j.jar.* This library can help us transform Chinese words into pinyin, which the combination of the correct pronouncing order of Chinese words and the tones. These can be easily sorted as they are English strings and follows dictionary order.

# Benchmarking

## Benchmarking is a method to test the performance of code. It can also be used to identify the CPU or memory efficiency of a piece of code. We will use benchmarking to test different sorting times to compare and obtain the optimal sorting method.

## SortWithPinYin4j & SortWithCollator

Because we need to test the running time in converting and sorting, so we need to create a function that we can change the input parameter value to define the array length, then initializing the array and sort this array in pinyin order.

Then we have used the above-mentioned sorting functions to sort name array in correct pinyin order.

**文本

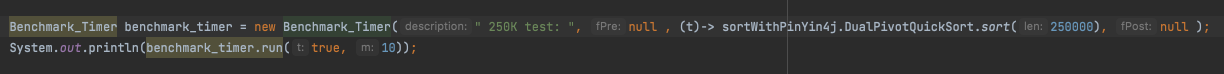
中度可信度描述已自动生成**

**文本

描述已自动生成**

## Benchmark

This package is gotten from the class repository, we can use Benchmark\_Timer class as a constructor to create any test objects in any benchmark main classes to test the running time in average, then print it in the console output.

****

# Outputs & Analysis

We have created two methods to covert Chinese into Pinyin with using the interface from Pinyin4j library and Collator library.

We can see the sorting plus converting time in each Benchmark class.

文本

描述已自动生成

Figure 3

Figure 3 shows the running time when we use the Pinyin4j library and Dual-pivot quick sort when the name length is up to 250k. We test the sorting and converting time in each benchmark method 10 times and get an average running time then repeating the benchmark function 8 times and get the output like figure 3 shows. We can see some other results from figures below.

文本

描述已自动生成

Figure 4

文本

描述已自动生成

Figure 6

文本

描述已自动生成

Figure 5

Figure 4 shows the running time when we use the Pinyin4j library and LSD radix sort when the name length is up to 4M.

Figure 5 shows the running time when we use the Collator library and Dual-pivot quick sort when the name length is up to 250K and 500K.

Figure 6 shows the running time when we use the Collator library and Timsort when the name length is from 250K to 4M. I just repeat the loop 3 times because it just wasted too much time.

Table 1 below shows the average result we get from each benchmark test.

The unit of result is millisecond.

Table 1

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name Length | Pinyin4j + Dual-pivot quick sort | Pinyin4j + Husky sort | Pinyin4j + LSD sort | Pinyin4j + MSD sort | Pinyin4j + three -way string quick sort | Pinyin4j + Tim sort | Collator + Dual-pivot quick sort | Collator + LSD sort | Collator + MSD sort | Collator + three -way string quick sort | Collator + Tim sort |
| 250K | 2214.6 | 2179.5 | 2365.6 | 2013.3 | 2052.5 | 2324.4 | 7663.8 | 2022.3 | 2038.7 | 2112.7 | 9487.7 |
| 500K | 2474.8 | 2482.8 | 2800.0 | 2137.7 | 2159.1 | 2753.2 | 14420.4 | 2078 | 2121.4 | 2240.0 | 17473.5 |
| 1M | 3094.2 | 3147.6 | 3598.4 | 2401.6 | 2456.3 | 3783.2 | 29122.1 | 2224.1 | 2355.3 | 2647.7 | 34395.0 |
| 2M | 4248.7 | 4062.3 | 5073 | 2923.1 | 2991.3 | 5901.6 | 59664.7 | 2437.1 | 2580.2 | 3326.0 | 68834.7 |
| 4M | 6400.0 | 5833.7 | 7931.6 | 3971.7 | 3927.5 | 10077 | 128182.4 | 2947.2 | 2950.2 | 4480.1 | 142393.2 |

# Conclusion

As we can analyze from the Table1. We assumed that in any sorts, the initializing array time and the converting time are same. The sorting time increases almost linearly with the increase of array length. Comparing the run time, we find that timsort and dual pivot quick sort is slower than other sorting algorithms. Since their time complexity are O(NlogN).

According to the run time data, we think basic MSD sort and LSD sort may be the best sorting algorithms in this case. And their time complexity is O(N). 3-way string quicksort is slightly slower or equal to MSD radix.

But according to what we had learned in class, three-way string quick sort should be an improved algorithm based on the MSD radix sort. We made an assumption that three-way string quick sort will do a better performance on longer strings. We set each names 10 times longer by adding themselves 10 times and shuffled each string. After we do the benchmark of MSD radix sort and three-way string quick sort on the latest data, we found that three-way string quick sort runs faster than MSD radix sort.

So, we made a conclusion that, MSD radix sort might have a better performance on short string arrays such as name arrays, and three-way string quick sort will have a better performance on longer string.

By using *Pinyin4j* library, we can sort Chinese words in two ways. At first, we don’t think these two sorting methods would affect the running time, but it truly happened. We think that’s because when sort with Collator, there are at most three Chinese characters to compare. And when sort with *pinyin4j*, there will be more English characters to sort with. However, when sorting with *Collator*, timsort and dual pivot quicksort run much slower than sorting with *pinyin4j*. That’s probably because the *Collator.compare()* runs slower than *String.compareTo(String)*.

# References

https://www.geeksforgeeks.org/timsort/

https://www.geeksforgeeks.org/dual-pivot-quicksort/

Hillyard, R. C., et al. “Huskysort.” ArXiv.org, 1 Dec. 2020, https://arxiv.org/abs/2012.00866.

https://www.geeksforgeeks.org/msd-most-significant-digit-radix-sort/

https://algs4.cs.princeton.edu/home/