Related Paper of MSD Radix Sort and Other Sorting Algorithms

Yuhan Yang(NUID:001094267) Luo Chen(NUID:001564677) Zhijie Li(NUID:001563872)

Abstract

The task of our project is to implement MSD radix sort for a natural language which uses Unicode characters. Since all three of us are both came from China, we choose Simplified Chinese as our target language. Additionally, we will compare your method with Timsort, Dual-pivot Quicksort, Huskysort, and LSD radix sort. We will have some background introduction in section 2; the analysis discussion, which is mainly the comparation with other sorting algorithms in section 3 and the theory of our algorithm in section 4.

Background

In this section, we will have some basic introduction of Huskysort and radix sort. Firstly, radix sort, a non-comparison sorting algorithm. Radix Exchange is an internal sorting method written for a fixed word length, internally binary machine. In general, a radix R sort involves the repeated distribution of the items into R "pockets" or areas according to a value of each digit in the key, expressed as a number to the base R using a positional representation[1]. In addition, there are two typical types of radix sort. MSD(the Most Significant Digit) radix sort and LSD(the Least Significant Digit) radix sort. The difference between this two types of radix sort is basically that MSD radix sort starts the sorting from the beginning of the string while the LSD radix sort starts from the end. As for Huskysort, it is a newly developed sorting algorithm combining quicksort with Timsort for object types that are costly to compare[2], including String, LocalDateTime, BigInteger, BigDecimal, and many user-defined types. Then main theory of Huskysort is to encode the item into long datatype, while comparing and switching the long-datatype item, switching the origin items as well. After that, run the system sorting algorithm to make sure the items are actually sorted. With the benchmark result, huskysort actually made quite a progress comparing with other sorting algorithm.

Analysis Discussion

In this section, we will mainly talking about the comparation between radix sort and other algorithm, which are came from the paper and research of other people. First we will take a look at the compare between radix sort and bucket sort. After that is the compare between various sorting algorithms, including the fundamental comparison sorting algorithm and radix sort.

3.1 Comparative Analysis of Bucket and Radix Sorting [3]

According to the paper by Sai Krishna Kovi, bucket sort and radix sort are two linear sorting algorithms, which in best case, can be possibly faster than most others' comparing algorithms. They are both stable if array is well processed. However, they both depends heavily on the input elements. Generally, bucket sort is quicker and better in dealing with pre-sorted array than radix sort while requiring more space.

Bucket sort requires extra memory for buckets, and in ideal case need to know maximum and minimum of the number. The property of recursively sort and counting on each bucket is applicable for parallelism. If the input is known in advance to be well dispersed, bucket sort is very suitable. By choosing good number of buckets, the performance can be lifted to linear. On the other hand, if number of buckets is too big or too small, or input is of same value, the elements would cluster in same bucket, which would obviously worst performance. As a result, bucket sort need pre-process and take an appropriate estimate of input array.

Radix sort is able to deal with bigger keys more productively, especially if arrays is partial-ordered. Compared with bucket sort, radix sort also requires extra memory, but far less than bucket sort, so in big amount input, radix sort is more acceptable than bucket sort.

3.2 A Comparative Study of Various Sorting Algorithms [4]

As the title suggested, this article is about the comparative study of the time complexity between various sorting algorithms, from the basic comparison sorting algorithms like bubble sort, insertion sort and selection sort, to the non-comparison sorting algorithms like radix sort, counting sort and bucket sort.

The article started with the different classifications of the sorting algorithms: Based on the way of generating the output, sorting algorithm can be value-based or index-based. Value-based sorting generates the output directly in the form of values being sorted while index-based sorting generates a list of indices which are rearranged in such a way that the values at those indices form a sorted list. On the other hand, based on the logic implemented to sort the values, sorting algorithms are broadly categorized into 2

classes: comparison sorting algorithms and non-comparison sorting algorithms, which we are quite familiar with.

The article focused on the classification of comparison and non-comparison algorithms. It pointed out that the lower bound of the comparison algorithms is overtaken by the non-comparison algorithms by using extra space.

By using the control flow graph combining with the logic and the code of different sorting algorithms, the article was able to analyze their time complexity. After the analyze of the time complexity, the article also mentioned about the space complexity and the overall benefits and limitations of each algorithm.

For example, by analyzing the code of radix sort, the author converted it into the control flow graph which has 13 nodes and start and end node. Inside the control flow graph there were six loops. By analyzing the time complexity of every loop, it came out with the conclusion that the overall time complexity of radix sort is O(n). Meanwhile, the radix sort is preferred to be implemented in the scenario when the length of all the values in the array is same. The space complexity of radix sort is O(n+k) while k is the number of distinct values in the array. At the end is the benefits and limitations of radix sort like radix sort takes more extra space.

After the analyze of six sorting algorithms, bubble sort, insertion sort, selection sort, counting sort, radix sort and bucket sort, the conclusion section is the last part of the article. The conclusion is that comparison sorting algorithms are preferred over non-comparison sorting algorithms for their simplicity when the number of values to be sorted is small, but as the size of data set increases, non-comparison sorting algorithms overtake them in terms of time complexity at the cost of extra space resources required to execute them.

Algorithm

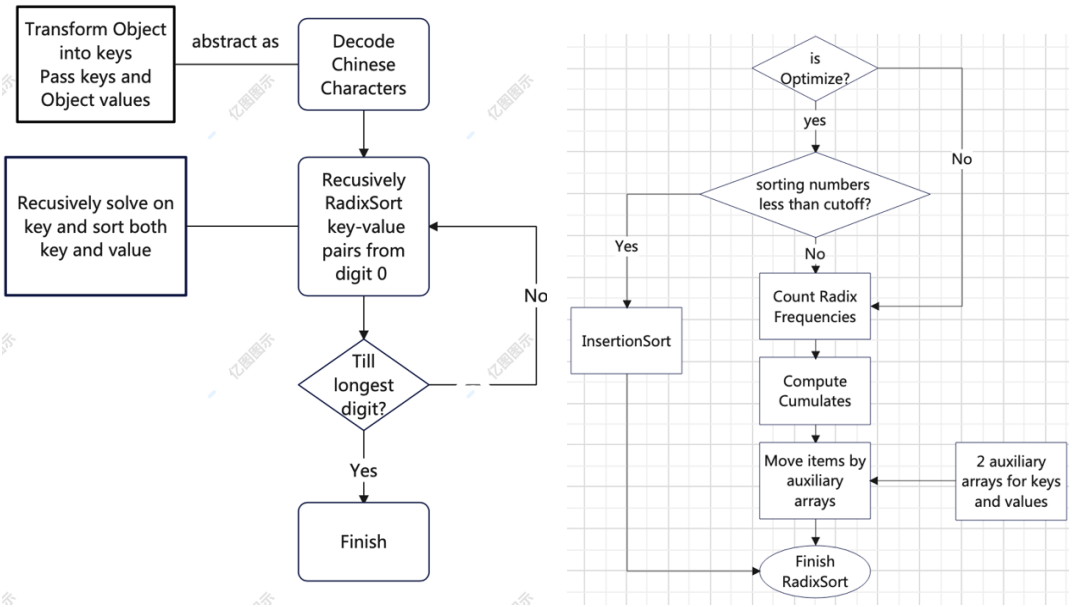

Fig. 1 Flow of MSD radix sort (top level)

Fig.2 Flow of Radix Sort

4.1 overview of MSD radix sort

Step 1: To encode Chinese Character into corresponding arrays of UNICODE (16-bit) by API of Collator class. (see Flow 1)

Step 2: Recursively sort key and value arrays from digit 0 to last digit by radix sort. By comparing UNICODE array (keys), move encoding arrays as well as original Chinese Characters arrays. (see Flow 2)

4.2 abstract class on the sorting pattern

We found that Husky sort and MSD radix sort are under the same pattern. When sorting objects or UTF-8 as Chinese Characters which are not comparable, the general way is to take them as values and transform them into keys using a way of encoding (into long in HuskySort) or UTF8 encoding (in MSD radix sort). As the key is comparable, every time compare the key, the key and value are both moved and original arrays get sorted. By this pattern, other UTF8 characters can be sorted as well. The sort pattern can be seen as an abstract class, though further work is needed for implementation.

Reference

[1] P. Hildebrandt and H. Isbitz, Radix exchange – an internal sorting method for digital computers, Journal of the ACM 6(2) (1959) 156–63.
[2] R C HILLYARD, Yunlu Liaozheng, and Sai Vineeth K R. 2020. Huskysort. ACM Trans. Graph. 37, 4, Article 111 (August 2020), 9 pages. https://doi.org/10.1145/1122445.1122456
[3] Kovi S K. Comparative Analysis of Bucket and Radix Sorting With Their Applications and Advantages[J].
[4] Gill S K, Singh V P, Sharma P, et al. A comparative study of various sorting algorithms[J]. International Journal of Advanced Studies of Scientific Research, 2019, 4(1).