Customizable Dictionary

ABSTRACT

The Customizable Dictionary is a dictionary that allows the full customization by a user. The full customization allows the user to define certain jargons or document new words on-the-go; it also opens the possibility of rapid lingual documentation during tribal (or similar) immersion. The program is written in Python 3 (guaranteed compatible 3.5 and above), thus it can be easily employed on a number of operating systems, including mobile; the program GUI is built on Python Tk, thus the guarantee of minimal computing requirements.

The Customizable Dictionary features its own data engine, the Quick-Insert-Export (QIE) Customizable Dictionary Data Engine. The engine offers a handful of features, primarily being based on Quick Sort, its main feature is a powerful and high-speed data searching and sorting. Its exporting power also allows the portability of dictionary data, allowing the use of dictionary data across numerous platforms, as long as the platform has the program in the system.

The Customizable Dictionary is designed to process customized dictionary data, exuding both power and speed using a custom data engine. Being built on Python, it is highly portable and able to run numerous platforms; the dictionary data can be used in any of these platforms. The dictionary opens numerous possibilities ranging from personal to professional to research use, as long as the matter is related to linguistics or languages.

INTRODUCTION

The Quick Sort algorithm is a successor to Insertion Sort, in terms of speed and data efficiency, designed for large-scale data [1]. Quick Sort, being an in-place sorting algorithm, it requires no additional memory space, making it more efficiency in data handling than similar sorting algorithms that require additional memory space. Moreover, the divide-and-conquer design also allows multithreading, therefore the possibility of having lower run time than other sorting algorithms significantly [2]–[4]. However, these same strengths also make Quick Sort, or other large-scale sorting algorithm rare in massive consumption, as there is rarely minimal data to be sorted by the mass consumer population.

The idea of the Customizable Dictionary derives from Quick Sort's original purpose, a dictionary sorting algorithm [5]. In lieu to this, the idea itself is self-explanatory, a fully-customizable dictionary by a user. It will, therefore, take advantage of Quick Sort at its core, considering the possible scale of data to be stored.

The Customizable Dictionary will be designed to efficiently insert-sort-search through the large-scale data in the dictionary, employing primarily a Quick Sort sorting algorithm and modified versions of Binary Search searching algorithm and Insertion Sort insertion-sorting algorithm. The algorithmic modification is made to fit the context of sorted data space beforehand by Quick Sort. In this matter, the are designed to have to have Quick Sort at its center.

The "Binary Chop" algorithm, known as the Binary Search, is a searching algorithm designed, traditionally, for sorted data spaces [6], [7]. The centroids of data is the point-of-comparison of the algorithm, therefore, having an almost similar logic to a divide-and-conquer algorithm. This makes it most suitable for this use, given the context of pre-sorted data, to minimize search times, in any data context, in comparison to linear sort [6], [8].

The Insertion Sort algorithm may seem to be an overkill or overuse of sorting power. However, in this case, it will be not necessarily used as a sorting utility, instead, it is an entry-insertion utility [1], [9]–[11]. It would be modified, in sense that it is in idea still Insertion Sort but having more of Quick Sort's algorithm logic. This method would reduce sorted insertion run times, instead of insert-then-sort methods, taking more advantage of the Quick Sort's speed while also reducing its instability [1], [9], [11]–[14].

In note, the Customizable Dictionary's intended consumer is the general public, opening the public to the sorting power brought by Quicksort. It will be designed to be more of a general public use instead of only serving a specific demographic. However, its consumer base can be limited to individuals who will be encountering jargons and nouveau words on a common basis and the linguistics fields.

REFERRENCES

[1] C. A. R. Hoare, "Algorithm 64: Quicksort," *Commun. ACM*, vol. 4, no. 7, p. 321, 1961, doi: 10.1145/366622.366644.

[2] P. Sanders and T. Hansch, "Efficient massively parallel quicksort," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 1253, pp. 13–24, 1997, doi: 10.1007/3-540-63138-0_2.

[3] R. S. Francis and L. J. H. Pannan, "A parallel partition for enhanced parallel QuickSort," *Parallel Comput.*, vol. 18, no. 5, pp. 543–550, 1992, doi: 10.1016/0167-8191(92)90089-P.

[4] P. Tsigas and Y. Zhang, "A simple, fast parallel implementation of Quicksort and its performance evaluation on SUN Enterprise 10000," *Proc. - 11th Euromicro Conf. Parallel, Distrib. Network-Based Process. Euro-PDP 2003*, pp. 372–381, 2003, doi: 10.1109/EMPDP.2003.1183613.

[5] S. U.-S. of E.-C. S. Department, "Tony Hoare >> Contributions >> Quicksort." https://cs.stanford.edu/people/eroberts/co

urses/soco/projects/2008-09/tony-hoare/quicksort.html (accessed Oct. 30, 2020).

[6] A. B. Almostafa, “Maximum-speed search Algorithm,” *2018 6th Int. Conf. Control Eng. Inf. Technol. CEIT 2018*, no. October, pp. 1–6, 2018, doi: 10.1109/CEIT.2018.8751856.

[7] A. Hatamlou, “In search of optimal centroids on data clustering using a binary search algorithm,” *Pattern Recognit. Lett.*, vol. 33, no. 13, pp. 1756–1760, 2012, doi: 10.1016/j.patrec.2012.06.008.

[8] R. Nowak, “Generalized binary search,” *46th Annu. Allert. Conf. Commun. Control. Comput.*, pp. 568–574, 2008, doi: 10.1109/ALLERTON.2008.4797609.

[9] T. SinghSodhi, S. Kaur, and S. Kaur, “Enhanced Insertion Sort Algorithm,” *Int. J. Comput. Appl.*, vol. 64, no. 21, pp. 35–39, 2013, doi: 10.5120/10761-5724.

[10] M. A. Bender, M. Farach-Colton, and M. A. Mosteiro, “Theory of Computing Insertion Sort Is O (n log n) ∗,” vol. 397, pp. 391–397, 2006.

[11] B. C. Dean, “A simple expected running time analysis for randomized ‘divide and conquer’ algorithms,” *Discret. Appl. Math.*, vol. 154, no. 1, pp. 1–5, 2006, doi: 10.1016/j.dam.2005.07.005.

[12] D. Cederman and P. Tsigas, “GPU-Quicksort,” *ACM J. Exp. Algorithmics*, vol. 14, no. 1, 2009, doi: 10.1145/1498698.1564500.

[13] J. L. Bentley and R. Sedgewick, “Fast algorithms for sorting and searching strings,” *Proc. Annu. ACM-SIAM Symp. Discret. Algorithms*, pp. 360–369, 1997.

[14] M. H. van Emden, “Algorithms 402: Increasing the efficiency of quicksort,” *Commun. ACM*, vol. 13, no. 11, pp. 693–694, 1970, doi: 10.1145/362790.362803.