Splaysort

In computer science, **splaysort** is an adaptive comparison sorting algorithm based on the splay tree data structure.[1]

Contents

Algorithm

The steps of the algorithm are:

1. Initialize an empty splay tree
2. For each data item in the input order, insert it into the splay tree
3. Traverse the splay tree in inorder to find the sorted order of the data

Thus, the algorithm may be seen as a form of insertion sort or tree sort, using a splay tree to speed up each insertion.

Analysis

Based on the amortized analysis of splay trees, the worst case running time of splaysort, on an input with n data items, is $O(n \log n)$, matching the time bounds for efficient non-adaptive algorithms such as quicksort, heap sort, and merge sort.

For an input sequence in which most items are placed close to their predecessor in the sorted order, or are out of order with only a small number of other items, splaysort can be faster than $O(n \log n)$, showing that it is an adaptive sort. To quantify this, let d_x be the number of positions in the input that separate x from its predecessor, and let i_x be the number of items that appear on one side of x in the input and on the other side of x in the output (the number of inversions that involve x). Then it follows from the dynamic finger theorem for splay trees that the total time for splaysort is bounded by

$$\sum_x \log d_x$$

and by

$$\sum_x \log i_x .$$[2]

Splaysort can also be shown to be adaptive to the entropy of the input sequence.[3]

Experimental results

In experiments by Moffat, Eddy & Petersson (1996), splaysort was slower than quicksort on tables of random numbers by a factor of 1.5 to 2, and slower than mergesort by smaller factors. For data consisting of larger records, again in a random order, the additional amount of data movement performed by quicksort significantly slowed it down compared to pointer-based algorithms, and the times for splaysort and mergesort were very close to each other. However, for nearly presorted input sequences (measured in terms of the number of contiguous monotone subsequences in the data, the number of inversions, the number of items that must be removed to make a sorted subsequence, or the number of non-contiguous monotone subsequences into which the input can be partitioned) splaysort became significantly more efficient than the other algorithms.[1]

Elmasry & Hammad (2005) compared splaysort to several other algorithms that are adaptive to the total number of inversions in the input, as well as to quicksort. They found that, on the inputs that had few enough inversions to make an adaptive algorithm faster than quicksort, splaysort was the fastest algorithm.[4]

Variations

Saikkonen & Soisalon-Soininen (2012) modify splaysort to be more strongly adaptive to the number of contiguous monotone subsequences in the input, and report on experiments showing that the resulting algorithm is faster on inputs that are nearly presorted according to this measure.[5]

References

1. Moffat, Alistair; Eddy, Gary; Petersson, Ola (July 1996), "Splaysort: Fast, Versatile, Practical", *Software Practice and Experience*, **26** (7): 781–797, doi:10.1002/(SICI)1097-024X(199607)26:7<781::AID-SPE35>3.3.CO;2-2 (https://doi.org/10.1002%2F%28SICI%291097-024X%28199607%2926%3A7%3C781%3A%3AAID-SPE35%3E3.3.CO%3B2-2)
2. Cole, Richard (2000), "On the dynamic finger conjecture for splay trees. II. The proof", *SIAM Journal on Computing*, **30** (1): 44–85, CiteSeerX 10.1.1.36.2713 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.2713), doi:10.1137/S009753979732699X (https://doi.org/10.1137%2FS009753979732699X), MR 1762706 (https://www.ams.org/mathscinet-getitem?mr=1762706).
3. Gagie, Travis (2005), *Sorting a low-entropy sequence*, arXiv:cs/0506027 (https://arxiv.org/abs/cs/0506027), Bibcode:2005cs.....6027G (https://ui.adsabs.harvard.edu/abs/2005cs.....6027G).
4. Elmasry, Amr; Hammad, Abdelrahman (2005), "An empirical study for inversions-sensitive sorting algorithms", *Experimental and Efficient Algorithms: 4th International Workshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*, Lecture Notes in Computer Science, **3503**, Springer, pp. 597–601, doi:10.1007/11427186_52 (https://doi.org/10.1007%2F11427186_52).
5. Saikkonen, Riku; Soisalon-Soininen, Eljas (2012), "A general method for improving insertion-based adaptive sorting", *Algorithms and Computation: 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012, Proceedings*, Lecture Notes in Computer Science, **7676**, Springer, pp. 217–226, doi:10.1007/978-3-642-35261-4_25 (https://doi.org/10.1007%2F978-3-642-35261-4_25).