ECS60

Xie Zhou

id: 912143385

Timetest Write-up

Average Run time (of 3 runs) for each ADT for each file

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | File1 | File2 | File3 | File4 |
| Linked List | 0.0574 | 82.1206 | 0.0414 | 53.2290 |
| Cursor List | 0.0429 | 375.9190 | 0.0546 | 208.2217 |
| Stack Array | 0.0356 | 0.0336 | 0.0336 | 0.0363 |
| Stack List | 0.0507 | 0.0405 | 0.0398 | 0.0432 |
| Queue Array | 0.0374 | 0.0365 | 0.0363 | 0.0388 |
| Skip List | 0.1681 | 0.1235 | 0.1413 | 0.3736 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Linked List | File1 | File2 | File3 | File4 |
| Individual insertion | O(1) | O(1) | O(1) | O(1) |
| Individual deletion |  | O(N) | O(1) | O(N) |
| Entire series of insertions | O(N) | O(N) | O(N) | O(N) |
| Entire series of deletions |  | O(N^2) | O(N) | O(N^2) |
| Entire file | 0(N) | O(N^2) | O(N) | O(N^2) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cursor List | File1 | File2 | File3 | File4 |
| Individual insertion | O(1) | O(1) | O(1) | O(1) |
| Individual deletion |  | O(N) | O(1) | O(N) |
| Entire series of insertions | O(N) | O(N) | O(N) | O(N) |
| Entire series of deletions |  | O(N^2) | O(N) | O(N^2) |
| Entire file | 0(N) | O(N^2) | O(N) | O(N^2) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stack Array | File1 | File2 | File3 | File4 |
| Individual insertion | O(1) | O(1) | O(1) | O(1) |
| Individual deletion |  | O(1) | O(1) | O(1) |
| Entire series of insertions | O(N) | O(N) | O(N) | O(N) |
| Entire series of deletions |  | O(N) | O(N) | O(N) |
| Entire file | O(N) | O(N) | O(N) | O(N) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stack List | File1 | File2 | File3 | File4 |
| Individual insertion | O(1) | O(1) | O(1) | O(1) |
| Individual deletion |  | O(1) | O(1) | O(1) |
| Entire series of insertions | O(N) | O(N) | O(N) | O(N) |
| Entire series of deletions |  | O(N) | O(N) | O(N) |
| Entire file | O(N) | O(N) | O(N) | O(N) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Queue Array | File1 | File2 | File3 | File4 |
| Individual insertion | O(1) | O(1) | O(1) | O(1) |
| Individual deletion |  | O(1) | O(1) | O(1) |
| Entire series of insertions | O(N) | O(N) | O(N) | O(N) |
| Entire series of deletions |  | O(N) | O(N) | O(N) |
| Entire file | O(N) | O(N) | O(N) | O(N) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Skip List | File1 | File2 | File3 | File4 |
| Individual insertion | O(logN) | O(logN) | O(logN) | O(logN) |
| Individual deletion |  | O(logN) | O(logN) | O(logN) |
| Entire series of insertions | O(NlogN) | O(NlogN) | O(NlogN) | O(NlogN) |
| Entire series of deletions |  | O(NlogN) | O(NlogN) | O(NlogN) |
| Entire file | O(NlogN) | O(NlogN) | O(NlogN) | O(NlogN) |

For the linked list, insertion requires very short time because every piece of data is inserted to the front of the linked list. Therefore, the complexity for each insertion is O(1). The difference in running time of the four files is due to the deletion process. File 1 and file 3 are obviously processed faster. Because there is no deletion in file 1, the total time complexity is O(N). For file 3, the insertion time is the same as file 1’s. But the processing time is very short because the deletion starts from the head of the list(the deletion starts from 250000 to 1 and data is ordered in this order already), which makes the time for each deletion O(1). Therefore, the total complexity is also O(N). File 2 and file 4 takes the longest running times (apprx. 82s, 53s). This situation is caused by how the linked list searches for data in itself (during deletion). File 2 creates the worst case. Because the deletion is from 1 to 250000, which is in a reversed order of the inserted data, the processor has to go to the end of the list to find the wanted piece of data and delete it. This worst case makes the time complexity of each deletion O(N) (the complexity of the entire deletion becomes O(N^2)). File 4 also has a long running time because its time complexity is O(N^2). Although not every individual deletion goes to the end of the list because the deletion is random, the big O of the deletion process is still O(N^2), making the total complexity O(N^2).

The Cursor list acts similarly to the linked list. With file1, which only contains insertion, the running time is very fast. With file3, which the deletions go exactly in order, the time is short as well. For the operations in file2 and file4, similar to the linked list, file 2 brings the worst case (goes to the end of the list every time) and file 4 also had the complexity of the worst case since it’s random. One noticeable thing that happened in both the linked list and the cursor list operations is that the deletions in file 2 take longer than those in file 4 (even though they have the same complexity). This is because in reality, the random deletion does not go to the end of the list for searching every time and that saves the running time.

The stack array, the stack list, and the queue array have similarly short running time when processing the data in all the files. This is because for stacks and queues, the insertions are simply adding data to the top of the data structure. Deletions are also simple-just taking one piece of data away from the top/bottom of the data structure. Because there is no “search-for-a-value” in stacks and queues but only popping and pushing, the running time is simply O(1\*N).

The skip list data structure is also very fast. It has an especially outstanding performance in processing file 2 and file 4. Different from other data structures, the skip list requires searching not only in deletions but also in insertions. Its binary insertion, which sorts the stored data, largely raised the efficiency in the later binary search for deletion. Thus while the insertions take a little longer time, time is saved for deletions (the most time consuming part).

At the first glance, the stacks and queues takes the shortest time when processing all the files. However, we have to take into account the limitation of these data structures. We cannot search through the data structure, which means data processing would become extremely difficult in most situations. Then it is the cursor list and linked lists. One interesting fact is that both of them have the same time complexity but different running time. The cursor list runs much slower than the linked list. The reason is because during every insertion or deletion, the cursor is moved. For both the cursor list and the linked list, the short running time for file1 and file 3 indicates that these two data structures works extremely well when insertion is the primary (or only) task and deletion time is negligible. However, this situation is very rare since you cannot expect the values to be in the exact order that minimize the deletion time, in the case of file 3: O(1). In most cases, the total deletion time (searching time) complexity is O(N). Therefore, clearly, searching through the whole list is very time consuming. On the other hand, the skip list performs better. Although the skip list requires an additional searching time when inserting (binary), because the list is sorted, the search for a value to be deleted (binary) is much faster than that of the cursor list and the linked list. It is also obvious in the running time table that the skip list has a good performance processing all 4 kinds of data while the lists only do well in processing 2 of the 4 files. Therefore, depending on the different types of data, linked lists and cursor lists (or stacks and queues) might be suitable for some, but overall, skip lists are better in dealing with all kinds of data.