COMP20003

Assignment 1: Dictionary BST
Student 826591

Introduction

A dictionary was implemented with a binary search to store athletes' information contained in the file specified in command line argument. Each record was stored in a separate node. The data structure enables query for athlete records, based on the keys provided from stdin.

In stage 1, duplicate keys are placed in the left side of the parent match. The program dict1 then continues execution to find further matches until a leaf node is found. Whereas, in stage 2 (dict2), the tree was constructed similarly except that duplicate keys are stored in a linked list. This enables duplicate keys to be found without further key comparisons.

The input files primarily used were derived from *athlete_events_filtered.csv,* to create *athlete_events_filtered_sorted.csv*, which was sorted in ascending order with regards to the athlete name, and *athlete_events_filtered_random.csv.*

Search keys were generated using shuf -n and cat *fileName.csv* | cut -d ',' -f2 command and were input to stdin during dict1 and dict2 execution with the randomised and sorted input files in the command line. 60 keys were generated in total for a file with 271,116 entries in the interest of time. Ideally, at least 100,000 keys would be searched.

The data structures implemented have the following worst case complexity analysis, a Binary Search Tree is O(n) in terms of searching and insertion but for a perfectly balanced tree it is O(logn). An unsorted linked list is O(1) in insertion and O(n) in searching.

Stages

Stage	Average key comparisons	input type
Stage1/ dict1	49.13	Random sorted
Stage1/ dict1	50,493.78	Sorted (ascending)
Stage 2/ dict2	30.43	Random sorted
Stage 2/ dict2	50,488.92	Sorted (ascending)

Table 1. Average key comparisons of stages 1 and 2 depending on input type

Comparison of the stages

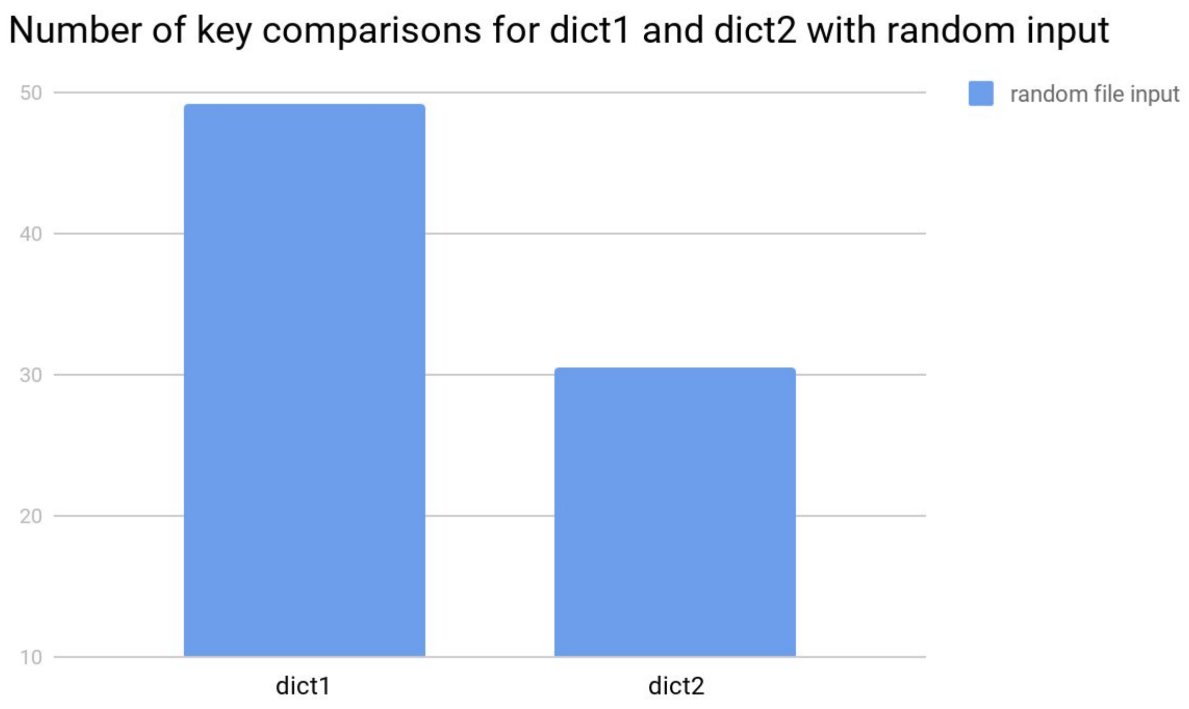

Figure 1. Number of key comparisons for dict1 and dicts with random sorted input

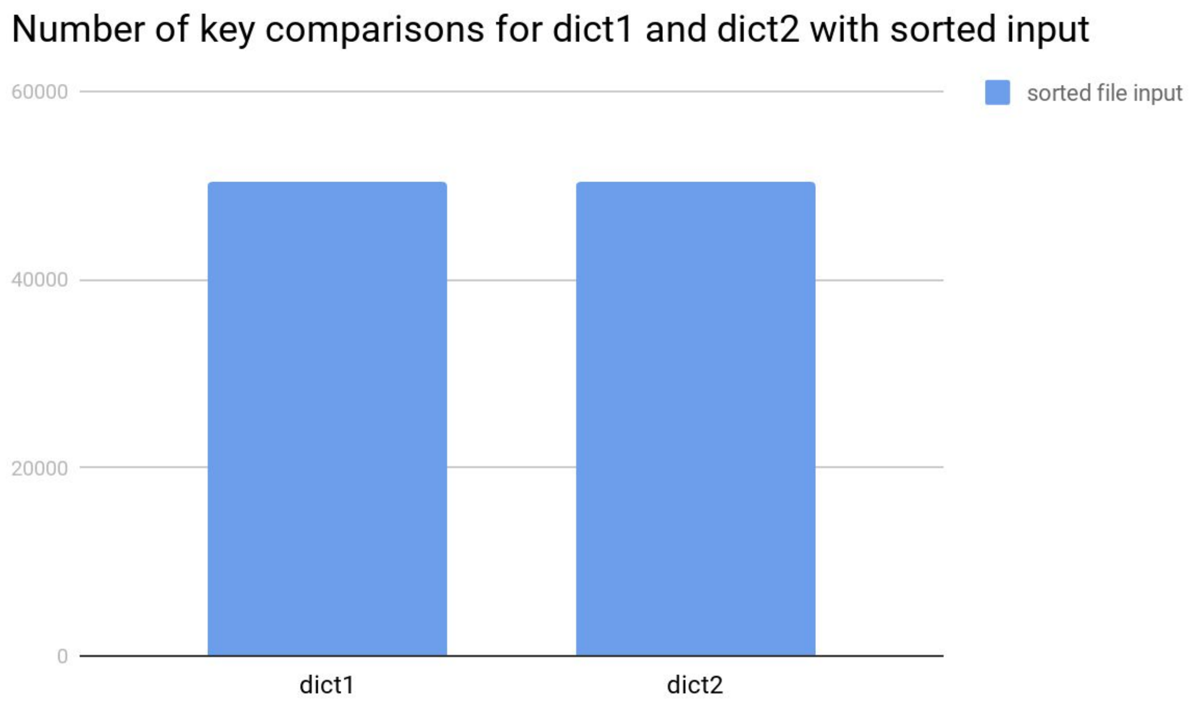

Figure 2. Number of key comparisons for dict1 and dict 2 with a sorted (ascending) input

Comparison with theory

Since the BST implemented is not self-balancing we can expect O(log n)>average number of key comparisons<O(n) closer to log n for a random sorted input file. This is not supported by the results (**Table 1** & **Figure 1**) although the average case of **Table 1** (dict 2 with random input) is just less than half more of the best case scenario. Whereas, for a sorted input file we can expect an average number of key comparisons (on the average case), n/2, or in the worst case, the key comparison average would equal to O(n) depending on the alphabetic value of the key. According to **Figure 2**, the test case is closer to average key comparison = n/5.

Discussion

Randomly sorted input

The results of the key comparisons for stage 1 with a randomly sorted input file were expected according to theory. The average of key comparisons for this stage with a randomly sorted input was ~49 (refer to **Table 1**), more than half of the best case log n, which is in this case should be ~18. As for stage 2, the implementation of a linked list in dict 2 reduced the average of key comparisons to ~30 (refer to **Table 1**), closer to the best case scenario. The search tree is not perfectly balanced therefore having a larger value to log n is to be expected.

Sorted (ascending) input

The number of key comparison for dict1 with a sorted input was ~50494 (refer to **table 1**) which ultimately just depended on the value of the key string. The average key comparison reflects the range of the key strings chosen. In contrast, dict2 generated an output of ~50489 (refer to **Table 1**) average of key comparisons, which reduced the average key comparison significantly less, compared to stage 1 (refer to **Figure 1** & **Figure 2**). Stage 1 was reduced by ~19 whilst stage 2 was only reduced by ~5. The amount reduced in the average key comparison is dependent on the number of duplicates stored in a linked list for the searched keys as well as the binary search tree being reduced to a stick. Compared with complexity analysis theory, there should be an O(n) search for a non-balancing tree, we can get this result by searching the last entry in the binary search tree using dict1.`For the test conducted, 271,116 entries and only 60 search keys, the little amount of search keys comparative to the entry may be the cause of not attaining the n/2 average case search in a non-balancing binary search tree.