Experimentation Assignment 1 (UNIX*yelp*)

Name: Maleakhi Agung Wijaya
User Name:maleakhiw
Student Number: 784091

Introduction

This assignment focus on creating a simplified UNIX*yelp* dictionary that supports insertion and search for key and values. Yelp database will be implemented using binary search tree as it's concrete data structure. Binary search tree is a type of hierarchical data structure which every node consists of pointer to left and right node following particular rules. For stage 1 implementation, the left child of every nodes is <= and the right child is >. In addition, search process will continue until reaching a leaf node as there are duplication on keys that every information needs to be printed. As for stage 2, all of information about the same key will be stored in the linked list. Therefore with this implementation, after a key is located, we just need to traverse the linked list without continuing to search until leaf. Both stage 1 and stage 2 input and output file will be given through command line arguments. The input will be csv files from yelp which consists of large number of data. As for the output, the key that are searched and the information about the key will be printed to specified ouput file. Apart from search result, we will also keep track of the number of comparisons which will be printed to stdout.

Result

The keys that are used on several data input conditions listed on the tables are generated randomly using generate_key.c script which we can control the number of keys (5000) and specified input files which the keys are taken (yelp csv). The comparison for stage 1 and stage 2 has been calculated as average.

Input File Condition	Number of Input	Keys Taken	Stage 1 Comparison (AVG)	Stage 2 Comparison (AVG)
Random	10	Same file from input	4.67	3.67
Random	10	Different file from input	4.74	4.74
Sorted	10	Same file from input	6.67	6.67
Sorted	10	Different file from input	5.00	5.00
Random	100	Same file from input	7.00	5.50
Random	100	Different file from input	8.82	7.69
Sorted	100	Same file from input	42.83	42.00
Sorted	100	Different file from input	23.57	22.75

Input File Condition	Number of Input	Keys Taken	Stage 1 Comparison (AVG)	Stage 2 Comparison (AVG)
Random	1000	Same file from input	19.83	7.33
Random	1000	Different file from input	19.36	7.69
Sorted	1000	Same file from input	62.83	53.00
Sorted	1000	Different file from input	44.86	33.91
Random	77,445	Same file from input	38.56	15.25
Random	77,445	Different file from input	22.04	20.48
Sorted	77,445	Same file from input	28,636.56	28,617.71
Sorted	77,445	Different file from input	27,759.91	27,759.66
Random	552,275	Same file from input	953.43	11.97
Random	552,275	Different file from input	39.22	21.43
Sorted	552,275	Same file from input	25,070.41	24,158.35
Sorted	552,275	Different file from input	24,893.87	24,888.66
Random, Unique	539,767	Same file from input	908.93	11.95
Random, Unique	539,767	Different file from input	39.10	21.43
Sorted, Unique	539,767	Same file from input	20,991.73	20,063.88
Sorted, Unique	539,767	Different file from input	21,730.78	21,691.35
Random, Duplicated	309,780	Same file from input	97.29	15.25
Random, Duplicated	309,780	Different file from input	25.81	20.63

Unique = No duplication in keys and data

Duplicated = Same keys and data

The test data used from result above is taken from yelp, mockaroo, and randomly generated csv files

Discussion

Based on theory, we can analyse the best, average case and worst case of insertion and search to binary search tree. The best case for insertion in binary search tree is O(1) during the first insertion at the root of the tree. As for search, the best case is also O(1) as when the key that are searched is at the root. This theoretical best case for insertion applied to both stage 1 and stage 2 as both insertions takes place at the tree's root. However, for search it is only possible for stage 2 (yelp2) when we were using the first element of the tree as search key (Example: search Mr Hoagie from yelp_academic_dataset_business.csv print number of comparison = 1). This search best case will never happen in stage 1 as there may be duplication of keys which then we need to traverse until leaf.

For average case, both insertion and search in binary search tree is O(log n). This occurs due to the structure of binary search tree which left child is <= and right child >. From this rule, in complete binary search tree or almost complete tree, the number of comparison performs is proportional to the height of the tree which is why the average complexity is $\log_2$. From the table above, it can be seen that the search that are taken to unsorted/ random data input approximates $\log_2(n)$ for both yelp1 and yelp2. That being said, yelp2 is more efficient than yelp1 in terms of searching as when a particular key is found, it will just need to traverse the linked list, not traversing until the leaf. The table also supports this theory as the number of comparison taken in stage 2 is less than number of comparison in stage 1.

For worst case, both insertion and search is O(n). This condition occurs when the binary search tree is degenerated into linked list. From the experimentation, it can be seen when the input data is sorted. Eventhough stage 2 can be more efficient than stage 1 if we search string starting with number, but when we search string starting with especially last half of alphabets, the number of comparisons will increase dramatically. Searching with number is much more efficient for stage 2 when the data is sorted as number will be inserted first, and thus the search process does not neccessarily go deep. As for stage 1, for every search it will go through until the leafs which is really bad.

We can avoid this worst case by implementing rotation every time a node is inserted. Tree which implement rotation is commonly known as AVL Trees which gives worst case of O(log n) as the tree stay balanced and thus can be used as further improvement for the problem of O(n) worst case.