**Project Summary**

In project 7, I implemented a HashMap that can be used to store data. Hash map is a data structure that that maps keys to a specific location in the array using a hash function (the same key will generate the same hash index), therefore, it’s efficient to add and locate data in a hash map because we only need to search for the location indexed by the hash function.

The goal of this project is to analyze the reddit files with both BSTMap and HashMap and compare their performances in different operations.

**Task Solutions**

HashMap

The class Hashmap also implements MapSet and it has a field for an object array to hold the KeyValuePairs. In HashMap class, I implemented opening hashing, where a BST is inserted when there’s collisions in the hash index.

The int field num\_collision keeps track of the occasion where different keys are hashed to the same location of the table. num\_filledPosisions keeps tracks of the number of positions that are filled in the table, and the array is expanded when it’s half-filled by the private method ensureCap().

WordCounter

I first modified the constructor of WordCounter class, allowing it to take in a Boolean parameter useHashMap, which decides whether the map field will initialize to a HashMap (true) or a BSTMap(false).

Then I modified the main function so that it uses a nested for loop to analyze each file passed in through command line for 5 times and compute the average time taken by dropping the highest and the lowest and takes the average of the remaining 3 time-values. The main function in WordCounter uses a HashMap to analyze the data and the program prints out the number of collisions that took place in the hashing, total word count and unique word count.

Then I added a getDepth method in BSTmap, which returns the number of levels in the BST. The method recursively traverses over the tree and compares the depths of the left sub-tree and the right sub-tree of a node and returns the larger value.

I created a separate class WordCounterBST to repeat the process using a BSTMap and computes the average time taken to analyze each file. The spreadsheet below shows the result of analyzing the 8 reddit files using both BST and HashMap.

FORM

Result analysis

To compare the performance of BSTMap and HashMap, I made the follow three plots to study the relationship between different variables in the analysis of reddit files and the performance of the two data structures.

1. Year vs. run time:

The performances of Hashmap and BSTMap have similar trends across the eight years, while HashMap is about 10 sec faster than BSTMap in general

1. Total word count vs. run-time

There’s a positive relationship between total word count and processing time for both data structures, but HashMap is still about 10 secs faster. The plot graph for BSTMap should have a trend of a logN relationship between its x and y axis, according to the data structure of BST

1. Unique word count vs. run time

HashMap is also faster in general than BSTMap and its advantage becomes more apparent as the word count increases.

Based on the results, we can conclude that the HashMap has a better performance in analyzing/reading in files of either the same size or the same total word count. The efficiency of HashMap could be the result of its direct indexing, instead of traversing the whole set of data as in a BST.

For a Hash table, there’s a positive relationship between the number of collisions that took place in putting the words and the time taken to analyze, as the plotted graph show below.

For a BSTMap, the relationship between the depth of BST trees and the run-time is not significant as the graph show below.

**Extensions**

Close-hashing table (CloseHashingMap) and its performance

I implemented another collision-handling method by using a closed hashing table in CloseHashingMap.java. The class also uses an object array to hold the data, but instead of inserting/expanding BSTs at each location in case of collisions, the class uses linear probing to find the next available space in the array and insert the data. The constructor takes in a comparator and an int s, which decides the step length of the linear probing. The hash function is the same as the one in the open HashMap.

The class also has a private method expandCap() that expands the array when it’s half filled. The method is called whenever a new element is put into the hash table.

Linear probing is used in the method put(), containsKey() and get(). For example, when a new KVP is put into the table, the put method will first generate an index using the hash function. Then I used a while loop to check if the indexed position in the array is filled. I used a comparator to compare the new\_key and the keys of the KVP at the indexed location, and if the new\_key is the same as the key at a location, the value associated with the key will be updated and the function will return the oldValue.

PIC

Otherwise, the while loop will keep probing in the table by the step length until it finds an empty space. If the index reaches the end of the array in the process of probing, it will wrap to the beginning of the table by taking a modulo of the current index over the length of the table.

PIC

To make sure that the number of collision is only incremented once if the first indexed space is filled, I created an int variable before the while loop to save the number of collisions before the probing, so that it doesn’t get incremented every time the while loop is repeated.

PIC

The method get and containsKey follow the same algorithm. I then tested the closed hashing map in the class CloseHashingTester, which has the same functions as the WordCounter class. it has a field of the CloseHashingMap object and I made the class to also read in the 8 reddit files each for 5 times and compared its performance with the open Hashmap.

It shows that there isn’t a significant difference in the run time between the closed and open hashmap, although the closed hashmap produces more collisions in general.

Analyzing efficiency of hash function in data distribution

As shown in the result of the base project, the number of collisions that took place during hashing does affect the efficiency of the Hashmap. Therefore, it’s important to have a good hash function that distributes the data evenly among the hash table. The second extension I did was to examine the efficiency of the hash function used in HashMap in distributing data among the array.

I first added a printTable method in HashMap which prints out the number of elements at each position of the array, which is also the size of the BST at the position. I tested it on the reddit file of 2008, and below is the number of elements stored in each (the first 100) positions of the final hash table array.

Credits:

Prof. Maxwell & his lecture notes

Prof. Layton

TA Tia Zhang

Roujia Zhong