Andrew Walters

CSC 2053

Cuckoo Hashing

The Cuckoo Separate Chaining Hash Symbol Table and Cuckoo Probing Hash Symbol Table were both implemented successfully with no known limitations.

Implementation:

Hashing Functions:

In Separate Chaining, I used a simple modulus and an offset modulus, ensuring that the hash functions would never be equal. Due to concerns of cycles in the probing implementation, the hashcode of each key was multiplied by a prime number prior to the modulus operation to ensure that cycles were less frequent. This required hash2 to implement a check and offset to prevent duplicate hash codes.

Separate Chaining:

Separate Chaining was implemented using an array of linked lists (SequentialSearchST). In the put function, the list with the smaller size was used, or a random choice was made if they were equal. Once the average list size surpassed 10, the table size was doubled.

Probing:

Probing was implemented with an array of generic objects cast as a key-value object called Item. The primary hurdle with probing was cycle prevention. I depth counter was used in recursive calls of put that limited the depth to 10 calls. Once 10 items had been replaced the table was doubled and rehashed.

Running Time Analysis:

For each input file, the Youtube Comedy Slam client was run 5 times and the time in the table represents the average of those trials. The Comedy Slam client reads from an input file and builds the data structures simultaneously, so the time to read is included in the timing for every structure. The input file is “comedy_comparisons.train” using the first 256K and 512K entries.

Data Structure	**Running Time for 256K (s)**	**Running Time for 512K (s)**
CuckooSeparateChainingHashST	1.259	2.377
CuckooProbingHashST	0.892	1.423
ST	1.097	1.712
RedBlackBST	0.987	1.573
SeparateChainingHashST	0.885	1.369
LinearProbingHashST	0.815	1.290

The hash tables clearly outperformed the tree data structures with the lone exception of my implementation of CuckooSeparateChainingHashST. This could be the result of some inefficient functions, such as size() which sums the size of each linked list instead of tracking the size in a variable. All of the data structures scale at approximately the same rate. The Cuckoo implementations were both slower than their non-Cuckoo equivalents, which could be a result of the implementation rather than the efficiency of the data structure.