Artiom Arutiunov

ID 504597668

Professor Smallberg

6/2/16

Project 4 Report

I used a hash table in my create delta algorithm. I decided to use linked lists to store any collisions that occurred when storing strings within my hash table. Instead of creating buckets that contained a pointer to a linked list, I instead just made the first element of the linked list to be the bucket. Thus the way I checked whether a bucket was “empty” or not was by whether the linked list’s size was 0. If it wasn’t 0, then I knew that the bucket (and the rest of the list) was empty. Each linked list node contained a struct I created called segment. Within each segment I had two variables: An integer for the offset value of the string from the old file and a string for the actual segment from the old file. I actually created a class for my hash table to have everything in one place. Within the class I had a one-parameter constructor of type in that I used to determine how many buckets the hash table should have. I also had the void functions insert (to insert a new segment into the hash table) and print (for debugging purposes). I also had a function of type int called hashFunc that takes the string segment to be inserted into the hash table and the number of buckets in the hash table. This is what I used to insert each segment into my hash table as I traversed through the old file’s string. Lastly, I created a constant integer called BUCKETMULTIPLIER with a value of 1.1 that gave me slightly more buckets in my hash table than the amount of characters in the hash table. The result: My hash table works pretty efficiently and quickly. In order to optimize what string length segment (referred to in the spec as N) to use to create the smallest delta file, I ran 25 iterations of different N values – 2 through 50 increasing by 2 with every iteration – to find the best N size value to use for the smallest delta file.