Justin Marion

Prof Long

CSE 13S

May 23, 2021

DESIGN.pdf

Bloom filters:

bf_delete():

If bv exists and bm exists:

bv_delete()

free(bf)

Bf = NULL

bf_insert():

index_0 = hash(insert_val)

index_1 = hash(insert_val)

index_2 = hash(insert_val)

Set index 0, 1, 2

bf_probe():

index_0 = hash(probe_val)

index_1 = hash(probe_val)

index_2 = hash(probe_val)

Bit vector:

This implementation will be the same implementation as the one used in previous assignments like asgn5

Hash Tables:

delete():

free(list)

Pointer to list -> NULL

free(table)

Pointer to table -> NULL

ht_lookup():

// Simple wrapper around ll_lookup

If ll exists

ll_loopup()

Else

Return NULL

ht_insert()

// Simple wrapper

If ll exists

ll_insert()

Else

Return NULL

LinkedList

ll_lookup()

Traverse through ll

If word is the same then save the pointer and break

If mtf is true

Move node to front

ll_insert()

If (it is not in lookup)

Insert node at the front

My banhammer.c will first parse all arguments and use helper functions to print the -h flag etc. I will use enum and an array to keep track of my arguments throughout the program.

In order to keep track of the nodes that contain the badspeak I will use my ll ADT as I am not sure of the length of the list.

The regex will look like:

[a-zA-Z0-9_]+(('|-)[a-zA-Z0-9_]+)*

The first part of the program matches all words and numbers and _ in a word, then a words can optionally have a - or ‘ connected to it and therefore we match any other word after that, the ()* lets us do this optionally.