

Assignment Purpose: To Build a Bloom filter which processes messages and returns a modified message that conforms to the specifications set. Then, to aid the Bloom filter in cases where it's output are inconclusive, build a hash table of linked lists to search through words which get hashed to the same index value to get a definitive answer for set membership.

Bloom filters test whether an element is in a set:

Return either:

possibly in the set (false positives can occur)

Error rate factor can be specified and in general, the larger the Bloom filter, the lower the false positive rate

definitely not in the set (false negatives cannot occur)

Lexical Analysis:

Words spoken by the citizens of GPRSC can include characters:

a-z, A-Z, 0-9, underscore, ', -

reg expressions for different word types :

single word/character : $[a-zA-Z]^+$

word with spaces and numbers : $[a-zA-Z0-9_]^+$

contraction : word + "'" + word

$([a-zA-Z]^+)'([a-zA-Z]^+)$

hyphenated word : word + "-" + word

$([a-zA-Z]^+)-([a-zA-Z]^+)$

For the regular expression to catch all words, link the word, word with apostrophe, and hyphenated words together with or expressions.

Note: Use the c library function `tolower` to convert every character of every word to lowercase if it is not so already.

Pseudocode :

```
word = file.next
for (i, i < strlen word, i++)
    word[i] = tolower(word[i])
```

The Bloom Filter ADT:

Bloom filter stores three salts, each 128 bits in size. **Due to bitwise storage of salts, the bit vector ADT will also be essential.

Bloom filter functions:

bf_create - constructor for new filter

bf_delete - destructor for filter

bf_size - return number of bits the filter can access

equal to the length of the bloom's bit vector

bf_insert - hashes an input word with the three salts, then sets the bits in the bloom filter's bit vector to reflect the inserted word

bf_probe - hashes an input word with the three salts, if the same vector bits are already set, the word was *most likely* added to the Bloom filter

bf_count - returns number of set bits in the bf

bf_print - prints out the filter



length = 16

salt 1: hash (Hello) = 3
salt 2: hash (Hello) = 4
salt 3: hash (Hello) = 5

- set bits -

salt 1: hash (Bye) = 7
salt 2: hash (Bye) = 8
salt 3: hash (Bye) = 9

salt 1: hash (Car) = 3
salt 2: hash (Car) = 4
salt 3: hash (Car) = 5

Despite not having been added,
filter returns True on Car.

↳ Returns true since all 3 bv values are 1.

Note, this illustrates the advantage of three salts, to trigger the above false positive, all three of the hashed values must coincidentally fall into indexes that are already set to one, not just only one.

To correct for these remaining values, a hash table containing linked lists is used.

The HashTable ADT:

The hash table stores values to keys. Using a key, finding the value is possible in constant time. To generate a hash value from a key, K, a hashing function $H(x)$ must be used where K is inserted as the input x.

HashTable functions:

ht_create - constructor for a hash table, sets the salt (hashing function) and initializes the table's linked list.

ht_delete - destructor for hash table

ht_size - return table size = the number of indices = the number of linked lists

ht_lookup

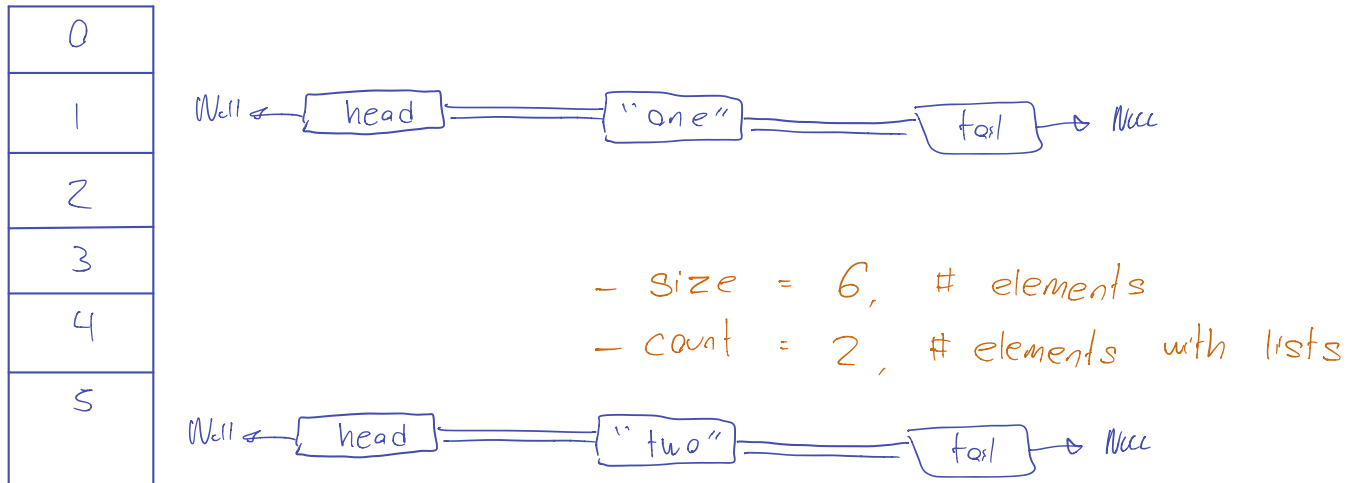
hash an input word *oldspeak*

the result of this hashing is the index of the linked list to perform lookup on

if a corresponding **node** is found return true, else false
ht_insert - inserts a word *oldspeak* and its translation *newspeak*
 index of the linked list to insert is the hashing of oldspeak
 if no linked list at index, create one, then add to it

ht_count - returns number of non NULL linked lists
ht_print - print out contents of hash table

Hash Table: Linked Lists



The Linked List ADT:

This will be a doubly linked list (each node has a pointer to a previous and a next node).

Each node contains an oldspeak word and, if there is one, its newspeak translation.
 Oldspeak without a newspeak translation is known as *badspeak*

Upon construction, a linked list begins with 2 sentinel nodes.

Linked list external variables:

Seeks - tracks number of lookups performed
 Links - tracks number of links traversed

Linked list functions:

ll_create

if Boolean mtf is true, found nodes are moved to front

ll_delete - destructor for the linked list, delete every node with node delete

ll_length - return length of linked list, the 2 sentinel nodes are not counted as a part of this

length

ll_lookup

if a linked list contains the node with asset oldspeak, return the node, else return NULL
 for every link in list

if link is not head or tail

if link.oldspeak == oldspeak, return node at link

note: *global variable seeks is added to with every function call, links is added to with every loop of this function.*

Additionally if MTF is specified, move the found node to the front of the LL.

ll_insert

Check if a duplicate node already exists, if so break
 else calls to create a new node, inserts the node in the LL
 Always insert new nodes at the *head*, right after the first sentinel node

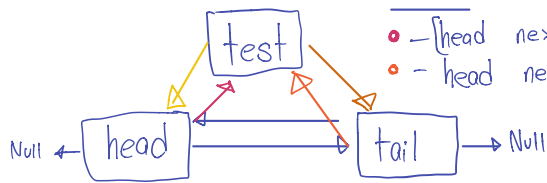
`ll_insert :`

`ll_lookup first` - return if node exists

Empty Linked List:



`insert (test)`

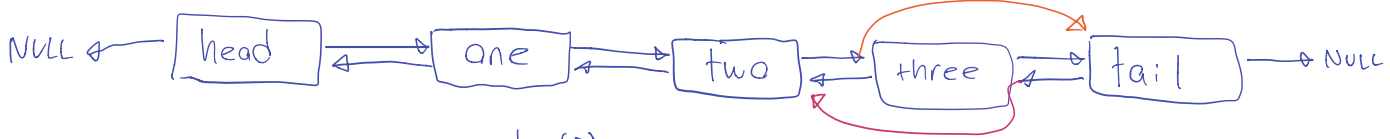


- - inserted prev \rightarrow head, always
- - inserted next \rightarrow head \rightarrow next, always
- - [head next] \rightarrow tail now becomes inserted
- - head next now becomes inserted

index the list length

`Move to front:` Search for '3', then move it to front

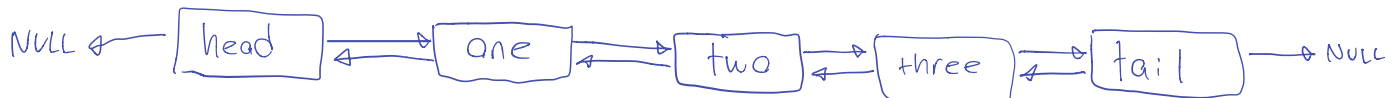
- `lookup(3)` - using std lookup
- remove (3)*
- insert (3) - using std insert



*remove a node (3)

- three \rightarrow prev \rightarrow three \rightarrow next
- three \rightarrow next \rightarrow three prev
- ↳ now nothing points to three node - delete (three)

`ll_delete:` - as used in MTF



To allow for for loop to work, delete previous, not current

Using the hash function:

The hash function returns the index of the bloom filter and hash table that are to be written to. However, this return value, by the nature of the hash function is a 10 digit number which far exceeds the reasonable size for either the bloom filter or hash table.

Indexes to HashTable and bloom filter are actually given by:

$$\text{index} = \text{Hash}(\langle \text{hashing salt} \rangle, \text{Oldspeak}) \% \text{HashTable} \mid \text{Bf} \rightarrow \text{vector} \cdot \text{length}$$