Assignment 7 - The Great Firewall of Santa Cruz

Teresa Joseph

CSE 13S - Professor Long

Fall 2021 - RD November 28/ FD December 3

Purpose

The purpose of this assignment is to implement a method of going through text and finding "oldspeak" terms and replacing them with "newspeak" words if possible. It will make use of bloom filters, hash tables, and binary trees to do this, their functions being written out in bf.c, ht.c, and bst.c respectively. A file that implements nodes for the binary search tree will be in node.c and a file that creates the bit vector operations needed for the bloom filter will be in bv.c. The main test harness is banhammer.c, which will take on command line options and filter through any badspeak/oldspeak and newspeak to print corresponding messages.

Pseudocode

bv.c

```
bv_create
    allocate bit vector memory with malloc
    if bit vector is NULL after allocation:
        return NULL (means allocation failed)
    set bit vector length to length
    for i from 0 to length:
        set index i of vector array to 0
    return bit vector
bv_delete
    free bit vector pointer
    set pointer to NULL
bv_length
    return bit vector struct member length
bv_set_bit
    if i is greater than or equal to bit vector length:
        return false
        set byte index to i divided by 8
        set bit index to i modulo 8
        create a mask of 1 left shifted by the bit index
        OR vector array at byte index with mask
        return true
    bv_clr_bit
        if i is greater than or equal to bit vector length:
            return false
        set byte index to i divided by 8
        set bit index to i modulo 8
        create a mask of 1 left shifted by the bit index
        AND vector array at byte index with mask
        return true
    bv_get_bit
        if i is greater than or equal to bit vector length:
            return false
        set byte index to i divided by 8
        set bit index to i modulo 8
        create a mask of 1 left shifted by the bit index
        if AND of vector array at byte index with mask is 1:
            return true
        otherwise:
            return false
    bv_print
        for i from 0 to length:
            if vector array at i is 1, print 1
            if vector array at i is 0, print 0
```

node.c

```
    node_create
        allocate node memory with malloc
        set left and right of node to NULL
        call strdup() on oldspeak and it to to node’s oldspeak
        call strdup() on newspeak and it to to node’s newspeak
        return node
    node_delete
        free pointer to oldspeak and newspeak
        free node and set its pointer to NULL
    node_print
        if both oldspeak and newspeak are not NULL:
            print both as shown in assignment document
        if oldspeak is not NULL and newspeak is NULL:
            print oldspeak as shown in assignment document
```

bst.c

```
    bst_create
        create pointer of node type and set to NULL
        return pointer
    bst_height
        if root is not NULL:
            recursively find the leftmost of the root
            recursively find the rightmost of the root
            find the greatest value of the two, add 1 to it, and return this
        else:
            return 0 (the root is NULL and there is no height)
    bst_size
        set an integer to be the size counter = 0
        if root is not NULL:
            increment the counter by 1
            recursively call with the left of the root
            recursively call with the right of the root
        return counter
    bst_find
        if root is not NULL:
```

```
        compare root's oldspeak to given oldspeak with strcmp()
            if 0 is returned, return the root (matches)
            if  > 0 returned, recursively call with left of root and oldspeak
            if < 0 returned, recursively call with right of root and oldspeak
            else, return NULL (oldspeak does not exist in bst)
        if this portion is reached, return the current root
bst_insert
    if root is not NULL:
        if root's oldspeak matches given oldspeak and root's newspeak matches
        given newspeak:
            return NULL
        find oldspeak position (similar process to bst_find)
        once found, check again that node does not exist
        create new node with given oldspeak and newspeak
        return pointer of created node
bst_print
    if the root is not NULL:
        recursively call with the left of the root
        call node_print() on the root
        recursively call with the right of the root
bst_delete
    if the root pointer is NULL:
        return (have reached the end)
    else:
        recursively call with the address of the left of the root
        recursively call with the address of the right of the root
        call node_delete() on the root
        set the root pointer to NULL
```

bf.c

```
bf_create
    allocate enough memory (of given size) for bloom filter
    if bloom filter pointer is not NULL:
        set primary[] indices to the lower and upper values in salts.h
        set secondary[] indices to the lower and upper values in salts.h
        set tertiary[] indices to the lower and upper values in salts.h
        set bit vector pointer filter
    return bloom filter pointer
bf_delete
    free bloom filter pointer
    set pointer to NULL
bf_size
    return length of bit vector
bf_insert
    hash given oldspeak with primary salt
    hash given oldspeak with secondary salt
    hash given oldspeak with tertiary salt
    for each of the returned hash values v:
        set bit vector at index v with bv_set_bit()
bf_probe
    hash given oldspeak with primary salt
    hash given oldspeak with secondary salt
    hash given oldspeak with tertiary salt
    for each of the returned hash values v:
        if index v of bit vector == 1:
            continue
        if index v of bit vector == 0:
            break and return false
    if this point is reached, return true
bf_count
    create counter and set to 0
    iterating through all indices of bloom filter:
        if value at index is 1:
                increment counter by 1
            else, continue
        return counter
    bf_print
        for i from 0 to end of bloom filter (to size):
            if bloom filter at i is 1, print 1
            if floom filter at i is 0, print 0
```

ht.c

```
    ht_create
        allocate enough memory (of given size) for hash table
        if hash table pointer is not NULL:
            set salts array indices to the lower and upper values in salts.h
            set hash table size to given size
            allocate enough memory for node pointer trees (of size Node * indices)
        return bloom filter pointer
    ht_delete
        for each index of the hash table:
            free corresponding tree with bst_delete
        free hash table pointer
        set pointer to NULL
    ht_size
        return hash table size (from struct)
    ht_lookup
        hash given oldspeak and save its return value
        use return value as index of hash table
        call bst_find() on the bst at the index to see if oldspeak is there
            if it is, return pointer
            if it is not, return NULL
    ht_insert
        hash oldspeak and save its return value
        use return value as index of hash table
        insert oldspeak and newspeak if relevant using bst_insert
    ht_count
        create counter and set it = 0
        iterating through all indices of the hash table:
            check if bst exists at index (if height > 0)
                if so, increment the counter
                else, continue
        return counter
    ht_avg_bst_size
        create bst counter
        create ht counter and set it = ht_count()
        iterating through all indices of the hash table:
            call bst_size() and increment bst counter by it
        return bst counter divided by ht counter
    ht_avg_bst_height
        create bst counter
        create ht counter and set it = ht_count()
        iterating through all indices of the hash table:
            call bst_height() and increment bst counter by it
        return bst counter divided by ht counter
    ht_print
        for all indices of hash table:
            print binary search tree with call to bst_print()
```

banhammer.c

```
    initialize bloom filter and hash table with bf_create and ht_create
    parse through command line options with getopt()
        -h prints help message
        -s prints statistics
        -t size specifies hash table size (default: 10000)
        -f size specifies bloom filter size (default: 2^20)
    call fscanf() to read badspeak words
    add each word to bloom filter with bf_insert() and ht_insert()
call fscanf() to read oldspeak and newspeak words
    add each oldspeak to bloom filter with bf_insert()
    add both oldspeak and newspeak to hash table with ht_insert()
parse through text from stdin to begin filtering
    for each word, check if in bloom filter with bf_probe()
        if word is not in bloom filter (false returned):
            continue parsing
        if word appears to be in bloom filter (true returned):
            check if word is in hash table with ht_lookup() and if it is:
                if there is no newspeak associated with it:
                    insert badspeak into list #1 to notify citizen
                if there is a newspeak associated with it:
                    insert oldspeak into list #2 with translation to notify
                if the word is not in the hash table:
                    continue (bf gave false positive)
if there are contents in list #1 only (meaning only thoughtcrime):
    print mixspeak message with list of badspeak
if there are contents in lists #1 and #2 (meaning thoughtcrime and counseling needed):
    print mixspeak message with list of badspeak and old/newspeak
if there are content in list #2 only (meaning only counseling needed):
    print goodspeak message with list of old/newspeak
```