Din Bostandzic
CSE 13S

Assignment 6 - Down the Rabbit Hole and Through the Looking Glass: Bloom Filters, Hashing, and the Red Queen's Decrees

Pre-Lab Answers

Part 1

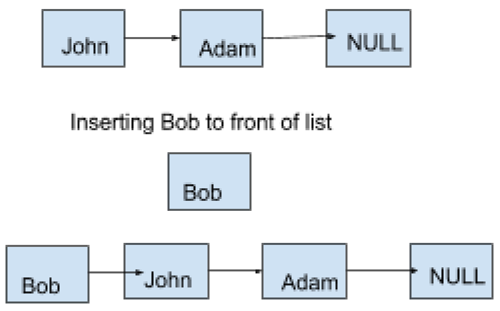
1. The bloom filter can also insert and probe for elements. The pseudocode for inserting and probing the bloom filter is:

```
Begin bf_insert(pass in bf as pointer to BloomFilter record, in key as string)
    Declare index of type unsigned 32 bit type, initialized to 0
     index = call hash(pass in bf->primary as array of unsigned 64 bit type, in key as
              string) % BFSIZE
     Call bv_set_bit(pass in bf->filter as array of BitVector, in index as unsigned 32
                  bit type)
     index =call  hash(pass in bf->secondary as array of unsigned 64 bit type, in key
            as string) % BFSIZE
     Call bv_set_bit(pass in bf->filter as array of BitVector, in index as unsigned 32
                  bit type)
     index =call  hash(pass in bf->tertiary as array of unsigned 64 bit type, in key
            as string) % BFSIZE
     Call bv_set_bit(pass in bf->filter as array of BitVector, in index as unsigned 32
                  bit type)
End bf_insert

Begin bf_probe(pass in bf as pointer to BloomFilter record, in key as string)
    Declare index1 of type unsigned 32 bit type, initialized to 0
    Declare index2 of type unsigned 32 bit type, initialized to 0
    Declare index3 of type unsigned 32 bit type, initialized to 0

     index1=  hash(bf->primary, key) % BFSIZE
     index2 = hash(bf->secondary, key) % BFSIZE
     index3= hash(bf->tertiary, key) % BFSIZE

    Begin if (bv_get_bit(bf->filter, index1) == 1 && bv_get_bit(bf->filter, index2)
==              1 && bv_get_bit(bf->filter, index3) == 1)
                    Return 1
    End if
    Return 0
End bf_probe
```

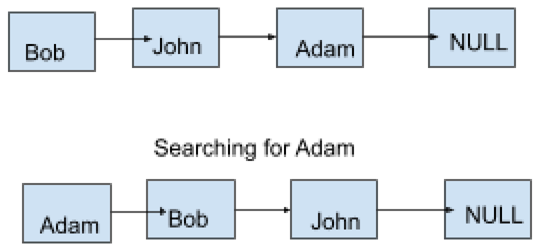
2. Assuming that a bloom filter with m bits and k hash functions is being created, the time complexity will be O(k) based on the number of hash functions being run on the data, while the space complexity is O(m) which is how much of the data is being stored inside the bloom filter.

Part 2

1. The pictures to show how elements are being inserted in different ways into the linked list:
   Insert at front of list:



   Insert at front of list but when searched for, move to front of list:



2. Pseudocode for all the linked list functions:

```
Declare move_to_front as boolean
Declare seek as static integer initialized to 0
Declare numlinks as static integer initialized to 0

Begin ll_node_create(Pass in gs as pointer to HatterSpeak structure)
        Dynamically create node of type pointer to ListNode structure, allocating size of
                ListNode
        Dynamically create the HatterSpeak structure to be used in node->gs

        Begin if (node == NIL)
                return (ListNode *)NIL
        End if
        Assign to node->gs->oldspeak Dynamically allocated space for 100 bytes
        Begin if (gs->oldspeak !=(char*) (NULL))
                Call strcpy(node->gs->oldspeak, gs->oldspeak)
```

```
    End if
    Begin else
        gs->oldspeak = (char*)NULL
    End else

    Assign to node->gs->hatterspeak Dynamically allocated space for 100 bytes

    Begin if (gs->hatterspeak !=(char*) (NULL))
        Call strcpy(node->gs->hatterspeak, gs->hatterspeak)
    End if
    Begin else
        gs->hatterspeak = (char*)NULL
    End else

    node->next = NIL
    return node
End ll_node_create

Begin ll_node_delete(pass in n as pointer to ListNode)
    free(n->gs)
    free(n)
End ll_node_delete


Begin ll_delete(pass in head as pointer to ListNode)
    Begin while (head->next != NIL)
        ll_node_delete(head->next)
        head->next = (head->next)->next
    End while
    ll_node_delete(head)
End ll_delete

Begin ll_insert(pass in head as pointer to pointer to ListNode, in gs as pointer to
      HatterSpeak structure)

     ListNode *node = ll_node_create(gs)
     node->next = *head
    *head = node
     return *head
End ll_insert
```

```
Begin ll_lookup(pass in head as pointer to pointer to ListNode, in key as string)
    Increment seek

    ListNode *headswap = *head
    ListNode *storehead = *head

    Begin if(call strcmp(storehead->gs->oldspeak, key) == 0)
        return storehead
    End if

    storehead = storehead->next

    Begin while (storehead != NIL)
        Begin if(strcmp(key, storehead->gs->oldspeak) == 0 && move_to_front == 0)
            return storehead
        End if
        Begin else if (strcmp(key, storehead->gs->oldspeak) == 0 && move_to_front == 1)
            HatterSpeak *keydataswap = storehead->gs
            storehead->gs = headswap->gs
            headswap->gs = keydataswap
            return headswap
        End else if
        Increment numlinks
        storehead = storehead->next
    End while

    return NIL
End ll_lookup
```

The program is the implementation of a bloom filter, which is made using previous implementation of BitVector, as well as a hashtable that uses linked list for collisions. Two files oldspeak.txt and hatterspeak.txt are read in, placed into the bloom filter (which is used to quickly determine if a certain element happens to probably be in hashtable, instead of doing string comparisons). Then for each oldspeak.txt word a structure is created containing as the key the word and as data value NULL, and placed into respective slot in hash table. The words from hatterspeak.txt are placed into their separate structures as well with the key being the first words in line, and data the word after the space in each line. Once placed in hashtable, user can then enter any word from keyboard, which will get parsed using a regex so that the words are actual words. Then using these words each is probed into the bloom filter and if found, searched in the hashtable to determine if it has a data value (the hatterspeak word) or not. If the word is not in bloom filter then it is ignored. The program also allows for I/O redirection by letting a file to be used as input. Once the input has been read in, and words have been found in hashtable, the words are stored in separate linked lists to be printed in end if the user is responsible for oldspeak words without translations or oldspeak words that do have hatterspeak translations, or both.

The following functional decomposition has been used along with the respective pseudocode below:

```
3.0 ASIGN 6
    3.1 file_readin(in bf as pointer to BloomFilter structure, in ht as pointer to HashTable structure)
    3.2 redir_input(in bf as pointer to BloomFilter structure, in ht as pointer to HashTable structure)
    3.3 printer(in node as pointer to ListNode structure)
    3.4 oldspeakprint(in node as pointer to ListNode Structure)
```

<u>main Data Design</u>

```
Define OPTIONS as string "mbh:f:s"
Define CONTRACTIONS as string "[a-zA-Z0-9]*[-_']*[a-zA-Z0-9]"

Declare tofront, nofront, stats, transwords, notrans as boolean type all initialized to 0
Declare input_num as string initialized to NULL
Declare default_hashtable as integer initialized to 10000
Declare default_bloom as integer initialized to 1048576
Declare c as character
```

<u>main Design</u>

```
Begin main(pass in argc as integer, in argv as string)
    Begin While ((c = getopt(argc, argv,OPTIONS)) != -1)
        Switch(c)
            Case ‘m’
                tofront = 1
                move_to_front = 1
                Break
            Case ‘b’
                nofront = 1
                move_to_front = 0
                Break
            Case ‘h’
                input_num = optarg
                To default_hashtable assign value as converted into integer of
                    input_num
                Break
            Case ‘f’
                input_num = optarg
                To default_bloom assign value as converted into integer of
                input_num
                Break
            Case ‘s’
                stats = 1
                Break
            Default case
                Display “Character not found in string”
                Return exit status fail
    End Switch
End while
Begin if (argc == 1)
    Display “No arguments supplied!”
    Return exit status fail
End if
Begin if (stats == 1)
    puts("stats are\n")
End if

 Begin if (tofront == 1 && nofront == 1)
    Display "this combination is not supported!”
    return -1;
End if

BloomFilter *bf = bf_create(default_bloom)
HashTable *ht = ht_create(default_hashtable)

 HatterSpeak *hs = hs_create((char*)NULL,(char*)NULL)
ListNode *node = ll_node_create(hs)

Begin for (unsigned integer 32 bit  i = 0, i < ht->length, i++)
    ht->heads[i] = node
End for
```

```
    file_readin(bf, ht)
    redir_input(bf, ht)
    return 0
End main
```

<u>file_read Design</u>

```
Begin file_read
    Create file as pointer to file, open oldspeak.txt in readmode
    Begin if (file == NULL)
        Display "could not open file!”
        return
    End if

    Create hatter as pointer to file, open hatterspeak.txt in readmode
      if (hatter == NULL)
        Display “could not open file!”
        Return
    End if

    size_t size = 1024
    Dynamically allocate buffer of string type with size of size
    Dynamically allocate buffer2 of string type with size of size

    Begin while (!feof(file))
        Read in strings from file to buffer
        bf_insert(bf, buffer)
        HatterSpeak *gs = hs_create(buffer, (char*)NULL)
        ht_insert(ht, gs)
    End while
    Close file
    Begin while (!feof(hatter))
        Read in space separated strings from hatter to buffer and buffer2
        bf_insert(bf, buffer)
        HatterSpeak *gs = hs_create(buffer, buffer2)
        ht_insert(ht, gs)
    End while
    Close hatter
End file_read
```

redir_input Design

```
Begin redir_input(in bf as pointer to BloomFilter structure, in ht as pointer to HashTable structure)
    HatterSpeak *hs = hs_create((char*)NULL,(char*)NULL)
    ListNode *stored_notranswords = ll_node_create(hs)
    ListNode *stored_transwords = ll_node_create(hs)

    int returncode
    regex_t regex
     returncode = regcomp(&regex, CONTRACTIONS, REG_EXTENDED)
    Begin if (returncode)
        Display "compilation unsuccessful"
        Return
    End if

    Begin while(!feof(stdin))
         char *matchedword = next_word(stdin, &regex)
        Begin if(matchedword !=NULL)

            Begin for(unsigned long i = 0, i < strlen(matchedword), i++)
                matchedword[i] = tolower(matchedword[i])
            End for

            bool passbf = bf_probe(bf, matchedword)
            Begin if(passbf == 1)
                ListNode *node =  ht_lookup(ht, matchedword)
                Begin if(node != NULL && isalpha(node->gs->hatterspeak[0]) != 0)
                     transwords = 1
                    ll_insert(&stored_transwords, node->gs)
                End if
                Else if (node != NULL && isalpha(node->gs->hatterspeak[0]) == 0)
                    notrans = 1
                    ll_insert(&stored_notranswords, node->gs)
                End else if
                Else if (node != NULL && (isalpha(node->gs->hatterspeak[0]) ==
                0 || isalpha(node->gs->hatterspeak[0]) !=0))

                     transwords = notrans = 1
                End else if
            End if
        End if
    End while
    Call clear_words()
```

```
        Begin if (transwords == 1 && notrans == 1)
                printer(stored_notranswords)
                printer(stored_transwords)
        End if
        Begin else if (transwords == 1 && notrans == 0)
                printer(stored_transwords);
        End else if
        Begin else if (transwords == 0 && notrans == 1)
                printer(stored_notranswords)
        End else if
End redir_input
```

<u>printer Design</u>

```
Begin printer
        Begin if((transwords == 1 && notrans == 1) && isalpha(node->gs->hatterspeak[0]) == 0)
                Display "Dear Comrade,”
                Display "You have chosen to use words that the queen has decreed oldspeak."
                Display “Due to your infraction you will be sent to the dungeon where you will"
                Display "   be taught hatterspeak.”
                Display "Your errors:”

                Begin while(node->next != NULL)
                        Display node->gs->oldspeak
                        node = node->next
                End while
        End if
        else if((transwords == 1 && notrans == 1) && isalpha(node->gs->hatterspeak[0]) != 0)
                Display "Appropriate hatterspeak translations.\n"
                Begin while(node->next != NULL)
                        Display node->gs->oldspeak, node->gs->hatterspeak)
                        node = node->next
                End while
        End else if
        Begin else if(transwords == 1)
                Display "Dear Wonderlander,"
                Display "The decree for hatterspeak finds your message lacking. Some of the”
                Display "   words that you used are not hatterspeak.”
                Display “The list shows how to turn the oldspeak words into hatterspeak.”
                Begin while(node->next != NULL)
                        Display node->gs->oldspeak, node->gs->hatterspeak)
                        node = node->next
                End while
        End else if
        Begin else if(notrans == 1)
                oldspeakprint(node)
        End else if
End printer
```

<u>oldspeakprint Design</u>

```
Begin oldspeakprint
        Display "Dear Wonderlander,"
        Display "You have chosen to use words that the queen has decreed oldspeak."
        Display "Due to your infraction you will be sent to the dungeon where you will"
        Display "   be taught hatterspeak."
        Display "Your errors:"
        Begin while(node->next != NULL)
                Display node->gs->oldspeak);
                node = node->next;
         End while
End oldspeakprint
```