Nicolas Ayala
nmayala@ucsc.edu

Pre-Lab 1)

1) bf_insert(Bloomfilter *bf, char *key){
   bv_set_bit(bf->filter, hash(bf->primary, key) % bf_length(bf));
   bv_set_bit(bf->filter, hash(bf->secondary, key) % bf_length(bf));
   bv_set_bit(bf->filter, hash(bf->tertiary, key) % bf_length(bf));
  }

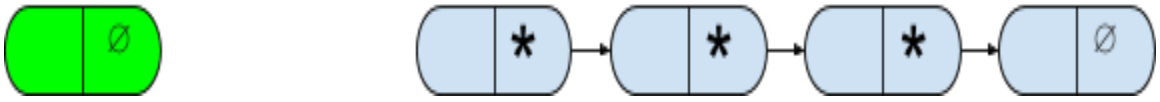
  //A delete function would ruin the purpose of a bloom filter
  //by creating the possibility of a false negative
  bf_delete(Bloomfilter *bf, char *key){
   bv_clr_bit(bf->filter, hash(bf->primary, key) % bf_length(bf));
   bv_clr_bit(bf->filter, hash(bf->secondary, key) % bf_length(bf));
   bv_clr_bit(bf->filter, hash(bf->tertiary, key) % bf_length(bf));
  }
2) The time complexity for inserting an element would be O(k)
   The time complexity for probing an element would be O(k)
   the space complexity would be O(m)

Pre-lab 2)

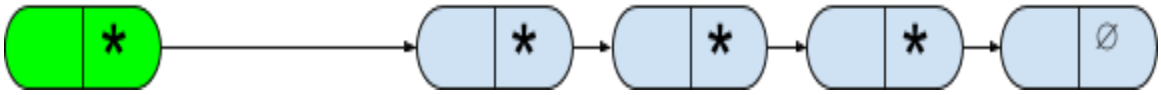
1)

Adding to head

Before

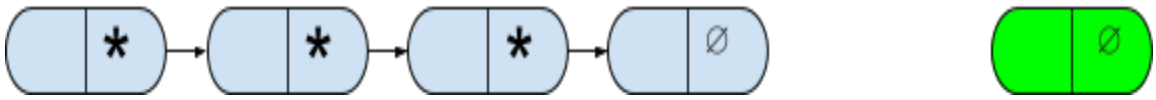


After

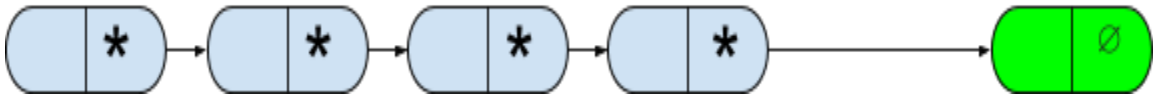


Adding to tail

Before

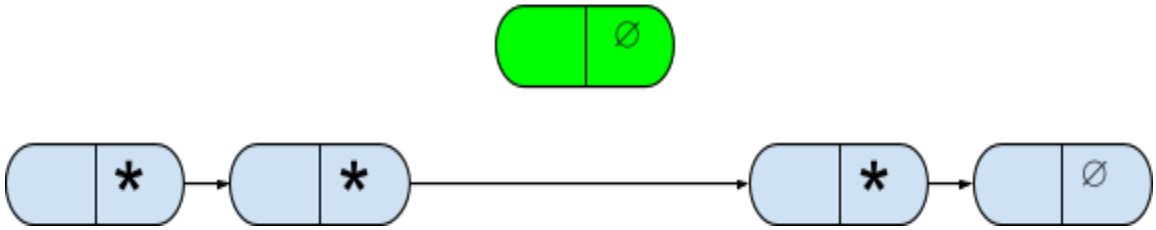


After

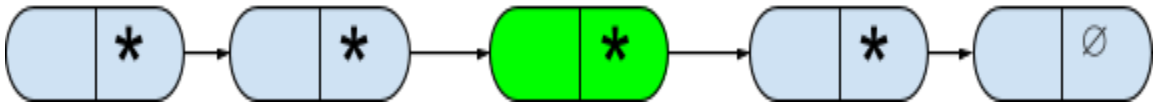


Inserting in the middle

Before



After



2)

```
ListNode *ll_node_create ( GoodSpeak *gs){
        ListNode fresh = (ListNode *) malloc (sizeof( struct ListNode));
        fresh->gs = gs;
        fresh->next = NULL;
}

void ll_node_delete ( ListNode *n){
        gs_delete(n->gs);
        free(n);
}

 void ll_delete ( ListNode * head ){
        ListNode *to_delete_next = head->next;
        free(head);
        while(to_delete_next){
                head = to_delete_next;
                to_delete_next = head->next;
```

```
                free(head);

        }
}

ListNode *ll_insert ( ListNode ** head , GoodSpeak *gs){
        ListNode *fresh = ll_node_create(gs);
        fresh->next = *head;
        head = &fresh;
        return fresh;
}

ListNode * ll_lookup ( ListNode ** head , char * key ){
        if(head){
                ListNode *look_up = (*head);
                while(look_up){
                        char* this_key = gs_oldspeak(look_up->gs);
                        if(strcmp( this_key, key) == 0){  //if they are equal
                                return look_up;
                        }
                        look_up = look_up->next;
                }
        }
        return NULL;
}

void ll_node_print(ListNode *n){
        if(n){
                GoodSpeak gs = n->gs;
                printf(gs->oldspeak);
                printf("\n");
                if(gs->newspeak){
                        printf(gs->newspeak);
                        printf("\n");
                }
        }
}

void ll_print( ListNode * head ){
        while(head){
                ll_node_print(head);
                head = head->next;
```

```
        }
    }
```

Design of program

There will be a file called newspeak.c which contains the main function

the main function will handle the following options:

-s , statistics flag

Instead of displaying censor message, display statistics

-h <size>, specifies size of hash table

default is 10000

-f <size>, specifies size of bloom filter

default is 2^20

-m, program will use move-to-front rule

-b, program will not use move-to-front rule

The flags -m and -b are incompatible.

There will be file called speck.c

This is an implementation of the SPECK cipher, which we will use as a hash

There will be a file called hash.c

This is an implementation of a hash table, with the following functions and pseudocode

```
//Struct definition
typedef struct HashTable{
        uint32_t ***table;
        uint32_t length;
}HashTable;


//Constructor for Hash Table
HashTable *ht_create(uint32_t length){
        HashTable *fresh = (HashTable *) malloc(sizeof(struct HashTable));
        fresh->table = (ListNode **) calloc(length, sizeof(uint32_t *));
        fresh->length = length;
}

//Destructor for Hash Table
void ht_delete(HashTable *h){
        free(h->table);
        free(h);
}

//Looks up the listnode corresponding to a key
ListNode *ht_lookup(Hashtable *h, char *key){
```

```
	uint64_t salt[2];
	salt[0] = 0;
	salt[1] = 0;
	uint32_t index = hash(salt, key);
	if( h->table[index] ){
		ListNode *node = ll_lookup(h->table[index], key);
		return node;
	}
	return NULL;
}

//Inserts a Good speak element into hash, linked lists are used to prevent collisions
void ht_insert(Hashtable *h, Goodspeak *gs){
	uint64_t salt[2];
	salt[0] = 0;
	salt[1] = 0;
	uint32_t index = hash(salt, key);
	if(h->table[index]){
		ll_insert(h->table[index], gs);
		return;
	}else{
		ListNode *node = ll_node_create(gs);
		if(node){
			h->table[index] = &node;
		}
	}
}

uint32_t ht_count(HashTable *h){
	uint32_t count = 0;
	if(h){
		for(uint32_t i = 0; i < h->length;i++){
			if(h->table[i]){
				count++;
			}
		}
	}
	return count;
}

void ht_print(HashTable *h){
	if(h){
		for(uint32_t i = 0; i < h->length;i++){
```

```
                        if(h->table[i]){
                                ll_print(*h->table[i]);
                        }
                }
        }
}
```

There will be a file called bf.c

This contains the implementation of a bloom filter

The bloom filter struct and functions will look like this

```
typedef struct BloomFilter {
        uint64_t primary [2]; // Provide for three different hash functions
        uint64_t secondary [2];
        uint64_t tertiary [2];
        BitVector * filter ;
} BloomFilter;

//Constructor for Bloom Filter
BloomFilter * bf_create ( uint32_t size ) {
        BloomFilter *bf = ( BloomFilter *) malloc ( sizeof ( BloomFilter ) ) ;
        if (bf) {
                bf - > primary [0] = 0 xfc28ca6885711cf7 ; // U.S. Constitution
                bf - > primary [1] = 0 x2841af568222f773 ;
                bf - > secondary [0] = 0 x85ae998311115ae3 ; // Il nome della rosa
                bf - > secondary [1] = 0 xb6fac2ae33a40089 ;
                bf - > tertiary [0] = 0 xd37b01df0ae8f8d0 ; // The Cremation of Sam
                bf - > tertiary [1] = 0 x911d454886ca7cf7 ; //Mcgee
                bf - > filter = bv_create ( size ) ;
                return bf;
        }
        return NULL;
}

//destructor for Bloom filter
void bf_delete(BloomFilter *b){
        bv_delete(b->filter);
        free(b);
}

//returns length of filter
uint32_t bf_length(BloomFilter *b){
        if(b && bf->filter){
                return bv_length(b->filter);
```

```
        }
    }
```

Implementation for bf_insert can be found in Pre-Lab 1)

```
    //Probes a key, produces a definite negative xor a possible positive
    bool bf_probe(BloomFilter *b, char *key){
        if (b){
            if (bv_get_bit(b->filter, hash( b->primary, key) % bf_length(b)) &&
            bv_get_bit(b->filter, hash( b->secondary, key) % bf_length(b)) &&
            bv_get_bit(b->filter, hash( b->tertiary, key) % bf_length(b)) ){
                return true;
            }
        }
        return false;
    }

    //Returns number of bits set in bloom filter
    uint32_t bf_count(BloomFilter *b){
        uint32_t count = 0;
        for(uint32_t i = 0; i < bf_length(b); i++){
            if(bv_get_bit(b->filter, i)){
                count++;
            }
        }
        return count;
    }
```

There will be a file called parser.c, which contains a simple regular expression parser

There will be a list of forbidden words, and translatable words.
A bloom filter will be filled with these words.
A GoodSpeak struct will be created for each of these words

There will be a Hash table, it will use the oldspeak from each GoodSpeak struct as a key and insert it into a linked list

A stream of words will be passed to the program, those words will consult the bloom filter. If any of those words are not definitely absent, then the Hash table is consulted

```
    if the word is present in the Hash table
        if the word is forbidden
            A thoughtcrime message is elicited
        if the word is not forbidden
```

A new speak message is elicited.

If no words are present in the Hash table

No message

The thought crime message:

“

Dear Comrade ,

You have chosen to use degenerate words that may cause hurt
feelings or cause your comrades to think unpleasant thoughts .
This is doubleplus bad . To correct your wrongthink and
preserve community consensus we will be sending you to joycamp
administered by Miniluv .

Your errors :

“

the forbidden words used are then listed

The new speak message:

“

Dear Comrade ,
Submitting your text helps to preserve feelings and prevent
badthink . Some of the words that you used are not goodspeak .
The list shows how to turn the oldspeak words into newspeak .

“

The oldspeak words found and their new speak translations are then displayed in the following format:

old speak word -> new speak word