



Computer Systems and Networks

ECPE 170 – Jeff Shafer – University of the Pacific

Performance Optimization Project

Lab Schedule

Activities

- **Today**
 - Discuss midterm exam
 - **Lab 7 – Performance Optimization Project**
- **Thursday**
 - **Lab 7 – Performance Optimization Project**

Assignments Due

- **** Midterm Exam ****
 - **Tuesday, October 14th**
- **Lab 7**
 - **Due by Oct 15th 5:00am**

Your Personal Repository

```
2014_spring_ecpe170\lab02
```

```
lab03
```

```
lab04
```

```
lab05
```

```
lab06
```

```
lab07
```

```
lab08
```

```
lab09
```

```
lab10
```

```
lab11
```

```
lab12
```

```
.hg
```

Hidden Folder!

(name starts with period)

Used by Mercurial to track all repository history (files, changelogs, ...)

Mercurial .hg Folder

- The existence of a `.hg` hidden folder is what turns a regular directory (and its subfolders) into a special Mercurial repository
- When you add/commit files, Mercurial looks for this `.hg` folder in the current directory or its parents

Lab 7

Performance Optimization Project



Lab Program

- Analyzes *n-gram* statistics of a text document
 - If $n=1$, it looks at individual words
 - If $n=2$, it looks at pairs of words
 - ...

- Print statistics
 - Top 10 *n-grams* in document
 - Total *n-grams*
 - Longest *n-gram*
 - ...

- Provided text files: Moby Dick, Shakespeare

```
unix> ./analysis_program -ngram 2 -hash-table-size <<REDACTED>> < moby.txt
Running analysis program...
```

```
Options used when running program:
ngram 2
details 10
hash-table-size <<REDACTED>>
N-gram size 2
```

Example Output

```
Running analysis... (This can take several minutes or more!)
  Initializing hash table...
  Inserting all n-grams into hash table in lowercase form...
  Sorting all hash table elements according to frequency...
```

Analysis Details:

(Top 10 list of n-grams)

```
1840 'of the'
1142 'in the'
714 'to the'
435 'from the'
375 'the whale' ← Very common in Moby Dick!
367 'of his'
362 'and the'
350 'on the'
328 'at the'
323 'to be'
```

Analysis Summary:

```
214365 total n-grams
114421 unique n-grams
91775 singleton n-grams (occur only once)
Most common n-gram (with 1840 occurrences) is 'of the'
Longest n-gram (4 have length 29) is 'phrenological characteristics'
Total time = 0.200000 seconds
```

*Study of size and shape of cranium
(as an indicator of mental abilities)*

Lab Objectives

1. **Fix memory leaks** so that Valgrind report is clean
 1. Missing a few calls to `free()` somewhere in the code
2. **Improve program performance by 80x**
 1. When compared to original code provided
3. **Document your code changes** by providing a “diff”
 1. Easy to do (1 command!) if you use version control properly and commit the original code before modifying it

Memory Leaks / Valgrind

➤ Reminder 1

➤ For each `malloc()` call, you need a `free()` call

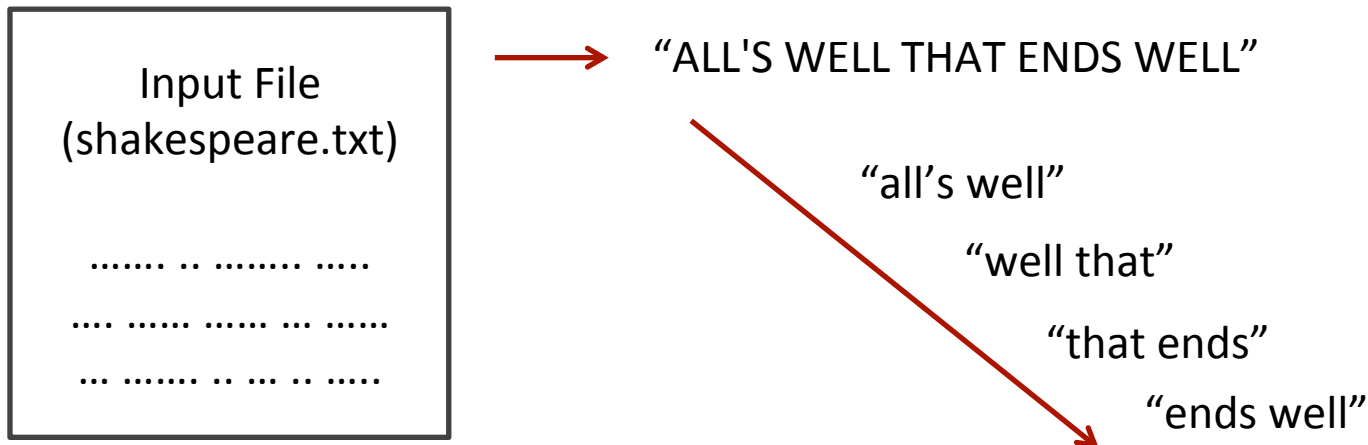
➤ Reminder 2

➤ The line of code that the Valgrind report identifies is where the `malloc()` was

➤ This is NOT where you want to call `free()`!

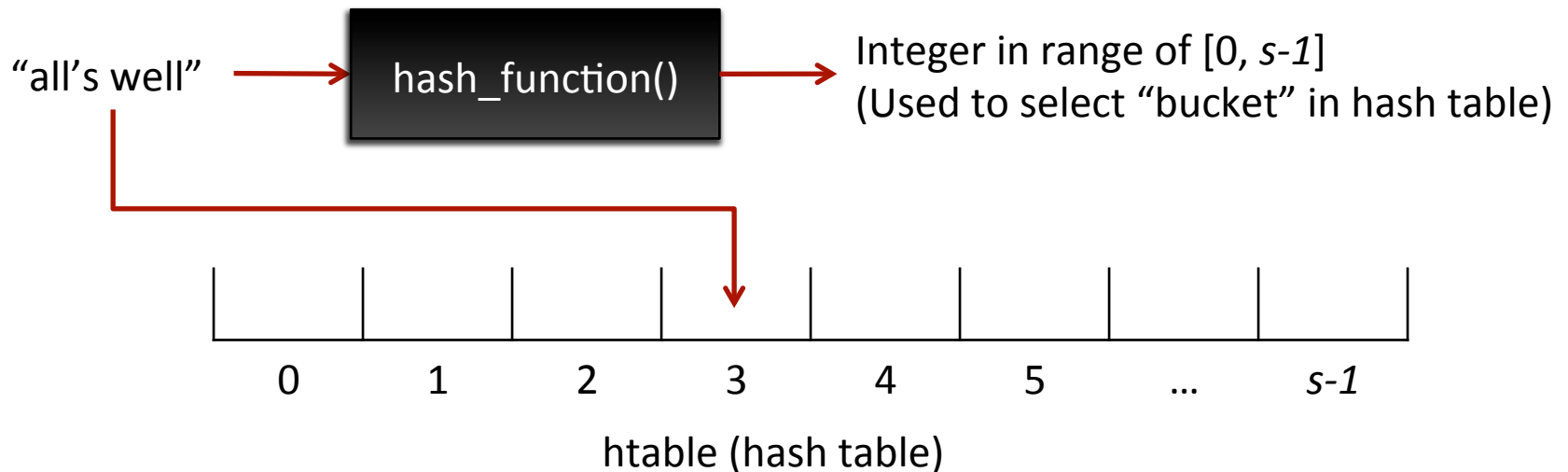
Program Operation (for $n=2$)

- Read each word from the file
- Combine adjacent words into *n-gram* strings
- Convert to lowercase



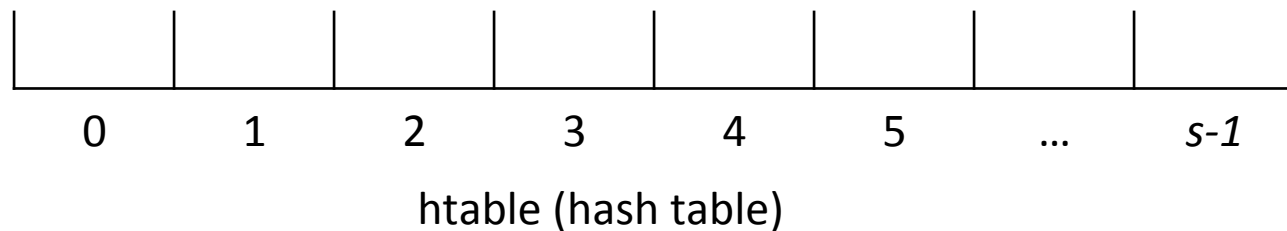
Program Operation

- Apply a **hash function** to each *n-gram* string
- Insert string into corresponding bucket in table



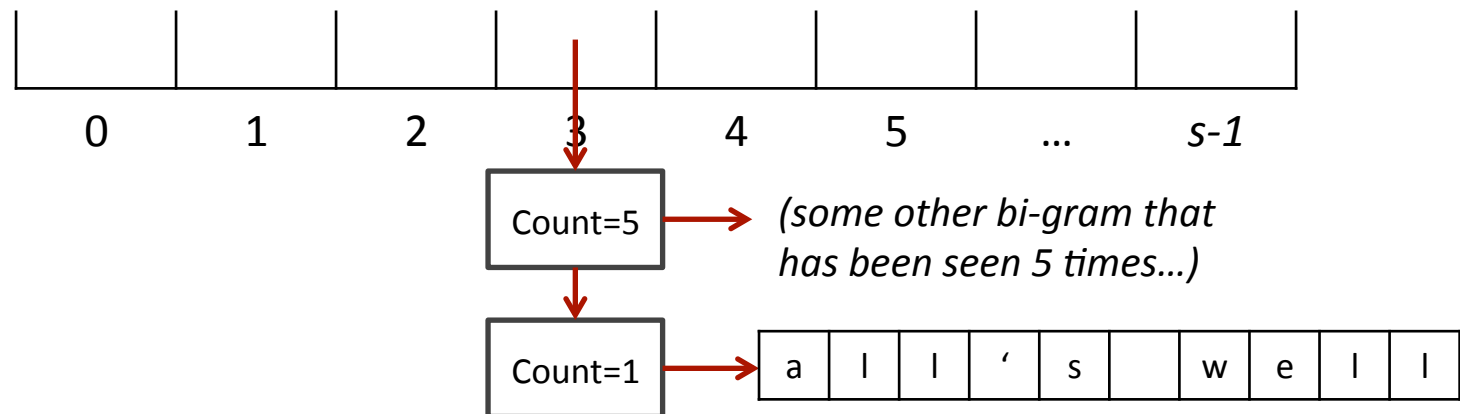
Program Operation

- This hash table is dynamically allocated in a single call to `malloc()`
 - (Technically, it is an array of pointers...)
 - **How many calls to `free()` will it take to clear it?**



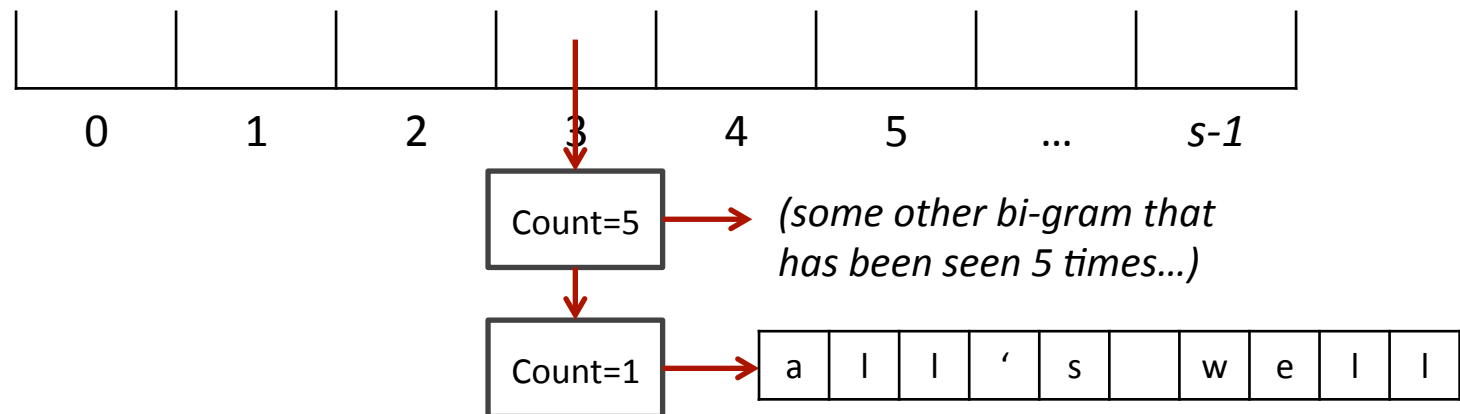
Program Operation

- Each bucket is organized as a linked list. **Search list**
 - If a matching string already exists in the linked list, its frequency counter is incremented
 - Otherwise, a new list element is added at the end with its frequency counter set to 1
 - List element points to char array containing n-gram



Program Operation

- Hash Table: One per program (`malloc()`)
- List element: One per unique word (`malloc()`)
- n-gram array: One per list element (`malloc()`)



Program Operation

- **So how many times will I need to call free() for:**
- **The hash table?**
 - Once! (only allocated once)
- **The list elements?**
 - Once per element (might want a loop?)
- **The unique word array?**
 - Once per word array (i.e. once per list element)

Program Operation

- File input finished
- **Sort** all elements in hash table according to frequency
 - This process is destructive to the hash table
 - All of the linked lists in the hash table are destroyed, and a **single new linked list** of all elements (in sorted order) is created
 - *The elements still exist, just the links have changed*
- Print statistics and exit

Performance Optimization

- The “tips” on the lab writeup are very helpful
- Sorting algorithm efficiency?
- Size of hash table?
 - **Do we want a hash table with lots of elements or fewer elements? (How does this affect the linked lists?)**
- Hash function?
 - **If I increase the size of my hash table, do I need to do anything about the hashing function?**