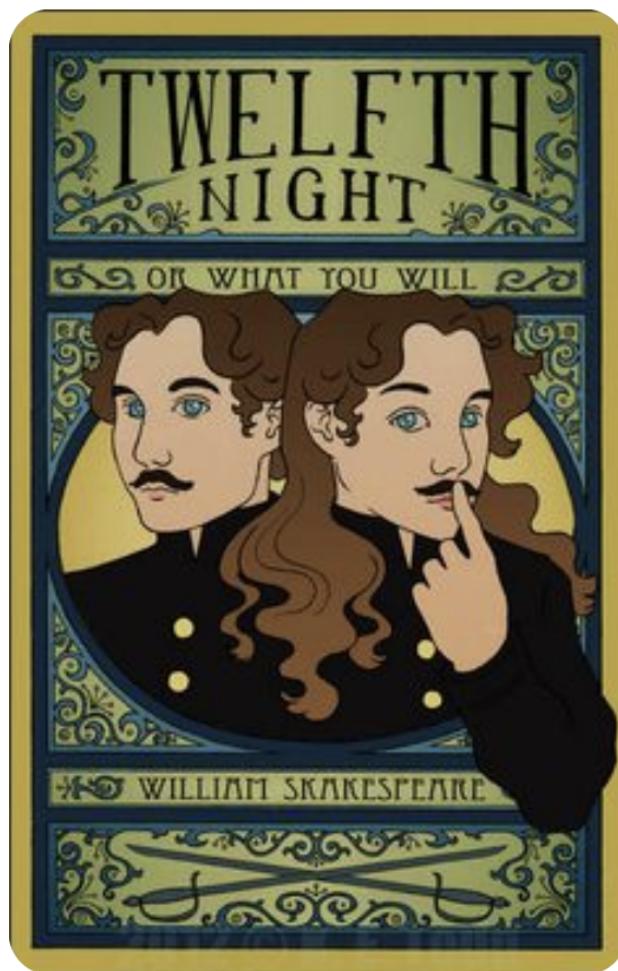


PROGRAMMING ASSIGNMENT #5
CS 2223 D-TERM 2024
CLOSED HASHING
&
DIJKSTRA'S ALGORITHM

ONE HUNDRED POINTS
DUE: THURSDAY, APRIL 18, 2024 11 PM



“If music be the food of love, play on . . .”

1. (10 Points) Read a file and hash the words:

Read a text file (there are three of the form TwelfthNight[something].txt on Canvas) and hash the words using the suggested hash function from *Levitin*, page 269:

$$h \leftarrow 0; \text{for } i \leftarrow 0 \text{ to } s - 1 \text{ do } h \leftarrow (h * C + ord(c_i)) \bmod m,$$

where:

h is the computed hash value,
 s is the length of the word being hashed,
 c_i is the i^{th} character of the word,
 $ord(c)$ is the numerical value of character c in the alphabet in use,
 C is a constant larger than every $ord(c_i)$, and
 m is the modulus defining the size of our hash table.

We will take $ord(c)$ to be the ASCII value of the characters in each word. Thus, $ord(c) \in \{39, 45, 65, 66, \dots, 90, 97, 98, \dots, 122\}$ for $c \in \{', -, A, B, \dots, Z, a, b, \dots, z\}$, respectively. All other characters are discarded; we define words as unbroken sequences (strings!) of consecutive alphabetic characters, both upper and lower case, plus apostrophes and hyphens¹, in the lines read for the file. (Java uses Unicode, but ASCII is still under there somewhere—you just need to dig it out.)

We'll start small with "TwelfthNightActOneScene1.txt" text file, and we will take $C = 123$ and $m = 997$, the largest three-digit prime. These represent parameters that can be changed to alter the shape and composition of our resulting hash table. We'll increase the size of our table when we use the larger files.

You are free to experiment with changing these parameters on your own, of course!

File	Table Size	Grading Comment	Extraneous Comment
TwelfthNightActOneScene1.txt	997	Test Version	'answers' to be supplied on weekend
TwelfthNightActOne.txt	1499	To Be Graded	
TwelfthNightActOne.txt	4000		Something Weird Happens (I think)
TwelfthNightFULL.txt	4000	5 Bonus Points	Something Wonderful Happens (Really!)

¹Thus, *brother's* and *Love-thoughts* are each individual words.

2. (20 Points) Create a Hash Table using Closed Hashing (Open Addressing):

- a. Build a hash table of size 997, i.e. entries from 000 to 996, from the hash values computed in Part 1 above. Load them into the table in the order they occur in the file, *discarding duplicates*. Some of our tables might get close to full, but you needn't worry about re-sizing them—we will examine the effects of their load factors in Part 3 below.
- b. Display the table, starting with table entry 0, in lines of the form:

Hash Address, Hashed Word, Hash Value of Word

(Remember, the Hash Address will not match the Hash Value when the resolution of a collision forces a word to be cascaded down the table. Also, remember that the table should be thought of as a circular list so that a word which cascades past the bottom end of the table gets wrapped to the top in looking for an open place in the table.)

Note: Your program may perform Parts 1 & 2 simultaneously, if you wish.

3. (25 Points) Explore the Hash Table:

Add to your code routines that answer these questions. Some answers may not be unique. You may resolve these issues by presenting either *any* correct answer or *all* correct answers.

- a. How many non-empty addresses are there in the table? What does that make the load factor, α , for our table?
- b. What is the longest empty area in the table, and where is it?
- c. What is the longest (largest) cluster in the table, and where is it?
Note: It might wrap from the end of the table back to the beginning.
- d. What hash value results from the greatest number of distinct words, and how many words have that hash value?
- e. What word is placed in the table farthest from its actual hash value, and how far away is it from its actual hash value?

4. (5 Points) Parameterize Your Code:

Instead of hardcoding your file-to-be-read and the table size, provide a console interface (scanner) allowing the user to enter the name of the file as a string and the size of the table as an integer.

Make sure your code prompts the user for the desired input so that your grader will know what to do.

You can assume that the table size will be large enough for the given file—that is, you need not check to see whether the combination of file and table size will result in an infinite loop.

You also need not check that the file exists, but this would be good practice for handling *file-not-found* exceptions.

I'll provide sample answers from a couple of different text files and different sizes of tables for you to check against as the due date draws closer.



“Nothing that can be can come between me and the full prospect of my hopes.”

5. (40 Points) Implement Dijkstra's Algorithm with weighted graphs

You can hardcode this matrix—we have a lot to get done in a short time—but you can also read it from a file if you prefer. Your code should ask for input from the console for start and destination nodes and use Dijkstra's algorithm to find the “length” of the shortest path between them. It should also display the sequence of nodes that constitute this shortest path.

Alternatively, you can merely ask for a source vertex and produce the shortest path to every vertex (in true Dijkstra form).

0	53	10	12	0	0	0	0	0	0
53	0	33	0	2	0	101	0	0	0
10	33	0	9	30	18	0	0	0	0
12	0	9	0	0	17	0	0	6	0
0	2	30	0	0	14	123	122	0	0
0	0	18	17	14	0	0	137	7	0
0	101	0	0	123	0	0	8	0	71
0	0	0	0	122	137	8	0	145	66
0	0	0	6	0	7	0	145	0	212
0	0	0	0	0	0	71	66	212	0

