Dinh, Bryan
CSCI 2114: Tashfeen's Data Structures
December 5, 2025

Homework 6

Question 1. Head over to the https://www.gutenberg.org and find an interesting book that is available in the text format. Perform word frequency analysis on the text of the book using the Java's built-in hash map. Before you start any counting, make sure you clean your text of any characters that are not lower-case English alphabet or an invisible space character, visibly rendered as "␣". Print out the top ten words with their respective frequencies in descending order. The words should be *at least six letters*. For example, a sample output for the George Orwell's novel *1984* is given bellow. Place the code for this question in a file called `Frequency.java` and copy any output in your PDF to answer this question.

```
|   winston | 282|
|   thought | 158|
|    seemed | 133|
|    moment | 131|
|    always | 122|
|    obrien | 119|
|    almost | 106|
|   another | 105|
|    though | 104|
|    before | 103|
```

The Strange Case of Dr. Jekyll and Mr. Hyde

```
utterson: 127
project:  88
gutenberg:  87
jekyll:  86
lawyer:  67
before:  48
thought:  39
seemed:  30
should:  30
moment:  28
myself:  28
```

Question 2. Let the 27 letter alphabet consist of a space and 26 lower case English letters: {␣, a, b, c, . . . , z}. We may further order the alphabet with $f = \{(␣, 0), (a, 1), (b, 2), (c, 3), (d, 4), \ldots, (z, 26)\}$. A *hash* function h takes any arbitrary string s of the alphabet's characters to a non-negative integer. Let s be of length n where s_i is the i^{th} character of s, $h(s)$ is given as,

$$h(s) \equiv \sum_{i=0}^{n-1} (f(s_i) \times 27^i) \mod m.$$

As trivial examples, for a sufficiently large value of m,

$$h(␣) \equiv 0 \equiv (0 \times 27^0), \quad h(a) \equiv 1 \equiv (1 \times 27^0), \quad h(b) \equiv 2 \equiv (2 \times 27^0) \mod m.$$

And, for a non-trivial example,

$$\begin{aligned} h(\text{bobby}) &\equiv (f(b) \times 27^4) + (f(o) \times 27^3) + (f(b) \times 27^2) + (f(b) \times 27^1) + (f(y) \times 27^0) \mod m \\ &\equiv (2 \times 27^4) + (15 \times 27^3) + (2 \times 27^2) + (2 \times 27^1) + (25 \times 27^0) \mod m \\ &\equiv 1062882 + 295245 + 1458 + 54 + 25 \mod m \\ &\equiv 1359664 \mod m \end{aligned}$$

Write a Java program that takes $h(s)$ and prints out s, e. g., `java Inverse 1359664` prints bobby. Use this program to figure out the string s for the following $h(s)$,

```
1446443617214273139220623316754031238673439502425204756407075972832616437749257375015894090763500398l4
```

Yes, you have to use `BigInteger`; you'll be okay. if a machine is expected to be infallible it cannot also be intelligent

Question 3. Download the professor's implementation of the above hash function in Java from the linked file `HashBrown.java`. Also download the list of 370,105 unique English words. Write a program that finds all the hash collisions among these words for each m_i given bellow.

$$m_1 = 2^{31} - 1 = \texttt{Integer.MAX\_VALUE}$$
$$m_2 = 2^{63} - 1 = \texttt{Long.MAX\_VALUE}$$

For every hash collision corresponding to each m_i give the hash value and the collided words. E. g., there are 30 hash collisions when $m = 2^{31} - 1$ with one of them being $h(\text{decoy}) = 2226796 = h(\text{eternish})$. Use the main method of `HashBrown.java` for this question.

```
881:[aeq, grillades]
13747:[elocutionists, rwd]
2115518164:[elkslip, vapourization]
491318560:[nonoptimistic, ploughgang]
1403560615:[pluvialis, primeness]
1183410379:[paleophytology, piscators]
395425281:[ketoketene, weenong]
2226796:[decoy, eternish]
1929526801:[paumari, uplinks]
1541549160:[cappelenite, postorbital]
1828421126:[cosinesses, outfits]
1147362968:[calumniative, deveined]
743636998:[submaniacally, trigeminus]
1862481874:[amblygonite, jillion]
1339762618:[homemaking, marattia]
513542510:[disroost, gymnastically]
276890:[decerning, nave]
52720974:[cremor, epimanikion]
1399643975:[encroachments, identic]
966567163:[myriologist, shiftable]
1520219554:[reseason, uniformal]
391988318:[orography, xanthide]
911563435:[assibilating, bingeys]
158695826:[araliophyllum, kapote]
194905270:[motets, verrucano]
1054238897:[birthbed, cumbrously]
2008476663:[counteragitate, discourteousness]
164046042:[rebinding, viviparously]
287790385:[decertified, tanged]
25127934:[blandishingly, praecordial]
5731686:[indistinctiveness, unmanipulatable]
```

31 total collisions.

Question 4. Read Cormen, et al. *Introduction to Algorithms* (3rd Edition) pages 647–652 and pages 658–659. You'll be implementing the Dijkstra's and the Bellman-Ford[1] algorithms. Download the files containing the adjacency matrices,

1) `dijkstra1.txt`
2) `dijkstra2.txt`
3) `bellmanford1.txt`
4) `bellmanford2.txt`

The above matrices define edge weights between vertices that should be named by taking as many labels as the rows/columns in the list: a, b, c, . . . , z.

Write a Java class `Graph.java` that implements both the Dijkstra's[2] and the Bellman-Ford algorithms. Use your Dijkstra's implementation to solve the files `dijkstra1.txt` and `dijkstra2.txt`. Similarly, use your Bellman-Ford algorithm to solve the files `bellmanford1.txt` and `bellmanford2.txt`. For each of the graphs and the algorithm, report the minimum distance with which each vertex can be reached and their respective parent vertex in your answers PDF. E. g, the output of Dijkstra's algorithm on `dijkstra1.txt` should look like,

```
a: 0 via a
b: 8 via d
c: 9 via b
d: 5 via a
e: 7 via d
```

You may verify the above by manually drawing the `dijkstra1.txt` graph on a piece of paper and running Dijkstra on it by hand.

Question 5. Can you run the Bellman-Ford algorithm on `dijkstra1.txt` and `dijkstra2.txt` to obtain the same result as you did by running the Dijkstra's algorithm? Similarly, can you run Dijkstra's algorithm on `bellmanford1.txt` and `bellmanford2.txt` to obtain the same results as you did by running Bellman-Ford algorithm?

```
Dijkstra 1 results:
a: 0.0 via a
b: 8.0 via d
c: 9.0 via b
d: 5.0 via a
e: 7.0 via d
Dijkstra 2 results:
a: 0.0 via a
b: 243.0 via t
c: 186.0 via p
d: 237.0 via y
e: 356.0 via b
f: 173.0 via a
g: 191.0 via p
h: 196.0 via p
i: 244.0 via k
j: 188.0 via u
k: 3.0 via a
l: 295.0 via a
m: 204.0 via t
n: 212.0 via f
o: 188.0 via z
p: 115.0 via a
```

[1] Also known as the "distance vector routing" algorithm in the networks field.

[2] Note that Java's class `PriorityQueue` is not dynamic and breaks ties arbitrarily.

```
q: 293.0 via t
r: 233.0 via u
s: 253.0 via v
t: 202.0 via k
u: 183.0 via f
v: 216.0 via y
w: 184.0 via k
x: 271.0 via d
y: 202.0 via j
z: 175.0 via p
Bellman Ford 1 results:
a: 0.0 via a
b: 2.0 via c
c: 4.0 via d
d: 7.0 via a
e: -2.0 via b
Bellman Ford 2 results:
a: 0.0 via a
b: 24.0 via z
c: -2.0 via a
d: 38.0 via s
e: 56.0 via p
f: 31.0 via d
g: -8.0 via a
h: -2.0 via l
i: 75.0 via u
j: 7.0 via c
k: 110.0 via l
l: -11.0 via g
m: 23.0 via p
n: -1.0 via u
o: -5.0 via h
p: 16.0 via v
q: 29.0 via b
r: 4.0 via g
s: 33.0 via x
t: 83.0 via v
u: -5.0 via r
v: 24.0 via j
w: 15.0 via c
x: 10.0 via c
y: -3.0 via l
z: 11.0 via u
```

Question 6. On a graph of exclusively positive edge weights, which algorithm is a better choice?
Dijkstra's algorithm

Question 7. Watch A* (A-Star) Pathfinding Algorithm Visualization on a Real Map on YouTube to see how a path finding algorithm known as the A* looks in execution on a real world graph. Note that A* is essentially the Dijkstra's algorithm with the addition of an "heuristic" function.

Submission Instructions

1) Turn in a PDF containing any outputs and answers from the homework.
2) Also turn in your `Frequency.java`, `Inverse.java`, `HashBrown.java` (with a main method) and finally the `Graph.java`.