Assignment 7 DESIGN.pdf

Ning Jiang

1. **Description of Program:** For this assignment, we will be creating a program that attempts to identify the most likely authors for an anonymous sample of text given a large database of texts with known authors.

Files to be included in directory "asgn7":

1. bf.h: Defines the interface for the Bloom filter ADT. Do not modify this.
2. bf.c: Contains the implementation of the Bloom filter ADT.
3. bv.h: Defines the interface for the bit vector ADT. Do not modify this.
4. bv.c: Contains the implementation of the bit vector ADT.
5. ht.h: Defines the interface for the hash table ADT and the hash table iterator ADT. Do not modify this.
6. ht.c: Contains the implementation of the hash table ADT and the hash table iterator ADT.
7. identify.c: Contains main() and the implementation of the author identification program.
8. metric.h: Defines the enumeration for the distance metrics and their respective names stored in an array of strings. Do not modify this.
9. node.h: Defines the interface for the node ADT. Do not modify this.
10. node.c: Contains the implementation of the node ADT.
11. parser.h: Defines the interface for the regex parsing module. Do not modify this.
12. parser.c: Contains the implementation of the regex parsing module.
13. pq.h: Defines the interface for the priority queue ADT. Do not modify this.
14. pq.c: Contains the implementation for the priority queue ADT.
15. salts.h: Defines the primary, secondary, and tertiary salts to be used in your Bloom filter implementation. Also defines the salt used by the hash table in your

hash table implementation.
16. speck.h: Defines the interface for the hash function using the SPECK cipher. Do not modify this.
17. speck.c: Contains the implementation of the hash function using the SPECK cipher. Do not mod-
ify this.
18. text.h: Defines the interface for the text ADT. Do not modify this.
19. text.c: Contains the implementation for the text ADT

Structure & explanation / Pseudocode:
Explanations for some of the equations are cited from asgn7.pdf.
ht.c
It is used to create a hash table to quickly store and retrieve unique words in a sample of text. We well use linear probing to resolving linear collection.
The hash table will contain slats, the size of the text, the node which supposed to be put into the hash table.

Iterating Over Hash Tables
This is ADT for iterate over all the entries in a hash table.

*Node *ht_iter(HashTableIterator *hti)*
Returns the pointer to the next valid entry in the hash table.

Node.c
This is the ADT which will create the node for hash table.
Wach node will contain a word and a it's count.

Bf.c
A Bloom filter can be represented as an array of m bits, or a bit vector. A Bloom filter should utilize k different hash functions. Using these hash functions, a set element added to the Bloom filter is mapped to at most k of the m bit indices, generating a uniform pseudo-random distribution.

Bv.c
A bit vector is an ADT that represents a one dimensional array of bits, the bits in which are used to denote if something is true or false (1 or 0).This is an efficient ADT since, in order to represent the truth or falsity of a bit vector of n items, we can use n/8 uint8_ts instead of n, and being able to access 8 indices with asingle integer access is extremely cost efficient. Since we cannot directly access a bit, we must use bitwise operations to

get, set, and clear a bit within a byte.

Text.c

We will need a new ADT to encapsulate the parsing of a text and serve as the in-memory representation of the distribution of words in the file. This is the Text ADT. It will contain a hash table, Bloom filter, and the count of how many words are in the text. This is not the number of unique words in the text, but the total number of words in the text.

*double text_dist(Text *text1, Text *text2, Metric metric)*
In this function, we will use three different method to calculate the distance:

2.1 Manhattan Distance

The *Manhattan distance* is the simplest of the three to compute. To compute it, you simply take the magnitude of the difference between each component of the vector. You can think of it as how far you would have to go if you had to follow the gridded streets of Manhattan. This is computed as:

$$\text{MD} = \sum_{i \in n} |u_i - v_i|$$

With the example texts we get the absolute value of the component distances before summing them:

- "hello" : $|u_1 - v_1| = |\frac{1}{2} - \frac{0}{3}| = \frac{1}{2}$
- "goodbye" : $|u_2 - v_2| = |\frac{0}{2} - \frac{2}{3}| = \frac{2}{3}$
- "world" : $|u_3 - v_3| = |\frac{1}{2} - \frac{1}{3}| = \frac{1}{6}$

This gives us a Manhattan distance of $\frac{4}{3}$.

2.2 Euclidean Distance

The *Euclidean distance* is calculated very similarly to the Manhattan distance, except you are able to go as the bird flies. Thus, we just need to remember how Euclid taught us to find the length of the hypotenuse of a triangle, so we add the squares of all the magnitudes together before getting the square root of the sum. This is computed as:

$$\text{ED} = \sqrt{\sum_{i \in n} (u_i - v_i)^2}$$

With the example texts we first square the component distances before summing them:

- "hello" : $(u_1 - v_1|)^2 = (\frac{1}{2} - \frac{0}{3})^2 = \frac{1}{4}$
- "goodbye" : $(u_2 - v_2|)^2 = (\frac{0}{2} - \frac{2}{3})^2 = \frac{4}{9}$
- "world" : $(u_3 - v_3|)^2 = (\frac{1}{2} - \frac{1}{3})^2 = \frac{1}{36}$

We then add all the values together to get a sum of $\frac{26}{36}$, giving us a Euclidean distance of $\sqrt{\frac{26}{36}} \approx 0.85$.

2.3 Cosine Distance

The *cosine distance* is different from the first two in that we derive it from the *cosine similarity* of two vectors, or the cosine of the angle between two vectors.

$$\text{Cos}(\Theta) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| \times |\vec{v}|}$$

If we are to normalize the vectors, the bottom of the fraction is just 1, and so we only need to get the dot product of the two vectors:

$$\vec{u} \cdot \vec{v} = \sum_{i \in n} u_i \times v_i$$

Note that the angle approaches 0 as the cosine of the angle approaches 1. To stay consistent with the other metrics, we instead get the cosine *distance* by subtracting the similarity from 1. This ensures that the texts that are more similar will have a smaller computed distance. For our example texts we multiply the vector components together before summing all the results:

- "hello" : $u_1 \times v_1 = \frac{1}{2} \times \frac{0}{3} = 0$
- "goodbye" : $u_2 \times v_2 = \frac{0}{2} \times \frac{2}{3} = 0$
- "world" : $u_3 \times v_3 = \frac{1}{2} \times \frac{1}{3} = \frac{1}{6}$

The computed cosine similarity is $\frac{1}{6}$. We subtract this from 1 for the cosine distance, giving us a cosine distance of $1 - \frac{1}{6} = \frac{5}{6}$.