Assignment 7 Writeup

Nhan Nguyen

March 2022

1 Introduction

For this writeup, I'll describe the behavior of the program based on a few changes to the parameters of the program:

- How the limitation of the noise affects the statistics
- How the text size affects the statistics
- How different metrics compare with each other (Euclidean, Manhattan, and Cosine)

2 main() description and code

2.1 Idea

The purpose of the main() program is to take the subject text and compare it with the author's text in the database and produce the distance between the author's text and the subject text
After acquiring all the distances, the program will display the top **k** (k is decided by the user) authors who is the most similar in terms of text to the subject text
To accomplish this task, the main() program will be utilizing the Text data structure (which comprises of both the hash table and the bloom filter), and the priority queue

2.2 Code

```
#define min(a, b) (a < b) ? a : b

int main(int argc, char **argv){
  int opt;
  FILE *data = fopen("lib.db", "r");
  FILE *noise = fopen("noise.txt", "r");
  if(data == NULL || noise == NULL){
    printf("Can't open the files\n");
    return EXIT_FAILURE;
```

```
}
uint32_t matches = 5;
uint32_t filter = 100;
Metric metric = EUCLIDEAN;
char *ptr;
while((opt = getopt(argc, argv, "d:n:k:l:emch")) != -1){
  switch(opt){
    case 'd':
      fclose(data);
      data = fopen(optarg, "r");
      if(data == NULL){
        printf("Can't open the file\n");
        return EXIT_FAILURE;
      }
      break;
    case 'n':
      fclose(noise);
      noise = fopen(optarg, "r");
      if(noise == NULL){
        printf("Can't open the file\n");
        return EXIT_FAILURE;
      }
      break;
    case 'k':
      matches = (uint32_t) strtoul(optarg, &ptr, 10);
      break;
    case 'l':
      filter = (uint32_t) strtoul(optarg, &ptr, 10);
      break;
    case 'e':
      metric = EUCLIDEAN;
      break;
    case 'm':
      metric = MANHATTAN;
      break;
    case 'c':
      metric = COSINE;
      break;
    case 'h':
      break;
    default:
      break;
  }
}
FILE *limitnoise = tmpfile();
char *inserts = malloc(100 * sizeof(char));
for(uint32_t i = 0; i < filter; i++){
  fgets(inserts, 100, noise);
  fprintf(limitnoise, "%s\n", inserts);
}
rewind(limitnoise);
Text *ban = text_create(limitnoise, NULL);
Text *subject = text_create(stdin, ban);
uint32_t cases;
fscanf(data, "%u", &cases);
PriorityQueue *pq = pq_create(cases);
char *author = malloc(100 * sizeof(char));
```

```
  char *path = malloc(100 * sizeof(char));
  fgets(path, 100, data);
  uint32_t cnt = 0;
  for(uint32_t i = 0; i < cases * 2; i++){
    if(i % 2){
      fgets(path, 100, data);
      path[strlen(path) - 1] = '\0';
      FILE *authors = fopen(path, "r");
      if(authors == NULL) continue;
      Text *source = text_create(authors, ban);
      //text_dist(source, subject, metric);
      enqueue(pq, author, text_dist(source, subject, metric));
      cnt++;
      fclose(authors);
      text_delete(&source);
    }else{
      fgets(author, 100, data);
      author[strlen(author) - 1] = '\0';
    }
  }
  uint32_t loops = min(matches, cnt);
  printf("Top %u, metric: %s, noise limit: %u\n", matches,
    metric_names[metric], filter);
        for(uint32_t i = 1; i <= loops; i++){
    char *outputa;
    double dist;
    dequeue(pq, &outputa, &dist);
    printf("%u) %s [%0.15f]\n", i, outputa, dist);
  }
  pq_delete(&pq);
  text_delete(&ban);
  text_delete(&subject);

  free(author);
  free(path);
  free(inserts);
  fclose(data);
  fclose(noise);
  fclose(limitnoise);
  return 0;
}
```

3 Interesting Observation

3.1 Observation 1: Noise limitation

First of all, what is a noise. A noise is the words that are commonly used in the English language like "a", "this", "is", etc. By filtering out all the words from both the subject text and the author's text, the subject's and the author's unique text are better represented and thus the estimation will become more accurate. Let's look at a few limitations to see the relationship: (Note that the metrics used will be Euclidean, and the subject text is William Shakespeare)

```
Top 10, metric: Euclidean distance, noise limit: 100
1) William Shakespeare [0.000000000000000]
2) William D. McClintock [0.026516404305548]
3) Dante Alighieri [0.026865145178084]
4) Edgar Allan Poe [0.027829100437704]
5) Johann Wolfgang von Goethe [0.027873601844561]
6) Henry Timrod [0.028370953256878]
7) Lew Wallace [0.028546232963695]
8) Saxo Grammaticus [0.029282174356051]
9) Rudyard Kipling [0.029385152618783]
10) Various [0.029410943073253]
```

This is the distance with the default noise limit at 100, now let's increase this limit to 5000:

```
Top 10, metric: Euclidean distance, noise limit: 5000
1) William Shakespeare [0.000000000000000]
2) Charles Dickens [0.032180445179866]
3) Tobias Smollett [0.033052832299086]
4) H. G. Wells [0.033064441635230]
5) Various [0.033095361444112]
6) Max Beerbohm [0.033205801285139]
7) Washington Irving [0.033230519415569]
8) Dante Alighieri [0.033341126210632]
9) Thomas Babington Macaulay [0.033762909883515]
10) Bret Harte [0.034145808813015]
```

Let's us also increase the noise limit to 10000 which is the entire noise.txt

```
Top 10, metric: Euclidean distance, noise limit: 10000
1) William Shakespeare [0.000000000000000]
2) Charles Dickens [0.036188235205489]
3) Dante Alighieri [0.037399677359559]
4) H. G. Wells [0.037819677285964]
5) Tobias Smollett [0.038401720335874]
6) Eric S. Raymond [0.038540312743114]
7) Max Beerbohm [0.038610518981084]
8) Washington Irving [0.038631133357005]
9) Various [0.039204224239672]
10) Thomas Babington Macaulay [0.039888610901936]
```

From these 3 iterations of the code, let's look closer to the text with author Various. Through all 3 iterations, we see the distance between William Shakespeare and Various kept on increasing. Why is this the case? This is because the Various uses more contemporary English when compared to old English in William Shakespeare. Therefore, the Various text will have much less unique words compared to Shakespeare's text. Ultimately, the distance between the 2 text increases, which in this case is more accurate.

3.2 Observation 2: Text size

For this observation, I will be using the example **identify** executable already provided in order to make a comparison on how accurate the calculations are. Let's start with an empty file text.txt, using the euclidean distance and the regular database:

```
nhan@nhan-VirtualBox:~/nhtrnguy/asgn7$ ./identify -k 10 < text.txt
Top 10, metric: Euclidean distance, noise limit: 100
1) Various [0.021911182964015]
2) Washington Irving [0.021960899873121]
3) Andrew Lang [0.022002466856375]
4) Tobias Smollett [0.022053456588987]
5) Thomas Babington Macaulay [0.022636768660189]
6) John Fiske [0.022640831954373]
7) Alphonse Daudet [0.022797695917442]
8) Bret Harte [0.023182188044944]
9) Thomas Carlyle [0.023276627604120]
10) Thomas Bailey Aldrich [0.023368688192132]
nhan@nhan-VirtualBox:~/nhtrnguy/asgn7$ ./identify-example -k 10 < text.txt
Top 10, metric: Euclidean distance, noise limit: 100
1) Various [0.021911182964015]
2) Washington Irving [0.021960899873121]
3) Andrew Lang [0.022002466856375]
4) Tobias Smollett [0.022053456588987]
5) Thomas Babington Macaulay [0.022636768660189]
6) John Fiske [0.022640831954373]
7) Alphonse Daudet [0.022797695917442]
8) Bret Harte [0.023182188044944]
9) Thomas Carlyle [0.023276627604120]
10) Thomas Bailey Aldrich [0.023368688192132]
```

As you can see not only is the order correct but the numbers are also correct, but this might be a speculation when we start to include more text from William Shakespeare's

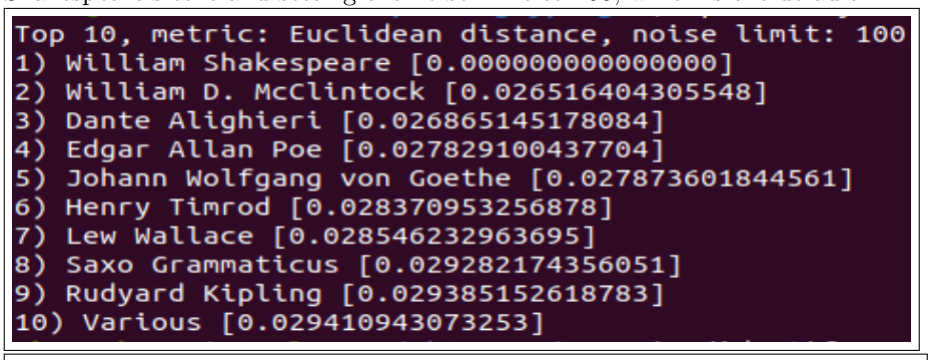
```
nhan@nhan-VirtualBox:~/nhtrnguy/asgn7$ ./identify -k 10 < text.txt
Top 10, metric: Euclidean distance, noise limit: 100
1) Henry Timrod [0.031715493615859]
2) George W. Sands [0.032362222514025]
3) Dante Alighieri [0.032478928487673]
4) Edgar Allan Poe [0.033686169562466]
5) William D. McClintock [0.034378594446271]
6) William Shakespeare [0.034969283942518]
7) Adelaide Anne Procter [0.036508159597910]
8) Henry Kendall [0.036712515694309]
9) Johann Wolfgang von Goethe [0.037019175242330]
10) Alfred Lord Tennyson [0.037086589279603]
nhan@nhan-VirtualBox:~/nhtrnguy/asgn7$ ./identify-example -k 10 < text.txt
Top 10, metric: Euclidean distance, noise limit: 100
1) Henry Timrod [0.031715493615859]
2) George W. Sands [0.032362222514026]
3) Dante Alighieri [0.032478928487673]
4) Edgar Allan Poe [0.033686169562469]
5) William D. McClintock [0.034378594446271]
6) William Shakespeare [0.034969283942530]
7) Adelaide Anne Procter [0.036508159597913]
8) Henry Kendall [0.036712515694309]
9) Johann Wolfgang von Goethe [0.037019175242330]
10) Alfred Lord Tennyson [0.037086589279605]
```

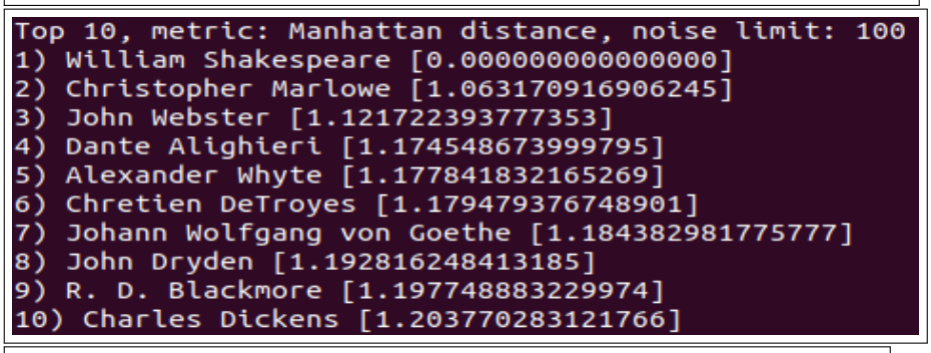
When we take the first 1000 lines from Shakespeare and insert it into the text, we can see that while the ordering was still correct the numbers start to differ. One of the reason might be because the statistic shown in the previous iteration (i.e text.txt is empty) the numbers are not truly equal as we have learned back in assignment 2, and from this inequality, the error kept on increasing and thus the number starts to differ. However, the error remains acceptable since it doesn't affect the ordering when we use the entire William Shakespeare's text.

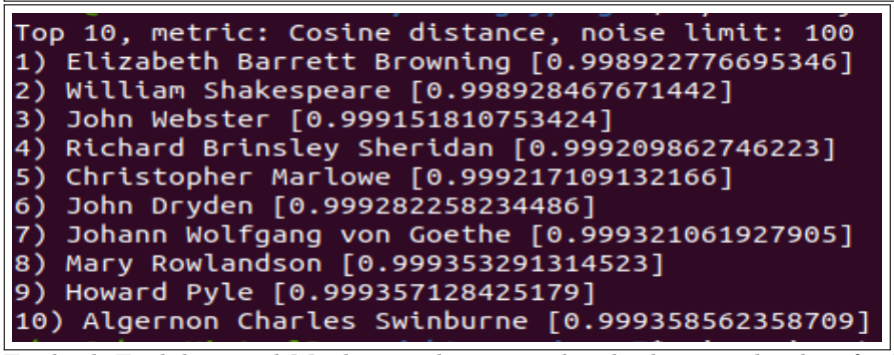
```
nhan@nhan-VirtualBox:~/nhtrnguy/asgn7$ ./identify -k 10 < texts/william-shakespeare.txt
Top 10, metric: Euclidean distance, noise limit: 100
1) William Shakespeare [0.000000000000000]
2) William D. McClintock [0.026516404305548]
3) Dante Alighieri [0.026865145178084]
4) Edgar Allan Poe [0.027829100437704]
5) Johann Wolfgang von Goethe [0.027873601844561]
6) Henry Timrod [0.028370953256878]
7) Lew Wallace [0.028546232963695]
8) Saxo Grammaticus [0.029282174356051]
9) Rudyard Kipling [0.029385152618783]
10) Various [0.029410943073253]
nhan@nhan-VirtualBox:~/nhtrnguy/asgn7$ ./identify-example -k 10 < texts/william-shakespeare.txt
Top 10, metric: Euclidean distance, noise limit: 100
1) William Shakespeare [0.000000000000000]
2) William D. McClintock [0.026516404305561]
3) Dante Alighieri [0.026865145178097]
4) Edgar Allan Poe [0.027829100437714]
5) Johann Wolfgang von Goethe [0.027873601844574]
6) Henry Timrod [0.028370953256889]
7) Lew Wallace [0.028546232963706]
8) Saxo Grammaticus [0.029282174356059]
9) Rudyard Kipling [0.029385152618790]
10) Various [0.029410943073263]
```

3.3 Observation 3: The different metrics

Node that while implementing the author authentication, we used 3 different metrics to identify the authors from the subject text: Euclidean, Manhattan, and Cosine. Below is the statistics of each metrics respectively using William Shakespeare's text and setting the noise limit to 100, which is the default.







For both Euclidean and Manhattan distances, they both correctly identify the William Shakespeare's text since the value is 0.0. However for the Cosine distance, the program incorrectly identify William Shakespeare, this is because for the cosine distance, the more the result increases towards 1 the more similar the subject text is to the author's, and since we're checking for the lowest results, the number 1 result that shows up might be the most inaccurate text compare to the subject text.

Between the Manhattan distance and the Euclidean distance, I can see that the

distance increase up much faster as we move down the list. This is due to 2 factors: we need to square root the result after the calculation and that we need to square up the difference for each word in terms of frequency. Since the ratio can either be equal or less than 1, once we square that difference between the ratio, it will just get even smaller, and that result will become even smaller once we square root it.