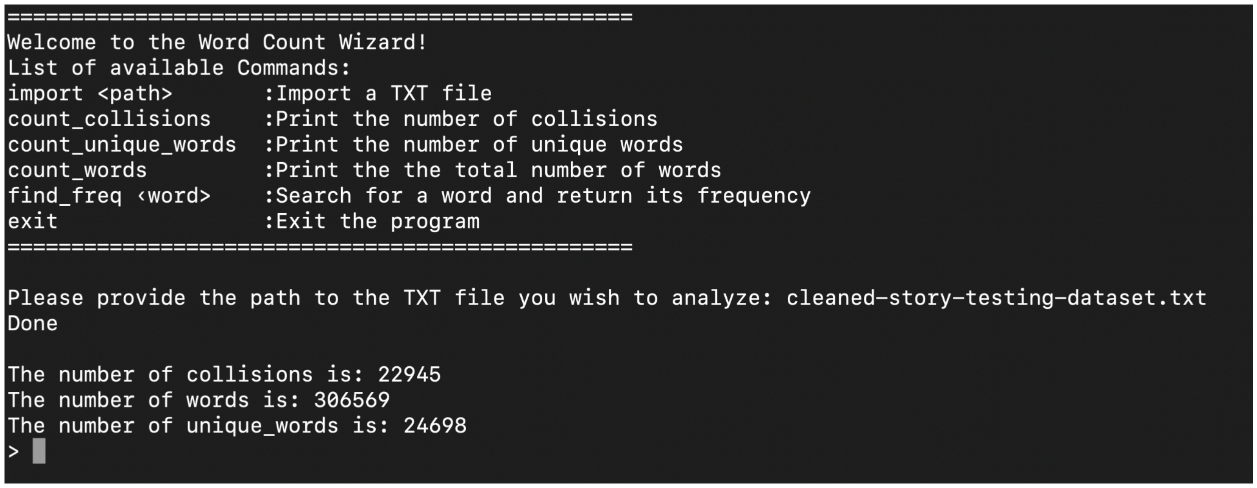
Hash Functions

1. Linear ASCII sum hashing
   (reference from Lecture #19, slide 23)



```
=================================================
Welcome to the Word Count Wizard!
List of available Commands:
import <path>       :Import a TXT file
count_collisions    :Print the number of collisions
count_unique_words  :Print the number of unique words
count_words         :Print the the total number of words
find_freq <word>    :Search for a word and return its frequency
exit                :Exit the program
=================================================

Please provide the path to the TXT file you wish to analyze: cleaned-story-testing-dataset.txt
Done

The number of collisions is: 22945
The number of words is: 306569
The number of unique_words is: 24698
> 
```

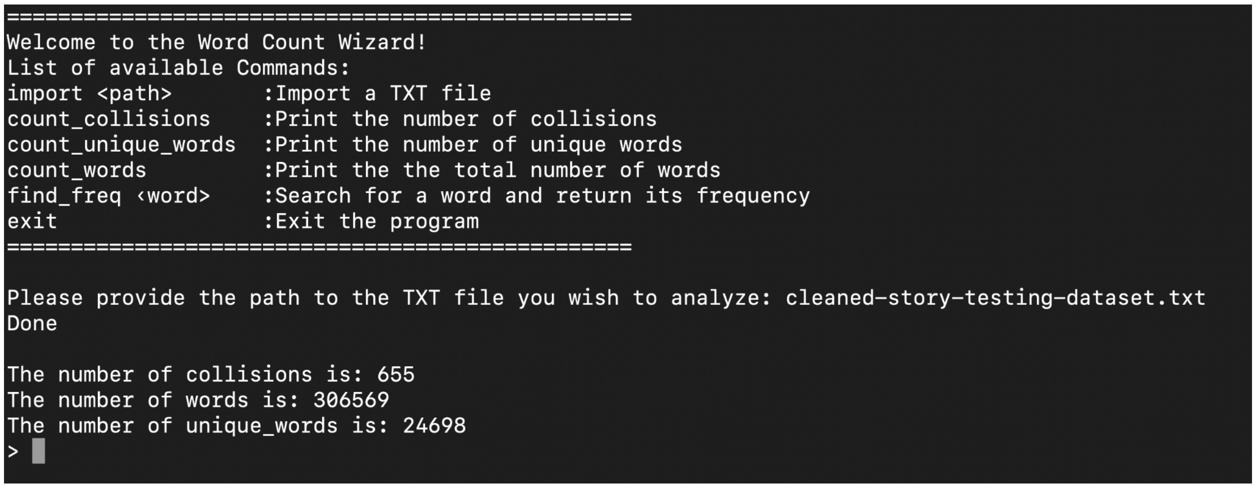
Implementation in Pseudocode:

```
Algorithm linear_ascii_sum_hash(key)
hash ← 0
for i ← 0 to key.size − 1 do
    hash ← hash + ASCII value of key[i]
return hash
```

Implementation in C++:

```
for (size_t i = 0; i < key.size(); ++i)
    {
        hash += key[i]; // add the ascii value of each character in the hashcode
    }
```

2. Polynomial hashing
(reference and inspiration from Lecture #19, slide 24)



```
==================================================
Welcome to the Word Count Wizard!
List of available Commands:
import <path>       :Import a TXT file
count_collisions    :Print the number of collisions
count_unique_words  :Print the number of unique words
count_words         :Print the the total number of words
find_freq <word>    :Search for a word and return its frequency
exit                :Exit the program
==================================================

Please provide the path to the TXT file you wish to analyze: cleaned-story-testing-dataset.txt
Done

The number of collisions is: 655
The number of words is: 306569
The number of unique_words is: 24698
> 
```

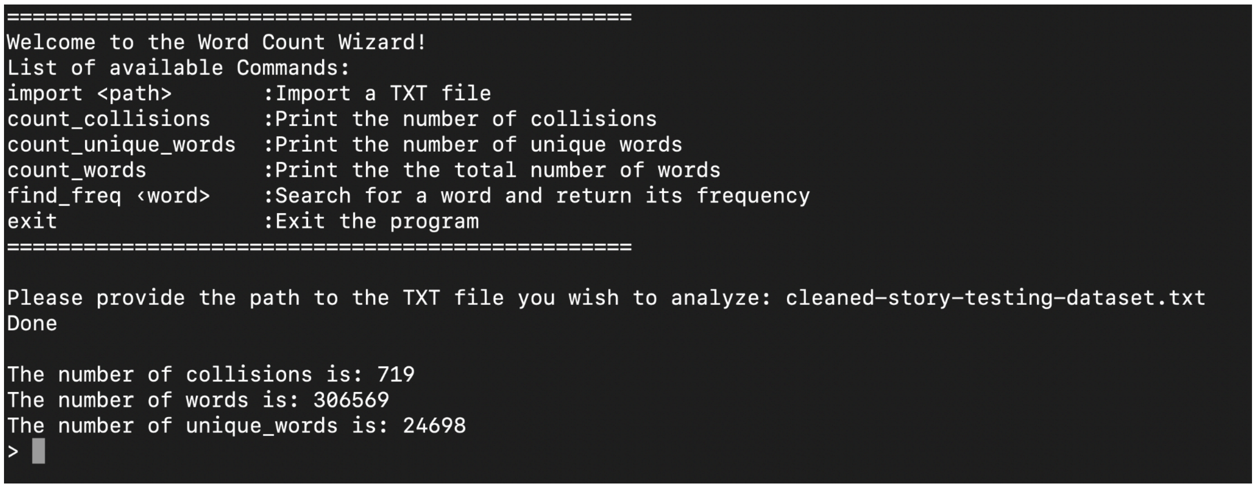
Implementation in Pseudocode:

```
Algorithm polynomial_hash(key)
  hash ← 0
  p ← 31  { Polynomial base }
  m ← capacity  { Modulus based on hash table size }
  for i ← 0 to key.size − 1 do
    hash ← (hash * p + ASCII value of key[i]) mod m
  return hash
```

Implementation in C++:

```
 unsigned int p = 31; // polynomial hash function base
        unsigned int m = capacity; // use table capacity for modulus operator
        for (unsigned int i = 0; i < key.length(); i++) // iterate over characters in the key
            {
                hash = (hash * p + key[i]) % m; // hash function
            }
```

3. Cycle Shift hashing
   (reference and inspiration from Lecture #19, slide 27)



```
=================================================
Welcome to the Word Count Wizard!
List of available Commands:
import <path>       :Import a TXT file
count_collisions    :Print the number of collisions
count_unique_words  :Print the number of unique words
count_words         :Print the the total number of words
find_freq <word>    :Search for a word and return its frequency
exit                :Exit the program
=================================================

Please provide the path to the TXT file you wish to analyze: cleaned-story-testing-dataset.txt
Done

The number of collisions is: 719
The number of words is: 306569
The number of unique_words is: 24698
> 
```

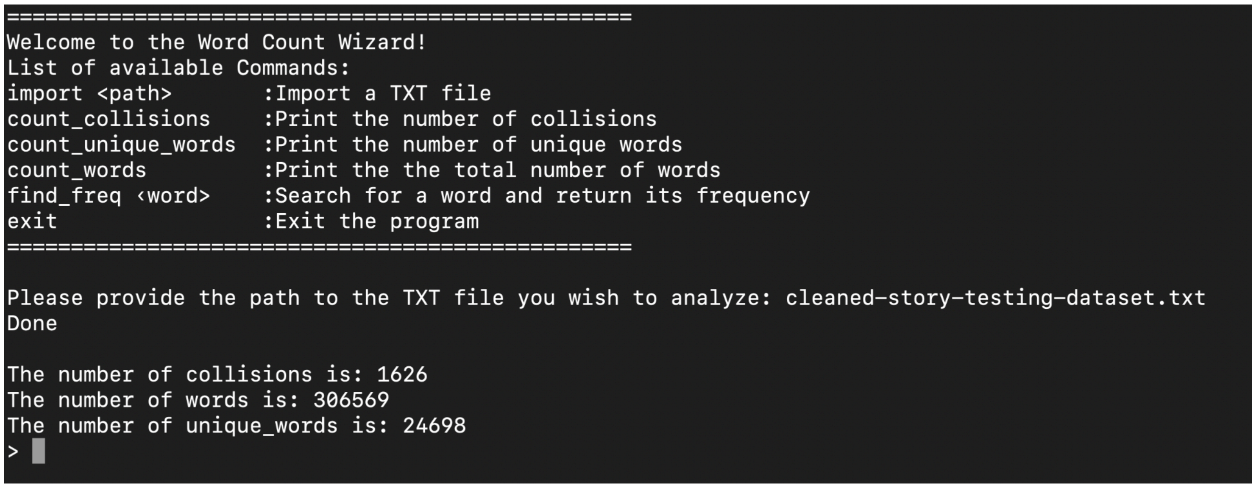
Implementation in Pseudocode:

```
Algorithm cycle_shift_hash(key)
  hash ← 0
  for i ← 0 to key.size − 1 do
    hash ← (hash << 5) OR (hash >> 27)  { Left rotate by 5 bits }
    hash ← hash XOR ASCII value of key[i]
  return hash
```

Implementation in C++:

```
hash = 0; // initialize hash
          for (size_t i = 0; i < key.size(); ++i)
          {
            hash = (hash << 5) | (hash >> (27)); // left rotate by 5 bits
            hash ^= key[i]; // XOR with the current character
          }
```

4. Linear ASCII product hashing
   (reference and inspiration from Lecture #19, slide 23 but I changed from being the sum of the ASCII values to their product)



```
=================================================
Welcome to the Word Count Wizard!
List of available Commands:
import <path>       :Import a TXT file
count_collisions    :Print the number of collisions
count_unique_words  :Print the number of unique words
count_words         :Print the the total number of words
find_freq <word>    :Search for a word and return its frequency
exit                :Exit the program
=================================================

Please provide the path to the TXT file you wish to analyze: cleaned-story-testing-dataset.txt
Done

The number of collisions is: 1626
The number of words is: 306569
The number of unique_words is: 24698
> 
```

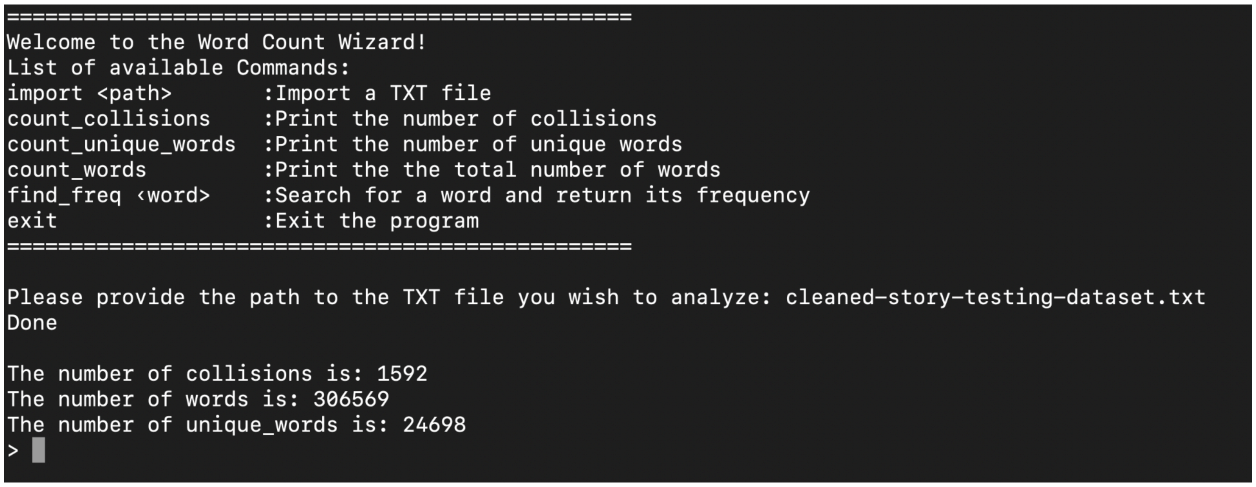
Implementation in Pseudocode:

```
Algorithm linear_modified_hash(key)
  hash ← 1  { Start with 1 to avoid multiplication by 0 }
  for i ← 0 to key.size − 1 do
    hash ← hash * ASCII value of key[i]
  return hash
```

Implementation in C++:

```
unsigned long hash = 1;
for (size_t i = 0; i < key.size(); ++i)
    {
      hash *= key[i]; // multiply the ascii value of each character in the hashcode
    }
```

5. Alternating linear ASCII sum and Polynomial hashing
   (I made a combination of linear sum and polynomial hashing alternating between different characters of a given word)



```
================================================
Welcome to the Word Count Wizard!
List of available Commands:
import <path>       :Import a TXT file
count_collisions    :Print the number of collisions
count_unique_words  :Print the number of unique words
count_words         :Print the the total number of words
find_freq <word>    :Search for a word and return its frequency
exit                :Exit the program
================================================

Please provide the path to the TXT file you wish to analyze: cleaned-story-testing-dataset.txt
Done

The number of collisions is: 1592
The number of words is: 306569
The number of unique_words is: 24698
> 
```

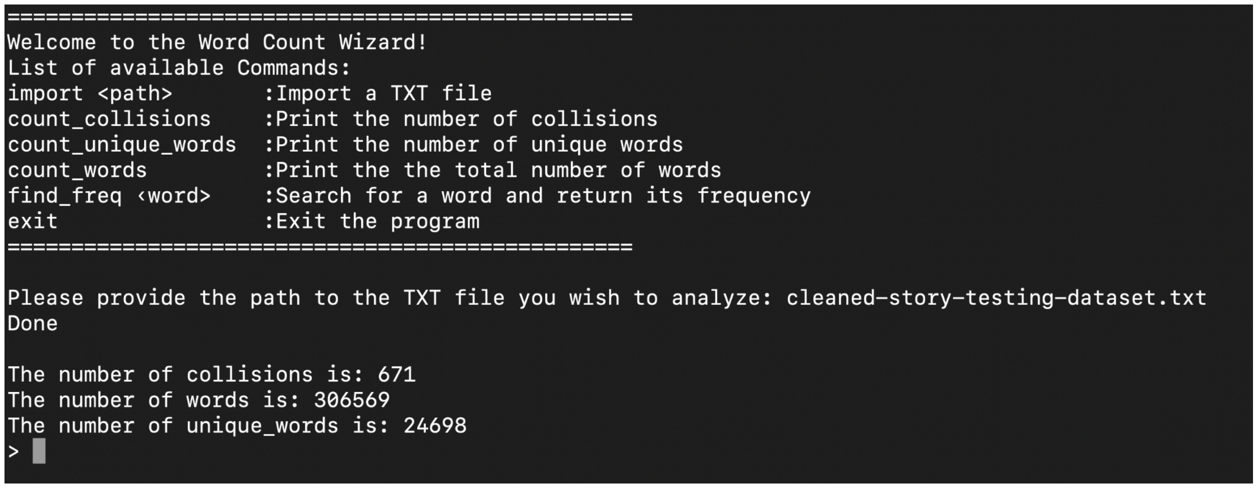
Implementation in Pseudocode:

```
Algorithm alternating_sum_polynomial_hash(key)
  hash ← 0
  alternator ← true
  p ← 31  { Polynomial base }
  m ← capacity  { Modulus based on hash table size }
  for i ← 0 to key.size − 1 do
    if alternator = true then
      hash ← hash + ASCII value of key[i]
    else
      hash ← (hash * p + ASCII value of key[i]) mod m
    alternator ← NOT alternator
  return hash
```

Implementation in C++:

```
bool alternator = true; // boolean for alternations
    for (size_t i = 0; i < key.size(); ++i)
    {
        if (alternator)
        {
            hash += key[i];  // Add the ASCII value of the character
        }
        else
        {
            unsigned int p = 31; // polynomial hash function base
            unsigned int m = capacity; // use table capacity for modulus operator
            hash = (hash * p + key[i]) % m;  // Subtract the ASCII value of the character
        }
        alternator = !alternator;  // Alternate between addition and polynomial hashing
    }
```

6. Alternating polynomial hashing and cycle shift hashing
   (I made a combination of cycle shift and polynomial hashing alternating between different characters of a given word)



```
================================================
Welcome to the Word Count Wizard!
List of available Commands:
import <path>       :Import a TXT file
count_collisions    :Print the number of collisions
count_unique_words  :Print the number of unique words
count_words         :Print the the total number of words
find_freq <word>    :Search for a word and return its frequency
exit                :Exit the program
================================================

Please provide the path to the TXT file you wish to analyze: cleaned-story-testing-dataset.txt
Done

The number of collisions is: 671
The number of words is: 306569
The number of unique_words is: 24698
> 
```

Implementation in Pseudocode:

```
Algorithm alternating_polynomial_cycle_shift_hash(key)
  hash ← 0
  alternator ← true
  p ← 31  { Polynomial base }
  m ← capacity  { Modulus based on hash table size }
  for i ← 0 to key.size − 1 do
    if alternator = true then
      hash ← (hash * p + ASCII value of key[i]) mod m
    else
      hash ← (hash << 5) OR (hash >> 27)  { Left rotate by 5 bits }
      hash ← hash XOR ASCII value of key[i]
    alternator ← NOT alternator
  return hash
```

Implementation in C++:

```
bool alternator = true; // boolean for alternations
    for (size_t i = 0; i < key.size(); ++i)
    {
      if (alternator)
      {
        unsigned int p = 31; // polynomial hash function base
        unsigned int m = capacity; // use table capacity for modulus operator
        hash = (hash * p + key[i]) % m;  // Subtract the ASCII value of the character
      }
      else
      {
            hash = (hash << 5) | (hash >> (27)); // left rotate by 5 bits
            hash ^= key[i]; // XOR with the current character
        }
        alternator = ! alternator;  // Alternate between polynomial hashing and cycle shift
    }
```

Choice of Hash Function

My default hash function is the second function that is Polynomial hashing because it has the least number of collisions (655) that makes it to have the smallest collisions per words ratio.