CS 2420 Program 4 – 20 points
Spring 2018

Who says I can't write poetry?

In this assignment, you will fit a bi-word language model to English and then use it to generate a poem. A uni-word model of English consists of a single probability distribution ***P(W)*** over the set of all words.

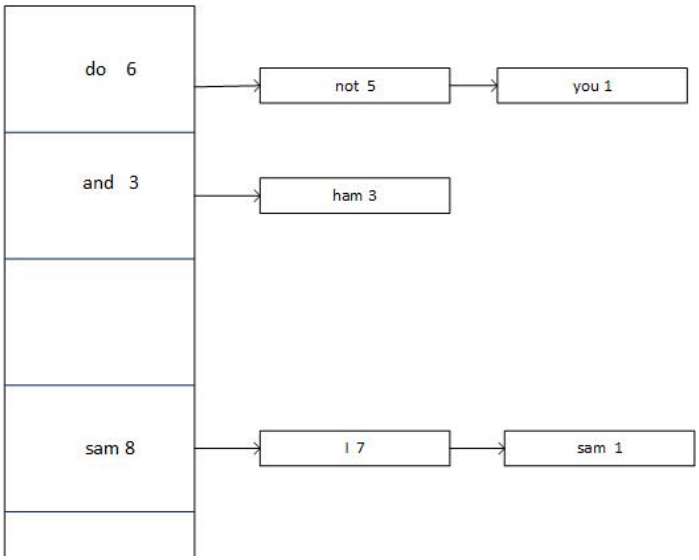
A bi-word model of English consists of two probability distributions: ***P(W)*** and ***P(W_i | W_{i-1})***. The first distribution is just the probability a word in a document. The second distribution is the probability of seeing word W_i given that the previous word was W_{i-1}.

Given a set of documents (in our case, various poems), your job in this assignment is to generate a poem from a given word. The method is simple. You just need to decide which word should follow a given word. The word selected is based on the probability in which it occurs (in the file) after the last word used. So for example, in the text below, if I ask, "What should follow 'sam'"?, you pick 'I' most of the time (7 out of 8 times) and 'sam' the rest of the time. No other word follows 'sam'.

I am Sam. I am Sam. Sam I Am.
That Sam I Am! That Sam I Am! I do not like that Sam I Am!
Do you like green eggs and ham?
I do not like them, Sam I Am.
I do not like green eggs and ham.
Would you like them here or there?
I would not like them here or there.
I would not like them anywhere.
I do not like green eggs and ham.
I do not like them, Sam I Am.

Thus you need to keep track of the number of times that each word follows every other word (so you can figure probabilities). I used a vector for this. A linked list would also work. You can use what you like.

Keep track of all words (and their probabilities) using a hash table in which you hash on word W. For the input above, partial contents of the hash table are shown below. Thus, "do" occurs 6 times in the file. Five times "do" is followed by "not" and one time it is followed by "you"



Similarly, for the input file, when you see the word "sam", seven times out of 8 "i" is the next word. One time out of 8, "sam" is the next work.

Given the information in this data structure, we can compute the conditional probability that "i" occurs once you are looking at word "sam" *P(i|sam)* as 7/8. Similarly, we can compute the probability *P(ham|and)* as 3/3 = 1. When I ask what should follow "sam", you will pick the next word to match the probabilities seen. In other words, 87.5% of the time you will want 'i' to come next (after sam).

Given a specific input file, you are to generate a poem of a given length from a specific starting word. So for example, poem("sam", 20) is to generate a poem of 20 words that begins with the word sam. Your poem might look like: "sam I am sam I am I am I would not like them here or there I am do not like" or "sam I would you like green eggs and ham I am do you like them sam I am I am I"

Specifics

You have been given the hash table code from your text for doing quadratic probing. Modify the code in at least two ways:

a. Use your own hash function, patterned after code we used in class.
b. Instead of quadratic probing, use double hashing.

Please retain the template structure of the hash table. Be careful not to add anything to the hash table that only makes sense for this specific problem.

For the data, remove all special characters and punctuation and change everything to lower case. My code to do that was as follows (but do it anyway you like):

```
void clean(std::string & nextToken)
{
	for (int i = 0; i < nextToken.length();)
	{
		if (nextToken[i] >255 || nextToken[i] < 0 || ispunct(nextToken[i]))
```

```
                nextToken.erase(i, 1);
            else {
                nextToken[i] = tolower(nextToken[i]);
                i++;
            }
        }

}
```

Output

There are several data files associated with the test. You will generate poems from the following specification:

File	Beginning word	Length of Poem	Print Hash Table?
green.txt	sam	20	Yes
clown.txt	go	20	Yes
inch.txt	computer	50	no
poe.txt	nevermore	50	no
seuss.txt	mordecai	50	no

For grading purposes, you are asked to print the contents of the hash table for some of the test cases. Because the next word is generated via a random number generator, you should not expect your output to match that of others.

Hints

In order to generate which word follows word WORD with the correct probability, I used the following strategy. I kept track of how many times WORD occurred (call it occurCt) and how many times each word followed WORD ($followCt_i$). I generated a random number between 0 and occurCt. Then I considered each word which followed WORD in order. I added up the followCts of each following word until the sum of the followCts passed the random number. When the sum passed the random number, that is the word I selected to be the next word in the poem.

I organized my data as shown below – but you can do anything you want.

```
HashTable<std::string, FirstWordInfo> wordFreq;


class FirstWordInfo
{
       std::string word;
       int count;
    public:
       std::vector<FollowingWordInfo> secondWordList;
       std::string toString();
       FirstWordInfo(std::string s, int c);
       void updateSecondWord(std::string w);
       void incrementCount(){ count++;}
};

class FollowingWordInfo
{
```

```
    public:
        std::string word;
        int count;
        std::string toString();
        FollowingWordInfo(std::string s, int c);
        void increment() {count++;  }
};
```

What to Turn In

Turn in your complete project file. In a readMe file, please write a one-paragraph analysis of the generated texts addressing the following points:

- What knowledge do human beings have that is NOT captured by this language model and that therefore causes the model to generate nonsense. Write down at least three facts that, if the computer could somehow know them, would allow it to generate more sensible language.
- Compare the different poems. If your experience is like mine, some will be more coherent than others. Speculate about why this is true.