Describe how you overcame each issue in the Challenges, particularly what you did for item #3.

Similarly, if you attempt the extra credit, explain what you did and how you did it.

## Majority Class/Unigram

To start, we used the majority class/unigram model the sequence. This was a simple matter of computing the most frequent symbol in the training set. Once this was done, the program would predict this symbol for everything in the testing set’s sequence.

## Bigram

The next step was to create to create a data structure for storing bigrams. We chose to use a hash-map for this process. The hash-map stores the current letter as a key and hash-map as the value. The value hash-map stores the next possible letters as keys and the number of occurrences of concatenated string as the value.

Figure - the hash-map data structure for bigrams on the string abbabcabbac

b

c

3

1

a

c

2

2

1

a

1

a

c

b

a

1

The next step was to implemented *ngrams*. That is, given any number n, our program would generate markov chains of that length from the training data, and apply them to the test data. This was a simple matter of scaling up the system from the previous step to store keys of *n-length* instead of only one.

## Further Improvements

-adds stuff from test into the hash-map

-backs off instead of making a random guess

-waited markov chain length length

# Accuracy

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm\Accuracy | **Trial 1** | **Trial 2** | **Trial 3** | **Trial 4** | **Trial 5** |
| **Unigram** |  |  |  |  |  |
| **Bigram** |  |  |  |  |  |
| **Simple Backoff** |  |  |  |  |  |
| **Weighted Markov** |  |  |  |  |  |

# How to Run

Pasty pasty:

a

b

c

a

c

a

b

c

a

c

b

a

b

c