William Clift
Data Structures
21 February 2020

Assignment 1 - Efficient Ngrams with Queues

Concepts:

For this assignment, we have an object class `ngramizer` that handles all of the algorithmic procedures to produce the ngrams. Using stdin to receive the sentence and number of grams (`n`), an object is created to process the sentence. `ngrams()` uses a queue of size `n` and iterates over each word of the sentence, pushing and popping words as it prints. For the `ngrams_backoff()` , the method `ngrams` is called starting at `n` and decrementing until it reaches 1, where it returns each of the ngrams n, n-1, to 1. Finally, the method returns and a `List<String>` is created to store each line of the ngrams breakdown. Using stdout, this is printed to the screen or to a file.

Implementation:

By choosing to create a multi-class program, we decrease the coupling. Our main class `ngrams` handles all of the input and output and creating of objects. Once the number of n-grams is sent as a command line argument to the program at runtime, an object is declared which requests a sentence and inputs it through stdin from a file or from keyboard interaction. At this point, the sentence as a String and number of n-grams is sent to the object constructor and stored in private variables.
The object `ngramizer` has three methods, `words` , `ngrams` and `ngrams_backoff` . The `words` method begins with the `split()` String method that takes the sentence and parses it into an array of each word (using " " as a delimiter). This puts the sentence into an easily accessible data structure so that we can process it in the following algorithm. In the `ngrams` main class, we create an object of type `ngramizer` and call the method `ngrams_backoff` on it. This uses the sentence array (`words[]`) and loads up a queue of size `n` (`Queue<String>` , using the `LinkedList` implementation) . We iterate through the queue to get the first ngram of size `n` . Then, we pop the first element and push the next element from `words[]` onto `q` .

Time Complexity:

`S` is the number of words, `N` is the number for n-grams.

Analytically --

`ngrams:` We use a nested `for` loop with the outter loop iterating `S-N-1` times and the inner loop is iterating over the Queue `N` times. Since `S` is sufficiently larger than `N` , we consider `S` as the scaling variable and thus, the algorithm is linear.

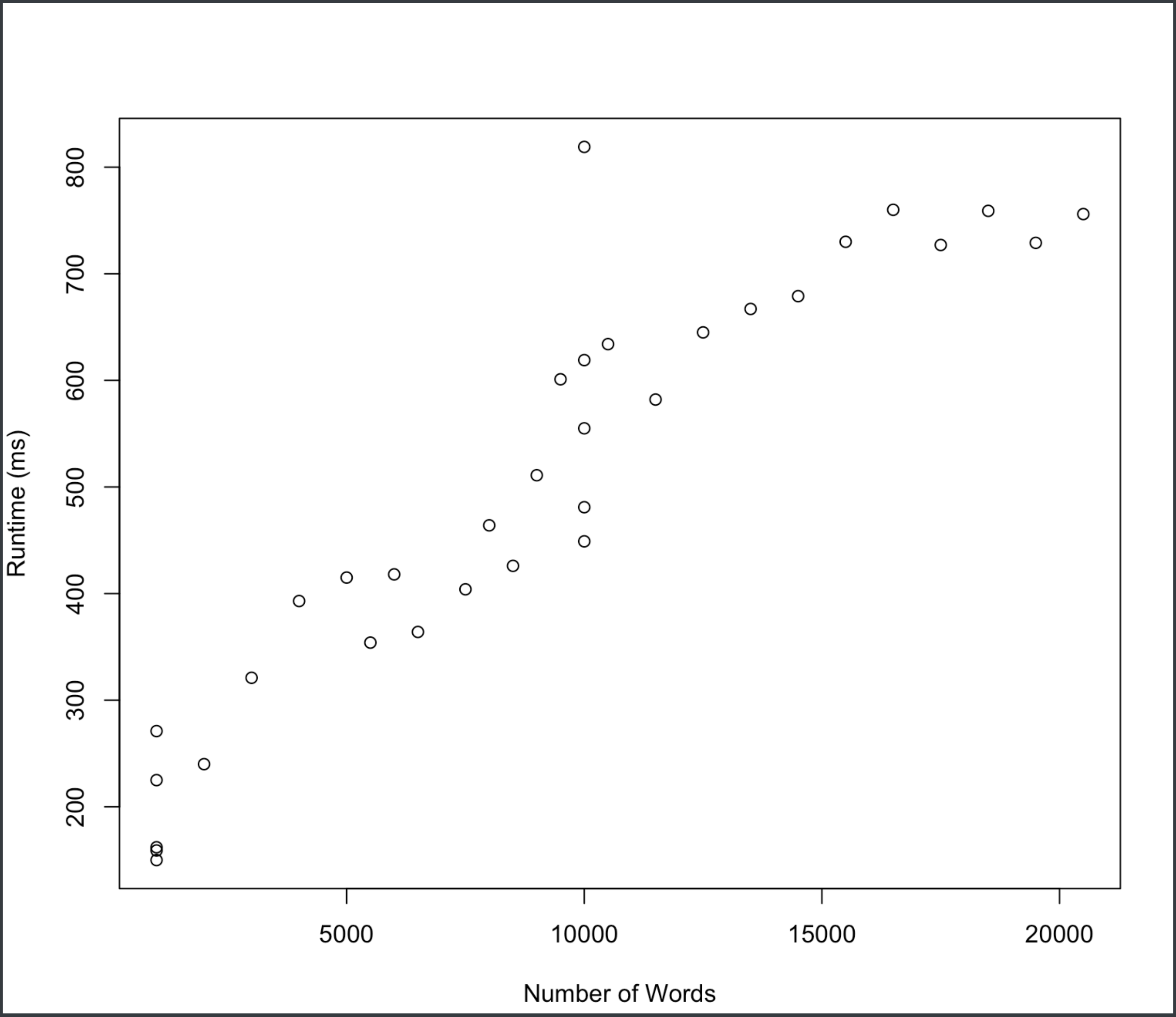
```
$O(N)$
```

`ngram_backoff:` Now, we use a while loop from `n-1` to `1` calling `ngrams()` each time. That gives us a complexity the same as `ngrams` and multiplies by the `n-1`, but the number of grams is very small with respect to the number of words, especially as it scales. So, the algorithm will still be linear.

```
$O(N)$
```

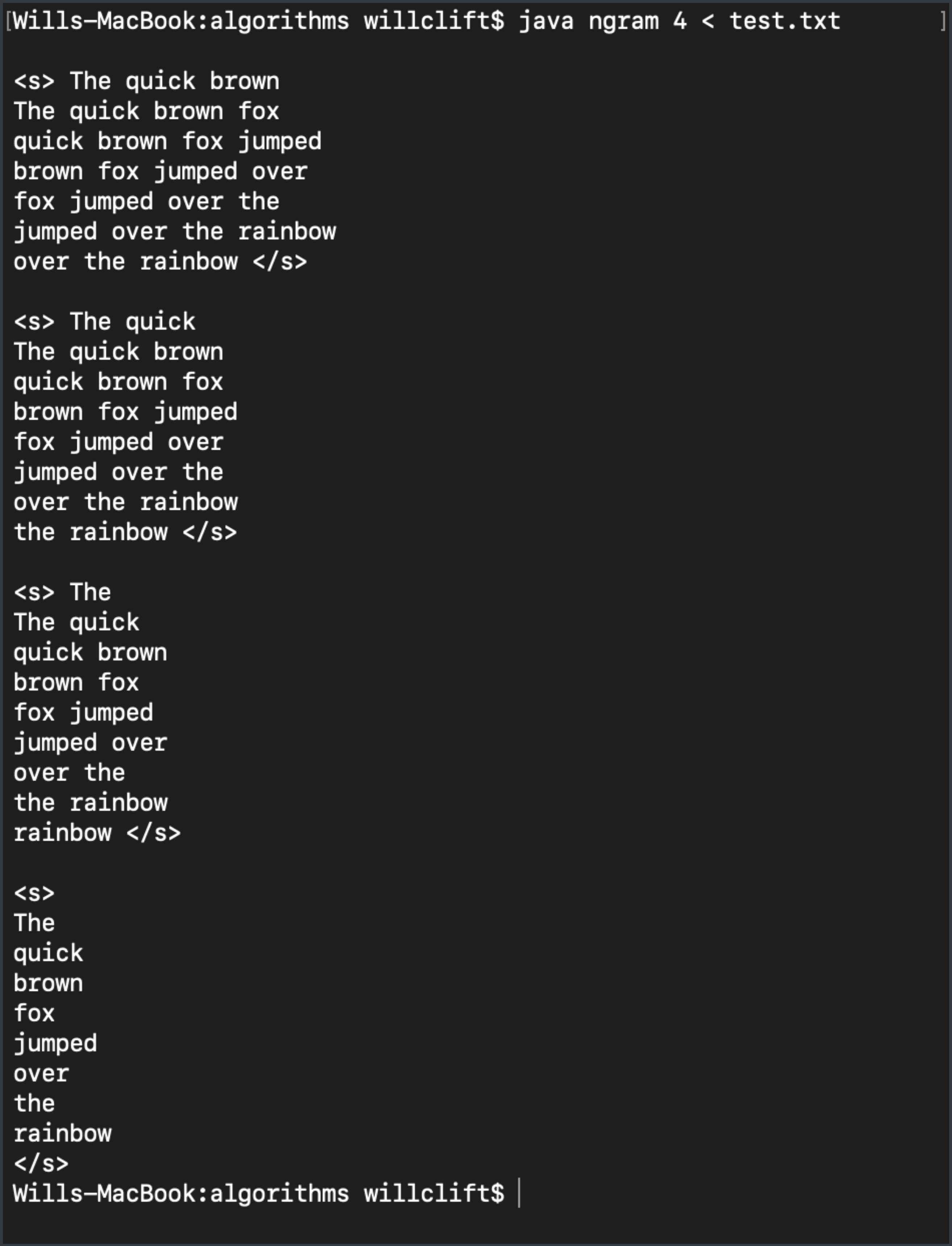
Empirically --

`ngrams_backoff`



This graph schows the runtime vs number of words for the program on `ngrams_backup` shows how this algorithm scales as the number of words gets really large. This graph gives us that the algorithm is linear because of its tendency to stay close to one line as N increases.

Here is an example of the output format on the command line.



```
[Wills-MacBook:algorithms willclift$ java ngram 4 < test.txt                ]

<s> The quick brown
The quick brown fox
quick brown fox jumped
brown fox jumped over
fox jumped over the
jumped over the rainbow
over the rainbow </s>

<s> The quick
The quick brown
quick brown fox
brown fox jumped
fox jumped over
jumped over the
over the rainbow
the rainbow </s>

<s> The
The quick
quick brown
brown fox
fox jumped
jumped over
over the
the rainbow
rainbow </s>

<s>
The
quick
brown
fox
jumped
over
the
rainbow
</s>
Wills-MacBook:algorithms willclift$ 
```