CSCE 350 Program 2 – Analysis

Naïve Versus Sophisticated String Matching

Text Analyzed

- Dickens: *A Tale of Two Cities*
- First 100 thousand digits of π
- DNA

Algorithms and Descriptions

Naïve String Matching

```
For each character in the text
    Match is true if the first character in the pattern matches the text character
    While Match is true
        Test each character in the pattern against the corresponding text character
    If Match is still true, allocate the match
```

Simple Index String Matching

1.) Map the text's character frequencies
2.) Index the text's characters by creating a location vector for each
3.) Begin the Match algorithm:

```
For each pattern
    Use the frequency map to find the lowest frequency character in the pattern
        Mark the location of that character within the pattern
    Use the character location index to associate a location vector

    Use the location vector and the position of the least frequent character within the pattern as a
      position to begin string comparison by character
```

Note for Analysis

When analyzing the algorithms, for the Naïve algorithm, “N” was considered the length of the text to be searched.

However, with the indexed algorithm, “N” is considered to be the length of the text to be searched **divided by the number of patterns searched for**.

Analysis

Of course, the note above implies that as the number of search patterns increase, the more useful the algorithm becomes. It's also clear that the algorithm could be more efficient by creating a stronger index, i.e. instead of basing the frequency and index on one character, two would be stronger, and improve the results. Another example might create an index based on the sum of the ascii character values for up to a specific number of string lengths. For example the string “11” could have the index key of 98, and the string “dd” would have the index key of 200. This would offer a minimal number of observable string comparisons, however with a large number of string lengths, the index could become quite large, and a stronger argument could be made for using “keywords.”

Here, the size of the index is confined to the character set used within the text. This implies pi would have an index of length 11 (including the period). It also implies that the Dickens text has a stronger index by default simply because it uses a larger character set. The effect of this would be doubled if the frequency / index was created for two characters. It should also be noted that the DNA character set is of length 4, and it thereby has the most useless index in this design.

It's understood that the worst case scenario for each search is text_length * pattern_length, but I didn't find this information much use while comparing these two algorithms. The summary and additional data below supports my claims. It should be noted that the data is slightly skewed because the amount of queries and the lengths input texts are not equivalent between tests.

Data Summary

Test	**Index Efficiency**
Dickens	93.44%
Pi	74.49%
DNA	52.89%

Data

Dickens	naïve	indexed	Efficiency
	793045	42152	94.68%
	793139	47822	93.97%
	753150	57132	92.41%
	758915	68374	90.99%
	771349	42468	94.49%
	818033	49336	93.97%
	756645	47269	93.75%
	754682	62789	91.68%
	742999	38498	94.82%
	784195	45980	94.14%
	820598	53060	93.53%
	797234	52700	93.39%
	792035	55024	93.05%
	834539	53594	93.58%
	791933	54922	93.06%
	761717	51674	93.22%
	783469	48239	93.84%
	828824	54425	93.43%
	796836	52972	93.35%
	783627	50761	93.52%

pi	naïve	indexed	efficiency
	23032	4900	78.73%
	23067	6934	69.94%
	23011	4809	79.10%
	23065	4831	79.05%
	23022	6889	70.08%
	23041	6908	70.02%
	23043	6910	70.01%
	22988	4831	78.98%

DNA	naïve	indexed	efficiency
	188	91	51.60%
	142	44	69.01%
	134	83	38.06%