COMP90049 Project2 Report

Qingyang Hong 629379

October 23, 2013

1 Introduction

In this project 2, the task focuses on approximate string matching, which is the technique to find strings that match a pattern approximately. With widespread use of Internet technology in the fast-tempo society, the demand for efficient online applications is growing day by day. More applications in computers are starting to be fed in high efficiency features by building on knowledge of Information retrieval, artificial intelligence and interactive processing. One point in the trend is to search similar words to make use of them in applications, like word processor error correction, email spam filtering, electronic dictionary spelling assistant, or search engine. The basic methodology is to find the nearest match to a string, and by a brutal-force way, a program may compare two strings character by character to calculate the "distance", which is obviously infeasible. While by loading text in a specific better structure and using some computational method, the problem can be solved within reasonable time and space constraints.

Firstly, the report will introduce some related work on the problem. Secondly it will describe the two basic methods, the "Trie" and the "Edit Distance", which are the essence in the experiments to handle the matching task. Thirdly, the report will present how the two methods are practically performed and the results will be gathered for analysis. Related explanation on the results will be about preliminary observations. Critical evaluation and comparison between two methods will be included in the discussion section, where the related analysis will also be given. Finally, a summary concludes all the achievements in the experiments and some future work will be derived.

2 Related Work on Approximate String Matching

Related work has started since 1970s, or even earlier from conceptual perspective. Basically, solutions to the problem can be divided into two categories, which are on-line method and off-line method. On-line method offers a pre-processed pattern for searching without indexes. The main idea of the method is to use dynamic programming to calculate the distance between words. On of the famous on-line method is the bitap algorithm, which performs well in short string search. With the development of computer technology and the growth of data, faster search needs more indexing and structured dataset. The indexing methods, such as suffix tree, n-gram are more likely to perform well in the practical context. By using them, any string search in a big text file can be achieved within decent time constraint. Generally speaking, both of the methods have their features, and they have utilities under different circumstances.

3 Methodology and Results

In our experiments, two kinds of algorithms are chosen to represent indexing and non-indexing approximate string matching methods, which are the "Trie" and the "Edit Distance". A trie is a tree data structure with strings in each node and is used to store a dynamic structure string array. The whole tree serves as a dictionary, expanding by adding new nodes to it. By following the edge from a node to another, a string can be produced character by character. As to cater to the approximate matching feature, the algorithm may allow slight mismatching during the traversal. As to the edit- distance, it also refers to the Levenshtein distance. The algorithm calculates difference between strings by operating insertions, deletions, replacements which have such cost treated as distance. The nearer two strings are, the more similar they are.

3.1 Trie and Edit-distance

In the practice, two kinds of trie algorithms are introduced. At the beginning, the program prepares a data structure, Node, which contains a character key and the list of the children, storing in a HashMap. The trie will be built on raw text(turgenev.txt), which should be processed to remove all non-alphabetic characters, other than spaces and converted to lower-case. The alphabet should be from a to z and a space character. A query is from the surname.txt. With prefixQuery, the search will try to find a node with the first character of searching string in the trie and search the next character in the children of the node, until nothing can be found or a matching success. With mismatchQuery, basic processes are the same with prefixQuery, but it allows mismatching. When encountering a mismatch between a character in string and a node, what to do is to hop to the next character to keep searching. It should be noted that times of mismatching should be recorded and fail if a search has excessive mismatching.

As to the edit-distance, the implementation is based on a distance matrix, which records the distance on each character between two strings. Every operations that change a character to another has cost. Usually, a matching costs nothing, but a mismatching, a deletion or a insertion costs 1. Intuitively, in a distance matrix, dmatrix[q.length][t.length] represents the minimum operations that can change string q to l. Like the mismatching in trie, the distance is also limited.

3.2 Results and Explanation

The first set of experiments are on two sets of data, the first of which has 1000 query strings and 10000 words in dictionary, and the second of which has the whole surname data set and preprocessed turgenev data set. The main point is to compare time complexity by recording the programs' running time. Intuitively, the trie method can process both the data sets and finish within decent time, while the editdistance method fail on the bigger data set and performs poorly on the small data set comparing to the trie method. Besides, prefix trie runs faster than mismatch trie on both data sets.(see Table 1).

Methods	1000/10000	31918(WholeQ)/74241(WholeDict)
Trie(mismatch)	573ms	1330ms
Trie(prefix)	386ms	1063ms
EditDistance	23259ms	N/A

Table 1: Algorithm Time cost

The second set of experiments are on the smaller data set so as to let both trie and editdistance finish the searching. The main point is to see the matching performance of the three methods. Both mismatch trie and editdistance succeed with two mismatchs. In term of editdistance, the query and match are seen as "aaby" and "ma_y". The prefix trie fails to match.(see Table 2).

Methods	Query	Match
Trie(mismatch)	aaby	away
Trie(prefix)	aaby	N/A
EditDistance	aaby	may

Table 2: Matching Comparison

4 Discussion

As seen from the first set of experiments, the trie is quite efficient, comparing to the edit distance. According to the related work mentioned above, it can be figured out that the trie is a tree data structure, which belongs to the so-called off-line method. The main point is that the data structure in edit distance is not indexed, but the trie has indexed every words from dictionary. Generally, the trie could have O(n) in time complexity and the edit distance could be O(n^2). Moreover, in terms of trie, prefix trie performs better in time, for it doesn't need to consider mismatch, but does straightforward matching.

Moving to the second set of experiments, editdistance could have more freedom in searching, for is has the concepts of deletion, insertion, replacement but not only matching. The prefix trie should be less free, for it can only move forward, but can't hop over one character as mismatch trie, which results in less matching success.

5 Conclusions

The experiments and analysis have generally explained the conceptions and features of both the approximate string matching methods. By going through the results and analysis, it can be seen that both the two methods have their utilities. The trie is much more efficient, and the edit distance has more freedom when comparing two single strings.

In the future work, one important task should be done is that the last node, which is reached in a query, should point back to the base to see how many indels are there before the start node of the query. Another task is to find better data structure for edit distance on efficiency. Both the considerations can improve the methods a lot.