Knowledge technology project 2

Name: Albert Wang

ID : 331793

**1 Introduction**

The system is built to accomplish the task of returning a best-matched string given a particular query. String search methods used in this system are Trie and Smith Waterman algorithm. The system is implemented by Python.

The text that will be searched from is preprocessed. All non-alphabetic characters, apart from space, are removed. And the file is tokenized.

A trie is then built upon the processed file.

**2 Implementation**

**2.0 Trie**

A multi level dictionary structure is used to build the trie. For example: a possible trie would be { ‘a’ : { ‘ac’: { ‘ace’:{} } ,

‘ad’: {} },

‘b’:{} ……. }.

Empty dictionary indicates an end of a particular branch. The system will try to find a match first. If this fails, the system will try to find the most match string.

In the system, the memory used by the trie is 3352 bytes.

**2.1 Edit Distance**

There are two main kinds of edit distance, local distance-Smith-Waterman algorithm, global distance-Needleman-Wunsch alogrithm.

Based on well-known Smith Waterman algorithm, a simple local edit distance function is implemented.

Score scheme

Match = +2

Deletion = Mismatch = Insertion = -1.

**3 Discussion**

**3.1 Result**

Searching by the trie structure is considerably faster than edit distance.

Trie:

ablan: result: able score: 2 time: 2.8133392334e-05

ables: result: able score: 1 time: 2.28881835938e-05

ablett: result: able score: 2 time: 2.50339508057e-05

abley: result: able score: 1 time: 2.31266021729e-05

abner: result: NO score: 2 time: 0.113157987595

aboba: result: abou score: 2 time: 3.00407409668e-05

abolitz: result: NO score: 2 time: 0.0883100032806

abraham: result: NO score: 2 time: 0.08154296875

Edit distance:

aaby : result : baby : score : 6 time: 2.60237598419

aachen : result : headacheshe : score : 9 time: 3.7491850853

aagaard : result : haggard : score : 9 time: 4.33059406281

aagesen : result : management : score : 10 time: 4.35487985611

aakerstreom : result : shoemakers : score : 10 time: 6.65360689163

aalto : result : altogether : score : 8 time: 3.15584111214

aam : result : himselfadame : score : 5 time: 2.00588583946

aamer : result : steamer : score : 8 time: 3.40172100067

aamoth : result : mother : score : 8 time: 3.78930401802

aanderud : result : wandered : score : 11 time: 4.94389986992

aanerud : result : wandered : score : 8 time: 4.39955496788

**3.2 Performance**

Space and time

Trie: To be specific, for this particular testing data, 3352 bytes are used to store the trie.

There are two possible ways to build the trie from the text file, one is to read the file as a whole string and build trie from it, another one is to tokenize it first and then build tire from a number of words.

In this system, I chose the second way which is proprossing the file into words. The time complexity for building the trie is, let n be the number of words and m be the number of letters in a word, the complexity is O(mn). Yet the time complexity for the other way of building trie is better, and it could be O(m).

There are other difference between those two ways of building trie.

Firstly, without tokenization, a query containing a space could have a exact match in the trie. Secondly, any adjacent letters could be a search result. For example, “aar” is a node in the trie and it could be a valid result, whereas, with tokenization, only the words at last level are valid. So it will miss some queries. However, the result returned in this case makes more sense since it could be more likely to be a really word. So it really depends on what are the requirements or what kind of search is needed.

As for the cost of space, the tire structure would require massive space to store all the nodes. Space is needed store the value of each node and also the children nodes of it.

Comparison cost:

Let q be the length of query, n be the number of nodes in the trie,

Best case, that is there is a match in the tire, the complexity is O(q) due to the nature of the trie.

Yet for worst case, that is there is no exact match in the trie, the complexity is O(qn).

Local edit distance:

As for time complexity, it is similar to a handshake problem that is each person needs to shake hands with every other in the room.

A handshake problem has the time complexity of O(n2).

Furthermore, as for space complexity, a two dimensional array or other similar structure is created for each comparision, so the space complexity is cn2.

So the complexity for both time and space is O(n2).

**3.2 Comparison**

It is always a trade of between space and time, when measure the merits of algorithms. The trie structure tends to use more space yet offering a fast search.

Edit distance uses less space yet more time to return a best match. Trie is not good at approximate string matching since it uses exhaustive search for this case.

Due to their strength and weakness, they are suitable for different purposes. And they could be used together to get a best result. For example, we could know there must be a match, then we could use trie. Otherwise, we could use edit distance to get a most-matched result. And if we use them together, handling a query by trie first and then edit distance, best performance could be achieved.

**Conclusion**