Habib University
CS 201 Data Structures II
Spring 2018

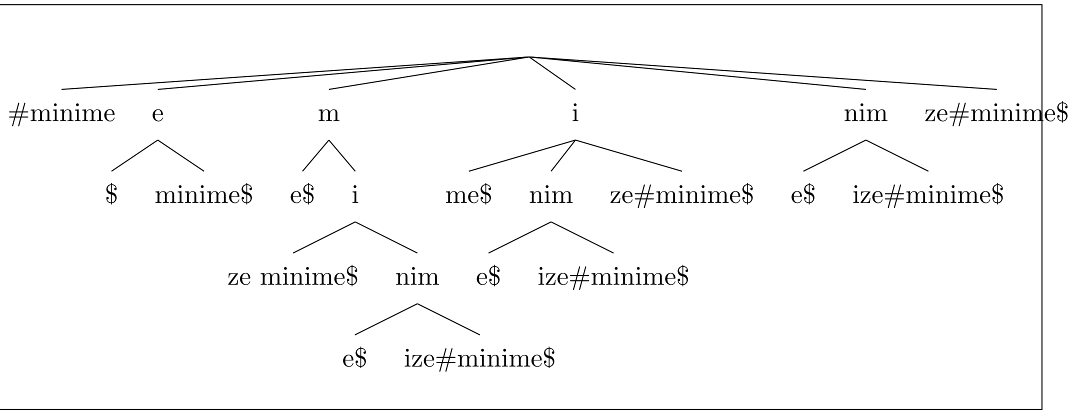
Emad Bin Abid - Saman Gaziani

$Team : hw - 4 - ea02893 - sg02494 - 4$

Homework 4
Submitted: March 12$^{\text{th}}$, 2018

1. 10 points Draw the compact representation of the suffix trie for the string: “minimize minime”.

Solution:



2. 10 points Give an efficient algorithm for deleting a string from a standard trie and analyze its running time.

Solution:
Algorithm 0.1: DELETE(T, S)

```
comment: Delete a string, s, from a standard trie, T, in O(n) time, supposing that we have
already searched for the word.

The idea of this algorithm is taken from geeksforgeeks.  The pseudocode is
however self-generated.

procedure DELETE_S_FROM_T(
)
 if (s not found)
   then
 return False
 u = end of s
 while (u.child≠ 0)
   do
 v = u
 u = u.parent
 delete v
```

3. 10 points Describe an algorithm for constructing the compact representation of a suffix trie, given its noncompact representation, and analyze its running time.

Solution:
To construct compact representation of a suffix trie, we will iterate through every leaf node, and check if it has any siblings.
If the node has sibling, the pointer will move towards the paretn node without any changes in the child node. But if it has no siblings, the node will combine with the parent node and will be treated as a single node now. The pointer will remain at the same node.
This procedure will be replicated till all the nodes are covered by the pointer.
Running time of the above algorithm is $O(n^2)$

The following questions refer to the 2 tables below.

term	df_t	idf_t
car	18165	1.65
auto	6723	2.08
insurance	19241	1.62
best	25235	1.5

Table 1

	Doc 1	Doc 2	Doc 3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

Table 2

4. 10 points Consider the table of term frequencies for 3 documents denoted Doc1, Doc2, Doc3 in Table 2. Compute the tf-idf weights for the terms `car, auto, insurance, best`, for each document, using the idf values from Table 1.

Solution:

	Doc 1	Doc 2	Doc 3
car	44.55	6.6	39.6
auto	6.24	68.64	0
insurance	0	53.46	46.98
best	21	0	25.5

5. $\boxed{\text{10 points}}$ Compute the Euclidean normalized document vectors for each of the documents in Table 2, where each vector has four components, one for each of the four terms.

Solution:
Doc1 = (44.55, 6.24, 0, 21)
$|Doc1| = \sqrt{44.55^2 + 6.24^2 + 0^2 + 21^2}$
= 49.65
$\frac{Doc1}{|Doc1|} = (\frac{44.55}{49.65}, \frac{6.24}{49.65}, \frac{0}{49.65}, \frac{21}{49.65})$
$\frac{Doc1}{|Doc1|} = (0.89, 0.12, 0, 0.42)$

Doc2 = (6.6, 68.64, 53.46, 0)
$|Doc2| = \sqrt{6.6^2 + 68.64^2 + 53.46^2 + 0^2}$
= 87.25
$\frac{Doc2}{|Doc2|} = (\frac{6.6}{87.25}, \frac{68.64}{87.25}, \frac{53.46}{87.25}, \frac{0}{87.25})$
$\frac{Doc2}{|Doc2|} = (0.07, 0.78, 0.52, 0)$

Doc3 = (39.6, 0, 46.98, 25.5)
$|Doc3| = \sqrt{39.6^2 + 0^2 + 46.98^2 + 25.5^2}$
= 68.67
$\frac{Doc3}{|Doc3|} = (\frac{39.6}{68.67}, \frac{0}{68.67}, \frac{46.98}{68.67}, \frac{25.5}{68.67})$
$\frac{Doc3}{|Doc3|} = (0.57, 0, 0.72, 0.37)$

6. With term weights as computed in the Question 5, rank the three documents from Table 2 by computed score for the query `car insurance`, for each of the following cases of term weighting in the query.

   (a) $\boxed{\text{5 points}}$ The weight of a term is 1 if present in the query, 0 otherwise.

   > **Solution:** q = [1, 0, 1, 0]
   > score(q,Doc1) = 0.897
   > score(q,Doc2) = 0.688
   > score(q,Doc3) = 1.302
   > **Ranking:** Doc3, Doc1, Doc2

   (b) $\boxed{\text{5 points}}$ Euclidean normalized idf.

   > **Solution:** q = [0.4778, 0.6024, 0.4692, 0.4344]
   > score(q,Doc1) = 0.6883
   > score(q,Doc2) = 0.7975
   > score(q,Doc3) = 0.7823
   > **Ranking:** Doc2, Doc3, Doc1

Programming Questions [0 points]

The python skeleton files for each of the folloiwng will be added shortly.

(a) Code a class to represent a generalized suffix tree. Words are added to it one at a time and it supports the usual query functions. Test it on a sample word list, e.g. the one at `http://thinkpython2.com/code/words.txt`.

(b) Code a class to represent an invertex index. Documents are added to it one at a time and it supports the usual query functions. Test it on a sample corpus, e.g. Reuters RCV1.