CMPS 130A
Hybrid Trie with Counts
Ankur Jain
(6234462)
May 23, 2003

Abstract
The project involves implementation of Hybrid Trie with Counts data structure. The data structure can store words in only small letters. It takes in values of two parameters AR2LL and LL2AR as input and changes the internal representation depending on the number of different letters stored in a Node. The implementation has been done in C++ and the performance results and analysis are presented to evaluate the performance of the data structure under different conditions.

1 Implementation and Results

The program first takes in two parameters, the first parameter is the value of AR2LL and the second parameter LL2AR. It then waits for the user command and perform the operations accordingly assuming no errors in the user input. Almost all the functions implemented in the code are recursive to make the code more structural and simple. The testing of the implementation was done using the test-files provided by the TA and some other testfile. The program was found to be working correctly under different conditions. The most critical operation were those which lead to change of the representation of the Trie. These operations include the Shrinkit and Replace operation in addition to Insert and the Delete operation. The program was verified to maintain correct representation for small examples, however testing for large examples is difficult.

1.1 The DataSet

The testing of the program was done on a set of 90000 words which included words of varying lengths however the longest word inserted was of length 19 and the shortest was of length 1. There were no duplicate words in the dataset as the program was tested on the insertion operation on duplicated words was tested separately. The Dataset contained words starting with different letters from a to z. However since the words are dictionary words there were more words with a than with z.

1.2 Results

Below are presented the results obtained for the above mentioned dataset. Most of the results were obtained for the Insert operation. The Membership operation was also tested. Testing membership for time is similar to testing the Print operation and is hence not tested. The testing of Membership operation involved inserting all the words and then testing for membership for all of them. Figure 2 shows the times taken for the Insert operation by the two representations. It is seen that the time taken for insertion less for pure array node. This result is obvious as accessing the nodes directly using the array index is faster also there is no node allocation
in the array representation which happens in linked allocation every time a new character is inserted. Also
insertion in the linked representation takes more time if the character to be added in the node is already there.
In this case we have to traverse the linked list completely to check whether the entry for the character is already
there. In case of array representation entry for a character in a node can be tested directly. Figure 1 shows the
times taken by the membership operation. The time taken for this operation is less for array representation in
general. For small number of words the time taken is same for both representations however for large data sets
array representation show better performance. This is due to the fact that array nodes can be accessed directly
whereas in linked representation the testing for the presence of a character takes more time.

Figure memusage shows the comparative memory usage for both the representations. It is seen that the
memory requirements are comparable for small number or words. However the memory taken by the array
node representation is much higher than using the linked representation. This is due to the fact the array
representation allocates space for every character whereas in linked representation the space is allocated only
for required characters. The figure does not show exact memory requirements, the values include memory taken
by the process itself which is a constant value and does not effect the trend of the curve.
Figure 4: Time taken by the insert operation on varying the AR2LL value. The above results were obtained in inserting 90000 words with LL2AR = 25

Figure 5: Time taken by the insert operation on varying the LL2AR value. The above results were obtained in inserting 90000 words with AR2LL = 8

Figure 5 and Figure 4 show the variance of time using different representations for the insert operation. The values for the time do not show much variation though. It is seen that keeping the value of LL2AR constant, value of AR2LL around 12 shows good results. Similarly value of LL2AR around 18 shows good results. This for optimum performance these value would give better performance.

1.3 Conclusions

The code for the Hybrid Trie representation is implemented and tested successfully. It is seen that pure array node representation gives much better performance in terms of time than linked representation. However this speed up in perform ace is at the cost of high memory requirements for the array node representation. Choosing a value of around 12 for AR2LL and a value of around 18 gives better performance.