

STRING INDEX
Assignment 3
Summer Research Intern NUS

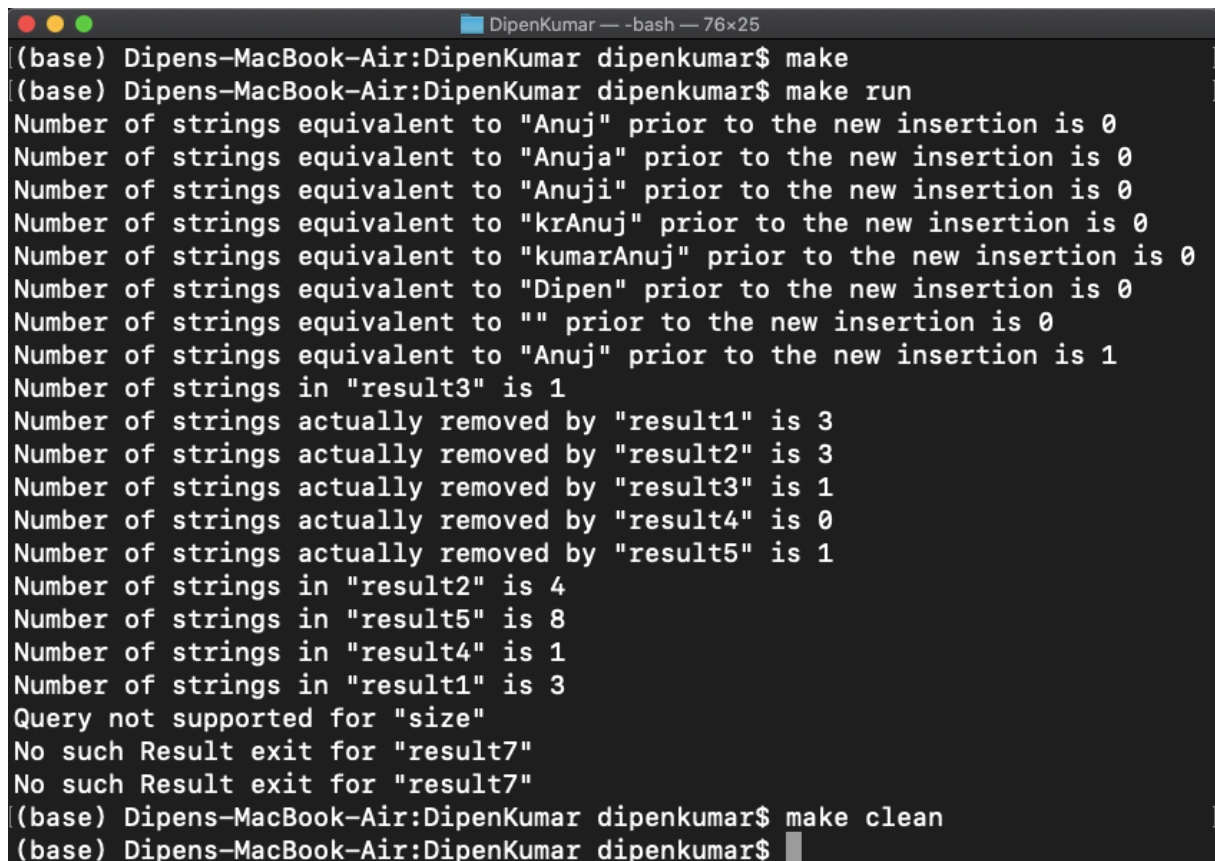
Dipen Kumar
Third-year undergraduate
Computer Science & Engineering
Indian Institute of Technology Delhi

Submitted directory contains following –

1. stridx.java
2. input.txt
3. Makefile
4. README

How to run the program –

1. Replace the sample input file i.e. “input.txt” with the file on which we are interested to test the program, keeping the filename same as “input.txt”.
2. There is a specific format for the input file i.e. “input.txt” which is explained in next section.
3. Open the terminal and run “make” command which will create the class files.
4. Next execute “make run” which will run the executable with one command line argument that is the input file name which in this case is “input.txt”.
5. Finally “make clean” command will delete all the class files keeping the directory clean.
6. Example –



```
DipenKumar — -bash — 76x25
(base) Dipens-MacBook-Air:DipenKumar dipenkumar$ make
(base) Dipens-MacBook-Air:DipenKumar dipenkumar$ make run
Number of strings equivalent to "Anuj" prior to the new insertion is 0
Number of strings equivalent to "Anuja" prior to the new insertion is 0
Number of strings equivalent to "Anuji" prior to the new insertion is 0
Number of strings equivalent to "krAnuj" prior to the new insertion is 0
Number of strings equivalent to "kumarAnuj" prior to the new insertion is 0
Number of strings equivalent to "Dipen" prior to the new insertion is 0
Number of strings equivalent to "" prior to the new insertion is 0
Number of strings equivalent to "Anuj" prior to the new insertion is 1
Number of strings in "result3" is 1
Number of strings actually removed by "result1" is 3
Number of strings actually removed by "result2" is 3
Number of strings actually removed by "result3" is 1
Number of strings actually removed by "result4" is 0
Number of strings actually removed by "result5" is 1
Number of strings in "result2" is 4
Number of strings in "result5" is 8
Number of strings in "result4" is 1
Number of strings in "result1" is 3
Query not supported for "size"
No such Result exit for "result7"
No such Result exit for "result7"
(base) Dipens-MacBook-Air:DipenKumar dipenkumar$ make clean
(base) Dipens-MacBook-Air:DipenKumar dipenkumar$
```

Format of input file –

1. Filename should be “input.txt”
2. It should be in the same directory containing “Makefile” and “stridx.java”.
3. There are five keywords which are case sensitive –
 - a. “Insert” – This word followed with any string which may be empty string, will add the string in the instantiated data structure.
 - b. “StringWithPrefix” – This is followed the two strings where the first string is the name given to the “Result” instance representing a snapshot of all strings with the prefix as the second followed string in the collection at invocation time. An empty string is an acceptable prefix.
 - c. “StringWithSuffix” – This is followed the two strings where the first string is the name given to the “Result” instance representing a snapshot of all strings with the suffix as the second followed string in the collection at invocation time. An empty string is an acceptable suffix.
 - d. “Size” – This is followed by the name of the “Result” instance which we had assigned using “StringWithPrefix” and “StringWithSuffix” keywords. This returns the number of strings in this Result, including duplicates.
 - e. “Remove” – This is also followed by the name of the “Result” instance which we had assigned using “StringWithPrefix” and “StringWithSuffix” keywords. This removes all (and only) the strings represented in this Result from the parent StringIndex collection.
4. If not given the required number of arguments while using the above keywords then the program will throw index out of bound exception.
5. Arguments to these keywords are case sensitive. Hence all operation are case sensitive.
6. If any other keyword is used then the program will respond with “Query not supported”.
7. Each keyword should be used in a newline.
8. If “Result” name is not declared then the program will respond with – “No such Result exit”

Design –

1. Ternary tree was implemented for StringIndex.
2. To avoid hard work, TreeMap in java was used for balanced tree in Ternary tree implementation.
3. The middle child in the Ternary tree is the TreeMap where it takes $\log(k)$ where k is the size of the Map and given that it is implemented using a balanced tree.
4. Each Node in the Map again has a pointer to another Map as its middle child as in case of Ternary tree.
5. My design uses two Ternary tree one each for prefix and suffix. In case of suffix I have just inverted the string and followed the same approach as in case of prefix that is select a node with the key as the first character of the string and then go deeper with its middle child and remove the first character of the string. Continue till we got have any character left. To save memory I only stored references in both trees pointing to the Data object which stored word. This instead of taking twice the memory for two Ternary tree took only the minimum required.
6. I could have used trie in my implementation but it would have taken more unused spaces which was not memory efficient but could have achieved insertion in

constant time. We were asked to implement a practical and scalable data structure. Hence this didn't gave the best result. With slightly more running time we have achieved optimized memory complexity using Ternary Tree. Ternary Tree only used memory as it is required helping us to scale our data structure and practically it is still fast.

7. I implemented the Result using Linked list of Data object storing the word along with two references to the two different nodes in two different Ternary Tree – prefix and suffix. Once strings in Result is deleted implies it is removed from both Ternary Tree and made null so that when other result delete they know since it is already pointing to null implies already deleted.

Running Time Analysis for Ternary Tree –

1. Insertion – Each time key length of key word is reduced when it select its mid child. It takes $\log(k)$ time to find the node in TreeMap with the given character and once it find the correct node it takes constant time to move to its middle child. Hence Number of such search for nodes is of order length of the word each take time of order $\log(k)$ where k is the size of TreeMap.
Hence Overall time complexity for insertion is $\text{len} * (\log(k)) \Rightarrow \log(k^{\text{len}})$
Now k^{len} is order size of words. Hence $O(\log(n))$ where n is the number of words in StringIndex. Once we reach the node we add the word in the linked list which cost constant time which implies $O(\log(n)) + O(1)$. Hence complexity is $O(\log(n))$.
2. Search – As discussed above it takes $\log(n)$ to reach the node with the given prefix and now we traverse the entire tree using DFS which is linear with the size of the words with that prefix (size of sub tree). Hence complexity is $O(\log(n) + m)$.
3. Maintaining two pointers in Data object along with the word help me save the time to search the tree again and hence I deleted in linear time in size of the Result. Hence complexity is $O(m)$

Proof of Correctness –

1. Pretty much the proof of correctness followed in the explanation of the Design.
2. Here I will prove that my implementation is thread-safe.
3. Insertions should be in order if not then their return values that is the number of string already present, will depends on random execution of threads
4. Insertion and search order cannot be changed because then search will miss or may include extra strings.
5. Insertion and Deletion order should also be maintained else insertion return value will be effected.
6. Point 3,4,5 together claim that if any writing or reading operation is in process then please wait for it to complete.
7. Search operations can change order among themselves as we are not changing the data structure and hence return value of search still remain same.
8. Search and deletion cannot change order of execution because if it then deletion will delete some strings and will change the return value of search.
9. Point 4,7,8 together proves that search should wait if any writing operation is going on and doesn't care if reading operations are on.
10. Deletion and deletion can't change order as return value i.e. number of words deleted by any Result will change according to order.
11. Point 5,8,10 proves that deletion should wait if any read or write operation is on.

12. We were asked to minimize unnecessary blocking of threads so that the data structure can support a greater number of concurrent operations. So these points- 6,9,11 shows the necessary blocking need. Rest are unnecessary and hence were not blocked.
13. We were given to assume that there will be significantly more reads than writes and that strict operation- level atomicity is not required. Hence my threaded implementation will be blocked very minimum because in case of search it is not blocked. It is only blocked if write is operating.
14. My implementation is scalable with no bound other than offered by the hardware. There is no bound on length of string. Can process infinitely large input file given that memory doesn't restrict. Regular deletion with regular insertion can run this program for indefinite period.

Example testcase –

1. To ensure that the above idea was implemented correctly, I as asked instantiated the data structure and executed a series of test routines on it to verify the correctness my implementation, when multiple threads are involved.
2. I created 10 threads and started them.
3. Each thread acquire the lock before it read the file to ensure the order of certain operations remain same as it affect the return values as discussed above.
4. If it is insertion or deletion the it doesn't releases the lock until it see its way clear that is 0 write operations and 0 read operations running at the time. If it is not 0 then it wait. When it is zero it increases it count in write operations at present and releases the lock.
5. If search acquired the lock then if only check for 0 write operation and wait till it is 0. Then it increase its counts in read operations and releases the lock.
6. Example input file –

```

Insert Anuj
Insert Anuja
Insert Anuji
Insert krAnuj
Insert kumarAnuj
Insert Dipen
Insert
StringsWithPrefix result1 Anuj
Insert Anuj
StringsWithSuffix result2 Anuj
StringsWithPrefix result3 Di
StringsWithSuffix result4 en
StringsWithSuffix result5
Size result1
Size result2
Size result3
Size result4
Size result7
Size result5
Remove result1
Remove result2

```

Remove result3
Remove result4
Remove result5
Remove result7
size result1

7. Example output –

Number of strings equivalent to "Anuj" prior to the new insertion is 0
Number of strings equivalent to "Anuja" prior to the new insertion is 0
Number of strings equivalent to "Anuji" prior to the new insertion is 0
Number of strings equivalent to "krAnuj" prior to the new insertion is 0
Number of strings equivalent to "kumarAnuj" prior to the new insertion is 0
Number of strings equivalent to "Dipen" prior to the new insertion is 0
Number of strings equivalent to "" prior to the new insertion is 0
Number of strings equivalent to "Anuj" prior to the new insertion is 1
Number of strings in "result3" is 1
Number of strings actually removed by "result1" is 3
Number of strings actually removed by "result2" is 3
Number of strings actually removed by "result3" is 1
Number of strings actually removed by "result4" is 0
Number of strings actually removed by "result5" is 1
Number of strings in "result2" is 4
Number of strings in "result5" is 8
Number of strings in "result4" is 1
Number of strings in "result1" is 3
Query not supported for "size"
No such Result exit for "result7"
No such Result exit for "result7"

8. First, Output display the insertion output in the same order as input because it was waiting for write to clear off.
9. When “Anuj” was added twice it return 1 while rest cases it was 0.
10. “Size” keyword return the initial size of Result. It is a constant. It neither changes StringIndex or Result nor effected by the change in Result or StringIndex. Hence it was not conditioned using any locks and is absolutely free from any constraint. Hence we see random order of “Size” output.
11. Also similar to insertion, deletion i.e. Remove, the output is the same order as input because it was waiting for write to clear off.
12. Search can shuffle their order among themselves as long as they are not violating order with insert or delete i.e. remove.
As seen from input result7 was not defined. Hence when queried for the Size of the result7 or when asked to delete result7, program say – No such Result exit for "result7"
13. Also when we enter wrong keyword to communicated with our data structure we receive – Query not supported
14. Since we receive Query not supported for “size”, this shows keywords are case sensitive and also are the arguments and all strings operations.