T-301-REIR, REIKNIRIT
HAUST 2019
D6 – STRINGS

Problem 1. Suppose you need to sort 10 million integers, each in the range 0 to 240. How would you do it? Which method, among the ones we have seen, gives the smallest tilde time complexity.

- I would use the key-indexed counting algorithm, which is based on keys within a specific range (for example integers between 0 and 240) which is used as an array index.

  The method counts the number of objects having an distinct key value, then calculates the position of each object in the output. Example:

Array unsorted:

index / number	a[i]
0	9
1	5
2	1
3	6
4	1
5	7
6	0
7	2
8	0
9	5
...	...

1) Take a count array to store the count of each unique object:

index / number		count[i]
0	↘	0
1	↘	2
2	↘	2
3	↘	1
4	↘	0
5	↘	0
6	↘	2
7	↘	1
8	↘	1
9	↘	0
...		1
		...

2) Modify the count array such that each element at each index stores the sum of previous counts:

index / number	count[i]
0	0
1	2
2	4
3	5
4	5
5	5
6	7
7	8
8	9
9	9
…	10
	…

The modified count array indicates the position of each object in the output sequence.

3) Output each object from the input sequence followed by increasing its count by one (1):

index / number	count[i]
0	0
1	3
2	4
3	5
4	5
5	6
6	7
7	8
8	9
9	10
…	10
	…

index / number	aux[i]
0	
1	
2	1
3	
4	
5	5
6	
7	
8	
9	9
…	…

Three numbers been sorted from the original input array

Final output (sorted array):

index / number	count[i]
0	2
1	4
2	5
3	5
4	5
5	7
6	8
7	9
8	9
9	10
…	10 …

index / number	aux[i]
0	0
1	0
2	1
3	1
4	2
5	5
6	5
7	6
8	7
9	9
…	…

The time complexity is proportional to N+R, where N is the number of integers and R is counting array created for each index.

Problem 2. Describe an input instance for which MSD is much faster than LSD.

- As the MSD method considers the most significant digit/character first, it sorts objects without having to iterate through all of the digits/characters in the string (at least when there is no duplicated items to be sorted), thereby sorting the entire range of objects on the first pass. This of course means that the range of objects have to be randomly distributed.

  MSD is also faster when the objects being sorted is of large quantity, as it holds hand in hand with starting with the most significant item, instead of the least significant item (as is done in LSD) and can therefore be sorted without iterating through every item.

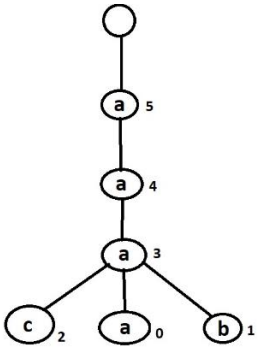
Problem 3. When is 3-way string quickSort the preferred sorting method?

- The 3-way string quicksort adapts well to handling equal keys, keys with long common prefixes, keys that fall into a small range, and small arrays. All these are situations where MSD string sort runs slowly. Of particular importance is that the partitioning adapts to different kinds of structure in different parts of the key. Also, like quicksort, 3-way string quicksort does not use extra space, which is an important advantage over MSD string sort, which requires space both for frequency counts and an auxiliary array.

Problem 4. Construct a set S of strings in lowercase (R = 26) for which the R-way trie uses space (in words) less than the total number of characters in strings in S. (The number of characters in the set {abc, bcd, aabd} is 10.)

- The R-way trie cannot use less space than the total number of characters in strings in S, because each node has R children – one for each possible character (R null links in each leaf).

  For example:
  The set {aaaa, aaab, aaac, aaa, aa, a} is total of 18 characters, and the R-way trie would look like:



  As can be seen in the picture above, the total nodes are 7 or 7 characters (which is less than is given in the set), but what is not shown is all the null links, which would then give a total of 7 * 26 = 182 characters.

Problem 5. (Final exam 2018) Check all the properties that hold for a trinary trie (R =3) that contains N strings of average length w.

a) A search hit takes time (in character comparisons) proportional to the length of the search key.

- The number of array accesses when searching in a trie is at most 1 plus the length of the key. Whatever algorithm or data structure we are using, we cannot know that we have found a key that we seek without examining all of its characters.

~~*b) Deleting a key is difficult.*~~

- The implementation of deleting a key can be accomplished with remarkably little code.

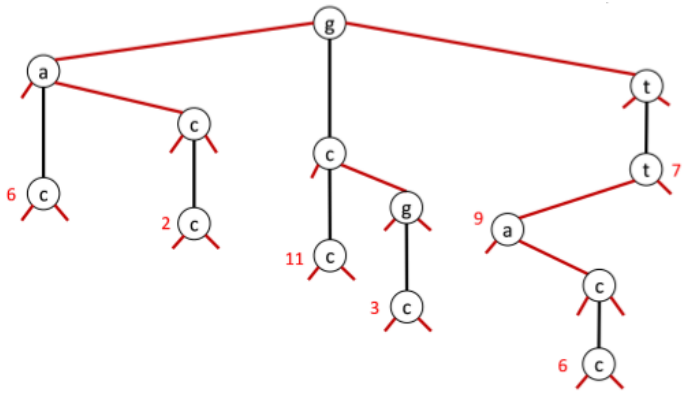
c) The space usage is significantly larger than an equivalent ternary search trie (TST).

- Yes, the space usage for R-way tries is significantly larger.

d) The number of empty links can be larger than Nw.

- Every key in the trie has a node containing its associated value that also has *R* links, so the number of links is at least *nR*. If the first character of all the keys are different, then there is a node with *R* links for every key character, so the number of links is *R* times the total number of key characters, or *wnR*.

Problem 6. (Final exam 2017) Consider the following Ternary Search Trie (TST), where the values are shown next to the nodes of the corresponding string keys.



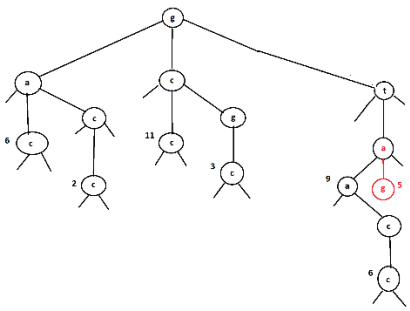
a) List the keys stored in the trie, in alphabetical order.

- ac – cc – gcc – ggc – ta – tt – tcc

b) Give one possible order in which the keys were inserted into the trie:

- gcc – ggc – tt – ta – tcc – ac - cc

c) Insert the key tag into the trie, with value 5. Show or describe how the trie changes.



- The key tt with the value 7 gets overwritten, and replaced by the new key 'tag' with the value 5 – see picture above.

Problem 7. Suppose you have a set X of one million DNA strings of length 20, and a long string S (a genome) of length 109. You want to answer which of the short strings occur in S as substrings.

a) *How long would it take to search for each string from X in S individually? (Assume, say, that your computer can perform 109 character comparisons per second.)*
    - It would then take 1 second to make the comparison, as the long string S is of length 109 and it takes one second to perform 109 character comparisons. So with one single iteration, you have checked all of the one million DNA strings which are (or are not) a substring in S.

b) *Explain how to achieve this efficiently using a trie. Estimate the amount of time it would take.*
    - You would do it in a way as shown in the picture below, where the word 'bananas' is the word being examined and the word is then broken down into pieces for each possible part in the word. Then we can iterate through each DNA string, to see if it appears in the word string S.