Jiapeng Pei S01435611

Module 10 Problems

1. [9 pts] Consider an input of size N consisting of exclusively numbers. Suppose you know the input values are ***not*** bounded. You do know, however, that the number of ***distinct*** input values is "small" Specifically, let M be the number of distinct input values. Then, you are guaranteed that M log M = O(N). Give an algorithm that has expected time O(N) to sort the items.
   Hint: What techniques do you know that make good use of expected-time behavior?

```python
def mySort(array):
    numFrequency = dict()
    ret = []
    for n in array:
        if n not in numFrequency:
            numFrequency[n] = 0
        numFrequency[n] += 1
    keys = list(numFrequency.keys())
    keys.sort()
    for key in keys:
        for i in range(numFrequency[key]):
            ret.append(key)

    return ret

nums = [1,253,254,10086,10086,9999,27,48,59,59,59,1,1,1]
print(mySort(nums))
# [1, 1, 1, 1, 27, 48, 59, 59, 59, 253, 254, 9999, 10086, 10086]
```

   My algorithm takes $O(MlogM)$ to sort the distinct values, which is $O(N)$. And it takes another $O(N)$ to push all the values back to the return array. So the overall complexity is $O(N)$.

2. [9 pts] Give a trace for the LSDsort applied to the following strings:
   no is th ti fo al go pe to co to th ai of th pa

   Here is how I am going to perform LSDsort: apply counting sort to sort the last digit, and sort last - 1 digit, ... and sort the first digit. Following is the trace:

   - Sort 2^{nd} digit: pa pe of th th th ti ai al no fo go to co to is
   - Sort 1^{st} digit: ai al co fo go is no of pa pe th th th ti to to

3. [9 pts] What modification to LSDsort would you make to cover variable length strings?

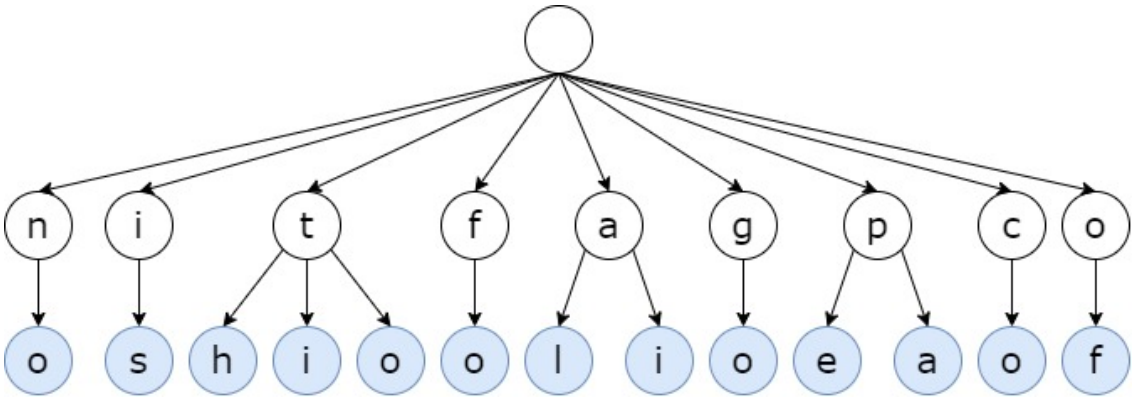
Suppose the maximum string length is m. For those strings that are less than m in length, I will fill its length to m using imaginary value β. β is smaller than every possible char value and equal to β. Then I could perform a normal LSD sort. For example, I need to sort bba, cd, aba, az:

- Sort 3^{rd} digit: cd(β), az(β), bba, aba
- Sort 2^{nd} digit: bba, aba, cd(β), az(β)
- Sort 1^{st} digit: aba, az(β), bba, cd(β)

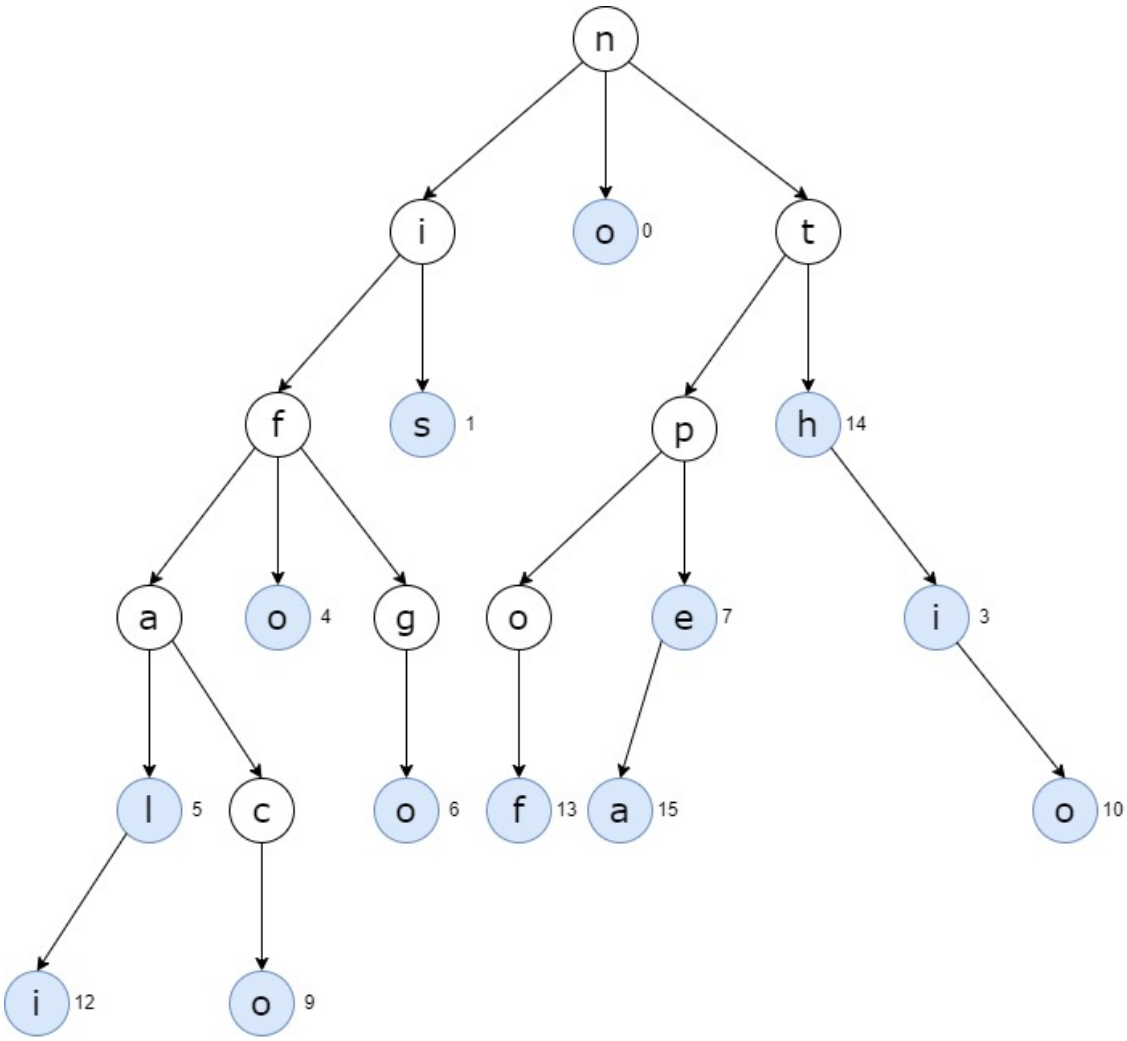
In this way, we can sort variable length strings in their dictionary order.

4. [7 pts] Draw the 26-way trie that results from inserting the following strings into the empty trie
no is th ti fo al go pe to co to th ai of th pa

Since in implementation, I use the hashmap to store the child nodes. So the relative order within a level of tree doesn't represent their actual stored order.



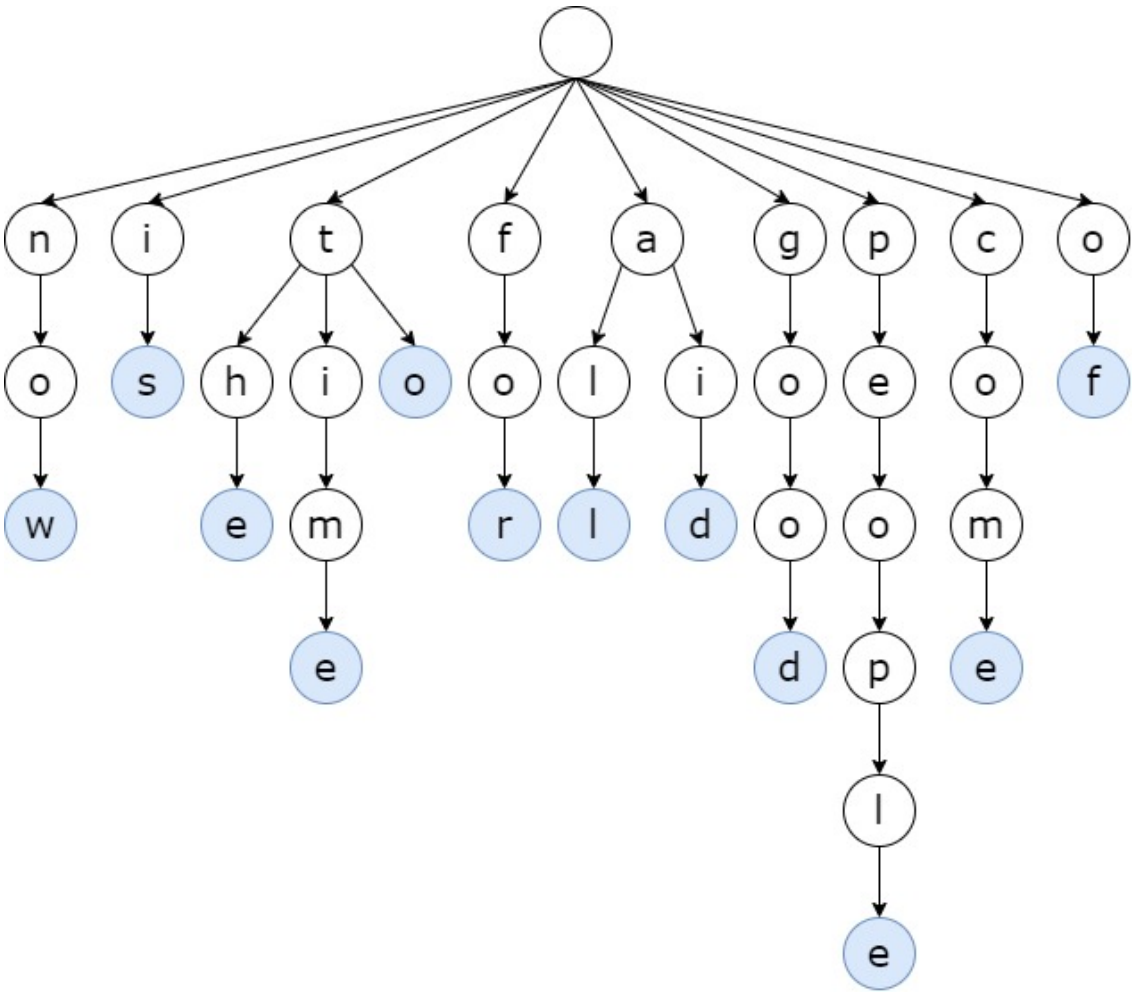
5. [7 pts] Draw the TST that results from insering the following strings into the empty trie
no is th ti fo al go pe to co to th ai of th pa



6. [7 pts] Draw the 26-way trie that results from inserting the following strings into an initially empty trie

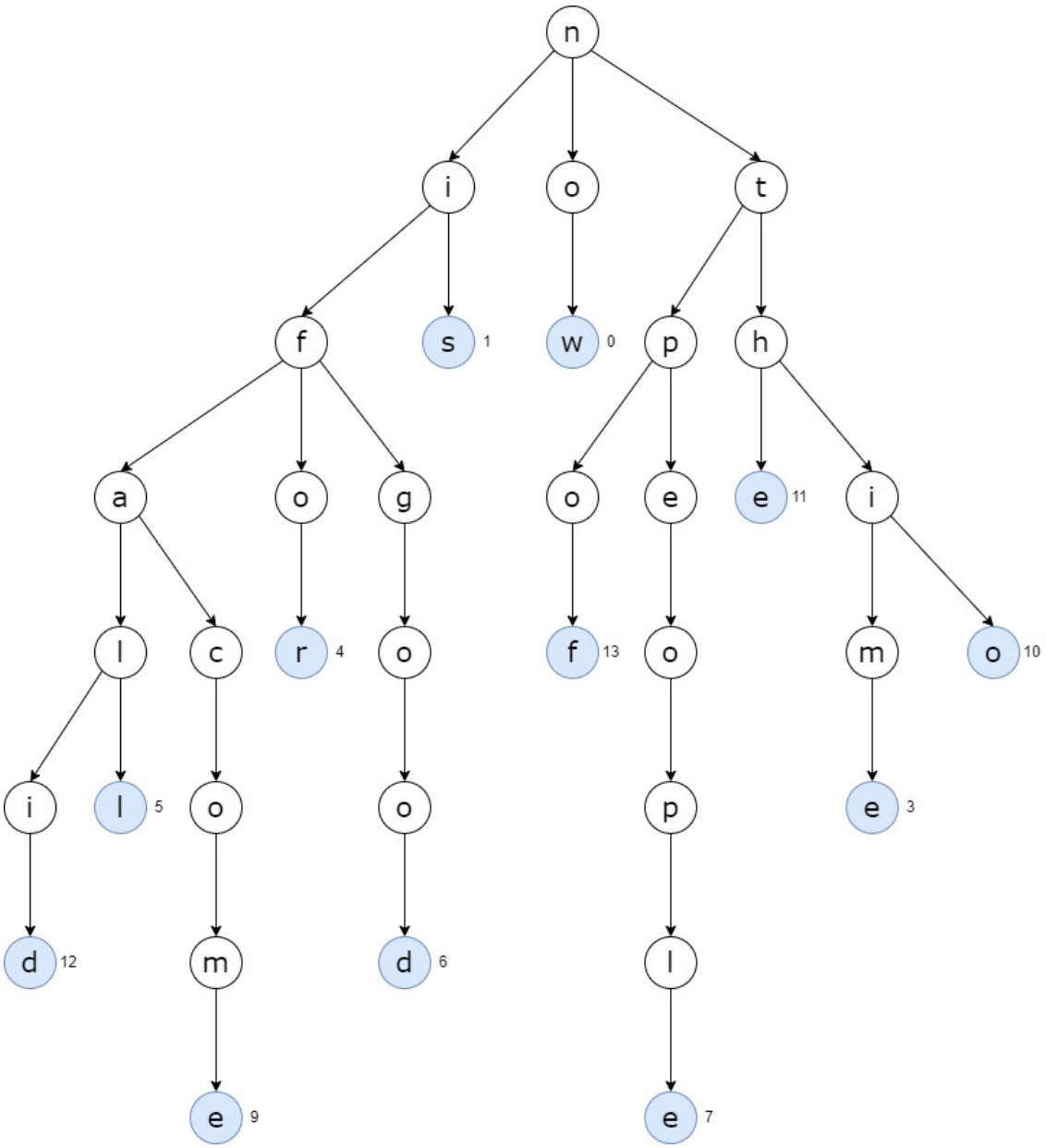
   now is the time for all good people to come to the aid of

   Since in implementation, I use the hashmap to store the child nodes. So the relative order within a level of tree doesn't represent their actual stored order.



7. [7 pts] Draw the TST that results from inserting the following strings into an initially empty trie

   now is the time for all good people to come to the aid of



8. [12 pts] Show the state transitions for DFA below

DFA

	0	1	2	3	4	5
A	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6

for the following input strings:

8.1 [6 pts] ABCAAABABABACAABB

- begin: 0
- A: 1
- AB: 2
- ABC: 0
- ABCA: 1
- ABCAA: 1
- ABCAAA: 1
- ABCAAAB: 2

- ABCAAABA: 3
- ABCAAABAB: 4
- ABCAAABABA: 5
- ABCAAABABAB: 4
- ABCAAABABABA: 5
- ABCAAABABABAC: 6
- Now reach the end and find the substring.

8.2 [6 pts] ABCAAABAABACAABBA

- begin: 0
- A: 1
- AB: 2
- ABC: 0
- ABCA: 1
- ABCAA: 1
- ABCAAA: 1
- ABCAAAB: 2
- ABCAAABA: 3
- ABCAAABAA: 1
- ABCAAABAAB: 2
- ABCAAABAABA: 3
- ABCAAABAABAC: 0
- ABCAAABAABACA: 1
- ABCAAABAABACAA: 1
- ABCAAABAABACAAB: 2
- ABCAAABAABACAABB: 0
- ABCAAABAABACAABBA: 1
- Reach the last character, and couldn't find substring.