Answer to Problem Set 10

- Student Name: Gaole Dai
- Student ID: S01435587

Problem 1

Because the distinct input value is small, and input have exclusively numbers, we could map all the items into a **hash map**, count the number of each items as map value and then sort key, the whole mapping process cost $\mathcal{O}(n)$, and the time complexity for sort is $M\log(M)$, which has an upper bound of $\mathcal{O}(n)$. Then go though the hash map and copying each key for value's time and obtain the final sorted result. The time complexity for the iteration is still $\mathcal{O}(n)$. Therefore, the time complexity for the whole algorithm is $\mathcal{O}(n)$.

The python code for the algorithm is shown below:

```
def sortDup(arr):
    hashmap = {};
    for i in range(len(arr)):
        if arr[i] in hashmap:
            hashmap[arr[i]] += 1
        else:
            hashmap[arr[i]] = 1
    sorted(hashmap)

    idx = 0
    for key in hashmap:
        for i in range(hashmap[key]):
            arr[idx] = key
            idx += 1

    return arr
```

Problem 2

```
def LSDsort(s, W){
  R = 256 // User-chosen Radix, normally 256
  aux = [0] * N
  # stable count sort on each char position
  for d in range (W-1, -1, -1):
    # Compute frequency counts.
    for i in range(N):
      count[a[i].key() + 1] += 1
    # Transform counts to indices.
    for r in range(R):
      count[r+1] += count[r]
    # Distribute the records.
    for i in range(N):
      aux[count[a[i].key()]+1] = a[i]
    # Copy back.
    for i in range(N):
        a[i] = aux[i]
```

Apply the algorithm, the trace could be shown in table below:

Data	d=1	d=0
no	pa	ai
is	pe	al
th	of	co
ti	th	fo
fo	th	go
al	th	is
go	ti	no
pe	ai	of
to	al	pa
co	no	pe
to	fo	th
th	go	th
ai	to	th
of	co	ti
th	to	to
pa	is	to

Problem 3

The modification is that we need to find the longest length of the string in input array, so that when d is larger or equal to the longest string, we add **extra 0** to make the string have same lengths.

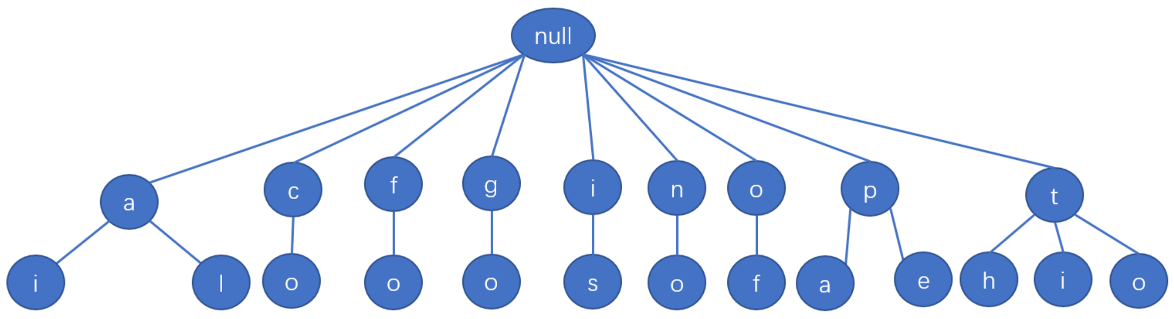
```
def findLongest(array):
    return len(max(array, key=len))

def LSD_Sort(array):
    size = len(array)
    #  extended ASCII alphabet size.
    R = 256
    aux = [0] * size
    count = [0] * (R + 1)
    w = findLongest(array)
    for d in range (w-1, -1, -1):
        #  compute frequency counts.
        for i in range(size):
            if d < 0 or d >= len(array[i]):
                count[1] += 1
            else:
                count[ord(array[i][d]) + 1] += 1
        #  compute cumulates
        for r in range(R):
            count[r+1] += count[r]

        #  Distribute the records.
        for i in range(size):
            if d < 0 or d >= len(array[i]):
                aux[count[0]] = array[i]
                count[0] += 1
            else:
                aux[count[ord(array[i][d])]] = array[i]
                count[ord(array[i][d])] += 1
        #  Copy back.
        for i in range(size):
            array[i] = aux[i]
```

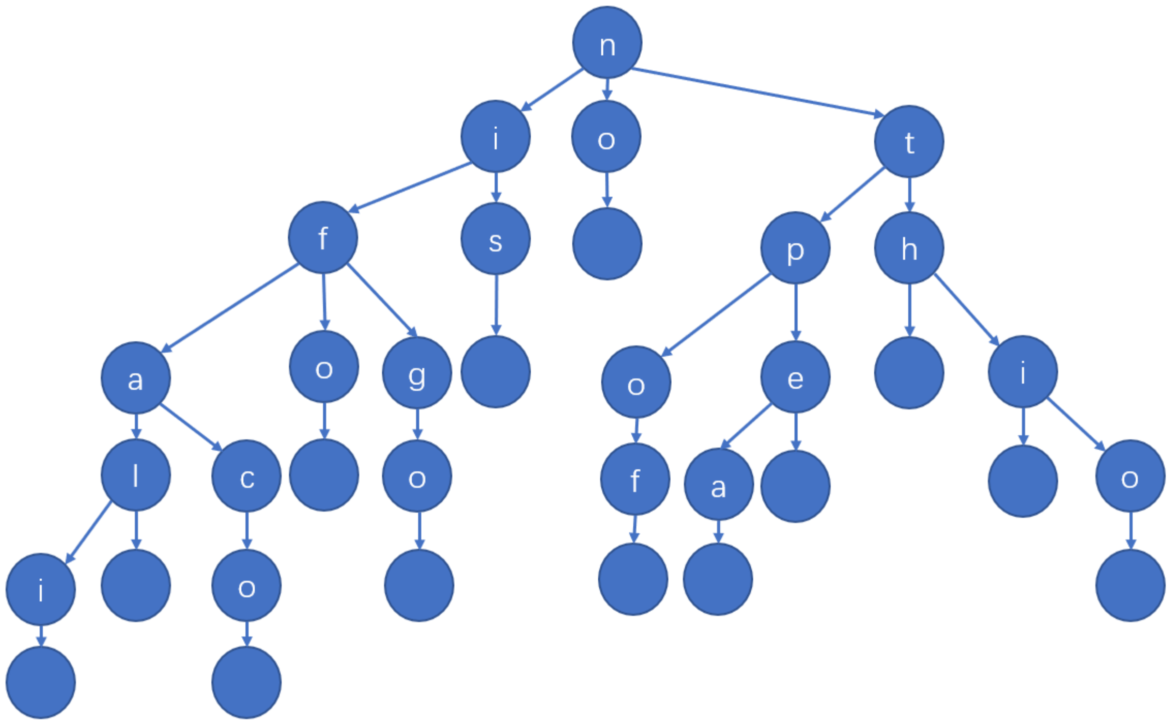
Problem 4

The 26-way trie that result from inserting the strings is:



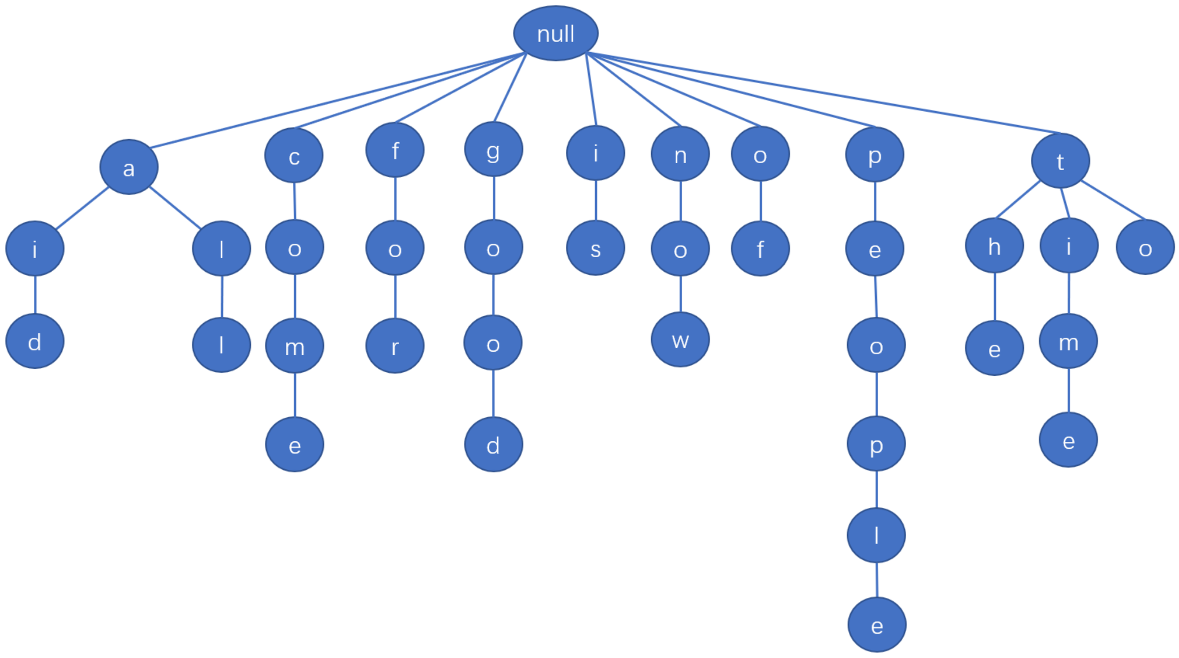
Problem 5

Each node has 3 links (children): smaller (left), equal (middle), larger (right). Insert each string into the tries, if the character larger than the parent node, then insert into left, if smaller than the parent node, insert into right, if equal insert into middle.



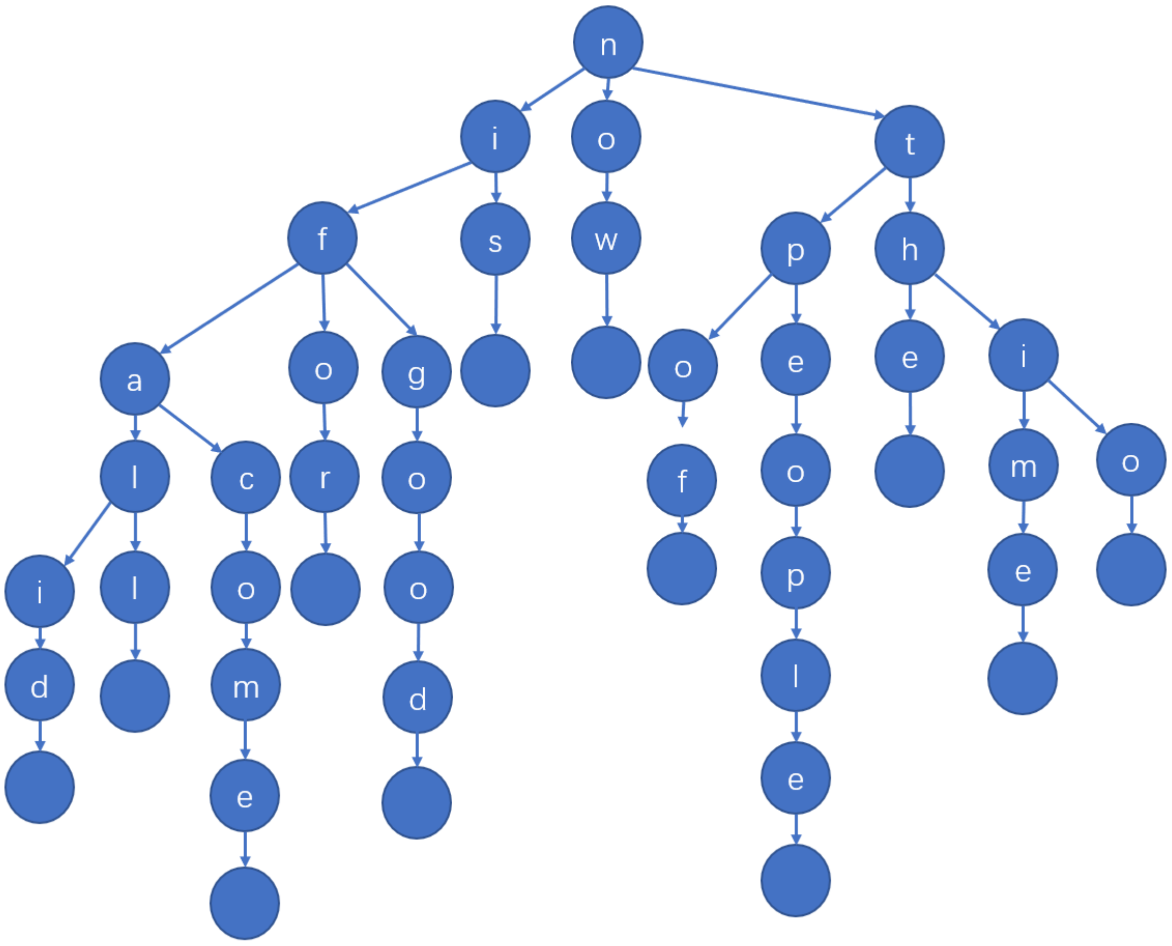
Problem 6

The 26-way trie that result from inserting the strings is:

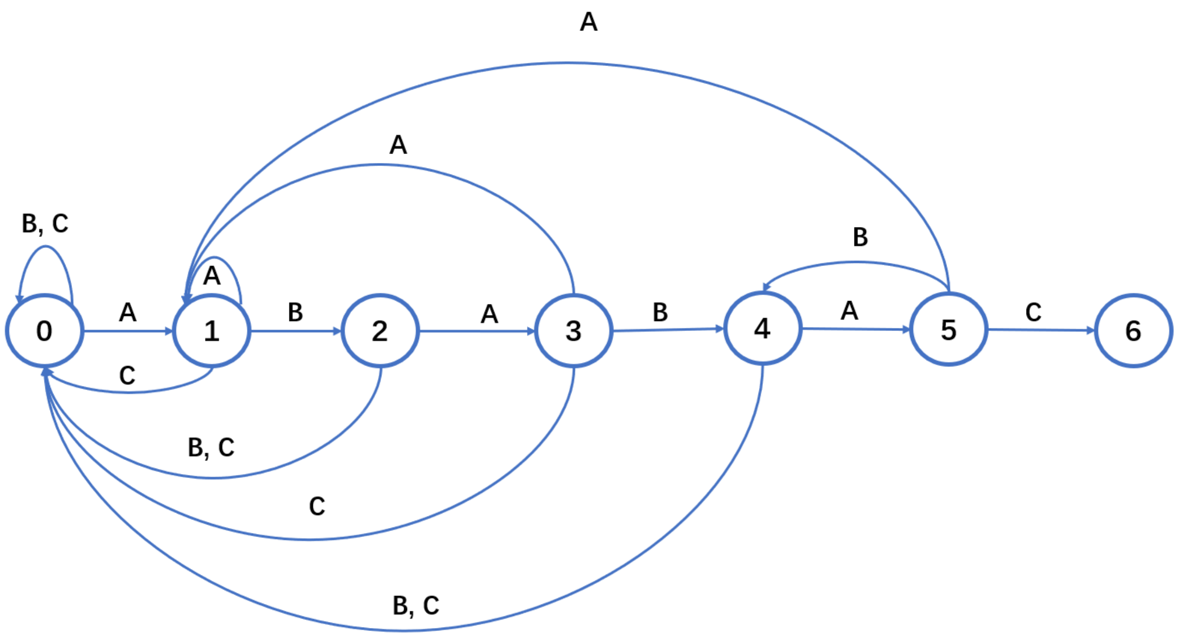


Problem 7

Similar method as problem 5.



Problem 8



8.1

Input	Cur State	Go to State
A	0	1
B	1	2

Input	Cur State	Go to State
C	2	0
A	0	1
A	1	1
A	1	1
B	1	2
A	2	3
B	3	4
A	4	5
B	5	4
A	4	5
C	5	**6**
Match and Stop		

8.2

Input	Cur State	Go to State
A	0	1
B	1	2
C	2	0
A	0	1
A	1	1
A	1	1
B	1	2
A	2	3
A	3	1
B	1	2
A	2	3
C	3	0
A	0	1
A	1	1

Input	**Cur State**	**Go to State**
B	1	2
B	2	0
A	0	**1**
No match		