homework 4

problem 1

The bus

[2 points]

We have following collection of strings called Patterns and a string called Text.

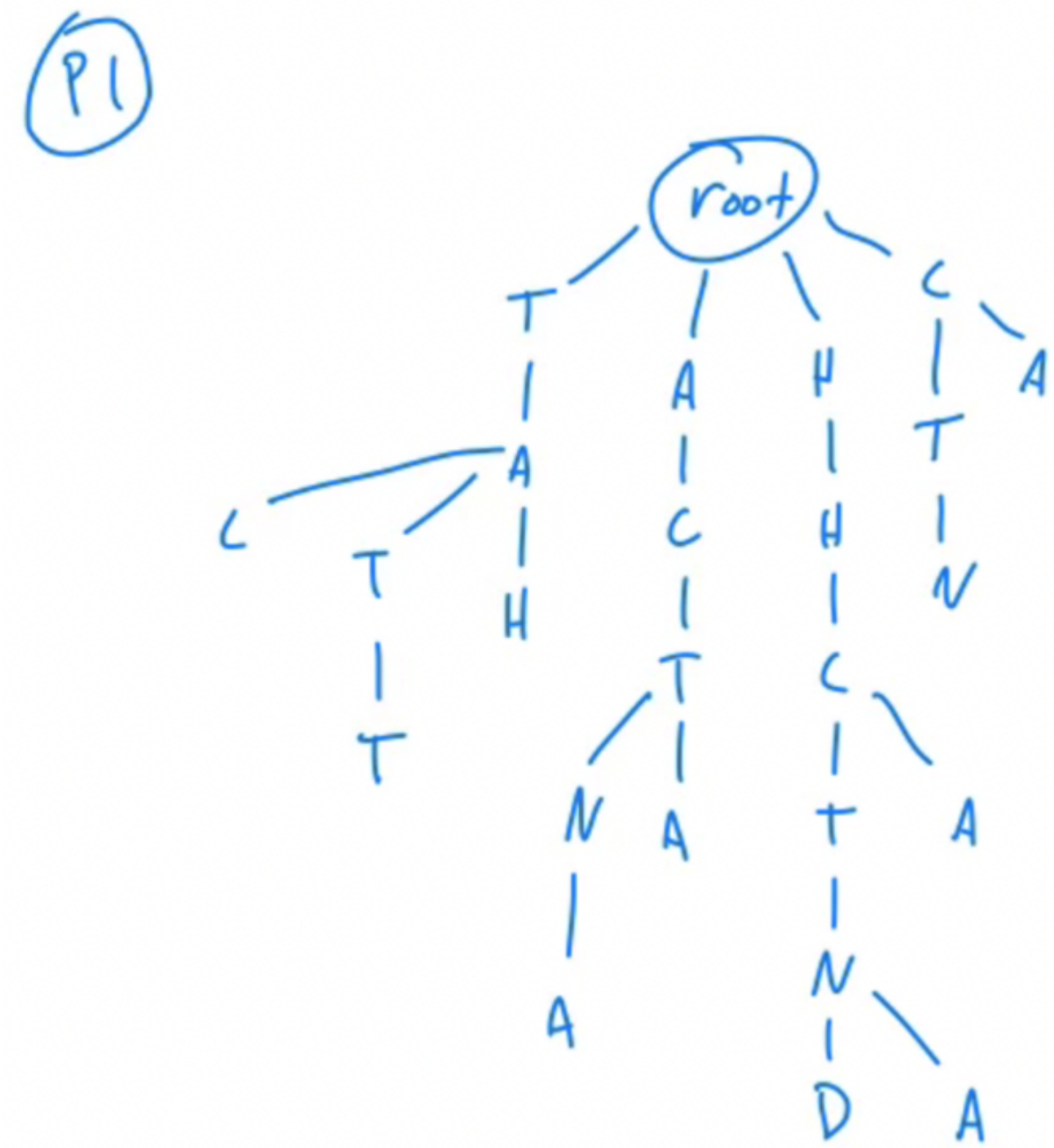
Text := tactnahhctndhhctna

Patterns :=

1. tah
2. acta
3. hhctnd
4. ca
5. hhca
6. tatt
7. tac
8. actna
9. hhctna
10. ctn

Construct *Trie(Patterns)* using the algorithm we learned in class. How many leaves does the trie have?

answer 1



it has 10 leaves

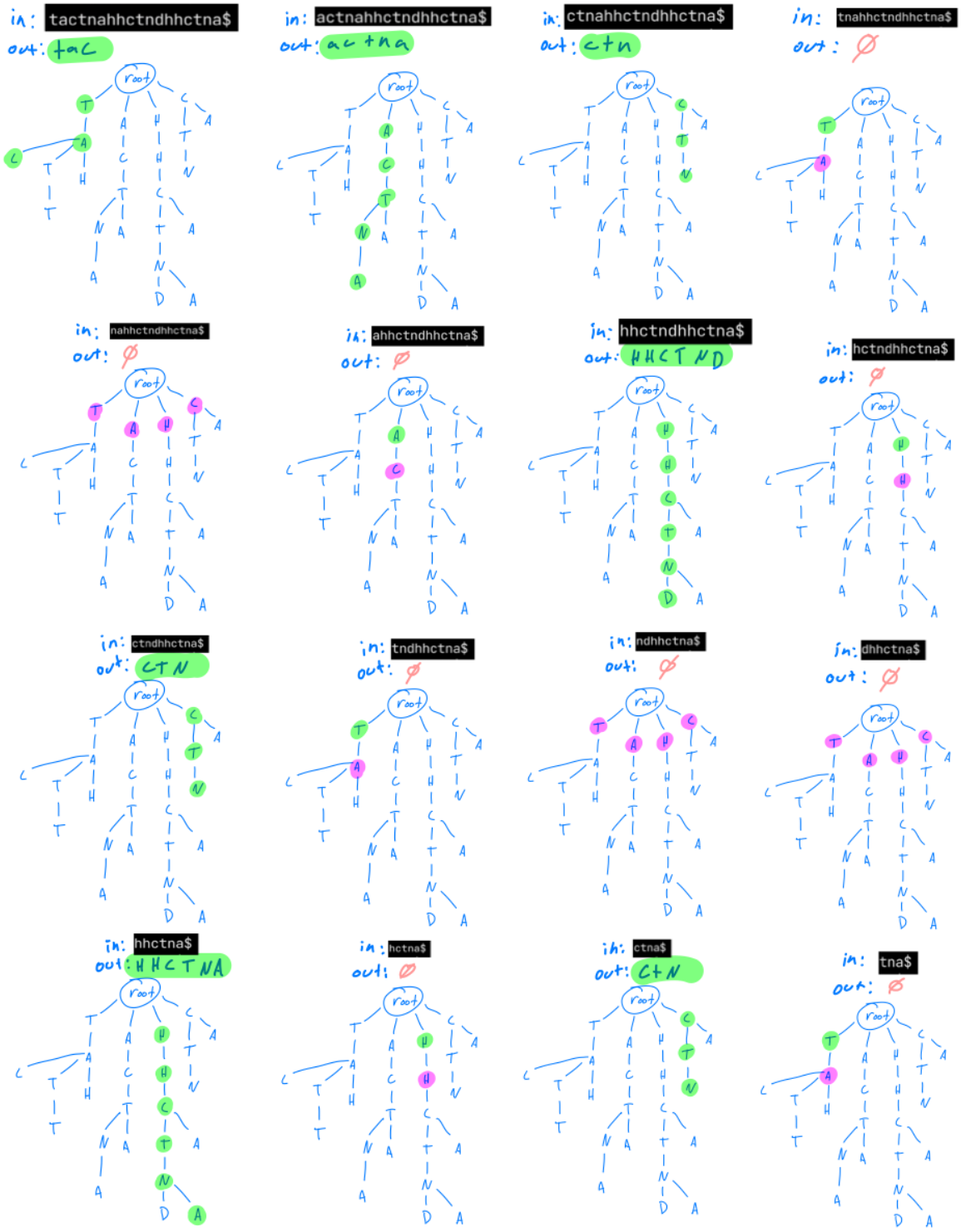
problem 2

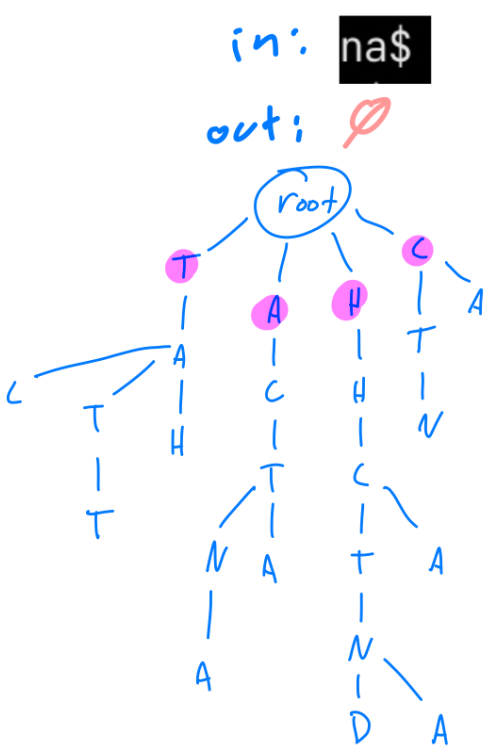
Bus stops

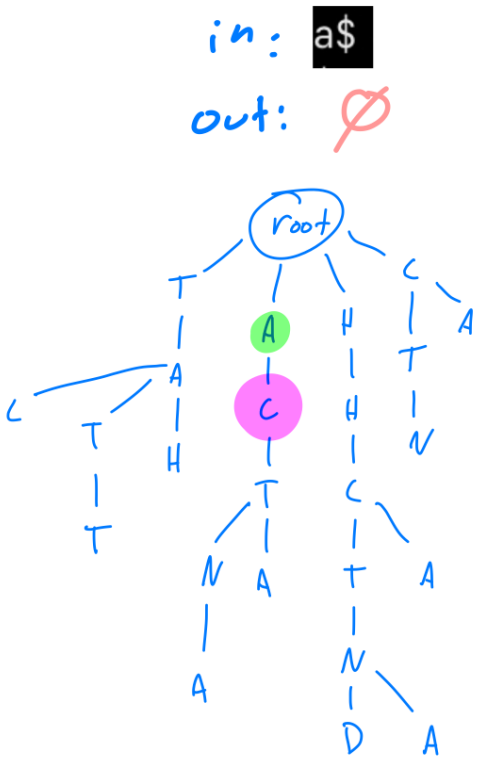
[3 points]

Perform TrieMatching between *Text* and *Trie(Patterns)* you generated in question 1 according to the algorithm we learned in class. That is, find all substrings of *Text* that match any strings in Patterns. Show all iterations of the TrieMatching algorithm; at each iteration, show the input and output by TrieMatching. How many matches were found? That is, how many substrings of *Text* matched one of strings in Patterns?

answer 2


in: tactnahhctndhhctna$
out: tac
in: actnahhctndhhctna$
out: actna
in: ctnahhctndhhctna$
out: ctn
in: tnahhctndhhctna$
out: ∅
in: nahhctndhhctna$
out: ∅
in: ahhctndhhctna$
out: ∅
in: hhctndhhctna$
out: HHCTND
in: hctndhhctna$
out: ∅
in: ctndhhctna$
out: CTN
in: tndhhctna$
out: ∅
in: ndhhctna$
out: ∅
in: dhhctna$
out: ∅
in: hhctna$
out: HHCTNA
in: hctna$
out: ∅
in: ctna$
out: CtN
in: tna$
out: ∅





- there are 7 matches

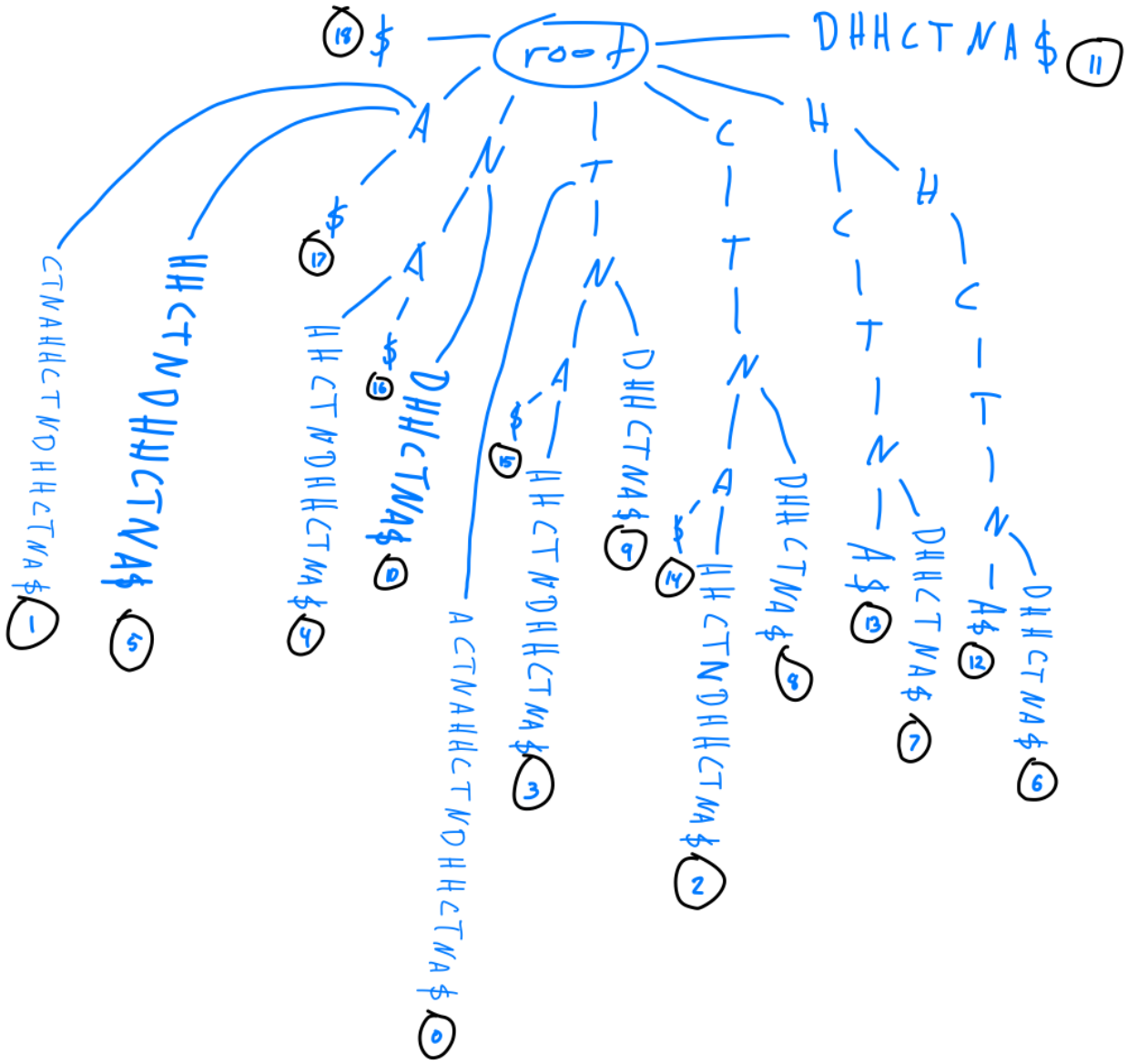
problem 3

Suffix trees

[5 points]

Create a suffix tree of *Text*. Show all suffixes of *Text* first and draw the tree. How many leaves are in the suffix tree? Is it the same as the number of leaves in Trie(Pattern)? Why or why not?

answer 3



- there are 19 leaves
- It is not the same as `Trie(Pattern).`
- There are more leaves in the suffix tree because the `Trie(Pattern)` contains only patterns and there are more suffixes to `Text` than there are patterns and some suffixes of `Text` are not accounted for by a pattern.

problem 4

Suffix arrays

[2 points]
Create a suffix array of *Text*. Show sorted suffixes and their starting positions.

answer 4

```
# number 4
for v in sorted([(s[i:],i) for i in list(range(len('tactnahhctndhhctna$')))]):
    print(v[1], v[0])
```

```
18 $
17 a$
1 actnahhctndhhctna$
5 ahhctndhhctna$
14 ctna$
2 ctnahhctndhhctna$
8 ctndhhctna$
11 dhhctna$
13 hctna$
7 hctndhhctna$
12 hhctna$
6 hhctndhhctna$
16 na$
4 nahhctndhhctna$
10 ndhhctna$
0 tactnahhctndhhctna$
15 tna$
3 tnahhctndhhctna$
9 tndhhctna$
```

problem 5

BWT

[5 points]
Create the Burrows-Wheeler transform of *Text*. First, create cyclic rotations matrix. Second, create M(*Text*) matrix. Lastly, output BWT(*Text*) from M(*Text*).

answer 5

```
s = 'tactnahhctndhhctna$'
for i in range(len(s)):
    rot = s[i:] + s[:i]
    print(rot)
```

```
# cyclic rotations matrix
tactnahhctndhhctna$
actnahhctndhhctna$t
ctnahhctndhhctna$ta
tnahhctndhhctna$tac
nahhctndhhctna$tact
ahhctndhhctna$tactn
hhctndhhctna$tactna
hctndhhctna$tactnah
ctndhhctna$tactnahh
tndhhctna$tactnahhc
ndhhctna$tactnahhct
dhhctna$tactnahhctn
hhctna$tactnahhctnd
hctna$tactnahhctndh
ctna$tactnahhctndhh
tna$tactnahhctndhhc
na$tactnahhctndhhct
a$tactnahhctndhhctn
$tactnahhctndhhctna
```

```
l = []
s = 'tactnahhctndhhctna$'
 for i in range(len(s)):
     rot = s[i:] + s[:i]
     l.append(rot)
 for v in sorted(l):
     print(v)
```

```
# M(Text)
$tactnahhctndhhctna
a$tactnahhctndhhctn
actnahhctndhhctna$t
ahhctndhhctna$tactn
ctna$tactnahhctndhh
ctnahhctndhhctna$ta
ctndhhctna$tactnahh
dhhctna$tactnahhctn
hctna$tactnahhctndh
hctndhhctna$tactnah
```

```
hhctna$tactnahhctnd
hhctndhhctna$tactna
na$tactnahhctndhhct
nahhctndhhctna$tact
ndhhctna$tactnahhct
tactnahhctndhhctna$
tna$tactnahhctndhhc
tnahhctndhhctna$tac
tndhhctna$tactnahhc
```

```
l = []
s = 'tactnahhctndhhctna$'
for i in range(len(s)):
    rot = s[i:] + s[:i]
    l.append(rot)
for v in sorted(l):
    print(v[-1],end='')
```

```
# BWT(Text)
antnhahnhhdattt$ccc
```

problem 6

BWT traversal

[5 points]

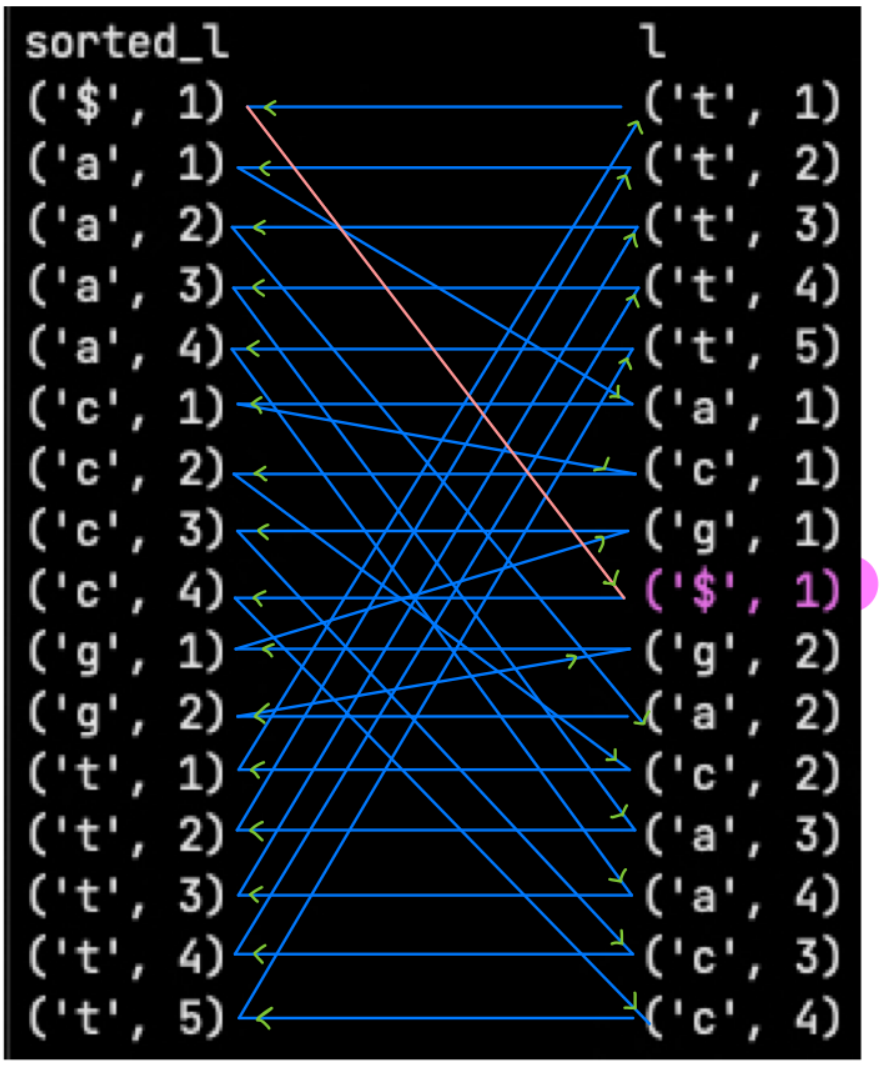
Reconstruct the string whose Burrows-Wheeler transform is *tttttacg$gacaacc*. Show how you reconstruct each letter using the First-Last property in each iteration as we learned in class (refer to Figure 9.12 in textbook). In other words, show the partial M(*Text*) matrix and the two arrows as in Figure 9.12 to indicate each letter you are reconstructing. Be sure to label the edges according to the order of how they should be traversed.

answer 6

```
s = 'tttttacg$gacaacc'
l = []
d = {}
for c in s:
    count = d.get(c,None)
```

```
    if not count:
        d[c] = 1
    else:
        d[c] += 1
    count = d[c]
    l.append((c,count))
    last_c = c
sorted_l = sorted(l)
print('sorted M(Text)','M(Text)',sep='\t')
for v1,v2 in zip(sorted_l,l):
    print(v1,v2,sep='\t')
```

```
sorted M(Text)  M(Text)
('$', 1)  ('t', 1)
('a', 1)  ('t', 2)
('a', 2)  ('t', 3)
('a', 3)  ('t', 4)
('a', 4)  ('t', 5)
('c', 1)  ('a', 1)
('c', 2)  ('c', 1)
('c', 3)  ('g', 1)
('c', 4)  ('$', 1)
('g', 1)  ('g', 2)
('g', 2)  ('a', 2)
('t', 1)  ('c', 2)
('t', 2)  ('a', 3)
('t', 3)  ('a', 4)
('t', 4)  ('c', 3)
('t', 5)  ('c', 4)
```



$C_4\ T_5\ A_4\ T_3\ A_2\ G_2\ G_1\ C_3\ T_4\ A_3\ T_2\ A_1\ C_1\ C_2\ T_1\ \$_1$

problem 7

BWMatching

[8 points]

Using the M(*Text*) matrix created in question 5, perform BWMatching between *Text* and the following 2 strings in Patterns. First, create a table that has i, FirstColumn, LastColumn, and LastToFirst(i) as in Figure 9.15 in textbook (or as in lecture slides). Then, for each string

in Pattern, show the values of top and bottom variables in each iteration of BWMatching, as in Figure 9.14 in textbook. How many times does each string in Pattern appear in *Text*?
Patterns:

1. ctn
2. cna

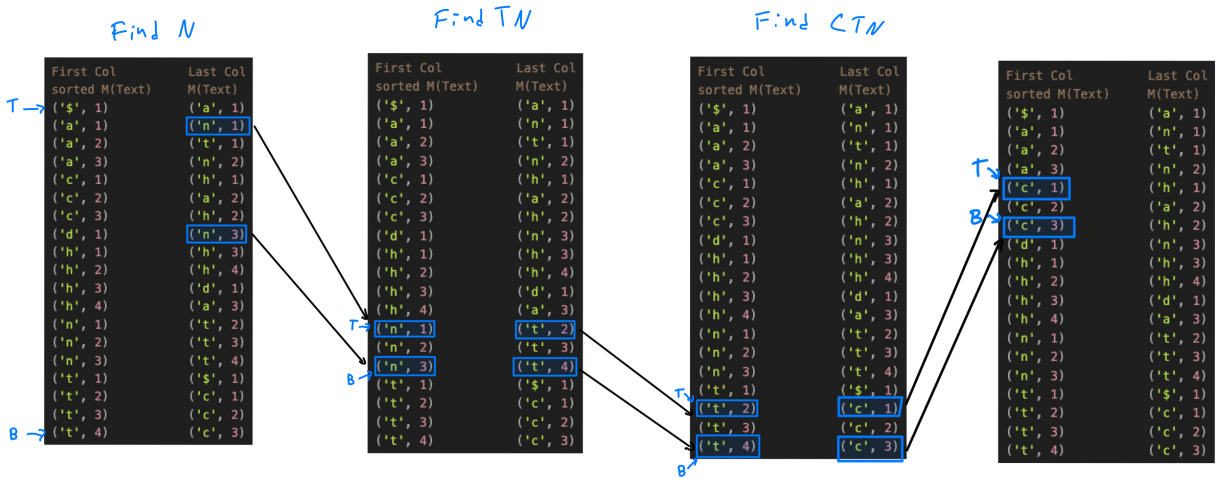
answer 7

```
s = 'antnhahnhhdattt$ccc'
l = []
d = {}
for c in s:
    count = d.get(c,None)
    if not count:
        d[c] = 1
    else:
        d[c] += 1
    count = d[c]
    l.append((c,count))
    last_c = c
sorted_l = sorted(l)
print('i','sorted M(Text)','M(Text)','\tlast to first',sep='\t')
for i,(v1,v2) in enumerate(zip(sorted_l,l)):
    print(i,end='\t')
    print(v1,v2,sep='\t', end='\t')
    print(sorted_l.index(v2))
```

```
#     First Col          Last Col
#i    sorted M(Text)     M(Text)     last to first
0     ('$', 1)           ('a', 1)    1
1     ('a', 1)           ('n', 1)    12
2     ('a', 2)           ('t', 1)    15
3     ('a', 3)           ('n', 2)    13
4     ('c', 1)           ('h', 1)    8
5     ('c', 2)           ('a', 2)    2
6     ('c', 3)           ('h', 2)    9
7     ('d', 1)           ('n', 3)    14
8     ('h', 1)           ('h', 3)    10
9     ('h', 2)           ('h', 4)    11
10    ('h', 3)           ('d', 1)    7
11    ('h', 4)           ('a', 3)    3
12    ('n', 1)           ('t', 2)    16
13    ('n', 2)           ('t', 3)    17
14    ('n', 3)           ('t', 4)    18
15    ('t', 1)           ('$', 1)    0
16    ('t', 2)           ('c', 1)    4
```

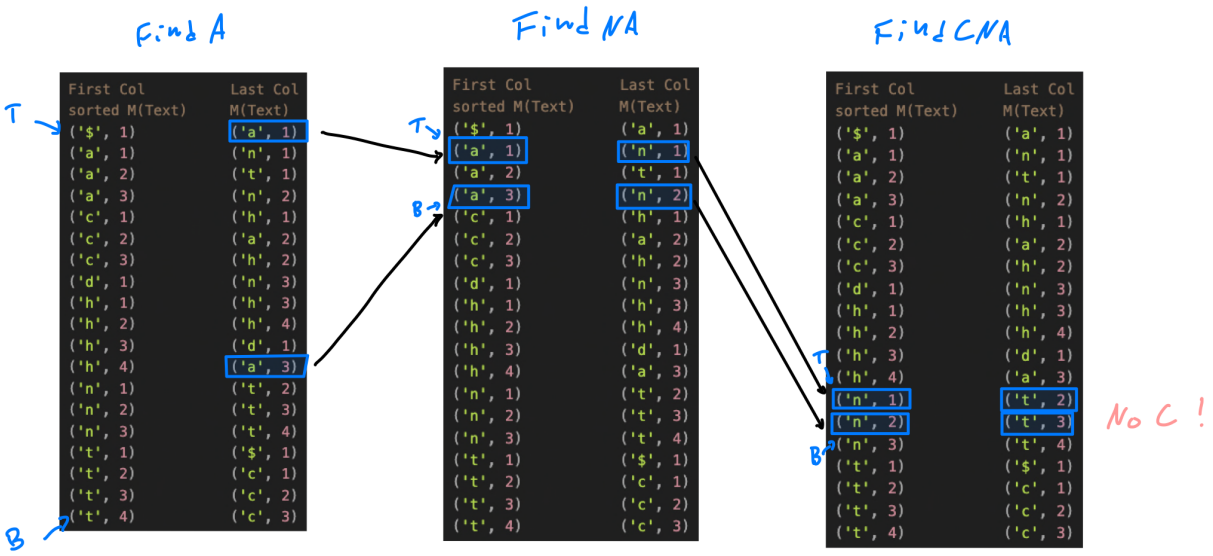
```
17    ('t', 3)          ('c', 2)    5
18    ('t', 4)          ('c', 3)    6
```

ctn



- T = 4
- B = 6
- 6 - 4 + 1 = 3
- ∴ `ctn` appears 3 times

cna



- T and B meet before we found `cna`
- ∴ `cna` does not appear