Todays Content:

a) Tries Intro

b) Insert / Search

Q) Given N strings & Q queries, for each query check if given query is in given N strings

Note: Assume max string length is l

Note: Insert/Delete/Search a string as key in hashset/hashmap TC: $O(l)$ // l is length of string.

Words:	Queries:
damp	data ✓
dark	draw ✓
data	drew ✓
drake	dump ✗
drawn	drawed ✗
drew	
dried	
drunk	
draw	

Idea1: Insert all N strings in hashset

For each query search in hashset.

TC: $O(N*l + Q*l)$ SC: $O(N*L)$

Idea2: Tries: Retrieval: To Retrive data

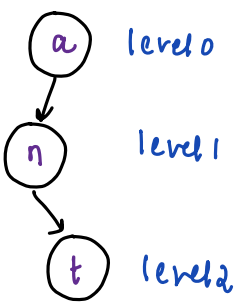
a) Tries is a Tree

b) Insert char by char level by level?

TC: $O(N*L + Q*L)$ SC: $O(N*L)$

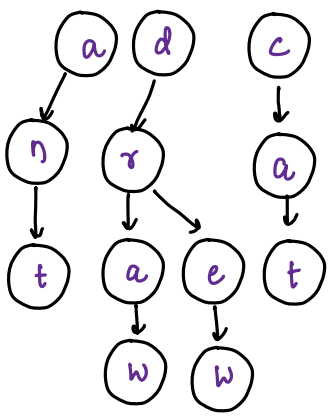
Obs: More words we insert, more optimized it is.

Data: ant

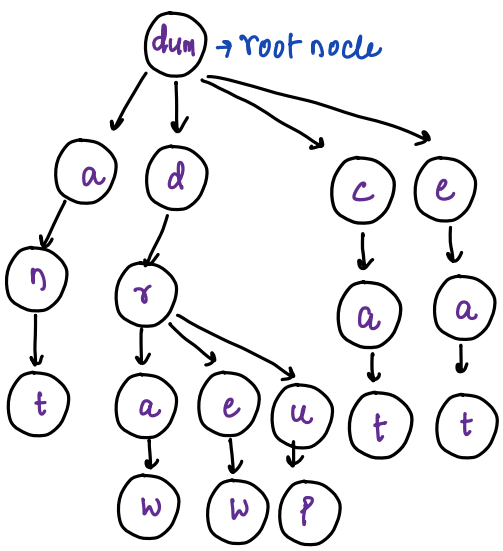


Data in Tries:

1. ant ✓
2. draw ✓
3. drew ✓
4. cat ✓

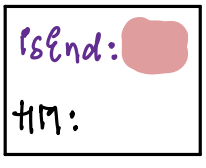


1. ant ✓
2. draw ✓
3. drew ✓
4. cat ✓
5. eat ✓
6. drup ✓



Note: Class structure of each Node

```
Class Node {
    bool isEnd; // will indicate if a word is ending at that node
    Hashmap<char, Node> hm;
     ↳ key   ↳ value : node address
    Node() {
        isEnd = false
        hm = new Hashmap<>();
    }
}
```



Words:

anki ✓ dde

answ ✓

ansk ✓

apq ✓

damp ✓

damper ✓

dam

0 1 2 3 4

date d

answ

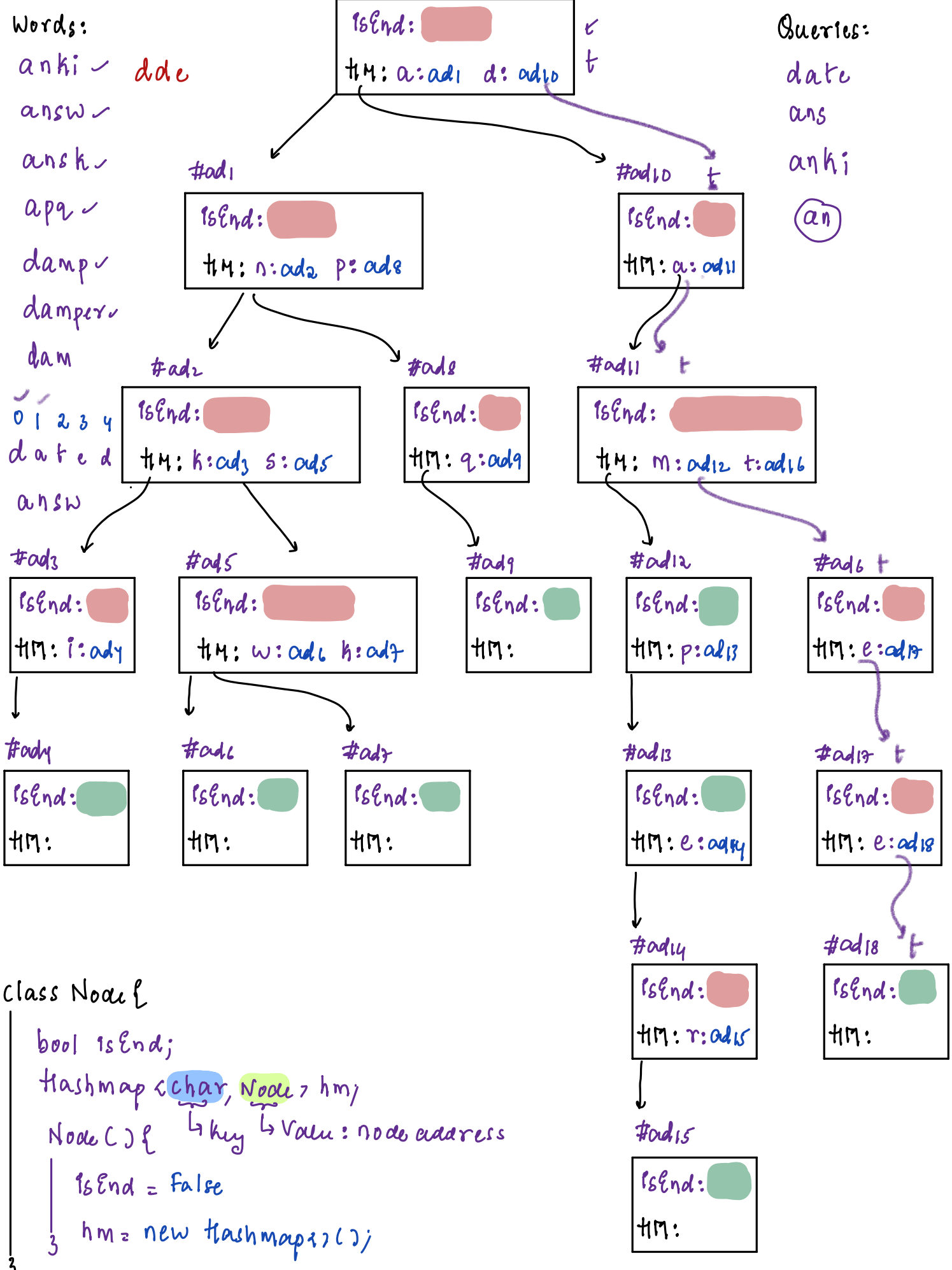
Queries:

date

ans

anki

an




```
Class Node {
    bool isEnd;
    Hashmap<char, Node> hm;
     ↳ key  ↳ value : node address
    Node() {
        isEnd = false
        hm = new Hashmap<>();
    }
}
```

```
void data(String inp[], String Que[]){ TC:
    int N = inp.length;                    SC:
    Node root = new Node();
    for(int i=0; i<N; i++){
        String word = inp[i];
        insert(root, word)
    }

    int Qs = Que.length;
    for(int i=0; i<Qs; i++){
        String word = Que[i]
        if( search(root, word)){
            print(present)
        }
        else{ print(absent)
    }
}
```

```
void insert(Node root, String data){ TC: O(l)   SC: O(l)
                                       ↳ // length of String:
    Node t = root;
    int l = data.length;
    for(int i=0; i<l; i++){
        // check for ith char in data.
        char ch = data.charAt(i)
        if( t.hm.containsKey(ch) == True){
            t = t.hm.get(ch) // update t to node
            // for char data[i] get address of node & update t
        }
        else{
            Node nn = new Node(); // Create new node.
            t.hm.put(ch, nn); // add in hashmap
            t = nn; // update temp to new node
        }
    }
    t.isEnd = True;
}
```

```
bool search(Node root, String data){ TC: O(L)
    Node t = root;
    int L = data.length;
    for(int i = 0; i < L; i++){
        // check for ith char in data.
        char ch = data.charAt(i)
        if( t.hm.containsKey(ch) == True){
            t = t.hm.get(ch) // update t to node
        else{ return false}
    }
    return t.isEnd; // if word is ending at that node = True
                    // if word is not ending that node = False;
}
```

⇒ Word:

- friend
- In your word document
  - a) all correct words are inserted in trie at start
  - b) Every word we write is a query, it searches in trie, if not there, it highlights

Note:

Tries = Prefix Trees

Q: Searching String in a bunch of, One Idea = Tries

Given N strings & Q queries, for each query check how many strings has prefix substring as given query // Q: Prefix search from bunch of strings?

Note: Assume max string length is l

Words:	Queries:
anki	dat : 4
answ	an : 4
ansk	apq : 1
apq	damp : 2
damp	
damper	
data	
date	
dated	
ankj	
date	

Idea: 1. Counting nodes with isEnd = True

TC: SC:

Note: Class structure of each Node

```
class Node {
    int cnt; // No: of words passing from that node
    HashMap<char, Node> hm; // For a node, since we are not aware no: of
    Node() {                    children we take a hashmap.
        cnt = 1;
        hm = new HashMap<>();
    }
}
```

Words:

anki ✓

answ ✓

ansk ✓

ansk ✓

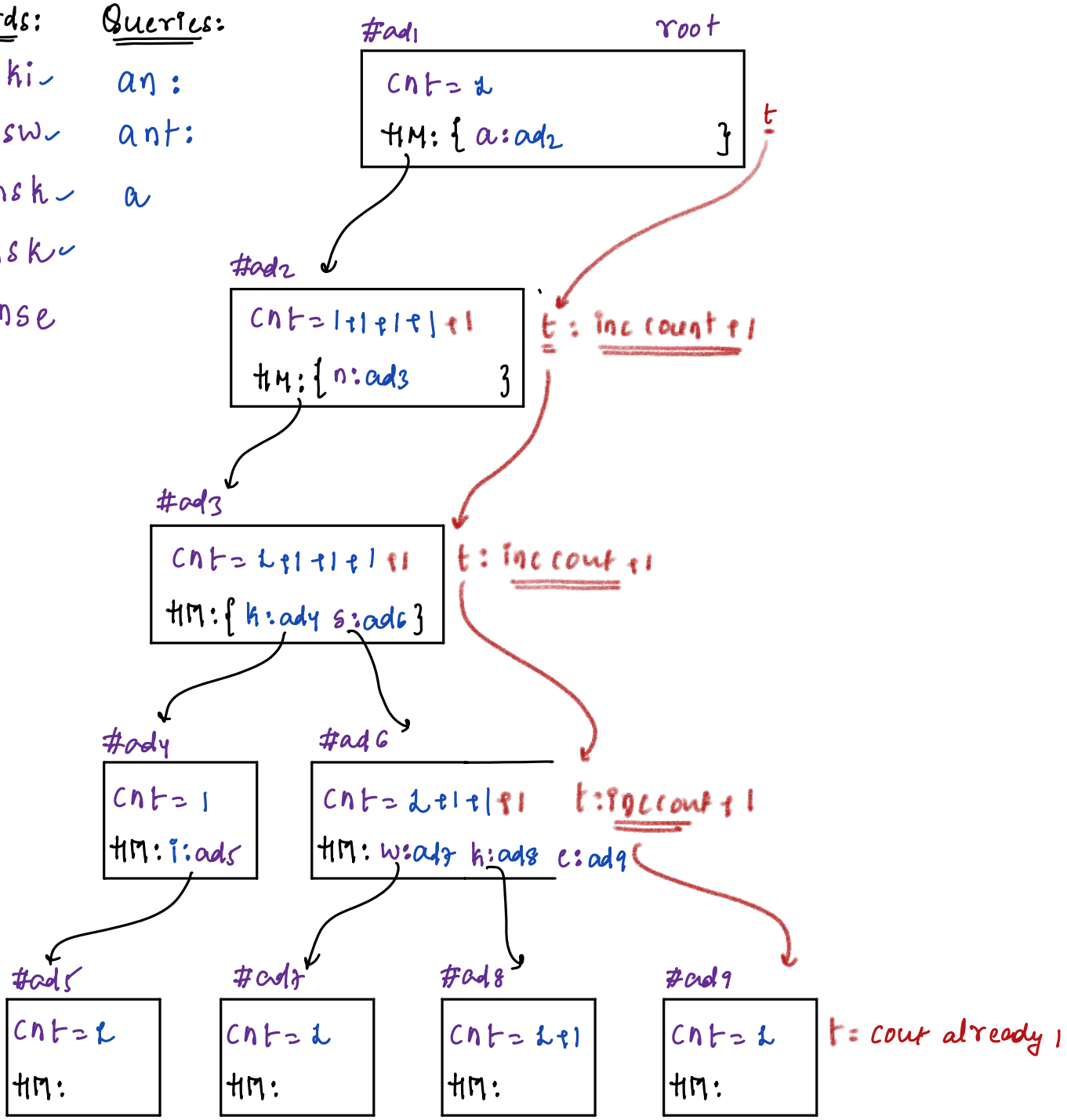
anse

Queries:

an :

ant:

a



```
void data(String inp[], String Que[]){
    int N = inp.length;
    Node root = new Node();
    for(int i=0; i<N; i++){
        String word = inp[i];
        insert(root, word)
    }
    int Q = Que.length;
    for(int i=0; i<Q; i++){
        String word = Que[i]
        print( freq(root, word)
    }
}
```

```
void insert(Node root, String data){ TC: O(l)  SC: O(l)
                                       ↳ // length of String
    Node t = root;
    int l = data.length;
    for(int i=0; i<l; i++){
        // check for ith char in data.
        char ch = data.charAt(i)
        if( t.hm.containsKey(ch) == True){
            t = t.hm.get(ch) // update t to node
            t.c = t.c+1;
        }
        else{
            Node nn = new Node(); // create new node.
            t.hm.put(ch, nn); // add in hashmap
            t = nn; // update temp to new node
        }
    }
}
```

```
int freq(Node root, String data){ TC: O(l)
    Node t = root;
    int l = data.length;
    for(int i = 0; i < l; i++){
        // check for ith char in data.
        char ch = data.charAt(i)
        if( t.hm.containsKey(ch) == True){
            t = t.hm.get(ch) // update t to node
        else{ return 0}
    }
    return t.c; // it will return no: of words passing that node.
}
```

Hint:

a. Search for a string In a bunch of String

b. Search for a string as Prefix In a bunch of strings

c. Search for a string as Suffix In a bunch of strings

d. Get freq of strings as Prefix/Suffix In a bunch of strings