

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.

Word:

Queries:

Ideal: Insert all N strings in hashset

for each query search in hashset.

damp

data ✓

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

dark

draw ✓

Ideal2: Tries: Retrieval: To Retrieve data

data

drew ✓

a) Tries is a Tree

drake

dump ✗

b) Insert char by char level by level?

drawn

drawed ✗

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

drew

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

Data: ant

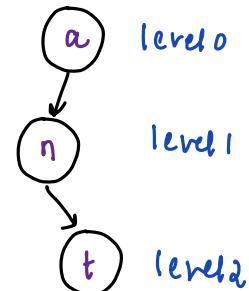
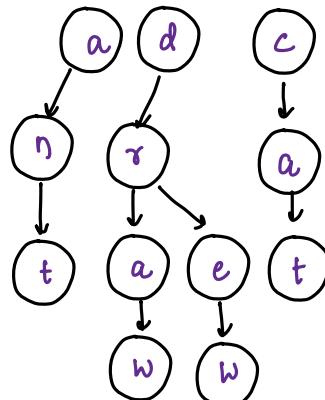
Data in Tries:

1. ant -

2. draw -

3. drew ✓

4. cat ✓



1. ant ✓

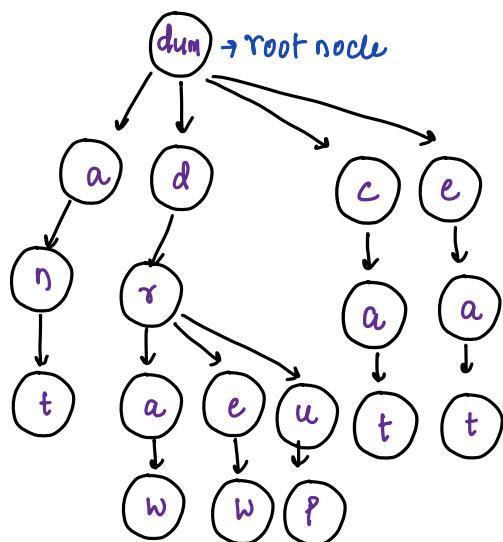
2. draw ✓

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;

 Node() { ↳ key ↳ Value: node address

 isEnd = false

 }; hm = new hashmap<>();

}

isEnd:



hm:

isEnd:



hm:

Words:

anki ~ date

answ ~

ansh ~

apq ~

damp ~

damper ~

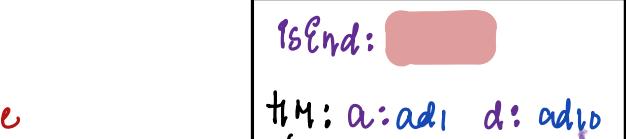
dam ~

0 1 2 3 4

date d

answ

#ad1



r
t

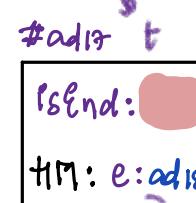
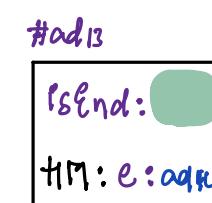
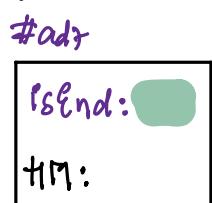
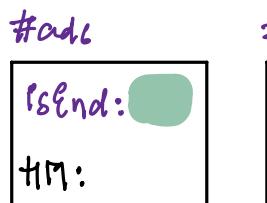
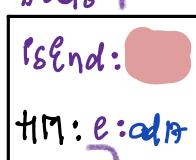
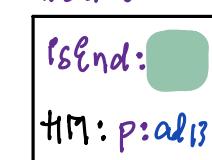
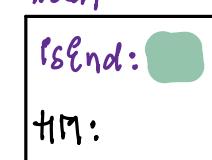
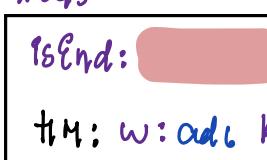
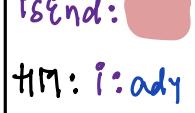
Queries:

date

ans

anki

(an)



#ad2

#ad3

#ad13

#ad17

#ad4

#ad5

#ad14

#ad18

#ad6

#ad7

#ad15

#ad19

#ad8

#ad9

#ad16

#ad20

#ad10

#ad11

#ad17

#ad21

#ad12

#ad13

#ad18

#ad22

#ad14

#ad15

#ad19

#ad23

#ad16

#ad17

#ad20

#ad24

#ad18

#ad19

#ad21

#ad25

#ad20

#ad21

#ad22

#ad26

#ad22

#ad23

#ad24

#ad27

#ad24

#ad25

#ad26

#ad28

#ad26

#ad27

#ad28

#ad29

#ad28

#ad29

#ad29

#ad30

#ad30

#ad31

#ad30

#ad31

#ad31

#ad32

#ad31

#ad32

#ad32

#ad33

#ad32

#ad33

#ad33

#ad34

#ad33

#ad34

#ad34

#ad35

#ad34

#ad35

#ad35

#ad36

#ad35

#ad36

#ad36

#ad37

#ad36

#ad37

#ad37

#ad38

#ad37

#ad38

#ad38

#ad39

#ad38

#ad39

#ad39

#ad40

#ad39

#ad40

#ad40

#ad41

#ad40

#ad41

#ad41

#ad42

#ad41

#ad42

#ad42

#ad43

#ad42

#ad43

#ad43

#ad44

#ad43

#ad44

#ad44

#ad45

#ad44

#ad45

#ad45

#ad46

#ad45

#ad46

#ad46

#ad47

#ad46

#ad47

#ad47

#ad48

#ad47

#ad48

#ad48

#ad49

#ad48

#ad49

#ad49

#ad50

#ad49

#ad50

#ad50

#ad51

#ad50

#ad51

#ad51

#ad52

#ad51

#ad52

#ad52

#ad53

#ad52

#ad53

#ad53

#ad54

#ad53

#ad54

#ad54

#ad55

#ad54

#ad55

#ad55

#ad56

#ad55

#ad56

#ad56

#ad57

#ad56

#ad57

#ad57

#ad58

#ad57

#ad58

#ad58

#ad59

#ad58

#ad59

#ad59

#ad60

#ad59

#ad60

#ad60

#ad61

#ad60

#ad61

#ad61

#ad62

#ad61

#ad62

#ad62

#ad63

#ad62

#ad63

#ad63

#ad64

#ad63

#ad64

#ad64

#ad65

#ad64

#ad65

#ad65

#ad66

#ad65

#ad66

#ad66

#ad67

#ad66

#ad67

#ad67

#ad68

#ad67

#ad68

#ad68

#ad69

#ad68

#ad69

#ad69

#ad70

#ad69

#ad70

#ad70

#ad71

#ad70

#ad71

#ad71

#ad72

#ad71

#ad72

#ad72

#ad73

#ad72

#ad73

#ad73

#ad74

#ad73

#ad74

#ad74

#ad75

#ad74

#ad75

#ad75

#ad76

#ad75

#ad76

#ad76

#ad77

#ad76

#ad77

#ad77

#ad78

#ad77

#ad78

#ad78

#ad79

#ad78

#ad79

#ad79

#ad80

#ad79

#ad80

#ad80

#ad81

#ad80

#ad81

#ad81

#ad82

#ad81

#ad82

#ad82

#ad83

#ad82

#ad83

#ad83

#ad84

#ad83

#ad84

#ad84

#ad85

#ad84

#ad85

#ad85

#ad86

#ad85

#ad86

#ad86

#ad87

#ad86

#ad87

#ad87

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

```
void insert(Node root, String data){ TC: O(l) SC: O(l)
```

```
    Node t = root; // length of string:  
    int l = data.length;  
    for(int i=0; i < l; i++) {  
        // check for  $i^{\text{th}}$  char in data.  
        char ch = data.charAt(i);  
        if(t.hm.containsKey(ch) == true) {  
            t = t.hm.get(ch); // update t to node  
        } 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

Idea: 1. Counting nodes with is End = True

answ an : 4

ansh app : 1

app damp : 2

TC:

SC:

damp

damper

data

date

dated

anki

date

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: Queries:

anki.

an;

#ad

root

answ.

ant:

CNT = 2

ans k -

a

#adz ✓

t := inc(count + 1)

ansku

Cnt = 1 + 1 + 1 + 1 + 1

H.M.; { n: add3 3

t := inc(count + 1)

#ad3

t: Inc count +1

CNT = Lg1 + 1 + 1

#ady

#ad 6

Cnt = 1

$$CnT = 2 + 1 + 1 \cancel{+ 1}$$

topic count + 1

4

#cad3

Fad. 8

End 9

CNT = 2

$$Cn t = \lambda$$

$$Cn t = 2 + 1$$

Cnt = 2

t: count already 1

```

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 Qs = que.length;
    for(int i=0; i < Qs; i++) {
        String word = que[i];
        print(freq(root, word));
    }
}

```

```

void insert(Node root, String data) { Tc: O(l) Sc: O(l)
    Node t = root;
    int l = data.length();
    // // length of string:
    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 {
            t.c = t.c + 1;
            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