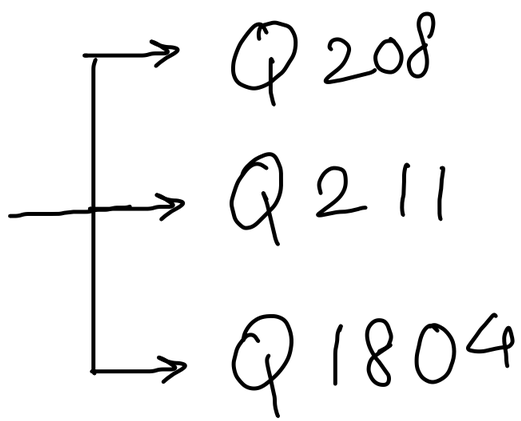
TRIE IMPLEMENTATION



Advantages & Disadvantages of Trie over Hashing

Applications of Tries

- Autocomplete System
- Word Suggestion System
- Spellchecker / Autocorrect
- Prefix Matching
- Searching { PhoneDirectory, Dictionary }

Dictionary Problems

Q720 Longest Word in Dictionary

Q677 Weighted Prefix Search

Q648 Replace Word with Prefix

Q 14 Longest Common Prefix

Q676 Magic Dictionary

IB Shortest Unique Prefix

GFG (Q1698) Count Distinct Substrings

GFG Most Frequent Word

Q1268 Search Suggestion Systems

Q1032 Stream of Characters

Q212 Word Search - II

Q 336 Palindrome Pairs

Q745 Prefix & Suffix Search

XOR Problems

Q421 Maximum XOR Pair - I

Q1707 Maximum XOR Pair - II

Q1803 XOR Pairs in Range

GFG Subarrays with XOR < K

GFG Unique Rows in Boolean Matrix

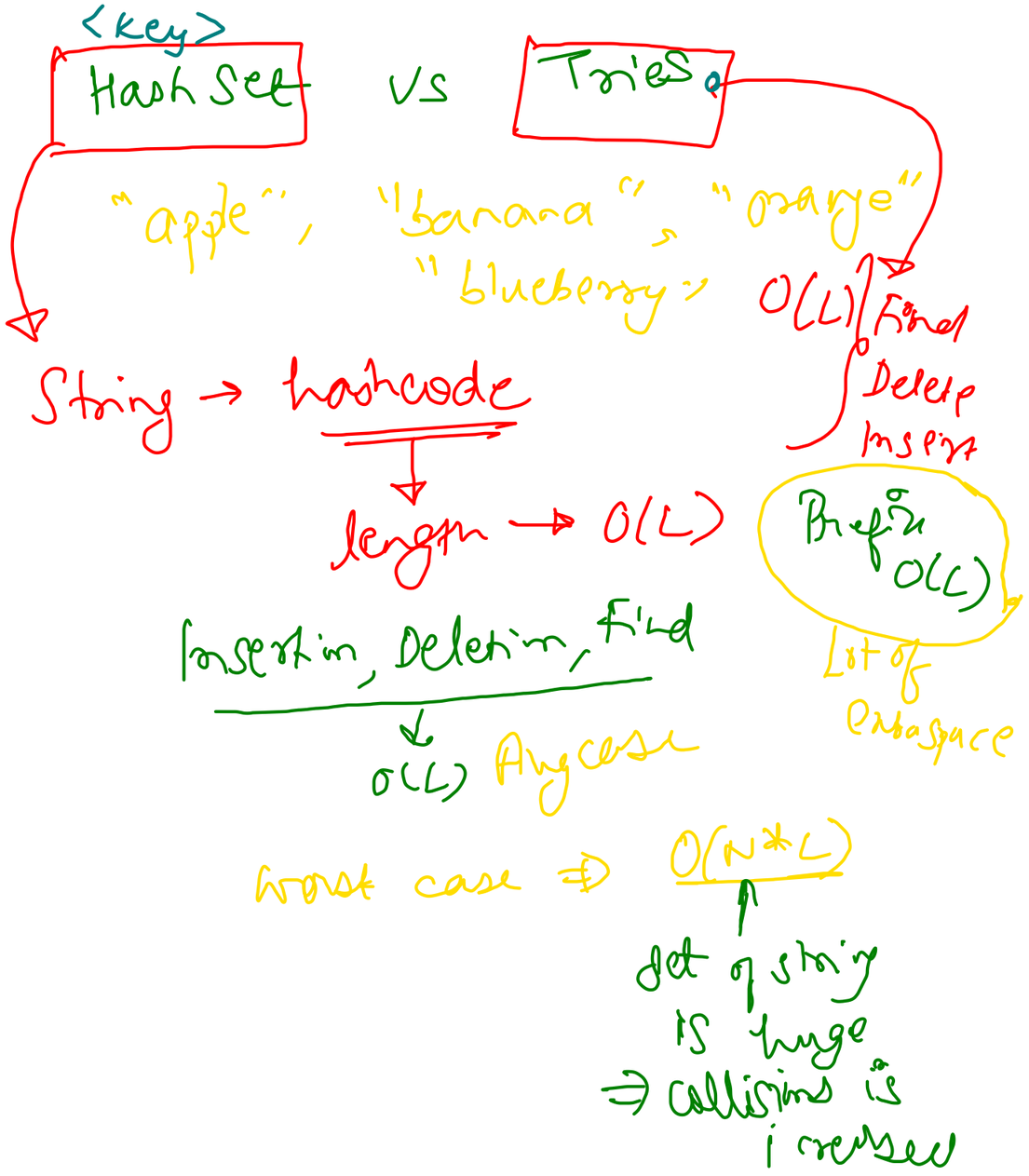
Bit Manipulation Basics

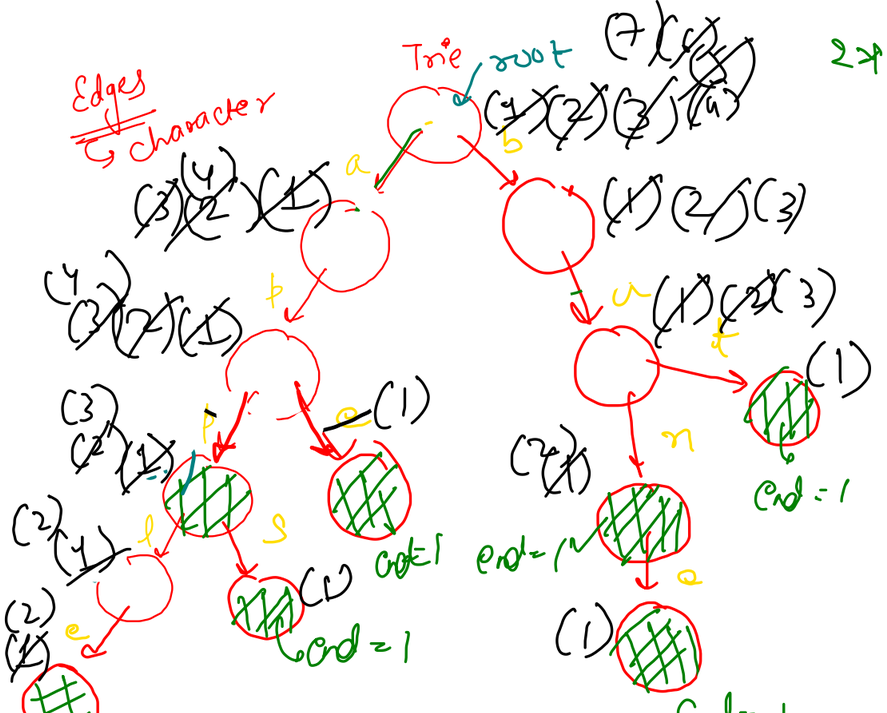
- Set, get, Unset
- left shift, rightshift
- XOR Properties

A **trie** (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

- `Trie()` Initializes the trie object.
- `void insert(String word)` Inserts the string `word` into the trie.
- `boolean search(String word)` Returns `true` if the string `word` is in the trie (i.e., was inserted before), and `false` otherwise.
- `boolean startsWith(String prefix)` Returns `true` if there is a previously inserted string `word` that has the prefix `prefix`, and `false` otherwise.



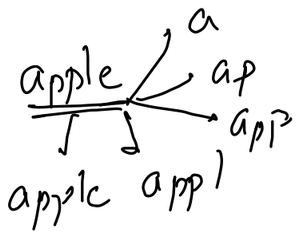


2 => insert("apple")
insert("bane")
insert("bad")
insert("ban")
insert("ap")
insert("apps")

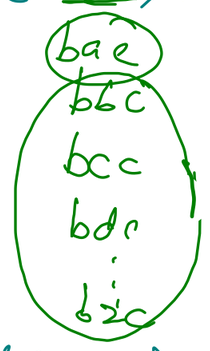
search("apple") True
search("bank") False
search("ap") False

prefix("apple") True
prefix("bank") False
prefix("ap") True

iSerch("ape")



search("b.c") F



search("b.t")

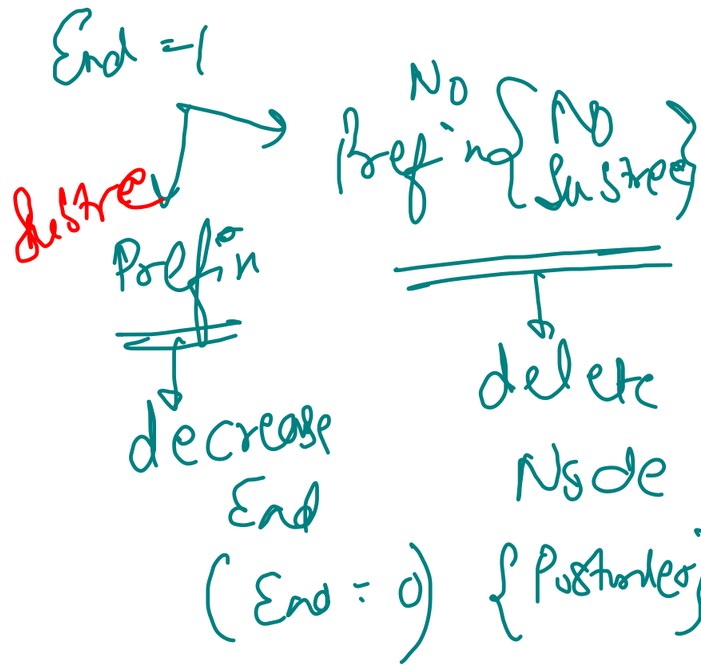
bat, bbt, bct, ... bzt

printprefix("ap") DFS: Broader

{ ape, app, apple, apples, apps }

delete("ban")

End > 2 : => decrease End



```java
public static class Node{
    private Node[] children = new Node[26];
    private boolean isEnd = false;

    public boolean contains(char ch){
        return (children[ch - 'a'] != null);
    }

    public Node get(char ch){
        return children[ch - 'a'];
    }

    public void set(char ch){
        children[ch - 'a'] = new Node();
    }

    public boolean getEnd(){
        return isEnd;
    }

    public void setEnd(){
        isEnd = true;
    }
}
```

```java
public void insert(String word) {
    Node curr = root;
    for(int i=0; i<word.length(); i++){
        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            curr.set(ch);

        curr = curr.get(ch);
    }

    curr.setEnd();
}

public boolean search(String word) {
    Node curr = root;
    for(int i=0; i<word.length(); i++){
        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            return false;

        curr = curr.get(ch);
    }

    return curr.getEnd();
}

public boolean startsWith(String word) {
    Node curr = root;
    for(int i=0; i<word.length(); i++){
        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            return false;

        curr = curr.get(ch);
    }

    return true;
}
```

Worst case

Take →

For 1 String { Insert, Delete, Find, Prefix } → O(L)

Space for entire Trie → O(NAL)

Q211 Leetcode

```
public boolean search(String word, int idx, Node curr){
    if(idx == word.length()){
        return curr.getEnd();
    }

    char ch = word.charAt(idx);

    if(ch != '.'){
        if(curr.contains(ch) == false) return false;
        return search(word, idx + 1, curr.get(ch));
    }

    for(char chn = 'a'; chn <= 'z'; chn++){
        if(curr.contains(chn) == false) continue;

        if(search(word, idx + 1, curr.get(chn)))
            return true;
    }

    return false;
}

public boolean search(String word) {
    return search(word, 0, root);
```

worst case

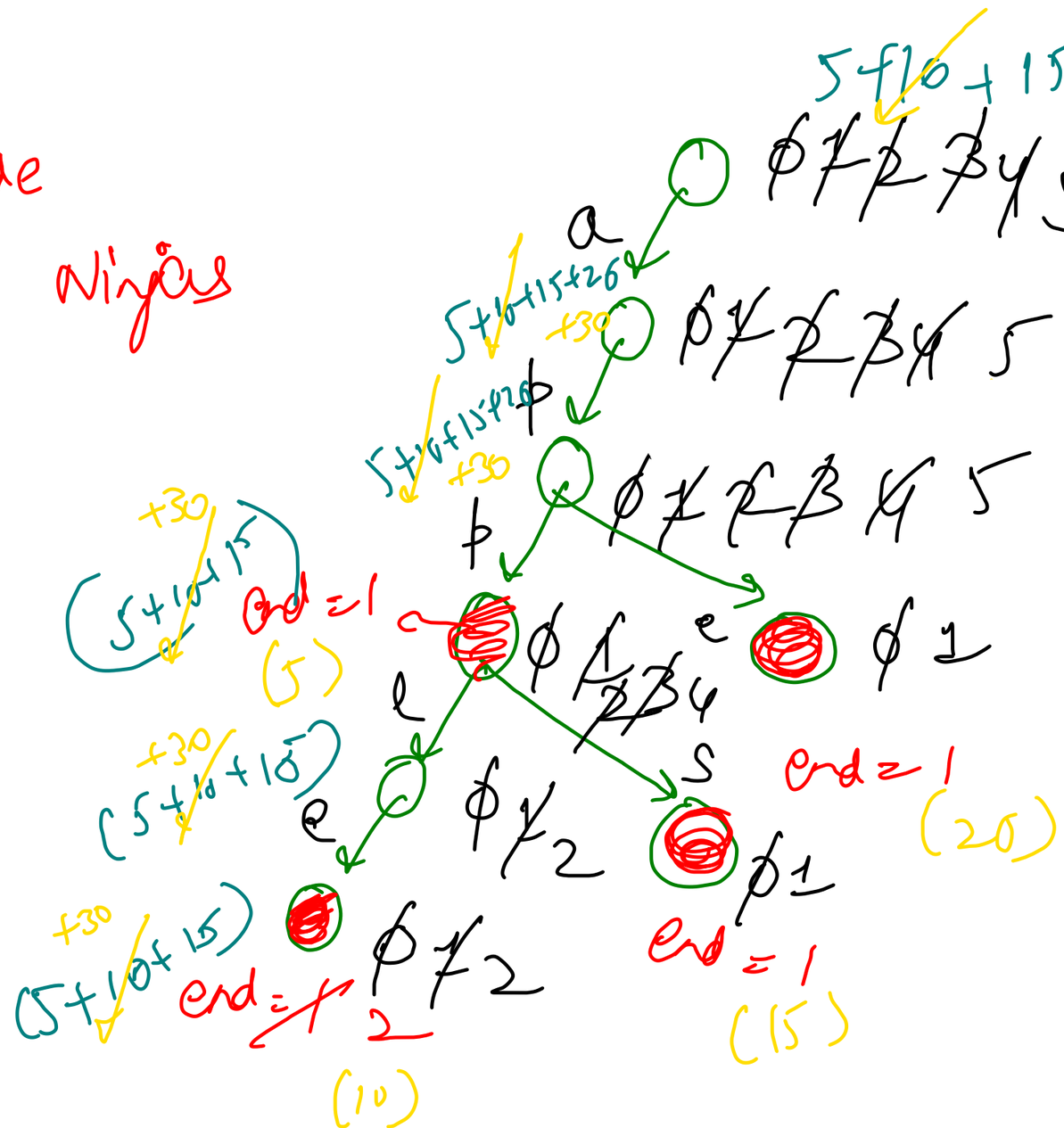
$(26)^h$

↳ All are dots (.)

Q1804

Leetcode

→ Coding Ninjas



insert("bapple") * 2

insert("ape")

insert("apps")

insert("app")

freq("apple") → 2

freq("apples") → 0

freq("app") → 1

pref("apple") → 2

pref("app") → 4

```java
public static class Node{
    private Node[] children = new Node[26];
    private int isEnd = 0;
    private int prefixCount = 0;

    public boolean contains(char ch){
        return (children[ch - 'a'] != null);
    }

    public Node get(char ch){
        return children[ch - 'a'];
    }

    public void set(char ch){
        children[ch - 'a'] = new Node();
    }

    public int getFreq(){
        return isEnd;
    }

    public int getPref(){
            return prefixCount;
    }

    public void increaseFreq(){
        isEnd++;
    }

    public void decreaseFreq(){
        isEnd--;
    }

    public void increasePref(){
        prefixCount++;
    }

    public void decreasePref(){
        prefixCount--;
    }
}
```

```java
Node root;
public Trie() {
    root = new Node();
}

public void insert(String word) {
    Node curr = root;

    for(int i=0; i<word.length(); i++){
        curr.increasePref();

        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            curr.set(ch);

        curr = curr.get(ch);
    }

    curr.increasePref();
    curr.increaseFreq();
}
```

O(L)

```java
public int countWordsEqualTo(String word) {
    Node curr = root;

    for(int i=0; i<word.length(); i++){
        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            return 0;

        curr = curr.get(ch);
    }

    return curr.getFreq();
}

public int countWordsStartingWith(String word) {
    Node curr = root;

    for(int i=0; i<word.length(); i++){
        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            return 0;

        curr = curr.get(ch);
    }

    return curr.getPref();
}
```

O(L)

O(L)

```java
public void erase(String word) {
    if(countWordsEqualTo(word) == 0){
        return;
    }

    Node curr = root;

    for(int i=0; i<word.length(); i++){
        curr.decreasePref();
        char ch = word.charAt(i);
        curr = curr.get(ch);
    }

    curr.decreasePref();
    curr.decreaseFreq();
}
```

(Q677) Map Sum Pairs

```
public static class Node{
    private Node[] children = new Node[26];
    private int val = 0;
    public int pref = 0;

    public Node get(char ch){
        return children[ch - 'a'];
    }

    public int getVal(){
        return val;
    }

    public void add(char ch){
        children[ch - 'a'] = new Node();
    }

    public void setVal(int val){
        this.val = val;
    }

    public boolean contains(char ch){
        return (children[ch - 'a'] != null);
    }
}
```

```
Node root;
public MapSum() {
    root = new Node();
}

public int search(String word){
    Node curr = root;
    for(int i=0; i<word.length(); i++){
        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            return 0;

        curr = curr.get(ch);
    }
    return curr.getVal();
}
```

```
public void insert(String word, int val) {
    int oldVal = search(word);

    Node curr = root;

    for(int i=0; i<word.length(); i++){
        curr.pref += (val - oldVal);

        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            curr.add(ch);

        curr = curr.get(ch);
    }

    curr.pref += (val - oldVal);
    curr.setVal(val);
}
```

Weighted Prefix Search

```
public int sum(String word) {
    Node curr = root;
    for(int i=0; i<word.length(); i++){
        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            return 0;

        curr = curr.get(ch);
    }
    return curr.pref;
}
```