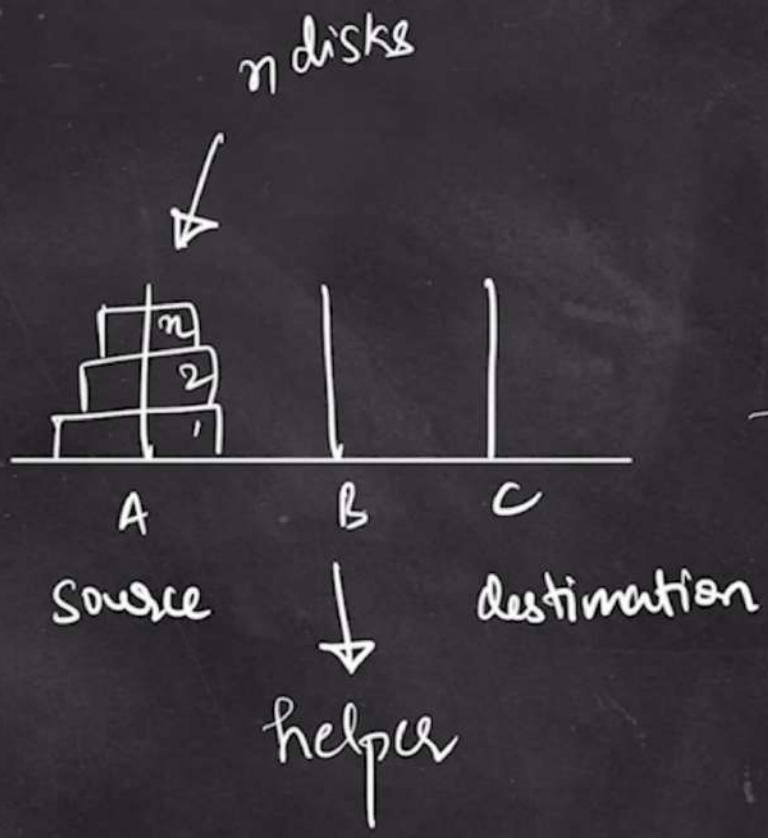
Qs. Tower of Hanoi

RULES

1. Only one disk transferred in 1 step
2. Smaller disks are slways kept on top of larger disks

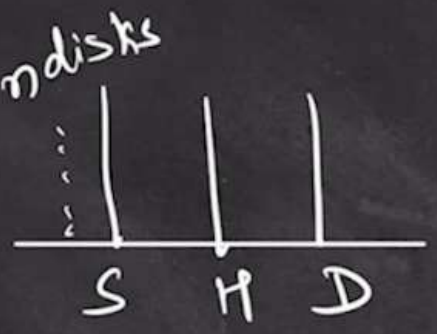


START

Steps

END

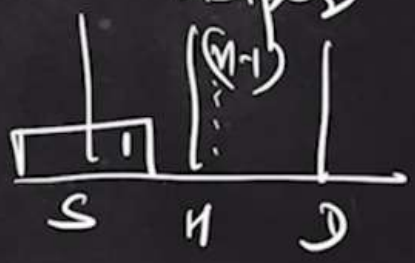
(1) "x"



(1) (n-1) disks

⇓

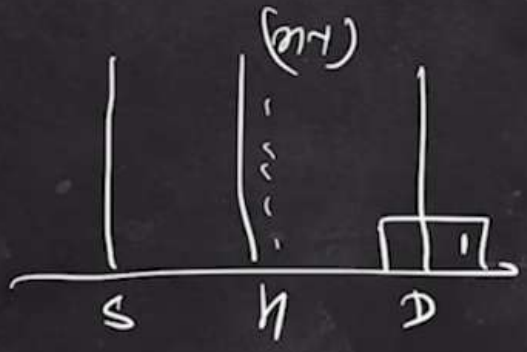
helper



"1"

(2) "1" disk $\underline{\underline{S \rightarrow D}}$

(3)

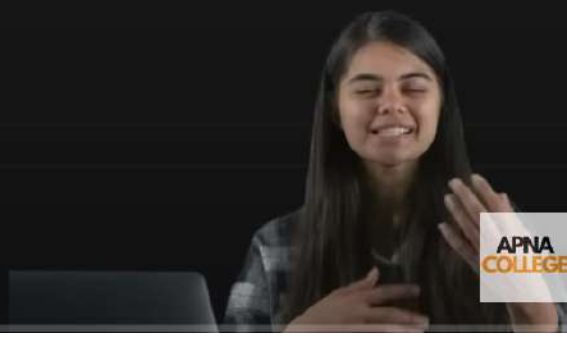


(n-1) disks

$H \xrightarrow{S} D$

helper

```java
public class Recursion2 {
    public static void towerOfHanoi(int n, String src, String helper, String dest) {
        if(n == 1) {
            System.out.println("transfer disk "+ n + " from "+src+" to "+dest);
            return;
        }
        towerOfHanoi(n-1, src, dest, helper);
        System.out.println("transfer disk "+ n + " from "+src+" to "+dest);
        towerOfHanoi(n-1, helper, src, dest);
    }

    Run | Debug
    public static void main(String args[]) {
        int n = 3;
        towerOfHanoi(n, "S", "H", "D");
    }
}
```



Qs. Tower of Hanoi

$$O(2^n - 1) \simeq O(2^n)$$

$$T(n) = 2T(n-1) + 1$$

Qs. Print a string in reverse

"abcd" → "dcba"

```java
public class Recursion2 {
    public static void printRev(String str, int idx) {
        if(idx == 0) {
            System.out.println(str.charAt(idx));
            return;
        }

        System.out.println(str.charAt(idx));
        printRev(str, idx-1);
    }
    Run | Debug
    public static void main(String args[]) {
        String str = "abcd";
        printRev(str, str.length()-1);
    }
```

Qs. Find the 1st & last occurance of an element in string

"abaacdaefaah" element

Recursion Class 2

Qs. Find the 1st & last occurance of an element in string

"abaacdaefaah" element = 'a'

idx=0 idx=10

```java
public class Recursion2 {
    public static int first = -1;
    public static int last = -1;

    public static void findOccurance(String str, int idx, char element) {
        if(idx == str.length()) {
            System.out.println(first);
            System.err.println(last);
            return;
        }
        char currChar = str.charAt(idx);
        if(currChar == element) {
            if(first == -1) {
                first = idx;
            } else {
                last = idx;
            }
        }

        findOccurance(str, idx+1, element);
```

```
public static void main(String args[]) {
    String str = "abaacdaefaah";
    findOccurance(str, 0, element);
}
```

Recursion Class 2

Qs. Check if an array is sorted (Strictly Increasing)

{1, 2, 3, 4, 5}

```java
public class Recursion2 {
    public static boolean isSorted(int arr[], int idx) {
        if(idx == arr.length-1) {
            return true;
        }

        if(arr[idx] < arr[idx+1]) {
            //array is sorted till now
            return isSorted(arr, idx+1);
        } else {
            return false;
        }
    }
    Run | Debug
    public static void main(String args[]) {
        int arr[] = {1, 3, 5};
        System.out.println(isSorted(arr, 0));
    }
```

```java
public class Recursion2 {
    public static boolean isSorted(int arr[], int idx) {
        if(idx == arr.length-1) {
            return true;
        }

        if(arr[idx] >= arr[idx+1]) {
            //array is unsorted
            return false;
        }
        return isSorted(arr, idx+1);
    }
    Run | Debug
    public static void main(String args[]) {
        int arr[] = {1, 3, 3};
        System.out.println(isSorted(arr, 0));
    }
}
```

Qs. Move all 'x' to the end of the string

"axbcxxd" → "abcdxxx"

newString

idx → curChar

count → track no. of x

newString →

newString → "

"a

```java
public static void moveAllX(String str, int idx, int count, String newString) {
    if(idx == str.length()) {
        System.out.println(newString);
        return;
    }

    char currChar = str.charAt(idx);
    if(currChar == 'x') {
        count++;
        moveAllX(str, idx+1, count, newString);
    } else {
        newString += currChar; //newString = newString + currChar
        moveAllX(str, idx+1, count, newString);
    }
}
Run | Debug
public static void main(String args[]) {
    String str = "axbcxxd";
    moveAllX(str, 0, 0, "");
```

Qs. Remove duplicates in a string

"abbccda"

Recursion Class 2

Qs. Remove duplicates in a string

"abbccda" → "abcd"

curchar − 'a' = ?

{ map INDEX }

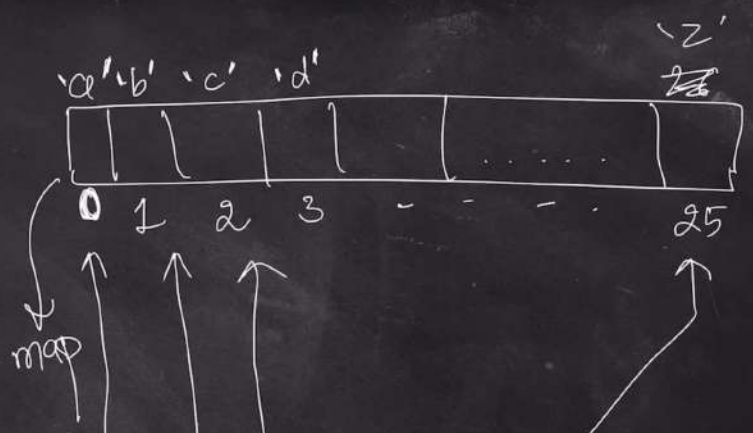
'a' − 'a' = 0

'b' − 'a' = 1

'c' − 'a' = 2

⋮

'z' − 'a' = 25



curr Char → True

newString ✗

curr char → False

newString ✓

map[pos] = True

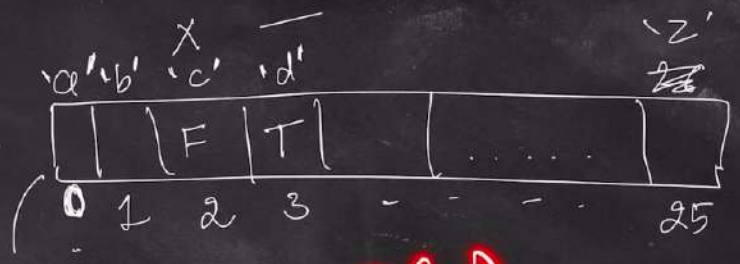
```java
public class Recursion2 {
    public static boolean[] map = new boolean[26];

    public static void removeDuplicates(String str, int idx, String newS
        if(idx == str.length()) {
            System.out.println(newString);
            return;
        }

        char currChar = str.charAt(idx);
        if(map[currChar - 'a']) {
            removeDuplicates(str, idx+1, newString);
        } else {
            newString += currChar;
            map[currChar - 'a'] = true;
            removeDuplicates(str, idx+1, newString);
        }
    }
```

```
Run | Debug
public static void main(String args[]) {
    String str = "abbccda";
    removeDuplicates(str, 0, "");
}
}
```

Recursion Class 2

Qs. Remove duplicates in a string

"abbccda" → "abcd"



O(n)

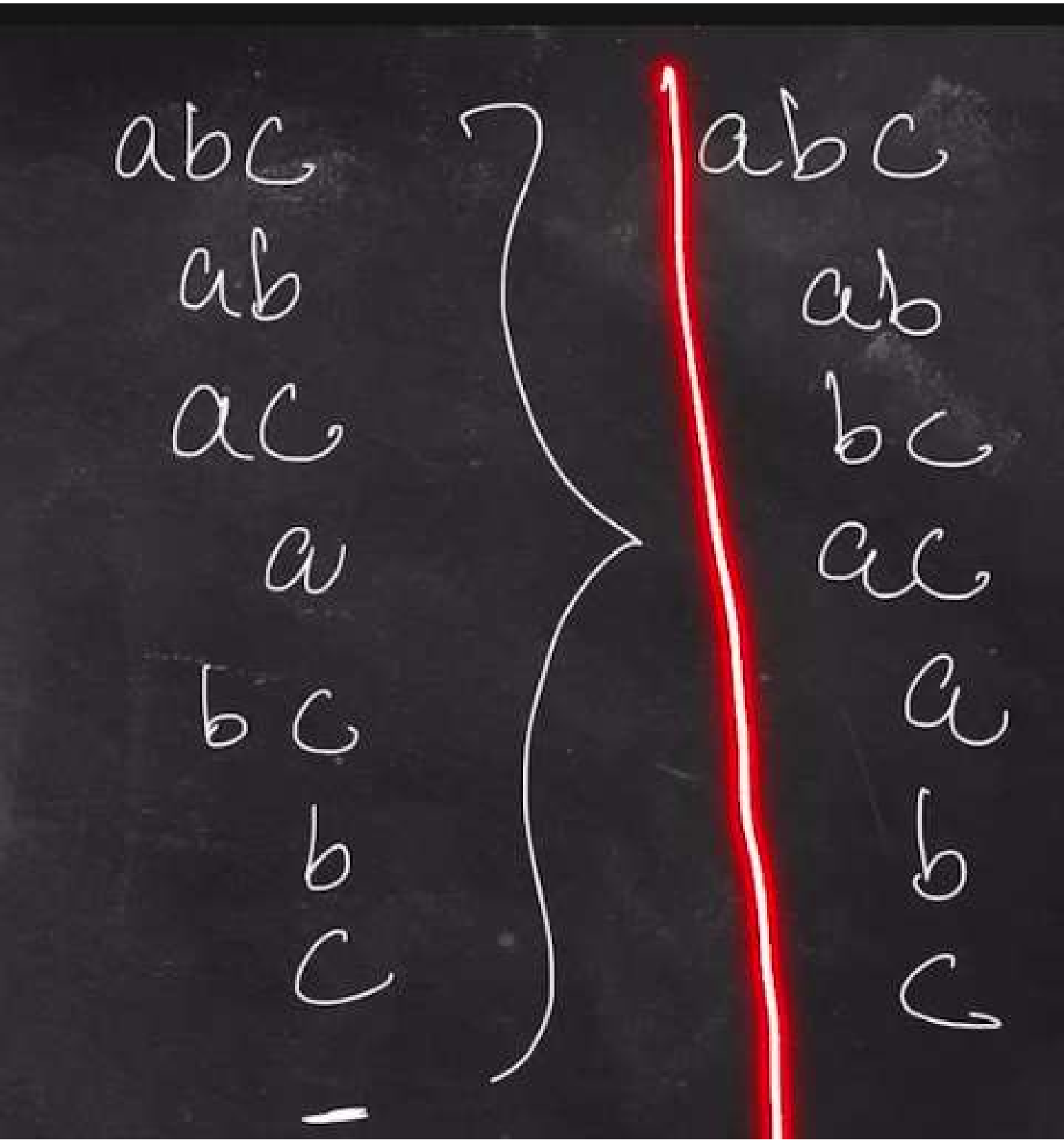
Recursion Class 2

Qs. Print all the subsequences of a string

"abc"

Qs. Print all the subsequences of a string

"abc"


abc
ab
ac
a
bc
b
c
abc
ab
bc
ac
a
b
c

Qs. Print all the subsequences of a string

"abc"

newString = " "

idx=0

choose call 1 newString + currChar

not choose call 2 newString

```java
public class Recursion2 {
    public static void subsequences(String str, int idx, String newString) {
        if(idx == str.length()) {
            System.err.println(newString);
            return;
        }
        char currChar = str.charAt(idx);

        //to be
        subsequences(str, idx+1, newString+currChar);

        void Recursion2.subsequences(String str, int idx, String newString)
        subsequences(str, idx+1, newString);
    }
    Run | Debug
    public static void main(String args[]) {
        String str = "abbccda";
        removeDuplicates(str, 0, "");
    }
```

```java
public class Recursion2 {
    public static void subsequences(String str, int idx, String newString) {
        if(idx == str.length()) {
            System.err.println(newString);
            return;
        }
        char currChar = str.charAt(idx);

        //to be
        subsequences(str, idx+1, newString+currChar);

        //or not to be
        subsequences(str, idx+1, newString);
    }
    Run | Debug
    public static void main(String args[]) {
        String str = "abbccda";
        removeDuplicates(str, 0, "");
    }
```

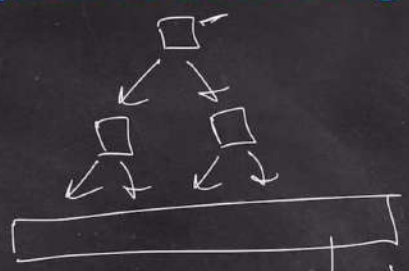
Recursion Class 2

Qs. Print all the subsequences of a string

"abc"

subseq = 2^n

$2 \times 2 \times 2 = 2^3$



total subseq

abc
ab
ac
a
bc
b
c

abc
ab
bc
ac
a
b
c

last level (nodes) = $2^n = (2^{n-1}) \times 2$

2^{n-1}

2^{n-2}

2^{n-3}

$\vdots$

1

function calls

GP - geometric progression

$2^n + 2^{n-1} + 2^{n-2} + 2^{n-3} + \dots 1$

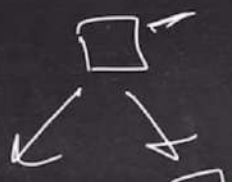
$= \frac{a(r^n - 1)}{(r-1)} = \frac{1(2^{n+1} - 1)}{(2-1)} = 2^{n+1} - 1$

Recursion Class 2

$O(2^n)$

Qs. Print all the subsequences of a string

"abc"



Recursion Class 2

Qs. Print all the subsequences of a string

"abc"

"aaa"

aaa

"aaa"

$\longrightarrow$ aaa

aa

aa

aa

a

a

a

$\underline{\underline{2^3}} = \underline{\underline{8}}$

Recursion Class 2

Qs. Print all the unique subsequences of a string

"aaa"

Q8. Print all unique subsequences of a string.

```
import java.util.HashSet;

public class Recursion2 {
   public static void printSubseq(String str, int idx, String res, HashSet<String>
allSubseq) {
       if(idx == str.length()) {
           if(allSubseq.contains(res)) {
               return;
           }
           allSubseq.add(res);
           System.out.println(res);
           return;
       }
```

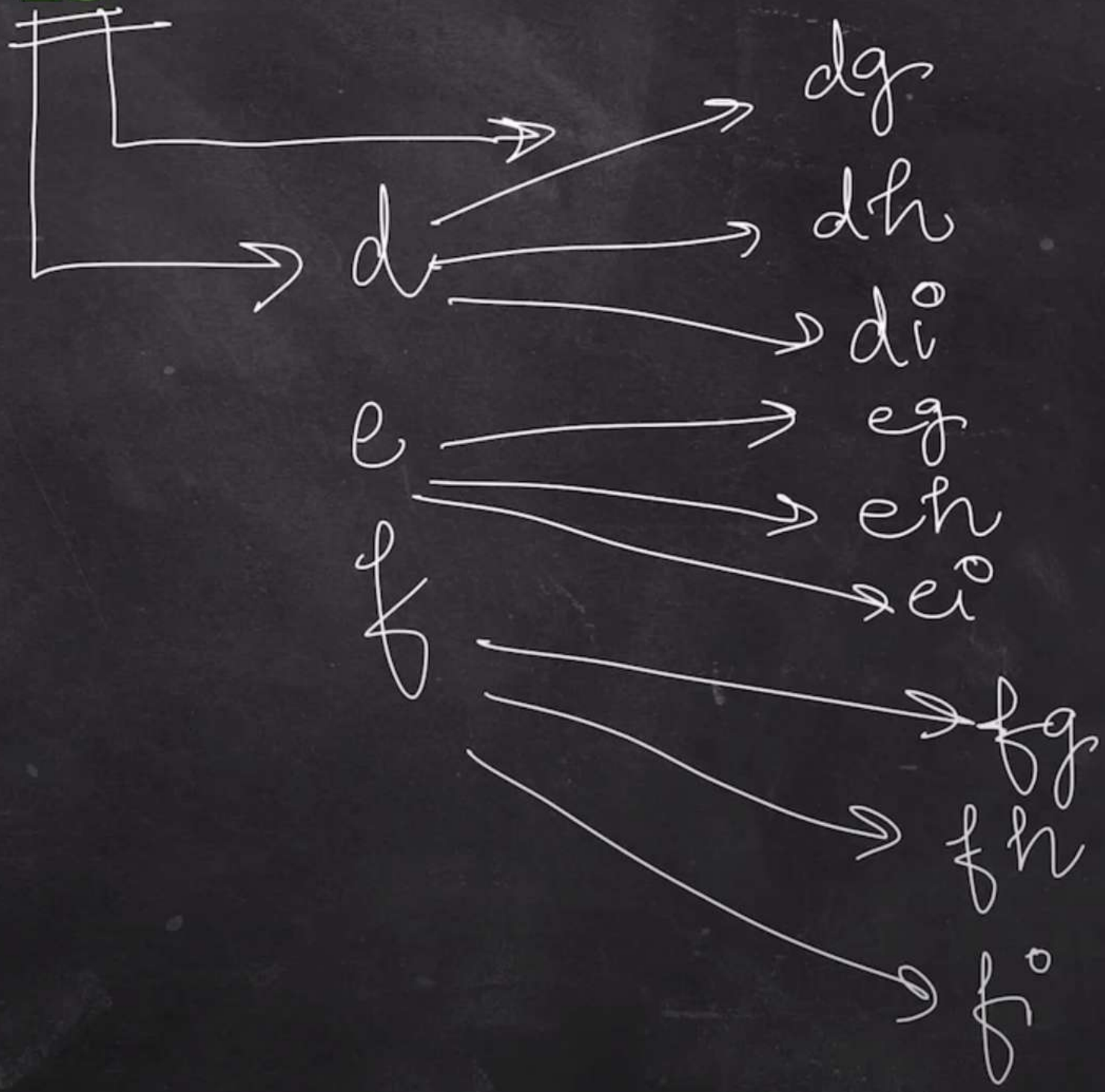
```
       //choose
       printSubseq(str, idx+1, res+str.charAt(idx), allSubseq);

       //don't choose
       printSubseq(str, idx+1, res, allSubseq);
   }
   public static void main(String args[]) {
       String str1 = "abc";
       String str2 = "aaa";
       HashSet<String> allSubseq = new HashSet<>();
       printSubseq(str2, 0, "", allSubseq);
   }
}
```

Recursion Class 2

Qs. Print keypad combination

0 -> .
1 -> abc
2 -> def
3 -> ghi
4 -> jkl
5 -> mno
6 -> pqrs
7 -> tu
8 -> vwx
9 -> yz

"23"



```java
import java.util.HashSet;

public class Recursion2 {
    public static String[] keypad = {".", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tu", "vwx", "yz"};

    public static void printComb(String str, int idx, String combination) {
        if(idx == str.length()) {
            System.out.println(combination);
            return;
        }
        char currChar = str.charAt(idx);
        String mapping = keypad[currChar - '0'];

        for(int i=0; i<mapping.length(); i++) {
            printComb(str, idx+1, combination+mapping.charAt(i));
        }
    }

    Run | Debug
    public static void main(String args[]) {
        String str = "23";
        printComb(str, 0, "");
    }
```

$$d(4^n)$$