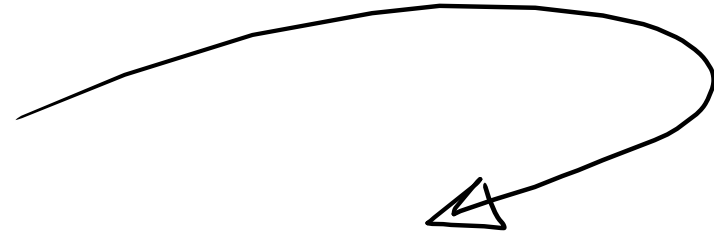


Recursion -

* STRING ENCODING

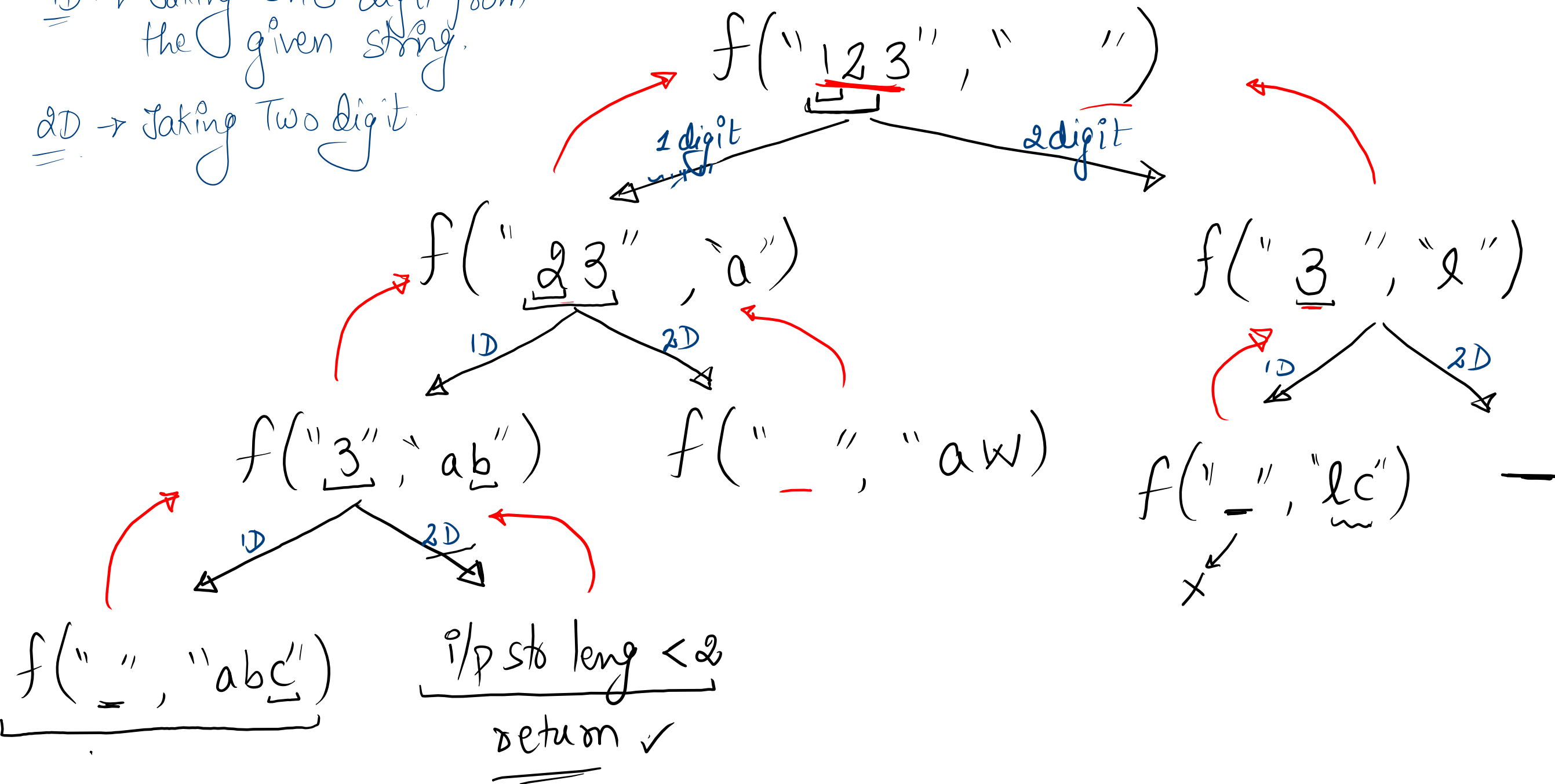
str \Rightarrow "126"

o/p \Rightarrow abf, lf, az



1D → Taking One digit from the given string.

2D → Taking Two digit.

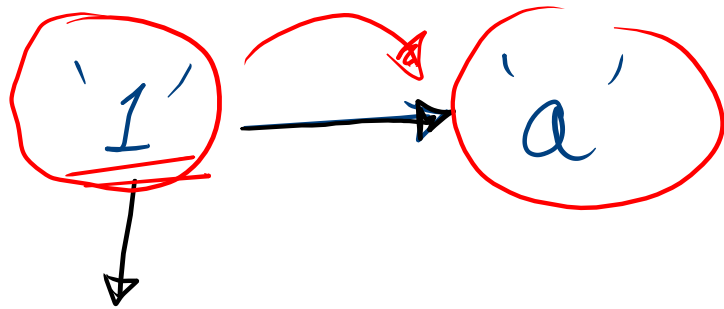


1 → 'a'
12 → 'l'
2 → 'b'
23 → 'w'
3 → 'c'

Output : abc, aw, lc

* One digit (1 to 9)

Ex -



char '1' - '0'
 int 1 + 'a'
'b' - 1 \Rightarrow a

Ex -

'4' \rightarrow 'd'
 \downarrow
'4' - '0'
4 + 'a' - 1 \Rightarrow 'e' x
 \rightarrow d

* two Digit (10 to 26)

23 \rightarrow 'l'

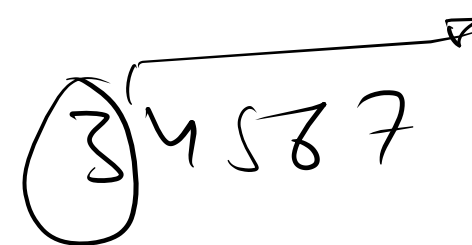
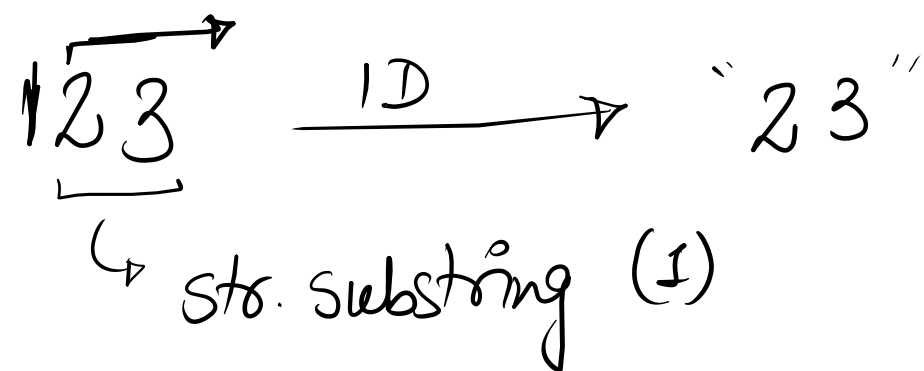
a + 1 \Rightarrow b , a + 2 \Rightarrow c
b + 1 \Rightarrow c

string "23" \rightarrow integer

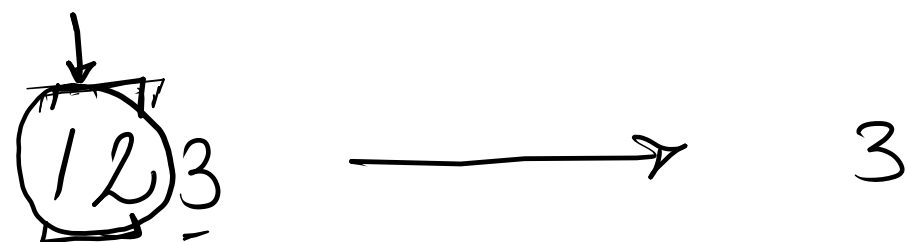
\downarrow string
 Integer.parseInt(23)

\downarrow
 int 23 + 'a' - 1
'm' x 'l' ✓

*



str.substring(1)



str.substring(i+2),

* Base Case -

if (idx == str.length())
 print(output);
 return;

```
public static void solve(String str , String output){
    //base case
    if(str.length() == 0){
        System.out.println(output);
        return ;
    }
    //process
    char first = str.charAt(0);
    if(first == '0') return ; // invalid

    int onedigit = first - '0' ;
    char currchar = (char)(onedigit + 'a' - 1) ;
    solve(str.substring(1),output+currchar);

    if(str.length() >= 2){
        int twodigit = Integer.parseInt(str.substring(0,2));
        if(twodigit >= 10 && twodigit <= 26){
            currchar = (char)(twodigit + 'a' - 1) ;
            solve(str.substring(2),output+currchar);
        }
    }
}

public static void printEncodings(String str) {
    solve(str,""); //solve(input , output);
}
```

* STRING PERMUTATION

str: abc

o/p:

abc
acb
bac
bca
cab
cba

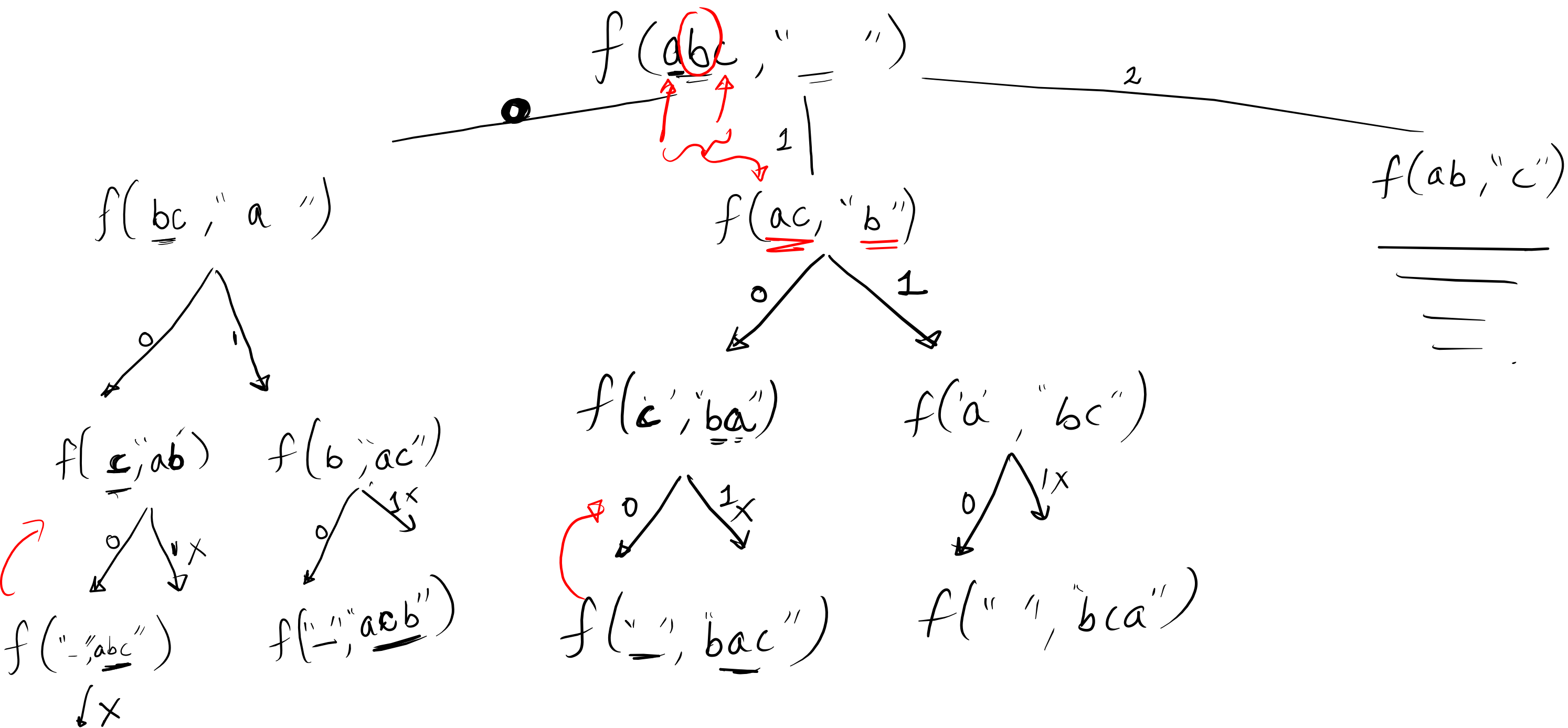
a b c d

a

b

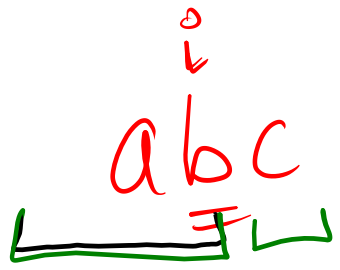
c


d

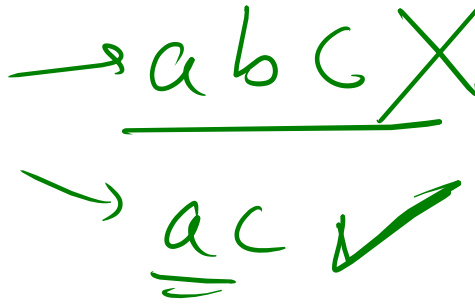


* Base Case

```
if (str.length() == 0) {  
    print (o/p)  
    return;  
}
```

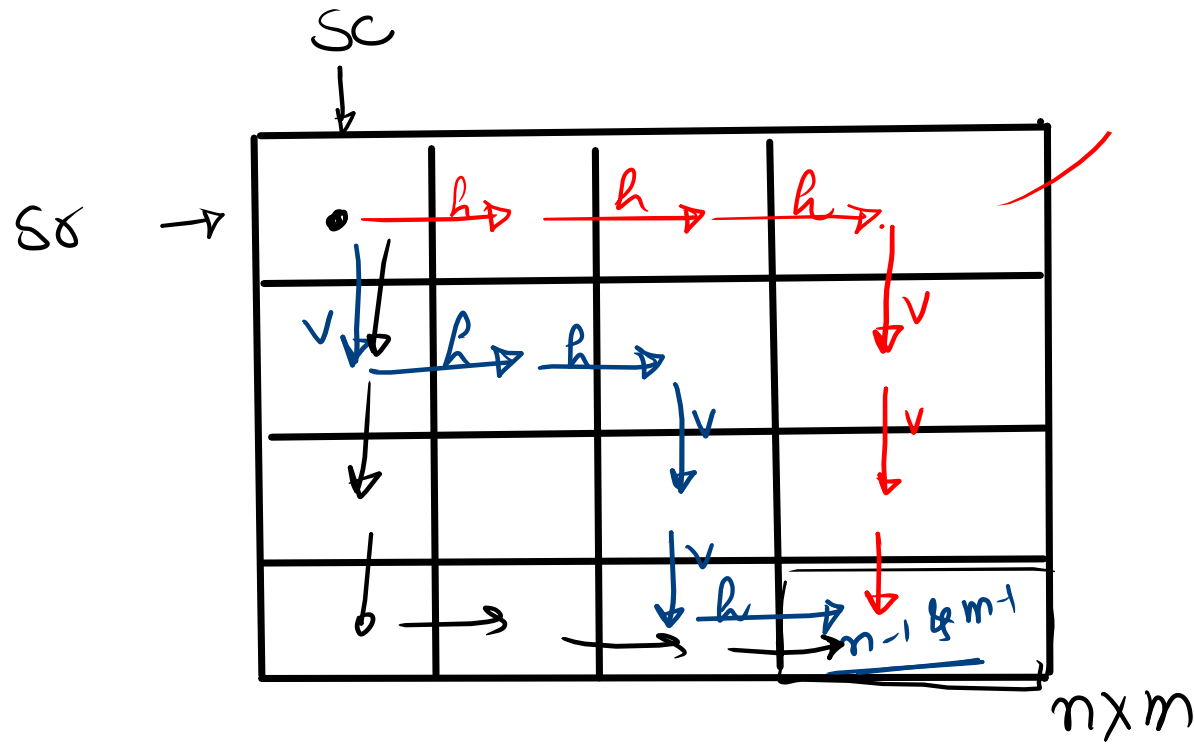
* 

generate?

(ac, "b")

output = output + str.charAt(i);
newString = str.substring(0, i) + str.substring(i+1);

include → exclude.


```
public void solve(String str , String output) {  
    //base case  
    if(str.length()==0) {  
        System.out.println(output);  
        return ;  
    }  
  
    //process  
    for(int i =0 ; i<str.length();i++){  
        if(i>0 && str.charAt(i)== str.charAt(i-1)) continue ;  
        String newString = str.substring(0,i) + str.substring(i+1);  
        // output = output + str.charAt(i);  
        solve(newString,output+ str.charAt(i));  
    }  
}  
  
public void printPermutations(String str) {  
    solve(str,"") ; // solve(input , output) ;  
}
```

* Maze Problem



RRR VVV (X)

h3 V3 ✓

h3 —
h3 V3 ✓

→ This is my one of the answers

V1h2V2h1 ✓

$\begin{matrix} n-1 \\ == \\ So \end{matrix}$

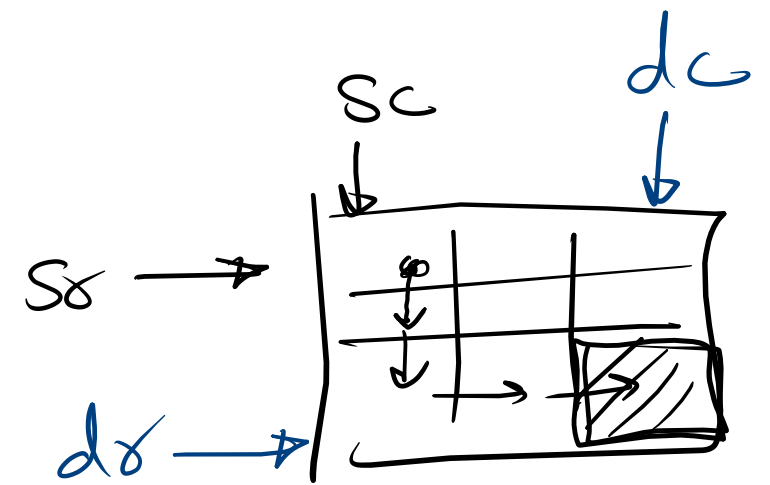
$\begin{matrix} m-1 \\ == \\ Sc \end{matrix}$

V3h3 ✓

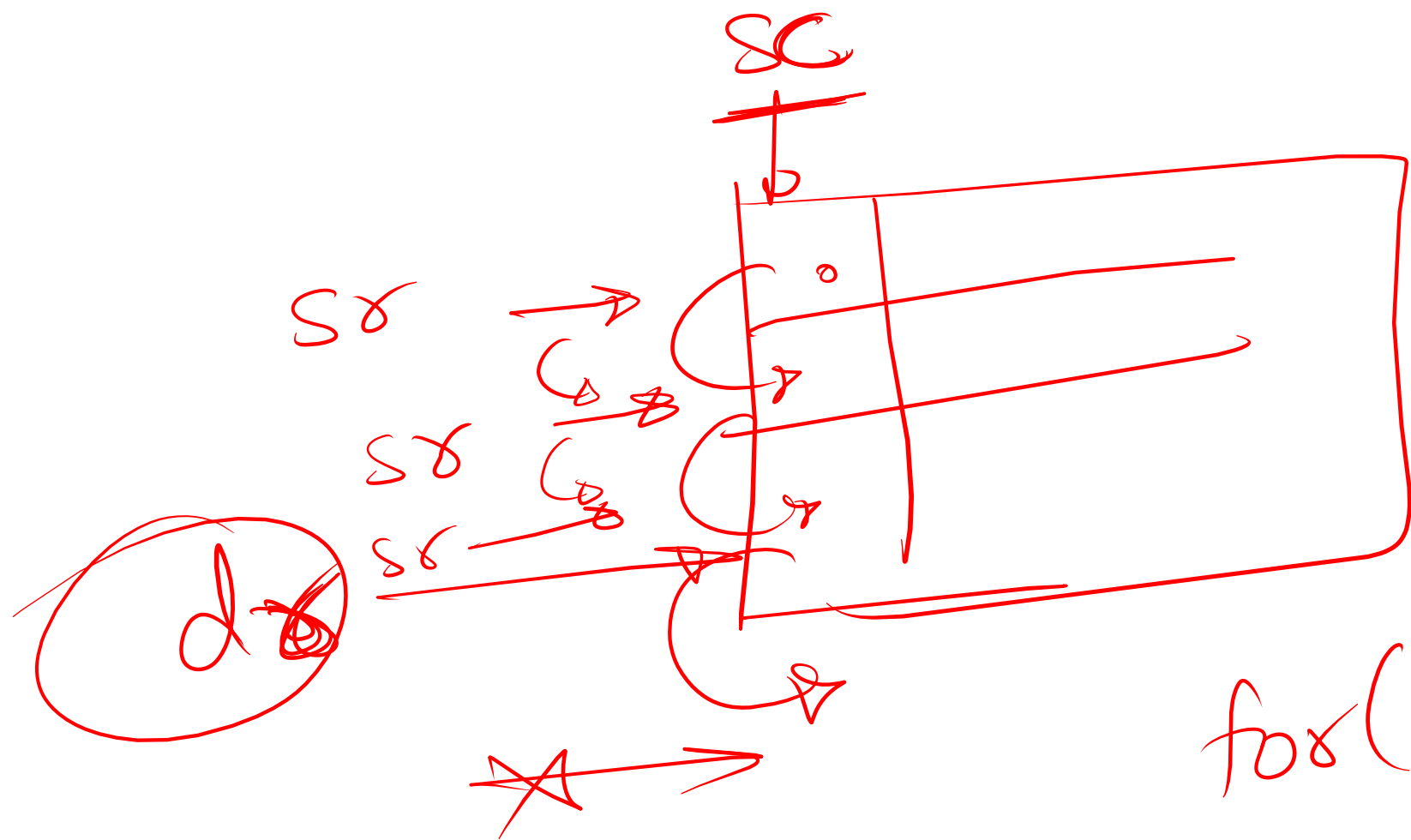
...

* Base base

```
if (sr == dr && sc == dc) {  
    print(output);  
    return;  
}  
}
```



$$\begin{matrix} dr & = & n-1 \\ dc & & m-1 \end{matrix}$$



$$sc + h \leq dx$$

for (int $v = 1$; $sx + v \leq dx$; $v++$) {

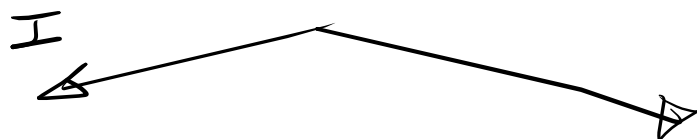
 solve($sx + v$, sc , dx , dc , $o + v + v$);

}

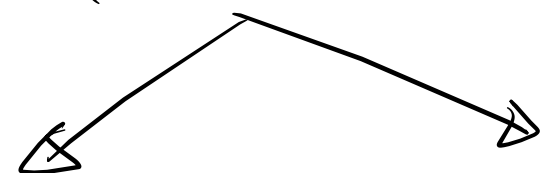
```
public static void solve(int sr, int sc, int dr, int dc, String output) {  
    //Write your code here  
    //base case  
    if(sr == dr && sc == dc){  
        System.out.println(output);  
        return;  
    }  
  
    //horizontal jumps  
    for(int h = 1 ; sc + h <= dc ; h++){  
        solve(sr,sc+h,dr,dc,output+'h'+h);  
    }  
  
    //vertical jumps  
    for(int v =1 ; sr + v <=dr ; v++ ){  
        solve(sr + v , sc , dr , dc , output+'v' + v);  
    }  
  
    //diagonal jumps  
    for(int d =1 ; sr + d <= dr && sc + d <= dc ; d++ ){  
        solve(sr + d , sc + d , dr , dc , output+'d' + d);  
    }  
}
```

* GET subsequence

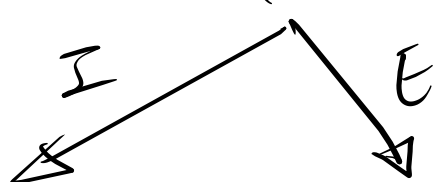
$f(a^0bc, " ")$



$f(a^1bc, " ")$

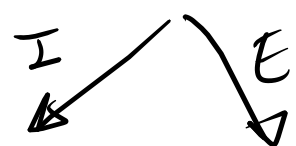


$f(a^1bc, "a")$



$f(a^1bc, "ab")$

$f(a^1bc, "a")$



$f(a^1bc, "abc")$ $f(a^1bc, "ab")$

ArrayList.add


```
public static void solve(String str , int idx , String output,ArrayList<String>ans){
    if(idx == str.length()){
        if(!output.isEmpty()){
            ans.add(output);
        }
        return ;
    }

    //include call
    solve(str,idx+1,output+str.charAt(idx),ans);

    //exclude call
    solve(str,idx+1,output,ans);
}

public static ArrayList<String> generateSubsequences(String str)
{
    ArrayList<String>ans = new ArrayList<>();
    solve(str,0,"",ans);
    Collections.sort(ans);
    return ans;
}
```


Old Phone Keypad

```
static String[] val={
    " ", "ABC", "DEF", "GHI", "JKL", "MNO", "PQRS", "TU", "VWX", "YZ"
};

static void solve(int n , int[]keys , int idx , String output , ArrayList<String>ans){

    //base case
    if(idx == n){
        ans.add(output);
        return;
    }

    //PROCESS
    int number = keys[idx];
    String temp = val[number];

    for(int i = 0 ; i<temp.length();i++){
        char ch = temp.charAt(i);
        solve(n,keys,idx+1,output+ch,ans);
    }

}

static ArrayList <String> OldPhone(int n, int[] keys){
    ArrayList <String>ans = new ArrayList<>();
    solve(n,keys,0,"",ans);
    return ans ;
}
```

Get Stair Paths

```
public static ArrayList<String> getStairPaths(int n) {  
    ArrayList<String> arr = new ArrayList<>();  
    helper(n, "", arr);  
    return arr;  
}  
  
static void helper(int n, String result, ArrayList<String> arr) {  
    if(n == 0) {  
        arr.add(result);  
        return;  
    }  
  
    if(n < 0) return;  
  
    for(int i=1; i<=3; i++) {  
        helper(n-i, result+i, arr);  
    }  
}
```

Get Maze Paths

```
public static ArrayList<String> getMazePaths(int sr, int sc, int dr, int dc) {  
    ArrayList<String> arr = new ArrayList<>();  
    solve(sr, sc, dr, dc, arr, "");  
    return arr;  
}  
  
static void solve(int sr, int sc, int dr, int dc, ArrayList<String> arr, String output) {  
  
    if(sr == dr && sc == dc) {  
        arr.add(output);  
        return;  
    }  
  
    //horizontal  
    if(sc < dc) {  
        solve(sr, sc+1, dr, dc, arr, output+"h");  
    }  
  
    //vertical  
    if(sr < dr) {  
        solve(sr+1, sc, dr, dc, arr, output+"v");  
    }  
}
```

Get Maze Paths

```
public static ArrayList<String> allPossiblePaths(int n, int m) {
    ArrayList<String> arr = new ArrayList<>();
    solve(0,0,n-1,m-1,arr,"");
    return arr;
}

static void solve(int sr,int sc,int dr,int dc,ArrayList<String> arr,String output){
    if(sr == dr && sc == dc){
        arr.add(output);
        return;
    }

    //horizontal moves
    for(int i=1;i<=2;i++){
        if(sc+i <= dc){
            solve(sr,sc+i,dr,dc,arr,output+"h"+i);
        }
    }

    //vertical moves
    for(int i=1;i<=2;i++){
        if(sr+i <= dr){
            solve(sr+i,sc,dr,dc,arr,output+"v"+i);
        }
    }

    //diagonal moves
    for(int i=1;i<=2;i++){
        if(sr+i <= dr && sc+i <= dc){
            solve(sr+i,sc+i,dr,dc,arr,output+"d"+i);
        }
    }
}
```