```
4

16 9

15 7

13 20

17 1
```

```
5

7 6

8 7

20 17

6 18

15 5
```

```
[4,5],
[4,6],
[6,7],
[2,3],
[1,1]]
```

1,1 2,3 4,5 4,6 6,7 → 1,1 2,3 4,6 4,5 6,7

dp = { 1, 3, 5, 6, 7 }

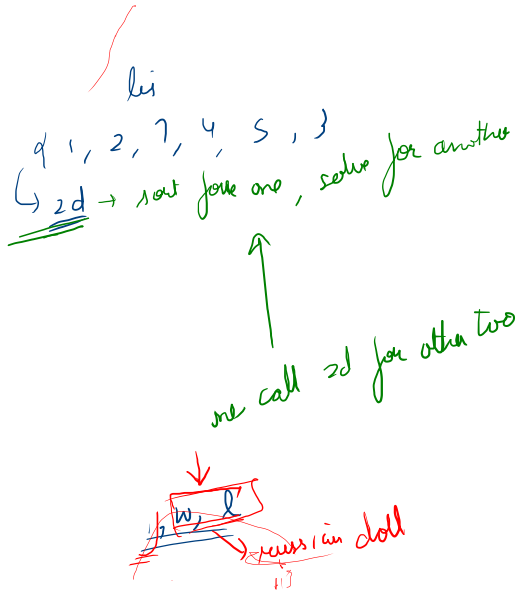
5

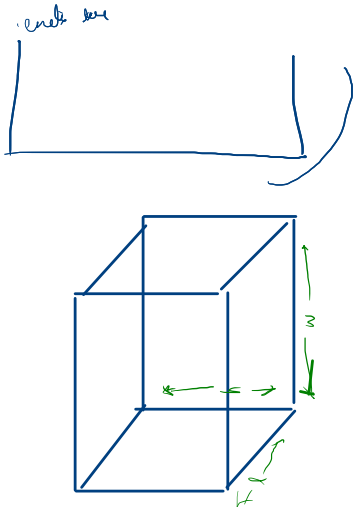
(1,1) < (2,3) < (4,5) < (6,7)
(4,6)

(20, 15)

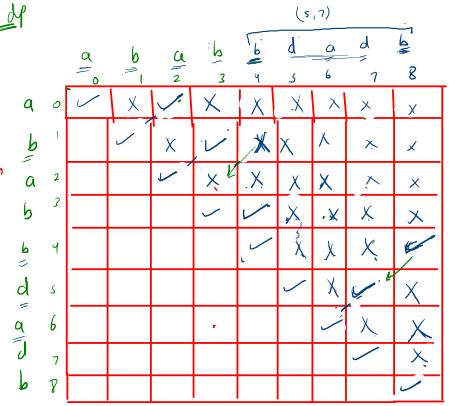
```java
public int maxEnvelopes(int[][] envelopes) {
    Arrays.sort(envelopes, (t, o)->{
        return t[0] - o[0];
    });

    int n = envelopes.length;
    ArrayList<Integer> dp = new ArrayList<>();

    for(int i=0; i<n; i++){
        int h = envelopes[i][1];

        int pos = findPos(dp, h);
        if(pos==dp.size()){
            dp.add(h);
        } else {
            dp.set(pos,h);
        }
    }

    return dp.size();
}
```

Run | Debug





String dp



substring len == 1

substring with len == 3

(4,8) → (5,7)

(i+1, j-1)

{ (i, j)

(i+1, j-1)

longest palindromic substring

boolean

81

len == 1

```
public void makePallindromeBoolenDP(String str){
    int n = str.length();
    boolean[][] dp = new boolean[n][n];

    for(int diag = 0; diag<n; diag++){
        for(int i=0,j=diag; j<n; i++,j++){
            if(diag==0){ // substrings of size always a pallindromw
                dp[i][j] = true;
            } else if(diag == 1){ // substrings of size2
                if(str.charAt(i)==str.charAt(j)){
                    dp[i][j] = true;
                }
            } else {
                if(str.charAt(i)==str.charAt(j)){
                    dp[i][j] = dp[i+1][j-1];
                }
            }
        }
    }
}
```

dp(i)(j) ⇒ j - i + 1 → str. substring (i, j+1)

pall in

(c1 == c3)

a b b a

pall

(c1 == c4)

dp(i)(j) → substring from i to j is a pall

pall

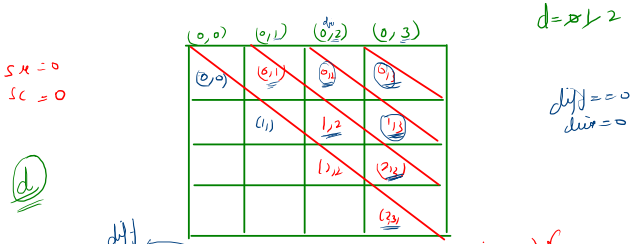
c1 c2 c3 c4 c5

(c1 == c5)

fc == lc → pall

dp(i+1)(j-1)

1)



sr = 0
sc = 0

d

d = 0 1 2

diff == 0
diag == 0

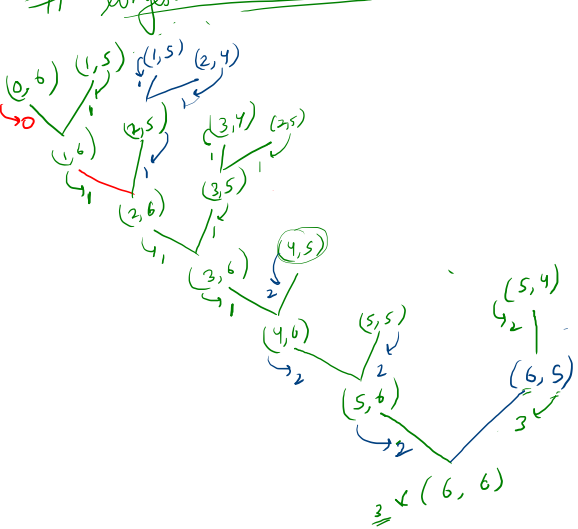
diff

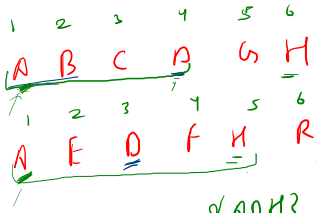
for (int diag = 0; diag < m; diag++) {
for (int i = 0, j = d; j < m; i++, j++) {

b

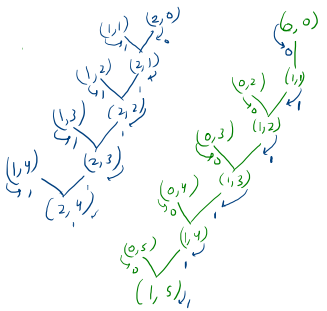
}

longest common subsequence (LCS)





(ADH)

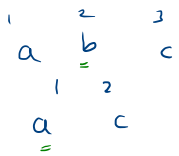


n == 0 || m == 0

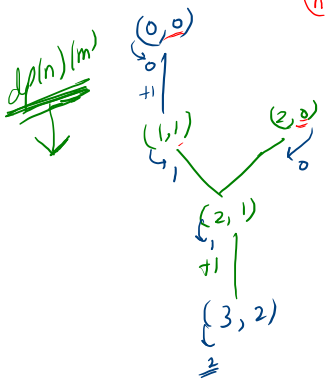
for(n == 0
m == 0

(n-1, m-1)

(n-1, m) (n, m-1)



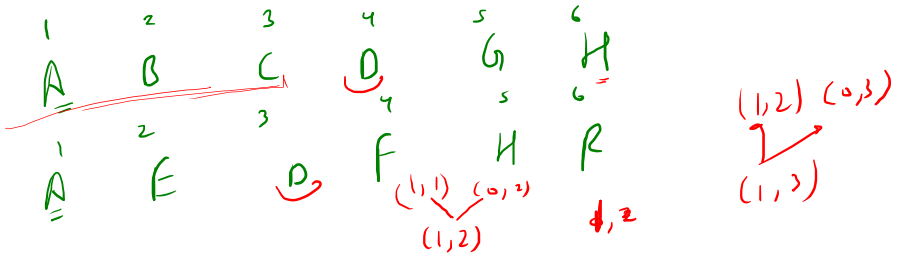
dp(n)(m)



```
public static int rec_memo(int n, int m, String s1, String s2,int[][] dp){
    if(n==0 || m==0){
        return 0;
    }

    if(dp[n][m]!=-1) return dp[n][m];

    if(s1.charAt(n-1) == s2.charAt(m-1)){
        return dp[n][m] = rec_memo(n-1, m-1, s1, s2,dp) + 1;
    } else {
        return dp[n][m] = Math.max(rec_memo(n,m-1,s1,s2,dp),rec_memo(n-1, m, s1, s2,dp));
    }
}
```

rec(n, m)

dp(n, m) ⇒ length of lcs with first n characters from s1
and m " " s2



S1 → / S2 ↓	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1
2	0	1	1	1	1	1	1
3	0	1	1	1	1	1	1
4	0	1	1	2	2	2	2
5	0	1	1	2	2	2	2
6	0	1	1	2	2	3	3

```java
public static int lcs_tab(int N, int M, String s1, String s2, int[][] dp){
    for(int n=0; n<=N; n++){
        for(int m=0; m<=M; m++){
            if(n==0 || m==0){
                dp[n][m]=0;
                continue;
            }

            if(s1.charAt(n-1) == s2.charAt(m-1)){
                dp[n][m] = dp[n-1][m-1] + 1; //rec_memo(n-1, m-1, s1, s2,dp) + 1;
            } else {
                dp[n][m] = Math.max(dp[n][m-1],dp[n-1][m]); //Math.max(rec_memo(n,m-1,s1,s2,dp
            }
        }
    }

    return dp[N][M];
}
```

break till 11:06

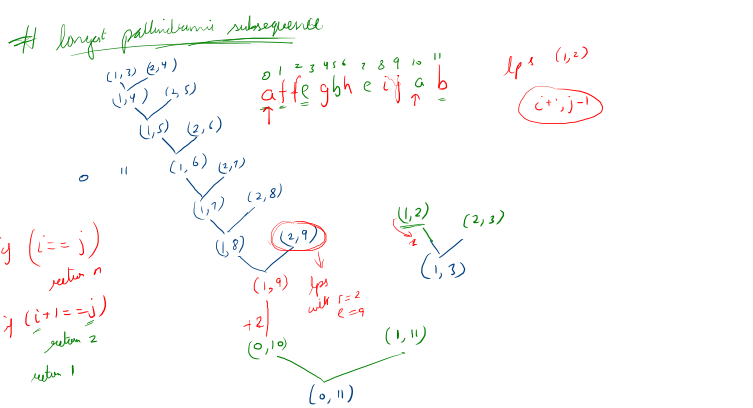
n - les ⇐ s e a ⟶ "a"

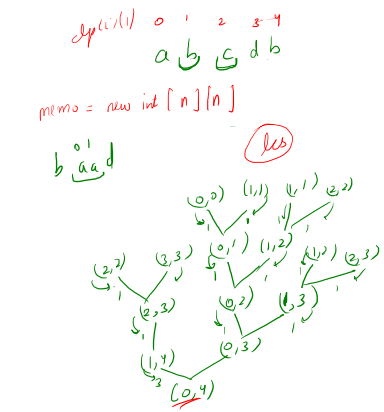
m - les ⇐ a t l

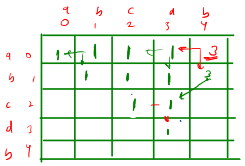
n - les + m - les

n+m - 2*les

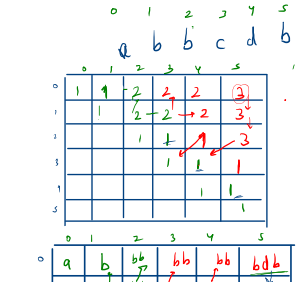
longest palindromic subsequence

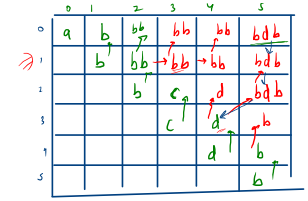






```
public static int lps_rec(String str, int i, int j){
    if(i==j){
        return 1;
    }

    if(str.charAt(i)==str.charAt(j)){
        if(i+1 == j){
            return 2;
        } else {
            return lps_rec(str, i+1, j-1) + 2;
        }
    } else {
        return Math.max(lps_rec(str,i,j-1),lps_rec(str, i+1, j));
    }
}
```





```
for(int diag=0; diag<n; diag++){
    for(i=0,j=diag; j<n; i++,j++){
        if(i==j){
            dp[i][j] = 1;
            sdp[i][j] = str.charAt(i)+"";
            continue;
        }

        if(str.charAt(i)==str.charAt(j)){
            if(i+1 == j){
                dp[i][j] = 2;
                sdp[i][j] = str.substring(i, i+2);
            } else {
                dp[i][j] = dp[i+1][j-1] + 2;//lps_memo(str, i+1, j-1,dp) + 2;
                sdp[i][j] = str.charAt(i) + sdp[i+1][j-1] + str.charAt(j);
            }
        } else {
            // dp[i][j] = Math.max(dp[i][j-1],dp[i+1][j]); //Math.max(lps_memo(str,i,j-1,dp),lps_
            if(dp[i][j-1] > dp[i+1][j]){
                dp[i][j] = dp[i][j-1];
                sdp[i][j] = sdp[i][j-1];
            } else {
                dp[i][j] = dp[i+1][j];
                sdp[i][j] = sdp[i+1][j];
            }
        }
    }
}
```

min. insertions to make a string palindrome.

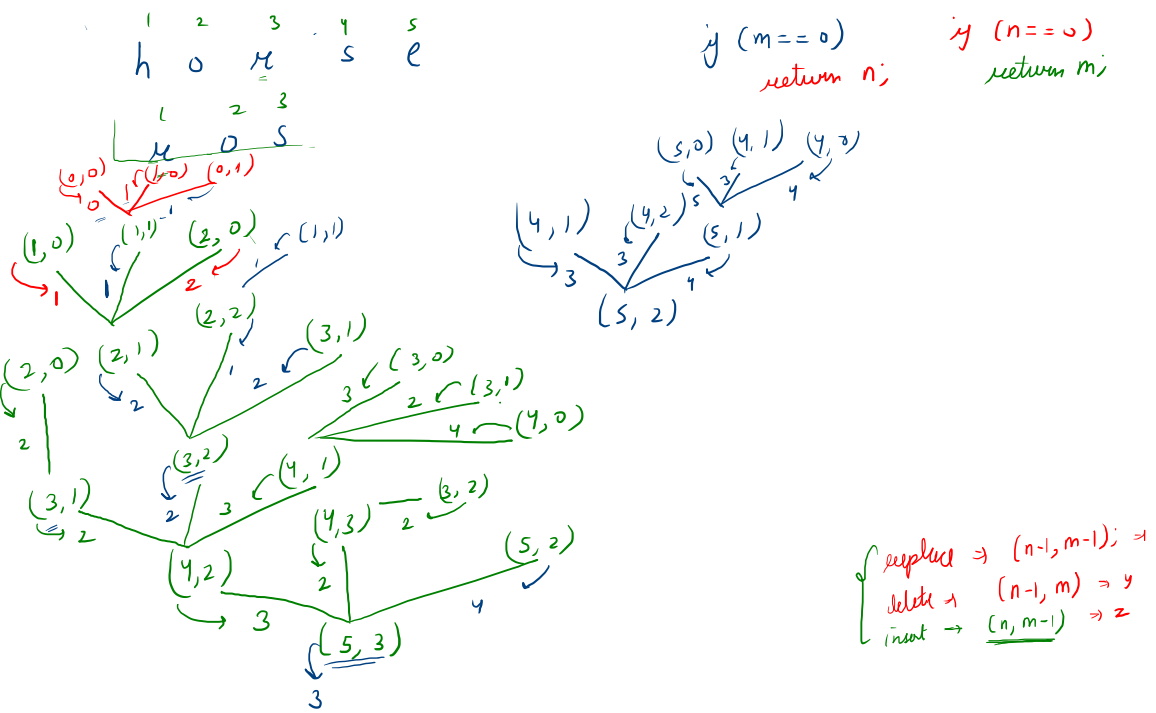
Ques

n - lps

mam ⇐ m b a d m ⇒ 2

bcb ⇐ a b c d b ⇒ 2

bcb ⇐ a b b c d b ⇒ 3

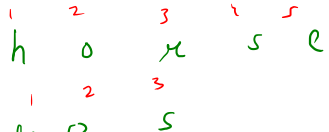


if (m == 0) return n;

if (n == 0) return m;

replace → (n-1, m-1); → x
delete → (n-1, m) → y
insert → (n, m-1) → z

min (x, y, z) + 1;

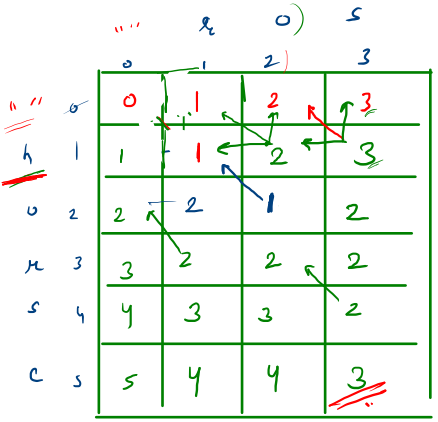


h o
r o

(1,1)

break till 10:50

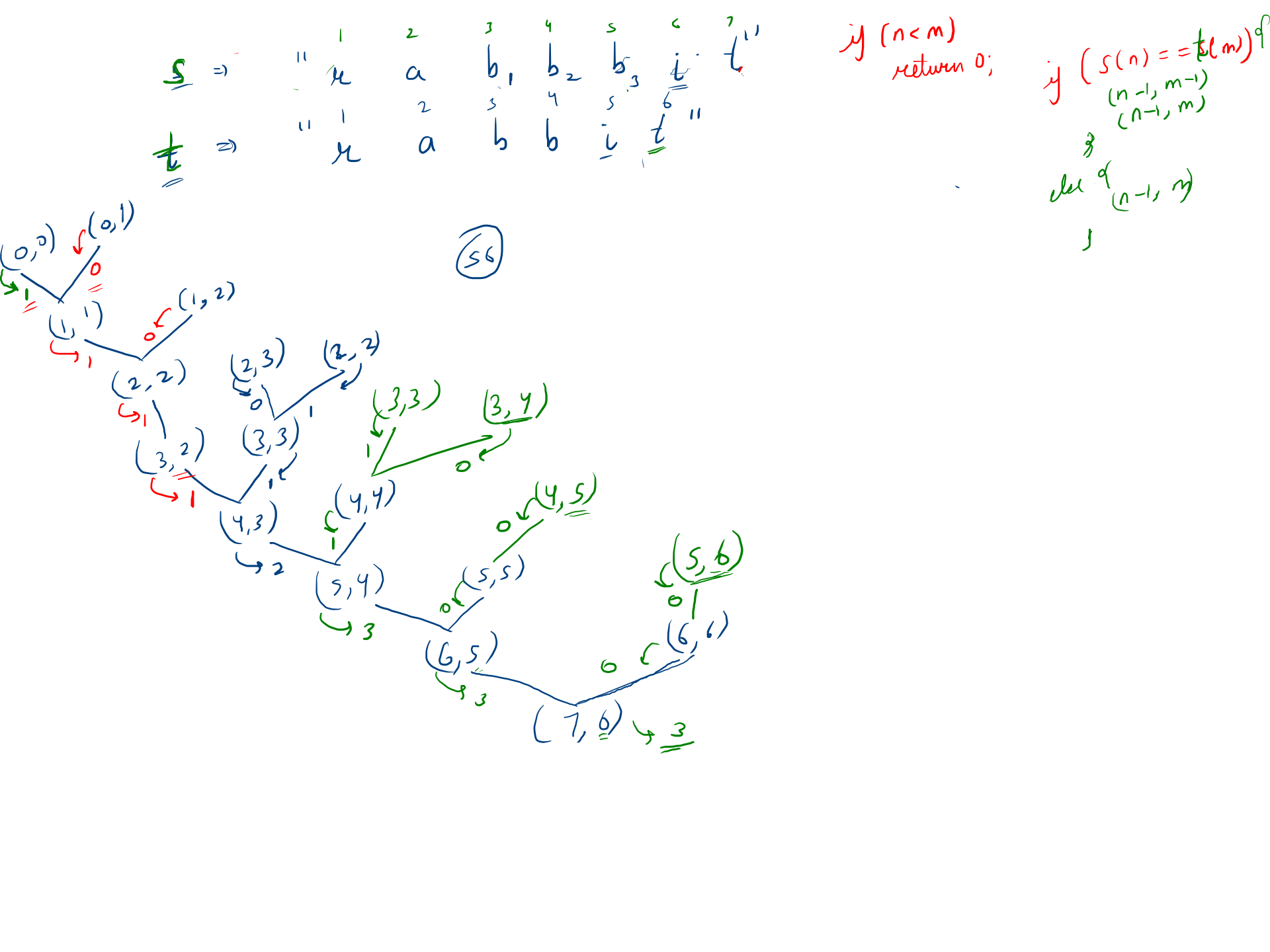
(2,2)



dp(i)(j)

```java
public int minDistance_tab(int N, int M, String w1, String w2,int[][] dp){
    for(int n=0; n<=N; n++){
        for(int m=0; m<=M; m++){
            if(n==0 || m==0){
                dp[n][m] = n==0 ? m : n;
                continue;
            }

            if(w1.charAt(n-1) == w2.charAt(m-1)){
                dp[n][m] = dp[n-1][m-1]; //minDistance_memo(n-1,m-1,w1, w2,dp);
            } else {
                int replace = dp[n-1][m-1]; //minDistance_memo(n-1, m-1, w1, w2,dp);
                int delete = dp[n-1][m]; //minDistance_memo(n-1, m, w1, w2,dp);
                int insert = dp[n][m-1]; //minDistance_memo(n, m-1, w1, w2,dp);

                dp[n][m] = Math.min(replace,Math.min(delete,insert)) + 1;
            }
        }
    }

    return dp[N][M];
}
```

dry run


S ⇒ " a a b1 b2 b3 i t "
t ⇒ " a a b b i t "
if (n < m)
return 0;
if (s(n) == t(m))
(n-1, m-1)
(n-1, m)
}
else
(n-1, m)
}
56
(0,0)
(0,1)
(1,1)
(1,2)
(2,2)
(2,3)
(2,2)
(3,2)
(3,3)
(3,3)
(3,4)
(4,3)
(4,4)
(4,5)
(5,4)
(5,5)
(5,6)
(6,5)
(6,6)
(7,6)