* Longest Common Subsequence (LCS) problem

Ex 1: S1 : a b c d e f g h i j

S2 : c d g i

∴ One subsequence is c d g i

- other subsequences could be d g i / g i / i

- But the longest subseq = c d g i

Ex: S1 : a b c d e f g h i j

S2 : e c d g i

Note: matching lines should not cross over

∴ e g i → is one subseq &

cdgi → could also be a subseq

dgi → " " " " "

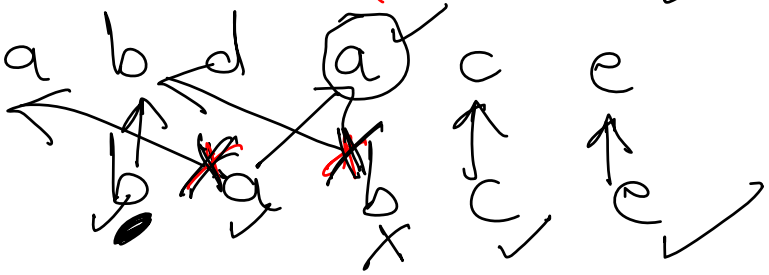
* cdgi is the longest one

---

1 2 3 4 5 6

Ex: 3 S1 : a b d a c e

S2 : b a b c e

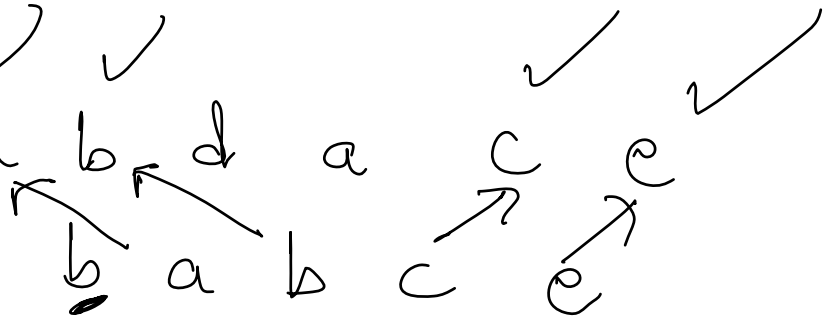


① b a c e → is one subsequence

- If we start with a letter of S2 instead of b letter than

S1 : a b d a c e

S2 : b a b c e



than ② a b c e is yet another subsequence

* Note :- There can be multiple subsequences of same length obtained

* Methods to solve LCS

1. LCS using Recursion (Memoization) ✓
   - used to reduce recursive calls
2. LCS using Dynamic Programming ✓

① LCS Using Recursion → Exponential time ($O(2^n)$) → top down approach

Algorithm

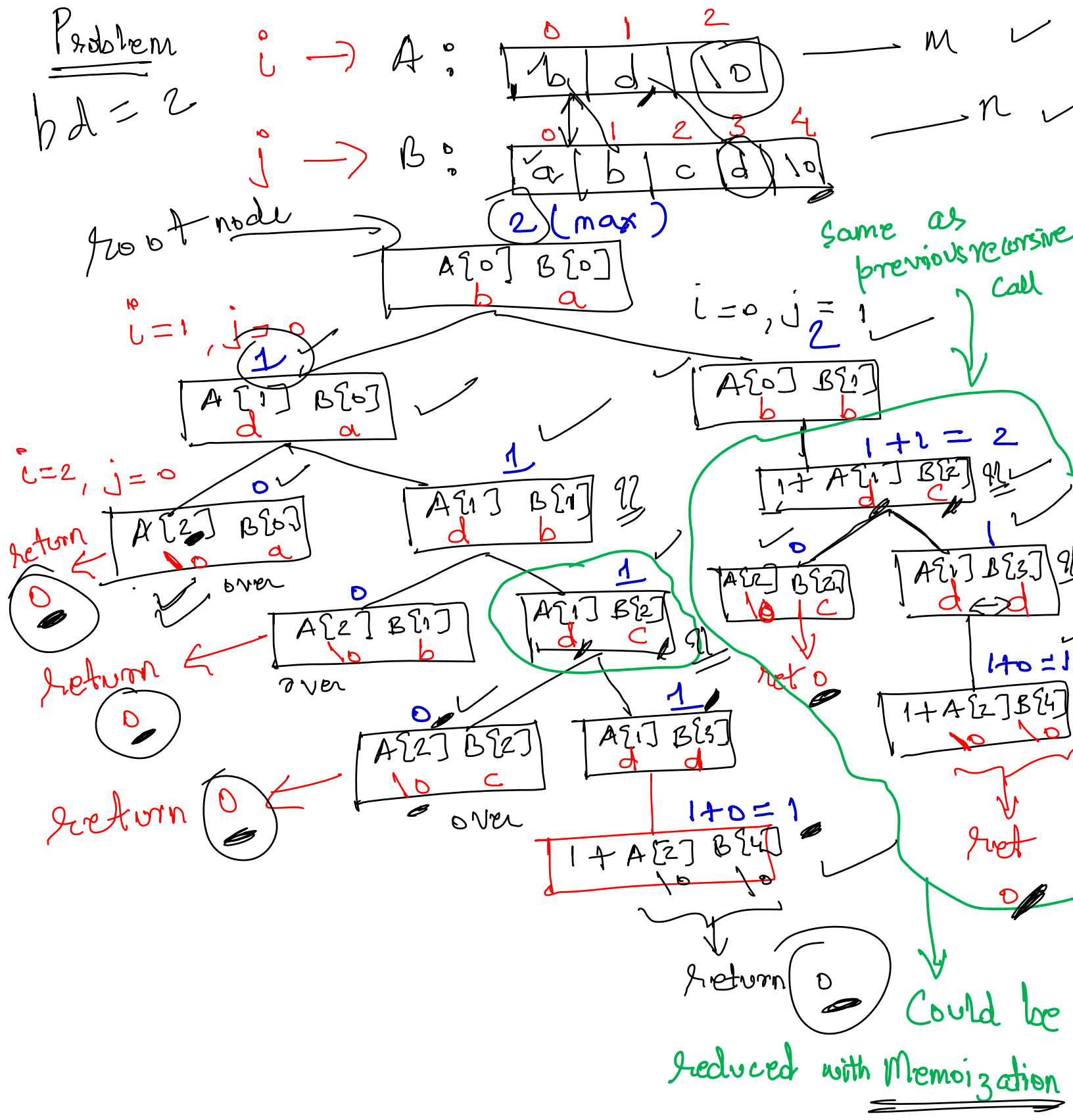
```
int LCS (i, j)
{
    if (A[i] == '\0' || B[i] == '\0')
        return 0;
    else if (A[i] == B[j])
        return 1 + LCS(i+1, j+1);
    else
        return Max(LCS(i+1, j), LCS(i, j+1));
}
```

perfect match → return 0;

else if (A[i] == B[j]) → return 1 + LCS(i+1, j+1);

if not a match → else return Max(LCS(i+1, j), LCS(i, j+1))

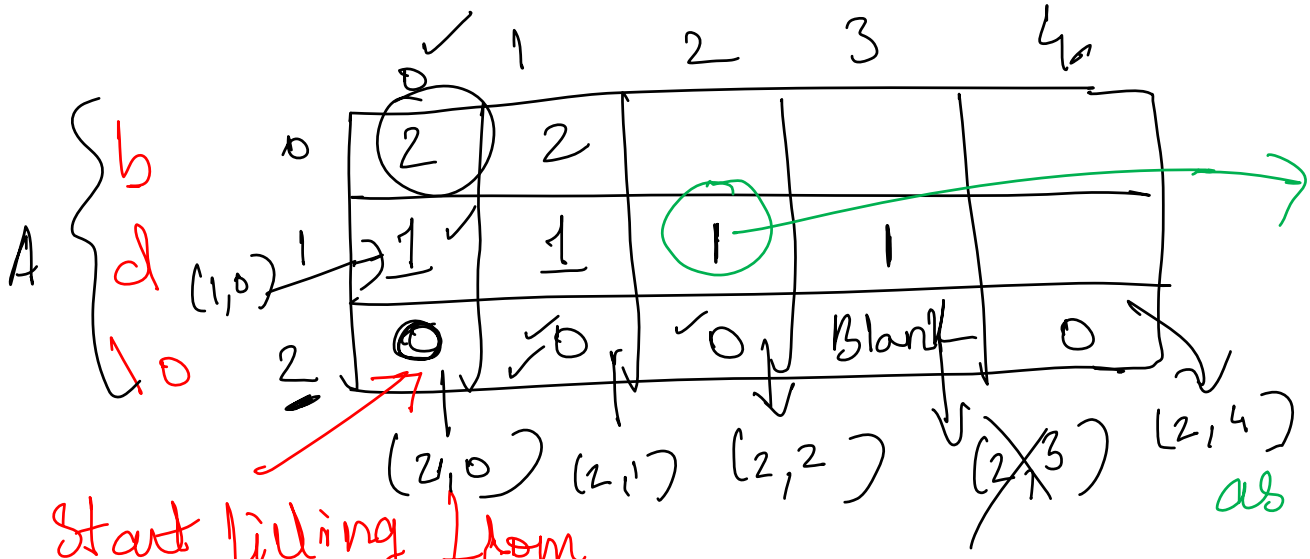
LC RC


Problem
bd = 2
i → A:
j → B:
m
n
root node
2 (max)
A[0] B[0]
b a
i=1, j=0
A[1] B[0]
d a
i=0, j=1
A[0] B[1]
b b
same as previous recursive call
i=2, j=0
A[2] B[0]
return 0
over
A[1] B[1]
d b
A[2] B[1]
return 0
over
A[1] B[2]
d c
1+1 = 2
1+A[1] B[2]
d c
A[2] B[2]
ret 0
A[1] B[3]
d d
1+0 = 1
A[2] B[3]
return 0
over
A[2] B[2]
c c
A[1] B[3]
d d
1+0 = 1
1+A[2] B[4]
return 0
1+0 = 1
1+A[2] B[4]
ret
Could be reduced with Memoization

* Memoization (to improve recursion & reduce time & fun^n calls)

B

a b c d \0



(1,2) call is made only once as its made on left side of tree

Start filling from here by referring tree calls

$O(mn)$ → time of memoization

* LCS Using Dynamic Programming

if (A[i] == B[j])

$$LCS[i,j] = 1 + LCS[i-1, j-1]$$

previous diagonal value

else

$$LCS[i,j] = \max\left(LCS[i-1,j], LCS[i,j-1]\right)$$

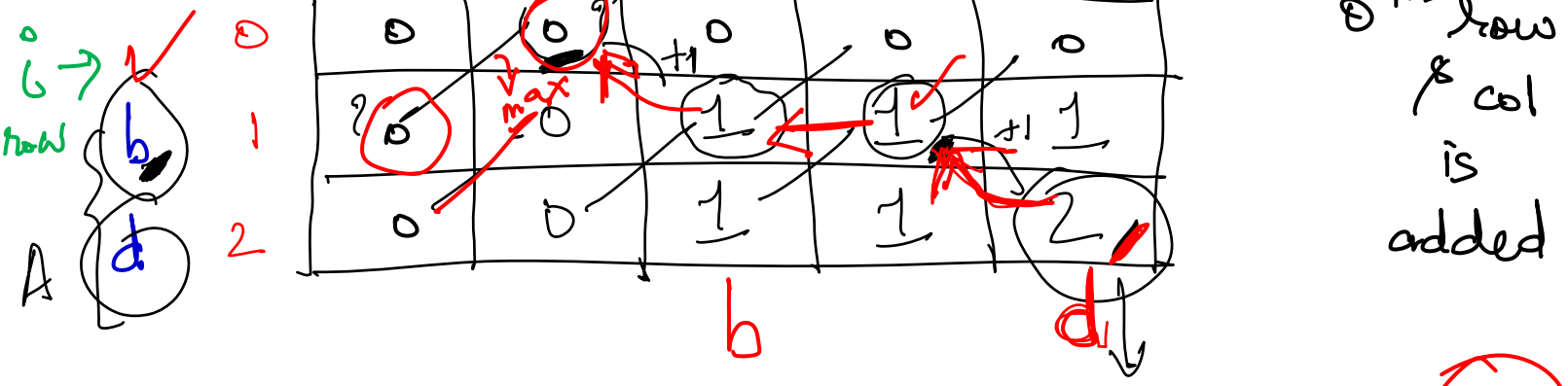
max of previous row or column entry

Eg ①

A : | b | d | → m

B : | a | b | c | d | → n

- index starts with 1



* Time : $\Theta(mn)$

Ex: 2 s1: s t o n e

s2: l o n g e s t