678. Valid Parenthesis String

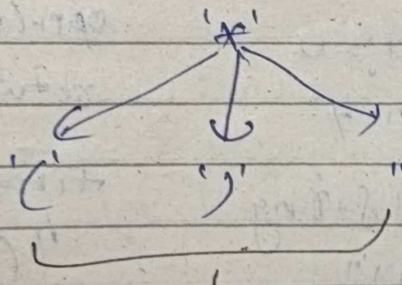
e.g. → "( )"
o/p → true

e.g. → "(*)"
o/p → True

e.g. → "(*))"
o/p → True;

Approach - 1 : DP

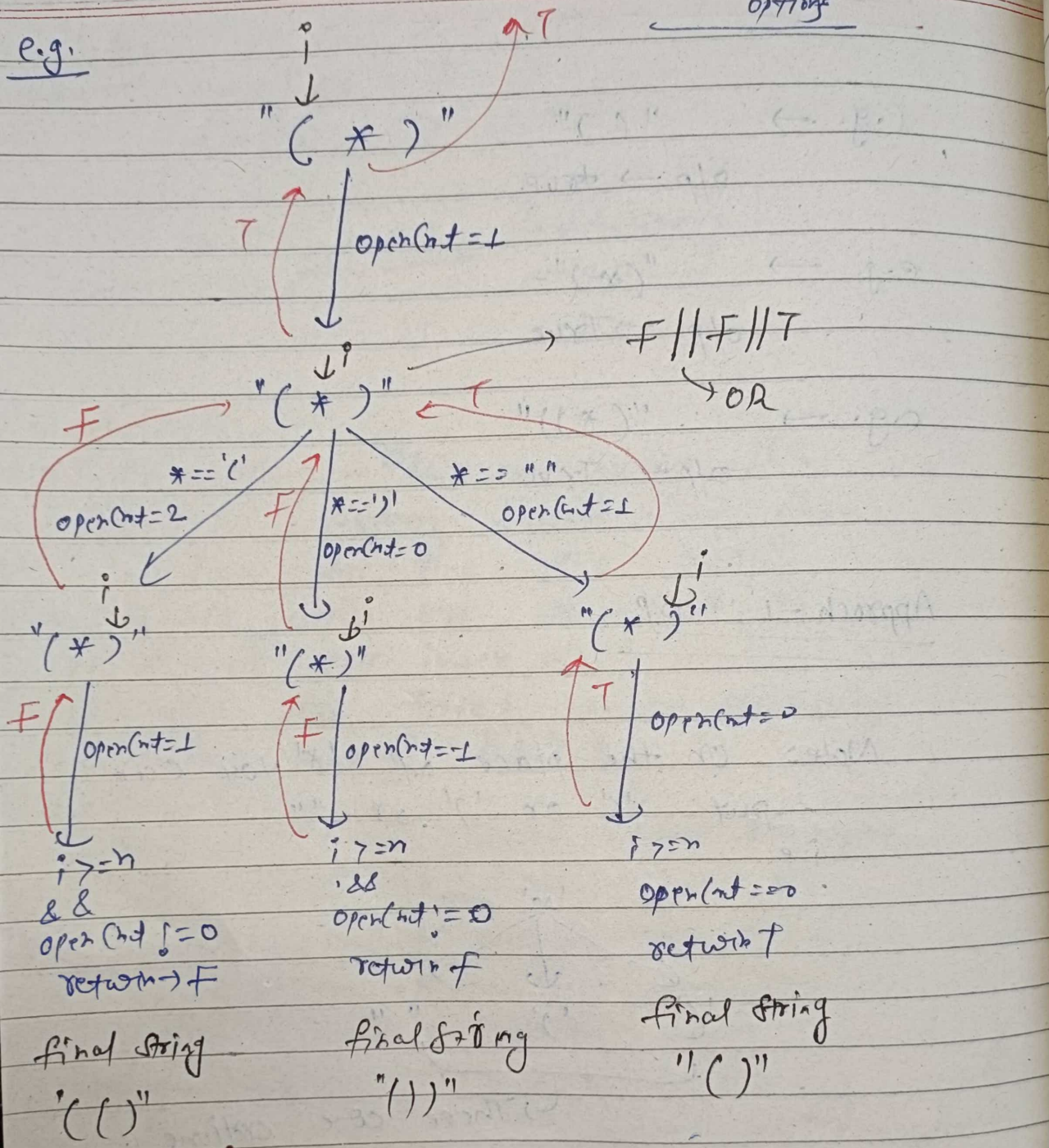
Note: on the place of '*' you can put '(' or ')' or "".
i.e.



↳ There are options available i.e. you can apply recursion & DP also.

$T.C. = 3^n$ ← options

e.g.



DP = ? → There is two changing variables.

(i) Idx

(ii) openCnt.

Teacher's Signature

solve (idx, open(nt) → This will gives you that till idx (index) how many open parenthesis ('(') are present in the string.

for DP we know that there must be overlaping sub problems.

↳ two or more sub problems should be same.

Here how it is possible.

e.g. "(*)*((*)"

0 1 2 3 4

"**(" → "*(" → "(" → open(nt=1

"**(" → "*" → "" → open(nt=0

suppose you are at idx = 4

"(*)*((*)"

idx

one way, * == "(" → * == ")"

String → "()((*)"

idx=4 open(nt=0

another way → * == '(' && * == ')'

String → "(())((*)"

idx=4 open(nt=0

overlaping subproblem

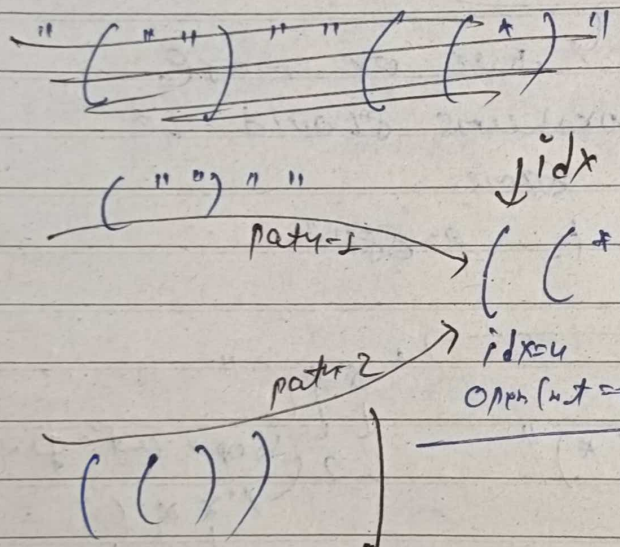
Teacher's Signature

Note:-

"(*)*((*)"

↓

At this idx you can reach via two/more different path like.



in recursion you are going for both the paths suppose from path-1 your answer is → false then from path-2 your answer will also false. but in recursion you go for both the paths.

If we store the answer of first calculated path-1 then we can be use the that answer and not check for path-2.

Teacher's Signature .....

for storing the answer memoization will come in picture.

for memoization you take a 2D array.

why 2D array bcz two changing variable.

idx & operCnt

~~the maxim~~

idx → 0 to n

operCnt → 0 to 2

↳ when string contain only open (.

dp[n+1][n+1];

$T.C. = O(3^n)$ → without memoization

$T.C. = O(n \times n)$

$S.C. = O(n \times n)$ } Memoization

Bottom-UP Approach

we have a 2D DP array of size

DP[n+1][n+1]

(in memoization we go & fill the DP[idx][open(it) & final our result store in

DP[0][0] bcz we call

solve(0, 0, s)
↑ idx ↑ open(it

So in Bottom Up you also have a 2D DP[n+1][n+1) i.e. you have to apply two for loops like that

```
for( idx =0 to n)
{ for (open(it =0 to n)
  {
```

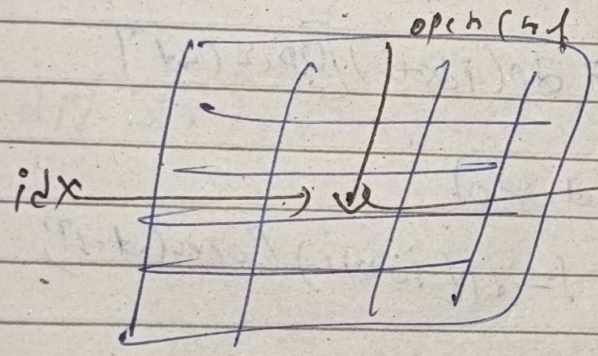
Teacher's Signature .....

But when you saw in the memoization code you get that the answer of solve(idx, open(nt) you get from the:

solve(idx+1, open(nt+1)) or

solve(idx+1, open(nt);

so like you have DP



you are at here & you come here from idx=0 to, open(nt=0 but if you want to fill this you should have the answer of

(idx+1, open(nt-1) or (idx+1, open(nt+1)

or

(idx+1, open(nt)

So you can't get the answer from idx=0 to n

open(nt=0 to n

Teacher's Signature

The for loop of open cnt change bcz you start traversing string from last, but open cnt is still =0

To like this

```
for(idx = n-1; idx >=0; idx--)
{
  for(open cnt=0; open cnt <=n; open cnt++)
  {
      if( s[idx] == '*')
      {
        isValid |= dp[idx+1][open cnt+1]
        isValid |= dp[idx+1][open cnt]
        if (open cnt > 0)
        { isValid |= dp[idx+1][open cnt-1];
        }
      } else if( s[idx] == ')')
        { isvalid |= dp[idx+1][open cnt+1];
        }
      else if( open cnt > 0)
        {
        }
      dp[idx][open cnt] = isValid
  }
}
return dp[0][0];
```

Teacher's Signature

dp[idx][openCnt] → can be True/False.

The meaning of this is like

0 1 2 3 4 5 6 7 8 9 10 11 12

"((*))(((****)))"

↑ idx, openCnt

suppose that you are at the idx index in string.

↳ There can be a value of openCnt.

dp[idx][openCnt] आपको यह बताता है कि यदि idx index तक openCnt इतना है तो क्या idx से till (n-1) index तक की string एक valid string बन सकती है या नहीं।

dp[0][0] → इसका मतलब है कि initially you have 0 open paranthesis then from 0-th index to till (n-1) index is this string can be valid or not.

Teacher's Signature ____________

Approach-3

↳ own approach

Track the index of open & star.

Approach-4

left to right traversing in string

right to left ↑↑

assumption

left to right '*' → '('

right to left '*' → ')'

e.g.:- "*(()) (*"

open cnt = 1 + 1 + 1 - 1 - 1 + 1 + 1 = 3

close cnt = 0 + 1 - 1 + 1 + 1 - 1 - 1 + 1 = 1

if any point of time

open cnt < 0 ⟶ return false

close cnt < 0 ⟶ "

open(++ < 0

↳ it means string में close parenthesis ज्यादा है

even though you are taking '*' as open.

close(++ > 0

↳ open parenthesis ज्यादा है even though you treating '*' as close