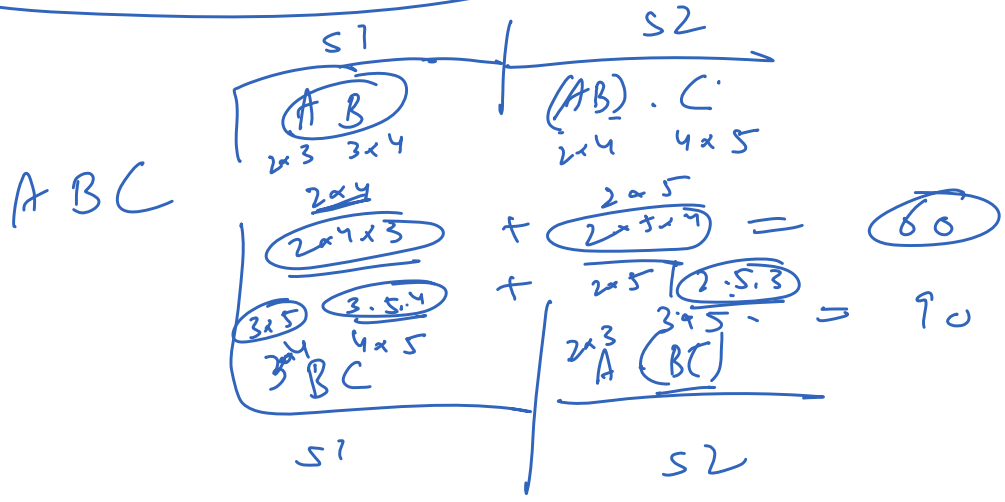
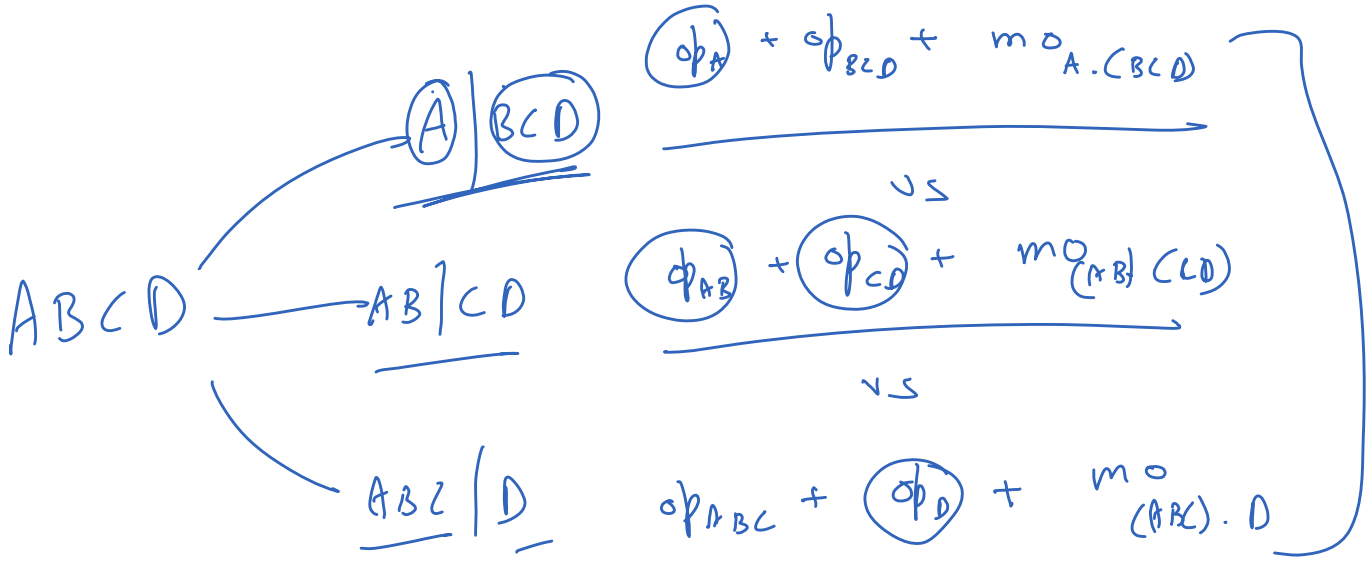
$A = 2 \times 3$

$B = 3 \times 4$

$AB = 2 \times 4 \mid 2 \times 4 \times 3$

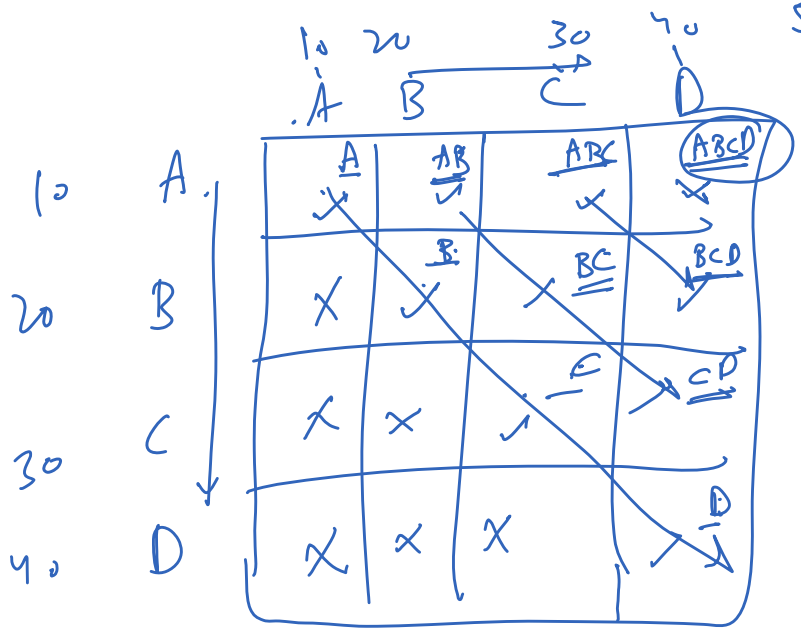




A B C D

10 x 20 x 30 x 40 x 50

S1 = AB	BC	CD
S2 = (AB).C	A.(BC)	AB
S3 = (ABC).D	(ABC).D	(AB)(CD)



AB

ABC

ABCD

11:17 — 11:22

921. Minimum Add to Make Parentheses Valid

Medium 2882 161 Add to List Share

A parentheses string is valid if and only if:

- It is the empty string,
- It can be written as `AB` (`A` concatenated with `B`), where `A` and `B` are valid strings, or
- It can be written as `(A)`, where `A` is a valid string.

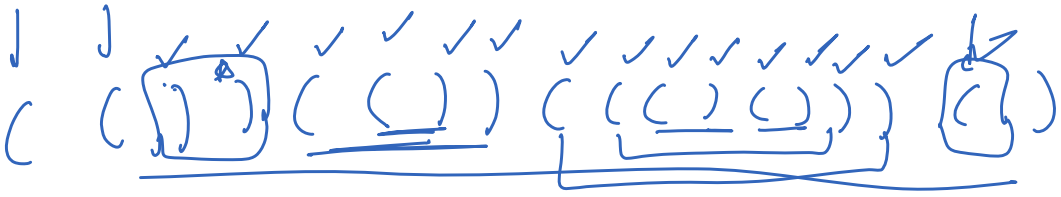
You are given a parentheses string `s`. In one move, you can insert a parenthesis at any position of the string.

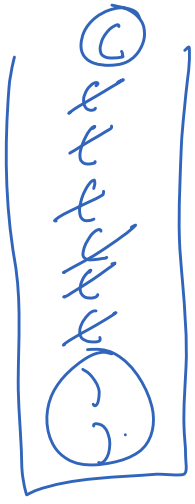
- For example, if `s = "()))"`, you can insert an opening parenthesis to be `"(()))"` or a closing parenthesis to be `"())))"`.

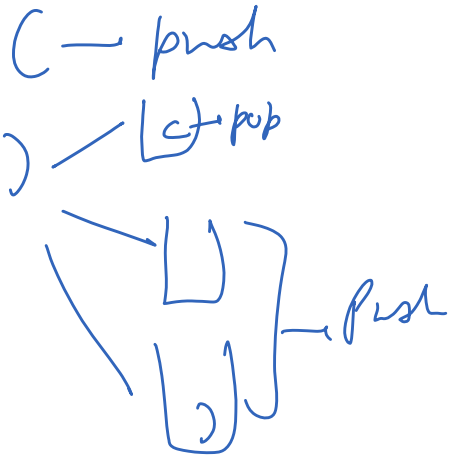
Return *the minimum number of moves required to make* `s` *valid.*

Example 1:

```
Input: s = "())"
Output: 1
```







9:15 - 9:20

Count the Reversals

Medium Accuracy: **50.95%** Submissions: **26920** Points: **4**

Given a string **S** consisting of only opening and closing curly brackets **'{'** and **'}'**, find out the minimum number of reversals required to convert the string into a balanced expression.

A reversal means changing **'{'** to **'}'** or vice-versa.

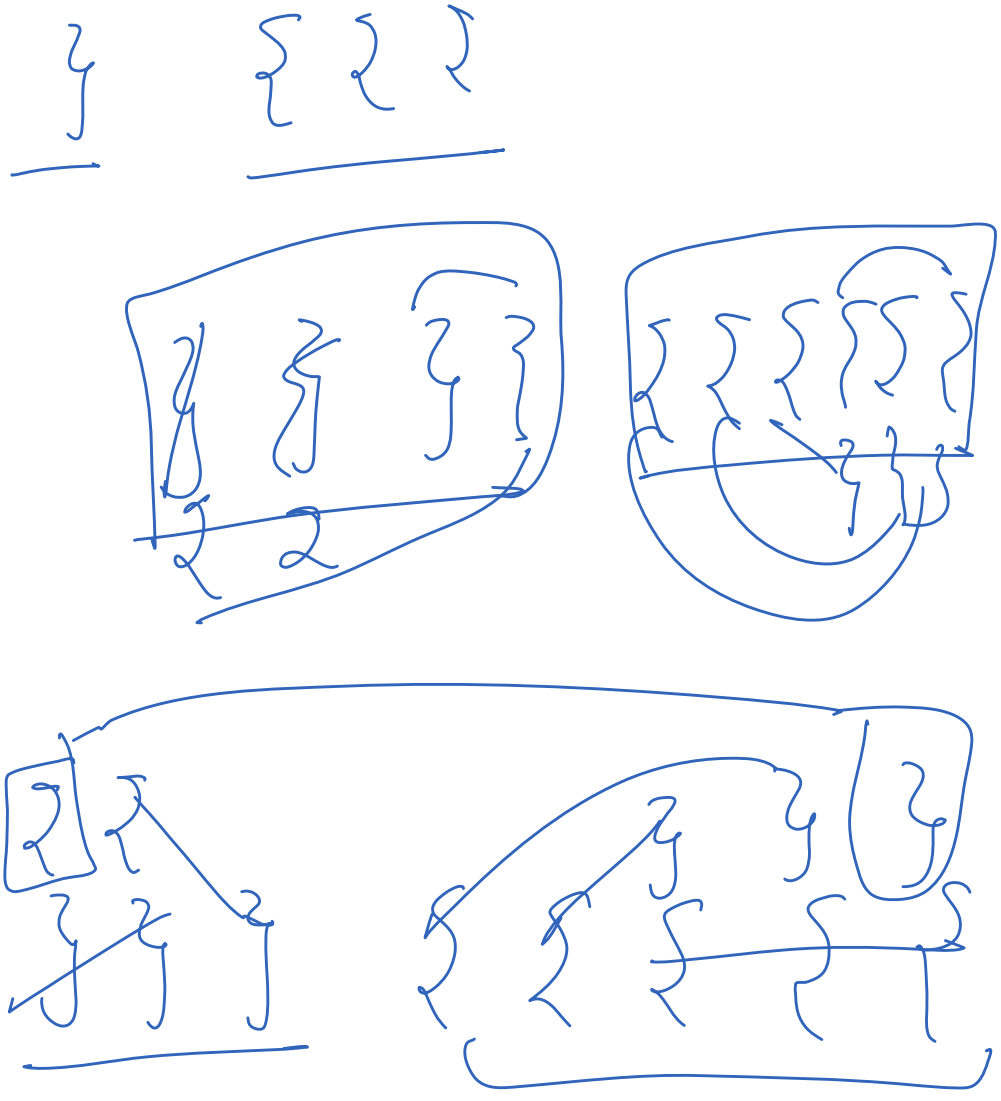
Example 1:

A reversal means changing '{' to '}' or vice-versa.

Example 1:

Input:
S = "}{{}}{{{"
Output: 3
Explanation: One way to balance is: "**{{{}}{}}**". There is no balanced sequence that can be formed in lesser reversals.

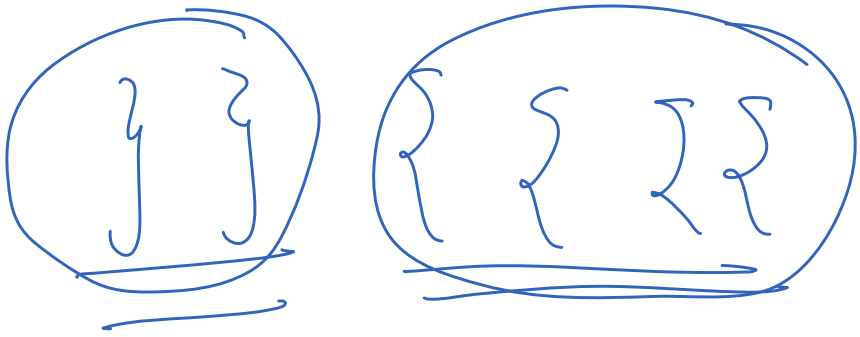
â€‹**Example 2:**

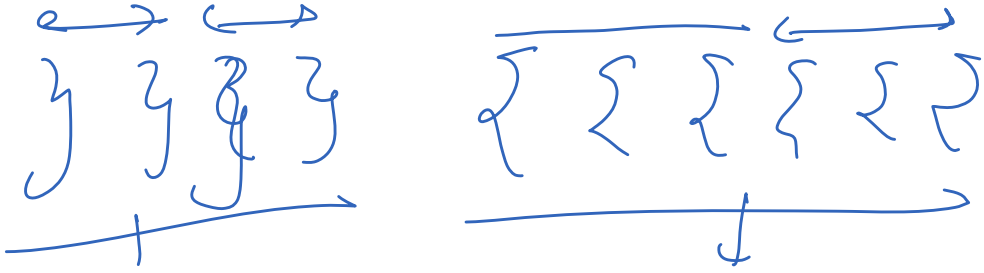


x/y 2 3

x/y → floor

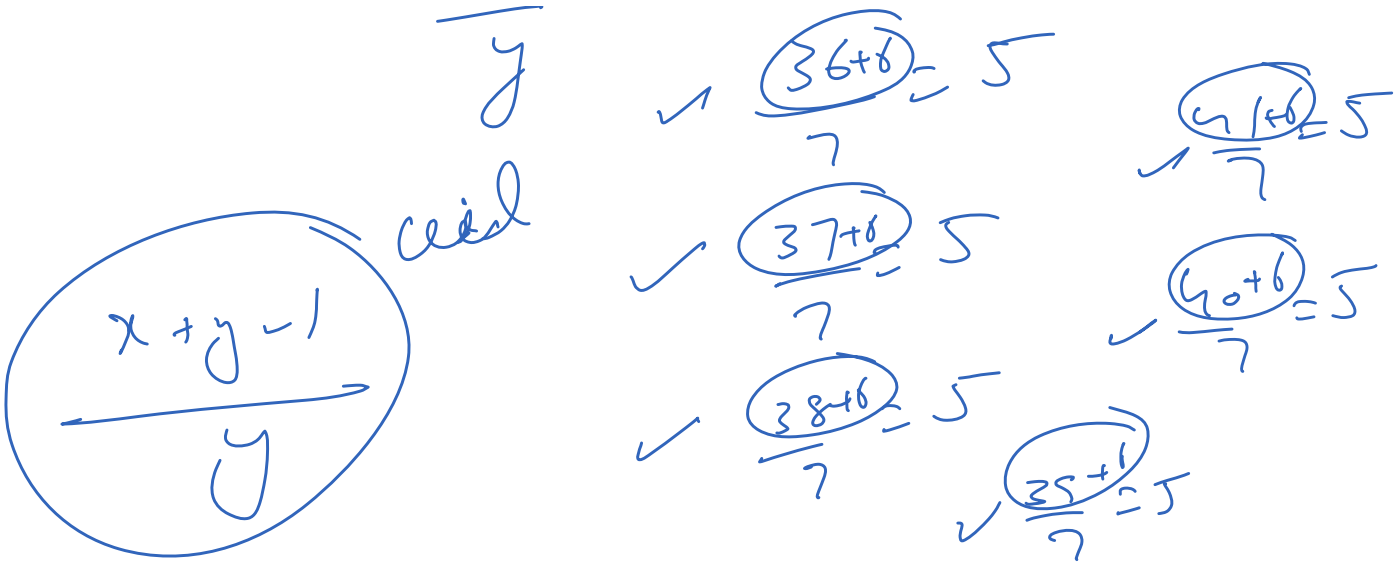
$(x+y-1)/y$ → ceil

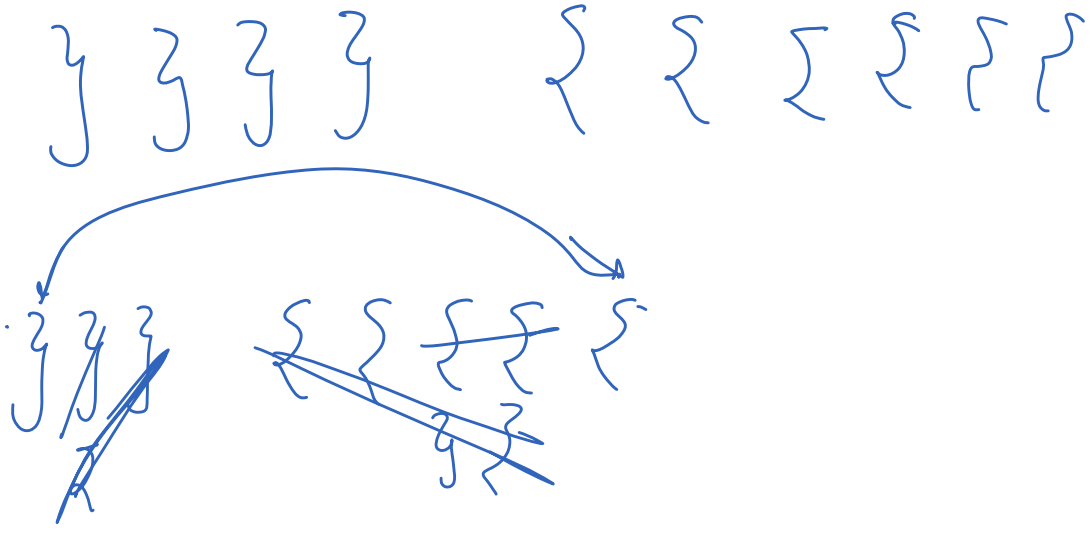


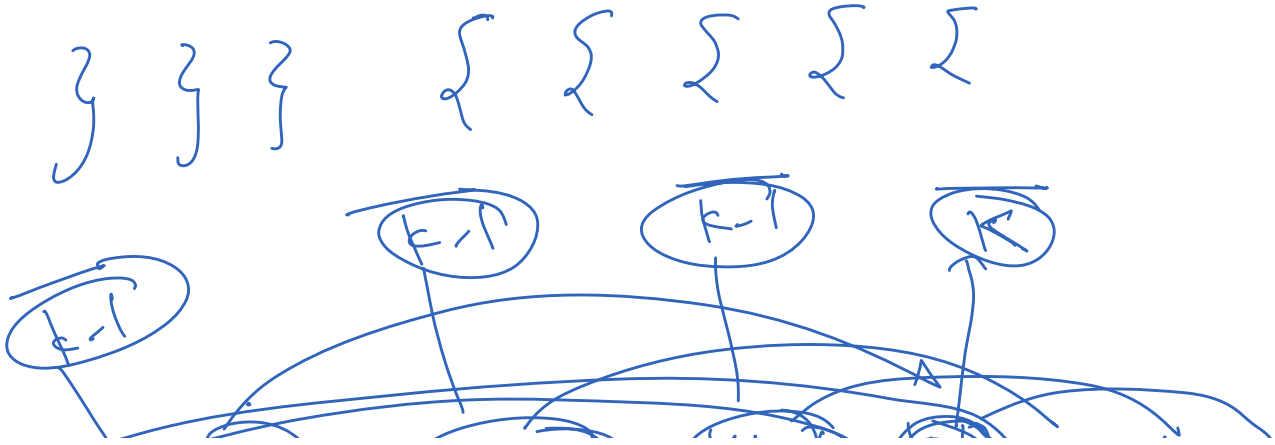


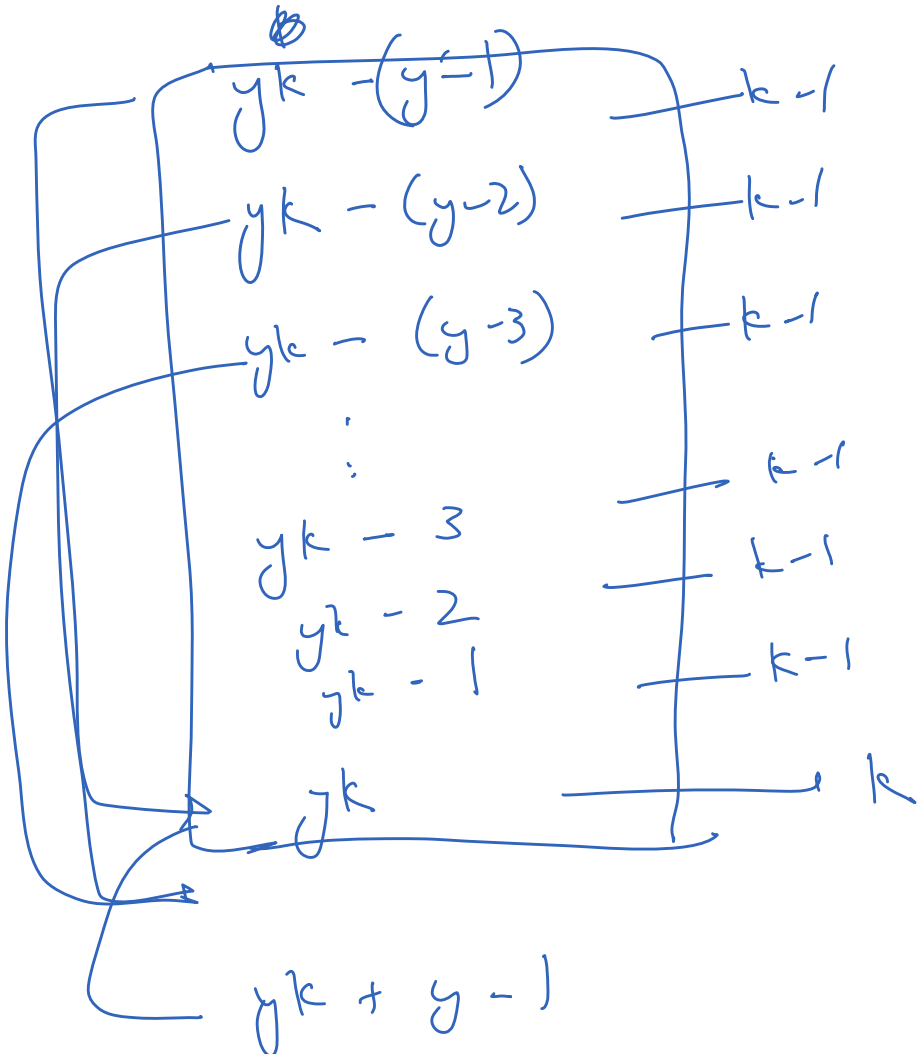
$\frac{35+6}{7} = 5$

$\frac{x}{y}$ $\frac{36+6}{7} = 5$


y
ceil
x + y - 1
y
36+6 = 5
7
37+6 = 5
7
38+6 = 5
7
41+6 = 5
7
40+6 = 5
7
35+1 = 5
7



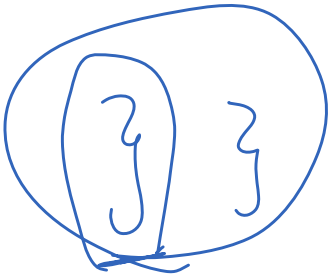

k-1
k-1
k-1
K



$yk + y - 1$

Floor $\frac{x}{y}$

Ceil $\frac{x + y - 1}{y}$



$((a+b)+(c+d))$

9:37 - 9:42

9:40 - 9:45

Programming / Stacks And Queues / Redundant Braces

Redundant Braces

Bookmark

Easy 36.7% Success 245 153

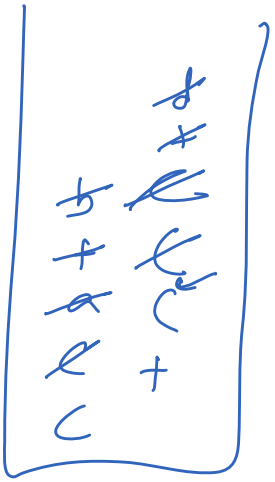
Asked In: Amazon

Problem Description

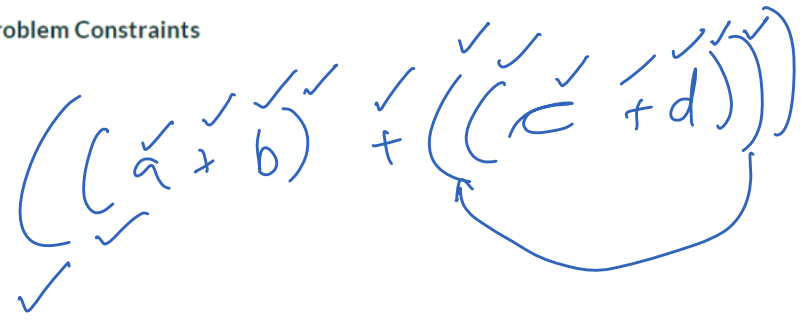
Given a string **A** denoting an expression. It contains the following operators `'+', '-', '*', '/'`.

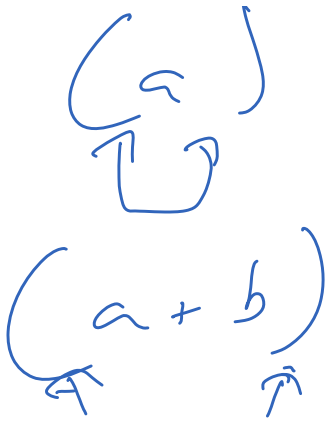
Chech whether A has redundant braces or not.

NOTE: A will be always a valid expression.



Problem Constraints



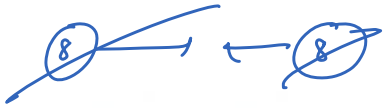


((a+b)+(e+(c*d)))/(a+c)

```
Input: asteroids = [5,10,-5]
Output: [5,10]
Explanation: The 10 and -5 collide resulting in 10. The 5 and 10 never collide.
```

Example 2:

```
Input: asteroids = [8,-8]
Output: []
Explanation: The 8 and -8 collide exploding each other.
```



Example 3:

```
Input: asteroids = [10,2,-5]
Output: [10]
Explanation: The 2 and -5 collide resulting in -5. The 10 and -5 collide resulting in 10.
```

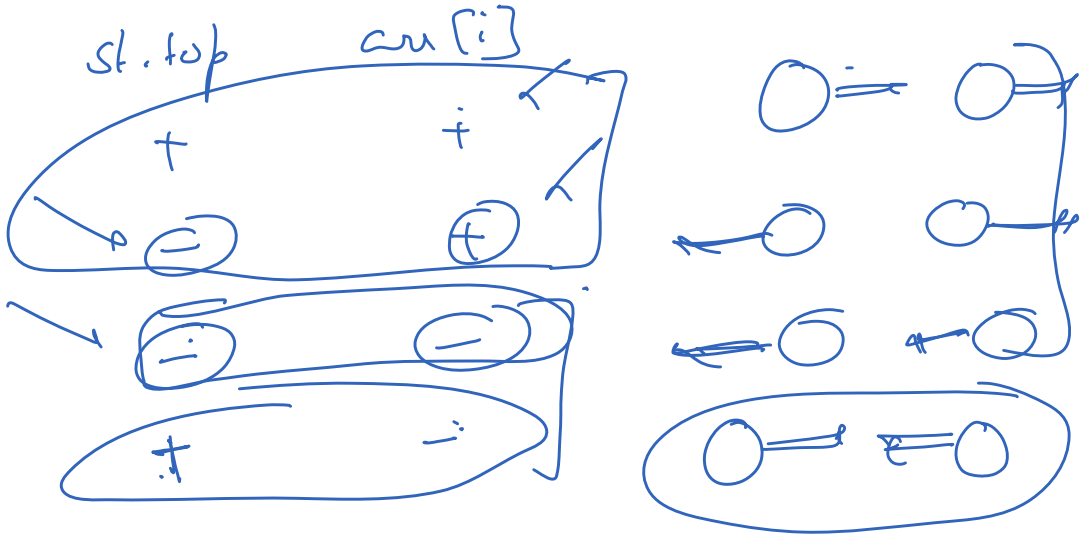
10:01 - 10:15 Try

2 5 -3 8 6 -7 4 -3

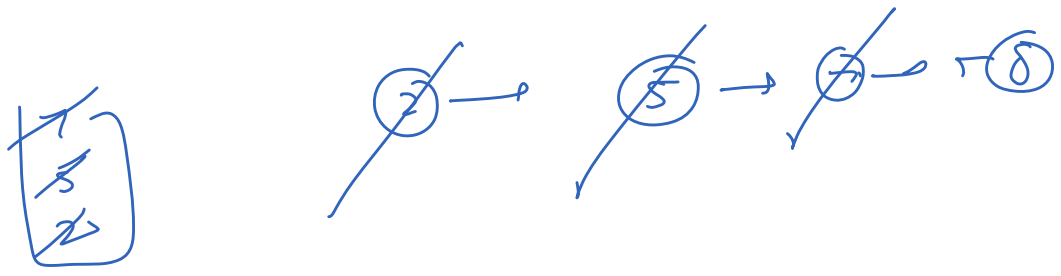
1. st.size == 0 → push

2. arr[i] > 0 → push

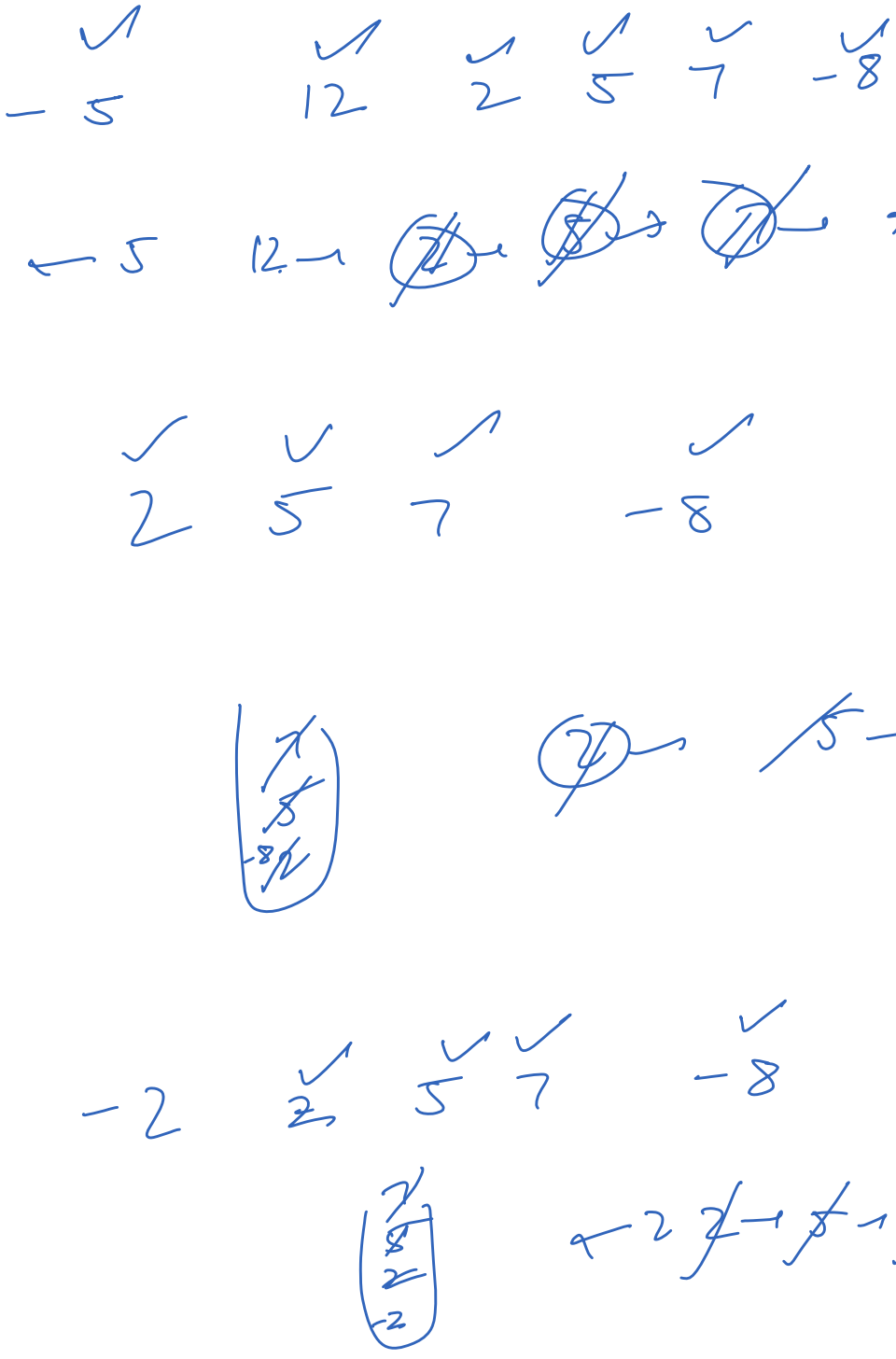
3. st.peek() < 0 → push

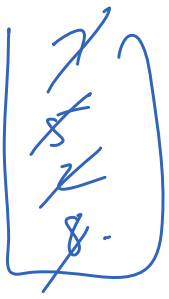


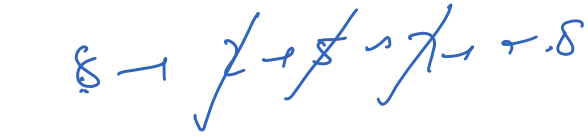
2 5 7 −8



−5 2 5 7 −8

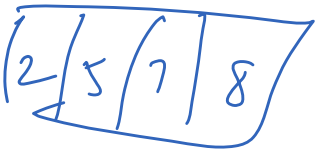






2 5 7 8

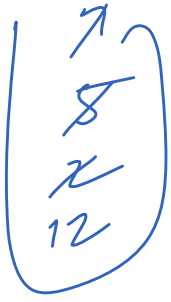
2 → 5 → 7 → 8.



12 2 5 7 −8

5 5. −5.

5.



5 5 −5

IPL 2021 - Final

Hard Accuracy: **48.48%** Submissions: **12481** Points: **8**

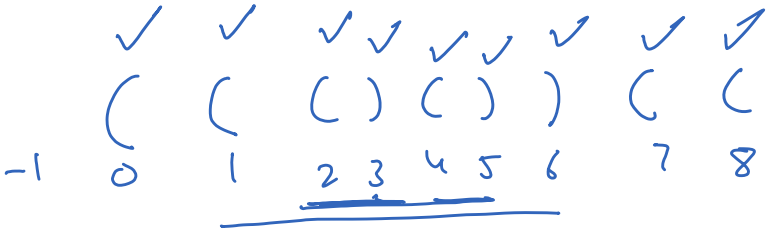
IPL 2021 Finals are here and it is between the most successful team of the IPL Mumbai Indians and the team striving to garb their first trophy Royal Challengers Banglore. Rohit Sharma, captain of the team Mumbai Indians has the most experience in IPL finals, he feels lucky if he solves a programming question before the IPL finals. So, he asked the team's head coach Mahela Jayawardene for a question. Question is, given a string **S** consisting only of opening and closing parenthesis 'ie **'('** and **')'**, the task is to find out the length of the longest valid parentheses substring.

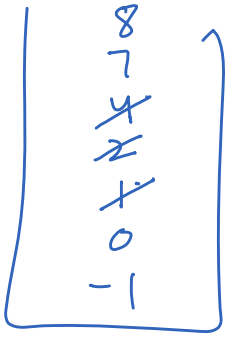
NOTE: The length of the smallest valid substring **( )** is **2**.

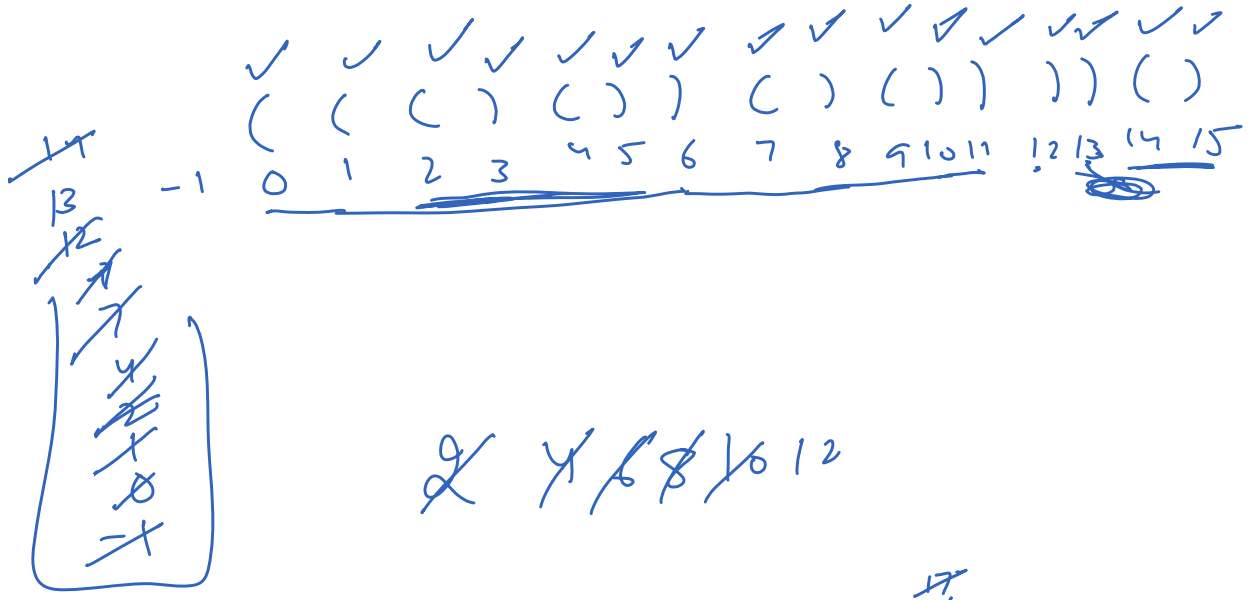
Example 1:

Input: S = "(()("
Output: 2
Explanation: The longest valid substring is "()". Length = 2.

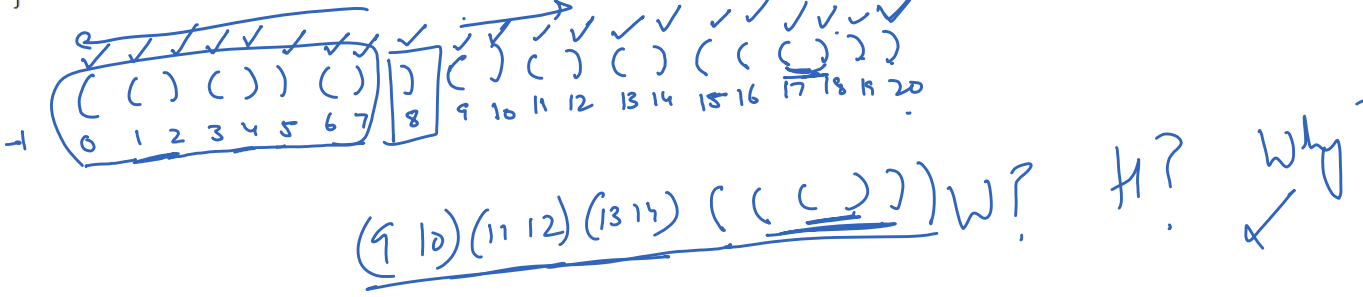
Example 2:

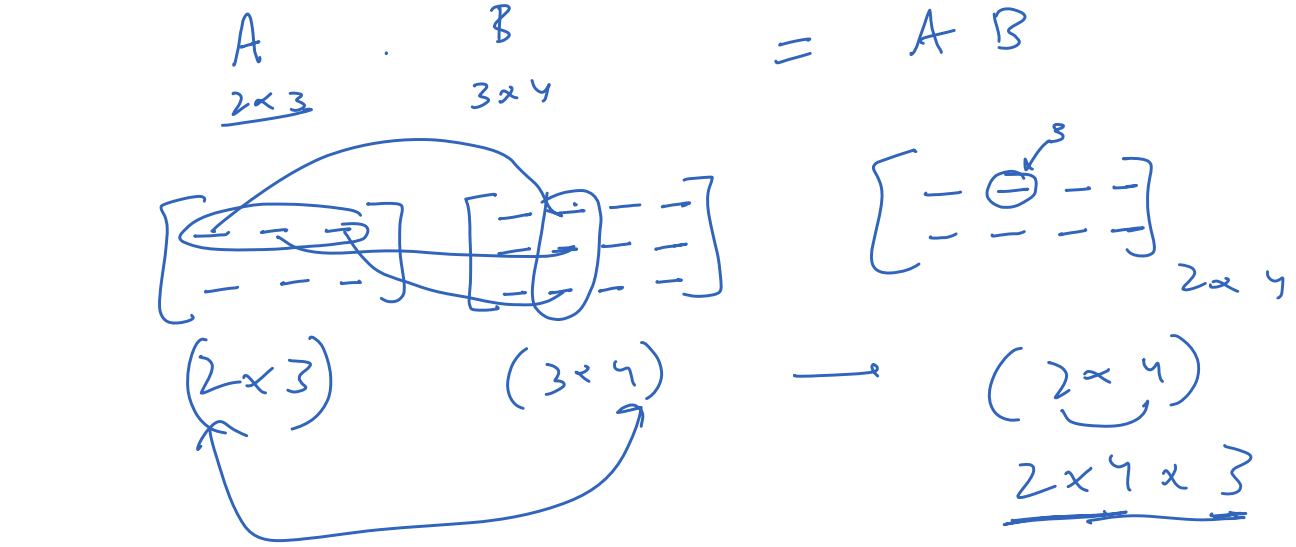


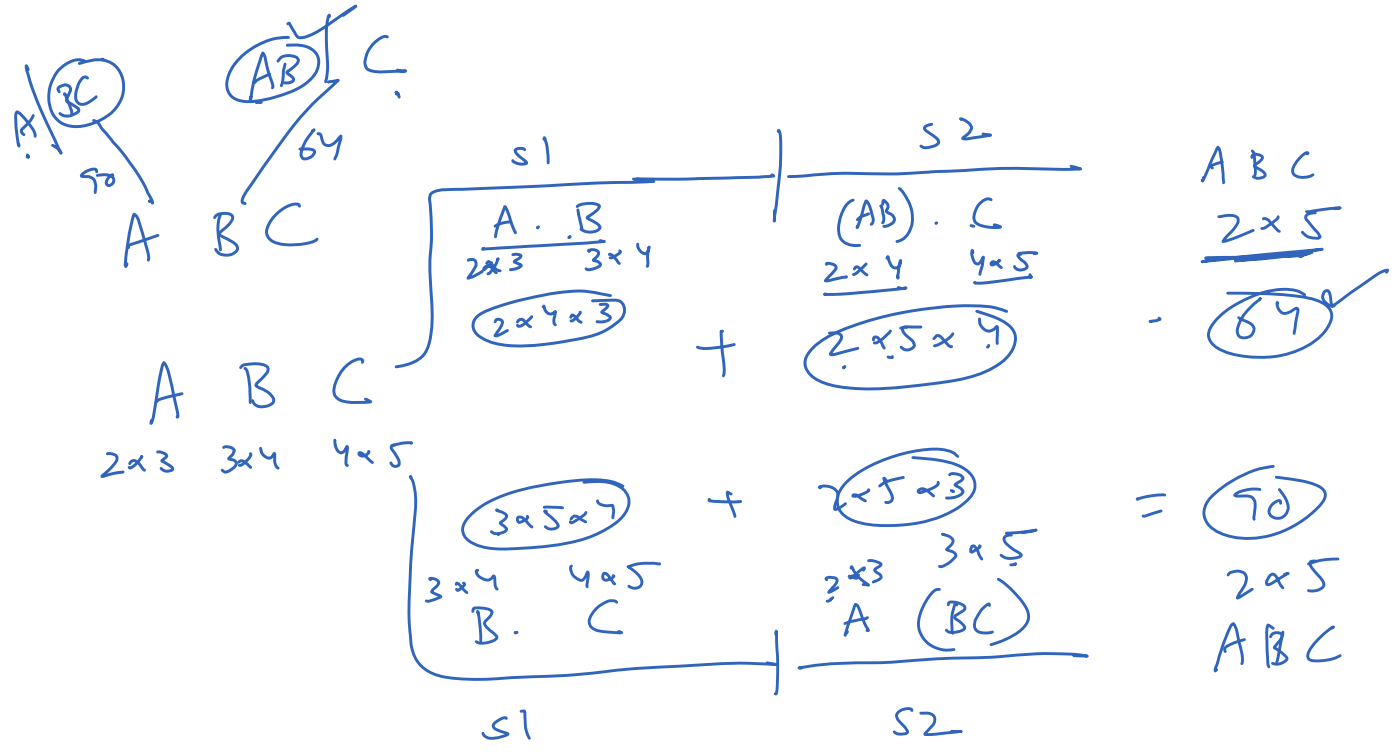


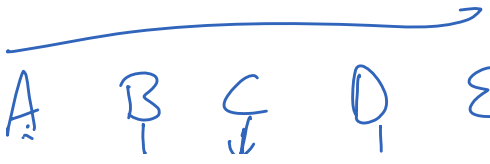


```
Stack<Integer> st = new Stack<>();
st.push(-1);
for(int i = 0; i < s.length(); i++){
    char ch = s.charAt(i);
    if(ch == '('){
        st.push(i);
    } else {
        st.pop();

        if(st.size() >0){
            int len = i - st.peek();
            res = Math.max(res, len);
        } else {
            st.push(i);
        }
    }
}
```




A . B = A B
2×3
3×4
2×4
(2×3) (3×4) → (2×4)
2×4×3


A (BC)
(AB) C
50
64
A B C
S1
S2
A . B
2×3 3×4
2×4×3
(AB) . C
2×4 4×5
2×5×4
A B C
2×5
64
A B C
2×3 3×4 4×5
3×5×4
3×4 4×5
B . C
2×5×3
2×3 3×5
A (BC)
= 90
2×5
A B C
S1
S2


A B C D E

$10 \times 20 \times 30 \times 40 \times 50 \times 60$

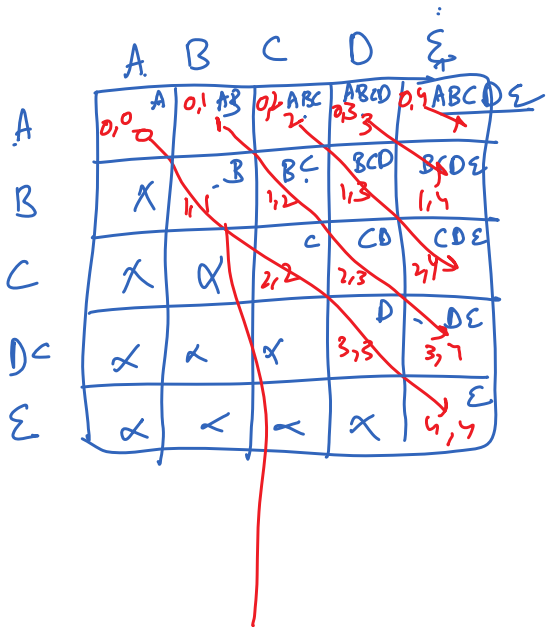
A – 10×20

B – 20×30

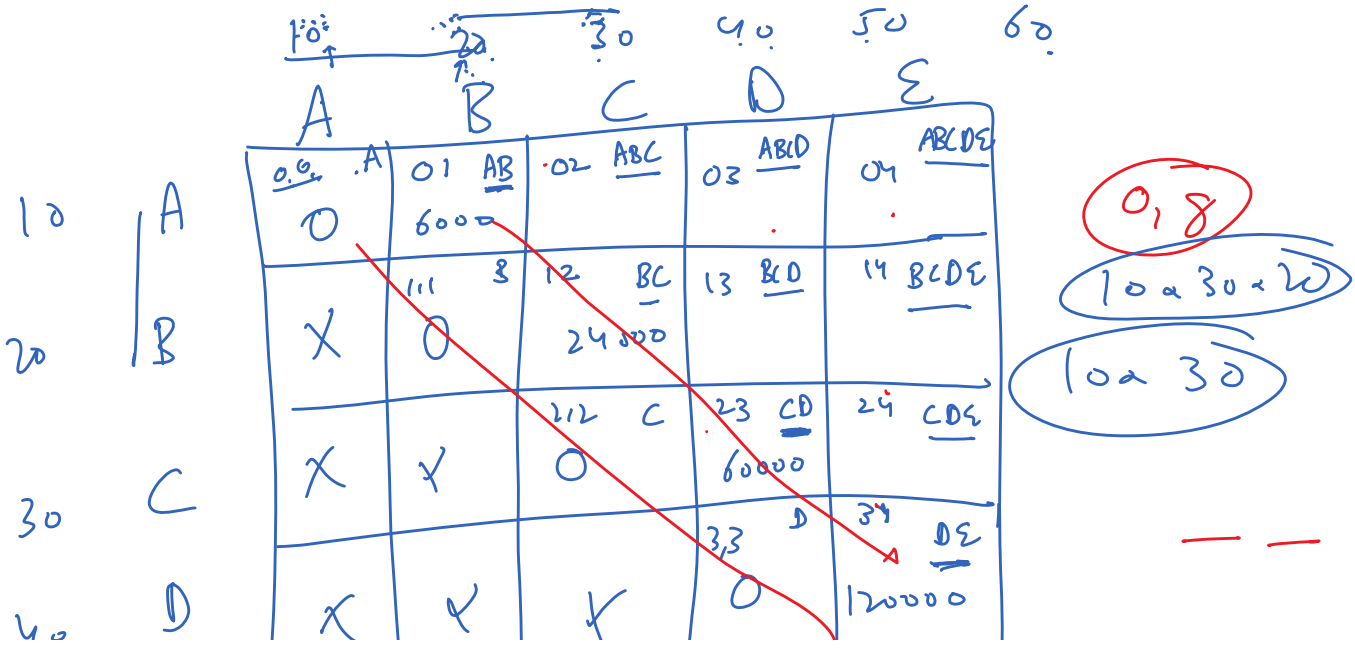
C – 30×40

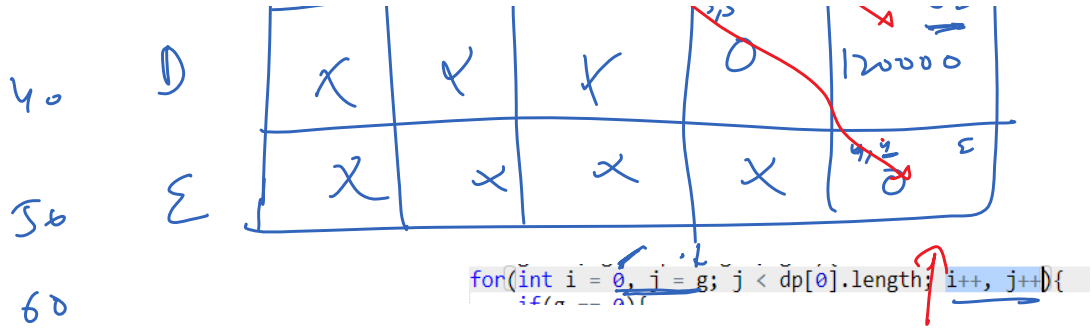
D – 40×50

E – 50×60



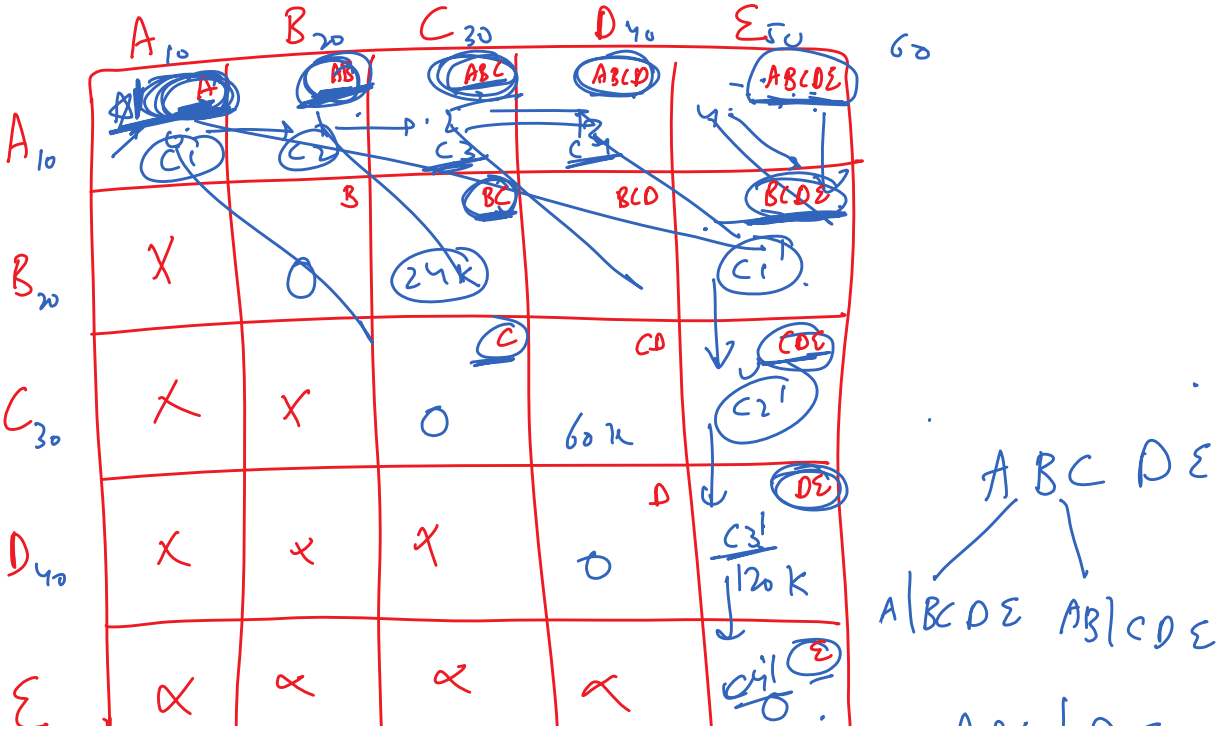
1. Store
2. Move
3. Calculate

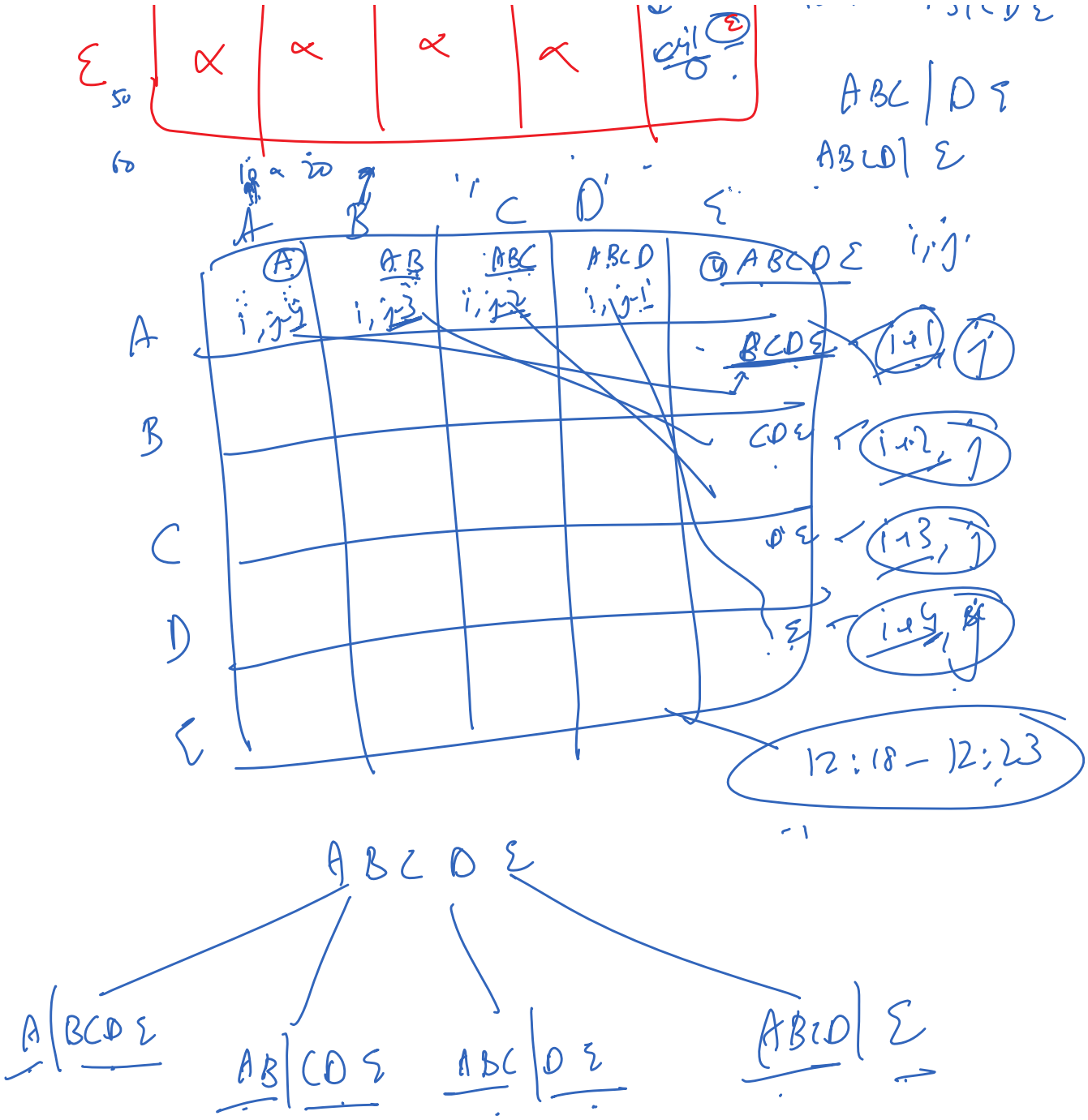




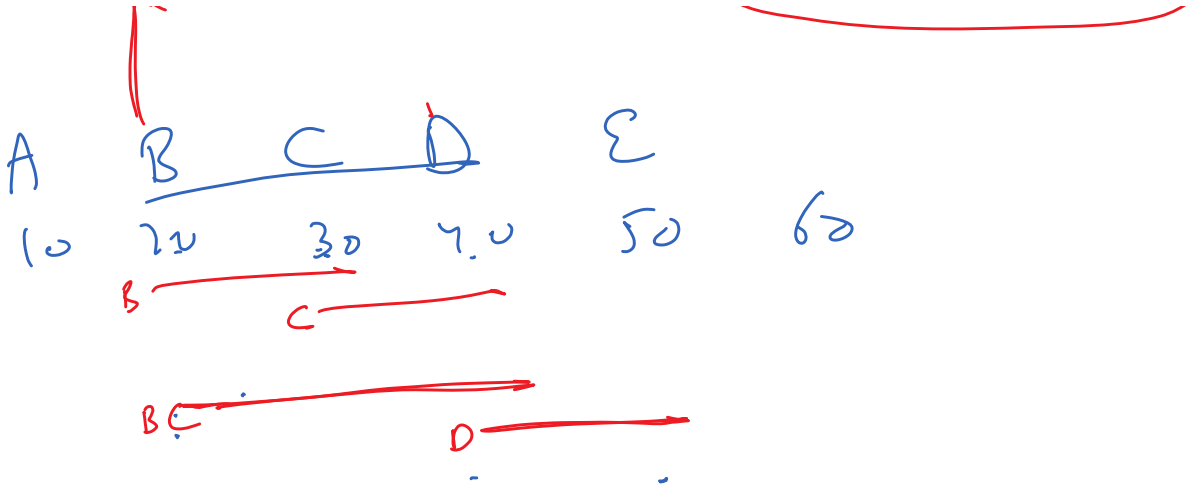
11:53 - 11:58

```
class Solution{
    static int matrixMultiplication(int N, int arr[])
    {
        // code here
        int[][] dp = new int[arr.length - 1][arr.length - 1];

        for(int g = 0; g < dp.length; g++){
            for(int i = 0, j = g; j < dp[0].length; i++, j++){
                if(g == 0){
                    dp[i][j] = 0;
                } else if(g == 1){
                    dp[i][j] = arr[i] * arr[j] * arr[j + 1];
                } else {
                    // bawaal
                }
            }
        }


        return dp[0][dp[0].length - 1];
    }
}
```






```
int left = dp[i][j - k2]; // arr[i] * arr[j - k2 + 1]
int right = dp[i + k1][j]; // arr[i + k1] * arr[ j + 1]
int multc = arr[i] * arr[j + 1] * arr[i + k1];

min = Math.min(min, left + right + multc);
```


A
B
C
D
E
10
20
30
40
50
60
B
C
BC
D