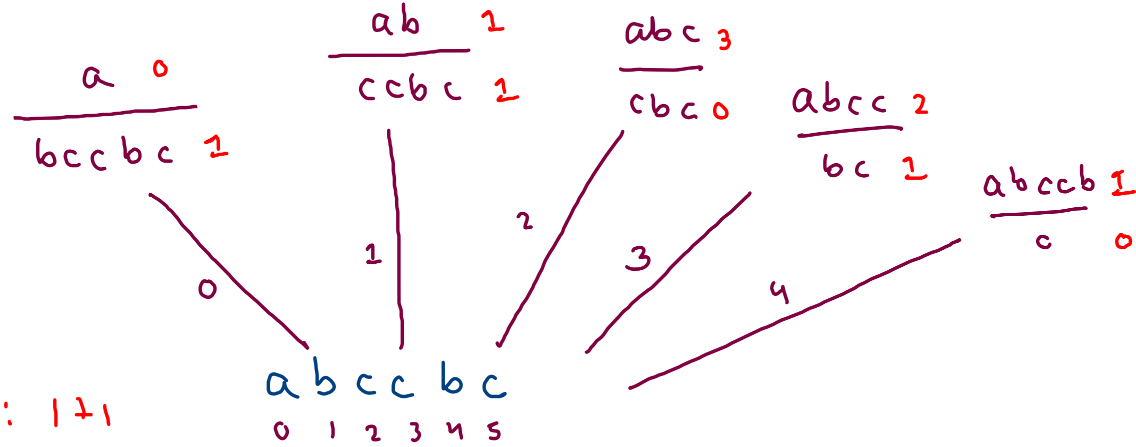
Minimum Palindromic Cut

a | b | c | c | b | c

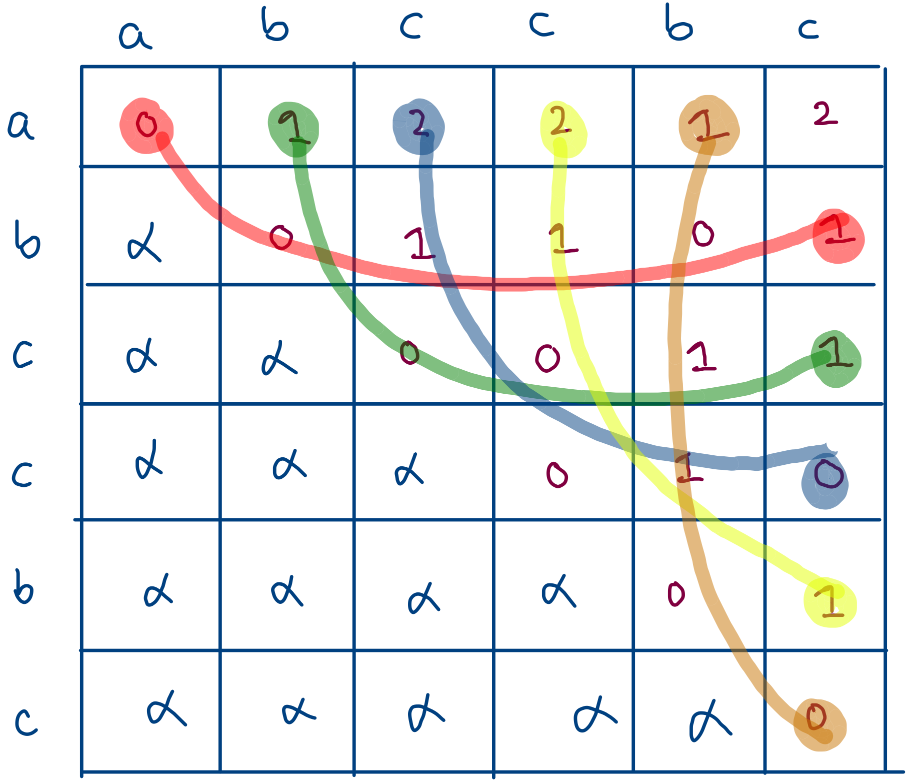
a | b | c c | b | c

a | b c c b | c

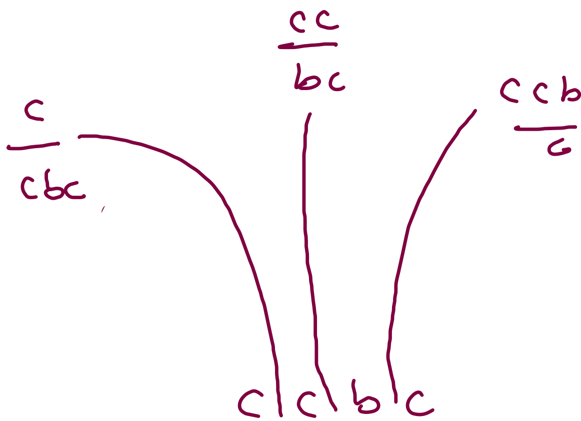


ans: 1+1

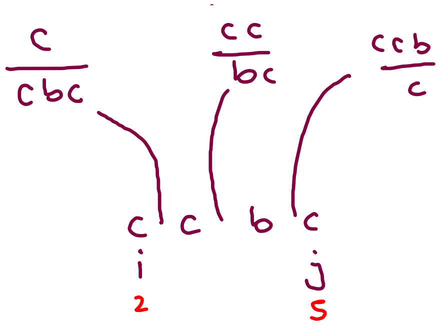
left - right strategy



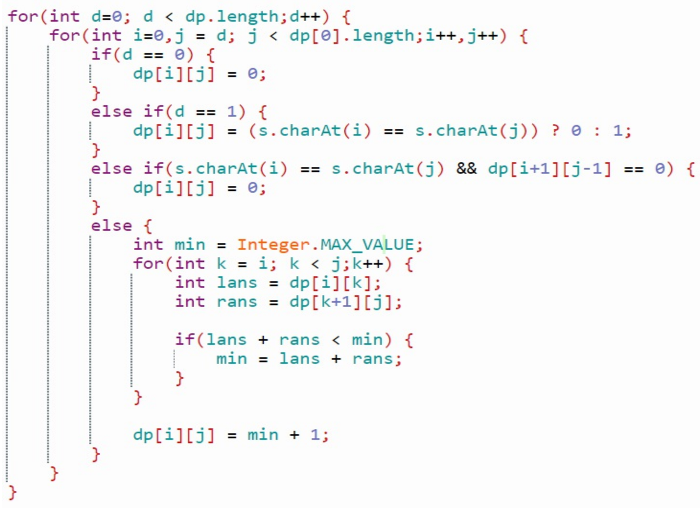
	a	b	c	c	b	c
a	0	1	2	2	1	2
b	α	0	1	1	0	1
c	α	α	0	0	1	1
c	α	α	α	0	1	0
b	α	α	α	α	0	1
c	α	α	α	α	α	0



	a_0	b_1	c_2	c_3	b_4	c_5
a_0	0	1	2	2	1	2
b_1	α	0	1	1	0	1
c_2	α	α	0	0	1	1
c_3	α	α	α	0	1	0
b_4	α	α	α	α	0	1
c_5	α	α	α	α	α	0

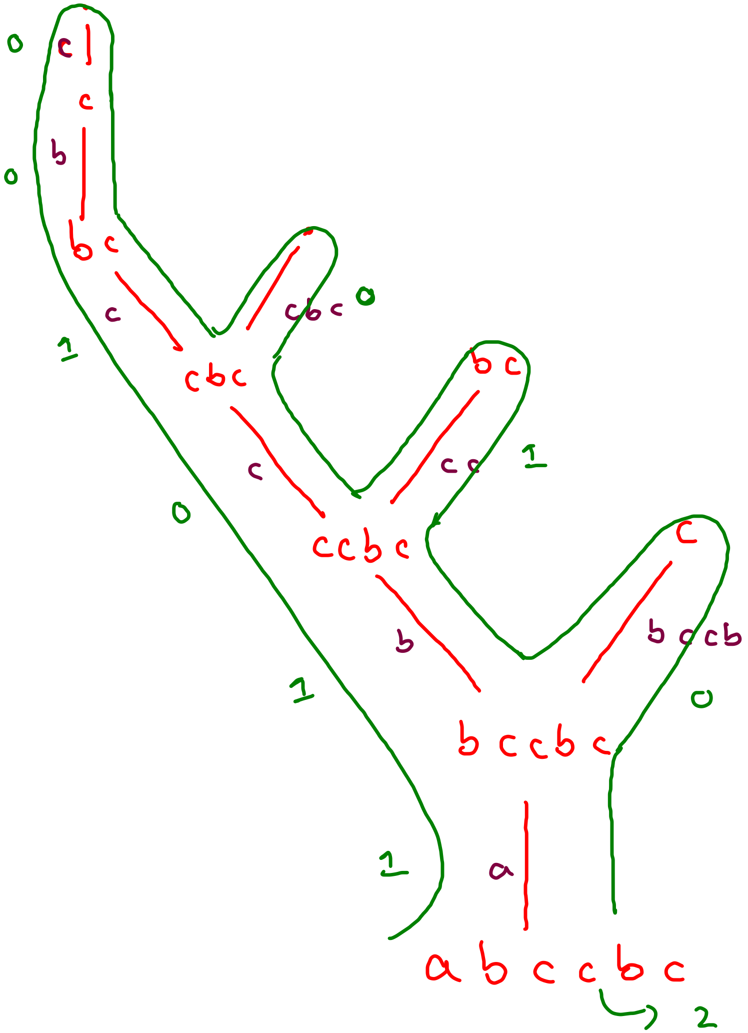


	dp	op
k : 2	2, 2	3, 5
k : 3	2, 3	4, 5
k : 4	2, 4	5, 5



```
for(int d=0; d < dp.length;d++) {
    for(int i=0,j = d; j < dp[0].length;i++,j++) {
        if(d == 0) {
            dp[i][j] = 0;
        }
        else if(d == 1) {
            dp[i][j] = (s.charAt(i) == s.charAt(j)) ? 0 : 1;
        }
        else if(s.charAt(i) == s.charAt(j) && dp[i+1][j-1] == 0) {
            dp[i][j] = 0;
        }
        else {
            int min = Integer.MAX_VALUE;
            for(int k = i; k < j;k++) {
                int lans = dp[i][k];
                int rans = dp[k+1][j];

                if(lans + rans < min) {
                    min = lans + rans;
                }
            }

            dp[i][j] = min + 1;
        }
    }
}
```

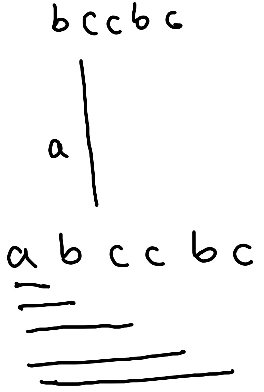
optimisation



$O(n^2)$

a	b	c	c	b	c
2	1	1	0	1	0
0	1	2	3	4	5

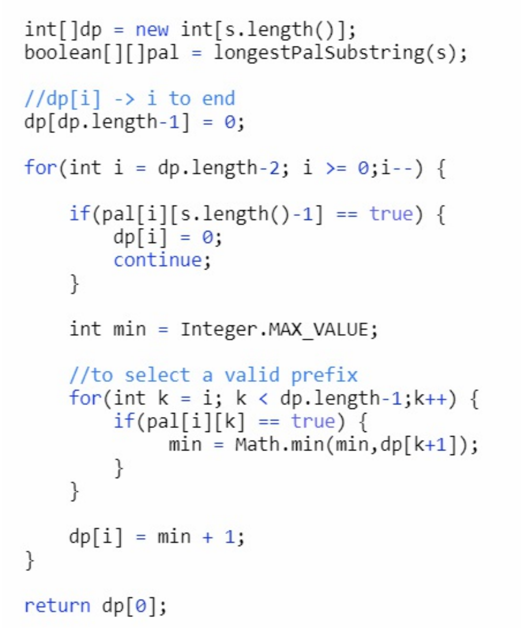
	a_0	b_1	c_2	c_3	b_4	c_5
a_0	T	F	F	F	F	F
b_1	α	T	F	F	T	F
c_2	α	α	T	T	F	F
c_3	α	α	α	T	F	T
b_4	α	α	α	α	T	F
c_5	α	α	α	α	α	T



$i \to$ i to end string

min cut to make each part palindromic.

	a_0	b_1	c_2	c_3	b_4	c_5
a_0	T	F	F	F	F	F
b_1	α	T	F	F	T	F
c_2	α	α	T	T	F	F
c_3	α	α	α	T	F	T
b_4	α	α	α	α	T	F
c_5	α	α	α	α	α	T

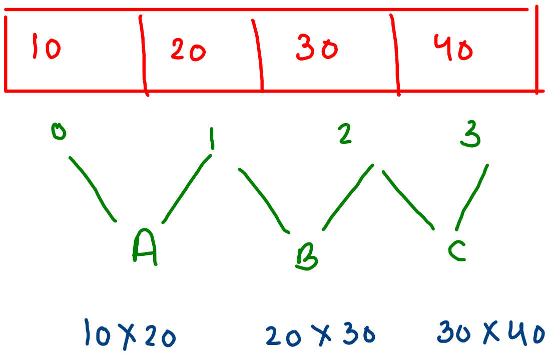


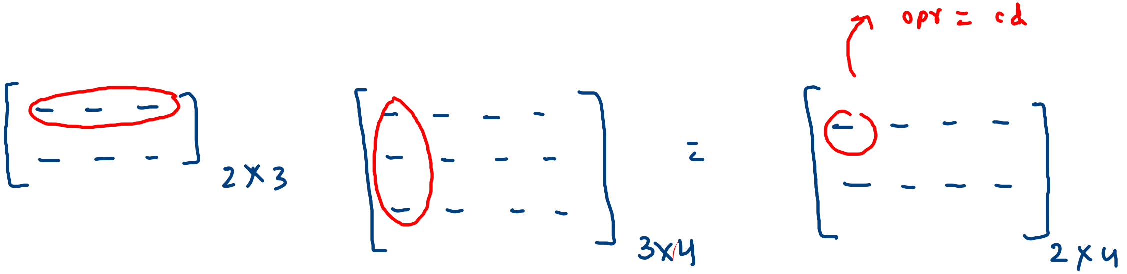
```
int[]dp = new int[s.length()];
boolean[][]pal = longestPalSubstring(s);

//dp[i] -> i to end
dp[dp.length-1] = 0;

for(int i = dp.length-2; i >= 0;i--) {

    if(pal[i][s.length()-1] == true) {
        dp[i] = 0;
        continue;
    }

    int min = Integer.MAX_VALUE;

    //to select a valid prefix
    for(int k = i; k < dp.length-1;k++) {
        if(pal[i][k] == true) {
            min = Math.min(min,dp[k+1]);
        }
    }

    dp[i] = min + 1;
}

return dp[0];
```

i -> i to end ss ans

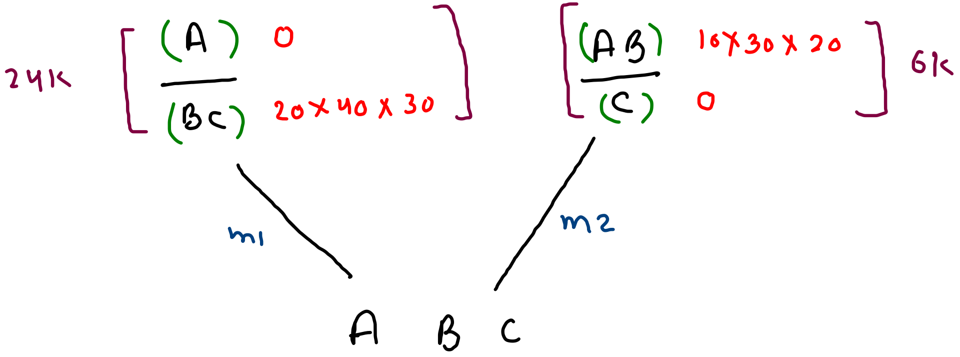
a	b	c	c	b	c
2	1	1	0	1	0
0	1	2	3	4	5
abccbc	bccbc	ccbc	cbc	bc	c

Matrix Chain Multiplication

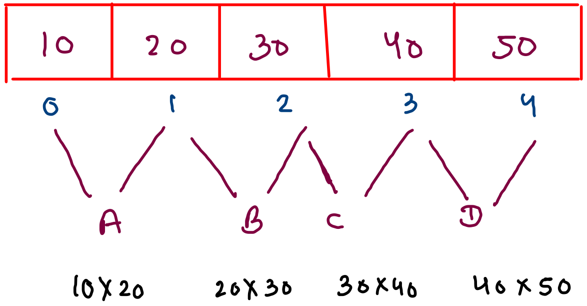


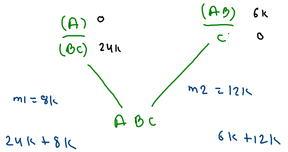


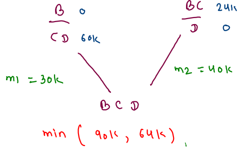
total opr = 2 * 4 * cd



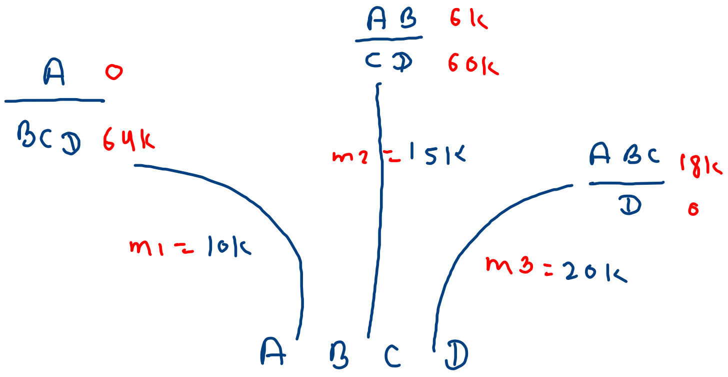
ans : min [(24k + m1) , (6k + m2)]







	A	B	C	D
A	0	6K	18K	38K
B	α	0	24K	64K
C	α	α	0	60K
D	α	α	α	0



A : 10X20   m1 = 10 * 20 * 50 = 10k

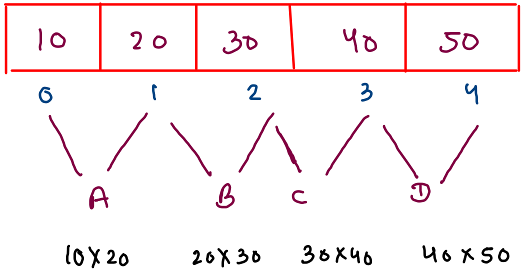
BCD ; 20X50

AB : 10X30   m2 = 10 * 30 * 50 = 15k

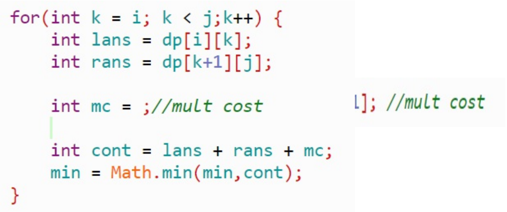
CD : 30 X 50

ABC : 10X40   m3 = 10 * 40 * 50 = 20k

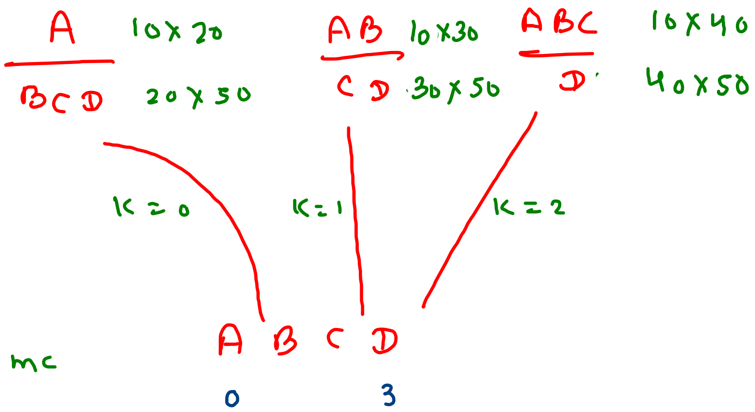
D : 40 X 50



		10	20	30	40	50
		A_0	B_1	C_2	D_3	
10	A_0	0	6k	18k	38k	
20	B_1	∝	0	24k	64k	
30	C_2	∝	∝	0	60k	
40	D_3	∝	∝	∝	0	
50						



```
for(int k = i; k < j;k++) {
    int lans = dp[i][k];
    int rans = dp[k+1][j];

    int mc = ;//mult cost                l]; //mult cost

    int cont = lans + rans + mc;
    min = Math.min(min,cont);
}
```

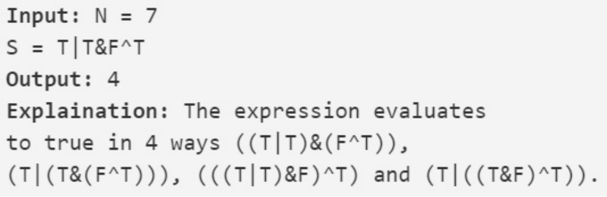


	lans	+	rans	+	mc
k = 0	0		64k		10k
k = 1	6k		60k		15k
k = 2	18k		0		20k

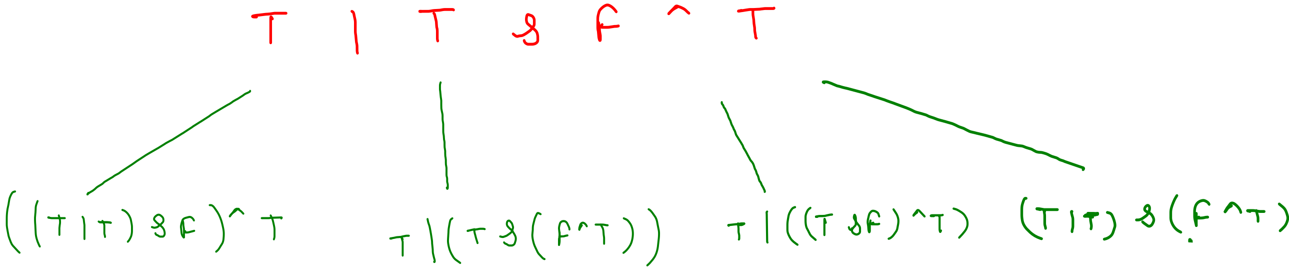
ld → i x (k+1)

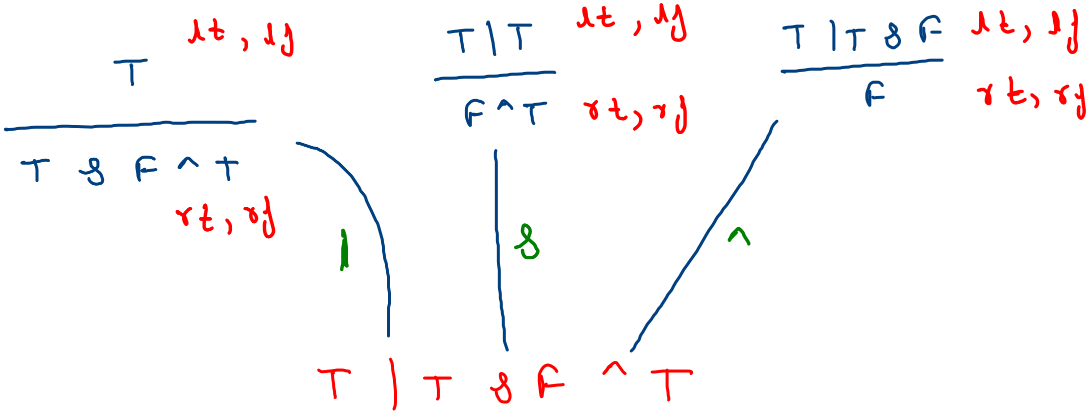
rd → (k+1) x (j+1)

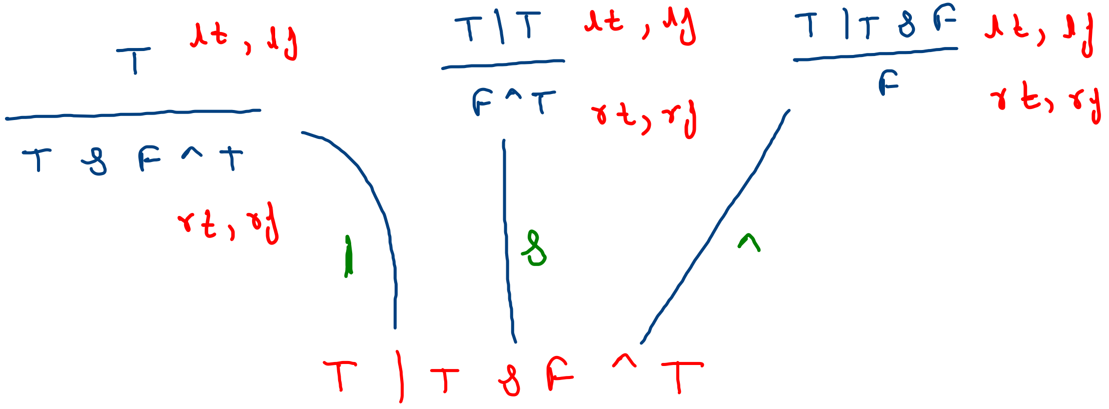
Boolean Parenthesization



Input: N = 7
S = T|T&F^T
Output: 4
Explaination: The expression evaluates to true in 4 ways ((T|T)&(F^T)), (T|(T&(F^T))), (((T|T)&F)^T) and (T|((T&F)^T)).







interaction [(F | F ^T) | (F ^ T ^ F)

lt = 2 rt = 2
lf = 0 rf = 0

opr	true count	false count
|	lt * rt + lt * rf + lf * rt	lf * rf
&	lt * rt	lt * rf + lf * rt + lf * rf
^	lt * rf + lf * rt	lf * rf + lt * rt

opr	true count	false count
|	lt * rt + lt * rf + lf * rt	lf * rf
&	lt * rt	lt * rf + lf * rt + lf * rf
^	lt * rf + lf * rt	lf * rf + lt * rt

T | T & F ^ T

	T |	T &	F ^	T
T |	1,0	1,0	1,1	4,1
T &	α	1,0	0,1	2,0
F ^	α	α	0,1	1,0
T	α	α	α	1,0

(tc, fc)

pair

T | T & F ^ T

S1 : T T F T

S2 : | & ^

$\frac{T}{F \wedge T}$ lt = 1, lf = 0 / rt = 1, rf = 0

$\frac{T \& F}{T}$ lt = 0, lf = 1 / rt = 1, rf = 0

t = 1
f = 0

&

^

t = 1
f = 0

T & F ^ T

↳ 2, 0