Count Substring of 0 and 1

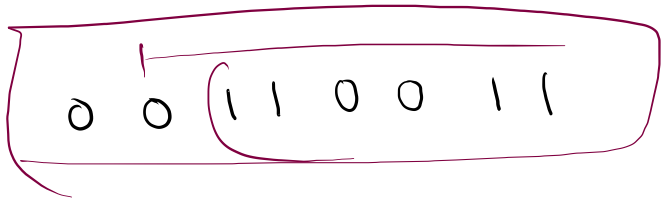
Sample Input 0

```
00110011
```

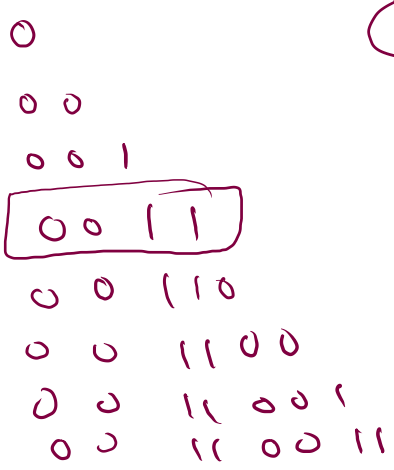
Sample Output 0

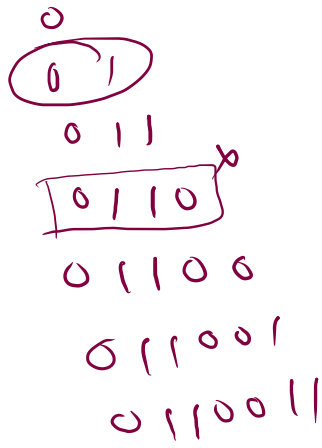
```
6
```

8 →



$\frac{8(9)}{2} = 36$

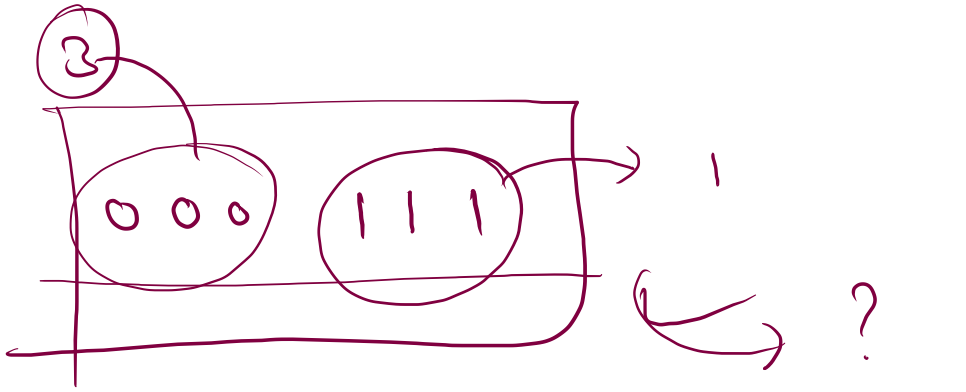


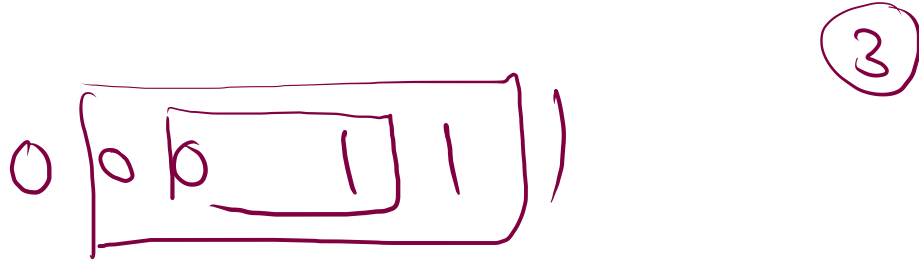


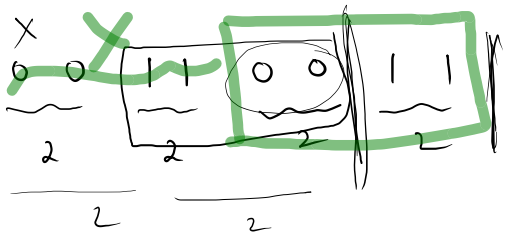
? 1100

0011

non-empty
or
len ≥ 1


3
0 0 0
1 1 1
1
?


3
0 0 0 1 1 1



ans

4 +2

= 6

C = X 2

X

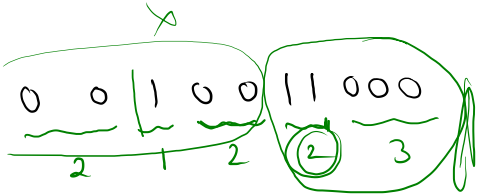
6

01

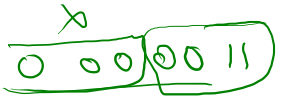
10

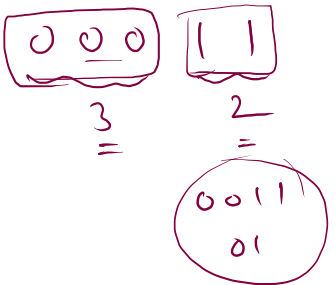
0011

01



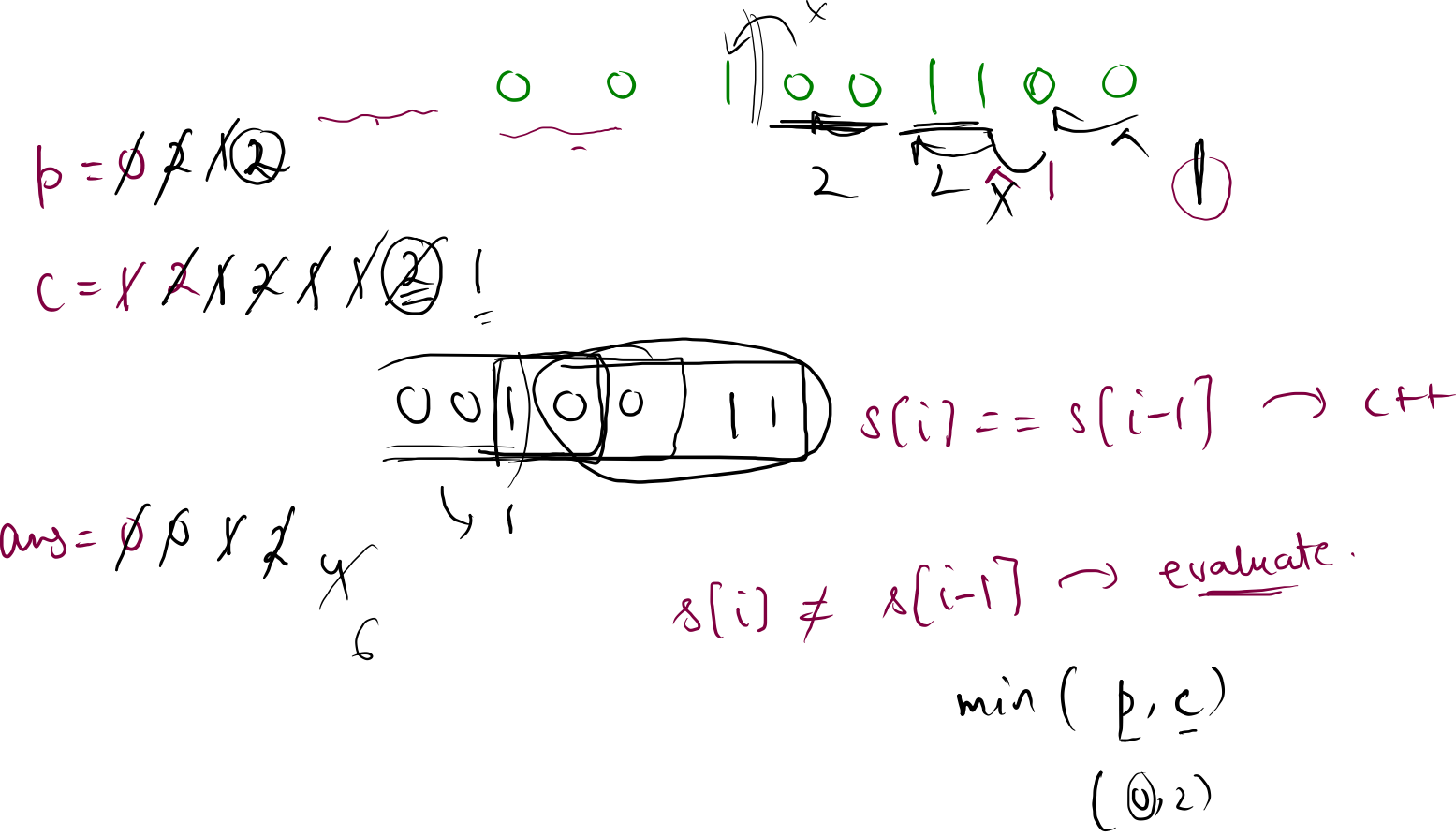
min

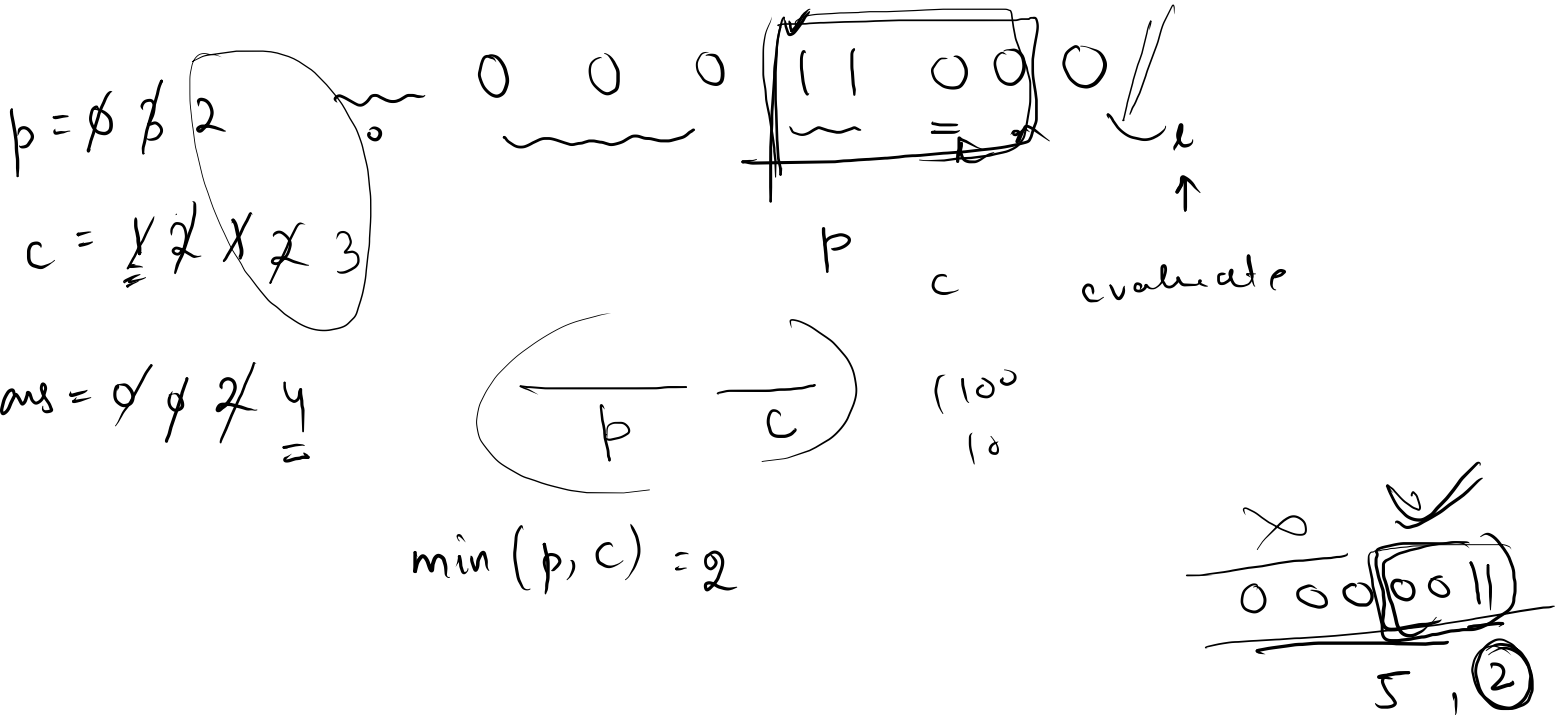




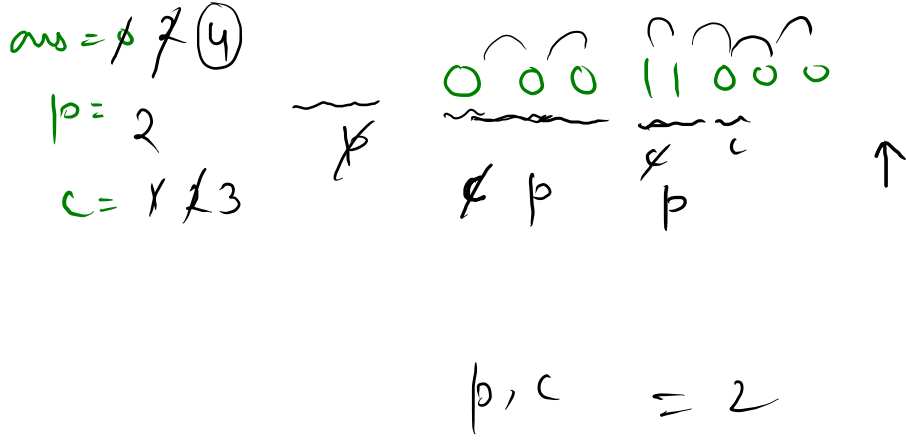
min (3, 2)

= 2


p = 0 2 1 2
c = 1 2 1 2 1 1 2 1
0 0 1 0 0 1 1
1
ans = 0 0 1 2 4
6
s[i] == s[i-1] → c++
s[i] ≠ s[i-1] → evaluate.
min (p, c)
(0, 2)


p = 0 0 2
c = 1 2 1 2 3
ans = 0 0 2 4
0 0 0 1 1 0 0 0
p
c
evaluate
p
c
100
10
min (p, c) = 2
0 0 0 0 0 1 1
5 , 2

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String s = scn.next();

    int ans = 0;
    int prev = 0;
    int curr = 1;

    for(int i = 1; i < s.length(); i++){
        if(s.charAt(i) == s.charAt(i-1)){
            curr++;
        }
        else{
            //evaluate
            ans += Math.min(prev, curr);
            prev = curr;
            curr = 1;
        }
    }

    ans += Math.min(prev, curr);

    System.out.println(ans);

}
```

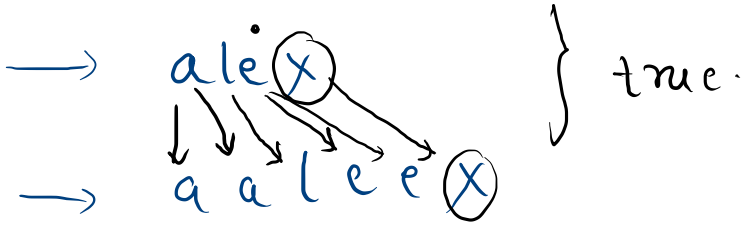


925. Long Pressed Name

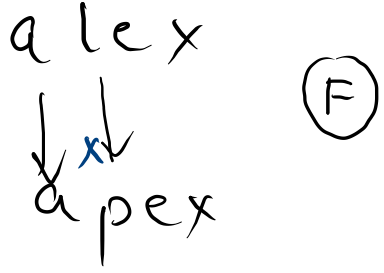
Easy 2230 320 Add to List Share

Your friend is typing his `name` into a keyboard. Sometimes, when typing a character `c`, the key might get *long pressed*, and the character will be typed 1 or more times.

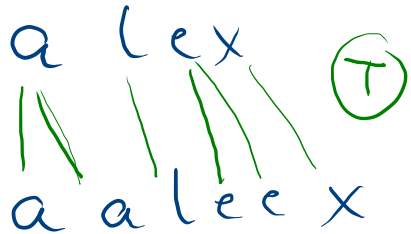
You examine the `typed` characters of the keyboard. Return `True` if it is possible that it was your friend's name, with some characters (possibly none) being long pressed.



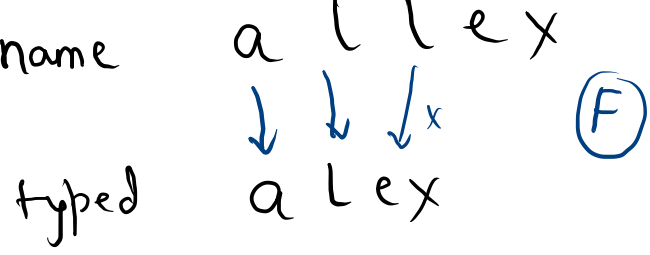

```
name = "alex", typed = "aaleex"
```



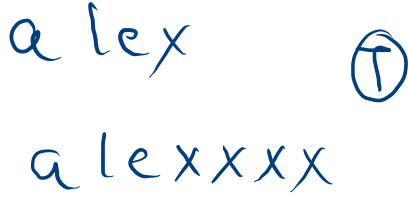
F



T



F



T

n → aalex

t → aaaalex

F

n → moon

t → mon

n → allex

t → allllllecxx

T

n → alex

t → aalex

T

n → aalex

t → alex

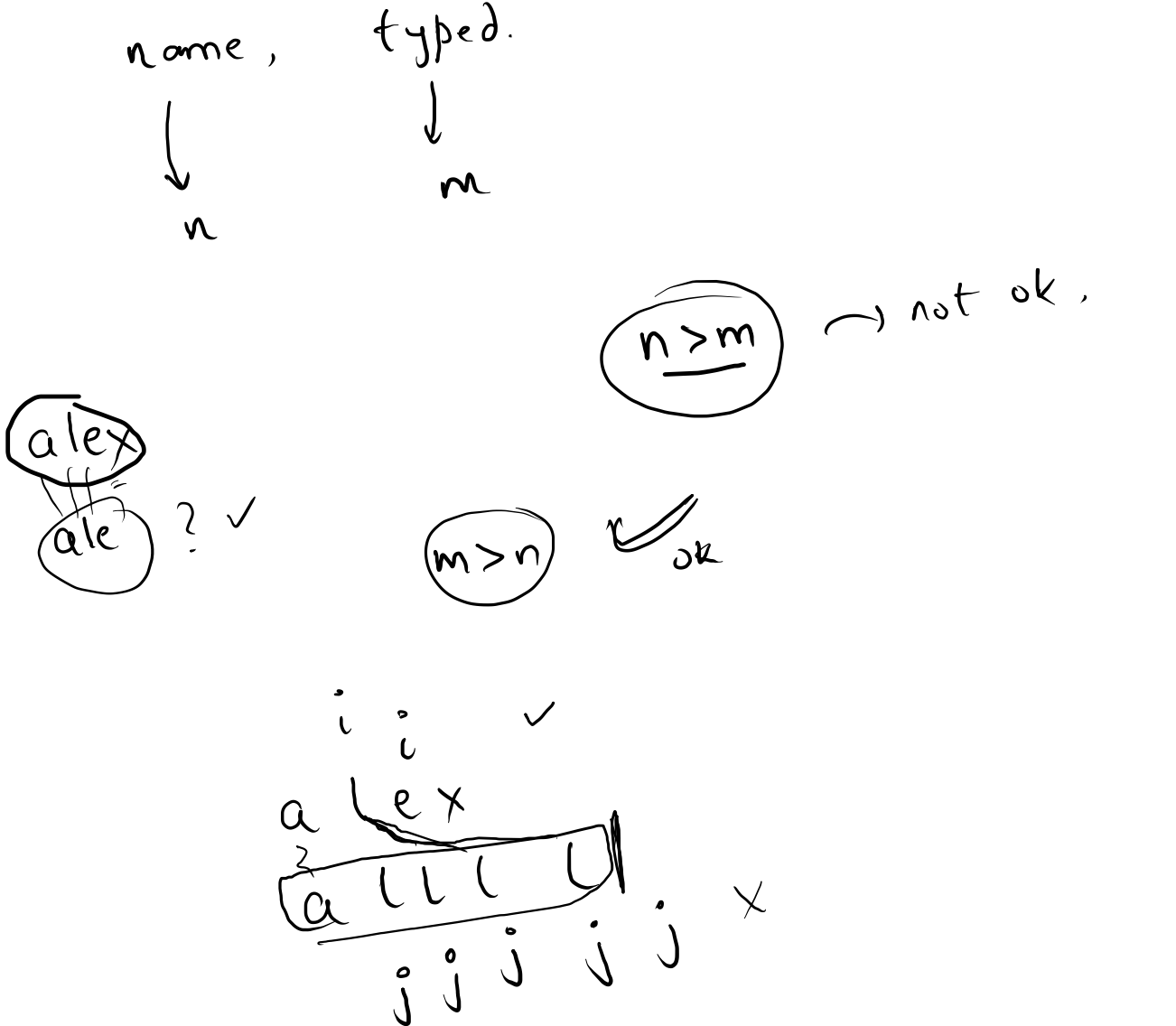
F

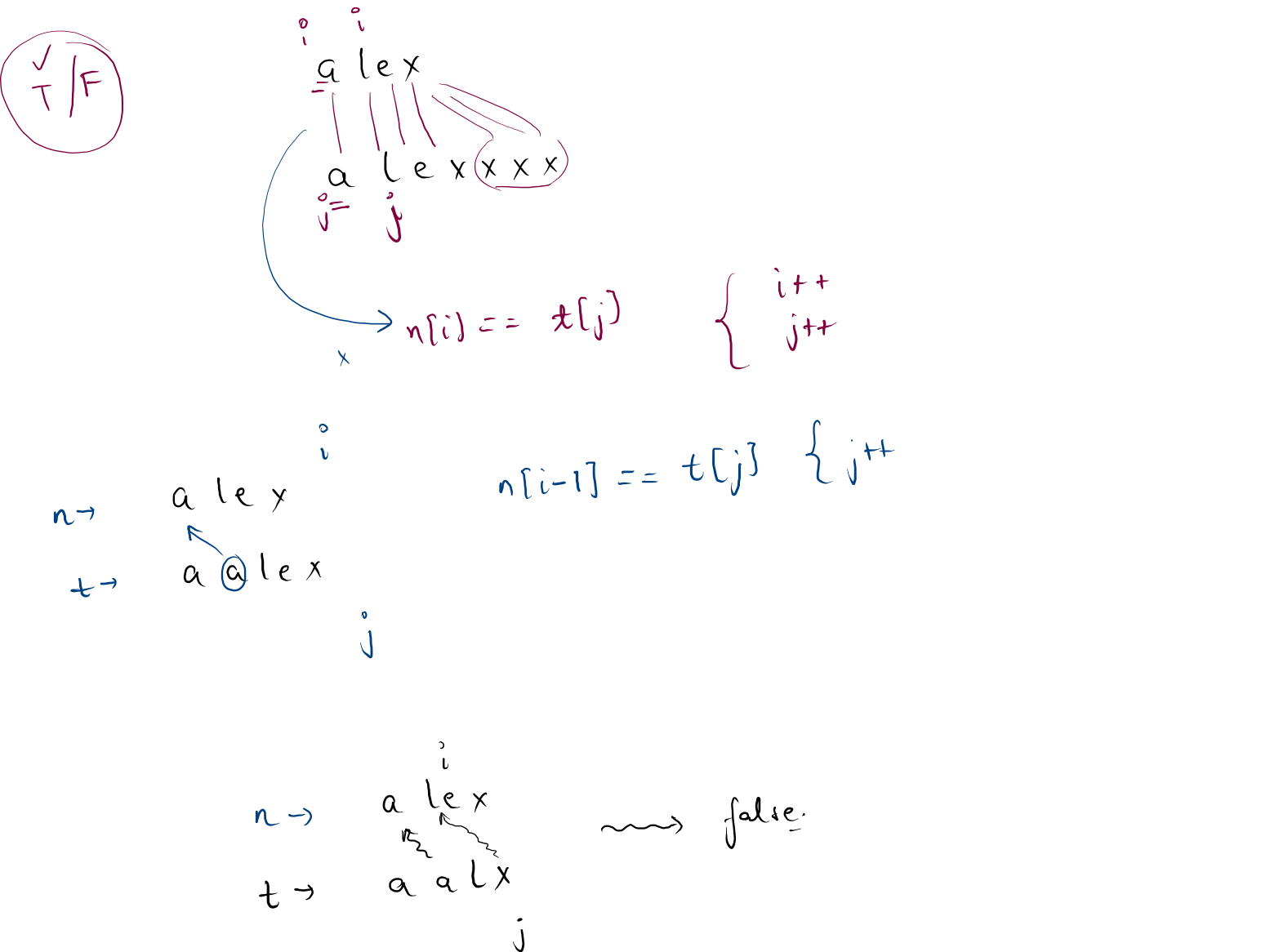
alex

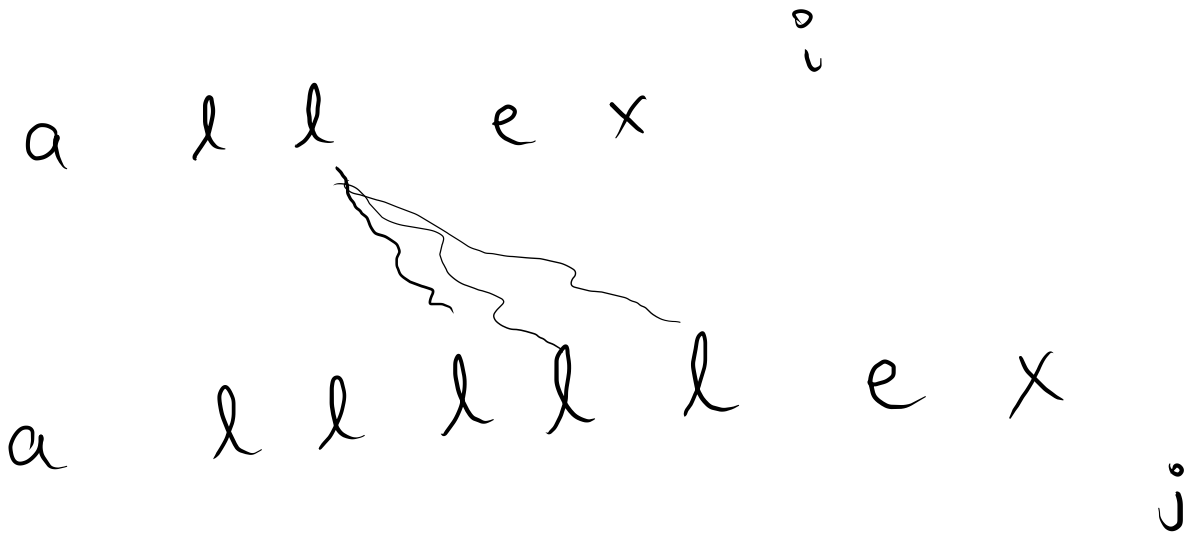
aalex

ch from 'n' can map with multiple ch from 't'

& oppsite is not possible.


name,
typed.
n
m
n>m
not ok.
alex
ale
?
m>n
ok
i
i
a
l
e
x
a
l
l
l
j
j
j
j
j
j
x


T/F
alex
alexxxx
i
j
n[i] == t[j]
i++
j++
x
n[i-1] == t[j] { j++
n →
t →
alex
a@lex
false.


a l l e x
a l l l l l e x

Critical.

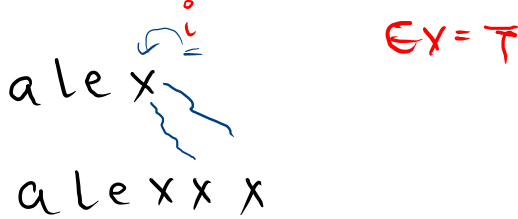
No.1

n → alex

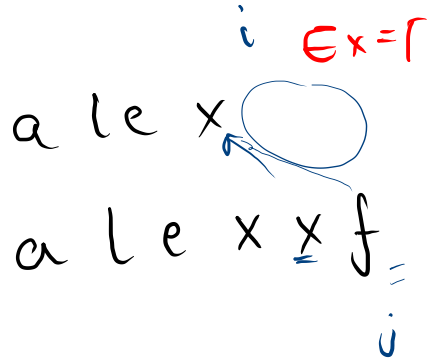
t → alexxxxf

```
//j is available and i ended
while(j < m){
    if(name.charAt(i-1) == typed.charAt(j)){
        j++;
    }
    else{
        return false;
    }
}
```

ij →



else →



imp.

i → should end.

↳ i == n

i

alex

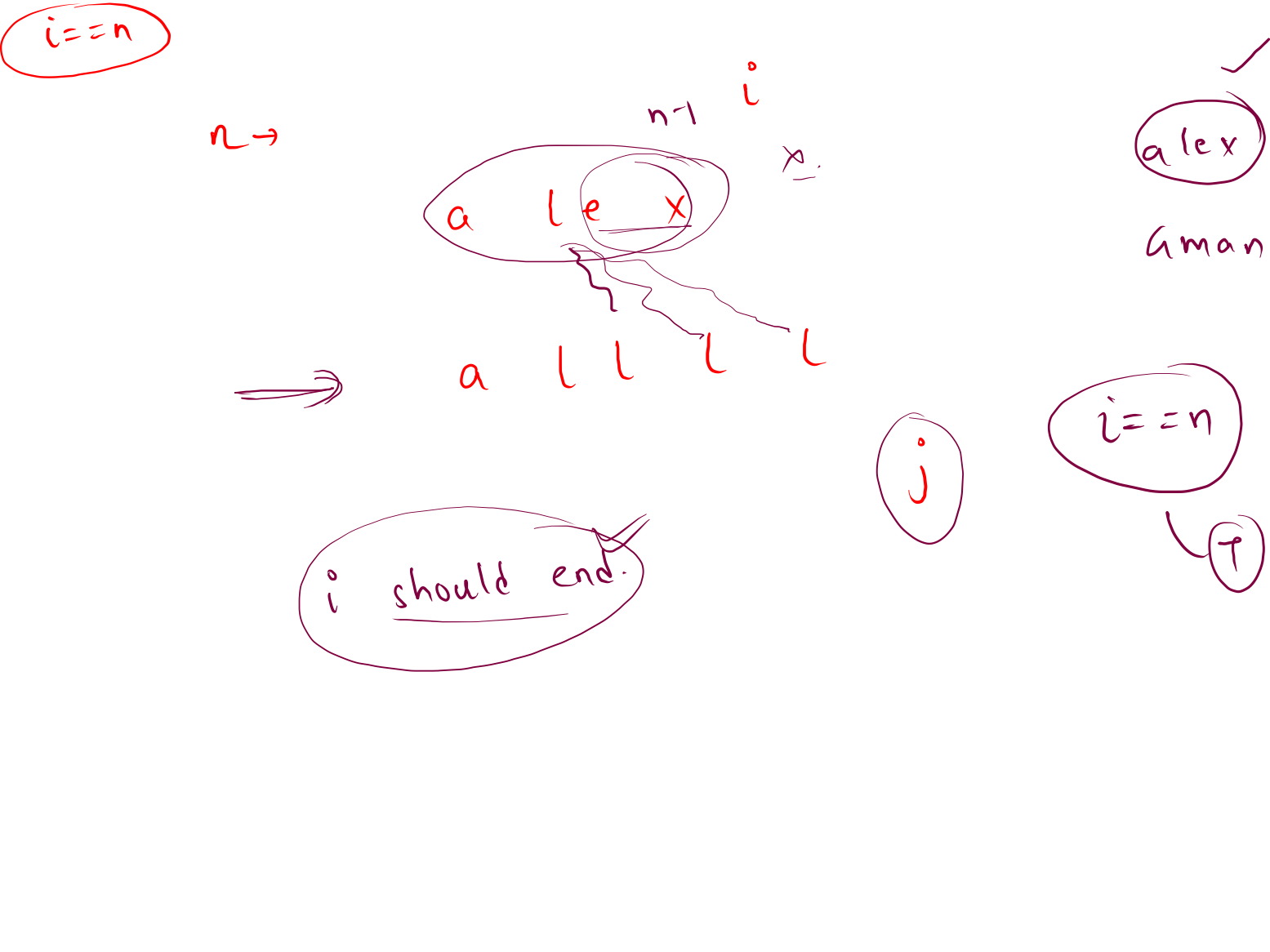
n →

t → alll L

false.

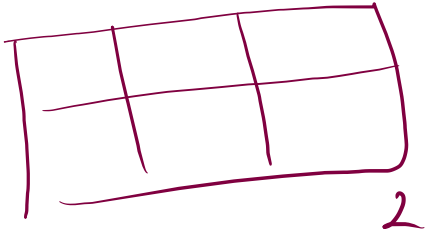
j

j ended i remaining.


i==n
n→
n-1
i
x
a l e x
a l l l l
alex
Aman
i==n
T
j
i should end.

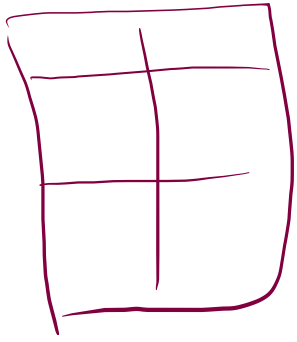
Matrix Multiplication.

A



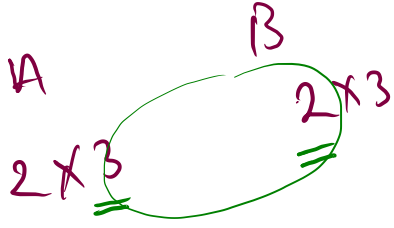
2 x 3

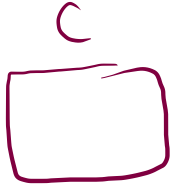
B



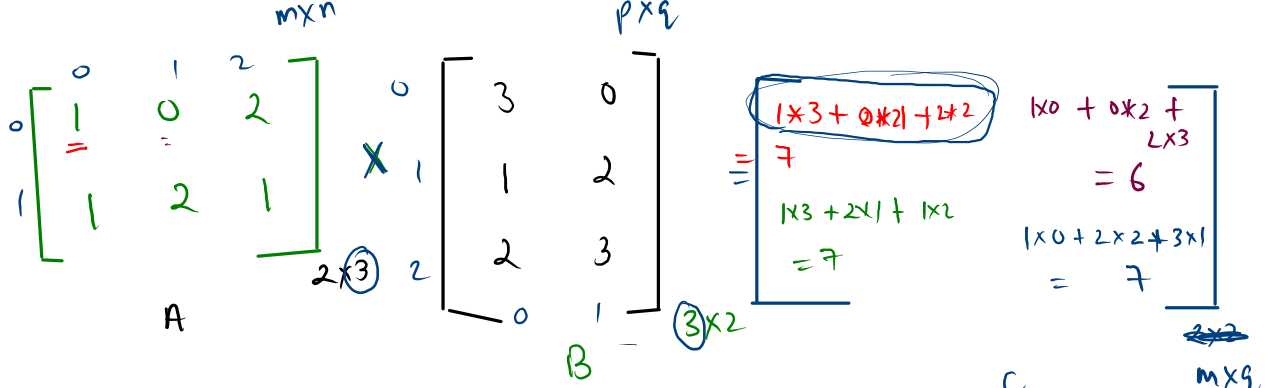
3 x 2

C = 2 x 2

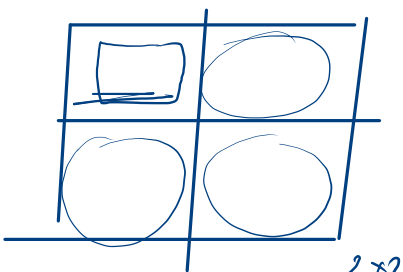




2 * 3 ✓




```
for(int i = 0; i < m; i++){
    for(int j = 0; j < q; j++){
        for(int k = 0; k < n; k++){      // k < p
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```



i=0
j=0

k=0 1 2

k<3

+ A[0][0] x B[0][0]
+ A[0][1] x B[1][0]
+ A[0][2] x B[2][0]