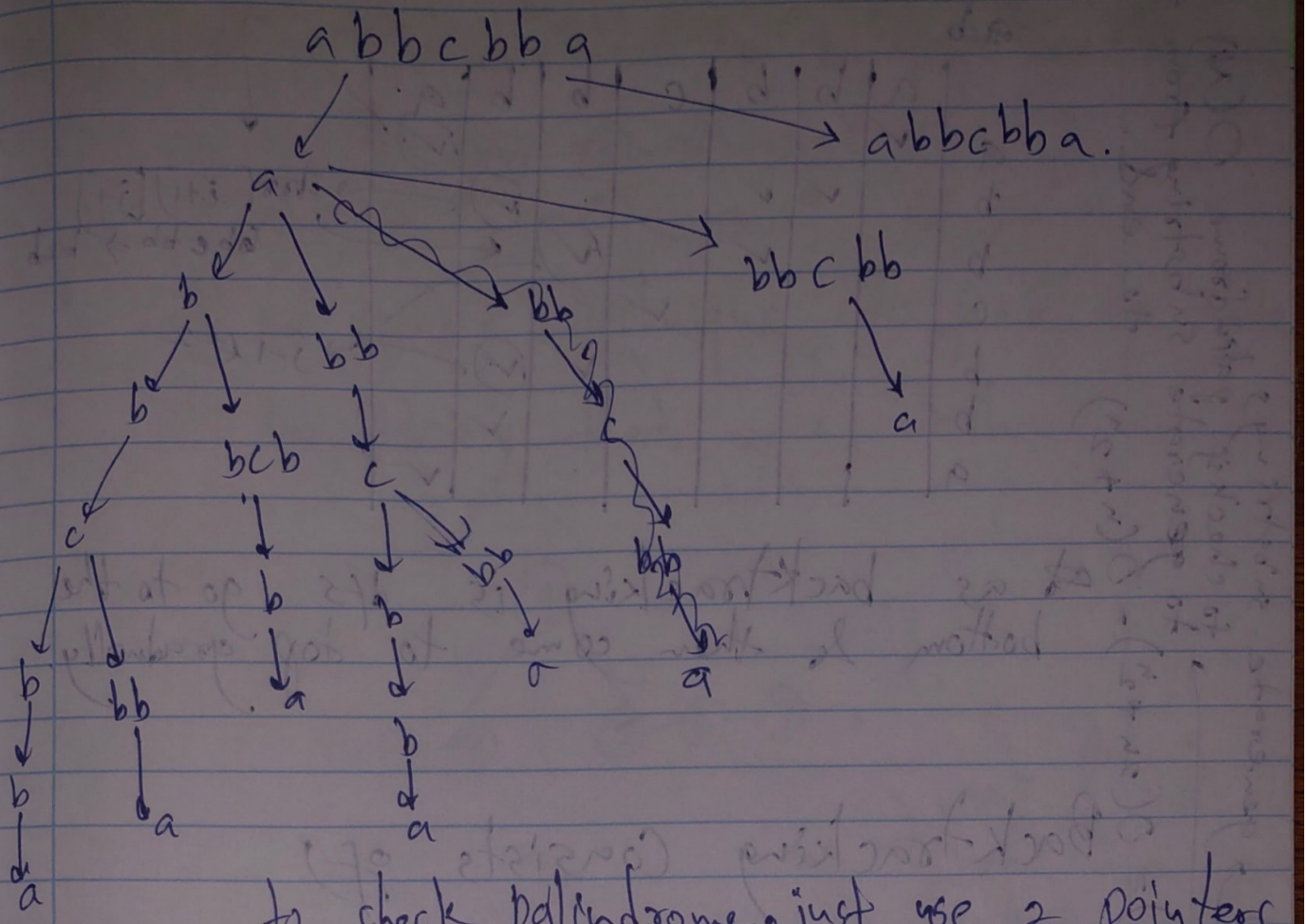
* Palindrome Partitioning

use backtracking. If taken substring is a palindrome, proceed with other letters. Else decline.



to check palindrome, just use 2 pointers in the substring & check.

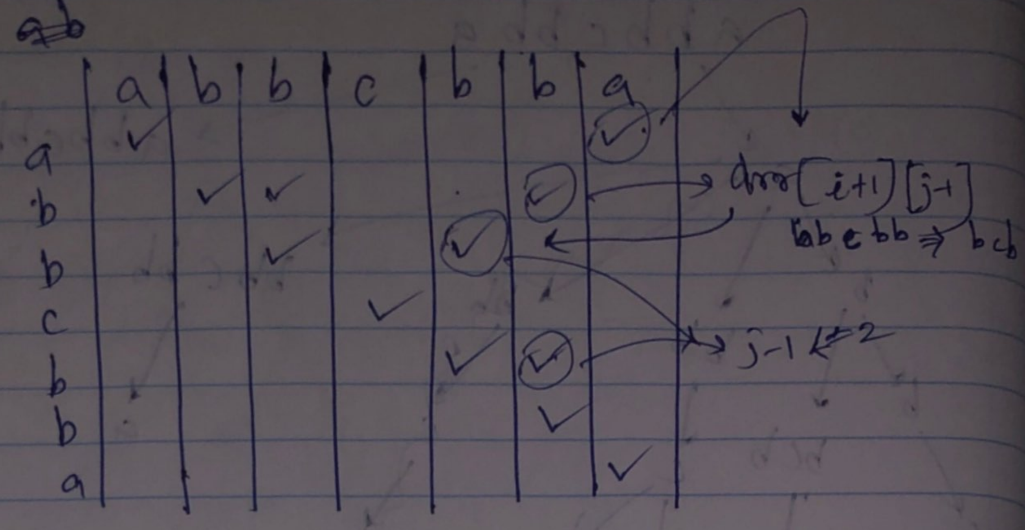
To optimize & avoid checking for palindrome in same string, can use dp.

bbcbb ⇒ bcb

if this true, then bbcbb should be true.

s.charAt(i) = s.charAt(j) ⇒ then

(j-i) <= 2 / dp[i+1][j-1] = true



as backtracking is dfs go to the bottom & then come to top gradually.

Backtracking consists of

- Choose → choose partition candidate (all substrings that could be generated from given string)
- Constraint → must be satisfied by choosen candidate (string must be palindrome)
- Goal → which determines if have found the required solution. & then we have to backtrack. (if we reached the end of string.)

time → $O(N * 2^n)$ → generate substrings to check if palindrome $O(N)$ to generate substring from start to end

$O((n+n) * 2^n) = O(2n * 2^n) = O(n * 2^n)$

Space → $O(N * N)$ in matrics.

in 1st approach $O(N)$

backtracking
- choose
- explore
- unchoose