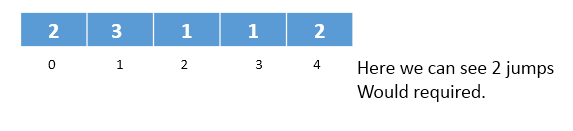
# Array Jumping

## Problem

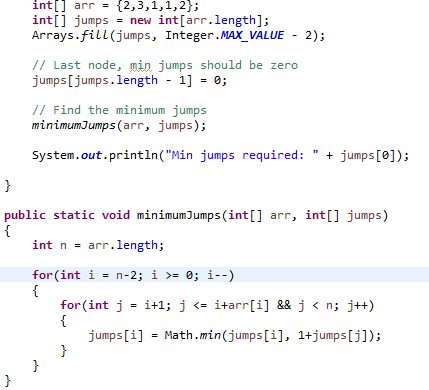
There is an array with values represents number of maximum positions from there can jump. We need to identify the minimum jumps we can make to reach last node in optimized way.



## Dynamic programming

Find the minimum jumps required to reach last node, and start from last node itself, so any before node the minimum jumps required should be

Jump[i] = From (I -> pos to -> i+arr[i]) Min(arr[i], 1+jumps[j])



### Time complexity with dynamic programming

Here the worst case time complexity is **O (n2)**

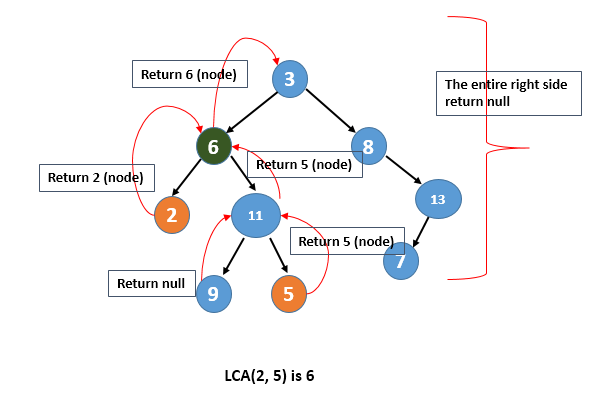
### Segment tree

Here the inner for loop does is just to find the minimum jumps from i+1 to last node. Which can be solved in Log n time using segment tree.

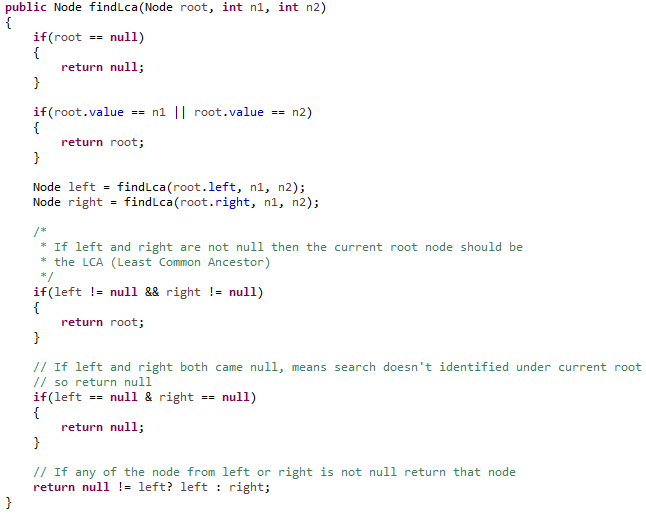
Hence the total time complexity will be **O (n Log n)**

# Least Common Ancestor

Here to find out the least common node between two pair of nodes.

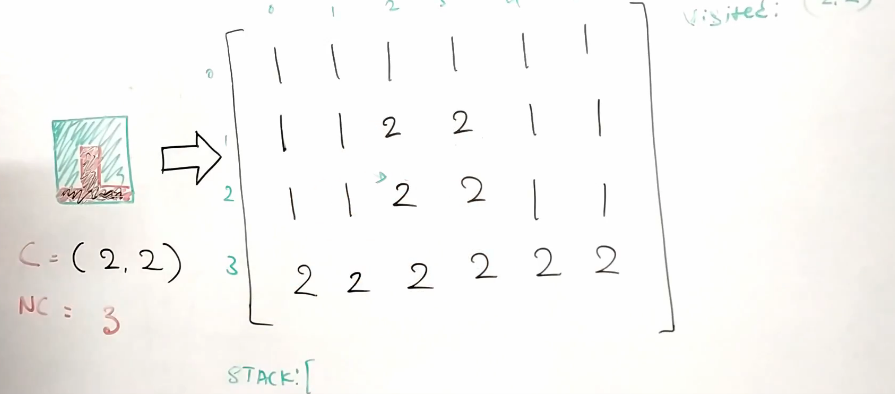


### Program



# Interview

## Flood Fill Algorithm



If someone selects values (3,1) and whose value is **2** and wants to fill with new color(value) i.e. **3** then the adjacent structure should fill with that color i.e. value **3**

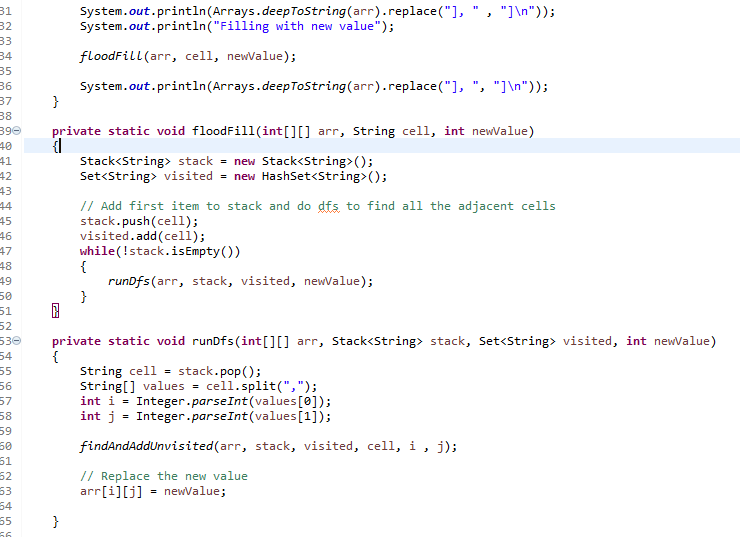
### Approach

Approach is very simple for this problem. We need to run **DFS** (with Stack) with helper of **visited** **array** to avoid multiple duplicate entries to stack.

Stack will contain all the cells whose value is the **old value** and **un-visited.**

**Pop** element from stack look for adjacent un-visited node and once done re-color it to new value **3.**

### Code



# Dynamic Programming

## Longest Common Subsequence

Given two arrays (strings) LCS is the longest common characters in between them.

LCS for input Sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4.

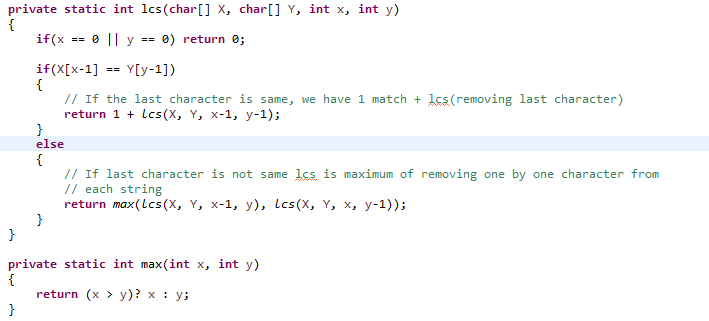
### Approach of solving

Start from last index

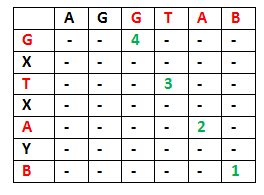
* If last index is matching we have **1** match + remove the last character from both arrays and do the match again
* If the last index doesn’t match LCS is MAX(lcs removing last character from 1st array, lcs removing last character from 2nd array)

### Recursive approach

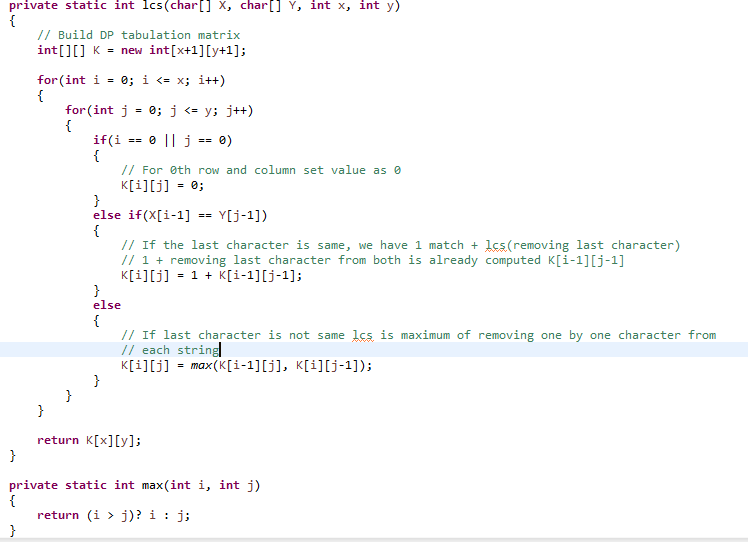
Time complexity is exponential **O (2n)**



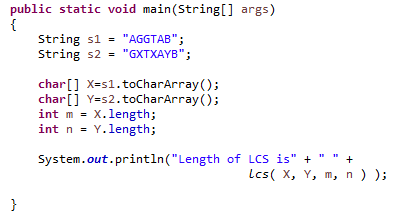
### Dynamic programming approach



**Program:**



**To test:**



With dynamic programming, the time complexity reduced to **O(mn)**