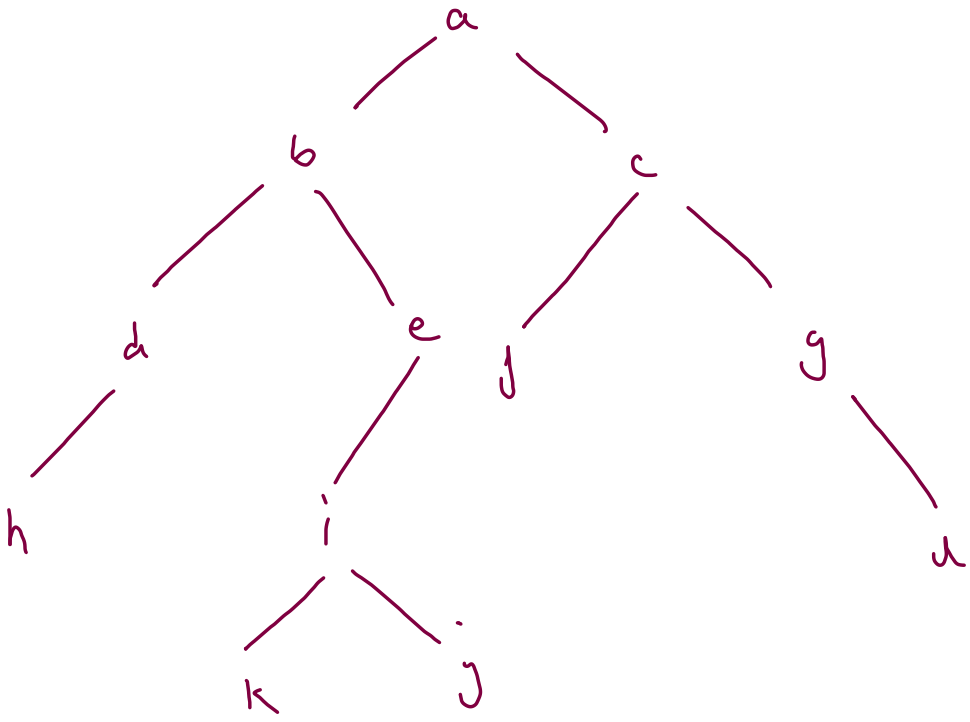
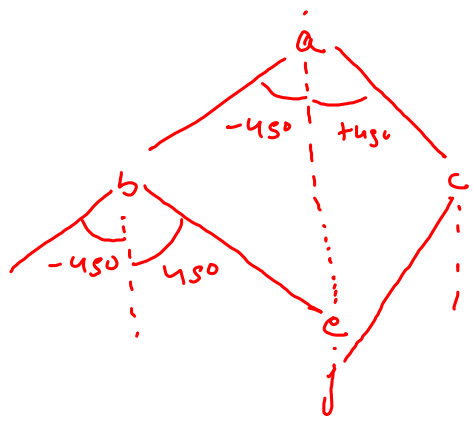
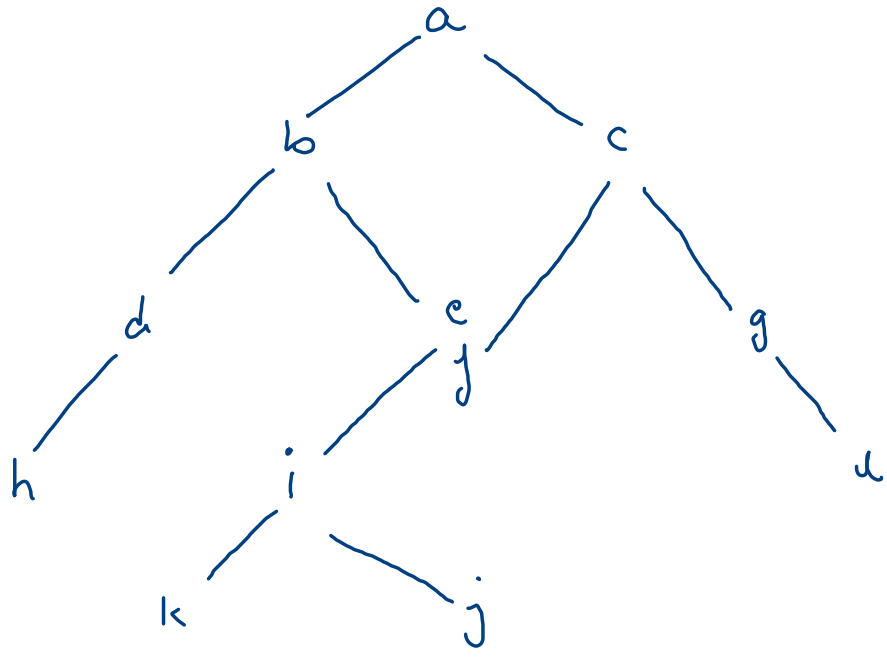
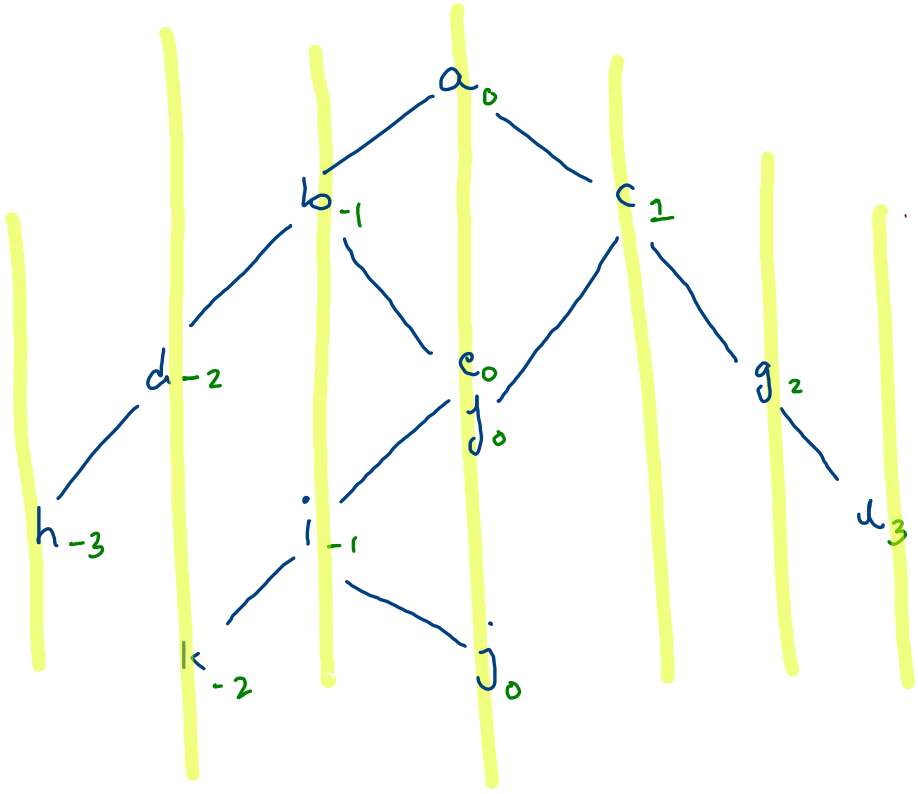

a
b
c
d
e
f
g
h
i
k
j


a
-45°
+45°
b
c
-45°
45°
e
f


a
b
c
d
e
f
g
h
i
k
j




pos (hd)

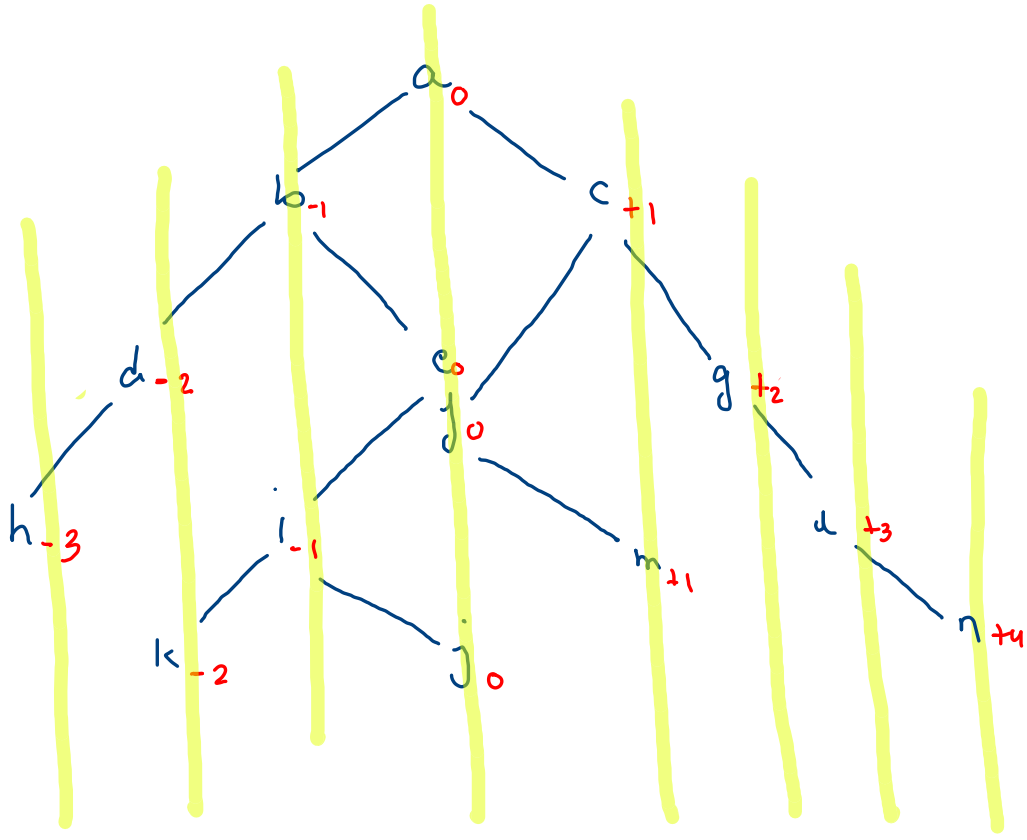
left child (hd-1)

right child (hd+1)

minhd = -3

maxhd = 3

width = (max hd - minhd + 1)



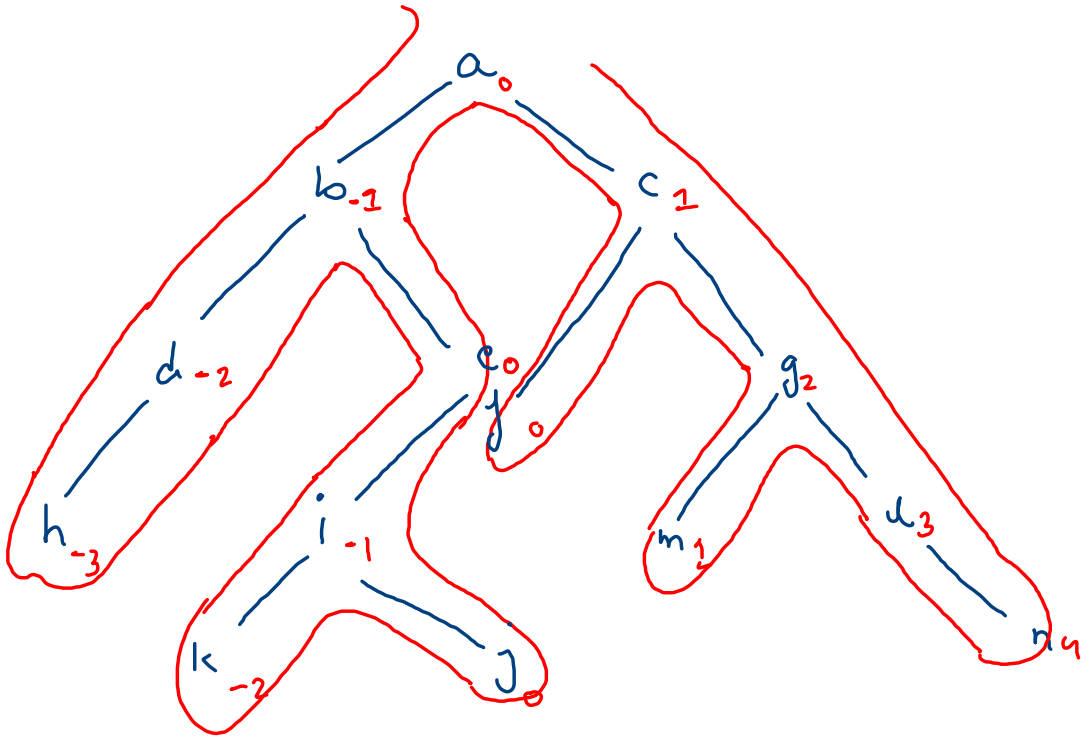

minhl = -3

maxhl = +4

width = (maxhl - minhl + 1)

= 4 - (-3) + 1

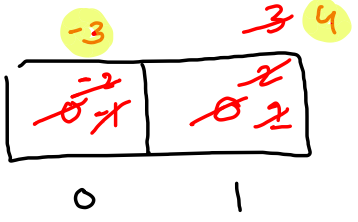
= 8




```
public static void width_helper(TreeNode root,int hl,int[]minmaxhl) {
    if(root == null) {
        return;
    }

    minmaxhl[0] = Math.min(minmaxhl[0],hl); //minhl
    minmaxhl[1] = Math.max(minmaxhl[1],hl) ; //maxhl

    width_helper(root.left,hl-1,minmaxhl);
    width_helper(root.right,hl+1,minmaxhl);
}

public static int width(TreeNode root) {
   int[]minmaxhl = new int[2];
   width_helper(root,0,minmaxhl); //min max horizontal level

   int minhl = minmaxhl[0];
   int maxhl = minmaxhl[1];
   int w = maxhl - minhl + 1;

   return w;
}
```

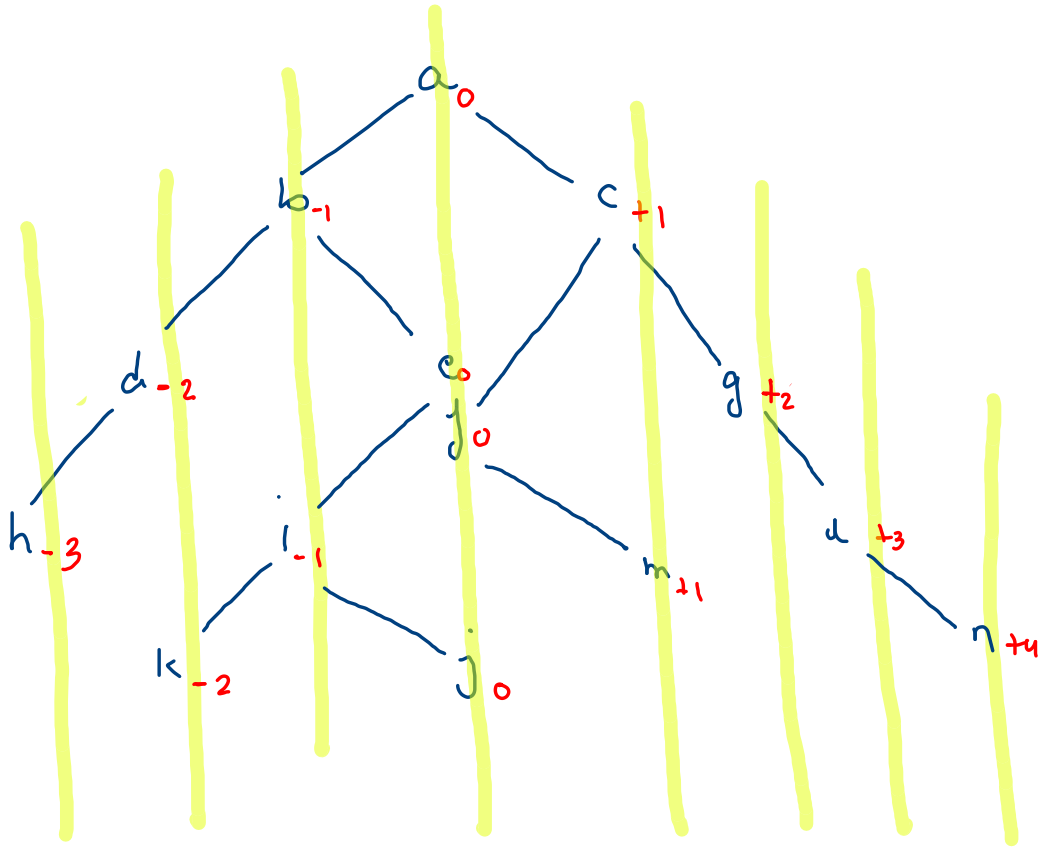
w : 4 - (-3) + 1

= 8




0 -> minhl

1 -> maxhl




-3 -> h

-2 -> d, k

-1 -> b, i

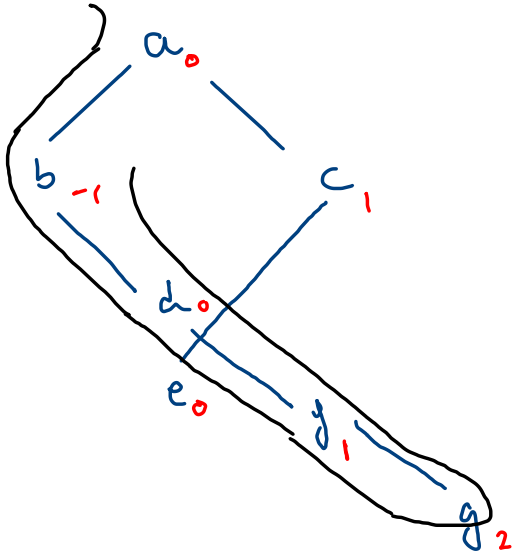
0 -> a, e, f, j

1 -> c, m

+2 -> g

+3 -> d

+4 -> n

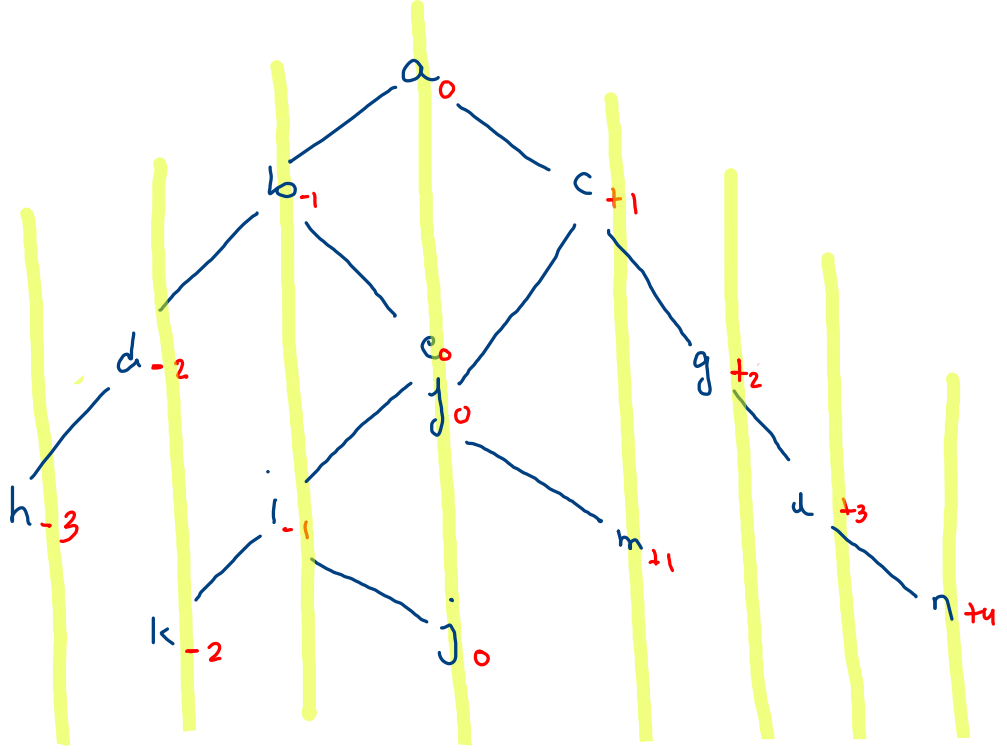



0 -> a, d

-1 -> b

1 -> f, c ✗

2 -> g




Pair {

node ;

hd ;

}

horz. dict → list

integer, AL

0 → a, e, f, j

-1 → b, i

+1 → c, m

-2 → d, k

2 → g

-3 → h

3 → d

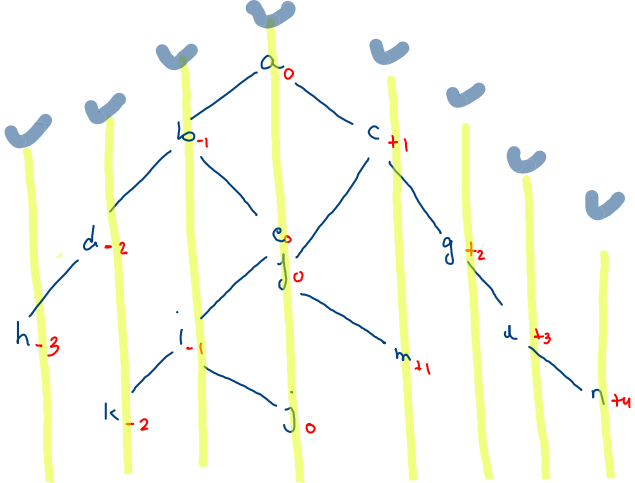
4 → n

minhd = -3

maxhd = 4

| a,0 | b,-1 | c,+1 | d,-2 | e,0 | f,0 | g,2 | h,-3 | i,-1 | m,1 | d,3 | k,-2 | j,0 | n,4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

remove, work, ac




```
while(q.size() > 0) {
    //remove
    Pair rem = q.remove();
    minhl = Math.min(minhl,rem.hl);
    maxhl = Math.max(maxhl,rem.hl);

    //work
    if(map.containsKey(rem.hl) == true) {
        ArrayList<Integer>list = map.get(rem.hl);
        list.add(rem.node.val);
    }
    else {
        ArrayList<Integer>list = new ArrayList<>();
        list.add(rem.node.val);
        map.put(rem.hl,list);
    }

    //add children
    if(rem.node.left != null) {
        q.add(new Pair(rem.node.left,rem.hl-1));
    }
    if(rem.node.right != null) {
        q.add(new Pair(rem.node.right,rem.hl+1));
    }
}

ArrayList<ArrayList<Integer>>ans = new ArrayList<>();
for(int hl = minhl; hl <= maxhl;hl++) {
    ans.add(map.get(hl));
}

return ans;
```

minhl = -3

maxhl = 4

Int , AL < Int >

0 -> a, e, f, j

-1 -> b, i

1 -> c, m

-2 -> d, k

2 -> g

-3 -> h

3 -> d

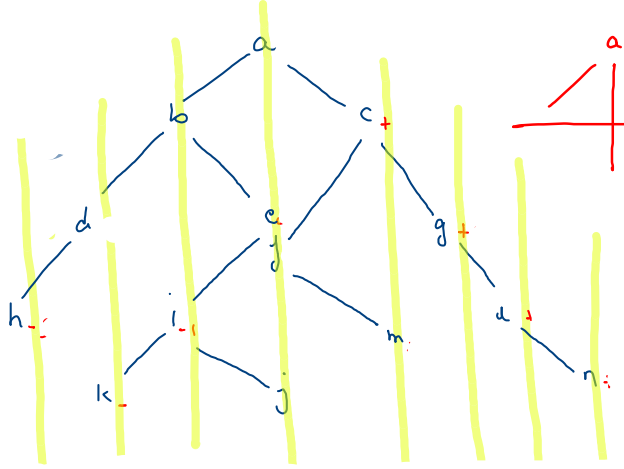
4 -> n

| a,0 | b,-1 | c,1 | d,-2 | e,0 | f,0 | g,2 | h,-3 | i,-1 | m,1 | d,3 | k,-2 | j,0 | n,4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

hl -> -3 to 4

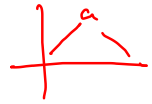
[ [h], [d,k], [b,i], [a,e,f,j], [c,m], [g], [l], [n] ]

-3 -2 -1 0 1 2 3 4










lw -> dw + (- min hd)

[ [h] [d,k] [b,i] [a, c, f, j] [c,m] [g] [l] n ]

0 1 2 3 4 5 6 7

(-3) (-2) (-1) (0) (2) (2) (3) (4)

0 1 2 3 4 5 6 7

-3 -> 0

-2 -> 1

-1 -> 2

0 -> 3

1 -> 4

2 -> 5

3 -> 6

4 -> 7

~~g,3 | b,2 | c,4 | d,2 | e,3 | f,3 | g,5 | h,0 | i,2 | m,4 | n,6 | k,1 | j,3 | n,4~~

1. Given a Binary Tree, print Vertical Order of it.
2. For each node at position (row, col), its left and right children will be at positions (row + 1, col - 1) and (row + 1, col + 1) respectively. The root of the tree is at (0, 0).
3. The vertical order traversal of a binary tree is a list of top-to-bottom orderings for each column index starting from the leftmost column and ending on the rightmost column. There may be multiple nodes in the same row and same column. In such a case, sort these nodes by their values.
4. For More Information Watch Question Video link below.

odd

-2 -> 16

-1 -> 9 14

0 -> 5 7 3 4

1 -> 14

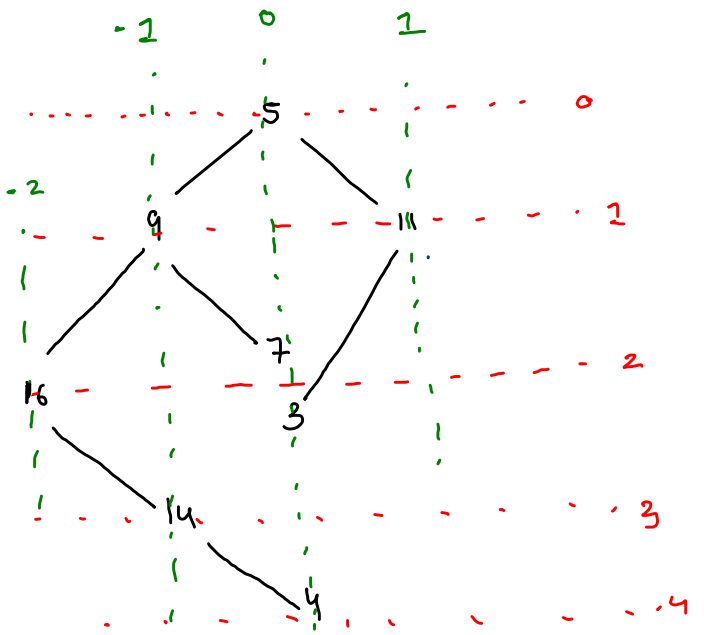
new

-2 -> 16

-1 -> 9 14

0 -> 5 3 7 4

1 -> 14




queue LO

Pair {

node,

x,

y

}

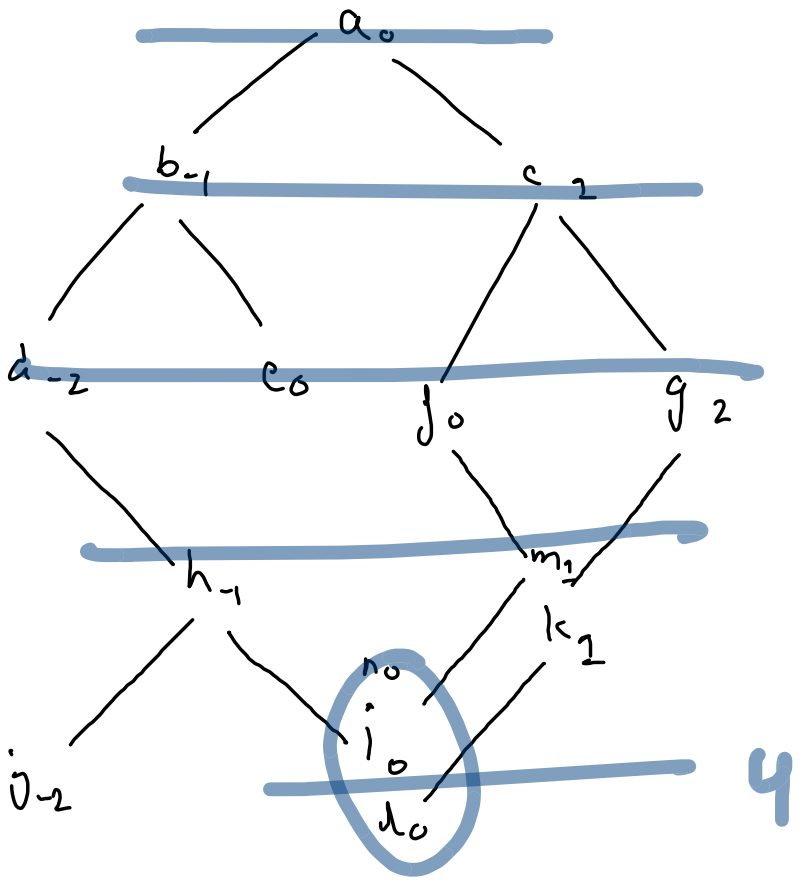
y 1st priority

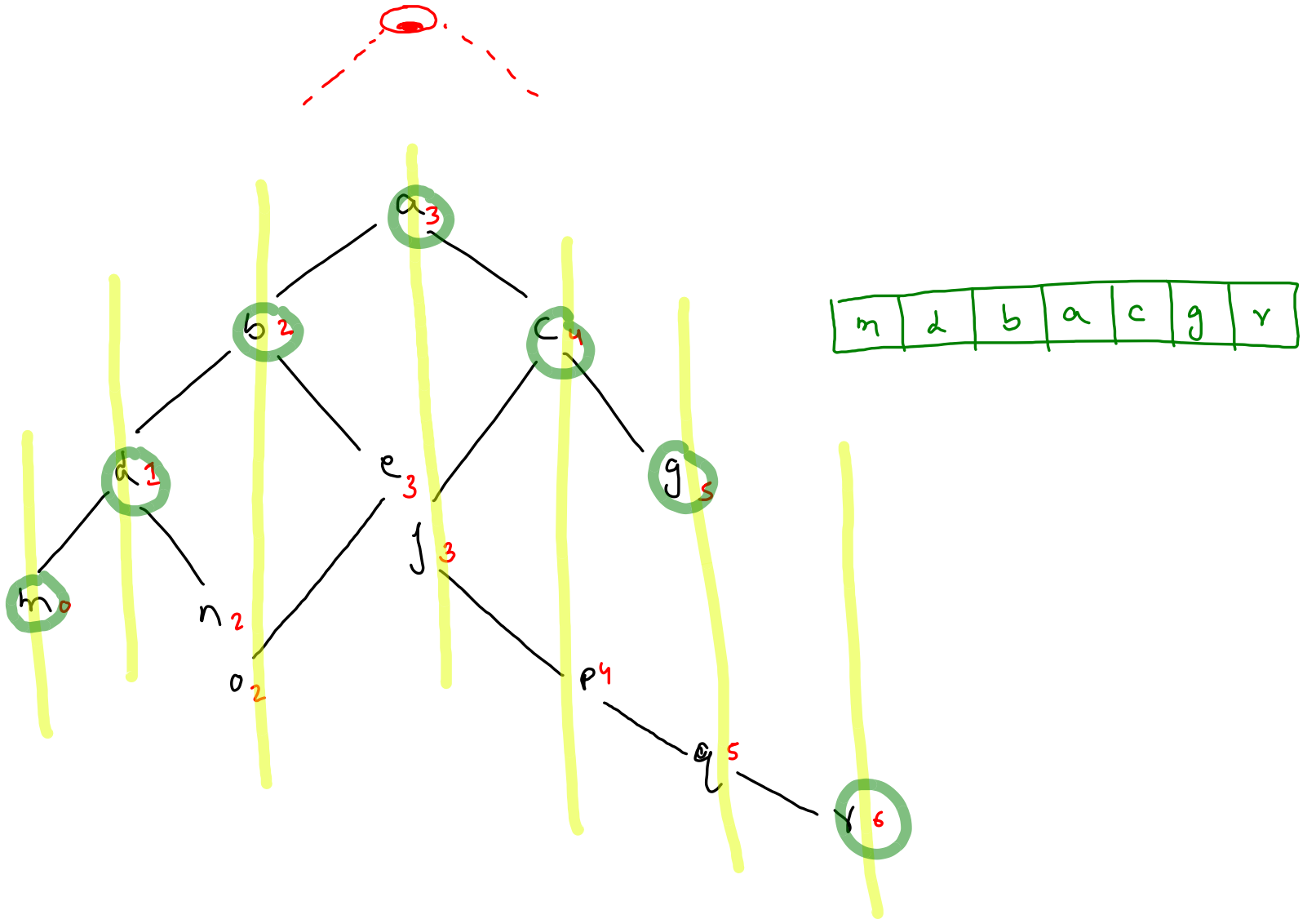
|| y same

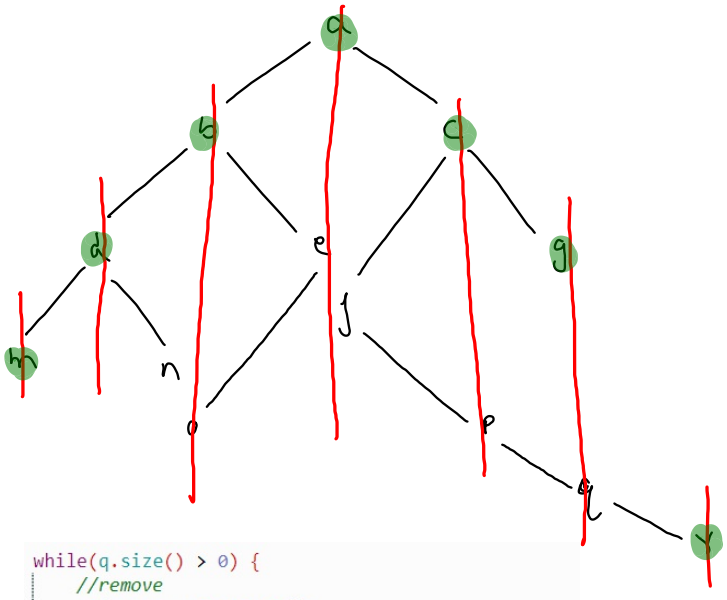
x 2nd priority

|| x same

[ node.val

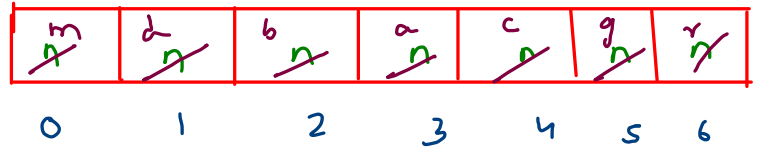

$a_0$
$b_{-1}$
$c_1$
$d_{-2}$
$e_0$
$f_0$
$g_2$
$h_{-1}$
$m_1$
$k_1$
$n_0$
$i_0$
$l_0$
$v_{-2}$
4


a 3
b 2
c 4
d 1
e 3
g 5
f 3
m 0
n 2
o 2
p 4
q 5
r 6
m d b a c g r

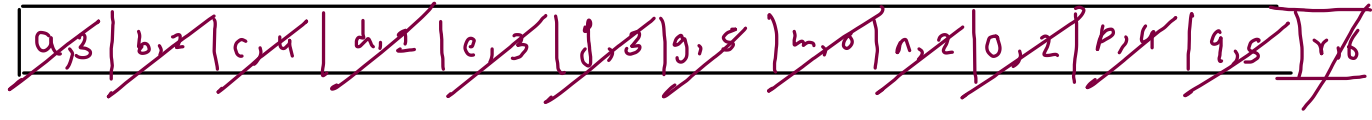


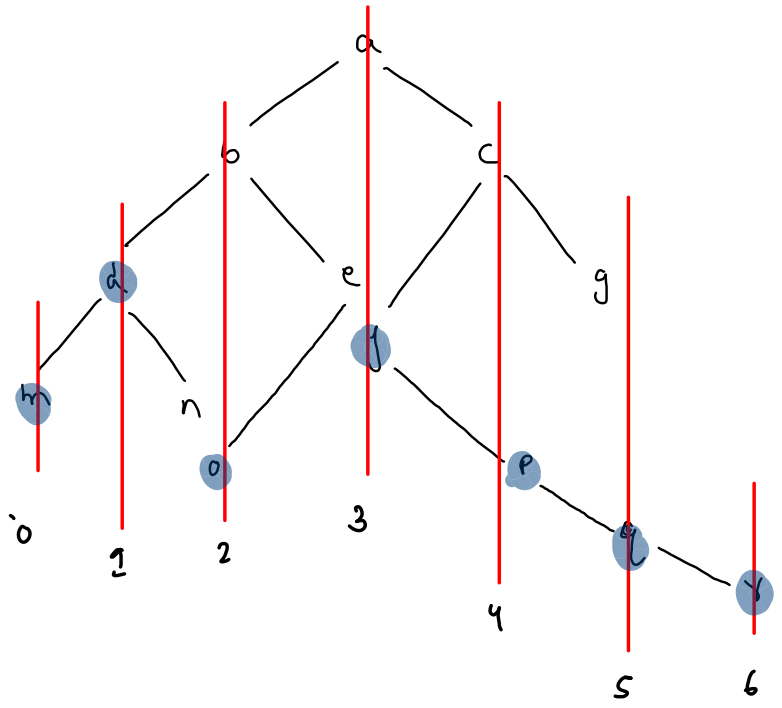

```
for(int i=0; i < w;i++) {
    ans.add(null);
}
```

w = 3 - (-3) + 1 = 7

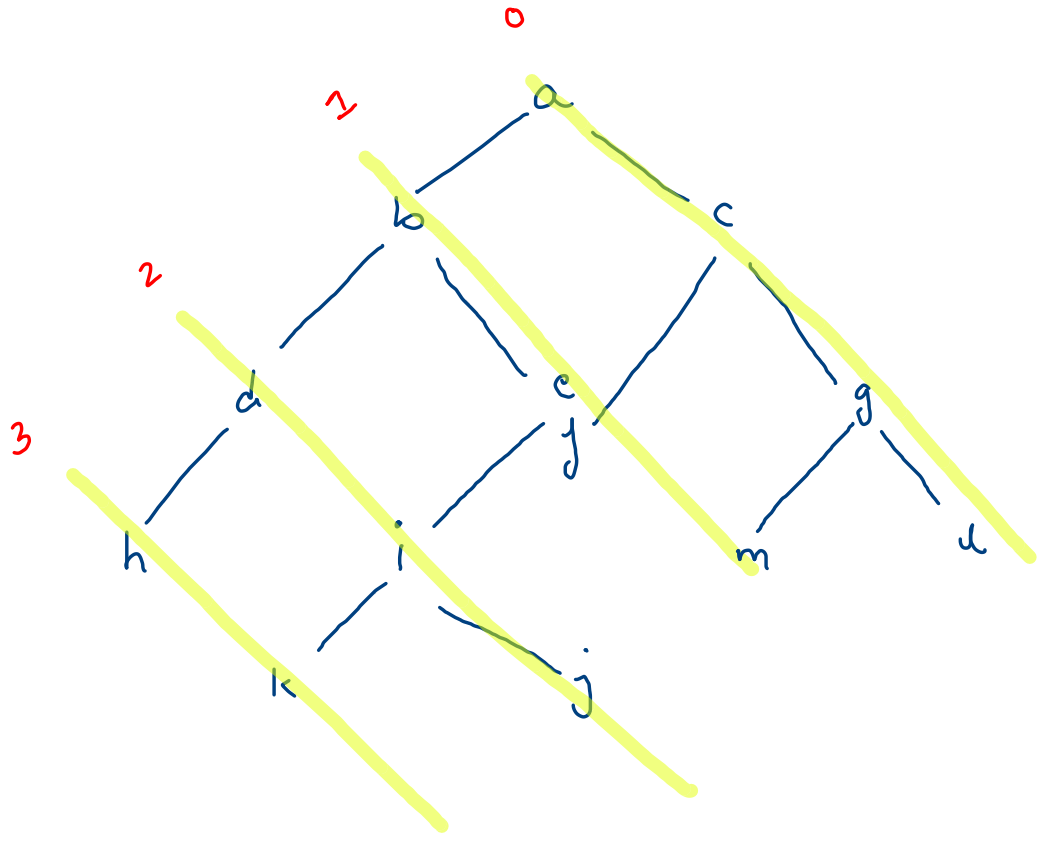



```
while(q.size() > 0) {
    //remove
    Pair rem = q.remove();

    //work
    if(ans.get(rem.hl) == null) {
        ans.set(rem.hl,rem.node.val);
    }

    //add children
    if(rem.node.left != null) {
        q.add(new Pair(rem.node.left,rem.hl-1));
    }
    if(rem.node.right != null) {
        q.add(new Pair(rem.node.right,rem.hl+1));
    }
}

return ans;
```






m d o f p q r




0 -> a c g l

1 -> b e f m

2 -> d i j

3 -> h k

dl

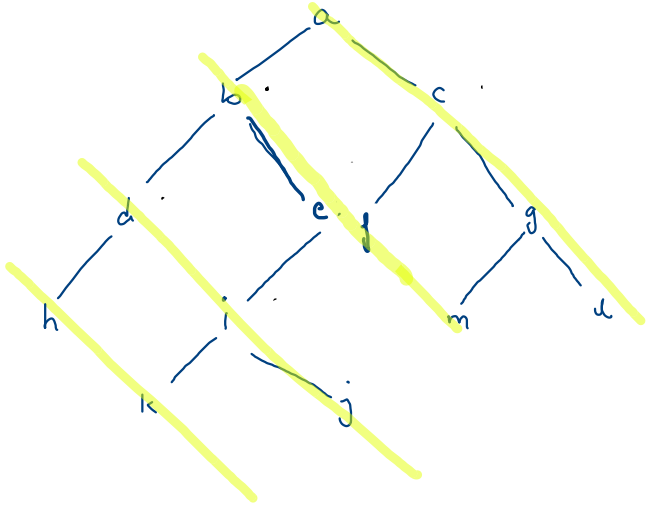
par.dl

left child

par.dl-1

right dl

par.dl




0 -> a c g l

1 -> b e f m

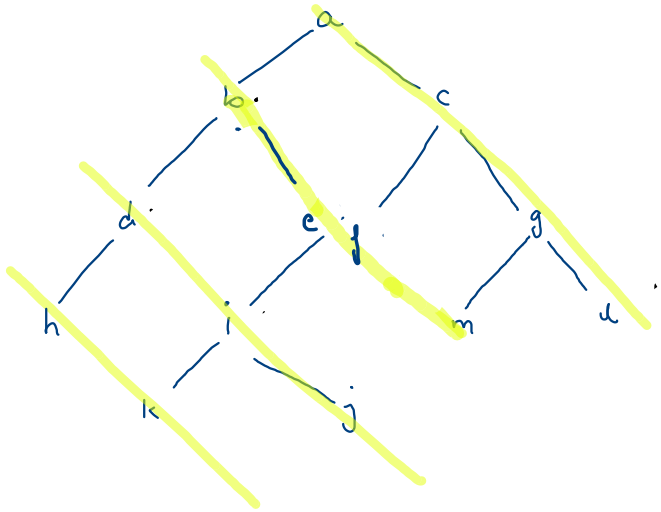
2 -> d i j

3 -> h k

component -> node,

till extrem
right

| a | b | j | h | d | i | b | k |
|---|---|---|---|---|---|---|---|




ans. [a c g l]

[b e f m]

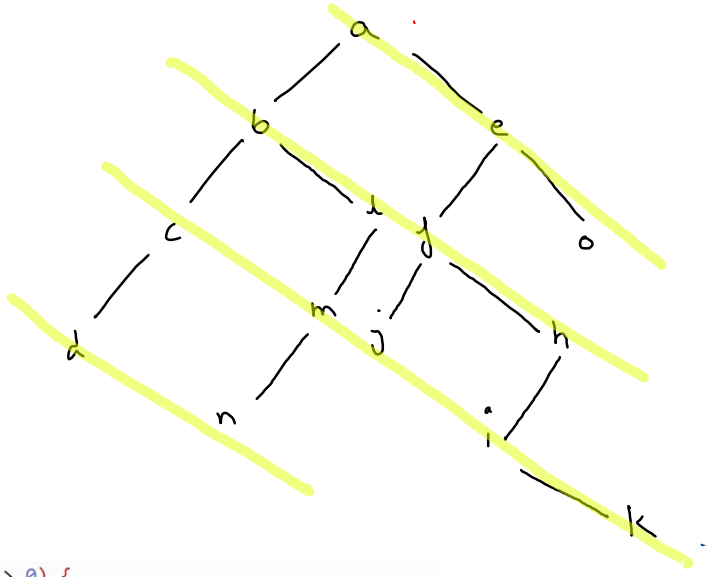
[d i j]

[h k]

diag = h k

```
while(q.size() > 0) {
    int count = q.size();

    //all components to make dth diagonal
    ArrayList<Integer>diag = new ArrayList<>();

    while(count-- > 0) {
        TreeNode rem = q.remove();

        //to travel a single component
        while(rem != null) {
            diag.add(rem.val);

            if(rem.left != null) {
                //to add components of next diagonal
                q.add(rem.left);
            }
            rem = rem.right;
        }
    }

    ans.add(diag);
}
```

| a | b | f | m | d | i | h | k |
|---|---|---|---|---|---|---|---|




ans. -> [a, e, o]

[b, i, j, h]

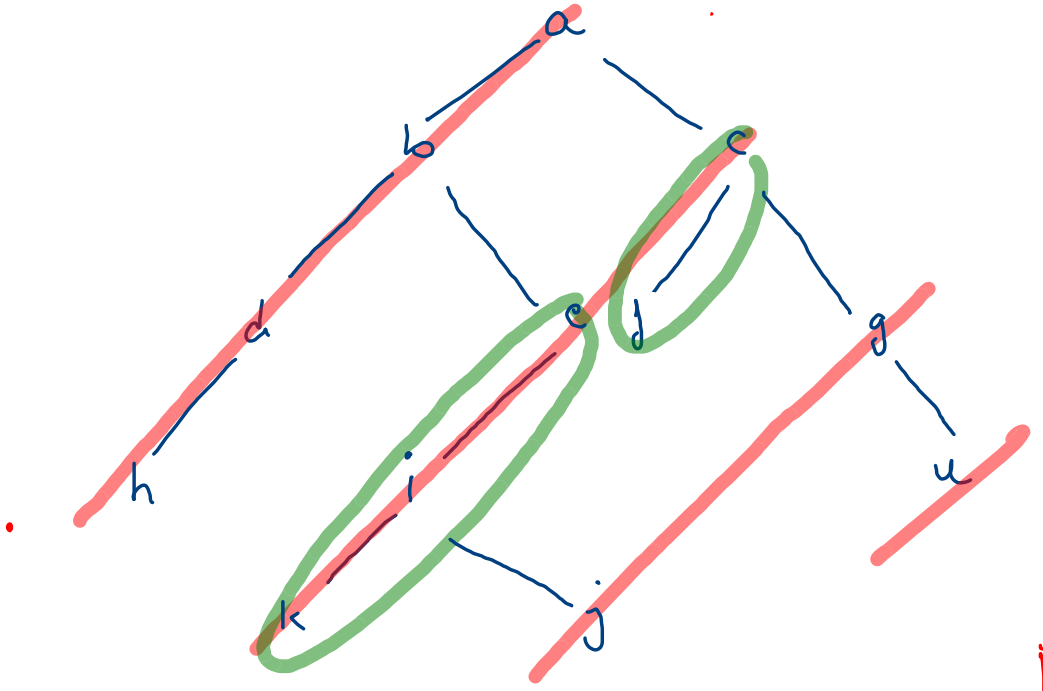
[c, m, j, i, k]

[d, n]

diag :

```
while(q.size() > 0) {
    int count = q.size();

    //all components to make dth diagonal
    ArrayList<Integer>diag = new ArrayList<>();

    while(count-- > 0) {
        TreeNode rem = q.remove();

        //to travel a single component
        while(rem != null) {
            diag.add(rem.val);

            if(rem.left != null) {
                //to add components of next diagonal
                q.add(rem.left);
            }
            rem = rem.right;
        }
    }

    ans.add(diag);
}
```

| a | b | j | c | m | j | i | d | n |
|---|---|---|---|---|---|---|---|---|




diag : a b d h