![Graphical user interface, application, Teams

Description automatically generated]()![Graphical user interface

Description automatically generated]()

|  |
| --- |
| /\* |
|  |  |
|  | Idea : To make a HashMap mapping the Col as Key and the Nodes in each column, prioritized as row, value per column. |
|  | So take a Priority Queue for this purpose and for each col, store the Nodes |
|  |  |
|  | The Nodes ordered by Row First; if in same row, then column. |
|  |  |
|  |  |
|  | Time Complexity : O(N LogN) |
|  | Space Complexity : O(N) |
|  | \*/ |
|  |  |
|  |  |
|  |  |
|  | class Solution { |
|  | private Map<Integer, Queue<int[]>> nodeEntries = new HashMap<>(); |
|  | public List<List<Integer>> verticalTraversal(TreeNode root) { |
|  | dfs(root, 0, 0);// root,row,col |
|  |  |
|  | int minCol = Collections.min(nodeEntries.keySet()); |
|  | int maxCol = Collections.max(nodeEntries.keySet()); |
|  |  |
|  | List<List<Integer>> output = new ArrayList<>(); |
|  |  |
|  | for(int col = minCol ; col <= maxCol; col++){ |
|  | output.add(new ArrayList<>()); |
|  |  |
|  | while(!nodeEntries.get(col).isEmpty()) { |
|  | int[] entry = nodeEntries.get(col).remove(); |
|  |  |
|  | output.get(output.size() - 1).add(entry[1]); |
|  | } |
|  | } |
|  |  |
|  | return output; |
|  | } |
|  |  |
|  | private void dfs(TreeNode root, int col, int row) { |
|  | if(root != null) { |
|  | nodeEntries.putIfAbsent(col , new PriorityQueue<>((e1, e2) -> { |
|  | // 0 - row. 1 - val |
|  |  |
|  | if(e1[0] != e2[0]) return e1[0] - e2[0]; |
|  | return e1[1] - e2[1]; |
|  | })); |
|  |  |
|  | nodeEntries.get(col).add(new int[]{row, root.val}); |
|  | dfs(root.left, col-1, row+1); |
|  | dfs(root.right, col+1, row+1); |
|  | } |
|  | } |
|  | } |