|  |  |
| --- | --- |
| **BottomView in C++** | |
| #include <iostream>  #include <map>  #include <queue>  #include <vector>  using namespace std;  // Definition for a binary tree node.  struct TreeNode {  int val;  TreeNode \*left;  TreeNode \*right;  TreeNode(int x) {  val = x;  left = nullptr;  right = nullptr;  }  };  vector<int> bottomView(TreeNode\* root) {  vector<int> bottomViewNodes;  if (!root) {  return bottomViewNodes;  }  // TreeMap equivalent in C++ is std::map  map<int, int> map;  queue<pair<TreeNode\*, int>> q;  q.push({root, 0});  while (!q.empty()) {  auto front = q.front();  q.pop();  TreeNode\* node = front.first;  int hd = front.second;  // Update the map with current node's value at its horizontal distance  map[hd] = node->val;  // Enqueue left child with horizontal distance hd - 1  if (node->left) {  q.push({node->left, hd - 1});  }  // Enqueue right child with horizontal distance hd + 1  if (node->right) {  q.push({node->right, hd + 1});  }  }  // Populate bottomViewNodes with values from map  for (const auto& pair : map) {  bottomViewNodes.push\_back(pair.second);  }  return bottomViewNodes;  }  // Utility function to create a new node  TreeNode\* newNode(int key) {  TreeNode\* node = new TreeNode(key);  return node;  }  int main() {  TreeNode\* root = newNode(1);  root->left = newNode(2);  root->right = newNode(3);  root->left->left = newNode(4);  root->left->right = newNode(5);  root->right->left = newNode(6);  root->right->right = newNode(7);  vector<int> result = bottomView(root);  // Print the result  for (int value : result) {  cout << value << " ";  }  cout << endl;  // Memory cleanup (optional in this example)  // You may need to delete nodes if not using smart pointers  return 0;  } | Binary Tree Structure: 1  / \  2 3  / \ / \  4 5 6 7 📘 Step-by-Step Dry Run Table We'll simulate the level order traversal using a queue storing (node, horizontal\_distance) and map hd → node->val.   | **Step** | **Queue Content** | **Popped Node** | **HD** | **Map After Step** | | --- | --- | --- | --- | --- | | 1 | (1, 0) | 1 | 0 | {0 → 1} | | 2 | (2, -1), (3, 1) | 2 | -1 | {-1 → 2, 0 → 1} | | 3 | (3, 1), (4, -2), (5, 0) | 3 | 1 | {-1 → 2, 0 → 1, 1 → 3} | | 4 | (4, -2), (5, 0), (6, 0), (7, 2) | 4 | -2 | {-2 → 4, -1 → 2, 0 → 1, 1 → 3} | | 5 | (5, 0), (6, 0), (7, 2) | 5 | 0 | {-2 → 4, -1 → 2, 0 → 5, 1 → 3} | | 6 | (6, 0), (7, 2) | 6 | 0 | {-2 → 4, -1 → 2, 0 → 6, 1 → 3} | | 7 | (7, 2) | 7 | 2 | {-2 → 4, -1 → 2, 0 → 6, 1 → 3, 2 → 7} |  🟢 Final Bottom View: Take values from the map in order of keys (i.e., horizontal distance):  -2 → 4  -1 → 2  0 → 6  1 → 3  2 → 7  🔽 **Output:**  4 2 6 3 7 |
| 4 2 6 3 7 | |