|  |  |
| --- | --- |
| **Node at distance K in C++** | |
| #include <iostream>  #include <queue>  using namespace std;  // Definition of a binary tree node  struct Node {  int data;  Node\* left;  Node\* right;  Node(int item) {  data = item;  left = nullptr;  right = nullptr;  }  };  // Function declaration  void printNodesDown(Node\* root, int k);  // Function to print nodes at distance k from the given node  int nodesAtDistanceKWithRootDistance(Node\* root, int node, int k) {  if (root == nullptr) {  return -1;  }  // If the current node is the target node, print nodes at distance k from it  if (root->data == node) {  printNodesDown(root, k);  return 0;  }  // Recursively search in left subtree  int leftHeight = nodesAtDistanceKWithRootDistance(root->left, node, k);  if (leftHeight != -1) {  // If the target node is found in the left subtree  if (leftHeight + 1 == k) {  cout << root->data << endl;  } else {  // Print nodes at distance k from the right subtree  printNodesDown(root->right, k - leftHeight - 2);  }  return leftHeight + 1;  }  // Recursively search in right subtree  int rightHeight = nodesAtDistanceKWithRootDistance(root->right, node, k);  if (rightHeight != -1) {  // If the target node is found in the right subtree  if (rightHeight + 1 == k) {  cout << root->data << endl;  } else {  // Print nodes at distance k from the left subtree  printNodesDown(root->left, k - rightHeight - 2);  }  return rightHeight + 1;  }  // If the target node is not found in either subtree  return -1;  }  // Function to print nodes at distance k from a given node downwards  void printNodesDown(Node\* root, int k) {  if (root == nullptr || k < 0) {  return;  }  // If reached the required distance, print the node  if (k == 0) {  cout << root->data << endl;  return;  }  // Recursively print nodes at distance k in both subtrees  printNodesDown(root->left, k - 1);  printNodesDown(root->right, k - 1);  }  // Function to initiate printing nodes at distance k from a given node value  void nodesAtDistanceK(Node\* root, int node, int k) {  nodesAtDistanceKWithRootDistance(root, node, k);  }  int main() {  // Hardcoded tree construction  Node\* root = new Node(2);  root->left = new Node(3);  root->left->left = new Node(4);  root->left->right = new Node(8);  root->left->left->left = new Node(4);  root->right = new Node(9);  root->right->right = new Node(2);  root->right->right->left = new Node(6);  // Call function to print nodes at distance k from node with value 3  nodesAtDistanceK(root, 3, 2);  // Clean up dynamically allocated memory  delete root->right->right->left;  delete root->right->right;  delete root->right;  delete root->left->left->left;  delete root->left->left;  delete root->left->right;  delete root->left;  delete root;  return 0;  } | Binary Tree Structure: 2  / \  3 9  / \ \  4 8 2  / /  4 6 🧠 Objective: Print all nodes that are **exactly k=2 distance** away from node with value 3. 📝 Dry Run Table:  | **Step** | **Function Call** | **Current Node** | **Action** | **Output** | **Return Value** | | --- | --- | --- | --- | --- | --- | | 1 | nodesAtDistanceK(root=2, node=3, k=2) | 2 | Call nodesAtDistanceKWithRootDistance |  |  | | 2 | nodesAtDistanceKWithRootDistance(root=2, node=3, k=2) | 2 | Not target → search left and right |  |  | | 3 | nodesAtDistanceKWithRootDistance(root=3, node=3, k=2) | 3 | 🎯 Target found! Call printNodesDown(3, 2) |  | 0 | | 4 | printNodesDown(root=3, k=2) | 3 | Go down to distance 2 |  |  | | 5 | printNodesDown(root=4, k=1) | 4 | Recurse to left → node 4 |  |  | | 6 | printNodesDown(root=4, k=0) | 4 (leaf) | ✅ Distance 0 → print 4 | 4 |  | | 7 | printNodesDown(root=8, k=1) | 8 | No children |  |  | | 8 | Back to step 2, leftHeight = 0 |  | Check if root (2) is at k=2? No → Call printNodesDown(right, k-2) |  |  | | 9 | printNodesDown(root=9, k=0) | 9 | ✅ Distance 0 → print 9 | 9 |  | | 10 | All done |  | Final output = 4, 9 |  |  |  ✅ Final Output: 4  9 |
| 4  9 | |