**MaxDepth (Finds the Maximum depth)**

15

9 16

4

2 7

1 3 5 11

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MD(15)  L=MD(9) 3  R=MD(16) 0  4 | | | | | | | | | | |
| MD(9)  L=MD(4) 2  R=MD(NULL) -1  **3** | | | | | | | | | MD(16)  L=MD(NULL) -1  R=MD(NULL) -1  **0** | |
| MD(4)  L=MD(2) 1  R=MD(7) 1  2 | | | | | | | | MD(NULL)  return -1 | MD(NULL)  return -1 | MD(NULL)  return -1 |
| MD(2)  L=MD(1) 0  R=MD(3) 0  **1** | | | | MD(7)  L=MD(5) 0  R=MD(11) 0  **1** | | | |  | | |
| MD(1)  L=MD(NULL) -1  R=MD(NULL) -1  0 | | MD(3)  L=MD(NULL) -1  R=MD(NULL) -1  0 | | MD(5)  L=MD(NULL) -1  R=MD(NULL) -1  0 | | MD(11)  L=MD(NULL) -1  R=MD(NULL) -1  0 | |
| MD(NULL)  Return -1 | MD(NULL)  Return -1 | MD(NULL)  Return -1 | MD(NULL)  Return -1 | MD(NULL)  Return -1 | MD(NULL)  Return -1 | MD(NULL)  Return -1 | MD(NULL)  Return -1 |

int maxDepth(struct node\* node) {

if (node==NULL) {

return(-1);

}

else {

// compute the depth of each subtree

int LDepth = maxDepth (node->left);

int RDepth = maxDepth (node->right);

// use the larger one

if (LDepth > RDepth)

return(LDepth +1);

else

return(RDepth +1);

}

}