# Diameter of a Binary Tree

The diameter of a tree (sometimes called the width) is the number of nodes on the longest path between two leaves in the tree. The diagram below shows two trees each with diameter nine, the leaves that form the ends of a longest path are shaded (note that there is more than one path in each tree of length nine, but no path longer than nine nodes).

diameter, 9 nodes, through root   *diameter, 9 nodes, NOT through root*
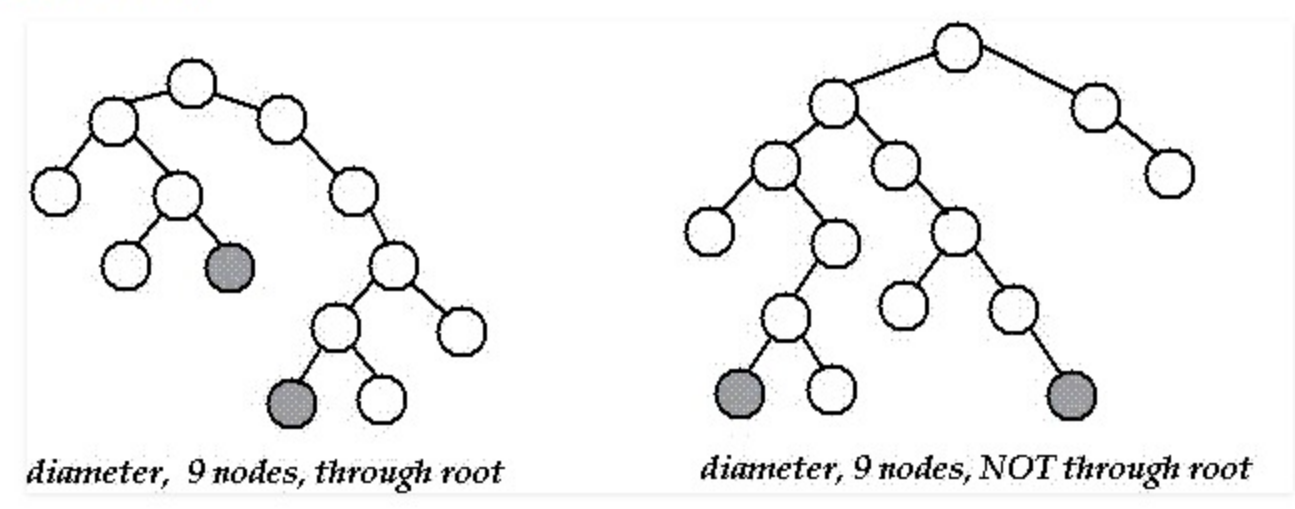
The diameter of a tree T is the largest of the following quantities:

* the diameter of T's left subtree
* the diameter of T's right subtree
* the longest path between leaves that goes through the root of T (this can be computed from the heights of the subtrees of T)

Implementation:

C   Java   Python

```c
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left, *right;
};

/* function to create a new node of tree and returns pointer */
struct node* newNode(int data);

/* returns max of two integers */
int max(int a, int b);

/* function to Compute height of a tree. */
int height(struct node* node);

/* Function to get diameter of a binary tree */
int diameter(struct node * tree)
{
    /* base case where tree is empty */
    if (tree == NULL)
      return 0;

    /* get the height of left and right sub-trees */
    int lheight = height(tree->left);
    int rheight = height(tree->right);

    /* get the diameter of left and right sub-trees */
    int ldiameter = diameter(tree->left);
    int rdiameter = diameter(tree->right);

    /* Return max of following three
     1) Diameter of left subtree
     2) Diameter of right subtree
     3) Height of left subtree + height of right subtree + 1 */
    return max(lheight + rheight + 1, max(ldiameter, rdiameter));
}

/* UTILITY FUNCTIONS TO TEST diameter() FUNCTION */

/*  The function Compute the "height" of a tree. Height is the
    number f nodes along the longest path from the root node
    down to the farthest leaf node.*/
int height(struct node* node)
{
    /* base case tree is empty */
    if(node == NULL)
        return 0;

    /* If tree is not empty then height = 1 + max of left
       height and right heights */
    return 1 + max(height(node->left), height(node->right));
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

/* returns maximum of two integers */
int max(int a, int b)
{
    return (a >= b)? a: b;
}

/* Driver program to test above functions*/
int main()
{

  /* Constructed binary tree is
            1
          /   \
         2     3
       /  \
      4    5
  */
  struct node *root = newNode(1);
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->left->left  = newNode(4);
  root->left->right = newNode(5);

  printf("Diameter of the given binary tree is %d\n", diameter(root));

  getchar();
  return 0;
}
```

`Run on IDE`

Time Complexity: O(n^2)

Output:

```
Diameter of the given binary tree is 4
```

Optimized implementation: The above implementation can be optimized by calculating the height in the same recursion rather than calling a height() separately. Thanks to Amar for suggesting this optimized version. This optimization reduces time complexity to O(n).

C   Java

```c
/*The second parameter is to store the height of tree.
   Initially, we need to pass a pointer to a location with value
   as 0. So, function should be used as follows:

   int height = 0;
   struct node *root = SomeFunctionToMakeTree();
   int diameter = diameterOpt(root, &height); */
int diameterOpt(struct node *root, int* height)
{
    /* lh --> Height of left subtree
       rh --> Height of right subtree */
    int lh = 0, rh = 0;

    /* ldiameter  --> diameter of left subtree
       rdiameter  --> Diameter of right subtree */
    int ldiameter = 0, rdiameter = 0;

    if(root == NULL)
    {
      *height = 0;
       return 0; /* diameter is also 0 */
    }

    /* Get the heights of left and right subtrees in lh and rh
       And store the returned values in ldiameter and rdiameter */
    ldiameter = diameterOpt(root->left, &lh);
    rdiameter = diameterOpt(root->right, &rh);

    /* Height of current node is max of heights of left and
       right subtrees plus 1*/
    *height = max(lh, rh) + 1;

    return max(lh + rh + 1, max(ldiameter, rdiameter));
}
```

`Run on IDE`

Time Complexity: O(n)

Output:

```
4
```

References:

http://www.cs.duke.edu/courses/spring00/cps100/assign/trees/diameter.html