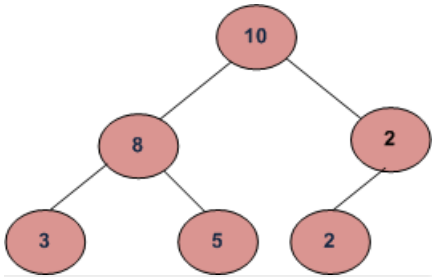
Given a binary tree, print all root-to-leaf paths

For the below example tree, all root-to-leaf paths are:

10 –> 8 –> 3

10 –> 8 –> 5

10 –> 2 –> 2



Algorithm:

Use a path array path[] to store current root to leaf path. Traverse from root to all leaves in top-down fashion. While traversing, store data of all nodes in current path in array path[]. When we reach a leaf node, print the path array.

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;
   struct node* right;
};

/* Prototypes for funtions needed in printPaths() */
void printPathsRecur(struct node* node, int path[], int pathLen);
void printArray(int ints[], int len);

/*Given a binary tree, print out all of its root-to-leaf
 paths, one per line. Uses a recursive helper to do the work.*/
void printPaths(struct node* node)
{
  int path[1000];
  printPathsRecur(node, path, 0);
}

/* Recursive helper function -- given a node, and an array containing
 the path from the root node up to but not including this node,
 print out all the root-leaf paths.*/
void printPathsRecur(struct node* node, int path[], int pathLen)
{
  if (node==NULL)
    return;

  /* append this node to the path array */
  path[pathLen] = node->data;
  pathLen++;

  /* it's a leaf, so print the path that led to here  */
  if (node->left==NULL && node->right==NULL)
  {
```

```
    printArray(path, pathLen);
  }
  else
  {
    /* otherwise try both subtrees */
    printPathsRecur(node->left, path, pathLen);
    printPathsRecur(node->right, path, pathLen);
  }
}


/* UTILITY FUNCTIONS */
/* Utility that prints out an array on a line. */
void printArray(int ints[], int len)
{
  int i;
  for (i=0; i<len; i++)
  {
    printf("%d ", ints[i]);
  }
  printf("\n");
}

/* utility 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);
}

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

  /* Constructed binary tree is
            10
          /   \
        8      2
      /  \    /
    3     5  2
  */
  struct node *root = newnode(10);
  root->left        = newnode(8);
  root->right       = newnode(2);
  root->left->left  = newnode(3);
  root->left->right = newnode(5);
  root->right->left = newnode(2);

  printPaths(root);

  getchar();
  return 0;
}
```

Java

```
// Java program to print all the node to leaf path

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
class Node
{
    int data;
    Node left, right;
```

```
    Node(int item)
    {
        data = item;
        left = right = null;
    }
}

class BinaryTree
{
    Node root;

    /*Given a binary tree, print out all of its root-to-leaf
      paths, one per line. Uses a recursive helper to do
      the work.*/
    void printPaths(Node node)
    {
        int path[] = new int[1000];
        printPathsRecur(node, path, 0);
    }

    /* Recursive helper function -- given a node, and an array
       containing the path from the root node up to but not
       including this node, print out all the root-leaf paths.*/
    void printPathsRecur(Node node, int path[], int pathLen)
    {
        if (node == null)
            return;

        /* append this node to the path array */
        path[pathLen] = node.data;
        pathLen++;

        /* it's a leaf, so print the path that led to here  */
        if (node.left == null && node.right == null)
            printArray(path, pathLen);
        else
        {
            /* otherwise try both subtrees */
            printPathsRecur(node.left, path, pathLen);
            printPathsRecur(node.right, path, pathLen);
        }
    }

    /* Utility function that prints out an array on a line. */
    void printArray(int ints[], int len)
    {
        int i;
        for (i = 0; i < len; i++)
        {
            System.out.print(ints[i] + " ");
        }
        System.out.println("");
    }

    // driver program to test above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(8);
        tree.root.right = new Node(2);
        tree.root.left.left = new Node(3);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(2);

        /* Let us test the built tree by printing Insorder traversal */
        tree.printPaths(tree.root);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Time Complexity: O(n)

References:
http://cslibrary.stanford.edu/110/BinaryTrees.html