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

struct Node
{
   int data;
   struct Node *left, *right;
};


struct Node *newNode(int data)
{
   struct Node *node = (struct Node*) malloc(sizeof(struct Node));
   node->data = data;
   node->left = node->right = NULL;
   return node;
}

// Utility method that actually removes the nodes which are not
// on the pathLen < k.
struct Node *removeShortPathlessthanK(struct Node *root,  int k)
{
   if(k==0)
     return root;
   //Base condition
   if (root == NULL)
      return NULL;


   root->left = removeShortPathlessthanK(root->left, k-1);
   root->right = removeShortPathlessthanK(root->right, k-1);


   if (root->left == NULL && root->right == NULL )
   {
      free (root);
      return NULL;
   }

   // Return root;
   return root;
}


//Method to print the tree in inorder fashion.
void printInorder(struct Node *root)
{
   if (root)
```

```
    {
        printInorder(root->left);
        printf("%d  ",root->data);
        printInorder(root->right);
    }
}

// Driver method.
int main()
{
    int k = 3;
    struct Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->left->left->left = newNode(7);
    root->right->right = newNode(6);
    root->right->right->left = newNode(8);
    printf("Inorder Traversal of Original tree");
    printInorder(root);
    printf("Inorder Traversal of Modified tree");
    struct Node *res = removeShortPathlessthanK(root, k);
    printInorder(res);
    return 0;
}
```

Output: Inorder Traversal of Original tree 7 4 2 5 1 3 8 6
Inorder Traversal of Modified tree 7 4 2 1 3 8 6

Time complexity: O(n)
Space complexity: O(n)