（7）

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

// 二叉树节点结构
typedef struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
} TreeNode;

// 路径节点结构（用于存储最长路径）
typedef struct PathNode {
    int data;
    struct PathNode* next;
} PathNode;

// 创建新节点
TreeNode* createNode(int data) {
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// 向路径中添加节点
PathNode* addToPath(PathNode* path, int data) {
    PathNode* newNode = (PathNode*)malloc(sizeof(PathNode));
    newNode->data = data;
    newNode->next = path;
    return newNode;
}

// 复制路径
PathNode* copyPath(PathNode* path) {
    PathNode* newPath = NULL;
    PathNode* temp = path;

    // 先反转原路径（因为路径是逆序存储的）
    PathNode* reversed = NULL;
    while (temp != NULL) {
        PathNode* newNode = (PathNode*)malloc(sizeof(PathNode));
        newNode->data = temp->data;
```

```
        newNode->next = reversed;
        reversed = newNode;
        temp = temp->next;
    }

    // 再复制反转后的路径
    temp = reversed;
    while (temp != NULL) {
        newPath = addToPath(newPath, temp->data);
        temp = temp->next;
    }

    return newPath;
}

// 释放路径内存
void freePath(PathNode* path) {
    while (path != NULL) {
        PathNode* temp = path;
        path = path->next;
        free(temp);
    }
}

// 查找最长路径的递归函数
void findLongestPath(TreeNode* root, PathNode* currentPath, int currentDepth,
                     PathNode** longestPath, int* maxDepth) {
    if (root == NULL) {
        return;
    }

    // 将当前节点添加到路径中
    currentPath = addToPath(currentPath, root->data);

    // 如果是叶子节点
    if (root->left == NULL && root->right == NULL) {
        if (currentDepth > *maxDepth) {
            *maxDepth = currentDepth;
            // 释放旧的最长路径
            freePath(*longestPath);
            // 复制当前路径作为新的最长路径
            *longestPath = copyPath(currentPath);
        }
    }
```

```
    // 递归遍历左右子树
    findLongestPath(root->left, currentPath, currentDepth + 1, longestPath, maxDepth);
    findLongestPath(root->right, currentPath, currentDepth + 1, longestPath, maxDepth);

    // 回溯：从路径中移除当前节点
    if (currentPath != NULL) {
        PathNode* temp = currentPath;
        currentPath = currentPath->next;
        free(temp);
    }
}

// 打印路径
void printPath(PathNode* path) {
    printf("最长路径：");
    while (path != NULL) {
        printf("%d", path->data);
        if (path->next != NULL) {
            printf(" -> ");
        }
        path = path->next;
    }
    printf("\n");
}

// 主函数：查找并输出最长路径
void getLongestPath(TreeNode* root) {
    if (root == NULL) {
        printf("二叉树为空！\n");
        return;
    }

    PathNode* longestPath = NULL;
    int maxDepth = 0;

    findLongestPath(root, NULL, 1, &longestPath, &maxDepth);

    printf("第一条最长路径的长度：%d\n", maxDepth);
    printPath(longestPath);

    // 释放内存
    freePath(longestPath);
}
```

```
// 测试代码
int main() {
    // 创建测试二叉树:
    //          1
    //        / \
    //       2   3
    //      / \   \
    //     4   5   6
    //          \
    //           7

    TreeNode* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->right = createNode(6);
    root->left->right->right = createNode(7);

    getLongestPath(root);

    return 0;
}
```

（8）

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

// 二叉树节点结构
typedef struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
} TreeNode;

// 路径节点结构（用于存储路径）
typedef struct PathNode {
    int data;
    struct PathNode* next;
} PathNode;

// 创建新节点
TreeNode* createNode(int data) {
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->data = data;
```

```
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// 向路径尾部添加节点
PathNode* addToPathEnd(PathNode* path, int data) {
    PathNode* newNode = (PathNode*)malloc(sizeof(PathNode));
    newNode->data = data;
    newNode->next = NULL;

    if (path == NULL) {
        return newNode;
    }

    PathNode* current = path;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
    return path;
}

// 向路径头部添加节点
PathNode* addToPathFront(PathNode* path, int data) {
    PathNode* newNode = (PathNode*)malloc(sizeof(PathNode));
    newNode->data = data;
    newNode->next = path;
    return newNode;
}

// 复制路径
PathNode* copyPath(PathNode* path) {
    PathNode* newPath = NULL;
    PathNode* temp = path;

    while (temp != NULL) {
        newPath = addToPathEnd(newPath, temp->data);
        temp = temp->next;
    }

    return newPath;
}
```

```
// 打印路径（从叶子到根）
void printPathFromLeafToRoot(PathNode* path) {
    printf("路径：");

    // 先反转路径（因为存储时是从根到叶子）
    PathNode* reversed = NULL;
    PathNode* current = path;
    while (current != NULL) {
        reversed = addToPathFront(reversed, current->data);
        current = current->next;
    }

    // 打印反转后的路径（从叶子到根）
    current = reversed;
    while (current != NULL) {
        printf("%d", current->data);
        if (current->next != NULL) {
            printf(" -> ");
        }
        current = current->next;
    }
    printf("\n");

    // 释放反转路径的内存
    while (reversed != NULL) {
        PathNode* temp = reversed;
        reversed = reversed->next;
        free(temp);
    }
}

// 打印路径（从根到叶子）
void printPathFromRootToLeaf(PathNode* path) {
    printf("路径：");
    PathNode* current = path;
    while (current != NULL) {
        printf("%d", current->data);
        if (current->next != NULL) {
            printf(" -> ");
        }
        current = current->next;
    }
    printf("\n");
}
```

```
// 释放路径内存
void freePath(PathNode* path) {
    while (path != NULL) {
        PathNode* temp = path;
        path = path->next;
        free(temp);
    }
}

// 递归函数：输出所有叶子节点到根节点的路径
void printAllLeafPaths(TreeNode* root, PathNode* currentPath) {
    if (root == NULL) {
        return;
    }

    // 将当前节点添加到路径中
    currentPath = addToPathEnd(currentPath, root->data);

    // 如果是叶子节点，打印路径
    if (root->left == NULL && root->right == NULL) {
        printf("叶子节点 %d 到根节点的路径：", root->data);
        printPathFromLeafToRoot(currentPath);
    }

    // 递归遍历左右子树
    printAllLeafPaths(root->left, currentPath);
    printAllLeafPaths(root->right, currentPath);

    // 回溯：从路径中移除当前节点（通过不保留修改）
    // 这里不需要手动移除，因为 currentPath 是值传递的副本
}

// 更简洁的递归实现（使用数组存储路径）
void printLeafPathsSimple(TreeNode* root, int path[], int pathLen) {
    if (root == NULL) {
        return;
    }

    // 将当前节点添加到路径数组
    path[pathLen] = root->data;
    pathLen++;

    // 如果是叶子节点，打印路径
```

```
    if (root->left == NULL && root->right == NULL) {
        printf("叶子节点 %d 到根节点的路径: ", root->data);
        for (int i = pathLen - 1; i >= 0; i--) {
            printf("%d", path[i]);
            if (i > 0) {
                printf(" -> ");
            }
        }
        printf("\n");
    } else {
        // 递归遍历左右子树
        printLeafPathsSimple(root->left, path, pathLen);
        printLeafPathsSimple(root->right, path, pathLen);
    }
}

// 主函数：输出所有叶子节点到根节点的路径
void printAllLeafToRootPaths(TreeNode* root) {
    if (root == NULL) {
        printf("二叉树为空！\n");
        return;
    }

    printf("=== 方法一：使用链表 ===\n");
    printAllLeafPaths(root, NULL);

    printf("\n=== 方法二：使用数组（更简洁） ===\n");
    int path[100]; // 假设路径长度不超过 100
    printLeafPathsSimple(root, path, 0);
}

// 测试代码
int main() {
    // 创建测试二叉树:
    //          1
    //        / \
    //       2   3
    //      / \   \
    //     4   5   6
    // 叶子节点：4, 5, 6

    TreeNode* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
```

```
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->right = createNode(6);

    printf("二叉树结构：\n");
    printf("            1\n");
    printf("          / \\\\\n");
    printf("         2     3\n");
    printf("       / \\\\     \\\\\n");
    printf("     4     5     6\n\n");

    printAllLeafToRootPaths(root);

    return 0;
}
```