Algorithm assignment #3

응용통계학과 20202850 김지민

1.

- **Code**

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

typedef struct ListNode {
    int data;
    struct ListNode* next;
} ListNode;


ListNode* newrecord(int value) {
    ListNode* newp;
    newp = (ListNode*)malloc(sizeof(ListNode));
    newp->data = value;
    newp->next = NULL;
    return newp;
}

// append : add a newp to the end of
void append(ListNode** head, ListNode* newp) {
    if ((*head) == NULL) {
        *head = newp;
    }
    else {
        ListNode* listp = (*head);

        while (listp->next != NULL) {
            listp = listp->next;
        }
        listp->next = newp;
    }
}
```

```c
// insert : add a newp next to listp
void insert(ListNode** head, ListNode* listp, ListNode* newp) {
    if (*head == NULL) {
        newp->next = NULL;
        *head = newp;
    }
    else if (listp == NULL) {
        newp->next = *head;
        *head = newp;
    }
    else {
        newp->next = listp->next;
        listp->next = newp;
    }

}

// delete : delete a removep next to listp
void delete(ListNode** head, ListNode* listp, ListNode* removep) {
    if (listp == NULL)
        *head = (*head)->next;
    else
        listp->next = removep->next;
    free(removep);
}

// traverse : print all records in the linked list
void traverse(ListNode** head) {
    for (ListNode* p = *head; p != NULL; p = p->next)
        printf("%d->", p->data);
    printf("NULL₩n");
}

// reverse : change the records in the linked list to reverse order
ListNode* reverse(ListNode** head)
{
    ListNode* listp, * nextp, * temp;
    listp = *head;
    nextp = NULL;

    while (listp != NULL)
    {
        temp = nextp;
        nextp = listp;
        listp = listp->next;
        nextp->next = temp;
    }
    return nextp;
}
```

```c
int main() {
    int random;
    ListNode* head = NULL;
    for (int i = 0; i < 10; i++) {
        random = rand() % 10;
        append(&head, newrecord(random));
    }
    printf("current linked list...₩n");
    traverse(&head);

    printf("₩nafter delete second node...₩n");
    delete(&head, head, head->next);
    traverse(&head);

    printf("₩nafter insert new node in second position...₩n");
    ListNode* n = newrecord(6);
    insert(&head, head, n);
    traverse(&head);

    printf("₩nafter the linked list is reversed...₩n");
    head = reverse(&head);
    traverse(&head);

    return 0;
}
```

- **Result**

Microsoft Visual Studio 디버그 콘솔

```
current linked list...
1->7->4->0->9->4->8->8->2->4->NULL

after delete second node...
1->4->0->9->4->8->8->2->4->NULL

after insert new node in second position...
1->6->4->0->9->4->8->8->2->4->NULL

after the linked list is reversed...
4->2->8->8->4->9->0->4->6->1->NULL

C:₩Users₩jimin.DESKTOP-8V20QSQ₩source₩repos₩Project1₩Debug₩Project1.
).
```

2.

- **Code**

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

typedef struct ListNode {
    int data;
    struct ListNode* next;
} ListNode;


ListNode* newrecord(int value) {
    ListNode* newp;
    newp = (ListNode*)malloc(sizeof(ListNode));
    newp->data = value;
    newp->next = NULL;
    return newp;
}

// append : add a newp to the end of
void append(ListNode** head, ListNode* newp) {
    if ((*head) == NULL) {
        *head = newp;
    }
    else {
        ListNode* listp = (*head);

        while (listp->next != NULL) {
            listp = listp->next;
        }
        listp->next = newp;
    }
}
```

```c
// insert : add a newp next to listp
void insert(ListNode** head, ListNode* listp, ListNode* newp) {
    if (*head == NULL) {
        newp->next = NULL;
        *head = newp;
    }
    else if (listp == NULL) {
        newp->next = *head;
        *head = newp;
    }
    else {
        newp->next = listp->next;
        listp->next = newp;
    }

}

// delete : delete a removep next to listp
void delete(ListNode** head, ListNode* listp, ListNode* removep) {
    if (listp == NULL)
        *head = (*head)->next;
    else
        listp->next = removep->next;
    free(removep);
}

// traverse : print all records in the linked list
void traverse(ListNode** head) {
    for (ListNode* p = *head; p != NULL; p = p->next)
        printf("%d->", p->data);
    printf("NULL₩n");
}
```

```c
// remove_duplicates : remove duplicates from an unsorted linked list
void remove_duplicates(ListNode** head) {
    ListNode* cur = (*head);
    ListNode* comp, * temp;
    while (cur != NULL) {
        comp = cur;
        while (comp->next != NULL) {
            // Duplicates found
            if (cur->data == comp->next->data) {
                temp = comp->next;
                comp->next = comp->next->next;
                free(temp);

            }
            // No duplicates found
            else {
                comp = comp->next;
            }
        }
        cur = cur->next;
    }
}

int main() {
    int random;
    ListNode* head = NULL;
    for (int i = 0; i < 20; i++) {
        random = rand() % 50 + 1;
        append(&head, newrecord(random));
    }
    printf("current linked list...₩n");
    traverse(&head);
    printf("₩n");

    printf("after remove duplicates of linked list...₩n");
    remove_duplicates(&head);
    traverse(&head);

    return 0;
}
```

- **Result**

Microsoft Visual Studio 디버그 콘솔

```
current linked list...
42->18->35->1->20->25->29->9->13->15->6->46->32->28->12->42->46->43->28->37->NULL

after remove duplicates of linked list...
42->18->35->1->20->25->29->9->13->15->6->46->32->28->12->43->37->NULL

C:\Users\jimin.DESKTOP-8V20QSQ\source\repos\Project1\Debug\Project1.exe(프로세스 24
).
```

3.

- **Code**

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

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

Node* initNode(int n, Node* n1, Node* n2) {
    Node* result = malloc(sizeof(Node));
    result->data = n;
    result->left = n1;
    result->right = n2;
    return result;
}

// getMin : return the minimum value of the tree
int getMin(Node* root) {
    Node* cur = root;
    while (cur != NULL && cur->left != NULL)
        cur = cur->left;
    return cur->data;
}

// getMax : return the maximum value of the tree
int getMax(Node* root) {
    Node* cur = root;
    while (cur != NULL && cur->right != NULL)
        cur = cur->right;
    return cur->data;
}
```

```c
// isBST : check if binary tree is a valid binary search tree or not
char* isBST(Node* root) {
    if (root == NULL)
        return "True";

    if (root->left != NULL && getMax(root->left) > root->data)
        return "False";


    if (root->right != NULL && getMin(root->right) < root->data)
        return "False";

    if (isBST(root->left) == "False" || isBST(root->right) == "False")
        return "False";

    return "True";
}
```

```c
int main() {
    Node* n7 = initNode(7, NULL, NULL);
    Node* n6 = initNode(4, NULL, NULL);
    Node* n3 = initNode(9, n6, n7);
    Node* n2 = initNode(3, NULL, NULL);
    Node* n1 = initNode(8, n2, n3);
    Node* root = n1;

    printf("Is binary tree a valid binary search tree? : %s₩n", isBST(root));

    return 0;
}
```

- **Result**

Microsoft Visual Studio 디버그 콘솔

```
Is binary tree a valid binary search tree? : False

C:₩Users₩jimin.DESKTOP-8V20QSQ₩source₩repos₩Project1₩Debug₩Project1
).
```

4.

- **Code**

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

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

Node* initNode(int n, Node* n1, Node* n2) {
    Node* result = malloc(sizeof(Node));
    result->data = n;
    result->left = n1;
    result->right = n2;
    return result;
}

// find_ans : find the ancestor of two numbers
Node* find_ans(Node* root, int num1, int num2) {
    if (root == NULL)
        return NULL;

    // If num1 and num2 are smaller than data of root node, then ancestor lies in left
    if (root->data > num1 && root->data > num2)
        return find_ans(root->left, num1, num2);

    // If num1 and num2 are greater than data of root node, then ancestor lies in right
    if (root->data < num1 && root->data < num2)
        return find_ans(root->right, num1, num2);

    return root;
}
```

```c
// find_num : verify that the data entered is in the tree
int find_num(Node* root, int num)
{
    if (root == NULL)
        return 0;

    // there is data in binary search tree
    if (root->data == num || find_num(root->left, num) || find_num(root->right, num))
        return 1;

    // there is no data in binary search tree
    return 0;
}

// print_ans : print out a common ancestor
void print_ans(Node* root) {
    int num1, num2;
    for (int i = 0; i < 3; i++) {
        scanf("%d %d", &num1, &num2);
        getchar();
        if (!find_num(root, num1) || !find_num(root, num2))
            printf("There is no common ancestor of the two numbers.\n");
        else
            printf("[%d %d] : %d\n", num1, num2, find_ans(root, num1, num2)->data);
    }
}

int main() {
    Node* n7 = initNode(9, NULL, NULL);
    Node* n6 = initNode(7, NULL, NULL);
    Node* n5 = initNode(3, NULL, NULL);
    Node* n4 = initNode(1, NULL, NULL);
    Node* n3 = initNode(8, n6, n7);
    Node* n2 = initNode(2, n4, n5);
    Node* n1 = initNode(6, n2, n3);
    Node* root = n1;

    print_ans(root);

    return 0;
}
```

- **Result**

Microsoft Visual Studio 디버그 콘솔

```
6 2
[6 2] : 6
1 3
[1 3] : 2
7 8
[7 8] : 8

C:\Users\jimin.DESKTOP-8V20QSQ\source\repos\Project1\Debug\Project1
.
```

5.

- **Code**

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

enum Color { RED, BLACK };

typedef struct Node {
    struct Node* parent;
    struct Node* left;
    struct Node* right;
    int data;
    enum Color color;

} Node;

typedef struct rbtree {
    Node* root;
} Tree;

Node* NIL = NULL;

Node* newNode(int data) {
    Node* node = malloc(sizeof(Node));

    node->parent = NULL;
    node->left = NULL;
    node->right = NULL;
    node->color = RED;
    node->data = data;

    return node;
}
```

```c
// right_rotate : Node x rotates to the right
void right_rotate(Tree* t, Node* x) {
    Node* child = x->left;

    x->left = child->right; // register the right child node of the left child as the left side of parent node

    if (child->right != NIL)
        child->right->parent = x;

    child->parent = x->parent;

    if (x->parent == NIL) // change left child to root if parent is NULL
        t->root = child;
    else {
        if (x == x->parent->left)
            x->parent->left = child;
        else // place the right child node where the parent was
            x->parent->right = child;
    }
    child->right = x;
    x->parent = child;
}

// left_rotate : Node x rotates to the left
void left_rotate(Tree* t, Node* x) {
    Node* child = x->right;

    x->right = child->left; // register the left child node of the right child as the right side of parent node

    if (child->left != NIL)
        child->left->parent = x;

    child->parent = x->parent;

    if (x->parent == NIL) // change right child to root if parent is NULL
        t->root = child;
    else {
        if (x == x->parent->right)
            x->parent->right = child;
        else // place the left child node where the parent was
            x->parent->left = child;
    }
    child->left = x;
    x->parent = child;
}
```

```c
// rebuild_tree : rebuild the tree
void rebuild_tree(Tree* t, Node* x) {
    while (x != t->root && x->parent->color == RED) {
        // The parent node is the left child of its parent node
        if (x->parent == x->parent->parent->left) {
            Node* uncle = x->parent->parent->right;
            if (uncle == NULL || uncle->color == BLACK) {
                if (x == x->parent->right) {
                    x = x->parent;
                    left_rotate(t, x);
                }
                x->parent->color = BLACK;
                x->parent->parent->color = RED;
                right_rotate(t, x->parent->parent);
            }
            // uncle node's color is red
            else {
                x->parent->color = BLACK;
                uncle->color = BLACK;
                x->parent->parent->color = RED;
                x = x->parent->parent;
            }
        }
        // The parent node is the right child of its parent node
        else {
            Node* uncle = x->parent->parent->left;
            if (uncle == NULL || uncle->color == BLACK) {
                if (x == x->parent->left) {
                    x = x->parent;
                    right_rotate(t, x);
                }
                x->parent->color = BLACK;
                x->parent->parent->color = RED;
                left_rotate(t, x->parent->parent);
            }
            // uncle node's color is red
            else {
                x->parent->color = BLACK;
                uncle->color = BLACK;
                x->parent->parent->color = RED;
                x = x->parent->parent;
            }
        }
    }

    t->root->color = BLACK;
}
```

```c
// insert : insert a new node
void insert(Tree* t, Node* NewNode) {
    Node* cur = NIL;
    Node* temp = t->root;
    // find where to insert nodes in the tree
    while (temp != NIL) {
        cur = temp;
        if (NewNode->data < temp->data)
            temp = temp->left;
        else
            temp = temp->right;
    }
    NewNode->parent = cur;

    if (cur == NIL) // tree is empty
        t->root = NewNode;
    else if (NewNode->data < cur->data)
        cur->left = NewNode;
    else
        cur->right = NewNode;
    NewNode->left = NIL;
    NewNode->right = NIL;
    NewNode->color = RED;

    rebuild_tree(t, NewNode);
}

char* color(enum COLOR color) {
    if (color == RED)
        return "RED";
    else
        return "BLACK";
}

void inorder(Tree* t, Node* n) {
    if (n != NIL) {
        inorder(t, n->left);
        printf("%d [%s] ", n->data, color(n->color));
        inorder(t, n->right);
    }
}

void preorder(Tree* t, Node* n) {
    if (n != NIL) {
        printf("%d [%s] ", n->data, color(n->color));
        preorder(t, n->left);
        preorder(t, n->right);
    }
}
```

```c
void postorder(Tree* t, Node* n) {
    if (n != NIL) {
        postorder(t, n->left);
        postorder(t, n->right);
        printf("%d [%s] ", n->data, color(n->color));
    }
}

int main() {
    Tree* t = malloc(sizeof(Tree));
    t->root = NIL;
    insert(t, newNode(41));
    insert(t, newNode(38));
    insert(t, newNode(31));
    insert(t, newNode(12));
    insert(t, newNode(19));
    insert(t, newNode(8));

    printf("inorder : ");
    inorder(t, t->root);
    printf("₩n");
    printf("preorder : ");
    preorder(t, t->root);
    printf("₩n");
    printf("postorder : ");
    postorder(t, t->root);

    return 0;
}
```

- **Result**

Microsoft Visual Studio 디버그 콘솔

```
inorder : 8 [RED] 12 [BLACK] 19 [RED] 31 [BLACK] 38 [BLACK] 41 [BLACK]
preorder : 38 [BLACK] 19 [RED] 12 [BLACK] 8 [RED] 31 [BLACK] 41 [BLACK]
postorder : 8 [RED] 12 [BLACK] 31 [BLACK] 19 [RED] 41 [BLACK] 38 [BLACK]
C:₩Users₩jimin.DESKTOP-8V20QSQ₩source₩repos₩Project1₩Debug₩Project1.exe(프로
).
```