Algorithm

Assignment 3 물리학과 20182326 이선민

Problem 1 - code

```
1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <time.h>
 6 typedef struct ListNode {
           int data;
           struct ListNode *next;
 9 } ListNode;
10
11 void print_data(ListNode *p)
12 {
13
           printf("%d -> ",p->data);
14
           if (p->next == NULL)
15
                   printf("NULL\n");
16 }
17
18 void free_node(ListNode *p)
19 {
           if (p == NULL)
20
21
                   return;
22
           free(p);
23 }
24
25 void error(char *message)
26 {
27
           fprintf(stderr, "%s\n", message);
28
           exit(1);
29 }
```

```
31 ListNode* Append(ListNode *list, int value)
32 {
33
           if (list == NULL)
34
35
                   list = (ListNode *)malloc(sizeof(ListNode));
36
                   list->data = value;
37
                   list->next = NULL;
38
                   return list;
39
40
           ListNode *p = (ListNode *)malloc(sizeof(ListNode));
41
           p->data = value;
42
           p->next = NULL;
43
           ListNode *tmp = list;
44
           while (tmp->next)
45
                   tmp = tmp->next;
46
           tmp->next = p;
47
           return list;
48 }
49
50 void Traverse(ListNode *list, void (*fp)(ListNode *))
51 {
52
           if (list == NULL)
53
54
                   printf("List is Empty.\n");
55
                   return ;
56
57
           ListNode *tmp = list;
58
           while (tmp)
59
60
                   tmp = list->next;
61
                   fp(list);
62
                   list = tmp;
63
           }
```

Problem1 - code

```
66 ListNode* Insert(ListNode *list, int index, int value)
67 {
           if (list == NULL && index == 0)
                  return Append(list, value);
           else if (list == NULL)
                  error("IndexError\n");
          ListNode *prev = list;
          ListNode *tmp = list->next;
           for (int i = 1; i < index; i++)
                   if (tmp == NULL)
                          Traverse(list, free_node);
                          error("IndexError\n");
                   tmp = tmp->next;
                   prev = prev->next;
           if (prev == NULL)
                  Traverse(list, free_node);
                   error("IndexError\n");
           ListNode *p = (ListNode *)malloc(sizeof(ListNode));
           p->data = value;
           if (index == 0)
                   p->next = list;
                   return p;
           p->next = tmp;
           prev->next = p;
           return list;
```

```
101 ListNode* Remove(ListNode *list, int index)
102 {
103
            if (list == NULL)
104
                    error("list is empty\n");
105
            if (index == 0)
106
107
                    ListNode *removed = list;
108
                    list = removed->next;
109
                    free(removed);
110
                    return list;
111
112
            if (index == 1 && list->next == NULL)
113
114
                    free(list);
115
                    list = NULL;
116
                    error("IndexError\n");
117
118
            ListNode *prev = list;
119
            ListNode *removed = list-> next;
120
            for (int i = 1; i < index; i++)
121
122
                    if (removed == NULL)
123
124
                            Traverse(list, free_node);
125
                            error("IndexError\n");
126
127
                    removed = removed->next;
128
                    prev = prev->next;
129
130
            if (removed == NULL)
131
132
                    Traverse(list, free_node);
133
                    error("IndexError\n");
134
135
            prev->next = removed->next;
136
            free(removed);
137
            return list;
138 }
```

Problem1 - code

```
140 ListNode *Reverse(ListNode *list)
141 {
142
            ListNode *rev = NULL;
143
            ListNode *tmp = list;
144
            while (tmp)
145
146
                    rev = Insert(rev, 0, tmp->data);
147
                    tmp = tmp->next;
148
149
            tmp = list;
150
            while (rev)
151
152
                    tmp->data = rev->data;
153
                    tmp = tmp->next;
154
                    rev = rev->next;
155
156
            return list;
157 }
158
159 int main(void)
160 {
161
            ListNode *head = NULL;
162
163
            int i;
164
            srand(time(NULL));
165
            for (i = 0; i < 10; i++)
166
167
                    int data = rand()\% 50;
168
                    head = Append(head, data);
169
170
            Traverse(head, print_data);
171
            Reverse(head);
172
            Traverse(head, print_data);
173
174
            Traverse(head, free_node);
175
            return 0;
176 }
```

Problem 1 - result

```
[~/2022_1/Algorithm/assignment3]$ gcc problem1.c
[~/2022_1/Algorithm/assignment3]$ ./a.out
9 -> 11 -> 11 -> 47 -> 13 -> 15 -> 5 -> 26 -> 31 -> 10 -> NULL
10 -> 31 -> 26 -> 5 -> 15 -> 13 -> 47 -> 11 -> 9 -> NULL
```

Problem2 - code

 The linked list code was used in the same way as Problem 1, but only the return value of the Remove function was changed. The return value was set as the node immediately before the deleted node.

```
LO1 ListNode* Remove(ListNode *list, int index)
           if (list == NULL)
                   error("list is empty\n");
           if (index == 0)
                   ListNode *removed = list;
                   list = removed->next;
                    free(removed);
                    return list;
111
           if (index == 1 && list->next == NULL)
113
114
                    free(list);
115
                    list = NULL;
116
                   error("IndexError\n");
           ListNode *prev = list;
           ListNode *removed = list-> next;
           for (int i = 1; i < index; i++)
                   if (removed == NULL)
                            Traverse(list, free_node);
                            error("IndexError\n");
                   removed = removed->next;
                   prev = prev->next;
           if (removed == NULL)
131
                   Traverse(list, free_node);
                   error("IndexError\n");
           prev->next = removed->next;
           free(removed);
            return prev;
```

Problem2

- newly added function, modifying main function

```
140 int Check_dup(ListNode *list, int data)
141 {
142
            int count = 0;
143
            if (!list)
                    return (0);
145
            while (list)
146
                    if (list->data == data)
148
                            return (1);
149
                    list = list->next;
150
           return (0);
152 }
154 void Remove_Duplicate(ListNode *list)
155 {
156
            ListNode *tmp = NULL;
            ListNode *list_tmp = list;
            int count = 0;
159
            while (list_tmp)
160
                    if (Check_dup(tmp, list_tmp->data))
                            list_tmp = Remove(list, count);
                    else
164
                            count++:
                            tmp = Append(tmp, list_tmp->data);
168
                    list_tmp = list_tmp->next;
169
170
            Traverse(tmp, free_node);
171 }
```

```
172
173 int main(void)
174 {
175
            ListNode *head = NULL;
176
177
            int i:
178
            srand(time(NULL));
179
            for (i = 0: i < 20: i++)
180
181
                    int data = rand()\% 50 + 1;
182
                    head = Append(head, data);
183
184
            Traverse(head, print_data);
            Remove_Duplicate(head);
185
186
            Traverse(head, print_data);
187
            Traverse(head, free_node);
188
            return 0;
189 }
```

Problem2 - result

```
[~/2022_1/Algorithm/assignment3]$ gcc problem2.c
[~/2022_1/Algorithm/assignment3]$ ./a.out
28 -> 46 -> 1 -> 44 -> 47 -> 20 -> 38 -> 47 -> 35 -> 1 -> 20 -> 29 -> 11 -> 6 -> 13 -> 20 -> 3 -> 27 -> 39 -> 9 -> NULL
28 -> 46 -> 1 -> 44 -> 47 -> 20 -> 38 -> 35 -> 29 -> 11 -> 6 -> 13 -> 27 -> 39 -> 9 -> NULL
```

Problem3 - code

```
1 #include <stdio.h>
 2 #include <stdlib.h>
 4 typedef struct _node{
           int data;
           struct _node * right;
           struct _node * left;
 8 } Node;
10 Node *Create(int root) {
       Node *tree = (Node *)malloc(sizeof(Node));
      tree->data = root;
      tree->right = NULL;
      tree->left = NULL;
15
       return tree;
16 }
18 void free_tree(Node *tree)
19 {
20
           if (tree)
                   free_tree(tree->left);
                   free_tree(tree->right);
                   free(tree);
26
27 }
29 int isBST(Node *node)
30 {
           if (node == NULL)
                   return 1;
           else if (node->left != NULL && node->left->data > node->data)
                   return 0;
           else if (node->right != NULL && node->right->data < node->data)
                   return 0:
           else if (!isBST(node->left) || !isBST(node->right))
38
                   return 0;
39
           return 1;
```

```
42 int main(void)
43 {
44
           Node *n1 = Create(8);
45
           Node *n2 = Create(3);
           Node *n3 = Create(9);
           Node *n4 = NULL;
           Node *n5 = NULL;
           Node *n6 = Create(4);
49
50
           Node *n7 = Create(7);
51
           n1->left = n2;
52
           n1->right = n3;
53
           n2 - left = n4;
54
           n2->right = n5;
55
           n3 - > left = n6;
56
           n3->right = n7;
57
           if (isBST(n1))
58
                   printf("It is BST\n");
59
           else
60
                   printf("It is not BST\n");
61
           free_tree(n1);
62 }
```

Problem3 - result

```
[~/2022_1/Algorithm/assignment3]$ gcc problem3.c [~/2022_1/Algorithm/assignment3]$ ./a.out It is not BST
```

Problem4 - code

```
1 #include <stdio.h>
 2 #include <stdlib.h>
4 typedef struct _node{
           int data;
           struct _node * right;
           struct _node * left;
8 } Node;
10 Node *Create(int root) {
           Node *tree = (Node *)malloc(sizeof(Node));
11
12
           tree->data = root;
13
           tree->right = NULL;
14
           tree->left = NULL;
15
           return tree;
16 }
17
```

```
18 Node *Insert(Node *tree, int data)
19 {
20
           if (tree == NULL)
21
22
                   tree = Create(data);
23
                  return tree;
24
25
          Node *parent = NULL;
26
           Node *tmp = tree;
27
          while (tmp)
28
29
                  parent = tmp;
30
                  if (data < tmp->data)
31
                           tmp = tmp->left;
32
                  else
33
                           tmp = tmp->right;
34
35
           Node *newNode = (Node *)malloc(sizeof(Node));
36
          newNode->data = data;
37
           newNode->right = NULL;
          newNode->left = NULL;
39
           if (newNode->data < parent->data)
40
                  parent->left = newNode;
41
           else
42
                  parent->right = newNode;
43
           return tree;
44 }
```

Problem4 - code

```
47 void Inorder(Node *tree) {
          if (tree)
50
                   Inorder(tree->left);
51
                  printf("%d ", tree->data);
52
                   Inorder(tree->right);
54 }
56 Node *Search(Node *tree, int data)
57 {
58
          if (tree == NULL || data == tree->data)
59
                   return tree;
60
           if (data < tree->data)
61
                   return Search(tree->left, data);
62
           else
63
                  return Search(tree->right, data);
64 }
66 void free_tree(Node *tree)
67 {
68
69
          if (tree)
70
71
                   free_tree(tree->left);
72
                   free_tree(tree->right);
73
                   free(tree);
74
75 }
```

```
77 int find_ancestor(Node *tree, int num1, int num2)
 78 {
            Node *tmp = tree;
 79
 80
            if (!Search(tree, num1) || !Search(tree, num2))
 81
                    exit(1);
 82
            while (1)
 83
 84
                    if (num1 == tmp->data || num2 == tmp->data)
 85
                            return tmp->data;
 86
                    else if (tmp->data < num1 && tmp->data < num2)</pre>
 87
                            tmp = tmp->right;
 88
                    else if (tmp->data > num1 && tmp->data > num2)
 89
                            tmp = tmp->left;
                    else
 91
                            return tmp->data;
 93 }
 95 int main(void)
 96 {
            Node *tree = NULL;
 98
            tree = Insert(tree, 6);
            Insert(tree, 2);
100
            Insert(tree, 8);
101
            Insert(tree, 1);
102
            Insert(tree, 3);
103
            Insert(tree, 7);
104
            Insert(tree, 9);
105
            printf("Binary search tree: ");
106
            Inorder(tree);
107
            printf("\n");
108
109
            int num1;
110
            int num2;
111
            printf("First node number: ");
112
            scanf("%d",&num1);
113
            printf("Second node number: ");
114
            scanf("%d",&num2);
115
            int ancestor = find_ancestor(tree, num1, num2);
            printf("The lowest common ancestor is %d\n", ancestor);
116
117
            free_tree(tree);
118 }
```

Problem4 - result

```
[~/2022_1/Algorithm/assignment3]$ gcc problem4.c
[~/2022_1/Algorithm/assignment3]$ ./a.out
Binary search tree: 1 2 3 6 7 8 9
First node number: 2
Second node number: 8
The lowest common ancestor is 6
[~/2022_1/Algorithm/assignment3]$ ./a.out
Binary search tree: 1 2 3 6 7 8 9
First node number: 1
Second node number: 7
The lowest common ancestor is 6
[~/2022_1/Algorithm/assignment3]$ ./a.out
Binary search tree: 1 2 3 6 7 8 9
First node number: 9
Second node number: 7
The lowest common ancestor is 8
```