

Algorithm

Assignment 3

물리학과 20182326 이선민

Problem 1 - code

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5
6 typedef struct ListNode {
7     int data;
8     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 {
20     if (p == NULL)
21         return;
22     free(p);
23 }
24
25 void error(char *message)
26 {
27     fprintf(stderr, "%s\n", message);
28     exit(1);
29 }
30
```

```
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 = tmp->next;
61         fp(list);
62         list = tmp;
63     }
64 }
65
```

Problem1 - code

```
66 ListNode* Insert(ListNode *list, int index, int value)
67 {
68     if (list == NULL && index == 0)
69         return Append(list, value);
70     else if (list == NULL)
71         error("IndexError\n");
72     ListNode *prev = list;
73     ListNode *tmp = list->next;
74     for (int i = 1; i < index; i++)
75     {
76         if (tmp == NULL)
77         {
78             Traverse(list, free_node);
79             error("IndexError\n");
80         }
81         tmp = tmp->next;
82         prev = prev->next;
83     }
84     if (prev == NULL)
85     {
86         Traverse(list, free_node);
87         error("IndexError\n");
88     }
89     ListNode *p = (ListNode *)malloc(sizeof(ListNode));
90     p->data = value;
91     if (index == 0)
92     {
93         p->next = list;
94         return p;
95     }
96     p->next = tmp;
97     prev->next = p;
98     return list;
99 }
```

```
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 }
139
```

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 -> 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.

```
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 prev;
138 }
139
```


Problem2

- newly added function, modifying main function

```
140 int Check_dup(ListNode *list, int data)
141 {
142     int count = 0;
143     if (!list)
144         return (0);
145     while (list)
146     {
147         if (list->data == data)
148             return (1);
149         list = list->next;
150     }
151     return (0);
152 }
153
154 void Remove_Duplicate(ListNode *list)
155 {
156     ListNode *tmp = NULL;
157     ListNode *list_tmp = list;
158     int count = 0;
159     while (list_tmp)
160     {
161         if (Check_dup(tmp, list_tmp->data))
162             list_tmp = Remove(list, count);
163         else
164         {
165             count++;
166             tmp = Append(tmp, list_tmp->data);
167         }
168         list_tmp = list_tmp->next;
169     }
170     Traverse(tmp, free_node);
171 }
172
```

```
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);
185     Remove_Duplicate(head);
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 -> 3 -> 27 -> 39 -> 9 -> NULL
[~/2022_1/Algorithm/assignment3]$
```


Problem3 - code

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct _node{
5     int data;
6     struct _node * right;
7     struct _node * left;
8 } Node;
9
10 Node *Create(int root) {
11     Node *tree = (Node *)malloc(sizeof(Node));
12     tree->data = root;
13     tree->right = NULL;
14     tree->left = NULL;
15     return tree;
16 }
17
18 void free_tree(Node *tree)
19 {
20     if (tree)
21     {
22         free_tree(tree->left);
23         free_tree(tree->right);
24         free(tree);
25     }
26 }
27
28 int isBST(Node *node)
29 {
30     if (node == NULL)
31         return 1;
32     else if (node->left != NULL && node->left->data > node->data)
33         return 0;
34     else if (node->right != NULL && node->right->data < node->data)
35         return 0;
36     else if (!isBST(node->left) || !isBST(node->right))
37         return 0;
38     return 1;
39 }
40 }
41
```

```
41
42 int main(void)
43 {
44     Node *n1 = Create(8);
45     Node *n2 = Create(3);
46     Node *n3 = Create(9);
47     Node *n4 = NULL;
48     Node *n5 = NULL;
49     Node *n6 = Create(4);
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>
3
4 typedef struct _node{
5     int data;
6     struct _node * right;
7     struct _node * left;
8 } Node;
9
10 Node *Create(int root) {
11     Node *tree = (Node *)malloc(sizeof(Node));
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;
38     newNode->left = NULL;
39     if (newNode->data < parent->data)
40         parent->left = newNode;
41     else
42         parent->right = newNode;
43     return tree;
44 }
45
```

Problem4 - code

```
47 void Inorder(Node *tree) {
48     if (tree)
49     {
50         Inorder(tree->left);
51         printf("%d ", tree->data);
52         Inorder(tree->right);
53     }
54 }
55
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 }
65
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 }
76
```

```
77 int find_ancestor(Node *tree, int num1, int num2)
78 {
79     Node *tmp = tree;
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)
87             tmp = tmp->right;
88         else if (tmp->data > num1 && tmp->data > num2)
89             tmp = tmp->left;
90         else
91             return tmp->data;
92     }
93 }
94
95 int main(void)
96 {
97     Node *tree = NULL;
98     tree = Insert(tree, 6);
99     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);
116     printf("The lowest common ancestor is %d\n", ancestor);
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
```