Informatics II, Spring 2023, Solution 8

Publication of exercise: April 23, 2023
Publication of solution: April 30, 2023
Exercise classes: May 2 - May 5, 2023

Binary Tree

Task 1

BC A A A

Task 2

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

struct TreeNode{
  int val;
  struct TreeNode* left;
  struct TreeNode* right;
};

void insert(struct TreeNode** root, int val) {
  struct TreeNode* newTreeNode = NULL;
  struct TreeNode* prev = NULL;
  struct TreeNode* current = *root;
  newTreeNode = malloc(sizeof(struct TreeNode));
  newTreeNode->val = val;
  newTreeNode->left = NULL;
  newTreeNode->right = NULL;
  while (current != NULL) {
    prev = current;
    if (val < current->val){
      current = current->left;
    } else{
      current = current->right;
    }
  }
  if (prev == NULL) {
    *root = newTreeNode;
  } else if (val < prev->val) {
    prev->left = newTreeNode;
  } else {
    prev->right = newTreeNode;
  }
}

struct TreeNode* search(struct TreeNode* root, int val) {
  struct TreeNode* current = root;
```

```
37    while (current != NULL && current->val != val) {
38      if (val < current->val){
39          current = current->left;
40      } else{
41          current = current->right;
42      }
43    }
44    return current;
45  }
46
47  void delete(struct TreeNode** root, int val) {
48    struct TreeNode* x = search(*root, val);
49    if (x == NULL){ //search did not find an element, hence do nothing.
50      return;
51    }
52    struct TreeNode* u = *root;
53    struct TreeNode* prev = NULL; // parent of tree node with value = val
54    while (u != x) {
55      prev = u;
56      if (x->val < u->val){
57        u = u->left;
58      } else{
59        u = u->right;
60      }
61    }
62    // Leaf and root case also handled in the no right or left branch. Since if it's leaf, its
          null anyway.
63    if (u->right == NULL) { // there is no right branch
64      if (prev == NULL){ // delete root
65        *root = u->left;
66      } else if (prev->left == u){ //if it's a left child, make left the new child
67        prev->left = u->left;
68      } else{
69        prev->right = u->left;
70      }
71    } else if (u->left == NULL) { // there is no left branch
72      if (prev == NULL){ // delete root
73        *root = u->right;
74      } else if (prev->left == u){ //if it's a left child, make right the new child
75        prev->left = u->right;
76      } else{
77        prev->right = u->right;
78      }
79    } else{
80      struct TreeNode* p = x->left;
81      struct TreeNode* q = p;
82      while (p->right != NULL) { //whilst right is null
83        q = p;
84        p = p->right;
85      }
86      if (prev == NULL){ // if we are at root
87        *root = p;
88      } else if (prev->left == u){ // if its a left child
89        prev->left = p;
90      } else{ //if its a right child
91        prev->right = p;
92      }
93      p->right = u->right;
94      if (q != p) {
95        q->right = p->left;
96        p->left = u->left;
97      }
98    }
99    free(u);
```

```
100 }
101
102 void printTreeRecursive(struct TreeNode *root, int level) {
103   if (root == NULL)
104     return;
105   if (root->left != NULL) {
106     printf("␣␣%d␣--␣%d␣:␣%d\n", root->val, root->left->val,level);
107     printTreeRecursive(root->left,level+1);
108   }
109   if (root->right != NULL) {
110     printf("␣␣%d␣--␣%d␣:␣%d\n", root->val, root->right->val,level);
111     printTreeRecursive(root->right,level+1);
112   }
113 }
114
115 void printTree(struct TreeNode *root) {
116   printf("graph␣g␣{\n");
117   printTreeRecursive(root, 1);
118   printf("}\n");
119 }
120
121 void pretraverseTree(struct TreeNode *root) {
122   printf("%d␣", root->val);
123   if(root->left)pretraverseTree(root->left);
124   if(root->right)pretraverseTree(root->right);
125 }
126
127 void intraverseTree(struct TreeNode *root) {
128   if(root->left)intraverseTree(root->left);
129   printf("%d␣", root->val);
130   if(root->right)intraverseTree(root->right);
131 }
132
133 void posttraverseTree(struct TreeNode *root) {
134   if(root->left)posttraverseTree(root->left);
135   if(root->right)posttraverseTree(root->right);
136   printf("%d␣", root->val);
137 }
138
139 void traverseTree(struct TreeNode *root) {
140   pretraverseTree(root);
141   printf("\n");
142   intraverseTree(root);
143   printf("\n");
144   posttraverseTree(root);
145   printf("\n");
146 }
147
148 int main() {
149   struct TreeNode* root= NULL;
150   printf("Inserting:␣4,␣2,␣3,␣8,␣6,␣7,␣9,␣12,␣1\n");
151   insert(&root, 4);
152   insert(&root, 2);
153   insert(&root, 3);
154   insert(&root, 8);
155   insert(&root, 6);
156   insert(&root, 7);
157   insert(&root, 9);
158   insert(&root, 12);
159   insert(&root, 1);
160   printTree(root);
161   printf("traverse:␣\n");
162   traverseTree(root);
163   printf("Deleting:␣4,␣12,␣2\n");
```

```
164   delete(&root, 4);
165   delete(&root, 12);
166   delete(&root, 2);
167   printTree(root);
168   printf("traverse:␣\n");
169   traverseTree(root);
170
171   return 0;
172 }
```

Task 3

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  struct TreeNode{
5    int val;
6    struct TreeNode* left;
7    struct TreeNode* right;
8  };
9
10 void insert(struct TreeNode** root, int val) {
11   struct TreeNode* newTreeNode = NULL;
12   struct TreeNode* prev = NULL;
13   struct TreeNode* current = *root;
14   newTreeNode = malloc(sizeof(struct TreeNode));
15   newTreeNode->val = val;
16   newTreeNode->left = NULL;
17   newTreeNode->right = NULL;
18   while (current != NULL) {
19     prev = current;
20     if (val < current->val){
21       current = current->left;
22     } else{
23       current = current->right;
24     }
25   }
26   if (prev == NULL) {
27     *root = newTreeNode;
28   } else if (val < prev->val) {
29     prev->left = newTreeNode;
30   } else {
31     prev->right = newTreeNode;
32   }
33 }
34
35 struct list{
36   int sum;
37   int len;
38   int data[20];
39 }list;
40
41 struct list * _lrlp(struct TreeNode *root){
42    if(root->left && root->right){
43        struct list * ll = _lrlp(root->left);
44        struct list * lr = _lrlp(root->right);
45        if(ll->sum > lr->sum){
46            ll->data[ll->len]=root->val;
47            ll->len ++;
48            ll->sum +=root->val;
49            return ll;
```

```
        }
        else{
            lr->data[lr->len]=root->val;
            lr->len ++;
            lr->sum +=root->val;
            return lr;
        }
    }
    else if(root->left){
            struct list * ll = _lrlp(root->left);
            ll->data[ll->len]=root->val;
            ll->len ++;
            ll->sum +=root->val;
            return ll;
    }
    else if(root->right){
            struct list * lr = _lrlp(root->right);
            lr->data[lr->len]=root->val;
            lr->len ++;
            lr->sum +=root->val;
            return lr;
    }
    else{
        struct list *l = (struct list *)malloc(sizeof(struct list));
        l->sum = root->val;
        l->len = 1;
        l-> data[0] = root->val;
        return l;
    }

}



int main() {
  struct TreeNode* root= NULL;
  printf("Inserting: 7, 5, 2, 15, 21, 10, 9, 13\n");
  insert(&root, 7);
  insert(&root, 5);
  insert(&root, 2);
  insert(&root, 15);
  insert(&root, 21);
  insert(&root, 10);
  insert(&root, 9);
  insert(&root, 13);
  struct list * l = _lrlp(root);
  printf("lrlp: ");
  for(int i=(l->len)-1;i>0;i--)printf("%d--",l->data[i]);
  printf("%d ",l->data[0]);
  printf("sum: %d",l->sum);
  return 0;
}
```