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;
  while (current != NULL && current->val != val) {
    if (val < current->val){
        current = current->left;
    } else{
        current = current->right;
    }
  }
  return current;
}

void delete(struct TreeNode** root, int val) {
  struct TreeNode* x = search(*root, val);
  if (x == NULL){ //search did not find an element, hence do nothing.
    return;
  }
  struct TreeNode* u = *root;
  struct TreeNode* prev = NULL; // parent of tree node with value = val
  while (u != x) {
    prev = u;
    if (x->val < u->val){
      u = u->left;
    } else{
      u = u->right;
    }
  }
  // Leaf and root case also handled in the no right or left branch. Since if it's leaf, its null anyway.
  if (u->right == NULL) { // there is no right branch
    if (prev == NULL){ // delete root
      *root = u->left;
    } else if (prev->left == u){ //if it's a left child, make left the new child
      prev->left = u->left;
    } else{
      prev->right = u->left;
    }
  } else if (u->left == NULL) { // there is no left branch
    if (prev == NULL){ // delete root
      *root = u->right;
    } else if (prev->left == u){ //if it's a left child, make right the new child
      prev->left = u->right;
    } else{
      prev->right = u->right;
    }
  } else{
    struct TreeNode* p = x->left;
    struct TreeNode* q = p;
    while (p->right != NULL) { //whilst right is null
      q = p;
      p = p->right;
    }
    if (prev == NULL){ // if we are at root
      *root = p;
    } else if (prev->left == u){ // if its a left child
      prev->left = p;
    } else{ //if its a right child
      prev->right = p;
    }
    p->right = u->right;
    if (q != p) {
      q->right = p->left;
      p->left = u->left;
    }
  }
  free(u);
```

```c
}

void printTreeRecursive(struct TreeNode *root, int level) {
  if (root == NULL)
    return;
  if (root->left != NULL) {
    printf("␣␣%d␣--␣%d␣:␣%d\n", root->val, root->left->val,level);
    printTreeRecursive(root->left,level+1);
  }
  if (root->right != NULL) {
    printf("␣␣%d␣--␣%d␣:␣%d\n", root->val, root->right->val,level);
    printTreeRecursive(root->right,level+1);
  }
}

void printTree(struct TreeNode *root) {
  printf("graph␣g␣{\n");
  printTreeRecursive(root, 1);
  printf("}\n");
}

void pretraverseTree(struct TreeNode *root) {
  printf("%d␣", root->val);
  if(root->left)pretraverseTree(root->left);
  if(root->right)pretraverseTree(root->right);
}

void intraverseTree(struct TreeNode *root) {
  if(root->left)intraverseTree(root->left);
  printf("%d␣", root->val);
  if(root->right)intraverseTree(root->right);
}

void posttraverseTree(struct TreeNode *root) {
  if(root->left)posttraverseTree(root->left);
  if(root->right)posttraverseTree(root->right);
  printf("%d␣", root->val);
}

void traverseTree(struct TreeNode *root) {
  pretraverseTree(root);
  printf("\n");
  intraverseTree(root);
  printf("\n");
  posttraverseTree(root);
  printf("\n");
}

int main() {
  struct TreeNode* root= NULL;
  printf("Inserting:␣4,␣2,␣3,␣8,␣6,␣7,␣9,␣12,␣1\n");
  insert(&root, 4);
  insert(&root, 2);
  insert(&root, 3);
  insert(&root, 8);
  insert(&root, 6);
  insert(&root, 7);
  insert(&root, 9);
  insert(&root, 12);
  insert(&root, 1);
  printTree(root);
  printf("traverse:␣\n");
  traverseTree(root);
  printf("Deleting:␣4,␣12,␣2\n");
  delete(&root, 4);
  delete(&root, 12);
  delete(&root, 2);
  printTree(root);
  printf("traverse:␣\n");
  traverseTree(root);

  return 0;
}
```

Task 3

```
#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 list{
  int sum;
  int len;
  int data[20];
}list;

struct list * _lrlp(struct TreeNode *root){
    if(root->left && root->right){
        struct list * ll = _lrlp(root->left);
        struct list * lr = _lrlp(root->right);
        if(ll->sum > lr->sum){
            ll->data[ll->len]=root->val;
            ll->len ++;
            ll->sum +=root->val;
            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;
}
```