浙江科技學院

Rapport de laboratoire sur PA3

Projet	TD5、TD6
Classe	BD212
Nom	Elie(金卓远)、Henri(叶秉文)
ID	1211024054、1210514066

Professeur MinRui

2023年 12 月 18 日

Rapport de PA3

一、 Préparation

a) Définir une structure TreeNode comme un nœud de l'arbre

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

typedef int typeElem;

typedef struct TreeNode {
    typeElem value;
    struct TreeNode *left;
    struct TreeNode *right;
} TreeNode;
```

b) Création d' un nouveau nœud

```c
TreeNode *consa(typeElem value, arbre leftTree, arbre rightTree) {
    TreeNode newNode;
    newNode.value = value;
    newNode.left = leftTree;
    newNode.right = rightTree;
    return &newNode;
}
```

c) Accès au sous-arbre gauche

```c
// Accès au sous-arbre gauche
TreeNode *left(arbre tree) {
    return tree->left;
}
```

d) Accès au sous-arbre droit

```c
// Accès au sous-arbre droit
TreeNode *right(arbre tree) {
    return tree->right;
}
```

e) Déterminer si l'arbre est vide

```c
bool isEmptyTree(arbre tree) {
    if (tree == NULL) {
        return true;
    } else {
        return false;
    }

}
```

f) Donnez la racine de l'arbre

```
typeElem root(arbre tree) {
    return tree->value;
}
```

g) Afficher l'arbre

i. Préfixe

```
// Affichage en préfixe
void prefixe(arbre tree) {
    if (!isEmptyTree(tree)) {
        printf("%d ", tree->value);
        prefixe( tree: tree->left);
        prefixe( tree: tree->right);
    } else {
        printf("null ");
    }
}
```

ii. Infixe

```
void infix(arbre tree) {
    if (!isEmptyTree(tree)) {
        infix( tree: tree->left);
        printf("%d ", tree->value);
        infix( tree: tree->right);
    } else {
        printf("null ");
    }
}
```

iii. Postfixe

```
// Affichage en postfixe
void postfixe(arbre tree) {
    if (!isEmptyTree(tree)) {
        postfixe( tree: tree->left);
        postfixe( tree: tree->right);
        printf("%d ", tree->value);
    } else {
        printf("null ");
    }
}
```

h) Afficher null pour les “enfants vides”

```
bool EnfantsVide(TreeNode* root){
    if(root->left==NULL || root->right==NULL)
        return true;
    else
        printf("null\n");return true;
}
```

Les autres fonctions:

Création d’un arbre binaire à partir d’un tableau

```
/// 通过数组创建二叉树
///index 初始值输入0
TreeNode *createTreeFromArray(int arr[], int index, int size) {
    TreeNode *root = NULL;

    // 判断当前索引是否在数组范围内
    if (index < size) {
        // 使用-1表示null
        if (arr[index] == -1) {
            return NULL;
        }

        root = createTreeNode( value: arr[index]);

        // 递归创建左子树
        root->left = createTreeFromArray(arr, index: 2 * index + 1, size);

        // 递归创建右子树
        root->right = createTreeFromArray(arr, index: 2 * index + 2, size);
    }

    return root;
}
```

Mise en place d’une file d’attente

```c
///---------------------------------
//Queue

// 定义队列结构
typedef struct QueueNode {
    struct TreeNode *data;
    struct QueueNode *next;
} QueueNode;

typedef struct Queue {
    struct QueueNode *front;
    struct QueueNode *rear;
} Queue;

// 初始化队列
void initQueue(Queue *q) {
    q->front = NULL;
    q->rear = NULL;
}

// 入队操作
void enqueue(Queue *q, TreeNode *node) {
    QueueNode *newNode = (QueueNode *) malloc(sizeof(QueueNode));
    newNode->data = node;
    newNode->next = NULL;

    if (q->rear == NULL) {
        q->front = newNode;
        q->rear = newNode;
    } else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
}

// 出队操作
TreeNode *dequeue(Queue *q) {
    if (q->front == NULL) {
        return NULL;
    }

    QueueNode *temp = q->front;
    TreeNode *data = temp->data;

    q->front = q->front->next;

    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return data;
}
```

Parcours BFS（Parcours en largeur）

```
// 判断队列里的值是否全是NULL
bool areAllValuesNull(struct Queue* q) {
    struct QueueNode* current = q->front;

    while (current != NULL) {
        if (current->data != NULL) {
            // 如果当前节点的数据不是NULL，则队列中存在非NULL值
            return false;
        }
        current = current->next;
    }

    // 遍历完成，所有值都是NULL
    return true;
}

// 主函数：广度优先搜索二叉树
void BFS(TreeNode *root) {
    if (root == NULL) {
        printf("null");
        return;
    }

    Queue q;
    initQueue(&q);

    enqueue(&q, node: root);

    while (q.front!=NULL) {
        struct TreeNode* current = dequeue(&q);

        if (current == NULL) {
            // 如果当前节点为 "null"，但队列中还有未处理的非 "null" 节点，继续打印
            if(!areAllValuesNull(&q))
                printf("null ");
        } else {
            printf("%d ", current->value);
            enqueue(&q, node: current->left);
            enqueue(&q, node: current->right);
        }
    }
}
```

Exemple: pour l'arbre démontré dans Figure 2

la fonction de parcours préfixe doit afficher

```
préfix:  5, 1, "null", "null", 4, 3, "null", "null", 6, "null", null"
```

```
void Ex1() {

    int root_f2[] = { [0]: 5,  [1]: 1,  [2]: 4,  [3]: -1,  [4]: -1,  [5]: 3,  [6]: 6}; // 使用-1表示null
    int size = sizeof(root_f2) / sizeof(root_f2[0]);
    TreeNode *Arbre_F2 = createTreeFromArray( arr: root_f2,  index: 0, size);
    prefixe( tree: Arbre_F2);
    /**
     * result: 5 1 null null 4 3 null null 6 null null
     */

}
```

Résulta:

```
D:\King\大三上\程序与算法3\devoir\TD5\cmake-build-debug\TD5.exe
5 1 null null 4 3 null null 6 null null
进程已结束，退出代码为 0
```

二、 Exercice2

Parce que l'infixe d'un arbre binaire trié correct est ascendant. Par conséquent, nous utilisons cette méthode pour vérifier que l'arbre de tri binaire est correct.

La fonction du code suivant est de parcourir un arbre binaire dans l'infixe et de stocker le résultat dans le tableau « res »

```
// 中序遍历二叉树
void infixetoArr(struct TreeNode* node, int* res, int* index) {
    if (node == NULL) return;

    infixetoArr( node: node->left, res, index);
    res[(*index)++] = node->value;
    infixetoArr( node: node->right, res, index);
}
```

le pseudo-code :

fonction isValidBST(root Arbre) :vrai ou faux

début

si root=NULL alors

retourner vrai ;

//Pour obtenir le nombre de nœuds dans l'arbre

nombreDeNoeud <- 0 ;

//créer un tableau pour mettre le résultat d'infixe

A<-tableau_vide ;

//Parcours en infixe et stockage des résultats dans un tableau

infixetoArr(root, A, nombreDeNoeude) ;

//Vérifier si le tableau est trié(si précédent plus suivant alors faux)

Pour i<- 1 au nombreDeNoeud faire

Si A[i]<=A[i-1] alors

Libérer A ;

Retourner faux ;

Fin si

Fin pour

Libérer A ;

Retourner vrai ;

Fin

```
// 验证二叉搜索树
bool isValidBST(struct TreeNode* root) {
    if (root == NULL) return true;

    // 获取树的节点数量
    int nodeCount = 0;
    int* result = (int*)malloc(sizeof(int) * 1000); // 假设树最多有1000个节点

    // 中序遍历树并将结果存入数组
    infixetoArr( node: root,  res: result,  index: &nodeCount);

    // 验证数组是否有序
    for (int i = 1; i < nodeCount; i++) {
        if (result[i] <= result[i - 1]) {
            free(result);
            return false;
        }
    }

    free(result);
    return true;
}
```

```c
//afficher le tab
void printArr(int arr[],int size){
    printf("Entreé: [");
    for (int i = 0; i < size; i++) {
        if (arr[i] == -1)
            printf("null,");
        else
            printf("%d,", arr[i]);
    }
    printf("\b]\n");
}
```

```c
///Ex2 Valider un arbre binaire de recherche (ABR)
void Ex2() {
    int root_f1[] = { [0]: 2, [1]: 1, [2]: 3};
    int size1 = sizeof(root_f1) / sizeof(root_f1[0]);
    TreeNode *F1=createTreeFromArray( arr: root_f1, index: 0, size: size1);

    printArr( arr: root_f1, size: size1);
    if (isValidBST( root: F1))
        printf("Sortie: true\n");
    else
        printf("Sortie: false\n");

    printf("------------------------------\n");

    int root_f2[] = { [0]: 5, [1]: 1, [2]: 4, [3]: -1, [4]: -1, [5]: 3, [6]: 6}; // 使用-1表示null
    int size2 = sizeof(root_f2) / sizeof(root_f2[0]);
    TreeNode *F2=createTreeFromArray( arr: root_f2, index: 0, size: size2);

    printArr( arr: root_f2, size: size2);
    if (isValidBST( root: F2))
        printf("Sortie: true\n");
    else
        printf("Sortie: false\n");
}
```

Résultat :

```
D:\King\大三上\程序与算法3\devoir\TD5\cmake-build-debug\TD5.exe
Entreé: [2,1,3]
Sortie: true
------------------------------
Entreé: [5,1,4,null,null,3,6]
Sortie: false
```

三、 Exercice3

Récupérer l’arbre binaire de recherche

D'après la question, seuls deux nœuds ont été échangés de manière incorrecte. J'ai utilisé le parcours de la racine centrale pour trouver les deux nœuds qui ont été échangés de manière incorrecte et je les ai échangés à nouveau.

le pseudo-code :

```
Fonction recoverTree(root Arbre) :vide
Début
    // Créer un tableau de nœuds pour enregistrer deux échanges d'erreurs
    Err[2]<- Tableau_Vide

    Si Arbre est vide alors
        Retourner ;
    Fin si

    //Pour enregistrer le nœud précédent
    Pre <- Nœud_Vide

    //définir une pile
    Stack {Data Tableau,top Entier};
    //initialisation de la pile
    Top de Stack <- -1;

    Tant que stack.top !=-1 et root != NULL faire
        Si root != NULL alors
        // Mettre le nœud courant sur la pile et se déplacer vers le
        sous-arbre de gauche
            Data<- root ;
            Left(root) ;
        Sinon
        // Sauter le nœud supérieur de la pile et vérifier s'il y a des
        erreurs
            Root<- Data[top] ;
            Si pre != NULL et value de pre > value de root alors
                // Enregistrer le premier nœud d'erreur
                Err[0]<- le premier nœud d'erreur
                // Enregistrer le deuxième nœud d'erreur
                Err[1]<- le deuxième nœud d'erreur
            Fin si
```

```
            // Mise à jour du nœud précédent au nœud actuel
            Pre<-root ;
            // Passer au sous-arbre de droite
            Right(root) ;
        Fin si
    //Échanger le mauvais nœud
    Tmp<-Err[0] ;
    Err[0]<-Err[1] ;
    Err[1]<-tmp ;
Fin
```

Rapport de PA3

```
//
// Created by 小金 on 2023/12/15.
//

#ifndef TD5_EX3_TT_H
#define TD5_EX3_TT_H
#include <stdio.h>
#include <stdlib.h>
#include "Operation_de_base.h"

// 恢复错误的二叉搜索树
void recoverTree(TreeNode* root) {
    // 创建一个数组用于存放两个错误交换的节点
    TreeNode* err[2] = { [0]: NULL, [1]: NULL};
    // 如果根节点为空，直接返回
    if (root == NULL) return;
    // 用于迭代的前一个节点
    TreeNode* pre = NULL;
    // 定义一个栈结构
    struct Stack {
        TreeNode* data[1000]; // 假设树最多有1000个节点
        int top;
    } stack;
    // 初始化栈
    stack.top = -1;
    // 中序遍历二叉树
    while (stack.top != -1 || root != NULL) {
        if (root != NULL) {
            // 将当前节点入栈，并移至左子树
            stack.data[++stack.top] = root;
//            root = root->left;
            root=left( tree: root);
        } else {
            // 弹出栈顶节点，并检查是否存在错误
            root = stack.data[stack.top--];
            if (pre != NULL && pre->value > root->value) {
                // 记录第一个错误的节点
                err[0] = (err[0] == NULL) ? pre : err[0];
                // 记录第二个错误的节点
                err[1] = root;
            }
            // 更新前一个节点为当前节点
            pre = root;
            // 移至右子树
//            root = root->right;
            root=right( tree: root);
        }
    }
    // 交换错误的节点值
    int tmp = err[0]->value;
    err[0]->value = err[1]->value;
    err[1]->value = tmp;
}
#endif //TD5_EX3_TT_H
```

Rapport de PA3

```
void Ex3(){
    int root_f3[] = { [0]: 1,  [1]: 3,  [2]: -1,  [3]: -1,  [4]: 2};
    int size3 = sizeof(root_f3) / sizeof(root_f3[0]);
    TreeNode *F3 = createTreeFromArray( arr: root_f3,  index: 0,  size: size3);

    int root_f4[] = { [0]: 3,  [1]: 1,  [2]: 4,  [3]: -1,  [4]: -1,  [5]: 2}; // 使用-1表示null
    int size4 = sizeof(root_f4) / sizeof(root_f4[0]);
    TreeNode *F4= createTreeFromArray( arr: root_f4,  index: 0,  size: size4);

    //3.1
    printf("(1)\n");
    printf("Input: root = [ ");
    BFS( root: F3);
    printf("]\n");
    recoverTree( root: F3);
    printf("Output: [ ");
    BFS( root: F3);
    printf("]\n");

    printf("--------------------------\n");

    printf("(2)\n");
    printf("Input: root = [ ");
    BFS( root: F4);
    printf("]\n");
    recoverTree( root: F4);
    printf("Output: [ ");
    BFS( root: F4);
    printf("]\n");
}
```

Résultat :

```
D:\King\大三上\程序与算法3\devoir\TD5\cmake-build-debug\TD5.exe
(1)
Input: root = [ 1 3 null null 2 ]
Output: [ 3 1 null null 2 ]
--------------------------
(2)
Input: root = [ 3 1 4 null null 2 ]
Output: [ 2 1 4 null null 3 ]
```

四、 Exercice4

Construction d’un arbre binaire à partir des parcours en préfixe et en infixe

le pseudo-code :

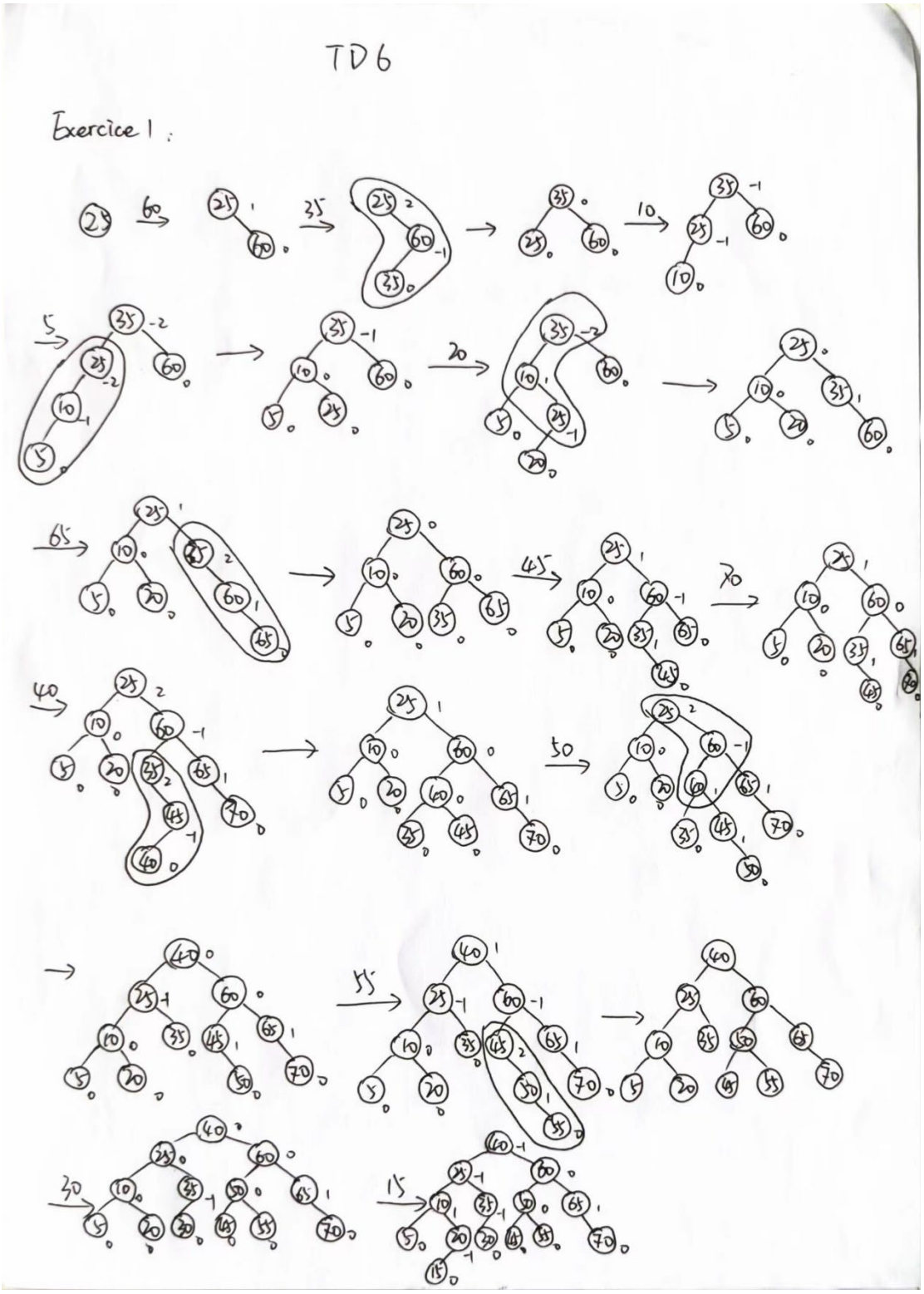
```
Fonction buildTree_prefixe_infixe(prefixe Tableau,infixe Tableau) :Arbre
    Début
        Si taille de préfixe est 0 alors
            Retourner Arbre vide ;
        Fin si

        // Créer le nœud racine et le placer sur la pile
        Root<-le premier élément de préfixe tableau
        Ajouter root au stack ;
        // Trouver la position du nœud actuel dans un tableau de traversée d'ordre moyen
        infixeIndex<-0 ;

        //Parcours du tableau en préfix (à partir du deuxième nœud)
        Pour i<-1 à taille de préfixe tableau faire
            Value_prefixe<-prefixe[i]
            Node <- le dernière élément de stack
            // Si la valeur du nœud actuel n'est pas égale à la valeur du tableau dinfixe
            Si value_node != infixe[infixeIndex] alors
                // Créer le nœud enfant de gauche et le placer sur la pile
                Node.left<-value_prefixe ;
                Ajouter Node.left au stack ;
            Sinon
            //Si la valeur du nœud actuel est égale à la valeur du tableau d’infixe
            // Retirer le nœud supérieur de la pile à tour de rôle jusqu'à ce que la pile soit vide ou que la valeur du nœud supérieur de la pile ne soit pas égale à la valeur du tableau de traversée d'infixe
                Tant que stack n’est pas vide et l’élément de stack =infixe[infixeIndex] faire
                Node<-le dernière de stack ;( Retirer l'élément de la pile)
                infixeIndex<-infixeIndex+1 ;

                //Créer un nœud enfant droit et le placer sur la pile
                Node.right<-value_préfixe;
                Ajouter node.right au stack ;
            Fin si
        Fin pour
        Retourner root ;
    Fin
```

Rapport de PA3



Rapport de PA3

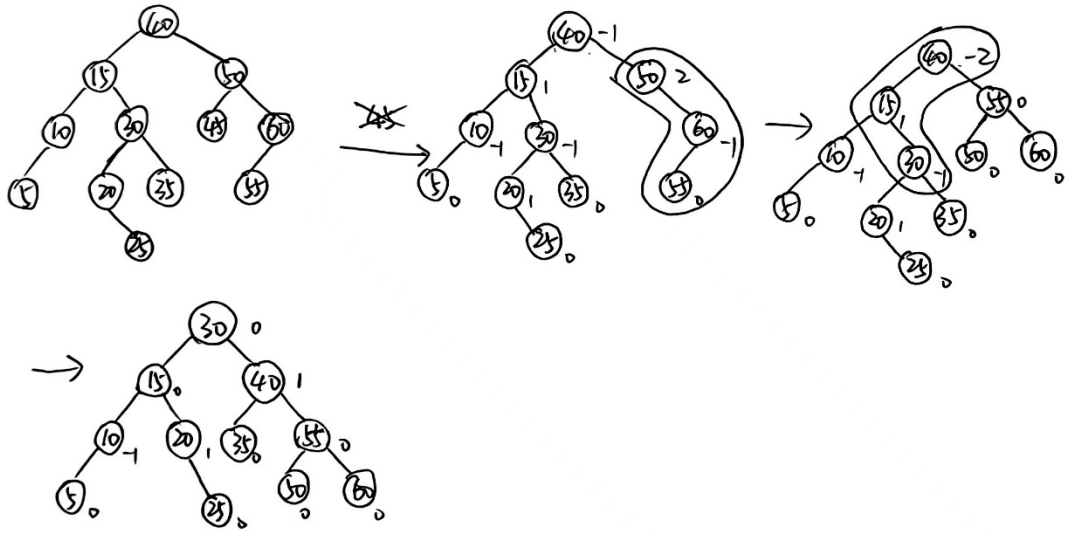
Exercice 2:

infixe : 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70

Il est en ordre croissant.

Non, juste quand Il est un arbre binaire recherche.

Exercice 3.



Exercice 4.