SAKHO OUSSEYNOU
PABOEUF ANTHONY

Structure de données TP6

Table des matières

SAKHO OUSSEYNOU
PABOEUF ANTHONY

Exercice 1

Algorithmes

```
fonction insertTree(noeud_t *ptree, int val) : noeud_t*
début

    si ptree n'est pas null alors
        si ptree.val > result.val alors
            insertTree(ptree.fils_gauche,val)
        sinon si ptree.val < result.val alors
            insertTree(ptree.fils_droit,val)
            // sinon alors il y a égalité, donc pas besoin de nouvelle node
            fin_si
        fin_si
    sinon
        noeud_t* ptree = espace mémoire de taille noeud_t
        ptree.val = val
    fin_si

    retourner result
fin

fonction inorderTree(noeud_t *ptree) : void
début
    si ptree n'est pas null alors
        inorderTree(ptree.fils_gauche)
        ecrire(ptree.val)
        inorderTree(ptree.fils_droit)
    fin_si
fin

fonction preorderTree(noeud_t *ptree) : void
début
    si ptree n'est pas null alors
        ecrire(ptree)
        preorderTree(ptree.fils_gauche)
        preorderTree(ptree.fils_droit)
    fin_si
fin

fonction postorderTree(noeud_t *ptree) : void
début
    si ptree n'est pas null alors
        postorderTree(ptree.fils_gauche)
        postorderTree(ptree.fils_droit)
        ecrire(ptree)
    fin_si
fin
```

```
fonction breadthTree(noeud_t *ptree) : void
début
    si ptree n'est pas null alors
        ecrire(ptree)
        si ptree.fils_gauche != NULL alors ecrire(ptree.fils_gauche)
        si ptree.fils_droit != NULL alors ecrire(ptree.fils_droit)
        reccu_breadthTree(ptree.fils_gauche)
        reccu_breadthTree(ptree.fils_droit)
    fin_si
fin

fonction reccu_breadthTree(noeud_t *ptree) : void
début // on n'écrit pas la node en question contrairement au début
      // pour le sommet
    si ptree n'est pas null alors
        si ptree.fils_gauche != NULL alors ecrire(ptree.fils_gauche)
        si ptree.fils_droit != NULL alors ecrire(ptree.fils_droit)
        reccu_breadthTree(ptree.fils_gauche)
        reccu_breadthTree(ptree.fils_droit)
    fin_si
fin
fonction maxTree(noeud_t *ptree) : int
début
    noeud_t* p_temp = ptree;
    si p_temp n'est pas null alors

        tant que p_temp.fils_gauche n'est pas null faire
            p_temp = p_temp.fils_gauche
        fin_tant_que
    sinon
        p_temp.val = -1
    fin_si
    retourner p_temp.val
fin

fonction minTree(noeud_t *ptree) : int
début
    noeud_t* p_temp = ptree;
    si p_temp n'est pas null alors

        tant que p_temp.fils_droit n'est pas null faire
            p_temp = p_temp.fils_droit
        fin_tant_que
    sinon
        p_temp.val = -1
    fin_si
    retourner p_temp.val
```

```
fin

fonction heightTree(noeud_t *ptree) : int
début
    si ptree n'est pas null alors

        int val_fils_gauche = 1 + heightTree(ptree.fils_gauche)
        int val_fils_droit = 1 + heightTree(ptree.fils_droit)

        si val_fils_gauche >= val_fils_droit alors
            retourner val_fils_gauche
        sinon
            retourner val_fils_droit
        fin_si
    sinon
        retourner 0
    fin_si
fin
fonction nbNodesTree(noeud_t *ptree) : int
début
    si ptree n'est pas null alors

        int val_fils_gauche = 1 + nbNodesTree(ptree.fils_gauche)
        int val_fils_droit = 1 + nbNodesTree(ptree.fils_droit)

        retourner val_fils_droit + val_fils_gauche
    sinon
        retourner 0
    fin_si
fin
fonction searchTree(noeud_t *ptree, int val) : noeud_t*
début
    noeud_t* res;
    si ptree n'est pas null alors
        si ptree.val == val alors
            res = ptree
        fin_si
        res = searchTree(ptree.fils_gauche,val)
        res = searchTree(ptree.fils_droit,val)
    fin_si

    retourner res
fin
fonction removeTree(noeud_t *ptree, int val) : noeud_t*
début
    noeud_t p_temp = searchTree(ptree,val)
    // impossible de retrouver le prédécesseur dans l'arbre
    // avec une méthode réccursive ici
```

```
    retourner p_temp
fin
```

Code C

```c
#include "arbre.h"

void *insertTree(noeud_t *ptree, int val) {

    if (ptree != NULL)
    {
        if (ptree->valeur > val)
        {
            insertTree(ptree->fils_gauche,val);
        }
        else if (ptree->valeur < val)
        {
            insertTree(ptree->fils_droit,val);
        }
    } else {
        ptree = malloc(sizeof(noeud_t));
        ptree->valeur = val;
    }
}
void inorderTree(noeud_t *ptree) {
    if (ptree != NULL) {
        inorderTree(ptree->fils_gauche);
        printf("%d /",ptree->valeur);
        inorderTree(ptree->fils_gauche);
    }
}
void preorderTree(noeud_t *ptree) {
    if (ptree != NULL) {
        printf("%d /",ptree->valeur);
        preorderTree(ptree->fils_gauche);
        preorderTree(ptree->fils_gauche);
    }
}
void postorderTree(noeud_t *ptree) {
    if (ptree != NULL) {
        postorderTree(ptree->fils_gauche);
        postorderTree(ptree->fils_gauche);
        printf("%d /",ptree->valeur);
    }
}
void breadthTree(noeud_t *ptree) {
    if (ptree != NULL) {
        printf("%d /", ptree->valeur);
```

```
        if (ptree->fils_gauche != NULL) {
            noeud_t* fg = ptree->fils_gauche;
            printf("%d /", fg->valeur);
        }
        if (ptree->fils_droit != NULL) {
            noeud_t* fd = ptree->fils_droit;
            printf("%d /", fd->valeur);
        }
        reccu_breadthTree(ptree->fils_gauche);
        reccu_breadthTree(ptree->fils_droit);
    }
}

void reccu_breadthTree(noeud_t *ptree) {
    if (ptree != NULL) {
        if (ptree->fils_gauche != NULL) {
            noeud_t* fg = ptree->fils_gauche;
            printf("%d /", fg->valeur);
        }
        if (ptree->fils_droit != NULL) {
            noeud_t* fd = ptree->fils_droit;
            printf("%d /", fd->valeur);
        }
        reccu_breadthTree(ptree->fils_gauche);
        reccu_breadthTree(ptree->fils_droit);
    }
}
int maxTree(noeud_t *ptree) {
    noeud_t* p_temp = ptree;
    if (p_temp != NULL) {
        while (p_temp->fils_gauche != NULL)
        {
            p_temp = p_temp->fils_gauche;
        }
    } else {
        p_temp->valeur = -1;
    }

    return p_temp->valeur;
}

int minTree(noeud_t *ptree) {
    noeud_t* p_temp = ptree;
    if (p_temp != NULL) {
        while (p_temp->fils_droit != NULL)
        {
            p_temp = p_temp->fils_droit;
        }
```

```
    } else {
        p_temp->valeur = -1;
    }

    return p_temp->valeur;
}
int heightTree(noeud_t *ptree) {
    if (ptree != NULL) {
        int val_fils_gauche = 1 + heightTree(ptree->fils_gauche);
        int val_fils_droit = 1 + heightTree(ptree->fils_droit);

        if (val_fils_gauche >= val_fils_droit) {
            return val_fils_gauche;
        } else {
            return val_fils_droit;
        }
    } else {
        return 0;
    }
}
int nbNodesTree(noeud_t *ptree) {
    if (ptree != NULL) {
        int val_fils_gauche = 1 + nbNodesTree(ptree->fils_gauche);
        int val_fils_droit = 1 + nbNodesTree(ptree->fils_droit);

        return val_fils_gauche + val_fils_droit;
    } else {
        return 0;
    }
}
noeud_t *searchTree(noeud_t *ptree, int val) {
    noeud_t* res;
    if (ptree != NULL) {
        if (ptree->valeur == val) {
            res = ptree;
        }
        res = searchTree(ptree->fils_gauche,val);
        res = searchTree(ptree->fils_droit,val);
    }

    return res;
}
noeud_t *removeTree(noeud_t *ptree, int val) ; // remove val from tree and
return the new tree

int main() {
    noeud_t *myTree = NULL; // empty tree
    insertTree(myTree, 50);
```

```
    insertTree(myTree, 45);
    insertTree(myTree, 65);
    insertTree(myTree, 55);
    insertTree(myTree, 54);
    insertTree(myTree, 56);
    insertTree(myTree, 80);
    insertTree(myTree, 70);
    insertTree(myTree, 85);
    insertTree(myTree, 30);
    insertTree(myTree, 47);


    inorderTree(myTree, 0);
    preorderTree(myTree, 0);
    postorderTree(myTree, 0);

    printf("max = %d\n", maxTree(myTree));
    printf("min = %d\n", minTree(myTree));
    printf("nb of nodes = %d\n", nbNodesTree(myTree));
    printf("height tree = %d\n\n", heightTree(myTree));
    breadthTree(myTree);
    printf("search for 55 = %d\n", searchTree(myTree, 55)->valeur);
    noeud_t *pnd = searchTree(myTree, 77);
    printf("search for 77 = %p\n", pnd);
    myTree = removeTree(myTree, 65) ;
}
```