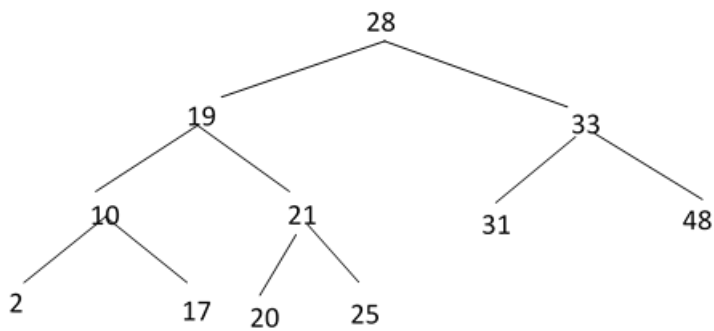


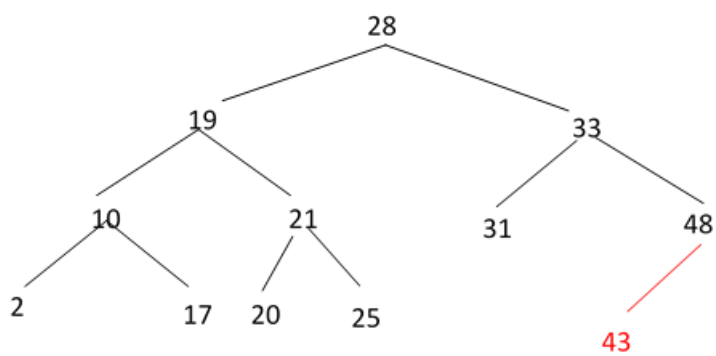
## Corrigé du test de contrôle

### Exercice 1 : (5 pts)

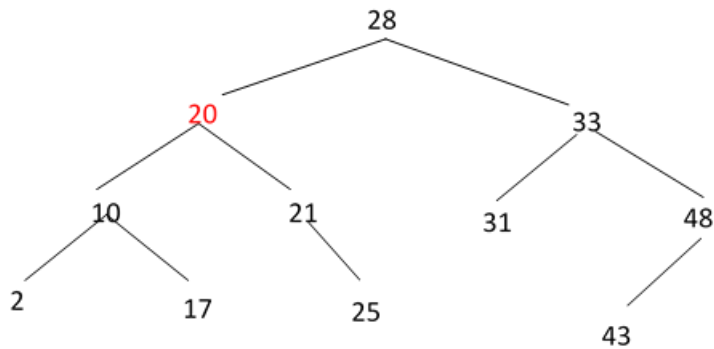
Considérer l'arbre binaire de recherche suivant :



- 1) insérer le nombre 43 dans l'arbre.



- 2) Ensuite supprimer le nombre 19 de l'arbre.



3) Ecrire un algorithme de suppression d'un élément d'un arbre binaire de recherche.

L'algorithme passe par 4 étapes qui sont :

1. Recherche de la valeur  $x$  dans l'arbre.
2. Recherche de la valeur minimale du sous-arbre droit du nœud contenant  $x$  ou bien de la valeur maximale du sous-arbre gauche du nœud contenant  $x$ . Soit  $y$ , cette valeur.
3. Suppression du nœud contenant  $y$ .
4. Remplacer  $x$  par  $y$ .

---

### **Algorithme Suppression d'un élément d'un arbre binaire de recherche**

#### **Programme principal**

**var** racine : ↑élément ; a : entier ;

**début** suppression (a) ; **fin**

**procédure** suppression ;

**entrée** : racine : ↑élément ; x : entier ;

**sortie** : l'arbre binaire de recherche sans le nœud contenant x ;

**var** nœud, p, prédécesseur, parent : ↑élément ; existe : booléen ; y : entier ;

**début** nœud := racine ;

existe := faux ;

**tant que** (nœud ≠ nil) et non existe **faire**

  (\* recherche de la clé x dans l'arbre \*)

**si** (x = nœud↑.clé) **alors** existe := vrai

**sinon début** prédécesseur := nœud ;

**si** (x < nœud↑.clé) **alors** nœud := nœud↑.fgauche

**sinon** nœud := nœud↑.fdroit ;

**fin** ;

**si** (nœud = nil) **alors** signaler ('x n'existe pas dans l'arbre')

**sinon si** (x = nœud↑.clé) **alors**

**début**

      p := nœud↑.fdroit ;

      (\* recherche de la clé minimale du sous-arbre droit \*)

**si** (p ≠ nil) **alors tant que** (p↑.fgauche ≠ nil) **faire**

**début** parent := p ;

        p := p↑.fgauche ;

**fin** ;

      (\* recherche de la clé maximale du sous-arbre gauche \*)

**sinon début** p := nœud↑.fgauche ;

```

    si (p ≠ nil) alors tant que (p↑.fdroit ≠ nil) faire
        début parent := p ;
            p := p↑.fdroit ;
        fin ;
    sinon (* le nœud contenant x n'a pas de successeur alors
suppression de ce nœud *)
        si (nœud = racine) alors racine := nil
        sinon prédecesseur↑.fdroit := nil ;
    fin ;
si (nœud↑.fgauche ≠ nil) ou (nœud↑.fdroit ≠ nil) alors
    début (* remplacer x par y *)
        y := p↑.clé ;
        nœud↑.clé := y ;
        (* supprimer le nœud contenant y *)
    si (p↑.fgauche = nil) alors parent↑.fgauche := p↑.fdroit
    sinon parent↑.fdroit := p↑.fgauche ;
    fin
fin ;
fin ;

```

---

4) Calculer la complexité de l'algorithme.

La recherche de l'élément  $x$  à supprimer se fait en  $O(\log_2(n))=p$  où  $p$  est la profondeur de l'arbre et  $n$  le nombre de nœuds. L'opération de recherche de l'élément du sous-arbre droit qui a la plus petite valeur pour remplacer  $x$  s'effectue aussi en  $O(\log_2(n))$ , de même pour la recherche de la clé maximale du sous-arbre gauche. La complexité est donc  $O(\log_2(n))$ .

### Exercice 2 : (5 pts)

1) Rappeler la représentation d'un ensemble  $S$  vue en cours.

Une représentation possible d'un ensemble  $S$ , qui facilite l'écriture des opérations union et intersection entre deux ensembles, utilise deux tableaux TV (Tableau des Valeurs) et VR (Vecteur Représentatif de l'ensemble  $S$ ). Le tableau TV contient toutes les valeurs susceptibles d'appartenir à un ensemble  $S$ .  $TV[max]$  contient le nombre de valeurs de TV. VR, le vecteur représentatif de  $S$ , est un vecteur de nombres booléens défini comme suit :

$$VR[i] = \begin{cases} 1 & \text{si } TV[i] \in S \\ 0 & \text{sinon} \end{cases}$$

2) Comment implémenter les opérations suivantes :

a. Appartenance d'un élément à un ensemble.

---

*Procédure appartenance ;*

*entrée : TV : tableau d'entiers ; VR : tableau de booléens ; x : entier ;*

*sortie : vrai ou faux ;*

*var p : entier ;*

*début p := 1 ;*

*tant que (p ≤ TV[max]) et (TV[p] ≠ x) faire p := p + 1 ;*

*si (TV[p] = x) et (VR[p] = 1) alors retourner(vrai)*

*sinon retourner(faux) ;*

*fin*

b. Union de deux ensembles.

---

*procédure union ;*

*entrée : TV : tableau d'entiers ; VR<sub>1</sub>, VR<sub>2</sub> : tableau de booléens ;*

*sortie : VR<sub>3</sub> : tableau de booléens;*

*var i : 1 ..max ;*

*début*

*pour (i := 1 à TV[max]) faire*

*VR<sub>3</sub>[i] := VR<sub>1</sub>[i] ou VR<sub>2</sub>[i] ;*

*(\* ou est l'opérateur booléen de disjonction \*)*

*fin ;*

---

c . Intersection de deux ensembles.

---

*procédure intersection ;*

*entrée : TV : tableau d'entiers ; VR<sub>1</sub>, VR<sub>2</sub> : tableau de booléens ;*

*sortie : VR<sub>3</sub> : tableau de booléens;*

*var i : 1 ..max ;*

*début*

*pour (i := 1 à TV[max]) faire*

*VR<sub>3</sub>[i] := VR<sub>1</sub>[i] et VR<sub>2</sub>[i] ;*

*(\* et est l'opérateur booléen de conjonction \*)*

*fin ;*

---

Calculer la complexité de chacune de ces opérations

La complexité de la procédure appartenance est  $O(TV[max])$  car au pire cas,  $x$  n'appartient pas à TV et la boucle aura un nombre d'itérations égal à  $TV[max]$ .

Les procédures *union* et *intersection* ont chacune une complexité en  $O(TV[max])$  car la boucle *pour*, a un nombre d'itérations égal à  $TV[max]$ .

3) Ecrire la procédure d'insertion d'un élément dans un ensemble et calculer sa complexité.

---

*Procédure insertion ;*

*entrée : TV : tableau d'entiers ; VR : tableau de booléens ; x : entier ;*

*sortie : TV : tableau d'entiers ; VR : tableau de booléens ;*

*var p : entier ;*

*début p := 1;*

*tant que (p <= TV[max]) et (TV[p] ≠ x) faire p := p+1 ;*

*si (TV[p] ≠ x) alors début TV[p] := x ;*

*TV[max] := TV[max] + 1 ;*

*fin ;*

*VR[p] := 1 ;*

*fin*

---

La complexité de la procédure insertion est  $O(TV[max])$  car au pire cas,  $x$  n'appartient pas à  $TV$  et la boucle aura un nombre d'itérations égal à  $TV[max]$ .

4) Ecrire la procédure de suppression d'un élément d'un ensemble et calculer sa complexité.

---

**Procédure suppression ;**

**entrée :  $TV$  : tableau d'entiers ;  $VR$  : tableau de booléens ;  $x$  : entier ;**

**sortie :  $TV$  : tableau d'entiers ;  $VR$  : tableau de booléens ;**

**var  $p$  : entier ;**

**début**       $p := 1$ ;

**tant que** ( $p \leq TV[max]$ ) **et** ( $TV[p] \neq x$ ) **faire**  $p := p + 1$  ;

**si** ( $TV[p] = x$ ) **alors**

**si** ( $VR[p] = 0$ ) **alors** afficher (' $x$  n'existe pas dans l'ensemble')

**sinon**  $VR[p] := 0$  ;

**fin**

---

La complexité de la procédure suppression est  $O(TV[max])$  car au pire cas,  $x$  n'appartient pas à  $TV$  et la boucle aura un nombre d'itérations égal à  $TV[max]$ .