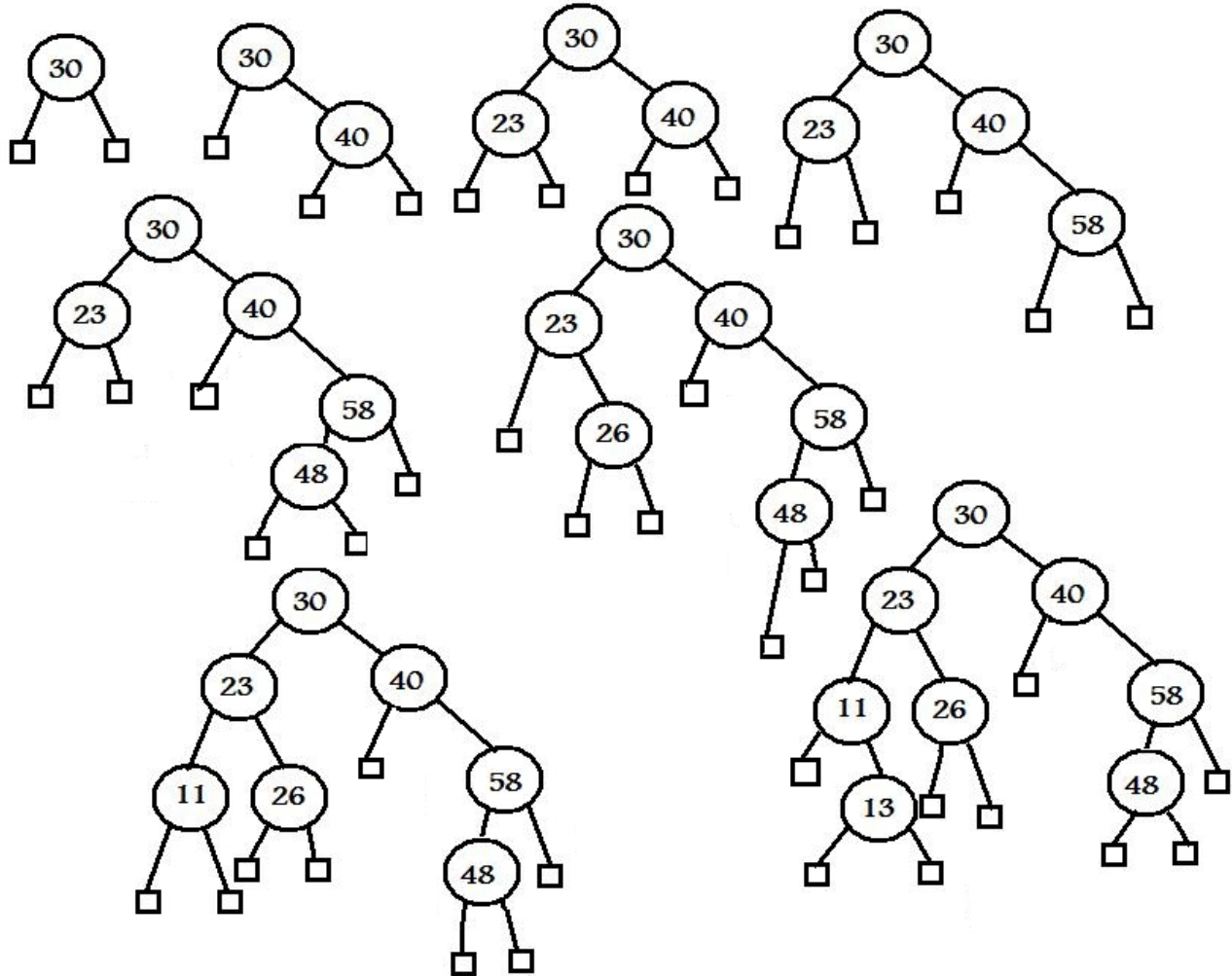


Travaux pratiques: Arbres de recherche

1. a) Insérer dans un arbre binaire de recherche avec les clés: 30, 40, 23, 58, 48, 26, 11,13.

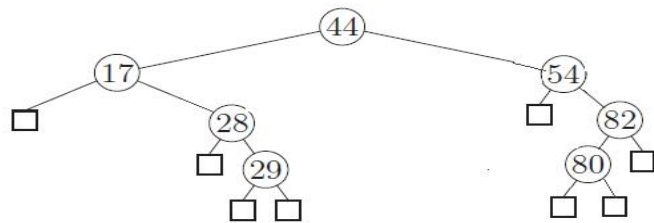
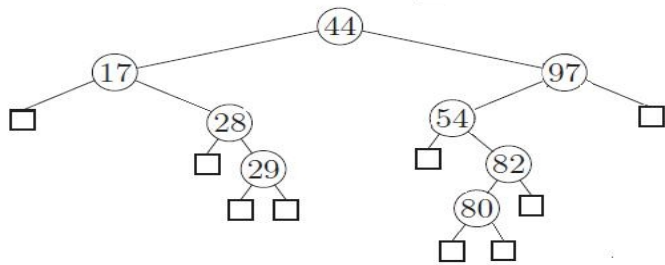
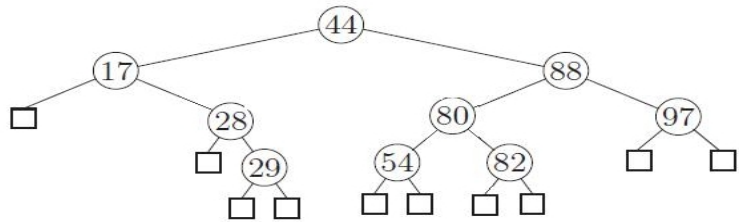
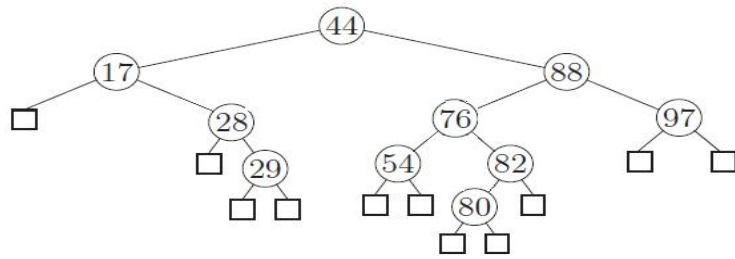
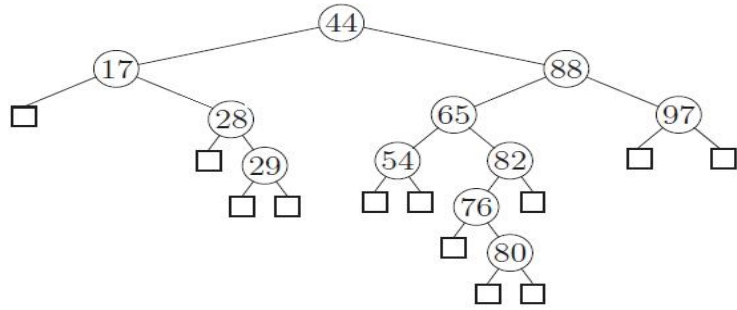
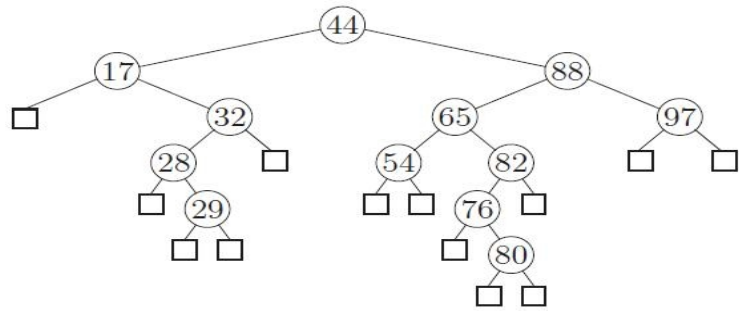


b) Supprimer dans l'arbre binaire A les clés suivantes: 32, 65, 76, 88, 97.

Plusieurs cas sont à considérer, une fois que le nœud à supprimer a été trouvé à partir de sa clé :

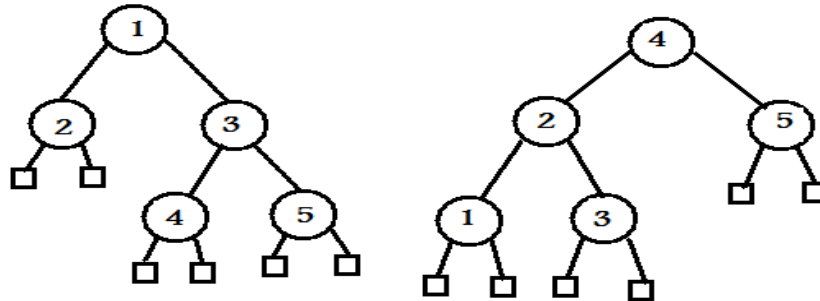
- **Suppression d'une feuille** : Il suffit de l'enlever de l'arbre vu qu'elle n'a pas de fils.
- **Suppression d'un nœud avec un enfant** : Il faut l'enlever de l'arbre en le remplaçant par son fils.
- **Suppression d'un nœud avec deux enfants** : Supposons que le nœud à supprimer soit appelé N . On le remplace alors par son successeur le plus proche (le nœud le plus à gauche du sous-arbre droit) ou son plus proche prédécesseur (le nœud le plus à droite du sous-arbre gauche). Cela permet de garder une structure d'arbre binaire de recherche. Puis on applique à nouveau la procédure de suppression à N , qui est maintenant une feuille ou un nœud avec un seul fils.

RAPPEL: DANS LE CADRE DE CE COURS, NOUS ALLONS LE REMPLACER PAR SON PLUS PROCHE SUCCESSEUR.



- c) Si on essaie d'insérer une séquence d'éléments dans un arbre binaire de recherche initialement vide, dans deux ordres différents, il est possible que l'on obtienne deux arbres différents. Construisez un exemple de ce phénomène avec une séquence d'au moins 5 clés.

Prenons les séquences suivantes composées du même ensemble de clés: 1, 3, 2, 5,4 et 4, 5, 2, 3,1. Nous obtenons deux arbres différents:



2. Voici un algorithme qui détermine si un arbre binaire de recherche de n éléments contient un élément dont la clé est dans un certain intervalle. Les entrées de l'algorithme devraient être un arbre binaire de recherche T et deux clés l et r , avec $l \leq r$. Si T contient au moins un élément de clé k , tel que $l \leq k \leq r$, l'algorithme retourne VRAI, sinon, il retourne FAUX.

Pour cet algorithme, nous allons utiliser les fonctions min et max qui retournent le nœud à la valeur minimale ou maximale d'un sous-arbre.

min (Arbre racine): retourne le min d'un arbre{

 courant = racine

 TANT QUE (courant → gauche != NULL)

 courant = courant → gauche }

 Retourner courant → valeur

}

max (Arbre racine): retourne le max d'un arbre{

 courant = racine

 TANT QUE (courant → droite != NULL) FAIRE

 courant = courant → droite }

 Retourner courant → valeur

}

estDansIntervalle(Arbre racine, l et r des entiers) {

 SI (nœud == NULL) ALORS

 Retourner Faux

 SI (nœud → valeur >= r ET nœud → valeur <= l) ALORS

 Retourner Vrai

 SI (nœud → gauche != NULL ET min (nœud → gauche) <= r

 ET max (nœud → gauche) >= l) ALORS

 estDansIntervalle(nœud → gauche, l, r)

 SINON

 estDansIntervalle(nœud → droit, l, r)

}

3. Arbres binaires de recherche différents dont l'ensemble des clés est :

a) $\{1,2,3\} \rightarrow$ 4 arbres binaires différents selon les ordres d'insertion suivants:

1. 1,2,3 et 3,2,1 donnent le même arbre
2. 1,3,2
3. 2,1,3 et 2,3,1 donnent le même arbre
4. 3,1,2

b) $\{1,2,3,4\} \rightarrow$ 14 arbres binaires différents selon les ordres d'insertion suivants:

1. 1,3,2,4 et 1,3,4,2 donnent le même arbre
2. 1,2,4,3
3. 1,4,2,3
4. 1,4,3,2
5. 2,3,1,4 et 2,3,4,1 et 2,1,3,4 donnent le même arbre
6. 2,4,1,3 et 2,4,3,1 et 2,1,4,3 donnent le même arbre
7. 3,1,2,4 et 3,1,4,2 et 3,4,1,2 donnent le même arbre
8. 3,2,1,4 et 3,4,2,1 et 3,2,4,1 donnent le même arbre
9. 4,1,2,3
10. 4,1,3,2
11. 4,2,1,3
12. 4,2,3,1
13. 4,3,1,2
14. 4,3,2,1 et 1,2,3,4 donnent le même arbre

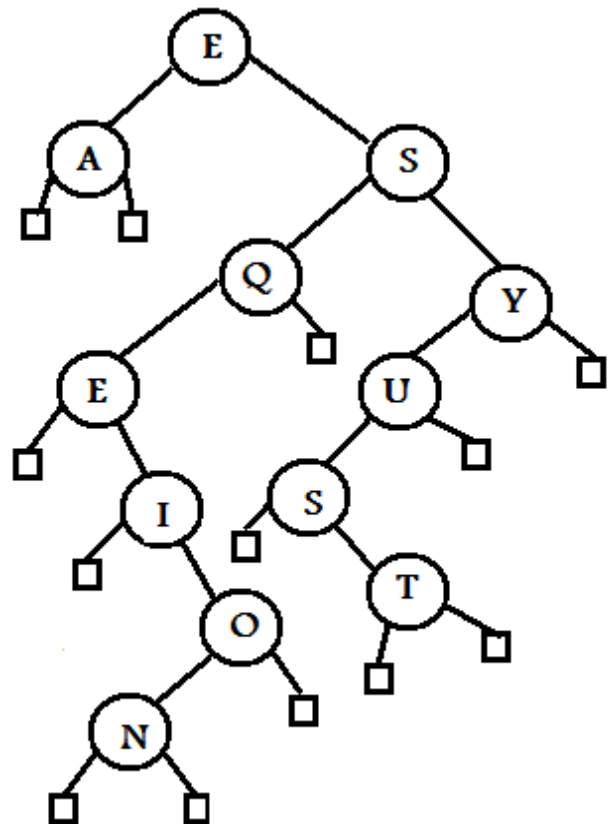
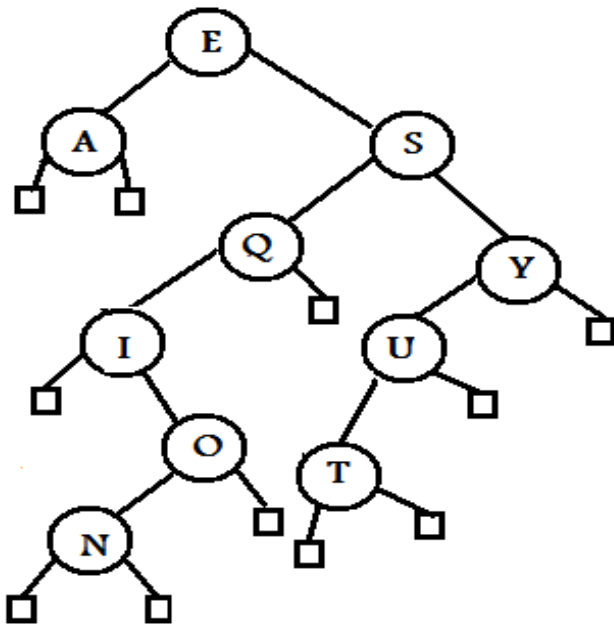
4. Donner un algorithme qui étant donné un arbre binaire contenant des clés entières dans ces nœuds internes, retourne VRAI, si l'arbre est un arbre binaire de recherche et FAUX, sinon.

```

EstArbreBinaire(Arbre nœud) retourne vrai ou faux {
  SI (le nœud est nul) ALORS
    Retourner Vrai
  SI (le nœud → gauche !=NULL) ET( nœud → gauche) >= nœud → valeur) ALORS
    Retourner Faux
  SI (le nœud → droite !=NULL) ET (nœud → droite) < nœud → valeur) ALORS
    Retourner Faux
  SI (EstArbreBinaire ( nœud → gauche) == FAUX)
    OU (EstArbreBinaire ( nœud → droite) ) == FAUX) ALORS
    Retourner Faux
}

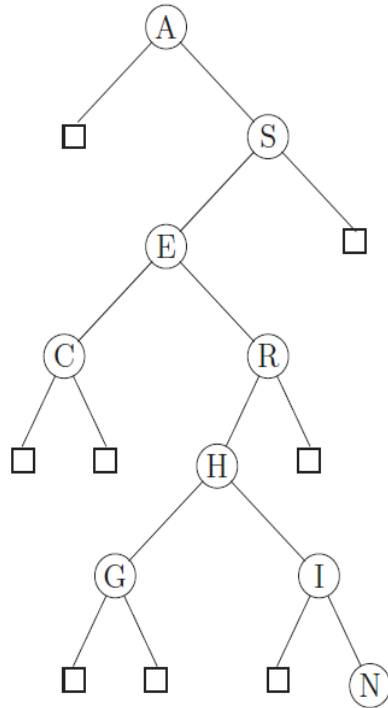
```

5. Dessiner l'arbre binaire de recherche résultant de l'insertion des clés E,A,S,Y,Q,U,T,I,O,N, dans cet ordre, dans un arbre initialement vide. L'ordre total sur les clés est l'ordre lexicographique.
6. Dessiner l'arbre binaire de recherche résultant de l'insertion des clés E,A,S,Y,Q,U,E,S,T,I,O,N, dans cet ordre, dans un arbre initialement vide.



RAPPEL: DANS LES ARBRES DE CE TP, LE FILS GAUCHE EST PLUS PETIT QUE LE NOEUD COURANT ET LE FILS DROIT EST PLUS GRAND OU ÉGAL.

7. L'insertion des clés A,S,E,R,H,I,N,G,C dans un arbre binaire de recherche initialement vide conduit à l'arbre ci-dessus. Donner 10 autres combinaisons de clés qui aboutissent au même arbre.



1. ASECRHIGN
2. ASECRHGIN
3. ASECRHING
4. ASERCHING
5. ASERHCING
6. ASERHCGIN
7. ASERHGINC
8. ASERHIGNC
9. ASERHICGN
10. ASERHIGCN

Deux lettres qui ne sont pas sur le chemin l'une de l'autre peuvent échanger leur ordre d'insertion sans que la structure de l'arbre ne soit affectée.

Par exemple: A est sur le chemin de C, donc changer leur ordre d'insertion affectera la structure de l'arbre. C n'est pas sur le chemin de R et pourra donc être inséré soit avant, soit après R sans que cela ne change la structure de l'arbre.

8. Écrire une méthode qui renvoie le nombre d'éléments d'un arbre binaire de recherche dont la clé est égale à une clé donnée.

```
compterCles(Arbre nœud, clé k, Entier Somme) {
```

```
    SI (nœud → valeur == k) ALORS
```

```
        Somme = Somme + 1
```

```
    SI (le nœud → gauche !=NULL ET max(nœud → gauche) < k)) ALORS
```

```
        compterCles (nœud → droite, k, Somme);
```

```
    SI (le nœud → droite !=NULL ET min(nœud → droite) > k)) ALORS
```

```
        compterCles (nœud → gauche, k, Somme);
```

```
    Retourner Somme;
```

```
}
```