Université Alger 1 Ben Youcef BenKhedda
Département Mathématiques et Informatique
Année universitaire 2016-2017

2ème année Informatique
Sections A et B
Rattrapage

Algorithmique et Structures de données Avancées

CHAQUE EXERCICE DOIT OBLIGATOIREMENT ÊTRE EFFECTUÉ SUR UNE FEUILLE SÉPARÉE

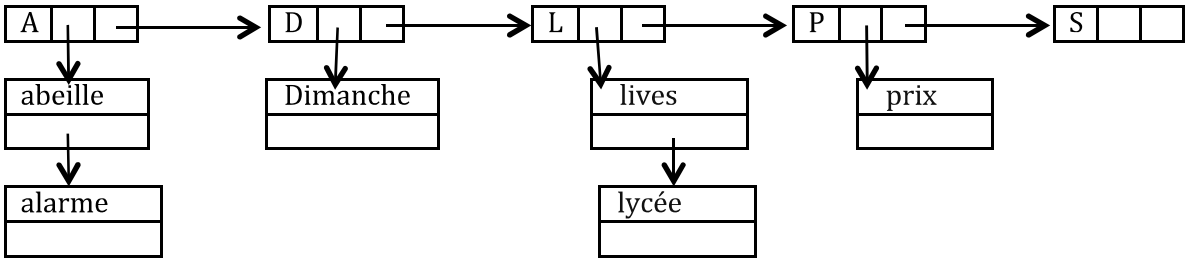
Exercice 1 (3.25 Pts)

On souhaite créer un dictionnaire en utilisant la structure de listes chainées.
Une liste contient les lettres de l'alphabet où chaque nœud pointe sur une liste des mots commençant par la lettre du nœud courant dans l'ordre alphabétique (voir le schéma).

1. Donner la déclaration de la structure**.**
2. On considère que la liste des lettres ne contient pas forcément toutes les lettres de l'alphabet de A à Z. Ecrire la fonction **Recherche_dic()** qui prend en paramètres la structure définie et un **mot** donné et cherche ce dernier dans la structure. La fonction retourne l'adresse de ce mot s'il existe, sinon elle l'insère à sa bonne position et retourne son adresse.

Pour simplifier, on utilise la fonction prédéfinie **Strcmp(M1, M2)** qui compare 2 chaines de caractères et renvoie l'une des 3 valeurs possibles suivantes:

0 si M1 et M2 sont identiques
-1 si M1 est avant M2 (M1< M2)
1 si M1 est après M2 (M1> M2)



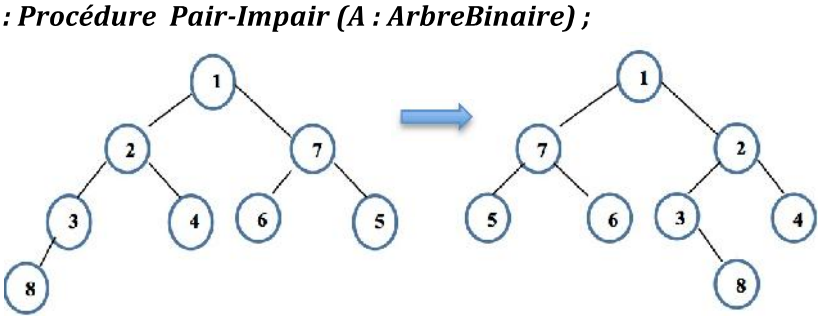
Exercice 2 : Les arbres binaires (4.5 Pts)

On considère une structure d'arbre binaire d'entiers, où chaque nœud contient un fils de clé paire et un fils de clé impaire.

Type ArbreBinaine { clé : Entier ; Fg, Fd : *ArbreBinaire } ;

1- Ecrire une procédure **récursive *Pair-Impair()*** qui sert à mettre les fils gauches impairs et les fils droits pairs. Autrement dit, si le fils gauche d'un nœud est pair et le fils droit est impair la procédure effectue une permutation entre les fils sans la création d'un nouveau nœud ou un nouvel arbre. Le schéma suivant illustre un exemple de permutation.

***Prototype* : Procédure Pair-Impair (A : ArbreBinaire) ;**



Algorithmique et Structures de données Avancées

2- Ecrire la fonction **récursive NB_Inf ()** qui retourne le nombre de clés inférieur à un élément « **x** » donné. **Prototype** : ***Fonction NB_Inf (A:* ArbreBinaire, x : entier*): entier;***

3- Ecrire la fonction **itérative *Liste-Prof ()*** qui revoie une liste chainée de type Liste qui contient toutes les clés des nœuds de profondeur « **p** » donnée.
Fonction Liste-Prof (A : ArbreBinaire, p : entier) : Liste ;

Note : Les fonctions sur les files *Enfiler ( ), Défiler ( ), Vide ( ) et Taille ()* peuvent être utilisées sans développement.

Exercice 3 : Les TAS, les tris, les ABRs (12.25 Pts)

On souhaite effectuer un tri **croissant** des lettres alphabétique on utilisant l'algorithme **tri par tas**. On considère l'ordre suivant A < B < …. < F < … K < O < P < …. < W < X < Y < Z.

1- Construire le **Max-Tas** correspondant à la séquence « **S** » suivante des éléments **qui arrivent dans l'ordre**.

S =

L	**I**	**K**	**R**	**D**	**V**	**B**	**O**	**F**	**T**

2- Donner toutes les étapes de **Tri par tas** pour trier la séquence « **S** » dans l'ordre **croissant** tout en montrant l'état du tas et du tableau trié durant toutes les étapes.

3- Construire l'arbre binaire de recherche « **ABR** » qui convient à la séquence « **S** » originale (**de la question 1**) en insérant les éléments dans l'ordre d'arriver.

4- Donner **le résultat** des parcours Préfixe, Infixe et Postfixe de l'arbre obtenu à la question précédente (question 3).

5- Supprimer de l'**ABR** obtenu à la question 3 la racine **L** en montrant tous les cas possible. Montrer l'état de l'arbre après chaque suppression.

6- On considère la déclaration d'un arbre binaire de recherche suivante (ABR) :

Type ArbreBinaineRecherche { clé : élément; Fg, Fd : * ArbreBinaineRecherche } ;

- Ecrire la fonction **itérative *Val_Sup()*** qui prend en paramètre un ABR et un élément « **x** » donné et retourne la valeur de la plus petite clé supérieure à « **x** ».
Prototype : ***Fonction Val_Sup (A:* ArbreBinaineRecherche, x : élément*): élément ;***

Corrigé type –Rattrapage-

ASDA – L2 – 2016/2017

Exercice 1: 3.25 Pts

1- Déclaration de la structure: 1.25/3.25

Type Liste {Clé : Chaine de caractères ; Svt : * Liste} ; 0.25 + 0.25

Type Liste-Dic {Tête : Caractère ; Mot : Liste ; Next : Liste-Dic} ; 0.25+ 0.25+ 0.25

2- Fonction Recherche dans le dictionnaire: 2/3.25

```
Fonction Recherche_dic ( F : Liste-Dic ; M : Chaine de caractères) : Liste
Début
{ T : Liste-Dic ; T=Nil ;
Tanque (F< > Nil et F.Tête < M [0]) faire
      T= F ; F= F.Next ;
Fait;
Si (F< > Nil et F. Tête = M[0] Alors
    Si (F.Mot = Nil) Alors
            New = Allouer (Liste) ; New.Clé= M;  F.Mot= New ; Retourner (New) ;
     Sinon
        N, P : Liste ; P= F.Mot ;  N= Nil ;
        Tanque (P < > Nil et  strcmp( P.Clé, M) = -1) faire
            N= P ; P= P.Svt ;
        Fait;
        Si (P < > Nil et strcmp( P. Clé, M) = 0 ) Alors Retourner (P) ;
         Sinon
              New = Allouer (Liste) ; New.Clé= M ; New.Svt= P;
               Si (N< > Nil) Alors  N.Svt= New ; Retourner (New) ;
                 Sinon  F.Mot = New ; Retourner (New) ;
              Fsi ;
        Fsi;
    Fsi;
    Sinon
       New = Allouer (Liste) ; New.Clé= M ; New.Svt= Nil ; New-Dic= Allouer (Liste-Dic) ;
       New-Dic. Tête =  M[0] ; New-Dic.Mot= New ; New-Dic.Next = F
       Si ( T < > Nil) Alors  ; T.Next= New-Dic; Fsi ;
       Retourner (New);
Fsi;
}FIN.
```

Exercice 2: 4. 5 Pts

1- Procédure récursive de permutation Pair-Impair: 1.5/4.5

```
Procedure Pair-Impair ( A: ArbreBinaire)
Début
{ Si  (A< > Nil) Alors
      Si ( A.Fg.clé  Mod 2 = 0) Alors
           S= A.Fg ; A.Fg= A.Fd ; A.Fd= S;
      Fsi;
      Pair-Impair (A.Fg);        Pair-Impair (A.Fd);
 Fsi;
}FIN.
```

2- Fonction récursive NB-Inf: 1.5/4.5

```
Fonction NB-Inf (A: ArbreBinaire, x : entier) : entier
Début
{Si A=Nil alors Retourner ( 0) ;
   Sinon
     Si A.clé(R)<x Alors
        Retourner (1 + NB-Inf(A.Fg) + NB-Inf(A.Fd) ;
      Sinon
        Retourner (NB-Inf(A.Fg) + NB-Inf(A.Fd) ;
     Fsi ;
   Fsi;
} FIN.
```

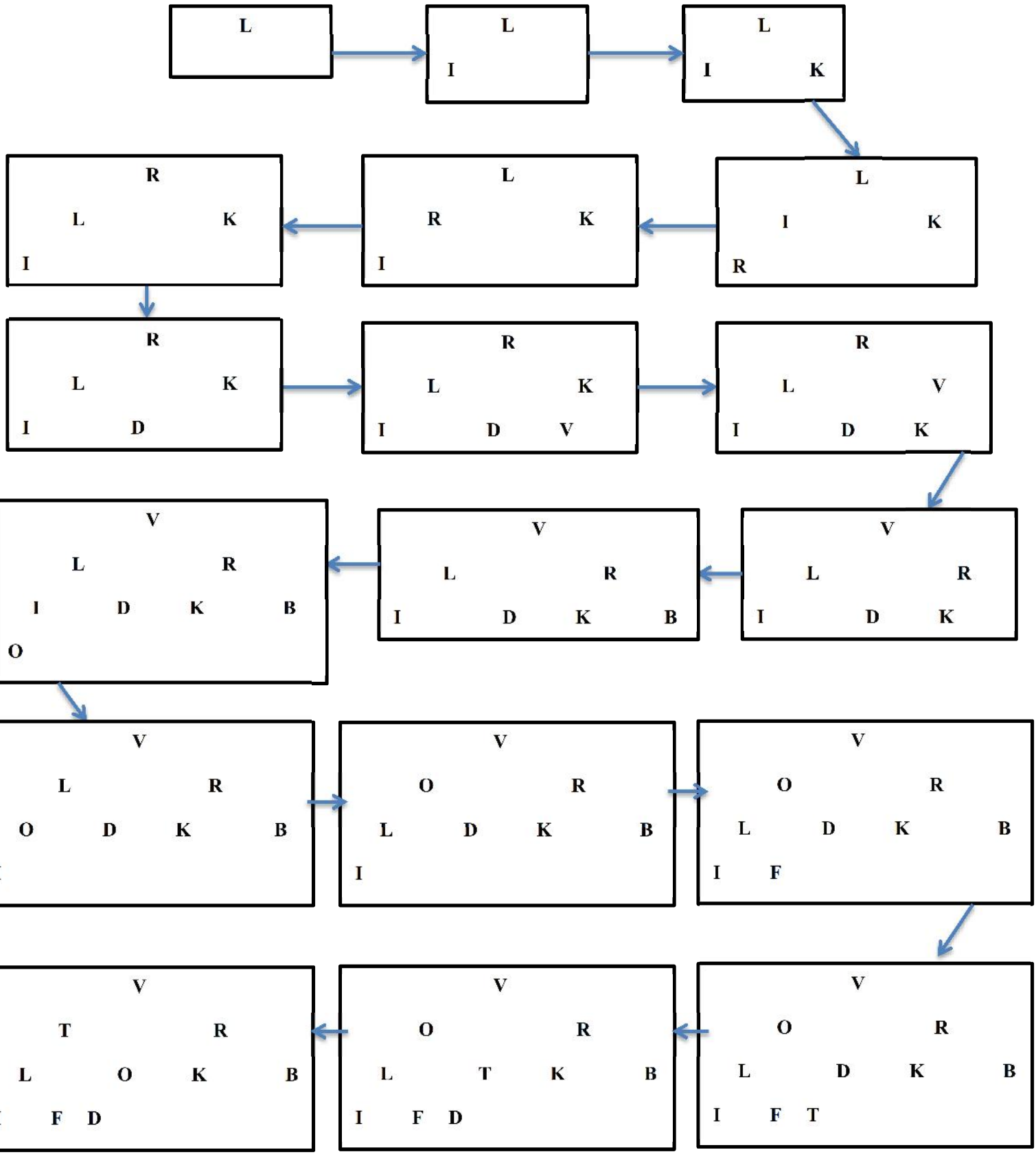
3- Fonction itérative Liste-Prof des nœuds d'une profondeur: 1.5/4.5

```
Fonction Liste-Prof (A: ArbreBinaire, p : entier) : Liste
Début
{Enfiler (F, A) ; D=0 ;
Tanque NonVide(F) Faire
   L= Taille (F) ; i=1 ;
   Tanque (i<L) Faire
     Tmp= Défiler (F) ;  i=i+1 ;
     Si (Tmp.Fg < > Nil) Alors Enfiler (F, Tmp.Fg) ;
     Si (Tmp.Fd < > Nil) Alors Enfiler (F, Tmp.Fd) ;
       Fait
   D=D+1 ;
   Si (D=P) Alors
      Thanque  NonVide(F) Faire
           Tmp = Défiler (F) ;        L= Insérer ( L, Tmp.clé) ;
      Fait
      Retourner (L) ;
    Fsi ;
Fait ; Free (F) ;
}FIN.
```

Exercice 3 : 12.25 Pts

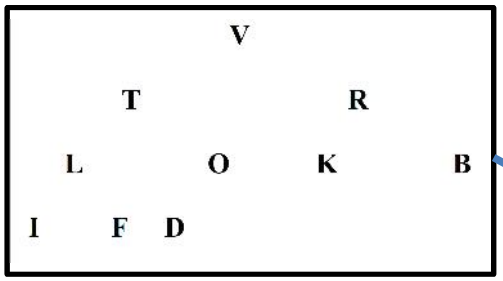
1- Construction du Max-Tas : 2.5/12.25

L	I	K	R	D	V	B	O	F	T

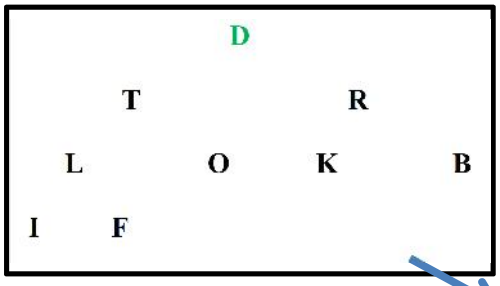


2- Tris par Tas dans l'ordre croissant déroulement : 3.25/12.25

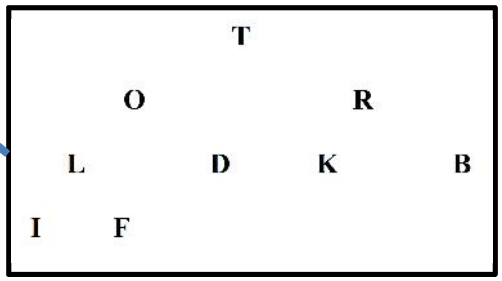
Après la construction du Max-Tas, on supprime à chaque itération la racine qui sera placée à sa bonne position dans tableau trié.



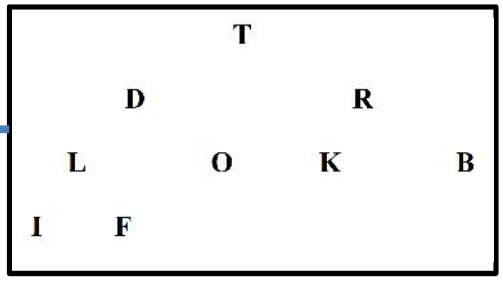
V	T	R	L	O	K	B	I	F	D



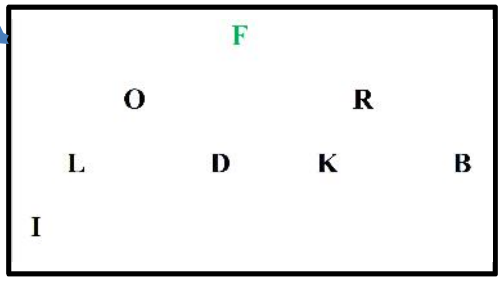
D	T	R	L	O	K	B	I	F	V



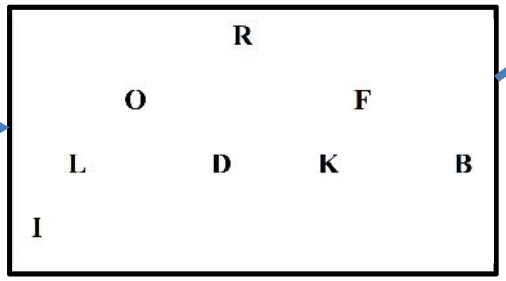
T	O	R	L	D	K	B	I	F	V



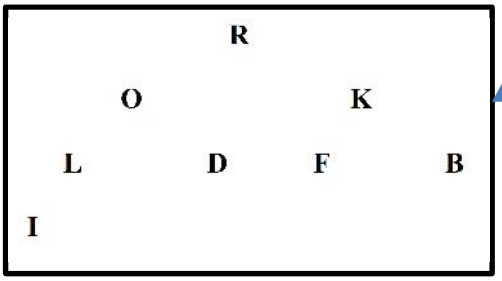
T	D	R	L	O	K	B	I	F	V



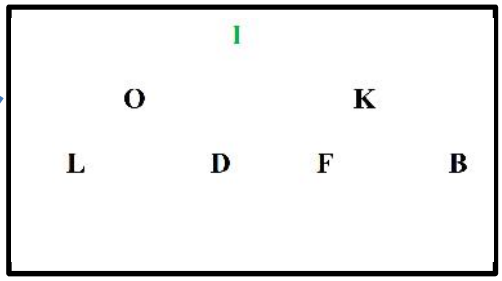
F	O	R	L	D	K	B	I	T	V



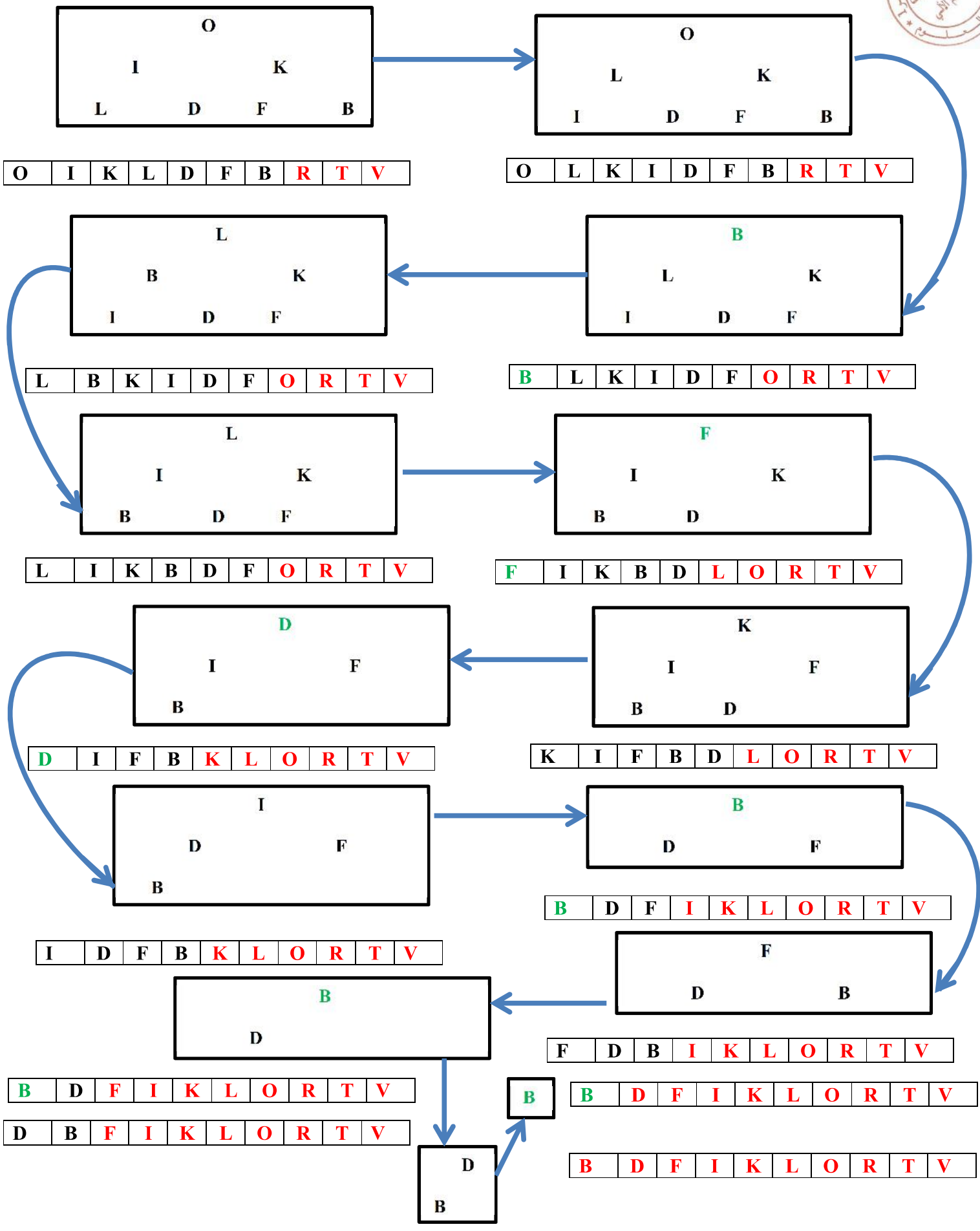
R	O	F	L	D	K	B	I	T	V



R	O	K	L	D	F	B	I	T	V

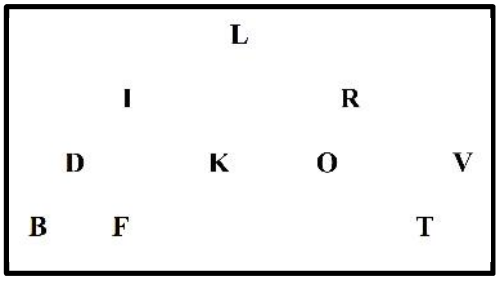


I	O	K	L	D	F	B	R	T	V


O I K L D F B
O I K L D F B R T V
O L K I D F B
O L K I D F B R T V
B L K I D F
B L K I D F O R T V
L B K I D F
L B K I D F O R T V
L I K B D F
L I K B D F O R T V
F I K B D
F I K B D L O R T V
K I F B D
K I F B D L O R T V
D I F B
D I F B K L O R T V
I D F B
I D F B K L O R T V
B D F
B D F I K L O R T V
F D B
F D B I K L O R T V
B D
B D F I K L O R T V
D B F I K L O R T V
D B
B
B D F I K L O R T V
B D F I K L O R T V

3- Construction de l'arbre binaire de recherche ABR : 2.5/12.25

L	I	K	R	D	V	B	O	F	T

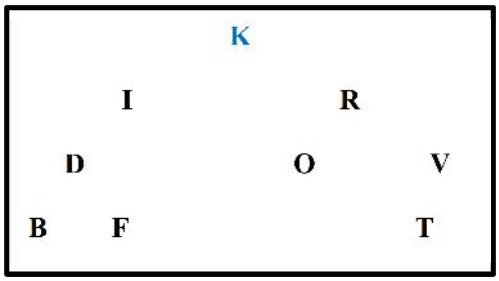


4- Parcours par profondeur : 1.5/12.25

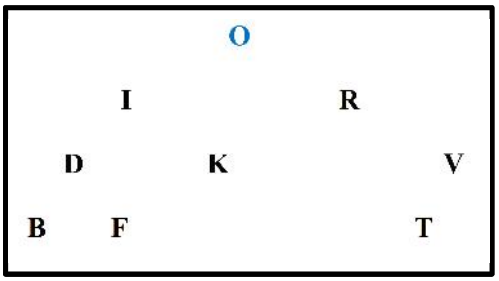
- **Préfixé :** L I D B F K R O V T
- **Infixé :** B D F I K L O R T V
- **Postfixé :** B F D K I O T V R L

5- Suppression de la Racine dans un ABR : 1/12.25

1) Remplacer la racine par le **Max à gauche**



2) Remplacer la racine par le **Min à droit**



6- Fonction itérative Val-Sup : 1.5/12.25

```
Fonction Val_Sup ( A : ArbreBinaireRecherche, x : élément) : élément
Début
{ Tanque (A < > Nil et A.clé < > x) Faire
      Si (A.clé > x ) Alors DG=A ; A=A.Fg ;
         Sinon  A= A.Fd ;
  Fait ;
Si  (A= Nil) Alors  Retourner (DG.clé) ;
     Sinon
        A=A.Fd ;
        Si (A=Nil) Alors Retourner (DG.clé) ;
          Sinon
              Tanque (A.Fg < >  Nil) Faire  A=A.Fg ; Fait ;
              Retourner (A.clé)
        Fsi ;
 Fsi ;
}FIN.
```