Université Larbi Ben M'hidi
Faculté : SESNV
Département : Mathématiques et Informatique

Module : ASD 3
Semestre : 3 (Informatique) Janvier 2024

Corrigé du contrôle (2023/2024)

Exercice 1 (06 pts):

Rappelons l'algorithme de tri à bulles du plus grand élément qui est une certaine forme de tri par sélection du maximum.

```
void TriBulles (int* t, int nb) {
    int i, k ;
    for (i = nb-1; i >0; i--) {
        for (k = 0; k < i; k++) {
            if (t[k] > t[k+1])      echanger(&t[k], &t[k+1]) ;  }
 }
}
```

Soit le tableau de caractère suivant : **I N F O R M A T**

1. Transformez l'algorithme pour qu'il puisse trier des caractères. **0,5pt**

   Il suffit de changer « void TriBulles (*int* t*, int nbElements) » par

   « void TriBulles (*char* t*, int nbElements) »

2. Donnez les états successifs du tableau à la fin de chaque étape de la boucle '*for*' interne lorsque i = 7. **1,5pt**

   K=0 : INFORMAT
   K=1 : IFNORMAT
   K=2: IFNORMAT
   K=3 : IFNORMAT
   K=4 : IFNOMRAT
   K=5 : IFNOMART
   K=6 : IFNOMART

3. Même question pour la fin de chaque étape de la boucle **'*for*'** externe (principale). **1,5 pt**

   I=6: FINMAORT
   I=5 : FIMANORT
   I=4 : FIAMNORT
   i=3 : FAIMNORT
   i=2 : AFIMNORT
   i=1 : AFIMNORT

4. Le tri par sélection consiste à chercher le plus petit élément de la tranche restante à trier et le placer au début de cette tranche. Il est possible aussi de faire un tri par sélection du plus grand élément. Donnez l'algorithme de tri par sélection du plus grand élément. Le tri doit se faire toujours par ordre croissant. **2,5 pts**

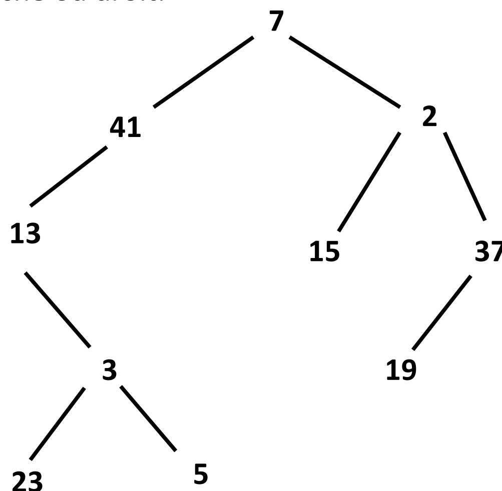
```
#include <stdio.h>

void permuter(int* a, int* b){
        int tmp = *a;
        *a = *b;
        *b = tmp;
}

void triSelection(int* t, int n) {
        int i, j;
        for (i=n-1; i>0; i--)       {
                int imax = i;
                for (j=i-1; j>=0; j--)
                        if (t[j]>t[imax])
                                imax = j;
                permuter(&t[i], &t[imax]);
        }
}
```

Exercice 2 (05 pts):

Soit le tableau suivant qui représente un arbre binaire T en triplets (info, gauche, droit) :

5	2	3	23	15	37	13	7	41	19
-1	4	3	-1	-1	9	-1	8	6	-1
-1	5	0	-1	-1	-1	2	1	-1	-1

La première colonne (indice 0) représente le nœud dont le champ info est 5 (valeur du nœud), le champ gauche est -1 (indice du fils gauche) et le champ droit est -1 (indice du fils droit), la seconde colonne (indice 1) représente le nœud dont le champ info est 2, le champ gauche est 4 et le champ droit est 5, et ainsi de suite.
La valeur (-1) indique l'absence d'un fils gauche ou droit.
1. Dessiner l'arbre binaire T. **1,5 pt**



2. Donner le code C de la structure de données pour représenter l'arbre T de cette manière. **1,5pt**

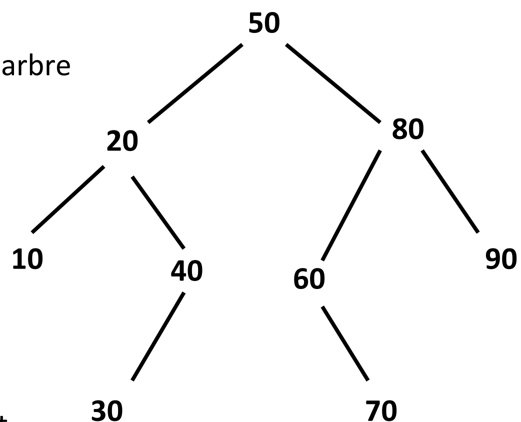
```
typedef struct Noeud
{           int info, gauche, droit;
} Noeud;

Noeud T[10] = {{5, -1, -1}, {2, 4, 5}, {3, 3, 0}, {23, -1, -1}, {15, -1, -1}, {37, 9, -1},

                    {13, -1, 2}, {7, 8, 1}, {41, 6, -1}, {19, -1, -1}};
```

3. Ecrire une fonction **nbrefeuilles** qui calcule le nombre de feuilles dans l'arbre T. **2pts**

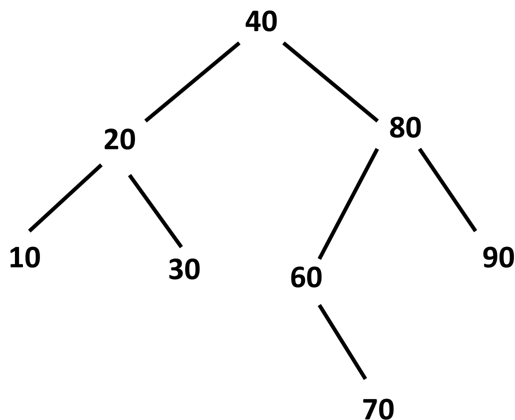
```
Int nbrefeuilles(int* T)  {

        int S=0;

        for(i = 0; i < 10; i++)

                if (T[i].gauche == -1 && T[i].droit == -1)
                        S=S+1 ;

        return S;          }
```

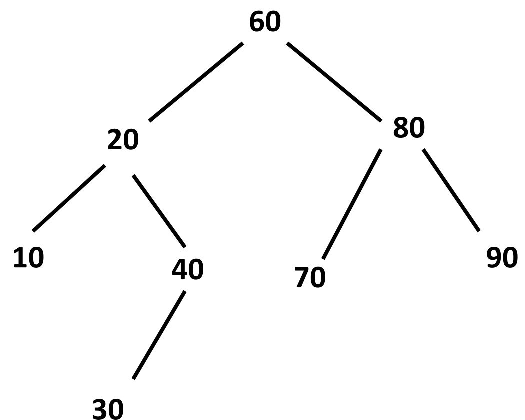
<u>Exercice 3</u> (05 pts):

1. Donner le résultat des parcours en profondeur de l'arbre binaire de recherche suivant : **1,5pt**
   Prefixe : 50 20 10 40 30 80 60 70 90
   Infixe : 10 20 30 40 50 60 70 80 90
   Postfixe : 10 30 40 20 70 60 90 80 50

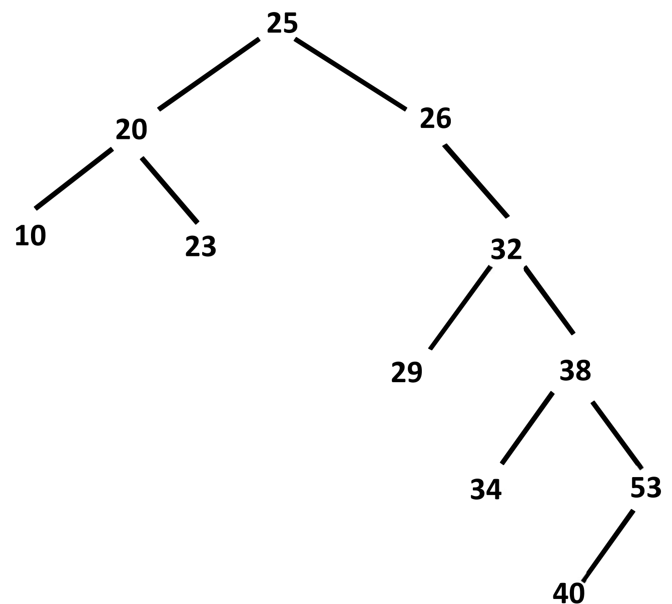


2. Donner les nouveaux arbres obtenus en supprimant l'élément 50. **1pt**





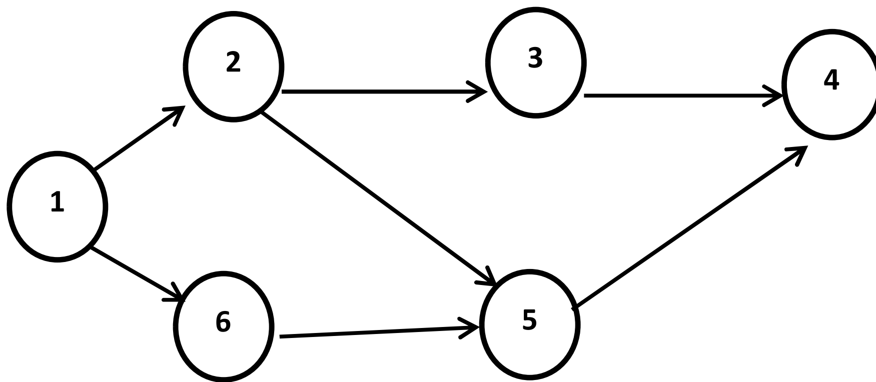
3. Construire l'arbre binaire de recherche dans l'ordre de la liste suivante : **1pt**
   25 26 20 32 38 53 10 29 34 23 40



4. Ecrire une fonction **Cherche_valeur** qui renvoie l'adresse du nœud de l'ABR contenant cette valeur ou NULL si cette valeur ne figure pas dans l'arbre. **1,5pt**

```
typedef struct Noeud {
        int valeur;
        struct Noeud* gauche;
        struct Noeud* droite;
} Noeud;

Noeud* Cherche_valeur(Noeud* racine, int valeur) {
  // Cas de base : l'arbre est vide ou la valeur est présente à la racine
        if (racine == NULL || racine->valeur == valeur) {
                return racine;          }

  // Si la valeur est plus petite que la racine, chercher dans le sous-arbre gauche
        if (valeur < racine->valeur) {
                return Cherche_valeur(racine->gauche, valeur);   }

  // Sinon, chercher dans le sous-arbre droit
        return Cherche_valeur(racine->droite, valeur);
}
```

Exercice 4 (04pts):

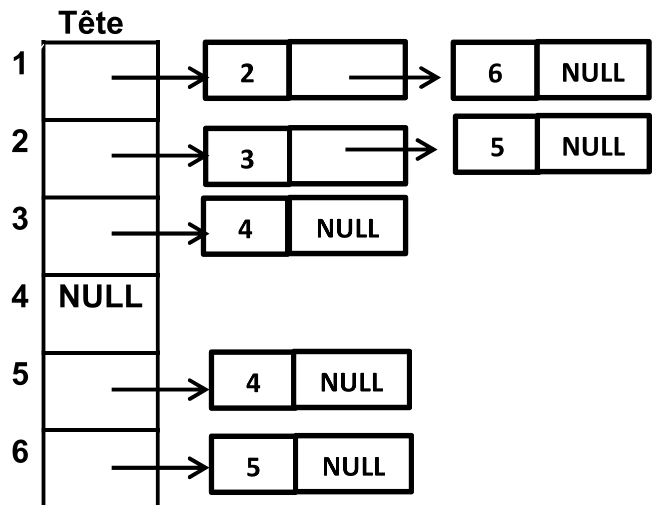


1. Donnez toutes les représentations de ce graphe. **2pts**

Représentation par matrice d'adjacence

	1	2	3	4	5	6
1	0	1	0	0	0	1
2	0	0	1	0	1	0
3	0	0	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	1	0

Représentation par listes chainées



Représentation par listes contigües

1	3	5	6	6	7	8	Tête
2	6	3	5	4	4	5	Succ

2. Donnez les résultats du parcours de ce graphe à partir du sommet 1. **2pts**

Parcours en largeur (BFS : Breadth-First Search) : 126354

Parcours en profondeur (DFS : Depth-First Search) : 123456