Université IBN KHALDOUN –TIARET- Faculté Des Mathématiques et de l'informatique
Département d'informatique, 2ière Année Licence : -2016/2017-

TP : Récursivité

Objectif : Ecrire un algorithme/programme récursif
Problème : écrire un algorithme récursif réalisant un certain traitement T sur des données D.

1- décomposer le traitement T en sous traitements de même nature mais sur des données plus petites
2- trouver la condition d'arrêt
3- tester éventuellement sur un exemple
4- écrire l'algorithme

Exercice 1 : Suite de Fibonacci

La suite de Fibonacci est définie comme suit :

$$Fib(n) = \begin{cases} 1 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ Fib(n-1) + Fib(n-2) & \text{sinon} \end{cases}$$

1. Ecrivez un algorithme itératif calculant Fib(n).
2. Ecrivez un algorithme récursif calculant Fib(n).
3. Calculez ça complexité en nombre d'additions.

Corrigé :

Fonction FIB(n :entier) :entier
si n=0 ou n=1 alors retourner 1
sinon retourner FIB(n-1) + FIB(n-2)

Exercice 2 (Calcul de pgcd) :

Soit a et b des entiers. On note que pgcd(a, b) = pgcd(a − b, b).

2. Écrire un algorithme récursif permettant de calculer le pgcd.

```
Entrées : a, b entiers
Résultat : le plus grand commun diviseur de a et b s'il existe, 0 sinon
fonction PGCD(a,b)
début
   si a=0 ou b=0 alors
      retourner 0
   sinon si a=b alors
      retourner a
   sinon si a > b alors
      retourner PGCD(a-b,b)
   sinon
      retourner PGCD(a,b-a)
fin
```

Exercice 3 : Ecrire un algorithme itératif et un autre récursif qui considèrent une phrase comme un tableau de caractères et d´énumérer les éléments de ce tableau dans l'ordre inverse.

```
void reverse(int a[],int start,int end)
{
     int temp;
     temp = a[start];
     a[start] = a[end];
     a[end] = temp;

    if(start==end ||start==end-1)
       return;
```

```
    reverse(a, start+1, end-1);
}
```

Exercice 4 : La recherche dichotomique

fonction avec retour entier dicho(entier[] t, entier n, entier i, entier j)
début
si (i>j ou t[(i+j)/2]=n) alors
si (i>j) alors
retourne -1;
sinon
retourne (i+j)/2;
finsi
sinon
si (t(i+j)/2] > n) alors
retourne dicho(t,n,i,(i+j)/2 – 1);
sinon
retourne dicho(t,n,(i+j)/2 + 1,j);
finsi
finsi
fin

Exercice 5 : Suite d'Ackermann

A(m,n) = n+1 si m = 0,

A(m,n) = A(m-1,1) si n=0 et m > 0

A(m,n) = A(m-1, A(m,n-1)) sinon

```
fonction avec retour entier ackerman(entier m, entier n)
début
    si (m = 0) alors
        retourne n+1;
    sinon
        si ((m>0) et (n=0)) alors
            retourne ackerman(m-1,1);
        sinon
            retourne ackerman(m-1,ackerman(m,n-1));
        finsi
    finsi
fin
```

Exercice 6 :

Soit une fonction qui vérifie si un élément ‘x’ appartient à une partie d’un tableau ‘A’ appelée IS_MEMBER() qui reçoit en paramètres le tableau A, l’élément ‘x’ et les indices de la partie du tableau ‘d’ et ‘f’ et retourne ‘true’ ou ‘false’.

a. Si l’ensemble ‘A’ n’est pas trié :
   1. Ecrivez un algorithme récursif pour cette fonction ;

b. Si l’ensemble ‘A’ est trié dans l’ordre croissant :
   2. Ecrivez un algorithme récursif pour cette fonction ;
   3. Utilisez la recherche dichotomique pour améliorer votre algorithme ;

Soit les fonctions :
   UNION(A,B) qui fait l’union de deux ensembles ;
   INTERSECT(A,B) qui fait l’intersection de deux ensembles ;

c. Si les ensembles ‘A’ et ‘B’ sont triés dans l’ordre croissant
   4. Ecrivez des algorithmes récursifs pour ces fonctions ;

a. Si l’ensemble ‘A’ n’est pas trié :
   1. Ecrivez un algorithme récursif pour cette fonction ;

IS_MEMBER(A, x, d, f) : booleen
si d>f alors retourner False

si A[d]=x alors retourner True
sinon retourner IS_MEMBER(A, x, d+1, f)

b. Si l'ensemble 'A' est trié dans l'ordre croissant :

2. Ecrivez un algorithme récursif pour cette fonction ;

IS_MEMBER(A, x, d, f):) : booleen
si d>f ou A[d]>x alors retourner False
si A[d]=x alors retourner True
sinon retourner IS_MEMBER(A, x, d+1, f)

3. Utilisez la recherche dichotomique pour améliorer votre algorithme ;

IS_MEMBER(A, x, d, f) :booléen
si d=f alors si A[d]=x alors retourner True
sinon retourner False
m←[(d+f)/2]
si A[m]=x alors retourner True
sinon si A[m]>x alors retourner IS_MEMBER(A, x, d, m-1)
sinon retourner IS_MEMBER(A, x, m+1, f)

Soit les fonctions :

UNION(A,B) qui fait l'union de deux ensembles ;
INTERSECT(A,B) qui fait l'intersection de deux ensembles ;

c. Si les ensembles 'A' et 'B' sont triés dans l'ordre croissant

4. Ecrivez des algorithmes récursifs pour ces fonctions ;

```
UNION(A, idxA, B, idxB, C)
   si (idxA ≤ taille(A) ou idxB ≤ taille(B)) alors
   ⌈      si (idxB > taille(B) ou B[idxB] > A[idxA]) alors
   |      ⌈      taille(C) ← taille(C) + 1
   |      |      C[taille(C)] ← A[idxA]
   |      ⌊      UNION(A, idxA+1, B, idxB, C)
   |      sinon
   |      ⌈      taille(C) ← taille(C) + 1
   |      |      C[taille(C)] ← B[idxB]
   |      |      si (idxA<=taille(A) et A[idxA]=B[idxB]) alors UNION(A,idxA+1,B,idxB+1,C)
   ⌊      ⌊      sinon UNION(A, idxA, B, idxB+1, C)
```

L'appel initial est :

```
UNION(A, 1, B, 1, C)
```

```
INTERSECT(A, idxA, B, idxB, C)
   si (idxA ≤ taille(A) et idxB ≤ taille(B)) alors
   ⌈      si (A[idxA] = B[idxB]) alors
   |      ⌈      taille(C) ← taille(C)+1
   |      |      C[taille(C)] ← A[idxA]
   |      ⌊      INTERSECT(A, idxA+1, B, idxB+1, C)
```

```
|       sinon ⌈si (B[idxB] > A[idxA]) alors INTERSECT(A, idxA+1, B, idxB, C)
⌊             ⌊sinon INTERSECT(A, idxA, B, idxB+1, C)
```

L'appel initial est :

```
INTERSECT(A, 1, B, 1, C)
```

Exercice 7 : (Maximum d'une liste)

1. Construire une fonction max qui renvoie le maximum de deux réels.
2. Écrire un algorithme récursif maximum donnant le maximum d'une liste de nombres réels quelconques. On pourra utiliser le fait que max(a, b, c) = max(a, max(b, c))

Exercice 8 : Récursivité croisée

La récursivité croisée consiste à écrire des fonctions qui s'appellent l'une l'autre.

Exemple :

```
// cette fonction renvoie vrai si l'entier est pair, faux sinon
// on suppose que l'entier est positif ou nul
fonction avec retour booléen estPair(entier n)
début
     si (m = 0) alors
         retourne VRAI;
     sinon
         retourne estImpair(n-1);
     finsi
fin
```

```
// cette fonction renvoie vrai si l'entier est impair, faux sinon
// on suppose que l'entier est positif ou nul
fonction avec retour booléen estImpair(entier n)
début
     si (m = 0) alors
         retourne FAUX;
     sinon
         retourne estPair(n-1);
     finsi
fin
```

Exercice 9 : Le problème des tours de Hanoï

1. Ecrire un programme C qui permette de résoudre le problème des tours de Hanoï.
2. Evaluer la complexité en nombre de déplacement des disques pour résoudre ce problème.