Récursivité

Présenté par D. ELGHANAMI
Pr. À l'EMI

Mai 2011

La récursivité

- Méthode de programmation puissante qui permet d'exprimer d'une manière élégante la solution de problèmes.
- Une fonction s'appelle elle-même
  - Une condition d'arrêt
  - Un appel
- Fonctionnement
  - La pile des appels
- Exemples :
  - Récurrences mathématiques classiques
  - Tour d'Hanoï
  - Tri rapide, tri fusion, …
  - Recherche dichotomique, …
  - ….

Les questions clé de la récursivité

1. Comment exprimer la solution du problème en terme de la solution du même problème ?
2. En quoi chaque appel récursif diminue-t-il la taille du problème ?
3. Quels cas du problème ont des solutions simples et serviront de cas de base?
4. Les appels récursifs convergent-ils vers un cas terminal ?
5. Les types de parcours et méthode de programmation.
6. Dans quel cas on peut utiliser l'Itération et/ou la récursivité.

- **Récursivité simple** : un appel récursif

  Exemple : La fonction puissance $x \rightarrow x^n$

$$x^k = \begin{cases} 1 & si\ k = 0 \\ x * x^{k-1} & sinon \end{cases}$$

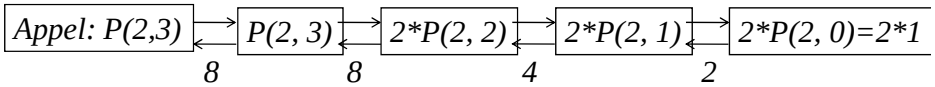
$x^k = 1/x * x^{k+1}$

Récursivité terminale ?

Programme en C :

```
int   puissance(float x, int k){
  if (k == 0)                      ← Condition d'arrêt
    return  1 ;
  else
    return  x*puissance(x, k–1);   ← Appel récursif
}
```

Code avant l'appel



- **Récursivité multiple** : plus d'un appel récursif.

Exemple : Calcul de combinaison

$$C_n^p = \begin{cases} 1 & si\ p = 0\ ou\ p = n \\ C_{n-1}^p + C_{n-1}^{p-1} & sinon \end{cases}$$

Récursivité imbriquée :

$$A(m,n) = \begin{cases} n+1 & si\ m = 0 \\ A(m-1, 1) & sinon\ \ si\ m > 0\ et\ n = 0 \\ A(m-1, A(m, n-1)) & sinon \end{cases}$$

Types de stratégies récursives

- Ascendante
- Descendante
- Par divisions successives (diviser pour régner)

Expemple : Somme des carrés de x à y

$x^2 + (x+1)^2 + \ldots. + y^2$

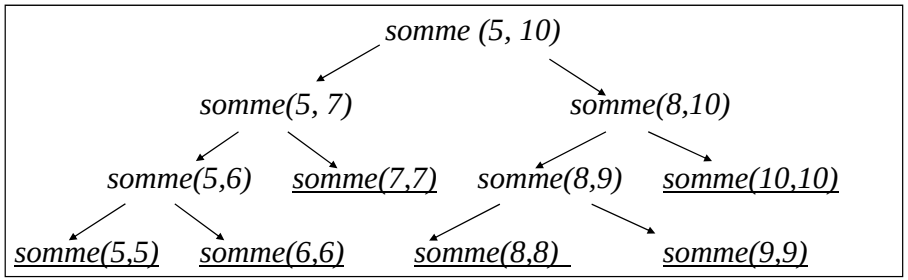
$$somme\ (x, y) = \begin{cases} x^2 & si\ x = y \\ x^2 + Somme(x+1, y) & sinon \end{cases}$$

$$somme\ (x, y) = \begin{cases} x^2 & si\ x = y \\ y^2 + Somme(x, y-1) & sinon \end{cases}$$

$$somme(x, y) = \begin{cases} x^2 & si\ x = y \\ somme(x, m) + somme(m+1, y) & sinon \end{cases}$$

avec $m = (x + y)/2$

Arbre de l'algorithme



Le mécanisme de la récursivité

Récursivité et boucle

- La récursivité peut simuler l'effet d'une boucle.
- Reçoit en paramètre le nombre d'itérations à réaliser (compteur)

```
void boucle (int i){
    if (i < N){
        Instructions;
        boucle(i+1);
    }
}
```

```
for(i=0; i<N; i++){
    Instructions;
}
```

Qu'arriverait-il si on inversait ces 2 opérations?

```
void boucle (int i){
    if (i < N){
        boucle(i+1);
        Instructions;
    }
}
```

Code exécuté après retour de l'appel

Récursivité et Itération

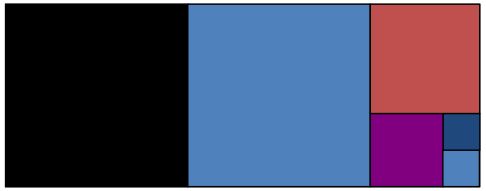
```
ALGORITHME P(U)
    si C(U) alors
        D(U)
        P(a(U))
    sinon
        T(U)
Fin.
```

```
ALGORITHME P(U)
    tant que C(U) faire
        D(U)
        U ← a(U)
    Fin tant que
    T(U)
Fin.
```


Algorithme d'Euclide pour le PGCD

- Trouver le plus grand carré permettant de le paver
- Découper en carré de côté = hauteur (largeur) du rectangle de façon à former un rectangle plus petit.
- Recommencer jusqu'à ce que la figure restante soit un carré
- Résultat : Le côté du carré est le pgcd de a et b.

Rectangle de cotés 65 et 25

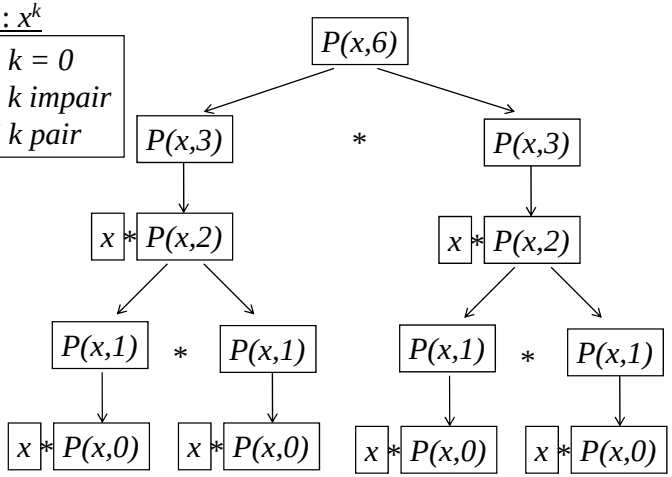


```
while (a != b) {
  if (a > b) a = a – b ;
  else b = b – a ; }
```

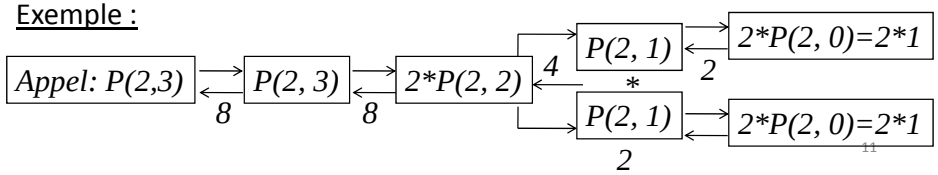
$$PGCD(a,b)=\begin{cases} PGCD(a-b,b) & si\ a>b \\ PGCD(a,b-a) & si\ a<b \\ a & sinon \end{cases}$$

Fonction puissance : x^k

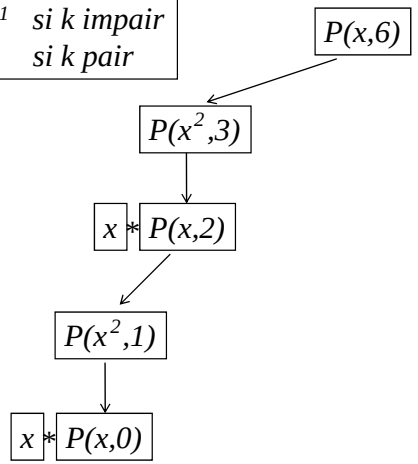
$$x^k=\begin{cases} 1 & si\ k=0 \\ x * x^{k-1} & si\ k\ impair \\ x^{k/2} * x^{k/2} & si\ k\ pair \end{cases}$$



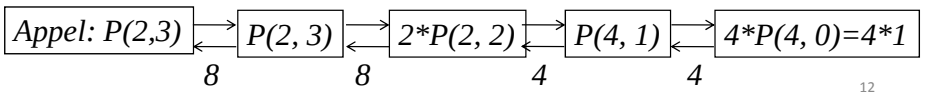
Exemple :



$$x^k=\begin{cases} 1 & si\ k=0 \\ x * x^{k-1} & si\ k\ impair \\ (x^2)^{k/2} & si\ k\ pair \end{cases}$$

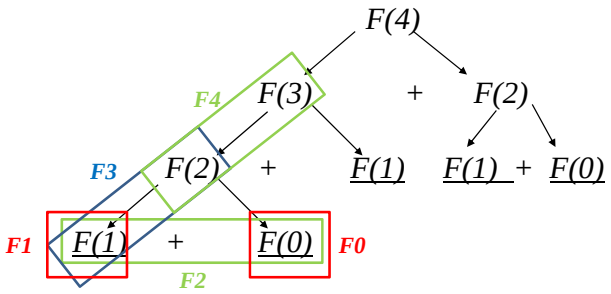


Exemple :



Suite de Fibonacci :

$F(0) = 1$
$F(1) = 1$
$F(n) = F(n-1) + F(n-2)$



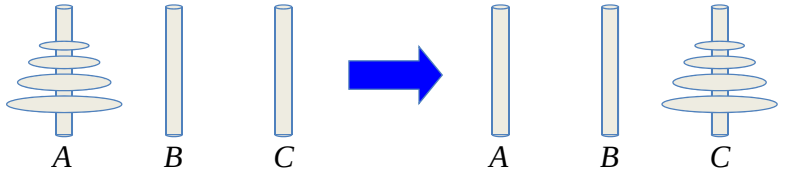
F[0]	F[1]	F[2]	F[3]	F[4]

Taille du tableau ?

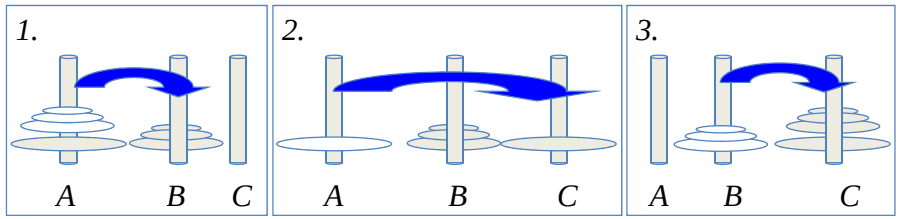
F[0]	F[1]	F[2]

Les Tours de Hanoï

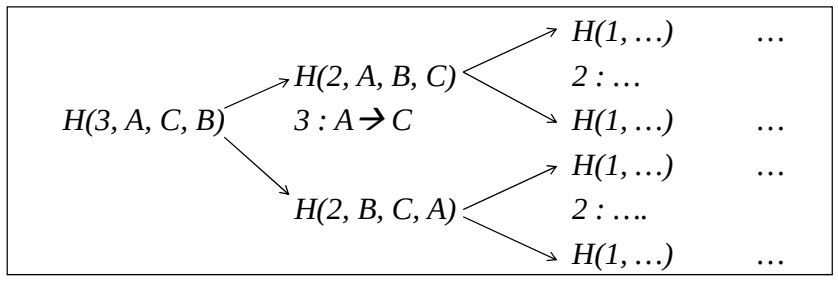
- Objectif : Déplacer tous les disques de la tige A à la tige C
  - La tige B sert d'emplacement temporaire
  - On ne peut empiler un disque sur un disque plus petit



- On décompose le problème en un problème plus simple
  1. On déplace les (n-1) disques, sauf le nème sur la tige B
  2. On déplace le nème disque sur la tige C
  3. On déplace tous les disques de la tige B sur la tige C



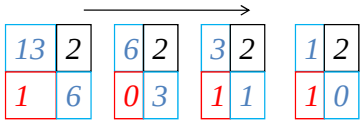

```
Hanoi (n, A, C, B)
   si (n >= 1)
     Hanoi (n – 1, A, B, C)
     déplacer n A → C
     Hanoi (n – 1, B, C, A)
   fin-si
```



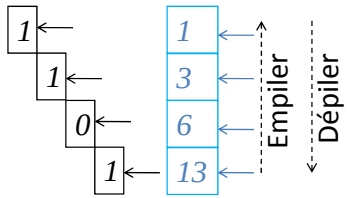
La pile des appels

- Une pile est utilisée pour mémoriser:
  - l'état des variables locales
- À chaque appel, les valeurs des variables locales sont empilées
  - l'instruction return ou fin provoque la descente de la pile

Exemple : convertir un entier décimal en base binaire : $(13)_{10} = (1011)_2$




```
void conversion (int  x){
  if( x != 0){
    conversion (x/2) ;
    printf("%d", x%2) ;
   }
}
```



Conversion d'un entier en chaîne de caractères :

Ver1

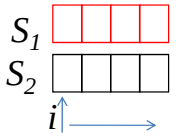
```
void EntCh (int  x, char T[], int* i){
  if( x != 0){
    EntCh (x/10, T, i) ;
    T[*i] = x%10 + '0' ;
    (*i)++;
    T[*i] = '\0' ;
  }
}
```

Ver2

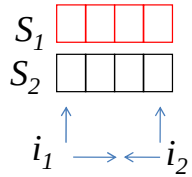
```
int EntCh (int  x, char T[], int i){
  if( x != 0){
    i = EntCh (x/10, T, i) ;
    T[i] = x%10 + '0' ;
    i++;
    T[i] = '\0' ;
  }
  return i ; }
```

Conversion d'une chaîne de caractères/entier en entier :

```
int ChEnt (char T[]){
  int i = 0, x = 0 ;
  while(T[i] != '\0'){
    x = x*10 + T[i] – '0' ;
    i++;
  }
  return x ; }
```

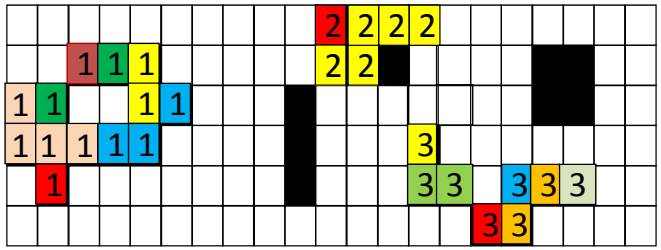
$$S_1 == S_2 \ ? \begin{cases} 0 & si\ l_1 \neq l_2 \\ 1 & sinon\ si\ i = l_1 \\ 0 & sinon\ si\ S_{1i} \neq S_{2i} \\ S_1 == S_2\ ?\ (\text{-1 case}) & sinon \end{cases}$$



Remarques:

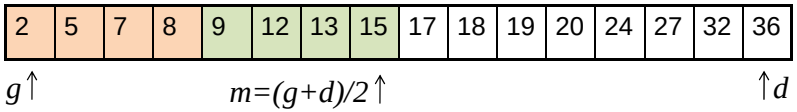
- Les programmes itératifs sont souvent plus efficaces,
- mais les programmes récursifs sont plus faciles à écrire.

Coloriage d'objets



```
void coloriageObjet( int  i , int  j , int  c ){
  if( i>=0  &&  i<N  &&  j>=0  &&  j<M  &&  T[i][j]==1 ){
     T[i][j] = c;
     for( k=i–1; k<= i+1; k++ )
       for( l=j–1; l<= j+1; l++ )
         if( k>=0  &&  k<N  &&  l>=0  &&  l<M  &&  T[k][l]==1 )
            coloriageObjet( k , l , c );
  }
}
```

Recherche dichotomique




```
int rechDicho (int x, int g, int d){
  int m ;
  if( g > d)  return –1;
  m = (g + d)/2;
  if (x == t[m])  return m;
  else
    if(t[m] < a)
      return rechercheDicho(x, m+1, d);
    else
      return rechercheDicho(x, g, m-1);
}
```

```
while( g <= d){
  m = (g + d)/2;
  if (x == t[m])
    return m;
  else
    if(t[m] < a)
      g = m+1 ;
    else
      d = m – 1 ;
}
return –1 ;
```

Tri rapide (QuickSort)

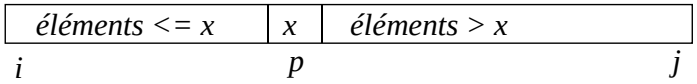
C'est un tri récursif comportant 3 étapes :

- Partition autour du pivot
  - Un élément pivot du tableau à trier est choisi
  - Le tableau est réorganisé afin que:
    - les éléments <= au pivot se retrouvent avant le pivot
    - Les éléments > pivot se retrouvent après le pivot

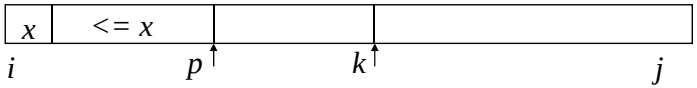
    Conséquence de cette étape : l'élément pivot est à sa position finale (lorsque le tableau est complètement trié)
- Un appel récursif pour trier les éléments avant le pivot
- Un appel récursif pour trier les éléments après le pivot

Fonctionnement de la fonction pivot

Objectif : x valeur du pivot p



À une certaine étape du processus

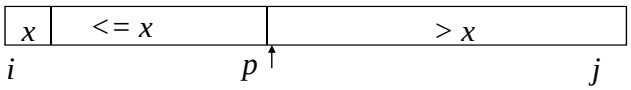


p pointe sur le dernier élément connu pour lequel T[p] >= x .

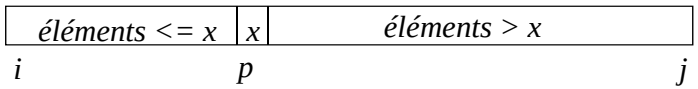
A sa droite, les éléments sont > x jusqu'à k qui pointe sur le premier élément non encore traité.

```
si T[k] <= x
        p ← p+1
        échanger(T[k], T[p])
fin-si
```


Après avoir parcouru le tableau



On échangea T[i] et T[p]




```
void  trier(int  t[ ], int  min, int  max){
    int  p, i;
    if(min<=max){
      p = min;
      for(i=p+1; i<=max; i++)
        if(t[i]<=t[min]){
          p++;
          permuter(t, i, p);
        }
      permuter(t, min, p);
      trier(t, min, p–1);
      trier(t, p+1, max);
    }
}
```

Exemples :

$$x^k = \begin{cases} 1 & si\ k = 0 \\ x * x^{k-1/2} * x^{k-1/2} & si\ k\ impair \\ x^{k/2} * x^{k/2} & si\ k\ pair \end{cases}$$

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

Tri Fusion

tri : 69 85 34 24 40 77 64
tri : 69 85 34 24
tri : 69 85
tri : 69
tri : 85
fusion : 69 et 85 → 69 85
tri : 34 24
tri : 34
tri : 24
fusion : 34 et 24 → 24 34
fusion : 69 85 et 24 34 → 24 34 69 85
tri : 40 77 64
tri : 40 77
tri : 40
tri : 77
fusion : 40 et 77 → 40 77
tri : 64
fusion : 64 → 64
fusion : 40 77 et 64 → 40 64 77
fusion : 24 34 69 85 et 40 64 77 → 24 34 40 64 69 77 85