18/11/2022

Algorithme

CHAPITRE 1 : La Récursivité

Définition : Une fonction (procédure) est dite récursive si elle s'appelle elle-même.

exemple : factoriel d'un entier naturel n.

$$Fact(n) = \begin{cases} 1 & Si\ (n=0)\ ou\ (n=1) \\ n * fact(n-1) & Sinon \end{cases}$$

$fact(0) = fact(1) = 1$

$fact(4) = 4*3*2*1$

$fact(5) = 5*4*3*2*1 = 5*fact(4)$

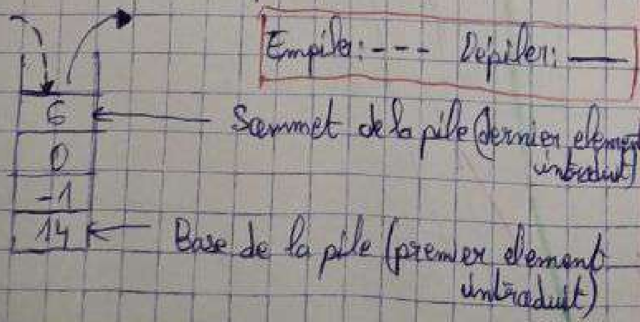
```
Fonction Fact(n:entier): entier;
Debut
   Si (n<=1) alors retourner 1; {Cas de base}
   Sinon retourner n*fact(n-1); {Cas générale}
Finsi;
Fin
```

*** Notion de Pile et File :**

Ce sont des structures de données ordonnées, mais qui ne permettent l'accès qu'à une seule donnée. Elles servent à memoriser des choses en attente de traitement. Elles sont souvent associés à des algorithmes récursifs.

La Pile :

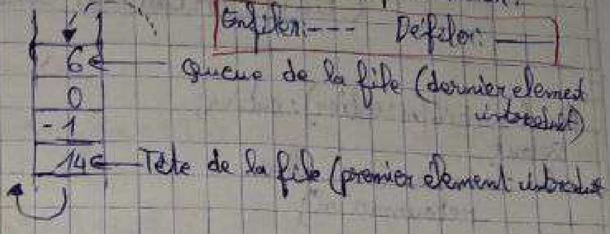
Les piles (Stack lifo : Last in First out) correspondent à une pile d'assiettes : on prend toujours l'element supérieur, le dernier empilé.



Applications :
1. memoriser les pages web visités
2. evaluation d'expressions arithmetiques
3. Appels de fonctions récursives
4. Memoriser les noeuds visités dans un algorithme de parcours en profondeur.

La File :

Les files (on dit aussi queues) (stack FIFO : First in First out) correspondent aux files d'attente : on prend toujours le premier élément donc le plus ancien.



Applications :
1. Traiter les requêtes par un serveur d'impression dans l'ordre qu'elles arrivent.
2. Memoriser les noeuds visités dans un algorithme de parcours en largeur.

*** Types de récursivité :**

I/ Première Classification :

1. La récursivité Terminale : un module récursif est dit terminal si aucun traitement n'est effectué à la remontée d'un Appel récursif (sauf le retour de resultat)

exemple : Reste de la division (a mod B = (a-B) mod B)

```
Fonction Reste (a: entier, B: entier): entier;
 Debut
  Si (a<B) alors retourner (a);
  Sinon
       retourner (Reste(a-B, B));
  Finsi;
 Fin
```

Reste(13,4) → Reste(9,4) → Reste(5,4) → Reste(1,4) → a<B → 1

Phase descente

Sur le dernier Appel (1,4) retourner le Resultat sans faire de calcule.

2. La récursivité non Terminale : un module récursif est dit non terminal si le resultat de l'appel récursif est utilisé pour réaliser un Traitement (en plus du retour du résultat).

exemple : Factoriel $n! = n*(n-1)!$

```
Fonction Fact(n: entier): entier
Debut
Si (n<=1) alors
      retourner (1);
Sinon
  retourner (n*Fact(n-1));
Finsi;
Fin
```

fact(4) = 24

4 * fact(3) = 24

3 * fact(2) = 6

2 * fact(1) = 2

1

II/ Deuxième classification:

1- La récursivité simple: où l'algorithme fait un seul appel récursif dans son corps.
exemple: (celui de la fonction factorielle).

2- La récursivité multiple: où l'algorithme fait plusieurs appels récursifs dans son corps.
exemple: le calcul de la suite de Fibonacci

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

```
Fonction Fib(n: entier): entier;
Debut
 Si (n=0) ou (n=1) alors
      retourner n;
 Sinon
      retourner Fib(n-1) + Fib(n-2);
 Finsi;
Fin
```

3- La récursivité mutuelle: où un module P appelle un autre module q qui fait à son tour un autre appel au module P.
exemple: La définition de la parité d'un entier peut être écrite de la manière suivante:

$$Pair(n) = \begin{cases} \text{vrai si } n=0 \\ Impair(n-1) \text{ Sinon} \end{cases} \quad \text{et} \quad Impair(n) = \begin{cases} \text{faux si } n=0 \\ Pair(n-1) \text{ Sinon} \end{cases}$$

```
Fonction Pair(n:entier): booléen;
Debut
Si(n=0) alors
  retourner vrai;
Sinon
retourner Impair(n-1);
Finsi;
Fin
```

```
Fonction Impair(n:entier): booléen;
Debut
Si (n=0) alors
 retourner faux;
Sinon
retourner Pair (n-1);
Finsi;
Fin
```

4- La récursivité imbriquée: consiste à faire un appel récursif à l'intérieur d'un autre appel récursif.
exemple: La fonction d'ackermann

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

```
Fonction Akermann(m:entier, n:entier): entier;
Debut
Si (m=0) alors retourner n+1;
Sinon Si (n=0) et (m>0) alors retourner akermann(m-1,1);
   Sinon retourner Akermann(m-1, Akermann(m,n-1));
Finsi;
Finsi;
Fin
```

* Schéma général de la récursivité

```
Algorithme Recur (Paramètres);
Si (Test - Arrêt) alors
    instruction du point d'arrêt
Sinon
    instructions;
    Recur (Paramètres changés);
    instructions
finsi
Fin
```

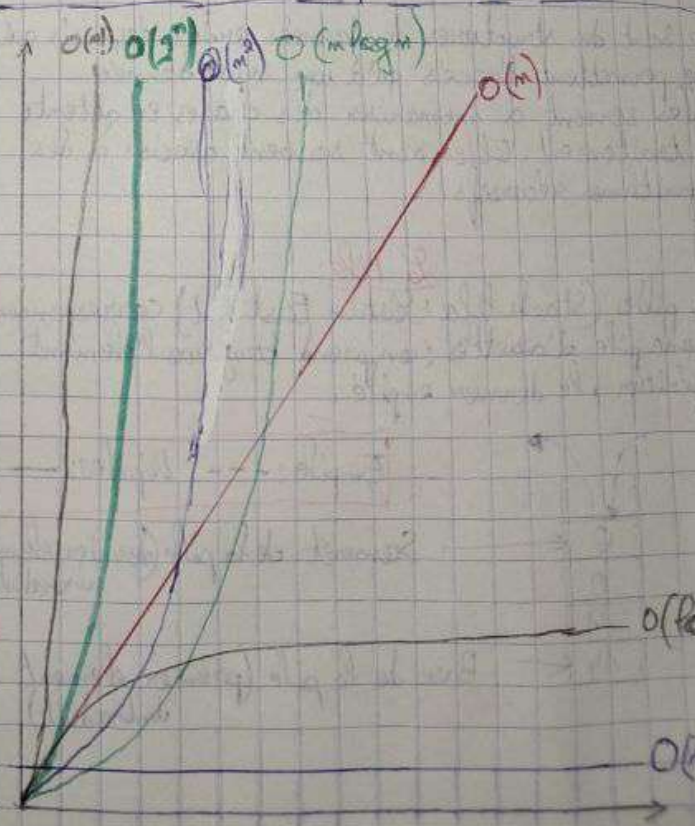
08/01/2023

CHAPITRE 2: La Complexité.

La complexité d'un ~~Algorithme consiste~~ en l'étude de la quantité de ressources (de temps ou d'espace) nécessaire à l'exécution de cet algorithme.
~~La complexité~~ temporelle d'un algorithme quantifie ~~le temps nécessaire à~~

* Type de Complexité:

Notation grand O	Nom
$O(1)$	complexité Constante
$O(\log(n))$	// logarithmique
$O(n)$	// linéaire
$O(n \log n)$	// quasi-linéaire
$O(n^2)$	// quadratique
$O(2^n)$	// exponentielle
$O(n!)$	// factorielle



* Règles de Calcul de la Complexité :

(1) Pour calculer la Complexité grand O d'un Algorithme il faut compter le nombre d'operations de base qu'il effectue comme :
- operation arithmétique ou logique (+, *, et, ou, ...)
- Operation d'affectation ($x \leftarrow 10$)
- Verification d'une Condition ($x > a$)
- operation d'entrée/Sortie (Ecrire ou lire)

➜ La Complexité de chaque operation de base est Constante ou O(1)

(2) La Complexité d'une Boucle est la complexité du bloc interne dans la Boucle multipliée par le nombre de fois que le bloc interne est répété

exemple (1)

Pour $i \leftarrow 1$ à n pas 1 faire ← Affectation : O(1)
 ecrire ("entrer un nombre a :") ← Ecriture : O(1)
 Lire (a) ← lecture : O(1)
 ecrire ("i x a :", a * i) ← ecriture O(1), operation Ari O(1)
Finpour
} x(n)

La Complexité de l'exemple ci-dessus est : O(5n)

(3) La Complexité de la structure Si / Sinon correspond à la Complexité de la Condition (O(1)) plus la Complexité la plus grande entre "Alors" et "Sinon"

exemple (2)

Si $n < 5$ alors ← Verification d'une Condition : O(1)
 ecrire ("Hello world!") ← ecriture O(1)
Sinon
 Pour $i \leftarrow 1$ à n pas 1 faire ← Affectation O(1)
 ecrire ("Hello world!") ← ecriture O(1) } O(2n)
 Finpour
Finsi

La Complexité de l'exemple ci-dessus est :
$O(1) + Max(O(1), O(2n)) \rightarrow O(2n+1)$

(4) La Complexité d'une Séquence de deux blocs d'instructions est egale à la plus grande des Complexités des deux blocs

exemple (1)
exemple (2) } $Max(O(5n), O(2n+1))$
⇓
$O(5n)$

- La Complexité d'un algorithme est un calcul de ses performances asymptotiques dans le pire des Cas.

- Asymptotique nous nous interessons aux données très Volumineuses car les petites valeurs ne sont pas informatives.

(5) Les Constantes multiplicatives sont remplacées par 1.
expl : $2 n^4 \Rightarrow 1 n^4$

(6) Les Constantes additives sont annulées.
expl : $n^4 + 8 \Rightarrow n^4$

(7) Le Terme le plus fort est Conservé
expl : $n^4 + n^2 + n \Rightarrow n^4$

exemple :

$O(3n^2 + 8n + 5) \overset{(5)}{\Rightarrow} O(1n^2 + 1n + 1) \overset{(6)}{\Rightarrow} O(1n^2 + 1n) \overset{(7)}{\Rightarrow} O(n^2)$

$O(2^n + 8n^3 + 13n^2) \overset{(5)}{\Rightarrow} O(2^n + 1n^3 + 1n^2) \overset{(6)}{\Rightarrow} O(2^n + 1n^3 + 1n^2) \overset{(7)}{\Rightarrow} O(2^n)$

$O(4n + 33 \log n + 1) \overset{(5)}{\Rightarrow} O(n + \log n + 1) \overset{(6)}{\Rightarrow} O(1n + 1 \log n) \overset{(7)}{\Rightarrow} O(n)$

* Arbre qui calcul le nombre de noeuds :

```
Type Arb_bin = ^noeud
     noeud = record
      clé : element
      FG, FD : Arb_bin
end
```

} déclaration ABR

```
Fonction Taille (A : Arb_bin) : entier;
Debut
  Si A = Nil alors //arbre Vide
     retourner (0)
  Sinon
     retourner (1 + Taille (A^.FG) + Taille (A^.FD)
  finsi
Fin
```

* Fonction qui calcule le nombre de feuilles :

8.

Le

```
Fonction Nbfeuilles (A : Arb_bin) : entier
  Debut
    Si A = Nil alors
       retourner (0);
    Sinon
       Si (A^.FG = Nil et A^.FD = Nil) alors
          retourner (1);
       sinon
          retourner (Nbfeuilles (A^.FG) + Nbfeuilles (A^.FD))
       finsi
    finsi
  fin
```

* Fonction qui calcule la Hauteur d'un Arbre :

```
Fonction Hauteur (A: Arb_bin): entier
  Debut
    Si A = Nil alors
      retourne (0);
    Sinon
      Si (A^.FG = Nil et A^.FD = Nil) alors
        retourne (1)
      Sinon
        retourner (1 + Max (Hauteur (A^.FG), Hauteur (A^.FD)))
      fsi
    fsi
  Fin
```

* Procedure qui permet l'insertion d'une clé X dans un arbre :

```
Procedure insertion (Var A: Arb_bin, X: element);
 Var P; Arb_bin;
  Debut
    Si A = Nil alors
      New (P);
      P^.clé := clé;
      P^.FG := Nil;
      P^FD := Nil;
      A := P;
    Sinon
      Si x < A^.clé alors
        insertion (A^.FG, X);
      Sinon
        Si x > A^.clé alors
          insertion (A^.FD, X);
        Sinon
          Ecrire ('x existe deja');
fin fsi  fsi  fsi
```

* Procedure de Suppression d'une clé x :

```
Procedure (Var A: arb-bin; x: element);
Var noeud, loc: arb-bin;
debut
    Si A <> nil alors
        Si x < A^.clé alors
            Supprimer (A^.FG, x);
        Sinon
            Si x > A^.clé alors
                Supprimer (A^.FD, x);
            Sinon // x = A^.clé
                Si A^.FG = nil alors
                    debut
                        loc := A;
                        A := A^.FD;
                        Dispose(loc);
                    Fin
                Sinon
                    Si A^.FD = nil alors
                        debut
                            loc := A;
                            A := A^.FG;
                            dispose (loc);
                        fin
                    Sinon // x = A^.clé
                        debut
                            noeud := Max (A^.FG);
                            A^.clé := noeud^.clé;
                            Supprimer (A^.FG, noeud^.clé);
                        fin
                finsi
            finsi
        finsi
    finsi
```

* Fonction qui recherche le Minimum dans un ABR:

```
Fonction Min (A: Arb_bin): entier;
Debut
  Si A = nil alors
  |  retourner (0);
  Sinon
  |  Si A^.FG = nil alors
  |  |  retourner (A);
  |  Sinon
  |  |  retourner (Min (A^.FG));
  |  fsi
  fsi
fin
```

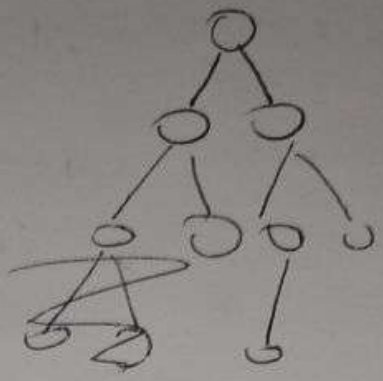
* Fonction qui supprime le Max de l'arbre:

```
Fonction SuppMax (A: Arb_bin): entier;
Var Pere, E: Arb_bin; x: entier;
Debut
  Si A <> nil alors
    Si A^.FD = nil alors  //Cas particulier où le Max est la racine
          Debut
          |  x := A^.clé
          |  E := A;
          |  A := A^.FG;
          |  dispose (E);
          |  retourner (x);
          fin
    Sinon
       Debut  Tantque A^.fd <> nil faire
                 |  Pere := A;
                 |  A := A^.FD;
              fait
       //on est sur le Max, la cellule pointée par //A à supprimer
          E := A; X := A^clé; Père^.fd := nil; dispose (E); retourner (x);
  finsi fsi  fin
```

* Arbre General :

I/ Representation standard :

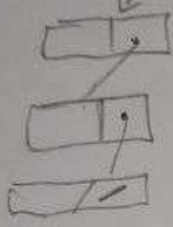
Num	1	2	3	4
Noeud	a	b	c	d
fils 1	2			
fils 2	3			
fils 3	4			



II/ Semi Statique / Semi-dynamique :

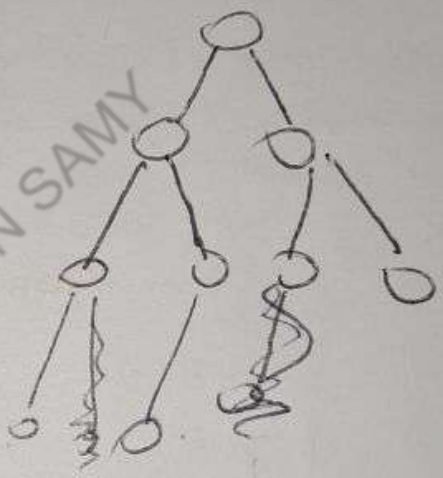
1	2	3	4	5
a	b	c	d	e

Chaque parente en lui fait leurs fils.



III/ Statique dense :

index	1	2	3	4
Noeud	a	d	g	h
indexP	/	1	1	2



* Arbre binaire :

I/ Representation Standard

Num			
Noeud	a	b	
filsG	2		
filsD		3	

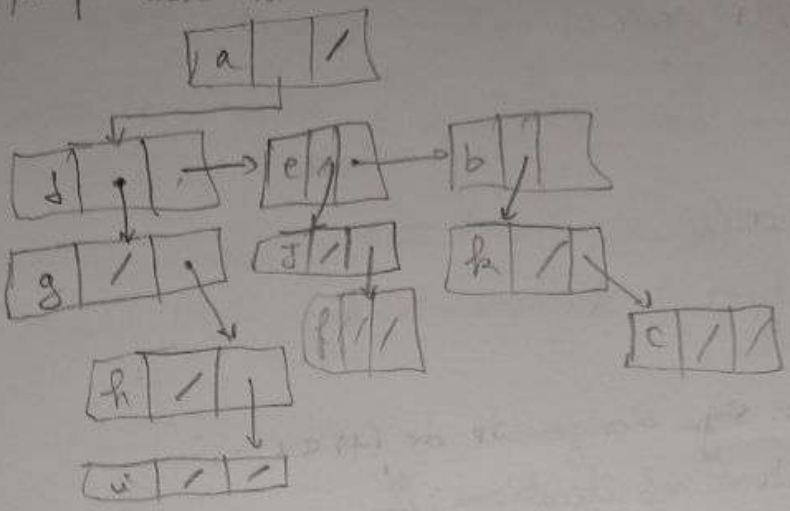
$$h = 2^n - 1$$

II/ Sequentielle Standard

2K : FG 2K+1 : FD

a	b											
1	2	3	4	5								

III/ dynamique par liste chaînée



* Tas Max :
le père > ses fils

* Tas Min :
Le Père < ses fils.

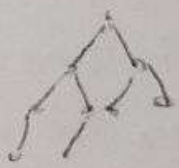
8.

Le * Arbre parfait :
c'est un arbre binaire où chaque nœud admet 2 ou 0 fils (pas 1)

* Arbre Complet :
c'est un arbre binaire où tous les niveaux sont remplis sauf le dernier

* La Taille d'un ABR :
Nombre Total de ses nœuds.

* Hauteur d'un ABR :
Le Nombre de nœuds du chemin le plus long dans l'arbre.



* Parcours préfixé : RGD

* Parcours infixé : GRD
on le note quand on peut plus descendre à gauche.

* Parcours postfixé : GDR
on le note quand on ne peut plus descendre ni à gauche ni à droite.

- **Exemple :**

*Complexité :

$O(\log n) < O(n^{\frac{1}{2}}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(n^4)$

logarithmique — racine Carré — linéaire — quasi-quadratique — quadratique — cubique — $n > 3$ polynomiale.

* La Boucle Pour : $((u - 1) + 1) * nbr\ Instr$

* La Boucle Tantque : nb iteration * nb instr
  + Test de Sortie

* Si le Compteur ~~sla~~ s'augmente de (+) ou (−) :
  donc nb iterations : $\frac{N}{Pas}$

* Si le Compteur s'augmente de (x) ou (/) :
  nb iterations $= \frac{\ln(N)}{\ln(Pas)} = \log(N)$

8 * La Boucle Repeter :
  ~~nb inst~~ nb itera * nb instru

L

* Facteur d'equilibre :

$$|FE(x)| \leq 1$$

$$FE(x) = H(fd(x)) - H(fg(x))$$

Recursivité :

Terminale ne contient aucun Traitement après l'appel recursif mais non Terminale le résultat de l'appel recursif est utilisés pour realiser un traitement.

* Complexité de la recherche d'une clé :
  $O(h)$

* Fonction qui
  Fonction M
  Debut
  Si A
  Sinon
  fsi
  fin

* Fonction
  Fonctio
  Var
  Debut
  Si