Algorithmique

tosaz@lri.fr

Cormen Leiserson Rirest (référence biblique)

Froideveaux Gaudel Soria

Mr Al Khwarismi[1] fut l'inventeur de l'algorithmique (d'après son nom)

Un algo résoud des problèmes P(D) avec D en entrée.

Exemple : D => 2 entiers, P calculer la somme, le produit.
D => tableau d'entier, P = les trier
D => un texte, P = vérifier que c'est un programme C correct et fournir l'executable.
D => (Programme, Entrée), P = savoir si P lancé sur E s'arrête (en supposant que la mémoire dispo est illimitée)

DEFINITION Algorithmique

C'est un procédé de calcul automatique et effectif qui résoud un problème P et qui s'arrête sur toutes les entrées.

REMARQUES

- Pas d'ordinateur ou d'informatique dans la définitions
- « qui s'arrête sur toutes données.)
  Problème d'arrêt. Par exemple Lancer P sur E et on observe
  - Si je vois P/E s'arrête alors je dis oui
  - Sinon … Je ne peux pas répondre !

  Ce n'est pas un algo parce que ça ne s'arrête pas toujours. Il s'agit d'un semi algo.
- Automatique executable sans réflechis. -> l'ordinateur peut le faire.
- Effectif : tout doit pouvoir être calculer effectivement.
  - Si dieu existe
    - N=0
    - Sinon N=1

    CE N'EST PAS UN ALGORITHME.
  - Arrêt tentative 2

[1] Savant Perse ~830

- X_n = {(P,E), |P|<=n , |E|<=n}
  X_n est un ensemble fini
- Y_n = {(P,E)| | |P|<=n ^ |E|<=n ^ P/E s'arrête}
  Y_n est fini.
- F(n) = max temps d'execution de P/E
  (P/E)€Y_n
- **Arrêt :** on lance P/E.
  - Si on observe que ça s'arrete, on répond oui.
  - Si on au bout de f(max(|P|, |E|) +1 ca ne s'arrête toujours pas on répond non.

COMMENT ON CALCULE F ??
CE N'EST PAS UN ALGO. CE N'EST PAS EFFECTIF.

- Comment décrire le problème, les données et l'algo ?
  - Le français ? C'est ambigu.
  - Techniques formelles. C'est el génie logiciel.
  - Pseudo code.
- Efficacité des algorithmiques : COMPLEXITE
  - Les ordres de grandeurs pour faire que $\pm\infty$ où à la rigueur qui retient $\geq \gamma \geq 0$

    F=O(g) f pas plus grand que g.

    $$\exists A f \leq A g$$

    $$\frac{f}{g} majoré\ en\ 100$$

    $$F = \Omega\ g\ \exists \lambda \geq 0$$

    …

    Texte en Français, sans gros chiffres. Sans plus de 6 caractère entre 2 mots.

    N_b = nombre de bits

    N_c= nombre de caractères

    N_m= nombre de mots

N_p= nombre de paragraphes

$N_b = 8n_c$ $\quad\quad$ $n_b = \Theta(n_c)$

$N_m <= n_c <= 32\ n_m$ $\quad\quad$ $n_c = \Theta(n_m)$

$N_p <= n_m$ nom majorable $\quad\quad$ $n_m \neq \Theta(n_p)$

- Taille d'un objet
  - Bit_taille (objet) = nombre de bit pour le coder.
  - Une grandeur g est une taille si et seulement si (bit_taille)

    N_b, n_c, n_m sont des tailles, mais pas n_p

    - Remarque : Le taille d'un entier est le nombre de chiffres pour l'écrire en base 2 (ou autre base).

      Il faut un $\log_2 n$ pour écrire n en base 2.

      Si n s'écrit avec k chiffres en base b :

      $$b^{k-1} < n < b^k$$

      Bit_taille(n) = $\log_2 n$

      Tout log convient $log_{10} n = \frac{log2}{log10} log_{10} n$

      La taille de T[1..N] d'entiers.

      En vrai, $\sum_{i=1}^{N} ln.T[i]$

      En pratique, on néglige l'impact des tailles des entiers. En info, tout int sera sur 32 ou 64 bits

      On prétendra que la taille est N.

      Bit_taille, mais avec quel codage ?

      - Le codage naturel.

$$Complexité\ (^{algo}_{données}, implementation, machine) = le\ temps\ de\ calcul$$

On veut ne pas s'occuper des 2 derniers. Et on ne veut pas passer trop de temps sur chaque données.

$$complexité\ en\ opération\ ^{algo}_{données} = nombres\ d'opérations faites.$$

Exemple : compter les doublons d'une liste :

```
Cpt <- 0

Pour i de 1 à N

        Pour j de i+1 à N

                Si ei=ej

                Alors c++

        Fpour

Fpour

Rendre cpt
```

Complexité en test « ei=ej »

$$\frac{N(N-1)}{2} \sim \frac{N^2}{2}, en\ \theta(N^2)$$

Complexité en « cpt++ » entre 0 et N(N-1)/2 fois en O(N²)

La différence entre Θ et O. dans les cas, on majore mais on ne minore que dans le premier.

Complexité en augmentation de i environ n fois.

Un opération et dite fondamentale si et seulement si la complexité de cette opération est Θ de la complexité sur une machine. Exemple : « Ei=j » est une OPÉRATION FONDAMENTALE, « cpt++ » n'est pas une OPÉRATION FONDAMENTALE.

Compléxité en faonction de la taille.

Compléxité au pire : $n \mapsto max_{\ d\ =n}\ complexité \sin d\ \ ou\ \ d\ \ \leq n$

Complexité moyenne : $n \mapsto moyenne_{\ d\ =n}\ complexité\ sur\ \ d$

Il faut une proba pas évident si les données est une graphe pour le tri si on peut avoir multi-occurrences

Vocabulaire

Complexité

- Constante Θ(1)
- Logarithmique Θ(lnn)
- Linéaire Θ (n)
- Quadratique Θ(n²)
- Polynomiale Θ ∃ $comp = O\ \ n^p\ \ lnn,\ \ \overline{n}, nlnn,\ c'est\ polynomial$

- Exponentiel.
    - C est exponentiel si et seulement si C n'est pas polynomiale.

$$^{\exists q}\ln C = O\ \ n^{q}$$

F(n)/n	10	30	100	1000	10^6	10^9
N	*	*	*	0.001	1	16 minutes
Nlnn	*	*	*	0.007	14	5h
N^2	*	0.001	0.01	1j	11J	32 000 ans
2^n	0.001	18minutes	4.10^{16}	10^{287}	10^{300000}	$10^{300000000}$
2^{2n}	10^{294}	10^{300000}	$10^{300000000}$	*	*	*

F le nombre d'opérations à faire.

P un problème.

Y a-t-il un algorithme pour ce problème ?

Vendredi 24 septembre 2010

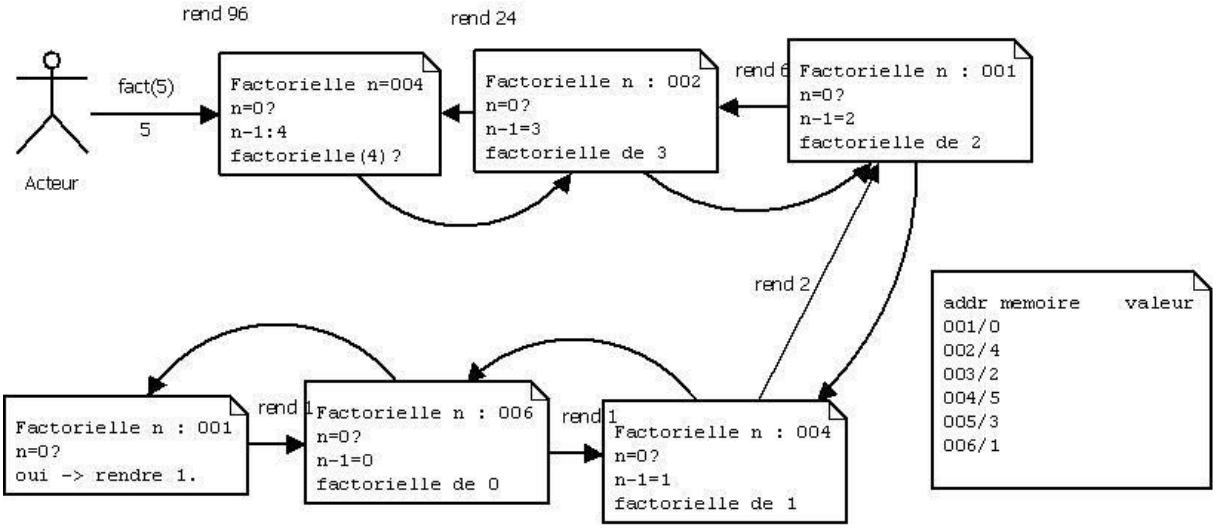
Algo récursif

Le fait qu'une fonction puisse s'appeler elle-même.

Exemple : Factorielle(n)

Si n= 0 alors rendre 1
sinon rendre n*Factorielle (n-1)

Comment ça se passe ?



Quid si je lance fact(-1) ?

- Sur une marchine de Turing : ne s'arrête pas.
- Sur un ordinateur réèl : Arret brutal : « out of memory »

Chercher x dans une liste. : chercher (X,L)

```
Si L vide alors rendre faux // X n'est pas dedans !
Sinon
        Si X=teteDeListe(L)
                alors rendre vrai
        Sinon
                Rendre chercher(X, Suite(L))
```

Fibonacci : $F_0 = F_1 = 1$;
Pour tout $n \geq 2$, $F_n = F_{n-1} + F_{n-2}$

```
Fibo(n)
If n=0 ou n=1
        alors rendre 1
sinon rendre fibo(n-1)+fibo(n-2)
```

X_n = Complexité en appels terminaux (Appele F_0, F_1)

Chaque calcul refait tous les calculs avants. (pas de mémoires des résultats précédants). Donc donc F100 prendra 2 fois plus de temps que F99

$$\overline{2}^{n} = 2^{n/2} \leq Xn \leq 2^{n}$$

X_n est exponentiel

X0=1
X1=1
$X_n = X_{n-1} + X_{n-2}$ avec X_{n-1} = le nombre d'appele quand je lance l'appelle F(n-1)

D'où $X_n = F1$

Yn = le nombre d'appeks
$Y_0 = Y_1 = 1$
$Y_n = 1 + Y_{n-1} + Y_{n-2}$
Zn = Yn + 1

Z0 = Z1 =Z
$Z_n = Z_{n-1} + Z_{n-2}$

$Z_n = 2F_n$
$Y_n = 2F_{n-1}$

Complexité de notre fonction Ffubonacci avec une complexité de :

$$\Theta = \frac{1+B}{2}^{n}$$

Est-ce que Fibo(100) produit un outOfMemory ?
Non, car il y a uniquement 100 appels récursif. Les re-calculs utilisent des appels, mais la mémoires utilisés par ceux-ci sont rendu en même temps que le résultat. La mémoire est réutilisé.

Comment calculer F(n) ?

On mémorise les resultats.

(ici complexité en temps en n et en espace en 1)

```
T[0.n]
Pour i de 0 à n
        T[i]← -1
Fpour

T[0]← 1
T[1]← 1

F(n ,inoutT[], out res)
Render res.
```

Mieu : Quitte à remplir T[], autant le faire itérativement et stocker chaque résultat dans une case.

Mais Fibonacci n'a pas besoin de tous les résultats précédents, uniquement des 2 dernier. Donc Mieu :

```
F(n)
Fp← 1
Fsuivant ← 1
/*F(p)=Fp, F(p+1)=F(pre) */
Pour I de 1 à n
        Temp ← Fsuiv
        Fsuiv ← Fsuiv + Fp
Fp← temps
Fpour
Render F(p)
```

Ou en recursif :

```
F(p, Fp, Fsuiv, n)
Si p=n alors
        Render Fp
Sino
        Rendre F(p+1, Fsuiv, Fp+Fsuiv, n)
Fsi
Fibonacci (nà
Rendre (O,1,1,n)
```

Quand appel recursif est Terminal (il n'y a rien à faire derrière), l'appelant peut se fermer directement et demander de l'appelé de passer le résultat directement dessus.

La complexité en espace est Θ(n) si le compilateur ne le fait pas. Θ(1) sinon.

Propreté du pseudo-code

Expression : Quelque chose qui a une valeur.
Par exemple : 0, Fact(7), P.val (le champs val de l'objet P)

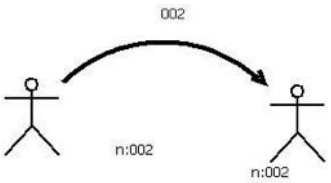
Instruction : quelque chose à faire, une action.
exemple : Affectation, If, then, else, print(-)

> Interdiction de faire la confusion !
>
> I← j++
> c'est une expression, on met dans I la valeur que j aura après l'incrémentation
> c'est une instruction, elle stocke J+1 dans j.
> c'est un truc batard à ne pas faire !

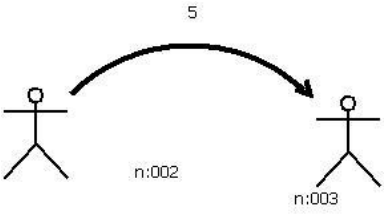
Fonction (Argument$_1$ à Argument$_n$) → de retour.
prend des arguments. Et REND un résultat. (exemple : Factorielle)
elle ne modifie PAS ses argument.

Procedure (Argument$_1$ à Argument$_n$)
elle Agit et ne rend pas de résultat.

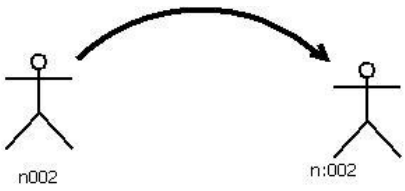
Argument (passage de)
in f(nà données de l'appelant à l'appelé.



002:5 en pratique il y a copie des variables



l'appelant donne une variabible que l'appelé peut modifier. Passage par référence.



k'appelant envoie une var à l'appelé que l'appelé va re-initialiser par var.