Un premier rappel: comment mesurer l'efficacité d'un algorithme

Complexité des algorithmes

Analyser la complexité d'un algorithme?

Analyser la complexité d'un algorithme doit permettre de mesurer l'efficacité de l'algorithme.

On pourrait d'abord se poser la question:

Qu'est-ce qu'un algorithme efficace?

Qu'est-ce qu'un algorithme efficace?

Tentative de définition: Un algorithme est efficace si, une fois implémenté, il s'exécute rapidement sur toute donnée "réelle".

- Première remarque: attention à l'implémentation
- Deuxième remarque: on devrait pouvoir mesurer l'efficacité indépendamment de la plate-forme
- Troisième remarque: qu'est-ce qu'une donnée réelle? Souvent les problèmes d'efficacité surgissent quand la taille des données grandit...
- Quatrième remarque: ne prend en compte que la ressource "temps".

Qu'est-ce qu'un algorithme efficace?

Deuxième Tentative de définition?
Un algorithme efficace est un algorithme qui se comporte de façon raisonnable indépendamment de la plate-forme même pour des données de taille assez grande...

On proposera une définition un peu plus précise tout à l'heure!

Analyser la complexité d'un algorithme c'est:

Exprimer le coût de l'algorithme - la quantité de ressources utilisées - *en fonction de la taille des données*

Idée: on essaie de prévoir le comportement en fonction de la taille des données.

QUELLES RESSOURCES?

la mémoire: *complexité spatiale*

le temps : *complexité temporelle*

ou le nb de processeurs, les communications,....

COMMENT MESURER LA TAILLE DES DONNÉES

. a priori la taille des données est **le nombre de bits de leur représentation en binaire**;

. même si dans certains cas, on étudie la complexité en fonction d'une autre notion de taille, comme le nombre d'éléments pour une liste, la dimension pour une matrice, . . .

COMMENT MESURER LA TAILLE DES DONNÉES

Q? Quelle est la taille d'un entier n?

Soit:
quel est le nombre de bits de la représentation en binaire d'un entier n?

environ $\log_2 n$: $\lceil \log_2 n + 1 \rceil$

QUEL EST LE COÛT DE L'ALGORITHME A POUR UNE DONNÉE d - NOTÉ $cout_A(d)$-?

. Pour évaluer ce coût indépendamment de la machine (physique) , on utilise un *modèle* de machines séquentielles "classiques", en général la RAM ("Random Access Machine").

. On considère sauf exception que chaque instruction "simple' ($+, *, -$, affectation, comparaison, accès mémoire ...) a un coût de 1: coût uniforme, i.e. indépendant de la taille de ses opérandes par opposition au coût logarithmique où on tient compte de la taille des données.

. On se borne souvent à "compter" certaines instructions: comparaisons pour un tri par comparaisons, accès à la mémoire externe pour un tri externe, opérations arithmétiques de base pour des produits de matrices...

QUELLE COMPLEXITÉ?

Pour deux données de taille n, le coût peut être différent!

- *La complexité dans le meilleur des cas:*

$$Inf_A(n) = inf\{cout_A(d)/d \text{ de taille } n\}$$

- *La complexité dans le pire des cas:*

$$Sup_A(n) = sup\{cout_A(d)/d \text{ de taille } n\}$$

- *La complexité en moyenne:* pour la définir, il faut disposer pour tout n d'une mesure de probabilité p sur l'ensemble des données de taille n;

$$Moy_A(n) = \sum_{d \text{ de taille } n} p(d) * cout(d)$$

En résumé...

Quand on parle de la complexité d'un algorithme sans préciser laquelle, c'est souvent de la complexité temporelle dans le pire des cas qu'on parle.

ORDRES DE GRANDEUR...

On ne calcule pas en général la complexité exacte mais on se contente de calculer son *ordre de grandeur asymptotique* voire de borner celui-ci.
Par exemple, si on fait $n^2 + 2n - 5$ opérations, on retiendra juste que l'ordre de grandeur est n^2. On utilisera donc les notations classiques sur les ordres de grandeur.

ORDRES DE GRANDEUR: DÉFINITIONS

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$f \in O(g)$ Ssi_{def} $\exists c \in \mathcal{R}^{+*}, \exists \mathcal{A}$, tels que
$\forall n > A, f(n) \leq c * g(n)$

On dit que f est dominée asymptotiquement par g.
On notera souvent $f = O(g)$.

ORDRES DE GRANDEUR: DÉFINITIONS

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$f \in \Omega(g)$ Ssi_{def} $\exists C \in \mathcal{R}^{+*}$, tels que
$\forall n > A, C * g(n) \leq f(n))$

On notera souvent $f = \Omega(g)$.
On dit que f domine g.
On a alors $g = O(f)$.

ORDRES DE GRANDEUR: DÉFINITIONS

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$f \in \Theta(g)$ Ssi_{def}
$\exists c, C \in \mathcal{R}^{+*}, \exists \mathcal{A}$,tels que
$\forall n > A, C * g(n) \leq f(n) \leq c * g(n)$

On dit alors que f et g sont de même ordre de grandeur asymptotique.
On notera souvent $f = \Theta(g)$.
On a $f = \Theta(g)$ Ssi $f = O(g)$ et $f = \Omega(g)$.

ORDRES DE GRANDEUR: DÉFINITIONS

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}$:

$f \in o(g)$ Ssi_{def} $\forall \epsilon \in \mathcal{R}^{+*}, \exists \mathcal{A}$, tels que
$\forall n > A, f(n) \leq \epsilon * g(n)$

On dit que f est négligeable asymptotiquement devant g.
On notera souvent $f = o(g)$

ORDRES DE GRANDEUR: EXEMPLES

$.n^3 + 3n + 7 \in \Theta(n^3)$

$.5 * n^3 + 3n + 7 \in \Theta(n^3)$

$.5 * n^3 + 3n + 7 \in O(n^3)$

$.5 * n^3 + 3n + 7 \in O(n^4)$

$.\log n \in o(n)$

ORDRES DE GRANDEUR: EXEMPLES

.$n * \log n \in \Theta(n^2)$? NON!

.$n * \log n \in O(n^2)$? Oui!

.$2^n \in \Theta(n^3)$? NON!

.$n^3 \in \Theta(2^n)$? Non!

.$n^3 \in O(2^n)$? Oui!

REMARQUE

Evaluer l'ordre de grandeur asymptotique du coût de l'algorithme en fonction de la taille des données plutôt que la complexité exacte se justifie si les données manipulées sont de grande taille.
Il ne faut pas négliger trop vite les constantes .

Q?: à partir de quelle taille de données, un algorithme de complexité exacte $200 * n * \log_2 n$ sera-t-il plus intéressant qu'un algorithme en n^2 ?

UN PEU DE VOCABULAIRE: UN ALGORITHME EST DIT...

. en temps **constant** si sa complexité dans le pire des cas est bornée par une constante.

. **linéaire** (resp. linéairement borné) si sa complexité (dans le pire des cas) est en $\Theta(n)$ (resp. $O(n)$).

. **quadratique** (resp. au plus quadratique) si sa complexité (dans le pire des cas) est en $\Theta(n^2)$ (resp. en $O(n^2)$).

. **polynomial** ou polynomialement borné, si sa complexité (dans le pire des cas) est en $O(n^p)$ pour un certain p.

. (au plus) **exponentiel** si elle est en $O(2^{n^p})$, pour un certain $p > 0$.

ETRE PRATICABLE OU NE PAS ÊTRE PRATICABLE?...

TELLE EST LA QUESTION QU'ON S EXPOSERA SOUVENT

Par "convention", un algorithme est dit **praticable** si il est polynomial, c.à.d. si sa complexité temporelle dans le pire des cas est polynomiale.

Les limites de la convention

- Si un algorithme est exponentiel dans le pire des cas, il peut être polynomial en moyenne; si les pires cas sont exceptionnels, il peut être praticable ...en pratique: certains algorithmes impraticables selon la définition ci-dessus sont ... pratiqués tous les jours (comme le simplexe).

-L'exécution d'un algorithme dont la complexité moyenne est de l'ordre de grandeur de n^5 microsecondes prendrait 30 ans si $n = 1000$.

MAIS CE N'EST PAS UNE SI MAUVAISE CONVENTION:

+ En pratique, il s'avère que la plupart des algorithmes polynomiaux ont un comportement asymptotique équivalent à un polynôme de faible degré, 2 ou 3.

+ De plus, la classe des algorithmes polynomiaux a de bonnes propriétés de clôture (par exemple, une "séquence" de deux algorithmes polynomiaux est polynomiale).

+ Enfin, et surtout, elle est indépendante du modèle séquentiel "classique" choisi: un algorithme polynômial pour le modèle des machines de Turing, le sera pour une RAM et vice-versa .

ORDRES DE GRANDEUR: EXEMPLES

Exemples de temps d'exécution en fonction de la taille de la donnée et de la complexité de l'algorithme, si on suppose qu'une instruction est de l'ordre de la μs;

T.\C.	$\log n$	n	$n \log n$	n^2	2^n
10	$3\mu s$	$10\mu s$	$30\mu s$	$100\mu s$	$1000\mu s$
100	$7\mu s$	$100\mu s$	$700\mu s$	$1/100s$	10^{14} *siècles*
1000	$10\mu s$	$1000\mu s$	$1/100s$	$1s$	*astronomique*
10000	$13\mu s$	$1/100s$	$1/7s$	$1,7mn$	*astronomique*
100000	$17\mu s$	$1/10s$	$2s$	$2,8h$	*astronomique*

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
 int max= t[0];
  for (i=1; i<=n-1; i++)
     if (max <t[i]) max=t[i];
```

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
 int max= t[0];
  for (i=1; i<=n-1; i++)
     if (max <t[i]) max=t[i];
```

Algorithme en $\Theta(n)$, donc linéaire.

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
  for (i=0; i<n-1; i++)
     for (j=0; j<n-1-i; j++)
       if (t[j+1] <t[j]) echanger(t[j],t[j+1]);
```

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
  for (i=0; i<n-1; i++)
     for (j=0; j<n-1-i; j++)
       if (t[j+1] <t[j]) echanger(t[j],t[j+1]);
```

Algorithme en $\Theta(n^2)$, donc quadratique.

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
 boolean permute=true;
 int last=n-1;
  while permute {
     permute=false;
     for (j=0; j<last; j++)
       if (t[j+1] <t[j])
          {permuter=true; echanger(t[j],t[j+1]);
     last=last-1;}
}
```

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
 boolean permute=true;
 int last=n-1;
  while permute {
     permute=false;
     for (j=0; j<last; j++)
       if (t[j+1] <t[j])
         {permuter=true; echanger(t[j],t[j+1]);
     last=last-1;}
}
```

dans le meilleur des cas en $\Theta(n)$

QUELQUES EXEMPLES:

```
// t tableau de n entiers
// n entier >0
 boolean permute=true;
 int last=n-1;
  while permute {
     permute=false;
     for (j=0; j<last; j++)
       if (t[j+1] <t[j])
         {permute=true; echanger(t[j],t[j+1]);
     last=last-1;}
}
```

dans le meilleur des cas en $\Theta(n)$
dans le pire des cas en $\Theta(n^2)$
Algorithme en $\Theta(n^2)$, donc quadratique.

QUELQUES EXEMPLES: MULTIPLICATION À LA RUSSE ... OU À L'ÉGYPTIENNE

Quelle est la complexité de:

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
while (b>0){
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b
```

Quelques exemples: multiplication à la russe ... ou à l'égyptienne

Quelle est la complexité de:

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
while (b>0){
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b
```

Le nombre d'opérations élémentaires est $\Theta(log_2(y))$.

Quelques exemples: multiplication à la russe ... ou à l'égyptienne

Quelle est la complexité de:

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
while (b>0){
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b
```

Le nombre d'opérations élémentaires est $\Theta(log_2(y))$.

L'algorithme est linéaire en coût uniforme, quadratique en coût logarithmique.....

QUELQUES EXEMPLES:

Soit l'algorithme suivant pour tester si un nombre est premier:

```
//n entier >1
boolean Premier(int n){
  int j =sqrt(n); \\racine carrée entière
  for (int i = 2; i<=j; i++)
    if (n mod i=0)  return false;
  return true;
}
```

ATTENTION À LA TAILLE DE LA DONNÉE!

```
//n entier >1
boolean Premier(int n){
   int j =sqrt(n); \\racine carree entiere
   for (int i = 2; i<=j; i++)
     if (n mod i=0)  return false;
   return true;}
```

Cet algorithme est-il polynomial? Non!!

ATTENTION À LA TAILLE DE LA DONNÉE!

```
//n entier >1
boolean Premier(int n){
  int rac =sqrt(n); \\racine carrée entière
  for (int i = 2; i<=rac; i++)
    if (n mod i=0)  return false;
  return true;
}
```

supposons que le coût d'une division soit constant, indépendant de la taille de la donnée, donc en $\Theta(1)$, alors la complexité dans le pire des cas de l'algo est $\Theta(\sqrt{(n)})$..

ATTENTION À LA TAILLE DE LA DONNÉE!

```
//n entier >1
boolean Premier(int n){
  int rac =sqrt(n); \\racine carrée entière
  for (int i = 2; i<=rac; i++)
    if (n mod i=0)  return false;
  return true;
}
```

supposons que le coût d'une division soit constant, indépendant de la taille de la donnée, donc en $\Theta(1)$, alors la complexité dans le pire des cas de l'algo est $\Theta(\sqrt{(n)})$..

mais la taille de la donnée $|d|$ est $\log_2 n$

donc le coût est en $\Theta(2^{|d|/2})$: l'algorithme est exponentiel.

A RETENIR SUR LA COMPLEXITÉ DES ALGORITHMES:

. Quand on parle de la Complexité c'est souvent la complexité temporelle dans le pire des cas

. Praticable=polynomial

. Evaluer l'ordre de grandeur de la complexité d'un algorithme ... n'exempte pas de tester et d'expérimenter

. Attention à la taille de la donnée!