Algorithmique et Structures de données
Complexité des algorithmes

Abdelkamel, Ben Ali

Université d'El Oued

Décembre 2020

Licence 2 d'informatique

Théorie de complexité

- Théorie de complexité, a pour but de :
  - Fournir les outils mathématiques nécessaires à l'analyse des performances d'un algorithme
  - Avoir une idée de ce qui est faisable et infaisable actuellement
- Ce qui permet de :
  - Améliorer les performances des problèmes faciles
  - Savoir comment aborder les problèmes difficiles

Complexité d'un algorithme

- Un programme qui s'exécute consomme de l'espace mémoire et du temps de calcul :
  - **Complexité temporelle** (plus intéressante)
  - **Complexité spatiale**
- Le but est d'exprimer les performances d'un algorithme à l'aide de fonctions qui dépendent de la **taille des données**

$$\mathbf{T_{ALG}(n) = ?}$$

Mesures de la complexité

Pour mesurer la complexité temporelle d'un algorithme, on s'intéresse plutôt aux opérations les plus coûteuses (FLOPS) pour le problème de calcul particulier :

- Racine carrée, Log, Exp, Addition réelle . . .
- Comparaisons d'éléments dans le cas des tris . . .

et on calcule le nombre d'opérations fondamentales exécutées par l'algorithme.

Problème	Opérations fondamentales
Recherche d'une valeur dans un tableau	comparaisons entre les éléments du tableau
Multiplication des matrices réelles	Multiplication scalaire
Addition des entiers binaires	Opération binaire

Mesures de la complexité

Le temps de l'exécution dépend de la taille de l'entrée. On veut considérer seulement la taille essentielle de l'entrée.
Cela peut être par exemple :

- le nombre d'éléments combinatoires dans l'entrée
- le nombre de bits pour représenter l'entrée
- etc ...

Problème	Taille de la donnée
Recherche d'une valeur dans un tableau	Nombre d'éléments dans le tableau
Multiplication des matrices réelles	Dimensions des matrices
Addition des entiers binaires	Nombre des bits pour représenter les entiers

- Complexité moyenne
- Complexité dans le pire des cas

- **Exemple** : Pour la recherche séquentielle (exercice TD) dans un tableau de n éléments, on fait $n/2$ comparaisons en moyenne et n comparaisons dans le pire des cas.

Soit A un algorithme
D_n l'ensemble des entrées de taille n
$I \in D_n$ une entrée

Définition :

1. $cout_A(I) =$ Nb d'opérations fondamentales exécutées par A sur I
2. la complexité de A en pire cas :

$$Max_A(n) = Max\{cout_A(I), I \in D_n\}$$

3. Soit Pr une distribution de probabilités sur D_n

   la complexité de A en moyenne :

$$Moy_A(n) = \sum_{I \in D_n} Pr[I].cout_A(I)$$

- **Exemple : Recherche Séquentielle**

```
int index_of_value(double v[], int n, int x)
{
    int j = 0;

    while (j < n)
    {
        if (v[j] == x)
            return j;
        j = j + 1;
    }

    return -1;
}
```

Opération de base : comparaison de x avec un élément de v.

Complexité en pire cas de RS :

$$Max_{RS}(n) = n$$

Complexité en moyenne de RS :

On suppose que :

- tous les éléments sont distincts
- $Pr[x \in v] = q$
- si $x \in v$ alors toutes les places sont équiprobables

Pour $1 \leq i \leq n$ soit

$$I_i = \{(v, x), x \in v\}$$

et

$$I_{n+1} = \{(v, x), x \notin v\}$$

On a :

$$Pr[I_i] = q/n \text{ pour } 1 \leq i \leq n \text{ et } cout_{RS}(I_i) = i$$

et

$$Pr[I_{n+1}] = 1 - q \text{ et } cout_{RS}(I_{n+1}) = n$$

$$
\begin{aligned}
Moy_{RS}(n) &= \sum_{i=1}^{n+1} Pr[I_i] \cdot cout_{RS}(I_i) \\
&= \sum_{i=1}^{n} \frac{q}{n}(i) + (1-q) \cdot n = \frac{q}{n} \sum_{i=1}^{n}(i) + (1-q) \cdot n \\
&= \frac{q}{n} \cdot \frac{n(n+1)}{2} + (1-q) \cdot n = (1-\frac{q}{2}) \cdot n + \frac{q}{2}
\end{aligned}
$$

- **si** $q = 1$ **alors** $Moy_{RS}(n) = (n+1)/2$
- **si** $q = 1/2$ **alors** $Moy_{RS}(n) = (3n+1)/4$

Notations utilisées

Il faut comparer les taux d'accroissement de différentes fonctions qui mesurent les performances d'un algorithme.

Notation "o"
On dit que $f(x) = o(g(x))$ pour $x \longrightarrow \infty$ si

$$\lim_{x \longrightarrow \infty} \frac{f(x)}{g(x)} = 0$$

Ce que veut dire que f croît plus lentement que g quand x est très grand. Par exemple :

$$x^2 = o(x^5) \qquad \sin x = o(x)$$
$$14.079\sqrt{x} = o(x/2 + 7\cos x) \qquad 23 \log x = o(x^{0.002})$$

Notation "O"
On dit que $f(x) = O(g(x))$ s'il existe k et x_0 tels que :

$$x > x_0 \quad f(x) < kg(x)$$

La notation o est plus précise que O, mais O est plus facile à calculer et suffisant. Par exemple :

$$\sin(x) = O(x) \quad \sin x = O(1)$$

Un classement des fonctions

$$\mathbf{\log(n) \ll \sqrt{n} \ll n \ll n\log(n) \ll n^2 \ll}$$
$$\mathbf{n^3 \ll 2^n \ll \exp(n) \ll n! \ll n^n \ll 2^{2^n}}$$

$\mathbf{O(1)}$	Constant
$\mathbf{O(\log(n))}$	Logarithmique
$\mathbf{O(n)}$	Linéaire
$\mathbf{n\log(n)}$	$n\log(n)$
$\mathbf{O(n^2)}$	Quadratique
$\mathbf{O(n^3)}$	Cubique
$\mathbf{O(2^n)}$	Exponentiel

- Complexité polynomiale $\Rightarrow$ souvent réalisable
  $\exists k > 0, \quad f(n) \in O(n^k)$.
- Complexité exponentielle $\Rightarrow$ en général irréalisable
  $\exists b > 1, \quad b^n \in O(f(n))$.
- Complexité doublement exponentielle
  par exemple : $f(n) = 2^{2^n}$.
- Complexité sous-exponentielle
  par exemple : $f(n) = 2^{\sqrt{n}}$.

$$\log(n) \ll \sqrt{n} \ll n \ll n\log(n) \ll n^2 \ll$$

$$n^3 \ll 2^n \ll \exp(n) \ll n! \ll n^n \ll 2^{2^n}$$

Comparaison entre les différents groupes de fonctions

	2	16	64	256
$\log \log \mathbf{n}$	0	2	2.58	3
$\log \mathbf{n}$	1	4	6	8
$\mathbf{n}$	2	16	64	256
$\mathbf{n} \log \mathbf{n}$	2	64	384	2048
$\mathbf{n^2}$	4	256	4096	65536
$\mathbf{2^n}$	4	65536	1.84467e+19	1.15792e+77
$\mathbf{n!}$	2	2.0923e+13	1.26887e+89	8.57539e+506
$\mathbf{n^n}$	4	1.84467e+19	3.9402e+115	3.2317e+616
$\mathbf{2^{n^2}}$	16	1.15792e+77	1.04438e+1233	2.00353e+19728

Méga $= 10^6(2^{20})$, Géga $= 10^9(2^{30})$, Tera $= 10^{12}(2^{40})$, Péta $= 10^{15}(2^{50})$

4 Ghz pendant 1 an $= 1,26 \times 10^{17}$

4 Ghz pendant 4 Milliards d'années $= 5 \times 10^{26}$

Applications

- Recherche dans un tableau
  - Recherche séquentielle
    - $O(N)$
    - 10000 comparaisons dans un tableau de 10000 éléments
  - Recherche dichotomique
    - $O(\log_2 N)$
    - 14 comparaisons dans un tableau de 10000 éléments

- Multiplication de matrices (exercice TD)
  Problème classique : $C = A \times B$ avec A et B matrices carrés d'ordre n.

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

```
for (i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
    {
        C[i][j] = 0.;
        for (k = 0; k < n; k++)
        {
            C[i][j] += A[i][k]*B[k][j];
        }
    }
}
```

Complexité : $O(N^3)$

- Multiplication de polynômes (Exercice TD)

$$P = \sum_{i=1}^{n} a_i x^i \qquad Q = \sum_{i=1}^{m} b_i x^i$$

$$PQ = \sum_{i=1}^{n+m} c_i x^i \quad \text{avec} \quad c_k = \sum_{i+j=k} a_i b_j$$

d'où le programme

```
for (i = 0; i < n + m; i++)
   C[i] = 0.;
for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
        C[i + j] += A[i]*B[j];
```

Et sa **complexité** en $O(mn)$

Applications

- Tri par sélection (Exercice TD)

```
void triSelection (float t[], int n)
{
    int i, j, i_min;
    for (i = 0; i < n - 1; i++)
    {
        i_min = i;
        for (j = i + 1; j < n; j++)
            if (t[j] < t[i_min])
                i_min = j;
        if (i_min > i)
            echanger(t, min , i);
    }
}
```

On fait $(n-1)+(n-2)+(n-3)+\cdots+2+1$ comparaisons

On fait donc $n(n-1)/2$ comparaisons $\rightarrow$ $O(n^2)$

Applications

- Exemple de tri par sélection

i

i_{min}

18	3	10	25	9	3	11	13	23	8
3	18	10	25	9	3	11	13	23	8
3	3	10	25	9	18	11	13	23	8
3	3	8	25	9	18	11	13	23	10
3	3	8	9	25	18	11	13	23	10
3	3	8	9	10	18	11	13	23	25
3	3	8	9	10	11	18	13	23	25
3	3	8	9	10	11	13	18	23	25
3	3	8	9	10	11	13	18	23	25
3	3	8	9	10	11	13	18	23	25