

CHAPITRE 2

COMPLEXITE D'UN ALGORITHME

Objectif :

- P un problème
- M une méthode pour résoudre le problème P

Algorithme : description de la méthode dans un langage algorithmique

- On veut Evaluer l'efficacité de la méthode comparer M avec une autre méthode M' indépendamment de l'environnement (machine, système, compilateur.

- Evaluation du nombre d'opérations élémentaires en fonction de la taille des données, de la nature des données.
- Notations :
 - ✓ n taille des données,
 - ✓ $T(n)$: nombre d'opérations élémentaires

Configurations caractéristiques

Meilleur cas,

Pire des cas

Cas moyen.

1. Définition :

La complexité : d'un algorithme est la quantité de ressources n nécessaires pour traiter des entrées. On la voit comme une fonction de la taille de l'entrée.

Les principales ressources mesurées sont le temps (nombre d'instructions utilisées) et l'espace (quantité d'espace mémoire nécessaire).

On distingue plusieurs types d'analyses de complexité : l'analyse dans le

meilleur des cas, le pire des cas et en moyenne. Pour ce cours on étudie exclusivement le pire des cas.

En informatique, la **complexité dans le pire des cas**, ou **complexité dans le cas le plus défavorable**, mesure la complexité (par exemple en temps ou en espace) d'un algorithme dans le pire des cas d'exécution possibles. Elle est exprimée comme une fonction de la taille de l'entrée de l'algorithme. Implicitement, on cherche à construire des algorithmes s'exécutant en utilisant le moins de ressources possible (e.g. le plus vite possible), et il s'agit par conséquent d'une borne supérieure des ressources requises par l'algorithme.

Par exemple, la complexité en temps dans le pire des cas correspond au temps d'exécution le plus long que puisse avoir l'algorithme, et permet d'en garantir la terminaison.

En informatique, dans le cadre de l'analyse de la complexité des algorithmes, la **complexité dans le meilleur des cas** correspond à la complexité (par exemple en temps) d'un algorithme dans le cas d'exécution le plus favorable possible. Elle est exprimée comme une fonction de la taille de l'entrée de l'algorithme. Implicitement, on cherche à construire des algorithmes s'exécutant en utilisant le moins de ressources possible (e.g. le plus vite possible), et il s'agit par conséquent d'une borne inférieure des ressources requises par l'algorithme.

Par exemple, la complexité en temps dans le meilleur des cas correspond au temps d'exécution le plus court que puisse avoir l'algorithme.

La **complexité en moyenne d'un algorithme** est la quantité d'une ressource donnée, typiquement le temps, utilisée par l'algorithme lors de son exécution pour traiter une entrée tirée selon une distribution donnée. Il s'agit par conséquent d'une moyenne de la complexité, pondérée entre les différentes entrées possibles selon la distribution choisie. Le plus souvent, on ne précise pas la distribution et on utilise implicitement une distribution uniforme (*i.e.* une loi uniforme discrète dans le cas discret), c'est-à-dire que toutes les entrées sont considérées comme équiprobables.

C'est une mesure importante pour l'analyse de la complexité des algorithmes, et elle s'oppose à la complexité dans le pire des cas qui considère la complexité maximale de l'algorithme sur toutes les entrées possibles.

Principales complexités

- $\mathcal{O}(1)$: temps constant,
- $\mathcal{O}(\log(n))$: complexité logarithmique (Classe L),
- $\mathcal{O}(n)$: complexité linéaire (Classe P),
- $\mathcal{O}(n \log(n))$
- $\mathcal{O}(n^2)$, $\mathcal{O}(n^3)$, $\mathcal{O}(n^p)$: quadratique, cubique, polynome

Evaluation de $T(n)$ (séquentiel)

Somme des coûts.

$$\left. \begin{array}{l} \text{Traitement1} \\ T_1(n) \end{array} \right\} T(n) = T_1(n)$$

Evaluation de $T(n)$ (embranchement)

Max des coûts.

$$\left. \begin{array}{l} \text{si } \langle \text{condition} \rangle \text{ alors} \\ \quad \text{Traitement1} \quad T_1(n) \\ \text{sinon} \end{array} \right\} \max(T_1(n), T_2)$$

Evaluation de $T(n)$ (boucle)

Somme des coûts des passages successifs

$$\left. \begin{array}{l} \text{tant que } \langle \text{condition} \rangle \text{ faire} \\ \quad \text{Traitement} \quad T_i(n) \\ \text{fin faire} \end{array} \right\} \sum_{i=1}^k T_i(n)$$

$T_i(n)$: coût de la $i^{\text{ème}}$ itération

fonction FunctionRecursive (*n*)

```
1  si (n > 1) alors  
2      FunctionRecursive(n/2),   coût T(n  
3      Traitement(n),           coût C(n  
4      FunctionRecursive(n/2),   coût T(n
```

Equation réursive

$$T(n) = 2 * T(n/2) + C(n)$$