Complexité Algorithmique

Dr Ahmed Wade

awade@ept.sn

Objectifs

- Comprendre la notion de complexité d'un algorithme.
- Savoir estimer la complexité d'un algorithme itératif ou récursif.
- Connaître les différent algorithme de tri et estimer leur complexité.

Question

Qui veut gagner 1 000 000 de dollars ?

Sommaire

1. Problème et Algorithme
2. Décidabilité
3. Terminaison
4. Complexité
5. Complexité d'une entrée
6. Complexité d'un algorithme
7. Complexité d'un problème

Problème

Un problème est une relation binaire reliant un ensemble d'instances (entrées) à un ensemble de solutions (sorties)

Exemple

- Problème du calcul du PGCD
  - A chaque couple {a,b}, on associe le PGCD de a et b
- Problème du tri
  - A chaque tableau d'entiers non triés, on associe le tableau d'entiers triés
- Problème de coloration de graphe
  - A chaque graphe G et chaque entier k, on associe la réponse « oui » si le graphe G peut être coloré avec k couleur et « non » sinon

- Un problème en général n'est pas fonctionnel :
  - une entrée peut admettre plusieurs sorties possibles

Problème

Un problème est une relation binaire reliant un ensemble d'instances (entrées) à un ensemble de solutions (sorties)

Exemple

- Problème du calcul du PGCD
  - A chaque couple {a,b}, on associe le PGCD de a et b
- Problème du tri
  - A chaque tableau d'entiers non triés, on associe le tableau d'entiers triés
- Problème de coloration de graphe
  - A chaque graphe G et chaque entier k, on associe la réponse « oui » si le graphe G peut être coloré avec k couleur et « non » sinon

- Un problème en général n'est pas fonctionnel :
  - une entrée peut admettre plusieurs sorties possibles

Problème

Un problème est une relation binaire reliant un ensemble d'instances (entrées) à un ensemble de solutions (sorties)

Exemple

- Problème du calcul du PGCD
  - A chaque couple {a,b}, on associe le PGCD de a et b
- Problème du tri
  - A chaque tableau d'entiers non triés, on associe le tableau d'entiers triés
- Problème de coloration de graphe
  - A chaque graphe G et chaque entier k, on associe la réponse « oui » si le graphe G peut être coloré avec k couleur et « non » sinon

- Un problème en général n'est pas fonctionnel :
  - une entrée peut admettre plusieurs sorties possibles

5

Algorithme

- Un algorithme est une représentation de la résolution par calcul d'un problème
- C'est un énoncé (une suite d'instructions) dans un langage bien défini
- Son implémentation consiste à automatiser son utilisation à l'aide d'une machine
- Un algorithme résout un problème si pour toute instance du problème, toute exécution de l'algo fournit une sortie spécifiée par le problème

Algorithme

- Un algorithme est une représentation de la résolution par calcul d'un problème
- C'est un énoncé (une suite d'instructions) dans un langage bien défini
- Son implémentation consiste à automatiser son utilisation à l'aide d'une machine
- Un algorithme résout un problème si pour toute instance du problème, toute exécution de l'algo fournit une sortie spécifiée par le problème

Algorithme

- Un algorithme est une représentation de la résolution par calcul d'un problème
- C'est un énoncé (une suite d'instructions) dans un langage bien défini
- Son implémentation consiste à automatiser son utilisation à l'aide d'une machine
- Un algorithme résout un problème si pour toute instance du problème, toute exécution de l'algo fournit une sortie spécifiée par le problème

Sommaire

1. Problème et Algorithme
2. Décidabilité
3. Terminaison
4. Complexité
5. Complexité d'une entrée
6. Complexité d'un algorithme
7. Complexité d'un problème

Décidabilité

- C'est quoi un problème de décision ?

Décidabilité

- C'est quoi un problème de décision ?
- Un problème de décision est un problème où chaque instance a pour solution soit « oui » soit « non »

Décidabilité

- C'est quoi un problème de décision ?
- Un problème de décision est un problème où chaque instance a pour solution soit « oui » soit « non »
- Un problème de décision est décidable s'il existe un algo qui pour toute instance I

Décidabilité

- C'est quoi un problème de décision ?
- Un problème de décision est un problème où chaque instance a pour solution soit « oui » soit « non »
- Un problème de décision est décidable s'il existe un algo qui pour toute instance I
  - Se termine en un nombre fini d'étapes
  - Répond « oui » si la solution de I est « oui »
  - Répond « non » si la solution de I est « non »

Décidabilité

- C'est quoi un problème de décision ?
- Un problème de décision est un problème où chaque instance a pour solution soit « oui » soit « non »
- Un problème de décision est décidable s'il existe un algo qui pour toute instance I
  - Se termine en un nombre fini d'étapes
  - Répond « oui » si la solution de I est « oui »
  - Répond « non » si la solution de I est « non »
- Question : peut-on décider tous les problèmes de décision ?

Sommaire

1. Problème et Algorithme
2. Décidabilité
3. Terminaison
4. Complexité
5. Complexité d'une entrée
6. Complexité d'un algorithme
7. Complexité d'un problème

Terminaison

À un même problème

- différentes solutions algorithmiques peuvent être proposées
  - certaines peuvent ne jamais terminer
  - boucles infinies
- La première qualité attendue d'un algorithme est sa terminaison
- Un second critère permet de les comparer et ainsi d'en distinguer de meilleures que d'autres
  - le temps
  - l'espace

Terminaison

À un même problème

- différentes solutions algorithmiques peuvent être proposées
  - certaines peuvent ne jamais terminer
  - boucles infinies
- La première qualité attendue d'un algorithme est sa terminaison
- Un second critère permet de les comparer et ainsi d'en distinguer de meilleures que d'autres
  - le temps
  - l'espace

Terminaison

À un même problème

- différentes solutions algorithmiques peuvent être proposées
  - certaines peuvent ne jamais terminer
  - boucles infinies
- La première qualité attendue d'un algorithme est sa terminaison
- Un second critère permet de les comparer et ainsi d'en distinguer de meilleures que d'autres
  - le temps
  - l'espace

Terminaison

- L'une des qualités attendus d'un algorithme est qu'il termine
  - il n'admette aucune instance pour laquelle l'exécution rentre dans une boucle infinie
- L'un des problèmes difficiles en informatique est de décider si un algorithme termine

Terminaison

Est ce que l'algorithme **syracuse** termine sur chaque entrée n ?

Algorithm 1

fonction syracuse (a : entierLong) : mot

    tant que a<>1 **faire**

        si a est pair **alors**

            a$\leftarrow \frac{a}{2}$

        sinon

            a$\leftarrow \frac{3.a+1}{2}$

        fin si

    fin tant que

fin fonction

Terminaison

- La communauté scientifique n'a pas réussi à prouver que algorithme **syracuse** termine sur chaque entrée de n
  - Même si il termine pour un très grand nombre de n
- Il n'existe pas de méthode universelle pour décider si un algo termine
- Le problème
  - Terminaison
  - Entrée : un algorithme A
  - Sortie : le booléen indiquant si A termine ou non
- Est un problème indécidable (Incalculable)

Terminaison

- La communauté scientifique n'a pas réussi à prouver que algorithme **syracuse** termine sur chaque entrée de n
  - Même si il termine pour un très grand nombre de n
- Il n'existe pas de méthode universelle pour décider si un algo termine
- Le problème
  - Terminaison
  - Entrée : un algorithme A
  - Sortie : le booléen indiquant si A termine ou non
- Est un problème indécidable (Incalculable)

Terminaison

- La communauté scientifique n'a pas réussi à prouver que algorithme **syracuse** termine sur chaque entrée de n
  - Même si il termine pour un très grand nombre de n
- Il n'existe pas de méthode universelle pour décider si un algo termine
- Le problème
  - Terminaison
  - Entrée : un algorithme A
  - Sortie : le booléen indiquant si A termine ou non
- Est un problème indécidable (Incalculable)

Attention

- Ne prenez pas prétexte de l'indécidabilité de la terminaison pour produire des algo qui ne terminent pas
- Il est possible lorsque vous écrivez un algorithme de vous assurer de façon formelle qu'il termine

Semi-décidabilité

- Un problème de décision est semi-décidable s'il existe un algorithme qui, pour toute instance I dont la solution est « oui », se termine en un nombre fini d'étapes en répondant « oui ».
- **Remarque** : le problème de l'arrêt est semi-décidable car il suffit de simuler l'exécution de l'algorithme A sur l'instance I et de répondre « oui » dès qu'il s'arrête

Sommaire

1. Problème et Algorithme
2. Décidabilité
3. Terminaison
4. Complexité
5. Complexité d'une entrée
6. Complexité d'un algorithme
7. Complexité d'un problème

Définition d'un problème

- Un grand nombre de problèmes fournissent dans la définition mathématique des objets mentionnés une solution algorithmique

Exemple

- Problème Puissance
- Entrée : un réel x, un entier a
- Sortie : le réel x^a

Solution itérative

- Si nous utilisons la définition de x^a vu en troisième ou en quatrième
  - x multiplié par lui même n fois

```
fonction puissance1 (x : réel; a : entier) : réel

    res ← 1.0;
    faire a fois
        res ← res · x
    retourner res
```

Solution itérative

- Si nous utilisons la définition de x^a vu en troisième ou en quatrième
  - x multiplié par lui même n fois

---

fonction puissance1 (x : réel ; a : entier) : réel

---

res $\leftarrow$ 1.0 ;
faire a fois
    res $\leftarrow$ res $\cdot$ x
retourner res

Solution récursive

- Si vous préférez la relation récursive apprise en seconde, x^a est égal à
  - 1 si a=0
  - $x \cdot x^{a-1}$ sinon

```
fonction puissance2 (x : réel; a : entier) : réel

    si (a=0) alors
        res ← 1.0;
    sinon
        res ← x · puissance2 (x, a-1);
    retourner res
```

Solution récursive

- Si vous préférez la relation récursive apprise en seconde, x^a est égal à
  - 1 si a=0
  - $x \cdot x^{a-1}$ sinon

---

fonction puissance2 (x : réel; a : entier) : réel

---

si (a=0) alors
  res $\leftarrow$ 1.0 ;
sinon
  res $\leftarrow$ x $\cdot$ puissance2 (x, a-1) ;
retourner res

Comparaison des deux solutions

- Ces deux solution sont tous les deux corrects
- Si nous considérons les ressources temps et l'espace
- Est ce qu'elles sont équivalentes ?

Comparaison des deux solutions

- Ces deux solution sont tous les deux corrects
- Si nous considérons les ressources temps et l'espace
- Est ce qu'elles sont équivalentes ?
- Le second utilise d'avantage de ressources espace que le premier

Sommaire

1. Problème et Algorithme
2. Décidabilité
3. Terminaison
4. Complexité
5. Complexité d'une entrée
   - Complexité d'un booléen
   - Complexité d'un entier
6. Complexité d'un algorithme

Complexité d'une entrée

- Avant de définir la complexité d'un algorithme
- Il faut définir la complexité d'une entrée (instance)

Définition

La complexité (ou taille) d'une entrée est le nombre d'octets nécessaires à sa représentation (le nombre d'octets que l'entrée occupe en mémoire)

Complexité d'une entrée

- Avant de définir la complexité d'un algorithme
- Il faut définir la complexité d'une entrée (instance)

Définition

La complexité (ou taille) d'une entrée est le nombre d'octets nécessaires à sa représentation (le nombre d'octets que l'entrée occupe en mémoire)

Complexité d'une entrée

- Avant de définir la complexité d'un algorithme
- Il faut définir la complexité d'une entrée (instance)

Définition

La complexité (ou taille) d'une entrée est le nombre d'octets nécessaires à sa représentation (le nombre d'octets que l'entrée occupe en mémoire)

Complexité d'un booléen

- Un booléen nécessite un octet pour sa représentation
  - En fait un bit suffit
- Nous dirons qu'il est de complexité (ou taille) constante
- Une matrice n lignes, m colonnes de booléens
  - est de complexité $n \cdot m$

Complexité d'un booléen

- Un booléen nécessite un octet pour sa représentation
  - En fait un bit suffit
- Nous dirons qu'il est de complexité (ou taille) constante
- Une matrice n lignes, m colonnes de booléens
  - est de complexité $n \cdot m$

Complexité d'un entier

- La taille de l'entier est fixe
  - Pour le type *entier* ou *int*
    - Indices ou entier apparaissant dans des tableaux et des matrices
  - Pour le type *entierLong*
    - Les algo manipulant de long entiers (plusieurs centaines de bits)

Complexité d'un entier

- La taille de l'entier est fixe
  - Pour le type *entier* ou *int*
    - Indices ou entier apparaissant dans des tableaux et des matrices
  - Pour le type *entierLong*
    - Les algo manipulant de long entiers (plusieurs centaines de bits)

Complexité d'un entier

- La taille de l'entier est fixe
  - Pour le type *entier* ou *int*
    - Indices ou entier apparaissant dans des tableaux et des matrices
- La taille de l'entier n est égal à son logarithme
  - Pour le type *entierLong*
    - Les algo manipulant de long entiers (plusieurs centaines de bits)

Sommaire

1. Problème et Algorithme
2. Décidabilité
3. Terminaison
4. Complexité
5. Complexité d'une entrée
6. Complexité d'un algorithme
   - Complexité en temps dans le pire des cas
   - Complexité en temps en moyenne
   - Complexité en temps dans le meilleur des cas

Complexité d'un algorithme

Un algorithme est un objet qui à partir de toute entrée permet d'exécuter des instruction en consommant deux ressources

- du temps
  - Le temps est évalué en considérant le nombre d'instructions élémentaires devant être exécutées
  - Une instruction est élémentaire si elle peut être exécutée en un temps fixe
- de l'espace
  - L'espace est le nombre d'octets utilisé par l'exécution de l'algorithme

Complexité d'un algorithme

Un algorithme est un objet qui à partir de toute entrée permet d'exécuter des instruction en consommant deux ressources

- du temps
  - Le temps est évalué en considérant le nombre d'instructions élémentaires devant être exécutées
  - Une instruction est élémentaire si elle peut être exécutée en un temps fixe
- de l'espace
  - L'espace est le nombre d'octets utilisé par l'exécution de l'algorithme

Complexité d'un algorithme

Un algorithme est un objet qui à partir de toute entrée permet d'exécuter des instruction en consommant deux ressources

- du temps
  - Le temps est évalué en considérant le nombre d'instructions élémentaires devant être exécutées
  - Une instruction est élémentaire si elle peut être exécutée en un temps fixe
- de l'espace
  - L'espace est le nombre d'octets utilisé par l'exécution de l'algorithme

Complexité d'un algorithme

Un algorithme est un objet qui à partir de toute entrée permet d'exécuter des instruction en consommant deux ressources

- du temps
  - Le temps est évalué en considérant le nombre d'instructions élémentaires devant être exécutées
  - Une instruction est élémentaire si elle peut être exécutée en un temps fixe
- de l'espace
  - L'espace est le nombre d'octets utilisé par l'exécution de l'algorithme

Instruction élémentaire

- Le caractère élémentaire d'une instruction dépend des hypothèses initiales
  - Si l'on manipule des entier de taille fixe
    - l'addition d'entiers doit être considérée comme élémentaire
  - Si l'on considère des entiers de grande taille (plusieurs centaines d'octets)
    - l'addition ne peut pas être considérée comme élémentaire
    - elle nécessite autant de manipulations de bits qu'il en a de présents

Espace utilisée par l'algorithme

- Dans les algorithme itératifs
  - l'espace utilisé = celui nécessaire pour représenter les nouvelles variables utilisées
- Dans les algorithme récursifs
  - il faut gérer l'espace requis pour gérer l'ensemble des appels récursifs
  - on utilise une pile d'appel dont la taille est égale au nombre d'appels récursifs
- Quel est l'espace utilisé par l'algorithme puissance1 ?
- Quel est l'espace utilisé par l'algorithme puissance2 ?

Espace utilisée par l'algorithme

- Dans les algorithme itératifs
  - l'espace utilisé = celui nécessaire pour représenter les nouvelles variables utilisées
- Dans les algorithme récursifs
  - il faut gérer l'espace requis pour gérer l'ensemble des appels récursifs
  - on utilise une pile d'appel dont la taille est égale au nombre d'appels récursifs
- Quel est l'espace utilisé par l'algorithme puissance1 ?
- Quel est l'espace utilisé par l'algorithme puissance2 ?

Espace utilisée par l'algorithme

- Dans les algorithme itératifs
  - l'espace utilisé = celui nécessaire pour représenter les nouvelles variables utilisées
- Dans les algorithme récursifs
  - il faut gérer l'espace requis pour gérer l'ensemble des appels récursifs
  - on utilise une pile d'appel dont la taille est égale au nombre d'appels récursifs
- Quel est l'espace utilisé par l'algorithme puissance1 ?
- Quel est l'espace utilisé par l'algorithme puissance2 ?

Espace utilisée par l'algorithme

- Dans les algorithme itératifs
  - l'espace utilisé = celui nécessaire pour représenter les nouvelles variables utilisées
- Dans les algorithme récursifs
  - il faut gérer l'espace requis pour gérer l'ensemble des appels récursifs
  - on utilise une pile d'appel dont la taille est égale au nombre d'appels récursifs
- Quel est l'espace utilisé par l'algorithme puissance1 ?
- Quel est l'espace utilisé par l'algorithme puissance2 ?

Exemple

```
fonction puissance1 (x : réel ; a : entier) : réel
```

```
res ← 1.0 ;
faire a fois
    res ← res · x
retourner res
```

```
fonction puissance2 (x : réel ; a : entier) : réel
```

```
si (a=0) alors
    res ← 1.0 ;
sinon
    res ← x · puissance2 (x, a-1) ;
retourner res
```

Exercice

fonction decalage (n : entier) : entier

---

si $n=0$ **alors**

  retourner 0

tant que estPair(n) faire

  n $\leftarrow \frac{n}{2}$

retourner n

- Que fait l'algorithme decalage et quelle est sa complexité en temps ?

Solution

- L'algorithme **decalage** permet de supprimer les bits nuls de poids faible
- Il a une complexité en temps
  - constante pour tout entier impair
  - égales à $\log(n)$ pour toute puissance de 2

Complexité en temps dans le pire des cas

Définition

La complexité en temps dans le pire des cas d'un algorithme A est la fonction qui à tout entier n associe le nombre d'instructions élémentaires maximal exécutées par A sur des entrées de taille n

Complexité en temps dans le pire des cas

Définition

La complexité en temps dans le pire des cas d'un algorithme A est la fonction qui à tout entier n associe le nombre d'instructions élémentaires maximal exécutées par A sur des entrées de taille n

- La complexité en temps dans le pire des cas de l'algorithme décalage est $\log(n)$

Complexité en temps en moyenne

Définition

On appelle complexité en moyenne le nombre moyen d'opérations effectuées par l'algorithme sur l'ensemble des entrées de taille n

- C'est une exercice souvent plus délicat que dans le pire des cas
- Question : Donner la complexité moyenne de l'algorithme **decalage** sur des entiers de taille 3.

Complexité en temps dans le meilleur des cas

Définition

La complexité en temps dans le pire des cas d'un algorithme A est la fonction qui à tout entier n associe le nombre d'instructions élémentaires minimal exécutées par A sur des entrées de taille n

- Quelle est la complexité en temps dans le meilleur des cas de l'algorithme **decalage** ?

Complexité en temps dans le meilleur des cas

Définition

La complexité en temps dans le pire des cas d'un algorithme A est la fonction qui à tout entier n associe le nombre d'instructions élémentaires minimal exécutées par A sur des entrées de taille n

- Quelle est la complexité en temps dans le meilleur des cas de l'algorithme **decalage** ?
- La complexité en temps dans le meilleur des cas de l'algorithme **decalage** est constante

Complexité en temps dans le meilleur des cas

Définition

La complexité en temps dans le pire des cas d'un algorithme A est la fonction qui à tout entier n associe le nombre d'instructions élémentaires minimal exécutées par A sur des entrées de taille n

- Quelle est la complexité en temps dans le meilleur des cas de l'algorithme **decalage** ?
- La complexité en temps dans le meilleur des cas de l'algorithme **decalage** est constante
- Dans l'algorithme **decalage** elle est atteinte par les entiers impaires

Complexité en espace

- Pour définir la complexité en espace
  - remplacer le terme "nombre d'instructions élémentaire exécutées par A" par "nombre d'octets utilisées lors de l'exécution de A"
- Il existe notamment trois complexités en espace
  - dans le pire des cas
  - en moyenne
  - dans le meilleur des cas

Sommaire

1. Problème et Algorithme
2. Décidabilité
3. Terminaison
4. Complexité
5. Complexité d'une entrée
6. Complexité d'un algorithme
7. Complexité d'un problème
   - Compromis espace-temps

Complexité d'un problème

- Une idée naturelle est d'évaluer la complexité en temps d'un problème
- Un problème admet plusieurs solutions algorithmique
- On peut comparer les algo. sous le critère de leur complexité en temps
- Considérer la plus faible comme la complexité du problème

Compromis espace-temps

- Il n'existe pas parfois de meilleur algorithme
- Un problème ayant en entrée des entrées de taille n peut admettre par exemple
  - Une solution algorithmique de complexité dans le pire des cas
    - n^2 en temps
    - constante en espace
  - Une solution algorithmique de complexité dans le pire des cas
    - n en temps
    - n en espace
- Ces deux solutions ne sont pas comparables
- L'une ne peut être considérée comme meilleure que l'autre

Compromis espace-temps

- Il n'existe pas parfois de meilleur algorithme
- Un problème ayant en entrée des entrées de taille n peut admettre par exemple
  - Une solution algorithmique de complexité dans le pire des cas
    - n^2 en temps
    - constante en espace
  - Une solution algorithmique de complexité dans le pire des cas
    - n en temps
    - n en espace
- Ces deux solutions ne sont pas comparables
- L'une ne peut être considérée comme meilleure que l'autre

Compromis espace-temps

- Il n'existe pas parfois de meilleur algorithme
- Un problème ayant en entrée des entrées de taille n peut admettre par exemple
  - Une solution algorithmique de complexité dans le pire des cas
    - n^2 en temps
    - constante en espace
  - Une solution algorithmique de complexité dans le pire des cas
    - n en temps
    - n en espace
- Ces deux solutions ne sont pas comparables
- L'une ne peut être considérée comme meilleure que l'autre

Compromis espace-temps

- Il n'existe pas parfois de meilleur algorithme
- Un problème ayant en entrée des entrées de taille n peut admettre par exemple
    - Une solution algorithmique de complexité dans le pire des cas
        - n^2 en temps
        - constante en espace
    - Une solution algorithmique de complexité dans le pire des cas
        - n en temps
        - n en espace
- Ces deux solutions ne sont pas comparables
- L'une ne peut être considérée comme meilleure que l'autre

Exercice 1

- Recherche d'un élément "x" dans un tableau T de taille n
- Si x appartient à T alors retourner vrai, retourner faux sinon

fonction recherche1 (x : entier, T : tableau) : boolean

```
x = 0

Répéter
si x=T[i] alors
    retourner vrai
sinon
    i=i+1
jusqu'à i> taille (T)
retourner faux
```

Exercice 2

- Recherche d'un élément "x" dans un tableau T de taille n
- Si x appartient à T alors retourner vrai, retourner faux sinon

---

fonction recherche2 (x : entier, T : tableau) : boolean

```
n = taille(T)

Pour i allant de 1 à n faire
    si x=T[i] alors
        retourner vrai
retourner faux
```

Exercice 3

- Recherche d'un élément "x" dans un tableau ordonné T de taille n
- Si x appartient à T alors retourner vrai, retourner faux sinon
- Donner un algorithme et déterminer sa complexité