LETTRES SORBONNE UNIVERSITÉ

2019-2020

M1SOL020

Epistémologie de l'Informatique

TD5
Complexité

Gaël Lejeune, Sorbonne Université

Inspiré d'Agnès Delaborde 2015-2016

Objectifs

- Comprendre la notion de complexité
- En déduire l'importance de l'algorithme
- Parcourir un tableau de façon efficace

1 Itérations

- Rédiger l'algorithme (la séquence d'instructions) d'une boucle de 1 à 100. A chaque pas de la boucle, une variable n prend la valeur du compteur. En sortie de boucle, on affiche n.
- Réaliser le même algorithme pour une boucle de 1 à 100*1000
- Codez ces deux algorithmes en PYTHON
- Mesurez la différence dans le temps d'exécution entre ces deux scripts : utilisez la méthode `clock` de la librairie `time` en stockant la valeur de `time.clock()` avant la boucle et en le comparant à sa valeur après la boucle.
- Quelle est la complexité de l'algorithme (unité: nombre d'itérations) ?

2 Parcours de tableau

- Ecrivez un algorithme de création d'un tableau de 4 par 4. Chaque cellule contient une lettre, et chaque colonne représente l'alphabet (on s'arrête ici au bout de la 4ème lettre puisqu'il s'agit d'un tableau de 4 par 4).

A	A	A	A
B	B	B	B
C	C	C	C
D	D	D	D

- Ecrivez un algorithme de parcours du tableau pour compter toutes les occurrences des lettres "A" et "C". Cet algorithme ne doit parcourir qu'une seule fois le tableau.
- Rédigez le même algorithme, sauf que vous parcourrez le tableau une fois pour compter la présence de "A", et une seconde fois pour compter la présence des "C".
- Quelle est la complexité de chacun des deux algorithmes (unité: nombre de passage dans la boucle) ?
- Remarquez les temps d'exécution pour le(s) parcours du tableau.
- Si nous avions, au lieu d'un tableau 4 par 4, un tableau de 1000 par 2000, quelle serait la complexité des deux parcours ?

3 Analyses de complexité

Quelle est la différence de complexité entre ces deux algorithmes qui vérifient la présence d'un entier dans un tableau (unité: nombre de passages dans la boucle tantque) ?

Algorithme 1

```
e : entier; T: tableau [1...n] d'entiers
i <- 1; trouve <- FAUX
tantque trouve = FAUX et i <= n faire
    si e = T[i] alors
        trouve <- VRAI
    i <- i + 1
```

Algorithme 2

```
e : entier; T: tableau [1...n] d'entiers
i <- 1; trouve <- FAUX
tantque i <= n faire
    si e=T[i] alors
        trouve <- VRAI
        sortir de la boucle tantque
    i <- i + 1
```

- Indiquez, pour l'algorithme 1 ci-dessus, la complexité dans le meilleur des cas, dans le pire des cas, et la complexité moyenne.
- Quelle est la complexité de l'algorithme ci-dessous ? (unité: passage dans la boucle)

```
i <- 0; n <- 4; v <- 1;
tantque i <= n faire
    v <- v x 2
    afficher v
```

- Quelle est la complexité de l'algorithme ci-dessous ? (nombre d'opérations effectuées)

```
i <- 50;
y <- 4.0 / 3.0 * 4 * i
afficher y
```

- Peut-on rendre plus efficace l'algorithme ci-dessous ?

```
fonction affichtab(tableau: liste d\'elements)
    i <- 0; n <- taille max de tableau
    tantque i <= n faire:
        afficher valeur de tableau à l'indice i
        i <- i + 1
s = "tableau d'entiers"
t: tableau d'entiers[1...10]
affiche "Voici le contenu de " et affiche s
appel a la fonction affichtab(t)
s = "chaîne de caractères"
t: tableau de caractères["a"..."z"]
affiche "Voici le contenu de " et affiche s
appel a la fonction affichtab(t)
```