NSI - Première

TD : Complexité des algorithmes

qkzk

2021/06/06

Exercice 1 - Ordre de grandeur des complexité

1. Tracer sur la calculatrice les fonctions suivantes :

$$f(x) = x^2,\ g(x) = x,\ h(x) = \ln(x),\ k(x) = x\ln(x)$$

2. Trier ces fonctions par *vitesse de croissance*. De celle qui explose le moins vite à celle qui explose le plus vite.

Le logarithme est une fonction très utilisé en science et qui dispose de beaucoup de propriétés.

En informatique on utilise notamment le fait qu'il permet de mesurer le nombre de bits nécessaires pour écrire un nombre.

2. Combien de bits sont nécessaires pour écrire $N = 123\ 567$ en mémoire ?
3. Calculer $\dfrac{\ln x}{\ln 2}$ à l'aide de la calculatrice.

Cette fonction est appelé *logarithme en base 2* et permet d'encadrer le nombre de bits utilisé par un entier.

4. Proposer une formule pour encadrer la taille d'un entier en mémoire avec le logarithme en base 2.
5. Recommencer avec le logarithme en base 10.
6. Reprenons les fonctions de la question 1.

   On considère maintenant F, G, H, K des algorithmes. Ils prennent tous en entrée une seule donnée dont la taille est x et leur coût est est donné par la fonction correspondante.

   Pour une valeur de x choisie, leur temps d'exécution est 1 seconde.

   On double la valeur de x. Estimer les temps d'exécution de chaque algorithme.
7. Recommencer avec l'exponentielle : $l(x) = 2^x$.

Exercice 2 - Complexité de quelques fonctions

1. On considère la fonction `indice_mini` utilisée dans le tri par sélection. Quelle est sa complexité ?
2. Que fait la fonction suivante ?

```
def mystere(tableau):
  somme = 0
  nb_elements = 0
  for element in tableau:
    somme = somme + element
    nb_elements = nb_elements + 1

  return somme / nb_elements
```

3. Quelle est la complexité de la fonction `mystere` ?

4. Recommencer avec la fonction suivante :

```
def mystere(tableau):
    somme = 0
    for i in range(len(tableau)):
        for j in range(i):
            somme = somme + tableau[j]
    return somme
```

   1. Que fait-elle ? Commencez par un exemple : `mystere([1, 2, 3, 4])`
   2. Estimer sa complexité.

5. Recommencer avec la fonction suivante :

```
def mystere(tableau):
    x = 0
    y = 0
    for i in range(len(tableau)):
        if tableau[i] < tableau[x]:
            x = i
    for j in range(len(tableau)):
        if tableau[j] > tableau[j]:
            y = j
    return (x, y)
```

6. Existe-t-il une relation immédiate entre la complexité d'un algorithme et le nombre de boucle qu'il contient ?

   Que dire du cas où les boucles sont *imbriquées* (= l'une dans l'autre) ?

Exercice 3 - Tableau `insert` et `append`

Python propose deux méthodes sur les tableaux :

```
>>> t = [2, 4, 6]
>>> t.append(8)
>>> t
[2, 4, 6, 8]
>>> t.insert(0, 12)
>>> t
[12, 2, 4, 6, 8]
>>> t.insert(2, 20)
>>> t
[12, 2, 20, 4, 6, 8]
```

1. Donner la signature de ces méthodes :
   - à quel objet s'appliquent-elles ?
   - que prennent-elles en entrée ?
   - que font-elles ?
   - que renvoient-elles ?
2. En mémoire, un objet `list` est représenté par un tableau de cases mémoires contigues (côte à côte).

Lorsqu'on crée une liste, Python réserve un grand nombre de cases à l'avance et celles-ci sont vides. Notons ce nombre N. Sa valeur n'a pas d'importance et elle a changé avec les versions de Python.

À chaque fois qu'on ajoute un élément dans le tableau, un attribut, *sa longueur* est augmenté.

Ainsi, on sait toujours combien d'éléments il contient.

3. D'après la description de précédente, quelle est la complexité de la fonction `len` ?

4. Toujours d'après cette description, laquelle des deux méthodes `insert` et `append` est la plus efficace ?

   Donner leur complexité.

5. Accéder à un élément.

Toujours en utilisant la description plus haut, proposer un moyen d'accéder à un élément d'un tableau par son indice :

```
>>> t = [4, 5, 6]
>>> t[0]
4
>>> t[1]
5
```

Va-t-on plus vite lorsqu'on souhaite accéder au premier élément ? Au dernier ?

Remarques :

- Pour retirer un élément, la méthode `pop` est très rapide. ELle renvoie le dernier élément (celui dont l'indice est `len(tableau) - 1`) et modifie la longueur du tableau, indiquant simplement qu'il a perdu un élément.

  Pour retirer un autre élément, d'indice quelconque, c'est plus délicat, il faut déplacer tous les éléments qui étaient après lui et cela prend du temps.

- Bien-sûr, lorsqu'on a ajouté N éléments au tableau... celui-ci est plein. Python réserve alors d'autres emplacements mémoire pour pouvoir continuer à agrandir arbitrairement le tableau.

5. Il existe une autre méthode pour ajouter des éléments à un tableau, l'opération `+`.

   ```
   >>> t1 = [1, 2, 3]
   >>> t2 = [4, 5, 6]
   >>> t1 + t2
   >>> [1, 2, 3, 4, 5, 6]
   >>> t1
   [1, 2, 3]
   >>> t2
   [4, 5, 6]
   ```

   Que fait l'opération `+` sur les tableaux ? Quels sont ses opérandes et son résultat ? Modifie-t-elle les objets sur lesquels elle s'applique ?

6. Proposer une autre manière d'ajouter un élément à la fin d'un tableau, ou au début de celui-ci en utilisant `+`.

7. Quelle est sa complexité ?