Algorithmes de base : parcours d'un tableau

On voit dans ce cours 3 algorithmes élémentaires dans le détail présents dans le programme.
On donnera pour chacun son énoncé en langage naturel et sa traduction en Python.

1 Recherche d'un élément

Il s'agit de déterminer l'occurrence (la présence ou non) d'un élément de type quelconque (int, float, str, tuple,list ...) dans un tableau (en général une liste mais cela peut être aussi un tuple, un dictionnaire, une chaîne de caractère).

Langage naturel

monElement est l'élément à trouver
Pour chaque element du tableau faire
si element = monElement alors Retourner vrai
Fin Pour
Retourner faux

Langage Python

```
def recherche(tableau):
    for element in tableau:
            if element == monElement:
                    return True
    return False
```

Pour le coût, on utilise le nombre de comparaisons.
En notant n le nombre d'éléments du tableau, au mieux on effectue une seule comparaison, le premier élément est celui cherché, au pire on effectue n comparaisons, le dernier élément est celui cherché ou il n'est pas présent dans la liste.
Le coût est donc au pire **linéaire**.

Une question importante en informatique est *"l'algorithme se termine-t-il ?"*.
On parle de **terminaison**.
L'algorithme de recherche donné repose sur l'utilisation d'une boucle : celle-ci se réalise pour tout tableau (le nombre d'itération maximal est la taille de ce tableau) donc l'**algorithme de recherche se termine**.

2 Recherche d'un extremum

On veut déterminer le plus grand élément (maximum) et/ou le plus petit élément (minimum) dans un tableau.

Langage naturel

min = max = premier élément du tableau
Pour chaque element du tableau faire
si element > max alors max = element
si element < min alors min = element
Fin Pour
Retourner min,max

Langage Python

```
def extremum(tableau):
  min=max=tableau[0]
  for element in tableau:
    if element > max:
      max=element
    if element < min :
      min=element
  return min,max
```

Pour le coût, on utilise le nombre de comparaisons.
En notant n le nombre d'élément du tableau, on effectue à chaque passage dans la boucle 2 comparaisons.
Le coût est alors $2n$ donc **linéaire**.

L'algorithme de recherche donné repose sur l'utilisation d'une boucle : celle-ci se réalise pour tout tableau (le nombre d'itération est la taille de ce tableau) donc l'**algorithme de recherche d'extremum se termine**.

3 Calcul de la moyenne

L'algorithme ici est assez connu, il trouve sa place ici car il contient aussi le parcours d'un tableau non vide.
Bien entendu, les valeurs du tableau sont ici des entiers ou décimaux.

Langage naturel

total=0
Pour chaque element du tableau faire
total=total+element
Fin Pour
Retourner total/taille du tableau

Langage Python

```
def moyenne(tableau):
    total=0
    for element in tableau:
        total=total+element
    return total/len(tableau)
```

Pour le coût, on utilise le nombre d'opérations.
En notant n le nombre d'élément du tableau, on effectue n additions dans la boucle et une division.
Le coût est alors $n+1$ donc **linéaire** (on pourra mathématiquement préciser affine mais le terme informatique est linéaire).
On utilise une boucle : celle-ci se réalise pour tout tableau (contenant des nombres entiers ou décimaux) donc l'**algorithme de calcul de la moyenne se termine**.

4 Variant et invariant

On a vu dans les trois cas la notion de terminaison. Cependant, une autre question fondamentale est l'**algorithme donne-t-il à chaque fois le résultat attendu ?**.
Pour justifier qu'un algorithme donne bien le résultat désiré, on aura souvent à effectuer une démonstration comme en mathématiques, et on aura besoin de certains outils.
La particularité des algorithmes précédents est l'utilisation d'une boucle.
Le variant d'une boucle est un élément qui est modifié à chaque passage dans la boucle : dans nos trois algorithmes, le variant est l'élément du tableau de rang l'itération de la boucle (tableau[0],tableau[1]...).
L'invariant d'une boucle est une propriété qui est vrai avant et après chaque passage dans la boucle : cette notion est plus complexe car il s'agit de trouver avant tout cette propriété. Celle-ci est en lien avec l'objectif de l'algorithme.
Pour le premier, l'invariant est le test *element==monElement* où element est le terme donné du tableau à chaque itération, on a une propriété qui donne vrai ou faux.
Pour le deuxième, l'invariant est donné par le couple *(min=plus petit élément parmi tableau[0] jusque tableau[i-1], max=plus grand élément parmi tableau[0] jusque tableau[i-1]) où i est le rang de element.*
Pour le troisième, l'invariant est *total = somme des éléments du tableau du rang 0 jusque i-1 où i est le rang de element.*