301 Algo-Prog TD6
Complexité Algorithmique

1 Le coût d'un algorithme

Un algorithme a un certain "coût", c'est à dire un certain **nombre d'instructions de bases** (comme le nombre d'opérations et d'affectations), exécutées en fonction des nombres des données **n**. Ce coût peut être différent entre deux algorithmes qui produiront un même résultat comme par exemple la recherche linéaire et la recherche dichotomique. A l'aide de ce coût, il sera possible de comparer "l'efficacité" d'un algorithme, c'est à dire celui qui a besoin du plus petit nombre d'opération pour résoudre le problème posé.

Nous nous intéressons donc non seulement à l'écriture d'algorithmes qui produisent des résultats corrects, mais également au nombre d'instructions qui permettront de résoudre le problème.

Par exemple, pour la fonction ci-dessous, le coût est égal à 5.

```
def function(n):
    if n % 3 == 0:  # 2 operations
        resultat = n/3 + 2 # 2 operations and 1 affectation
    else:
        resultat = n*2 + 1 # 2 operations and 1 affectation
    return resultat
```

Le coût dépend généralement de la taille des données qu'il manipule. On note en général n cette taille et on cherche "formule(n)" qui représente le nombre d'opérations en fonction de n pour lequel "formule" dépend de l'algorithme.

Reprendre l'exercice 1.1.1 du TD4, où on utilise une boucle pour chercher un élément dans une liste. Une solution pourrait être :

```python
def recherche(liste,element):
    for i in range(len(liste)):
        if liste[i]==element:
            return i
    return -1
```

Au niveau de la boucle for, il y a 1 addition (sur i), 1 affectation (i) et 1 comparaison (test si $i < len(liste)$). Dans la boucle for, il y a un test if $liste[i] == element$. Au final, si n est la taille de liste, il y a $4n$ opérations élémentaires.

2 À vous

1. Calculer le coût de la fonction somme :

```python
def somme(n):
    somme = 0
    for i in range(n):
        somme += i
    return somme
```

2. Reprendre l'exercice 3.1 du TD2 pour l'affichage de la première figure, une solution pourrait être :

```python
def figure(n):
    for i in range(n+1):
        for j in range(i):
            print("*", end="")
        print()
```

   Quelle est le coût de cette fonction ?

3 Ordre de grandeur asymptomatique et complexité d'un algorithme

Quand n est relativement grand, c'est difficile de calculer précisément le coût. Nous allons plutôt estimer un ordre de grandeur $\mathcal{O}$.

On dit qu'une fonction f est :
— **bornée** ou **constante** si $f = \mathcal{O}(1)$
— **logarithmique** si $f = \mathcal{O}(\log n)$
— **linéaire** si $f = \mathcal{O}(n)$
— **quadratique** si $f = \mathcal{O}(n^2)$
— **exponentielle** s'il existe une constante $c > 0$ telle que $f = \mathcal{O}(c^n)$

Lorsque $f = \mathcal{O}(g)$ on dit que g **est un ordre de grandeur de** f**.**

Quel est l'ordre de grandeur de la fonction somme ? Et de la fonction figure ?

4 La recherche dichotomique

On veut représenter une zone de terrain de largeur x hauteur au moyen d'une liste de deux éléments :

```
largeur = ...
hauteur = ...
zone = [[0, 0], [largeur - 1, hauteur - 1]]
```

Le premier élément de zone décrit les coordonnées du coin supérieur gauche, le deuxième élément décrit les coordonnées du coin inférieur droit. Les coordonnées vont donc de la gauche vers la droite et du haut vers le bas.

Au lancement du programme, celui-ci calcule les coordonnées d'un point aléatoire de la zone : on suppose qu'un trésor sera caché dans cette case.

Le principe consiste à déplacer un robot (dont les coordonnées sont, elles aussi, choisies au hasard) pour qu'il se rende le plus vite possible sur la case du trésor.

L'algorithme du programme principal est donc :

```
largeur = lire("Entrez la largeur de la zone : ")
hauteur = lire("Entrez la hauteur de la zone : ")
zone = [[0, 0], [largeur - 1, hauteur - 1]]

x_tresor, y_tresor = coord_hasard(zone)
x_robot, y_robot = coord_hasard(zone)

nb_essais = 0
TQ (x_robot, y_robot) != (x_tresor, y_tresor) FAIRE
    direction = chercher(coord_robot, coord_tresor)
```

```
    nb_essais = nb_essais + 1
    diminue la zone de recherche en fonction de direction
    mise   jour des coordonnes du robot
    afficher("le robot se dplace en ...")
FTQ

Afficher("Trouve en ", nb_essais, " essais")
```

Vous devez donc écrire :

- la fonction coord_hasard qui renvoie un couple de coordonnées valide (par rapport à la zone qui lui est passée en paramètre)
- la fonction chercher qui renvoie la direction à suivre pour accéder au trésor.
  Son code consiste à regarder où se trouve la case du trésor par rapport à la position courante du robot (ces deux informations sont passées en paramètres à la fonction). Elle renvoie une chaîne "H" (haut), "B" (bas), "G" (gauche), "D" (droite), "HG" (haut gauche), "HD" (haut droit), "BG" (bas gauche), "BD" (bas droite) en fonction de la position du trésor.
  La réduction de la zone de recherche doit exploiter le principe de la dichotomie vu en cours.