Performances des algorithmes et Preuves en algorithmie

1 Structure itérative et suite numérique

1.1 Exemple : Récurrence simple

On pose $u_0 = 1$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = u_n + 2n - 1$.

1. Écrire une fonction Python d'argument n retournant une liste `U` contenant tous les termes u_i ;
2. Écrire une fonction Python d'argument n retournant l'unique terme u_n.

1.2 Synthèse

2 Preuves en algorithmie

Un algorithme ou un programme nécessite une vérification de ses performances. Les principales performances sont :

— **la terminaison** : l'algorithme s'arrête-t-il ?
— **la correction** : le résultat est-il juste, correct ?
— **la compléxité** : de quelle quantité de ressources (temps ou mémoire) l'algorithme a-t-il besoin ?

Les deux premières performances nécessitent une vérification formelle sous forme de **preuves**.

On rappelle qu'une **itération d'une boucle** est l'exécution du bloc d'instructions compris dans la boucle.

2.1 Terminaison

Il est nécessaire de vérifier qu'un algorithme s'arrête et ne tourne pas à l'infini. Evidemment, une boucle du type **for** exécute le bloc d'instructions un nombre N fini de fois . La boucle se termine donc.

La preuve qu'une boucle **while** se termine s'appuie sur une quantité appelée **variant de boucle** dépendant de l'état du programme et variant à chaque itération de boucle. Cette quantité est similaire à une suite numérique, dépendant de i le compteur d'itération.

2.1.1 notation

On note : v_0 la valeur initiale du variant de boucle et v_i sa valeur à la fin de la $i^{\text{ème}}$ itération. Il faut vérifier :

$$\begin{aligned} v_i &\in \mathbb{Z} \\ v_0 &> 0 \\ v_{i+1} &< v_i \end{aligned}$$

Alors la suite v_i est une suite strictement décroissante d'entiers , qui prendra une valeur nulle ou négative à partir du rang n.

La condition de boucle $v > 0$ devient fausse et l'algorithme s'arrête.

2.1.2 Application : division Euclidienne

On considère le programme suivant, où par hypothèse a et b sont deux entiers strictement positifs :

```python
a=int(input("dividende ?"))
b=int(input("diviseur ?"))
r, q = a, 0
while r >= b:
    r = r - b
    q = q + 1
print("quotient: ",q,"reste: ",r)
```

Justifier que cet algorithme se termine.

2.2 Correction

Pour justifier qu'un algorithme est correct, c'est-à-dire qu'il renvoie le résultat recherché, il faut mettre en place une preuve de sa correction. Cette preuve s'appuie sur un **invariant de boucle**, qui est une propriété $P(i)$ sur les variables à la fin de la $i^{\text{ème}}$ itération de boucle. Cette propriété $P(n)$ au rang n caractérise le résultat à trouver.

2.2.1 Méthodologie

1. Choisir et exprimer un invariant $P(i)$ judicieux ;
2. (*initialisation*) Démontrer que $P(0)$ est vrai avant d'entrer dans la boucle
3. (*héréidité*) Pour tout $i > 0$, $P(i) \Longrightarrow P(i+1)$ (Utiliser le corps de la boucle)

2.2.2 Application : division Euclidienne

Justifier que l'algorihtme de la division Euclidienne est correct.

3 Exercices

3.1 Programme mystère

On donne le programme suivant, où l'utilisateur renseigne deux entiers stritement positifs a et n.

```python
a=int(input("entier ?"))
n=int(input("puissance ?"))
N=n
R=1
while N>0 :
    R=R*a
    N=N-1
print(R)
```

1. Quel est le résultat de ce programme ?
2. En utilisant le variant $v = N$, justifier que le programme se termine.
3. En utilisant la propriété $\mathbf{P} : R \cdot a^N = a^n$, justifier que le programme est correct.
4. Compléxité en temps : combien de fois la boucle est-elle parcourue ?

3.2 Exponentiation rapide

On donne le programme suivant, où l'utilisateur renseigne deux entiers stritement positifs a et n.

```python
a=int(input("entier ?"))
n=int(input("puissance ?"))
A=a
N=n
R=1
while N>0 :
    if N%2==0 :
        A=A**2
        N=N/2
    else :
        R=R*A
        N=N-1
print(R)
```

1. En utilisant le variant $v = N$, justifier que le programme se termine.
2. En utilisant la propriété $\mathbf{P} : R \cdot A^N = a^n$, justifier que le programme est correct.
3. A quoi sert ce programme ?

3.3 Contamination

— Écrire un programme doit d'abord lire un entier, la population totale de la ville. Sachant qu'une personne était malade au jour 1 et que chaque malade contamine deux nouvelles personnes le jour suivant (et chacun des jours qui suivent), afficher à partir de quel jour toute la population de la ville sera malade.
— Prouver la terminaison de ce programme.
— Prouver la correction de ce programme.