DS 3
Informatique
Suite de Robinson associée à une liste
Extrait E3A PSI 2018

Simon Dauguet
simon.dauguet@gmail.com

Mercredi 11 Février 2026

Le devoir dure 1h.

La qualité de la rédaction et de la présentation seront prises en compte dans la notation. On prendra bien garde à la justesse et la précision des justifications.

Si un candidat repère ce qui lui semble être une erreur d'énoncé, il l'identifiera clairement sur la copie et explicitera les décisions qu'il sera amené à prendre.

La calculatrice n'est pas autorisée.

Le sujet comporte 3 pages.

1. Proposer en python une fonction `maxi` prenant en argument une liste d'entiers naturels `L` et renvoyant le maximum des entiers de cette liste. Puis calculer sa complexité au pire. *On n'utilisera pas de fonction spécifique de python déterminant ce maximum.*
2. Écrire une fonction `ind` prenant en argument une liste d'entiers naturels `L` et renvoyant la liste des indices $[i_1, i_2, ..., i_r]$ avec $i_1 < i_2 < ... < i_r$ telle que pour tout $k \in [\![1, r]\!]$, $L[i_k]$ soit non nul.

   Par exemple, si `L=[0,1,3,0,7]`, alors `ind(L)` renvoie `[1,2,4]`.
3. Écrire une fonction `nb_oc` prenant en argument une liste d'entiers naturels `L` et renvoyant la liste `T` de longueur `M = maxi(L)+1` où, pour tout $i \in [\![0; M]\!]$, `T[i]` est le nombre d'occurrences dans la liste `L` de l'entier i.

   Par exemple, si `L=[3,1,4,1,5]`, alors `T=[0,2,0,1,1,1]` (Remarque : on pourra utiliser la fonction `maxi`).
4. (a) Soit `L` une liste d'entiers naturels. Déterminer le nombre de fois, noté n, où la liste `L` est parcourue lors de l'exécution de `nb_oc(L)`.

   (b) On veut que n soit indépendant de `M`. Si c'est le cas, expliquer en quoi la condition est respectée. Si ce n'est pas le cas, modifier la fonction `nb_oc` afin de respecter cette condition.

   (c) Calculer la complexité de la fonction `nb_oc`.
5. Soit A une suite d'entiers. On définit alors la suite de Robinson $(L_n)_{n\in\mathbb{N}}$ associée à la liste A par récurrence comme suit :

   - $L_0 = A$

- si L_n est construite, alors :
  - on détermine $T_n = nb\_oc(L_n)$
  - on détermine $I_n = ind(T_n)$
  - si $I_n = [i_1, ..., i_r]$, alors $L_{n+1} = [T[i_r], i_r, ..., T[i_1], i_1]$

Par exemple, si $A = [4, 4, 1, 2]$ on a :

- $L_0 = [4, 4, 1, 2]$
- $L_1 = [2, 4, 1, 2, 1, 1]$ (il y a deux '4', un '2' et un '1' dans la liste L_0)
- $L_2 = [1, 4, 2, 2, 3, 1]$ (il y a un '4', deux '2' et trois '1' dans la liste L_1)

(a) On donne $A = [2, 0, 4, 1, 3, 3, 2, 3, 1, 1]$. Déterminer L_3 et L_{2026}.

(b) On donne $B = [2, 4, 1, 1, 1, 2]$. Si l'on suppose que $L_1 = B$, donner toutes les solutions possibles pour L_0

(c) On donne $C = [2, 4, 1, 0]$. Si l'on suppose que $L_1 = C$, donner toutes les solutions possibles pour L_0

On propose le code suivant :

```
1   def rob(A,n) :
2       i = 0
3       L = A[:]     # <=> A.copy()
4       while i < n :
5           T = nb_oc(L)
6           I = ind(T)
7           L = []
8           for e in I :
9               L = [T[e], e] + L
10          i += 1
11      return L
```

(d) Remplir le jeu de tests avec $A = [3, 1, 4, 1]$ et $n = 2$ en annexe.

(e) Prouver la terminaison de la fonction **rob**.

(f) Prouver la correction de la fonction **rob**.

Jeu de tests :

ligne	i	L	i<n	T	I	e	T[e]	Sortie