TSI2. Concours Blanc 2018

Épreuve d'informatique

Durée 3 heures. Les calculatrices sont interdites

Les quatre exercices sont indépendants et peuvent être traités dans un ordre quelconque.

Pour les exercices 03 et 04, on pourra utiliser les cinq fonctions qui opèrent sur les piles :
cree_pile( ), *empile*(x, p), *depile*(p), *sommet*(p), *est_vide*(p)
Rappelons leur conception.

```
>>> def cree_pile( ) :
        return([ ])

>>> def empile(x, p) :
        p.append(x)

>>> def depile(p) :
        assert(len(p) > 0)
        return(p.pop( ))

>>> def est_vide :
        return(len(p) == 0)

>>> def sommet(p) :
        n = len(p)
        assert(n > 0)
        return(p[n − 1])
```

Exercice 01

Formats de papier

Le format papier A_0 est un format rectangulaire de largeur $841\,mm$ et de longueur $1189\,mm$. Cela donne une surface de $1\,m^2$.
Pour tout entier p, le format A_{p+1} est un format ractangulaire dont la longueur est la largeur du format A_p et la largeur est la moitié de la longueur du format A_p.

1. Donner (à la main) la largeur et la longueur d'une feuille de format A_1 puis A_2 puis A_3.
2. Quels sont les surfaces des feuilles de format A_1 jusqu'à A_4 ?
3. Écrire une fonction récursive $format(p)$ qui renvoie le couple (largeur, longueur) (en mm) d'une feuille de format A_p.

T.S.V.P →

Exercice 02

Le baguenaudier

Le jeu du baguenodier est constitué d'une réglette comportant n cases, numérotées de 1 à n. Chaque case peut être soit vide, soit contenir un pion. Initialement, toutes les cases sont remplies. Le but du jeu est de vider toutes les cases en suivant les règles suivantes :

(i) ***on ne peut modifier l'état que d'une case à la fois (la remplir si elle est vide ; la vider sinon)***
(ii) ***on peut toujours modifier l'état de la case numéro*** 1 ***(celle la plus à gauche)***
(iii) ***pour tout entier*** $i \in [\![2, n]\!]$, ***on peut modifier l'état de la case numéro*** i ***si et seulement si :***
- ***la case numéro*** $i-1$ ***est occupée***
- ***les cases*** 1, 2, ..., $i-2$ ***sont libres.***

Voici par exemple une solution lorsque $n = 3$:

1. Exhiber une solution dans le cas $n = 2$ puis $n = 4$. On dessinera toutes les étapes.
2. Écrire une fonction Python $vider\_case(n)$ qui affiche juste le message « vider la case numéro n » ainsi qu'une fonction Python$remplir_case(n)$ de fonctionnement analogue.
   Attention, il faut dissocier la chaîne de caractères de la valeur n.
3. À partir de maintenant, on décide d'appliquer les algorithmes mutuellements récursifs suivants, pour soit vider complétement une règlette de n cases initialement remplies, soit pour remplir complétement une règlette de n cases initialement vides.

   On suppose $n \geqslant 2$.
   • ***Algorithme « Vider ». Pour vider la règlette de*** n ***cases, on peut***
   étape 1 : commencer par vider les $n-2$ ***premières cases (sans toucher aux deux dernières) ;***
   étape 2 : la dernière case est alors précédée d'une case pleine, les autres étant toutes vides, on la vide à son tour ;
   étape 3 : on remplit maintenant les $n-2$ ***premières cases (sans toucher aux deux dernières) : les*** $n-1$ ***premières cases sont alors pleines et la dernière est vide ;***
   étape 4 : on termine en vidant ces $n-1$ ***cases.***

   • ***Algorithme « Remplir ». Pour remplir une règlette de*** n ***cases, on peut :***
   étape 1 : remplir les $n-1$ ***premières cases ;***
   étape 2 : on vide ensuite les $n-2$ ***premières cases ;***
   étape 3 : on remplit la dernière case ;
   étape 4 : on remplit les $n-2$ ***premières.***

   (a) Adapter les deux algorithmes au cas $n = 1$.

   (b) Appliquer à la main l'algorithme « Remplir » pour les cas $n = 2$, $n = 3$ et $n = 4$. On fera tous les dessins. On partira d'une règlette vide pour obtenir une règlette remplie.

   (c) Appliquer à la main l'algorithme « Vider » pour le cas $n = 5$. On dessinera encore une fois toutes les étapes. On partira d'une règlette remplie pour obtenir une règlette vide.

(d) Écrire deux fonctions Python mutuellement récursives *vider* et *remplir*, inspirées des algorithmes précédents, telles que :
vider(n) affiche la succession des coups à jouer pour vider la règlette (initialement remplie) à n cases.
remplir(n) affiche la succession des coups à jouer pour remplir la règlette (initialement vide) à n cases (donc *vider*(n) donne la solution du jeu).
Remarque : pour le cas $n = 0$, il n'y a rien à afficher (aucune case à vider ou remplir) ; on pourra alors renvoyer la valeur *None* (qui ne provoquera aucun affichage).

4. On note :
• v_n le nombre de coups à jouer pour vider la règlette (initialement remplie) à n cases ;
• r_n le nombre de coups à jouer pour remplir la règlette (initialement vide) à n cases.
On convient que $v_0 = r_0 = 0$.

   (a) Avec les exemples prcédédents, donner les valeurs de v_k pour $k \in [\![1, 5]\!]$ et de r_k pour $k \in [\![1, 4]\!]$. Que remarque t-on ?

   (b) Écrire les relations de récurrence vérifiées par les deux suites (v_n) et (r_n).

   (c) Démontrer que, pour tout entier n, on a : $r_n = v_n$.

   (d) Déterminer l'unique constante l telle que la suite constante égale à l satisfasse les relations de récurrence trouvées à la question **4)b)**.

   (e) On note désormais $x_n = v_n - l$, où l a été trouvé à la question **4)d)**.
   Déterminer la relation de récurrence vérifiée par cette nouvelle suite.

   (f) En déduire l'expression de v_n en fonction de n. Retrouver v_n pour $n \in [\![1, 5]\!]$.

Exercice 03

Affiches visibles

On colle des affiches sur une palissade, toutes à partir du bord gauche de celle-ci. On suppose que les affiches occupent toute la hauteur de la palissade, mais elles n'ont pas toutes la même largeur. Lorsqu'on colle une affiche par dessus les précédentes, la nouvelle les recouvre partiellement ou totalement. On cherche à déterminer combien d'affiches sont encore (partiellement) visibles après le collage d'une liste d'affiches dont on donne les largeurs. Par exemple, si la liste des largeurs des affiches est (dans l'ordre) $[4, 5, 3, 1, 2]$, la première affiche (de largeur 4) est collée puis entièrement recouverte par l'affiche suivante, de largeur 5, elle-même partiellement recouverte par la suivante (de largeur 3). À la fin, restent visibles (partiellement) les affiches de largeurs 5, 3 et 2 (donc trois affiches).

1. Si la liste des largeurs des affiches est dans l'ordre d'affichage $[3, 7, 6, 8, 1, 3, 2, 5]$, combien d'affiches restent visibles à la fin ?
2. Écrire une fonction Python *decompte*(p) qui renvoie le nombre d'éléments présents dans une pile p. La fonction pourra vider la pile (on n'impose pas que, après l'appel à *decompte*(p) , la pile soit dans le même état qu'avant l'appel de la fonction).
   On pourra utiliser les fonctions *est_vide* et *depile*.
3. Écrire une fonction *nombre_visibles*(l) dont le paramètre l est la liste des largeurs des affiches dans l'ordre d'affichage et qui renvoie le nombre d'affiches encore partiellement visibles à l'issue du collage. On commencera par créer une pile dans la procédure.
   On pourra utiliser les cinq fonctions qui gérent les piles, rappelées en début de sujet.

Exercice 04

Tri avec une pile

Pour tout entier $n \geqslant 1$, on appelle ***séquence de longueur*** n une liste formée des n entiers de l'intervalle $[\![1, n]\!]$ (il y a donc $n!$ séquences de longueur n). On dit qu'une telle séquence est ***triable par une pile*** si et seulement s'il est possible, à partir d'une liste l et d'une pile p, initialement vides, de ranger tous les éléments de la séquence par ordre croissant dans la liste au moyen des seules opérations suivantes :

• ***s'il reste des éléments dans la séquence, prendre le premier et l'empiler sur la pile ;***

• ***si la pile est non vide, dépiler le sommet et le ranger dans la liste en cours de formation (à la fin de la liste).***

Par exemple, la séquence $(1, 3, 2)$ peut être triée par la succession d'opérations suivantes :

• empiler l'élément 1 ;
• dépiler (on récupère l'élément 1, que l'on place dans la liste l) ;
• empiler l'élément 3 ;
• empiler l'élément 2 ;
• dépiler (on ajoute 2 à la liste l) ;
• dépiler (on ajoute 3 à la liste).

On représente cette suite d'opérations sous la forme $EDEEDD$ (E pour empiler et D pour dépiler).

On obtient $l = [1, 2, 3]$. C'est bien dans l'ordre croissant.

1. Parmi les séquences suivantes, lesquelles peuvent être triée par une pile ?

   $$(2, 4, 1, 3) \quad (3, 1, 2, 5, 4).$$

   on demande une justification.

2. Montrer que lors d'un tri réussi, les éléments de la pile sont, à chaque instant, rangés par ordre croissant (en partant du sommet de la pile).

3. Justifier que si l'une séquence est triable par une pile, il existe une unique manière de la trier.

4. Écrire en Python une fonction $tri\_pile(s)$ qui renvoie un booléen indiquant si une séquence, représentée par la liste s, est triable par une pile.