DS - Algorithmique - Master 1 Bio-informatique

15 décembre 2016 - Durée : 1h30 minutes

Les documents, les ordinateurs et les téléphones ne sont pas autorisés.

Exercice 1

Proposez un algorithme qui prends en paramètres une liste Python d'entiers l et un entier e et qui renvoie vrai si l'élément e est dans l et faux sinon.

Déterminer la complexité dans le pire cas de votre algorithme.

Solution

```
1 def recherche(l, e):
2         for i in range(len(l)):
3                 if e == l[i]:
4                         return True
5         return False
```

Soit $n = len(n)$, alors,
Complexité de la lisge 4 : (1)
Complexité des la ligne 3 à 4 : (1)
Complexité de la ligne 2 à 4 : $n \times (1) = (n)$

La complexité totales est donc : $(1) + (n) = (n)$.

Exercice 2

Donnez la définition d'une file. Illustrez votre définition par un exemple.

Solution

Les files F sont des structures de données contenant des éléments qui peuvent apparaître en plusieurs exemplaires et possédant 2 opérations :

- `enfiler(F,e)` qui ajoute l'élément e dans F ;
- `defiler(F)` qui enlève un élément de la file F ;

et qui vérifient la propriété suivante : si un élément e est ajouté avant l'élément f alors l'élément e sera enlevé avant f.

Exercice 3

Dans cet exercice, vous devez utilisez uniquement les piles données à l'aide des fonctions primitives suivantes :

```
def creer_pile( ):
    # renvoie une nouvelle pile vide
```

```
def empiler( p, e ):
    # empile dans la pile p l'élément e.

def depiler( p ):
    # dépile l'élément situé en heut de la pile p et le renvoie.

def taille( p ):
    # renvoie le nombre d'éléments contenus dans la pile p.
```

Dans cet exercice, il n'est pas autorisé d'utiliser des listes, tuples et dictionnaires du langage python.

1. Proposez un algorithme *copie_et_renverse*(p) qui prends en paramètre une pile p et qui renvoie une copie de p dont les éléments sont ordonnées dans l'ordre inverse. La pile p doit, à la fin du processus, rester inchangée.
2. On rappelle qu'un mot w est un palindrome si son renversé est égal à w. Par exemple, *aabaa* est un palindrome, mais pas *aabab*.
   Proposez un algorithme qui n'utilise que les primitives de piles et qui détermine si un mot passé en paramètre est un palindrome.

Solution

1.
```
1  def copier_et_renverser(p, e):
2      resultat = creer_pile()
3      tmp = creer_pile()
4      for i in range( taille(p) ):
5          e = depiler(p)
6          empiler( resultat, e )
7          empiler( tmp, e )
8      for i in range( taille(tmp) ):
9          empiler( p, depiler(tmp) )
10     return resultat
```

2.
```
1  def est_palindrome(w):
2      renverse_w = copier_et_renverser( w )
3      copie_w = copier_et_renverser( p1 )
4      for i in range( taille(w) ):
5          if depiler(copie_w) != depiler(renverse_w):
6              return False
7      return True
```

Exercice 4

Le but de cet exercice est d'écrire un programme récursif qui renvoie tous les sous-ensembles d'un ensemble $\{1, 2, \ldots, n\}$. On appelle ces sous-ensembles les *parties* de n.

Par exemple, les parties de 3 sont :

$$\{\},\ \{1\},\ \{2\},\ \{3\},\ \{1,2\},\ \{1,3\},\ \{2,3\} \text{ et } \{1,2,3\}.$$

En fait, on peut remarquer que les parties de n sont la réunion

— des parties de $n-1$ et

— des sous-ensembles obtenus à partir des parties de $n-1$ en leurs ajoutant l'entier n.

Par exemple :

$$parties(0) = \{\{\}\ \}$$

$$parties(1) = \{\ \underbrace{\{\}}_{parties(0)}\ ,\ \underbrace{\{1\}}_{partie(0)\text{ auxquels on ajoute 1 à chaque ensemble}}\ \}.$$

$$parties(2) = \{\ \underbrace{\{\},\{1\}}_{parties(1)}\ ,\ \underbrace{\{2\},\{1,2\},\{2,2\}}_{partie(1)\text{ auxquels on ajoute 2 à chaque ensemble}}\ \}.$$

$$parties(3) = \{\underbrace{\{\},\{1\},\{2\},\{1,2\}}_{parties(2)},\ \underbrace{\{3\},\{1,3\},\{2,3\},\{1,2,3\}}_{partie(2)\text{ auxquels on ajoute 3 à chaque ensemble}}\ \}.$$

Nous allons utiliser cette dernière propriété pour implémenter les parties de n.

1. Commencez par écrire un programme $ajouter\_entier(l, e)$ qui prends en paramètres une liste python l qui contient des listes d'entiers et un entier e. Ce programme fait une copie l' de l, ajoute l'entier e à la fin de toutes les listes python présentent dans l' et retourne l.

   Par exemple, `ajouter_entier( [ [], [2,4]], 3 )` renvoie la liste $[[3],[2,4,3]]$.

2. Proposez une fonction récursive $parties(n)$ qui prends en paramètre un entier n et qui renvoie une liste de listes qui code toutes les parties de n.

   Par exemple, la fonction $parties(4)$ pourra renvoyer la liste suivante :

   $$[[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]$$

   Aucune contrainte d'ordre est requise dans la fonction $parties(n)$, les listes ne doivent cependant pas contenir de doublons.

   Par exemple, $partie(3)$ peut renvoyer la liste suivante :

   $$[[2],[1,2],[3],[1,3],[],[1],[2,3],[1,2,3]]$$

   Par contre il ne peut pas renvoyer la liste suivante :

   $$[[],[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]$$

   car la liste [] est en deux exemplaires. Il ne pourra pas renvoyer non plus

   $$[[],[1,1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]$$

   car dans $[1,1]$ l'entier 1 est présent deux fois.

Solution

1.

```
1  def copier_liste(l):
2      resultat = []
3      for i in range(len(l)):
4          resultat.append( l[i] )
5      return resultat
6
7  def ajouter_entier(l, e):
8      resultat = []
9      for li in l:
10         copie_li = copier_liste(li)
11         copie_li.append( e )
12         resultat.append( copie_li )
13     return resultat
```

```
1  def parties(n):
2      if n == 0:
3          return [[]]
4      retrun parties(n) + ajouter_entier( parties(n-1), n )
```