UNIVERSITÉ LIBRE DE BRUXELLES
École polytechnique de Bruxelles

Août 2015

INFO-H-100 – Informatique – Prof. Th. Massart
1ère année du grade de Bachelier en Sciences de l'Ingénieur
Interrogation de Août

5 questions, 4 pages.

Remarques préliminaires

- On vous demande de répondre aux questions sur les feuilles de l'énoncé.
- Vous pouvez continuer la réponse sur le verso de la page.
- Des feuilles de brouillon vous sont fournies en fin d'énoncé.
- N'oubliez pas d'inscrire votre nom, prénom et numéro de matricule sur chaque feuille.
- Vous disposez de trois heures et vous ne pouvez pas utiliser de notes.
- Si du code vous est demandé,
  - la réponse à la question doit comprendre le code *Python* structuré et conforme aux règles de bonne pratique et conventions,
  - sauf mention contraire, vous ne pouvez utiliser aucune fonction de librairies (pas d'`import`)
  - veillez à découper votre réponse en fonctions de manière pertinente
  - veillez à utiliser des structures de données appropriées.
- Veillez à écrire lisiblement et n'hésitez pas à commenter votre code si ça peut le rendre plus clair.

Question 1 - Théorie (5 points)

Pour chacun des codes suivants, et en supposant que `s` est une chaine de n caractères, que `mat` est une matrice carrée de dimansion $n \times n$ et que `dico` est un dictionnaire dont les clés et valeurs sont respectivement des chaînes de caractères de taille maximum 10 et des ensembles de clés existentes.

- expliquez ce qu'il fait et ce qu'il imprime
- donnez l'état du programme lors de chaque `print` grâce à des diagrammes d'état (comme fait au cours).
- donnez la complexité moyenne et maximale ($\mathcal{O}()$) de la **fonction** `foo_`i, $i = 1..3$) en temps d'exécution. Précisez bien les paramètres utilisés pour exprimer la complexité et justifiez bien vos réponses (le résultat seul ne suffit pas pour obtenir des points).

Solution

1.
```python
def foo_0(s):                                   # moyenne   maximale
    if len(s) == 1 or len(s) ==0:               #   O(1)      O(1)
        res  = True                             #   O(1)      O(1)
    else:
        res = s[0] == s[-1] and foo_0(s[1:-1])  #   O(k) sans la récursivité idem
    print(s)                                    #   O(k)      O(k)
    return res                                  #   O(1)      O(1)

s = "hannah"
print(foo_0(s))
```

- `foo_0` teste si une séquence reçue (string ou autre) est un palindrôme.
- Pour l'exemple, le code imprime :

```
  (string vide)
nn
anna
hannah
True
```

- Pour visualiser les diagrammes d'état, vous êtes invité à introduire le code dans pythontutor.org et à l'exécuter : chaque appel récursif à la fonction `foo_0` créée une nouvelle Frame (espace de nom) avec deux variables locales `s` et `res`.

— Complexités moyennes et maximales : voir code plus haut pour chaque instance où k est la taille du string reçu en paramètre. Au total $0(k)$ avec $k = n, n-2, n-4, ..., 2, 0$ soit $O(n)$ appels et de l'ordre de n+(n-2)+(n-4)+... soit une complexité moyenne et maximale en $O(n^2)$.

2.
```
def foo_1(mat):                                  #  moyenne        maximale
   n = len(mat)                                  #    O(1)           O(1)
   for i in range(n-1):                          #    O(n^3)         O(n^3)
      i_sel = i                                  #    O(1)           O(1)
      for j in range(i+1,n):                     #    O(n^2)         O(n^2)
         if  min(mat[j]) < min(mat[i_sel]):      #    O(n)           O(n)
            i_sel = j                            #    O(1)           O(1)
      mat[i_sel],mat[i] =  mat[i],mat[i_sel]     #    O(1)           O(1)
      print(mat)                                 #    O(n^2)         O(n^2)

m = [[1,2,6],[3,4,7],[5,0,8]]
foo_1(m)
print(m)
```

— `foo_1` trie une matrice (technique du tri par sélection) en réorganisant les lignes par ordre croissant en fonction de la valeur de l'élément minimal de chaque ligne.
— Pour l'exemple, le code imprime :
```
[[5, 0, 8], [3, 4, 7], [1, 2, 6]]
[[5, 0, 8], [1, 2, 6], [3, 4, 7]]
[[5, 0, 8], [1, 2, 6], [3, 4, 7]]  (après le retour)
```
— Pour visualiser les diagrammes d'état, vous êtes invité à introduire le code dans pythontutor.org et à l'exécuter.
— Complexités moyennes et maximales : voir commentaire dans le code $O(n^3)$ pour les deux complexités. Voir complexité du tri par sélection mais en plus la fonction `min` prend un temps linéaire dans la taille de la ligne $= n$. D'où les résultats.

3.
```
def foo_3(dico):                  #                  moyenne     maximale
   res = set({})                  #                    O(1)        O(1)
   for elem in dico:              # (nbre d'it O(n)) => O(n^2)     O(n^3)
      c = dico[elem] - {elem}     #                    O(1)        O(n)
      for a in c:                 # (nbre d'it O(n)) => O(n)       O(n^2)
         if elem in dico[a]:      #                    O(1)        O(n)
            res.add(elem)         #                    O(1)        O(n)
   return res                     #                    O(1)        O(1)

dico = { "Jean"       : {"Jean", "Catherine"} ,
         "Germaine"   : {"Catherine"},
         "Catherine"  : {"Jean", "Germaine"} ,
         "Luc"        : set({}),
         "Michel"     : {"Luc"}}
print(foo_3(dico))
```

— `foo_3` renvoie toutes les personnes qui "connaissent" quelqu'un (dans l'ensemble correspondant à sa clé, et dont au moins une de ces connaissances le connait.
— Pour l'exemple, le code imprime :
`{'Jean', 'Catherine', 'Germaine'}`
— Pour visualiser les diagrammes d'état, vous êtes invité à introduire le code dans pythontutor.org et à l'exécuter.
— Complexités moyennes et maximales : Hypothèse pour la complexité moyenne : chaque entrée à $n/2$ valeurs ; voir commentaires dans le code $O(n^2)$ en moyenne et $O(n^3)$ au maximum. La différence est dûe aux accès au dictionnaire et aux manipulations (rajoute et enlève un élément) des ensembles, qui en moyenne est en $O(1)$ mais au maximum est en $O(n)$ où n est le nombre d'entrée dans le dictionnaire ou l'ensemble.

Question 2 - Fonctionnement de Python 2 (2 points)

Expliquer le mode de fonctionnement d'un compilateur et d'un interpréteur. Python de base est-il interprété ou compilé ? Expliquez quels en sont les avantages et les inconvénients pour ce langage.

Solution

Voir cours.

Question 3 - Manipulation de fichiers (5 points)

Écrivez une fonction `selection(f_in, criteres, f_out)` qui ouvre et lit le contenu du fichier de données dont le nom est donné par le paramètre `f_in`, fichier contenant une liste de mots, et qui crée un nouveau fichier de mots en résultat dont le nom est donné par le paramètre `f_out` contenant tous les mots du fichier de données qui satisfont un certain nombres de critères donnés par le paramètre `criteres`.

`criteres` est un dictionnaire dont les clés sont des lettres alphabétiques (on suppose en minuscules sans accents) et dont les valeurs sont des nombres entiers positifs ou nuls. Un mot du fichier respecte les critères s'il a, pour chaque lettre donnée en clé du dictionnaire, le nombre exact de cette lettre donné par la valeur correspondante.

Par exemple avec

```
criteres = { 'a' : 0,
             'e' : 1,
             'i' : 2}
```

Tous les mots du fichier de données contenant aucune fois la lettre 'a', exactement une fois la lettre 'e' et deux fois la lettre 'i', doivent être écrits dans le fichier de résultat.

Le fichier de données a exactement un mot par ligne et il est supposé bien formaté, chaque mot contenant uniquement des lettres alphabétiques minuscules sans accent (il n'y a pas de lignes vides ou autres tests à effectuer sur le contenu de ce fichier). Le fichier de résultat aura également un mot par ligne et devra être fermé par la fonction en fin de traitement.

Exemple :

Fichier `exemple.txt` :

```
cueillir
prairie
difficile
examen
fossilise
```

Fichier résultat `resultat.txt` :

```
cueillir
fossilise
```

L'appel à la fonction `selection("exemple.txt", criteres, "resultat.txt")` produit le fichier nommé "`resultat.txt`", contenant tous les mots correspondants à `criteres` (voir ci-dessus) trouvés dans le fichier "`exemple.txt`". Notez que le mot `prairie` n'est pas repris, car, malgré qu'il contienne un `e` et deux `i`, il contient également un `a`. De même `difficile` contient un `e`, aucun `a`, mais trois (et non deux) `i`.

Solution

```
def criteria_match(string, criteres):
  occurences = {}
  for car in string:
    occurences[car] = occurences.get(car,0) + 1
  keys = list(criteres.keys())
  i = 0
  res = True
  while i < len(keys) and res:
    res = occurences.get(keys[i],0) == criteres[keys[i]]
    i += 1
  return res

def selection(f_in, criteres, f_out):
  # we keep the "\n" at the end of the line,
  # since we will write it back to the result file
  words = open(f_in).readlines()
  f = open(f_out,"w")
  for word in words:
    if criteria_match(word, criteres):
      f.write(word)
  f.close()

criteres = {'a': 0, 'e': 1, 'i': 2}
selection("exemple.txt", criteres, "resultat.txt")
```

Question 4 - Recherche dichotomique (4 points)

Écrivez une fonction `worst_passed(grade_list)` qui renvoie le nom de l'étudiant ayant réussi avec la moins bonne note, c'est-à-dire celui dans la liste ayant la plus faible cote supérieure ou égale à 10/20. La liste `grade_list` contient des tuples de la forme `(nom, cote)`, triée de manière descendante sur la cote. On vous demande d'écrire la fonction en utilisant une recherche dichotomique.

Exemple :

```
>>> grade_list = [ ("Lisa", 19), ("Krusty", 15), ("Ned", 14), ("Apu",11), ("Marge", 10), ("Bart", 8), ("Homer", 1) ]
>>> worst_passed(grade_list)
'Marge'
```

Solution

```
def worst_passed(grade_list):
  bi,bs = 0, len(grade_list)
  m = (bi+bs)//2
  while bi < bs and grade_list[m][1] != 10:
    if grade_list[m][1] < 10:
      bs = m-1
    else:
      bi = m+1
    m = (bi+bs)//2
  return grade_list[m][0]
```

Question 5 - Récursivité (4 points)

Soit une liste de nombres pouvant contenir des sous-listes. Écrivez une fonction `list_depth(ls)` qui renvoie la "profondeur" de la liste `ls`. On convient qu'une liste `ls` ne contenant pas de sous-liste a une profondeur nulle : la fonction renvoie la valeur zéro dans ce cas.

Exemples :

```
>>> list_depth([1, 2, 3])
0
>>> list_depth([1, [2, 3], 4])
1
>>> list_depth([1, [[[2, 3], 4], 5, []], 6])
3
```

Solution

```
def list_depth(ls):
  depth = 0
  for x in ls:
    if type(x) == list:
      d = list_depth(x) + 1
      if d > depth:
        depth = d
  return depth
```