UNIVERSITE LIBRE DE BRUXELLES
Ecole polytechnique de Bruxelles

9 juin 2012

INFO-H-100 – Informatique – Programmation – Prof. Th. Massart
1ère année du grade de Bachelier en Sciences de l'Ingénieur
Correction de l'examen de juin

Question 1 - Complexité (3 points)

```
def equality_dic(a,b):                                  O(n^2)
    res = False                                         O(1)
    if len(a)==len(b):                                  O(n^2)
       dic_a = {}                                       O(1)
       dic_b = {}                                       O(1)
       for i in a:                                      O(n^2)
           dic_a[i]=dic_a.get(i,0)+1                    O(n)
       for i in b:                                      O(n^2)
           dic_b[i]=dic_b.get(i,0)+1                    O(n)
       res = True                                       O(1)
       for j in dic_a:                                  O(n^2)
           res = res and (dic_a[j] == dic_b.get(j,0))   O(n)
       for j in dic_b:                                  O(n^2)
           res = res and (dic_b[j] == dic_a.get(j,0))   O(n)
    return res                                          O(1)
```

Explications :
La complexité au pire des cas est celle où `len(a) = len(b) = n`. L'algorithme est un $\mathcal{O}(n^2)$ car

- une séquence d'instructions simples en $\mathcal{O}(1)$ et d'une `if` qui lui même contient une séquence avec des instructions en $\mathcal{O}(1)$ et des `for` qui sont tous au pire en $\mathcal{O}(n^2)$.
- Chaque `for` tourne de l'ordre de `n` fois et chaque itération fait un ou 2 accès à un élément du dictionnaire en lecture ou écriture (au pire en $\mathcal{O}(n)$).
- Les règles du `for` (multiplie les complexités du corps et du nombre d'itérations) du `if` et des séquences (max des complexités) nous donnent le résultat.

```
def equality_list(a,b):                 O(n^2)
    res = False                         O(1)
    if len(a)==len(b):                  O(n^2)
       res = True                       O(1)
       for i in a:                      O(n^2)
           if i in b:                   O(n)
               a.remove(i)              O(n)
               b.remove(i)              O(n)
           else:
               res = False              O(1)
    return res
```

Explications : La complexité au pire des cas est celle où `len(a) = len(b) = n`
L'algorithme est un $\mathcal{O}(n^2)$ car

- le test `if i in b` ainsi que les `remove` sont en $\mathcal{O}(n)$ (recherche ou enlève dans une liste à n éléments
- le `if i in a` complet est donc formé d'une séquence qui au pire des cas est 3 fois en $\mathcal{O}(n)$ donc en $\mathcal{O}(n)$
- le `for` boucle de l'ordre de $\mathcal{O}(n)$ fois donc au total on obtient une comlexité pour l'instruction for en $\mathcal{O}(n^2)$
- le `if len(a)==len(b)` est donc aussi en $\mathcal{O}(n^2)$ ainsi que l'algorithme complet.

Question 2 - Compilation / interprétation (2 points)

Voir notes sur l'Université Virtuelle.
Il fallait parler

- des 6 phases (3 d'analyse (lexicale, syntaxique et sémantique) et 3 de synthèse (génération du code intermédiaire, optimisation et génération du code final)
- expliquer que l'interpréteur décode chaque instruction python une par une et exécute l'effet voulu. Contrairement à un langage compilé où on a 2 étapes bien séparées (compilation et exécution).
- que la définition du langage implique une gestion d'un stack (pile) et d'un heap (tas) run time que le langage soit compilé ou interprété.

Question 3 - Hachage (4 points)

```
def isAlpha(c):
    """True ssi c est une lettre alphabetique"""
    return ('A' <= c <= 'Z') or ('a' <= c <= 'z')

def hachage(texte):
    texte = filter(isAlpha, texte)          # Ne garde que les alphabetiques
    ls    = map(ord, texte)                 # Liste des codes ASCII de chaque lettre
    return reduce(lambda a,b: a+b, ls, 0)   # ou: return sum(ls)
```

Question 4 - Flatten (5 points)

```
def flatten(l):
    r = []
    for el in l:
        if type(el) == list:
            r.extend(flatten(el))
        else:
            r.append(el)
    return r
```

Question 5 - Cryptage de fichier (6 points)

```
def openTrans(transfile):
        f = open(transfile)
        data = f.read().split()
        trans = {}
        for i in range(0,len(data),2):
                trans[data[i]] = data[i+1]
        f.close()

        return trans

def removeExtension(filename):
        i = len(filename)-1
        while i >= 0 and filename[i] != '.':
                i -= 1

        if i == -1:
                result = filename
        else:
                result = filename[:i]

        return result

def openSource(sourcefile):
        f = open(sourcefile)
        data = f.read()
        f.close()
        return data

def writeCrypt(crypt, cryptFile):
        f = open(cryptFile,'w')
        f.write(crypt)
        f.close()

def createCrypt(trans, source):
        source = list(source)    # travailler avec une liste et pas un string permet d'etre en temps lineaire
        for i in range(len(source)):
                source[i] = trans.get(source[i], source[i])

        return ''.join(source)

def translate(transfile, sourcefile):
        source = openSource(sourcefile)
        trans = openTrans(transfile)
        crypt = createCrypt(trans, source)
        # crypt = ''.join(map(lambda x: trans.get(x,x), source))        # ou avec un map
        cryptFile = removeExtension(sourcefile) + ".crypt"
        # cryptFile = '.'.join(sourcefile.split('.')[:-1] + ["crypt"])  # ou en travaillant sur le string
        writeCrypt(crypt, cryptFile)

translate("trans.txt", "bonjour.txt")
```