Devoir d'informatique 3

D.Malka – MPSI 2017-2018 – Lycée Saint-Exupéry

11.01.2018

Durée de l'épreuve : 2h00

L'usage de la calculatrice est autorisé.

L'énoncé de ce devoir comporte 4 pages.

- *Si, au cours de l'épreuve, vous repérez ce qui vous semble être une erreur d'énoncé, signalez le sur votre copie et poursuivez votre composition en expliquant les raisons des initiatives que vous êtes amené à prendre.*
- *Il ne faudra pas hésiter à formuler des commentaires. Le barème tiendra compte de ces initiatives ainsi que des qualités de rédaction de la copie.*
- *La numérotation des exercices doit être respectée. Les résultats doivent être systématiquement encadrés. Les pages doivent être numérotées de la façon suivante : n°page courante/nombre total de page.*

Données pour tous les exercices.

- La méthode `append()` ajoute l'argument en queue de liste. Appel : `l.append(elt)`.
- La fonction `len()` renvoie un entier égal à la longueur de la liste ou de la chaîne de caractères passée en argument. Appel : `len(L)`.
- La fonction `float()` convertit l'argument en flottant. Appel `float(n)`.
- La fonction `int()` convertit l'argument en un entier. Si l'argument est un flottant, la fonction `int()` renvoie la partie entière de ce nombre. Appel `int(x)`.
- La fonction `range(a,b,p)` génère la liste suivante : $[a, a+p, a+2p, \dots, a+jp, \dots, b-p]$. L'appel `range(a,b)` est équivalent à `range(a,b,1)` et l'appel `range(b)` est équivalent à `range(0,b,1)`.
- La fonction `linspace(a,b,N)` génère la liste suivante : $[a, a+p, a+2p, \dots, a+jp, \dots, a+(N-1)p]$ avec $p = \frac{b-a}{N}$.
- Slicing : l'expression `l[i:j]` renvoie la sous-liste comprenant les éléments d'indice i inclus à j exclu de la liste `l`.

Exercice 1 – Faut-il trier pour mieux rechercher ?

Dans ce problème, on se demande s'il est intéressant de trier une liste avant d'y rechercher un élément ou s'il est préférable d'opérer la recherche directement dans la liste.

1. Ecrire une fonction `rech_simple` de complexité linéaire renvoyant la position d'un élément `e` dans un liste `L` quelconque ou `None` si l'élément n'appartient pas à la liste.
2. Si la liste est triée, on sait qu'une recherche dichotomique est plus efficace. On en donne l'algorithme en langage Python :

```
1  def rech_dicho(e,L):
2      g=0
3      d=len(L)-1
4      while g<=d:
5          m=(d+g)//2
6          if e>L[m]:
7              g=m+1
8          elif e<L[m]:
9              d=m-1
10         else :#x==T[m]:
11             return m
12     return None
```

2.1 Soit la liste `L=[10, 23, 30, 33, 38, 39, 40, 43, 52, 66, 90, 95, 97, 115, 117]`. Donner le nombre de comparaisons opérées lors de l'appel `rech_dicho(L,27)`.

2.2 Montrer que la complexité au pire de la fonction de recherche dichotomique est $O(\log_2(n))$ où n est la longueur de la liste passée en argument.

3. Pour trier la liste, on propose la fonction `tri` suivante :

```
1 def tri(L):
2     n = len(L)
3     for i in range(n):
4         for j in range(i,0,-1):#parcourt la liste a l'envers
5             if L[j]<L[j-1]:
6                 L[j],L[j-1] = L[j-1],L[j]
```

3.1 Appliquer la fonction `tri` à la liste `[11, 48, 25, 6, 24]`. On représentera l'état de la liste pour chaque itération sur `i`.

3.2 Soit `L` une liste non vide d'entiers ou de flottants. Montrer que « la liste `L[0:i+1]` (avec la convention Python) est triée par ordre croissant à l'issue de l'itération i » est un invariant de boucle. En déduire que `tri(L)` trie la liste `L`.

3.3 Que vaut la complexité au pire de la fonction `tri` ?

4. On suppose qu'en moyenne, on est amené à réaliser quotidiennement R recherches dans des listes et E enregistrement de listes. En supposant toutes les listes de même taille n, pour quelless valeurs du rapport $\frac{R}{E}$ est-il intéressant de trier les listes avant d'y rechercher un élément plutôt que d'utiliser simplement la fonction `rech_simple` ?

5. Même question si on utilise un algorithme de tri de complexité $O(n\log_2(n))$.

Exercice 2 – Séquençage génomique

Ce problème aborde quelques aspects du séquençage génomique.

1. On considère une séquence nucléique d'ADN telle que

`ATATACGGATCGGCTGTTGCCTGCGTAGTAGCGT`

La distance de Hamming mesure la différence entre deux séquences de même taille en dénombrant les positions pour lesquelles les bases des deux séquences sont différentes.

1.1 Calculer la distance de Hamming entre les séquences :

`ATATACAGATCGGCTGTTGCGTGCGTAGTTGCGT`
`ATATACGGATCGGCTGTTGCCTGCGTAGTAGCGT`

1.2 Ecrire un programme calculant la distance de Hamming entre deux séquences nucléiques.

2. L'acide ribonucléique messager (ARN messager ou ARNm) est une copie transitoire d'une portion de l'ADN correspondant à un ou plusieurs gènes. L'ARNm est utilisé comme intermédiaire par les cellules pour la synthèse des protéines. Il est constitué d'un enchaînement des bases A,U, G et C. On appelle *codon* un ensemble de trois bases adjacentes. Chaque codon code un acide aminé sauf AUG qui donne le début de la séquence codante et UGA qui en code la fin. La protéine codée par le gène est alors constituée de l'enchaînement des acides aminés codés par chaque codon (fig.1).

2.1 Ecrire une fonction `Start` qui recherche la position de la première occurrence du codon AUG dans une séquence d'ARNm.

2.2 Expliquer ce que fait le programme fig.2 et donner sa complexité. La complexité de `rech(codon)` est constante.

1re base	2e base U		2e base C		2e base A		2e base G	
U	UUU	F Phe	UCU	S Ser	UAU	Y Tyr	UGU	C Cys
	UUC	F Phe	UCC	S Ser	UAC	Y Tyr	UGC	C Cys
	UUA	L Leu	UCA	S Ser	UAA	STOP Ocre	UGA	STOP Opale / U Sec / W Trp
	UUG	L Leu / START	UCG	S Ser	UAG	STOP Ambre / O Pyl	UGG	W Trp
C	CUU	L Leu	CCU	P Pro	CAU	H His	CGU	R Arg
	CUC	L Leu	CCC	P Pro	CAC	H His	CGC	R Arg
	CUA	L Leu	CCA	P Pro	CAA	Q Gln	CGA	R Arg
	CUG	L Leu	CCG	P Pro	CAG	Q Gln	CGG	R Arg
A	AUU	I Ile	ACU	T Thr	AAU	N Asn	AGU	S Ser
	AUC	I Ile	ACC	T Thr	AAC	N Asn	AGC	S Ser
	AUA	I Ile	ACA	T Thr	AAA	K Lys	AGA	R Arg
	AUG	M Met & START	ACG	T Thr	AAG	K Lys	AGG	R Arg
G	GUU	V Val	GCU	A Ala	GAU	D Asp	GGU	G Gly
	GUC	V Val	GCC	A Ala	GAC	D Asp	GGC	G Gly
	GUA	V Val	GCA	A Ala	GAA	E Glu	GGA	G Gly
	GUG	V Val / START	GCG	A Ala	GAG	E Glu	GGG	G Gly

FIGURE 1 – Traduction des codons de l'ARNm en acide aminé

```
1  def ARNm_to_prot(seq,code):
2      d=start(seq)
3      f=stop(seq)
4      prot=[]
5      a=""
6      i=d
7      j=i+3
8      while a !="Stop" and j<=f:
9          codon=seq[i:j]
10         a=rech(codon)
11         prot.append(a)
12         i=j
13         j=i+3
14     return prot
```

FIGURE 2 – Algorithme 2

Exercice 3 – Autour des tables et des fonctions de hachage

Un fonction de hachage est une fonction h qui associe à une chaîne de caractères c quelconque une chaîne de caractères de longueur fixée p, $h(c)$, appelée *empreinte* de c. Les fonctions de hachage sont par exemple utilisées en cryptographie, pour authentifier des données ou des utilisateurs et dans une structure de donnée appelée table de hachage.

On appelle *collision* d'une fonction de hachage tout couple (c_1, c_2) ayant même empreinte i.e. tel que $h(c_1) = h(c_2)$.

1. Un (mauvais) exemple de fonction de hachage :

$$h : c \to c\%255$$

où c est un entier naturel (c'est-à-dire une chaîne de chiffres !)

1.1 Calculer l'empreinte de 1753.

1.2 Montrer que la fonction h n'est pas résistante aux collisions c'est-à-dire qu'on peut aisément trouver deux chaînes c_1 et c_2 de même empreinte ($h(c_1) = h(c_2)$).

2. Mots de passe

2.1 On recommande de choisir une suite aléatoire de n caractères parmi les 95 caractères imprimables de la table ASCII pour mot de passe plutôt que, par exemple, un mot du dictionnaire. Expliquer pourquoi par un argument de complexité en temps.

2.2 Les mots de passe sont stockées sur une machine distante ou locale. Mais ils ne sont évidemment pas conservés en clair : une empreinte est générée par application d'une fonction de hachage et c'est cette empreinte qui est stockée puis comparée à l'empreinte générée par hachage de la chaîne de caractères entrée au moment de l'authentification. On note m la taille de l'empreinte.

2.2.1 Ecrire une fonction `authentification` prenant en entrée l'empreinte `e` du mot de passe, le mot de passe `p` entrée par la personne cherchant à s'authentifier et renvoyant `True` si l'authentification est réussie, `False` sinon. On pourra supposer définie la fonction de hachage `h`.

2.2.2 Expliquer pourquoi il est impératif que la fonction `h` soit résistante aux collisions.

3. Table de hachage

Un *dictionnaire* est une structure de données permettant de lire, supprimer ou insérer une donnée avec un complexité quasi-constante. On associe des clé à des valeur. Pour simplifier, on suppose que la clé est l'indice d'un tableau `D` de longueur m (les clés appartiennent donc à $\{0, m-1\}$). On stocke au total n valeurs dans le dictionnaire avec, évidemment, $n < m$. L'accès à la valeur de clé `c` se fait par l'instruction `D[c]` avec un temps constant.

3.1 Exemple : pour $m = 10$, représenter le dictionnaire dont la clé est un entier et dont la valeur indique par `True` ou `False` si ce nombre est premier.

3.2 Le plus souvent, $n \ll m$. Quel est alors l'inconvénient de l'implémentation décrite en introduction ?

3.3 Pour palier ce problème, l'indice n'est plus la clé c mais sa valeur hachée $h(c)$ et on suppose que $h(c)$ est à valeur dans $\{0, n-1\}$. C'est le principe d'une table de hachage. Quelle doit être la complexité de la fonction h pour que la lecture et l'écriture reste de complexité constante. On ignorera la gestion des collisions.