Informatique — MPSI/PCSI, 2023–2024

Devoir surveillé n° 1

Durée : 2 heures

L'usage de calculatrices est interdit. — Aucun document n'est autorisé.

La présentation, la lisibilité, l'orthographe, la qualité de la rédaction, la clarté et la précision des raisonnements entreront pour une part importante dans l'appréciation des copies. En particulier, ***les candidats devront apporter les explications et commentaires suffisants à la compréhension de leurs programmes*** *et veilleront à utiliser des noms de variables explicites. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.*

Le sujet est composé de trois exercices indépendants.

Exercice 1 – Arrêts aux stations essences

Un conducteur désire parcourir un long trajet en voiture, en s'arrêtant au besoin à des stations services le long de la route. Il dispose pour cela d'un réservoir de capacité c unités de carburant. Par commodité, on suppose que la voiture consomme une unité de carburant par km.

La liste des stations sera donnée sous la forme d'une liste de nombres entiers positifs, chaque entier désignant la distance entre la station et la station précédente. Par exemple, la liste `L=[58, 11, 79, 84, 55, 56]` signifie qu'il y a six stations possibles sur la route, la première se trouvant à 58 km du départ, la seconde à 11 km de la première, etc. On considérera que la dernière station correspond au terminus du voyage.

Pour que le trajet soit possible, il faut que la distance à parcourir entre deux stations, ou entre le départ et la première station, ne dépasse jamais la capacité du réservoir c.

Question 1.1. *Écrire une fonction* `possible` *qui prend en arguments une liste d'entiers* `L` *et une capacité entière* `c` *et détermine si le trajet est possible ou non.*

| On supposera par la suite que cette condition est systématiquement réunie.

On s'intéresse maintenant à un premier conducteur, M Pressé, qui désire faire le moins d'arrêts possibles. Il applique pour cela l'algorithme suivant : partant du départ, il rejoint la station la plus lointaine qu'il peut atteindre avec la capacité de son réservoir. Arrivé à cette station, il refait le plein, puis repart pour aller de nouveau le plus loin possible, et ainsi de suite.

Question 1.2. *Écrire une fonction* `prochaine_station` *qui prend en arguments une liste de stations* `L`*, une capacité* `c`*, un entier* `i` *représentant l'indice de la station déjà atteinte (avec* `i` *pour convention égal à* `-1` *si le trajet n'a pas encore commencé) et qui retourne l'indice de la station la plus lointaine que l'on peut atteindre sans refaire le plein. (En particulier, dans le cas où il est possible de terminer le trajet, on retournera l'indice de la dernière station.)*

Question 1.3. *Écrire une fonction* `arrets` *qui prend en arguments une liste de station* `L` *et une capacité* `c` *et retourne la liste des arrêts effectués sous la forme d'une liste d'entiers représentant les indices des stations où l'on s'arrête, la dernière étant le terminus. Par convention, cette liste devra toujours commencer par* `-1`*, l'indice correspondant au départ.*

On s'intéresse maintenant à un autre conducteur, M Eco, qui ne désire pas forcément faire le moins d'arrêts possibles, mais faire le plein là où le carburant est le moins cher possible. Pour cela, on suppose à partir de maintenant que l'on dispose d'une liste de stations L où chaque élément de la liste est un couple du type $[d_i, p_i]$ où d_i est un entier positif représentant la distance à la station précédente et p_i un nombre flottant positif représentant le prix du carburant par unité en euro à cette station. Par exemple :

```
L=[[23, 0.16], [48, 0.24], [86, 0.2], [54, 0.19], [20, 0.23], [28, 0.23]]
```

signifie que la première station est à 23 km du départ et vend le carburant 0,16 € par unité, la deuxième station à 48 km de la première et vend le carburant à 0,24 € par unité, etc.

Question 1.4. *Écrire une fonction* `prochaine_station_eco` *qui prend en arguments une liste de stations* `L`*, une capacité* `c`*, un entier* `i` *représentant l'indice de la station déjà atteinte (avec* `i` *pour convention égal à -1 si le trajet n'a pas encore commencé) et qui retourne l'indice de la station, parmi celles que l'on peut atteindre, qui vend le carburant le moins cher.*

Question 1.5. *Comment modifier la fonction* `arrets` *de la question 1.3 pour qu'elle retourne désormais la liste des arrêts à effectuer en appliquant la stratégie « économique » ?*

On désire maintenant calculer combien le conducteur va dépenser en pleins de carburant pour le trajet, en supposant qu'il refait le plein arrivé à destination pour effectuer le retour.

Question 1.6. *Écrire une fonction* `cout_pleins` *qui prend en arguments une liste des stations* `L` *et une liste des arrêts* `A` *et renvoie le coût total des pleins effectués le long du parcours.*

Exercice 2 – Quelques éléments de cryptographie

La cryptographie (du grec *crypto*, caché et *graphie*, écrire) est la science des codes secrets. Elle remonte à l'antiquité et Jules César l'a employée pour coder ses messages. Il utilisait le système le plus simple, celui des alphabets décalés d'un ou plusieurs crans (où l'on remplace, par exemple, A par B, B par C, etc.).

Dans cet exercice, on pose la chaîne de caractères :

```
Alph = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ '
```

On appelle **texte** toute chaîne de caractères (non vide) ne comportant que des caractères parmi les 27 qui apparaissent dans `Alph`. Par exemple, la chaîne de caractères `"LYCEE JEAN PERRIN"` est un texte. On considérera de plus qu'un texte ne peut pas commencer ni finir par une espace.

Cryptographie affine

La cryptographie affine une méthode de cryptographie basée sur un chiffrement par substitution mono-alphabétique, c'est-à-dire que la lettre d'origine n'est remplacée que par une unique autre lettre. On commence par remplacer chaque lettre par son rang dans l'alphabet en commençant au rang 0 :

clair	A	B	C	D	E	⋯	X	Y	Z
rang	0	1	2	3	4	⋯	23	24	25

Deux entiers a et b sont choisis comme clef.
Chaque lettre claire est d'abord remplacée par son équivalent numérique x puis chiffrée par le calcul du reste de la division euclidienne par 26 de l'expression affine $ax+b$, soit l'unique entier $y \equiv ax+b\ [26]$ congru à $ax+b$ modulo 26 dans l'intervalle $[\![0, 25]\!]$. On peut montrer que le chiffrement est possible si, et seulement, si a et 26 sont premiers entre eux.

Par exemple, pour $(a, b) = (3, 5)$, on obtient :

clair	A	B	C	D	E	⋯	X	Y	Z
rang	0	1	2	3	4	⋯	23	24	25
crypté	F	I	L	O	R	⋯	W	Z	C

Dans tout l'exercice, il sera possible d'utiliser les fonctions `chr`, qui prend en argument un point de code Unicode (entier) i et qui renvoie la caractère c associé et `ord` qui prend en argument un caractère Unicode et renvoie le point de code Unicode (entier) associé. Par exemples :

```
>>> ord('A'),ord('B'),ord('Z')
(65, 66, 90)
>>> chr(65),chr(66),chr(90)
('A', 'B', 'Z')
```

Utilisation des alphabets clairs et chiffrés

Question 2.1. *Implémenter la fonction* `alph_clair` *qui ne prend pas d'argument et qui renvoie les liste des car caractères de l'alphabet « clair », c'est-à-dire la liste* `['A','B',...,'Z']`.

Question 2.2. *Implémenter la fonction* `alph_crypt` *prenant en argument les entiers* `a` *et* `b` *constituant la clef de chiffrement et qui renvoie la liste des caractères de l'alphabet chiffré. Le premier élément de la liste correspond au caractère qui code le caractère* `'A'` *et le dernier élément celui qui code le caractère* `'Z'`.

Pour la suite de cette partie, on initialise les variables globales suivantes, la clé de chiffrement ayant été choisie au préalable :

```
clair = alph_clair()
crypt = alph_crypt(a,b)
```

Question 2.3. *Ecrire la fonction* `indexcar` *qui prend en argument une liste et un des éléments de cette liste (par exemple un caractère et l'un des alphabets, clair ou chiffré et qui renvoie l'indice, supposé unique, de cet élément dans la liste.*

Question 2.4. *En utilisant les fonctions précédentes, implémenter la fonction* `texte_crypt` *prenant en argument un texte « en clair » et renvoyant le texte chiffré. Les espaces ne seront pas modifiés.*

Question 2.5. *Implémenter de même la fonction* `texte_decrypt` *prenant comme argument un texte « chiffré », et renvoyant le texte déchiffré.*

Chiffrement et déchiffrement direct

Question 2.6. *Implémenter la fonction* `texte_crypt_dir` *prenant en arguments un texte « en clair »* `T` *et les entiers* `a` *et* `b` *et renvoyant le texte chiffré, par* ***chiffrement direct****, c'est-à-dire sans utiliser les alphabets clair et chiffré.*

Pour déchiffrer un texte chiffré, il faut trouver la clé de déchiffrement. Le caractère chiffré associé à un caractère « clair » de rang $x \in [\![0, 25]\!]$ est le caractère de rang $y = ax + b$ [26] $\in [\![0, 25]\!]$. Inversement, on admettra que le caractère « clair » associé au caractère chiffré y est le caractère de rang $x = u(y - b)$ [26] $\in [\![0, 25]\!]$ où u est l'inverse modulaire de a dans l'ensemble $\mathbb{Z}/26\mathbb{Z}$. Pour le calculer, on admettra qu'il suffit de trouver le premier entier supérieur ou égal à 1 vérifiant $au \equiv 1$ [26] .

Question 2.7. *Implémenter la fonction* `cle_decrypt` *prenant l'entier* `a` *comme argument qui sert à chiffrer le teste et renvoyant son inverse modulaire* `u`.

Question 2.8. *En déduire une implémentation de la fonction* `texte_decrypt_dir` *prenant comme arguments un texte chiffré* `T` *et les entiers* `a` *et* `b` *et renvoyant directement le texte déchiffré, en utilisant la clé de déchiffrement.*

Crackage du cryptage affine

Le chiffrement par substitution monoalphabétique d'un texte un peu long est malheureusement très facile à casser par **analyse fréquentielle**. En effet, dans une langue donnée, la distribution statistique des lettres dans un texte est assez stable. En français, la lettre la plus fréquente est le `'E'`, puis viennent `'S'`, `'A'`, `'N'` etc. La comparaison des fréquences du texte chiffré et de la langue d'origine du texte «en clair» permet de d'identifier un certain nombre de caractères, le `'E'` en particulier. L'ensemble du procédé est expliqué dans la nouvelle d'Edgar Poe : *Le scarabée d'or*.

Question 2.9. *Implémentation la fonction* `an_freq` *prenant comme argument un texte chiffré et renvoyant la liste du nombre d'occurrences de chaque caractère dans le texte. Le premier élément de la liste correspondra au nombre d'occurrences du caractère* `'A'` *dans le texte chiffré, le dernier élément au nombre d'occurrences du caractère* `'Z'`.

Exercice 3 – Codage de brins d'ADN

L'acide désoxyribonucléique (ADN) est une molécule biologique présente dans toutes les cellules. Elle est constituée de deux brins en hélice. Chaque hélice est une longue chaîne formée par l'enchainement de nucléotides reliés entre eux par des liaisons $5' \to 3'$. Chaque nucléotide unit, par une liaison N-glycosidique, un désoxyribose 5'phosphate à une **base** purique (adénine [A], guanine [G]) ou pyrimidique (thymine [T], cytosine [C]). Les deux chaînes s'associent entre elles par complémentarité des bases :

— l'adénine ne peut se lier qu'à la thymine, avec deux liaisons hydrogène ;
— la cytosine ne peut se lier qu'à la guanine, avec trois liaisons hydrogène.

L'unité d'information élémentaire est le triplet de nucléotide ou **codon**. Chaque **gène** est une séquence de codons qui porte l'information nécessaire à la construction d'une protéine ou d'une molécule d'acide ribo nucléique (ARN) fonctionnelle. Cette séquence commence toujours par une séquence ATG (codon méthionine ou *Met*) et se termine par un des codons stop (TAA, TGA ou TAG). L'ensemble des zones d'ADN codant des protéines ne représente qu'une toute petite fraction (environ 10 %) de la totalité de l'ADN. La majorité de l'ADN est constitué de séquences d'une dizaine de bases répétées un grand nombre de fois.

Lorsqu'une cellule a besoin d'une protéine, le gène est transcrit en ARN messager (ARNm), avec la même séquence que la partie codante du gène, la base T étant remplacée par la base U (uracile). On appelle **code génétique** l'ensemble des relations faisant correspondre les différents triplets avec les différents acides aminés.

UUU	*Phe*	UCU	*Ser*	UAU	*Tyr*	UGU	*Cys*	CUU	*Leu*	CCU	*Pro*	CAU	*His*	CGU	*Arg*
UUC		UCC		UAC		UGC		CUC		CCC		CAC		CGC	
UUA	*Leu*	UCA		UAA	*stop*	UGA	*stop*	CUA		CCA		CAA	*Gln*	CGA	
UUG		UCG		UAG		UGG	*Trp*	CUG		CCG		CAG		CGG	
AUU	*Ile*	ACU	*Thr*	AAU	*Asn*	AGU	*Ser*	GUU	*Val*	GCU	*Ala*	GAU	*Asp*	GGU	*Gly*
AUC		ACC		AAC		AGC		GUC		GCC		GAC		GGC	
AUA		ACA		AAA	*Lys*	AGA	*Arg*	GUA		GCA		GAA	*Glu*	GGA	
AUG	*Met*	ACG		AAG		AGG		GUG		GCG		GAG		GGG	

FIGURE 1 – Le code génétique.

Il est devenu habituel de dire que l'ADN se présente comme un texte composé à l'aide de quatre lettres A, C, G et T qui s'enchaînent sans interruption et qui est orienté avec un début et une fin ; par exemple :

```
B = "AATCGCTTAATGAGCTTTACGTGCATAGCAGATGAAGAAGTTGATTAGACCATGAGCTTTACAGCAGATGTGATTAG"
```

On appellera **brin** une telle chaîne de caractères.

Dans cet exercice la clause `in` et les comparateurs `==` et `!=` agissant sur les séquences (listes, chaînes de caractères) sont interdits.

Question 3.1. *Définir une fonction* `base` *qui prend comme argument un caractère* `b` *associé à un nucléotide et qui renvoie le booléen* `True` *si c'est une des base azotée d'ADN,* `False` *sinon.*

Question 3.2. *Définir une fonction* `ADN` *qui prend comme argument une chaîne de caractères* `B` *correspondant à un brin et qui renvoie le booléen* `True` *si c'est un brin d'ADN,* `False` *sinon.*

Question 3.3. *Définir une fonction* `ARNm` *qui prend comme argument une chaîne de caractères* `B` *correspondant à un brin d'ADN et qui renvoie le brin d'ARN messager correspondant.*

Question 3.4. *Définir une fonction* `codon` *qui prend comme arguments deux chaines de caractères* `A` *et* `B` *correspondant chacune à un codon et qui renvoie le booléen* `True` *s'ils sont égaux,* `False` *sinon.*

Question 3.5. *Définir une fonction* `codons` *qui prend comme arguments une chaine de caractères* `C` *et une liste de chaînes de caractères* `L`*, chacune correspondant à un codon, et qui renvoie le booléen* `True` *si le codon* `C` *est dans* `L`*,* `False` *sinon.*

Question 3.6. *Définir une fonction* `gene` *qui prend comme argument une chaîne de caractères* `B` *correspondant à un brin de nucléotides et qui renvoie le booléen* `True` *si c'est un gène,* `False` *sinon.*

Question 3.7. *Définir une fonction* `genes` *qui prend comme argument une chaîne de caractères* `B` *correspondant à un brin de nucléotides et qui renvoie les différents gènes du brin.*

On suppose que l'on dispose d'une fonction `AA` qui à chaque codon associe son acide aminé. On rappelle qu'une **protéine** est une séquence d'acides aminés issue de la traduction d'un gène, privé de ses codons *Met* (start) et stop.

Question 3.8. *Définir une fonction* `proteine` *qui prend comme argument un gène sous forme de chaîne de caractères* `G` *et qui renvoie la protéine transcodée correspondante.*

Question 3.9. *Définir une fonction* `proteines` *qui prend comme argument une chaîne de caractères* `B` *correspondant à un brin de nucléotides et qui renvoie les différents protéines transcodées avec les gènes du brin.*

— Fin de l'énoncé —