Concours Blanc - Epreuve d'informatique

Consignes

- La calculatrice n'est **pas autorisée**.
- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

❑ Exercice 1 : *Somme de termes consécutifs*

■ Partie I : Somme des éléments d'une liste

Q1– Ecrire une fonction `somme` *itérative* qui prend en paramètre une liste d'entiers `lst` et renvoie la somme de ses éléments. Par exemple, `somme([4, 8, 3, 1])` renvoie `16`.

Q2– Ecrire une version *récursive* de la fonction `somme`.

■ Partie II : Somme maximale de k termes consécutifs

On s'intéresse maintenant au calcul de la somme maximale de k termes consécutifs d'une liste d'entiers. Par exemple, si $k = 3$ et que la liste est $[2, 7, -1, 3, 8, -5]$, la somme maximale de 3 termes consécutifs est 10 (correspondant à la somme $(-1) + 3 + 8$). Dans tout l'exercice, on notera `lst` la liste d'entiers et `n` sa taille, et on supposera que `n>0` (la liste est non vide) et que `k` est inférieur ou égal à `n`.

Q3– Ecrire une fonction `sommek` qui prend en paramètre une liste `lst`, un entier `k` et un entier `i` et renvoie la somme des `k` termes consécutifs de `lst` à partir de l'indice `i`. C'est à dire que `sommek(lst, k, i)` renvoie `lst[i] + lst[i+1] + ... + lst[i+k-1]`. On supposera que cette somme est définie, c'est à dire que `i>=0` et `i+k-1` est inférieur strictement à `n`.

Q4– En utilisant la fonction `sommek`, écrire une fonction `sommek_max` qui prend en paramètre une liste `lst` et un entier `k` et renvoie la somme maximale de `k` termes consécutifs de `lst`. On procédera en testant toutes les valeurs possibles de l'indice de départ `i` pour calculer la somme de `k` termes consécutifs.

Q5– On veut maintenant exprimer la complexité de `sommek_max` en nombre d'additions. Donner le nombre d'additions lors d'un appel à `sommek` et en déduire en fonction de `k` et `n` le nombre d'additions effectuées par `sommek_max`.

Q6– En observant que la somme des k premiers termes à partir de l'indice `i+1` s'obtient à partir de celle à l'indice `i` en effectuant seulement deux additions, proposer et écrire une nouvelle version plus efficace de la fonction `sommek_max`.

Q7– Quelle est la complexité en nombre d'additions de cette nouvelle version en fonction de `n` et de `k` ?

❑ Exercice 2 : *anagrammes*

Deux mots *de même longueur* sont anagrammes l'un de l'autre lorsque l'un est formé en réarrangeant les lettres de l'autre. Par exemple :

— *chien* et *niche* sont des anagrammes.

— *epele* et *pelle*, ne sont pas des anagrammes, en effet bien qu'ils soient formés avec les mêmes lettres, la lettre *l* ne figure qu'à un seul exemplaire dans *epele* et il en faut deux pour écrire *pelle*.

Le but de l'exercice est d'écrire une fonction `anagrammes` qui prend en argument deux chaines de caractères et qui renvoie `True` si ces deux chaines sont des anagrammes et `False` sinon.

■ Partie I : Une approche récursive

Dans cette partie, on utilise un algorithme récursif afin de tester si deux chaines de caractères sont des anagrammes. Si les deux chaines sont vides alors ce sont des anagrammes, sinon on supprime le premier caractère de la première chaine de la seconde et on effectue un appel récursif sur ce qu'il reste des deux chaines. Par exemple sur `chien` et `niche`, le premier appel récursif s'effectuerait entre `hien` et `nihe` car on supprime le `c` des deux chaines.

Q8– Ecrire une fonction `supprime_premier` qui prend en argument un caractère `car` et une chaine `chaine` et renvoie la chaine obtenue en supprimant la premier occurence de `car` dans `chaine`.
Par exemple `supprime_premier("c","niche")` renvoie `"nihe"`. Si `car` n'est pas dans chaine alors on renvoie `chaine` sans modification.
Par exemple `supprime_premier("l","Python")` renvoie `"Python"`

Q9– Ecrire une fonction récursive `anagrammes_rec` qui prend en argument deux chaines de caractères et renvoie `True` si ce sont des anagrammes l'une de l'autre et `False` sinon.
Par exemple, `anagrammes_rec("niche","chien")` renvoie `True`.

■ **Partie II** : Une approche itérative

Dans cette partie, on utilise une approche itérative en manipulant les dictionnaires de Python.

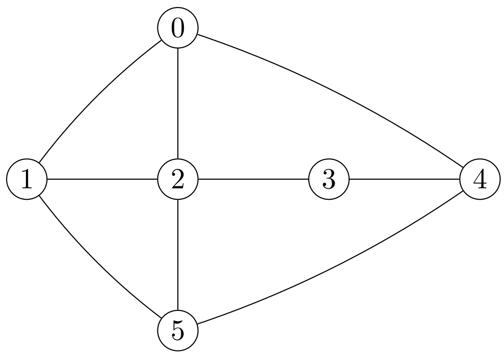
Q10– Ecrire une fonction `cree_dico` qui prend en argument une chaine de caractères et renvoie un dictionnaire dont les clés sont les caractères composant la chaine et les valeurs leur nombre d'apparition.
Par exemple, `cree_dico("epele")` renvoie le dictionnaire `{'e':3, 'p':1, 'l':1 }` en effet dans le mot `'epele'`, `'e'` apparaît à trois reprises et `'l'` et `'p'` chacun une fois.

Q11– Ecrire une fontionc `egaux` qui prend en argument deux dictionnaires et renvoie `True` si ces deux dictionnaires sont égaux (c'est-à-dire contiennent exactement les mêmes clés avec les mêmes valeurs) et `False` sinon.
Par exemple, `egaux({'e':3, 'p':1, 'l':1 },{'p':1,'e':2,'l':2})` renvoie `False`
⚠ on s'interdit ici d'utiliser le test d'égalité `==` entre deux dictionnaires et on écrira un parcours de dictionnaire.

Q12– Ecrire une fonction `anagrammes_iter` qui prend en argument deux chaines de caractères et renvoie `True` si ce sont des anagrammes et `False` sinon.

❑ **Exercice 3** : *Coloration d'un graphe*

Dans toute la suite de l'exercice, on considère un graphe non orienté $G = (S, A)$ où chaque sommet est identifié par un entier. On supposera ce graphe représenté en Python par un dictionnaire dont les clés sont les sommets et les valeurs les listes d'adjacence. Par exemple, le graphe G suivant :



est représenté par le dictionnaire `ex` suivant :

```
ex = {0: [1, 2, 4],
 1: [0, 2, 5],
 2: [0, 1, 3, 5],
 3: [2, 4],
 4: [0, 3, 5],
 5: [1, 2, 4]}
```

■ **Partie I** : Questions préliminaires

Q13– Rappeler la définition du *degré* d'un sommet dans un graphe et écrire une fonction `degre` qui prend en argument un graphe (représenté par un dictionnaire tel que ci-dessus) et un sommet et renvoie son degré.

Q14– Ecrire une fonction `appartient` qui prend en argument une liste d'entiers `lst` et un entier `x` et renvoie `True` si `x` est dans `lst` et `False` sinon.

Q15– En utilisant la fonction `appartient`, écrire une fonction `sont_adjacents` qui prend en argument deux sommets et un graphe et renvoie `True` si ces deux sommets sont adjacents et `False` sinon.

■ **Partie II** : Coloration d'un graphe

La coloration de graphe consiste à attribuer une couleur à chacun de ses sommets de manière que deux sommets reliés par une arête soient de couleur différente. On s'intéresse généralement une coloration utilisant un *minimum* de couleurs.

Q16– Montrer que le graphe G ci-dessus peut être colorer avec seulement 3 couleurs en dessinant ce graphe et en faisant figurer à côté de chaque sommet un chiffre indiquant sa couleur.

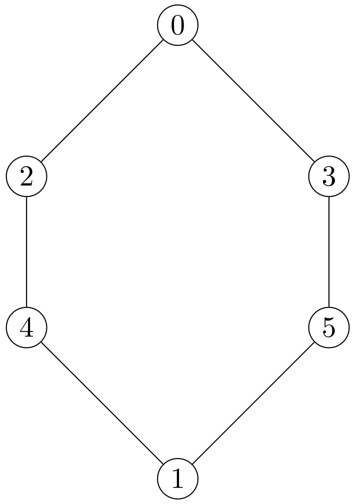
Q17– Donner un exemple de graphe à n sommets dont la coloration nécessite au moins n couleurs.

Q18– On représente une coloration d'un graphe à n sommet par une liste de n entiers. L'élément d'indice i de cette liste est la couleur du sommet i. Par exemple, pour le graphe G donné en exemple la coloration suivante : $[1, 3, 3, 1, 2, 3]$ indique que les sommets 0 et 3 sont de couleur 1, les sommets 1, 2 et 5 sont de couleur 3 et le sommet 4 est de couleur 2. Ecrire une fonction `est_valide` qui prend en argument un graphe et une coloration et renvoie `True` si la coloration est valide (c'est-à-dire si deux sommets adjacents n'ont pas la même couleur) et `False` sinon.

■ **Partie III** : Coloration gloutonne d'un graphe

On propose la méthode gloutonne suivante afin de colorier un graphe : on parcourt les sommets *dans leur ordre de numérotation* et on leur attribue la plus petite couleur disponible. Sur le graphe G donné en exemple, le sommet 0 reçoit la couleur 1, puis le sommet 1 la couleur 2, le sommet 2 la couleur 3. Ensuite le sommet 3 reçoit la couleur 1 (car comme il n'est pas adjacent au sommet 0 cette couleur est disponible), puis le sommet 4 reçoit la couleur 1 et enfin le sommet 5 reçoit la couleur 1. La coloration finale obtenue est donc $[1, 2, 3, 1, 2, 1]$.

Q19– On considère maintenant le graphe H ci-dessous :



Montrer que H peut-être coloré avec seulement deux couleurs, puis donner le résultat de la coloration avec la méthode gloutonne, que peut-on en conclure ?

Q20– Montrer qu'on peut renuméroter les sommets de H de façon à ce que l'algorithme glouton fournisse une coloration n'utilisant que deux couleurs.

Q21– Ecrire une fonction `colorie` qui prend en argument un graphe et renvoie une coloration valide de ce graphe en utilisant la méthode gloutonne décrite ci-dessus.