Interrogation d'informatique n°2. *Barème sur 29.5 pts.*

Exercice A. Chemins multicolores dans un graphe colorié

On considère un ensemble S de n villes reliées par des routes (les routes pouvant être à sens unique).

Les villes sont numérotées de 0 à $n-1$. Ainsi, on considère $S = \{0, 1, ..., n-1\}$.

Pour signifier qu'il existe une route reliant une ville x à une ville y, on utilise la notation $x \to y$.

Le graphe ainsi défini est codé par une liste G de n listes : pour toute ville $x \in S$, la liste $G[x]$ est la liste des villes y pour lesquelles il existe une route reliant x à y, c'est-à-dire : $y \in G[x]$ ssi $x \to y$.

Pour chaque ville x, on note m_x le nombre de villes voisines de x, c'est-à-dire la longueur de $G[x]$.

On pose $m = \sum_{x \in S} m_x$.

Par exemple, le graphe $\begin{array}{ccccc} & & 1 & \to & 0 \\ & \swarrow & & \searrow & \downarrow \\ 4 & \leftarrow & 3 & \rightleftharpoons & 2 \end{array}$ est codé par `G = [[2],[0,2,4],[3],[2,4],[]]`.

On a ici $n = 5$ (nombre de villes) et $m = 7$ (nombre de routes).

On fixe deux villes s et $t \in S$.

D'autre part, à chaque ville est attribuée une couleur selon le pays où elle se trouve.

Chaque couleur est codée par un entier $k \in \{0, 1, ..., r+1\}$. Il y a donc $(r+2)$ couleurs possibles.

On suppose de plus que les villes sont colorées selon la règle suivante :

- la ville s est colorée par la couleur 0 et la ville t est colorée par la couleur $r+1$

- toutes les autres villes sont colorées par une couleur appartenant à $\{1, 2, ..., r\}$.

Le coloriage est codé en PYTHON par un tableau C de longueur n, de sorte que pour tout sommet x, la valeur de $C[x]$ donne la couleur de x.

Ainsi, $C[s]$ vaut 0 et $C[t]$ vaut $r+1$, et pour $x \notin \{s, t\}$, $C[x] \in \{1, 2, ..., r\}$ est la couleur de la ville x.

Un ensemble A de villes de S, c'est-à-dire une partie de S, peut être codée de deux façons :

- soit par la liste L des éléments de A

- soit par son tableau caractéristique, c'est-à-dire un tableau T de longueur n tel que :

pour toute ville $x \in S$, $T[x]$ vaut 1 si x appartient à L, et vaut 0 sinon.

Par exemple, la partie $A = \{1, 4, 5\}$ de $S = \{0, 1, ..., 6\}$ peut être codée

- soit par la liste $L = [1, 4, 5]$ dont la longueur est le nombre d'éléments de A

- soit par le tableau $T = [0, 1, 0, 0, 1, 1, 0]$ de longueur n (où n est le cardinal de S).

1) [4 pts] Écrire une fonction `voisins(T,G,C,k)` qui étant donnée une partie A de S codée par son tableau caractéristique T, un graphe G, un entier k compris entre 1 et r, renvoie le tableau caractéristique R de **l'ensemble des villes y de couleur k telles que $\exists\, x \in A$, $x \to y$.**

Autrement dit, $R[y] = 1$ ssi on a $C[y] = k$ et s'il existe $x \in A$ tel que $y \in G[x]$.

Préciser *sans justification* la complexité de la fonction en fonction de n et de $\sum_{x\in A} m_x$.

2) [3 pts] Écrire une fonction `cheminColore(G,C,s,t,r)` qui renvoie `True` ssi il existe un chemin $s \to v_1 \to ... \to v_r \to t$ reliant s à t et tel que $\forall k \in [\![1, r]\!]$, $C[v_k] = k$.

Autrement dit : On cherche s'il existe un chemin traversant tous les pays par ordre croissant de couleur et en visitant à chaque fois une ville par pays.

On utilisera une boucle `for`, et à chaque étape de la boucle, on fera appel à la fonction `voisins`.

3) [2 pts] Montrer que la complexité de `CheminColore` est en $O((r+1)n+m)$.

Indication : Noter A_k la partie de S obtenue à l'étape k, et faire intervenir les sommes $\sum_{x\in A_k} m_x$..

4) [2 pts] On souhaite améliorer l'efficacité de l'algorithme afin d'obtenir une complexité en $O(n+m)$.

Écrire une fonction `partition(C,r)` qui renvoie une liste S de $(r+2)$ listes de sorte que pour tout $k \in \{0, 1, 2, ..., r+1\}$, la liste $S[k]$ contienne la liste des villes de couleur k.

On demande une fonction de complexité $O(n)$.

On va dans la suite reprendre l'algorithme utilisé aux questions 1) et 2) mais en utilisant un seul tableau caractéristique.

5) [2 pts] Écrire une procédure `voisins2(G,T,S,C,k)` qui étant donnés :

- la liste de listes G codant le graphe
- un tableau caractéristique T codant une partie A de S
- une liste S de $(r+2)$ listes codant la partition de S par couleurs (cf question 4))
- un tableau donnant les couleurs des villes
- un entier $k \in \{1, 2, ..., r+1\}$

modifie T de sorte à attribuer dans T la valeur 1 à toute ville de couleur k qui est voisine de l'une des villes de couleur $(k-1)$ appartenant à A. Les valeurs de T en les villes de couleur différente de k restent inchangées.

6) [2 pts] En déduire une nouvelle fonction `CheminColore2` de complexité $O(n+m)$.

7) *Question supplémentaire* : Justifier la complexité en $O(n+m)$.

Exercice B. Mots de Dyck (= expressions bien parenthésées) et permutations de pile

Soit L une liste de longueur n dont chaque élément vaut 1 ou -1.

On dit que L est un mot de Dyck ssi la liste L contient autant de 1 que de -1, et si pour tout $k \leq n$, les k premiers termes de la liste contiennent au moins autant de 1 que de -1.

Il s'agit en fait de toutes les formules correspondant à des parenthésages corrects :
Par exemple, la formule parenthésée $e = (()(()()))$ correspond au mot de Dyck `[1,1,-1,1,1,-1,1,-1,-1,-1]`.

De même, la formule $e = ()()$ correspond à `[1,-1,1,-1]`.
La liste vide `[ ]` est aussi un mot de Dyck.

1) [2.5 pts] Écrire en PYTHON une fonction `Dycko(L)` qui étant donnée une liste L dont tous les éléments valent 1 ou -1, renvoie `True` si L est un mot de Dyck, et `False` sinon.

On demande un algorithme dont le nombre d'opérations est en $O(n)$, où n est la longueur de L.

Permutations de piles et mots de Dyck

Une permutation σ de $E = \{0, 1, 2, ..., n-1\}$ est codée en PYTHON par la liste $A = [\sigma(0), \sigma(1), \sigma(2)..., \sigma(n-1)]$.

On étudie l'entrée et le départ de n éléments dans une pile.

Les éléments sont numérotés de 0 à $n-1$ par ordre croissant d'ajout dans la pile :

- La pile est initialement vide

- Chaque élément i est ajouté une seule fois dans la pile et en est extrait une seule fois aussi.

- L'élément de numéro i est entré dans la pile après les éléments de numéro $j < i$ et avant les éléments de numéro $j > i$. Par exemple, l'élément 0 est entré en premier dans la pile.

- Les ajouts et les suppressions sont toujours en tête de pile. On note $\sigma(0), \sigma(1), .., \sigma(n-1)$ les numéros des éléments par ordre de leur sortie de pile. Par exemple, $\sigma(0) = 2$ signifie que le premier élément extrait de la pile est l'élément 2 (c'est-à-dire celui qui a été ajouté à la pile après les éléments 0 et 1). Et $\sigma(n-1)$ est le numéro du dernier élément sorti de la pile (ensuite, la pile est à nouveau vide).

On définit ainsi une permutation de $\{0, 1, 2, ..., n-1\}$. On dit alors que σ est une permutation de pile.

A chaque permutation de pile σ, on associe une liste $L = f(\sigma)$ définie de la façon suivante : on associe le chiffre $(+1)$ lorsqu'un élément est ajouté dans la pile, et le chiffre (-1) lorsqu'un élément est supprimé de la pile. L'ordre des termes dans la liste L correspond à la succession temporelle de ces opérations dans la pile. Ainsi, la liste commence par $(+1)$ car on commence par ajouter un élément dans la pile.

Exemple : Sur une pile initialement vide, on effectue successivement les opérations suivantes :
- on ajoute 0 à la pile
- on ajoute 1 à la pile
- on retranche 1 à la pile ($\rightarrow$ donc $\sigma(0) = 1$)
- on ajoute 2 à la pile
- on retranche 2 à la pile ($\rightarrow$ donc $\sigma(1) = 2$)
- on retranche 0 à la pile ($\rightarrow$ donc $\sigma(2) = 0$)

On obtient respectivement la permutation $A = [1, 2, 0]$ et la liste $L = [1, 1, -1, 1, -1, -1]$.

On *admet* que les listes L ainsi obtenues sont les mots de Dyck de longueur $2n$.
Plus précisement, on admet qu'on établit même ainsi une bijection $f : \sigma \rightarrow L$ de l'ensemble $\mathcal{P}$ des permutations de pile de $\{0, 1, 2, ...n-1\}$ sur l'ensemble $\mathcal{D}$ des mots de Dyck de longueur $2n$.

2) [1 pt] On considère $L = [1, 1, ...1, 1, -1, -1, ..., -1]$ et $L' = [1, -1, 1, -1, 1, -1, ..., 1, -1]$ de longueur $2n$.

Préciser sans justification les permutations de pile σ et σ' associées respectivement aux listes L et L'.

3) [2 pts] Écrire une fonction `permut_de_dyck(L)` qui étant donné un mot de Dyck L renvoie la permutation σ de pile associée (c'est-à-dire qui étant donnée la liste L renvoie $A = f^{-1}(L)$).

On proposera un algorithme de complexité $O(n)$.

Par exemple, `permut_de_dyck([1,1,-1,1,-1,-1])` renvoie `[1,2,0]`.

Indication : Utiliser une variable locale représentant la pile de l'algorithme.

4) [1 pt] On considère $\boxed{s = \sigma^{-1}}$ la bijection réciproque de σ. Écrire une fonction `inverse(A)` qui étant donnée une permutation σ codée par A renvoie le code B de la permutation s.

On proposera un algorithme de complexité $O(n)$ qui parcourt *une seule fois* la liste A.

5) [0.5 pt] Avec les notations de c), que représente $s(i)$ lorsque σ est une permutation de pile ?

6) [2.5 pts] (★) Écrire une fonction `dyck_de_permut(A)` qui étant donnée une permutation de pile σ codée par A renvoie le mot de Dyck associé (son image par f).

On demande un algorithme de complexité $O(n)$.

7) [1 pt] En déduire une fonction `test(A)` qui étant donnée une permutation teste son appartenance à $\mathcal{P}$.

On proposera un algorithme de complexité $O(n)$.

Exercice C. Tri Shell

Pour $A = [x_0, ..., x_{n-1}]$ et $p \in \mathbb{N}^*$, on considère la procédure suivante :

```
01.    def tri(A : (int), p :  int) :
02.        n = len(A) ; assert p>0
03.        for r in range(p) :          # r décrit {0,1,...,p-1}
04.            for i in range(r,n,p) :      # i décrit {r, r+p, r+2p, ...}
05.                j = i
06.                while j >= p and A[j-p] > A[j] :
07.                    A[j],A[j-p] = A[j-p],A[j]
08.                    j = j - p
```

1) [1 pt] Donner le type de tri utilisé dans `tri(A,p)` et indiquer les sous-tableaux ainsi triés.

2) [1 pt] Pour tout $0 \leq i < n$, on note $m(i)$ le nombre d'entiers $j \in [\![0, i]\!]$ tels que $x_j > x_i$.
Exprimer en fonction des $m(i)$ **le nombre d'échanges** N effectués ligne 07 lorsqu'on effectue `tri(A,1)`.

Préciser la valeur maximale de N (c'est-à-dire la valeur de N dans le "pire" cas).

3) [2 pts] (★) Déterminer le nombre maximal N' d'échanges effectués durant l'exécution des instructions :

```
tri(A,2) ; tri(A,1)
```

Comparer N et N' lorsque n tend vers $+\infty$. Commenter le résultat obtenu.