Interrogation d'informatique n°3. *Barème sur 24 pts*

Exercice A. Réduction des chemins dans un graphe

On considère un graphe orienté G. On note S l'ensemble des sommets.

Pour deux sommets x et $y \in S$, on note $x \to y$ pour signifier qu'il existe une arête reliant x à y dans G.

Un chemin reliant un sommet s à un sommet t est une liste $L = [x_0, ..., x_{n-1}]$ de sommets telle que

$$s = x_0 \to x_1 \to x_2 \to ... \to x_{n-1} = t$$

Le chemin $\Gamma = [x_0, ..., x_{n-1}]$ est codé en PYTHON par la liste de ses n sommets.

Un chemin extrait de Γ est une sous-liste de L formant **un chemin reliant aussi s à t**, en sélectionnant une partie **des sommets et des arêtes** de Γ.

On dit qu'un chemin est **réduit** s'il ne repasse jamais par le même sommet, c'est-à-dire $i \neq j \Rightarrow x_i \neq x_j$.

Exemple : On peut extraire du chemin $\Gamma = [\mathbf{0}, 1, 2, 3, 2, \mathbf{4}]$ le chemin réduit $\Gamma' = [\mathbf{0}, 1, 2, \mathbf{4}]$.

Soit un chemin Γ reliant s à t dans G.

Dans la suite, on propose un algorithme qui **extrait** de Γ un chemin **réduit** Γ' reliant s à t

1) [1 pt] Montrer par un exemple qu'il n'y a pas nécessairement unicité du chemin réduit extrait.

2) On propose ici un algorithme de complexité linéaire construisant un chemin réduit Γ' extrait de Γ.

L'idée est de considérer pour chaque sommet sa dernière occurrence dans Γ pour déterminer son successeur dans Γ'. Ainsi, pour construire Γ', dès qu'on arrive sur un sommet x, on va choisir comme successeur de x dans Γ' le sommet qui suit la dernière occurrence de x dans Γ (et l'algorithme termine si $x = t$). Chaque sommet x n'est donc visité dans Γ' qu'une seule fois.

a) [2.5 pts] On utilise un dictionnaire V de sorte que pour tout sommet $x \in \Gamma$,

$$V[x] = \max\{j \in [\![0, n-1]\!] \mid L[j] = x\}$$

Remarque : Ainsi, l'ensemble des clés du dictionnaire est l'ensemble des sommets du chemin Γ, et la valeur de tout sommet x est l'indice de la dernière occurrence de x dans L.

Écrire une fonction `dico(L)` qui étant donné un chemin renvoie le dictionnaire V.

Par exemple, `dico([a,b,c,c,b,e,b,d,g,d,f])` renvoie le dictionnaire :

```
{ a :  0 ; b :  4 ; c :  3 ; d :  9 ; e :  6 ; f :  10 ; g :  8 }
```

On demande une fonction de complexité $O(n)$, où n est la longueur de L.

Rappels des instructions élémentaires sur les dictionnaires :

- création d'un dictionnaire vide : `V = {}`

- test booléen d'appartenance d'une clé au dictionnaire : `clé in V`

- ajout et modification d'un élément : `V[clé] = valeur`

b) [3.5 pts] En déduire une fonction `reduction(L,V)` qui étant donnés un chemin Γ codé par L et son dictionnaire V, renvoie un chemin réduit Γ' extrait de Γ.
On demande une complexité $O(p)$, où p est la longueur de Γ'.

Par exemple, `reduction([a,b,c,c,b,e,b,d,g,d,f],V)` renvoie `[a,b,d,f]`

L'algorithme utilisé peut se schématiser sur cet exemple par : `a` $\rightarrow$ `b` ... `b` $\rightarrow$ `d` ... `d` $\rightarrow$ `f`

3) [1 pt] L'algorithme de Dijkstra permet de déterminer dans un graphe un chemin de longueur minimale reliant deux sommets donnés.

Expliquer brièvement comment cet algorithme permet d'obtenir un chemin réduit Γ' extrait de Γ.

Remarque : Il s'agit ici d'expliciter le graphe auquel on applique l'algorithme de Dijkstra.

Exercice B. Recherche du médian et problème de sélection

Soit $L = [x_0, ..., x_{n-1}]$ un tableau de n réels deux à deux distincts (codé en PYTHON par une liste de flottants).

Pour $0 \leq g < d \leq n$ (avec g pour gauche et d pour droit), $L[g : d]$ désigne le sous-tableau $[x_g, ..., x_{d-1}]$.

On dit qu'un élément x est de rang p dans L ssi il existe dans L exactement p éléments strictement inférieurs à x.

L'élément de rang 0 est ainsi le minimum de L. L'élément de rang $n-1$ est le maximum de L.

Le problème de sélection consiste à trouver l'élément de rang p dans L, c'est-à-dire l'élément d'indice p lorsque L est trié par ordre croissant (avec $0 \leq p < n$).

La recherche du médian est un cas particulier du problème de sélection : il correspond au cas $p = n//2$.

On se propose de trouver une méthode de complexité optimale déterminant l'élément de rang p.

1) [1 pt] On se donne $0 \leq p < n$.
Expliquer brièvement comment on peut obtenir l'élément de rang p en $O(n \ln n)$ opérations.

2) [2.5 pts] Écrire une procédure-fonction `TriPartiel(p,L)` qui renvoie l'élément de rang p du tableau L, en procédant par des échanges sur les éléments de L de sorte à placer dans les $(p+1)$ premières cases de L les $(p+1)$ plus petits éléments, classés par ordre croissant.

Par exemple, `TriPartiel(1,[7,1,4,3,6])` renvoie `3`

On proposera une fonction de complexité $O(np)$.
Dans la suite on utilisera cette fonction uniquement sur des tableaux de petite taille.

3) [1 pt] Écrire une fonction `moyenne(g,d,L)` qui étant donné un tableau L de longueur n et deux entiers g et d (gauche et droite) vérifiant $0 \leq g < d \leq n$, renvoie la valeur moyenne des entiers du sous-tableau $L[g : d]$.

Par exemple, `moyenne(1,5,[1,2,3,4,10])` renvoie $\frac{1}{4}(2+3+4+10) = 4.75$

4) On se donne un réel v, appelé pivot.

On veut définir une procédure-fonction `partition(v,g,d,L)` qui étant donné un tableau L de longueur n et deux entiers g et d vérifiant $0 \leq g < d \leq n$**,**

- **modifie** l'ordre des éléments du sous-tableau $L[g : d]$ de sorte que tous ceux qui ont une valeur inférieure ou égale au pivot v soient situés avant ceux qui ont une valeur strictement supérieurs à v.

- **renvoie** l'entier k tel que $\begin{cases} \text{pour tout } i \text{ vérifiant } g < i < k, \text{ on a } L[i] \leq v \\ \text{pour tout } j \text{ vérifiant } k \leq j < d, \text{ on a } L[j] > v \end{cases}$

On propose un algorithme effectuant $O(d - g)$ opérations :

- on utilise deux curseurs i et j, initialisés par $i = g$ et $j = d - 1$

- à chaque étape jusqu'au test d'arrêt (*à déterminer*), on effectue les opérations suivantes :

    - si $L[i] \leq v$, on effectue $i = i + 1$

    - sinon, et si $L[j] > v$, on effectue $j = j - 1$

    - sinon, on échange $L[i]$ et $L[j]$.

a) [2.5 pts] Écrire la procédure-fonction `partition(v,g,d,L).`

b) [1.5 pt] Justifier à l'aide d'invariants de boucles (c'est-à-dire de propriétés vraies à chaque étape, portant ici sur les valeurs des éléments d'indice $< i$ et celles des éléments d'indice $> j$) que l'entier k renvoyé par l'algorithme vérifie bien les propriétés demandées**.**

5) [3 pts] En déduire une procédure **récursive** `selection(p,g,d,L)` qui étant donnés trois entiers g, d et p renvoie **l'élément de rang** p **dans le sous-tableau** $L[g : d]$ en procédant comme suit :

Si le sous-tableau contient au moins deux éléments, on effectue une partition **en choisissant comme pivot** v **la valeur moyenne** de ce sous-tableau : cette partition permet de se ramener à la recherche de l'élément dans un sous-tableau **strict** du tableau $L[g : d]$.

Par exemple, `selection(0,g,d,L)` renvoie le plus petit élément de $L[g : d]$.

Remarque : On suppose en particulier que $0 \leq p < d - g$.

Attention : Ne pas oublier le test d'arrêt dans la fonction récursive.

6) Ainsi, si L est de longueur n, `selection(p,0,n,L)` renvoie l'élément de rang p de L.

On suppose ici que $n = 2^N$ et que les pivots ont des valeurs "idéales" de sorte que chaque partition utilisée dans l'algorithme conduit à deux sous-tableaux de même longueur.

[1.5 pt] Montrer brièvement que la complexité de la recherche de l'élément de rang p est alors en $O(n)$.

Dans la suite, on s'intéresse à optimiser le coût dans les pires cas.

En effet, dans la situation générale, il se peut que les deux sous-tableaux associés aux partitions soient de longueurs très inégales, ce qui peut conduire à une complexité en $O(n^2)$ dans le pire des cas.

L'idée est de procéder à un prétraitement afin d'optimiser le choix du pivot (pour éviter les pires cas).

On considère la division euclidienne de n par 5, c'est-à-dire $n = 5q + r$, avec $r \in \{0, 1, 2, 3, 4\}$.

On regroupe les éléments du tableau **par paquets de 5 éléments** (sauf éventuellement pour le dernier groupe qui contient entre 1 et 5 éléments).

Pour tout $0 \leq k < q$, on note y_k le médian de l'ensemble $\{x_{5k}, ..., x_{5k+4}\}$ de cardinal 5.
On rappelle que le médian correspond à la valeur médiane.
On ne s'occupe pas du dernier groupe s'il contient moins de 5 éléments.

Le tableau $M = [y_0, ..., y_{q-1}]$ est appelé **tableau des médians locaux** de L, avec $q = \left\lfloor \frac{n}{5} \right\rfloor$.

Par exemple, si $L = [2, 1, 5, 4, 3, 7, 8, 9, 10, 6, 12, 11]$, les paquets sont $[2, 1, 5, 4, 3]$ et $[7, 8, 9, 10, 6]$, donc le tableau M des médians locaux est $[3, 8]$.

Pour déterminer l'élément d'ordre p dans L, on considère l'algorithme récursif suivant :
- on détermine en temps $O(n)$ le tableau $M = [y_0, y_1, ..., y_{q-1}]$ des médians, avec $q = \left\lfloor \frac{n}{5} \right\rfloor$
- lorsque $n \geq 5$, c'est-à-dire $q \geq 1$, on calcule (par appel récursif) le médian v de M
- on utilise ce médian v pour appliquer l'algorithme de partition précédemment utilisé ; on se ramène ainsi à un problème de sélection dans un sous-tableau de longueur n' qu'on résout par un appel récursif.

7) [1.5 pt] Écrire une fonction `medians(g,d,L)` qui étant donnés un tableau L de longueur n et deux entiers $0 \leq g < d \leq n$ renvoie **le tableau M des médians locaux du sous-tableau `L[g:d]`**.

On utilisera des expressions de la forme `triPartiel(2,L[i:i+5])`.

8) [1.5 pt] Écrire une nouvelle fonction **récursive** `selection2(p,g,d,L)` qui étant donnés une liste L de longueur n, des entiers $0 \leq g < d \leq n$ et $0 \leq p < d - g$, renvoie l'élément de rang p du sous-tableau `L[g:d]` en utilisant l'algorithme décrit ci-dessus. Le médian v du tableau M des médians locaux s'obtient par un appel **récursif**.

Remarque : Le test d'arrêt porte sur les cas où $d - g < 5$.

9) *Question supplémentaire*

a) (★) On reprend la notation n' utilisé dans l'énoncé avant 7). Justifier que $n' \leq n - 3\left\lfloor \frac{q}{2} \right\rfloor$, où $q = \left\lfloor \frac{n}{5} \right\rfloor$.

b) On note $c(n)$ le nombre d'opérations nécessaires pour le problème de sélection dans un tableau de longueur n (pour une valeur arbitraire de p, avec $0 \leq p < n$). On *admet* que $n \longmapsto c(n)$ est croissante.

Justifier brièvement qu'il existe une constante K telle que $c(n)$ vérifie l'inégalité

$$\forall n \geq 5,\ \ c(n) \leq K\ n + c\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + c\left(n - 3\left\lfloor \frac{q}{2} \right\rfloor\right) \quad , \text{ où } q = \left\lfloor \frac{n}{5} \right\rfloor$$

c) (★★) En déduire l'existence d'une constante E telle que $\forall n \in \mathbb{N}^*$, $c(n) \leq E\ n$.

Remarque : La complexité obtenue est linéaire grâce au fait que $\frac{1}{5} + (1 - \frac{3}{2}\frac{1}{5}) = \frac{1}{5} + \frac{7}{10} = \frac{9}{10} < 1$.

Mais elle n'aurait pas été linéaire si on avait regroupé par 3, car $\frac{1}{3} + (1 - \frac{1}{2}\frac{1}{3}) = \frac{1}{3} + \frac{2}{3} = 1$.