DEVOIR D'INFORMATIQUE N°2 (24/02/2017)

Sandwich au jambon

Le problème de *Stone-Tukey*, aussi appelé théorème du *sandwich au jambon*[1], s'énonce (de façon très simplifiée) de la manière suivante : un ensemble de n points en dimension d peut toujours être séparé en deux parties de cardinal au plus $\left\lfloor \frac{n}{2} \right\rfloor$ par un hyperplan de dimension $d-1$ (certains points peuvent être dans l'hyperplan), où $\left\lfloor \frac{n}{2} \right\rfloor$ désigne la partie entière de $\frac{n}{2}$. De manière concrète, un ensemble de points dans l'espace peut être séparé en deux parties quasi-égales par un plan. De même un ensemble de points dans le plan peut être séparé en deux par une droite et même en 4 à l'aide de deux droites. Ce sujet porte sur la résolution algorithmique de ce problème et de problèmes connexes selon différentes méthodes.

Tout au long du sujet, il est possible d'utiliser les fonctions ou procédures demandées dans les questions précédentes du sujet, même si ces questions n'ont pas été traitées.

Nous attacherons la plus grande importance à la lisibilité du code produit par les candidats ; aussi, nous encourageons les candidats à introduire des procédures ou fonctions intermédiaires lorsque cela en simplifie l'écriture.

Complexité. La complexité, ou le temps d'exécution, d'un programme P (fonction ou procédure) est le nombre d'opérations élémentaires (addition, soustraction, multiplication, division, affectation, etc...) nécessaires à l'exécution de P. Lorsque cette complexité dépend d'un paramètre n, on dira que P a une complexité en $O(f(n))$ s'il existe $K > 0$ tel que la complexité de P est inférieure à $K\,f(n)$ pour tout n.

De manière générale, on s'attachera à garantir une complexité aussi petite que possible dans les fonctions écrites. Le candidat y sera tout particulièrement sensible lorsque la complexité d'une fonction est demandée. Dans ce cas, il devra de plus justifier cette dernière si elle ne se déduit pas directement de la lecture du code.

Listes en Python. On rappelle qu'en Python, les listes sont des tableaux dynamiques à une dimension. Sur les listes, on dispose des opérations suivantes, qui sont de complexité constante :

- [ ] crée une liste vide c'est-à-dire ne contenant aucun élément.
- si n est un entier, **[x]*n** crée une liste de longueur n où chaque élément est égal à la valeur de **x**.
- **len**(**liste**) renvoie la longueur de la liste **liste**.
- **liste**[i] renvoie l'élément d'indice i de la liste **liste** s'il existe, ou produit une erreur sinon (noter que les éléments sont indicés à partir de 0).
- **liste**[$-i$] renvoie l'élément d'indice **len**(**liste**)-i de la liste **liste** s'il existe ou produit une erreur sinon. En particulier, **liste**[-1] renvoie le dernier élément de la liste.
- **liste**[$i : j$] renvoie la liste extraite de **liste** formée des éléments d'indices i *inclus* à l'indice j *non inclus*. En particulier, **liste**[i :**len**(**liste**)], que l'on peut abréger en **liste**[i :], représente tous les éléments de la liste à partir de celui d'indice i, et **liste**[$0 : i$], que l'on peut abréger en **liste**[: i], représente les i premiers éléments de la liste.
- **liste.append**(**x**) ajoute la valeur de **x** à la fin de la liste **liste** qui s'allonge ainsi d'un élément.
- **liste.extend**(**liste2**) ajoute tous les éléments de la liste **liste2** à la fin de la liste **liste**.

1. car il s'exprime de façon imagée en disant qu'on peut couper en quantités égales, d'un seul coup de couteau, le jambon, le fromage et le pain d'un sandwich.

Enfin, l'instruction **x in liste** renvoie le booléen **True** ou **False** selon que **x** appartient à la liste **liste** ou pas. Cette opération est de complexité $O(n)$ où n est la longueur de la liste.

Important : L'usage de toute autre fonction sur les listes telle que **liste.insert(i,x)**, **liste.remove(x)**, **liste.index(x)**, ou encore **liste.sort(x)** est rigoureusement interdit (ces fonctions devront être programmées explicitement si nécessaire).

I. Grand, petit et médian

Dans cette partie, nous supposerons donné un tableau **tab** contenant n nombres réels. Les indices du tableau vont de 0 à $n-1$. Nous dénoterons par **tab[a..b]** le tableau pris entre les indices a et b c'est-à-dire les cellules $tab[a], tab[a+1], \ldots, tab[b-1], tab[b]$. Nous supposerons dans l'énoncé que $a \leqslant b$.

Nous utiliserons le tableau de taille 11 suivant pour nos exemples :

3	2	5	8	1	34	21	6	9	14	8

Question 1

Écrire une fonction **CalculeIndiceMaximum(tab, a, b)** qui renvoie l'indice d'une case où se trouve le plus grand réel de **tab[a. .b]**. Sur le tableau précédent avec $a = 0$ et $b = 10$, la fonction renverra 5 car la case 5 contient la valeur 34.

Question 2

Écrire une fonction **NombrePlusPetit(tab, a, b, val)** qui renvoie le nombre d'éléments dans le tableau **tab[a. .b]** dont la valeur est plus petite ou égale à val. Sur le tableau exemple, pour une valeur de val égale à 5, et $a = 0, b = 10$, la fonction devra renvoyer la valeur 4 car seuls les nombres $3, 2, 5, 1$ sont inférieurs ou égaux à 5.

Nous allons maintenant calculer un médian d'un tableau. Rappelons qu'une valeur médiane m d'un ensemble E de nombres est un élément de E tel que les deux ensembles $E_{<m}$ (les nombres de E strictement plus petits que m) et $E_{>m}$ (les nombres de E strictement plus grands que m) vérifient $\text{card}(E_{<m}) \leqslant \left\lfloor \frac{n}{2} \right\rfloor$ et $\text{card}(E_{>m}) \leqslant \left\lfloor \frac{n}{2} \right\rfloor$. Notez que le problème du médian est une reformulation de problème dit du *sandwich au jambon* pris en dimension 1.

Question 3

Une méthode naïve pour trouver un médian consiste à parcourir les éléments de l'ensemble et à calculer pour chacun d'eux les valeurs de $\text{card}(E_{<m})$ et $\text{card}(E_{>m})$.

Écrire une fonction **MedianNaif(tab, a, b)** qui renvoie un médian du tableau **tab[a. .b]** en utilisant cette méthode.

Quelle est sa complexité dans le meilleur des cas ? et dans le pire des cas ? En supposant que le médian du tableau se situe de façon équiprobable à chaque position, calculez la complexité moyenne.

Question 4

Une méthode plus efficace consiste à trier le tableau par ordre croissant puis prendre la cellule du milieu dans le tableau trié.

a) Écrire une fonction **Fusion(A, B)** qui prend comme arguments deux listes **A** et **B** de nombres réels, que l'on suppose triées par ordre croissant, et qui retourne comme valeur la liste fusionnée de tous les éléments de **A** et **B**, triée par ordre croissant.

b) Écrire une fonction **TriFusion(L)** qui prend en argument une liste **L** de nombres réels et qui retourne comme valeur la liste de tous les éléments de **L** triée par ordre croissant.

Quelle est la complexité de cette procédure (on ne demande pas de justification) ?

c) Écrire une fonction **MedianAvecTri(tab, a, b)** qui renvoie un médian du tableau **tab**[**a. .b**] en utilisant la méthode décrite ci-dessus.

II. Recherche du médian en temps linéaire

Il existe une méthode optimale en temps linéaire $\mathcal{O}(n)$ pour trouver le médian d'un ensemble de n éléments. Cette partie a pour but d'en proposer une implémentation.

Une fonction annexe nécessaire pour cet algorithme consiste à savoir séparer en deux un ensemble de valeurs. Soit un tableau *tab* et un réel appelé pivot $p = tab[i]$, il s'agit de réordonner les éléments du tableau en mettant en premier les éléments strictement plus petits que le pivot $tab_{<p}$, puis les éléments égaux au pivot p, et en dernier les éléments strictement plus grands $tab_{>p}$. Sur le tableau exemple, en prenant comme valeur de pivot 8 on obtiendra le tableau résultat suivant :

3	2	5	1	6	**8**	**8**	21	34	9	14

Notez que dans le résultat les nombres plus petits que le pivot 3, 2, 5, 1, 6 peuvent être dans n'importe quel ordre les uns par rapport aux autres.

Question 5

Ecrire une fonction **Partition(tab, a, b, indicePivot)** qui prend en paramètre un tableau d'entiers $tab[a..b]$ ainsi qu'un entier *indicePivot* dans $[\![a\,;b]\!]$. Soit $p = tab[indicePivot]$. La fonction devra réordonner les éléments de $tab[a..b]$ comme expliqué précédemment en prenant comme pivot le nombre p. La fonction retournera le nouvel indice de la case ou se trouve la valeur p.

Remarque : il existe des algorithmes qui réalisent ce partitionnement « en place », c'est-à-dire sans utiliser de tableau supplémentaire.[2] *Ici, on pourra se contenter de faire le partitionnement en utilisant une liste auxiliaire de même longueur que* **tab**.
Cependant, il est indispensable pour la suite (questions 6 à 8.b) que le tableau **tab** *soit effectivement modifié c'est-à-dire partitionné après l'exécution de la fonction*

Question 6

Remarquons que le $\left\lfloor \frac{n}{2} \right\rfloor$-ème élément dans l'ordre croissant d'un tableau de taille n est un élément médian du tableau considéré[3]. Nous allons donc non pas programmer une méthode pour trouver le médian mais plus généralement pour trouver le k-ème élément d'un ensemble. Nous allons utiliser l'algorithme suivant.

On cherche le k-ème élément du tableau $tab[a..b]$**.**

- Si $a = b$ (donc $k = 1$) alors renvoyer $tab[a]$.

2. il s'agit du célèbre problème du *drapeau tricolore* de Dijkstra.
3. Attention ! En Python, le k-ième élément d'un tableau ordonné est situé à l'indice $k - 1$...

- Sinon, soit $p = tab[a]$. Partitionner le tableau $tab[a..b]$ en utilisant le pivot p en mettant en premier les éléments plus petits que p. Soit i l'indice de p dans le tableau résultant.
  - Si $i-a+1 > k$, chercher le k-ème élément dans $tab[a..i-1]$ et renvoyer cet élément.
  - Si $i-a+1 = k$, renvoyer le pivot.
  - Si $i-a+1 < k$, chercher le $(k-i+a-1)$-ème élément dans $tab[i+1..b]$ et renvoyer cet élément.

Écrire une fonction récursive **ElementK (tab, a, b, k)** qui réalise l'algorithme de sélection du k-ème élément dans le tableau $tab[a..b]$ et renvoie cet élément.

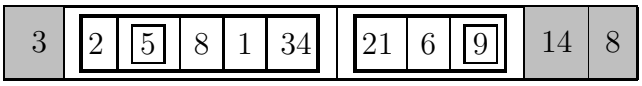
Question 7 :

Supposons que dans l'algorithme précédent nous voulions rechercher le premier élément mais qu'à chaque étape le pivot choisi est le plus grand élément, quel est un ordre de grandeur du nombre d'opérations réalisées par votre fonction ?

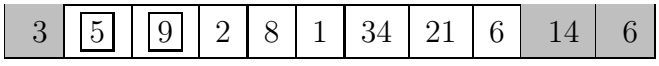
L'algorithme précédent ne semble donc pas améliorer le calcul du médian. Le problème vient du fait que le pivot choisi peut être mauvais c'est-à-dire qu'à chaque étape un seul élément du tableau a été éliminé. En fait, si l'on peut choisir un pivot p dans $tab[a..b]$ tel qu'il y ait au moins $\frac{b-a}{5}$ éléments plus petits et $\frac{b-a}{5}$ plus grands, alors on peut montrer que l'algorithme précédent fonctionne optimalement en temps $\mathcal{O}(n)$.[4] [5]

Pour choisir un tel élément dans $tab[a..b]$, on réalise l'algorithme **ChoixPivot** suivant, où chaque étape sera illustrée en utilisant le tableau donné en introduction en prenant $a = 1$ et $b = 8$.

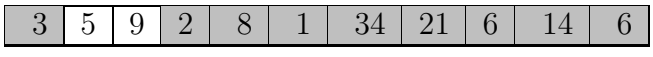
- On découpe le tableau en paquets de 5 éléments plus éventuellement un paquet plus petit. On calcule l'élément médian de chaque paquet (on pourra utiliser la fonction **MedianNaif**).



- S'il n'y a qu'un paquet on renvoie son médian. Sinon on place ces éléments médians au début du tableau.



- On réalise **choixPivot** sur les médians précédents. Dans notre exemple on recommence donc les étapes précédentes en prenant $a = 1$ et $b = 2$.



Question 8

a) Écrire la fonction **ChoixPivot(tab, a, b)** (c'est une fonction récursive) qui réalise l'algorithme précédent et renvoie l'indice du pivot.

b) Modifier la fonction **ElementK** pour utiliser cette fonction.

c) Écrire finalement une fonction **IndiceMedian(tab, a, b)** qui recherche un élément médian du tableau $tab[a..b]$ et renvoie l'indice de cet élément. Contrairement aux précédentes, cette fonction ne doit pas modifier l'ordre des éléments du tableau tab.

4. Voir Aho, Hopcroft, Ullmann, Data Structures and Algorithms .

5. Une autre solution consiste à choisir un pivot aléatoire, voir Cormen, Introduction to Algorithms. C'est évidemment plus facile à programmer, mais moins efficace.

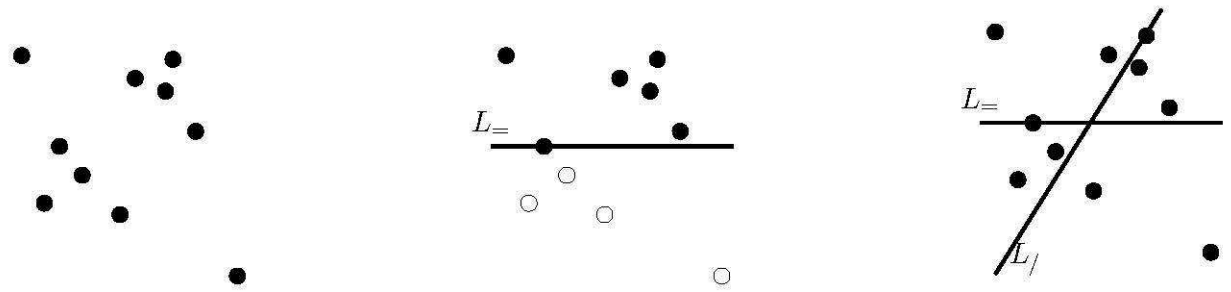
III. De la 1D vers la 2D, des nombres aux points.

Pour un réel $x \geqslant 0$, on note dans cette partie $\lceil x \rceil$ la partie entière supérieure de x, c'est-à-dire le plus petit entier qui est plus grand ou égal à x : $\lceil x \rceil - 1 < x \leqslant \lceil x \rceil$. On supposera disposer d'une fonction **ceil**(x) qui renvoie la partie entière supérieure $\lceil x \rceil$ pour tout réel $x \geqslant 0$.

Dans la partie précédente, nous avons étudié le problème du médian en dimension 1. On supposera donc que vous disposez maintenant de la fonction **IndiceMedian(tab, a, b)** décrite ci-dessus.

Dans cette partie, nous généralisons l'algorithme de manière à trouver deux droites dans le plan séparant un ensemble de n points en 4 parties de cardinal plus petit que $\left\lceil \frac{n}{4} \right\rceil$.

Soit E un ensemble de n points tel qu'il n'existe pas trois points alignés. Nous allons chercher deux lignes $L_=$ et $L_/$ séparant cet ensemble de points en 4 parties comme le montre la troisième figure ci-dessous. En effet, dans cette figure chaque partie est composée d'exactement 2 points, les points situés sur les lignes n'étant pas pris en compte. Nous supposerons donnés dans cette partie deux tableaux **tabX** et **tabY** de taille n contenant les coordonnées des n points. Ainsi le point i a comme abscisse $tabX\ [i]$ et comme ordonnée $tabY\ [i]$.



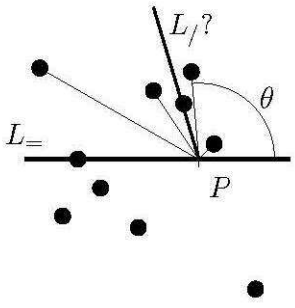
La première étape est de séparer les points en deux ensembles de même cardinal. Il suffit de remarquer que l'on peut toujours effectuer cette séparation par une ligne horizontale notée $L_=$ passant par un point d'ordonnée médiane comme le montre le second schéma ci-dessus.

Question 9

Écrire une fonction **CoupeY(tabX, tabY)** qui renvoie l'ordonnée d'une ligne horizontale séparant les points en deux parties de cardinal au plus $\left\lfloor \frac{n}{2} \right\rfloor$.

La seconde ligne est plus difficile à trouver. Nous allons en réalité trouver le point d'intersection des deux lignes $L_=$ et $L_/$.

Soit P un point sur la droite horizontale $L_=$ précédente, de coordonnées (x, y) . On veut vérifier si ce point peut être le point d'intersection des deux lignes $L_=$ et $L_/$. Nous allons trouver dans un premier temps l'angle entre $L_=$ et $L_/$ utilisant le fait que $L_/$ doit séparer en deux parties de cardinal proche les points au dessus de $L_=$. Ensuite nous allons vérifier si la droite $L_/$ ainsi devinée sépare les points en dessous de $L_=$ en deux parties presque égales.



Concrètement on considère les demi-droites partant de $P = (x, y)$ et joignant les k points de E au dessus strictement de $L_=$ comme indiqué sur le schéma ci-dessus. On calcule ensuite les angles θ entre $L_=$ et ces demi-droites. Remarquez alors que toute demi-droite d'angle médian partage en deux parties de cardinal $\leqslant \left\lfloor \frac{k}{2} \right\rfloor$ les points au dessus de $L_=$.

Nous supposerons donnée une fonction **Angle(x, y, x2, y2)** qui calcule et renvoie l'angle formé par une droite horizontale partant de $(x,\ y)$ et celle joignant (x, y) et $(x2, y2)$. La valeur retournée sera comprise dans l'intervalle $[0\,;\pi[$ (ce sont des angles de droites).

Question 10

Écrire une fonction **DemiDroiteMedianeSup(tabX, tabY, x, y)** qui calcule et renvoie un angle médian entre $L_=$ et les segments reliant $P = (x,\ y)$ avec les points de E dont l'ordonnée est strictement supérieure à y.

Pour un point P donné, nous avons déterminé l'angle que doit prendre $L_/$ pour couper les points au dessus de $L_=$ en 2 parties de taille au plus moitié. Il faut maintenant vérifier que cette droite $L_/$ coupe aussi les points en dessous de $L_=$ en 2 parties quasi-égales. Il suffit de vérifier que l'angle de $L_/$ partitionne les angles formés entre P et les ℓ points en dessous de $L_=$ en deux parties de cardinal $\leqslant \left\lceil \frac{\ell}{2} \right\rceil$.

Question 11

Écrire une fonction **VerifieAngleSecondeDroite(tabX, tabY, x, y, theta)** qui calcule les ℓ angles formés entre $L_=$ et les points strictement au dessous de $L_=$. Votre fonction devra renvoyer :

- 0 si theta est une médiane des ℓ angles ;
- -1 si le nombre d'angles strictement plus petits que theta est $> \left\lceil \frac{\ell}{2} \right\rceil$;
- 1 si le nombre d'angles strictement plus grands que theta est $> \left\lceil \frac{\ell}{2} \right\rceil$.

Si *xmin* est l'abscisse minimale des points de E et *xmax* l'abscisse maximale, alors il est évident que l'intersection entre $L_=$ et $L_/$ a une abscisse entre *xmin* et *xmax*. Nous allons donc rechercher cette abscisse en utilisant l'algorithme suivant :

1. Trouver $L_=$ d'ordonnée y.
2. Soit $\alpha = xmin$ et $\beta = xmax$.
3. On calcule P le point au milieu de (α, y) et (β, y) .
4. On calcule l'angle possible de la droite $L_/$ par a fonction **DemiDroiteMedianeSup**.
5. Soit d la valeur donnée par la fonction **VerifieAngleSecondeDroite** avec l'angle trouvé précédemment. Si $d = 0$ alors on a trouvé $L_/$. Si $d = -1$ on recommence à partir du point **3** en prenant l'abscisse de P à la place de β. Si $d = 1$ on recommence mais en prenant l'abscisse de P à la place de α.

Question 12

Écrire la fonction **SecondeMediane(tabX, tabY, y)** qui à partir d'un ensemble E de points donnés par leurs coordonnées et de l'ordonnée y de la droite $L_=$ calcule et renvoie l'abscisse x du point d'intersection de $L_=$ et de $L_/$ ainsi que l'angle de $L_/$.