**Informatique**

**Composition Hors Classement**

*Robert Cori, Yves Robert*

30 Novembre 1999 13h30 -15h30

## I- Plus longue coupe équilibrée

Soit $F$ un tableau unidimensionnel d'entiers de taille $n \geq 1$ dont tous les éléments valent 0 ou 1. Une coupe $[i..j[$ de $F$, avec $0 \leq i \leq j \leq n$, est dite équilibrée si elle comporte autant de 0 que de 1:

$$\text{card}\{k \mid F[k] = 0,\ i \leq k < j\} = \text{card}\{k \mid F[k] = 1,\ i \leq k < j\}.$$

La longueur de la coupe $[i..j[$ est son nombre d'éléments $j - i$. *Le but de l'exercice est de déterminer la longueur de la plus longue coupe équilibrée du tableau $F$; il est représentatif d'une large classe de problèmes traitant de parcours de tableaux. Nous allons construire des solutions de plus en plus efficaces.*

On définit la classe :

```
class Tableau {
final static int n = 100; // définition d'une constante
static int[] F = new int[n];



méthodes (fonctions) à écrire ...
...}
```

On supposera que $F$ a été initialisé par une méthode extérieure.

**1.**– Quel serait l'ordre de grandeur $O(n^x)$ du coût de la méthode suivante, (trop naïve pour que nous l'écrivions en Java): examiner toutes les coupes $[i..j[$ du tableau $F$ et prendre le maximum des longueurs de celles qui sont équilibrées?

**2.**–

**a)** Ecrire une fonction `static int equi1(int i)` qui renvoie la longueur de la plus longue coupe équilibrée du tableau $F$ qui commence en $i$. On cherche donc

$$\max_j\{j - i \mid 0 \leq i \leq j \leq n,\ [i..j[ \text{ équilibrée}\}.$$

On veut que le coût de cette fonction `equi1` soit en $O(n)$.

**b)** Faire appel à la fonction précédente `equi1` pour écrire une fonction `static int longueur1()` qui renvoie la longueur de la plus longue coupe équilibrée de $F$. Quel est l'ordre de grandeur $O(n^x)$ du coût de cette fonction ?

**3.**– On cherche maintenant une fonction `static int longueur2()` qui renvoie la longueur de la plus longue coupe équilibrée de $F$, dont le coût soit linéaire en $n$. Pour cela, on introduit le déséquilibre $\text{deseq}(i,j)$ d'une coupe $[i..j[$, c'est la différence entre son nombre de 1 et son nombre de 0:

$$\text{deseq}(i,j) = \text{card}\{k \mid F[k] = 1,\ i \leq k < j\} - \text{card}\{k \mid F[k] = 0,\ i \leq k < j\}.$$

Clairement, $\text{deseq}(i,j)$ est un entier compris entre $-n$ et $n$.

L'idée est de parcourir une fois le tableau $F$ et de calculer, pour chaque indice $i$, $0 \leq i \leq n$, le déséquilibre $\text{deseq}(0,i)$. Plus précisément, on va remplir un tableau auxiliaire `aux` de taille $2n+1$, tel que :

$$aux[d+n] = i \text{ ssi } i \text{ est le plus petit indice tel que } \text{deseq}(0,i) = d.$$

On pourra initialiser à $-1$ tous les éléments de `aux`.

**a)** Ecrire une fonction `static void prepareLongueur2()` qui effectue ce travail (ne pas oublier que le déséquilibre 0 est obtenu pour une coupe vide: ainsi $aux[n] = 0$).

On remarque alors que la coupe $[i..j[$ est équilibrée si et seulement si $\text{deseq}(0,i) = \text{deseq}(0,j)$.

**b)** Modifier la fonction `prepareLongueur2()` pour obtenir la fonction `longueur2()` qui calcule la longueur de la plus longue coupe équilibrée de $F$ avec un coût linéaire en $n$.

## II- Scrutin de listes

*On se propose d'écrire des fonctions de manipulation de listes chaînées, on illustre ceci par le pretexte de listes de candidat(e)s à des élections.*

On représente une liste de candidats à des élections par une liste chaînée. Chaque cellule d'une telle liste représente un candidat ou une candidate et contient trois champs :

- L'identité de la personne, on supposera pour simplifier qu'il s'agit d'un nombre entier.
- Un nombre égal à 0 si c'est un candidat et à 1 s'il s'agit d'une candidate.
- Un pointeur sur la cellule suivante.

Un exemple de telle liste est donnée ci-dessous.

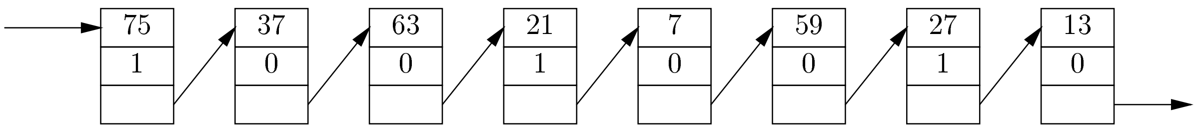



Figure 1: Une liste de candidats

**1.**– Précisez la structure de donnée sous forme d'une classe `ListeCand` en Java. Ecrire une fonction [1]

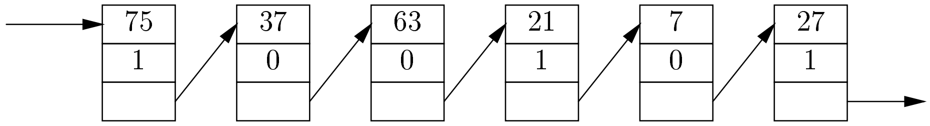
[1]toutes les fonctions sont proposées dans ce texte sous forme de méthodes statiques, vous avez toutefois la possibilité de fournir des réponses sous la forme de méthodes dynamiques si vous le souhaitez

`static boolean estParitaire(ListeCand x)`
qui détermine si une liste `x` contient autant d'hommes que de femmes; on procèdera par comptage du nombres de candidats et de candidates de la liste et par comparaison des deux nombres obtenus.

**2.**– On suppose que l'on a une liste `x` qui contient un plus grand nombre d'hommes que de femmes et on veut la rendre paritaire en supprimant certains hommes. Une première étape est d'écrire une fonction

```
static ListeCand enleveH(int a, ListeCand x)
```

qui rend pour résultat une liste dans laquelle tous les hommes au delà des `a` premiers ont été supprimés dans `x`, les autres candidats et toutes les candidates conservant les mêmes positions respectives. Ainsi sur la liste donnée plus haut l'action de `enleveH(3,x)` doit produire la liste :




Utiliser cette fonction dans une fonction

```
static ListeCand rendParitaire (ListeCand x)
```

qui renvoie une liste paritaire obtenue par suppression dans `x` d'un certain nombre d'hommes situés à la fin de cette liste.

Dans toute la suite on note $n$ le nombre de candidates qui figurent sur une liste, on suppose que deux candidats du même sexe ont des identités différentes et que les identités des candidates sont tous les nombres compris entre 0 et $n-1$.

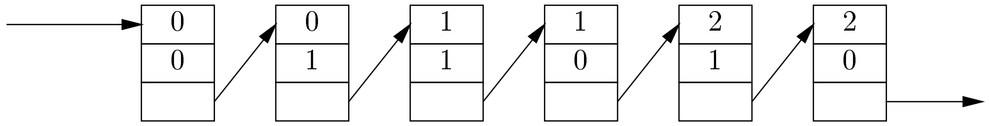



Figure 2: Une liste paritaire

A chaque liste $L$ on associe un tableau de longueur $n$, que l'on appelle ici *tableau de positions*, on le note $h_L$; pour $0 \leq i \leq n-1$, $h_L[i]$ est le nombre d'hommes situés avant la candidate d'identité $i$ dans la liste $L$.
Sur l'exemple ci-dessus le tableau de positions est donné par

| | 0 | 1 | 2 |
|---|---|---|---|
| $h_L$ | 1 | 1 | 2 |

**3.**– Ecrire une fonction

```
static int[] postionH (ListeCand x)
```

qui calcule le tableau de positions $h_L$ associé à la liste `x`. *(Un algorithme en $O(n)$ sera mieux apprécié qu'un algorithme en $O(n^2)$)*

**4.**– Ecrire une fonction

```
static ListeCand ajouterF(ListeCand x, int a, int i)
```

qui étant donnée une liste `x` (contenant $i$ hommes ou plus) ajoute à cette liste une candidate d'identité $a$ dans une position telle qu'elle a exactement $i$ hommes placés avant elle.
Utiliser celle-ci pour écrire une fonction :

```
static ListeCand construire(int[] h)
```

qui étant donné un tableau $h$ construit une liste paritaire dont le tableau de positions est $h$ et dont les candidats hommes apparaissent dans l'ordre des identités croissantes de 0 à $n-1$.

**5.**– Une liste de candidats est dite normalisée si elle est paritaire, si tous les candidats de même sexe ont des identités distinctes comprises entre 0 et $n-1$ et s'ils apparaissent dans l'ordre des identités croissantes (la liste de la Figure 2 est normalisée).
On dit qu'une liste $L$ est compatible avec un tableau de positions $h$ si pour tout $i < n$, on a $h_L[i] \leq h[i]$. On demande d'écrire une fonction

```
static void genereListes(in[] h)
```

qui affiche toutes les listes normalisées de candidats qui sont compatibles avec un tableau `h` de longueur $n$. On pourra supposer que $h[0] \leq h[1] \ldots \leq h[n]$. On affichera une liste par la fonction :

```
static void afficheListe(ListeCand x)
```

que l'on supposera donnée. On ne cherchera pas à optimiser la complexité mais à proposer une solution claire.