Informatique Composition Hors Classement : Corrigé

1. Plus longue coupe équilibrée

Question 1 Tester une coupe $[i..j[$ a un coût proportionnel à sa longueur qui vaut $j-i$. Comme i varie de 0 à $n-1$ et j de i à n, le coût total est en $O(n^3)$.

Question 2

1. Première fonction `equi1` de coût $O(n)$ (au passage, on ne teste pas les coupes de longueur impaire) :

```
static int equi1(int i) {
   int nb0=0, nb1=0;
   int lmax = 0;\\
   for (int k=i+2; k <= n; k+=2) {
      if (F[k-2]==1) nb1++; else nb0++;
      if (F[k-1]==1) nb1++; else nb0++;
      if (nb0 == nb1) lmax = k-i; // [i..k[ équilibrée
      }
   return lmax;
}
```

2. Fonction `longueur1()` de coût $O(n^2)$:

```
static int longueur1() {
   int lmax = 0;
   for (int i = 0; i <= n - 2; i++)
      if (equi1(i) > lmax) lmax = equi1(i);
    return lmax;
}
```

Question 3

1. On initialise `aux` puis on parcourt F pour le remplir :

```
static void prepareLongueur2() {
   int[] aux = new int[2*n+1];
   for (int k = 0; k < 2*n+1; k++)
      aux[k] = -1;
   int d = 0; //invariant: d = nb1 - nb0 dans la coupe [0..i[
   aux[n] = 0; //valeur pour la coupe vide [0..0[
   for (int i=1; i < n; i++) {
      if (F[i-1]==0) d++; else d- -;
      if (aux[d+n] == -1) aux[d+n] = i;
   }
}
```

2. C'est la seule question un peu difficile : en parcourant le tableau F, quand on rencontre pour la coupe $[0..i[$ le déséquilibre d :
   - ou bien d est rencontré pour la première fois (`aux[d+n] = -1`), et on stocke la valeur de i dans `aux[d+n]`
   - ou bien d a déjà été rencontré (`aux[d+n]` $\neq$ `-1`), et on sait que la coupe `F[aux[d+n]..i[` est équilibrée. On met alors à jour le compteur `lmax` :

```
static int longueur2() {
   int[] aux = new int[2*n+1];
   for (int k=0; k<2*n+1; k++)
      aux[k] = -1;
   int lmax = 0;
   int d = 0; //invariant: d = nb1 - nb0 dans F[1..i]
   aux[n] = 0; // déséquilibre 0 pour i=0
   for (int i=1; i<=n; i++) {
      if (F[i-1]==1) d++ ; else  d- -;
      if (aux[d+n] == -1) aux[d+n] = i;
      else // coupe équilibrée [(d+n)..i[
         if (i-aux[d+n] > lmax) lmax = i-aux[d+n];
   }
    return lmax;
}
```

2. Scrutin de listes

1. La structure de liste chaînée est classique :

```
class ListeCand{
  static int compte = 0;
  int  identite;
  int sexe;
  ListeCand suivant;
```

On utilisera souvent le constructeur :

```
  ListeCand(int a, int s, ListeCand x){
     identite = a;
     sexe = s;
     suivant = x;}
```

Pour savoir si une liste est paritaire on compte les hommes et les femmes.

```
static boolean estParitaire (ListeCand x){
      int nh = 0;
      int nf = 0;
      while (x != null){
        if (x.sexe == 0) nh++; else nf++;
        x = x.suivant;}
      return (nh == nf);}
```

2. Voici une fonction qui construit une nouvelle liste en laissant la première intacte, une fonction avec effet de bord aurait tout aussi bien fait l'affaire :

```
 static ListeCand enleveH(int a, ListeCand x){
      if (x == null) return x;
      else if  (x.sexe == 1)
         return new ListeCand(x.identite, x.sexe, enleveH(a, x.suivant));
     else if (a > 0)
         return new ListeCand(x.identite, x.sexe, enleveH(a - 1, x.suivant));
     else return enleveH(a, x.suivant);}
```

Il est utile de connaître le nombre de femmes dans une liste par une fonction comme ;

```
  static int nbF (ListeCand x) {
      int nf = 0;
      while (x != null){
         if (x.sexe == 1) nf++;
         x = x.suivant;}
     return nf;}
```

On obtient alors :

```
  static ListeCand rendParitaire(ListeCand x){
      int nf = nbF(x);
      return enleveH(nf, x);
   }
```

3. La taille de h est déterminée par le nombre de femmes dans `x`. On parcourt la liste de candidats, chaque fois que l'on rencontre un homme on incrémente le nombre d'hommes, chaque fois que l'on rencontre une femme d'identité a on a obtenu $h[a]$:

```
static int[] positionH (ListeCand x) {
      int n = nbF(x);
      int[] h = new int[n];
      int nh = 0;
      while (x != null) {
             if (x.sexe == 0) nh++;
             else h[x.identite] = nh;
```

```
            x = x.suivant;
      }
     return h;}
```

4. Ici on a donné une fonction avec effet de bord :

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

     if (i == 0) x = new ListeCand(a, 1, x);
     else  if(x.sexe == 0) x.suivant = ajouterF(x.suivant, a, i - 1);
           else x.suivant = ajouterF(x.suivant, a, i);
     return x;
   }

 static ListeCand construire (int[] h){
      int n = h.length;
      ListeCand x = null;
      for (int i = 0; i < n; ++i)  x = new ListeCand(n - i - 1, 0, x);
      for (int i = n - 1; i >=  0  ; --i) x = ajouterF(x, i, h[i]);
      return x;}
```

5. A chaque appel récursif on ajoute un homme ou une femme à une liste `x`. Un homme peut être ajouté à chaque itération, une femme ne peut l'être que s'il ne reste pas trop d'hommes à ajouter.

```
static void genere(int[] h, int nf, int nh, ListeCand x){
    if (nf == -1)
        if (nh == -1) affiche(x);
        else genere(h, nf, nh - 1, new ListeCand(nh, 0, x));
    else {if (h[nf] > nh)
              genere(h, nf - 1, nh , new ListeCand(nf, 1, x));
          if (nh >= 0)  genere(h, nf, nh - 1, new ListeCand(nh, 0, x));
 }
  }
```

Cette fonction devra être appelée par :

```
genere(h, h.length - 1, h.length - 1, null);
```