Concours Polytechnique filières PSI, PT : corrigé

Jean-Loup Carré

Informatique commune – 2005

Question 1. Pour chaque candidat présent dans le tableau a, on compte son nombre de voix, et on vérifie s'il obtient plus de $n/2$ voix.

```
def gagnant_tour1(a):
    for g in a:
        v = 0
        for x in a:
            if g == x:
                v += 1
        if v > len(a)//2:
            return g
    return -1
```

Les deux `for` imbriqués nous coûtent un temps en $\mathcal{O}(n^2)$.

Question 2.
```
def gagnant_tour1_bis(a):
    g = a[0]
    v = 1
    for k in range(1, len(a)):
        if a[k] == g:
            v += 1
        else : # On passe au candidat suivant
            g = a[k]
            v = 1
        if v > len(a)/2:
            return g
    return -1
```

On ne parcourt plus qu'une seule fois la liste, d'où une complexité en $\mathcal{O}(n)$.

Question 3. On commence par trier le tableau en temps $\mathcal{O}(n\ln(n))$ puis on trouve l'éventuel gagnant en temps $\mathcal{O}(n)$. Au total, cela nous coûte un temps $\mathcal{O}(n\ln(n)) + \mathcal{O}(n) = \mathcal{O}(n\ln(n))$.

Question 4.
```
def gagnant_tour2(a):
    m = 0  # Nombre de voix du ou des gagnants
    LG = []  # Liste des gagnants
    for g in a:
        v = 0
        for x in a:
            if g == x:
                v += 1
        if v > m:  # Ce candidat bat tous les candidats déjà vus
            m = v
            LG = [g]
        elif v == m and g not in LG: # si on a des ex aequo
            LG.append(g)
    for g in LG:
        print(g)
```

Question 5.
```
def gagnant_tour2_bis(a):
    g = a[0]
    v = 1
    m = 1
```

```
    LG = [g]
    for k in range(1, len(a)):
        if a[k] == g:
            v += 1
        else :
            g = a[k]
            v = 1
        if v > m:
            m = v
            LG = [g]
        elif v == m:
            LG.append(g)
    for g in LG:
        print(g)
```

Question 6. Par le même argument qu'à la **question 3**, il nous faut un temps en $\mathcal{O}(n\ln(n))$.

Question 7. Soit k le nombre d'occurences de x dans le tableau a ; comme x est majoritaire dans a, on a $k > n/2$.
Soit b le tableau a privé de y et z. La longueur de b vaut donc $n-2$. Comme $y \neq z$, en retirant y et z on retire au plus une occurence de a, donc il y a au moins $k-1$ occurences de x dans b. Or comme $k > n/2$ on a $k-1 > (n-2)/2$.

Question 8. Soit k_0 le nombre d'occurrences de x dans $b =< a[0], \ldots, a[i-1] >$ et k_1 le nombre d'occurences de x dans $c =< a[i], \ldots, a[n-1] >$.
Comme il n'y a pas d'élément majoritaire dans b, $k_0 < i/2$, comme x est majoritaire dans a, $k_0 + k_1 > n/2$ d'où $k_1 > n/2 - k_0 > n/2 - i/2 = (n-i)/2$. Donc x est majoritaire dans c (qui a $n-i$ éléments).

Question 9. On commence par écrire une fonction `gagnant_potentiel` qui renvoie la gagnant s'il existe, et qui renvoie sinon n'importe quel candidats.

```
def gagnant_potentiel(a):
    g = a[0]
    v = 1
    for k in range(1, len(a)):
        if v == 0:
            g = a[k]
        elif a[k] == g:
            v += 1
        else :
            v -=1
    return g
```

Cette fonction maintient l'invariant suivant : « En supprimant des paires (y, z) d'éléments différents dans $<a[0]\ldots a[k-1]>$, on peut obtenir un tableau ne contenant que v occurences de g ».
Ainsi, à la fin de la fonction, on peut en supprimant des paires d'éléments différents du tableau a, obtenir le tableau $f =< \underbrace{g, \ldots, g}_{v \text{ fois}} >$. Or, d'après la **question 7**, si a a un gagnant, alors f a le même gagnant.
On en déduit que si a a un gagnant, alors c'est forcément g.
Il nous reste à écrire une fonction qui vérifie si le gagnant potentiel trouvé par notre première fonction a vraiment gagné.

```
def gagnant_tour1_ter(a):
    g = gagnant_potentiel(a)
    v = 0
    for x in a:
        if x == g:
            v +=1
    if v > len(a)/2:
        return g
    else:
        return -1
```