Buller, voter et multiplier

INFORMATIQUE COMMUNE - Devoir n° 3 - Olivier Reynet

Consignes :

1. utiliser une copie différente pour chaque partie du sujet,
2. écrire son nom sur chaque copie,
3. écrire de manière lisible et intelligible,
4. préparer une réponse au brouillon avant de la reporter sur la feuille.

Les trois parties de ce contrôle sont indépendantes.

A Buller

Le tri bulle est un tri comparatif. Son principe est décrit par l'algorithme 1.

Algorithme 1 Tri bulle

```
1 : Fonction FAIRE_MONTER(t, dernier)
2 :     pour j de 0 à dernier-1 répéter
3 :         si t[j] > t[j+1] alors
4 :             ÉCHANGER(t[j], t[j+1])
5 : Fonction TRI_BULLE(t)
6 :     n ← taille(t)
7 :     pour i de n-1 à 1 répéter
8 :         FAIRE_MONTER(t, i)
```

A1. Sur le tableau `[9,5,0,3,1,2,6,4]`, appliquer à la main l'algorithme du tri bulle en détaillant le résultat de chaque itération de boucle de la fonction TRI_BULLE sur le tableau.

A2. Écrire une fonction de prototype `echanger(t,i,j)` qui implémente la fonction ÉCHANGER, c'est-à-dire qui échange l'élément d'indice `i` avec celui d'indice `j` dans une liste `t`.

A3. Écrire une fonction de signature `faire_monter(t,dernier)` qui implémente la fonction homonyme de l'algorithme 1. Celle-ci place l'élément le plus grand du début du tableau à la dernière place possible, c'est-à-dire à l'indice `dernier`.

A4. Écrire une fonction de signature `tri_bulle(t)` qui implémente le tri bulle d'un tableau d'entiers.

R S'il n'y a pas d'échanges lors d'un parcours des éléments qui restent à trier, on peut en conclure que tout le tableau est trié. L'algorithme 2 présente cette optimisation de l'algorithme précédent.

A5. Quelle est la complexité temporelle de la fonction OPT_TRI_BULLE ? On prendra soin de bien préciser de quel(s) paramètre(s) dépend la complexité, de préciser s'il y a un pire ou un meilleur cas et de calculer précisément la complexité.

A6. Écrire une fonction de signature `opt_tri_bulle(t)` qui implémente le tri bulle optimisé.

A7. Citer un tri dont la complexité est en $O(n \log n)$ dans tous les cas. Expliquer son principe et illustrer, sur la liste `[9,5,0,3,1,2,6,4]`, le fonctionnement de cet algorithme dans le détail.

Algorithme 2 Tri bulle optimisé

```
1 : Fonction OPT_TRI_BULLE(t)
2 :     n ← taille(t)
3 :     trié ← faux
4 :     i ← n − 1
5 :     tant que t non trié et i > 0 répéter
6 :         trié ← vrai
7 :         pour j de 0 à i-1 répéter
8 :             si t[j] > t[j+1] alors
9 :                 échange(t[j], t[j+1])          ▷ faire monter la bulle
10 :                trié ← faux                    ▷ On n'a donc pas fini de trier...
11 :        i ← i − 1
```

B Voter

La méthode Condorcet [1] est un mode de scrutin, c'est-à-dire une méthode de vote pour faire un choix. Lors d'un tel scrutin, **les électeurs classent tous les candidats dans l'ordre de leur préférence.**

Condorcet a énoncé le principe suivant : «si un choix est préféré à tout autre par une majorité ou une autre, alors ce choix doit être élu.» **Le vainqueur du scrutin, s'il existe, est le candidat qui bat tous les autres en duel.** Un tel candidat est appelé **vainqueur de Condorcet**.

■ **Exemple 1 — Scrutin de Condorcet.** Soit trois candidats A,B, et C à une élection. On note $\succ$ la relation «est préféré à». Ainsi $A \succ B$ signifie que A est préféré à B. Le bulletin type d'un électeur pour ce scrutin est donc : $A \succ C \succ B$.

Lors du dépouillement, on a compté :

- 23 votants : $A \succ C \succ B$;
- 19 votants : $B \succ C \succ A$;
- 16 votants : $C \succ B \succ A$;
- 2 votants : $C \succ A \succ B$.

Lorsqu'on compare deux à deux tous les choix, on obtient :

1. Marie Jean Antoine Nicolas de Caritat, marquis de Condorcet, dit Condorcet (1743-1794) était un scientifique brillant, mathématicien, philosophe engagé, homme politique et éditeur français, représentant des Lumières.

$\succ$	A	B	C
A	0	25	23
B	35	0	19
C	37	41	0

On en déduit que la préférence majoritaire est $C \succ B \succ A$ et que C est le choix préféré à tous les autres.

On peut construire à partir de ce tableau la **matrice des duels** M. Dans cette matrice, chaque terme m_{ij} indique le nombre de votes en faveur de la préférence $i \succ j$. Par ailleurs, A a pour indice 0, B a pour indice 1 et C 2.

$$M = \begin{pmatrix} 0 & 25 & 23 \\ 35 & 0 & 19 \\ 37 & 41 & 0 \end{pmatrix}$$

Dans l'optique d'automatiser le scrutin Condorcet, on représente :

- chaque candidat par un entier distinct, en commençant à zéro. Pour trois candidats, on aura donc les candidats 0,1 et 2.
- un bulletin de vote d'un électeur par une liste contenant les candidats dans l'ordre de préférence de l'électeur,
- un scrutin est une liste de bulletins,
- une matrice de duels est une liste de listes.

B1. Écrire une fonction de signature `matrice_nulle(n)` qui renvoie une matrice de taille (n, n) ne contenant que des zéros sous la forme d'une liste de liste.

■ **Exemple 2 — Comptabiliser un bulletin.** On cherche à effectuer le dépouillement, c'est-à-dire à comptabiliser les votes de chaque électeur.
On suppose qu'il y a trois candidats 0, 1 et 2. Soit `b = [2,0,1]` un bullletin.
Soit `M = [[0, 25, 23], [35, 0, 19], [37, 41, 0]]` la matrice des duels déjà partiellement construite.
Lorsqu'on aura comptabilisé ce bulletin, la matrice `M` vaudra :
`[[0, 26, 23], [35, 0, 19], [38, 42, 0]]`.

B2. Écrire une fonction de signature `depouiller_un_bulletin(bulletin, M)` où `bulletin` est un bulletin d'électeur et `M` la matrice des duels à compléter. Cette fonction complète la matrice des duels d'après le choix de l'électeur.

B3. Écrire une fonction de signature `depouiller(bulletins)` où `bulletins` est un scrutin, c'est-à-dire la liste de tous les bulletins des électeurs. Cette fonction renvoie la matrice des duels `M` complétée de ce scrutin.

■ **Exemple 3 — Duels gagnés.** Lorsqu'on cherche à déterminer le vainqueur du scrutin de Condorcet, on a besoin de savoir le nombre de duels que chaque candidat a remporté. Soit M la matrice des duels. On sait que i gagné son duel avec $j \neq i$ si $m_{ij} > m_{ji}$. Le nombre de duels gagnés par i est donc incrémenté à chaque fois que cette inégalité est vraie.

B4. Écrire une fonction de signature `nb_duels_gagnes(M, i)` où `M` est la matrice des duels et `i` un numéro de candidat. Cette fonction renvoie le nombre de duels gagnés par le candidat `i`.

B5. Dans un scrutin de Condorcet, on a décompté :

- 41 bulletins `[0, 1, 2]`
- 33 bulletins `[1, 2, 0]`

- 22 bulletins `[2, 0, 1]`

À quelle condition un candidat a-t-il gagné ? Y-a-t-il un vainqueur de Condorcet ?

B6. Écrire une fonction de signature `determiner_vainqueur(M)` où `M` est la matrice des duels d'un scrutin de Condorcet. Cette fonction renvoie le vainqueur de Condorcet s'il existe et `None` sinon.

C Multiplier

L'algorithme de Karatsuba permet de multiplier rapidement deux nombres de n chiffres, plus rapidement qu'avec la méthode naïve que l'on utilise à la main. Dans cette partie, on se propose de le programmer.

C1. Soit a et b deux nombres de n chiffres. Combien de multiplications et combien d'additions sont nécessaires pour calculer le produit $a \times b$ avec la méthode naïve (à la main) ?

C2. Quelle est la complexité de cet algorithme ?

C3. Écrire une fonction de signature `chiffres_b10(a : int)—> int` où `a` est un nombre entier positif à n chiffres en base 10. Cette fonction renvoie n et procède de manière itérative avec une boucle `while`.

C4. Prouver que la fonction `chiffres_b10` termine.

C5. En vous appuyant sur la division entière, écrire une fonction de signature `r_chiffres_b10(a : int)—> int`. Cette fonction renvoie le même résultat que la fonction `chiffres_b10` mais procède de manière **récursive**.

C6. Quelle est la complextié de la fonction `r_chiffres_b10` ?

L'idée à la base de l'algorithme 3 de Karatsuba est l'observation suivante. Dans le calcul qui suit, le développement 2 ne nécessite que trois multiplications au lieu de quatre pour le développement 1 :

$$\left(a \times 10^k + b\right)\left(c \times 10^k + d\right) = ac \times 10^{2k} + (ad + bc) \times 10^k + bd \quad (1)$$

$$= ac \times 10^{2k} + (ac + bd - (a-b)(c-d)) \times 10^k + bd \quad (2)$$

On réutilise en effet deux fois ac et bd.

Considérons deux nombres à n chiffres a et b. En prenant $k = \frac{n}{2}$ et en effectuant la division euclidienne avec 10^k sur chaque nombre, la méthode peut être appliquée de manière récursive pour les calculs de ac, bd et $(a-b)(c-d)$ en scindant à nouveau a, b, c et d en deux et ainsi de suite.

Algorithme 3 Karatsuba

```
1 : Fonction KARATSUBA(x, y)
2 :     si x < 10 ou y < 10 alors
3 :         renvoyer x × y                    ▷ Condition d'arrêt : un seul chiffre
4 :     n ← le nombre de chiffres maximum en base 10 de x et y
5 :     k ← n//2
6 :     a, b ← DIVMOD(x, 10^k)
7 :     c, d ← DIVMOD(y, 10^k)
8 :     ac ← KARATSUBA(a, c)
9 :     bd ← KARATSUBA(b, d)
10 :    ad_bc ← KARATSUBA(a + b, c + d) − ac − bd
11 :    renvoyer ac × 10^(2k) + ad_bc × 10^k + bd
```

C7. Écrire une fonction de signature `karatsuba(x, y)` qui implémente l'algorithme 3. L'usage de la fonction `divmod(a,b)` qui renvoie le quotient (`a//b`) et le reste (`a%b`) de la division euclidienne de `a` par `b` est autorisé.

C8. À quelle grande famille d'algorithmes appartient l'algorithme de Karatsuba ? Citer un autre algorithme de cette famille dont vous donnerez la complexité.

C9. Proposer une relation de récurrence qui caractérise la complexité de la fonction `karatsuba`. On précisera quel est le paramètre agissant sur la complexité. L'opération `divmod(a,b)` a une complexité constante par rapport au nombre de chiffres de `a` et de `b`.

C10. On peut montrer que la complexité de `karatsuba` $T(n)$ est telle que : $T(n) \leqslant 3T(n/2)$. En supposant que $n = 2^k$, c'est-à-dire n est une puissance de 2, et en introduisant une suite auxiliaire, montrer que : $T(n) = O(n^{\log_2 3})$. Est-ce mieux que l'algorithme naïf[2] ?

2. $\log_2 3 \approx 1.585$