



Master BioInformatique

ANNÉE : 2013/2014

SEMESTRE DE DÉCEMBRE 2013

PARCOURS : Master 1

UE J1BS7202 : Algorithmique et Programmation

Épreuve : Examen

Date : Jeudi 19 décembre 2013

Heure : 9 heures

Durée : 2 heures

Documents : autorisés

Épreuve de M. Alain GRIFFAULT

Code d'anonymat :

Avertissement

- La plupart des questions sont indépendantes.
- **À chaque question, vous pouvez au choix répondre par un algorithme ou bien par un programme python.**
- Les indentations des fonctions écrites en Python doivent être respectées.
- L'espace laissé pour les réponses est suffisant (sauf si vous utilisez ces feuilles comme brouillon, ce qui est fortement déconseillé).

| Question | Points | Score |
|--------------------------|--------|-------|
| Mise en bouche | 7 | |
| Algorithmes de rang | 14 | |
| Liste doublement chaînée | 9 | |
| Total: | 30 | |

Exercice 1 : Mise en bouche

(7 points)

- (a) (1 point) Deux nombres sont opposés si leur somme est égale à 0. Deux nombres sont inverses si leur produit est égal à 1. Écrire un algorithme `sontInvOuOpp(a,b)` où `a` et `b` sont deux nombres, qui retourne `Vrai` si `a` et `b` sont inverses ou opposés, `Faux` sinon.

- (b) (2 points) Écrire un algorithme `existeInvOuOppConsecutifs(T)` où `T` est un tableau de nombres, qui retourne `Vrai` si `T` contient deux nombres **consécutifs** opposés ou inverses, `Faux` sinon.

- (c) (2 points) Écrire un algorithme `existeInvOuOpp(T)` où T est un tableau de nombres, qui retourne **Vrai** si T contient deux nombres, **ayant des indices différents**, opposés ou inverses, **Faux** sinon.

- (d) (2 points) Écrire un algorithme `nbInvOuOpp(T)` où T est un tableau de nombres, qui retourne le nombre de paires d'indices (i, j) telles que : d'une part $i < j$; d'autre part $T[i]$ et $T[j]$ soient des nombres opposés ou inverses.

Exercice 2 : Algorithmes de rang**(14 points)**

Le problème de la sélection consiste à trouver dans un tableau de nombres l'élément dit de rang i .

Pour cet exercice, du fait que les indices d'un tableau T sont compris entre 0 et $\text{longueur}(T)-1$, nous admettrons que l'élément de rang 0 est le plus petit élément du tableau, et que l'élément de rang $\text{longueur}(T)-1$ est le plus grand.

Exemple : Soit $T = [8, 6, 53, 8, 2, 9, 3, 10]$, alors :

- Les éléments de rang <0 sont indéfinis.
- L'élément de rang 0 est 2.
- L'élément de rang 1 est 3.
- L'élément de rang 2 est 6.
- L'élément de rang 3 est 8.
- L'élément de rang 4 est 8.
- L'élément de rang 5 est 9.
- L'élément de rang 6 est 10.
- L'élément de rang 7 est 53.
- Les éléments de rang >7 sont indéfinis.

Remarque 1 : Une solution simple au problème de la sélection consiste à utiliser un algorithme quelconque de tri, puis de retourner l'élément de rang souhaité.

Algorithme 1: Rang(T ,rang)

Données : Un tableau T de nombres, et rang un entier

Résultat : Si rang est un indice, alors $T[\text{rang}]$ après avoir trié T

si $\text{rang} < 0$ OU $\text{rang} \geq \text{longueur}(T)$ alors

 | retourner nil;

Trier(T);

retourner $T[\text{rang}]$;

Remarque 2 : Il est facile de se persuader qu'il n'est pas utile de trier *tout* le tableau pour avoir une solution au problème de la sélection. Dans cet exercice, nous allons adapter des algorithmes de tri vus en cours afin d'obtenir des algorithmes de rang plus *efficaces* que le précédent.

Dans toute la suite de l'exercice, vous pourrez utiliser la fonction classique **Echange**(T, i, j) qui échange les valeurs du tableau T indicées par i et j .

```
def echange(T, i, j):
    TMP = T[i]
    T[i] = T[j]
    T[j] = TMP
```

Algorithme 2: Echange(T, i, j)

Données : Un tableau T de nombres, et deux indices i et j

Résultat : $T[i]$ et $T[j]$ échangés

aux $\leftarrow T[i]$;

$T[i] \leftarrow T[j]$;

$T[j] \leftarrow \text{aux}$;

(a) Solution adaptée du tri par sélection vu en cours.

```
def triSelection(T):
    for i in range(len(T)):
        iMin = i
        for j in range(i+1, len(T)):
            if T[j] < T[iMin]:
                iMin = j
        if iMin != i:
            echange(T, i, iMin)
```

Algorithme 3: TriSelection(T)

Données : Un tableau T de nombres
Résultat : Le tableau T trié en ordre croissant

```
pour i=0 à longueur(T)-1 faire
    iMin ← i;
    pour j=i+1 à longueur(T)-1 faire
        si T[j] < T [iMin] alors
            iMin ← j;
    si i ≠ iMin alors
        Echange(T,i,iMin);
```

Il semble évident qu’une fois la valeur désirée *bien placée* dans le tableau, il est inutile de continuer le tri.

i. (2 points) Écrire un algorithme `rangSelection(T,r)` fortement inspiré de l’algorithme ou du programme python `triSelection(T)` qui résout le problème de la sélection. Ne pas oublier de s’assurer que le rang désiré correspond à un indice du tableau.

ii. (1 point) Compléter le tableau des complexités en fonction de $n=\text{longueur}(T)$ et du rang r .

Rappel : Les complexités ne dépendent pas de valeurs particulières des paramètres n et r , mais de valeurs particulières contenues dans le tableau.

| | TriSelection(T) | RangSelection(T,r) |
|---------------------------|--------------------|--------------------|
| Temps (meilleur des cas) | $\Omega(n^2)$ | |
| Temps (pire des cas) | $\mathcal{O}(n^2)$ | |
| Espace (meilleur des cas) | $\Omega(1)$ | |
| Espace (pire des cas) | $\mathcal{O}(1)$ | |

(b) Solution adaptée du tri à bulle vu en cours.

```
def triBulle(T):
    for i in range(len(T)-1,0,-1):
        for j in range(i):
            if T[j]>T[j+1]:
                echange(T,j,j+1)
```

Algorithme 4: TriBulle(T)

Données : Un tableau T de nombres
Résultat : Le tableau T trié en ordre croissant

```
pour i=len(T)-1 à 1 décroissant faire
    pour j=0 à i-1 faire
        si T[j] > T[j+1] alors
            Echange(T,j,j+1);
```

Il semble évident qu’une fois la valeur désirée *bien placée* dans le tableau, il est inutile de continuer le tri.

i. (2 points) Écrire un algorithme `rangBulle(T,r)` fortement inspiré de l’algorithme ou du programme python `triBulle(T)` qui résout le problème de la sélection. Ne pas oublier de s’assurer que le rang désiré correspond à un indice du tableau.

ii. (1 point) Compléter le tableau des complexités en fonction de $n=\text{longueur}(T)$ et du rang r .

| | TriBulle(T) | RangBulle(T,r) |
|---------------------------|--------------------|----------------|
| Temps (meilleur des cas) | $\Omega(n^2)$ | |
| Temps (pire des cas) | $\mathcal{O}(n^2)$ | |
| Espace (meilleur des cas) | $\Omega(1)$ | |
| Espace (pire des cas) | $\mathcal{O}(1)$ | |

(c) Solution adaptée du tri rapide vu en cours.

Soit la variante suivante de l’algorithme de partition basée sur l’algorithme du drapeau Hollandais vu en cours.

Cet algorithme *partitionne* le tableau en trois zones : la première contient des valeurs strictement inférieures à la valeur du pivot ; la seconde contient des valeurs égales à la valeur du pivot ; et la troisième des valeurs strictement supérieures à la valeur du pivot.

```
def troisPartitionner(T,g,d):
    pivot = T[g]
    i = g
    j = i
    k = d
    while j <= k:
        if T[j] == pivot:
            j += 1
        elif T[j] < pivot:
            echange(T,i,j)
            i += 1
            j += 1
        else:
            echange(T,j,k)
            k -= 1
    return i,j,k
```

Algorithme 5: TroisPartitionner(T,g,d)

Données : Un tableau T de nombres, et deux indices g et d

Résultat : i,j,k tel que $T[g..i-1] < T[i..k]=pivot < T[k+1..d]$

```
pivot ← T[g];
i ← g;
j ← i;
k ← d;
tant que j ≤ k faire
    si T[j] = pivot alors
        j ← j+1;
    sinon si T[j] < pivot alors
        Echanger(T,i,j);
        j ← j+1;
        i ← i+1;
    sinon
        Echanger(T,j,k);
        k ← k-1;
retourner i,j,k;
```

Rappel : La complexité, vue en cours, de troisPartitionner(T,g,d) est $\Theta(d - g + 1)$.

i. (1 point) Compléter le tableau suivant afin de simuler l’exécution de troisPartitionner.

$$T = [17, 3, 21, 13, 17, 25, 4], g = 0, d = 6$$

| | | Temps → |
|---------------------------|----|---------|
| pivot | 17 | |
| couple d’indices échangés | | |
| i | 0 | |
| j | 0 | |
| k | 6 | |

- ii. (2 points) Cette version *améliorée* du tri rapide tire profit des trois zones, en ne faisant pas d'appel récursif sur la zone intermédiaire, car les valeurs de cette zone sont correctement placées.

```
def triRapideRec(T,g,d):
    if g < d:
        i,j,k = troisPartitionner(T,g,d)
        triRapideRec(T,g,i-1)
        triRapideRec(T,k+1,d)

def triRapide(T):
    triRapideRec(T,0,len(T)-1)
```

Algorithme 6: TriRapideRec(T,g,d)

Données : Un tableau T de nombres, et deux indices g et d

Résultat : Le tableau T[g..d] trié en ordre croissant

si $g < d$ **alors**

```
(i,j,k) ← troisPartitionner(T,g,d);
TriRapideRec(T,g,i-1);
TriRapideRec(T,k+1,d);
```

Algorithme 7: TriRapide(T)

Données : Un tableau T de nombres

Résultat : Le tableau T trié en ordre croissant

```
TriRapideRec(T,0,longueur(T)-1);
```

Écrire des algorithmes `rangRapide(T,r)` et `rangRapideRec(T,g,d,r)` fortement inspirés des algorithmes `triRapide(T)` et `triRapideRec(T,g,d)`, qui résolvent le problème de la sélection. Ne pas oublier de s'assurer que le rang désiré correspond à un indice du tableau.

- iii. (1 point) Compléter le tableau des complexités en fonction de $n = \text{longueur}(T)$ et du rang r .

| | TriRapide(T) | RangRapide(T,r) |
|---|--------------------|-----------------|
| Temps (meilleur des cas) (Toutes les valeurs identiques) | $\Omega(n)$ | |
| Temps (pire des cas) (tableau trié) | $\mathcal{O}(n^2)$ | |
| Espace (meilleur des cas) | $\Omega(1)$ | |
| Espace (pire des cas) | $\mathcal{O}(n)$ | |

- (d) La solution *naturelle* au problème de sélection basé sur le tri rapide est une solution récursive. En examinant le déroulement de votre programme, vous devez vous apercevoir qu'aucun calcul *pertinant* n'est réalisé dès que l'on commence à dépiler les appels récursifs de la pile d'exécution. Dans de tel cas, il est assez facile de transformer une solution récursive en une solution itérative.
- i. (2 points) Écrire un algorithme `rangRapideIteratif(T,r)` obtenu à partir de votre solution à la question précédente.

- ii. (1 point) Compléter le tableau des complexités en fonction de $n = \text{longueur}(T)$ et du rang r .

| | TriRapide(T) | RangRapideIteratif(T,r) |
|---|--------------------|-------------------------|
| Temps (meilleur des cas) (Toutes les valeurs identiques) | $\Omega(n)$ | |
| Temps (pire des cas) (tableau trié) | $\mathcal{O}(n^2)$ | |
| Espace (meilleur des cas) | $\Omega(1)$ | |
| Espace (pire des cas) | $\mathcal{O}(n)$ | |

- (e) (1 point) En pratique, le cas *standard* correspond à un tableau initial non trié, et ayant peu de valeurs répétées. L'algorithme de partitionnement retourne alors souvent trois zones telles que l'intermédiaire est *petite* et à peu près au centre du tableau.

Compléter le tableau en fonction de $n = \text{longueur}(T)$ et du rang r pour ce cas *moyen*.

| | TriRapide(T) | RangRapide(T,r) | RangRapideIteratif(T,r) |
|---|------------------------|-----------------|-------------------------|
| Temps moyen (zone 2 petite et centrée) | $(n \times \log_2(n))$ | | |

Exercice 3 : Liste doublement chaînée**(9 points)**

Définition : Une *liste doublement chaînée* est une structure dynamique L composée de cellules ayant chacune :

- Un champ **info** pour stocker les données de la liste.
- Un pointeur **succ** qui contient l'adresse de la cellule suivante dans la liste si elle existe, la valeur **nil** sinon.
- Un pointeur **pred** qui contient l'adresse de la cellule précédente dans la liste si elle existe, la valeur **nil** sinon.

La liste L est un pointeur ayant pour valeur **nil** si la liste est vide, sinon l'adresse de l'unique cellule dont le pointeur **pred** est égal à **nil**.

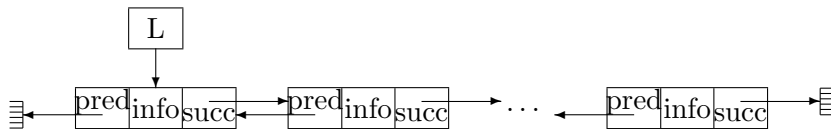


FIGURE 1 – Une liste doublement chaînée

Les listes doublement chaînées sont manipulées par les primitives suivantes :

1. **Derniere_Cellule(L)** qui retourne un pointeur sur la dernière cellule de la liste si elle existe, la valeur **nil** sinon.
 2. **Inserer_Apres(L,X,P)** qui insère dans la liste L la cellule X après la cellule pointée par P.
 3. **Inserer_Tete(L,X)** qui insère en tête de la liste L la cellule X.
 4. **Concatener(L1,L2)** qui retourne le résultat de la concaténation des listes L1 et L2 dans la liste L1.
- (a) (2 points) Écrire l'algorithme de la primitive **Derniere_Cellule(L)**.

(b) (2 points) Écrire l'algorithme de la primitive `Inserer_Apres(L,X,P)`. Vous supposerez que `P` est un pointeur valide sur une cellule de `L`.

(c) (2 points) Écrire l'algorithme de la primitive `Inserer_Tete(L,X)`.

(d) (3 points) Écrire l'algorithme de la primitive `Concatener(L1,L2)`.