

L3-INFO Algorithmique

Quick 1 – 6 octobre 2023 – durée 1 h

Une feuille A4 manuscrite recto-verso autorisée. Les calculatrices et les téléphones (incluant smartphone, tablettes,... tout ce qui contient une interface réseau) sont interdits.

Les dictionnaires pour les personnes de langue étrangère sont autorisés.

Le barème est indicatif.

Vous êtes vivement encouragés à illustrer vos recherches et vos réponses par des schémas.

Question 1 : Écriture d'algorithme et complexité (6 points)

On considère l'algorithme suivant :

CHOIX(T)

Données : Un tableau T de n éléments comparables

Résultat : ???

```
1  $i := 0$ 
2  $j := n - 1$ 
3 while  $i < j$ 
4   if  $T[i] < T[j]$ 
5      $i := i + 1$ 
6   else
7      $j := j - 1$ 
7 return  $i$ 
```

- (2 pts) Décrivez précisément ce que représente le résultat renvoyé par cet algorithme.
- (1 pts) Quel est le coût au pire de cet algorithme, en nombre de comparaisons d'éléments de T ?
Donnez une réponse précise, pas seulement un ordre de grandeur.
- (1 pts) Quel est le coût au mieux de cet algorithme, en nombre de comparaisons d'éléments de T ?
Donnez une réponse précise, pas seulement un ordre de grandeur.
- (2 pts) On suppose maintenant que le tableau T est trié au préalable en ordre croissant : proposez un algorithme plus efficace que le précédent et évaluez sa complexité.

Réponse 1 :

- Il recherche l'indice du maximum dans T (la première occurrence s'il y en a plusieurs).
En effet, à partir d'une certaine itération, $T[i]$ ou $T[j]$ contient le maximum de T (puisque i et j balayaient tous les indices de 0 à $n - 1$ dans un ordre indéterminé).
À partir de cette itération :

- soit l'autre élément considéré est inférieur, et alors on l'élimine ;
- soit il est égal et alors on conserve le maximum dans $T[i]$ (ce qui assure qu'on finira sur la première occurrence).

2. Dans tous les cas l'écart $j - i$ diminue de 1 à chaque itération.

Comme il est initialement de $n - 1$ et qu'on ne peut pas sortir prématurément de la boucle, on effectue exactement $n - 1$ comparaisons.

3. Le coût au mieux est le même puisqu'on effectue systématiquement $n - 1$ itérations.

4. **return** $n - 1$ est un algorithme en temps constant.

Remarquons que c'est bien $n - 1$ et pas $T[n - 1]$ qui est attendu (l'indice du maximum).

Pour être exact, dans le cas où il y a plusieurs maximums, il faudrait renvoyer le plus à gauche d'entre eux. On écrirait donc plutôt :

```

1  $i := n - 1$ 
2 while  $i \geq 0$  et  $T[i] = T[n - 1]$ 
3    $i := i - 1$ 
4 return  $i + 1$ 

```

Dans le pire des cas (tableau dont le contenu est constant) on retrouve notre coût linéaire ; cependant il est probable que ce cas soit très rare, et que la plupart des tableaux reçus en entrée contiennent un nombre borné de maximums.

Question 2 : Structure de données (14 points)

L'objectif de cet exercice est de **concevoir** et de **réaliser** une structure de données permettant :

- d'insérer un nouvel élément ;
- de consulter et/ou d'extraire l'élément le plus ancien présent dans la structure ;
- de consulter et/ou d'extraire l'élément le plus récent présent dans la structure.

Par convention, on appellera cette structure *Chenille*.

1. Opérations (2 pts)

On identifie 7 opérations nécessaires à la manipulation d'une structure de *Chenille* :

Vide, Insérer, EstVide, Tête, Queue, Étêter, Équeuter.

Précisez le profil de chaque opération (types des arguments, type de retour). Comme vu en cours, on privilégie un point de vue *fonctionnel* (pas d'effet de bord).

2. Préconditions (1 pt)

Quelles opérations ne peuvent pas être utilisées lorsque la *Chenille* est vide ?

3. Axiomes (4 pts)

Recopiez et complétez les 8 égalités suivantes, qui permettent de spécifier le comportement des opérations choisies.

EstVide(...) = ...

EstVide(...) = ...

Tete(Insérer(*e*, *c*)) = ...

Tete(...) = ...

Queue(...) = ...

Étêter(Insérer(*e*, *c*)) = ...

Étêter(...) = ...

Équeuter(...) = ...

4. Simulations (2 pts)

Est-il possible de simuler une pile à l'aide de cette structure *Chenille*, et si oui comment ?

Est-il possible de simuler une file à l'aide de cette structure *Chenille*, et si oui comment ?

5. Implémentation (5 pts)

Écrivez une implémentation du type *Chenille* basée sur un tableau d'éléments, et si besoin un nombre fini de variables.

Pour cette question, vous utiliserez des effets de bord.

Un schéma préliminaire vous sera d'un grand secours et sera apprécié par le correcteur.

Plus vous parviendrez à réaliser d'opérations en temps constant, plus votre solution sera valorisée.

Bonus : Comme notre tableau est de taille limitée, la *Chenille* pourra être « pleine » ; décrivez le nombre maximal d'éléments qu'on peut mémoriser (en fonction de la taille du tableau), et prévoyez une condition précise pour afficher un message d'erreur si on cherche à insérer un élément de trop.

1. **Opérations**
- Vide : $void \rightarrow Chenille$
 - Insérer : $Element \times Chenille \rightarrow Chenille$
 - EstVide : $Chenille \rightarrow bool$
 - Tete : $Chenille \rightarrow Element$
 - Queue : $Chenille \rightarrow Element$
 - Étêter : $Chenille \rightarrow Chenille$
 - Équeuter : $Chenille \rightarrow Chenille$

2. **Préconditions**

Pour les 4 dernières opérations, la *Chenille* est supposée non vide.

3. **Axiomes**

- EstVide(Vide()) = vrai
- EstVide(Insérer(e , p)) = faux
- Tete(Insérer(e , c)) = Tete(c) si c est non vide
- Tete(Insérer(e , Vide())) = e
- Queue(Insérer(e , c)) = e
- Étêter(Insérer(e , c)) = Insérer(e , Étêter(c)) si c est non vide
- Étêter(Insérer(e , Vide())) = Vide()
- Équeuter(Insérer(e , c)) = c

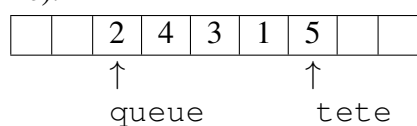
4. Sans les opérations Tête et Étêter, on obtient une pile.

Sans les opérations Queue et Équeuter, on obtient une file.

5. Comme dans le cours, on peut utiliser un tableau "circulaire" avec deux indices pour la tête et la queue de la structure.

Il est commode ici que l'indice `tete` désigne l'élément en tête de la structure, et l'indice `queue` l'élément le plus récent. On suppose les éléments rangés de gauche à droite du plus récent au plus ancien.

Par convention, la structure est vide si `tete = queue - 1` (le cas d'égalité correspond à une chenille de taille 1). Par conséquent avec un tableau de taille n on ne pourra stocker que $n - 1$ éléments (sinon on ne peut pas distinguer la chenille vide de la chenille pleine).



6. Les opérations s'écrivent alors :

— Vide :

```
tete := n-1
queue := 0
```

— Insérer(e , c) :

```
Si (tete + 2) % n == queue :
  // Le modulo est pour le cas où la tete est tout à droite
  // Ici on prend tete + 2 pour éviter d'avoir n éléments
  raise "La Chenille est pleine"
queue := queue - 1
Si queue < 0 :
  queue := n-1 // circulaire : on reprend à partir de la droite
T[queue] := e
```

— EstVide(c) :

```
Return (tete + 1) % n == queue
```

```
— Tête(c):  
    Return (T[tete])  
— Étêter(c):  
    tete := tete - 1  
    Si tete < 0 :  
        tete := n-1  
— Queue(c):  
    Return (T[queue])  
— Équeuter(c):  
    queue := queue + 1  
    Si queue == n :  
        queue := 0
```