DM - Algorithmique - Master 1 Bio-informatique

2023 - 2024

Rev. ~~1~~3 - modifié le 14/11/2023

Exercice 1: Tri rapide

~~Ecrivez un algorithme non~~

1. Écrivez un algorithme récursif, qui prend en paramètre un tableau d'entier non trié et qui trie ce tableau à l'aide du tri rapide.

   Le principe du tri rapide consiste à choisir un élément p du tableau, appelé pivot, puis à trier le tableau en mettant les éléments plus petits que p à gauche de p et les éléments plus grands que p à droites de p. Ensuite, on recommence le processus sur le tableau de gauche d~~'~~'une part (c~~'~~'est à dire les éléments situés à gauche du pivot) et sur le tableau de droite d~~'~~'autre part. L~~'algorithme s'arête~~ 'algorithme s'arrête quand le tableau est complètement trié.
2. Proposez une version non récursive de cet algorithme.

Exercice 2: Tri fusion

~~Ecrivez un algorithme irécursif~~ Écrivez un algorithme récursif qui prend en paramètre un tableau d'entier non trié et qui trie ce tableau à l'aide du tri fusion.

Le tri fusion consiste à couper le tableau en 2 de tailles identiques (à un élément près), à trier le tableau de gauche en utilisant l~~'~~'algorithme de tri fusion, à trier le tableau de droite avec le même algorithme, puis à fusionner les deux tableaux.

Exercice 3: File circulaire

Un tableau circulaire de taille n est un tableau qui contient n cellules dont la cellule d'index $n-1$ est aussi adjacente à la cellule d'index 0.

La figure 1 montre un exemple de tableau circulaire.

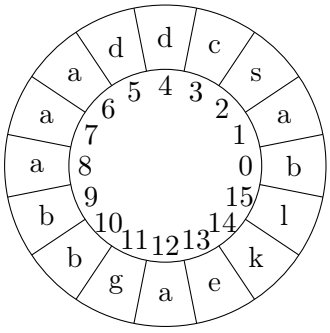

FIGURE 1 – un tableau circulaire

Dans un tableau circulaire, il n'y a pas de dernier élément, car toute cellule a toujours un successeur. En effet, le successeur de la cellule d'index $n-1$ est la cellule d'index 0.

Comme en informatique, il n'existe de pas de mémoire circulaire, les tableaux circulaires sont implémentés à l'aide de tableaux classiques. Ainsi, il n'y a pas de différence, en mémoire, entre un tableau classique et un tableau circulaire.

Par exemple, dans la figure 2, vous trouverez l'implémentation en mémoire du tableau circulaire de la figure 1.

	b	a	s	c	d	d	a	a	a	b	b	g	a	e	k	l	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

FIGURE 2 – l'implémentation en mémoire du tableau de la figure 1

L'objectif de cet exercice est d'implémenter ~~un~~ une file circulaire. Une file circulaire est une file implémentée à l'intérieur d'un tableau circulaire.

~~Un~~ Une file est un tableau contenant au moins une cellule. La première cellule est appelée la ~~queue~~ tête et la dernière cellule est appelé la ~~tête~~queue.

La figure 3 montre 3 exemples de files.

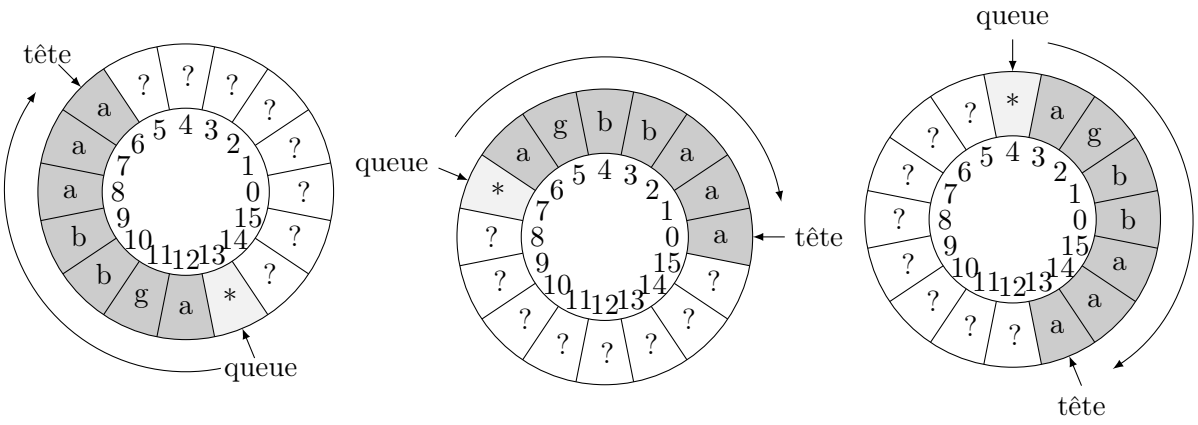

FIGURE 3 – La même file située à différents endroits dans le tableau circulaire

Les éléments d'une file sont les éléments contenus dans les cellules qui vont de la queue (~~inclue~~exclue) à la tête (~~exclue~~inclue).

En fait, la figure 3 montre différents états de la mémoire pour une même file contenant 8 cellules et les 7 même éléments. ~~La différence de position~~ Les différentes positions de la file dépendent de l'historique d'utilisation de la file.

On dit que la file est pleine si la file utilise tout les ~~ces cellules du tableaux~~cellules du tableau. Une file est vide si elle est réduite à sa tête (c'est à dire qu'elle ne contient qu'une cellule et aucun élément).

Par exemple, la figure 4 montre un exemple de file pleine (la file de gauche) et de file vide (la file de droite).

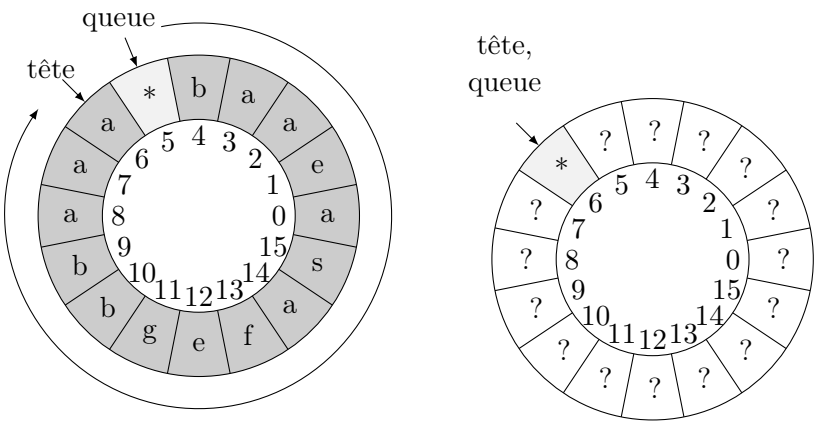

FIGURE 4 – Exemple de file pleine et vide

Dans ~~un file~~une file, deux opérations sont possibles : enfiler un élément et défiler un élément.

— Enfiler une élément :
Si la file n'est pas pleine, c'est à dire si la queue ne suit pas la tête, alors il est possible d'enfiler un élément en décalant la ~~tête~~ queue sur la cellule suivante.
La figure 5 montre l'ajout de l'élément r dans la file.

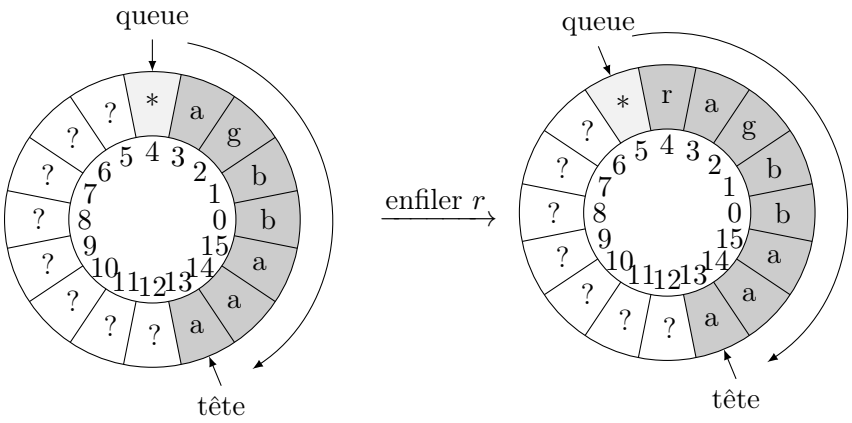

FIGURE 5 – Enfiler un élément r dans une ~~filer~~file

En fait, la ~~tête~~ queue de la file est utilisée comme une mémoire tampon servant à préparer une future donnée à insérer dans la file. C'est à dire que lorsque l'on souhaite ajouter un élément, on commence par copier cet élément dans la ~~tête~~ queue de la file avant de décaler le ~~tête~~queue.

— Défiler un élément :
Si la file n'est pas vide, c'est à dire si la queue et la tête ne représente pas la même cellule, alors il est possible de défiler un élément en décalant la ~~queue~~ tête sur le cellule suivante.
La figure 6 montre la suppression d'un élément de la file.

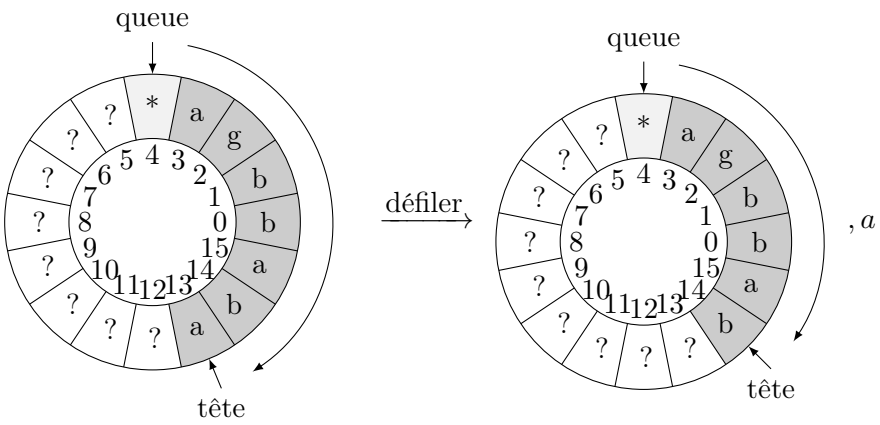

FIGURE 6 – Défiler un élément d'une file

Proposer une structure de ~~donée~~ données et une implémentation de l'API suivante qui permet de manipuler des files circulaires.

```
1  def creer_file(capacite):
2      """
3      Retourne une file vide de capacitée maximale `cpacite`.
4
5      capacite : un entier
6      """
7      return NotImplemented
8
9  def enfiler( f, e ):
10     """
11     Ajoute l'élément `e` dans la file `f`.
12     Si la file est pleine, l'élément n'est pas enfilé dans la file.
13
14     La fonction retourne vrai en cas de succes et faux sinon.
15
16     f : une file
17     e : un element
18     """
19     return NotImplemented
20
21 def defiler( f ):
22     """
23     Retire le dernier élément de la file `f` et le renvoie.
24     Si la file est vide, la fonction renvoie None.
25
26     f : une file
27     e : un element
28     """
29     return NotImplemented
```

```
30
31 def file_est_vide( f ):
32     """
33     Renvoie Vrai si la file est vide.
34
35     f : une file
36     """
37     return NotImplemented
38
39 def file_est_pleine( f ):
40     """
41     Renvoie Vrai si la file est pleine.
42
43     f : une file
44     """
45     return NotImplemented
46
47 def inserer_file( f, e ):
48     """
49     Ajaute l'élément e dans la file.
50
51     Si, au moment d'ajouter e, la file est pleine, cette fonction
52     double sa capacité avant d'effectuer l'opération.
53
54     f : une file
55     e : un element
56     """
57     return NotImplemented
58
59 def redimentionner_file( f, capacite ):
60     """
61     Redimentionne la capacité de f de sorte que la capacité de f
62     devienne égale à 'capacite', si c'est possible.
63
64     En cas de succès, la fonction renvoie Vrai.
65     En cas d'échec (quand la taille de la file est plus grande que
66     la nouvelle capacité), la file n'est pas modifiée et la
67     fonction renvoie Faux.
68
69     f : file
70     capacite : un entier
71     """
72     return NotImplemented
```

Proposez un programme qui teste votre implémentation de file circulaire. Ce programme doit pouvoir tester tous les cas limites, file vide, file pleine, différentes positions de têtes et queues, etc... Justifiez vous choix de tests.

Quels sont les avantages et les inconvénients d'utiliser une file circulaire ?

Exercice 4: Union de deux listes chaînées

On rappelle qu'un ensemble est une collection sans répétition d'éléments. Les ensembles sont généralement codés par des listes triées ou des listes sans répétition d'éléments.

Dans cet exercice, les ensembles seront codés par des listes doublement chaînées en utilisant l'API suivante :

```
def create_list():
    """
    Créer et retourne une liste vide.
    """
    return NotImplemented
%DIF >
def create_cell(value, successeur):
"""
    Créer et retourne une cellule contenant la valeur
    `valeur` et le successeur `successeur`.
"""
    return NotImplemented
%DIF >
def change_cell(cell, valeur, successeur):
"""
    Change les attributs `valeur` et `successeur` de la cellule
    `cell` passée en paramètre.
    """
    raise NotImplementedError()
%DIF >
def push_front(l, e):
    """
    Ajoute l'élément `e` au début de la liste `l`.
    Cette fonction créé une nouvelle cellule et l'insère
    en début de liste.
    """
    return NotImplemented
%DIF >
def remove_front(l):
    """
    Retire le premier élément de la liste si il existe.
    """
    return NotImplemented
%DIF >
def first_cell( l ):
    """
    Retourne la première cellule de la liste `l` et None
    si la liste est vide.
    """
    return NotImplemented
```

```
%DIF >
def next_cell( cell ):
    """
    Retourne le successeur de la cellule `cell`.
    """
    return NotImplemented
%DIF >
def value_cell( cell ):
    """
    retourne la valeur de la cellule `cell`.
    """
    return NotImplemented
%DIF >
def push_back(l, e):
    """
    Ajoute l'élément `e` à la fin de la liste `l`.
    """
    return NotImplemented
%DIF >
def last_cell(l):
    """
    Renvoie la dernière cellule de la liste `l`.
    """
    return NotImplemented
%DIF >
def prev_cell( cell ):
    """
    Retourne la cellule prédécesseur à la cellule `cell`.
    """
    raise NotImplementedError()
```

1. Écrivez un algorithme qui prend en paramètre deux ensembles, codés par des listes doublement chaînées non triées et qui renvoie une nouvelle liste chaînée qui est l'union des deux ensembles.
2. Maintenant, les ensembles sont codés par des listes doublement chaînées triées. Proposez un algorithme plus efficace qui fait l'union des deux ensembles.