Année 2017-2018 — 1 session

ENSEIRB MATMECA BORDEAUX

ATELIER ALGORITHMIQUE ET PROGRAMMATION
PG116
G. EYROLLES, F. HERBRETEAU, D. RENAULT, S. NGUYEN

Filière : Informatique — Année : 1 — Semestre : 6

Date de l'examen : 26/03/2018 — Durée de l'examen : 1h20

Documents autorisés ☐ non autorisés ☒
Calculatrice autorisée ☐ non autorisée ☒

Les exercices peuvent être traités indépendamment les uns des autres.

Une dequeue (double-ended queue) est une structure de donnée séquentielle qui permet l'ajout et le retrait d'éléments en temps constant à chacune de ses extrémités. Cette structure est utilisée notamment pour certains algorithmes d'ordonnancement de tâches, où les tâches plus prioritaires sont insérées en tête de séquence, et les tâches moins prioritaires sont insérées en queue de séquence.

Contrairement à un tableau, les éléments d'une dequeue ne sont pas nécessairement stockés de manière contiguë. Une dequeue peut par exemple être mise en œuvre par une liste chaînée de tableaux de taille fixe d'éléments. L'ajout en tête est plus rapide que dans le cas d'un tableau puisqu'aucun décalage n'est nécessaire. Il suffit en effet d'ajouter un nouveau tableau en tête de la liste de tableaux. L'occupation en mémoire est cependant plus élevée : certains tableaux de la liste ne sont que partiellement occupés.

Le but de ce devoir est de mettre en œuvre une structure dequeue par une liste chaînée de tableaux d'entiers positifs ou nuls, en programmant différentes fonctions agissant sur cette structure.

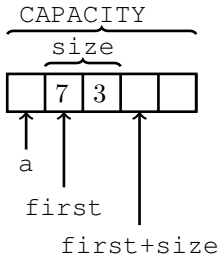
Toutes les définitions de structures et les déclarations de fonctions nécessaires sont rappelées à la fin du document.

1 `ex1.c` : Tableaux avec insertion en tête et en queue (5 points)

Mettre en œuvre les fonctions déclarées dans le fichier `ex1.c` à partir des informations suivantes :

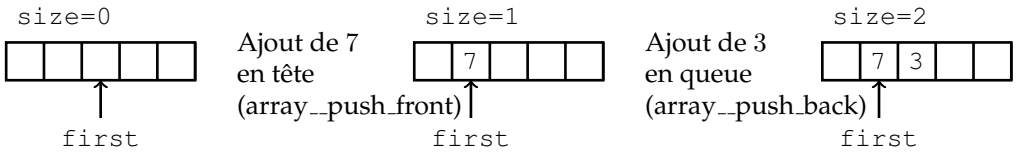
La structure **`struct`** `array` représente des tableaux d'entiers positifs ou nuls. Les éléments du tableau `a` ne sont pas stockés à partir de l'indice 0, mais à partir de l'adresse pointée par `first`. Cela permet d'ajouter des éléments en tête de tableau sans décalage (lorsque `first` est supérieur à `a`). Enfin, `size` donne le nombre d'éléments stockés à partir de la position `first`.

La figure ci-dessous représente une dequeue contenant la séquence `7; 3`. Le pointeur `first` indique le premier élément de la séquence.



L'initialisation d'une **`struct`** `array` place le pointeur `first` à la moitié (`CAPACITY / 2`) du tableau `a`.

L'exemple précédent été obtenue à partir d'une structure **`struct`** `array` initialisée avec les opérations suivantes :



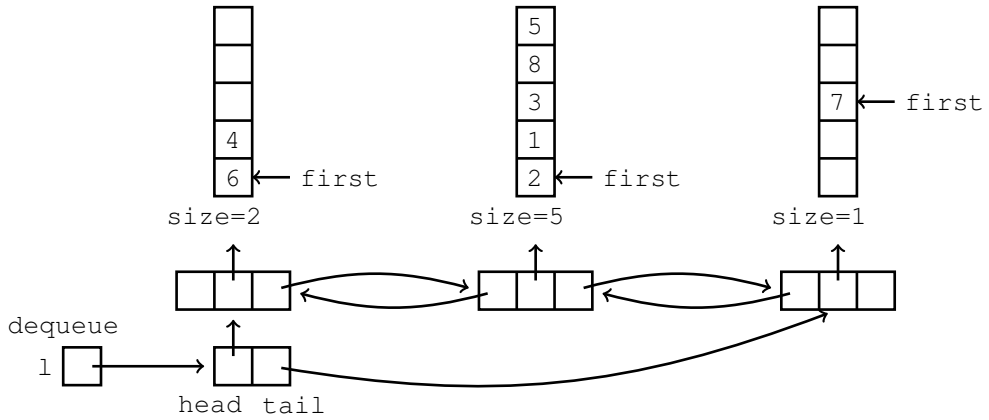
Les messages d'erreurs des tests affichent la valeur des trois champs de la **`struct`** `array`. L'affichage `{{-88, -88, -88, -88, -88}, size = 6789, first = (nil)}`) correspond à une structure **`struct`** `array` avant initialisation.

2 `ex2.c` :Double-ended queue (8 points)

Mettre en œuvre d'une partie des fonctions de dequeue déclarées dans le fichier `ex2.c` à partir des informations suivantes :

La mise en œuvre de dequeue utilise une liste doublement chaînée dont la réalisation vous est fournie (**`struct`** `link`).

La figure ci-dessous représente une dequeue contenant la séquence d'entiers : `6; 4; 2; 1; 3; 8; 5; 7`, répartie dans trois tableaux chaînés.



L'ajout en tête (fonction `dequeue__push_front`) sur cette dequeue ne pourra pas ajouter le nouvel entier au tableau situé en tête de liste puisque `first` pointe en début du tableau. Il faut donc tout d'abord allouer un nouveau tableau, et placer celui-ci en début de liste, avant de lui ajouter le nouvel entier.

L'ajout en queue (fonction `dequeue__push_back`) sur cette dequeue ajoutera le nouvel entier au-dessus du `7`, dans le tableau situé en fin de liste.

On remarque que le scénario précédent gaspille la mémoire puisqu'un nouveau tableau est alloué alors qu'il serait possible de faire de la place dans le tableau actuellement en tête en décalant ses éléments. L'optimisation de la mémoire n'est pas considérée dans ce sujet. On acceptera donc que les tableaux, à part celui du milieu, soient au mieux à moitié remplis.

Les messages d'erreurs des tests utilisent le caractère `>` pour un appel à `dequeue__push_front` et `<` pour un appel à `dequeue__push_back`. L'exemple de la figure ci-dessus est donné par l'affichage suivant `6>4>2>1>dequeue_empty<3<8<5<7`.

Noto bene : **Contraintes à suivre dans votre mise en œuvre**

— *Les mises en œuvre de* **`struct`** `link` *et de* **`struct`** `array` *vous sont fournies.*

— Le retour d'erreur de la fonction `malloc` n'est pas à traiter.

— La liste ne doit pas contenir de tableau vide.

— Vos fonctions doivent éviter d'allouer de la mémoire inutilement, et de parcourir les structures de données plus de fois que nécessaire.
— La manipulation des structures **`struct array`**, **`struct link`** et **`struct lelement`** doit se faire uniquement à travers les fonctions fournies pour ces structures. *Excepté pour le champs `array` de **`struct`** `lelement`, tout accès direct aux champs de ces structures vaudra la note* 0 *à cet exercice.*

Remarques par rapport à la mise en œuvre des listes vue en TD :

— la liste est doublement chaînée : elle permet d'accéder aux cellules précédentes et suivantes d'une cellule donnée en temps constant ;
— **`struct`** `link` offre un accès temps constant à la dernière cellule de la liste. Pour une liste `l`, le pointeur `l->tail` pointe sur la dernière cellule contenant une donnée valide, s'il y en a une (i.e. `llm__is_end_mark(l->tail)` retourne `0`, c'est-à-dire `false`, si la liste n'est pas vide).

3 Paramétrage du code

3.1 `ex3.txt` : Opérations `pop` du dequeue (4 points)

Cet exercice a pour but de factoriser le code commun aux fonctions `dequeue__pop_front` et `dequeue__pop_back`.

```
/**
 * Ajoute x en tete de la dequeue,
 * retourne 0 en cas de succes et -1 en cas d'erreur.
 **/
int dequeue__pop_front(struct dequeue * dequeue)
{
  if (dequeue__is_empty(dequeue))
    return -1;

  if (array__pop_front(lnk__first(dequeue->l)->array) == -1)
    return -1;

  if (array__size(lnk__first(dequeue->l)->array) == 0) {
    struct lelement * llm = lnk__remove_head(dequeue->l);
    free_dequeue_element(llm);
  }

  return 0;
}

/**
 * Ajoute x en queue de la dequeue,
 * retourne 0 en cas de succes et -1 en cas d'erreur.
 **/
int dequeue__pop_back(struct dequeue * dequeue);
```

À partir de l'implémentation ci-dessus de la fonction `dequeue__pop_front`, répondre aux questions contenues dans le fichier `ex3.txt`.

3.2 `ex4.txt` : Paramétrer le type des valeurs stockées (3 points)

On souhaite faire évoluer nos dequeues pour stocker des valeurs dont le type n'est pas fixé à la compilation. On s'intéresse aux modifications à apporter à **`struct`** `array` pour y parvenir.

Répondre aux questions contenues dans le fichier `ex4.txt` en justifiant vos réponses.

```c
//array
#define CAPACITY 5

struct array {
  int a[CAPACITY]; // tableau d'entier
  int * first; // pointeur sur 1er element
  size_t size; // nombre d'elements stockes dans a
};

void array__empty(struct array * array);
size_t array__size(struct array const * array);

int array__push_front(struct array * array, int x);
int array__front(struct array const * array);
int array__pop_front(struct array * array);

int array__push_back(struct array * array, int x);
int array__back(struct array const * array);
int array__pop_back(struct array * array);

//link
struct link {
  struct lelement * head;
  struct lelement * tail;
};

struct lelement {
  struct array * array;
  struct lelement * next;
  struct lelement * prev;
};

struct link lnk__empty(void);
struct lelement * lnk__first(struct link const * l);
struct lelement * lnk__last(struct link const * l);

int llm__is_end_mark(struct lelement const * e);
struct lelement * llm__next(struct lelement const * e);
struct lelement * llm__prev(struct lelement const * e);

void lnk__add_head(struct link * l, struct lelement * e);
struct lelement * lnk__remove_head(struct link * l);
void lnk__add_tail(struct link * l, struct lelement * e);
struct lelement * lnk__remove_tail(struct link * l);
void lnk__remove(struct link * l, struct lelement * e);

//dequeue
struct dequeue {
  struct link * l;
};

struct dequeue * dequeue__empty();
void dequeue__free(struct dequeue * dequeue);

int dequeue__is_empty(struct dequeue const * dequeue);
size_t dequeue__size(struct dequeue const * dequeue);

int dequeue__front(struct dequeue const * dequeue);
int dequeue__push_front(struct dequeue * dequeue, int x);

int dequeue__pop_front(struct dequeue * dequeue)
int dequeue__pop_back(struct dequeue * dequeue)
int dequeue__push_back(struct dequeue * dequeue, int x);
int dequeue__back(struct dequeue const * dequeue)
```