

Files d'attente

Algorithmique et structures de données

Pierre Künzli

17-01-2023

Adapté des cours de Paul Albuquerque et Orestis Malaspinas

La file d'attente

- Structure de données abstraite permettant le stockage d'éléments.
- *FIFO*: First In First Out, ou première entrée première sortie.
- Analogue de la vie "réelle":
 - File à un guichet,
 - Serveur d'impressions,
 - Mémoire tampon, ...

Fonctionnalités

La file d'attente

- Structure de données abstraite permettant le stockage d'éléments.
- *FIFO*: First In First Out, ou première entrée première sortie.
- Analogie de la vie "réelle":
 - File à un guichet,
 - Serveur d'impressions,
 - Mémoire tampon, ...

Fonctionnalités

- Enfiler: ajouter un élément à la fin de la file.
- Défiler: extraire un élément au devant de la file.
- Tester si la file est vide.

La file d'attente

- Structure de données abstraite permettant le stockage d'éléments.
- *FIFO*: First In First Out, ou première entrée première sortie.
- Analogie de la vie "réelle":
 - File à un guichet,
 - Serveur d'impressions,
 - Mémoire tampon, ...

Fonctionnalités

- Enfiler: ajouter un élément à la fin de la file.
- Défiler: extraire un élément au devant de la file.
- Tester si la file est vide.
- Lire l'élément de la fin de la file.
- Lire l'élément du devant de la file.
- Créer une liste vide.
- Détruire une liste vide.

La file d'attente

Implémentation possible

- La structure file, contient un pointeur vers la tête et un vers le début de la file.
- Entre les deux, les éléments sont stockés dans une liste chaînée.

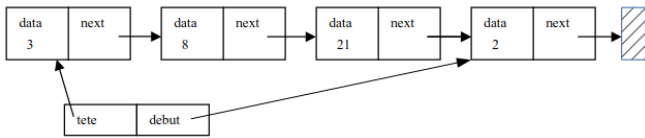


Figure 1: Illustration d'une file d'attente.

Structure de données en C?

La file d'attente

Implémentation possible

- La structure file, contient un pointeur vers la tête et un vers le début de la file.
- Entre les deux, les éléments sont stockés dans une liste chaînée.

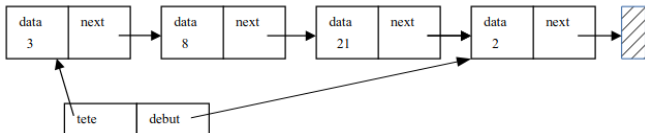


Figure 1: Illustration d'une file d'attente.

Structure de données en C?

```
typedef struct _element { // Élément de liste
    int data;
    struct _element* next;
} element;

typedef struct _queue { // File d'attente:
    element* head; // tête de file d'attente
    element* tail; // queue de file d'attente
} queue;
```

Fonctionnalités d'une file d'attente

Creation et consultations

Fonctionnalités d'une file d'attente

Creation et consultations

```
void queue_init(queue *fa); // head = tail = NULL
bool queue_is_empty(queue fa); // fa.head == fa.tail == NULL
int queue_tail(queue fa); // return fa.tail->data
int queue_head(queue fa); // return fa.head->data
```

Manipulations et destruction

Fonctionnalités d'une file d'attente

Creation et consultations

```
void queue_init(queue *fa); // head = tail = NULL  
bool queue_is_empty(queue fa); // fa.head == fa.tail == NULL  
int queue_tail(queue fa); // return fa.tail->data  
int queue_head(queue fa); // return fa.head->data
```

Manipulations et destruction

```
void queue_enqueue(queue *fa, int val);  
// adds an element before the tail  
int queue_dequeue(queue *fa);  
// removes the head and returns stored value  
void queue_destroy(queue *fa);  
// dequeues everything into oblivion
```

Enfilage

Deux cas différents:

1. La file est vide (dessin):

Enfilage

Deux cas différents:

1. La file est vide (dessin):

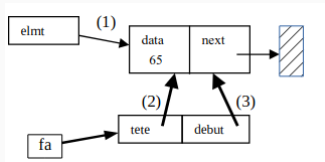


Figure 2: Insertion dans une file d'attente vide.

2. La file n'est pas vide (dessin):

Enfilage

Deux cas différents:

1. La file est vide (dessin):

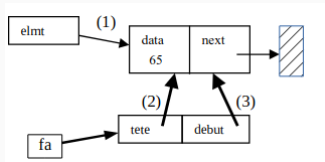


Figure 2: Insertion dans une file d'attente vide.

2. La file n'est pas vide (dessin):

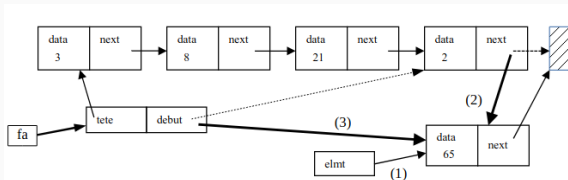


Figure 3: Insertion dans une file d'attente non-vide.

Implémentation

Implémentation

```
void queue_enqueue(queue *fa, int val) {
    element elmt = malloc(sizeof(*elmt));
    elmt->data = val;
    elmt->next = NULL;
    if (queue_is_empty(*fa)) {
        fa->head = elmt;
        fa->tail = elmt;
    } else {
        fa->tail->next = elmt;
        fa->tail = elmt;
    }
}
```

Trois cas différents

1. La file a plus d'un élément (dessin):

Trois cas différents

1. La file a plus d'un élément (dessin):

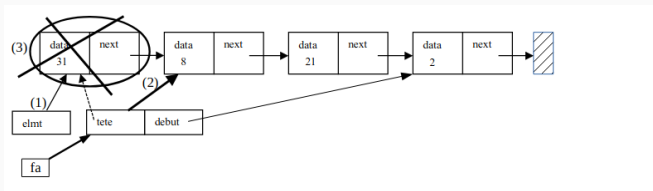


Figure 4: Extraction d'une file d'attente

2. La file un seul élément (faire un dessin):

Trois cas différents

1. La file a plus d'un élément (dessin):

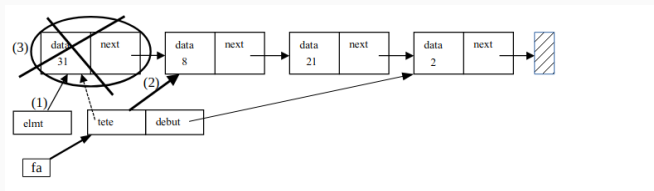


Figure 4: Extraction d'une file d'attente

2. La file un seul élément (faire un dessin):

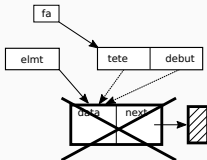


Figure 5: Extraction d'une file d'attente de longueur 1.

Trois cas différents

1. La file a plus d'un élément
2. La file un seul élément
3. La file est vide (problème)

Implémentation

Implémentation

```
int queue_dequeue(queue *fa) {  
    element* elmt = fa->head;  
    int val = elmt->data;  
    fa->head = fa->head->next;  
    free(elmt);  
    if (NULL == fa->head) {  
        fa->tail = NULL;  
    }  
    return val;  
}
```

Implémentation

```
int queue_dequeue(queue *fa) {  
    element* elmt = fa->head;  
    int val = elmt->data;  
    fa->head = fa->head->next;  
    free(elmt);  
    if (NULL == fa->head) {  
        fa->tail = NULL;  
    }  
    return val;  
}
```

Problème avec cette implémentation?

Implémentation

```
int queue_dequeue(queue *fa) {  
    element* elmt = fa->head;  
    int val = elmt->data;  
    fa->head = fa->head->next;  
    free(elmt);  
    if (NULL == fa->head) {  
        fa->tail = NULL;  
    }  
    return val;  
}
```

Problème avec cette implémentation?

Si la file est vide

Comment on faire la désallocation?

Comment on faire la désallocation?

On défile jusqu'à ce que la file soit vide!

Quelle sont les complexité de:

- Enfiler?

Quelle sont les complexité de:

- Enfiler?

$O(1)$

- Défiler?

Quelle sont les complexité de:

- Enfiler?

$O(1)$

- Défiler?

$O(1)$

- Détruire?

Quelle sont les complexité de:

- Enfiler?

$O(1)$

- Défiler?

$O(1)$

- Détruire?

$O(N)$

- Est vide?

Quelle sont les complexité de:

- Enfiler?

$O(1)$

- Défiler?

$O(1)$

- Détruire?

$O(N)$

- Est vide?

$O(1)$

Implémentation alternative

Comment implémenter la file autrement?

Comment implémenter la file autrement?

- Données stockées dans un tableau;
- Tableau de taille connue à la compilation ou pas (réallouable);
- `tail` serait un indice du tableau;
- `capacity` seraient la capacité maximale;
- On *enfile* “au bout” du tableau, au défile au début (indice 0).

Implémentation alternative

Comment implémenter la file autrement?

- Données stockées dans un tableau;
- Tableau de taille connue à la compilation ou pas (réallouable);
- `tail` serait un indice du tableau;
- `capacity` seraient la capacité maximale;
- On *enfile* “au bout” du tableau, au *défile* au début (indice 0).

Structure de données

```
typedef struct _queue {  
    int *data;  
    int tail, capacity;  
} queue;
```


- Initialisation?

File basée sur un tableau

- Initialisation?

```
void queue_init(queue* q){  
    q->capacity = INIT_CAPACITY;  
    q->tail = -1;  
    q->data = malloc(q->capacity*sizeof(int));  
}
```

- Est vide?

File basée sur un tableau

- Initialisation?

```
void queue_init(queue* q){  
    q->capacity = INIT_CAPACITY;  
    q->tail = -1;  
    q->data = malloc(q->capacity*sizeof(int));  
}
```

- Est vide?

```
bool queue_is_empty(queue q){  
    return q.tail == -1;  
}
```

- Enfiler?

- Enfiler?

```
void queue_enqueue(queue* q, int val){  
    if(q->tail < q->capacity-1){  
        q->tail++;  
        q->data[q->tail] = val;  
    }  
}
```

- Défiler?

File basée sur un tableau

- Défiler?

```
int queue_dequeue(queue* q, int* val){
    if(q->tail >= 0){
        *val = q->data[0];
        for(int i=1; i<= q->tail){
            q->data[i-1] = q->data[i];
        }
        q->tail--;
        return 0;
    }
    return 1;
}
```

Quelle sont les complexités de:

- Initialisation?

Quelle sont les complexités de:

- Initialisation?

$O(N)$

- Est vide?

Quelle sont les complexités de:

- Initialisation?

$O(N)$

- Est vide?

$O(1)$

- Enfiler?

Quelle sont les complexités de:

- Initialisation?

$O(N)$

- Est vide?

$O(1)$

- Enfiler?

$O(1)$

- Défiler?

Quelle sont les complexités de:

- Initialisation?

$O(N)$

- Est vide?

$O(1)$

- Enfiler?

$O(1)$

- Défiler?

$O(N)$

Une file plus efficace

Comment faire une file plus efficace?

- Quel est le problème ?

Une file plus efficace

Comment faire une file plus efficace?

- Quel est le problème ?
- Défiler est particulièrement lent $\mathcal{O}(N)$.

Solution?

Une file plus efficace

Comment faire une file plus efficace?

- Quel est le problème ?
- Défiler est particulièrement lent $\mathcal{O}(N)$.

Solution?

- Utiliser un indice séparé pour head.

```
typedef struct _queue {  
    int *data;  
    int head, tail, capacity;  
} queue;
```

Une file plus efficace (implémentation)

Enfilage

```
void queue_enqueue(queue *fa, int val) {
    if ((fa->head == 0 && fa->tail == fa->capacity-1) ||
        (fa->tail == (fa->head-1))) {
        return; // queue is full
    }
    if (fa->head == -1) { // queue was empty
        fa->head = fa->tail = 0;
        fa->data[fa->tail] = val;
    } else if (fa->tail == fa->capacity-1 && fa->head != 0) {
        // the tail reached the end of the array
        fa->tail = 0;
        fa->data[fa->tail] = val;
    } else {
        // nothing particular
        fa->tail += 1;
        fa->data[fa->tail] = val;
    }
}
```


Une file plus efficace (implémentation)

Défilage

```
void queue_dequeue(queue *fa, int *val) {
    if (fa->head == -1) {
        return; // queue is empty
    }
    *val = fa->data[fa->head];
    if (fa->head == fa->tail) { // that was the last element
        fa->head = fa->tail = -1;
    } else if (fa->head == fa->capacity-1) {
        fa->head = 0;
    } else {
        fa->head += 1;
    }
}
```