

Filas

Estruturas de Dados e Algoritmos



Prof. Daniel Saad Nogueira
Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

- 1 Introdução
- 2 Vetores Dinâmicos
- 3 Implementação em vetores
- 4 Filas
- 5 Exemplo



Sumário

1 Introdução



Introdução

Filas

- Filas são um TAD em qual os elementos são mantidos em uma ordem específica. Esta ordem é a ordem FIFO (First-in-First-Out).
- A ordem FIFO se caracteriza pelo fato dos primeiros elementos a fazerem parte da estrutura, também serão os primeiros elementos a deixarem a estrutura.



Operações em Filas

- Algumas das operações suportadas por uma fila devem ser:
 - ▶ Enfileiramento de elementos;
 - ▶ Desenfileiramento de elementos;
 - ▶ Acessar a frente da fila;
 - ▶ Verificar se a fila está vazia;
 - ▶ Verificar o tamanho da fila;



Filas





Representação de filas

- Assim como listas, filas podem ser representadas de várias maneiras, duas delas são por meio de:
 - 1 Vetores;
 - 2 Estruturas auto-referenciadas;



Sumário

2 Vetores Dinâmicos

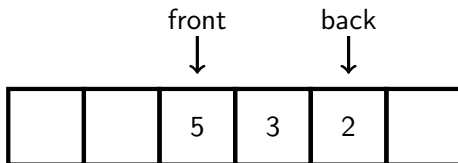


Representação de filas em vetores

- Para representar filas em vetores, usamos uma estrutura composta por um vetor e dois índices, que apontam, respectivamente, para a posição da frente (início) da fila e a posição de trás (fim) da fila.



Representação de Filas em Vetores





Sumário

- 2 Vetores Dinâmicos
 - Enfileiramento
 - Desenfileiramento

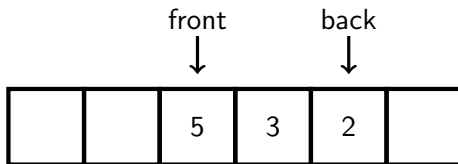


Enfileiramento em vetores

- Para inserir um elemento ao final da fila, incrementamos o índice do fim e colocamos o novo elemento nessa posição.

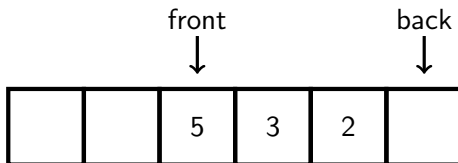


Enfileiramento em vetores



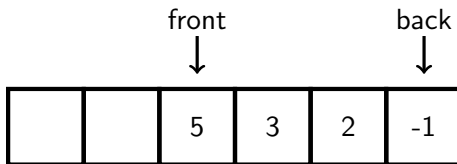


Enfileiramento em vetores





Enfileiramento em vetores



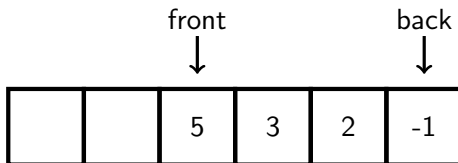


Enfileiramento em vetores

- Quando o índice do fim estiver no final do vetor, podemos simplesmente “dar a volta” pra reutilizar o espaço!

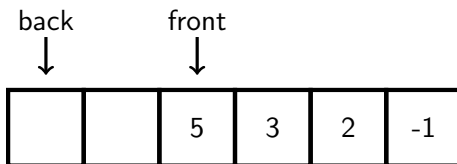


Enfileiramento em vetores





Enfileiramento em vetores





Enfileiramento em vetores

- Caso o vetor venha a ficar totalmente preenchido, duplicamos a sua capacidade.
- Contudo, existem dois casos que devemos observar:
 - 1 Índice do início menor que o do fim.
 - 2 Índice do início menor que o do fim.



Enfileiramento em vetores

Índice do início menor que o do fim

- Só precisamos duplicar a capacidade do vetor.



Enfileiramento em vetores

Índice do início menor que o do fim

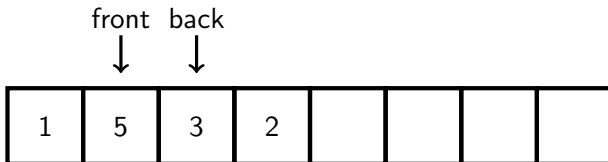
- Só precisamos duplicar a capacidade do vetor.



Enfileiramento em vetores

Índice do início menor que o do fim

- Só precisamos duplicar a capacidade do vetor.





Enfileiramento em vetores

Índice do início maior que o do fim

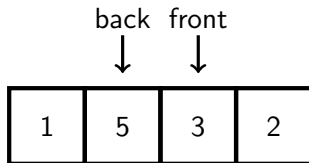
- Para deixar a fila em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.



Enfileiramento em vetores

Índice do início maior que o do fim

- Para deixar a fila em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.

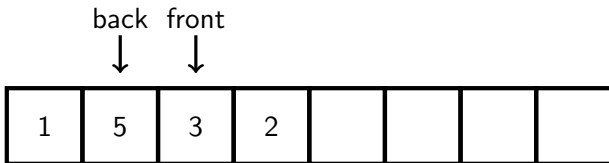




Enfileiramento em vetores

Índice do início maior que o do fim

- Para deixar a fila em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.

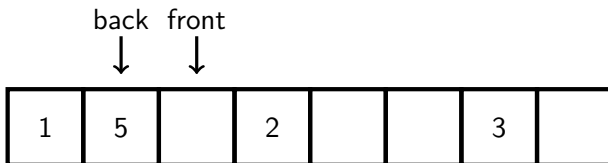




Enfileiramento em vetores

Índice do início maior que o do fim

- Para deixar a fila em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.





Enfileiramento em vetores

Índice do início maior que o do fim

- Para deixar a fila em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.





Enfileiramento em vetores

- Copiar os elementos ao duplicar o vetor não é uma operação tão custosa ao longo do tempo, pois só fazemos isso cada vez que o vetor está cheio. Isso só ocorre após $\frac{n}{2}$ inserções, logo, o custo é amortizado.



Sumário

- 2 Vetores Dinâmicos
 - Enfileiramento
 - Desenfileiramento

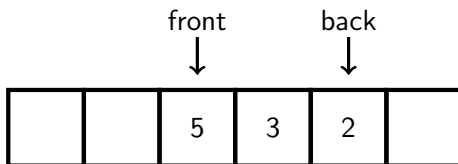


Desenfileiramento em vetores

- Desenfileirar consiste em incrementar o índice do início.

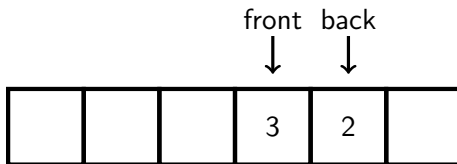


Desenfileiramento em vetores





Desenfileiramento em vetores



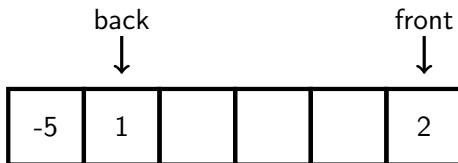


Desenfileiramento em vetores

- Utilizamos a mesma estratégia do enfileiramento: quando o índice do início estiver no final do vetor, damos a volta.

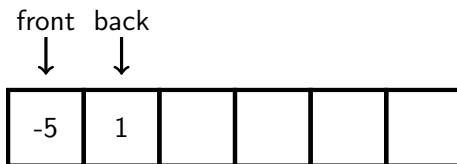


Desenfileiramento em vetores





Desenfileiramento em vetores



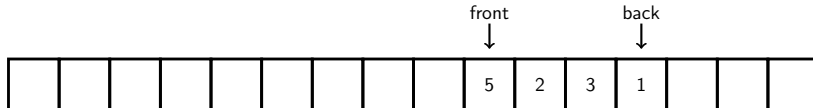


Desenfileiramento em vetores

- Quando a ocupação do vetor está em $\frac{1}{4}$ de sua capacidade, devemos reduzir a sua capacidade pela metade.
- Todos os elementos da fila são movidos para o início do vetor realocado, independente da posição dos índices de início e fim.

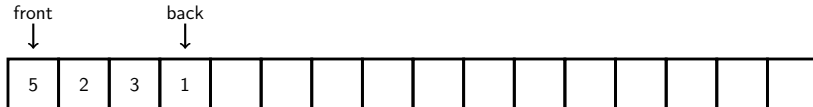


Desenfileiramento em vetores



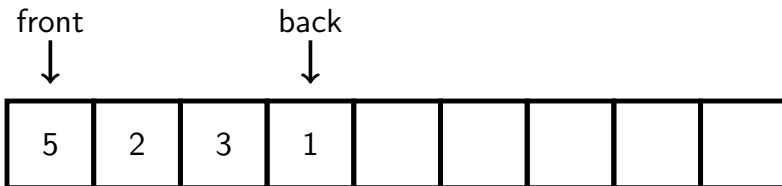


Desenfileiramento em vetores



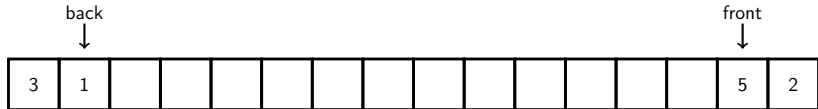


Desenfileiramento em vetores



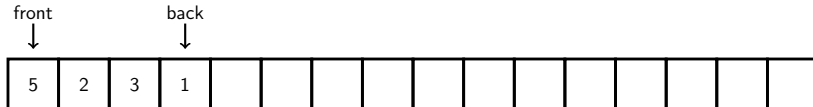


Desenfileiramento em vetores



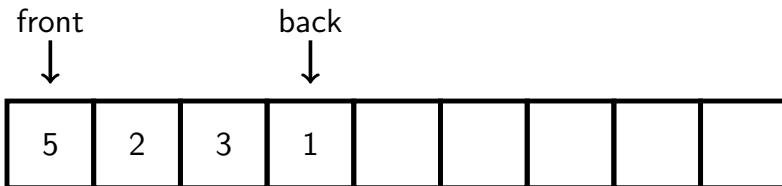


Desenfileiramento em vetores





Desenfileiramento em vetores





Desenfileiramento em vetores

- Copiar os elementos ao reduzir o vetor pela metade não é uma operação tão custosa ao longo do tempo, pois só fazemos isso quando o vetor está a $\frac{1}{4}$ de sua capacidade. Dessa forma, o custo é amortizado.



Sumário

3 Implementação em vetores



Sumário

3 Implementação em vetores

- Definição
- Inicialização
- Funções auxiliares
- Enfileirar
- Desenfileirar
- Acessar a frente
- Limpeza
- Análise



Definição

```
7 typedef struct queue_t {  
8     size_t front;  
9     size_t back;  
10    size_t size;  
11    size_t capacity;  
12    int *queue;  
13 } queue_t;
```



Sumário

3 Implementação em vetores

- Definição
- **Inicialização**
- Funções auxiliares
- Enfileirar
- Desenfileirar
- Acessar a frente
- Limpeza
- Análise



Inicialização

```
5 void queue_initialize(queue_t **q) {  
6     (*q) = mallocx(sizeof(queue_t));  
7     (*q)->front = 0;  
8     (*q)->back = 3;  
9     (*q)->size = 0;  
10    (*q)->capacity = 4;  
11    (*q)->queue = mallocx(sizeof(int) * 4);  
12 }
```




Sumário

3 Implementação em vetores

- Definição
- Inicialização
- **Funções auxiliares**
- Enfileirar
- Desenfileirar
- Acessar a frente
- Limpeza
- Análise



Funções auxiliares

```
72  size_t queue_size(queue_t *q) {  
73      return q->size;  
74  }
```



Funções auxiliares

```
76 bool queue_empty(queue_t *q) {  
77     return queue_size(q) == 0;  
78 }
```



Sumário

3 Implementação em vetores

- Definição
- Inicialização
- Funções auxiliares
- **Enfileirar**
- Desenfileirar
- Acessar a frente
- Limpeza
- Análise



Enfileirar

```
20 void queue_push(queue_t *q, int data) {
21     if (q->size == q->capacity) {
22         size_t old_capacity = q->capacity;
23         q->capacity *= 2;
24         q->queue = reallocx(q->queue, sizeof(int) * q->capacity);
25         if (q->front > q->back) {
26             for (size_t i = q->front; i < old_capacity; i++) {
27                 q->queue[i + old_capacity] = q->queue[i];
28             }
29             q->front = q->front + old_capacity;
30         }
31     }
32     q->back++;
33     if (q->back == q->capacity)
34         q->back = 0;
35     q->queue[q->back] = data;
36     q->size++;
37 }
```



Sumário

3 Implementação em vetores

- Definição
- Inicialização
- Funções auxiliares
- Enfileirar
- **Desenfileirar**
- Acessar a frente
- Limpeza
- Análise



Desenfileirar

```
39 void queue_pop(queue_t *q) {
40     assert(q->size > 0);
41     if (q->size == q->capacity / 4 && q->capacity > 4) {
42         size_t new_capacity = q->capacity / 2;
43         if (q->front <= q->back) {
44             for (size_t i = q->front, j = 0; i <= q->back; i++, j++) {
45                 q->queue[j] = q->queue[i];
46             }
47         } else {
48             size_t front_len = q->capacity - q->front;
49             for (int i = q->back; i >= 0; i--) {
50                 q->queue[i + front_len] = q->queue[i];
51             }
52             for (size_t i = q->front, j = 0; i < q->capacity; i++, j++) {
53                 q->queue[j] = q->queue[i];
54             }
55         }
56         q->front = 0;
57         q->back = q->size - 1;
58         q->capacity = new_capacity;
59         q->queue = reallocx(q->queue, q->capacity * sizeof(int));
60     }
61     q->front++;
62     q->size--;
63     if (q->front == q->capacity)
64         q->front = 0;
65 }
```



Sumário

3 Implementação em vetores

- Definição
- Inicialização
- Funções auxiliares
- Enfileirar
- Desenfileirar
- **Acessar a frente**
- Limpeza
- Análise



Acessar a frente

```
67 int queue_front(queue_t *q) {  
68     assert(q->front < q->capacity);  
69     return q->queue[q->front];  
70 }
```



Sumário

3 Implementação em vetores

- Definição
- Inicialização
- Funções auxiliares
- Enfileirar
- Desenfileirar
- Acessar a frente
- **Limpeza**
- Análise



Limpeza

```
14 void queue_delete(queue_t **q) {  
15     free((*q)->queue);  
16     free(*q);  
17     *q = NULL;  
18 }
```



Sumário

3 Implementação em vetores

- Definição
- Inicialização
- Funções auxiliares
- Enfileirar
- Desenfileirar
- Acessar a frente
- Limpeza
- Análise



Filas: análise

Complexidade das Operações

Operação	Complexidade
Enfileirar	$\Theta(1)$ amortizado
Desenfileirar	$\Theta(1)$ amortizado
Acessar a frente	$\Theta(1)$



Sumário

4 Filas



Representação de Filas em Listas

- Filas também podem ser implementadas por meio de estruturas auto-referenciadas.
- Uma das estruturas que podem prover as funcionalidades de uma Fila é uma Lista encadeada simples.



Operação de Filas em Listas

- Usando listas, a operação de verificar se uma fila está vazia é realizada ao verificar se a lista está vazia.
- O tamanho da fila é o tamanho da lista.
- Para enfileirar um elemento, basta inserir o elemento na cauda.
- Para desenfileirar um elemento, basta retirar o elemento da cabeça.
- Para acessar a frente da fila, basta acessar o elemento da cabeça.



Sumário

- 4 Filas
 - Análise



Filas

Filas: complexidade

Operação	Complexidade
Enfileirar	$\Theta(1)$
Desenfileirar	$\Theta(1)$
Verificar frente	$\Theta(1)$



Sumário

5 Exemplo



Exemplo de Utilização da Biblioteca

```
1  #include "alloc.h"
2  #include "queue.h"
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void) {
7      int i;
8      queue_t *q;
9      queue_initialize(&q);
10     for (i = 0; i < 16; i++) {
11         printf("Inserindo %d\n", i);
12         queue_push(q, i);
13     }
14     while (!queue_empty(q)) {
15         printf("Retirando %d\n", queue_front(q));
16         queue_pop(q);
17     }
18     queue_delete(&q);
19     return 0;
20 }
```