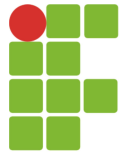
Filas

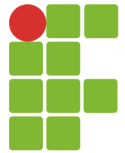
Estruturas de Dados e Algoritmos – Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

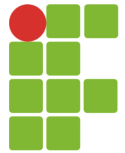
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário



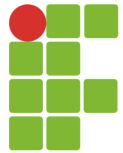
Sumário



Introdução

Filas

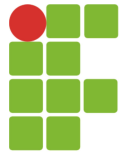
- Filas são um TAD em qual os elementos são mantidos em uma ordem específica. Esta ordem é a ordem FIFO (First-in-First-Out).
- A ordem FIFO se caracteriza pelo fato dos primeiros elementos a fazerem parte da estrutura, também serão os primeiros elementos a deixarem a estrutura.



Operações em Filas

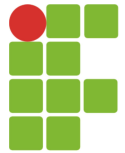
- Algumas das operações suportadas por uma fila devem ser:
  - Enfileiramento de elementos;
  - Desenfileiramento de elementos;
  - Verificar a frente da fila;
  - Verificar se a fila está vazia;

Filas



Representação de Filas

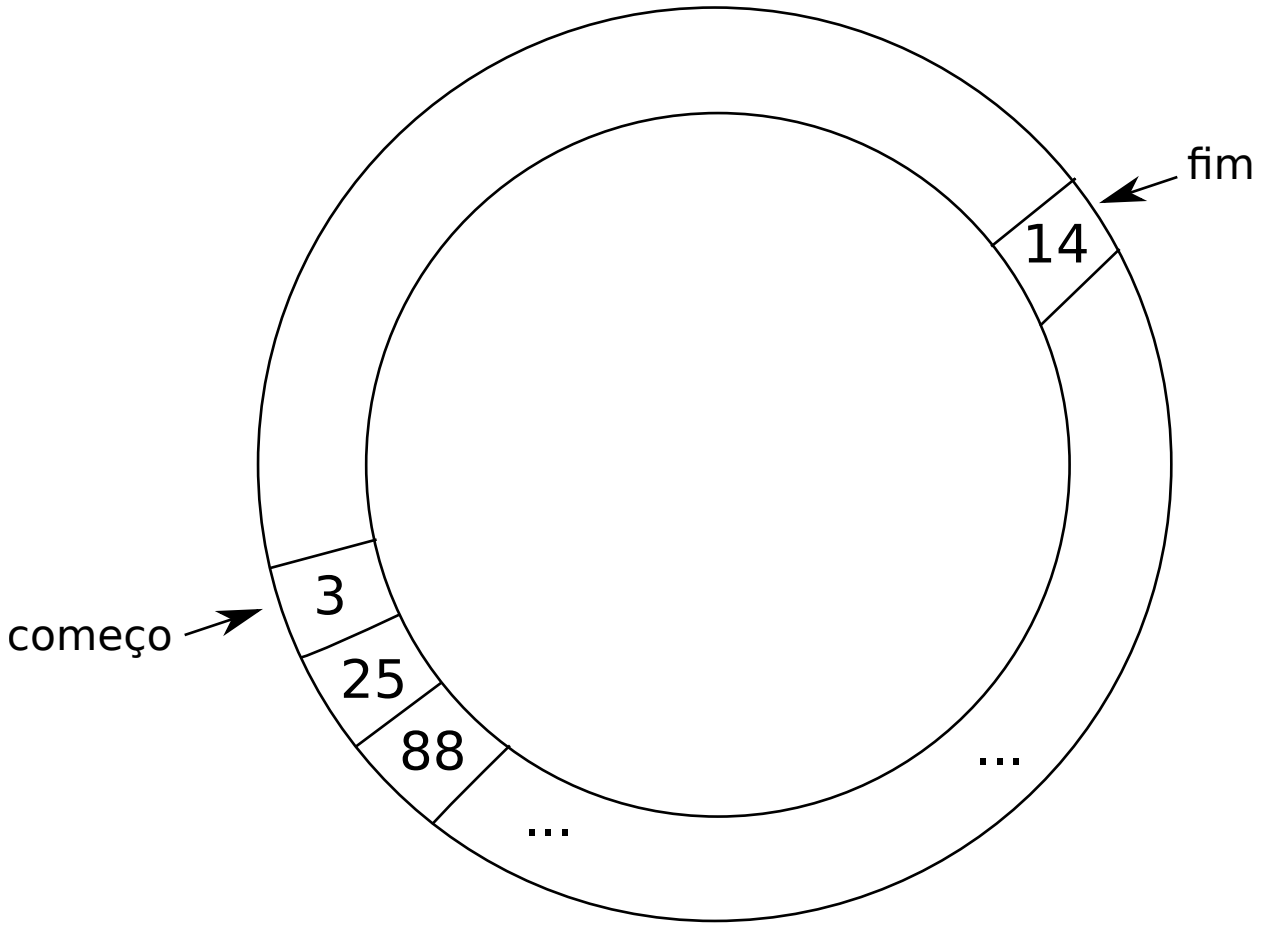
- Assim como listas, filas podem ser representadas de várias maneiras, duas delas são por meio de:
  1. Vetores;
  2. Estruturas auto-referenciadas;

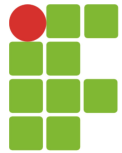


Representação de Filas em Vetores

- Para representar Filas em Vetores, usamos uma estrutura composta por um vetor e dois índices apontando respectivamente para a posição da frente (começo) da fila e a posição de trás (fim) da Fila.
- Para evitar no entanto um estouro dos índices do vetor, usa-se vetores circulares, implementados facilmente usando operações de resto nas linguagens de programação.

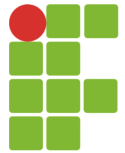
Representação de Filas em Vetores





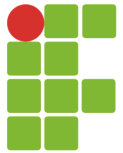
Operações de Filas em Vetores

- Para verificar se a fila está vazia, basta examinar os ponteiros de começo e fim.
- Para acessar a frente da Fila na representação em vetores circulares, basta acessar o elemento do vetor apontado pelo índice começo.
- Para enfileirar um elemento, basta escrever na posição do vetor apontada por fim.
- Por fim, para desenfileirar um elemento, basta ler o elemento do vetor apontado por começo.
- O único cuidado é manipular os índices começo e fim de modo a manter a consistência da estrutura.

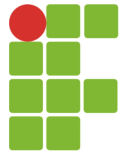


Problemas com a Representação

- Esta representação não é muito adequada em cenários dinâmicos mais complexos, pois, como o vetor tem limite, podemos sobreescrever a cabeça ao inserir um elemento na cauda.
- Estruturas auto-referenciadas tem uma gerência de memória mais simples e não precisamos nos preocupar com os limites da lista.



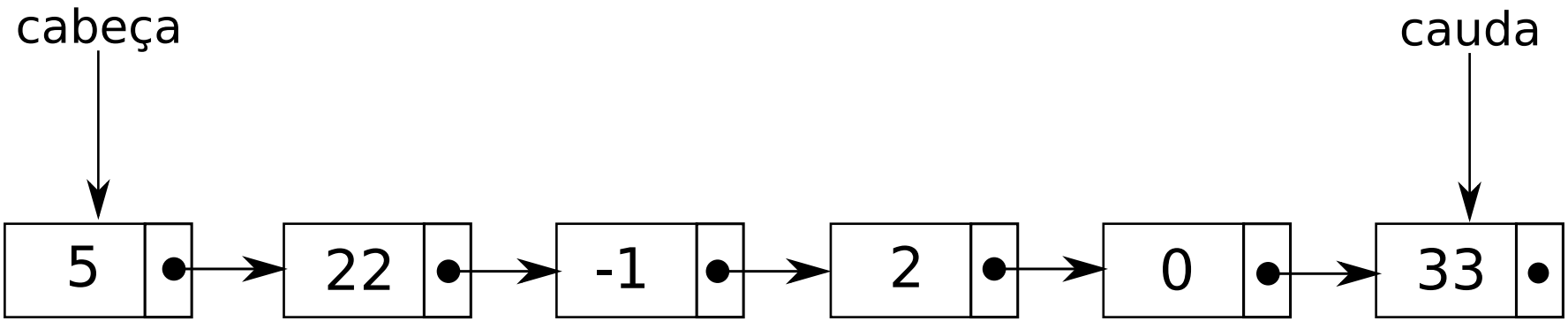
Sumário

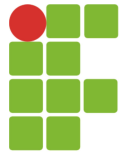


Representação de Filas em Listas

- 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.

Representação de Filas em Listas

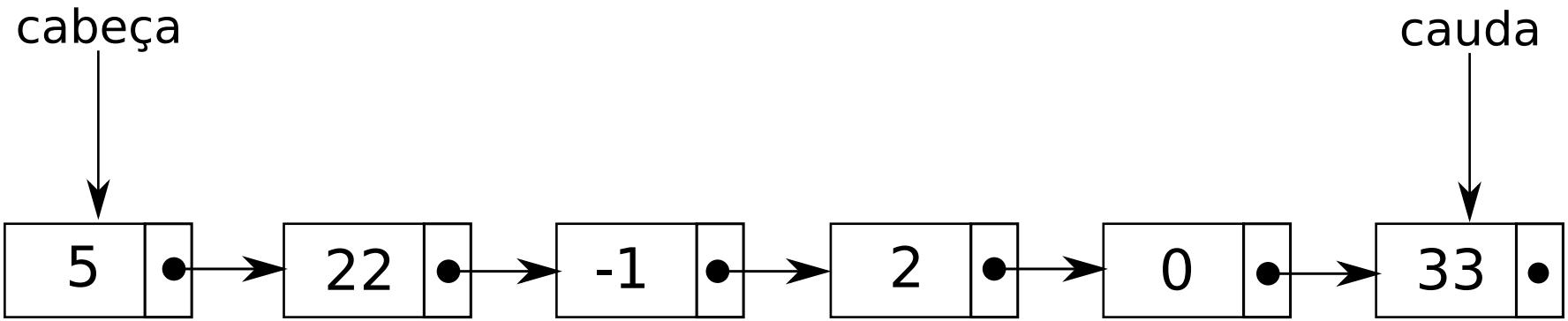


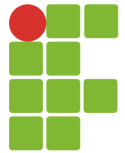


Operação de Filas em Listas

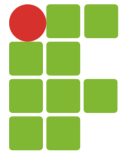
- Usando Listas, a operação de verificar se uma Fila está vazia é realizada ao verificar se a Lista está vazia.
- Para enfileirar um elemento, basta inserir o elemento na cauda.
- Para desenfileirar um elemento, basta retirar o elemento da cabeça.
- Para verificar a frente da fila, basta acessar o elemento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

Representação de Filas em Listas



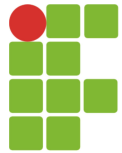


Sumário



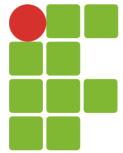
Filas: Definição

```
6  typedef void* (*queue_node_constructor_fn) (void*);
7  typedef void (*queue_node_destructor_fn)(void *);
```



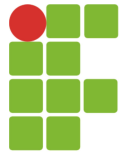
Filas: Definição

```
12 typedef struct queue_node_t{
13     void* data;
14     struct queue_node_t* next;
15 }queue_node_t;
```

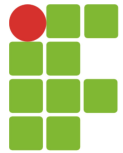


Filas: Definição

```
19 typedef struct queue_t{
20     queue_node_t* front;
21     queue_node_t* back;
22     queue_node_constructor_fn constructor;
23     queue_node_destructor_fn destructor;
24     size_t size;
25 }queue_t;
```

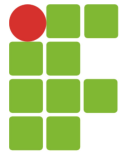


Sumário

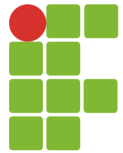


Filas: Inicialização

```
7  void queue_initialize(queue_t** q,queue_node_constructor_fn constructor,
8                        queue_node_destructor_fn destructor){
9      (*q) = mallocx(sizeof(queue_t));
10     (*q)->back = NULL;
11     (*q)->front = NULL;
12     (*q)->size = 0;
13     (*q)->constructor = constructor;
14     (*q)->destructor = destructor;
15 }
```

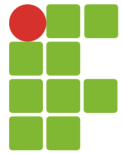


Sumário



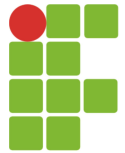
Filas: Funções Auxiliares

```
60  size_t queue_size(queue_t* q){
61      return(q->size);
62  }
```

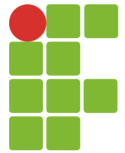


Filas: Funções Auxiliares

```
64  size_t queue_empty(queue_t *q){
65      return queue_size(q)==0 ? 1 : 0;
66  }
```

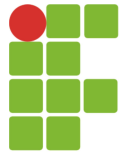


Sumário

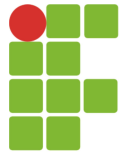


Filas: Enfileirar

```
46  void queue_push(queue_t* q,void* data){
47      queue_node_t* new_node = mallocx(sizeof(queue_node_t));
48      new_node->data = q->constructor(data);
49      new_node->next = NULL;
50      if(queue_size(q)==0){
51          q->front = new_node;
52      }
53      else{
54          q->back->next = new_node;
55      }
56      q->back = new_node;
57      q->size++;
58  }
```

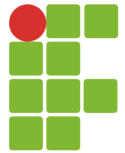


Sumário

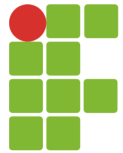


Filas: Acesso

```
41 void* queue_front(queue_t* q){
42     assert(!queue_empty(q));
43     return(q->front->data);
44 }
```

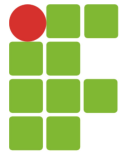


Sumário

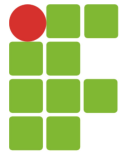


Filas: Enfileirar

```
26  void queue_pop(queue_t* q){
27      assert(!queue_empty(q));
28      queue_iterator_t it = q->front;
29      if(queue_size(q)==1){
30          q->front = NULL;
31          q->back = NULL;
32      }
33      else{
34          q->front = q->front->next;
35      }
36      q->destructor(it->data);
37      free(it);
38      q->size--;
39  }
```

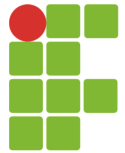


Sumário

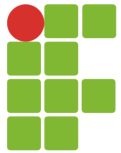


Filas: Enfileirar

```
17  void queue_delete(queue_t** q){
18      while(!queue_empty(*q)){
19          queue_pop(*q);
20      }
21      free(*q);
22      *q = NULL;
23  }
```



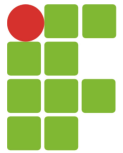
Sumário



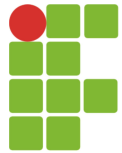
Filas

Complexidade das Operações

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

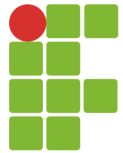


Sumário



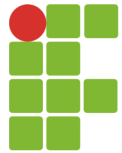
Exemplo de Utilização da Biblioteca

```
6  typedef struct pessoa{
7      char nome[30];
8      char cpf[20];
9      int idade;
10 }pessoa;
```



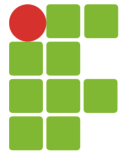
Exemplo de Utilização da Biblioteca

```
12  void* constructor_pessoa(void* data){
13      void* ptr = mallocx(sizeof(pessoa));
14      memcpy(ptr,data,sizeof(pessoa));
15      return ptr;
16  }
```



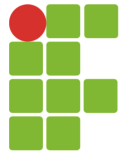
Exemplo de Utilização da Biblioteca

```
35  void destructor_pessoa(void* data){
36      free(data);
37  }
```



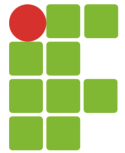
Exemplo de Utilização da Biblioteca

```
18 void my_getline(char* str,size_t size){
19     int i;
20     char c;
21     for(i=0;i<size-1;i++){
22         c = getchar();
23         if(c=='\n'){
24             str[i] = '\0';
25             break;
26         }
27         str[i] = c;
28     }
29     str[size-1]='\0';
30     while(c!='\n'){
31         c = getchar();
32     }
33 }
```



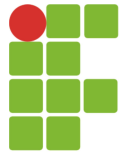
Exemplo de Utilização da Biblioteca

```
39  void cadastra_pessoa(pessoa* p){
40      printf("Nome: ");
41      my_getline(p->nome,30);
42      printf("CPF: ");
43      my_getline(p->cpf,20);
44      printf("Idade: ");
45      scanf("%d%*c",&p->idade);
46  }
```



Exemplo de Utilização da Biblioteca

```
48  void imprime_pessoa(const pessoa* p){
49      printf("Nome: ");
50      printf("%s\n",p->nome);
51      printf("CPF: ");
52      printf("%s\n",p->cpf);
53      printf("Idade: ");
54      printf("%d\n",p->idade);
55  }
```



Exemplo de Utilização da Biblioteca

```
57  int main(void){
58      int i;
59      queue_t* q;
60      pessoa p;
61      queue_initialize(&q,constructor_pessoa,destructor_pessoa);
62      for(i=0;i<5;i++){
63          printf("Cadastrando pessoa %d\n",i+1);
64          cadastra_pessoa(&p);
65          queue_push(q,&p);
66      }
67      while(!queue_empty(q)){
68          printf("\n**Imprimindo pessoa**\n");
69          p =  *(pessoa*) queue_front(q);
70          queue_pop(q);
71          imprime_pessoa(&p);
72          printf("\n");
73      }
74      queue_delete(&q);
75      return 0;
76  }
```