

Filas e Deques

Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

Baseado nos slides do Prof. Rudinei Goularte

Conteúdo

- Filas
- Deques

Fila

- O que é?
- Para que serve?






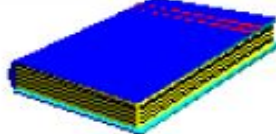

Exemplo de Aplicação

- Problema: automação de uma biblioteca
 - ◆ Todos os livros devem ser cadastrados
 - ◆ O sistema deve informar se um livro está disponível ou não nas estantes
 - ◆ Caso o livro não esteja disponível, o usuário pode aguardar em uma fila de espera
 - ◆ Quando o livro for devolvido, o primeiro da fila de espera pode retirá-lo
- Sua tarefa: desenvolver esse sistema

Modelagem do Problema

- 1º passo: abstração
 - ◆ Identificar os elementos do mundo real que são relevantes para a solução do problema
 - Quais são eles?

Modelagem do Problema

fila de espera para o livro	livros do acervo	disponível?
<p>último ---->  <---- 1º</p>	<p> trigonometria</p>	não
<p>último ---->  <---- 1º</p>	<p> química inorgânica</p>	não
<p>fila vazia!</p>	<p> estruturas de dados</p>	sim

Modelagem do Problema

→ Elementos relevantes

- ◆ Um cadastro de livros
- ◆ Indicação da disponibilidade dos livros
- ◆ Uma fila de espera para cada livro, com indicação da ordem das pessoas
- ◆ Primeiro e último da fila
- ◆ Cadastro de pessoas: nome, endereço e telefone

Modelagem do Problema

- 2º passo: quais são as operações possíveis nas filas?
 - ◆ Entrar na fila
 - Quem entra, entra onde?
 - ◆ Sair da fila
 - Quem sai, sai de onde?
 - ◆ Outras?

Fila (Queue)

→ O que é?

- ◆ *É uma estrutura para armazenar um conjunto de elementos, que funciona da seguinte forma*
 - Novos elementos sempre entram no fim da fila
 - O único elemento que se pode retirar da fila em um dado momento é seu primeiro elemento

Fila (Queue)

→ Para que serve?

- ◆ Modelar situações em que é preciso armazenar um conjunto ordenado de elementos, no qual o primeiro elemento a entrar no conjunto será também o primeiro elemento a sair do conjunto, e assim por diante
 - Política FIFO: first in, first out

Aplicações

- Exemplos de aplicações de filas
 - ◆ Filas de espera e algoritmos de simulação
 - ◆ Controle por parte do sistema operacional de recursos compartilhados, tais como impressoras
 - ◆ Buffers de Entrada/Saída
 - ◆ Estrutura de dados auxiliar em alguns algoritmos como a busca em largura

TAD Fila

→ Operações principais

- ◆ `fila_criar()`: cria uma fila F vazia
- ◆ `fila_entra(F,x)`: insere o elemento x no final da fila F. Retorna **true** se foi possível inserir, **false** caso contrário
- ◆ `fila_sai(F)`: remove o elemento no início de F, e retorna esse elemento. Retorna NULL se não foi possível remover

TAD Fila

→ Operações auxiliares

- ◆ `fila_frente(F)`: retorna o elemento no início de F , sem remover
- ◆ `fila_tamanho(F)`: retorna o número de elementos em F
- ◆ `fila_vazia(F)`: indica se a fila F está vazia
- ◆ `fila_cheia(F)`: indica se a fila F está cheia (útil para implementações estáticas)

TAD Fila - exemplo

Operação	Fila	resultado
fila_criar()	1º da fila □	
fila_entra(F,a)	1º da fila □ a	
fila_entra(F,b)	1º da fila □ a, b	
fila_entra(F,c)	1º da fila □ a, b, c	
fila_sai(F)	1º da fila □ b, c	return (a)
fila_entra(F,d)	1º da fila □ b, c, d	
fila_sai(F)	1º da fila □ c, d	return (b)

Implementação

→ Alocação sequencial

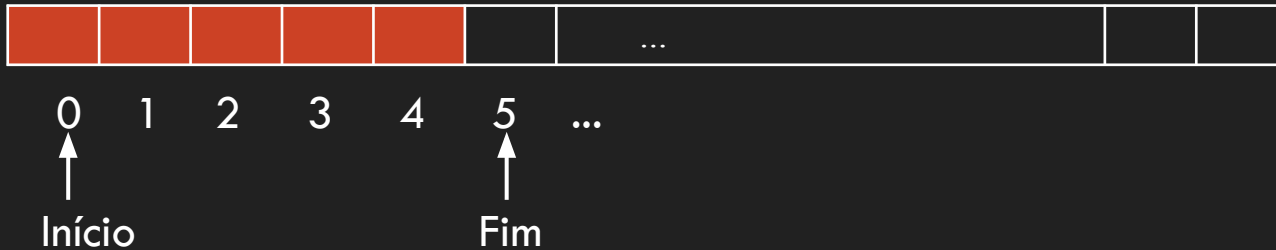
- ◆ Os elementos da fila ficam, necessariamente, em sequência (um ao lado do outro) na memória

→ Alocação estática

- ◆ Todo o espaço de memória a ser utilizado pela fila é reservado (alocado) em tempo de compilação
- ◆ Todo o espaço reservado permanece reservado durante todo o tempo de execução do programa, independentemente de estar sendo efetivamente usado ou não

Implementação de Fila

- **Início:** aponta para/indica o primeiro da fila, ou seja, o primeiro elemento a sair
- **Fim:** aponta para/indica o fim da fila, ou seja, onde o próximo elemento entrará



Implementação de Fila

- Qual a condição inicial, quando a fila é criada?
- Qual a condição para fila vazia?
- Qual a condição para fila cheia?

Exemplo de uso

- Array com 6 posições
- `fila_criar()`
 - ◆ `Início = 0, Fim = 0`
- `fila_entra(F, a), fila_entra(F, b), fila_entra(F, c)`
 - ◆ Qual o estado de `F`, `Início` e `Fim`?
- `fila_entra(F, z), fila_entra(F, r), fila_entra(F, s)`
 - ◆ `fila_cheia(F) == TRUE`
- `fila_sai(F), fila_sai(F)`
 - ◆ Qual o problema?

Implementação Sequencial

- As inserções (fila_entra) fazem com que o contador f seja incrementado ($O(1)$)



- Mas as remoções (fila_sai) requerem deslocar todos os elementos ($O(n)$)
- É possível melhorar isso?

Implementação Sequencial Circular

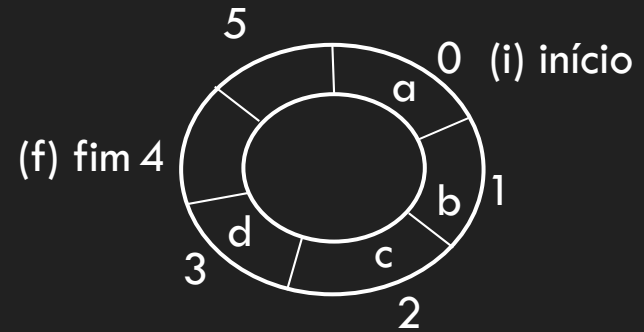
- A solução é fazer com que o início não seja fixo na primeira posição do vetor
 - ◆ Pode-se permitir que f volte para o início do vetor quando esse contador atingir o final do vetor
- Essa implementação é conhecida como fila circular
- Na figura abaixo a fila circular possui 4 posições vagas e $f < i$



Implementação Sequencial Circular

→ Portanto, deve-se “ver” a fila como um “anel” – Fila Circular

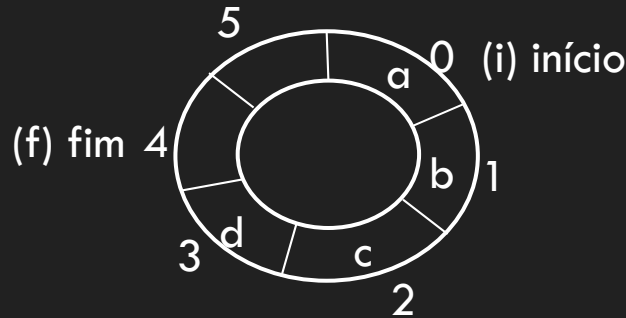
- Qual a condição inicial (quando a fila é criada)?
- Qual a condição para fila vazia?
- Qual a condição para fila cheia?



Implementação Sequencial Circular

→ **Solução:** campo extra para guardar número de elementos

- Qual a condição inicial (quando a fila é criada)?
- Qual a condição para fila vazia?
- Qual a condição para fila cheia?



Total = 4

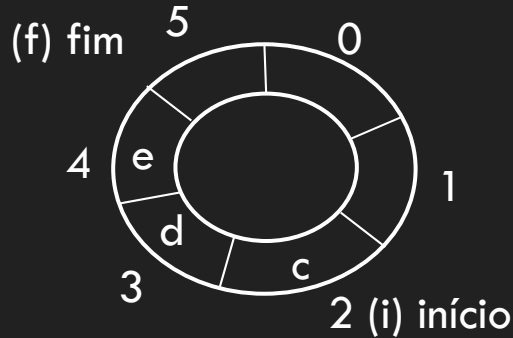
Implementação Sequencial Circular

- Qual a condição inicial (quando a fila é criada)?
 - ◆ $Total=0, início=0, fim=0$
- Qual a condição para fila vazia?
 - ◆ $Total=0$
- Qual a condição para fila cheia?
 - ◆ $Total=tamanho\ da\ fila$

Implementação Sequencial Circular

→ Exemplo

- ◆ $\text{inicio} = 2$, $\text{fim} = 5$, $\text{total} = 3$
- ◆ $\text{entra}(\text{f})$, $\text{entra}(\text{g})$, $\text{entra}(\text{h})$



Implementação Sequencial Circular na Heap

- A fila é sequencial, como no caso de alocação estática. Contudo, está alocada na Heap.
- ◆ Facilidade de modelagem e implementação como TAD.
- ◆ Chamaremos daqui em diante apenas de Fila Sequencial Circular

Definição de Tipos

```
(fila.h)
...
#include "item.h"
#define TAM 100
#define TRUE 1
#define FALSE 0
typedef struct fila_ FILA;

FILA *fila_criar(void);
boolean fila_inserir(FILA *fila, ITEM *item);
boolean fila_remove(FILA *fila);
ITEM *fila_busca(FILA *fila, int chave);
int fila_tamanho(FILA *fila);
...
```

Definição de Tipos

```
(fila.c)
```

```
...
```

```
#include "fila.h"
```

```
struct fila_{
```

```
    ITEM *itens[TAM];
```

```
    int inicio; /*posicao do 1o elemento da fila*/
```

```
    int fim;    /*posicao do ultimo elemento da fila*/
```

```
    int tamanho;
```

```
};
```

```
...
```

Implementação Sequencial Circular

```
/*Cria logicamente uma fila, inicialmente vazia*/
FILa *fila_criar(void){
    /*pré-condição: existir espaço na memória.
    Na implementação estática não há o que verificar*/
    FILa *fila = (FILa *) malloc(sizeof(FILa));
    fila->inicio = 0;
    fila->fim = 0;
    fila->tamanho = 0; /* fila vazia*/
    return (fila);
}
int fila_cheia(FILa *fila) {
    return (fila->tamanho == TAM);
}
int fila_vazia(FILa *fila) {
    return (fila->tamanho == 0);
}
```

Implementação Sequencial Circular

```
/*Insere um elemento no fim da fila. FILA NÃO ORDENADA. Fila Circular. */
boolean fila_inserir(FILA *fila, ITEM *item){
    if (fila != NULL && (!fila_cheia(fila)) ){
        fila->itens[fila->fim] = item;
        fila->fim = (fila->fim+1) % TAM;
        fila->tamanho ++;
        return(TRUE);
    }
    return(FALSE);
}
```

Implementação Sequencial Circular

```
ITEM *fila_remove(FILA *fila){
    if (fila != NULL && (!fila_vazia(fila)) ) {
        ITEM *ret = item_criar(item_get_chave(fila_inicio(fila));
        item_apagar(&fila->itens[fila->inicio]);
        fila->itens[fila->inicio] = NULL;
        fila->inicio = (fila->inicio + 1) % TAM;
        fila->tamanho --;
        return (ret);
    }
    return (NULL);
}
```

Implementação Sequencial Circular

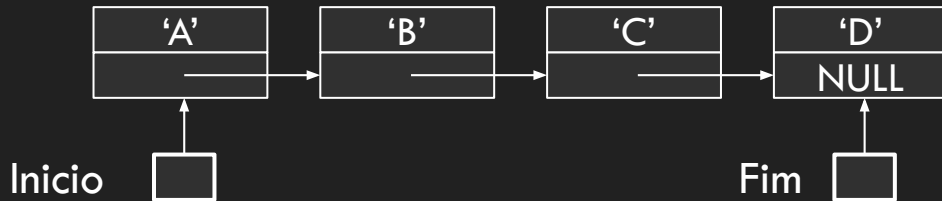
```
boolean fila_remove(FILA *fila){
    if (fila != NULL && (!fila_vazia(fila)) ) {
        item_apagar(&fila->itens[fila->inicio]);
        fila->itens[fila->inicio] = NULL;
        fila->inicio = (fila->inicio + 1) % TAM;
        fila->tamanho --;
        return (TRUE);
    }
    return (FALSE);
}
```

Implementação Sequencial Circular

```
int fila_tamanho(FILA *fila) {  
    if (fila != NULL)  
        return (fila->tamanho);  
    return (ERRO);  
}
```


Implementação Encadeada

- A implementação encadeada* pode ser realizada mantendo-se dois ponteiros, um para o início e outro para o final da fila
- Com isso pode-se ter acesso direto às posições de inserção e remoção
- ◆ * Supõe-se aqui encadeamento com alocação dinâmica



Implementação Encadeada

- As operações **inserir** e **remover** implementadas dinamicamente são bastante eficientes
- Deve-se tomar cuidado apenas para que os ponteiros para início e final tenham valor NULL quando a fila estiver vazia

Sequencial versus Encadeada

Operação	Sequencial	Encadeada
Criar	$O(1)$	$O(1)$
Apagar	$O(1)^*$	$O(n)$
Inserir	$O(1)$	$O(1)$
Remover	$O(1)$ (circular)	$O(1)$
Frente	$O(1)$	$O(1)$
Vazia	$O(1)$	$O(1)$
Cheia	$O(1)$	$O(1)$
Tamanho	$O(1)$	$O(1)$ (c/ contador)

* Se a Fila for implementada como um array na *stack*. Caso seja implementada como um array na *heap* (como temos feito), será $O(n)$.

Sequencial versus Encadeada

→ Sequencial

- ◆ Implementação simples
- ◆ Tamanho da fila definido a priori

→ Encadeada

- ◆ Alocação dinâmica permite gerenciar melhor estruturas cujo tamanho não é conhecido a priori ou que variam muito de tamanho

Dequeues

- *Double Ended QUEUE*
- Deques são estruturas que permitem inserir e remover de ambos os extremos

Aplicações

- Alguns Sistemas de escalonamento de processos com múltiplas CPU's
- Verificadores de palíndromos (sequências de itens que são iguais quando lidos da esquerda para direita ou da direita para esquerda)
- Opção desfazer/refazer em programas de edição de imagem, texto etc.

TAD Deques

→ Operações principais

- ◆ `inserir_inicio(D,x)`: insere o elemento `x` no início da deque `D`. Retorna **true** se foi possível inserir e **false** caso contrário
- ◆ `inserir_fim(D,x)`: insere o elemento `x` no final da deque `D`. Retorna **true** se foi possível inserir e **false** caso contrário
- ◆ `remover_inicio(D)`: remove o elemento no início de `D`, e retorna esse elemento. Retorna `NULL` se não foi possível remover
- ◆ `remover_fim(D)`: remove o elemento no final de `D`, e retorna esse elemento. Retorna `NULL` se não foi possível remover

TAD Deques

→ Operações auxiliares

- ◆ primeiro(D): retorna o elemento no início de D. Retorna NULL se o elemento não existe
- ◆ ultimo(D): retorna o elemento no final de D. Retorna NULL se o elemento não existe
- ◆ contar(D): retorna o número de elementos em D
- ◆ vazia(D): indica se a deque D está vazia
- ◆ cheia(D): indica se a deque D está cheia (útil para implementações estáticas)

TAD Deques

- Como deques requerem inserir e remover elementos em ambos os extremos
 - ◆ Implementação sequencial circular
 - ◆ Implementação dinâmica duplamente encadeada
- Nesses casos, as operações do TAD são $O(1)$

Implementação Sequencial Circular

- Exercício: Implementar uma DEQUE
 - ◆ Aproveite a implementação de uma fila circular estática e acrescente as duas operações que faltam:
 - remover do fim; e
 - inserir no início
 - ◆ As outras operações são as mesmas

Implementação - Inserir no Início

→ Se $x \geq 0$, então $(x + N) \% N = x$

→ Se $x = -1$, então $(x + N) \% N = N - 1$

◆ x é a posição que se deseja inserir: `deque->inicio - 1`

```
boolean deque_inserir_inicio(DEQUE *deque, ITEM *item) {  
    if (deque != NULL && !deque_cheia(deque)) {  
        deque->inicio = (deque->inicio - 1 + TAM) % TAM;  
        deque->itens[deque->inicio] = item;  
        deque->tamanho ++;  
        return (TRUE);  
    }  
    return (FALSE);  
}
```

Implementação - Remover do Fim

```
ITEM *deque_remove_fim(DEQUE *deque) {  
    if (deque != NULL && !deque_vazia(deque)) {  
        deque->fim = (deque->fim - 1 + TAM) % TAM;  
        ITEM* i = deque->itens[deque->fim];  
        deque->itens[deque->fim] = NULL;  
        deque->tamanho --;  
        return (i);  
    }  
    return (NULL);  
}
```

→ Se $x \geq 0$, então $(x + N) \% N = x$

→ Se $x = -1$, então $(x + N) \% N = N - 1$

Exercícios

- Implemente uma fila dinâmica
- Implemente uma deque dinâmica
- Implemente um procedimento recursivo capaz de esvaziar uma fila
- Implemente um procedimento para inverter uma fila (o primeiro elemento se tornará o último e vice-versa)

Referências

- ZIVIANI, N. Projeto de Algoritmos, Thomson, 2a. Edição, 2004.