Professor
Marcio Feitosa

CURSO: Estruturas de Dados

TAD FILA

O TAD Fila (*queue*, no inglês) é uma estrutura linear com dois lados de acesso, sendo que a inserção (ou entrada) de dados só é possível por um dos lados e a remoção (ou saída) apenas pelo outro.

O lado de saída é chamado de **início** ou *head* (cabeça) e o lado de entrada é o chamado **final** ou *tail* (rabo).

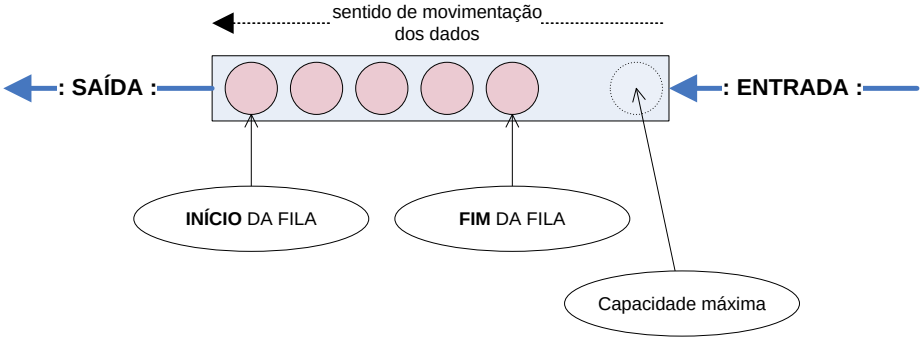

Figura 1

O objetivo do TAD Fila é representar problemas que envolvam enfileiramento de elementos que necessitem de algum tipo de atendimento.

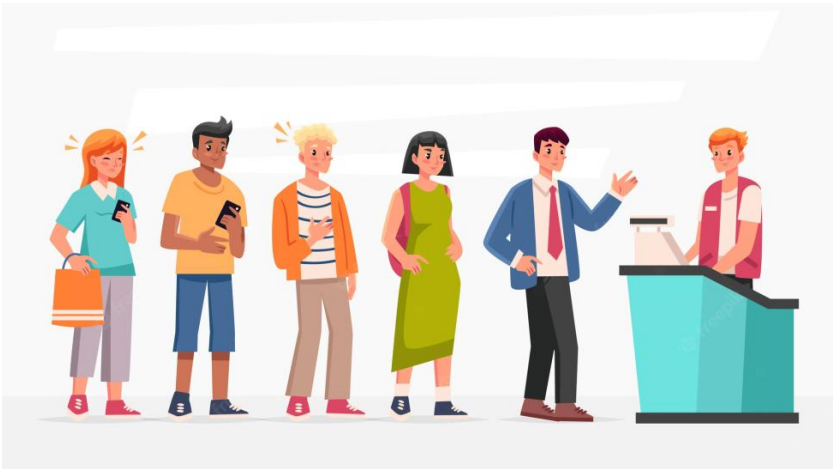

Figura 2

A Figura 2 ilustra uma fila de pessoas a serem atendidas pelo rapaz que está no balcão verde.

Tecnicamente o elemento atendente é chamado de **servidor**. Também existem modelos de fila com atendimento multiservidor, conforme ilustrado na Figura 3 abaixo.

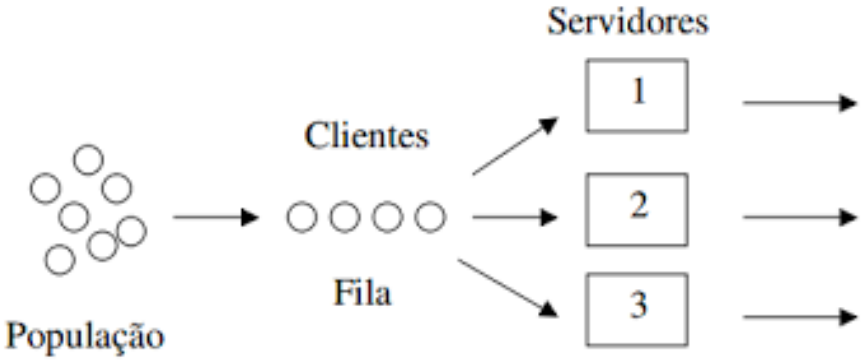

Figura 3

Neste modelo existe uma única fila e diversos servidores para atender os elementos da fila.

Modo de operação

A fila opera no modelo chamado FIFO (*first in first out*), ou, traduzindo, o primeiro a entrar será o primeiro a sair.

O enfileiramento de elementos sempre ocorre quando[1]:

- O servidor está indisponível e elementos precisam ser atendidos para passar para outra etapa.
- O servidor está disponível, mas o taxa de chegada de elementos por unidade de tempo é maior que a capacidade de atendimento do servidor.

[1] Na realidade o problema das filas, conhecido como Teoria das Filas, é extremamente complexo e não está contemplado no nosso conteúdo. Está sendo dada apenas uma "pincelada" muito básica.

Professor Marcio Feitosa

Tipicamente precisamos dos seguintes métodos:

- Para a operação de inserção – **enqueue(elemento)** – (enfileirar)
- Para a operação de extração – **dequeue()** – (desenfileirar)

A possibilidade de leitura, ou não, dos elementos da fila depende da natureza da aplicação.

Embora os dois métodos relacionados acima sejam suficientes para a operação da fila, outros três métodos costumam aparecer nas implementações mais básicas:

- **peek()** – lê o primeiro elemento da fila sem removê-lo.
- **isFull()** – verifica se o tamanho da fila está na capacidade máxima do contêiner.
- **isEmpty()** – verifica se a fila está vazia (0 elemento).

Estamos utilizando a nomenclatura convencional internacional, mas encontramos inúmeras implementações com nomenclatura no idioma do país (caso não seja inglês).

Questões de desempenho

Comumente as implementações de fila são feitas em vetores. Diferentemente da pilha que tem apenas um ponto móvel (topo), a fila tem dois, o início e o final.

Se implementarmos o TAD Fila à imagem e semelhança da realidade, teremos o seguinte problema: ao ser chamado pelo servidor, o elemento do início sai da fila e deixa a posição vaga, que, no mundo real é ocupada pelo anteriormente segundo colocado da fila, a sua posição pelo terceiro e assim por diante. Ou seja, todos os elementos da fila se deslocam uma posição à frente.

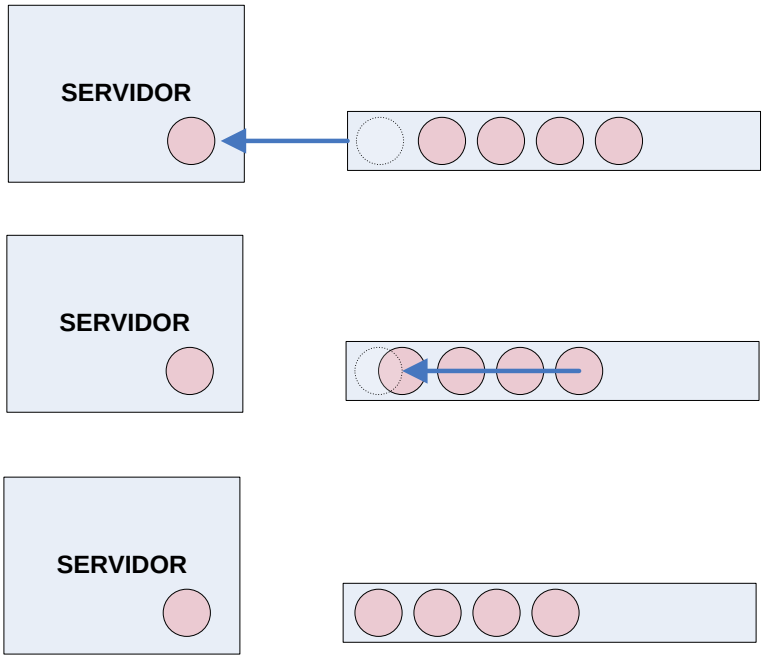

Figura 4

Esse problema aumenta na medida da quantidade de elementos na fila.

A solução para isto está em desvincular o início da fila da posição 0 (zero) do vetor e vinculá-lo a um ponteiro móvel.

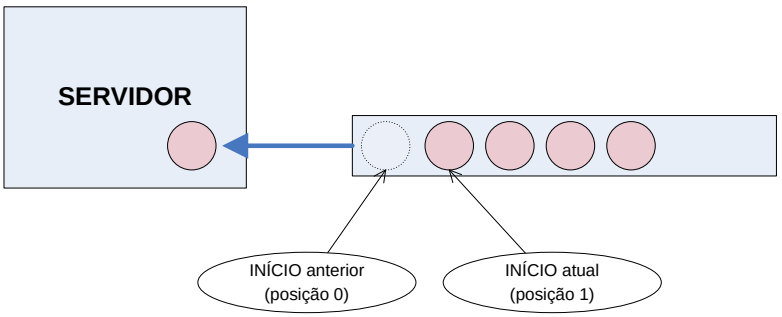

Figura 5

Da mesma forma o final também é móvel, mas neste modelo de **início** móvel, o ponteiro do **final** só se move quando entrar um novo elemento na fila[2].

Agora fazemos a seguinte pergunta: muito bem, mas e quando o ponteiro do final chegar ao final do vetor, o que fazer? É aí que entra o conceito do vetor circular, onde "teoricamente" o final se liga ao início. Dessa forma quando o ponteiro do **final** chegar à última posição do vetor, passará à posição zero na próxima inserção.

[2] No modelo de **início** fixo em zero, o ponteiro de **final** se moveria também com o deslocamento para a frente de todo o conjunto devido à saída de um elemento.

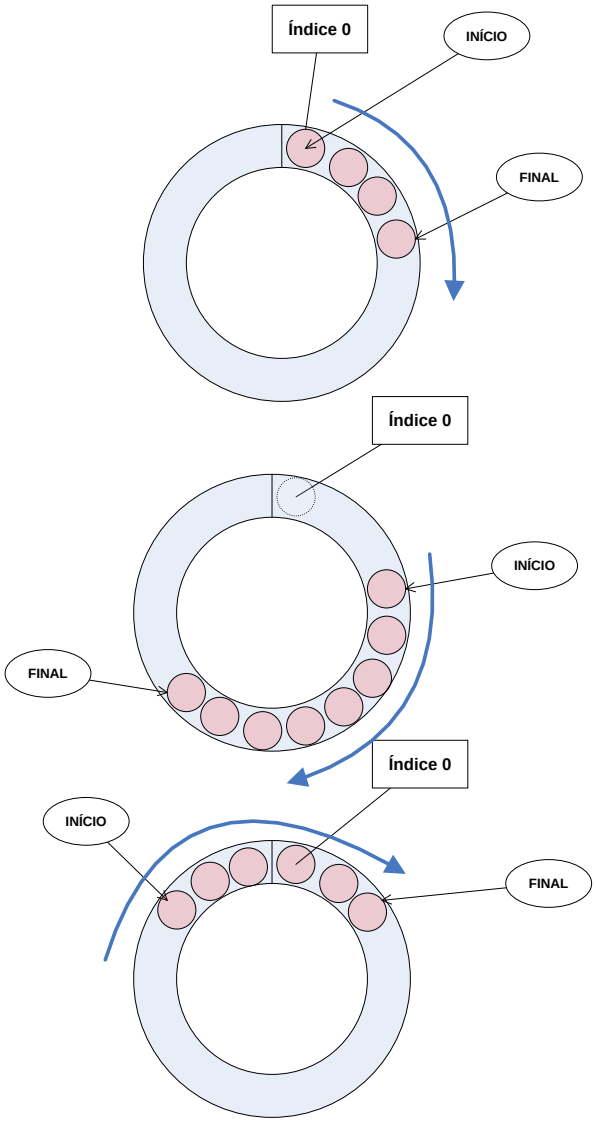

Figura 6

Observamos que na situação em que o ponteiro de final passa à posição zero, fica em numeração inferior ao ponteiro de início.

Ora, como calculamos a nova posição dos ponteiros, por ocasião de inserção ou remoção, uma vez que podem voltar a zero quando completarem a "volta" no vetor?

A forma mais simples é incrementando o ponteiro (ele deve ser incrementado de qualquer forma) e obtendo o resto da divisão inteira pelo tamanho máximo do container da fila, que aqui iremos implementar com vetor.

novo índice = (índice atual + 1) % tamanho vetor

Exemplo (tanto para início como para final):

- Tamanho vetor (capacidade máxima) = 10 elementos (índices de 0 a 9)
- Ponteiro no índice 8 (nona posição)
- Novo índice = (8 + 1) % 10

Na divisão inteira, não é possível se dividir 9 por 10, portanto o resultado é zero e sobram 9 (o resto é 9). Nova posição do ponteiro = 9.

Na próxima movimentação:

- Novo índice = (9 + 1) % 10

10 / 10 = 1 e sobram 0, portanto o próximo índice é zero.

E assim os índices vão rodando pelo vetor sempre no mesmo sentido.

Como fazer esta implementação?

Há mais de uma forma. Inicializando os ponteiros em -1, inicializando em zero, considerando o ponteiro de final apontando para o elemento, considerando o ponteiro apontando para a próxima posição vaga.

A nossa implementação será a seguinte:

- Ponteiros de início e fim inicializados em zero
- Ponteiro de início aponta para o primeiro elemento
- Ponteiro de fim aponta para a próxima posição vaga.
- Flag indicador da última operação realizada: -1 = saída; +1 = entrada

Para obter informações sobre a fila:

isEmpty() – se **início** == **fim** e a última operação foi de remoção.

isFull() – se **início** == **fim** e a última operação foi de inserção.

size()

- se inicio < fim → *size* = fim - inicio
- se início == fim
  - tamanho = 0 se vazia
  - tamanho = max se cheia
- se inicio > fim → *size* = max – inicio + fim

Professor
Marcio Feitosa

Mas, vamos ao código!!

----- 10101010101010 -----