Alocação sequencial - filas

Filas

A estrutura de dados Fila também é bastante intuitiva. A analogia é com uma fila de pessoas aguardando para serem atendidas no guichê de um banco, ou aguardando o ônibus. Se houver respeito, todos obedecem a fila, isto é, quando chega um novo elemento, este se posiciona no final da fila e quando libera um lugar quem vai é quem está no início da fila.

Assim, uma fila é caracterizada pela seqüência:
O primeiro a entrar é o primeiro a sair.
Ou
O último que entrou é o último a sair.

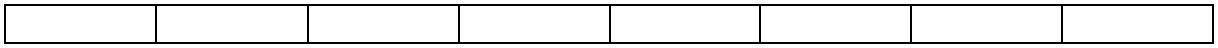
As expressões em inglês que definem esta estrutura:
`FIFO – first in first out` ou
`LILO – last in last out`

Considere o exemplo abaixo que mostra a evolução de uma fila. Uma letra significa "adicione um elemento na fila" e um ponto significa "retire um elemento da fila".

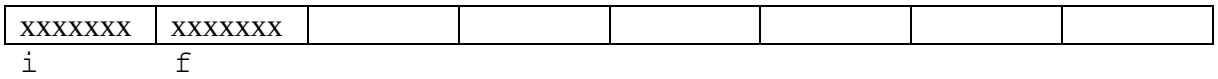
Operação	Retirado	Fila
F		F
I		FI
L		FIL
.	F	IL
A		ILA
.	I	LA
D		LAD
E		LADE
.	L	ADE
.	A	DE
E		DEE
L		DEEL
E		DEELE
M		DEELEM
E		DEELEME
.	D	EELEME
N		EELEMEN
.	E	ELEMEN
T		ELEMENT
.	E	LEMENT
O		LEMENTO
S		LEMENTOS
.	L	EMENTOS

Para se implementar uma fila num vetor de elementos são necessários 2 apontadores. Um indica o início da fila (primeiro elemento) e o outro o fim da fila (último elemento). Vejamos uma fila implementada num vetor de 8 elementos (`v[0],v[1],....,v[7]`)

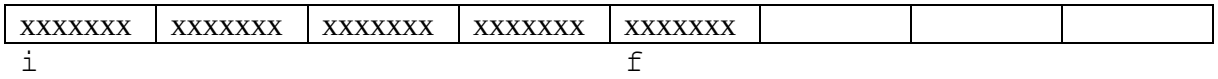
Fila vazia: `i=-1; f=-1;`



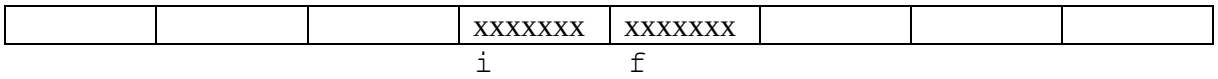
Inserir 2 elementos: `i=0; f=1;`



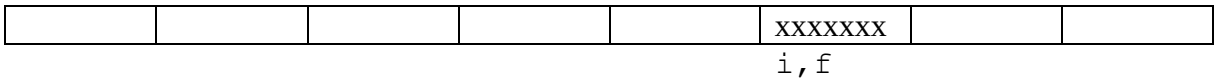
Inserir mais 3 elementos: `i=0; f=4;`



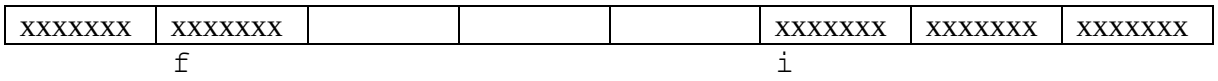
Remover 3 elementos: `i=3; f=4;`



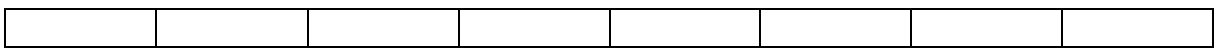
Inserir 1 elemento e remover 2: `i=5; f=5;`



Inserir 4 elementos: neste caso temos um problema, pois não cabem mais elementos no vetor e temos elementos vazios no início. Para resolver este problema, podemos fazer uma fila circular, isto é, fazer com que o último elemento do vetor seja seguido do primeiro. Ou seja, basta somar um (módulo o tamanho do vetor). Neste exemplo teríamos: `i=5; f=1;`



Remover 5 elementos: neste caso também temos um problema, pois a fila ficaria vazia e qual seria o valor de `i` e `f`? Se deixarmos `i=f=2`, por exemplo, não conseguiremos distinguir fila vazia de fila com um só elemento. É melhor neste caso forçar `i=f=-1`.



Agora vamos fazer os algoritmos numa fila de inteiros de MAX elementos:

```
#define MAX 100
```

```
int i, f;
int fila[MAX];

void inicia_fila() {
   i=f=-1;
}

int insere_fila(int x) {
    /* insere x no fim da fila */
    int prox;
    prox = (f+1) % MAX;
    if (prox == i) return -1; //não há mais lugar-fila cheia
    f = prox;
    fila[f] = x;
    // se a fila era vazia, altera o i
    if (i==-1) i=0;
    return 0;
}

int remove_fila(int *x) {
    /* remove elemento da fila em x */
    if (i == -1) return -1; /* fila vazia */
    *x = fila[i];
    /* verifica se fila vai ficar vazia */
    if (i == f) i=f=-1;
    else i=(i+1) % MAX;
    return 0;
}
```

Existem outras maneiras de implementar as funções para manipulação de filas. Uma delas que pode simplificar o tratamento dos casos particulares (fila vazia, fila cheia), é manter além dos indicadores `i` e `f`, um contador com o número de elementos da fila `n_elem`.

```
#define MAX 100
int i, f, n_elem;
int fila[MAX];

void inicia_fila() {
   i=f=-1;
   n_elem=0;
}
```

Fica com exercício, refazer as funções `insere_fila` e `remove_fila`.

O exemplo acima mostra uma fila de inteiros, mas podemos ter filas com elementos de quaisquer tipos inclusive de elementos do tipo `struct`. Para tanto mudaríamos apenas algumas declarações na implementação acima:

```
#define MAX 100
struct elemento {
   ....
}
int i, f;
struct elemento fila[MAX];

void inicia_fila() {
   i=f=-1;
}

int insere_fila(struct elemento x) {
    /* insere x no fim da fila */
    int prox;
    prox = (f+1) % MAX;
    if (prox == i) return -1; /* não há lugar-fila cheia */
    f = prox;
    fila[f] = x;
    /* se a fila era vazia, altera o i */
    if (i==-1) i=0;
    return 0;
}

int remove_fila(struct elemento *x) {
    /* remove elemento da fila em x */
    if (i == -1) return -1; // fila vazia
    *x = fila[i];
    /* verifica se fila vai ficar vazia */
    if (i == f) i=f=-1;
    else i=(i+1) % MAX;
    return 0;
}
```

Aplicações

Existem muitos algoritmos que precisam manipular estrutura de dados do tipo fila.
É usada principalmente para comunicação entre processos que possuem velocidades diferentes de geração de pedidos e atendimento de pedidos.
Por isso que não é comum o uso desta estrutura na programação seqüencial normal, pois se os pedidos são consumidos na mesma velocidade em que são gerados, não há necessidade de filas.
Por isso, esses algoritmos aparecem principalmente em comunicação entre processos e dentro do sistema operacional, que recebe requisições de vários programas ao mesmo tempo e tem que atendê-las sequencialmente.

Citamos alguns:

a) O controlador do disco é um processador dedicado para controlar o acesso ao disco. Vários usuários podem requisitar acesso a determinados setores do disco ao mesmo tempo. Como o atendimento a cada requisição demora alguns milisegundos, o controlador tem que gerenciar a fila de requisições dos usuários.

b) Um servidor de páginas web que atende um usuário de cada vez recebe ao mesmo tempo várias requisições de páginas de vários clientes da rede. Como só um cliente é atendido ao mesmo tempo, as várias requisições pendentes têm que permanecer numa fila. Neste caso, a fila é gerenciada pelo protocolo TCP que está em contato com a rede, recebendo as requisições dos usuários.

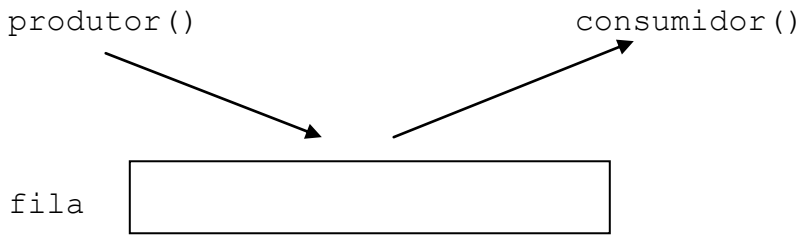
c) Uma parte importante do núcleo do sistema operacional é o escalonamento de processos. Os vários processos se alternam na execução, pois a CPU é uma só. O sistema operacional deve gerenciar a fila de processos prontos, esperando a disponibilidade da CPU. Neste caso na verdade além de usa posição na fila, existe a prioridade do processo. Isso é implementado pelo sistema operacional com uma fila de prioridades ou com várias filas, uma para cada prioridade.

d) A fila da impressora. Num ambiente em rede, a impressora é usada por vários usuários. Num determinado instante, vários arquivos podem ser enviados à impressora para serem impressos. A impressora tem que colocar esses arquivos numa fila, pois a velocidade de impressão é menor que a velocidade de recepção dos arquivos.

Os exemplos com filas são usados em geral quando o processamento é feito por processos assíncronos funcionando paralelamente. A fila regula a comunicação entre os processos. Existem os processos “produtores” de elementos para a fila e os processos “consumidores” de elementos da fila. A produção e consumo de elementos da fila ocorre em velocidades diferentes. Daí a necessidade da fila.

```
void produtor() {
  produza (msg);
  insere_fila(msg);
}

void consumidor() {
  /* se fila está vazia fica esperando */
  remove_fila(msg);
  consuma(msg);
}
```



Filas com prioridades

Muitas vezes uma estrutura de fila simples não atende. Ocorre quando os elementos da fila têm prioridades diferentes para serem atendidos. Isso ocorre também no caso das filas da vida real. Veja o caso da fila dos clientes no banco. Tem que haver atendimento especial para idosos, gestantes, clientes preferenciais, etc.

Há duas soluções:

a) Várias filas. Uma para cada prioridade. Naturalmente ao retirar um elemento, sempre procuramos na fila de maior prioridade. Quando ela está vazia, tenta-se a fila com prioridade seguinte e assim por diante.
b) Uma só fila, só que quando se coloca um elemento na fila, não se coloca no final e sim no lugar correspondente à sua prioridade. Isso não é bem uma fila, pois não estamos usando o princípio de inserir no final e retirar do início.

Como já devem ter notado, essas soluções podem fazer com que elementos com baixa prioridade demorarem muito para serem atendidos e talvez nem o sejam. Portanto a administração de filas de prioridades requer algo mais elaborado, mas esse é assunto para outros cursos.