BCC202 - Estruturas de Dados I

Aula 12: Filas

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: **silvap@ufop.edu.br**

2021

Conteúdo

Conteúdo

Tipo Abstrato de Dados com as seguintes características:

- O **primeiro** elemento a ser inserido é o **primeiro** a ser retirado/ removido
- FIFO - *First in, first out*
- TAD conhecida como queue.

Só podemos inserir um elemento no final da fila e só podemos retirar do início.

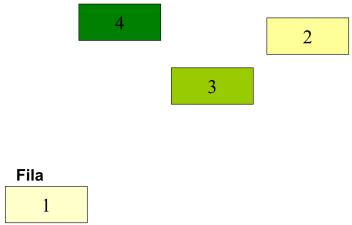
- Analogia natural com o conceito de fila que usamos no dia a dia.
  - Fila bancária.
  - Fila do cinema.
- Usos:
  - Sistemas Operacionais: Fila de impressão, Fila de processamento, etc.

Ilustração: enfileirar e desinfileirar

Fila

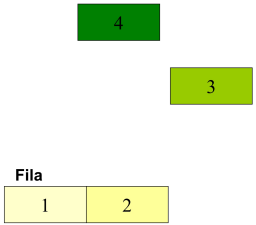
Fila vazia.

Ilustração: enfileirar e desinfileirar



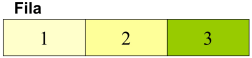
Enfileirou.

Ilustração: enfileirar e desinfileirar



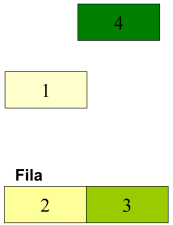
Enfileirou.

Ilustração: enfileirar e desinfileirar



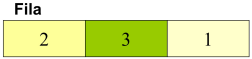
Enfileirou.

Ilustração: enfileirar e desinfileirar



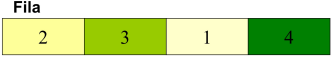
Desenfileirou.

Ilustração: enfileirar e desinfileirar



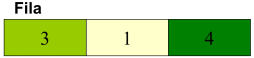
Enfileirou.

Ilustração: enfileirar e desinfileirar



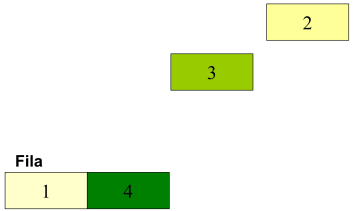
Enfileirou.

Ilustração: enfileirar e desinfileirar



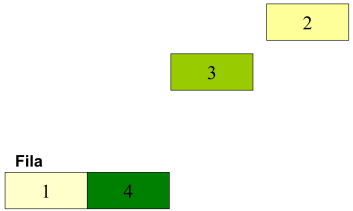
Desenfileirou.

Ilustração: enfileirar e desinfileirar



Desenfileirou.

Ilustração: enfileirar e desinfileirar



Fila nada mais é do que uma **Lista** com uma restrição:

O **primeiro** elemento a ser inserido é o **primeiro** a ser retirado.

Exemplos

Implementação de *(i)* fila de impressão; *(ii)* fila de processamento.

Conteúdo

fila.h

```
1  #ifndef fila_h
2  #define fila_h
3
4  #include <stdio.h>
5
6  /*definindo um novo tipo*/
7  typedef struct { ... } TFila;
8
9  void TFila_Inicia(TFila **);
10 void TFila_EhVazia(TFila*);
11 void TFila_Enfileira(TFila*, int);/*insere no final*/
12 int TFila_Desenfileira(TFila*);/*retira do início */
13 void TFila_Libera(TFila**);
14
15 #endif /* fila_h */
```

Existem várias opções de estruturas de dados que podem ser usadas para representar filas. As duas representações mais utilizadas são: vetor e ponteiros.

Conteúdo

Vetor

Estratégia simplista: inserção de novos elementos no final do vetor. Retirada de elementos no início do vetor.

Inserção e Remoção: Extremidades Opostas

- A operação `Enfileira` faz a parte de trás da fila expandir-se.
- A operação `Desenfileira` faz a parte da frente da fila contrair-se.
- A fila tende a caminhar pela memória do computador, ocupando espaço na parte de trás e descartando espaço na parte da frente.

Quais as desvantagens dessa estratégia?

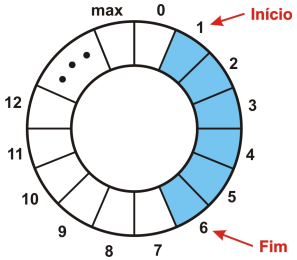
Com poucas inserções e retiradas, a fila vai ao encontro do limite do espaço da memória alocado para ela.

Quantidade de elementos

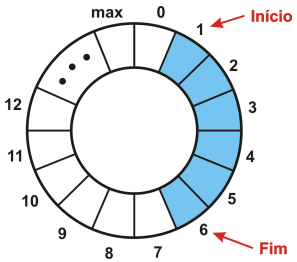
- a lista pode ter um limite conhecido (vetor estático): é necessário verificar a sua capacidade a cada inserção.
- a lista pode usar um vetor dinâmico
  (dimensão atualizada – uso do *realloc* – sempre que necessário).

Fila Circular

- **Dica**: imaginar o *array* como um círculo. A primeira posição segue a última.
- Por essa característica, a fila é denominada Fila Circular.

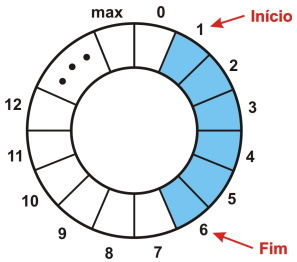


Fila Circular



- A fila ocupa posições contíguas de memória, delimitada pelos índices **Início** e **Fim**.
  - `Início` indica a posição do primeiro elemento
  - `Fim` a primeira posição vazia (posição após o último elemento)

Fila Circular



- Para `enfileirar`, basta mover o índice `Fim` uma posição no sentido horário.
- Para `desenfileirar`, basta mover o índice `Início` uma posição no sentido horário.

Fila Circular

Fila após inserção de quatro elementos

0	1	2	3	4	5	···
1.4	2.2	3.5	4.0			
↑				↑		
ini				*fim*		

Para essa implementação, os índices do vetor são incrementados de maneira que seus valores progridam "circularmente".

Fila Circular

Fila após retirada de dois elementos

0	1	2	3	4	5	⋯
1.4	2.2	3.5	4.0			
		↑		↑		
		ini		*fim*		

De modo geral:

- $ini = ini\%dim$, onde *dim* é a dimensão da fila.
- $fim = (ini + n)\%dim$, onde n é quantidade de elementos da fila e *dim* sua dimensão.

Struct Fila utilizando Vetores

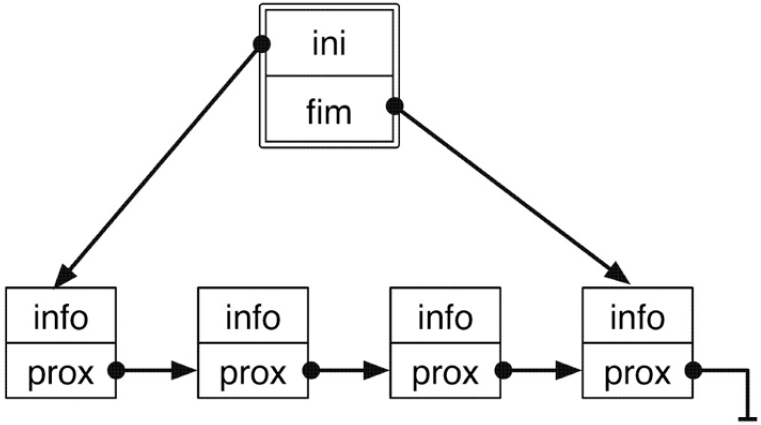
```c
/*alt: alocar vet dinamicamente e realocar sempre que preciso. */
#define MAXTAM 1000

typedef struct {
  int n;   /* número de elementos armazenados */
  int ini; /* índice do início da fila */
  int vet[MAXTAM];  /* vetor de elementos */
} TFila;

// implementação das operações de fila para manipular esses dados.
```

Conteúdo

Filas usando ponteiros

A implementação será similar à lista encadeada com CABEÇA.

- Elementos serão inseridos e retirados de extremidades opostas da fila.
- Há uma célula cabeça para facilitar a implementação das operações Enfileira e Desenfileira quando a fila está vazia.
- Serão mantidos dois ponteiros: ini e fim da fila:
  - respectivamente, primeiro e último elemento da fila.
- Quando a fila está vazia, os apontadores primeiro e último apontam para a célula cabeça.
- Para enfileirar um novo item, basta criar uma célula nova, ligá-la após a célula que contém x_n e colocar nela o novo item.
- Para desenfileirar o item x_1, basta "desligar" a célula após a cabeça da lista.



E se mantivéssemos somente um ponteiro para a primeira posição, tal como implementamos a lista?

A especificação, é a mesma descrita para a implementação utilizando vetor.

fila_ponteiro.h

```
#ifndef fila_h
#define fila_h
#include <stdio.h>
/* Decisão de implementação,
    poderia ser vetor.*/
typedef struct {
    int chave;
} TItem;

typedef struct Celula {
    TItem item;
    struct Celula *prox;
} TCelula;

typedef struct {
    TCelula *cabeca;
    TCelula *fim;
} TFila;
...
```

```
...

void TFila_Inicia(Fila**);

void TFila_EhVazia(Fila*);

/*insere no final*/
void TFila_Enfileira(Fila*, int);

/*retira do início */
int TFila_Desenfileira(Fila*);

void TFila_Libera(Fila**);

#endif /* fila_h */
```

O mesmo que a *Lista*: nós encadeados, contendo informações e ponteiro para o próximo nó.

fila_ponteiro.c

```
1  void TFila_Inicia(TFila *f) {
2      f->cabeca = (TCelula*) malloc(sizeof(TCelula));
3      f->fim = f->cabeca;
4  }
5
6  int TFila_EhVazia(TFila *f) {
7      return f->cabeca == f->fim;
8  }
9
10 void TFila_Enfileira(TFila *f, TItem item) {
11     TCelula *aux = (TCelula*) malloc(sizeof(TCelula));
12     aux->item = item;
13     aux->prox = NULL;
14     f->fim->prox = aux;
15     f->fim = aux;
16 }
```

fila_ponteiro.c

```
1  int TFila_Desenfileira(TFila *f, TItem *item) {
2      if (TFila_EhVazia(f))
3          return 0;
4      TCelula *aux = f->cabeca->prox;
5      f->cabeca->prox = f->cabeca->prox->prox;
6      *item = aux->item;
7      free(aux);
8      return 1;
9  }
10
11 void TFila_Limpa(TFila *f) {
12     TItem t;
13     while (TFila_Desenfileira(f, &t));
14 }
```

Conteúdo

- Fila: Tipo Abstrato de Dado.
  - Vetor.
  - Ponteiro.
- Protocolo bem definido: (First In, First Out - FIFO).
- Qual a complexidade de tempo para enfileirar e para desinfileirar?

Algoritmos de ordenação.

Conteúdo

Bibliografia

Os conteúdos deste material, incluindo figuras, textos e códigos, foram extraídos ou adaptados do livro-texto indicado a seguir:


Celes, Waldemar and Cerqueira, Renato and Rangel, José
Introdução a Estruturas de Dados com Técnicas de Programação em C.
Elsevier Brasil, 2016.
ISBN 978-85-352-8345-7.

Conteúdo

Exercício

Seja TLista o TAD Lista que implementa as seguintes funções:

```
1  /* faz com que a lista fique vazia */
2  void TLista_Inicia(TLista *pLista);
3
4  /* insere pItem na posicao pos da lista */
5  void Lista_Insere(TLista *pLista, TItem* pItem, int pos);
6
7  /* retira o item  na posicao pos da lista e retorna em pItem */
8  int Lista_Retira(TLista* pLista, TItem* pItem, int pos);
9
10 int Lista_Tamanho(Tlista *lista);
```

Implemente as operações abaixo da TAD TFila utilizando TLista. Esta implementação é eficiente? Justifique sua resposta.

```
1  void TFila_Inicia(Fila**);
2  void TFila_Enfileira(Fila*, int);
3  int TFila_Desenfileira(Fila*);
```