Universidade Federal de Ouro Preto
Campus João Monlevade

CSI 488 – Algoritmos e Estruturas de Dados I

TAD – Filas

Prof. Mateus Ferreira Satler

Índice

1. Introdução

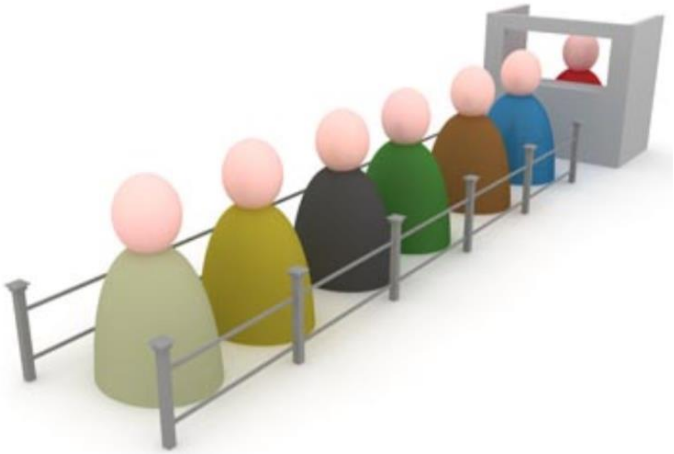
- O que é uma **Fila** em nosso cotidiano?
  - Alinhamento de uma série de indivíduos ou objetos em sequência, de modo que um esteja imediatamente atrás do outro.
- Funcionamento de uma fila.
- **Fila** e **Lista** são a mesma coisa?
  - Faça uma lista de pessoas.
  - Faça uma fila de pessoas.

1. Introdução

- A **Fila** é uma estrutura de dados bastante usada em computação.
  - Admite remoção e inserção de elementos.
  - Os acessos aos elementos também seguem uma regra.
  - A estrutura de fila é uma analogia natural com o conceito de fila que usamos no nosso dia a dia.

1. Introdução

- Uma **Fila** é um conjunto de itens a partir do qual podem-se eliminar itens numa extremidade (chamada início da fila) e no qual podem-se inserir itens na outra extremidade (chamada final da fila).



1. Introdução

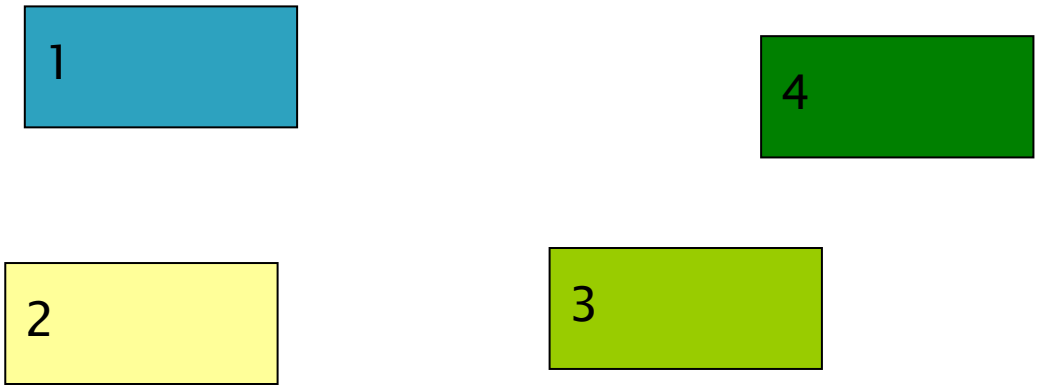
- Filas são casos especiais de listas.
  - Nas listas, quando precisávamos criar um novo elemento, poderíamos inseri-lo ou removê-lo **de qualquer posição da lista.**
  - Exemplos:
    - Na primeira posição;
    - Na última posição; ou
    - Em qualquer parte no meio da lista.

1. Introdução

- Numa fila existe uma regra básica a ser seguida:
  - **Primeiro a chegar é o primeiro a sair**
  - Do inglês: ***FIFO*** – *First In, First Out*
- Um novo elemento da fila somente pode ser inserido na última posição (fim da fila).
- Um elemento só pode ser removido da primeira posição (inicio da fila).

1. Introdução

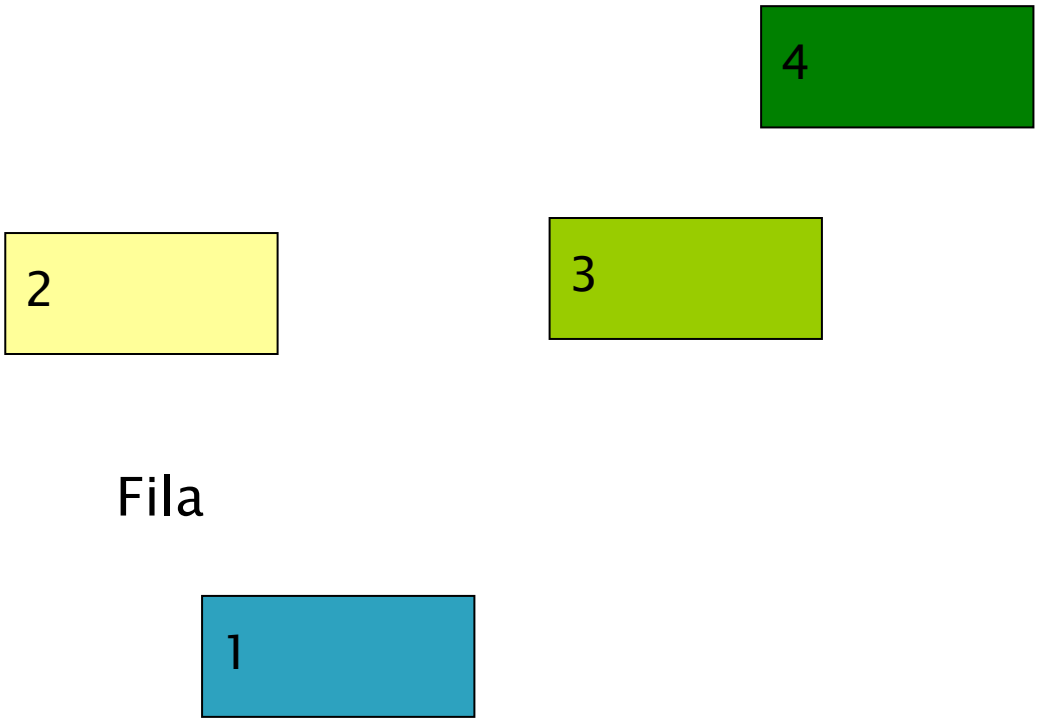
- **Funcionamento da Fila:** Fila vazia



Fila

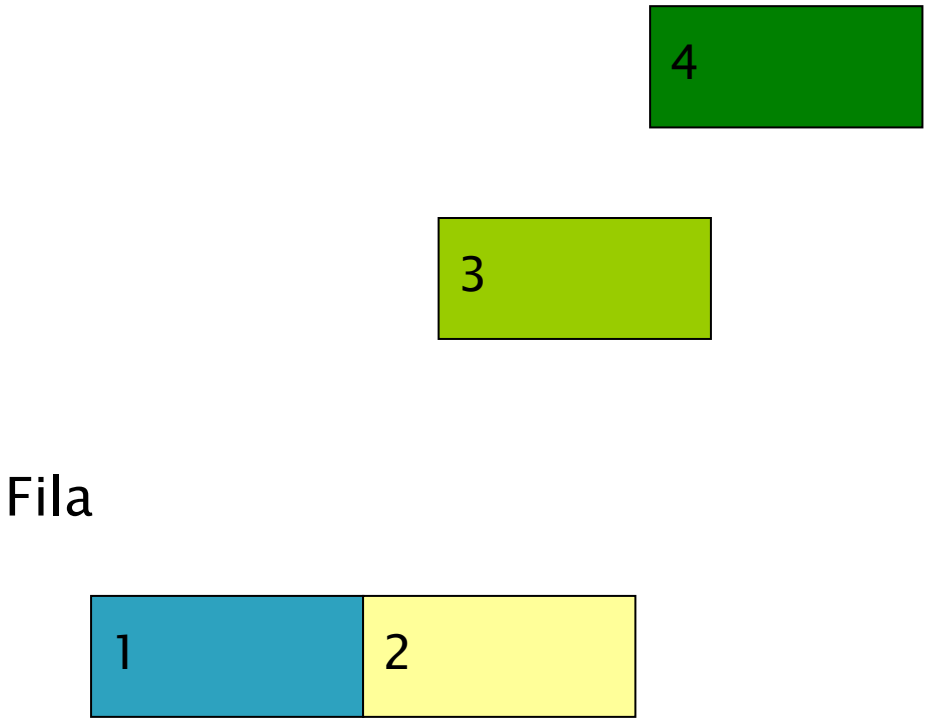
1. Introdução

- **Funcionamento da Fila:** Enfileirou



1. Introdução

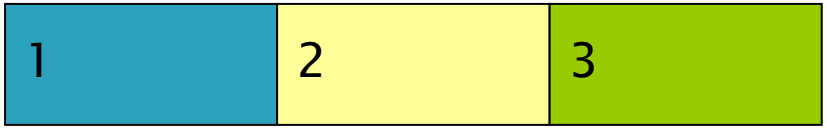
- **Funcionamento da Fila:** Enfileirou



1. Introdução

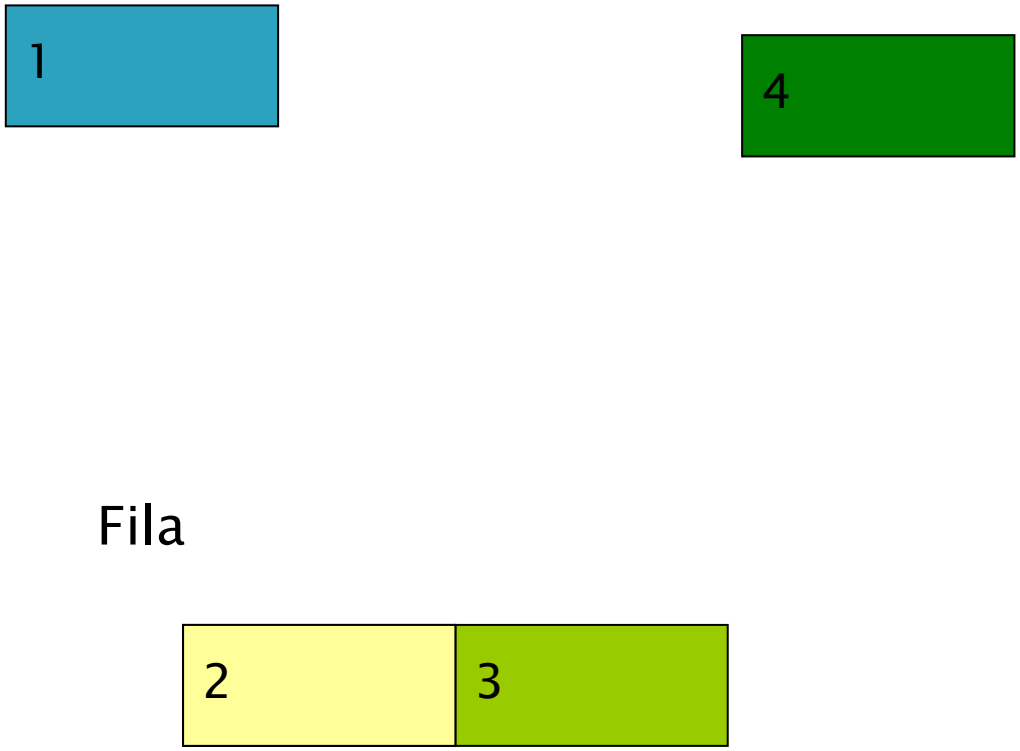
- **Funcionamento da Fila:** Enfileirou

Fila



1. Introdução

- **Funcionamento da Fila:** Desenfileirou



1. Introdução

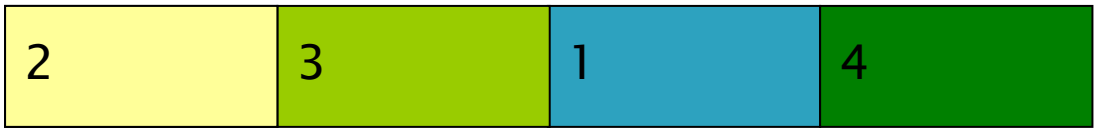
- **Funcionamento da Fila:** Enfileirou

Fila

1. Introdução

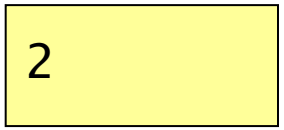
- **Funcionamento da Fila:** Enfileirou

Fila



1. Introdução

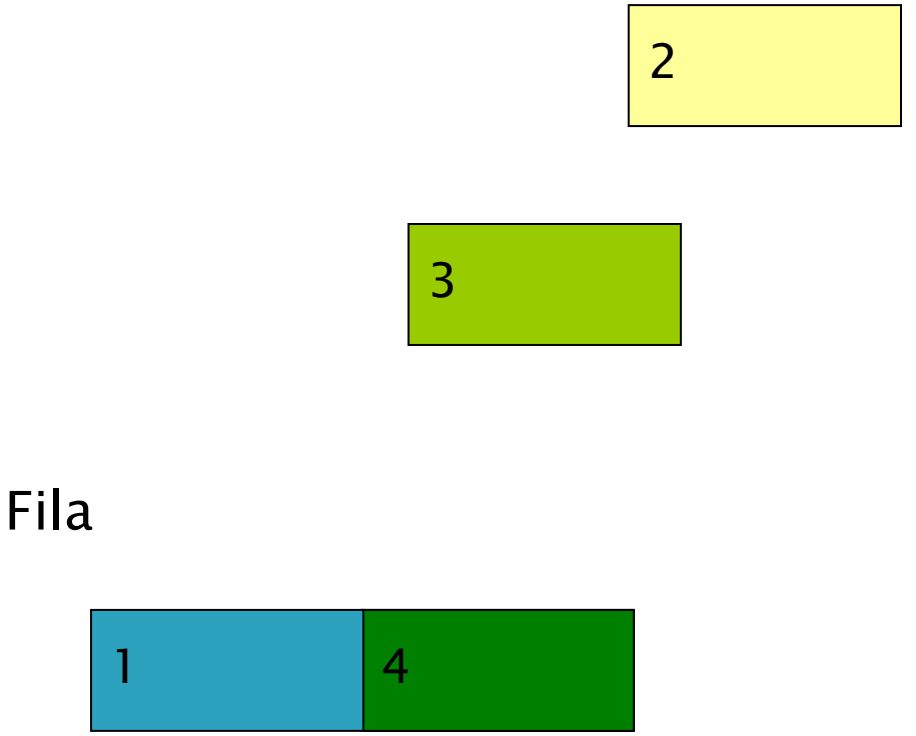
- **Funcionamento da Fila:** Desenfileirou



Fila

1. Introdução

- **Funcionamento da Fila:** Desenfileirou



1. Introdução

- Exemplos de uso de filas na computação:
  - **Filas de impressão**: Impressoras tem uma fila, caso vários documentos sejam impressos, por um ou mais usuários, os primeiros documentos impressos serão de quem enviar primeiro.
  - **Filas de processos**: Vários programas podem estar sendo executados pelo sistema operacional. O mesmo tem uma fila que indica a ordem de qual será executado primeiro.
  - **Filas de tarefas**: Um programa pode ter um conjunto de dados para processar. Estes dados podem estar dispostos em uma fila, onde o que foi inserido primeiro, será atendido primeiro.

2. TAD Fila

- O que o TAD Fila deveria conter?
  - Representação do tipo da fila.
  - Conjunto de operações que atuam sobre a fila.
- Quais operações deveriam fazer parte da fila?
  - Depende de cada aplicação.
  - Mas, um conjunto padrão pode ser definido.

2. TAD Fila

- Operações necessárias à grande maioria das aplicações:

FFVazia (Fila): faz a fila ficar vazia.

Fvazia (Fila): esta função retorna ***true*** se a fila está vazia; senão retorna ***false***.

Enfileira (Fila, x): insere o item x no final da fila.

Desenfileira (Fila, x): retorna o item x no início da fila, retirando-o da fila.

2. TAD Fila

- Existem várias opções de estruturas de dados que podem ser usadas para representar filas.

- As duas representações mais utilizadas são:

  - Implementação por **arrays** (vetores).

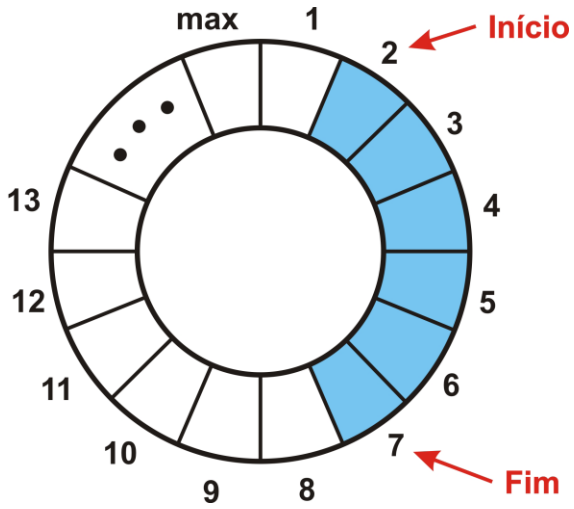
  - Implementação por **ponteiros**.

3. Implementação por Array

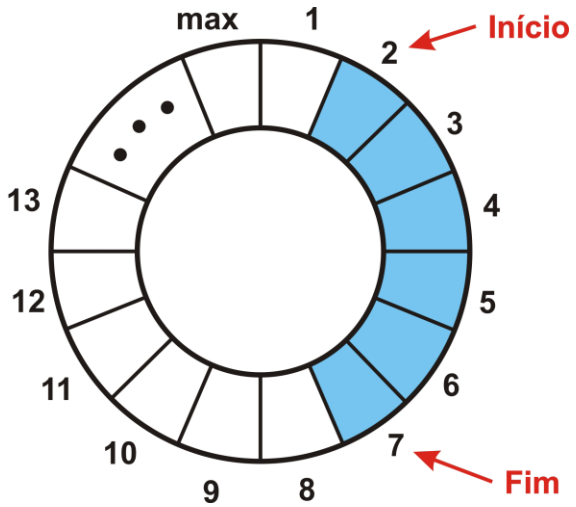
- Os itens são armazenados em posições contíguas de memória.
- A operação **`Enfileira`** faz a parte de trás da fila expandir-se.
- A operação **`Desenfileira`** faz a parte da frente da fila contrair-se.
- A fila tende a caminhar pela memória do computador, ocupando espaço na parte de trás e descartando espaço na parte da frente.

3. Implementação por Array

- Com poucas inserções e retiradas, a fila vai ao encontro do limite do espaço da memória alocado para ela.
- **Solução**: imaginar o array como um círculo. A primeira posição segue a última.
- Por esta característica, a fila é denominada **Fila Circular**.

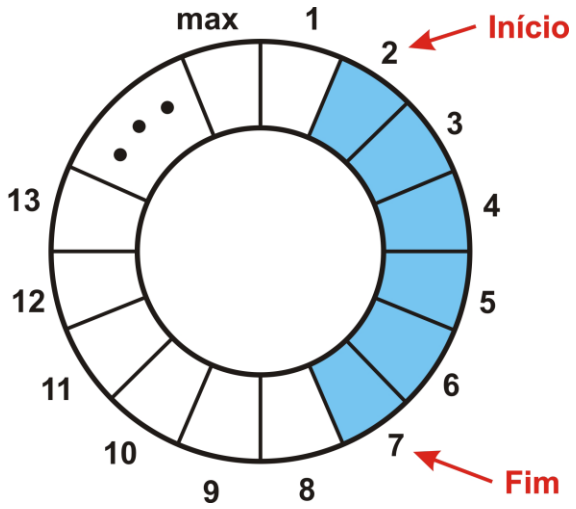


3. Implementação por Array



- A fila se encontra em posições contíguas de memória, em alguma posição do círculo, delimitada pelos apontadores **`Início`** e **`Fim`**.
  - **`Início`** indica a posição do primeiro elemento
  - **`Fim`** a primeira posição vazia (posição após o último elemento)

3. Implementação por Array



- Para **`enfileirar`**, basta mover o apontador **`Fim`** uma posição no sentido horário.

- Para **`desenfileirar`**, basta mover o apontador **`Início`** uma posição no sentido horário.

3. Implementação por Array

- Estrutura:

```
#define MAXTAM 1000

typedef int Tchave;

typedef struct {
  TChave Chave;
  /* outros componentes */
} TItem;

typedef struct {
  TItem vItem[MAXTAM+1];
  int iFrente, iTras;
} TFila;
```

3. Implementação por Array

- Nos casos de fila cheia e fila vazia, os apontadores **Frente** e **Trás** apontam para a mesma posição do círculo.
- Uma saída para distinguir as duas situações é deixar uma posição vazia no array.
- Neste caso, a fila está cheia quando **Trás+1** for igual a **Frente**:

```
void FFVazia (TFila* pFila) {
  pFila->iFrente = 0;
  pFila->iTras = pFila->iFrente; }

int Fvazia (TFila* pFila) {
  return (pFila->iFrente == pFila->iTras); }
```

3. Implementação por Array

```
int Enfileira (TFila* pFila, TItem* pItem) {

  if(((pFila->iTras+1)%(MaxTam+1)) == pFila->pFrente)
    return 0; /* fila cheia */

  pFila->vItem[pFila->iTras] = *pItem;
  pFila->iTras = (pFila->iTras+1) % (MaxTam+1);

  //Alternativa
  //if (pFila->iTras == MaxTam) pFila->iTras = 0;
  //else pFila->iTras++;

  return 1;
}
```

3. Implementação por Array

```
int Desenfileira (TFila* pFila, TItem* pItem) {

  if (FVazia(pFila))
    return 0;

  *pItem = pFila->vItem[pFila->iFrente];
  pFila->iFrente = (pFila->iFrente+1) % (MaxTam+1);

  //Alternativa
  //if (pFila->iFrente == MaxTam) pFila->iFrente = 0;
  //else pFila->iFrente++;

  return 1;
}
```

4. Implementação por Ponteiro

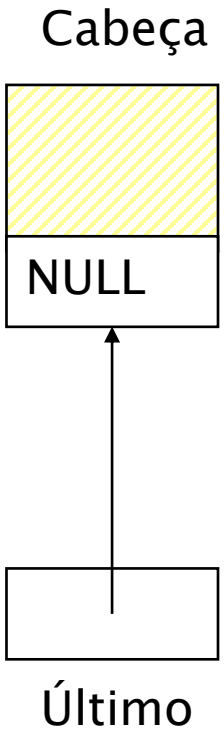
- A fila é implementada por meio de células.
- Cada célula contém um item da fila e um apontador para outra célula.
- Há uma célula **`cabeça`** para facilitar a implementação das operações **`Enfileira`** e **`Desenfileira`** quando a fila está vazia.

4. Implementação por Ponteiro

- A fila contém um apontador para o início da fila (célula **Cabeça**) e um apontador para a parte de trás da fila (**Fim**).
- Quando a fila está vazia, os apontadores **Cabeça** e **Fim** 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

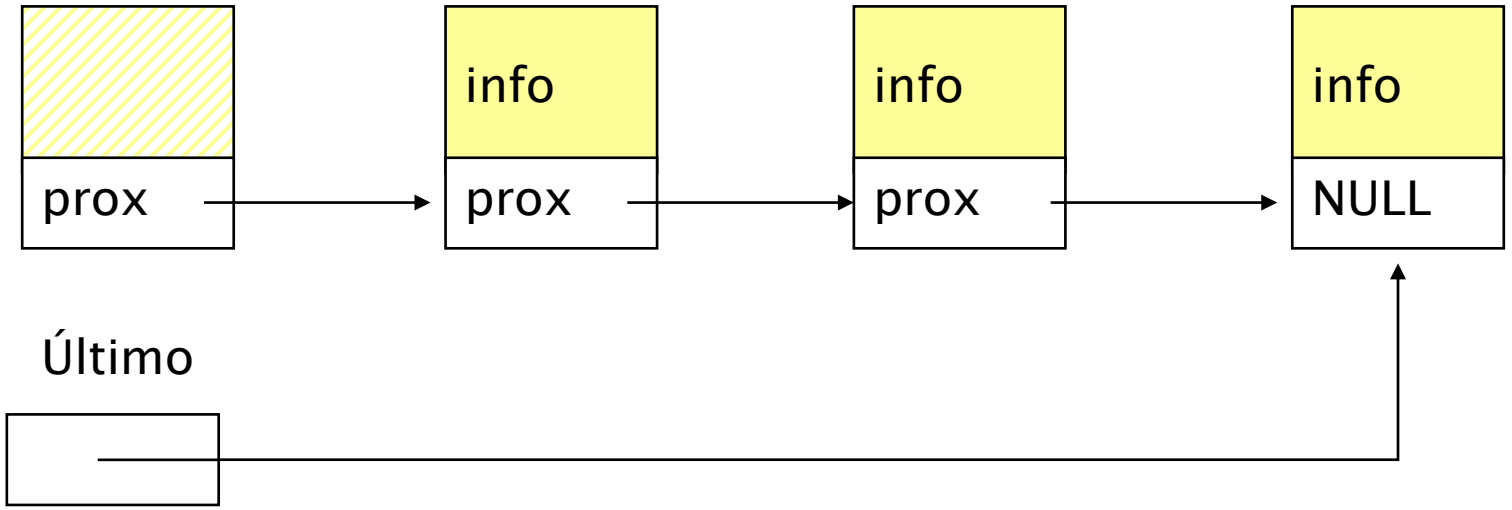
4. Implementação por Ponteiro

- Cria Fila Vazia



4. Implementação por Ponteiro

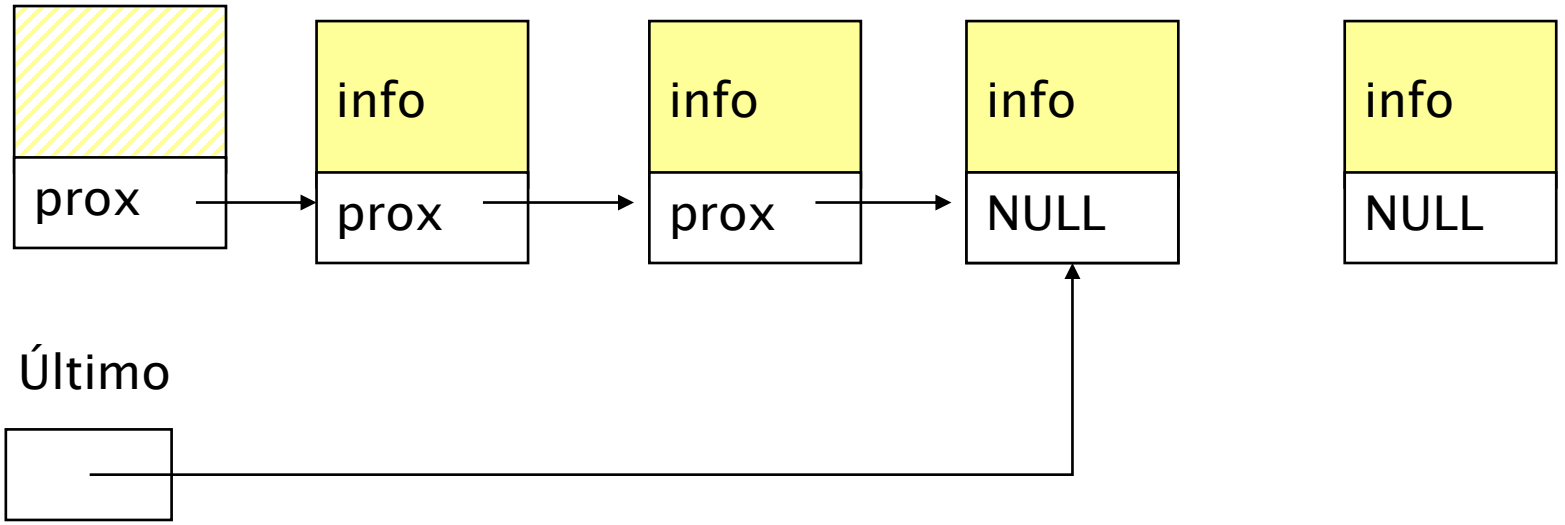
- Opção única de posição onde se pode **inserir**:



  - **Final** da fila, ou seja, **última** posição.

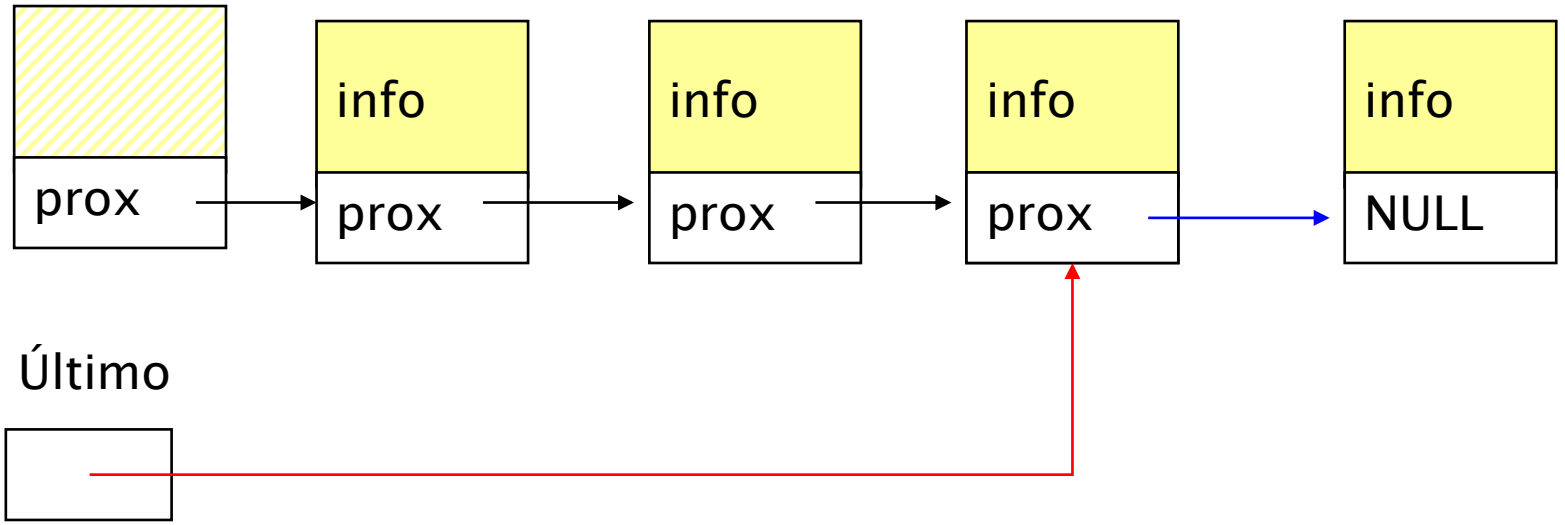
4. Implementação por Ponteiro

- **Inserir** na **última posição (1/3)**



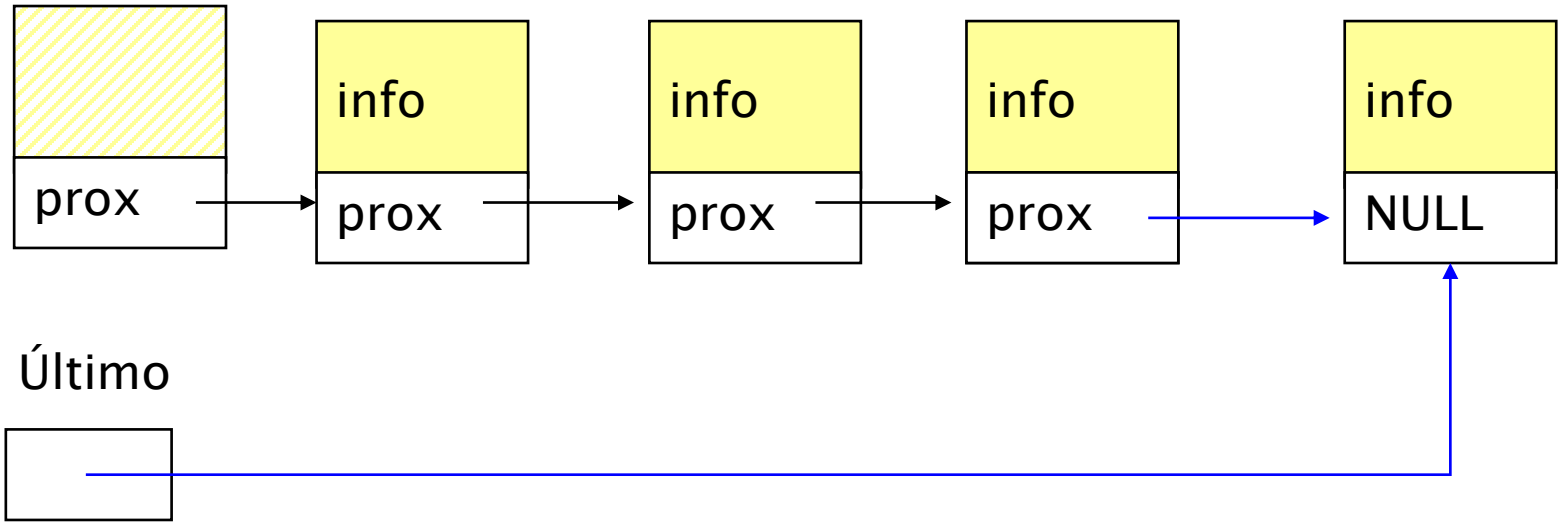
4. Implementação por Ponteiro

- **Inserir** na **última posição (2/3)**



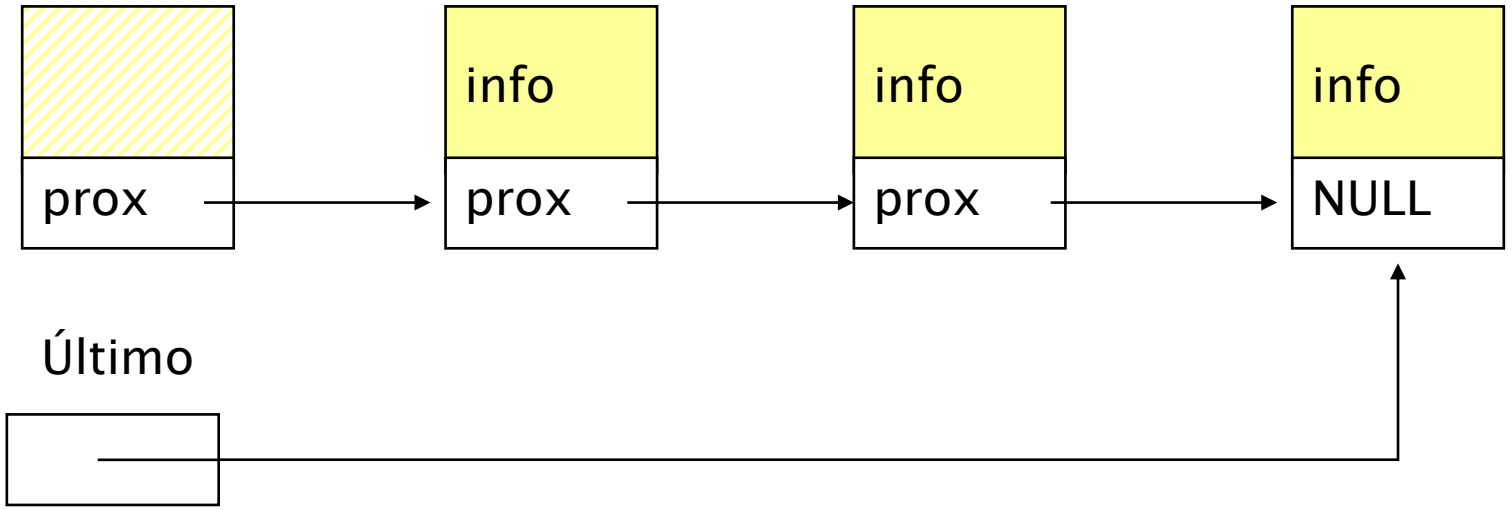
4. Implementação por Ponteiro

- **Inserir** na **última posição (3/3)**



4. Implementação por Ponteiro

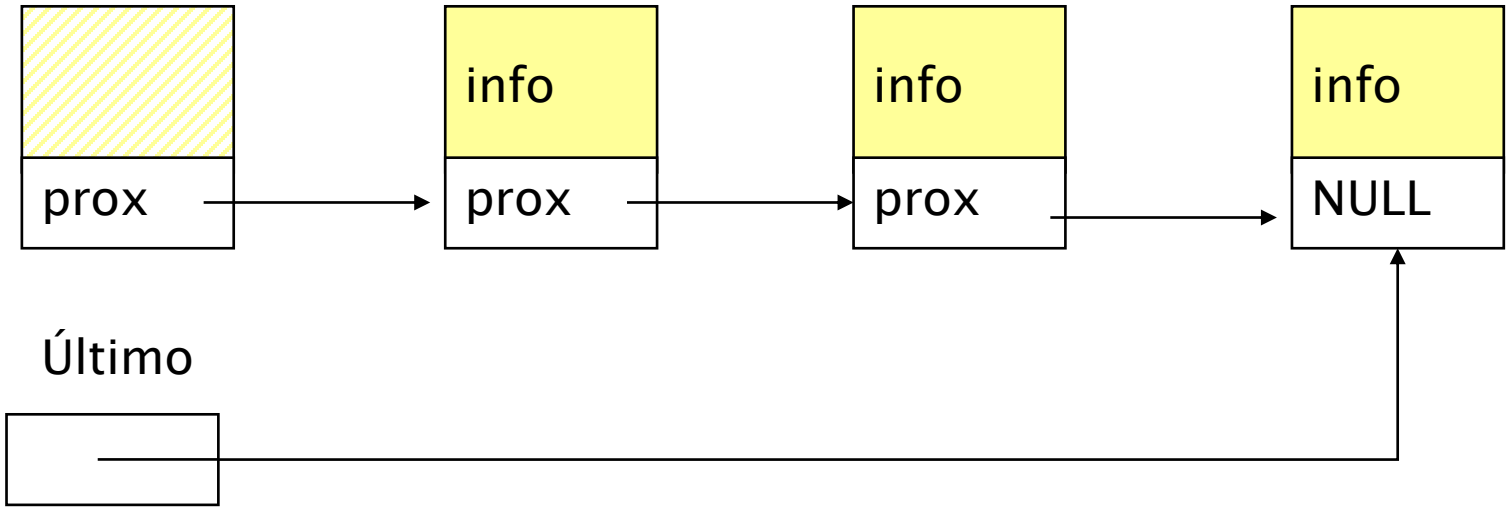
- Opção única de posição onde se pode **retirar**:



  - **Início** da fila, ou seja, **primeira** posição.

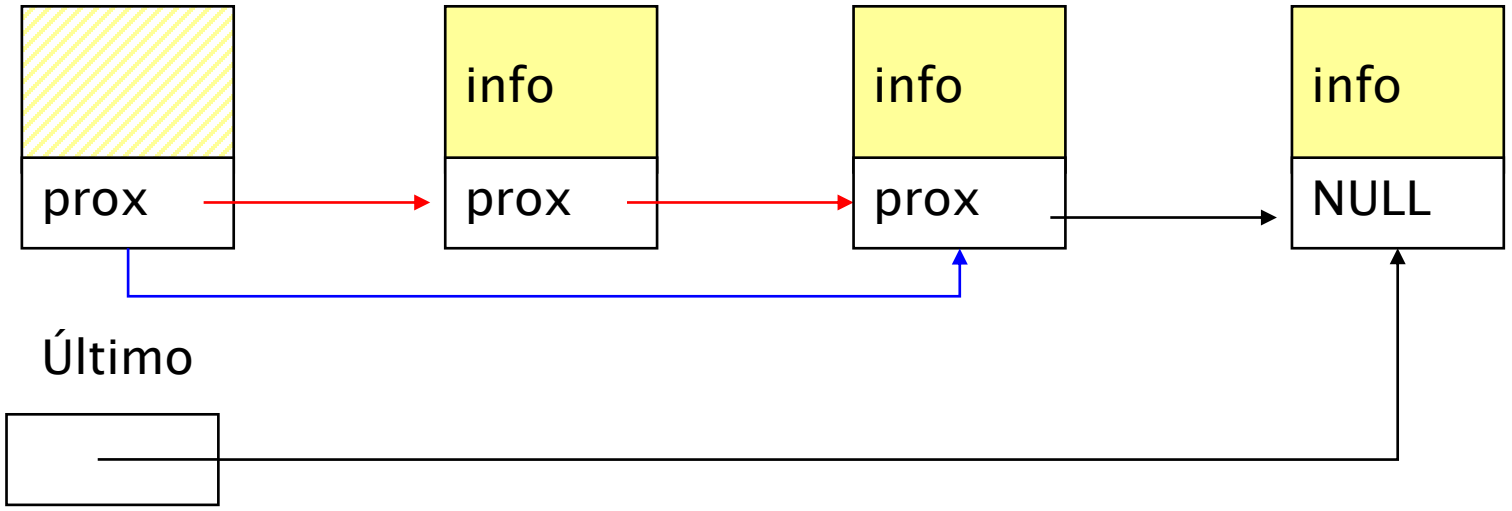
4. Implementação por Ponteiro

- **Retirar** da **1ª posição (1/3)**



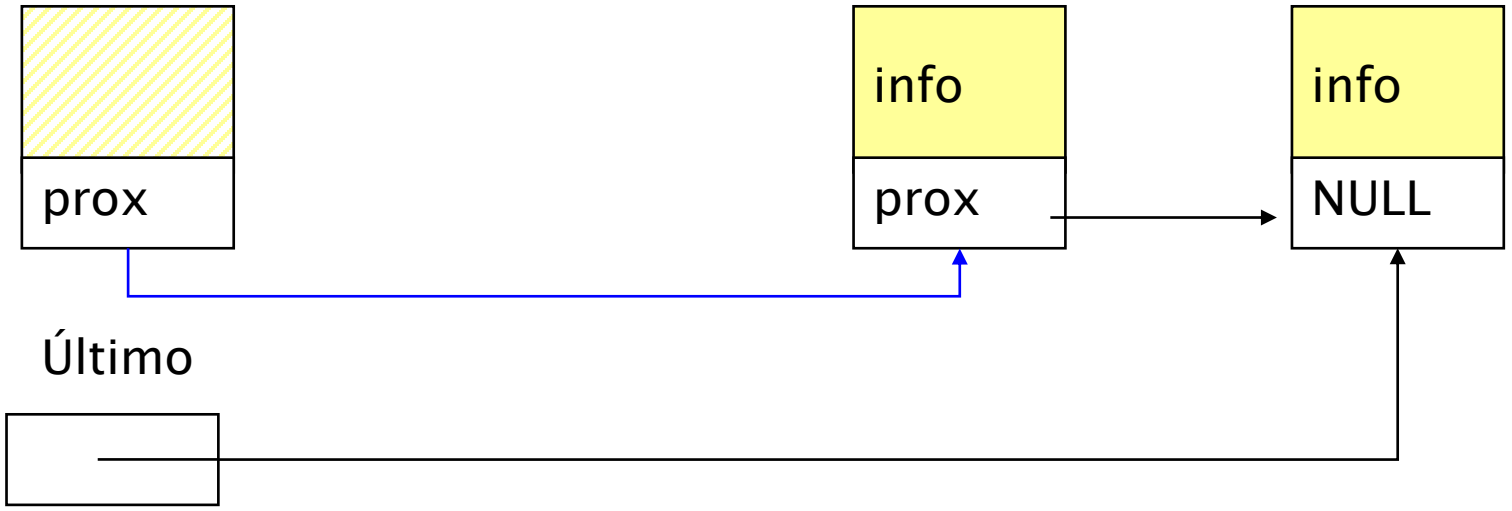
4. Implementação por Ponteiro

- **Retirar** da 1ª posição (2/3)



4. Implementação por Ponteiro

- **Retirar** da 1ª posição (3/3)



4.1. Estrutura da Fila

```
typedef int Tchave;

typedef struct TItemEst {
  TChave Chave;
  /* outros componentes */
} TItem;

typedef struct TCelulaEst {
  TItem item;
  struct TCelulaEst* pProx;
} TCelula;

typedef struct TfilaEst {
  Celula* pFrente;
  Celula* pTras;
} TFila;
```

4.2. Operações com Fila

- **Com cabeça**

```
void FFVazia (TFila* pFila) {
  pFila->pFrente = (Celula*)malloc(sizeof(TCelula));
  pFila->pTras = pFila->pFrente;
  pFila->pFrente->pProx = NULL;
}

int Fvazia (TFila* pFila) {
  return (pFila->pFrente == pFila->pTras);
}
```

4.2. Operações com Fila

- **Sem cabeça**

```
void FFVazia (TFila* pFila) {
  pFila->pFrente = NULL;
  pFila->pTras = pFila->pFrente;
}

int Fvazia (TFila* pFila) {
  return (pFila->pFrente == NULL);
}
```

4.2. Operações com Fila

- Com cabeça

```
int Enfileira (TFila *pFila, TItem* pItem) {
  Celula* pNovo;
  pNovo = (Celula*) malloc(sizeof(TCelula));

  if(pNovo == NULL) return 0;

  pFila->pTras->pProx = pNovo;
  pFila->pTras = pNovo;
  pNovo->item = *pItem;
  pNovo->pProx = NULL;

  return 1;
}
```

4.2. Operações com Fila

- **Sem cabeça**

```
int Enfileira (TFila *pFila, TItem* pItem) {
  Celula* pNovo;
  pNovo = (Celula*) malloc(sizeof(TCelula));

  if(pNovo == NULL) return 0;
  if(pFila->pTras != NULL)
    pFila->pTras->pProx = pNovo;
  pFila->pTras = pNovo;
  pNovo->item = *pItem;
  pNovo->pProx = NULL;
  if(pFila->pFrente == NULL)
    pFila->pFrente = pNovo;
  return 1;
}
```

4.2. Operações com Fila

- **Com cabeça**

```
int Desenfileira (TFila* pFila, TItem* pItem) {
  Celula* pAux;

  if(FVazia(pFila)) return 0;

  pAux = pFila->pFrente;
  pFila->pFrente = pFila->pFrente->pProx;
  *pItem = pFila->pFrente->item;
  free(pAux);

  return 1;
}
```

4.2. Operações com Fila

- **Sem cabeça**

```
int Desenfileira (TFila* pFila, TItem* pItem) {
  Celula* pVelho;

  if(FVazia(pFila)) return 0;

  pVelho = pFila->pFrente;
  pFila->pFrente = pFila->pFrente->pProx;
  *pItem = pVelho->item;
  free(pVelho);

  if(pFila->pFrente == NULL)
    pFila->pTras = NULL;

  return 1;
}
```

5. Referências

- Material de aula dos Profs. Luiz Chaimowicz e Raquel O. Prates, da UFMG: https://homepages.dcc.ufmg.br/~glpappa/aeds2/AEDS2.1%20Conceitos%20Basicos%20TAD.pdf
- DEITEL, P; DEITEL, H. *C How to Program*. 6a Ed. Pearson, 2010.
- LANGSAM,Y.; AUGENSTEIN, M.J.; TENENBAUM, A.M. Data Structures using C and C++, 2a edição . Prentice Hall of India. 2007.
- CORMEM, T. H.; et al. Introduction to algorithms, 3a edição, The MIT Press.
- DROZDEK A. Estrutura de dados e algoritmos em C++,1a edição Cengage Learning.