Algoritmos e Estrutura de Dados II

Prof. Fellipe Guilherme Rey de Souza

Aula 05 – Fila (Implementação)

Agenda

- Possíveis implementações
- Fila com vetor
- Fila com alocação dinâmica

Possíveis Implementações

- Existem duas principais formas de implementarmos uma Fila:
  - Usando um vetor (fila circular) para armazenar os valores
  - Usando alocação dinâmica de memória para armazenar os valores

- A seguir, veremos um pouco mais das duas implementações, incluindo os seus prós e contras.

Possíveis Implementações

- Importante salientar que a Fila é um **tipo abstrato de dado**. Isso quer dizer que existe mais de uma implementação possível para a Fila.

- Os conteúdos que serão abordados na aula de hoje mostram **uma possível implementação** de Fila, seja utilizando vetores ou seja utilizando alocação dinâmica de memória.

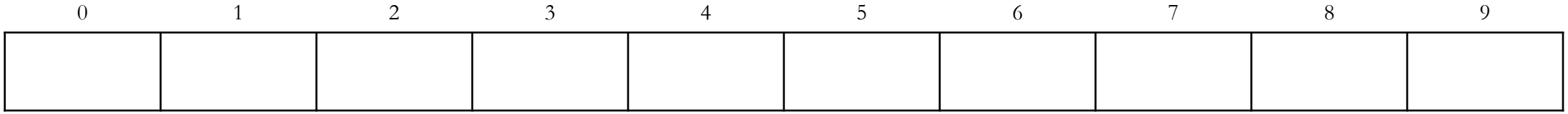
Fila com Vetores

- A implementação da Fila com vetores utiliza-se da estrutura já existente das linguagens de programação (vetor/array) para armazenar seus valores.

- A implementação é simples: Basta usar um vetor (do tipo que for a Fila) e criar três novas variáveis adicionais: **comeco**, **fim** e **tamanho**.

Filas com Vetores

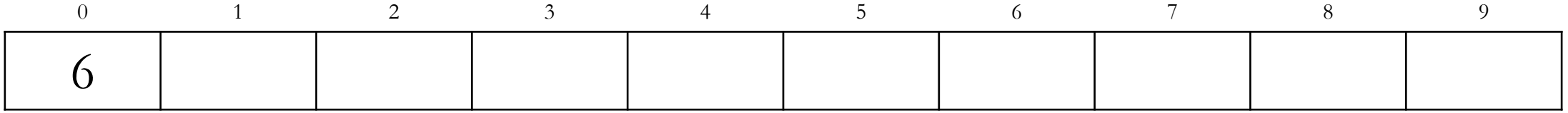
t_0



Começo: 0 / **Fim**: -1 / **Tamanho**: 0

Ação: Enfileirar o item "6"

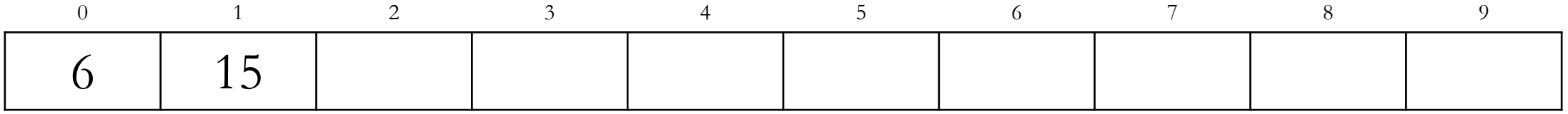
t_1



Começo: 0 / **Fim**: 0 / **Tamanho**: 1

Ação: Enfileirar o item "15"

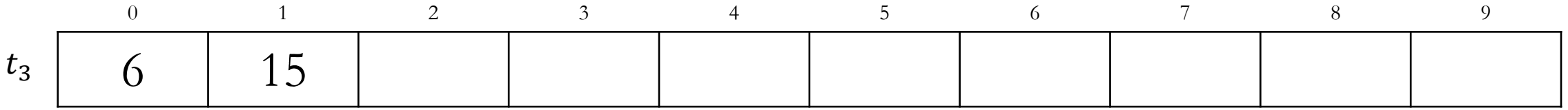
t_2



Começo: 0 / **Fim**: 1 / **Tamanho**: 2

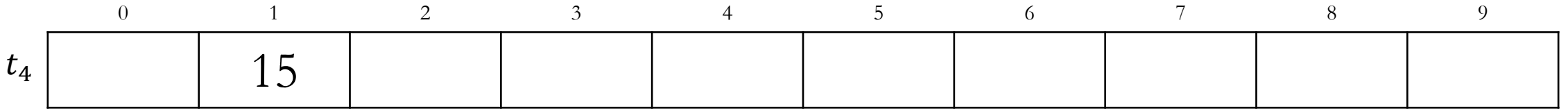
Filas com Vetores

Ação: Ver o primeiro (Retorna o "6" sem mexer na Fila)



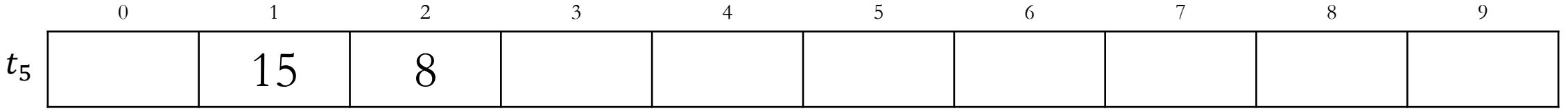
Começo: 0 / **Fim**: 1 / **Tamanho**: 2

Ação: Desenfileirar



Começo: 1 / **Fim**: 1 / **Tamanho**: 1

Ação: Enfileirar o item "8"



Começo: 1 / **Fim**: 2 / **Tamanho**: 2

Filas com Vetores

```c
#include<stdio.h>
#define TAM 8

typedef struct _Pessoa {
    char nome[20];
    int idade;
} Pessoa;

int main() {
    Pessoa fila[TAM];
    int comeco = 0;
    int fim = -1;
    int tamanho = 0;
```

- Definição da Fila
  - Defini um tamanho máximo para a Fila (neste exemplo, o tamanho é 8)
  - Criei uma Fila do tipo Pessoa (poderia ser de qualquer outro tipo)
  - Iniciei o começo com 0 (também poderia ser -1 com algumas alterações na implementação), o fim com -1 e o tamanho com 0.

Filas com Vetores

- Enfileirar

```c
//Enqueue
void insereNaFila(Pessoa *fila, Pessoa pessoa, int *fim, int *tamanho) {
    if (*tamanho == TAM) {
        printf("Fila Cheia! \n");
    } else {
        *fim = (*fim == TAM-1) ? 0 : *fim + 1;
        fila[*fim] = pessoa;
        *tamanho = *tamanho + 1;
    }
}
```

  - Para Enfileirar, precisamos receber a Fila, o fim, o tamanho e o item a ser enfileirado (inserido).
  - Inicialmente, verifico se a Fila está cheia. Se estiver, não consigo enfileirar
  - Caso a Fila não esteja cheia, incremento o fim e o tamanho e adiciono o novo item (pessoa) ao fim da fila.

Filas com Vetores

- Desenfileirar

```c
// Dequeue
Pessoa* retiraDaFila(Pessoa *fila, int *comeco, int *tamanho) {
    if (*tamanho == 0) {
        printf("Fila vazia!\n");
        return NULL;
    }
    *tamanho = *tamanho - 1;
    int aux = *comeco;
    *comeco = (*comeco == TAM-1) ? 0 : *comeco + 1;
    return &(fila[aux]);
}
```

  - Para desenfileirar, precisamos receber a Fila, o começo e o tamanho.
  - Inicialmente, verifico se a Fila está vazia. Se estiver, não há o que desenfileirar.
  - Caso a Fila não esteja vazia, decremento o tamanho, incremento o começo e retorno o item que será removido da fila (o antigo primeiro).

Filas com Vetores

- **Vantagens:**

  i. <u>Acesso rápido aos elementos</u>

  - Como um vetor é uma estrutura de dados contígua na memória, o acesso aos elementos é muito rápido, com complexidade **O(1)**, tanto para inserção quanto remoção.

  ii. <u>Simplicidade de Implementação</u>

  - A Fila pode ser facilmente implementada usando um vetor, o que torna o código simples e direto.

Mesmas vantagens da implementação da Pilha com vetores!

Filas com Vetores

- **Desvantagens:**
  i. Tamanho fixo (em vetores estáticos)
     - Se o vetor for alocado com um tamanho fixo, ele pode levar a um desperdício de memória se o vetor não for completamente utilizado ou a necessidade de realocar o vetor quando ele se lota, o que pode ser custoso (complexidade **O(n)** na realocação e cópia).

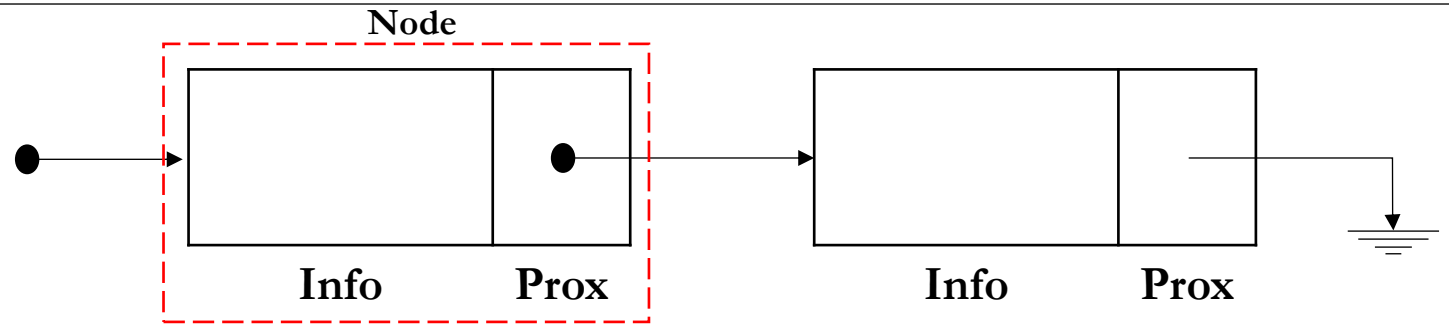
  ii. Complexidade na Remoção de Elementos (em implementações não circulares)
     - Ao usar um vetor não circular, ao remover um item da fila (normalmente na frente), pode ser necessário deslocar todos os outros elementos, o que pode gerar um custo **O(n)** em operações de dequeue.

Filas com Alocação Dinâmica

- Como vimos, uma das desvantagens da Fila com vetores é o tamanho fixo. A solução para esta desvantagem veio através da alocação dinâmica de memória.

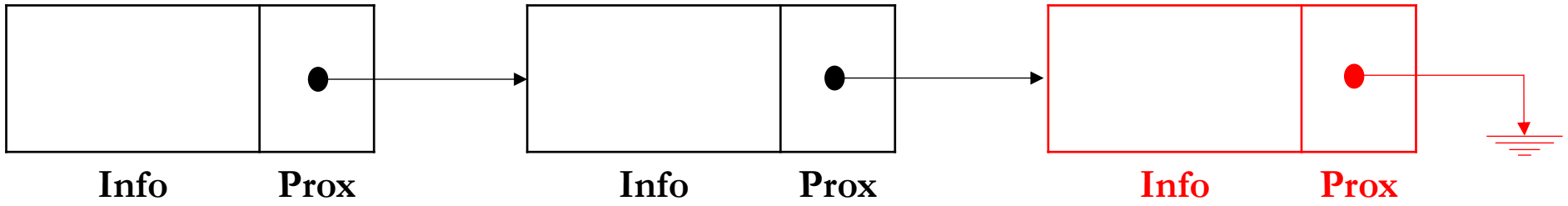
- Para isso, é preciso definirmos uma nova estrutura que será utilizada para armazenar o conteúdo de nossa Fila.

Filas com Alocação Dinâmica



- Assim como fizemos para a Pilha, iremos definir a mesma estrutura chamada Node, que contém dois campos:
  - **Info**, que armazena o conteúdo de cada item da nossa Fila.
  - **Prox**, que armazena o endereço de memória do próximo item da nossa Fila.

Filas com Alocação Dinâmica



- Utilizando esta estrutura, a cada novo item a ser inserido (enfileirado), basta criar outro Node e realizar as atribuições para que este node seja o último da Fila.

Filas com Alocação Dinâmica

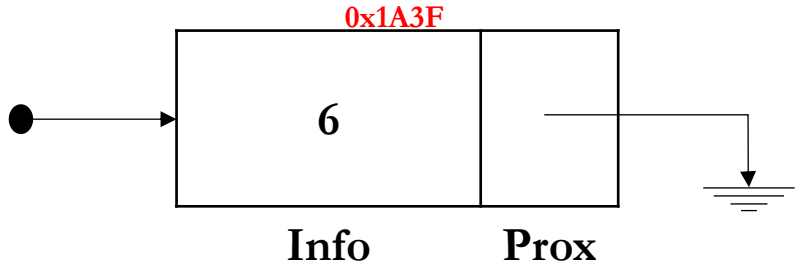
t_0

NULL

Começo: NULL

Ação: Enfileirar o item 6

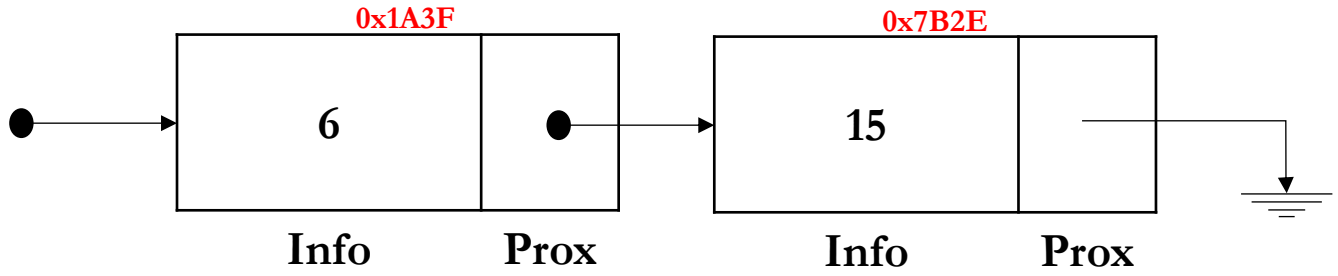
t_1



Começo: **0x1A3F**

Ação: Enfileirar o item 15

t_2

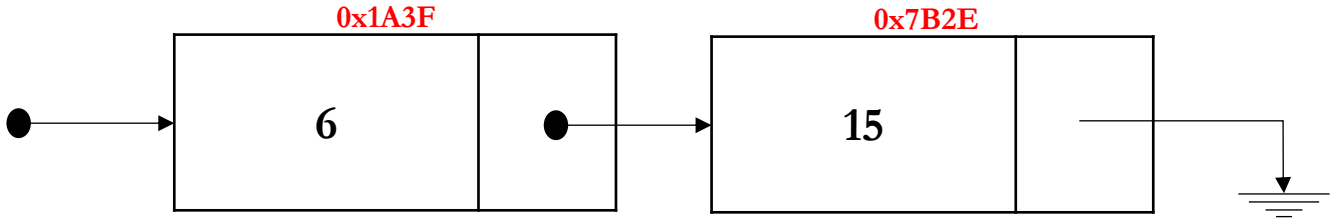


Começo: **0x1A3F**

Filas com Alocação Dinâmica

Ação: Ver o primeiro (Somente retorna o Node 0x1A3F sem mexer na Fila)

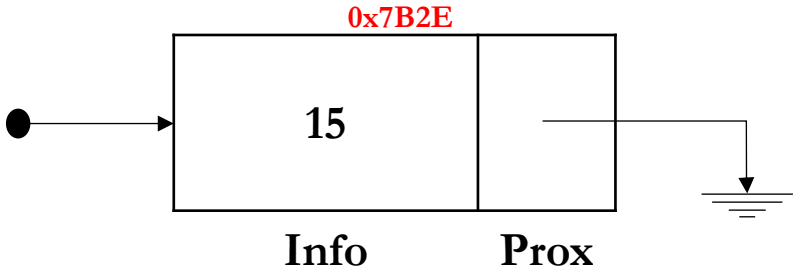
t_3



Começo: 0x1A3F

Ação: Desenfileirar (Retorna 0x1A3F)

t_4



Começo: **0x7B2E**

Ação: Desenfileirar (Retorna 0x7B2E)

t_5 **NULL**

Começo: **NULL**

Filas com Alocação Dinâmica

```c
typedef struct no {
    int info;
    struct no *prox;
} Node;

typedef struct {
    struct no *comeco;
} Fila;
```

- Estrutura
  - Para criar a estrutura, precisamos criar dois tipos novos: Node e Fila.
  - O Node possui dois atributos: info (do tipo dos elementos da Fila) e o prox (do tipo no) – exatamente a mesma estrutura usada no exemplo da Pilha.
  - A Fila possui um no inicial, chamado comeco.

Filas com Alocação Dinâmica

- Enfileirar
  - Para Enfileirar, inicialmente criamos um novo elemento usando alocação dinâmica de memória.
  - Caso o elemento criado seja nulo, quer dizer que não temos espaço suficiente na memória para criar outro nó e, portanto, não é possível Enfileirar.

```c
void insere(Fila *fila, int info) {
    Node *newElement = (Node*) malloc(sizeof(Node));
    if (!newElement) {
        printf("Falha na alocação de memória!\n");
        return;
    }

    newElement->info = info;
    newElement->prox = NULL;
    if (fila->comeco == NULL) {
        fila->comeco = newElement;
    } else {
        Node *aux = fila->comeco;
        while (aux->prox != NULL) {
            aux = aux->prox;
        }
        aux->prox = newElement;
    }
}
```

Filas com Alocação Dinâmica

- Enfileirar *(cont.)*
  - Posteriormente, fazemos com que nosso novo nó receba o conteúdo a ser enfileirado e definimos que o próximo dele é NULL (fim da fila).
  - Se o começo da fila for NULL, significa que nossa Fila é vazia. Assim, o item inserido passa a ser o começo da Fila.

```c
void insere(Fila *fila, int info) {
    Node *newElement = (Node*) malloc(sizeof(Node));
    if (!newElement) {
        printf("Falha na alocação de memória!\n");
        return;
    }

    newElement->info = info;
    newElement->prox = NULL;
    if (fila->comeco == NULL) {
        fila->comeco = newElement;
    } else {
        Node *aux = fila->comeco;
        while (aux->prox != NULL) {
            aux = aux->prox;
        }
        aux->prox = newElement;
    }
}
```

Filas com Alocação Dinâmica

- Enfileirar *(cont.)*
  - Caso a fila não esteja vazia, utiliza uma variável auxiliar para, a partir do começo da fila, buscar o último elemento dela (ou seja, o fim da fila).
  - Após encontrar o fim da fila, define que o novo nó criado é o próximo do antigo fim.

```c
void insere(Fila *fila, int info) {
    Node *newElement = (Node*) malloc(sizeof(Node));
    if (!newElement) {
        printf("Falha na alocação de memória!\n");
        return;
    }

    newElement->info = info;
    newElement->prox = NULL;
    if (fila->comeco == NULL) {
        fila->comeco = newElement;
    } else {
        Node *aux = fila->comeco;
        while (aux->prox != NULL) {
            aux = aux->prox;
        }
        aux->prox = newElement;
    }
}
```

Filas com Alocação Dinâmica

- Desenfileirar

```c
int remover(Fila *fila) {
    if (fila->comeco == NULL) {
        printf("Fila Vazia. Impossível remover!\n");
        return NULL;
    }
    Node *aux = fila->comeco;
    fila->comeco = aux->prox;
    int result = aux->info;
    free(aux);
    return result;
}
```

  - Para desenfileirar, verificamos se o começo é nulo (se for, a nossa fila está vazia e, assim, não há nada para desenfileirar).
  - Posteriormente, criamos um novo Node que recebe o antigo começo, fazemos com que o próximo elemento seja o começo, liberamos o espaço da memória do elemento desenfileirado e, por fim, retornamos o conteúdo do info.

Filas com Alocação Dinâmica

- **Vantagens:**
  i. Flexibilidade de tamanho
     - A Fila pode crescer e encolher conforme necessário, sem se preocupar com a capacidade inicial ou a realocação constante. Isso elimina o risco de ter um tamanho fixo e proporciona um uso mais eficiente da memória. Cada novo elemento é alocado de forma dinâmica conforme a necessidade.

  ii. Eficiência de uso de memória
     - Ao usar alocação dinâmica de memória, a memória é alocada exatamente para o número de elementos armazenados, sem desperdício. Não é necessário reservar espaço extra como em vetores dinâmicos ou arrays de tamanho fixo.

Filas com Alocação Dinâmica

- **Desvantagens:**
  i. Sobrecarga de memória adicional
     - A alocação e desalocação dinâmica de memória envolvem uma sobrecarga adicional de processamento. Cada operação de inserção ou remoção pode exigir a alocação/liberação de novos blocos de memória, impactando no desempenho de filas frequentemente alteradas.

  ii. Fragmentação de memória
     - A memória pode ser fragmentada ao longo do tempo, o que pode tornar o uso ineficiente. Isso é especialmente importante em sistemas com recursos limitados, como sistemas embarcados, onde a fragmentação pode levar a um uso não ideal da memória.

Filas com Alocação Dinâmica

- **Desvantagens** (cont.):
  - iii. Gerenciamento de memória mais complexo
    - É necessário garantir que a memória seja liberada corretamente após o uso para evitar vazamentos de memória. A alocação e desalocação dinâmica de memória podem ser mais lentas do que simplesmente manipular um vetor estático.
  - iv. Desempenho pior na inserção
    - Caso não se utilize dois ponteiros para indicar o começo e outro para indicar o fim (não usamos este último), a operação para inserir na fila passa a ser **O(n)**, porque devemos percorrer nossa fila procurando onde é o fim dela para inserir um novo item.

Algoritmos e Estrutura de Dados II

Prof. Fellipe Guilherme Rey de Souza

Aula 05 – Fila (Implementação)