Outras Estruturas

Algoritmos e Estrutura de Dados 1
Prof. Luiz Gustavo Almeida Martins

Fila de Prioridade

- Estrutura de Dados na qual a classificação intrínseca dos elementos determina os resultados das operações básicas.
  - **Analogia:** pode ser vista como uma mistura de **Lista** com critério de ordenação implícito e **Fila.**
  - Prioridade **afeta a inserção ou a remoção.**
- ∃ 2 tipos básicos:
  - **Fila de Prioridade Ascendente (FPA):**
    - O elemento com **MENOR** "prioridade" deve ser removido.
  - **Fila de Prioridade Descendente (FPD):**
    - O elemento com **MAIOR** "prioridade" deve ser removido.

Fila de Prioridade

- Existem 2 abordagens possíveis para implementar FPA ou FPD:

  1) Uso de uma lista não-ordenada:

  - **Inserção:** Insere no final (como fila simples).
  - **Remoção:** busca e remove o MENOR/MAIOR elemento da lista.

Fila de Prioridade

2) Uso de uma lista ordenada:

- **Inserção**: insere ordenado, ou seja, busca a posição mais adequada do elemento de acordo com a sua prioridade (valor).
- **Remoção**: remove no início (como fila simples).

Fila de Prioridade

- Análise da "complexidade" das 2 abordagens:
  - **1ª Abordagem:**
    - **Inserção**: 1 passo.
    - **Remoção**: ***N*** passos, para fila de ***N*** elementos.
  - **2ª Abordagem:**
    - **Inserção**: ***K*** passos, com ***K*** variando de 1 a ***N***.
    - **Remoção**: 1 passo.

Fila de Prioridade

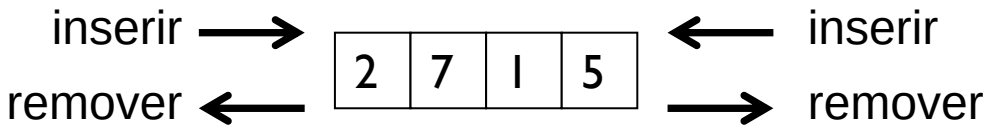
- Notação BIG-O (análise do pior caso):
  - **1ª Abordagem**: 1 passo inserção e ***N*** passos remoção.
  - **2ª Abordagem**: ***N*** passos inserção e 1 passo remoção.
- Portanto, ambas têm complexidade **O(N).**

Fila de Prioridade

- Análise do caso médio:
  - **1ª Abordagem**: ***N*** passos remoção.
  - **2ª Abordagem**: ***N*/2** passos inserção.
- Embora a complexidade das abordagens seja O(1) para uma operação e O(*N*) para a outra, a **2ª abordagem é + vantajosa para o caso médio.**

Fila Dupla ou Deque

- **E.D. que permite remoção/inserção nas 2 extremidades**



- É como se em uma mesma estrutura existissem **duas filas**, uma inversa da outra.

Fila Dupla ou Deque

- **TAD: operações básica**
  - inicializar_deque
  - deque_vazio
  - deque_cheio
  - inserir_início
  - inserir_final
  - remover_início
  - remover_final

Fila Dupla ou Deque

- **Técnicas de Implementação**

  a) Estática/Sequencial:

  - Uso do **incremento circular** (insere_final e remove_início) e do **decremento circular** (insere_início e remove_final)

  - **Diferenciação Fila Cheia/Vazia:**

    a.1 - **Desprezar 1 posição:**

    vazia => ini = fim

    cheia => ini =(fim+1) % max

Fila Dupla ou Deque

a.2 – **Uso de contador:**

vazia => cont = 0

cheia => cont = MAX

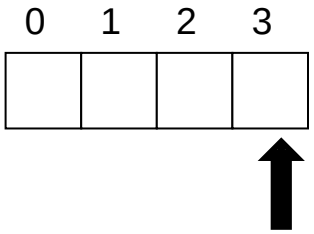
- **Decremento circular:**

$$F \ominus 1 = \begin{cases} F-1, \text{ se } F>0 \\ Max-1, \text{ se } F=0 \end{cases}$$

Fila Dupla ou Deque

- **Exemplo: Solução que despreza 1 posição**

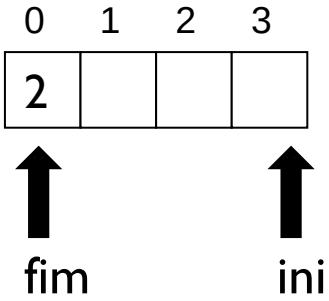
1) Inicializa



max = 4

ini = fim = max - 1

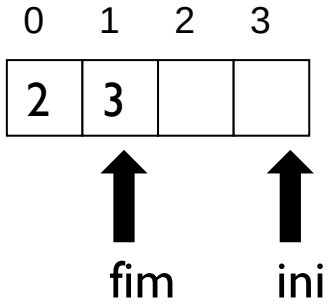
2) Insere_final(2)



Fim = fim ⊕ 1
Vet[fim] = 2

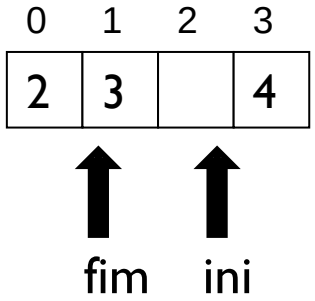
Fila Dupla ou Deque

3) Insere_final(3):



Fim = fim ⊕ 1
Vet[fim] = 3

4) Insere_início(4):

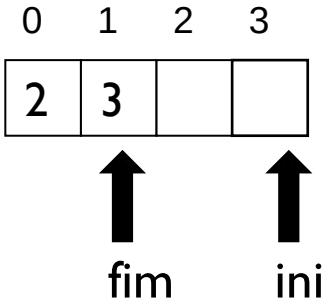


Vet[ini] = 4
ini = ini ⊖ 1

5) Insere_final(5): Impossível: Fila Cheia (ini = fim ⊕ 1)

Fila Dupla ou Deque

6) remove_início(&x):

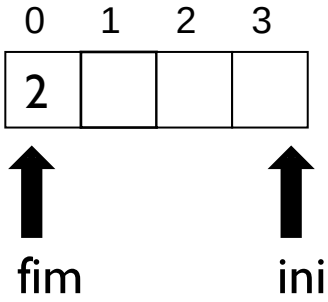


x = 4

Ini = ini ⊕ 1

x = vet[ini]

7) remove_final(&x):



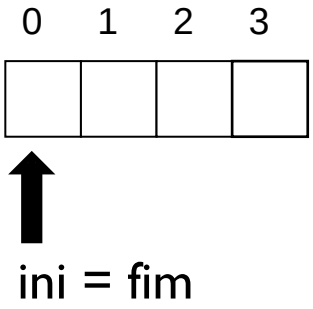
x = 3

x = vet[fim]

fim = fim ⊖ 1

Fila Dupla ou Deque

8) remove_início(&x):



x = 2

Ini = ini ⊕ 1

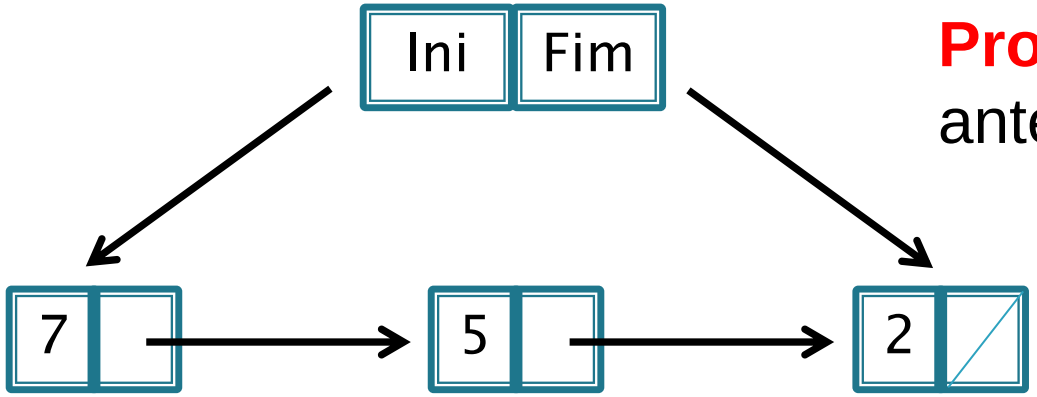
x = vet[ini]

9) remove_final(&x): Impossível: Fila vazia (ini = fim)

Fila Dupla ou Deque

b) Dinâmica/Encadeada

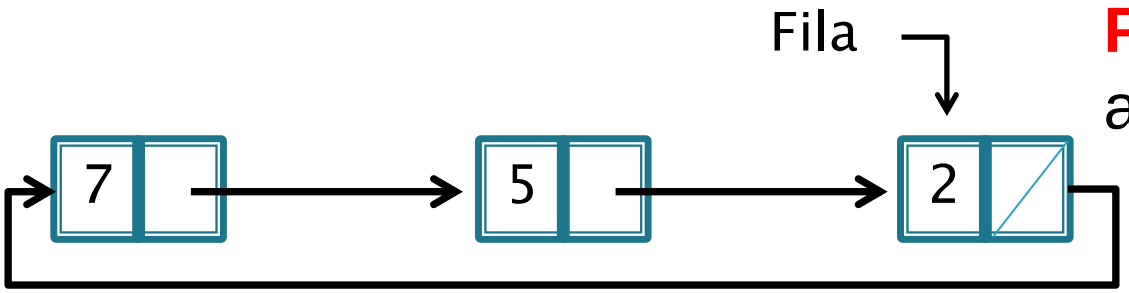
- Encadeamento simples: não é eficiente para remoção_final



Problema: Qual nó antecede Fim?

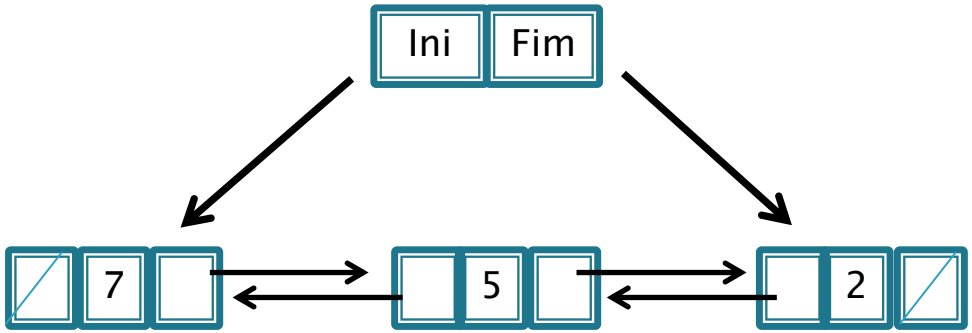
Fila Dupla ou Deque

- Encadeamento circular: também não é eficiente



Problema: Qual nó antecede Fila?

- Solução usual: USO DE ENCADEMENTO DUPLO

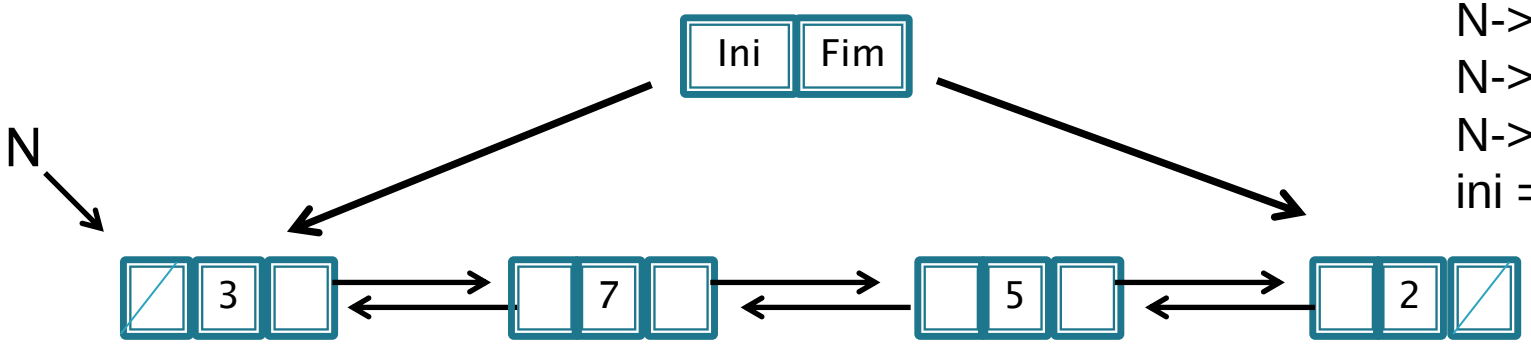


Nó:

Fila Dupla ou Deque

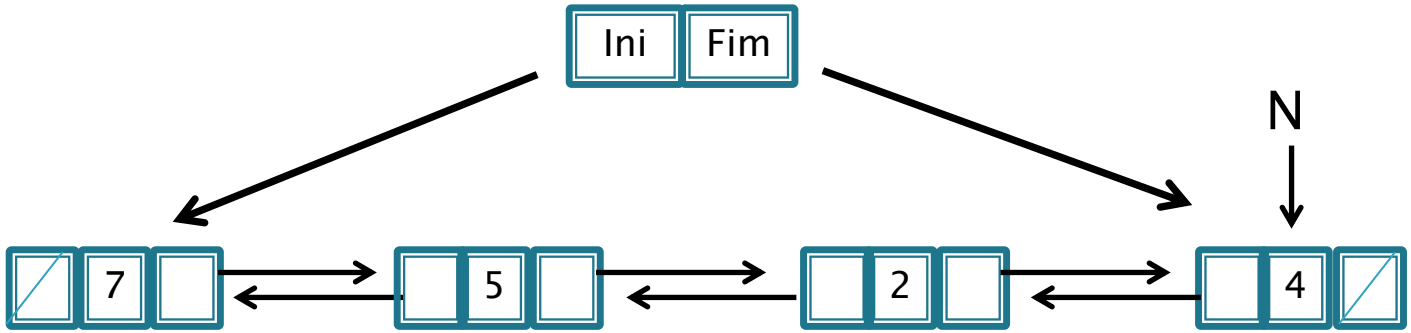
Insere_início(3):

N = aloca_novo
N->info = 3
N->prox = ini
N->ant = NULL
ini = N



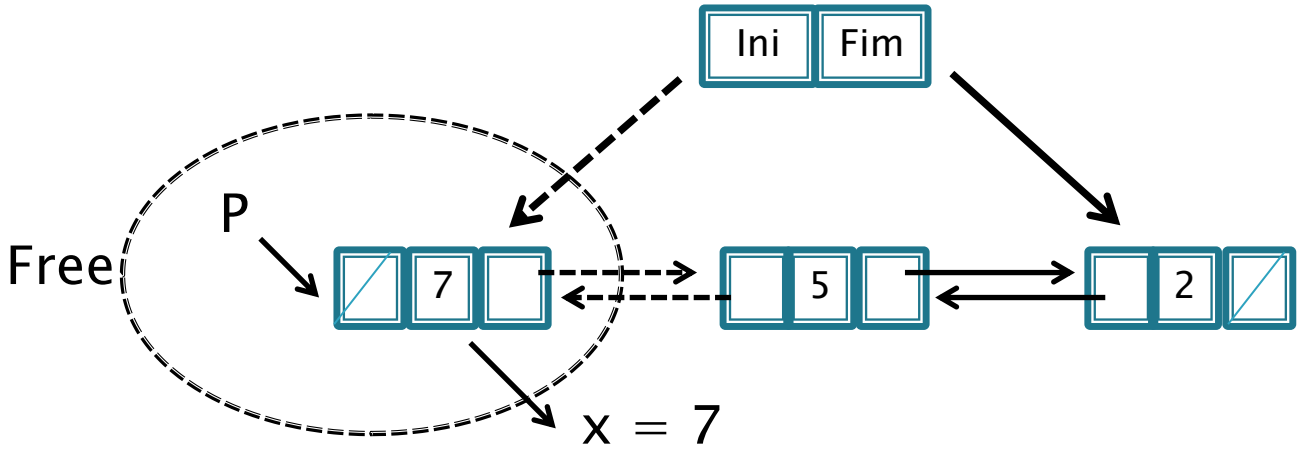
Insere_final(4):

N = aloca_novo
N->info = 4
N->prox = NULL
N->ant = fim
fim->prox = N
fim = N

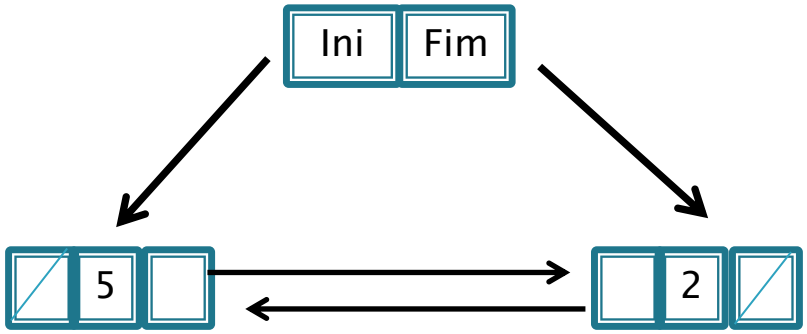


Fila Dupla ou Deque

Remove_início(&x):



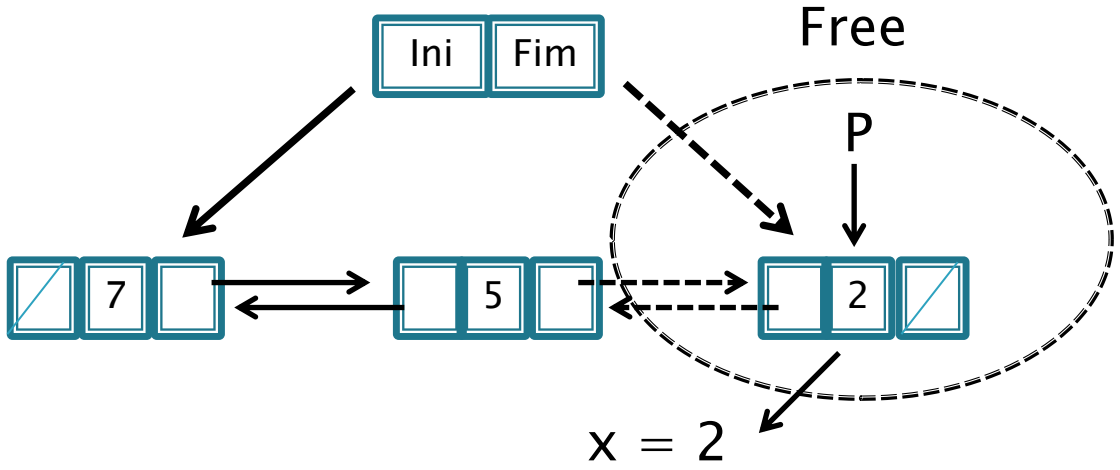
P = Ini
x = P->info
Ini = P->prox
Ini->ant = NULL
free(P)

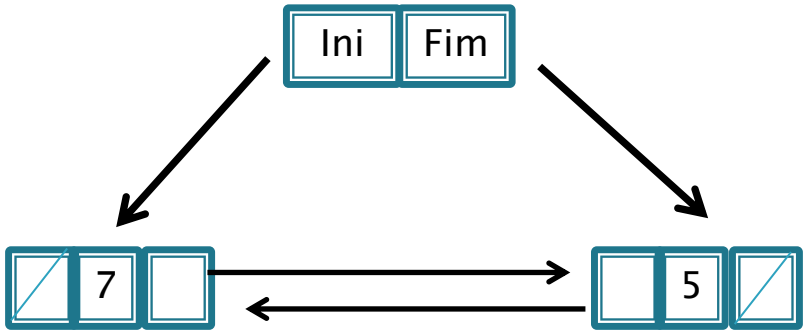


Fila Dupla ou Deque

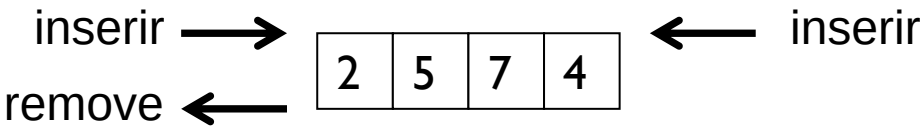
Remove_final (&x):

P = Fim
x = P->info
Fim = P->ant
Fim->prox = NULL
free(P)





Fila Dupla com saída restrita

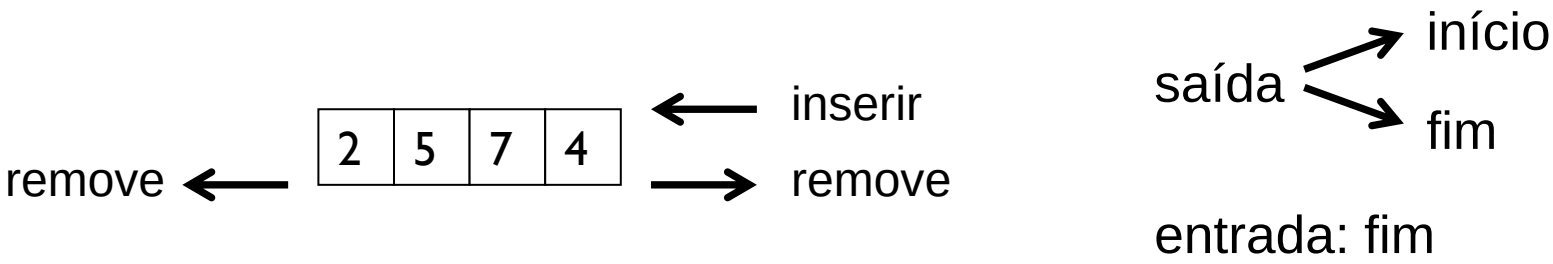


saída: início

entrada → início
entrada → fim

- Funciona como uma fila especial onde eventualmente podemos dar prioridade a um elemento, inserindo-o no início.

Fila Dupla com entrada restrita



- Funciona como uma pilha especial (o final é o topo e o início é a base), onde eventualmente podemos remover um elemento da base.
  - **Ex:** quando ocorrer um estouro da pilha (pilha cheia), pode ser implementada uma remoção do elemento da BASE que é o **mais antigo** da pilha.

Listas Simples e Listas Estruturadas

- A maioria dos exemplos vistos trata de listas simples, onde o elemento de dado é um único campo. A única exceção foi a lista do polinômio.

```
struct no {
        int Info;
        struct no *Prox;
};
```

```
struct no {
        char Info;
        struct no *Prox;
};
```

```
struct termo {
        int Coef;
        int Expo;
        struct termo *Prox;
};
```

Listas Simples e Listas Estruturadas

- Uma opção para se trabalhar com listas mais complexas é o uso de uma estrutura para representar o elemento.

```
struct dado {
        int cod-matricula;
        char *nome;
        char *endereço;
        int idade;
};
```

Acesso: l -> Info.idade

```
struct no {
        struct dado Info;
        struct no *Prox;
};
```

OU

```
struct no {
        struct dado *Info;
        struct no *Prox;
};
```

Acesso: l -> Info -> idade

Listas Homogêneas e Heterogêneas

- Geralmente, os elementos de dado são sempre do mesmo tipo (lista homogênea).

- Entretanto, é possível a construção de **listas heterogêneas**.

  - **Ex**: uma lista de caracteres e números inteiros.

  - Pode ser implementada com o uso do **UNION** ou de um ponteiro **VOID**.

Listas Homogêneas e Heterogêneas

- **Uso do UNION:**

```
struct no {
        int tipo;
        union {
                int i_int;
                char i_char;
        } info;
        struct no *Prox;
};
```

Exemplo de acesso:

```
SE lista->tipo = 0 ENTÃO
        lista->info.i_int = x;
SENÃO
        lista->info.i_char = x;
FIM-SE
```

- **Obs:** Os diferentes membros de uma união se sobrepõem. **O espaço é alocado para o tipo de maior tamanho.**

Listas Homogêneas e Heterogêneas

- **Uso de um ponteiro para VOID:**

```
struct no{
        int tipo;
        void *info;
        struct no *Prox;
};
```

Exemplo de acesso:

```
int *i;
char *c;

if (lista->tipo==0)
{
        *i = x;
        lista->info = i;
}
else
{
        *c = x;
        lista->info = c;
}
```