

MC-202

Filas e Pilhas

Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

2º semestre/2019

Filas

- Uma impressora é compartilhada em um laboratório
- Alunos enviam documentos quase ao mesmo tempo



Como gerenciar a lista de tarefas de impressão?

Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

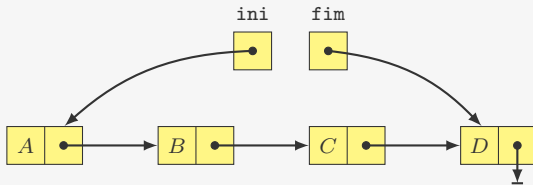
Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo:

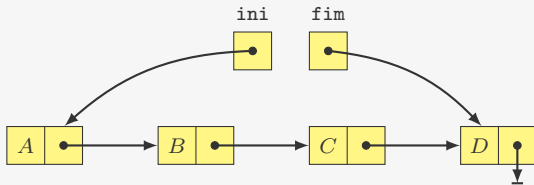


Fila: implementação com lista ligada



```
1 typedef struct {  
2     p_no ini, fim;  
3 } Fila;  
4  
5 typedef Fila * p_fila;
```

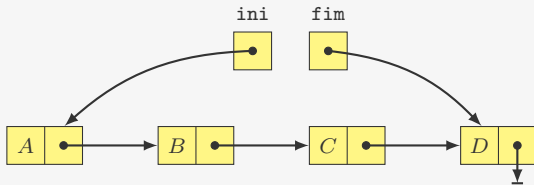
Fila: implementação com lista ligada



```
1 p_fila criar_fila() {
2   p_fila f;
3   f = malloc(sizeof(Fila));
4   f->ini = NULL;
5   f->fim = NULL;
6   return f;
7 }

1 void destruir_fila(p_fila f) {
2   destruir_lista(f->ini);
3   free(f);
4 }
```

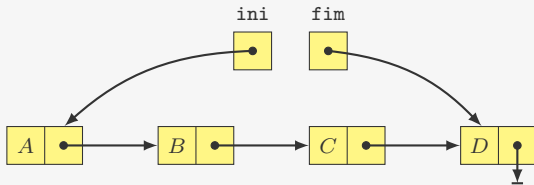
Fila: implementação com lista ligada



Inserir no final:

```
1 void enfileira(p_filha f, int x) {
2     p_no novo;
3     novo = malloc(sizeof(No));
4     novo->dado = x;
5     novo->prox = NULL;
6     if (f->ini == NULL)
7         f->ini = novo;
8     else
9         f->fim->prox = novo;
10    f->fim = novo;
11 }
```

Fila: implementação com lista ligada

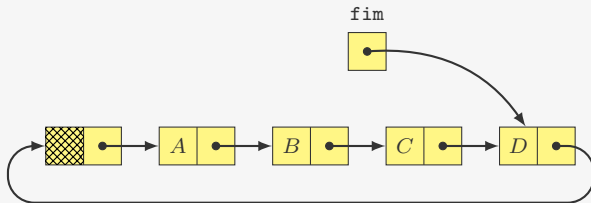


Remove do início:

```
1 int desenfileira(p_filha f) {  
2     p_no primeiro = f->ini;  
3     int x = primeiro->dado;  
4     f->ini = f->ini->prox;  
5     if (f->ini == NULL)  
6         f->fim = NULL;  
7     free(primeiro);  
8     return x;  
9 }
```

Supõe que a lista não é vazia...

Fila: implementação com lista ligada (outra opção)



Enfileira:

- Fazer novo nó apontar para **fim**->**prox**
- Atualizar campo **fim**->**prox** para novo nó
- Mudar **fim** para apontar para o novo nó

Desenfileira:

- Basta remover o nó seguinte ao nó dummy
 - i.e., **fim**->**prox**->**prox**

Exercício: implemente em C essa versão de fila

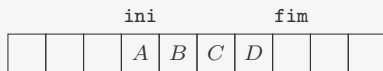
Fila: implementação com vetor

Primeira ideia:

- Inserimos no final do vetor: $O(1)$
- Removemos do começo do vetor: $O(n)$

Segunda ideia:

- Variável **ini** indica o começo da fila
- Variável **fim** indica o fim da fila

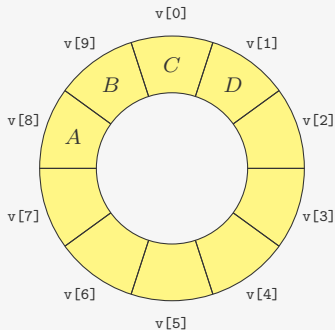


E se, ao inserir, tivermos espaço apenas à esquerda de **ini**?

- podemos mover toda a fila para o começo do vetor
- mas isso leva tempo $O(n)$...

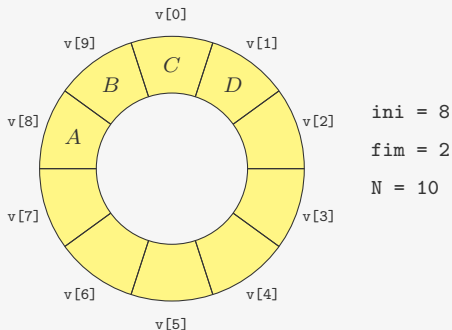
Fila: implementação com vetor (fila circular)

Solução: considerar o vetor de tamanho N de maneira **circular**



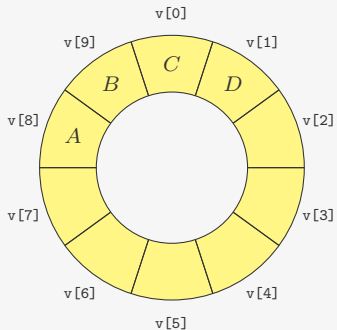
As manipulações de índices são realizadas módulo N

Fila circular - Estrutura



```
1 typedef struct {  
2     int *v;  
3     int ini, fim, N, tamanho;  
4 } Fila;  
5  
6 typedef Fila * p_fila;
```

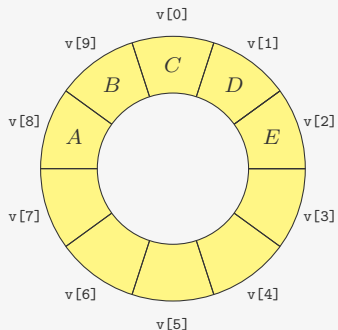
Fila circular - Criando



`ini = 8`
`fim = 2`
`N = 10`
`tamanho = 4`

```
1 p_fila criar_fila(int N) {  
2     p_fila f;  
3     f = malloc(sizeof(Fila));  
4     f->v = malloc(N * sizeof(int));  
5     f->ini = 0;  
6     f->fim = 0;  
7     f->N = N;  
8     f->tamanho = 0;  
9     return f;  
10 }
```

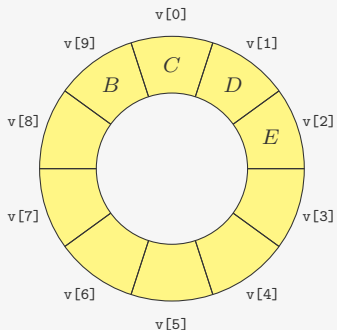
Fila circular - Enfileira



`ini = 8`
`fim = 3`
`N = 10`
`tamanho = 5`

```
1 void enfileira(p_fila f, int x) {  
2     f->v[f->fim] = x;  
3     f->fim = (f->fim + 1) % f->N;  
4     f->tamanho++;  
5 }
```

Fila circular - Desenfileira



`ini = 9`
`fim = 3`
`N = 10`
`tamanho = 4`
`x = A`

```
1 int desenfileira(p_fila f) {  
2     int x = f->v[f->ini];  
3     f->ini = (f->ini + 1) % f->N;  
4     f->tamanho--;  
5     return x;  
6 }
```

Um cliente simples

```
1 int main() {
2     int n, x, i;
3     p_fila f;
4     f = criar_fila(100);
5     scanf("%d", &n);
6     for (i = 0; i < n; i++) {
7         scanf("%d", &x);
8         enfileira(f, x);
9     }
10    while(!fila_vazia(f)) {
11        x = desenfileira(f);
12        printf("%d ", x);
13    }
14    printf("\n");
15    destroi_fila(f);
16    return 0;
17 }
```

Qual é o problema do código acima?

- E se **n** for maior do que **100**?
 - poderíamos usar listas ligadas

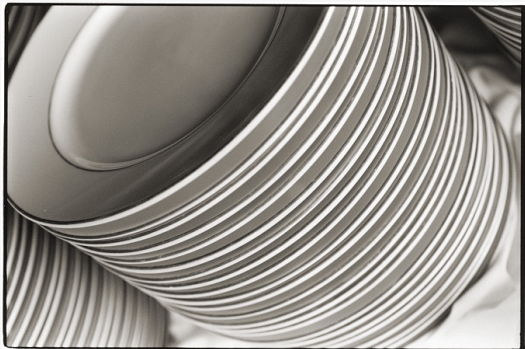
Exemplos de aplicações

Algumas aplicações de filas:

- Gerenciamento de fila de impressão
- Buffer do teclado
- Escalonamento de processos
- Comunicação entre aplicativos/computadores
- Percurso de estruturas de dados complexas (grafos etc.)

Pilha

- Remove primeiro objetos **inseridos há menos tempo**
- **LIFO** (*last-in first-out*): último a entrar é primeiro a sair



É como uma pilha de pratos:

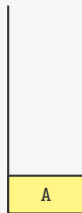
- **Empilha** os pratos limpos sobre os que já estão na pilha
- **Desempilha** o prato de cima para usar

Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo:



Pilha: implementação com vetor

Definição:

```
1 typedef struct {
2     int *v;
3     int topo;
4 } Pilha;
5
6 typedef Pilha * p_pilha;
```



Inserção:

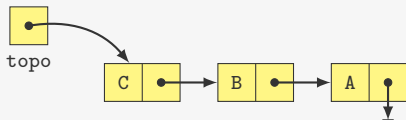
```
1 void empilhar(p_pilha p, int i) {
2     p->v[p->topo] = i;
3     p->topo++;
4 }
```

Remoção:

```
1 int desempilhar(p_pilha p) {
2     p->topo--;
3     return p->v[p->topo];
4 }
```

Pilha: implementação com lista ligada

Após empilhar A, B e C:

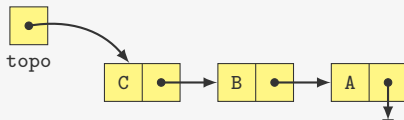


Estrutura:

```
1 typedef struct {  
2     p_no topo;  
3 } Pilha;  
4  
5 typedef Pilha * p_pilha;
```

Pilha: implementação com lista ligada

Após empilhar A, B e C:

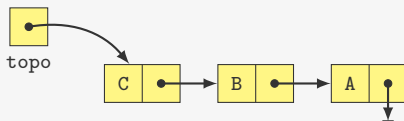


Empilhando:

```
1 void empilhar(p_pilha pilha, int x) {  
2     p_no novo = malloc(sizeof(No));  
3     novo->dado = x;  
4     novo->prox = pilha->topo;  
5     pilha->topo = novo;  
6 }
```

Pilha: implementação com lista ligada

Após empilhar A, B e C:



Desempilhando:

```
1 int desempilhar(p_pilha pilha) {  
2     p_no topo = pilha->topo;  
3     int x = topo->dado;  
4     pilha->topo = pilha->topo->prox;  
5     free(topo);  
6     return x;  
7 }
```

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...
- Cálculo e conversão de notações
 - pré-fixa
 - pós-fixa
 - infixa (com parênteses)
- Percurso de estruturas de dados complexas (grafos etc.)
- Recursão

Veremos algumas dessas aplicações na próxima unidade

Exercício

Um *deque* (*double-ended queue*) é uma estrutura de dados com as operações: `insere_inicio`, `insere_fim`, `remove_inicio`, `remove_fim`.

Implemente um *deque* utilizando listas ligadas.