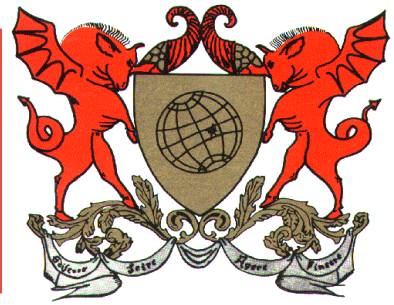


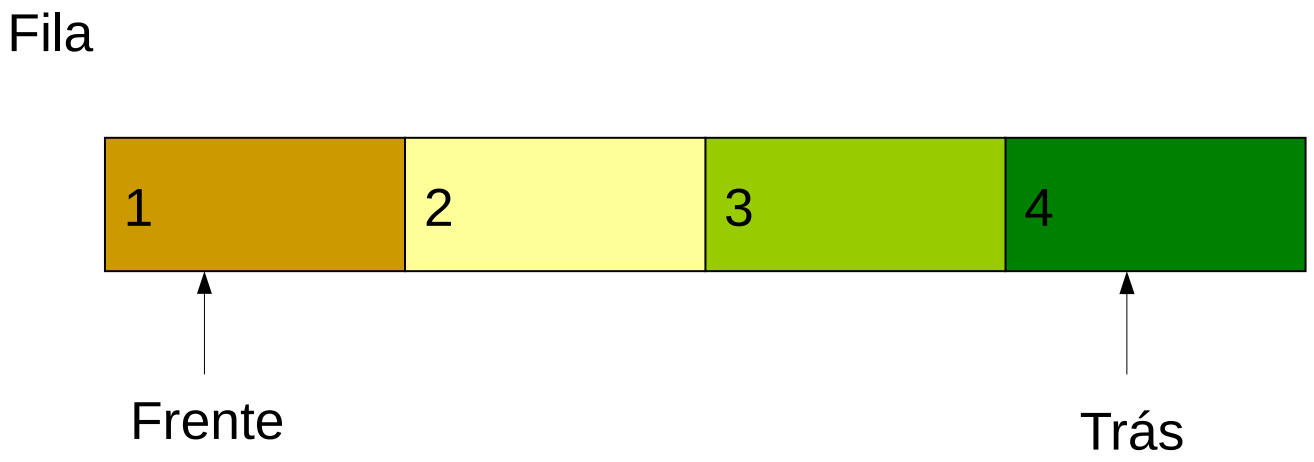
Universidade Federal de Viçosa
Campus de Florestal

Algoritmos e Estruturas de Dados I (CCF 211)

Filas (Cap03 – Seção 3.3- Ziviani)

Profa.Thais R. M. Braga Silva
<thais.braga@ufv.br>

O que é uma fila?



Fila

- É uma lista linear em que todas as inserções são realizadas em um extremo da lista, e todas as retiradas e, geralmente, os acessos são realizados no outro extremo da lista.
- O modelo intuitivo de uma fila é o de uma fila de espera em que as pessoas no início da fila são servidas primeiro e as pessoas que chegam entram no fim da fila
- São chamadas de listas FIFO (“*first in, first out*”)

Fila

- Existe uma ordem linear para filas que é a “ordem de chegada”
- São utilizadas quando desejamos processar itens de acordo com a ordem “primeiro-que-chega, primeiro-atendido”
- Sistemas operacionais utilizam filas para regular a ordem na qual tarefas devem receber processamento e recursos devem ser alocados a processos

TAD Fila

- Tipo Abstrato de dados com a seguinte característica:
  - O **primeiro** elemento a ser inserido é o **primeiro** a ser retirado/ removido

    (FIFO – *First in First Out*)

- Analogia: fila bancária, fila do cinema.

- Usos: Sistemas operacionais: fila de impressão, processamento

TAD Fila

- Operações:
  - 1. FFVazia(Fila). Faz a fila ficar vazia.
  - 2. FEnfileira(Fila, x). Insere o item x no final da fila.
  - 3. FDesenfileira(Fila, x). Retorna o item x no início da fila, retirando-o da fila.
  - 4. FEhVazia(Fila). Esta função retorna true se a fila está vazia; senão retorna false.

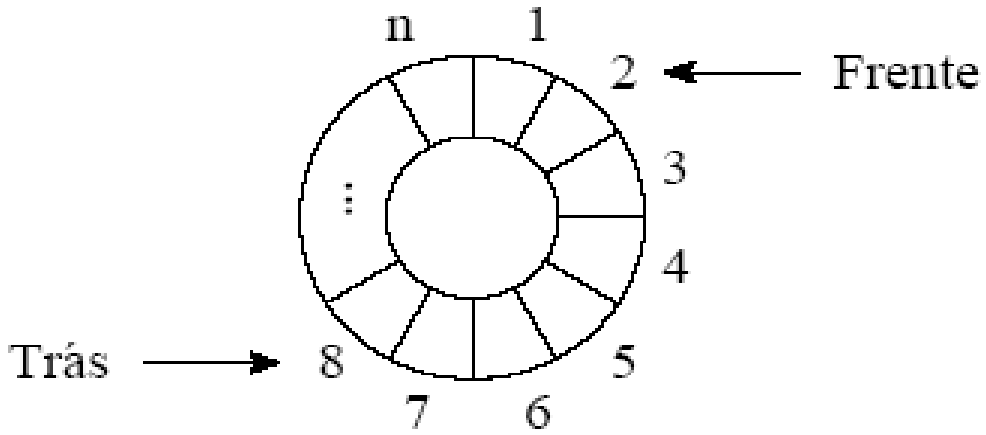
Implementação de Filas através de Arranjos

- **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.**

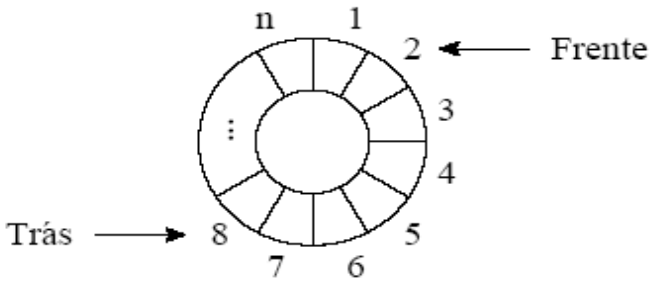
Implementação de Filas através de Arranjos

- **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.**



Implementação de Filas através de Arranjos



- **A fila se encontra em posições contíguas de memória, em alguma posição do círculo, delimitada pelos apontadores Frente e Trás. (Frente indica a posição do primeiro elemento, trás a primeira posição vazia)**

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

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

Estrutura da Fila usando Arranjo

```
#define MaxTam  1000

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

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

Estrutura da Fila usando Arranjo

```
void FFVazia(TFila* pFila);

int FEhVazia(TFila* pFila);

int FEnfileira(TFila* pFila,TItem* pItem);

int FDesenfileira(TFila* pFila,TItem* pItem);
```

Operações sobre Filas usando Posições Contínuas de Memória

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

Operações sobre Filas usando Posições Contínuas de Memória

```
int FEnfileira(TFila* pFila,
               TItem* pItem)
{
  if (((pFila->iTras+1)%(MaxTam+1)) == pFila->iFrente)
    return 0; /* fila cheia */

  pFila->vItem[pFila->iTras] = *pItem;
  pFila->iTras = (pFila->iTras + 1) % (MaxTam+1);
/*
  if (pFila->iTras == MaxTam) pFila->iTras = 0;
  else                        pFila->iTras++;
*/
  return 1;
} /* FEnfileira */
```

Operações sobre Filas usando Posições Contínuas de Memória

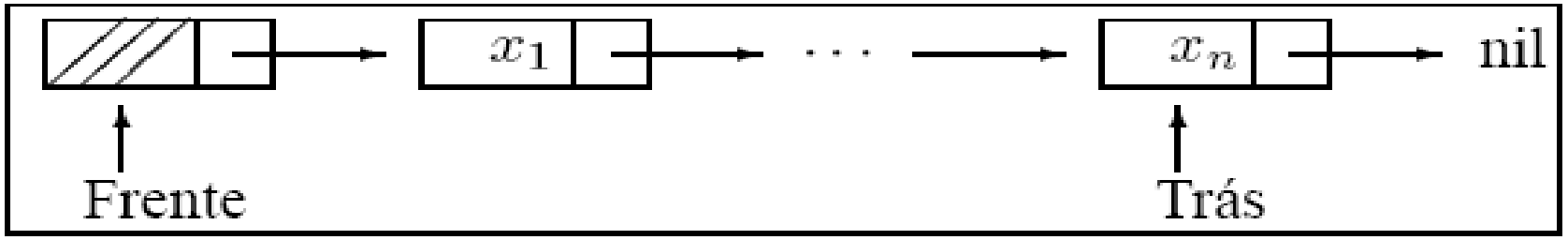
```
int FDesenfileira(TFila* pFila,
                  TItem* pItem)
{
  if (FEhVazia(pFila))
    return 0;

  *pItem = pFila->vItem[pFila->iFrente];
  pFila->iFrente = (pFila->iFrente + 1) % (MaxTam+1);
/*
  if (pFila->iFrente == MaxTam) pFila->iFrente = 0;
  else                          pFila->iFrente++;
*/
  return 1;
} /* FDesenfileira */
```

Implementação de Filas usando Apontadores

- **Há uma célula cabeça para facilitar a implementação das operações Enfileira e Desenfileira quando a fila está vazia.**
- **Quando a fila está vazia, os apontadores Frente e Trás 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 cabeça da lista e a célula que contém x_1 passa a ser a célula cabeça.

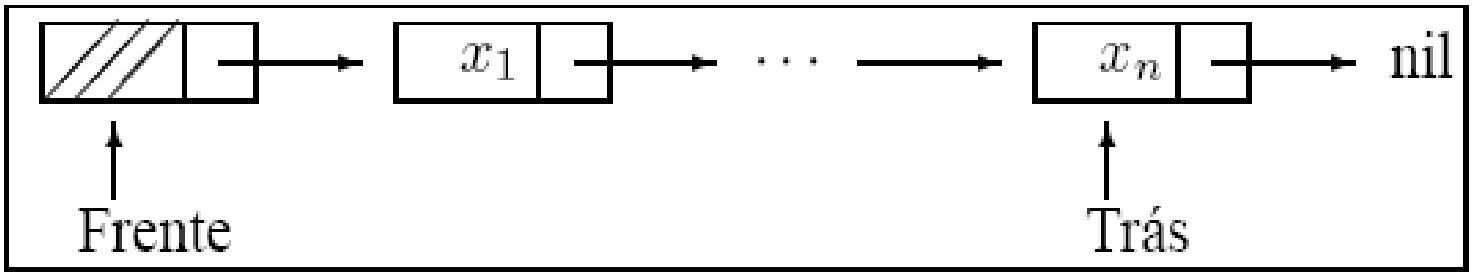


Estrutura da Fila usando Apontadores

- **A fila é implementada por meio de células**

- **Cada célula contém um item da fila e um apontador para outra célula.**

- **A estrutura TipoFila contém um apontador para a frente da fila (célula cabeça) e um apontador para a parte de trás da fila.**



Estrutura da Fila usando Apontadores

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

typedef struct Celula* Apontador;
typedef struct Celula {
  TItem Item;
  struct Celula* pProx;
} TCelula;

typedef struct TFila {
  Apontador pFrente;
  Apontador pTras;
} TFila;
```

Operações sobre Filas usando Apontadores

```
void FFVazia(TFila* pFila);

int FEhVazia(TFila* pFila);

int FEnfileira(TFila* pFila,TItem* pItem);

int FDesenfileira(TFila* pFila,TItem* pItem);
```

Operações sobre Filas usando Apontadores

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

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

Operações sobre Filas usando Apontadores

```
int FEnfileira(TFila *pFila,
               TItem* pItem)
{
  Apontador pNovo;
  pNovo = (Apontador)malloc(sizeof(TCelula));
  if (pNovo == NULL) return 0;

  pFila->pTras->pProx = pNovo;
  pFila->pTras = pNovo;
  pNovo->Item = *pItem;
  pNovo->pProx = NULL;
  return 1;
} /* FEnfileira */
```

Operações sobre Filas usando Apontadores

```
int FDesenfileira(TFila* pFila,
                  TItem* pItem)
{
  Apontador pAux;
  if (FEhVazia(pFila)) return 0;

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

  return 1;
} /* FDesenfileira */
```

Exercícios

Considere uma fila F não vazia e uma pilha P vazia. Usando apenas uma variável temporária, as operações desempilha, empilha, desenfileira, enfileira e os testes se a pilha e fila estão vazias, escreva um algoritmo para reverter a ordem dos elementos em F.

Exercícios

Implemente uma fila de espera de um consultório médico. O programa deve permitir:

- inserir um paciente no final da fila

- chamar o paciente a ser atendido

- informar quantos pacientes existem na fila de espera