Estruturas de Informação
Aula 13: Filas

07/05/2009

Fontes Bibliográficas

- Livros:
  - Projeto de Algoritmos (Nivio Ziviani): Capítulo 3;
  - Introdução a Estruturas de Dados (Celes, Cerqueira e Rangel): Capítulo 10;
  - Estruturas de Dados e seus Algoritmos (Szwarefiter, et. al): Capítulo 2;
  - Algorithms in C (Sedgewick): Capítulo 3;
- Slides baseados nas transparências disponíveis em: http://www.dcc.ufmg.br/algoritmos/transparencias.php

Filas

- 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.
- Ex.: fila de um banco
- São chamadas de estruturas do tipo fifo ("first-in", "first-out").
- 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

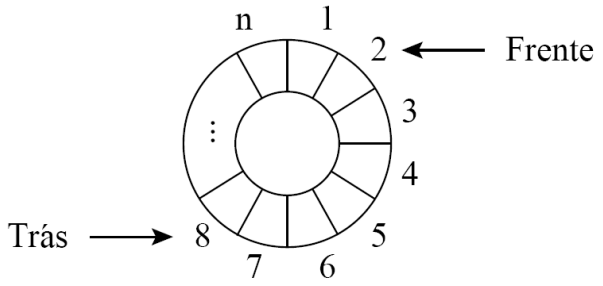
Alguns exemplos de operações:

- FFVazia(Fila). Faz a fila ficar vazia.
- Enfileira(x, Fila). Insere o item x no final da fila.
- Desenfileira(Fila, x). Retorna o item x no início da fila, retirando-o da fila.
- Vazia(Fila). Esta função retorna *true* se a fila está vazia; senão retorna *false*.

Implementação Fila usando Vetores

- 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.
- 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 Fila usando Vetores (2)



Implementação Fila usando Vetores (3)

- A fila se encontra em posições contíguas de memória, em alguma posição do círculo, delimitada pelos ponteiros (índices) Frente e Trás.
- Para enfileirar, basta mover o ponteiro Trás uma posição no sentido horário.
- Para desenfileirar, basta mover o ponteiro Frente uma posição no sentido horário.

Estrutura Fila usando Vetores (Fila.h)

```
#ifndef FILA_H_
#define FILA_H_

typedef int TipoChave;
typedef struct tipoitem TipoItem;
typedef struct tipofila TipoFila;

TipoFila* InicializaFila ();
void FFVazia(TipoFila* Fila);
int Vazia(TipoFila* Fila);
void Enfileira(TipoItem* x, TipoFila* Fila);
void Desenfileira(TipoFila* Fila, TipoItem* Item);
void Imprime(TipoFila* Fila);
TipoItem* InicializaTipoItem (TipoChave n);

#endif /*FILA_H_*/
```

Estrutura Fila usando Vetores (Fila.c)

- O tamanho máximo do **Vetor** circular é definido pela constante MaxTam.
- Os outros campos do registro TipoPilha contêm ponteiros (índices) para a parte da frente e de trás da fila

```
#include <stdio.h>
#include <stdlib.h>
#include "Fila.h"
#define MaxTam 1000

typedef int Ponteiro;

struct tipoitem {
   TipoChave Chave; /* outros campos */
};

struct tipofila{
   TipoItem Item[MaxTam];
  Ponteiro Frente, Tras;
};
```

Operações do TAD Fila (1)

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

Operações do TAD Fila (2)

```
TipoFila* InicializaFila(){
  TipoFila* fila =
  (TipoFila*)malloc(sizeof(TipoFila));
  return fila;
}
```

Operações do TAD Fila (3)

```
void FFVazia(TipoFila* Fila)
{
  Fila->Frente = 0;
  Fila->Tras = Fila->Frente;
}

int Vazia(TipoFila* Fila)
{
  return (Fila->Frente == Fila->Tras);
}
```

Operações do TAD Fila (4)

```
void Enfileira(TipoItem* x, TipoFila *Fila)
{
  if ((Fila->Tras+1) % MaxTam == Fila->Frente)
      printf(" Erro fila esta cheia\n");
  else {
      Fila->Item[Fila->Tras] = *x;
      Fila->Tras = (Fila->Tras+1) % MaxTam;
  }
}
```

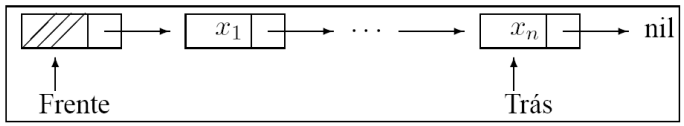
Operações do TAD Fila (5)

```
void Desenfileira(TipoFila* Fila, TipoItem* Item)
{
  if (Vazia(Fila))
      printf(" Erro fila esta vazia\n");
  else
  {
      *Item = Fila->Item[Fila->Frente];
      Fila->Frente = (Fila->Frente+1) % MaxTam;
  }
}
```

Operações do TAD Fila (6)

```
void Imprime(TipoFila* Fila)
{
  int Aux;
  printf("Imprime fila com vetores\n");
  for (Aux = Fila->Frente; Aux <= (Fila->Tras -
  1); Aux++)
      printf("%d\n", Fila->Item[Aux].Chave);
}
```

Filas com alocação não sequencial e dinâmica



Filas com alocação não sequencial e dinâmica

- Há uma célula cabeça para facilitar a implementação das operações Enfileira e Desenfileira quando a fila está vazia.
- Quando a fila está vazia, os ponteiros 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 xn e colocar nela o novo item.
- Para desenfileirar o item x1, basta desligar a célula cabeça da lista e a célula que contém x1 passa a ser a célula cabeça.

Estrutura da Fila com Alocação não Sequencial e Dinâmica

- A fila é implementada por meio de células.
- Cada célula contém um item da fila e um ponteiro para outra célula.
- O registro TipoFila contém um ponteiro para a frente da fila (célula cabeça) e um ponteiro para a parte de trás da fila.

Estrutura da Fila com Alocação não Sequencial e Dinâmica (2)

```
#include <stdio.h>
#include <stdlib.h>
#include "Fila.h"

typedef struct Celula_str* Ponteiro;

struct tipoitem {
   TipoChave Chave; /* outros componentes */
};

typedef struct Celula_str {
   TipoItem Item;
   Ponteiro Prox;
}Celula;

struct tipofila {
   Ponteiro Frente, Tras;
};
```

Operações do TAD Fila usando ponteiros (1)

```
TipoFila* InicializaFila(){
  TipoFila* fila = (TipoFila*)malloc(sizeof(TipoFila));
  return fila;
}
void FFVazia(TipoFila* Fila)
{
  Fila->Frente = (Ponteiro) malloc(sizeof(Celula));
  Fila->Tras = Fila->Frente;
  Fila->Frente->Prox = NULL;
}

int Vazia(TipoFila* Fila)
{
  return (Fila->Frente == Fila->Tras);
}
```

Operações do TAD Fila usando ponteiros (2)

```
void Enfileira(TipoItem* x, TipoFila
  *Fila)
{
  Fila->Tras->Prox = (Ponteiro)
  malloc(sizeof(Celula));
  Fila->Tras = Fila->Tras->Prox;
  Fila->Tras->Item = *x;
  Fila->Tras->Prox = NULL;
}
```

Operações do TAD Fila usando ponteiros (3)

```
void Desenfileira(TipoFila* Fila, TipoItem* Item)
{
  Ponteiro q;
  if (Vazia(Fila))
  { printf(" Erro fila esta vazia\n");
    return;
  }
  q = Fila->Frente;
  Fila->Frente = Fila->Frente->Prox;
  *Item = Fila->Frente->Item;
  free(q);
}
```

Operações do TAD Fila usando ponteiros (4)

```
void Imprime (TipoFila* Fila)
{
  Ponteiro Aux;
  Aux = Fila->Frente->Prox;
  while (Aux != NULL)
  {
     printf("%d\n", Aux->Item.Chave);
     Aux = Aux->Prox;
  }
}
```

Exercício

Implemente o tipo fila, usando duas pilhas como estruturas de dados para representar as filas