

AED1 – Aula 5

Filas

Prof. Dr. Aldo Díaz
aldo.diaz@ufg.br

Instituto de Informática
Universidade Federal de Goiás

INF

INSTITUTO DE
INFORMÁTICA



UFG

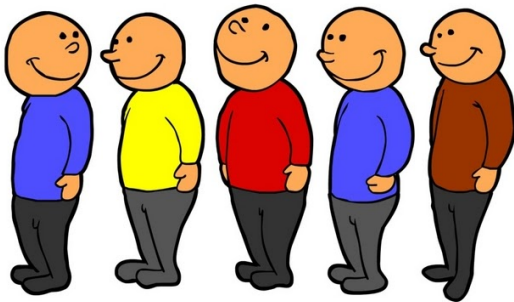
UNIVERSIDADE
FEDERAL DE GOIÁS

01/12/2022

- 1 Fila
- 2 Representação por contiguidade
- 3 Representação por encadeamento
- 4 Fila de prioridade
- 5 Saiba mais ...

Fila

Estrutura de dados dinâmica que permite operações de **inserção** e **remoção** de objetos.

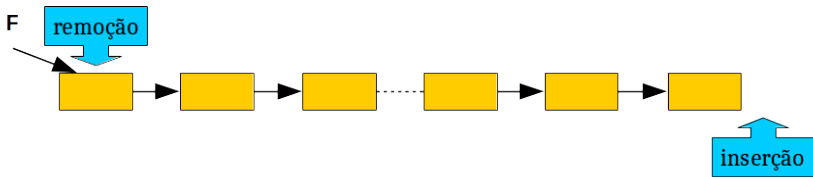


Fila – Política FIFO

Filas seguem uma política FIFO:

“O primeiro elemento inserido é o primeiro a ser removido”

Estas 2 operações são realizadas nas extremidades da fila:



Fila – Exemplos

- Pessoas aguardando por atenção num banco, callcenter, ...
- Arquivos numa fila de impressão
- Processos esperando recursos da CPU, GPU
- Dados aguardando transmissão numa porta serial (USB, RS232, SPI, I2C, Bluetooth)



Fila – Operações

As **operações fundamentais** sobre filas são:

- Criar fila
- Quantificar os elementos presentes
- Inserir um elemento
- Remover um elemento

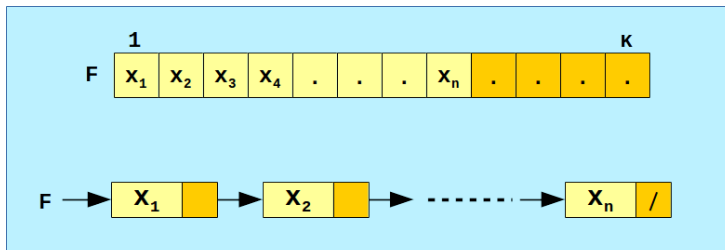
Outras **operações auxiliares** são:

- Mostrar os elementos da fila
- Concatenar duas filas
- Consultar se uma *chave* está presente num dos elementos

Fila – Representação

Há 2 maneiras para representar uma fila

1. Por contiguidade (alocação estática)
2. Por encadeamento (alocação dinâmica)



Representação por contiguidade

É uma implementação simples (um *buffer*) baseada em um TAD:

- Utiliza-se 1 vetor para representar a fila
- Utilizam-se 2 variáveis de controle:
 - Tamanho atual: O número de elementos presentes na fila naquele momento
 - Tamanho máximo: O número de elementos que a fila poderá conter como máximo

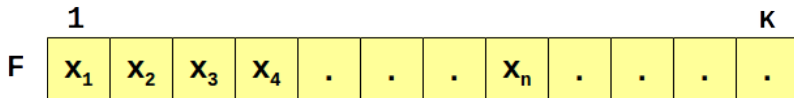
Desvantagens:

- O vetor pode nem sempre ser completamente utilizado pela fila
- O tamanho do vetor pode ser insuficiente para armazenar todos os elementos da fila

Representação por contiguidade – Implementação

Desafio – Implemente uma fila utilizando um TAD.

TamanhoFila = n



Representação por contiguidade – Implementação

lista.h

```
1 // Definicoes
2 #define SUCESSO 1
3 #define FALHA 1
4 #define CHAVE_INVALIDA 0
5 #define TAMANHO_MAXIMO_FILA 100
6
7 // Variaveis globais
8 typedef struct elemento Elemento;
9 typedef struct fila Fila;
10
11 // Prototipo das operacoes (funcoes)
12 int criarFilaVazia(Fila* fila);
13 int criarFilaChave(Fila* fila, Elemento elemento);
14 int tamanhoFila(Fila fila);
15 void mostrarElemento(Elemento elemento);
16 void mostrarFila(Fila fila);
17 Elemento consultaPosicao(Fila fila, unsigned int intPosicao);
18 int insFinal(Fila* fila, Elemento elemento);
19 Elemento remInicio(Fila* fila);
```

Representação por contiguidade – Implementação

lista.c

```
1 #include <stdio.h>
2 #include "lista.h"
3
4 // Definicao dos TAD
5 struct elemento {
6     unsigned int chave;
7     unsigned int dado;
8 };
9
10 struct fila {
11     Elemento elementos[TAMANHO_MAXIMO_FILA];
12     unsigned int tamanho;
13 };
```

Representação por contiguidade – Implementação

lista.c

```
1 // Criar fila, por contiguidade – Versao 1
2 int criarFilaVazia(Fila* fila) {
3     fila->tamanho = 0;
4
5     return(SUCESSO);
6 }
7
8 // Criar fila, por contiguidade – Versao 2
9 int criarFilaVazia(Fila* fila) {
10    fila->tamanho = 0;
11    for(int i=0; i<TAMANHO_MAXIMO_FILA; i++)
12        fila->elementos[i].chave = CHAVE_INVALIDA;
13
14    return(SUCESSO);
15 }
```

Representação por contiguidade – Implementação

lista.c

```
1 // Criar fila com um unico elemento, por contiguidade
2 int criarFilaChave(Fila* fila, Elemento elemento) {
3     fila->elementos[0] = elemento;
4     fila->tamanho = 1;
5
6     return (SUCESSO);
7 }

1 // Determinar o tamanho da fila, por contiguidade
2 int tamanhoFila(Fila fila) {
3     if(fila.tamanho >= 0)
4         return(fila.tamanho);
5     else
6         return(FALHA);
7 }
```

Representação por contiguidade – Implementação

lista.c

```
1 // Mostrar um elemento da fila, por contiguidade
2 void mostrarElemento(Elemento elemento) {
3     printf("Chave.....: %u\n", elemento.chave);
4     printf("Dado.....: %u\n", elemento.dado);
5 }

1 // Mostrar toda a fila, por contiguidade
2 void mostrarFila(Fila fila) {
3     if(fila.tamanho == 0)
4         printf("Atencao: A fila esta vazia.\n");
5     else {
6         printf("A fila linear possui %u elementos.\n\n", fila.tamanho);
7         for(int i=0; i<fila.tamanho; i++) {
8             printf("Elemento n.: %u\n", (i+1));
9             mostrarElemento(fila.elementos[i]);
10        }
11    }
12 }
```

Representação por contiguidade – Implementação

lista.c

```
1 // Consultar elemento pela posicao dele, por contiguidade
2 Elemento consultaPosicao(Fila fila, unsigned int intPosicao) {
3     Elemento elementoResultado;
4
5     if((intPosicao > 0) && (intPosicao <= fila .tamanho))
6         elementoResultado = fila.elementos[intPosicao-1];
7     else
8         elementoResultado.chave = CHAVE_INVALIDA;
9
10    return(elementoResultado);
11 }
```

Representação por contiguidade – Implementação

lista.c

```
1 // Inserir elemento no final da fila, por contiguidade
2 int insFinal(Fila* fila, Elemento elemento) {
3     unsigned int i;
4     Elemento auxiliar;
5
6     if(fila->tamanho == TAMANHO_MAXIMO_FILA)
7         return(FALHA);
8     else {
9         fila->elementos[fila->tamanho] = elemento;
10        fila->tamanho++;
11        return(SUCESSO);
12    }
13 }
```

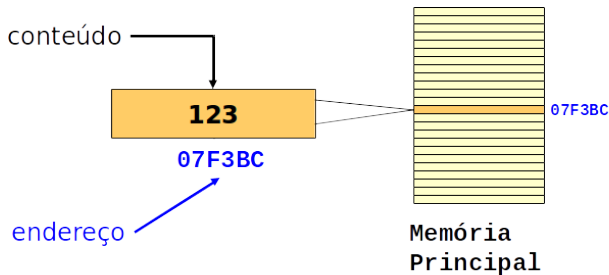

Representação por contiguidade – Implementação

lista.c

```
1 // Remover elemento no inicio da fila, por contiguidade
2 Elemento remInicio(Fila* fila ) {
3     Elemento elementoResultado;
4
5     if(fila->tamanho == 0) {
6         elementoResultado.chave = CHAVE_INVALIDA;
7         return(elementoResultado);
8     }
9     else {
10        elementoResultado = fila->elementos[0];
11        for(int i=0; i<(fila->tamanho-1); i++)
12            fila->elementos[i] = fila->elementos[i+1];
13        fila->tamanho--;
14        return(elementoResultado);
15    }
16 }
```

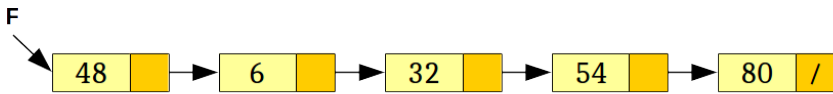
Representação por encadeamento

Uma outra maneira de representar uma fila é por **encadeamento**, o que demanda o uso de **ponteiros** para endereços de memória.

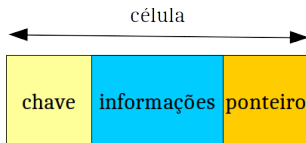


Representação por encadeamento

Assim, pode-se imaginar a seguinte maneira de organizar informações:



Onde cada elemento possui:

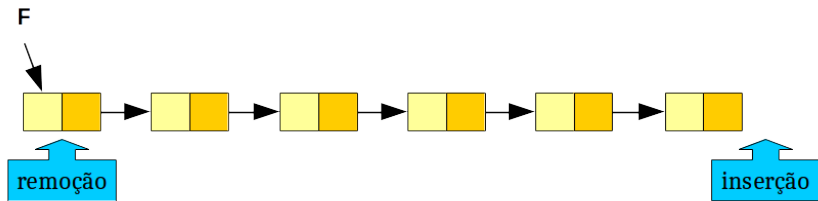


- a. Uma **chave**
- b. **Dados** (48, 6, 32, 54, 80)
- c. Um **ponteiro** para o elemento seguinte

Representação por encadeamento

Observações:

- Há um apontador especial para o **primeiro elemento** da fila
- No **último elemento**, utiliza-se um ponteiro NULL para indicar o fim da fila
- Em uma **fila vazia**, o ponteiro para o primeiro elemento será NULL



Representação por encadeamento – Implementação

Para implementar uma **fila por encadeamento** podemos utilizar 4 variações das **listas lineares (LL)**:

1. Simplesmente encadeada (LLSE)
2. Simplesmente encadeada com nó Descritor (LLSEcD)
3. Duplamente encadeada (LLDE)
4. Duplamente encadeada com nó Descritor (LLDEcD)

Representação por encadeamento – Implementação

Uma abordagem suficientemente adequada é utilizar uma **LLSEcD**, pois permite:

- Acesso direto ao primeiro e o último elemento da fila (existem ponteiros específicos no nó descritor)
- Consulta direta do tamanho da fila (contida no nó descritor)

Representação por encadeamento – Implementação

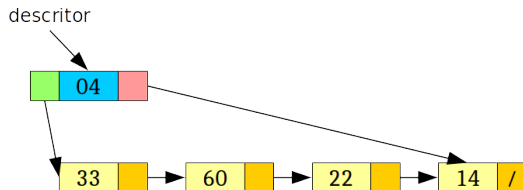


Figura: Implementação LLSEcD de uma fila.

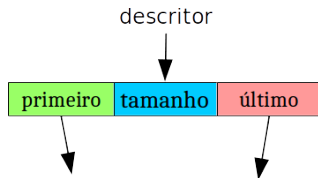


Figura: Conteúdo do nó descriptor.

Representação por encadeamento – Implementação

Operações fundamentais

A implementação **LLSEcD** de uma **fila** deve seguir a **política FIFO**, ou seja:

- A **inserção** sempre é realizada no **final** da fila
- A **remoção** sempre é realizada no **início** da fila

Neste contexto:

- A **inserção** é chamada de *enqueue* (enfileirar)
- A **remoção** é chamada de *dequeue* (desenfileirar)

Representação por encadeamento – Desafio

1. Estude a implementação LLSEcD disponível no repositório `1e05-filas/LLSE`
2. Elabore um diagrama de fluxo para cada uma das funções envolvidas
3. Modifique o código para implementar uma *fila por encadeamento*

<https://github.com/2022-2-INF0286-AED1/coursework>

Fila de prioridade



Fila de prioridade

Em computação é frequente a necessidade de se manter um conjunto de elementos **ordenados de acordo com uma certa prioridade**.

A **prioridade** possibilita que os elementos sejam obtidos por sua **importância** em relação aos demais elementos e não pela sua ordem de inserção.

Fila de prioridade

A **prioridade** pode ser expressa por:

- Uma categoria
- Um número
- a & b

Exemplos:

- Prioridade categórica:** {baixa, média, alta}
- Prioridade numérica:** {0, 1, 2, 3}
- Prioridade combinada:**
 - baixa \rightarrow {0, 1, 2, 3}
 - média \rightarrow {0, 1, 2, 3}
 - alta \rightarrow {0, 1, 2, 3}

Fila de prioridade

Numa **fila de prioridade** a **classificação de seus elementos** determina os resultados das operações de **inserção** e **remoção**.

Em relação à **chave**, há 2 tipos de fila:

1. **Ascendente** – Elementos podem ser inseridos em ordem arbitrária, mas apenas o elemento com o **menor valor** pode ser removido a cada vez
2. **Descendente** – Elementos podem ser inseridos em ordem arbitrária, mas apenas o elemento com o **maior valor** pode ser removido a cada vez

Fila de prioridade

As **operações fundamentais** sobre filas de prioridade são:

- Criar fila
- Inserir elemento de acordo com sua prioridade
- Remover elemento com maior/ menor prioridade
- Consultar elemento com maior/menor prioridade

Outras **operações auxiliares** são:

- Quantificar os elementos presentes na fila
- Identificar os elementos com uma certa prioridade
- Alterar a prioridade de determinado elemento
- Combinar duas filas de prioridade

Fila de prioridade – Implementação

Uma fila de prioridade pode ser implementada utilizando estruturas de dados como:

- Vetor ordenado
- Lista linear duplamente encadeada – LLDE ordenada

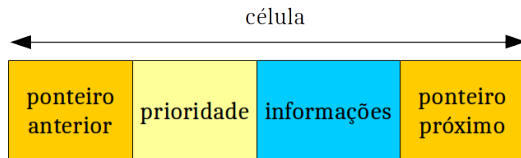
Fila de prioridade – Implementação por vetor ordenado

Um **vetor ordenado** é:

- Eficiente para consultar/remover o elemento com maior/menor prioridade
- Ineficiente na manutenção da ordem das prioridades
 - A manutenção das prioridades somente será eficiente quando “poucas” operações de inserção/remoção sejam realizadas
 - O **tamanho máximo da fila** não deve ser exageradamente grande

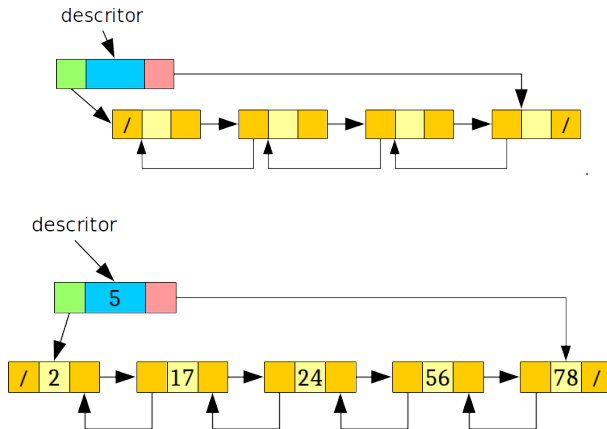
Fila de prioridade – Implementação por LLDE

Uma **Lista Linear Duplamente Encadeada (LLDE)** pode ser adaptada fazendo o campo **chave** corresponder à **prioridade** de um elemento:



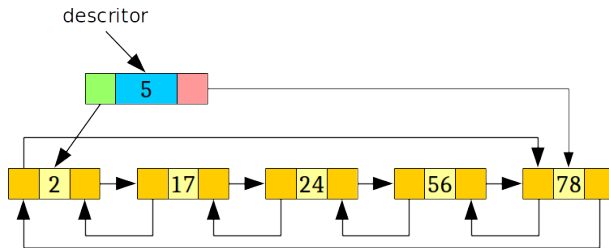
Fila de prioridade – Implementação por LLDE

A LLDE pode ter um *nó descritor* para proporcionar mais eficiência em algumas operações:



Fila de prioridade – Implementação por LLDE

A LLDE pode ser circular:



Fila de prioridade – Outras implementações


Pesquise sobre outras estruturas de dados que podem ser utilizadas para implementar uma **fila de prioridade**:

- a. Heap binário
- b. Bounded Height Priority Queue (fila de prioridade com altura limitada)
- c. Binary Search Tree (árvore binária de busca)
- d. Fibonacci Heap
- e. Pairing Heap

Saiba mais ...

youtu.be

- Canal do [Prof. Wanderley de Souza Alencar](#) (INF/UFG)



Wanderley de Souza Alencar
@wanderleydesouzaalencar7933
329 subscribers

[HOME](#)
[VIDEOS](#)
[PLAYLISTS](#)
[COMMUNITY](#)
[CHANNELS](#)
[ABOUT](#)

[filas](#)

Listas Lineares

1ª Parte

por Wanderley de Souza Alencar
wanderleydesouzaalencar7933

Universidade Federal de Goiás - UFG
Instituto de Informática - INF

Wanderley de Souza Alencar
@wanderleydesouzaalencar7933

September 27, 2020

16:48

Aula sobre Listas Lineares (1ª parte)

Wanderley de Souza Alencar • 530 views • 2 years ago

Videoaula sobre "Listas Lineares" para os(as) estudantes dos cursos de bacharelado do Instituto de Informática da Universidade Federal de Goiás, período noturno do 1º semestre letivo de...

AE01 - Aula 00

Apresentação da disciplina

Wanderley de Souza Alencar
wanderleydesouzaalencar7933

Universidade Federal de Goiás - UFG
Instituto de Informática - INF

Wanderley de Souza Alencar
@wanderleydesouzaalencar7933

22 de maio de 2021

41:48

Algoritmos e Estruturas de Dados 2 - 2022/1 [Aula de apresentação da disciplina].

Wanderley de Souza Alencar • 93 views • 5 months ago

Disciplina "Algoritmos e Estruturas de Dados 2", semestre letivo 2022/1 do Instituto de Informática da Universidade Federal de Goiás (INF/UFG). Aula de apresentação da disciplina para...

Listas Lineares Simplesmente Encadeadas (LLSE)

Wanderley de Souza Alencar • 477 views • 4 years ago

Aula sobre Listas Lineares Simplesmente Encadeadas (LLSE) para a turma da disciplina "Algoritmos e Estruturas de Dados - 1º do Curso de Bacharelado em Sistemas de Informação, Universidade...

Saiba mais ...

1. FORBELLONE, A. L. V.; EBERSPACHER, H. F. *Lógica de programação – A construção de algoritmos e estruturas de dados*. 3 ed. São Paulo: Prentice Hall. 2005. *Cap. 07*.
2. GOODRICH, M. T.; TAMASSIA, R. *Estruturas de dados e algoritmos em Java*. 4 ed. Porto Alegre: Bookman. 2007. *Cap. 3, 5, 6, 8*.
3. CORMEN, T. H.; et al. *Introduction to Algorithms*. 3 ed. MIT Press. 2009. *Caps. 10, 12, 19*.
4. SEDGEWICK, R.; WAYNE, K. *Algorithms*. 4 ed. Addison-Wesley. 2011. *Cap. 3, 5*.
5. STEPHENS, R. *Essential Algorithms: A Practical Approach to Computer Algorithms*. 1 ed. John Wiley & Sons. 2013. *Cap. 10*.
6. FERRARI, R.; et al.. *Estruturas de dados com jogos*. 1 ed. São Paulo: Elsevier. 2014. *Caps 3, 4, 5*.