Tipos Abstratos de Dados
FILAS estáticas

Wanderley de Souza Alencar
adaptado por Raphael Guedes

October 3, 2024

Fila – no Dicionário

Um alinhamento de uma série de indivíduos, ou objetos, em sequência, de modo que um esteja imediatamente atrás do outro; fileira.

Sequência de pessoas dispostas de maneira alinhada pelos mais diversos critérios (ordem de chegada, altura, etc.) e para os mais diversos objetivos.

Fila – na Computação

É um tipo de estrutura de dados dinâmica que permite a realização de operações de inserção e de remoção de elementos (ou objetos) de tal maneira que se obedeça ao seguinte critério:

"*O elemento a ser removido é sempre aquele que, dentre todos os presentes na fila em determinado instante, foi o primeiro a ser nela inserido.*"

Fila – na Computação

É um tipo de estrutura de dados, de tamanho variável, em que se estabelece a política FIFO (*First In, First Out*) para a inserção/remoção de elementos nela presentes.

Fila – na Computação

Assim podemos conceber a ideia de *filas* de:

- pessoas;
- objetos concretos;
- objetos abstratos (números, imagens, etc.).

Fila – na Computação

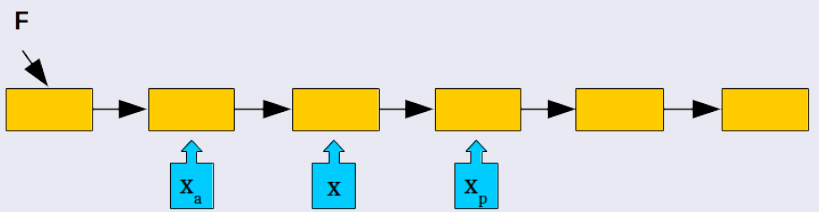
Assim podemos conceber a ideia de *filas* de:

- pessoas para atendimento num banco;
- arquivos aguardando impressão, numa determinada impressora;
- de dados aguardando transmissão numa linha serial de comunicação;
- de processos esperando para ter acesso à CPU (num SO multitarefa);

Fila – na Computação

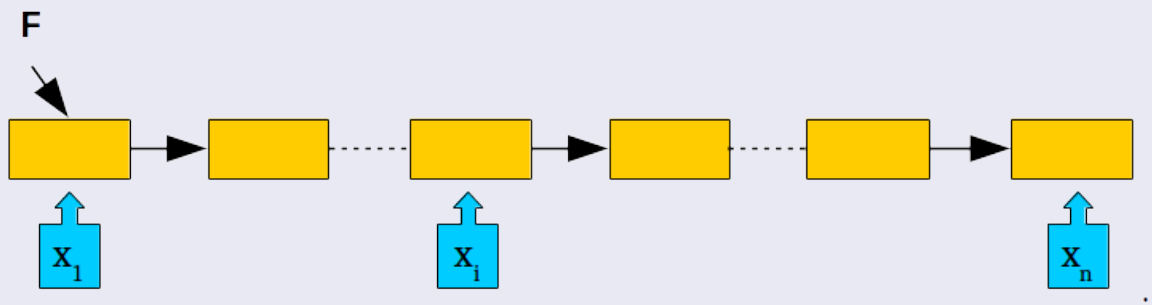
Uma fila é uma estrutura de dados formada por uma sequência finita de elementos – células, componentes, nodos ou nós – organizada de tal maneira que reflita uma relação linear de ordem entre eles.

A *relação linear de ordem* define que um elemento x tenha um, e somente um, elemento x_a que lhe seja imediatamente anterior e outro, também único, x_p que lhe seja imediatamente posterior.



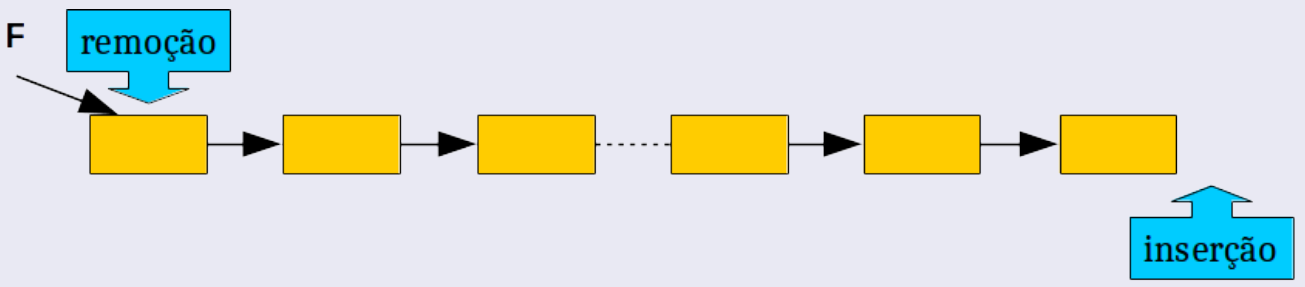
Fila – na Computação

Evidentemente, o primeiro elemento da fila, x_1, não possui elemento anterior e o último elemento, digamos x_n, não possui elemento posterior, considerando que $n \in \mathbb{N}^*$.



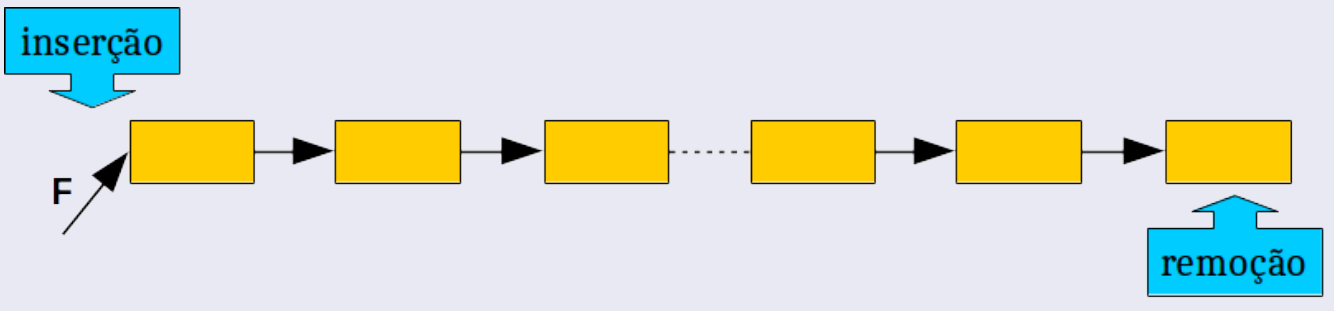
Fila – na Computação

Para obedecer à política FIFO, as operações de *inserção* e *remoção* devem obrigatoriamente serem realizadas em "*extremidades opostas*" da fila:



Fila – na Computação

Para obedecer à política FIFO, as operações de *inserção* e *remoção* devem obrigatoriamente serem realizadas em "*extremidades opostas*" da fila:



Operações Fundamentais

...

As operações fundamentais sobre filas são:

1. criar uma fila (inicialmente *vazia*);
2. determinar a quantidade de elementos presentes na fila;
3. inserir um elemento na fila;
4. remover um elemento da fila.

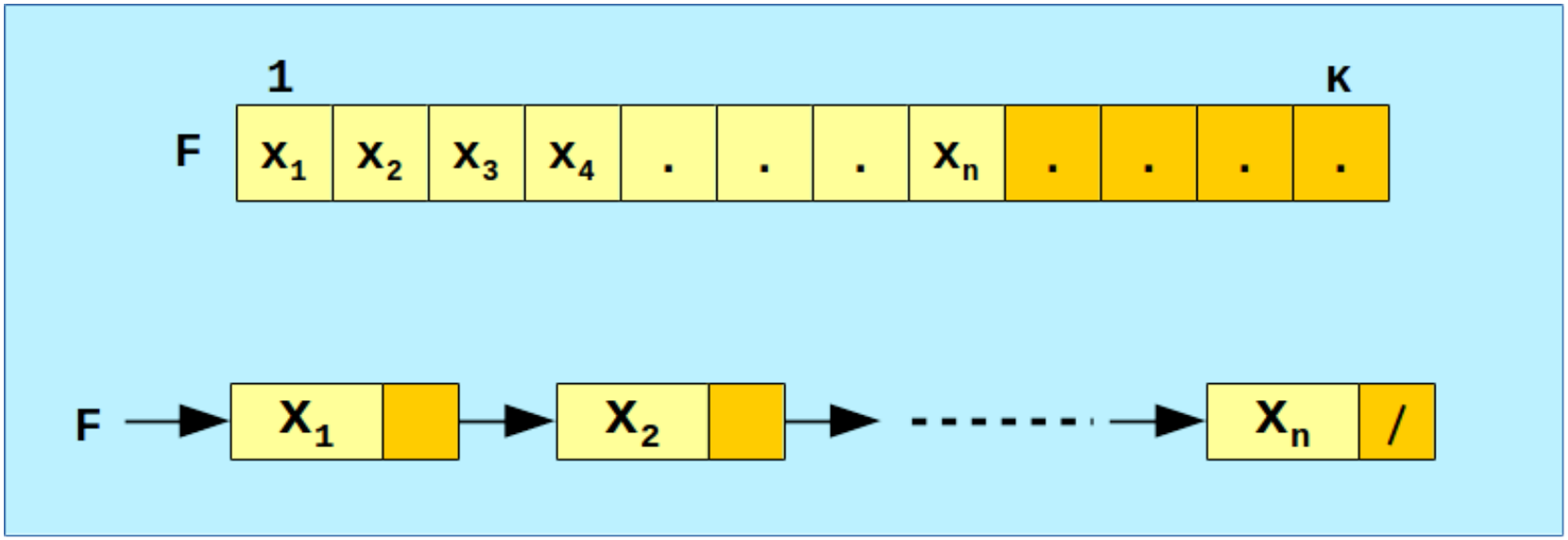
Operações Fundamentais

...

Outras operações sobre filas podem ser:

1. mostrar um determinado elemento da fila (ou todos os elementos);
2. consultar se uma determinada *chave* está num dos elementos da fila;
3. concatenar duas filas, gerando uma única fila;

Representações



Representações de Filas

Há, esssencialmente, duas maneiras para representar computacionalmente uma fila:

1. por contiguidade (alocação estática);
2. por encadeamento (alocação dinâmica).

Representações

A escolha da maneira mais conveniente a ser utilizada deve ser balizada por fatores como:

1. qual o tamanho do armazenamento exigido por elemento da fila?
2. quais os tamanhos, mínimo e máximo, esperados para a fila?
3. qual a frequência de realização de cada operação prevista?

Por Contiguidade

A proposta básica desta implementação é a sua simplicidade: utilizar um vetor para conceber um TAD que represente a estrutura de dados *fila*.

A implementação é facilitada, mas paga-se o preço ao optar por previamente alocar um vetor que, nem sempre, está sendo completamente utilizado pela fila ou que pode, em certo momento, ser insuficiente para armazenar a quantidade elementos da fila.

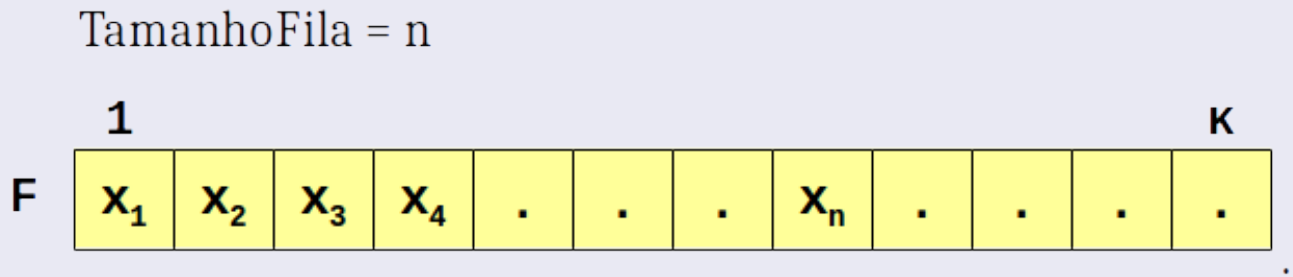
Por Contiguidade

Dada uma fila $F = (x_1, x_2, \ldots, x_n)$, com $n \in \mathbb{N}^*$, utiliza-se um vetor para representá-la, além de duas variáveis de controle:

- `TamanhoFila`: conterá o número de elementos presentes na fila naquele momento;
- `TamanhoMaximoFila`: conterá o número máximo de elementos que a fila poderá conter, correspondendo ao tamanho do vetor que foi definido para representá-la, digamos, $\kappa \in \mathbb{N}^*$.

Por Contiguidade

Vetor, de tamanho κ, representando uma fila F:



Por Contiguidade

Note que:

- a fila ocupa as posições de 1 a n do vetor;
- as posições de $(n+1)$ a κ correspondem à área livre do vetor, ou seja, área destinada ao crescimento da fila e que está previamente alocada;
- a fila está limitada ao tamanho do vetor, que é κ;

Por Contiguidade

Note que para conservar a política FIFO, é necessário que:

- uma inserção seja realizada na posição $(n+1)$, desde que $n < \kappa$;
- a remoção deve ser realizada na posição 1, desde que $n \geq 1$.

Declaração das estruturas para lista, por contiguidade:

```
1  //
2  // Arquivo lista.h (parte 01)
3  //
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define SUCESSO 1
8  #define FALHA -1
9  #define CHAVE_INVALIDA 0
10
11 #define TAMANHO_MAXIMO_FILA 100
```

Representações

Declaração das estruturas para lista, por contiguidade:

```
1  //
2  // Arquivo lista.h (parte 02)
3  //
4  typedef struct {
5        unsigned int chave;
6        unsigned int dado;
7  } Elemento;
8
9  typedef struct fila {
10     Elemento elementos [TAMANHO_MAXIMO_FILA];
11     unsigned int tamanho;
12 } Fila;
```

Declaração das estruturas para lista, por contiguidade:

```
1  //
2  // Arquivo lista.h (parte 03)
3  //
4
5  int criarFilaVazia (Fila * fila);
6  int criarFilaChave (Fila * fila, Elemento elemento);
7  int tamanhoFila (Fila fila);
8  void mostrarElemento(Elemento elemento);
9  void mostrarFila (Fila fila);
10 Elemento consultaPosicao(Fila fila, unsigned int intPosicao);
11 int insFinal (Fila * fila, Elemento elemento);
12 Elemento remInicio (Fila * fila);
```

Representações

Criar uma fila inicialmente vazia, por contiguidade:

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

Representações

Criar uma fila com um único elemento, por contiguidade:

```
1 int criarFilaChave(Fila * fila, Elemento elemento) {
2     fila -> elementos[0] = elemento;
3     fila -> tamanho = 1;
4     return (SUCESSO);
5 }
```

Representações

Determinar o tamanho da fila, por contiguidade:

```
1 int tamanhoFila(Fila fila ) {
2
3       if (fila.tamanho >= 0) {
4           return(fila .tamanho);
5       }
6       else {
7           return(FALHA);
8       }
9 }
```

Mostrar um elemento da fila, por contiguidade:

```
1 void mostrarElemento(Elemento elemento) {
2     printf("Chave.....: %u\n", elemento.chave);
3     printf("Dado.....: %u\n", elemento.dado);
4 }
```

Mostrar toda a fila, por contiguidade:

```
1  void mostrarFila(Fila fila) {
2      unsigned int i;
3
4      if (fila.tamanho == 0) {
5          printf("Atencao: A fila esta vazia.\n");
6      }
7      else {
8          printf("A fila linear possui %u elementos.\n\n", fila.tamanho);
9          for (i = 0; (i < fila.tamanho); i++) {
10             printf("Elemento n.: %u\n", (i+1));
11             mostrarElemento(fila.elementos[i]);
12         }
13     }
14 }
```

Consultar elemento pela posição dele, por contiguidade:

```
1  Elemento consultaPosicao(Fila fila , unsigned int intPosicao) {
2
3      Elemento elementoResultado;
4
5      if ((intPosicao > 0) && (intPosicao <= fila .tamanho)) {
6          elementoResultado = fila.elementos[intPosicao - 1];
7      }
8      else {
9          elementoResultado.chave = CHAVE_INVALIDA;
10     }
11     return(elementoResultado);
12 }
```

Inserir elemento no final da fila, por contiguidade:

```
1  int insFinal (Fila * fila , Elemento elemento) {
2      unsigned int i;
3      Elemento auxiliar;
4
5      if (fila->tamanho == TAMANHO_MAXIMO_FILA) {
6          return(FALHA);
7      }
8      else {
9          fila->elementos[fila->tamanho] = elemento;
10         fila->tamanho++;
11         return(SUCESSO);
12     }
13 }
```

Remover elemento no início da fila, por contiguidade:

```
1  Elemento remInicio (Fila * fila ) {
2      unsigned int i;
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 (i = 0; (i < (fila->tamanho - 1)); i++) {
12             fila->elementos[i] = fila->elementos[i+1];
13         }
14         fila->tamanho--;
15         return(elementoResultado);
16     }
17 }
```

Saiba Mais...

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. and STEIN, C.; *Algoritmos* – traduação da 2 edição norte-americana; Elsevier; 2002. *Capítulos: 12, 19 e 20*;
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. and STEIN, C.; *Introduction to Algorithms*; 3^{rd} edition; MIT Press; 2009. *Chapters: 10, 12 and 19*.

Saiba Mais...

- FERRARI, Roberto; RIBEIRO, Marcela X.; DIAS, Rafael L. e FALVO, Maurício; *Estruturas de dados com jogos*; 1 ed.; São Paulo: Elsevier; 2014. *Capítulos: 03, 04 e 05*;
- FORBELLONE, André Luiz V. e EBERSPACHER, Henri Frederico; *Lógica de programação – a construção de algoritmos e estruturas de dados*; 3 ed.; São Paulo: Prentice Hall do Brasil; 2005. *Capítulo 07*;
- GOODRICH, Michael T. e TAMASSIA, Roberto; *Estruturas de dados e algoritmos em Java*; 4 ed.; Porto Alegre: Bookman; 2007. *Capítulos: 03, 05, 06 e 08*.

Saiba Mais...

- SCZWARCFITER, Jayme Luiz e MARKEZON, Lilian; *Estruturas de dados e seus algoritmos*; 1^a ed.; Rio de Janeiro: LTC; 1994. *Capítulo 06*;
- SEDGEWICK, R. and WAYNE, K.; *Algorithms*; 4^{th} edition; Addison-Wesley; 2011. *Chapters: 03 and 05*;
- STEPHENS, Rod; *Essential Algorithms: A Practical Approach to Computer Algorithms*; 1^{st} edition; John Wiley & Sons; 2013. *Chapter 10*.