ESTRUTURAS DE DADOS

prof. Alexandre César Muniz de Oliveira

Sugestão bibliográfica:

- **ESTRUTURAS DE DADOS USANDO C**
Aaron M. Tenenbaum, et alli

- **DATA STRUCTURES, AN ADVANCED APPROACH USING C**
Jeffrey Esakov & Tom Weiss

- **ESTRUTURAS DE DADOS E ALGORITMOS EM JAVA (2ED)**
Michael Godrich & Roberto Tamassia

RECURSIVIDADE

1. Definição

Um processo é dito recursivo quando ele pode ser definido em função de si mesmo. Pode-se modelar um processo em forma de função ou procedimento chamado recursivo.

2. Exemplos: *(obs.: Interrupção processo)*

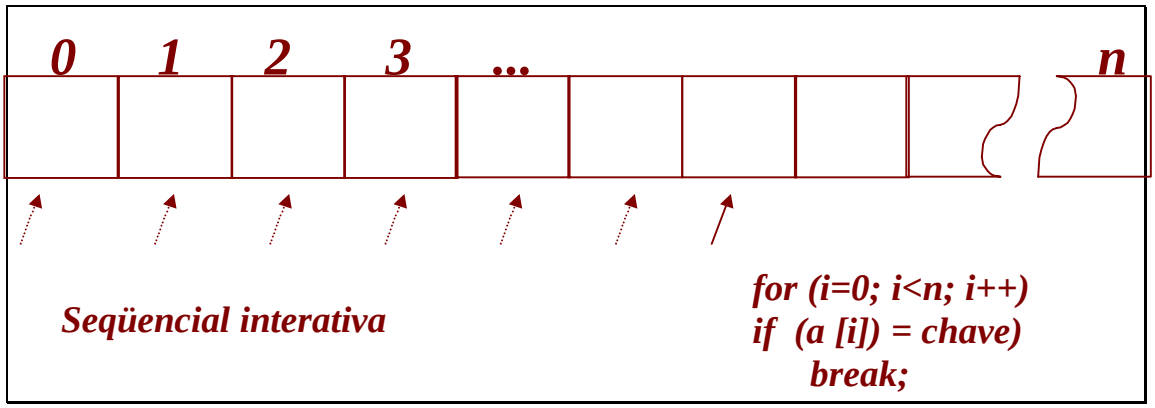
a) fatorial ∴ n! = n * (n - 1)!
= n * (n - 1) * (n - 2)!
...

b) multiplicação N)
∴ m * n = m * (n - 1) + m
= m * (n - 2) + m + m
= m * (n - 3) + m + m + m

c) Seqüência de Fibonacci
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
n = 0 → 0; n = 1 → 1; para n > 1

∴fib (n) = fib (n - 2) + fib (n - 1)
fib (n - 3) + fib (n - 4)

3. Busca seqüencial recursiva

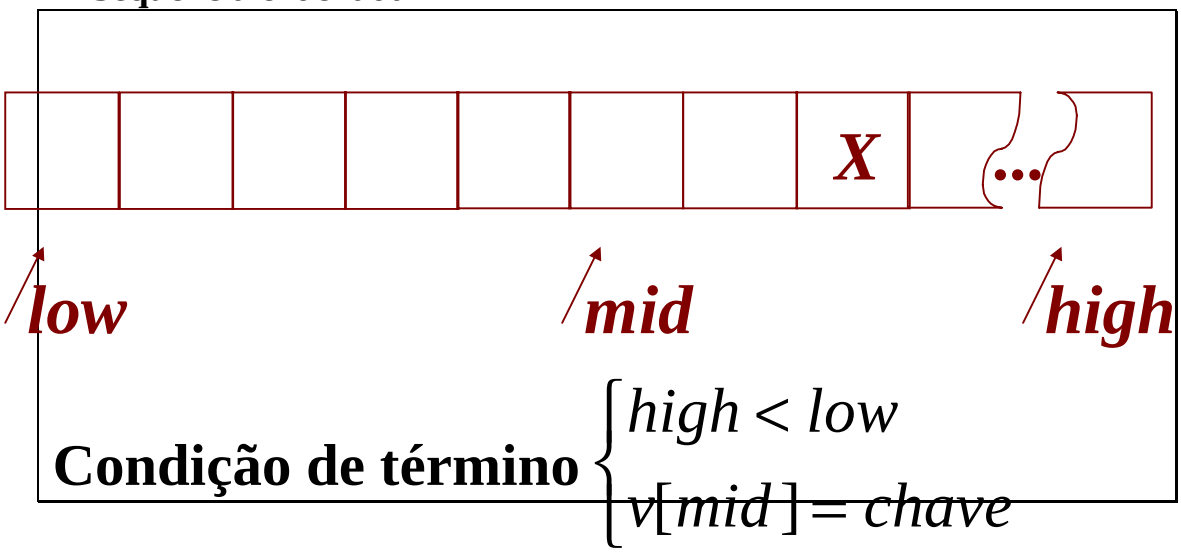


RECURSIVIDADE

Recursivamente
necessário definir função

```
pesqseq (vetor, pos, fim, chave)
   if (pos > fim) return (NULL);
   if (vetor [pos] == chave)
      return (&vetor);
   resp = pesqseg (vetor, pos+1, fim, chave)
   return (resp);
fim_pesqseg;
```

4. Busca binária recursiva
Seqüência ordenada



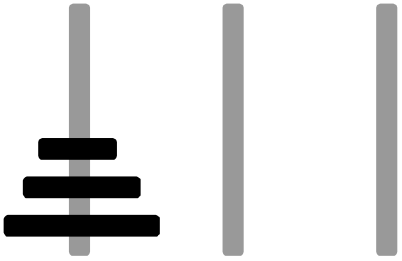
Condição de término $\begin{cases} high < low \\ v[mid] = chave \end{cases}$

Procurar : novos limites
executar mesmo procedimento ***(recursivo)***

```
pesqbin (vetor, chave, low, high)
   mid : integer;    resp; ^vetor;
   if (low > high )    return (null);
   mid = (low + high ) / 2;
   if (chave == vetor [mid]
      return (&vetor [mid]);
   if (chave < vetor [mid])
      resp = pesqbin ( vetor , chave, low, mid - 1);
   else
      resp = pesqbin (vetor, chave, mid + 1, high);
   return (resp);
```

RECURSIVIDADE

5. Torres de Hanói



Dado 3 pinos A, B, C com n discos de diferentes tamanhos, inicialmente em A, movê-los para C. Restrição:

Objetivo:

Mover todos os discos de A→ C, usando o auxiliar B

N = 1 A B C → A B C

N = 2 $\overline{A}$ B C → A B C → A B C → A B C

N = 3 $\overline{\overline{A}}$ B C → A B C → A B C → A B C

A $\overline{B}$ $\overline{\overline{C}}$ → A $\overline{B}$ $\overline{\overline{C}}$ → A B $\overline{\overline{C}}$ → A B $\overline{\overline{C}}$

Para n = 1 move o disco de A para C

Para n > 1 move os (n - 1)
discos menores de A para B;
move o disco A para C
move os (n - 1) de B para C

Torres de Hanói

O problema consiste em resolver o objetivo primário que é definido em função de resolver objetivos secundários idênticos ao primário.

RECURSIVIDADE

Algoritmo

```
programa Hanói;
   lê número discos ( n > 0);
   fhanoi ( 'A', 'B', 'C', N);
fim;

fhanoi (origem, aux, destino, n)
  caracter: origem, aux, destino;
  integer: n;
  {       se (n = 1)
                  escreva (origem, 'para',  destino);
          senão {
                  fhanoi (origem, destino, aux, n - 1);
                  fhanoi (origem, aux, destino, 1);
                  fhanoi (aux, origem,destino, n-1);
          }
fim;
```

FILA

1. Conceito

Conjunto de itens a partir do qual pode-se eliminar itens numa extremidade *(início)* e pode-se inserir itens em outra *(final)*

	I→ 0 ← F
insere (A)	I→ A ←F
insere (B)	I→ A B ← F
insere (C)	I→A B C ←F
remove (C)	I→ A B ←F

FIRST IN FIRST OUT (FIFO)

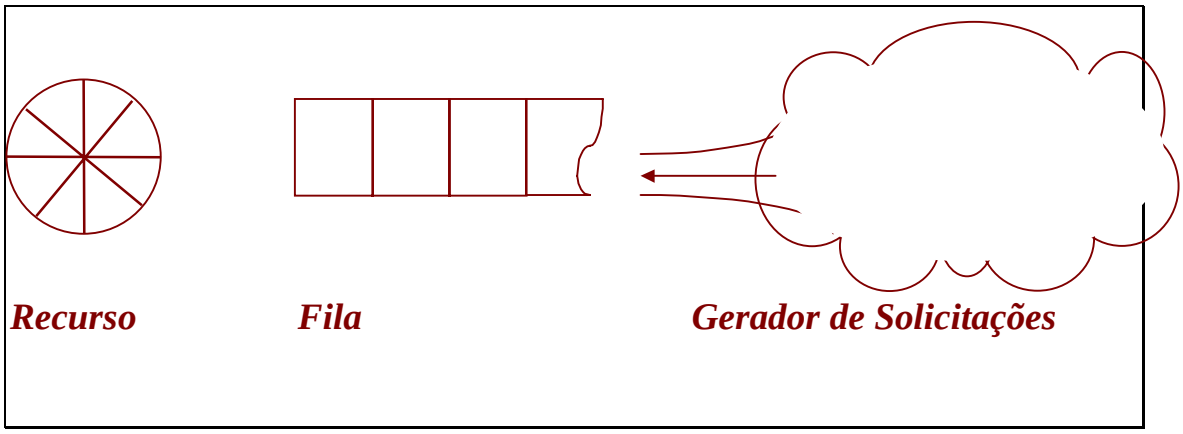
2. Primitivas

Insert (Q, X) **sempre**
X = remove (Q) **lenght (Q) > 0**

Empty (Q) $\begin{cases} V \\ F \end{cases}$

FILA

3. Aplicação: Contenção de recurso usando fila simples



Inicializa parâmetros

$$\begin{cases} prob(chegada) \\ prob(saida) \\ tempodesimulaçao \end{cases}$$

```
Enquanto durar simulação faça
    Chegada (prob-chegada)
    Se chegou então
        Se recurso-ocupado então
            insere na fila
        Senão utiliza-Recurso ( ),
    Se recurso-ocupado então
        Saída (prob-saída)
        Se saiu se fila não vazia
            Remove da Fila
            utiliza recurso ( )
Fim
```

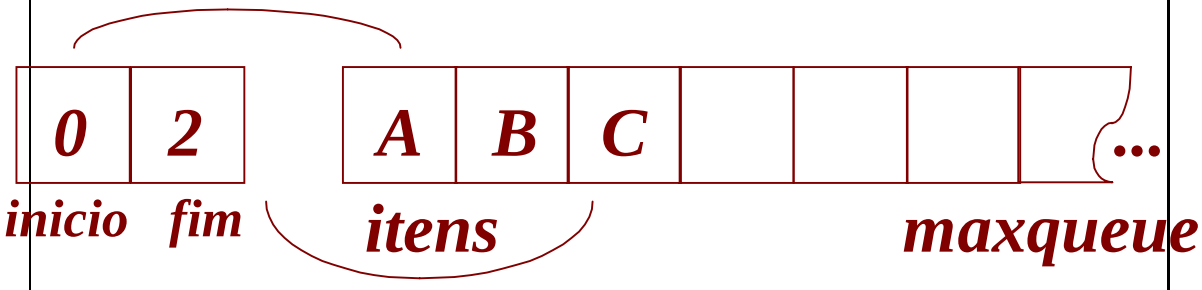
FILA

4. Implementação

```
# define MAXQUEUE 100
typedef {
    int itens [MAXQUEUE];
    int inicio, fim;
}queue;
```

```
queue q;
```

$$\begin{cases} inicio: \\ q.inicio = 0 \\ q.\,fim = 1 \end{cases}$$

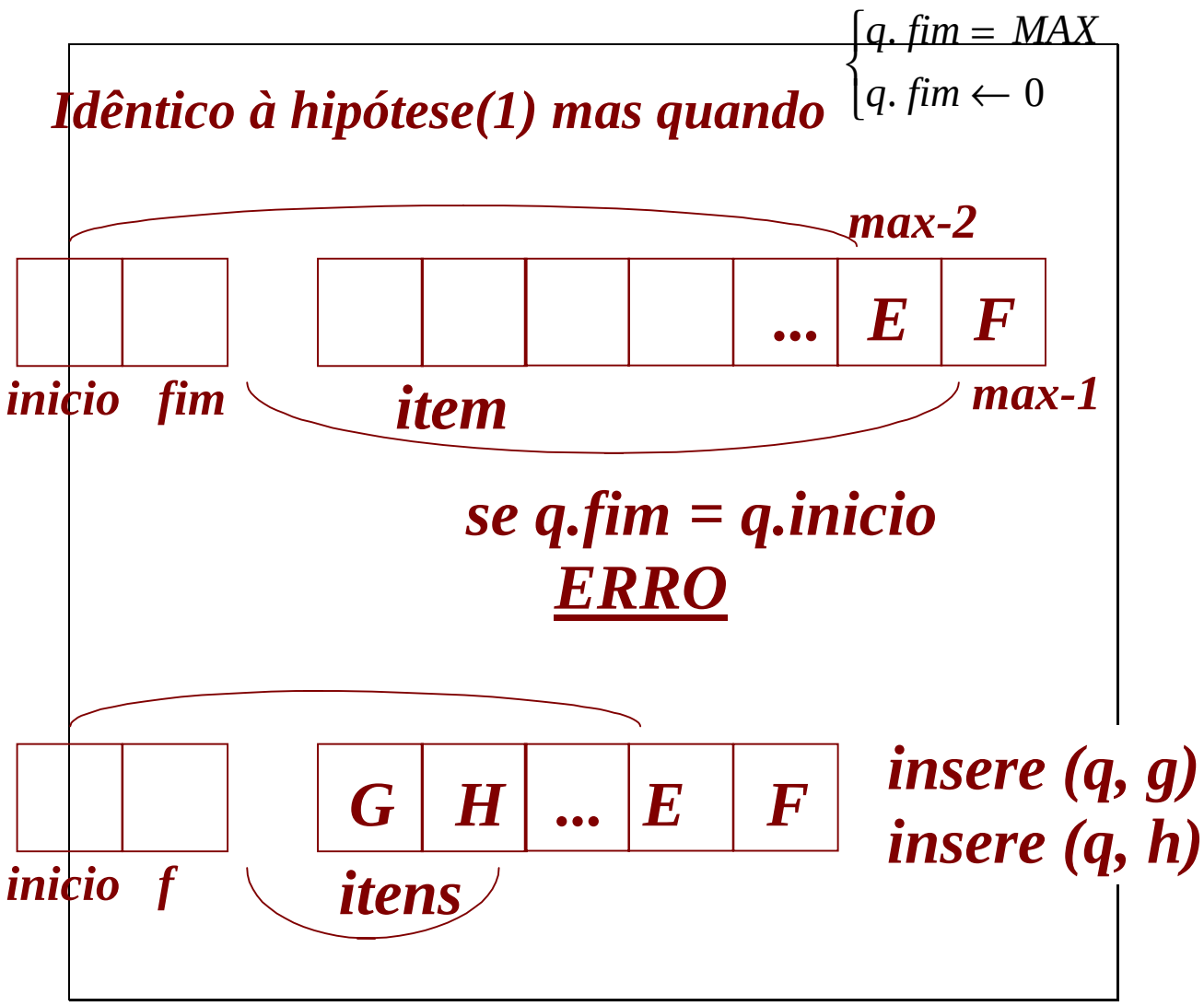



```
Hipótese ( 1 )
   Vazia se  q.inicio > q.fim
    insere ( ) q. Item[++q.fim] = x
    remove ( )  x = q.item [q.inicio ++];
    problemas:

Hipótese (2 )
Vazia se q.fim = -1
q.inicio = 0

remove:  x = a.itens[0];
for ( I = 0; I< q.fim; I++)
q.items[I] = q.item [I+1];
q.item --;
```

FILA

4.1. Implementação em vetor circular

```
struct queue{
   int itens[MAX];
   int inicio, fim;
} q;
```



Sempre que inserir itens: fim ++
Sempre que remover itens: inicio ++
Vazia se q.inicio = q.fim
Inicialmente q.inicio = q.fim = MAX - 1

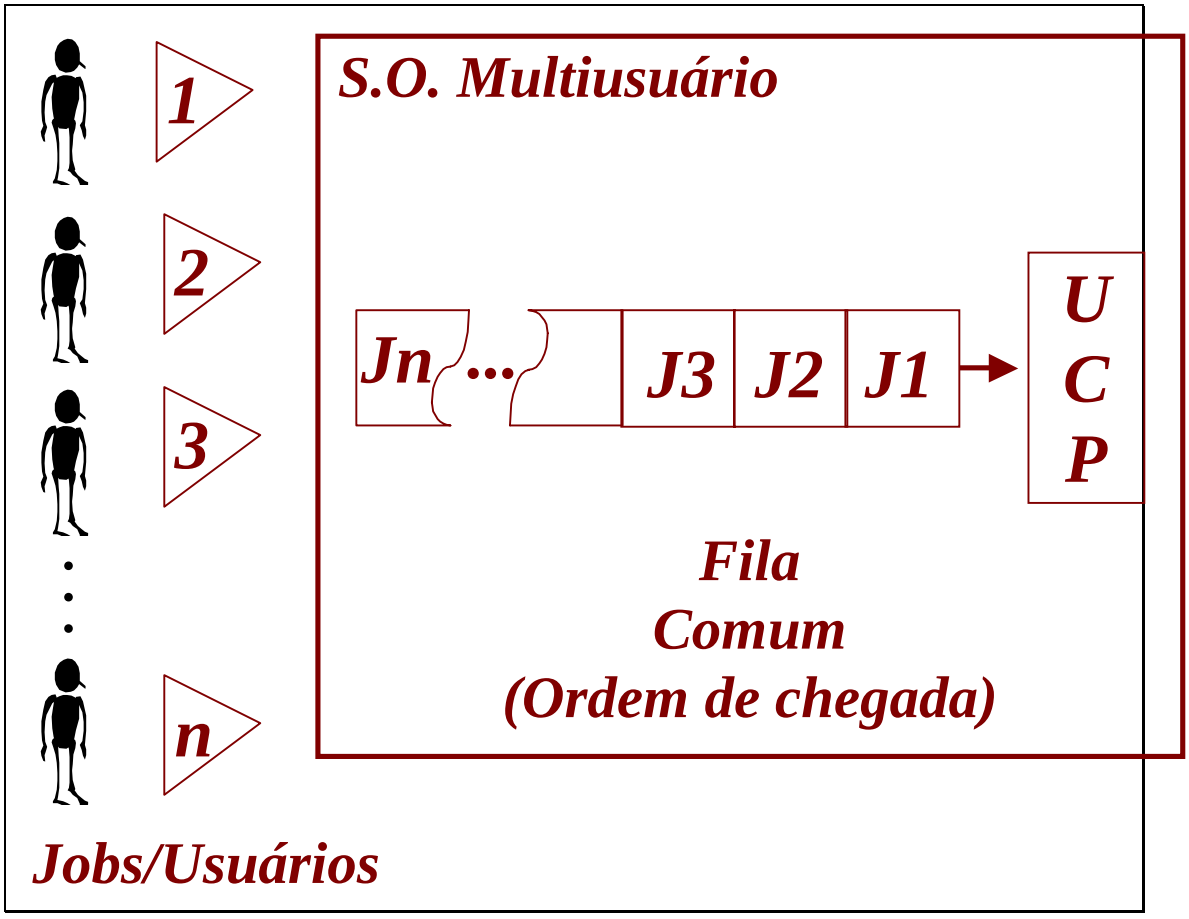
FILA

Implementação das primitivas

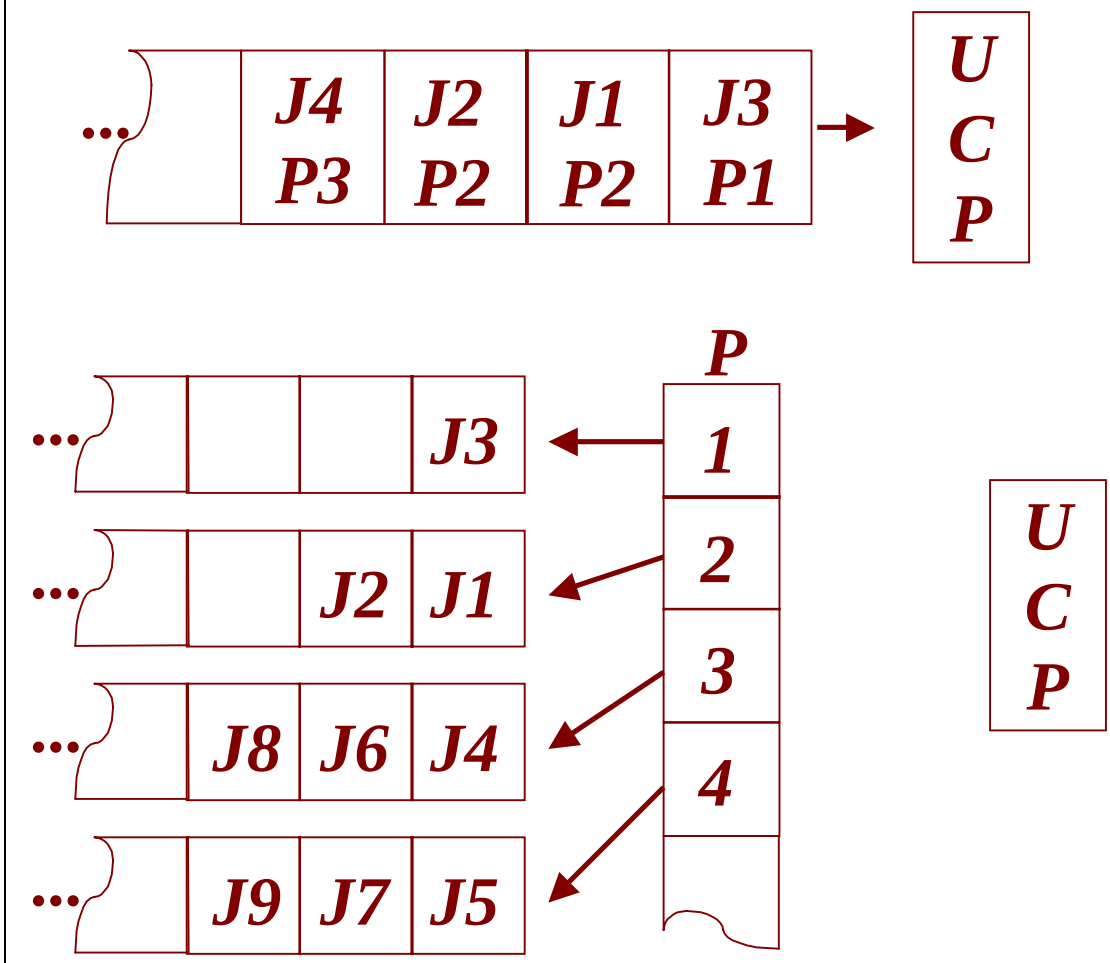
```
void insert (struct queue *pq, int x)
{
        if (pq→fim == MAX -1) pq→fim = 0;
        else (pq → fim)++;
        if (pq → fim == pq → inicio)    {
                puts ("queue overflow");
                exit (0);
        }
        pq →itens [pq →fim] = x;
}

int remove (struct queue * pq)
{
        if (empty(pq)  {
                puts ("queue underflow');
                exit (0);
        }
        if (pq→inicio==MAX-1)
                pq→inicio = 0;
        else    pq → inicio ++;
        return (pq → itens[pq →inicio]);
}

int empty (struct queue * pq)
{
        return  ((pq→inicio = pq→fim)?TRUE:FALSE)
}
```

FILA

5. Fila de Prioridades



Ou considerando prioridades



Múltiplas filas, cada qual contendo somente jobs com a mesma prioridade

Remoção: Retirar da fila o elemento com maior prioridade.
Reagrupar a fila ou marcar a posição "desocupada"

Inserção: Inserir o elemento na fila correspondente, para o final. Se estourar o limite, procurar lugar marcado ou inserir no local exato.

Obs: Constantes reorganização sugere outro tipo de estrutura.