

AULA 18

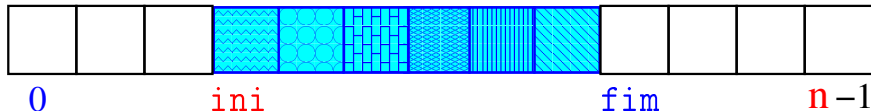
Mais implementações de filas

PF 5.3

<http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html>

Fila implementada em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.



O índice `ini` indica o `primeiro` da fila.

O índice `fim-1` indica o `último` da fila.

`fim` é a `primeira posição vaga` da fila.

A fila está `vazia` se "`ini == fim`".

A fila está `cheia` se "`fim == n`".

Interface item.h

```
/*  
 * item.h  
 */  
typedef int Item;
```

Interface queue.h

```
/*
 * queue.h
 * INTERFACE: funcoes para manipular uma
 * fila
 */
void queueInit(int);
int queueEmpty();
void queuePut(Item);
Item queueGet();
void queueFree();
```

Implementação queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "item.h"
/*
 * FILA: implementacao em vetor
 */
static Item q;
static inicio;
static fim;
```

Implementação queue.c

```
void
queueInit(int n)
{
    q = mallocSafe(n * sizeof(Item));
    inicio = fim = 0;
}

int
queueEmpty()
{
    return inicio == fim;
}
```

Implementação queue.c

```
void
queuePut(Item item)
{
    q[fim++] = item;
}

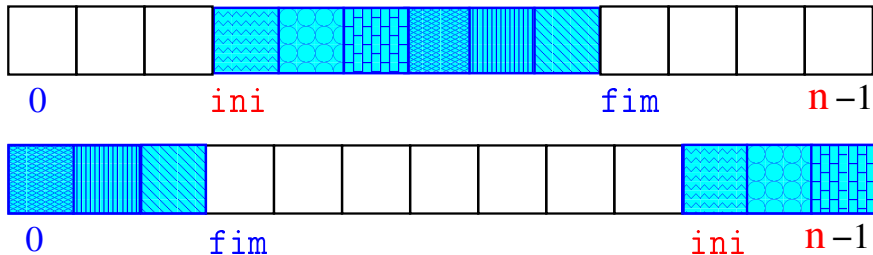
Item
queueGet()
{
    return q[inicio++];
}
```


Implementação queue.c

```
void  
queueFree()  
{  
    free(q);  
}
```

Uma implementação circular em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.



Temos que $0 \leq ini < n$ $0 \leq fim < n$

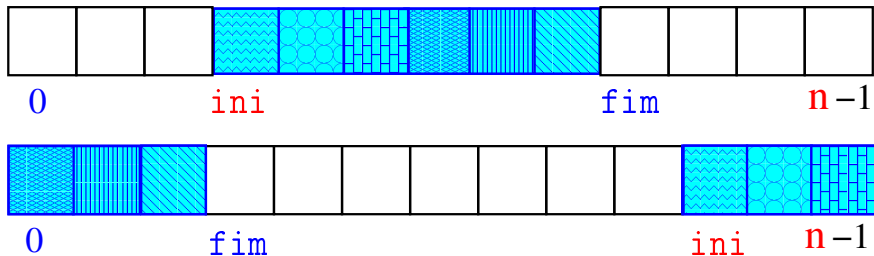
Não supomos $ini \leq fim$

O índice ini indica o primeiro da fila.

fim é a primeira posição vaga da fila.

Uma implementação circular em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.



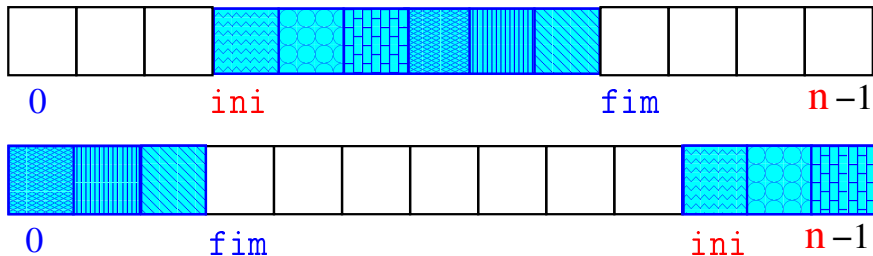
A fila está **vazia** se “ $ini == fim$ ”

A fila está **cheia** se “ $fim+1 == ini$ ” ou

“ $fim+1 == n$ e $ini == 0$ ”

Uma implementação circular em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.

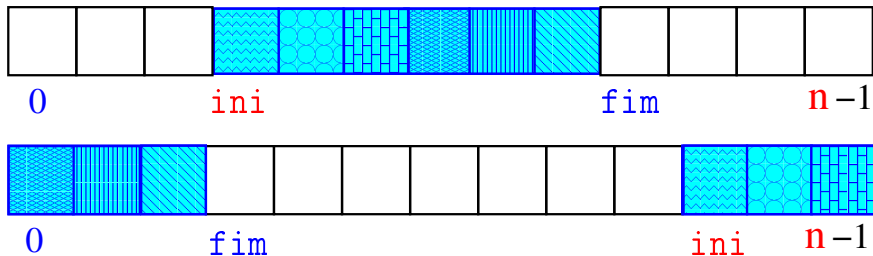


A fila está **vazia** se " $ini == fim$ "

A fila está **cheia** se " $(fim+1) \% n == ini$ "

Uma implementação circular em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.

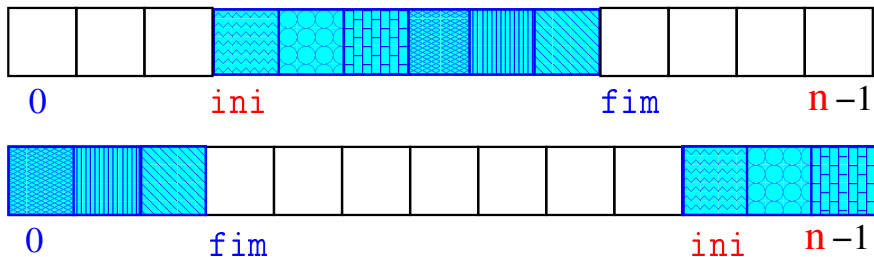


A posição fim **sempre está desocupada**

Isto é importante para distinguir fila **vazia** de **cheia**

Uma implementação circular em um vetor

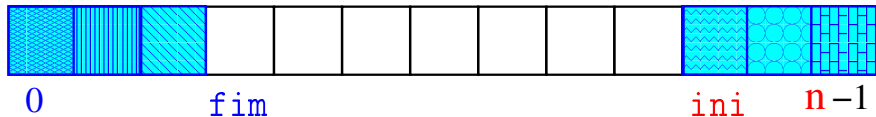
A fila será armazenada em um vetor $q[0 \dots n-1]$.



Para *remover* (= *dequeue* = *get*) um elemento faça

```
x = q[ini++];  
if (ini == n) ini = 0;
```

Uma implementação circular em um vetor



Para *inserir* (= *queue* = *put*) um elemento faça

```
if((fim+1)%n == ini) {  
    printf("Fila cheia!\n");  
    exit(EXIT_FAILURE);  
}  
q[fim++] = x;  
if(fim == n) fim = 0;
```

Interface item.h

```
/*  
 * item.h  
 */  
typedef int Item;
```


Interface queue.h

```
/*  
 * queue.h  
 * INTERFACE: funcoes para manipular uma  
 * fila  
 */  
void queueInit(int);  
int queueEmpty();  
void queuePut(Item);  
Item queueGet();  
void queueFree();
```

Implementação queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "item.h"
/*
 * FILA: implementacao em vetor
 */
static Item q;
static int n; /* tamanho do vetor */
static int inicio;
static int fim;
```

Implementação queue.c

```
void
queueInit(int N)
{
    n = N + 1;
    q = mallocSafe(n * sizeof(Item));
    inicio = fim = 0;
}

int
queueEmpty()
{
    return inicio == fim;
}
```

Implementação queue.c

```
void
queuePut(Item item)
{
    if ((fim+1)%n == ini) {
        printf("Fila vai transbordar!\n");
        exit(EXIT_FAILURE);
    }
    q[fim++] = item;
    if (fim == n) fim = 0;
}
```

Implementação queue.c

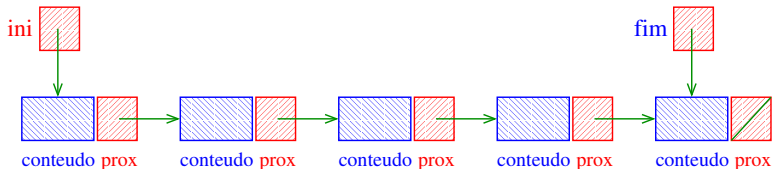
Item

```
queueGet()  
{  
    int i = ini;  
    ini = (ini + 1) % n;  
    return q[i];  
}
```

```
void  
queueFree()  
{  
    free(q);  
}
```

Fila implementada em uma lista encadeada

A fila será armazenada em uma lista encadeada.



O ponteiro **ini** aponta para o **primeiro** da fila .

O ponteiro **fim** aponta para o **último** da fila .

ini->**conteudo** é o **primeiro elemento** da fila .

fim->**conteudo** é o **último elemento** da fila .

A fila está **vazia** se “**ini** == NULL” .

Implementação queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "item.h"
/*
 * FILA: uma implementacao em lista
 * encadeada
 */
typedef struct queueNode* Link;
struct queueNode {
    Item conteudo;
    Link prox;
};
static Link ini, fim;
```

Implementação queue.c

```
static Link  
new(Item conteudo, Link prox)  
{  
    Link p = mallocSafe(sizeof *p);  
    p->conteudo = conteudo;  
    p->prox = prox;  
    return p;  
}
```


Implementação queue.c

```
void
queueInit(int n)
{
    ini = NULL;
}

int
queueEmpty()
{
    return ini == NULL;
}
```

Implementação queue.c

```
void
queuePut(Item item)
{
    if (ini == NULL)
    {
        ini = fim = new(conteudo, NULL);
        return;
    }
    fim->prox = new(conteudo, NULL);
    fim = fim->prox;
}
```

Implementação queue.c

Item

queueGet()

{

Link p = ini;

Item conteudo = ini->conteudo;

ini = ini->prox;

free(p);

return conteudo;

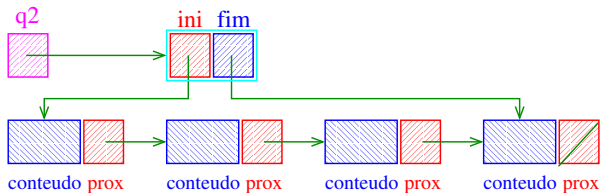
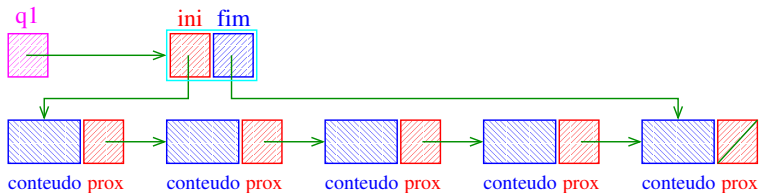
}

Implementação queue.c

```
void
queueFree()
{
    while (ini != NULL)
    {
        Link t = ini->prox;
        free(ini);
        ini = t;
    }
}
```

Fila implementada em lista encadeada

As filas serão armazenada em lista encadeada.



Fila implementada em lista encadeada

Uma fila `q` é um ponteiro para uma `struct` com campos `ini` e `fim`.

Para cada fila `q` há ponteiros `ini` e `fim`.

`q->ini->conteudo` é o primeiro elemento da fila `q`.

`q->fim->conteudo` é o último elemento da fila `q`.

A fila `q` está vazia se “`q->ini == NULL`”.

Interface item.h

```
/*  
 * item.h  
 */  
typedef int Item;
```

Interface queue.h

```
/*
 * queue.h
 * INTERFACE: funcoes para manipular filas
 * ATENCAO: Esta interface permite que
 * varias filas sejam utilizadas.
 */
typedef struct queue *Queue;

Queue queueInit(int);
int queueEmpty(Queue);
void queuePut(Queue, Item);
Item queueGet(Queue);
void queueFree(Queue);
```


distancias

A função `distancias` recebe um inteiro `n`, uma matriz `A` representando uma rede de estradas entre `n` cidades e uma cidade `c` e devolve um vetor `d` que registra a distância da cidade `c` a cada uma das outras: `d[i]` é a distância de `c` a `i`.

```
int *
distancias (int n, int **A, int c) {
    int *d; /* d[i] = distancia de c a i*/
    int j;
    Queue q;
```

distancias

```
/* aloque vetor de distancias */  
d = mallocSafe(n* sizeof(int));  
  
q = queueInit(n); /* crie uma fila */  
  
/* inicialize o vetor de distancias */  
for (j = 0; j < n; j++)  
    d[j] = n; /* distancia n = infinito */  
d[c] = 0;  
  
queuePut(q, c); /* coloque c na fila */
```

distancias

```
while (!queueEmpty(q)) {
    int i = queueGet(q);
    int di = d[i];
    for (j = 0; j < n; j++)
        if (A[i][j] == 1 && d[j] > di+1) {
            d[j] = di + 1;
            queuePut(q, j);
        }
}
queueFree(q);
return d;
}
```

Implementação queue.c

```
/*  
 * FILA: uma implementacao em lista  
 * encadeada  
 */  
typedef struct queueNode* Link;  
struct queueNode {  
    Item conteudo;  
    Link prox;  
};  
struct queue {  
    Link ini, fim;  
};  
typedef struct queue *Queue;
```

Implementação queue.c

```
static Link  
new(Item conteudo, Link prox)  
{  
    Link p = mallocSafe(sizeof *p);  
    p->conteudo = conteudo;  
    p->prox = prox;  
    return p;  
}
```

Implementação queue.c

Queue

```
queueInit(int n)
{
    Queue q = mallocSafe(sizeof *q);
    q->ini = NULL;
    return q;
}
```

```
int
queueEmpty(Queue q)
{
    return q->ini == NULL;
}
```

Implementação queue.c

```
void
queuePut(Queue q, Item item)
{
    if (q->ini == NULL)
    {
        q->ini = new(conteudo, NULL);
        q->fim = q->ini;
        return;
    }
    q->fim->prox = new(conteudo, NULL);
    q->fim = q->fim->prox;
}
```

Implementação queue.c

Item

```
queueGet(Queue q)
{
    Link p = q->ini;
    Item conteudo = q->ini->conteudo;

    q->ini = q->ini->prox;
    free(p);
    return conteudo;
}
```


Implementação queue.c

```
void
queueFree(Queue q)
{
    while (q->ini != NULL)
    {
        Link t = q->ini->prox;
        free(q->ini);
        q->ini = t;
    }
    free(q);
}
```