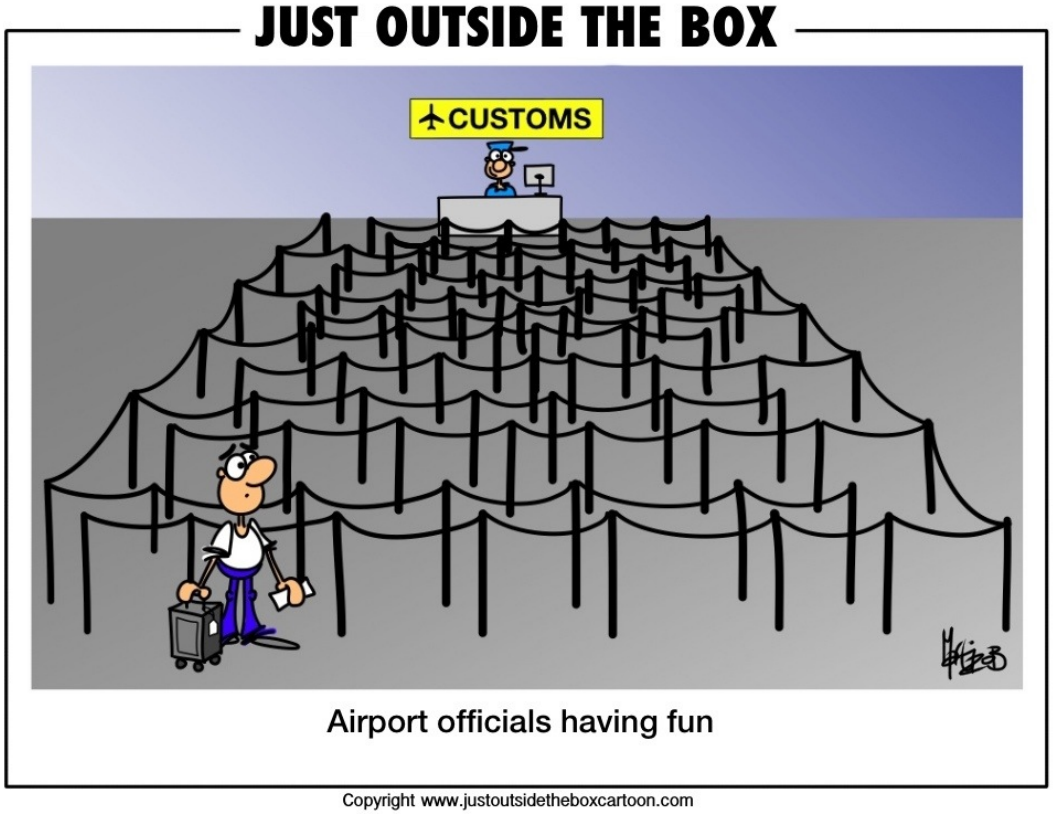
Melhores momentos

AULA 12

Filas



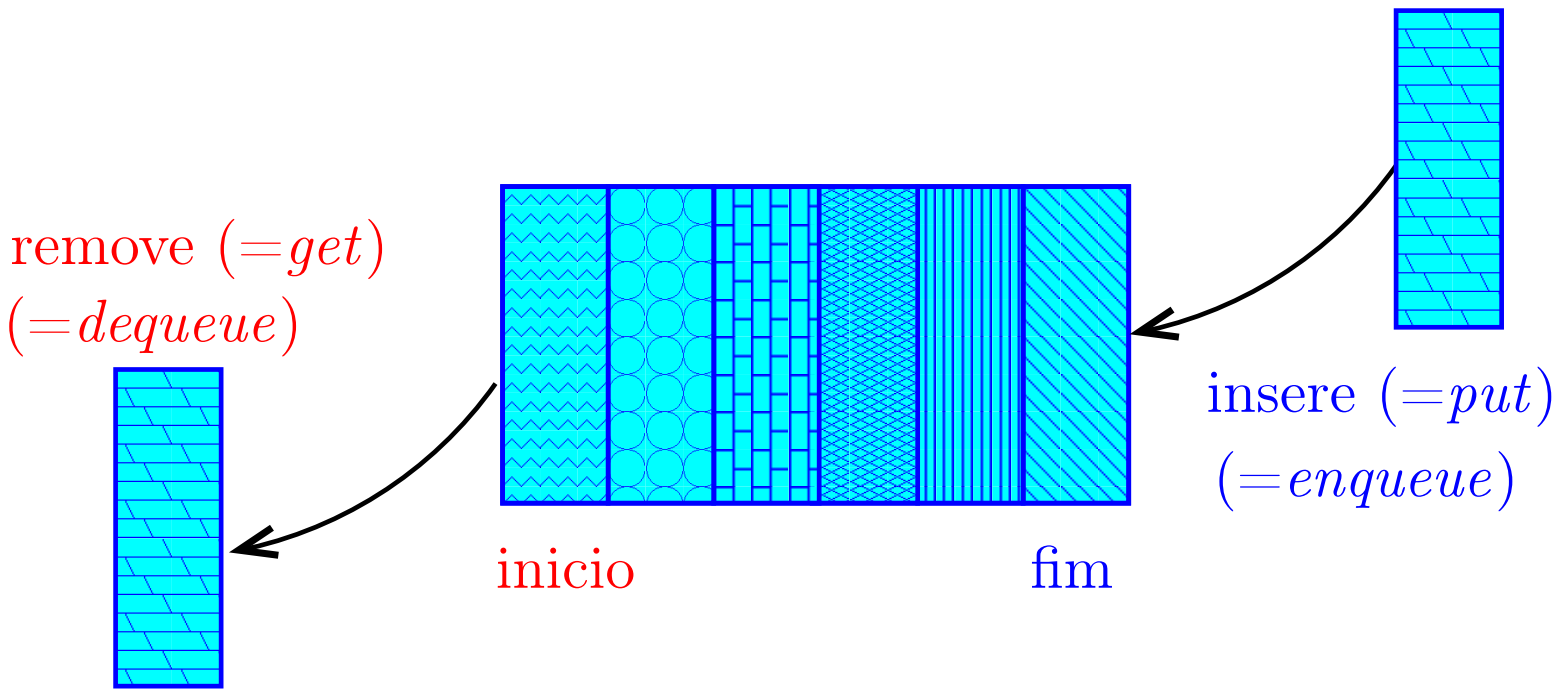
Fonte: http://justoutsidetheboxcartoon.com/

PF 5.1

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

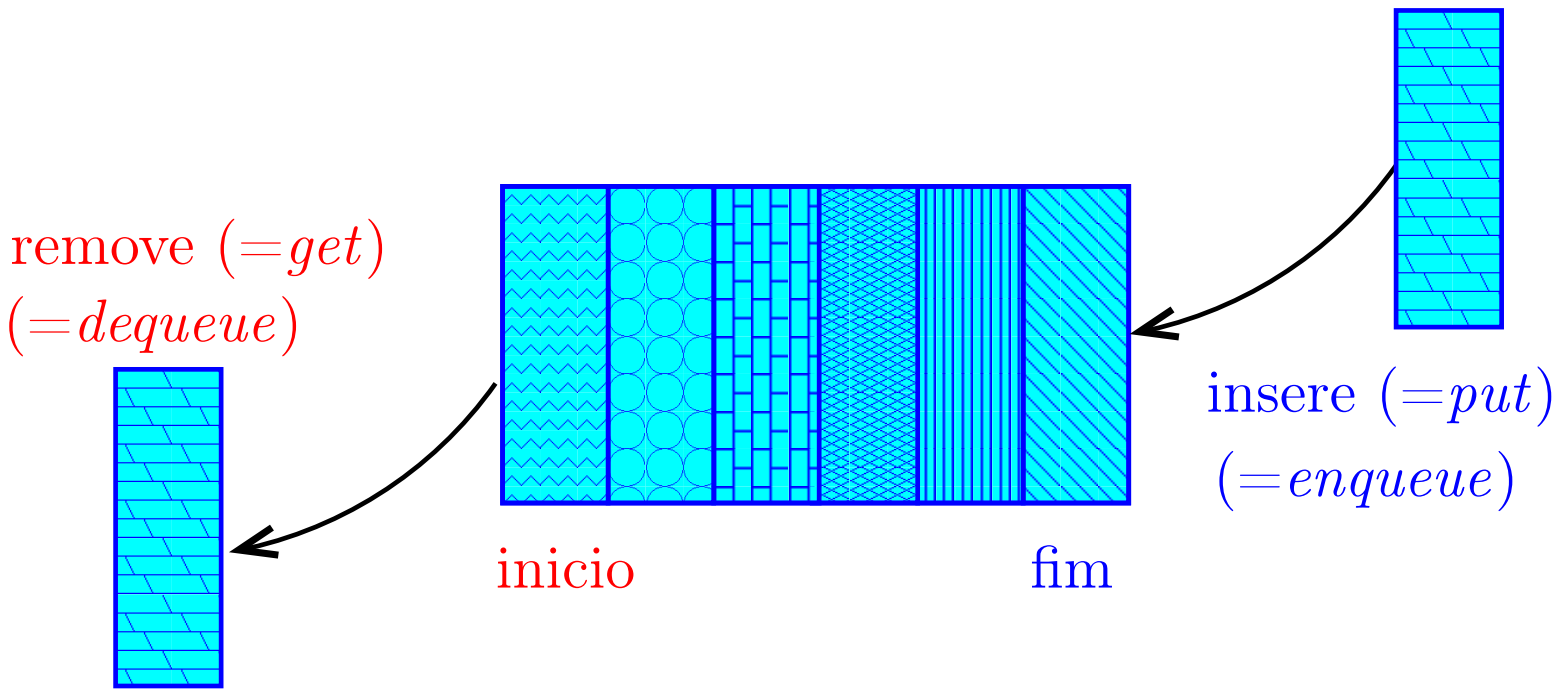
Filas

Uma **fila** (=*queue*) é uma lista dinâmica em que todas as inserções são feitas em uma extremidade chamada de **fim** e todas as remoções são feitas na outra extremidade chamada de **início**.



Filas

Assim, o primeiro objeto a ser removido de uma fila é o primeiro que foi inserido. Esta política de manipulação é conhecida pela sigla **FIFO** (=*First In First Out*)



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 dist que registra a distância da cidade c a cada uma das outras: dist[i] é a distância de c a i.

```
int *
distancias (int n, int **a, int c) {
   int *q; /* guarda a fila */
   int ini; /* q[ini] = 1o.  */
   int fim; /* q[fim-1] = ultimo */
   int *dist; /* dist[i] = distancia de c a
i*/
   int j;
```

distancias

```
/* queueInit(n):  inicialize a fila */
q = mallocSafe(n*sizeof(int));

ini = 0; fim = 0; /* fila vazia */

/* aloque vetor de distancias */
dist = mallocSafe(n*sizeof(int));

/* inicialize o vetor de distancias */
for (j = 0; j < n; j++)
   dist[j] = n; /* distancia n = infinito
*/
dist[c] = 0;

/* queuePut(c):  coloque c na fila */
q[fim++] = c;
```

distancias

```
  while (ini != fim) { /*!queueEmpty()*/
     int i = q[ini++]; /* i = queueGet() */
     int di = dist[i];
     for (j = 0; j < n; j++)
         if (a[i][j] == 1 && dist[j] > di+1){
             dist[j] = di + 1;
             q[fim++] = j; /* queuePut(j) */
         }
  }
  free(q); /* queueFree() */
  return dist;
}
```

Relações invariantes

No início de cada iteração do `while`, a fila consiste em

> *zero ou mais cidades à distância* `k` *de* `c`*,*
> *seguidas de zero ou mais cidades à distância*
> `k`*+1 de* `c`*,*

para algum `k`.

Isto permite concluir que, no início de cada iteração, para toda cidade `i`, se `dist[i] != n` então `dist[i]` é a distância de `c` a `i`.

Consumo de tempo

O consumo de tempo da função `distancias` é proporcional a $\mathtt{n}^2$

O consumo de tempo da função `distancias` é $\mathrm{O}(\mathtt{n}^2)$

AULA 13

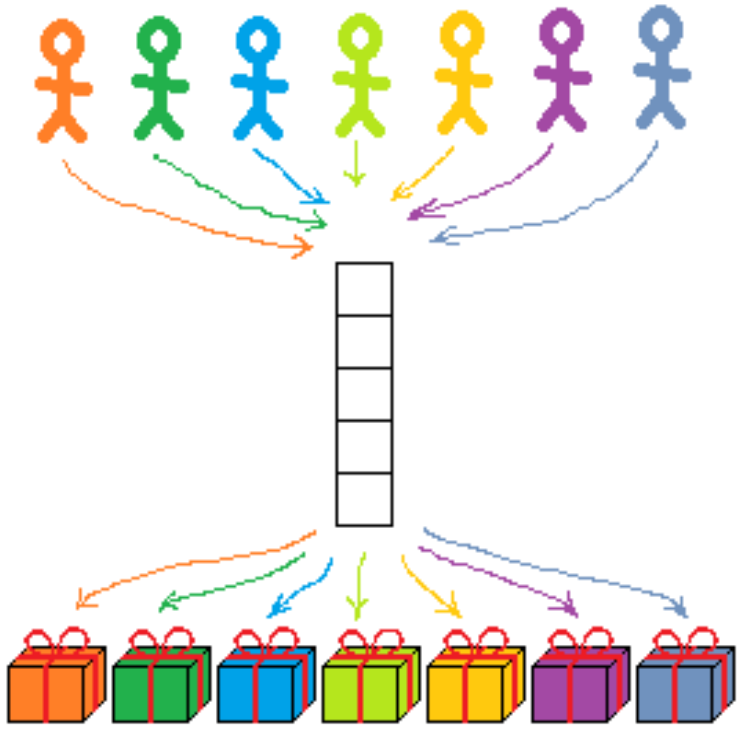
Condição de inexistência

Se `dist[i] == n` para alguma cidade `i`, então

$$S = \{v : \texttt{dist}[v] < n\}$$
$$T = \{v : \texttt{dist}[v] == n\}$$

são tais que toda estrada entre cidades em S e cidades em T tem seu início em T e fim em S.

Interface para filas



Fonte: http://yosefk.com/blog

S 4.6, 4.8

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();
```

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 dist que registra a distancia da cidade c a cada uma das outras: dist[i] é a distância de c a i.

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

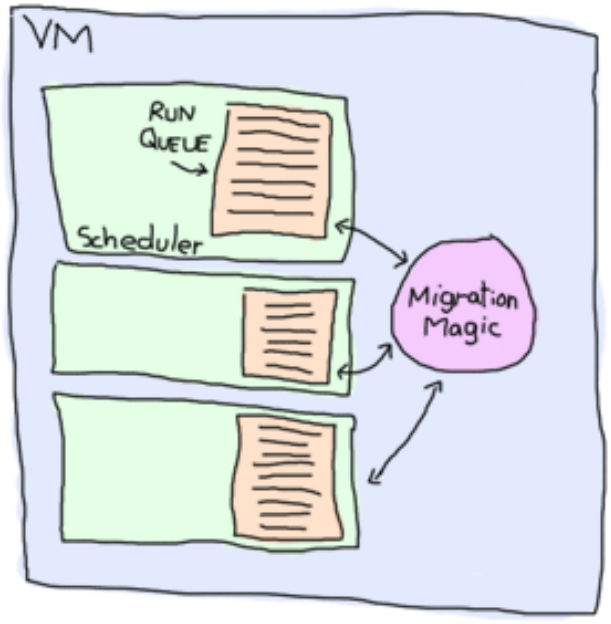
distancias

```
queueInit(n); /* inicialize a fila */

/* aloque vetor de distancias */
dist = mallocSafe(n*sizeof(int));

/* inicialize o vetor de distancias */
for (j = 0; j < n; j++)
   dist[j] = n; /* distancia n = infinito
*/
dist[c] = 0;

queuePut(c); /* coloque c na fila */
```

distancias

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

Implementações de filas

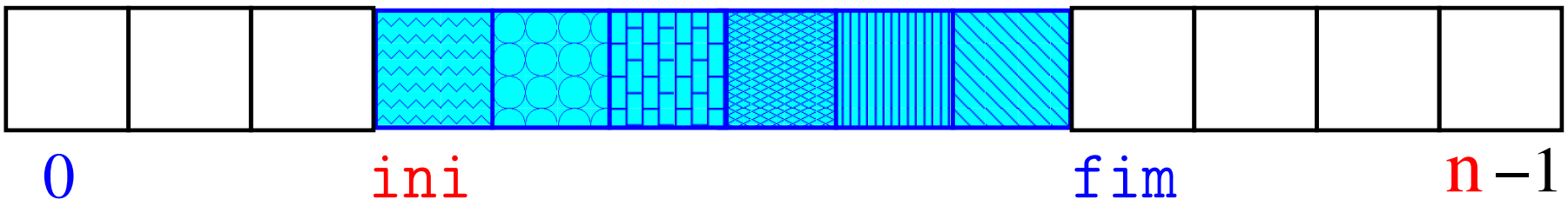


Fonte: http://learnyousomeerlang.com/

PF 5.3
http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html

Filas em vetores

A fila será armazenada em um vetor `q[0 . . n-1]`.



O índice `ini` indica o primeiro da fila.

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

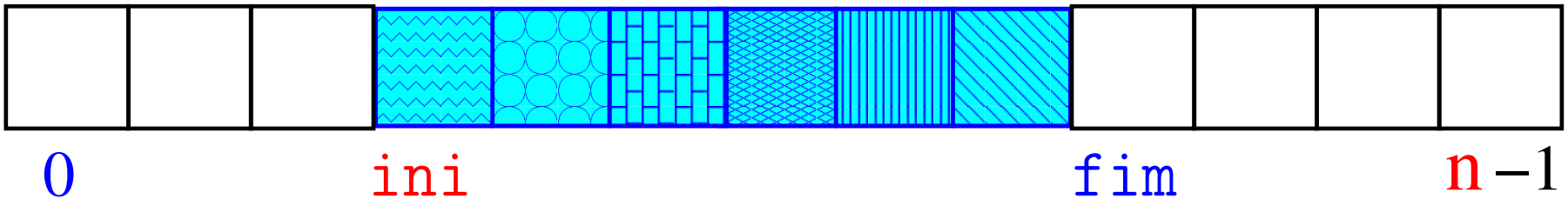
`fim` é a primeira posição vaga da fila.

A fila está vazia se "`ini == fim`"

A fila está cheia se "`fim == n`".

Filas em vetores

A fila será armazenada em um vetor `q[0 . . n-1]`.



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

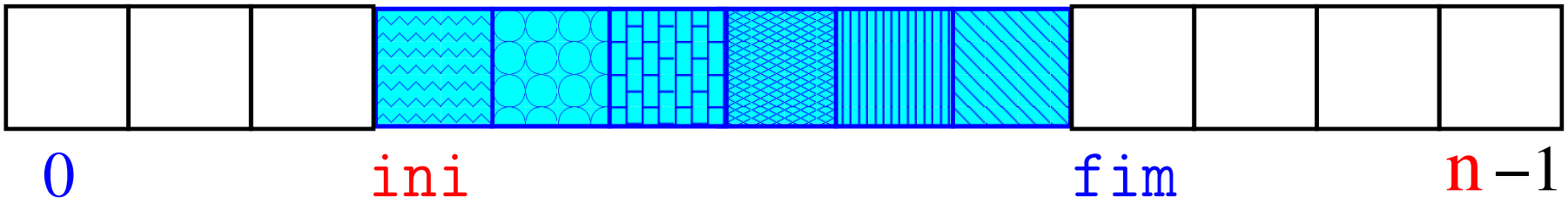
```
x = q[ini++];
```

que é equivalente a

```
x = q[ini];
ini += 1;
```

Filas em vetores

A fila será armazenada em um vetor `q[0 . . n-1]`.



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

```
q[fim++] = x;
```

que é equivalente a

```
q[fim] = x;
fim += 1;
```

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 ini;
static int fim;
```

Implementação queue.c

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

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

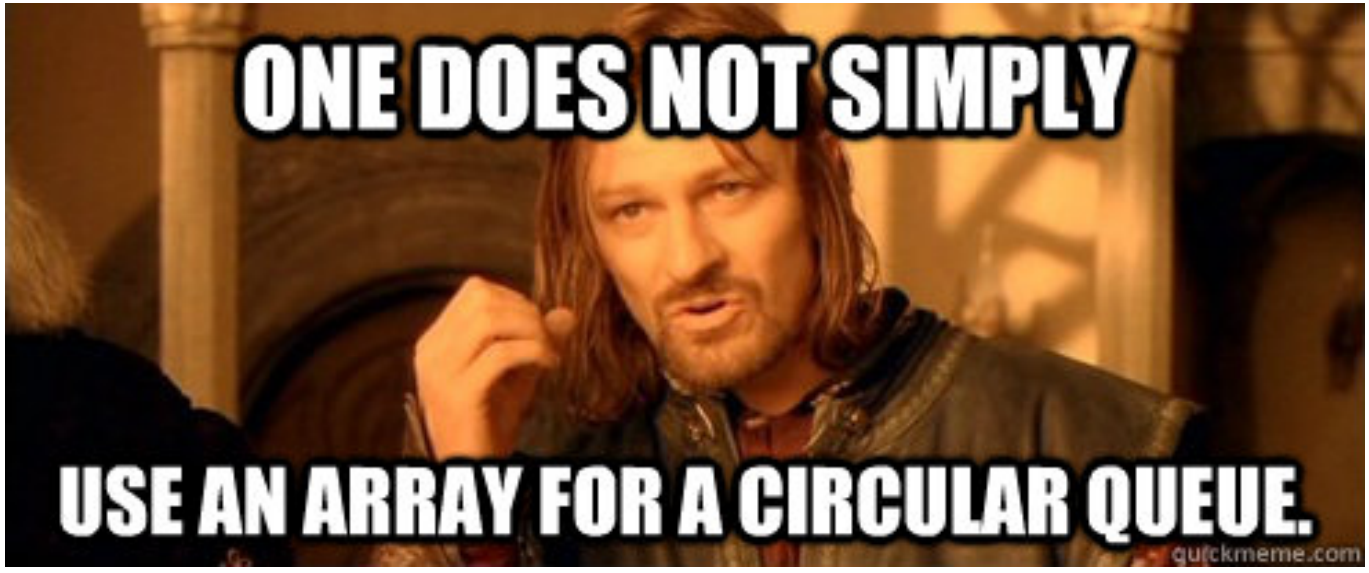
Implementação queue.c

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

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

Implementação queue.c

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

Filas em vetores circulares

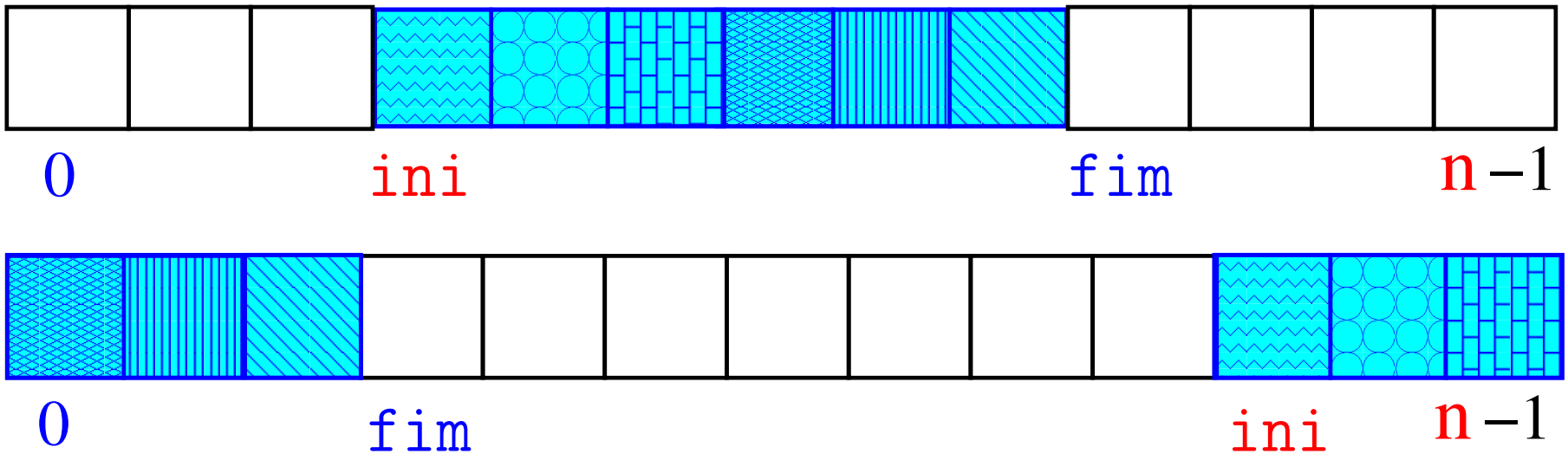


Fonte: http://www.quickmeme.com/

PF 5.1
http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html

Fila implementada circularemente em um vetor

A fila será armazenada em um vetor `q[0 . . n-1]`.



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

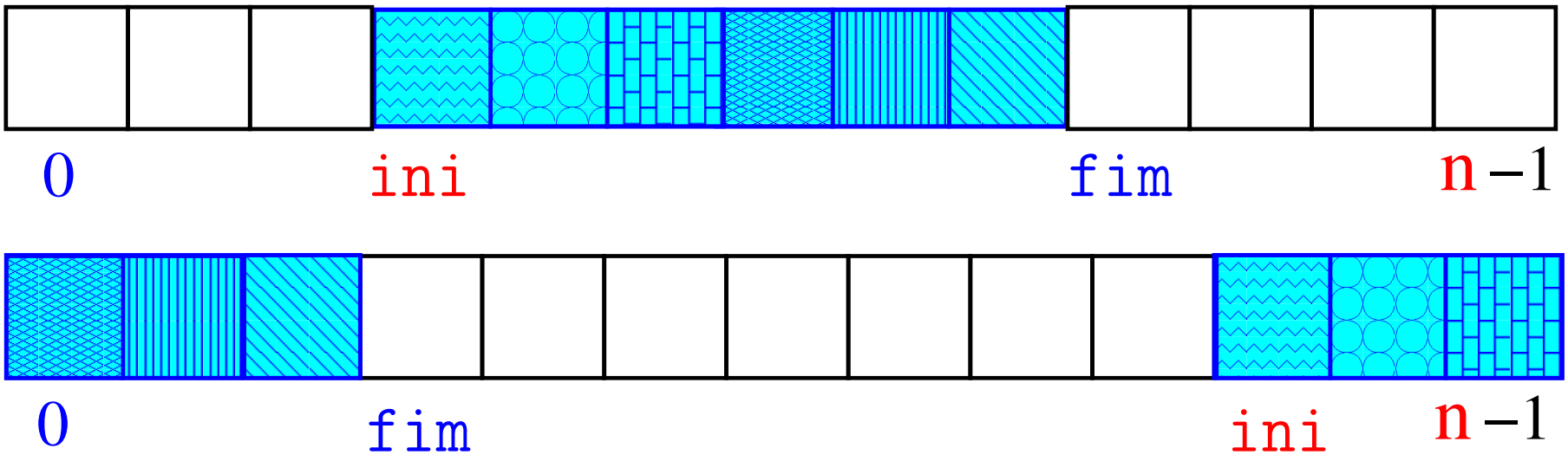
Não supomos `ini` $\leq$ `fim`

O índice `ini` indica o primeiro da fila.

`fim` é a primeira posição vaga da fila.

Fila implementada circularemente em um vetor

A fila será armazenada em um vetor `q[0 . . n-1]`.



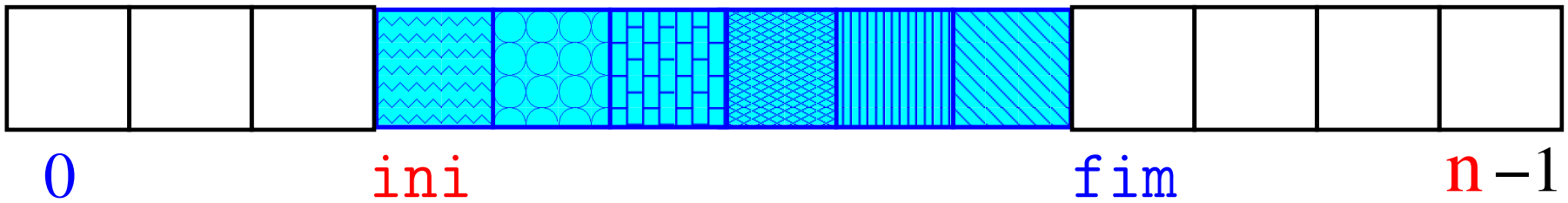
A fila está vazia se "`ini == fim`"

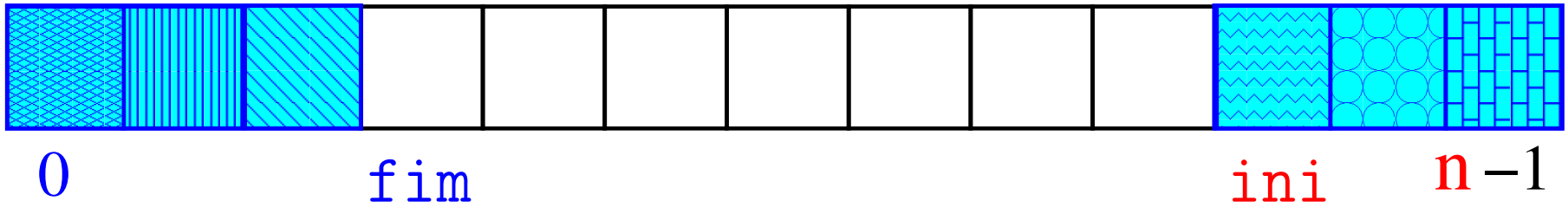
A fila está cheia se "`fim+1 == ini`" ou

"`fim+1 == n` **e** `ini == 0`"

Fila implementada circularemente em um vetor

A fila será armazenada em um vetor `q[0 . . n-1]`.



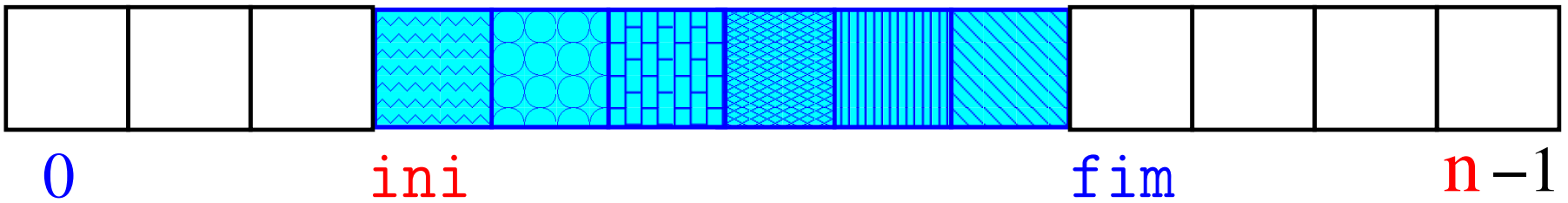


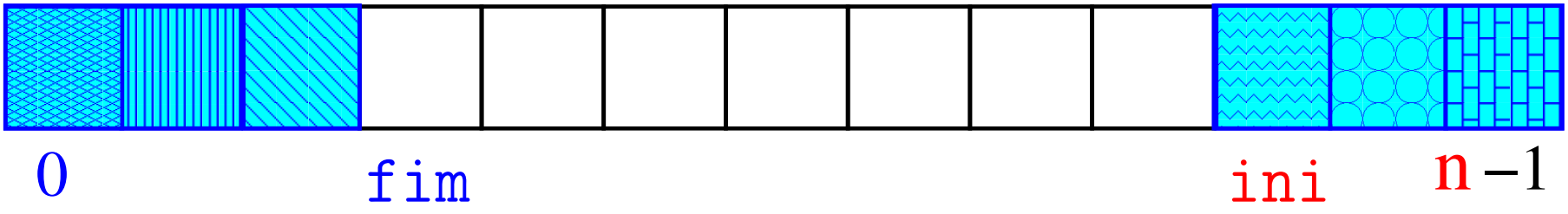
A fila está vazia se "`ini == fim`"

A fila está cheia se "`(fim+1) % n == ini`"

Fila implementada circularemente em um vetor

A fila será armazenada em um vetor `q[0 . . n-1]`.



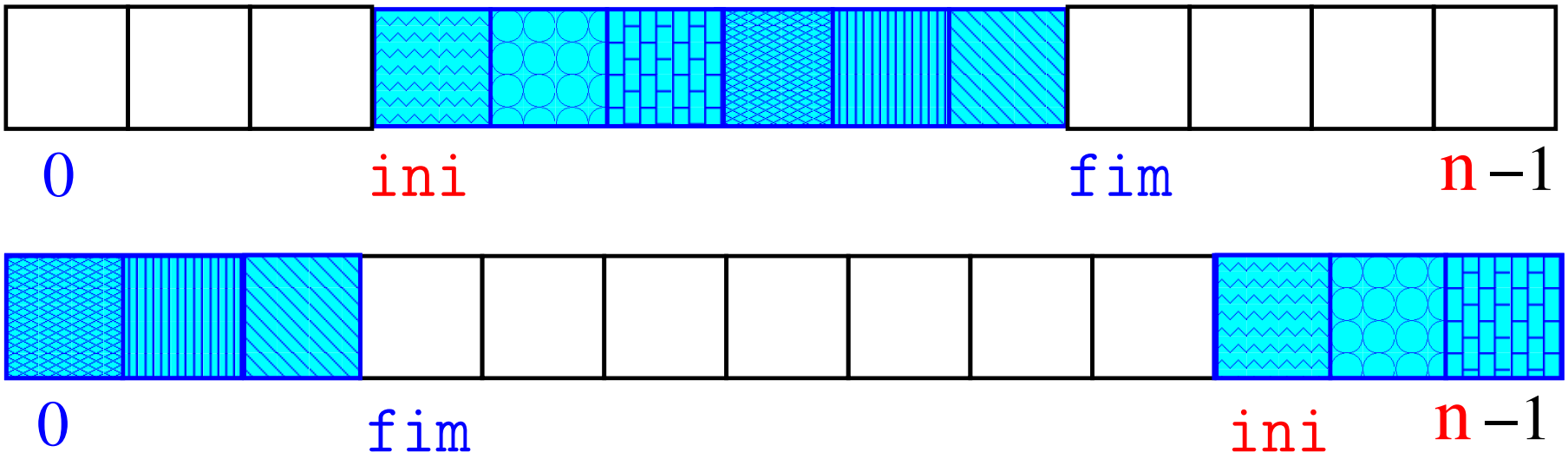


A posição `fim` **sempre está desocupada**

Isto é importante para distinguir fila vazia de cheia

Fila implementada circularemente em um vetor

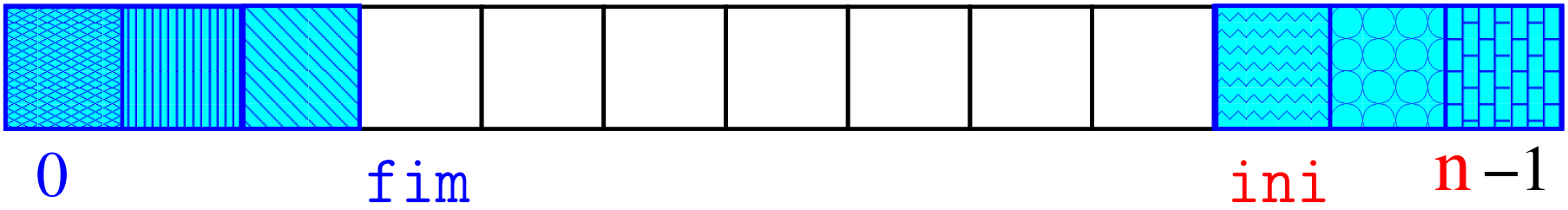
A fila será armazenada em um vetor `q[0 . . n-1]`.



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 ini;
static int fim;
```

Implementação queue.c

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

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