

Melhores momentos

AULA 14

Interface queue.h

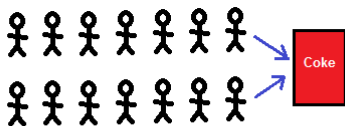
```
/*  
 * queue.h  
 * INTERFACE: funcoes para manipular uma  
 * fila  
 */  
void queueInit(int);  
int queueEmpty();  
void queuePut(Item);  
Item queueGet();  
void queueFree();
```

Implementação queue.c sem cabeça

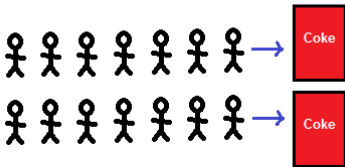
```
void
queuePut(Item item)
{
    if (ini == NULL)
    {
        ini = fim = new(item, NULL);
        return;
    }
    fim->prox = new(item, NULL);
    fim = fim->prox;
}
```

AULA 15

Filas em listas encadeadas com cabeça



Concurrent: 2 queues, 1 vending machine



Parallel: 2 queues, 2 vending machines

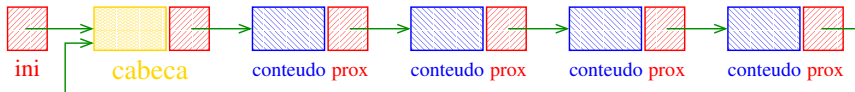
Fonte: <http://yosefk.com/>

PF 5.1

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

Fila implementada em uma lista com cabeça

A fila será armazenada em uma **lista encadeada circular** com **cabeça**.



O ponteiro **ini** aponta para a **cabeça** da lista.

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

A fila está **vazia** se "**ini**->**prox** == **ini**".

Implementação queue.c

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

Implementação queue.c

```
void  
queueInit(int n)  
{  
    ini = mallocSafe(sizeof *ini);  
    ini->prox = ini;  
}
```

```
int  
queueEmpty()  
{  
    return ini->prox == ini;  
}
```

Implementação queue.c

```
void
queuePut(Item item)
{
    Link nova = mallocSafe(sizeof *nova);

    nova->prox = ini->prox;
    ini->prox = nova;

    /* insira item na celula cabeca (!) */
    ini->conteudo = item;

    /* mude a cabeca para nova (!) */
    ini = nova;
}
```

Implementação queue.c

Item

queueGet()

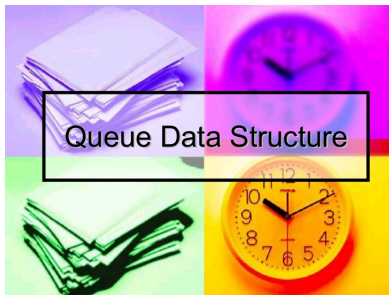
```
{  
    Link p = ini->prox;  
    Item item = p->conteudo;  
  
    ini->prox = p->prox;  
    free(p);  
    return item;  
}
```

Implementação queue.c

```
void
queueFree()
{
    Link p = ini->prox;

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

FilaS em listaS encadeadaS

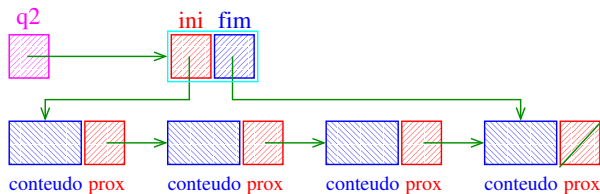
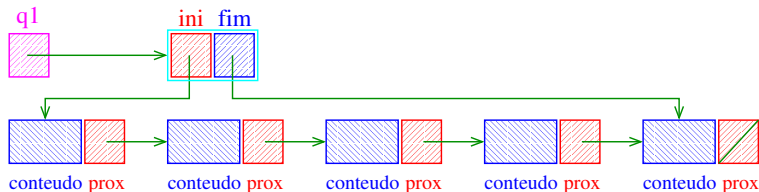


Fonte: <http://www.slideshare.net/>

S 4.8

FilaS implementadaS em listaS encadeadaS

As filas serão armazenada em listas encadeadas.



FilaS implementadaS em listaS encadeadaS

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 item, Link prox)  
{  
    Link p = mallocSafe(sizeof *p);  
    p->conteudo = item;  
    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(item, NULL);  
        q->fim = q->ini;  
        return;  
    }  
    q->fim->prox = new(item, NULL);  
    q->fim = q->fim->prox;  
}
```

Implementação queue.c

Item

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

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

Implementação queue.c

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

Compilação

```
cria o obj queue.o  
> gcc -Wall -O2 -ansi -pedantic -Wno-unused-result  
-c queue.c
```

```
cria o obj distancias.o  
> gcc -Wall -O2 -ansi -pedantic -Wno-unused-result  
\  
-c distancias.c
```

```
cria o executável distancia  
> gcc queue.o distancia.o -o distancia
```

Makefile

Hmmm. Ler o tópico **Makefile** no fórum.

```
distancia:  distancia.o queue.o
            gcc distancia.o queue.o -o distancia
```

```
distancia.o:  distancia.c
              gcc -Wall -O2 -ansi -pedantic \
                -Wno-unused-result -c distancia.c
```

```
queue.o:  queue.c item.h
           gcc -Wall -O2 -ansi -pedantic \
             -Wno-unused-result -c queue.c
```