# Algoritmos, Filas

Prof. Ricardo Reis
Universidade Federal do Ceará
Sistemas de Informação
Estruturas de Dados

20 de Junho de 2012

## 1 Definição

Uma *fila* (ou *queue* em inglês) é uma estrutura de dados linear elementar que permite armazenar e recuperar objetos sobre as seguintes restrições,

1. Objetos devem ser inseridos (enfileirados) numa extremidade (*final* da fila) e removidos (desenfileirados) da outra extremidade (*início* da fila). Ver Figura-1.
2. Só é possível ler dados dos objetos nas extremidades da fila (início e final). Os demais objetos, denominados objetos *internos*, não podem ser acessados diretamente.
3. Uma fila que não comporta mais objetos (*fila cheia*) deve reportar tentativas de enfileiramentos sem sucesso ao passo que uma fila *sem* objetos (*fila vazia*) deve reportar tentativas de desenfileiramento sem sucesso.

O número de objetos enfileirados em uma fila define o *comprimento da fila*. O máximo de objetos que uma fila suporta é denominado *capacidade da fila*. Uma fila vazia tem comprimento zero e uma fila cheia possui comprimento igual a sua capacidade.

A ordem de enfileiramento em uma fila segue a mesma ordem de desenfileiramento. Logo o primeiro objeto enfileirado será o primeiro a ser desenfileirado, o segundo enfileirado será o segundo desenfileirado e assim por diante. Por essa razão as filas são denominadas estruturas FIFO (do inglês, *First In First Out*, ou, *primeiro a entrar primeiro a sair*).

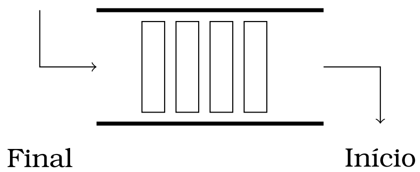



**Figura 1: Esquema de fila**

## 2 Operadores de Filas

Os operadores básicos de uma fila, como estrutura de dados, são,

- **Create**: Operador responsável por construir uma nova fila.
- **Enqueue**: Operador que enfileira um novo objeto no final de uma fila consequentemente aumentando seu comprimento em uma unidade quando a estrutura não estiver cheia.
- **Dequeue**: Operador que desenfileira o objeto no início da fila consequentemente diminuindo seu comprimento em uma unidade quando a estrutura não estiver vazia.
- **Begin**: Operador que retorna o valor no objeto no início da fila.
- **End**: Operador que retorna o valor no objeto no final da fila.
- **Empty**: Operador que testa se a fila está vazia.
- **Full**: Operador que testa se a fila está cheia.
- **Destroy**: Operador que elimina uma dada fila e seus objetos se existirem.

## 3 Filas Sequenciais

Uma fila sequencial é aquela construída utilizando-se um vetor como estrutura base. De forma similar às pilhas, simula-se numa fila que não existe o acesso aleatório no vetor base.

Tabela 1: Descritor de uma fila Sequencial

| **Atributo** | **Descrição** |
|---|---|
| $M$ | vetor base |
| $n$ | capacidade da fila |
| $i$ | índice de $M$ que corresponde ao início da fila |
| $f$ | índice de $M$ que corresponde ao final da fila |
| $t$ | comprimento da fila |

| **Operador** | **Argumentos** | **Descrição** |
|---|---|---|
| CREATE | $n$ | Cria uma fila sequencial de capacidade $n$ |
| ENQUEUE | $\mathcal{F}$, $x$ | Enfileira chave $x$ no final da fila $\mathcal{F}$ |
| DEQUEUE | $\mathcal{F}$ | Desenfileira objeto do início da fila $\mathcal{F}$ |
| BEGIN | $\mathcal{F}$ | Retorna valor no início da fila $\mathcal{F}$ |
| END | $\mathcal{F}$ | Retorna valor no final da fila $\mathcal{F}$ |
| EMPTY | $\mathcal{F}$ | Verifica se a fila $\mathcal{F}$ está vazia |
| FULL | $\mathcal{F}$ | Verifica se a pilha $\mathcal{F}$ está cheia |
| DESTROY | $\mathcal{F}$ | Elimina a fila $\mathcal{F}$ |

A Tabela-1 ilustra o descritor de uma fila sequencial. O atributo $M$ representa o vetor base e $n$ seu respectivo comprimento que nesse caso

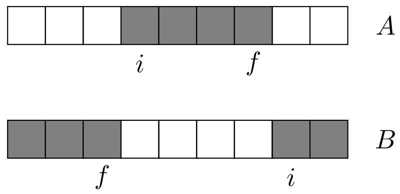



**Figura 2: Esquemas de distribuição de chaves numa fila sequencial**

retrata a capacidade da fila. Os atributos $i$ e $f$ representam respectivamente os índices de $M$ onde se localizam início e final da fila. Note que não necessariamente $f$ é maior que $i$. Na Figura-2, por exemplo, são ilustrados duas situações, $A$ e $B$, de preenchimento para o vetor $M$ de uma fila sequencial. No caso $A$, que pode ser obtido por sete chamadas seguidas a ENQUEUE e depois três a DEQUEUE, $i$ é de fato menor que $f$ existindo nas extremidades de $M$ células ainda por serem ocupadas. No caso $B$, que pode ser obtido por mais cinco enfileiramentos e quatro desenfileiramentos seguidos sobre o estado $A$, $f$ é menor que $i$, mas as células preenchidas ocorrem agora nas extremidades de $M$ e as por preencherem, no meio.

Esta forma de aproveitamento do espaço de $M$ é construída em $O(1)$ como será demonstrado nas implementações dos algoritmos dos operadores de de filas sequenciais a seguir.

O Algoritmo-1 apresenta os operadores de construção (CREATE) e destrução (DESTROY) de uma fila sequencial. A finalidade maior do construtor é alocar o vetor de base $M$ (e sua capacidade $n$) além de definir os índices $i$ e $f$ para zero acusando que eles apontam para fora da fila (lembre-se que definimos como sendo 1 o índice mínimo dos vetores). Este *apontar para fora* indica fila vazia e é equivalente a atribuição zero que se faz ao parâmetro $t$ (comprimento da fila). De fato $t$ não é necessário, mas é mantido para que não se precise recalcular o comprimento da fila (a partir de $i$ e $j$) sempre que requisitado. A nova fila vazia criada pelo operador CREATE é retornada como saída. O operador DESTROY faz o caminho inverso de CREATE. Ele recebe uma fila $\mathcal{F}$ de entrada, desaloca seu vetor base $M$ e anula os demais campos para zero desabilitando a fila.

**Algoritmo 1** Operadores de construção e destrução de fila sequencial

**Função** CREATE($n$)
  $\mathcal{F}.M \leftarrow$ **Alocar**$(n)$
  $\mathcal{F}.n \leftarrow n$
  $\mathcal{F}.i \leftarrow \mathcal{F}.f \leftarrow 0$
  $\mathcal{F}.t \leftarrow 0$
  **Retorne** $\mathcal{F}$

**Função** DESTROY(**ref** $\mathcal{F}$)
  **Desalocar** $\mathcal{F}.M$
  $\mathcal{F}.i \leftarrow \mathcal{F}.f \leftarrow \mathcal{F}.t \leftarrow 0$

O Algoritmo-2 implementa o operador de enfileiramento em fila sequencial. Ele recebe como entrada uma referência $\mathcal{F}$ para a fila alvo e a chave $x$ a ser enfileirada. Quando a fila está vazia ($\mathcal{F}.t = 0$, linha-2) então os valores de $\mathcal{F}.i$ e $\mathcal{F}.f$ (que valem zero) são mudados para $1$ (linha-3) posição esta onde é enfileirado o valor de $x$ (linha-4). Quando a fila não está vazia calcula-se a posição $p$ de enfileiramento (linha-6) que corresponde a $f + 1$ quando $f < n$ e $1$ quando $f = n$, ou seja, uma vez extrapolado o final do vetor $M$ a próxima posição a ser considerada deverá ser $1$. Este mecanismo é conhecido como *circularidade* e é implementado utilizando-se o operador $\mathrm{mod}$ (resto de divisão entre inteiros) conforme indica a expressão,

$$p \leftarrow 1 + \mathcal{F}.f \bmod \mathcal{F}.n$$

na linha-6.

Quando o $p$ calculado coincide com $\mathcal{F}.i$ significa que a fila $\mathcal{F}$ está cheia e logo $x$ não pode ser enfileirado (o operador retorna então na linha-9). Do contrário $x$ pode ser enfileirado para a posição $p$ (linha-8) e $\mathcal{F}.f$ atualizado para o valor de $p$ (linha-9). Na linha-12 o comprimento da fila é incrementado (note que isso só ocorre quando o enfileiramento se procede). O operador retorna **Verdadeiro** quando o enfileiramento obtém sucesso (linha-13) e **Falso** (linha-11) do contrário.

**Algoritmo 2** Operador de enfileiramento em fila sequencial

1: **Função** ENQUEUE(**ref** $\mathcal{F}$, $x$)
2:     **Se** $\mathcal{F}.t = 0$ **então**
3:         $\mathcal{F}.i \leftarrow \mathcal{F}.f \leftarrow 1$
4:         $\mathcal{F}.M[1] \leftarrow x$
5:     **senão**
6:         $p \leftarrow 1 + \mathcal{F}.f \bmod \mathcal{F}.n$
7:         **Se** $p \neq \mathcal{F}.i$ **então**
8:             $\mathcal{F}.M[p] \leftarrow x$
9:             $\mathcal{F}.f \leftarrow p$
10:         **senão**
11:             **Retorne Falso**
12:     $\mathcal{F}.t \leftarrow \mathcal{F}.t + 1$
13:     **Retorne Verdadeiro**

O operador de desenfileiramento em fila sequencial é implementado pelo Algoritmo-3. O operador DEQUEUE recebe a fila $\mathcal{F}$ e lhe efetua um desenfileiramento. Quando $\mathcal{F}$ não está vazia (teste da linha-2 obtém sucesso) o operador trata duas situações: a de fila contendo um elemento e a de fila contendo mais de um elemento. Se a fila contém apenas um elemento o teste da linha-3 obtém êxito e os índices $\mathcal{F}.i$ e $\mathcal{F}.f$ são *apontados* para fora da fila (linha-4). Do contrário, se o teste na linha-3 falha então o início da fila $\mathcal{F}.i$ (que é a extremidade de retiradas) precisa ser recalculado. Similar ao que ocorre no enfileiramento, o reajuste de $\mathcal{F}.i$ deve ser feito pelo mecanismo da circularidade que garante que o índice voltará a 1 ao tentar extrapolar o final de $\mathcal{F}.M$. O novo valor de $\mathcal{F}.i$ é dado por,

$$1 + \mathcal{F}.i \bmod \mathcal{F}.n$$

conforme implementado na linha-6. Quando $\mathcal{F}$ está vazia o teste na linha-3 falha e então procede-se a linha-8 que retorna **Falso** indicando que nenhum desenfileiramento foi realizado devido a fila estar vazia. Na linha-9 o comprimento da fila $\mathcal{F}.t$ é diminuído em uma unidade (note que esta linha só é executada se de fato um desenfileiramento ocorrer). O **Verdadeiro** retornado na linha-10 indica desenfileiramento com sucesso.

**Algoritmo 3** Operador de Desenfileiramento em fila sequencial

1: **Função** DEQUEUE(**ref** $\mathcal{F}$)
2:     **Se** $\mathcal{F}.t > 0$ **então**
3:         **Se** $\mathcal{F}.t = 1$ **então**
4:             $\mathcal{F}.i \leftarrow \mathcal{F}.f \leftarrow 0$
5:         **senão**
6:             $\mathcal{F}.i \leftarrow 1 + \mathcal{F}.i \bmod \mathcal{F}.n$
7:     **senão**
8:         **Retorne Falso**
9:     $\mathcal{F}.t \leftarrow \mathcal{F}.t - 1$
10:     **Retorne Verdadeiro**

No Algoritmo-4 são implementados os demais operadores de filas sequenciais cada qual recebendo a fila $\mathcal{F}$ de operação. Respectivamente BEGIN e END retornam os valores das chaves no início e final da fila $\mathcal{F}$ (valores das chaves de $\mathcal{F}.M$ nas posições $\mathcal{F}.f$ e $\mathcal{F}.i$). EMPTY e FULL testam se a fila está vazia ($\mathcal{F}.t = 0$) e cheia ($\mathcal{F}.t = \mathcal{F}.n$) respectivamente.

**Algoritmo 4** Demais operadores em filas sequenciais.

**Função** BEGIN(**ref** $\mathcal{F}$) ▷ Valor da chave no início da fila
    **Retorne** $\mathcal{F}.M[\mathcal{F}.i]$

**Função** END(**ref** $\mathcal{F}$) ▷ Valor da chave no final da fila
    **Retorne** $\mathcal{F}.M[\mathcal{F}.f]$

**Função** EMPTY(**ref** $\mathcal{F}$) ▷ Testa se fila está vazia
    **Retorne** $\mathcal{F}.t = 0$

**Função** FULL(**ref** $\mathcal{F}$) ▷ Testa se fila está cheia
    **Retorne** $\mathcal{F}.t = \mathcal{F}.n$

# 4 Filas Encadeadas

Uma fila encadeada é aquela que utiliza uma lista simplesmente encadeada como estrutura base. Em tais filas são mantidas duas referências, $i$ e $f$, sendo $i$ dedicada a apontar para o nódo inicial da lista

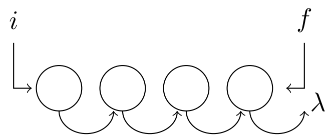



**Figura 3: Esquema de fila encadeada**

base e $f$ para o nódo final. Quando a fila está vazia as duas referências apontam para $\lambda$.

Na Figura-3 é ilustrada uma fila encadeada. O nódo-raiz da lista de base é apontado por $i$ ao passo que o último nódo é apontado por $f$. Esta ordem de escolha dos rótulos das extremidades não é aleatória. De fato ela é responsável por manter em $O(1)$ as operações de enfileiramento e desenfileiramento em filas encadeadas. Para entender isso basta imaginar como ficam operações de inserção e remoção de nódos em cada extremidade. Na extremidade inicial, inserção e remoção podem ser feitas em $O(1)$. Entretanto na extremidade final, apesar de a inserção poder ser feita em $O(1)$ (lembre-se que a referência ao nódo final é conhecida), a remoção deve ocorrer em $O(n)$ haja vista o último nódo não ter acesso ao penúltimo (para atualizar a referência).

A Tabela-2 ilustra o descritor de uma fila encadeada. Os atributos neste caso são reduzidos, em relação às filas sequenciais, às referências $i$ e $f$ respectivamente para início e final de fila, além do comprimento $t$ (quantidade de nódos). Os operadores são os mesmos das filas sequenciais distinguindo-se apenas pelo construtor, que neste caso não possui argumentos de entrada, e pela ausência do operador FULL haja vista estruturas encadeadas não possuírem restrições de capacidade.

Tabela 2: Descritor de uma fila Encadeada

| **Atributo** | **Descrição** |
|---|---|
| $i$ | Referência para o nódo inicial da fila |
| $f$ | Referência para o nódo final da fila |
| $t$ | comprimento da fila |

| **Operador** | **Argumentos** | **Descrição** |
|---|---|---|
| CREATE | - | Cria uma fila encadeada |
| ENQUEUE | $\mathcal{F}$, $x$ | Enfileira chave $x$ no início da fila $\mathcal{F}$ |
| DEQUEUE | $\mathcal{F}$ | Desenfileira objeto do início da fila $\mathcal{F}$ |
| BEGIN | $\mathcal{F}$ | Retorna valor no início da fila $\mathcal{F}$ |
| END | $\mathcal{F}$ | Retorna valor no final da fila $\mathcal{F}$ |
| EMPTY | $\mathcal{F}$ | Verifica se a fila $\mathcal{F}$ está vazia |
| DESTROY | $\mathcal{F}$ | Elimina a fila $\mathcal{F}$ |

A Tabela-3 ilustra o descritor de um nódo de uma fila encadeada. É exatamente a mesma composição de um nódo de lista encadeada, ou seja, possui um campo para a chave ($chave$) e outro para referenciamento do nódo seguinte ($prox$).

Tabela 3: Descrição de um nódo de uma fila encadeada

| **Campo** | **Descrição** |
|---|---|
| *chave* | Valor da chave no nódo |
| *prox* | Referência ao nódo seguinte (vale $\lambda$ quando é o último nódo) |

O Algoritmo-5 implementa os operadores de construção e destrução de fila encadeada. O construtor CREATE configura os campos $i$ e $f$ de uma nova fila encadeada $\mathcal{F}$ para $\lambda$ além de anular o campo $t$. A fila criada é retornada e representa uma fila vazia. O operador DESTROY desaloca a fila encadeada recebida via argumento $\mathcal{F}$. O processo é semelhante a desalocação de listas encadeadas: um laço principal (linha-6) faz avançar a raiz da estrutura (linha-8) eliminando gradativamente os nódos que vão ficando para trás (linha-9). Note que esse laço conduz $\mathcal{F}.i$ a $\lambda$ sendo ainda necessário, para anulação completa da fila, fazer $\mathcal{F}.f$ apontar para $\lambda$ (linha-10) e $\mathcal{F}.t$ receber zero (linha-11).

**Algoritmo 5** Operadores de construção e destrução em fila encadeada

1: **Função** CREATE
2: $\quad \mathcal{F}.i \leftarrow \mathcal{F}.f \leftarrow \lambda$
3: $\quad \mathcal{F}.t \leftarrow 0$
4: $\quad$ **Retorne** $\mathcal{F}$

5: **Função** DESTROY(**ref** $\mathcal{F}$)
6: $\quad$ **Enquanto** $\mathcal{F}.i \neq \lambda$ **faça**
7: $\quad\quad \mathcal{N} \leftarrow \mathcal{F}.i$
8: $\quad\quad \mathcal{F}.i \leftarrow \mathcal{F}.i.prox$
9: $\quad\quad$ **Desalocar** $\mathcal{N}$
10: $\quad \mathcal{F}.f \leftarrow \lambda$
11: $\quad \mathcal{F}.t \leftarrow 0$

No Algoritmo-6 é implementado o operador ENQUEUE de enfileiramento de uma chave $x$, passada como argumento, em uma fila encadeada $\mathcal{F}$. Neste operador é criado um nódo $\mathcal{N}$ cujos campos *chave* e *prox* são respectivamente configurados com o valor de $x$ e $\lambda$. O novo nódo então é conectado ao final da fila (linha-4) e $\mathcal{F}.f$ atualizado para o novo último nódo (linha-5). O comprimento da fila é aumentado em uma unidade na linha-6. Note que como a estrutura é encadeada então não existem testes de verificação no algoritmo.

No Algoritmo-7 é implementado o operador DEQUEUE de desenfileiramento em uma fila encadeada $\mathcal{F}$ passada como argumento. Este operador testa se $\mathcal{F}$ não está vazia (linha-2) e em caso de êxito elimina o primeiro nódo da lista base. A eliminação possui as três etapas já conhecidas: (i) reserva do primeiro nódo através de uma variável auxiliar (linha-3); (ii) deslocamento da raiz $\mathcal{F}.i$ para a posição adiante (linha-4); (iii) eliminação do nódo reservado (linha-5). Na linha-6 o comprimento da fila é corrigido. A linha-7 e a linha-9 são dedicadas a retorno de

**Algoritmo 6** Operador de enfileiramento em fila encadeada

**Função** ENQUEUE(**ref** $\mathcal{F}$, $x$)
  $\mathcal{N}.chave \leftarrow x$
  $\mathcal{N}.prox \leftarrow \lambda$
  $\mathcal{F}.f.prox \leftarrow \mathcal{N}$
  $\mathcal{F}.f \leftarrow \mathcal{N}$
  $\mathcal{F}.t \leftarrow \mathcal{F}.t + 1$

função: **Verdadeiro** indica que houve desenfileiramento e **Falso** que a fila estava vazia e nenhum desenfileiramento pôde ser efetuado.

**Algoritmo 7** Operador de desenfileiramento em fila encadeada

**Função** DEQUEUE(**ref** $\mathcal{F}$)
  **Se** $\mathcal{F}.t > 0$ **então**
    $\mathcal{N} \leftarrow \mathcal{F}.i$
    $\mathcal{F}.i \leftarrow \mathcal{F}.i.prox$
    **Desalocar** $\mathcal{N}$
    $\mathcal{F}.t \leftarrow \mathcal{F}.t - 1$
    **Retorne Verdadeiro**
  **senão**
    **Retorne Falso**

Os demais operadores de filas encadeadas são implementados no Algoritmo-8. BEGIN e END retornam respectivamente as chaves do primeiro e último nódos da fila $\mathcal{F}$ (argumento de entrada) lendo o campo $chave$ dos nódos apontados pelos campos $i$ e $f$. EMPTY testa se a a fila de entrada $\mathcal{F}$ está vazia apenas verificando se o campo $t$ é nulo.

**Algoritmo 8** Demais operadores em filas encadeadas

**Função** BEGIN($\mathcal{F}$)
  **Retorne** $\mathcal{F}.i.chave$

**Função** END($\mathcal{F}$)
  **Retorne** $\mathcal{F}.f.chave$

**Função** EMPTY($\mathcal{F}$)
  **Retorne** $\mathcal{F}.t = 0$

# 5 Aplicações de Filas

## 5.1 Bucketsort

Um algoritmo *Bucketsort* é uma categoria de algoritmos de ordenação que se baseia na divisão das chaves de um vetor $U$, a ser ordenado, em *Buckets* (baldes). Consistem em etapas, chamadas *distribuições*, onde as chaves de $U$ são divididas entre os buckets, seguindo algum critério, e em seguida recolhidas dos buckets e levadas de volta para $U$ cujo novo leiaute se mostra *mais ordenado* que o anterior. Após uma dada quantidade de distribuições o vetor $U$ se torna ordenado.

*Radixsort*, ou *ordenação por raiz*, é um algoritmo da categoria bucketsort de complexidade linear, ou seja, $O(n)$, comumente utilizado para ordenar números inteiros podendo ser expandido para outras formas de dados. Na ordenação por raiz de números inteiros, em base decimal, cada distribuição é efetuada de acordo com uma família de dígitos das chaves, ou seja, a primeira distribuição ocorre de acordo com a unidade das chaves, a segunda de acordo com as dezenas, a terceira de acordo com as centenas e assim sucessivamente. A quantidade de buckets necessária deve ser dez (referente a cada dígito na base decimal) e o total de distribuições deve ser igual a quantidade de dígitos do maior inteiro (em módulo) de $U$.

A ordenação por raiz de um vetor $U$ de $n$ números inteiros em base decimal utiliza dez filas que funcionam como buckets e são rotuladas de $0$ a $9$ que anunciam seus dígitos representativos na base decimal. Em cada distribuição as chaves de $U$ são levadas aos buckets cujos dígitos atuais (unidade, dezena, centena e etc) eles rotulam. Depois que as chaves são todas enfileiradas nos buckets são desenfileiradas dos buckets, e levadas de volta a $U$. Os desenfileiramentos ocorrem em ordem crescente de rótulos dos buckets e ocorrem, em cada bucket, em sequência até esvaziamento. A ordem de desenfileiramento é também a ordem de redisposição em $U$. O número de distribuições efetuadas é igual ao número máximo $d$ de dígitos que ocorre entre as chaves de $U$.

A listagem a seguir ilustra as distribuições executadas na ordenação por raiz do vetor,[9713, 3678, 8800, 8940, 799, 1178, 1667, 4324, 8958, 7301]. Note que como a quantidade máxima de dígitos entre as chaves é quatro então se procedem quatro distribuições. Para cada distribuição é mostrado o vetor antes dos enfileiramentos, os leiautes dos buckets após os enfileiramentos e o vetor após os desenfileiramentos. Note que como cada bucket é uma fila (definimos, neste esquema, enfileiramentos à esquerda e desenfileiramentos à direita) então a ordem de chaves nos buckets é inversa a de aparição no vetor antes dos enfileiramentos e analogicamente a ordem das chaves no vetor pós-desenfileiramentos é inversa à ordem em que aparecem nos buckets.

```
 Distribuicao  1 :
 Antes:  [9713, 3678, 8800, 8940, 799, 1178, 1667, 4324, 8958, 7301]
 Buckets:
  - Bucket 0: | 8940 8800 |
  - Bucket 1: | 7301 |
  - Bucket 2: | |
  - Bucket 3: | 9713 |
  - Bucket 4: | 4324 |
  - Bucket 5: | |
  - Bucket 6: | |
  - Bucket 7: | 1667 |
  - Bucket 8: | 8958 1178 3678 |
  - Bucket 9: | 799 |
 Depois:  [8800, 8940, 7301, 9713, 4324, 1667, 3678, 1178, 8958, 799]

Distribuicao  2 :
 Antes:  [8800, 8940, 7301, 9713, 4324, 1667, 3678, 1178, 8958, 799]
 Buckets:
  - Bucket 0: | 7301 8800 |
  - Bucket 1: | 9713 |
```

```
  - Bucket 2: | 4324 |
  - Bucket 3: | |
  - Bucket 4: | 8940 |
  - Bucket 5: | 8958 |
  - Bucket 6: | 1667 |
  - Bucket 7: | 1178 3678 |
  - Bucket 8: | |
  - Bucket 9: | 799 |
 Depois:  [8800, 7301, 9713, 4324, 8940, 8958, 1667, 3678, 1178, 799]

Distribuicao  3 :
 Antes:  [8800, 7301, 9713, 4324, 8940, 8958, 1667, 3678, 1178, 799]
 Buckets:
  - Bucket 0: | |
  - Bucket 1: | 1178 |
  - Bucket 2: | |
  - Bucket 3: | 4324 7301 |
  - Bucket 4: | |
  - Bucket 5: | |
  - Bucket 6: | 3678 1667 |
  - Bucket 7: | 799 9713 |
  - Bucket 8: | 8800 |
  - Bucket 9: | 8958 8940 |
 Depois:  [1178, 7301, 4324, 1667, 3678, 9713, 799, 8800, 8940, 8958]

Distribuicao  4 :
 Antes:  [1178, 7301, 4324, 1667, 3678, 9713, 799, 8800, 8940, 8958]
 Buckets:
  - Bucket 0: | 799 |
  - Bucket 1: | 1667 1178 |
  - Bucket 2: | |
  - Bucket 3: | 3678 |
  - Bucket 4: | 4324 |
  - Bucket 5: | |
  - Bucket 6: | |
  - Bucket 7: | 7301 |
  - Bucket 8: | 8958 8940 8800 |
  - Bucket 9: | 9713 |
 Depois:  [799, 1178, 1667, 3678, 4324, 7301, 8800, 8940, 8958, 9713]
```

O Algoritmo-9 mostra a implementação da ordenação por raiz. A função RADIXSORT ordena o vetor $L$ (primeiro argumento) de comprimento $n$ (segundo argumento) e cujas chaves possuem no máximo $d$ dígitos (terceiro argumento). Na primeira parte (linha-2 a linha-3) é construído o vetor $buckets$ constituído de 10 filas que funcionarão como os buckets (os buckets possuem índices entre 1 e 10, mas se referem respectivamente aos dígitos de 0 a 9). O laço da linha-5 refere-se ao controle das distribuições e por essa razão efetua $d$ iterações. O laço entre a linha-6 e a linha-8 executa os enfileiramentos, em uma dada distribuição, das chaves em $L$ para seus devidos buckets. Como cada distribuição ocorre de acordo com uma família de dígitos (primeiro unidade, depois dezena, depois centena e assim sucessivamente) então em cada distribuição a coleta de dígitos ocorre de forma especializada através da expressão,

$$dig \leftarrow \lfloor L[j]/m \rfloor \bmod 10$$

(linha-7) onde $dig$ denota o dígito procurado. O valor $m$, nesta expressão, denota o valor pelo qual as chaves devem ser divididas de forma a se eliminar dígitos já processados ao passo que $\bmod 10$ age no intuito de extrair o dígito procurado (o resto de uma divisão por 10 de um número inteiro positivo representa seu último dígito). O valor de $m$ inicia em 1 (linha-4) e, como a base é decimal, é multiplicado por 10 ao final de cada distribuição (linha-15) permitindo um descarte gradualmente crescente de dígitos. Como os valores de dígitos calculados ficam na faixa $0..9$ então o mapeamento do bucket correspondente é feito pela expressão $dig + 1$. Os enfileiramentos propriamente ditos ocorrem na linha-8. Entre a linha-10 e linha-14 está implementada a última parte das distribuições, ou seja, os desenfileiramentos sequenciais dos buckets acoplados às sobreposições das chaves de $L$. O laço da linha-10 é responsável em varrer todos os buckets ao passo que o laço da linha-11 tem por responsabilidade desenfileirar sequencialmente cada um destes até esvaziamento. Cada chave desenfileirada dos buckets é copiada para $L$ na posição $k$ (inicialmente $k$ vale 1 conforme linha-9) constantemente atualizada na linha-14. Note que sem $k$ as chaves não poderiam ser devolvidas apropriadamente para $L$.

**Algoritmo 9** Ordenação por Raiz (RadixSort)

1: **Função** RADIXSORT(**ref** $L$, $n$, $d$)
2:     **Para** $i \leftarrow 1..10$ **faça**
3:         $buckets[i] \leftarrow$ CREATE() ▷ Cria 10 filas vazias
4:     $m \leftarrow 1$
5:     **Para** $i \leftarrow 1..d$ **faça**
6:         **Para** $j \leftarrow 1..n$ **faça**
7:             $dig \leftarrow \lfloor L[j]/m \rfloor \bmod 10$
8:             ENQUEUE($buckets[dig + 1]$, $L[j]$)
9:         $k \leftarrow 1$
10:         **Para** $i \leftarrow 1..10$ **faça**
11:             **Enquanto não** EMPTY($buckets[i]$) **faça**
12:                 $L[k] \leftarrow$ BEGIN($buckets[i]$)
13:                 DEQUEUE($buckets[i]$)
14:                 $k \leftarrow k + 1$
15:         $m \leftarrow m \times 10$

## 5.2 Preenchimento em Imagens

Uma imagem $I$ é uma matriz $m \times n$ de células denominadas *píxels*, que armazenam, cada uma, um valor de cor normalmente representado por um ou mais números inteiros. Diz-se que $I$ possui *resolução* $m \times n$ com $m$ linhas e $n$ colunas. A posição de um pixel é dada pelo par formado por sua linha e sua coluna. Valores de linhas estão no intervalo $1..m$ e de coluna no intervalo $1..n$. O pixel mais superior e mais à esquerda de uma imagem possui posição $(1, 1)$ e o pixel mais inferior e mais à direita possui posição $(m, n)$.

Consideremos o problema de preenchimento de uma região fechada em uma imagem $I$. Para simplificação consideremos uma resolução $10 \times 10$ e píxels podendo assumir três cores apenas: **BRANCO**, **CINZA** e **PRETO**. Inicialmente $I$ contém uma região branca limitada por um contorno preto fechado como ilustrado na Figura-4. Nosso objetivo é preencher em **PRETO** a região limitada pelo contorno. A ideia é partir de um pixel interno ao contorno e efetuar um preenchimento que pinta

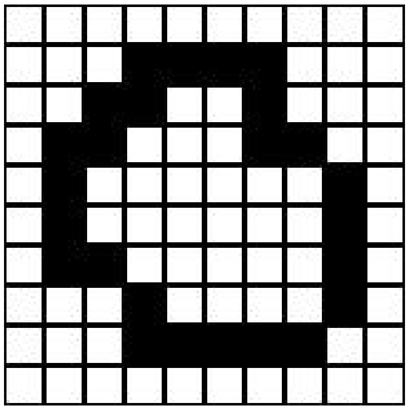


**Figura 4: Imagem com região limitada por uma linha fechada.**

gradativamente as vizinhanças e se expande até que o contorno seja encontrado (veja Figura-5).

As etapas de um algoritmo de preenchimento são descritas a seguir,

1. Tomar a posição $col$, $lin$ do ponto $Q$ a partir do qual se desenvolverá o preenchimento.

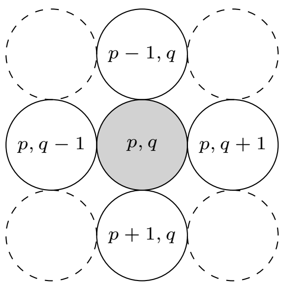
2. Tomar a cor de $Q$ (em nosso exemplo é necessariamente **BRANCO**) como cor de referência, ou seja, todos os píxels de mesma cor nas vizinhanças deverão ser pintados de **PRETO**.

3. Criar uma *fila de posições*, $\mathcal{F}$.

4. Colorir a posição $\{lin, col\}$ em **CINZA** e depois enfileirá-la em $\mathcal{F}$.

5. Promover o desenfileiramento sequencial de $\mathcal{F}$ até seu esvaziamento e para cada posição $\{p, q\}$ desenfileirada deve-se,

   (a) Colorir em **CINZA** e depois enfileirar em $\mathcal{F}$ todas as posições *vizinhas cardeais* à $\{p, q\}$ cujas cores sejam a mesma da cor de referência (note que isso descarta os píxels que formam o contorno). As posições vizinhas cardeais correspondem àquelas diretamente ao norte ($\{p-1, q\}$), ao sul ($\{p+1, q\}$), à oeste ($\{p, q-1\}$) e à leste ($\{p, q+1\}$) conforme esquema ilustrativo,

   (b) Colorir a posição $\{p, q\}$ em **PRETO**.

Note que, no algoritmo anterior, antes mesmo de assumir **PRETO**, os píxels, inicialmente em **BRANCO**, são pintados de **CINZA**. Isso ocorre no dado instante em que suas posições devem ser enfileiradas em $\mathcal{F}$. Somente quando todos os píxels em posições vizinhas cardeais são *resolvidos* (enfileirados ou descartados por pertencerem ao contorno) é que então são pintados de preto. Isso cria um efeito como o mostrado pela Figura-5 o qual ilustra uma *frente cinza* que avança em direção ao contorno e cujos píxels possuem suas posições enfileiradas à espera de serem resolvidas (pintadas de **PRETO**).

O Algoritmo-10 implementa o algoritmo descrito anteriormente. Como argumentos são repassadas uma imagem $I$ e a posição $\{lin, col\}$ a partir da qual se desenvolverá o preenchimento. Na linha-2 é criada a fila de posições, $\mathcal{F}$. Na linha-3 é determinada a cor de referência, $cor$. Na linha-4 é enfileirada a posição $\{lin, col\}$. O laço da linha-5 é o responsável pelo esvaziamento de $\mathcal{F}$. Na linha-6 e na linha-7 ocorre um desenfileiramento que é armazenado na posição $\{p, q\}$. Na linha-8 é criado o vetor $V$ das quatro posições vizinhas cardeais a $\{p, q\}$ que são candidatas a serem enfileiradas. O laço da linha-9 varre $V$ e tenta fazer os enfileiramentos. Na linha-10 as variáveis $i$ e $j$ são utilizadas para recuperar cada uma das posições vizinhas cardeais de $V$. O teste da linha-11 testa se de fato a cor da posição vizinha cardeal vigente possui a cor de referência, $cor$ (quando este teste falha é porque um pixel do contorno foi encontrado). As posições que passam no teste anterior são coloridas em **CINZA** (linha-12) e enfileiradas em $\mathcal{F}$ (linha-13). As posições cujas vizinhanças já estão resolvidas são pintadas de **PRETO** conforme linha-14.

**Algoritmo 10** [reenchimento de região interna a um contorno em uma figura dada.

1: **Função** FLOODFILL(**ref** $I$, $lin$, $col$)
2: $\quad \mathcal{F} \leftarrow$ CREATE()
3: $\quad cor \leftarrow I[lin, col]$
4: $\quad$ ENQUEUE($\mathcal{F}$, $\{lin, col\}$)
5: $\quad$ **Enquanto não** EMPTY($\mathcal{F}$) **faça**
6: $\qquad p, q \leftarrow$ BEGIN($\mathcal{F}$)
7: $\qquad$ DEQUEUE($\mathcal{F}$)
8: $\qquad V \leftarrow [(p-1, q), (p+1, q), (p, q-1), (p, q+1)]$
9: $\qquad$ **Para** $k \leftarrow 1..4$ **faça**
10: $\qquad\quad i, j \leftarrow V[k]$
11: $\qquad\quad$ **Se** $I[i, j] = cor$ **então**
12: $\qquad\qquad I[i, j] \leftarrow$ **CINZA**
13: $\qquad\qquad$ ENQUEUE($\mathcal{F}$, $V[k]$)
14: $\qquad\quad I[p, q] \leftarrow$ **PRETO**

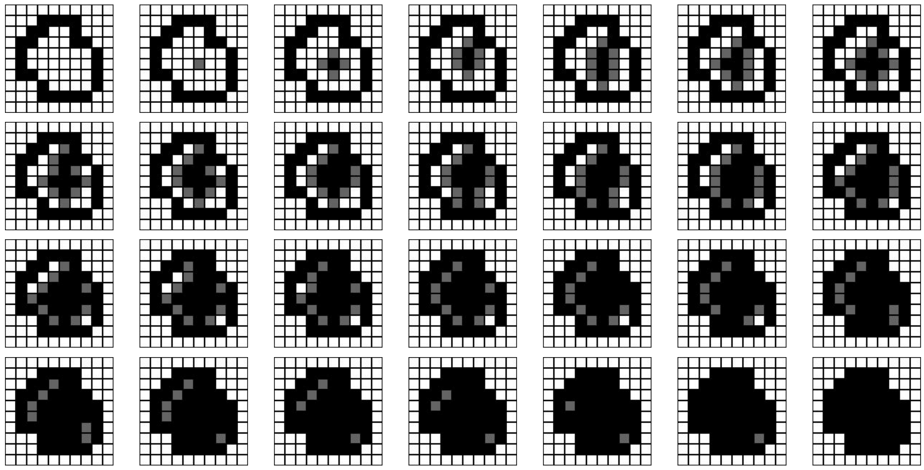


**Figura 5: Preenchimento em uma figura de uma região limitada**

# 6 Exercícios

1. Construa a partir de duas pilhas uma estrutura que se comporte como uma fila. O que se pode afirmar em relação ao desempenho dessa nova estrutura?
2. Um *operador de transparência* em uma fila $\mathcal{F}$, é um operador que permite acessar (somente leitura) quaisquer umas das chaves de $\mathcal{F}$. A ideia é repassar um índice virtual, $j$, entre 1 e $\mathcal{F}.t$, e mapear chaves entre o início e final da fila (note que quando $j = 1$ a posição real é $\mathcal{F}.i$ e quando $j = \mathcal{F}.t$ então a posição real $\mathcal{F}.f$). Implemente um operador de transparência para um fila.
3. Utilize os operadores tradicionais de filas e mais o operador de transparência (questão 2) para resolver o seguinte problema: Dado um vetor $L$ de inteiros de comprimento $n$ e um inteiro $m$, tal que $m < n$, determinar a subsequencia de $L$ de comprimento $m$ cuja soma seja máxima.
4. Uma *deque* é uma fila especial que permite enfileiramento e desenfileiramento em ambas extremidades. Uma deque sequencial usa de base um vetor alocado dinamicamente cujo comprimento é invariante durante a existência da fila. Implementar,
   (a) Descritor de deque sequencial.
   (b) Construtor e destrutor de deque sequencial.
   (c) Operadores PUSHFRONT e PUSHBACK de respectivos enfileiramentos no início e final da deque.
   (d) Operadores POPFRONT e POPBACK de respectivos desenfileiramentos no início e final da deque.
5. Implemente uma deque (ver questão 4) que utilize uma lista encadeada como estrutura de base (note que esta lista precisa ser *duplamente* encadeada).

6. Uma variação do BucketSort para ordenação de números inteiros utiliza a representação em binário dos números e consequentemente dois buckets apenas. Implemente esta variante de BucketSort para inteiros positivos de até 32-bits.
7. Modifique o algoritmo de preenchimento de forma a aceitar contornos abertos. O que muda neste caso?