Algoritmos, Filas

Prof. Ricardo Reis
Universidade Federal do Ceará
Estruturas de Dados

9 de Maio de 2013

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.

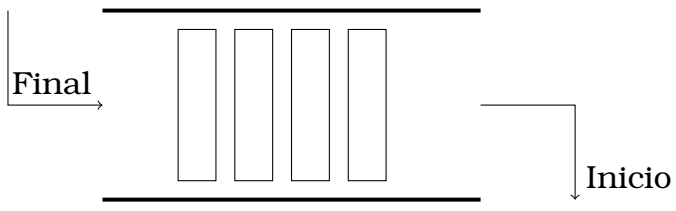


Figura 1: Esquema de fila

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*).

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.

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 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 o operador de construção (CREATE) e o Algoritmo-2 o de destrução (DESTROY) de uma fila sequencial. A finalidade maior do construtor é alocar o vetor de base M (e sua capacidade n)

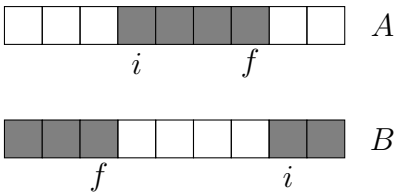


Figura 2: Esquemas de distribuição de chaves numa fila sequencial

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 $\mathcal{Q}$ de entrada, desaloca seu vetor base M e anula os demais campos para zero desabilitando a fila.

Algoritmo 1 Construtor de fila sequencial

```
1: Função CREATE(n)
2:     Q.M ← Alocar(n)
3:     Q.n ← n
4:     Q.i ← Q.f ← 0
5:     Q.t ← 0
6:     Retorne Q
```

Algoritmo 2 Destrutor de fila sequencial

```
1: Função DESTROY(ref Q)
2:     Desalocar Q.M
3:     Q.i ← Q.f ← Q.t ← 0
```

O Algoritmo-3 implementa o operador de enfileiramento em fila sequencial. Ele recebe como entrada uma referência $\mathcal{Q}$ para a fila alvo e a chave x a ser enfileirada. Quando a fila está vazia ($\mathcal{Q}.t = 0$, linha-2) então os valores de $\mathcal{Q}.i$ e $\mathcal{Q}.f$ (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

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	$\mathcal{Q}$, x	Enfileira chave x no final da fila $\mathcal{Q}$
DEQUEUE	$\mathcal{Q}$	Desenfileira objeto do início da fila $\mathcal{Q}$
BEGIN	$\mathcal{Q}$	Retorna valor no início da fila $\mathcal{Q}$
END	$\mathcal{Q}$	Retorna valor no final da fila $\mathcal{Q}$
EMPTY	$\mathcal{Q}$	Verifica se a fila $\mathcal{Q}$ está vazia
FULL	$\mathcal{Q}$	Verifica se a pilha $\mathcal{Q}$ está cheia
DESTROY	$\mathcal{Q}$	Elimina a fila $\mathcal{Q}$

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 $\bmod$ (resto de divisão entre inteiros) conforme indica a expressão,

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

na linha-6.

Quando o p calculado coincide com $\mathcal{Q}.i$ significa que a fila $\mathcal{Q}$ 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 $\mathcal{Q}.f$ 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.

O operador de desenfileiramento em fila sequencial é implementado pelo Algoritmo-4. O operador DEQUEUE recebe a fila $\mathcal{Q}$ e lhe efetua um desenfileiramento. Quando $\mathcal{Q}$ não está vazia (teste da linha-2 obtém sucesso) o operador trata duas situações: a de

Algoritmo 3 Enfileiramento em fila sequencial

```
 1: Função ENQUEUE(ref Q, x)
 2:     Se Q.t = 0 então
 3:         Q.i ← Q.f ← 1
 4:         Q.M[1] ← x
 5:     senão
 6:         p ← 1 + Q.f mod Q.n
 7:         Se p ≠ Q.i então
 8:             Q.M[p] ← x
 9:             Q.f ← p
10:         senão
11:             Retorne Falso
12:     Q.t ← Q.t + 1
13:     Retorne Verdadeiro
```

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 $\mathcal{Q}.i$ e $\mathcal{Q}.f$ são *apontados* para fora da fila (linha-4). Do contrário, se o teste na linha-3 falha então o início da fila $\mathcal{Q}.i$ (que é a extremidade de retiradas) precisa ser recalculado. Similar ao que ocorre no enfileiramento, o reajuste de $\mathcal{Q}.i$ deve ser feito pelo mecanismo da circularidade que garante que o índice voltará a 1 ao tentar extrapolar o final de $\mathcal{Q}.M$. O novo valor de $\mathcal{Q}.i$ é dado por,

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

conforme implementado na linha-6. Quando $\mathcal{Q}$ 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 $\mathcal{Q}.t$ 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 4 Desenfileiramento em fila sequencial

```
 1: Função DEQUEUE(ref Q)
 2:     Se Q.t > 0 então
 3:         Se Q.t = 1 então
 4:             Q.i ← Q.f ← 0
 5:         senão
 6:             Q.i ← 1 + Q.i mod Q.n
 7:     senão
 8:         Retorne Falso
 9:     Q.t ← Q.t − 1
10:     Retorne Verdadeiro
```

No Algoritmo-5 são implementados os demais operadores de filas sequenciais cada qual recebendo a fila $\mathcal{Q}$ de operação. Respectivamente BEGIN e END retornam os valores das chaves no início e final da fila $\mathcal{Q}$ (valores das chaves de $\mathcal{Q}.M$ nas posições $\mathcal{Q}.f$ e $\mathcal{Q}.i$). EMPTY e FULL testam se a fila está vazia ($\mathcal{Q}.t = 0$) e cheia ($\mathcal{Q}.t = \mathcal{Q}.n$) respectivamente.

Algoritmo 5 Demais operadores em filas sequenciais.

```
 1: Função BEGIN(ref Q)
 2:     Retorne Q.M[Q.i]                ▷ Chave inicial

 3: Função END(ref Q)
 4:     Retorne Q.M[Q.f]                ▷ Chave final

 5: Função EMPTY(ref Q)
 6:     Retorne Q.t = 0                 ▷ Vazia?

 7: Função FULL(ref Q)
 8:     Retorne Q.t = Q.n               ▷ Cheia?
```

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 nodo inicial da lista base e f para o no do final. Quando a fila está vazia as duas referências apontam para λ.

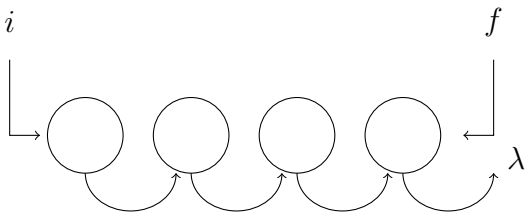


Figura 3: Esquema de fila encadeada

Na Figura-3 é ilustrada uma fila encadeada. O no do-raiz da lista de base é apontado por i ao passo que o último no do 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 nodos 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 no do final é conhecida), a remoção deve ocorrer em $O(n)$ haja vista o último no do 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 nodos). 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.

A Tabela-3 ilustra o descritor de um no do de uma fila encadeada. É exatamente a

Tabela 2: Descritor de uma fila Encadeada

Atributo	**Descrição**
i	Referência para o no do inicial da fila
f	Referência para o no do final da fila
t	comprimento da fila

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

Tabela 3: Descrição de um no do de uma fila encadeada

Campo	**Descrição**
$chave$	Valor da chave no no do
$prox$	Referência ao no do seguinte (vale λ quando é o último no do)

mesma composição de um no do de lista encadeada, ou seja, possui um campo para a chave ($chave$) e outro para referenciamento do no do seguinte ($prox$).

O Algoritmo-6 implementa o operador de construção e o Algoritmo-7 implementa o operador de destrução de fila encadeada. O construtor CREATE configura os campos i e f de uma nova fila encadeada $\mathcal{Q}$ para λ 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 $\mathcal{Q}$. O processo é semelhante a desalocação de listas encadeadas: um laço principal (linha-2) faz avançar a raiz da estrutura (linha-4) eliminando gradativamente os nodos que vão ficando para trás (linha-5). Note que esse laço conduz $\mathcal{Q}.i$ a λ sendo ainda necessário, para anulação completa da fila, fazer $\mathcal{Q}.f$ apontar para λ (linha-6) e $\mathcal{Q}.t$ receber zero (linha-7).

Algoritmo 6 Construtor de fila encadeada

```
1: Função CREATE
2:     Q.i ← Q.f ← λ
3:     Q.t ← 0
4:     Retorne Q
```

Algoritmo 7 Destrutor de fila encadeada

```
1: Função DESTROY(ref Q)
2:     Enquanto Q.i ≠ λ faça
3:         N ← Q.i
4:         Q.i ← Q.i.prox
5:         Desalocar N
6:     Q.f ← λ
7:     Q.t ← 0
```

No Algoritmo-8 é implementado o operador ENQUEUE de enfileiramento de uma chave x, passada como argumento, em uma fila encadeada $\mathcal{Q}$. Neste operador é criado um no do $\mathcal{N}$ cujos campos $chave$ e $prox$ são respectivamente configurados com o valor de x e λ. O novo no do então é conectado ao final da fila (linha-4) e $\mathcal{Q}.f$ atualizado para o novo último no do (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-9 é implementado o operador DEQUEUE de desenfileiramento em uma fila encadeada $\mathcal{Q}$ passada como argumento. Este operador testa se $\mathcal{Q}$ não está vazia

Algoritmo 8 Enfileiramento em fila encadeada

```
1: Função ENQUEUE(ref Q, x)
2:     N.chave ← x
3:     N.prox ← λ
4:     Q.f.prox ← N
5:     Q.f ← N
6:     Q.t ← Q.t + 1
```

(linha-2) e em caso de êxito elimina o primeiro no do da lista base. A eliminação possui as três etapas já conhecidas: (i) reserva do primeiro no do através de uma variável auxiliar (linha-3); (ii) deslocamento da raiz $\mathcal{Q}.i$ para a posição adiante (linha-4); (iii) eliminação do no do reservado (linha-5). Na linha-6 o comprimento da fila é corrigido. A linha-7 e a linha-9 são dedicadas a retorno de função: **Verdadeiro** indica que houve desenfileiramento e **Falso** que a fila estava vazia e nenhum desenfileiramento pôde ser efetuado.

Algoritmo 9 Desenfileiramento em fila encadeada

```
1: Função DEQUEUE(ref Q)
2:     Se Q.t > 0 então
3:         N ← Q.i
4:         Q.i ← Q.i.prox
5:         Desalocar N
6:         Q.t ← Q.t − 1
7:         Retorne Verdadeiro
8:     senão
9:         Retorne Falso
```

Os demais operadores de filas encadeadas são implementados no Algoritmo-10. BEGIN e END retornam respectivamente as chaves do primeiro e último nodos da fila $\mathcal{Q}$ (argumento de entrada) lendo o campo $chave$ dos nodos apontados pelos campos i e f. EMPTY testa se a a fila de entrada $\mathcal{Q}$ está vazia apenas verificando se o campo t é nulo.

Algoritmo 10 Demais operadores em filas encadeadas

```
1: Função BEGIN(Q)
2:     Retorne Q.i.chave

3: Função END(Q)
4:     Retorne Q.f.chave

5: Função EMPTY(Q)
6:     Retorne Q.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 dígi-

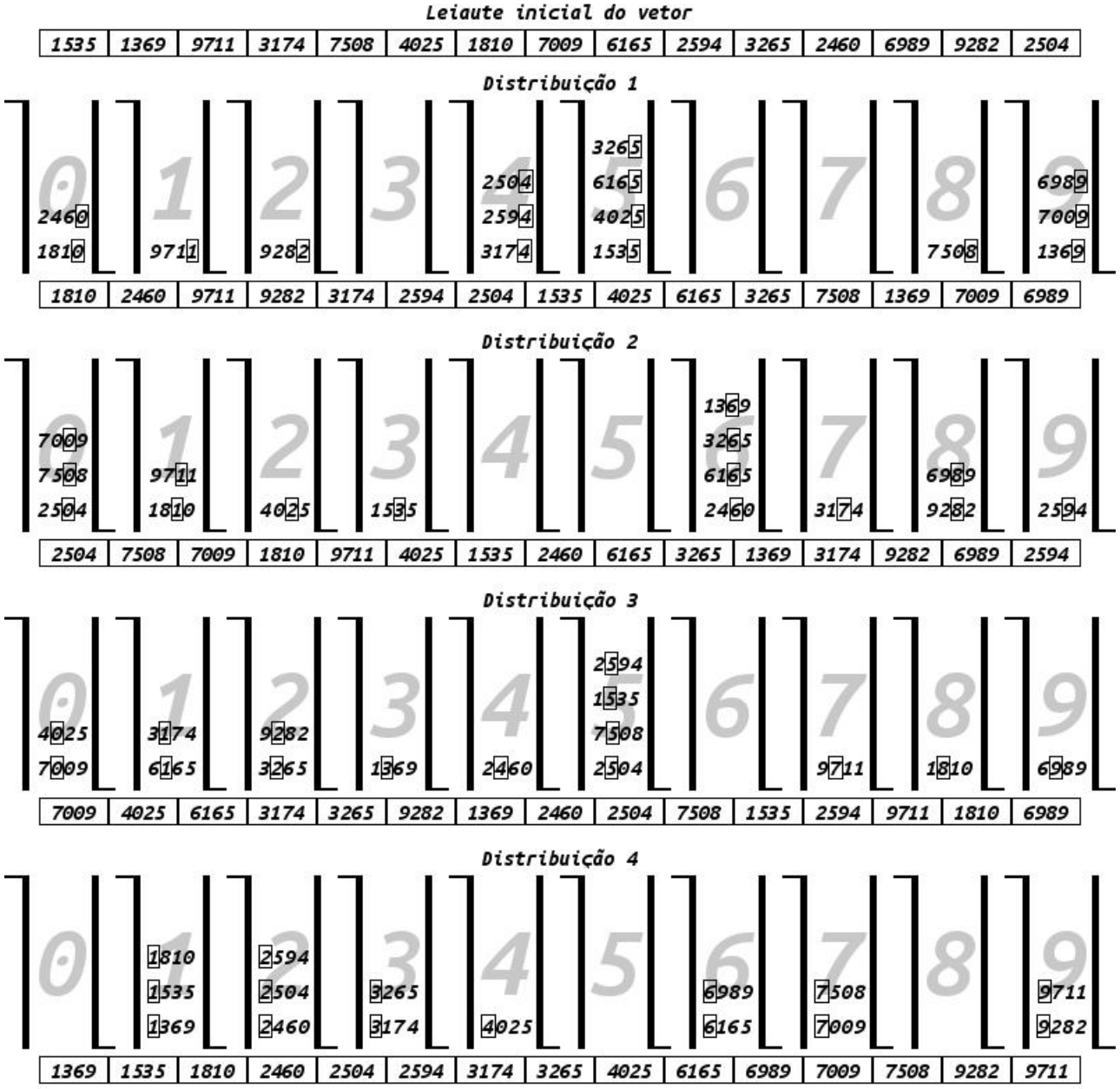


Figura 4: Distribuições efetuadas em uma ordenação por raiz (BucketSort/RadixSort)

tos de controle 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 enfileiradas nos buckets, elas são desenfileiradas e levadas de volta a U. Os desenfileiramentos ocorrem em ordem crescente de rótulos dos buckets e, em cada bucket, procedem continuamente até esvaziamento. O repovoamento de U ocorre sequencialmente, desde a primeira posição, e avança conforme as chaves são desenfileiradas dos buckets e reinseridas em U. O número de distribuições efetuadas é igual ao número máximo d de dígitos que ocorre entre as chaves de U.

A Figura-4 ilustra as distribuições executadas num dado vetor cujas chaves (inteiros) possuem até quatro dígitos (por isso ocorre um total de quatro distribuições). Em cada distribuição é mostrado o leiaute de buckets (filas) após a transparência de chaves do vetor (enfileiramento por cima e desenfileiramento por baixo) e o leiaute do vetor após esvaziamento dos buckets. Os rótulos numéricos em marca d'água em cada bucket indicam a que dígito na base decimal ele se refere. Os dígitos de controle

Algoritmo 11 Ordenação por Raiz (RadixSort)

```
1: Função RadixSort(ref L, n, d)
2:     Para i ← 1..10 faça
3:         buckets[i] ← Create()                ▷ Cria dez filas vazias para representar os buckets
4:     m ← 1                                    ▷ Controlador de dígitos de referência
5:     Para i ← 1..d faça                       ▷ Laço das distribuições
6:         Para j ← 1..n faça                   ▷ Povoamento de buckets da distribuição corrente
7:             dig ← ⌊L[j]/m⌋ mod 10
8:             Enqueue(buckets[dig + 1], L[j])  ▷ Enfileiramentos
9:         k ← 1
10:        Para i ← 1..10 faça                  ▷ Esvaziamento de buckets com repovoamento de L
11:            Enquanto não Empty(buckets[i]) faça
12:                L[k] ← Begin(buckets[i])
13:                Dequeue(buckets[i])          ▷ Desenfileiramentos
14:                k ← k + 1
15:        m ← m × 10
```

de cada distribuição são envolvidos por retângulos.

O Algoritmo-11 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 dez filas que funcionam como buckets (os índices estão 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 efetua d iterações. Para cada uma destas iterações o laço da linha-6 enfileira as chaves de L nos buckets. Tais enfileiramentos ocorrem de acordo com a família de dígitos das chaves (unidade, dezena, centena e etc) da distribuição em progresso. Os dígitos mapeiam os buckets de destino correspondentes.

O k-ésimo dígito de um inteiro x (medido de trás para frente) é dada por,

$$dig_k(x) \leftarrow \left\lfloor \frac{x}{10^{k-1}} \right\rfloor \bmod 10 \qquad (1)$$

Na linha-7 o dígito de controle da distribuição corrente é determinado por uma reformulação da Equação-1 que substitui 10^{k-1} pela variável m. A ideia é usar a estrutura do próprio laço e fazer m (que inicia com valor 1, linha-4), em cada iteração, ser multiplicado por 10 (linha-15). O efeito é equivalente a potência de dez da Equação-1, porém computacionalmente menos dispendioso. Note ainda que, como os valores de dígitos calculados ficam na faixa $0..9$, o mapeamento do bucket correspondente é feito pela expressão $dig + 1$ (linha-8).

Os enfileiramentos ocorrem na linha-8 e os desenfileiramentos na linha-13 os quais esvaziam os buckets e repovoam L. No repovoamento de L cada chave desenfileirada dos buckets é copiada para L na posição k (inicialmente k vale 1, linha-9). O valor de k é incrementado e cada iteração na linha-14.

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

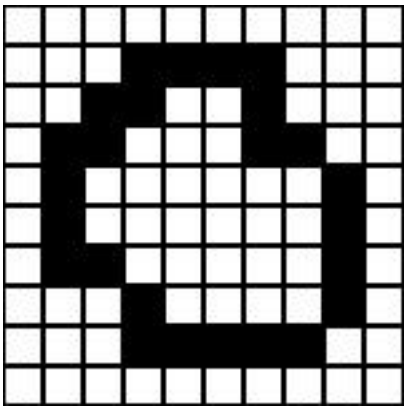


Figura 5: Imagem com região limitada por uma linha fechada.

$(m,\, n)$.

Consideremos o problema de preenchimento de uma região fechada em uma imagem I, ou seja, é dada uma imagem contendo uma disposição de píxels de mesma cor que formam uma linha de contorno fechada cujo interior deseja-se preencher (com a mesma cor do contorno).

Seja uma imagem I dada de resolução 10×10 cujos píxels podem assumir três cores: branco, cinza e preto. Inicialmente I é uma região branca contendo um contorno preto fechado como ilustrado na Figura-5. Deseja-se preencher em preto a região branca limitada pelo contorno preto. A ideia é partir de um pixel interno ao contorno (branco) e efetuar um preenchimento que pinta gradativamente as vizinhanças e se expande até que o contorno seja encontrado (veja Figura-6).

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.

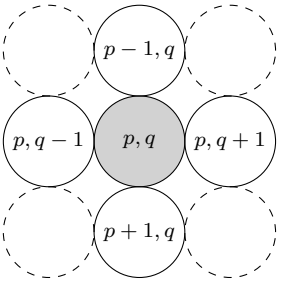
2. Tomar a cor de Q (em nosso exemplo é necessariamente **BRANCO**) como cor de referência, ou seja, todos os píxels de mesma cor nas vizinhanças deverão ser pintados de **PRETO**.

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

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

5. Promover o desenfileiramento sequencial de $\mathcal{Q}$ até seu esvaziamento e para cada posição $\{p,\, q\}$ desenfileirada devese,

   (a) Colorir em **CINZA** e depois enfileirar em $\mathcal{Q}$ 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 $\mathcal{Q}$. 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-6 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-12 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, $\mathcal{Q}$. 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 $\mathcal{Q}$. Na

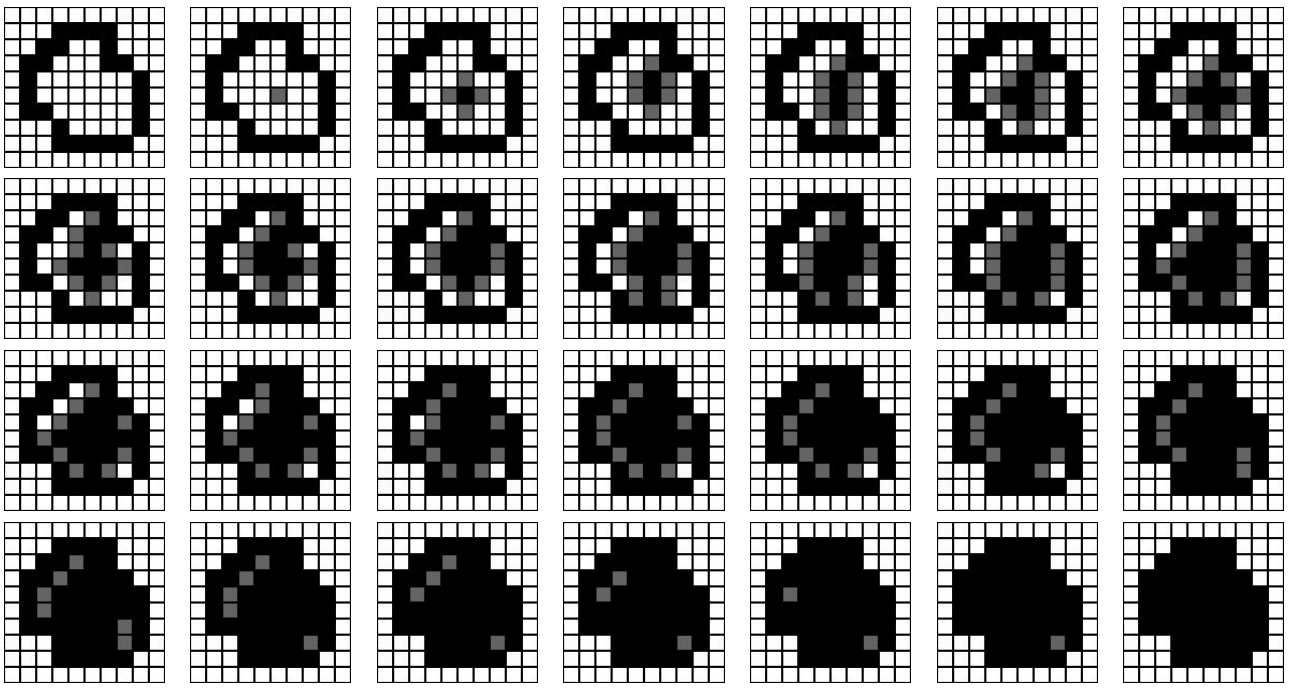


Figura 6: Preenchimento em uma figura de uma região limitada

linha-6 e na linha-7 ocorre um desenfileiramento que é armazenado na posição $\{p, q\}$. Na linha-9 é criado o vetor V das quatro posições vizinhas cardeais a $\{p, q\}$ que são candidatas a serem enfileiradas. O laço da linha-10 varre V e tenta fazer os enfileiramentos. Na linha-11 as variáveis i e j são utilizadas para recuperar cada uma das posições vizinhas cardeais de V. O teste da linha-12 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-13) e enfileiradas em $\mathcal{Q}$ (linha-14). As posições cujas vizinhanças já estão resolvidas são pintadas de **PRETO** conforme linha-15.

Algoritmo 12 Preenchimento de região interna a um contorno em uma figura

```
 1: Função FloodFill(ref I, lin, col)
 2:     Q ← Create()
 3:     cor ← I[lin, col]
 4:     Enqueue(Q, {lin, col})
 5:     Enquanto não Empty(Q) faça
 6:         p, q ← Begin(Q)
 7:         Dequeue(Q)
 8:         V ← [(p − 1, q), (p + 1, q),
 9:               (p, q − 1), (p, q + 1)]
10:         Para k ← 1..4 faça
11:             i, j ← V[k]
12:             Se I[i, j] = cor então
13:                 I[i, j] ← CINZA
14:                 Enqueue(Q, V[k])
15:             I[p, q] ← PRETO
```

6 Exercícios

Ilustrar todas as distribuições na ordenação por raiz dos vetores a seguir,

1. 259, 385, 718, 557, 548, 855, 895, 974, 725, 282, 470, 872
2. 435, 803, 826, 587, 338, 138, 984, 470, 905, 642, 731, 729
3. 7794, 8039, 8040, 8870, 7426, 661, 3812, 6649, 5447, 7341, 1991, 2457

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,

4. Descritor de uma deque sequencial.
5. Construtor e destrutor de uma deque sequencial.
6. Operadores PUSHFRONT e PUSHBACK de respectivos enfileiramentos no início e final de uma deque sequencial.
7. Operadores POPFRONT e POPBACK de respectivos desenfileiramentos no início e final de uma deque sequencial.
8. Implemente uma deque que utilize uma lista encadeada como estrutura de base (note que esta lista precisa ser *duplamente* encadeada).

Resolva os problemas seguintes relacionados a BucketSort,

9. Como se pode modificar o RadixSort de forma que números inteiros negativos possam também estar presentes? Porque na versão do Algoritmo-11 isso não era possível?
10. Modifique o Algoritmo-11 de forma que se aplique a ordenação de strings.
11. Proponha um algoritmo de ordenação BucketSort para ordenação de números de ponto flutuante.
12. Construa uma variação do Algoritmo-11 para ordenação de números inteiros positivos utilizando suas representações binárias (note que isso implica utilização de apenas dois buckets).
13. Ao ordenar-se inteiros de 32 bits pelo algoritmo da questão-12 devem ser realizadas 32 distribuições. Na prática isso é um desperdício pois em muitos casos o maior inteiro (em bits) de um vetor precisa bem menos que 32 bits. Proponha um teste a ser feito no final de uma distribuição que determine se a distribuição seguinte precisa ou não ser executada. Qual a complexidade deste teste?

Resolva os problemas a seguir referentes ao algoritmo de preenchimento em imagens,

14. Modifique o algoritmo de preenchimento em imagens de forma a aceitar contornos abertos. O que muda neste caso?
15. O que muda no algoritmo de preenchimento no caso de figuras com muitas cores? O contorno pode ser composto por mais de uma cor? A fila precisa de mais alguma condição de enfileiramento?
16. Seja I uma imagem de resolução $n \times n$. Qual a ordem de comprimento máxima a que pode chegar uma fila utilizada no preenchimento de I?
17. Se, no Algoritmo-12, ao invés dos quatro píxels cardeais fossem utilizados os oito píxels de vizinhança o que mudaria? Seia vantajoso ou desvantajoso?
18. Seja I uma imagem em escala cinza, ou seja, os píxels podem assumir apenas tonalidades de cinza. Em geral um tom de cinza pode ser representado por um número de ponto flutuante entre zero e um onde *zero* é preto e *um* é branco. Suponhamos que I contenha um fundo branco e uma região A fechada contornada por uma linha preta. Um preenchimento em *degradê* de A, a partir de um pixel P pertencente a A, corresponde a um processo de coloração dos píxels de A onde P é mantido branco e os demais píxels pintados em tons de cinza em valores decrescentes de tons desde P (onde vale *um*) até um píxel do contorno (onde vale *zero*). Proponha e implemente um algoritmo de preenchimento em degradê.

Resolva os seguintes problemas,

19. 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?
20. Um *operador de transparência* em uma fila $\mathcal{Q}$, é um operador que permite acessar (somente leitura) quaisquer umas das chaves de $\mathcal{Q}$. A ideia é repassar um índice virtual, j, entre 1 e $\mathcal{Q}.t$, e mapear chaves entre o início e final da fila (note que quando $j = 1$ a posição real é $\mathcal{Q}.i$ e quando $j = \mathcal{Q}.t$ então a posição real $\mathcal{Q}.f$). Implemente um operador de transparência para um fila.

21. Utilize os operadores tradicionais de filas e mais o operador de transparência (questão 20) 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.