Estruturas de Informação I - Semana 3

2. Listas baseadas em Arrays

João Araujo Ribeiro

jaraujo@uerj.br

Departamento de Engenharia de Sistemas e Computação

Universidade do Estado do Rio de Janeiro

Resumo

Nesta aula vamos estudar as listas e filas cuja estrutura de base é um array.

`http://www.opendatastructures.org/ods-python/2_Array_Based_Lists.html`

2. Listas Baseadas em Arrays
    - 2.1 ArrayStack: Operações rápidas usando um Array
    - 2.2 Cópia otimizada de Array
    - 2.3 Fila Baseada em Array
    - 2.4 ArrayDeque
    - 2.5 DualArrayDeque
    - 2.6 RootishArrayStack
    - 2.7 Exercícios

Tabela dos tempos de execução das operações nas estruturas de dados desta aula:

	$\text{get}(i)/\text{set}(i,x)$	$\text{add}(i,x)/\text{remove}(i)$
ArrayStack	$O(1)$	$O(n-i)$
ArrayDeque	$O(1)$	$O(\min\{i, n-i\})$
DualArrayDeque	$O(1)$	$O(\min\{i, n-i\})$
RootishArrayStack	$O(1)$	$O(n-i)$

Em uma lista sequencial, o sucessor de um elemento ocupa a posição física subsequente deste elemento. Uma das formas mais comuns de se implementar uma lista sequencial é utilizando ARRAY.

Armazenamento

Em um array associamos a cada elemento um índice (denominamos elemento a_i).

Desta forma, estamos armazenando o elemento a_i e a_{i+1}nas posições consecutivas *i* e *i+1* do array.

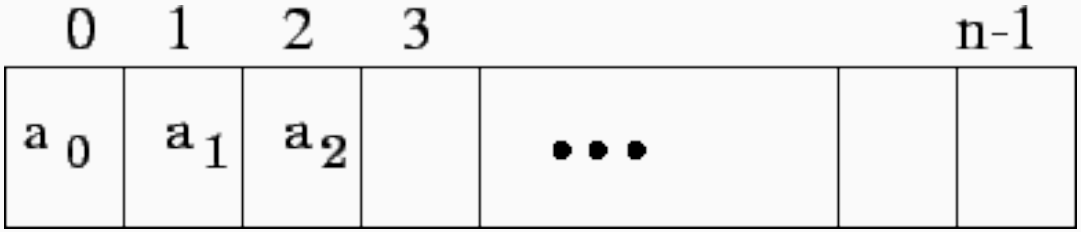


Figura 1: Armazenamento de Lista Sequencial

Vantagens em se usar array

1. Rápido acesso aos elementos
2. Facilidade em modificar informações

Os elementos do array podem ser acessados em tempo constante, por exemplo, na maioria das linguagens, o elemento que está na posição *i* de um array *a* pode ser recuperado utilizando-se *a[i]*

Desvantagens

1. Definição prévia do tamanho do array
2. Dificuldade para inserir (e remover) um elemento entre dois outros já existentes

Na definição de um array precisamos especificar o número máximo de elementos que serão armazenados e manter um registro do número de elementos armazenados

Inserção

Suponha que queiramos inserir um novo elemento x em uma posição i já ocupada de um vetor a. Para realizar esta tarefa teremos que deslocar os elementos para as posições seguintes. Note que no pior caso (inserir na primeira posição) esta operação leva tempo $O(n)$.

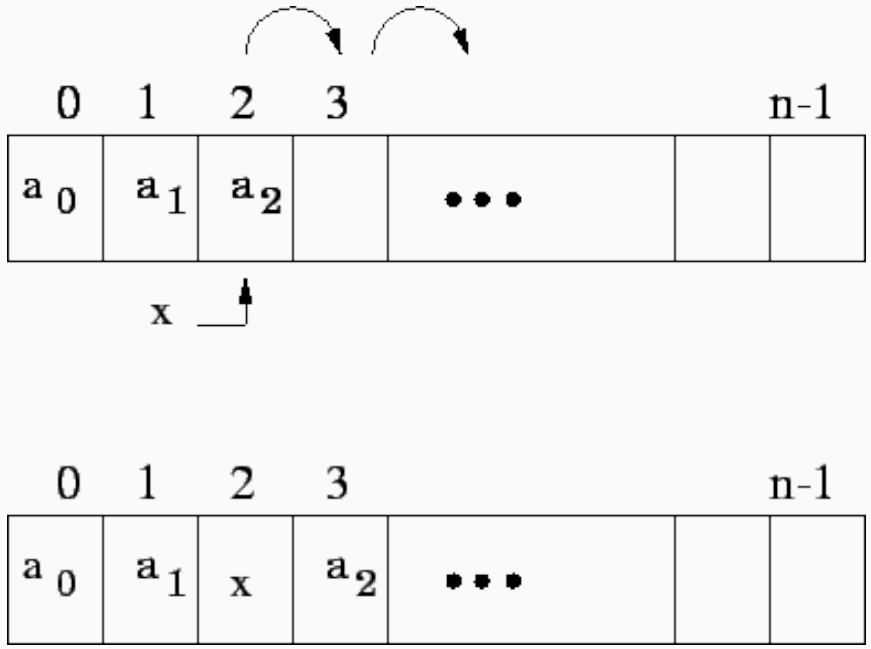


Figura 2: Inserção de elemento

2. Listas Baseadas em Arrays

2.1 ArrayStack: Operações rápidas usando um Array

Inicialização

Um elemento da lista com índice i é armazenado em $a[i]$. Na maioria das vezes, a é maior que o estritamente necessário, assim um inteiro n é usado para manter um registro do número de elementos realmente armazenados em a. Deste modo, os elementos da lista são armazenados em $a[0], \ldots, a[n-1]$ e, a todo instante, $\text{length}(a) \geq n$.

```
initialize()
    a ← new_array(1)
    n ← 0
```

Operações Básicas

$\text{get}(i)$

    return $a[i]$

$\text{set}(i, x)$

    $y \leftarrow a[i]$

    $a[i] \leftarrow x$

    return y

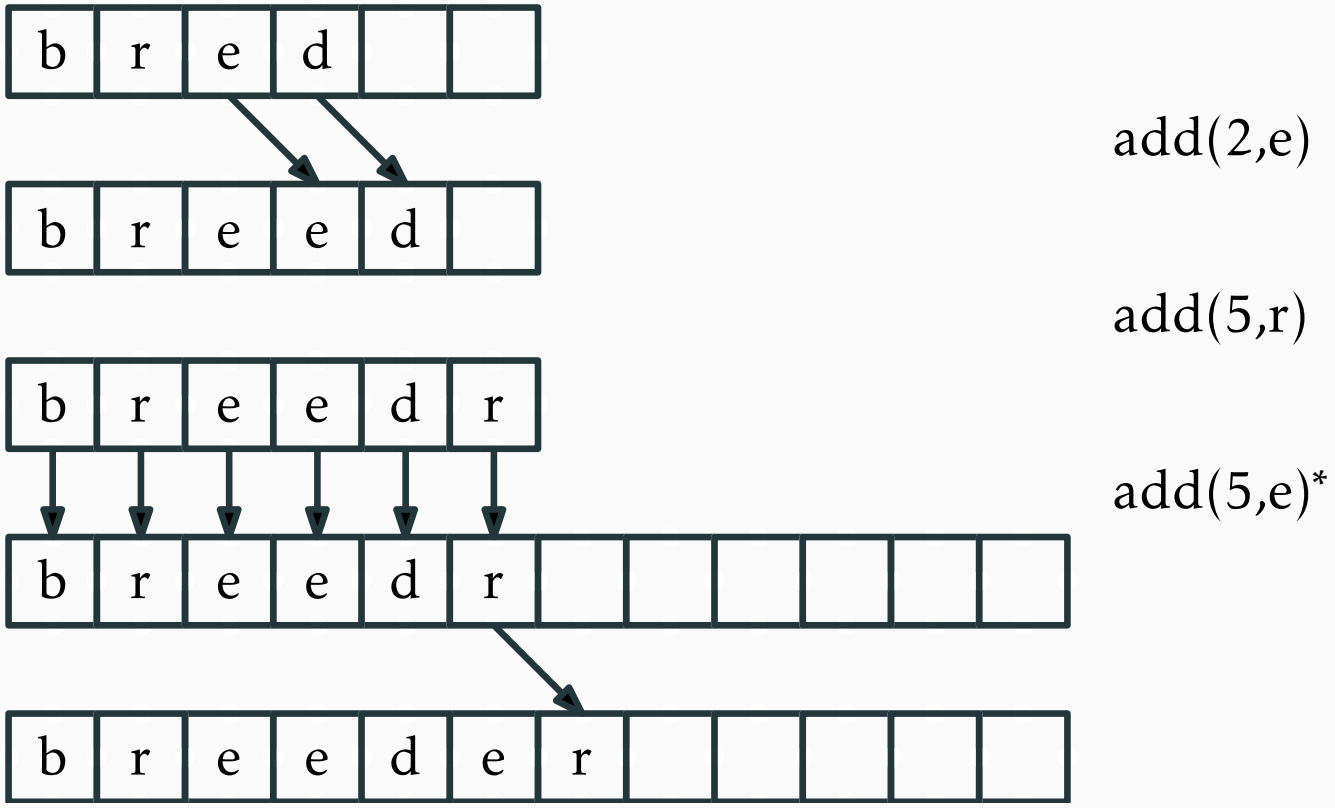


Figura 3: Uma sequência de operações $\mathrm{add}(i, x)$ em uma ArrayStack. Setas indicam elementos sendo copiados. Operações que implicam uma chamada para $\mathrm{resize}()$ são marcadas com asterisco.

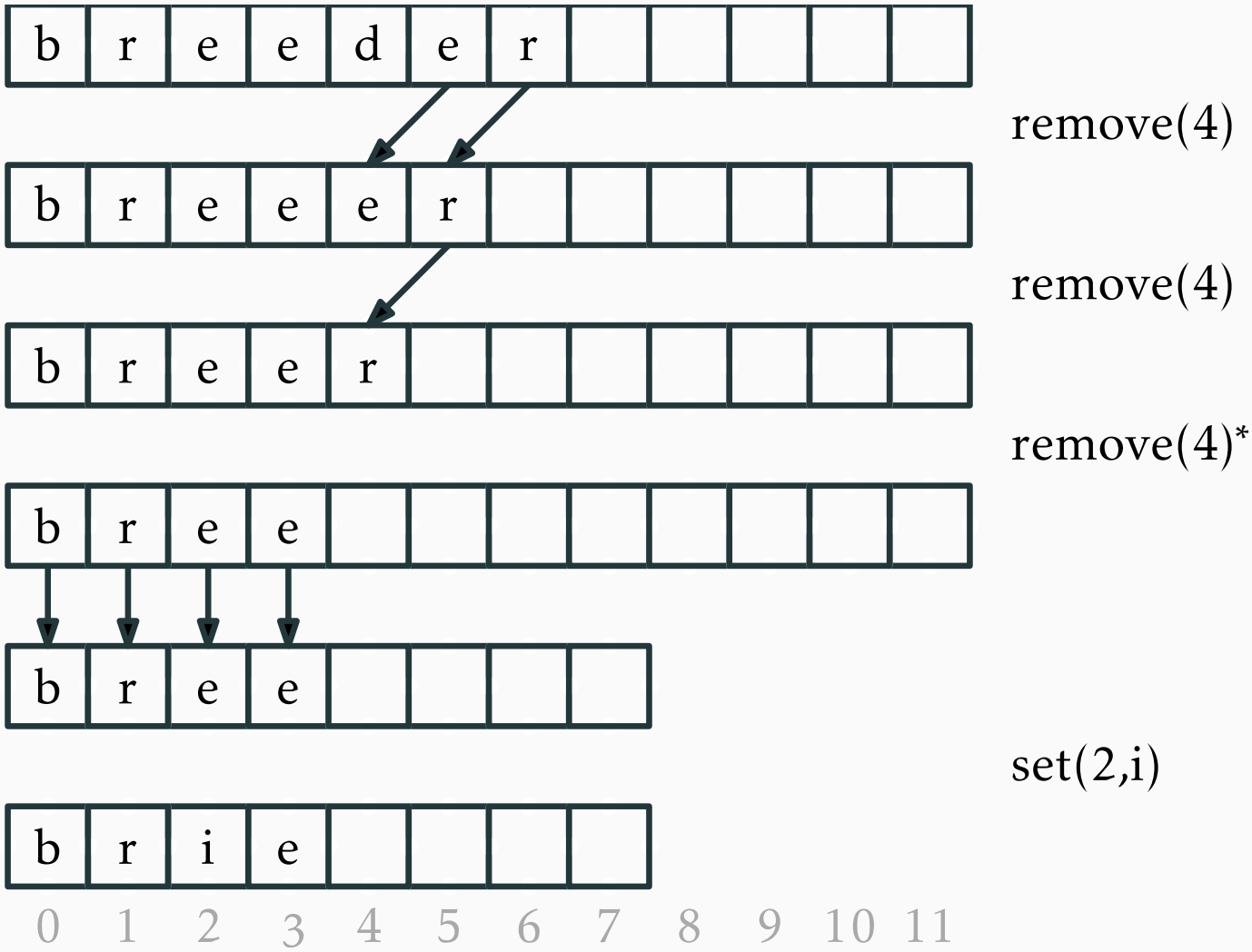


Figura 4: Uma sequência de operações $\mathrm{remove}(i)$.

Add

$\text{add}(i, x)$
    if $n = \text{length}(a)$ **then** $\text{resize}()$
    $a[i+1, i+2, \ldots, n] \leftarrow a[i, i+1, \ldots, n-1]$
    $a[i] \leftarrow x$
    $n \leftarrow n + 1$

Custo da operação (ignorando $\text{resize}()$) = $O(n - i)$

Remove

```
remove(i)
    x ← a[i]
    a[i, i + 1, ..., n − 2] ← a[i + 1, i + 2, ..., n − 1]
    n ← n − 1
    if length(a) ≥ 3 · n then resize()
    return x
```

Crescendo e Encolhendo

resize()

$b \leftarrow \text{new\_array}(\max(1, 2 \cdot n))$

$b[0, 1, \ldots, n-1] \leftarrow a[0, 1, \ldots, n-1]$

$a \leftarrow b$

Uma ArrayStack é uma maneira eficiente de implementar uma Stack. Particularmente, podemos implementar $\mathrm{push}(x)$ como $\mathrm{add}(n, x)$ e $\mathrm{pop}()$ como $\mathrm{remove}(n-1)$, em cada caso, estas operações executaram em um tempo amortizado de $O(1)$.

2. Listas Baseadas em Arrays

2.2 Cópia otimizada de Array

2.2 Cópia otimizada de Array

Algumas linguagens permitem uma cópia otimizada de áreas de memória, acelerando a operação de resize. Na linguagem C temos as funções $\mathrm{memcpy}(d, s, n)$ e $\mathrm{memmove}(d, s, n)$. Na linguagem C++ temos $\mathrm{stdcopy}(a_0, a_1, b)$. Java possui o método $\mathrm{System.arraycopy}(s, i, d, j, n)$.

2. Listas Baseadas em Arrays

2.3 Fila Baseada em Array

2.3 Fila Baseada em Array

A estrutura ArrayQueue implementa uma fila FIFO (first-in-first-out) na qual os elementos são removidos (usando a operação $\mathrm{remove}()$) da fila na mesma ordem em que são inseridos (usando a operação $\mathrm{add}(x)$).

Como implementar?

Por que não ArrayStack?

Como implementar?

Array infinito? Com j indicando o elemento a ser removido e n o número de elementos na Fila.

Os elementos são armazenados em:

$[a[j], a[j+1], \ldots, a[j+n-1]\ \ ]$

Inicialmente, ambos j e n são 0. Para acrescentar um elemento, devemos colocá-lo em $a[j+n]$ e incrementar n. Para remover um elemento, devemos removê-lo de $a[j]$, incrementar j, e decrementar n.

Qual o problema desta solução?

Aritmética modular

Podemos simular um Array infinito usando um array finito a e *aritmética modular*.

Esta também é chamada de aritmética do relógio. Por exemplo 10:00 mais cinco dá 3:00. Formalmente, dizemos que

$$10 + 5 = 15 \equiv 3 \pmod{12} \ .$$

Que lemos como "15 é congruente a 3 módulo 12." Também podemos tratar mod como um operador, de modo que

$$15 \bmod 12 = 3 \ .$$

Array finito tratado como infinito

As posições passam a ser: $a[j \bmod length(a)], a[(j + 1) \bmod length(a)], \ldots, a[(j + n - 1) \bmod a)]$.]

Isto trata o Array a como um Array *circular* cujos índices que são maiores que $length(a) - 1$ "voltam" ao início do array.

Inicialização

initialize()

$a \leftarrow \text{new\_array}(1)$

$j \leftarrow 0$

$n \leftarrow 0$

Inserindo e removendo de um ArrayQueue

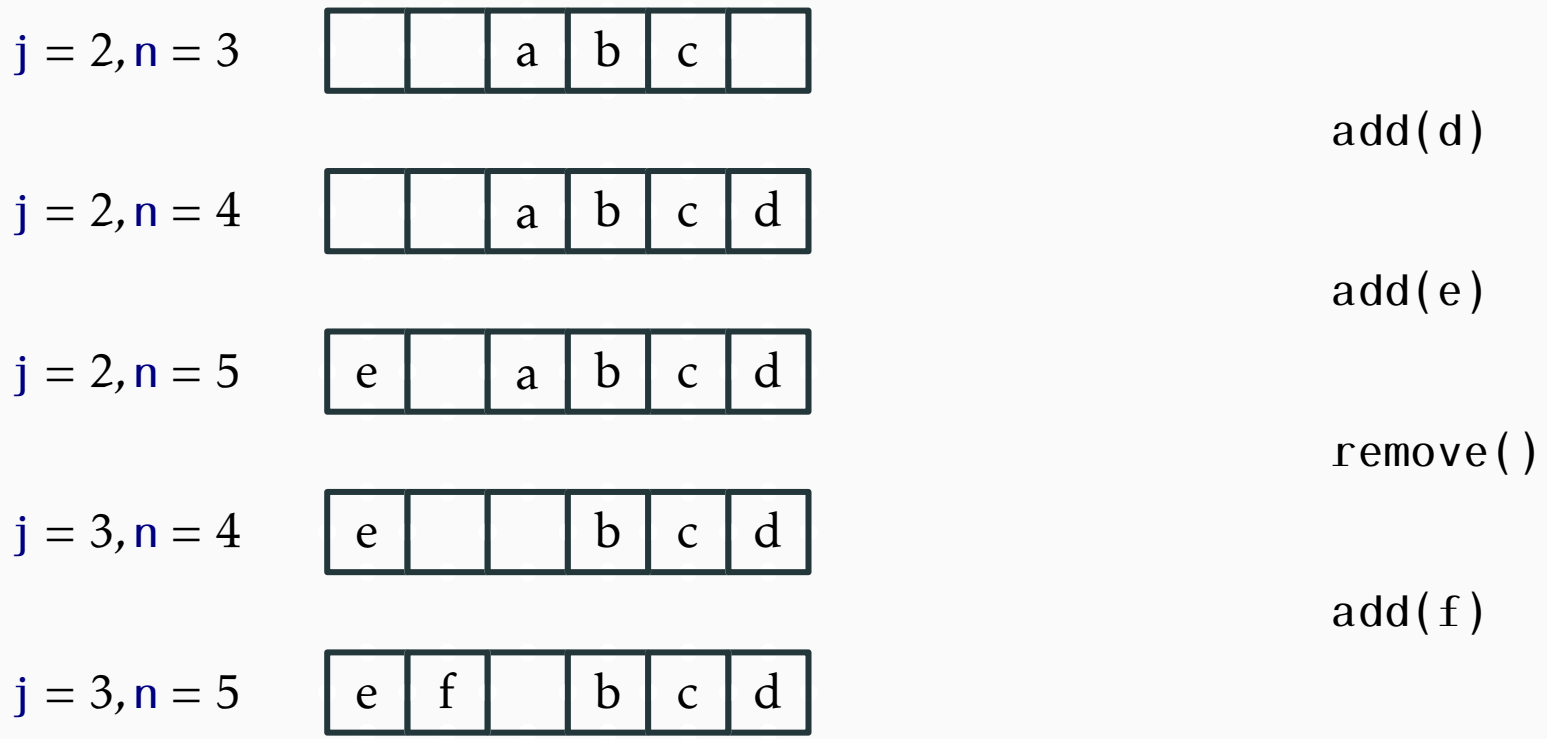


Figura 5: Uma sequência de $\mathrm{add}(x)$ e $\mathrm{remove}(i)$.

Inserindo e removendo de um ArrayQueue

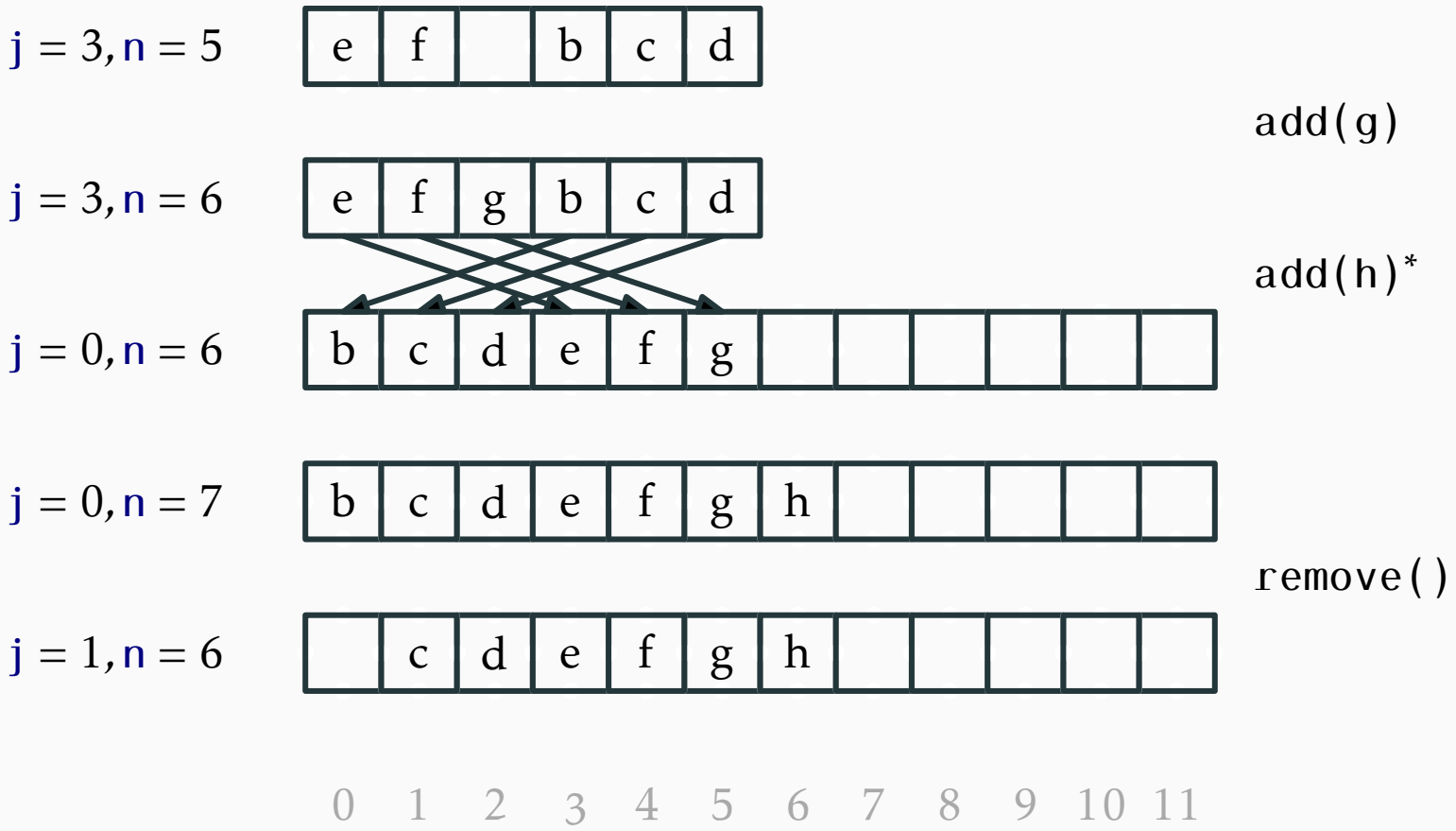


Figura 6: Uma sequência de $\mathrm{add}(x)$ e $\mathrm{remove}(i)$.

Add()

```
add(x)
    if n + 1 > length(a) then resize()
    a[(j + n) mod length(a)] ← x
    n ← n + 1
    return true
```

remove()

$x \leftarrow a[j]$

$j \leftarrow (j + 1) \bmod \text{length}(a)$

$n \leftarrow n - 1$

if $\text{length}(a) \geq 3 \cdot n$ **then** resize()

return x

resize()

resize()

$b \leftarrow \text{new\_array}(\max(1, 2 \cdot n))$

for k **in** $0, 1, 2, \ldots, n-1$ **do**

$b[k] \leftarrow a[(j+k) \bmod \text{length}(a)]$

$a \leftarrow b$

$j \leftarrow 0$

2. Listas Baseadas em Arrays

2.4 ArrayDeque

Usando a mesma estrutura circular.

```
initialize()
    a ← new_array(1)
    j ← 0
    n ← 0
```

$\text{get}(i)$

    return $a[(i + j) \bmod \text{length}(a)]$

$\text{set}(i, x)$

    $y \leftarrow a[(i + j) \bmod \text{length}(a)]$

    $a[(i + j) \bmod \text{length}(a)] \leftarrow x$

    return y

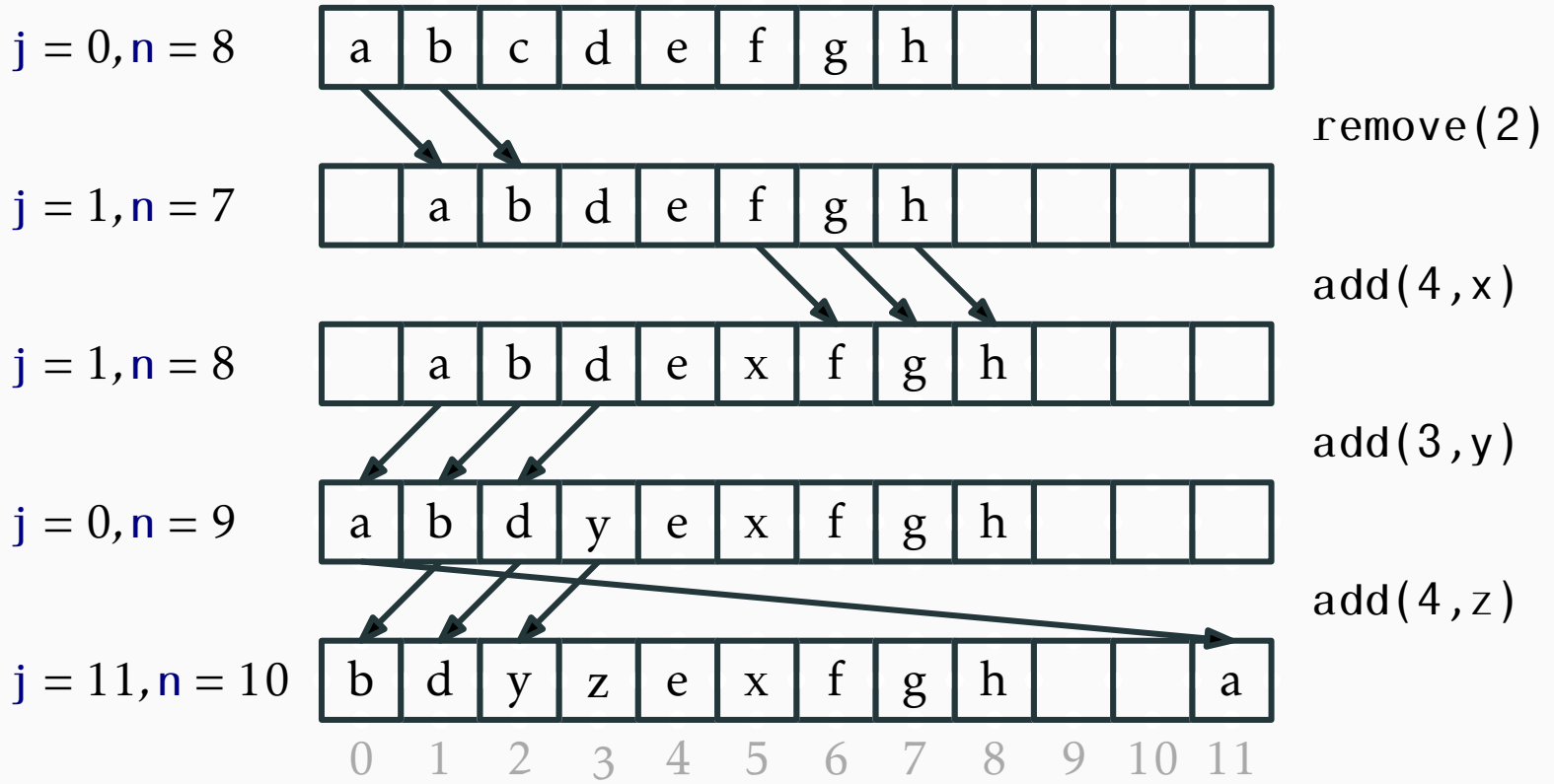


Figura 7: Uma sequência de operações $\mathrm{add}(i, x)$ e $\mathrm{remove}(i)$ sobre um ArrayDeque.

$Add(i, x)$

```
add(i, x)
    if n = length(a) then resize()
    if i < n/2 then
        j ← (j − 1) mod length(a)
        for k in 0, 1, 2, ..., i − 1 do
            a[(j + k) mod length(a)] ←
a[(j + k + 1) mod length(a)]
    else
        for k in n, n − 1, n − 2, ..., i + 1 do
            a[(j + k) mod length(a)] ←
a[(j + k − 1) mod length(a)]
    a[(j + i) mod length(a)] ← x
    n ← n + 1
```

remove(i)
    $x \leftarrow a[(j+i) \bmod \text{length}(a)]$
    if $i < n/2$ **then**
        for k **in** $i, i-1, i-2, \ldots, 1$ **do**
            $a[(j+k) \bmod \text{length}(a)] \leftarrow a[(j+k-1) \bmod \text{length}(a)]$
        $j \leftarrow (j+1) \bmod \text{length}(a)$
    else
        for k **in** $i, i+1, i+2, \ldots, n-2$ **do**
            $a[(j+k) \bmod \text{length}(a)] \leftarrow a[(j+k+1) \bmod \text{length}(a)]$
    $n \leftarrow n-1$
    if $\text{length}(a) \geq 3 \cdot n$ **then** resize()

Implementando ArrayDeque

- $\text{get}(i)$ e $\text{set}(i, x)$ em $O(1)$ tempo por operação e
- $\text{add}(i, x)$ and $\text{remove}(i)$ em um tempo $O(1 + \min\{i, n - i\})$ por operação.

2. Listas Baseadas em Arrays

2.5 DualArrayDeque

Construindo uma fila com duas pilhas

Uma DualArrayDeque usa duas ArrayStack para contruir uma lista.

Inicialize()

```
initialize()
    front ← ArrayStack()
    back ← ArrayStack()
```

```
size()
    return front.size() + back.size()
```

A pilha $front$ armazena os elementos cujos índice são $0, \ldots, front.\text{size}() - 1$, mas o faz na ordem reversa. A pilha $back$ contém a lista com os elementos de índices $front.\text{size}(), \ldots, \text{size}() - 1$ na ordem normal. Deste modo, $\text{get}(i)$ e $\text{set}(i, x)$ traduzem nas chamadas apropriadas para $\text{get}(i)$ ou $\text{set}(i, x)$ nas respectivas $front$ ou $back$, com um tempo de $O(1)$ por operação.

```
get(i)
    if i < front.size() then
        return front.get(front.size() − i − 1)
    else
        return back.get(i − front.size())

set(i, x)
    if i < front.size() then
        return front.set(front.size() − i − 1, x)
    else
        return back.set(i − front.size(), x)
```

Operações *add* e *remove*

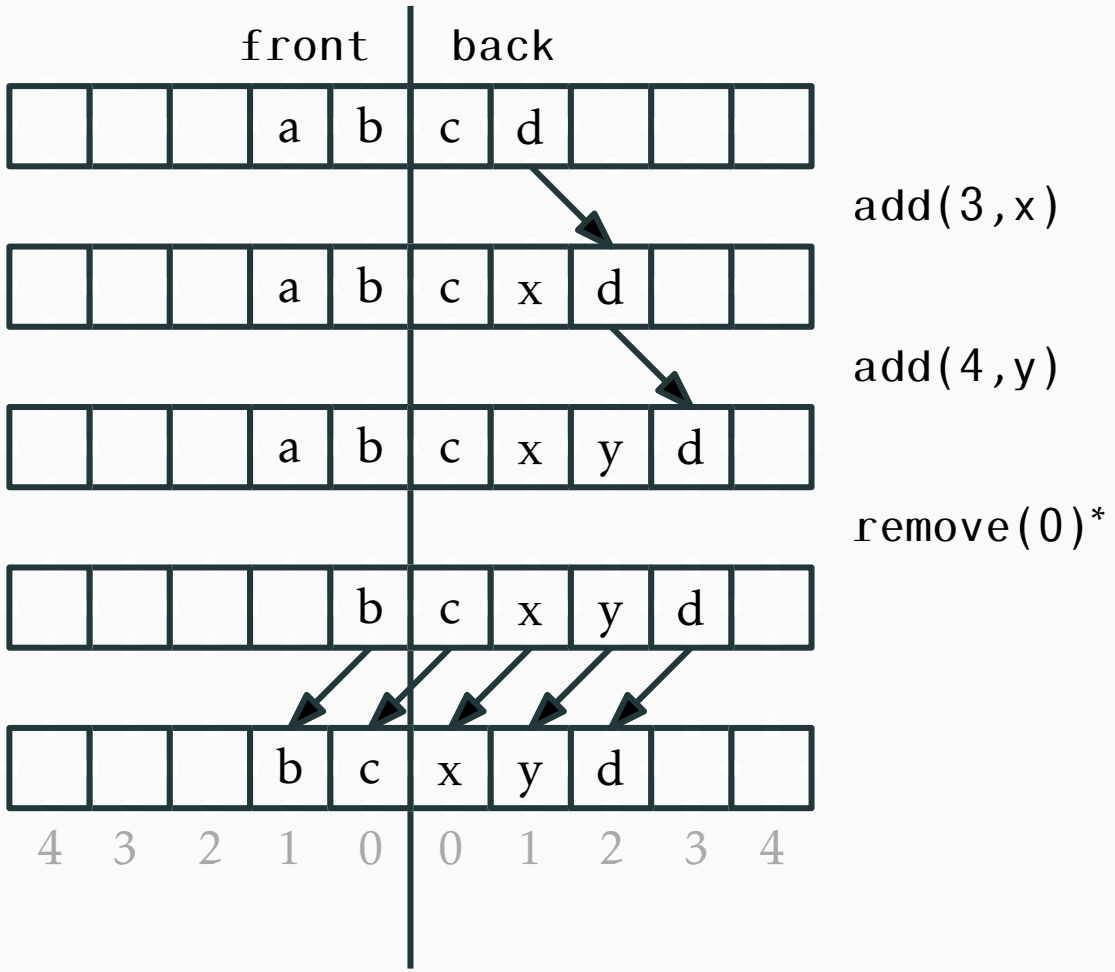


Figura 8: Uma sequência de operações de $\mathrm{add}(i, x)$ and $\mathrm{remove}(i)$ em uma DualArrayDeque.

```
add(i, x)
    if i < front.size() then
        front.add(front.size() − i, x)
    else
        back.add(i − front.size(), x)
    balance()
```

$balance()$ garante que, a menos que $\text{size}() < 2$, $front.\text{size}()$ e $back.\text{size}()$ não diferem em tamanho por um fator maior que 3. Particularmente, $3 \cdot front.\text{size}() \geq back.\text{size}()$ e $3 \cdot back.\text{size}() \geq front.\text{size}()$.

```
remove(i)
    if i < front.size() then
        x ← front.remove(front.size() − i − 1)
    else
        x ← back.remove(i − front.size())
    balance()
    return x
```

balance()

```
balance()
    n ← size()
    mid ← n div 2
    if 3 · front.size() < back.size() or
3 · back.size() < front.size() then
        f ← ArrayStack()
        for i in 0, 1, 2, . . . , mid − 1 do
            f.add(i, get(mid − i − 1))
        b ← ArrayStack()
        for i in 0, 1, 2, . . . , n − mid − 1 do
            b.add(i, get(mid + i))
        front ← f
        back ← b
```

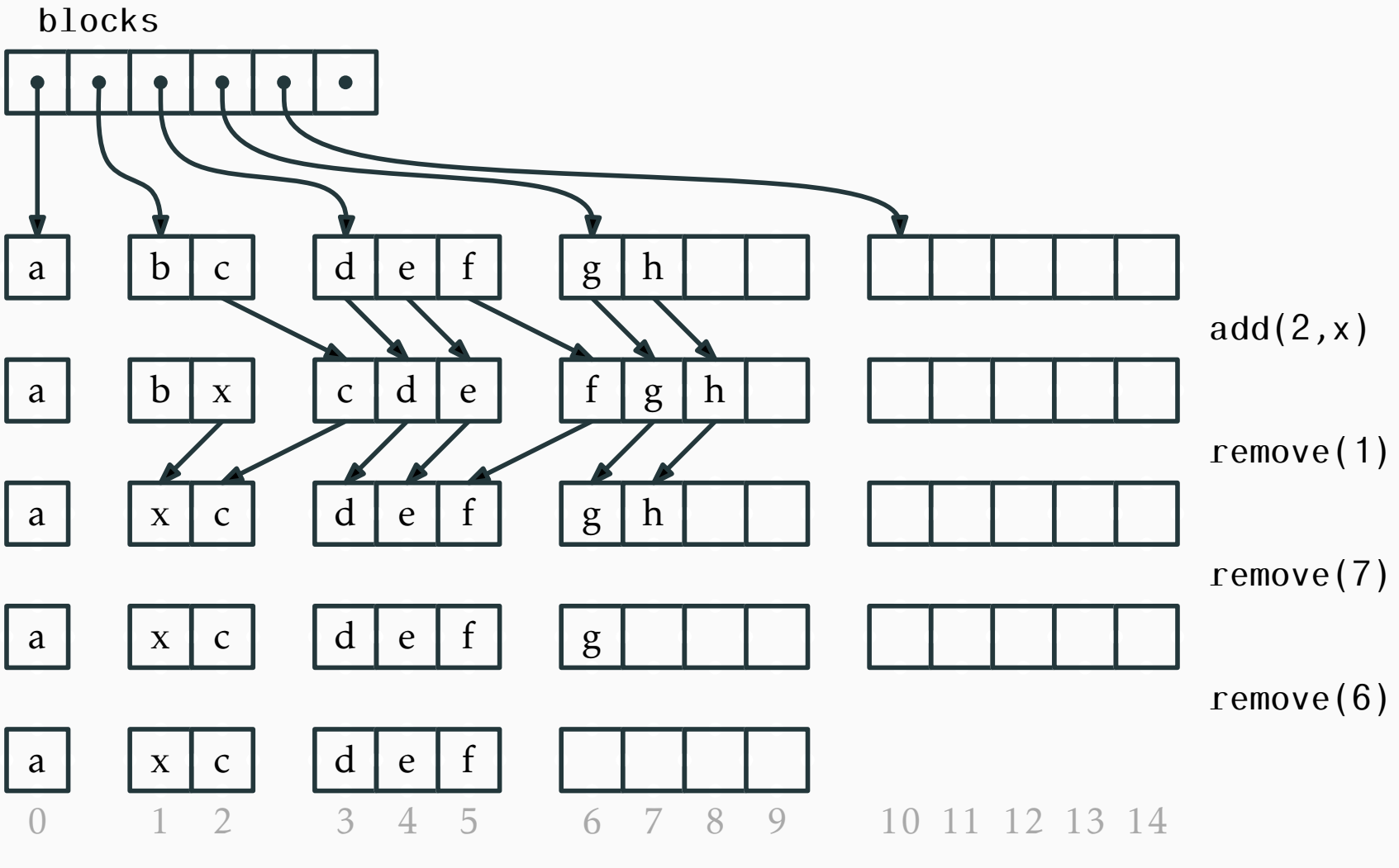
2. Listas Baseadas em Arrays

2.6 RootishArrayStack

RootishArrayStack

Uma RootishArrayStack armazena n elementos usando $O(\sqrt{n})$ arrays. Nestes arrays, pelo menos $O(\sqrt{n})$ posições do array estão vazias em qualquer momento. Todo espaço restante é usado para armazenar dados. Assim, estas estruturas desperdiçam no máximo $O(\sqrt{n})$ do espaço quando estão armazenando elementos n elementos.

RootishArrayStack



initialize()

initialize()

$n \leftarrow 0$

$blocks \leftarrow$ ArrayStack()

O elemento da lista com índice 0 é armazenado no bloco 0, os elementos com índices 1 e 2 são armazenados no bloco 1, os elementos com índices 3, 4, e 5 são armazenados no bloco 2, e assim por diante.

Se o índice i está no bloco b, então o número de elementos nos blocos $0, \ldots, b-1$ é $b(b+1)/2$. Assim, i está armazenado na posição

$$j = i - b(b+1)/2$$

dentro do bloco b.

Cálculo de b

O número de elementos que possuem índices menores que ou iguais a i é $i+1$. Por outro lado, o número de elementos nos blocos $0, \ldots, b$ é $(b+1)(b+2)/2$. Assim, b é o menor inteiro tal que

$$(b+1)(b+2)/2 \geq i+1 \ .$$

Podemos escrever esta equação como

$$b^2 + 3b - 2i \geq 0 \ .$$

A equação quadrática correspondente é $b^2 + 3b - 2i = 0$ que possui duas soluções: $b = (-3 + \sqrt{9 + 8i})/2$ e $b = (-3 - \sqrt{9 + 8i})/2$.

Assim, obtemos a solução $b = (-3 + \sqrt{9 + 8i})/2$. Procuramos simplesmente o menor inteiro b tal que $b \geq (-3 + \sqrt{9 + 8i})/2$. Isto é simplesmente

$$b = \left\lceil (-3 + \sqrt{9 + 8i})/2 \right\rceil \quad .$$

```
i2b(i)
    return int_value(ceil((-3.0 + sqrt(9 + 8 · i))/2.0)
```

$\text{get}(i)$

$b \leftarrow \text{i2b}(i)$

$j \leftarrow i - b \cdot (b+1)/2$

return $blocks.\text{get}(b)[j]$

$\text{set}(i, x)$

$b \leftarrow \text{i2b}(i)$

$j \leftarrow i - b \cdot (b+1)/2$

$y \leftarrow blocks.\text{get}(b)[j]$

$blocks.\text{get}(b)[j] \leftarrow x$

return y

$add()$

$\text{add}(i, x)$
    $r \leftarrow blocks.\text{size}()$
    if $r \cdot (r+1)/2 < n+1$ **then** $\text{grow}()$
    $n \leftarrow n+1$
    for j **in** $n-1, n-2, n-3, \ldots, i+1$ **do**
        $\text{set}(j, \text{get}(j-1))$
    $\text{set}(i, x)$

```
grow()
    blocks.append(new_array(blocks.size() + 1))
```

```
remove(i)
    x ← get(i)
    for j in i, i + 1, i + 2, . . . , n − 2 do
        set(j, get(j + 1))
    n ← n − 1
    r ← blocks.size()
    if (r − 2) · (r − 1)/2 ≥ n then shrink()
    return x
```

```
shrink()
    r ← blocks.size()
    while r > 0 and (r − 2) · (r − 1)/2 ≥ n do
        blocks.remove(blocks.size() − 1)
        r ← r − 1
```

2. Listas Baseadas em Arrays

2.7 Exercícios

2.7 Exercícios

1. O método de Lista $add\_all(i, c)$ insere todos os elementos de uma coleção c na posição i da lista. (O método $add(i, x)$ é um caso especial onde $c = x$.) Explique por que, para as estruturas desta aula, não seria eficiente implementar $add\_all(i, c)$ com chamadas repetidas a add(i,x). Projete e implemente uma forma mais eficiente de implementação.

FIM