SINTRA
TECNOLOGIAS DIGITAIS
ECONOMIA E SOCIEDADE

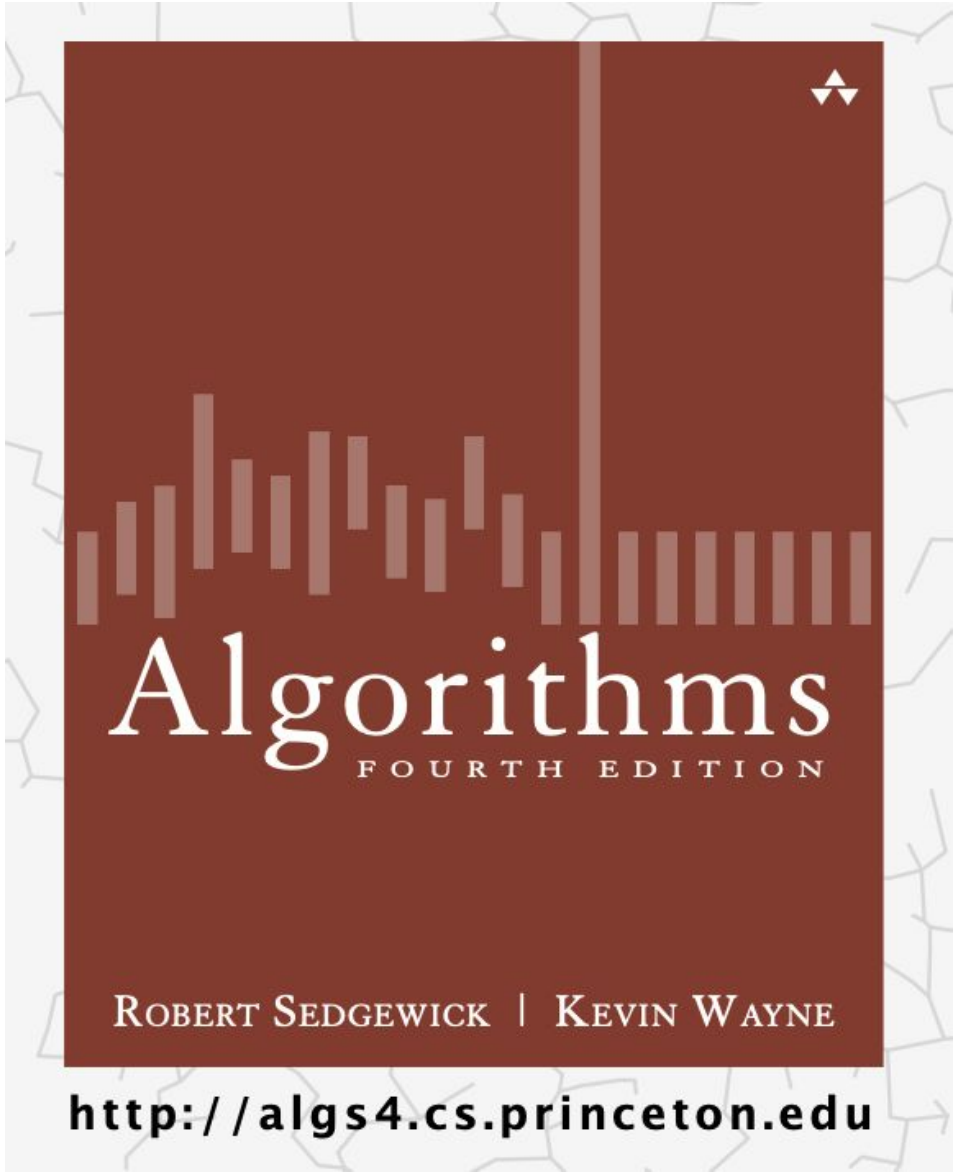
2º Semestre

2023/2024

Aula 4 (7 e 8)

Algoritmia e Estruturas de Dados

Joana Martinho Costa, João Matos, João Monge, Miguel Sales Dias, Paulo Vieira


Algorithms
FOURTH EDITION
ROBERT SEDGEWICK | KEVIN WAYNE
http://algs4.cs.princeton.edu

No episódio anterior...

- ...trabalhámos a análise de algoritmos

No episódio anterior...

```
1  count = 0
2  for i in range(N):
3      if a[i] == 0:
4          count += 1
```

N

1 Acessos a array

No episódio anterior...

```
1  count = 0
2  for i in range(N):
3      for j in range(i+1, N):
4          if a[i] + a[j] == 0:
5              count += 1
```

N

N-1

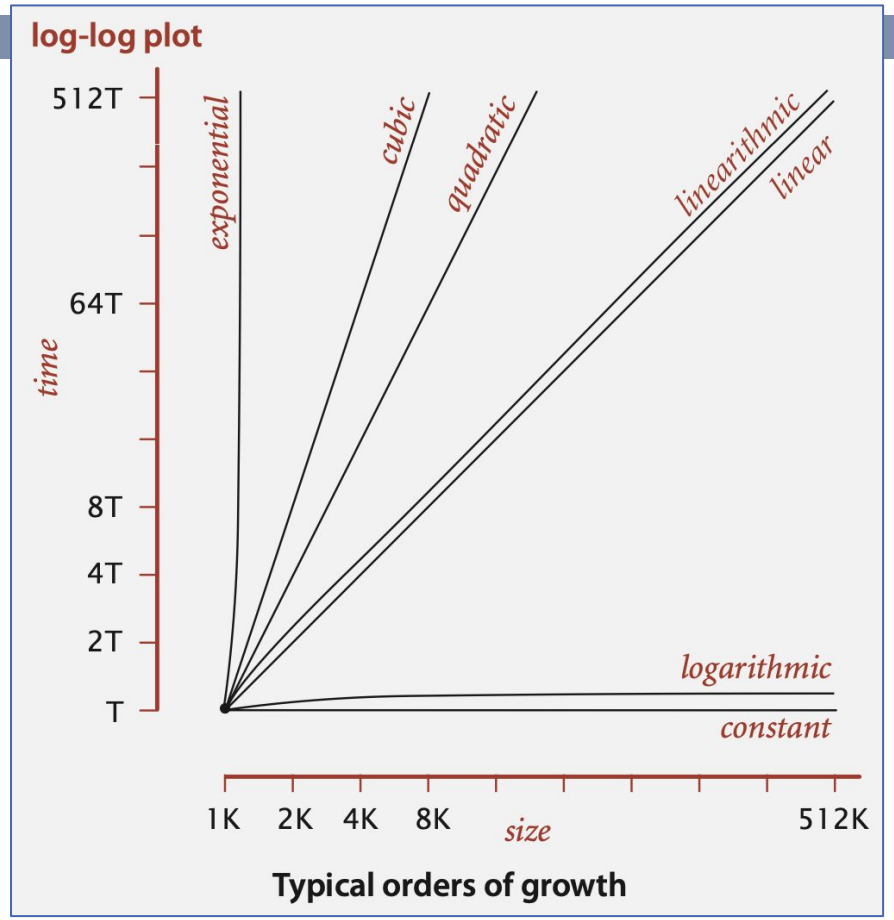
2 Acessos a array

No episódio anterior...

```
1  count = 0
2  for i in range(N):
3      for j in range(i+1, N):
4          for k in range(j+1, N):
5              if a[i] + a[j] + a[k] == 0:
6                  count += 1
```

3 Acessos a array

No episódio anterior...



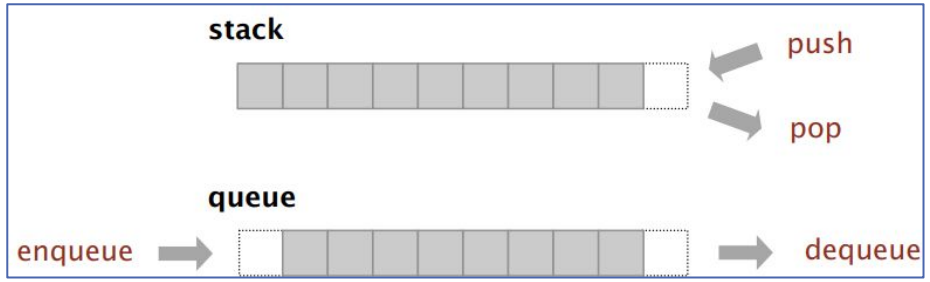
order of growth	name	typical code framework	description	example
1	constant	`a = b + c;`	statement	add two numbers
log N	logarithmic	`while (N > 1) { N = N / 2; ... }`	divide in half	binary search
N	linear	`for (int i = 0; i < N; i++) { ... }`	loop	find the maximum
N log N	linearithmic	[see mergesort lecture]	divide and conquer	mergesort
N^2	quadratic	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }`	double loop	check all pairs
N^3	cubic	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }`	triple loop	check all triples
2^N	exponential	[see combinatorial search lecture]	exhaustive search	check all subsets

Stacks e Queues

- Stacks
- Resizing arrays
- Queues
- Aplicações

Stacks e Queues

- Tipos de dados fundamentais
  - Operações: insert, remove, iterate, test (para testar se está vazio)



- Stack: Examinar o item que foi adicionado mais recentemente
  - Estrutura LIFO: Last In First Out
- Queue: Examinar o item que foi adicionado há mais tempo
  - Estrutura FIFO: First In First Out

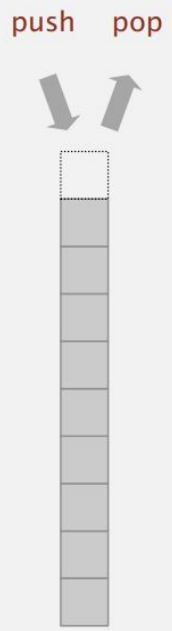
Generalizamos

- Criamos estruturas genéricas que possam ser reutilizadas em diversos contextos

Stack API (Application Programming Interface)

- Contexto: Stack de Strings

public class StackOfStrings	
StackOfStrings()	*create an empty stack*
void push(String item)	*insert a new string onto stack*
String pop()	*remove and return the string most recently added*
boolean isEmpty()	*is the stack empty?*
int size()	*number of strings on the stack*



Stack – Cliente de Teste

```
def main():
    for line in sys.stdin:
        stack = StackOfStrings()
        for item in line.split():
            if item == "-":
                print(stack.pop(), " ", end='')
            else:
                stack.push(item)
```

1. Lê uma String
2. Se a String for igual a “-”, executa pop da stack e imprime
3. Se não, faz push para a stack

```
% more tobe.txt
to be or not to - be - - that - - - is

% java StackOfStrings < tobe.txt
to be not that or be
```

Como construir os métodos da Stack?

- Através de uma estrutura Linked-List
- Através de um array

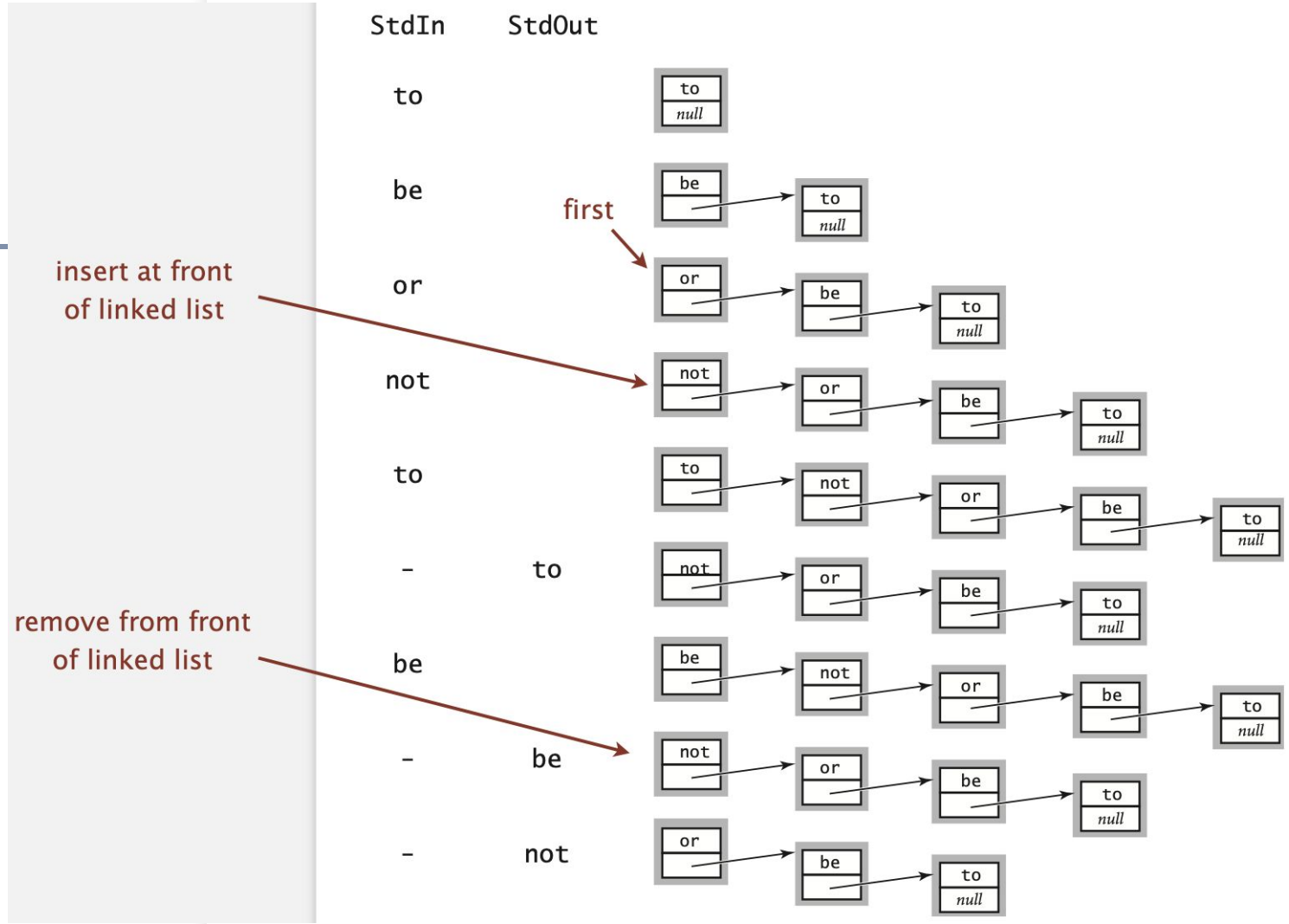
Versão 1: Stack com Linked-List

Joana Martinho Costa, João Matos, João Monge, Miguel Sales Dias, Paulo Vieira

O que é uma linked-list?

- Uma lista ligada (ou encadeada) é uma estrutura de dados que liga vários nós.
- Cada nó tem um apontador para o nó seguinte.

- O exemplo da figura é uma lista de Strings, onde cada nó é composto por uma String e um apontador para o nó seguinte



Stack com Linked-List

1º: Criar uma classe “Nó” para representar os nós da lista

```
class Node:
    def __init__(self, item, next_node):
        self.item = item
        self.next = next_node
```

Stack com Linked-List

2º: Preparar a funcionalidade pop (retirar da stack):

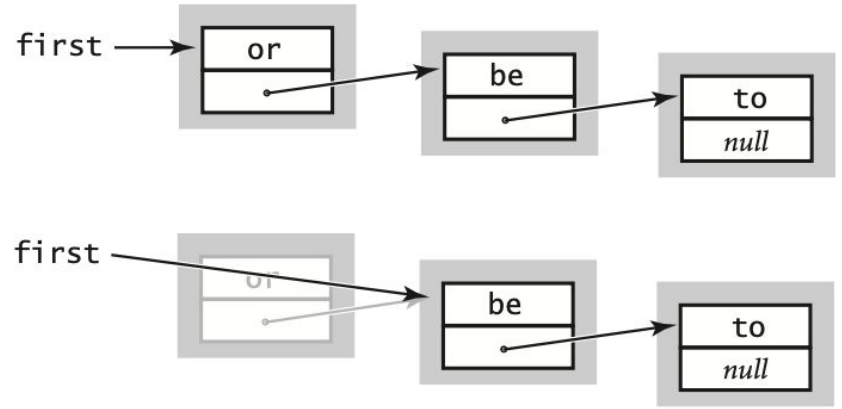
para retirar o nó da stack, troca-se o apontador do primeiro nó (Nó first) para o segundo nó

save item to return

```
String item = first.item;
```

delete first node

```
first = first.next;
```



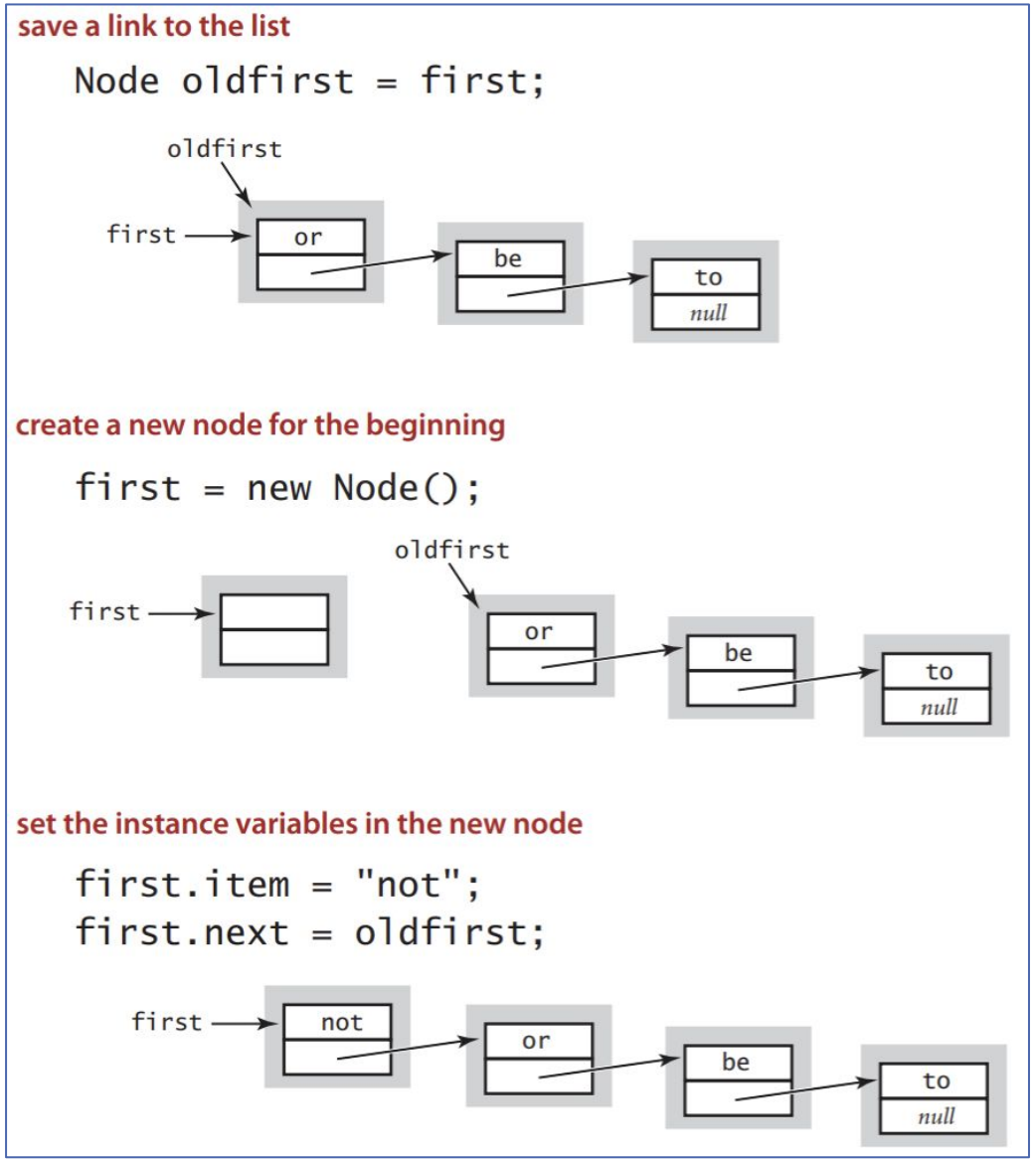
return saved item

```
return item;
```

Stack com Linked-List

2º: Preparar a funcionalidade push (colocar da stack):

Para adicionar um nó na stack, guarda-se a ligação ao nó first, cria-se um novo nó e liga-se o novo nó ao antigo nó first



Stack com Linked-List

- Versão completa

```
class Node:
    def __init__(self, item, next_node):
        self.item = item
        self.next = next_node
```

```
class LinkedStackOfStrings:
    def __init__(self):
        self.first = None

    def is_empty(self):
        return self.first is None

    def push(self, item):
        oldfirst = self.first
        self.first = Node(item, oldfirst)

    def pop(self):
        item = self.first.item
        self.first = self.first.next
        return item
```

No mínimo, incrementa-se o i+=1 (N), no máximo incrementa-se o i+=1 e o count+=1 (2N)

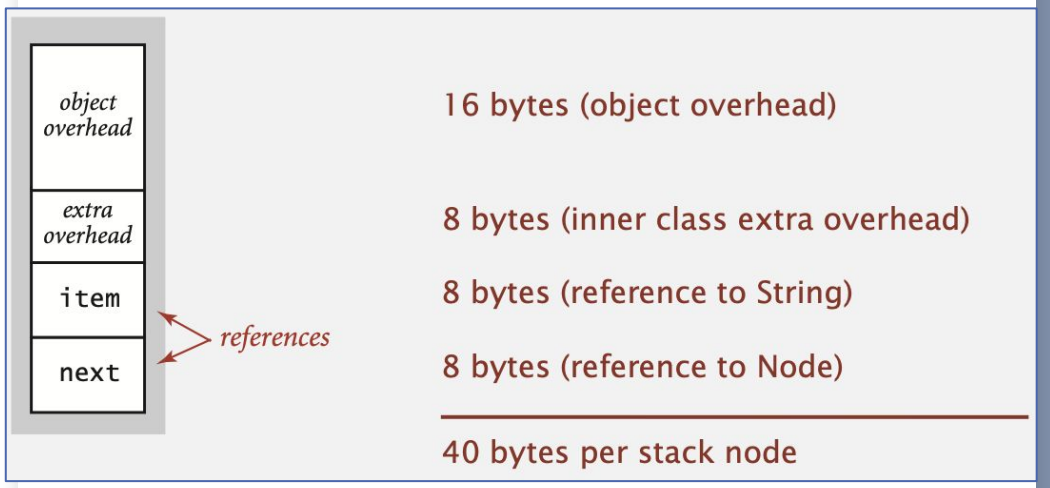
Tarefa

1. Criar a classe LinkedStackofInts com nós do tipo inteiro
2. Cria uma função main com:
    a) Dois push (inteiros 34 e 67)
    b) Um pop
        a) Imprime o valor que saiu da stack

Análise da performance da Stack com Linked-List

Todas as operações (criação, verificar se está vazio, push e pop) têm um custo de execução constante no pior caso

Em relação ao custo de memória, uma stack com N itens tem um custo de 40N bytes (excluindo o custo do próprio tipo de dados de cada nó, como por exemplo, o custo associado a String)



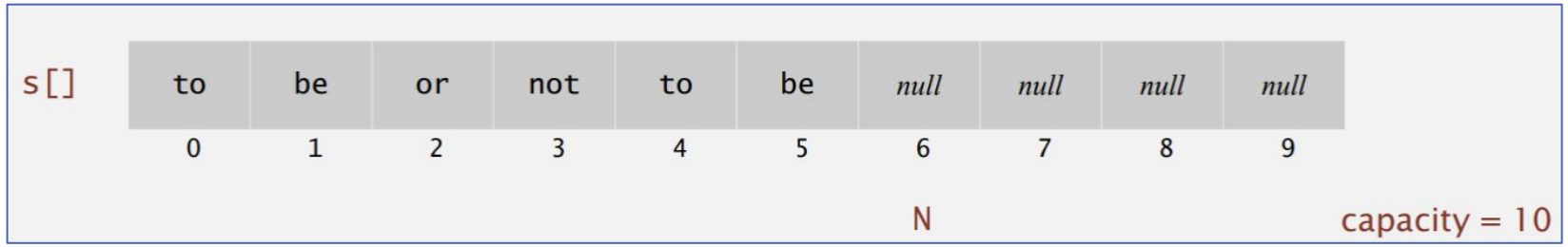

```
class Node:
    def __init__(self, item, next_node):
        self.item = item
        self.next = next_node
```

Versão 2: Stack com Array

Joana Martinho Costa, João Matos, João Monge, Miguel Sales Dias, Paulo Vieira

Stack com Array

- Usar um array s[] para guardar N itens numa stack
- Método push: adiciona itens ao s[N]
- Método pop: retira itens do s[N-1]



- Se o número de itens a adicionar for maior que a capacidade do array: stack overflow

Implementação Stack com Array

```
class FixedCapacityStackOfStrings:
    def __init__(self, capacity):
        self.N = 0
        self.s = []
        for i in range(capacity):
            self.s.append("")

    def is_empty(self):
        return self.N == 0

    def push(self, item):
        self.s[self.N] = item
        self.N += 1

    def pop(self):
        self.N -= 1
        return self.s[self.N]
```

```
def main():
    for line in sys.stdin:
        stack = FixedCapacityStackOfStrings(10)
        for item in line.split():
            if item == "-":
                print(stack.pop(), " ", end='')
            else:
                stack.push(item)
```

Stack com Array

- Características
  - Lança uma exceção se chamar o pop() numa stack vazia
  - Se a stack encher, é necessário fazer o resizing ao array
  - Valores null podem ser inseridos
  - Existe um problema de loitering que consiste em manter em memória valores que já não são usados

Stack com Array

```
def pop(self):
        self.N -= 1
        return self.s[self.N]
```

Loitering

```
def pop(self):
        self.N -= 1
        item = self.s[self.N]
        self.s[self.N] = None
        return item
```

this version avoids "loitering":
garbage collector can reclaim memory
only if no outstanding references

Resizing

Joana Martinho Costa, João Matos, João Monge, Miguel Sales Dias, Paulo Vieira

Resizing

- Como aumentar o array?

- Tentativa 1
  - Push(): aumenta o tamanho do array s[] para +1
  - Pop(): diminui o tamanho do array s[] para -1

Tem um custo demasiado elevado ☹
Em cada push/pop seria necessário copiar todos os itens para um novo array

Inserir N itens tem um custo proporcional a 1+2+...+N – aproximadamente N^2/2

Resizing

Tentativa 2: Se o array encher, duplica-se o tamanho e copiam-se todos os itens para o novo array

Inserir N itens tem um custo proporcional a N (e não N^2)

```
public ResizingArrayStackOfStrings()
{  s = new String[1];  }

public void push(String item)
{
   if (N == s.length) resize(2 * s.length);
   s[N++] = item;
}

private void resize(int capacity)
{
   String[] copy = new String[capacity];
   for (int i = 0; i < N; i++)
      copy[i] = s[i];
   s = copy;
}
```

Resizing

- Como diminuir o array?

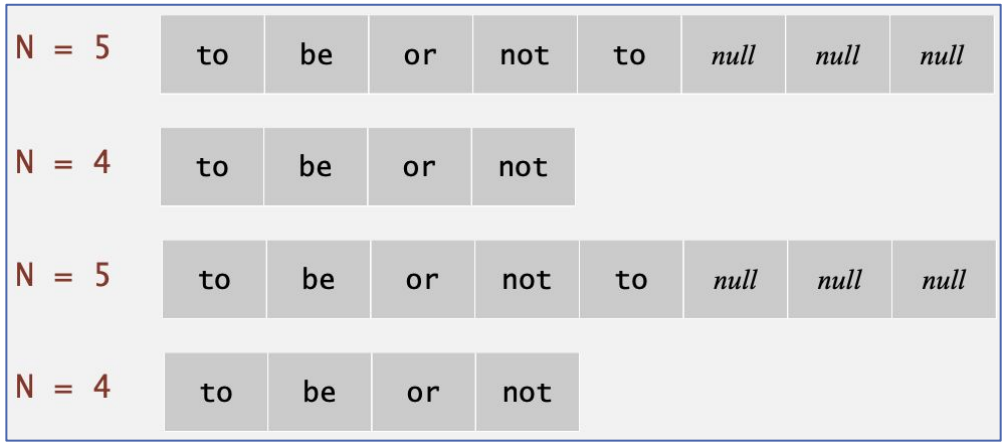
- Tentativa 1
  - Push(): duplica o array s[] quando está preenchido
  - Pop(): reduz o array s[] para metade quando está metade preenchido

Tem um custo demasiado elevado ☹️
Em cada push/pop seria necessário copiar todos os itens para um novo array

Inserir N itens tem um custo proporcional a 1+2+...+N – aproximadamente N^2/2

Resizing

- Custo elevado!
  - Se considerarmos uma sequência contínua push-pop-push-pop... Quando o array está cheio, cada operação tem um custo proporcional a N



Resizing

- Como diminuir o array?

- Solução mais eficiente
  - Push(): duplica o array s[] quando está preenchido
  - Pop(): reduz o array s[] para metade quando está metade **¼ preenchido**

```
public String pop()
{
   String item = s[--N];
   s[N] = null;
   if (N > 0 && N == s.length/4) resize(s.length/2);
   return item;
}
```

Stack com Linked-List ou com Resizing-Array?

- Implementação com linked-list
  - Todas as operações têm um custo constante no pior caso
  - Utiliza tempo e espaço extra para trabalhar com as ligações entre os nós

- Implementação com resizing array
  - Todas as operações têm um custo amortizado N
  - Menos espaço desperdiçado

Implementação de Stacks

- Botão “Retroceder” no web browser
- Botão “Anular” no processador de texto

Tarefa

1. Criar a classe ResizingArrayStackOfInt
   Cria uma função main com:
   a) Quatro push (inteiros 34, 42, 89 e 67)
   b) Dois pop
   c) Imprime o tamanho do vetor a cada operação push ou pop

Tarefa

- Stack Fixa

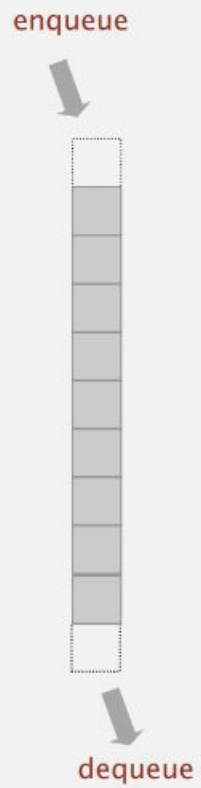
iscte
SINTRA
TECNOLOGIAS DIGITAIS
ECONOMIA E SOCIEDADE

Filas

Joana Martinho Costa, João Matos, João Monge, Miguel Sales Dias, Paulo Vieira

Queues

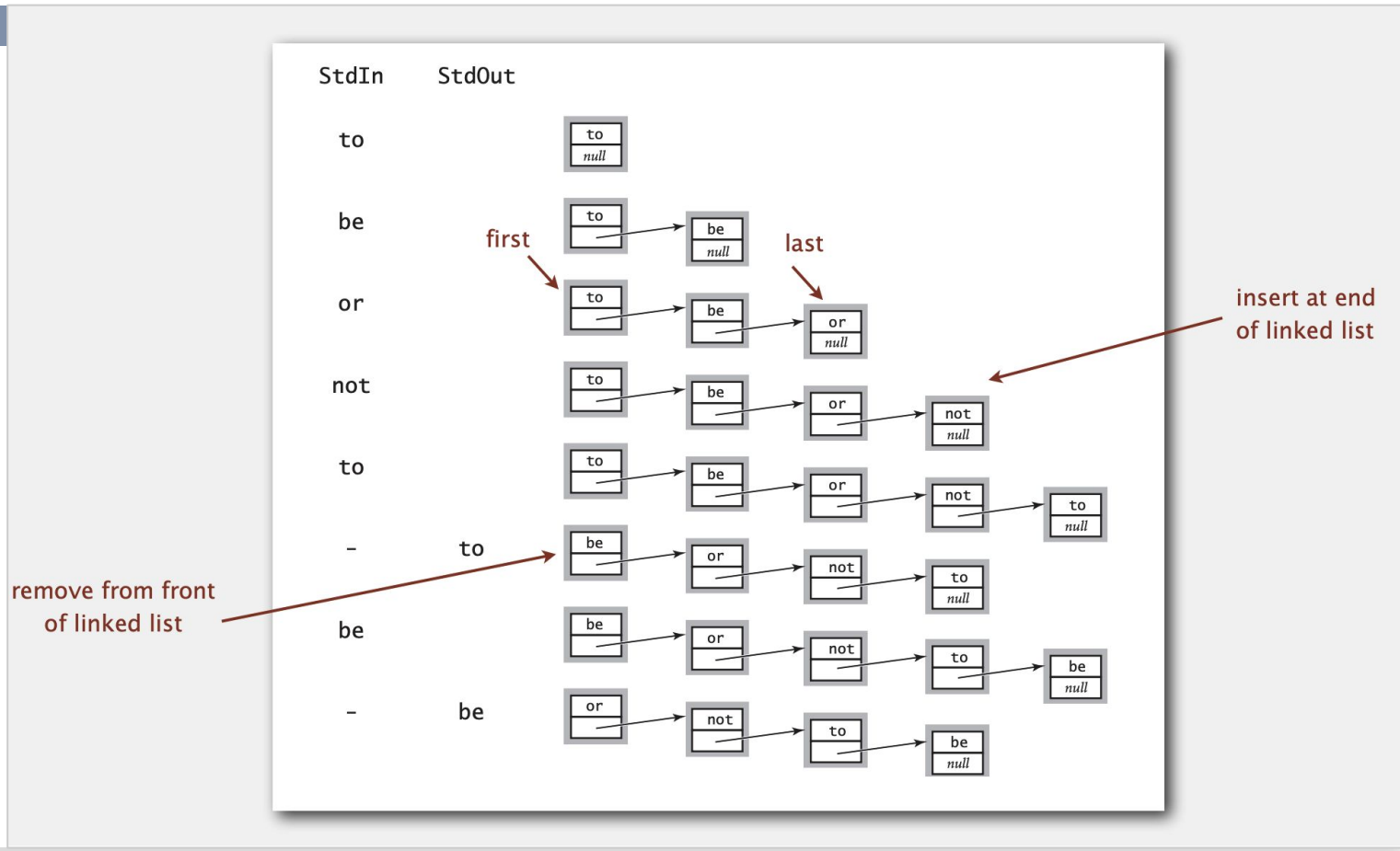
public class QueueOfStrings	
QueueOfStrings()	*create an empty queue*
void enqueue(String item)	*insert a new string onto queue*
String dequeue()	*remove and return the string least recently added*
boolean isEmpty()	*is the queue empty?*
int size()	*number of strings on the queue*



Queues

Queue com Linked-list

- Ao contrário do Stacks, temos de manter referência para dois elementos: first e last



Queue com Nodes

- Versão Completa

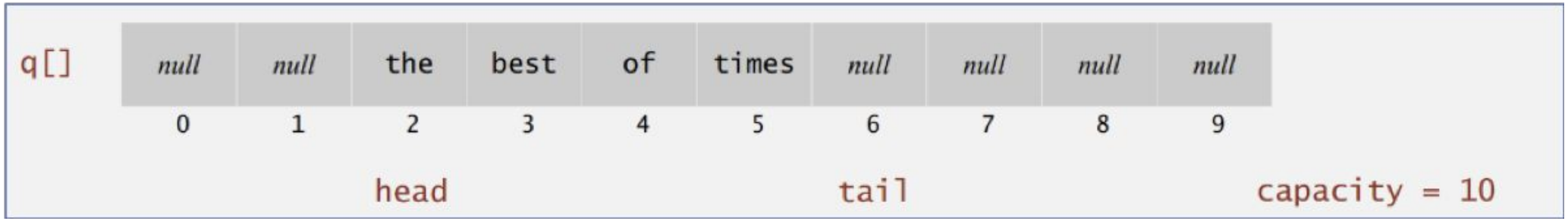
```
class QueueOfStrings:
    def __init__(self):
        self.first = None
        self.last = None

    def enqueue(self, item):
        oldlast = self.last
        self.last = Node(item, None)
        oldlast.next = self.last

    def dequeue(self):
        item = self.first.item
        self.first = self.first.next
        return item

    def is_empty(self):
        return self.first is None
```

Queue com Resizing



- Utiliza o array [] para guardar os items na fila
- enqueue(): adiciona o novo elemento a q[tail]
- dequeue(): remove o item da q[head].
- Atualiza os apontadores head e tail

Tarefa

- Implementar a estrutura de dados Queue com resizing

Tarefa

- Tarefa Semanal 4