Algoritmos em Python extraídos do PDF

Pilhas usando listas:

```python
class StackList:
    def __init__(self):
        self.items = []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop() if self.items else None
    def peek(self):
        return self.items[-1] if self.items else None
    def isempty(self):
        return not self.items
# Uso:
stacklist = StackList()
stacklist.push(1)
stacklist.push(2)
stacklist.push(3)
print("Pilha usando lista:", stacklist.items)
print("Pop:", stacklist.pop())
print("Peek:", stacklist.peek())
```

Pilhas usando deque:

python

```python
from collections import deque
class StackDeque:
    def __init__(self):
        self.items = deque()
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop() if self.items else None
    def peek(self):
        return self.items[-1] if self.items else None
    def isempty(self):
        return not self.items
# Uso:
stackdeque = StackDeque()
stackdeque.push(1)
stackdeque.push(2)
stackdeque.push(3)
print("Pilha usando deque:", stackdeque.items)
print("Pop:", stackdeque.pop())
print("Peek:", stackdeque.peek())
```

Filas usando listas:

python

```python
class QueueList:
    def __init__(self):
        self.items = []
    def enqueue(self, item):
        self.items.append(item)
    def dequeue(self):
        return self.items.pop(0) if self.items else None
    def front(self):
        return self.items[0] if self.items else None
    def isempty(self):
        return not self.items
# Uso:
queuelist = QueueList()
queuelist.enqueue(1)
queuelist.enqueue(2)
queuelist.enqueue(3)
print("Fila usando lista:", queuelist.items)
print("Dequeue:", queuelist.dequeue())
print("Front:", queuelist.front())
```

Filas usando deque:

python

```python
from collections import deque
class QueueDeque:
    def __init__(self):
        self.items = deque()
    def enqueue(self, item):
        self.items.append(item)
    def dequeue(self):
        return self.items.popleft() if self.items else None
    def front(self):
        return self.items[0] if self.items else None
    def isempty(self):
        return not self.items
# Uso:
queuedeque = QueueDeque()
queuedeque.enqueue(1)
queuedeque.enqueue(2)
queuedeque.enqueue(3)
print("Fila usando deque:", queuedeque.items)
print("Dequeue:", queuedeque.dequeue())
print("Front:", queuedeque.front())
```

Exemplos com tratamento de overflow/underflow:

python

```python
class Stack:
    def __init__(self, capacity):
        self.items = []
        self.capacity = capacity
    def push(self, item):
        if len(self.items) == self.capacity:
            raise OverflowError("Stack overflow")
        self.items.append(item)
    def pop(self):
        if not self.items:
            raise IndexError("Stack underflow")
        return self.items.pop()
    def peek(self):
        return self.items[-1] if self.items else None
    def isempty(self):
        return len(self.items) == 0
    def isfull(self):
        return len(self.items) == self.capacity
# Uso:
stack = Stack(3)
stack.push(10)
stack.push(20)
stack.push(30)
print("Pilha após pushes:", stack.items)
try:
    stack.push(40)
except OverflowError as e:
    print(e)
print("Elemento removido (pop):", stack.pop())
print("Pilha após pop:", stack.items)
try:
    stack.pop()
    stack.pop()
    stack.pop()
except IndexError as e:
    print(e)
```

Fila circular exemplo:

python

```python
class CircularQueue:
    def __init__(self, capacity):
        self.queue = [None] * capacity
        self.capacity = capacity
        self.front = self.rear = -1
    def isfull(self):
        return (self.rear + 1) % self.capacity == self.front
    def isempty(self):
        return self.front == -1
    def enqueue(self, item):
        if self.isfull():
            raise OverflowError("Queue overflow")
        if self.front == -1:
            self.front = 0
        self.rear = (self.rear + 1) % self.capacity
        self.queue[self.rear] = item
    def dequeue(self):
        if self.isempty():
            raise IndexError("Queue underflow")
        item = self.queue[self.front]
        if self.front == self.rear:
            self.front = self.rear = -1
        else:
            self.front = (self.front + 1) % self.capacity
        return item
    def frontelement(self):
        return None if self.isempty() else self.queue[self.front]
# Uso:
cq = CircularQueue(3)
cq.enqueue(10)
cq.enqueue(20)
cq.enqueue(30)
print("Fila circular após enqueues:", cq.queue)
try:
    cq.enqueue(40)
except OverflowError as e:
    print(e)
print("Elemento removido (dequeue):", cq.dequeue())
print("Fila circular após dequeue:", cq.queue)
print("Elemento na frente (front):", cq.frontelement())
```

Simulação de fila de atendimento ao cliente:

python

```python
class CustomerServiceQueue:
    def __init__(self):
        self.queue = []
    def arrive(self, customer):
        self.queue.append(customer)
    def serve(self):
        if self.isempty():
            return None
        return self.queue.pop(0)
    def isempty(self):
        return len(self.queue) == 0
    def size(self):
        return len(self.queue)
# Uso:
csq = CustomerServiceQueue()
csq.arrive("Cliente 1")
csq.arrive("Cliente 2")
csq.arrive("Cliente 3")
print("Fila após chegada:", csq.queue)
print("Atendendo:", csq.serve())
print("Fila após atender:", csq.queue)
```