Educação Profissional Paulista

Técnico em **Ciência de Dados**

Secretaria da Educação

SÃO PAULO
GOVERNO DO ESTADO

Lógica de programação e algoritmos

Pilhas e filas

Aula 3

Código da aula: [DADOS]ANO1C3B4S28A3

Mapa da Unidade 1 Componente 4

Lógica de programação e algoritmos

semana 21

Recursividade

semana 23

Busca e ordenação

semana 25

Prática de busca e ordenação

semana 27

Algoritmos de contagem e acumulação

semana 28

Você está aqui!

Pilhas e filas

Lógica de programação e algoritmos

Mapa da Unidade 1 Componente 4

Você está aqui!

28

Pilhas e filas

Aula 3

Código da aula:
[DADOS]ANO1C3B4S28A3

Objetivos da aula

- Introduzir os fundamentos das estruturas de dados pilhas e filas.

Recursos didáticos

- Recurso audiovisual para exibição de vídeos e imagens;
- Acesso ao laboratório de informática e/ou internet.

Duração da aula

50 minutos.

Competências técnicas

- Identificar e resolver problemas relacionados a dados e análises;
- Compreender e dominar técnicas de manipulação de dados;

Competências socioemocionais

- Adaptar-se a novas tecnologias, técnicas e tendências sem perder o foco, as metas e os objetivos da organização;
- Colaborar efetivamente com outros profissionais, como cientistas de dados e engenheiros de dados; trabalhar em equipes multifuncionais colaborando com colegas, gestores e clientes.

Construindo o **conceito**

Pilhas e filas: problemas e soluções comuns

Problemas comuns com pilhas

- Overflow e underflow;
- Balanceamento de parênteses;
- Conversão de notação infixa para pós-fixa.

Problemas comuns com pilhas

- Implementação de filas circulares;
- Simulação de atendimento;
- Gerenciamento de processos em sistemas operacionais.

Soluções e implementações

- Implementação de pilha com tratamento de overflow e underflow;
- Implementação de fila circular.

Exemplos de uso em simulação e gerenciamento de processos

Construindo o **conceito**

Problemas comuns com pilhas

- **Overflow e underflow:** ocorrem quando tentamos adicionar um elemento a uma pilha cheia (overflow) ou remover um elemento de uma pilha vazia (underflow).

- **Balanceamento de parênteses:** verificação de expressões matemáticas ou de código para assegurar que todos os parênteses estão corretamente balanceados.

- **Conversão de notação infixa para pós-fixa:** algoritmos para converter expressões matemáticas da notação infixa (convencional) para pós-fixa (notação polonesa reversa).

Construindo o **conceito**

Problemas comuns com filas

- **Implementação de filas circulares:** resolver a limitação de espaço e melhorar a eficiência das operações de fila.
- **Simulação de atendimento:** modelagem de sistemas de atendimento ao cliente para otimização de recursos.
- **Gerenciamento de processos em sistemas operacionais:** utilização de filas para gerenciar processos em execução, prontos e bloqueados.

Construindo o **conceito**

Soluções e implementações

- **Implementação de pilha com tratamento de overflow e underflow:** abordagens para evitar erros de overflow e underflow em implementações de pilhas.
- **Implementação de fila circular:** estruturas e algoritmos para implementar filas circulares que utilizam espaço de forma eficiente.
- **Exemplos de uso em simulação e gerenciamento de processos:** aplicações práticas de pilhas e filas em simulação de atendimento ao cliente e gerenciamento de processos.

Construindo o **conceito**

Exemplos práticos

Vamos praticar!
Implemente este código utilizando Python.

Exemplo 1:
Implementação de pilha com tratamento de overflow e underflow:

```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):
        if not self.items:
            return None
        return self.items[-1]

    def is_empty(self):
        return len(self.items) == 0

    def is_full(self):
        return len(self.items) == self.capacity
```

```python
# Exemplo de uso
stack = Stack(3)
stack.push(10)
stack.push(20)
stack.push(30)
print("Pilha após pushes:", stack.items)
print("Tentativa de overflow:")
try:
    stack.push(40)
except OverflowError as e:
    print(e)
print("Elemento removido (pop):", stack.pop())
print("Pilha após pop:", stack.items)
print("Tentativa de underflow:")
try:
    stack.pop()
    stack.pop()
    stack.pop()
except IndexError as e:
    print(e)
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Construindo
o **conceito**

Exemplos práticos

Implemente este código utilizando Python.

Exemplo 1: implementação de pilha com tratamento de overflow e underflow:

```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):
        if not self.items:
            return None
        return self.items[-1]

    def is_empty(self):
        return len(self.items) == 0

    def is_full(self):
        return len(self.items) == self.capacity
```

```python
# Exemplo de uso
stack = Stack(3)
stack.push(10)
stack.push(20)
stack.push(30)
print("Pilha após pushes:", stack.items)
print("Tentativa de overflow:")
try:
    stack.push(40)
except OverflowError as e:
    print(e)
print("Elemento removido (pop):", stack.pop())
print("Pilha após pop:", stack.items)
print("Tentativa de underflow:")
try:
    stack.pop()
    stack.pop()
    stack.pop()
except IndexError as e:
    print(e)
```

Construindo
o **conceito**

Exemplos práticos

Implemente este código utilizando Python.

Exemplo 2: implementação de fila circular:

```python
class CircularQueue:
    def __init__(self, capacity):
        self.queue = [None] * capacity
        self.capacity = capacity
        self.front = self.rear = -1

    def is_full(self):
        return (self.rear + 1) % self.capacity == self.front

    def is_empty(self):
        return self.front == -1

    def enqueue(self, item):
        if self.is_full():
            raise OverflowError("Queue overflow")
        if self.front == -1:
            self.front = 0
        self.rear = (self.rear + 1) % self.capacity
        self.queue[self.rear] = item
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Construindo
o **conceito**

Exemplos práticos

Implemente este código utilizando Python.

Exemplo 2: implementação de fila circular:

```
    def dequeue(self):
        if self.is_empty():
            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 front_element(self):
        if self.is_empty():
            return None
        return self.queue[self.front]
```

Construindo
o **conceito**

Exemplos práticos

Implemente este código utilizando Python.

Exemplo 2: implementação de fila circular:

```
# Exemplo de uso
cq = CircularQueue(3)
cq.enqueue(10)
cq.enqueue(20)
cq.enqueue(30)
print("Fila circular após enqueues:", cq.queue)
print("Tentativa de overflow:")
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.front_element())
```

Colocando em **prática**

Exercício: simulação de atendimento ao cliente

Trabalhem em duplas para implementar a classe CustomerServiceQueue usando listas em Python.

A classe deve permitir a chegada de clientes na fila e a simulação do atendimento a esses clientes.

Após a implementação, executem a sequência de operações fornecida e verifiquem se a fila está sendo gerenciada corretamente.

Compartilhem suas implementações e resultados com a classe.

20 minutos

Duplas

Código de programação

Colocando em **prática**

Exercício: simulação de atendimento ao cliente

20 minutos

Duplas

Código de programação

IMPLEMENTAÇÃO DA CLASSE CUSTOMERSERVICEQUEUE

1. Implemente a classe CustomerServiceQueue com as seguintes operações:

- arrive(customer): Adiciona um cliente ao final da fila.
- serve(): Remove e retorna o cliente do início da fila.
- is_empty(): Verifica se a fila está vazia.
- size(): Retorna o número de clientes na fila.

2. Teste sua implementação com a sequência de operações fornecida.

EXECUÇÃO E VERIFICAÇÃO

- Adicione três clientes à fila: "Cliente 1", "Cliente 2" e "Cliente 3".
- Sirva os clientes um por um e verifique o estado da fila após cada atendimento.
- Verifique o comportamento da fila quando não há mais clientes para atender.

Ser
sempre +

Situação

Imagine que você está trabalhando como desenvolvedor em uma empresa de tecnologia.

Durante um projeto importante, você e seu colega foram designados para trabalhar juntos na implementação de um sistema de filas para um novo serviço de atendimento ao cliente.

No entanto, seu colega constantemente faz comentários negativos sobre suas ideias e trabalho, o que está afetando sua motivação e produtividade.

Como você lida com essa situação para garantir que o projeto seja concluído com sucesso e mantenha um ambiente de trabalho saudável?

Relato fictício elaborado especialmente para o curso.

Ser
sempre +

Ação

Para lidar com essa situação desafiadora, você decide adotar uma abordagem construtiva e buscar soluções que melhorem a comunicação e a colaboração com seu colega. Isso inclui uma reunião de feedback, definição de expectativas claras e estabelecimento de um plano de ação conjunto.

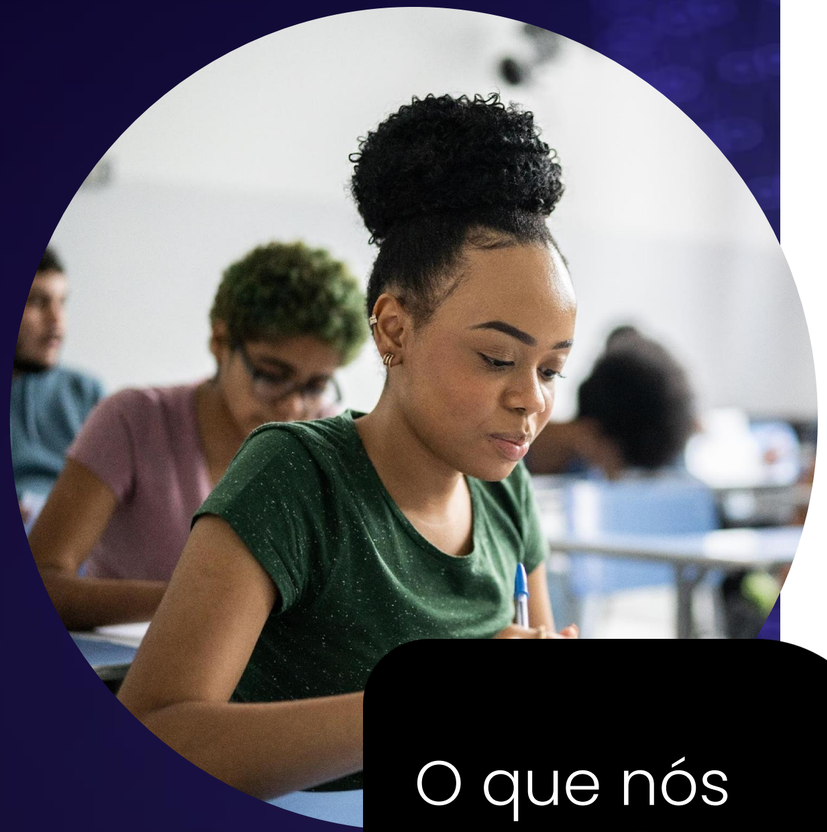
1. Como você abordaria seu colega para discutir os comentários negativos de forma construtiva?

2. Quais estratégias você utilizaria para melhorar a comunicação e a colaboração entre vocês dois?

3. Como você garantiria que o projeto continue avançando de forma produtiva, mesmo enfrentando esse desafio?

Relato fictício elaborado especialmente para o curso.

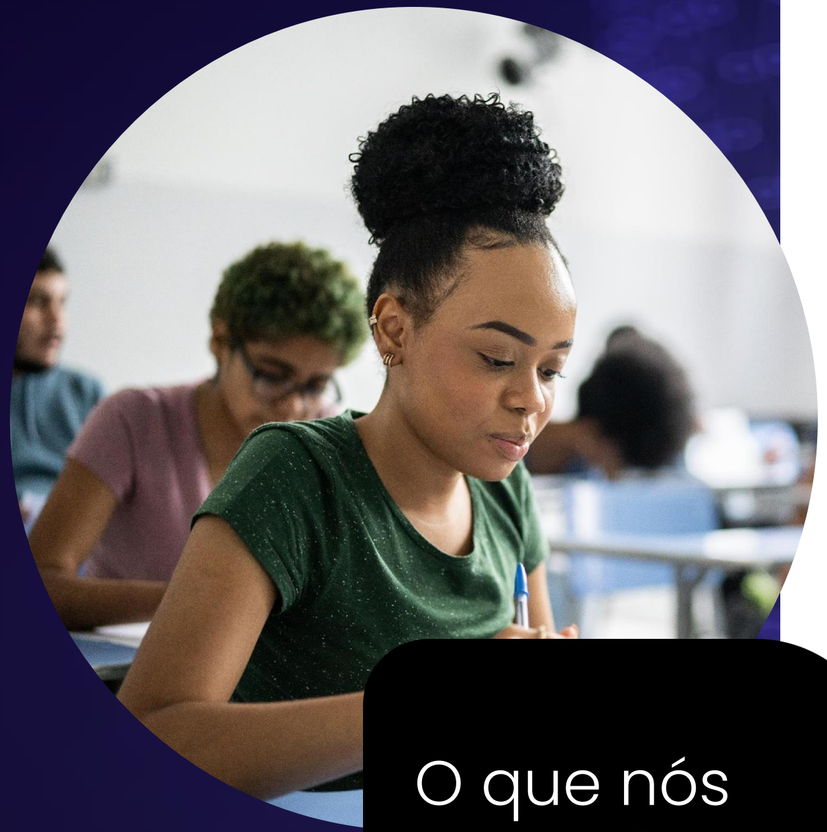
Diálogo



© Getty Images

O que nós **aprendemos hoje?**

Nuvem de palavras



© Getty Images

O que nós **aprendemos hoje?**

Então ficamos assim...

1. Na terceira aula, abordamos problemas comuns com pilhas, como overflow, underflow, balanceamento de parênteses e conversão de notação infixa para pós-fixa. Explicamos como tratar esses problemas em implementações práticas.

2. Discutimos problemas e soluções para filas, incluindo a implementação de filas circulares, simulação de atendimento e gerenciamento de processos em sistemas operacionais, destacando a importância dessas estruturas em aplicações reais.

3. Os alunos implementaram exercícios práticos, como a simulação de atendimento ao cliente, consolidando o entendimento de pilhas e filas e suas aplicações em cenários reais. A seção "Ser Sempre +" focou em competências socioemocionais para lidar com desafios profissionais.

Saiba mais

Será que você realmente precisa saber algoritmos e estruturas de dados para trabalhar com programação?

Acesse o conteúdo e descubra!

ALURA. Algoritmos e Estrutura de Dados. Hipsters #186. 3 fev. 2020. Disponível em: https://cursos.alura.com.br/extra/hipsterstech/algoritmos-e-estrutura-de-dados-hipsters-186-a375. Acesso em: 24 jul. 2024.

Referências da aula

FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de programação**: a construção de algoritmos e estruturas de dados com aplicações em Python. Porto Alegre: Bookman, 2022.

Educação Profissional Paulista

Técnico em **Ciência de Dados**

Secretaria da Educação

SÃO PAULO
GOVERNO DO ESTADO