Conteúdo Extra

Estruturas de Dados: Filas e Pilhas

Material Complementar

Contents

PILHAS E FILAS EM OUTRAS LINGUAGENS

Linguagem de Programação	Fila (Queue)	Pilha (Stack)
C++	std::queue	std::stack
Java/Kotlin	java.util.Queue *(ex: java.util.LinkedList)*	java.util.Stack *(observação: java.util.Stack é uma classe legada. Recomenda-se que você use o java.util.Deque + java.util.LinkedList no lugar)* *(observação 2: apesar disso, em sistemas corporativos encontramos muito o java.util.Stack. Por isso que usamos ele na Atividade Formativa)*
Python	queue.Queue *(ex: queue.Queue() ou queue.PriorityQueue())*	collections.deque
C#	System.Collections.Queue	System.Collections.Stack *(observação: System.Collections.Stack é uma classe legada. Recomenda-se que você use o System.Collections.Generic.Stack no lugar)*
JavaScript/TypeScript	Array *(observação: você pode imitar uma fila usando push() e shift())*	Array *(observação: você pode imitar uma pilha usando push() e pop())*
Ruby	Queue (do módulo thread)	Array *(observação: você pode imitar uma pilha usando push() e pop())*
PHP	SplQueue	SplStack
Swift	Queue	Array *(observação: você pode imitar uma pilha usando append() e popLast())*
Go	container/list (pode ser usada como uma fila)	container/list (pode ser usada como uma pilha)
Rust	std::collections::VecDeque	std::collections::Vec *(observação: você pode usar o método push para adicionar elementos e pop para remover o último)*
Scala	scala.collection.mutable.Queue	scala.collection.mutable.Stack

Lua	Table *(observação: você pode usar o método table.insert para adicionar elementos e table.remove para remover)*	Table *(observação: você pode usar o método table.insert para adicionar elementos e table.remove para remover)*
R	Queue *(observação: do pacote data.table)*	Vector *(observação: você pode usar o método append para adicionar elementos e length para obter o tamanho)*

EM ENTREVISTAS

Cada empresa é uma cultura, e cada entrevistador possui um padrão diferente. A única coisa que posso garantir é que nenhum entrevistador será igual. Nem todo entrevistador técnico é formado em ADS: podemos ter entrevistadores formados em Ciência da Computação, Engenharia da Computação, Sistemas da Informação ou outros cursos. Cada entrevistador terá um nível de conhecimento diferente, e é importante ter uma noção sobre o tipo de pergunta que você poderá receber. Novamente: existe uma possibilidade relativamente alta de que **não** te perguntem sobre filas e pilhas, mas esta possibilidade não é nula. Sendo assim, vamos a alguns exemplos de perguntas que você poderia receber:

Perguntas gerais

1. **O que é uma pilha (stack) e como ela funciona?**
    a. *Uma pilha é uma estrutura de dados que segue o princípio LIFO (Last In, First Out), onde o último elemento inserido é o primeiro a ser removido. As operações básicas em uma pilha são **push** (para adicionar um elemento) e **pop** (para remover um elemento).*
2. **Quando você usaria uma pilha em um programa?**
    a. *As pilhas são úteis quando a **ordem de processamento é importante** e quando você precisa **desfazer operações na ordem inversa em que foram feitas**. Exemplos incluem a implementação de desfazer/refazer em editores de texto, a execução de operações matemáticas reversas (como em expressões pós-fixas) e o rastreamento de chamadas de função em linguagens de programação.*
3. **O que é uma fila (queue) e como ela funciona?**
    a. *Uma fila é uma estrutura de dados que segue o princípio FIFO (First In, First Out), onde o primeiro elemento inserido é o primeiro a ser removido. As operações básicas em uma fila são **enqueue** (para adicionar um elemento) e **dequeue** (para remover um elemento).*
4. **Quando você usaria uma fila em um programa?**
    a. *Filas são úteis quando você precisa processar elementos na ordem em que foram recebidos. Elas são normalmente usadas em **sistemas de processamento de dados em lotes, gerenciamento de tarefas em sistemas operacionais e em sistemas de comunicação entre processos**.*
5. **Como você implementaria uma pilha ou uma fila em uma linguagem de programação como Java?**
    a. *Em Java, você pode implementar uma pilha usando a classe **Stack** (que é uma subclasse de **Vector**, mas é recomendado usar **Deque** da interface **java.util.Deque** com a implementação **ArrayDeque** para melhor desempenho) e uma fila usando a interface **Queue** com a implementação **LinkedList** ou **ArrayDeque**.*

6. **Quais são as complexidades de tempo das operações comuns em pilhas e filas?**
    a. *Para uma pilha e uma fila implementadas de forma eficiente, as operações **push**, **pop**, **enqueue** e **dequeue** têm complexidade de tempo O(1), ou seja, são operações de tempo constante.*

Para back-end

7. **Como você implementaria um sistema de cache com capacidade limitada usando uma fila?**
    a. *Em um sistema de cache com capacidade limitada, você pode usar uma fila para armazenar os itens em ordem de acesso mais recente. Ao adicionar um novo item à fila, verifique se a fila já atingiu a **capacidade máxima**. Se sim, remova o item mais antigo da fila antes de adicionar o novo item.*
8. **Como você implementaria um sistema de mensagens assíncronas entre microserviços usando filas?**
    a. *Em um sistema de mensagens assíncronas entre microserviços, você pode usar uma fila para enviar mensagens de um microserviço para outro. Cada microserviço pode ter sua própria fila de entrada para receber mensagens e processá-las em seu próprio ritmo. Isso ajuda a desacoplar os microserviços e a garantir que as mensagens sejam processadas mesmo se um dos microserviços estiver temporariamente indisponível.*
9. **Como você lidaria com a concorrência ao usar uma fila em um ambiente multi-threaded?**
    a. *Ao lidar com filas em um ambiente multi-threaded, é importante garantir que as operações de manipulação da fila (por exemplo, adicionar e remover elementos) sejam **thread-safe**. Você pode usar estruturas de dados que oferecem suporte a operações thread-safe, como ConcurrentLinkedQueue em Java.*
10. **Como você implementaria um sistema de agendamento de tarefas usando uma fila de prioridade?**
    a. *Em um sistema de agendamento de tarefas, você pode usar uma fila de prioridade para armazenar as tarefas a serem executadas, onde a prioridade é baseada no tempo de execução ou em alguma outra métrica relevante. Ao adicionar uma nova tarefa à fila, ela é inserida na posição correta de acordo com sua prioridade. Ao executar uma tarefa, você remove a tarefa com a maior prioridade da fila.*

Para front-end (mais difícil de acontecer, mas não impossível)

11. **Como você lidaria com a renderização de uma lista grande de elementos de forma eficiente em uma página web?**
    a. *Uma abordagem eficiente seria usar técnicas de virtualização, como o uso de bibliotecas como React Virtualized ou vue-virtual-scroller, que renderizam apenas os elementos visíveis na tela, melhorando o desempenho da aplicação.*
    b. *Observação: renderização em lote = LIFO = deixamos de renderizar os elementos quando não precisamos mais deles.*
12. **Como você implementaria um sistema de notificações em tempo real em uma aplicação web?**
    a. *Para implementar um sistema de notificações em tempo real, você poderia usar WebSockets para estabelecer uma conexão bidirecional entre o cliente e o servidor. Quando uma nova notificação é gerada no servidor, ela é enviada para o cliente através da conexão WebSocket e exibida na interface do usuário.*
    b. *Observação: receber notificações na ordem em que chegam = FIFO = filas.*
13. **Como você lidaria com o gerenciamento de estado em uma aplicação web complexa?**
    a. *Para lidar com o gerenciamento de estado em uma aplicação web complexa, você pode utilizar bibliotecas como Redux (para React) ou Vuex (para Vue.js) para centralizar o estado da aplicação e facilitar a comunicação entre os componentes.*
    b. *Observação: estas bibliotecas podem usar conceitos de LIFO (pilha) para manter o histórico de navegação.*

EXERCÍCIOS DE FIXAÇÃO

*A ideia de todos estes exercícios é o de praticar casos de uso de **filas e pilhas**. Lembre-se:*

- *São exercícios de fixação. Logo, você pode fazê-los opcionalmente. Não há a necessidade de entrega.*
- *Você pode fazer **preferencialmente (e não obrigatoriamente)** em Java. Como são exercícios de prática, você também pode testar na sua linguagem favorita.*
- *No final das questões você terá acesso às respostas. Elas são somente uma sugestão. Existem implementações mais refatoradas ou mais performáticas para todos estes exemplos.*

Exercícios

1. Você foi contratado por uma **empresa de logística** para ajudar a implementar um sistema de rastreamento de encomendas. A empresa recebe diversas encomendas por dia e precisa garantir que todas sejam despachadas corretamente. Para isso, você irá utilizar uma pilha para manter o controle das encomendas recebidas e prontas para despacho.

    - Crie uma classe **Encomenda** com os seguintes atributos:

        1. **cliente** (String): o nome do cliente que fez a encomenda.
        2. **endereco** (String): o endereço de entrega da encomenda.

    - Crie uma classe **EmpresaDeEntregas** com um método **main** que realiza as seguintes ações:

        1. Cria uma pilha de encomendas.
        2. Simula a chegada de pelo menos três encomendas, adicionando-as à pilha.
        3. Exibe na tela as encomendas prontas para despacho, retirando-as da pilha.

    - Certifique-se de usar os métodos **push** para adicionar encomendas à pilha e **pop** para retirá-las.
    - Teste o seu programa e verifique se as encomendas são exibidas corretamente na tela.
    - *Dica: Use a classe **Stack** do Java para implementar a pilha.*
    - *Dica 2: A própria documentação diz que existe um "Deque". O que acha de procurar na documentação onde está este Deque? E o que acha de substituir o Stack pelo Deque?*
    - *Dica 3: O que seria "testar"? Como comprovar que o seu programa funciona? O que acha de usar um loop com o while para iterar pela sua pilha até não existirem mais elementos? O método **isEmpty()** poderá te ajudar nisso. Isto também está na documentação.*

2. Você foi contratado por uma **rede de supermercados** para implementar um sistema de atendimento aos clientes que estão na fila do caixa. A empresa deseja utilizar uma fila para organizar o atendimento, garantindo que os clientes sejam atendidos na ordem correta de chegada.

- Crie uma classe **Cliente** com o seguinte atributo:
    1. **nome** (String): o nome do cliente.
- Crie uma classe **CaixaDeSupermercado** com um método **main** que realiza as seguintes ações:
    1. Cria uma fila de clientes.
    2. Simula a chegada de pelo menos três clientes, adicionando-os à fila.
    3. Mostra na tela os clientes sendo atendidos, removendo-os da fila.
- Certifique-se de usar os métodos **add** para adicionar clientes à fila e **poll** para removê-los.
- Teste o seu programa e verifique se os clientes são atendidos na ordem correta de chegada.
- *Dica: Use a interface **Queue** e a implementação **LinkedList** do Java para implementar a fila.*
- *Dica 2: A documentação possui uns exemplos interessantes. Qual é a diferença entre os métodos **add()/offer()**, **remove()/poll()** e **element()/peek()**?*

3. Você foi contratado pelo **departamento de TI de uma rede multinacional de fast-food** para desenvolver um sistema de gerenciamento de pedidos. Cada restaurante terá uma fila para representar os pedidos que os clientes fazem e uma outra fila para representar os pedidos que já foram preparados e estão prontos para serem entregues aos clientes.
    - Crie uma classe **Pedido** com os seguintes atributos:
        1. **numero** (int): o número do pedido.
        2. **descricao** (String): uma descrição do pedido.
    - Crie uma classe **Restaurante** com os seguintes métodos:
        1. **adicionarPedido(Pedido pedido)**: adiciona um pedido à fila de pedidos.
        2. **prepararPedido()**: move um pedido da fila de pedidos aguardando preparo para a fila de pedidos preparados.
        3. **entregarPedido()**: remove um pedido da fila de pedidos preparados, simulando a entrega ao cliente.
        4. **listarPedidos()**: exibe na tela os pedidos das duas filas, indicando quais estão prontos para serem entregues.
    - No método **main**, simule o funcionamento do restaurante, adicionando alguns pedidos à fila, preparando-os e entregando-os.
    - Explore a documentação oficial do Java para entender como usar as classes **Queue** e **Stack** para implementar a fila e a pilha, respectivamente.

- Teste o seu programa e verifique se os pedidos são gerenciados corretamente.
- *Dica: o que deve acontecer no seu código caso não existam mais pedidos?*

4. ***[Desafio]*** Você foi contratado para trabalhar em uma **empresa multinacional**. Nessas empresas é comum transferirmos dados entre diferentes sistemas. Como um exemplo, empresas como Volvo, Boticário, BRF, Volkswagen e outras transferem dados de sensores de máquinas da linha de produção para os sistemas de gestão usando conceitos de internet das coisas (IoT). As filas de mensagens são preferíveis em relação a APIs em cenários onde há necessidade de comunicação assíncrona entre sistemas. Elas promovem o desacoplamento, permitindo que sistemas operem independentemente, facilitando a escalabilidade e a manutenção. Além disso, as filas de mensagens garantem a tolerância a falhas ao armazenar mensagens até que o sistema esteja pronto para processá-las, e oferecem suporte a processamento assíncrono, permitindo que sistemas processem mensagens em seu próprio ritmo. A capacidade de suportar múltiplos consumidores e o balanceamento de carga são benefícios adicionais, distribuindo a carga de trabalho de forma equilibrada entre os sistemas. Portanto, o seu objetivo é o de desenvolver um **sistema de comunicação entre duas aplicações usando uma fila de mensagens**. Uma aplicação será responsável por enviar mensagens para a fila (Produtor) e a outra será responsável por receber essas mensagens (Consumidor). Portanto, você deverá fazer o seguinte:
   - Crie uma classe que representará o sistema de comunicação por message queues (filas de mensagem)Main. Esta classe deve utilizar uma **LinkedBlockingQueue** para representar a fila de mensagens.
   - Implemente a classe **Produtor** que será responsável por enviar mensagens para a fila. O **Produtor** deve enviar pelo menos cinco mensagens, uma a cada segundo, para simular algum processamento entre o envio de cada mensagem.
   - Implemente a classe **Consumidor** que será responsável por receber mensagens da fila. O **Consumidor** deve receber e exibir as mensagens recebidas.
   - Utilize **threads** para simular a comunicação assíncrona entre as duas aplicações.
   - Teste o seu programa e verifique se as mensagens são enviadas e recebidas corretamente entre as aplicações.
   - *Dica: vários conceitos novos são introduzidos neste desafio – por isso o nome. Eu, se fosse você, começaria entendendo o que temos de novo aqui:*
     1. *LinkedBlockingQueue é um tipo de Queue. O que acha de ler a documentação para entender a diferença entre ambos?*
     2. *Threads permitem execuções **em paralelo** de algoritmos. O que acha de entender melhor as threads? Primeiro, pela documentação oficial. Depois, por uma explicação mais prática.*
     3. *Falei sobre "message queues". O que acha de pesquisar sobre isso no Google?*
     4. *Falei sobre as vantagens de usarmos filas de mensagem entre sistemas, mas não falei sobre as desvantagens. O que acha de pesquisar no Google sobre "when not to use message queues" (sem aspas)?*
   - *Dica 2: Use os métodos **put** e **take** da **LinkedBlockingQueue** para enviar e receber mensagens, respectivamente.*

RESPOSTAS

1. Empresa de logística:

```
import java.util.Stack;

public class Main {
    static class Encomenda {
        private String cliente;
        private String endereco;

        public Encomenda(String cliente, String endereco) {
            this.cliente = cliente;
            this.endereco = endereco;
        }

        public String getCliente() {
            return cliente;
        }

        public String getEndereco() {
            return endereco;
        }
    }

    public static void main(String[] args) {
        Stack<Encomenda> pilhaDeEncomendas = new Stack<>();

        // Simulando a chegada de algumas encomendas
        pilhaDeEncomendas.push(new Encomenda("Cliente 1", "Rua A, 123"));
        pilhaDeEncomendas.push(new Encomenda("Cliente 2", "Rua B, 456"));
        pilhaDeEncomendas.push(new Encomenda("Cliente 3", "Rua C, 789"));

        // Mostrando as encomendas prontas para despacho
        System.out.println("Encomendas prontas para despacho:");
        while (!pilhaDeEncomendas.isEmpty()) {
            Encomenda encomenda = pilhaDeEncomendas.pop();
            System.out.println("Cliente: " + encomenda.getCliente() + ", Endereço: " + encomenda.getEndereco());
        }
    }
}
```

2. Supermercado

```
import java.util.LinkedList;
import java.util.Queue;

public class Main {
    public static void main(String[] args) {
        Queue<Cliente> filaDeClientes = new LinkedList<>();

        // Simulando a chegada de alguns clientes
        filaDeClientes.add(new Cliente("Cliente 1"));
        filaDeClientes.add(new Cliente("Cliente 2"));
        filaDeClientes.add(new Cliente("Cliente 3"));

        // Atendendo os clientes da fila
        System.out.println("Atendendo os clientes:");
        while (!filaDeClientes.isEmpty()) {
            Cliente cliente = filaDeClientes.poll();
            System.out.println("Cliente atendido: " + cliente.getNome());
        }
    }

    static class Cliente {
        private String nome;

        public Cliente(String nome) {
            this.nome = nome;
        }

        public String getNome() {
            return nome;
        }
    }
}
```

3. Rede de Fast-Food

```
import java.util.LinkedList;
import java.util.Queue;

public class Main {
    public static void main(String[] args) {
        Queue<Pedido> filaDePedidos = new LinkedList<>();
        Queue<Pedido> pedidosProntos = new LinkedList<>();

        // Simulando a chegada de alguns pedidos
        filaDePedidos.add(new Pedido(1, "Pizza"));
        filaDePedidos.add(new Pedido(2, "Hambúrguer"));
        filaDePedidos.add(new Pedido(3, "Salada"));
        filaDePedidos.add(new Pedido(4, "Sorvete"));
        filaDePedidos.add(new Pedido(5, "Hambúrguer Duplo"));

        // Preparando os pedidos
        while (!filaDePedidos.isEmpty()) {
            Pedido pedido = filaDePedidos.poll();
            System.out.println("Preparando pedido " + pedido.getNumero() + ": " + pedido.getDescricao());
            pedidosProntos.add(pedido);
        }

        // Entregando os pedidos
        System.out.println("\nEntregando pedidos:");
        while (!pedidosProntos.isEmpty()) {
            Pedido pedido = pedidosProntos.poll();
            System.out.println("Pedido " + pedido.getNumero() + " entregue: " + pedido.getDescricao());
        }
    }

    static class Pedido {
        private int numero;
        private String descricao;

        public Pedido(int numero, String descricao) {
            this.numero = numero;
            this.descricao = descricao;
        }

        public int getNumero() {
            return numero;
        }

        public String getDescricao() {
            return descricao;
        }
    }
}
```

4. Desafio

```
import java.util.concurrent.LinkedBlockingQueue;

public class Main {

    public static void main(String[] args) {
        LinkedBlockingQueue<String> queue = new LinkedBlockingQueue<>();

        // Produtor envia mensagens para a fila
        Thread produtor = new Thread(() -> {
            try {
                for (int i = 0; i < 5; i++) {
                    String mensagem = "Mensagem " + i;
                    queue.put(mensagem);
                    System.out.println("Produtor enviou: " + mensagem);
                    Thread.sleep(1000); // Simula algum processamento
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        // Consumidor recebe mensagens da fila
        Thread consumidor = new Thread(() -> {
            try {
                for (int i = 0; i < 5; i++) {
                    String mensagem = queue.take();
                    System.out.println("Consumidor recebeu: " + mensagem);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        produtor.start();
        consumidor.start();
    }
}
```

5.

PROFESSOR-AUTOR

Wellington Rodrigo Monteiro

PUCPR
GRUPO MARISTA