**Exercício 1: Simulador de Elevador**

**Descrição:**

Neste exercício, você deverá desenvolver um simulador de elevador para um prédio de 10 andares, numerados de 0 a 9. O simulador deve gerenciar o funcionamento básico do elevador, incluindo:

* **Movimentação do elevador** entre os andares,
* **Atendimento de chamadas** feitas nos andares ou dentro do elevador,
* **Abertura e fechamento das portas** em cada parada.

**Funcionalidades esperadas:**

1. **Chamadas externas**: Quando um usuário em qualquer andar chama o elevador (para subir ou descer), a chamada deve ser registrada.
2. **Chamadas internas**: Dentro do elevador, o usuário pode selecionar o andar desejado. O elevador deve registrar esses pedidos.
3. **Movimentação**: O elevador deve mover-se na direção apropriada para atender as chamadas de forma eficiente. Ele deve otimizar o trajeto, atendendo as chamadas de acordo com sua posição e a direção atual.
4. **Abertura e fechamento de portas**: Quando o elevador chega ao andar solicitado, ele deve abrir as portas, simular um tempo de espera (como se os usuários estivessem entrando e saindo) e depois fechar as portas antes de continuar a operação.
5. **Restrições**: O elevador deve mover-se apenas entre os andares 0 e 9, e não pode abrir portas enquanto estiver em movimento.

**Requisitos:**

* Utilize uma linguagem de programação como Python, JavaScript, ou qualquer outra de sua escolha.
* Crie uma classe Elevador que vai gerenciar todas as funcionalidades do simulador.
* O simulador deve ser interativo: permita que o usuário faça chamadas (externas e internas) e visualize o movimento do elevador em tempo real ou por meio de mensagens de status.

**Sugestão de extensão:**

* Implemente funcionalidades adicionais, como um sistema para registrar quantas pessoas estão no elevador, ou defina uma capacidade máxima de usuários.

### ****Exercício 7: Busca em Largura (BFS) em Grafos****

#### Descrição:

Neste exercício, você deverá explicar e implementar o algoritmo de **Busca em Largura (BFS)** em grafos. A BFS é uma técnica de busca que explora um grafo por níveis ou camadas, encontrando todos os vértices a uma distância fixa antes de avançar para os mais distantes.

#### Objetivo:

Implementar o algoritmo de Busca em Largura para encontrar o **caminho mais curto** entre dois vértices em um grafo **não ponderado**.

#### Explicação do Algoritmo:

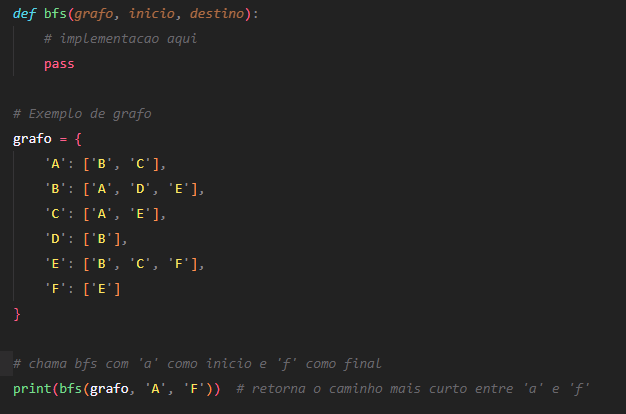
1. **Inicialização**: Comece a partir de um vértice de origem. Marque esse vértice como visitado e adicione-o a uma fila (estrutura de dados do tipo FIFO).
2. **Exploração**: Enquanto houver vértices na fila, remova o primeiro vértice e explore seus vizinhos não visitados. Cada vértice visitado é adicionado à fila.
3. **Caminho mais curto**: Durante a exploração, se o vértice de destino for encontrado, o algoritmo termina e retorna o caminho percorrido. A BFS garante que o primeiro caminho encontrado para o destino será o mais curto em termos de número de arestas, pois ela visita vértices por "camadas".
4. **Encerramento**: Se todos os vértices possíveis forem visitados e o destino não for encontrado, significa que não há caminho entre os dois vértices.

#### 

#### Requisitos:

1. Implementar a função bfs(grafo, inicio, destino) que recebe um grafo, um vértice inicial e um vértice de destino, e retorna o caminho mais curto entre eles.
2. O grafo pode ser representado como um dicionário onde as chaves são os vértices e os valores são listas de vértices adjacentes (vizinhos).

Exemplo de Implementação:



#### Sugestão de extensão:

* Implemente uma versão que também conte o número de arestas percorridas.
* Adapte o algoritmo para grafos ponderados usando o algoritmo de Dijkstra, para encontrar o caminho mais curto baseado no peso das arestas.

Respostas: