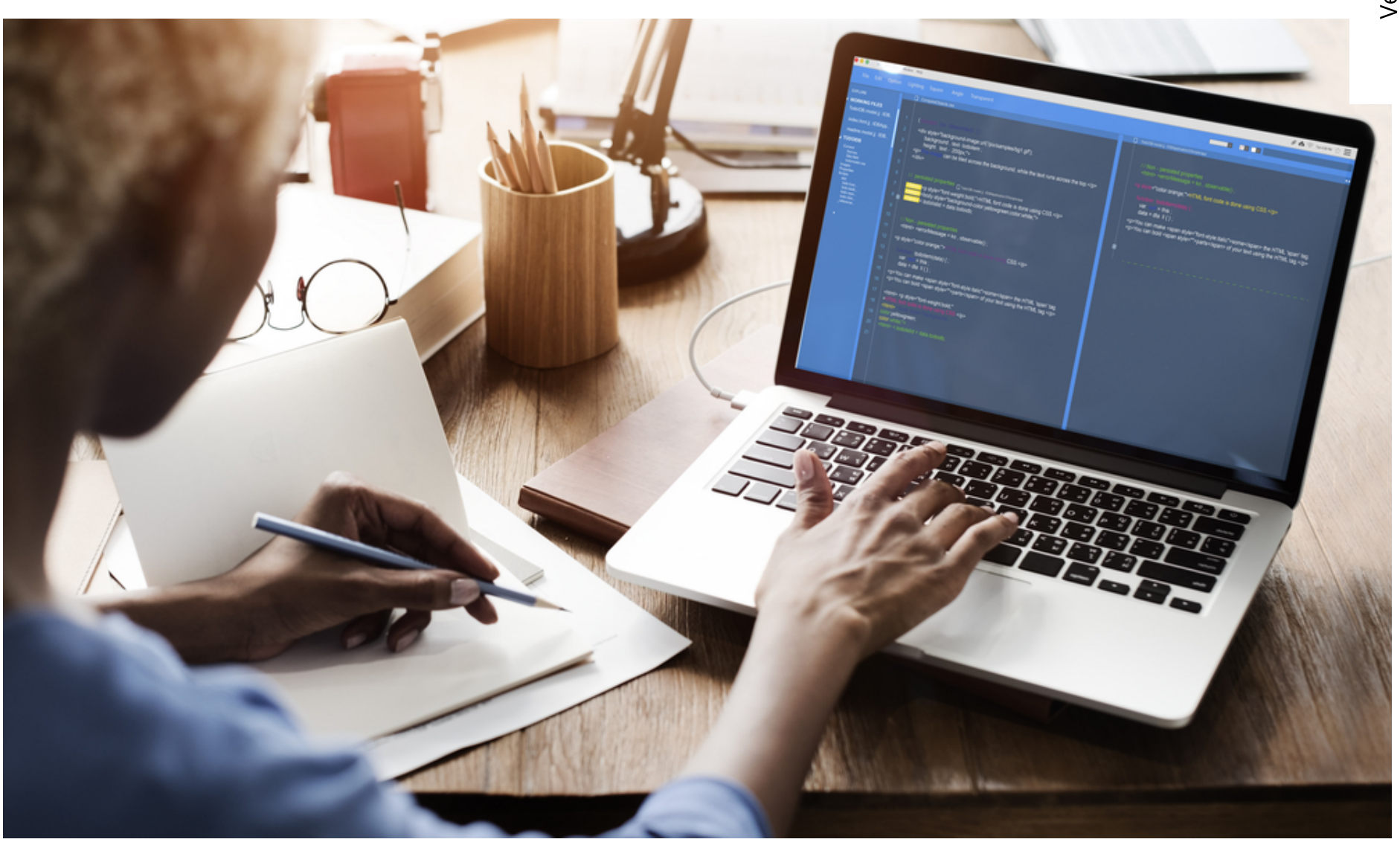
FOCO NO MERCADO DE TRABALHO

ALGORITMOS DE BUSCA

Vanessa Cadan Scheffer

A ESCOLHA DE UM ALGORITMO DE BUSCA

Por meio dos algoritmos de busca, podemos tratar dados de diversas maneiras.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

DESAFIO

Empresas têm investido na contratação de profissionais com conhecimento na área de análise de dados. Entre as diversas especialidades que envolvem essa área, os engenheiros de dados são responsáveis por capturar, tratar e disponibilizar os dados para as áreas de análise e negócio. Entre os vários

tratamentos que precisam ser feitos nos dados, o *dedup* é um deles. *Dedup* é a abreviação para deduplicação de dados, que consiste em elimiar registros duplicados dos dados.

Como desenvolvedor em uma empresa de consultoria, você foi alocado para atender um cliente que precisa fazer a ingestão de dados de uma nova fonte e tratá-los. Alocado em uma equipe ágil, você se responsabilizou por implementar uma solução que recebe uma lista de CPFs e a retorna com a transformação de dedup. Além de fazer o dedup, também foi solicitado a você retornar somente a parte numérica do CPF e verificar se eles possuem 11 dígitos. Se não não possuírem, o CPF deve ser eliminado da lista.

Como você fará o dedup usando um algoritmo de busca? É mais apropriado usar uma busca linear ou binária? Como vai remover os caracteres *ponto* e *traço* que costumam aparecer nos CPFs? Como vai verificar se os CPFs possuem 11 dígitos?

RESOLUÇÃO

Você foi alocado em um projeto no qual precisa implementar uma função que faça a transformação de dedup em uma lista de CPFs. Além do dedup, você também precisa remover os caracteres *ponto* e *traço* do CPF, deixando somente números, e verificar se eles possuem 11 dígitos [caso contrário, o CPF não deve ser incluído na lista de CPFs válidos].

Para essa implementação, vamos utilizar um algoritmo de busca linear, uma vez que não podemos garantir que os CPFs virão ordenados, o que é uma exigência da busca binária. Poderíamos, internamente, fazer essa ordenação, mas qual seria o custo de um algoritmo de ordenação? Como não temos essa resposta, vamos adotar a busca linear. Uma vez que os CPFs podem ter números, traço e ponto, eles devem ser tratados como string, e, como vantagem, ganhamos a função *replace()*, que troca um caractere por outro. Podemos usar a função *len()* para verificar se o CPF tem 11 dígitos. Veja uma possível implementação para a solução

In [11]:

```
# Parte 1 - Implementar o algoritmo de busca linear
def executar_busca_linear(lista, valor):
    tamanho_lista = len(lista)
    for i in range(tamanho_lista):
        if valor == lista[i]:
            return True
    return False
```

Ver anotações 0

In [12]:

```
# Parte 2 - Criar a função que faz o dedup e os tratamentos no cpf
def criar_lista_dedup(lista):
    lista_dedup = []
    for cpf in lista:
        cpf = str(cpf)
        cpf = cpf.replace("-", "").replace(".", "")
        if len(cpf) == 11:
            if not executar_busca_linear(lista_dedup, cpf):
                lista_dedup.append(cpf)

    return lista_dedup
```

In [13]:

```
# Parte 3 - Criar uma função de teste
def testar_funcao(lista_cpfs):
    lista_dedup = criar_lista_dedup(lista_cpfs)
    print(lista_dedup)



lista_cpfs = ['111.111.111-11', '11111111111', '222.222.222-22', '333.333.333-
33', '22222222222', '444.44444']
testar_funcao(lista_cpfs)
```

```
['11111111111', '22222222222', '33333333333']
```

Implementamos nossa solução em três etapas. As etapas 1 e 2 são, de fato, parte da solução. A etapa 3 é uma boa prática de programação, já que, para toda solução, é viável ter uma ou mais funções de teste. Na primeira parte, implementamos a função de busca linear, a qual vai receber uma lista e um valor ser procurado. Lembre-se de que, nesse algoritmo, toda a lista é percorrida até encontrar (ou não) o valor.

Na segunda parte da solução, implementamos a função que faz ajustar o CPF, qu valida seu tamanho e que faz o dedup. Vamos entender o processo.

- Linha 3 - criamos uma estrutura de dados, do tipo lista, vazia. Essa lista armazenará os valores únicos dos CPFs.
- Linha 4 - criamos a estrutura de repetição, que percorrerá cada CPF da lista original.
- Linha 5 - fazemos o cast (conversão forçada) do CPF para o tipo string. Quando percorremos uma lista, cada elemento pode ter um tipo diferente. Convidamos você a fazer um teste: antes de fazer o cast, peça para imprimir o tipo do dado `print(type(cpf))`. Você verá que, nos casos em que o CPF só tem número, o tipo é inteiro.
- Linha 6 - usamos um encadeamento da função *replace* para substituir o traço por "nada". Quando usamos `replace("-", "")`, o primeiro elemento é o que queremos substituir e o segundo é o que queremos colocar no lugar. Nesse caso, veja que não há nada dentro das aspas, ou seja, o traço será apenas removido. O mesmo acontece para o ponto.
- Linha 7 - checamos se o tamanho do CPF é 11. Se não for, nada acontece, e o loop passa para a próxima iteração. Se for, então passamos para a linha 8.
- Linha 8 - aqui está a "mágica" do dedup. Invocamos a função de busca, passando como parâmetro a lista_dedup (que é a lista final que queremos) e o CPF. Essa função vai procurar, na lista_dedup, o valor. Caso o encontre, retornará True; caso não o encontre, retornará False. Queremos justamente os

casos falsos, pois isso quer dizer que o CPF ainda não está na lista correta, razão pela qual usamos `if not`. Caso não esteja, então passamos a linha 9.

- Linha 9 - adicionamos à lista_dedup o CPF, já validado, transformado e com a garantia de que não está duplicado.

Na terceira parte da solução, apenas fazemos um teste. Dada uma sequência fictícia de CPFs, veja que o resultado é exatamente o que queremos: CPFs somen numéricos sem duplicações. Veja como o algoritmo foi capaz de lidar com CPFs que contêm traços e pontos da mesma forma que o fez com os somente numéricos.

DESAFIO DA INTERNET

O site https://www.hackerrank.com/ é uma ótima opção para quem deseja treinar as habilidades de programação. Nesse portal, é possível encontrar vários desafios, divididos por categoria e linguagem de programação. Na página inicial, você encontrará a opção para empresas e para desenvolvedores. Escolha a segunda e faça seu cadastro.

Após fazer o cadastro, faça login para ter acesso ao dashboard (quadro) de desafios. Navegue até a opção de algoritmos e clique nela. Uma nova página será aberta, do lado direito da qual você deve escolher o subdomínio "search" para acessar os desafios pertinentes aos algoritmos de busca. Tente resolver o desafio "Missing Numbers"!

Você deve estar se perguntando: " por que eu deveria fazer tudo isso?". Acreditamos que o motivo a seguir pode ser bem convincente: algumas empresas utilizam o site HackerRank como parte do processo seletivo. Então, é com você!