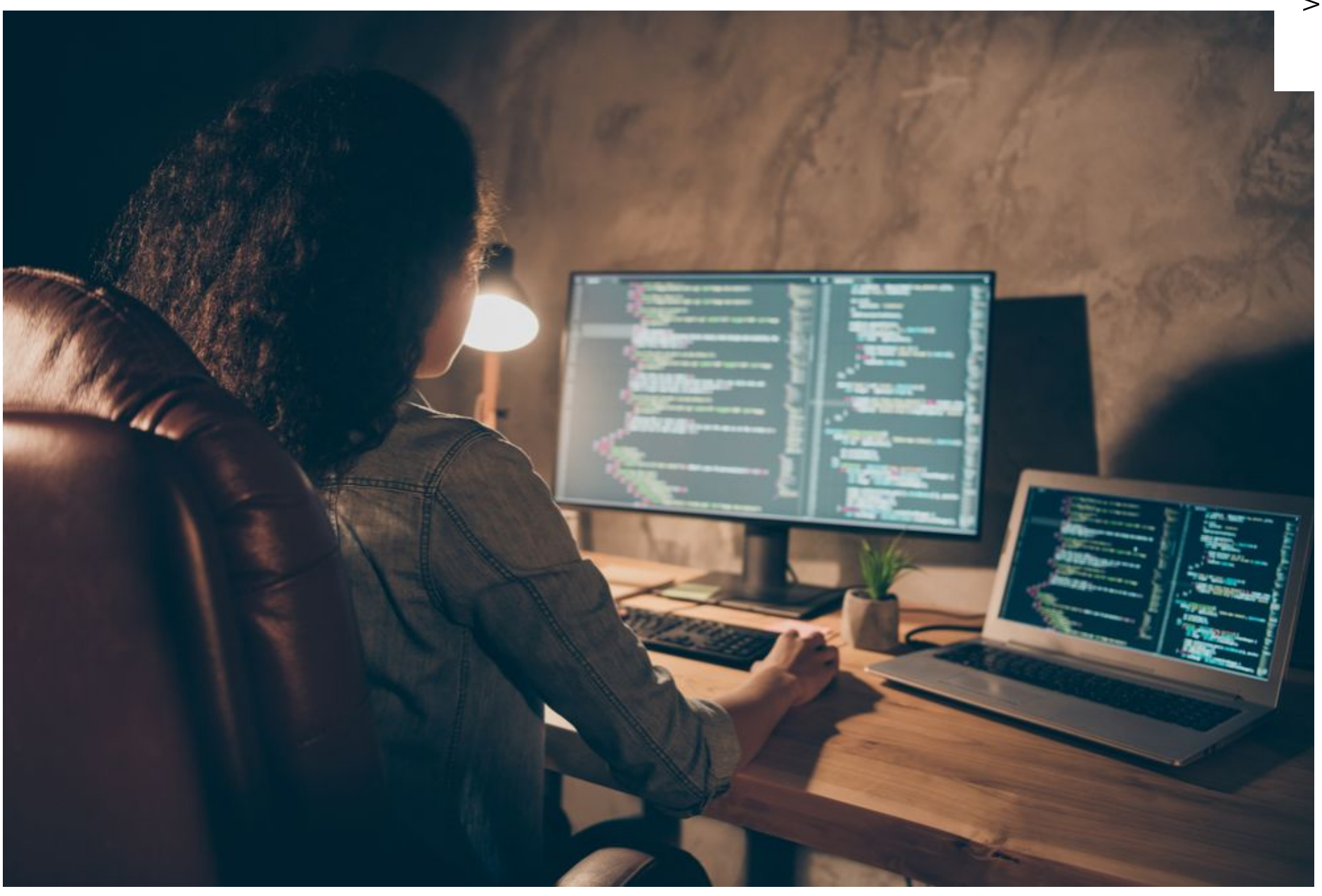
FOCO NO MERCADO DE TRABALHO

ALGORITMOS DE ORDENAÇÃO

Vanessa Cadan Scheffer

COMO APRESENTAR OS DADOS DE FORMA ORDENADA?

Implementando algoritmos de ordenação.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

DESAFIO

Como desenvolvedor em uma empresa de consultoria, você continua alocado para atender um cliente que precisa fazer a ingestão de dados de uma nova fonte e tratá-las. Você já fez uma entrega na qual implementou uma solução que faz o

dedup em uma lista de CPFs, retorna somente a parte numérica do CPF e verifica se eles possuem 11 dígitos.

Os clientes aprovaram a solução, mas solicitaram que a lista de CPFs válidos fosse entregue em ordem crescente para facilitar o cadastro. Eles enfatizaram a necessidade de ter uma solução capaz de fazer o trabalho para grandes volumes de dados, no melhor tempo possível. Uma vez que a lista de CPFs pode crescer exponencialmente, escolher os algoritmos mais adequados é importante nesse caso.

Portanto, nesta nova etapa, você deve tanto fazer as transformações de limpeza e validação nos CPFs (remover ponto e traço, verificar se tem 11 dígitos e não deixar valores duplicados) quanto fazer a entrega em ordem crescente. Quais algoritmos você vai escolher para implementar a solução? Você deve apresentar evidências de que fez a escolha certa!

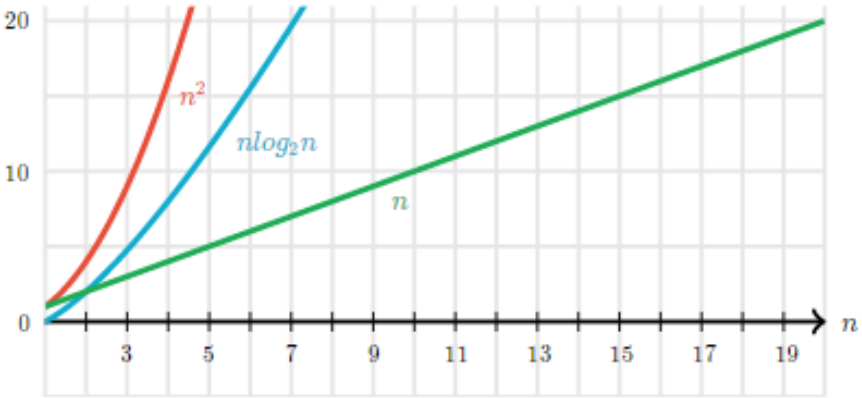
RESOLUÇÃO

Alocado em um novo desafio, é hora de escolher e implementar os melhores algoritmos para a demanda que lhe foi dada. Consultando a literatura sobre algoritmos de busca, você encontrou que a busca binária performa melhor que a busca linear, embora os dados precisem estar ordenados. No entanto, agora você já sabe implementar algoritmos de ordenação!

O cliente enfatizou que a quantidade de CPFs pode aumentar exponencialmente, o que demanda algoritmos mais eficazes. Ao pesquisar sobre a complexidade dos algoritmos de ordenação, você encontrou no portal Khan Academy (2020) um quadro comparativo (Figura 2.10) dos principais algoritmos. Na Figura 2.10, fica evidente que o algoritmo de ordenação *merge sort* é a melhor opção para o cliente, visto que, para o pior caso, é o que tem menor complexidade de tempo.

Figura 2.10|Comparação de algoritmos de ordenação

Algoritmo	Tempo de execução no pior caso	Tempo de execução no melhor caso	Tempo de execução médio
ordenação por seleção (selection sort)	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Insertion sort	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$
Ordenação por combinação (merge sort)	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$



Fonte: Khan Academy (2020, [s.p.]).

Agora que já decidiu quais algoritmos implementar, é hora de colocar a mão na massa!

In [9]:

```
# Parte 1 - Implementar o algoritmo de ordenação merge sort


def executar_merge_sort(lista, inicio=0, fim=None):
    if not fim:
        fim = len(lista)

    if fim - inicio > 1:
        meio = (inicio + fim) // 2
        executar_merge_sort(lista, inicio, meio)
        executar_merge_sort(lista, meio, fim)
        executar_merge(lista, inicio, meio, fim)
    return lista



def executar_merge(lista, inicio, meio, fim):
    esquerda = lista[inicio:meio]
    direita = lista[meio:fim]
    topo_esquerda = topo_direita = 0
    for p in range(inicio, fim):
        if topo_esquerda >= len(esquerda):
            lista[p] = direita[topo_direita]
            topo_direita += 1
        elif topo_direita >= len(direita):
            lista[p] = esquerda[topo_esquerda]
            topo_esquerda += 1
        elif esquerda[topo_esquerda] < direita[topo_direita]:
            lista[p] = esquerda[topo_esquerda]
            topo_esquerda += 1
        else:
            lista[p] = direita[topo_direita]
            topo_direita += 1
```

In [10]:

```
# Parte 2 - Implementar o algoritmo de busca binária


def executar_busca_binaria(lista, valor):
    minimo = 0
    maximo = len(lista) - 1
    while minimo <= maximo:
        meio = (minimo + maximo) // 2
        if valor < lista[meio]:
            maximo = meio - 1
        elif valor > lista[meio]:
            minimo = meio + 1
        else:
            return True
    return False
```

In [11]:

```
# Parte 3 - Implementar a função que faz a verificação do cpf, o dedup e devolve
o resultado esperado


def criar_lista_dedup_ordenada(lista):
    lista = [str(cpf).replace('.','').replace('-','') for cpf in lista]
    lista = [cpf for cpf in lista if len(cpf) == 11]
    lista = executar_merge_sort(lista)


    lista_dedup = []
    for cpf in lista:
        if not executar_busca_binaria(lista_dedup, cpf):
            lista_dedup.append(cpf)
    return lista_dedup
```

In [12]:

```
# Parte 4 - Criar uma função de teste


def testar_funcao(lista_cpfs):
    lista_dedup = criar_lista_dedup_ordenada(lista_cpfs)
    print(lista_dedup)



lista_cpfs = ['44444444444', '111.111.111-11', '11111111111', '222.222.222-22',
'333.333.333-33', '22222222222', '444.44444']
testar_funcao(lista_cpfs)
```

```
['11111111111', '22222222222', '33333333333', '44444444444']
```

Ver anotações 0

Implementamos os algoritmos de ordenação (merge sort) e de busca (binária), conforme aprendemos nas aulas. No algoritmo de ordenação, fazemos um tratamento na variável fim para que não precise ser passada explicitamente na primeira chamada. Vamos focar na função *criar_lista_dedup_ordenada*. Na linha 4, usamos uma list comprehension para remover o ponto e o traço dos CPFs. Na linha 5, criamos novamente uma list comprehension, agora para guardar somente os CPFs que possuem 11 dígitos. Em posse dos CPFs válidos, chamamos a função de ordenação. Agora que temos uma lista ordenada, podemos usar a busca binária para verificar se o valor já está na lista; caso não estiver, então ele é adicionado. Na quarta parte da nossa solução, implementamos uma função de teste para checar se não há nenhum erro e se a lógica está correta.

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 encontra 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 "**sort**" para acessar os desafios pertinentes aos algoritmos de busca. Tente resolver o desafio "Insertion Sort - Part 1"!

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