Folha de Consulta para Entrevista

Da entrevista Master The Coding de Andrei Neagoie: estruturas de dados + algoritmos

Os 3 pilares de um bom código:

1. Legível 2.
Complexidade de tempo
3. Complexidade de espaço

Quais habilidades o entrevistador está procurando:

- Habilidades analíticas - Como você pode pensar em problemas e analisar as coisas?
- Habilidades de codificação - Você codifica bem, escrevendo um código limpo, simples, organizado e legível?
- Conhecimento técnico - Você conhece os fundamentos da vaga para a qual está se candidatando?
- Habilidades de comunicação: Sua personalidade combina com a cultura da empresa?

Passo a passo através de um problema:

1. Quando o entrevistador fizer a pergunta, anote os pontos-chave no topo (ou seja, classificados variedade). Certifique-se de ter todos os detalhes. Mostre como você é organizado.

2. Certifique-se de verificar novamente: Quais são as entradas? Quais são as saídas?

3. Qual é o valor mais importante do problema? Você tem tempo, espaço e memória, etc. Qual é o objetivo principal?

4. Não seja irritante e faça muitas perguntas.

5. Comece com a abordagem ingênua/força bruta. Primeira coisa que vem em mente. Isso mostra que você é capaz de pensar bem e criticamente (você não precisa escrever este código, apenas falar sobre ele).

6. Diga a eles por que essa abordagem não é a melhor (ou seja, O(n^2) ou superior, não legível, etc...)

7. Analise sua abordagem, comente as coisas e veja onde você pode quebrar as coisas. Alguma repetição, gargalos como O(N^2) ou trabalho desnecessário? Você usou todas as informações que o entrevistador lhe deu? Gargalo é a parte do código com o maior Big O. Concentre-se nisso. Às vezes, isso também ocorre com o trabalho repetido.

8. Antes de começar a codificar, percorra seu código e anote as etapas que você seguirá seguir.

9. Modularize seu código desde o início. Divida seu código em belos pedaços pequenos e adicione apenas comentários, se necessário.

10. Comece realmente a escrever seu código agora. Lembre-se de que quanto mais você preparar e entender o que precisa codificar, melhor será o quadro branco. Portanto, nunca comece uma entrevista no quadro branco sem ter certeza de como as coisas vão funcionar. Essa é uma receita para o desastre.
Lembre-se: muitas entrevistas fazem perguntas que você não conseguirá responder completamente a tempo. Então pense: O que eu posso mostrar para mostrar que eu posso fazer isso e sou melhor que os outros? codificadores. Divida as coisas em Funções (se você não consegue se lembrar de um método, apenas crie uma função e pelo menos você a terá lá. Escreva algo e comece com a parte fácil.

11. Pense nas verificações de erros e em como você pode quebrar esse código. Nunca faça suposições sobre a entrada. Suponha que as pessoas estejam tentando quebrar seu código e que Darth Vader esteja usando sua função. Como você vai protegê-lo? Sempre verifique se há entradas falsas que você não deseja. Aqui está um truque: Comente no código, as verificações que você deseja fazer, escreva a função e diga ao entrevistador que você escreveria testes agora para fazer sua função falhar (mas você não precisará realmente escrever os testes).

12. Não use nomes ruins/confusos como i e j. Escreva um código que leia bem.

13. Teste seu código: verifique se não há parâmetros, 0, indefinido, nulo, arrays massivos, código assíncrono, etc. Pergunte ao entrevistador se podemos fazer suposições sobre o código. Você pode fazer a resposta retornar um erro? Faça furos em sua solução. Você está se repetindo?

14. Por fim, fale com o entrevistador onde você melhoraria o código. Funciona? Existem abordagens diferentes? É legível? O que você buscaria no Google para melhorar? Como o desempenho pode ser melhorado? Possivelmente: pergunte ao entrevistador qual foi a solução mais interessante que você viu para esse problema

15. Se o seu entrevistador estiver satisfeito com a solução, a entrevista geralmente termina aqui. Isso é também É comum que o entrevistador faça perguntas de extensão, como como você lidaria com o problema se toda a entrada for muito grande para caber na memória ou se a entrada chegar como um fluxo.
Esta é uma pergunta de acompanhamento comum no Google, onde eles se preocupam muito com a escala. A resposta geralmente é uma abordagem de dividir e conquistar, executar o processamento distribuído dos dados e apenas ler certos pedaços da entrada do disco na memória, gravar a saída de volta no disco e combiná-los posteriormente.

Boa lista de verificação de código:

[]Funciona
[]Bom uso de estruturas de dados
[]Reutilização de código/ Não se repita []Modular
- torna o código mais legível, sustentável e testável []Menos que O(N^2). Queremos
evitar loops aninhados, se pudermos, pois são caros. Dois loops separados são melhores do que 2 loops aninhados
[]Baixa Complexidade de Espaço --> A recursão pode
causar estouro de pilha, a cópia de arrays grandes pode exceder a memória da máquina

Heurísticas para responder à pergunta:

[]Mapas de hash geralmente são a resposta para melhorar a complexidade de
tempo []Se for uma matriz classificada, use a árvore binária para obter O(log N). Dividir e conquistar - Divida um conjunto de dados em partes menores e, em seguida, repita um processo com um subconjunto de dados. A pesquisa binária é um ótimo exemplo
disso []Tente Classificar sua
entrada []Tabelas de hash e informações pré-computadas (ou seja, classificadas) são algumas das melhores maneiras de otimizar seu código
[]Observe a compensação Tempo x Espaço. Às vezes, armazenar estado extra na memória pode ajudar o tempo.
(Tempo de execução)
[]Se o entrevistador está lhe dando conselhos/dicas/sugestões. Siga-os
[]Compensações de espaço-tempo: Hastables geralmente resolvem isso muitas vezes. Você ocupa mais espaço, mas consegue uma otimização de tempo no processo. Na programação, muitas vezes você pode usar um pouco mais de espaço para obter um tempo mais rápido

---

E lembre-se sempre: comunique seu processo de pensamento o máximo possível. Não se preocupe em terminar rápido. Cada parte da entrevista é importante.