Complexidade - Matheus Gonzaga

1. Um dos pontos importantes a serem considerados ao se tomar a decisão de utilizar um algoritmo para a solução de um problema, é a sua correção. Um algoritmo incorreto não produzirá os resultados esperados e não pode ser utilizado. Outro ponto importante a ser considerado, quando temos mais de uma opção de algoritmos tradicionais para resolver um problema ou quando podemos projetar um algoritmo para um problema de diversas maneiras, é a questão de sua eficiência.

a) O que significa a eficiência de tempo de um algoritmo.

A eficiência de tempo de um algoritmo refere-se à quantidade de tempo que o algoritmo leva para executar, medida em termos de operações ou passos de execução em relação ao tamanho da entrada. Em outras palavras, é uma medida da rapidez com que um algoritmo resolve um problema em função do tamanho do conjunto de dados de entrada.

b) Quais as grandezas físicas que influenciam a eficiência de tempo de um algoritmo na prática.

O número de operações executadas, o uso de memória e detalhes específicos do hardware.

c) Quais dessas grandezas nós realmente utilizamos para expressar a eficiência de tempo de um algoritmo, utilizando-as para o nosso modelo de computação?

As grandezas utilizadas são o número de operações executadas e o uso de memória por parte do algoritmo.

d) De quais nós abstraímos e por quê?

Se abstrairmos dos detalhes específicos do hardware, como a velocidade exata da CPU ou a arquitetura específica do sistema.

e) Que notação utilizamos na prática para expressar a complexidade de tempo de um algoritmo?

É utilizada a notação Big O para expressar a complexidade de tempo de um algoritmo, pois verifica sem tanta rigidez o máximo de recursos que o algoritmo vai utilizar.

2. Uma das possíveis formas de se descrever a complexidade de um algoritmo é a chamada Notação-Big-Oh. Explique o que você entendeu por esta definição.

A notação Big-O descreve a ordem de grandeza da complexidade de um algoritmo, ignorando fatores constantes e termos de menor ordem. Ela é usada para dar uma visão geral do desempenho relativo de algoritmos para conjuntos de dados grandes. A notação Big-O é representada por O(f(n)), onde "f(n)" é uma função que descreve a taxa de crescimento do algoritmo em relação ao tamanho da entrada "n".

3. Calcule, passo a passo, a complexidade do algoritmo abaixo.

O algoritmo em questão fornece uma prova convincente de que, não importando os tipos de dados ou a quantidade de memória que consome, sua complexidade de tempo está sempre, no mínimo, em $O(n^2)$ e, no máximo, em $O(n^2 * 2n)$. A fundamentação dessa prova reside no emprego de dois loops aninhados, destinados a examinar todas as possíveis modificações nos elementos da matriz ***x***. A complexidade de tempo é definida pela

quantidade de instruções realizadas no cenário mais desfavorável. O algoritmo contém dois loops aninhados. O primeiro loop, indexado por ***i***, executa ***n*** vezes. Já o segundo loop, indexado por ***j***, é condicional ao valor de ***i***. Se ***i*** for ímpar, o segundo loop é executado ***2n*** vezes. Se for par, o segundo loop executa apenas ***n*** vezes. Dado que ***n*** pode crescer indefinidamente, a complexidade de tempo ***O(n²)*** representa o pior cenário. Enquanto isso, a complexidade de tempo ***O(n² * 2^n)*** representa o melhor cenário, onde o segundo loop é sempre executado 2n vezes. Portanto, a complexidade de tempo deste algoritmo é efetivamente ***O(n²)***.

4. No .cpp.