Lista de Revisão/Alinhamento – FPAA

Data: 09/08/2022

Aluno: Lucas Lage e Silva

1) Primeiramente, complexidade de um algoritmo está relacionado ao custo computacional necessário para a sua execução, ou seja, um algoritmo é classificado com complexidade alta caso ele realize uma grande quantidade de operações. Em segundo lugar, o custo de um algoritmo está ligado a execução desse algoritmo, de modo a medir qual vai ser a dificuldade de processamento baseado na entrada dada.

2) Quando se diz sobre análise de complexidade de um algoritmo existem 3 parâmetros de medidas para se entender o funcionamento de um algoritmo em diferentes situações. O ‘melhor caso’, ‘pior caso’ e ‘caso médio’ representam valores relacionados ao custo computacional necessário para execução de um algoritmo em casos extremos e no caso mais comum de entradas. Para exemplificar o uso desses parâmetros irei utilizar o algoritmo de busca binária em uma lista. Ex: Dada uma lista com itens aleatórios, gostaria de encontrar um item específico utilizando o algoritmo de busca binária(divisão e conquista). Considerando que a lista de elementos está ordenada, ao analisarmos o algoritmo é possível perceber que, por ele buscar o objeto que está na metade do vetor atual existem 3 possíveis casos:

• O valor buscado estiver na posição do meio → Melhor caso → O(1)
• O valor buscado não está no vetor → Pior caso → O(log n)
• O valor está no vetor, mas não na posição do meio → Caso médio → O(log n) Desse modo podemos entender que caso você queira encontrar um item em um vetor utilizando abusca binária, a complexidade do algoritmo será da ordem de O(1) ou O(log n).

3)

a) O algoritmo de ordenação por seleção parte da ideia de que para ordenar um conjunto aleatório de valores ele irá percorrer todos os valores da lista até encontrar o menor deles, esse valor será colocado na primeira posição. Após isso ele reiniciará o processo percorrendo agora os n-1 elementos da fila para encontrar o menor entre eles e colocar na segunda posição e fará e processo continuamente até chegar na última posição, o que fará com que o vetor esteja ordenado. A partir do funcionamento descrito acima podemos perceber que não há melhor caso, pior caso ou caso médio visto que qualquer possibilidade de vetor vai resultar na realização de duas iterações entre todas as posições do vetor, o que vai resultar de uma complexidade única de O(n2). Desse modo, a complexidade desse algoritmo é percebida de acordo com as entradas logo que, por mais que a complexidade seja a mesma para qualquer vetor, quanto maior o número de elementos maior vai ser o custo de execução desse algoritmo visto que o crescimento de custo é quadrático segundo número de elementos, ou seja, caso haja um número pequeno de elementos o algoritmo será executado rapidamente porem com a adição de mais elementos ao vetor ele pode se tornar extremamente lento.

b) O algoritmo de ‘quick sort’ parte da ideia de sempre particionar um array para realizar sua ordenação através de um ‘pivô’ sendo um dos elementos do array e que será utilizado para realizar comparações. O objetivo desse algoritmo é recursivamente colocar todos os valores menores do que o pivô à esquerda dele e os maiores a direita, de modo que ao final do processo recursivo o array esteja devidamente ordenado. Para isso, inicialmente o pivô é colocado no final do array e se procura o primeiro valor menor do que o pivô(percorrendo o array sequencialmente) e o primeiro valor menor do que o pivô (percorrendo o arrray inversamente), caso o índex do primeiro não for maior que do segundo eles trocarão de posição e esse processo será realizado repetidamente. Caso o índex do primeiro elemento maior do que o pivô seja maior que o index do primeiro elemento menor a repetição irá parar e o elemento maior irá trocar de posição com o pivô, sendo que após isso, recursivamente, será escolhido um novo pivô e o processo recomeçará. Análise de complexidade:

- Quando o pivô é o maior ou menor elemento da lista → O(n2) → Pior caso
- Quando se produz 2 sub listas com tamanhos n/2 e (n/2) −1 → O(n*log n) → melhor caso
- Entrada diferente das condições anteriores → O(n * log n) → Caso médio

O tamanho do array de entrada com certeza é um importante elemento para a definição do custo atrelado a algoritmo de quick sort, porem, a função do pivô é essencial na redistribuição dos sub arrays na qual será executada de forma recursiva o algoritmo, ou seja, a escolha de um bom pivô, evitando escolha do maior ou menor valor do array, pode reduzir drasticamente o custo computacional desse algoritmo (simplesmente reduzindo a curva de crescimento dele de um algoritmo quadrático para um n * log n).