INSTITUTO FEDERAL DE SÃO PAULO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDA MARTINS DA SILVA

JOÃO THIAGO DA SILVA SOUZA

3SUM

Projeto 01 - Projeto e Análise de Algoritmos

São João da Boa Vista/SP

Funcionamento

1.1 *Three Sum*: Força bruta

O algoritmo de *Three Sum* utiliza o paradigma da força bruta, ou seja, itera por todo o *array* checando todas as possíveis combinações de números presentes no *array* fornecido que satisfaçam a condição de que a soma da tripla resulte em 0.

Para isso, ele possui três *loops* de repetição do tipo *for* aninhados, resultando em uma complexidade de $O(n^3)$. O *loop* mais externo vai iniciar sua iteração declarando uma variável *i* partindo da posição 0 do *array* e a cada laço incrementando *i*, finalizando ao chegar na antepenúltima posição do *array* (n-2).

O próximo *loop* aninhado declara e utiliza uma variável denominada *j*, que partirá da posição *i* + 1, já que a posição *i* já foi checada anteriormente, incrementando e finalizando na penúltima posição do *array* (n-1).

O último *loop* aninhado vai declarar e utilizar uma variável chamada *k*, que vai partir da posição *i* + 2, finalizando suas iterações na última posição do *array*. Dentro do último *loop* será onde a condição de formar uma tripla será checada (*A[i] + A[j] + A[k]* == 0), caso satisfaçam a condição, a tripla será exibida na saída e o contador da quantidade de triplas encontradas será incrementado.

1.2 *Three Sum*: Melhorado

O algoritmo melhorado para resolver o problema *Three Sum* utiliza, além da própria função *Three Sum*, as funções *Merge Sort* e Busca Binária.

A função *threeSumMelhorado* inicialmente chama a função *mergeSort* para ordenar o *array* fornecido pelo usuário, que possui complexidade aproximada de n*lg n. Em seguida, a função ImprimeArray, que consiste em um *loop* que percorre e imprime o *array* de forma ordenada (complexidade de n).

Após finalizar o *loop* anterior, há a invocação de um novo *loop*, que usa a variável *i*, iniciando em 0 e para no antepenúltimo elemento do *array* ordenado. Dentro desse *loop*, há

um novo, que para cada elemento percorrido no *loop* externo, este será chamado, iterando do elemento *i* + 1, até o penúltimo elemento (n - 1).

Logo em sequência, a função buscaBinaria (de complexidade lg(n)) é chamada, ela calcula o valor a ser encontrado através da operação ((*A[i] + A[j]*) * -1) e o índice encontrado por ela será armazenado em uma variável chamada "indice".

Após realizar a busca por esse valor, caso o índice for diferente de -1, significa que uma tripla foi encontrada, sendo ela composta pelos elementos *A[i]*, *A[j]* e *A[indice]*, logo ocorre a impressão da tripla e o incremento da quantidade de triplas encontradas. Caso o índice seja igual a -1, o laço de repetição continua para checar os próximos elementos do *array*.

Complexidade dos algoritmos

2.1 *Three Sum*: Força bruta

A complexidade do algoritmo *Three Sum* que utiliza o paradigma da força bruta é **T(n) = n^3**, aproximadamente. Sua complexidade é **n^3** por conta de seus três laços de repetição aninhados.

2.2 *Three Sum*: Melhorado

Para o algoritmo *Three Sum* melhorado, é necessário considerar as funções *Merge Sort* e Busca Binária para sua análise.

Começando pelo *Mergesort*, que é chamado apenas uma vez na função, ele possui uma complexidade **T(n) = O(n*lg n)**. Após o *mergeSort* há a impressão dos elementos do *array* (complexidade **T(n) = n**).

Sabendo que o algoritmo Busca Binária é de uma complexidade **T(n) = O(lg n)**: após a chamada do *mergeSort*, há dois *loops* aninhados, o que resulta em uma complexidade de **n^2**. Sendo assim, juntando as informações, a complexidade dos *loops*, que no *loop* mais interno há chamada da busca binária, há complexidade **T(n) = O(n^2 * lg n)**.

Concluindo, a complexidade do algoritmo melhorado é **$T(n) = n + (n * \lg n) + (n^2 * \lg n)$.**