Trabalho Prático 1: O problema da mediçãoo de Rick Sanchez
Breno de Castro Pimenta
RA: 2017114809

Introdução: O problema pode ser resumido como uma uma equação diofantina de várias variáveis, sendo elas inteiras, onde as constantes seriam os frascos que o Rick Sanchez possui (mais especificamente o volume de cada um), o resultado seria a medida que se busca alcançar e as variáveis seriam a quantidade de movimentos que devem ser feitos. A fórmula abaixo representa a equação, onde os F's representam os volumes dos frascos, os M's representam os movimentos necessários e o VF a medida do volume final que busca-se alcançar.

F1*M1 + F2*M2 + ... +Fn*Mn = Medida

No caso para encontrar o número de movimentos há que somar o módulo das variáveis, pois há que levar em conta que o uso de um frasco para retirar um valor é contado como um movimento, mas deve ser considerado na equação como valor negativo.

Implementação: Há a possibilidade da solução de equações diofantinas através do algoritmo de Euclides, no entanto a equação desse algoritmo pode escalar em quantidade de frascos, o que é equivalente a quantidade de variáveis, por esse motivo a solução através do algoritmo de Euclides traria uma complexidade temporal e espacial altíssima.

Optou-se por uma solução customizada utilizando como base lista duplamente encadeada. Houve a implementação de duas classes: Lista e CelulaEspecial, que permitem utilizar essa estrutura. Essa lista possui métodos que fornecem um auxilio direto à solução do problema como addValor e removerValor, pois os frascos do Rick são mantidos em tempo de execução como uma lista, dessa forma tornando simples a remoção e adição dos frascos.

Para a realização dos cálculos foi criada uma outra classe a ControleCamadas, onde camadas são listas que representam os valores calculados a partir de uma certa quantidade de movimentações com os frascos.

O consumo de memória demonstrou ser um ponto crítico para o projeto, para otimizar esse processo duas medidas foram tomadas:

1. Optou-se por não armazenar as camadas conforme são calculadas. Armazena-se a camada atual até a próxima ser calculada. E assim que a camada correspondente é verificada como a resposta a mesma é deletada.
2. Antes de adicionar um novo valor durante a construção de uma nova camada verifica-se se esse valor já foi adicionado, evitando-se assim valores repetidos.
3. Ao adicionar novos valores à nova camada que está sendo calculada, utiliza-se uma lógica de soma e subtração entre a camada anterior e os frascos do Rick de forma a não criar futuros valores repetidos. Ou seja, caso algum dos dois valores seja maior do que a medida que se busca, deve ser realizado uma subtração e caso ambos os valores sejam menores, deve se realizar uma soma. Essa lógica foi implementada com if's de forma a aumentar a complexidade do algoritmo apenas de forma constante.

<u>Instruções de compilação e execução:</u>

- **make**: apenas compila
- **make run**: compila e executa o programa
- **make test**: executa testes de entrada e saída localizados na pasta 'tests', na raiz do projeto.
- **make test_code**: compila e executa testes referentes ao código, que se encontram na pasta 'src/testes'.
- **make clean**: deleta arquivos compilados.
- **make clean_all**: deleta arquivos compilados e executáveis.

Análise de complexidade: Será tomado 'n' como valor referente ao número de entradas. Ao analisar a complexidade do algoritmo pode-se dividi-lo em duas partes, sendo a primeira a gestão da Camada Base (lista) referente aos frascos do Rick. Essa camada possui duas ações básicas: adição, cada uma a custo O(1) e remoção, que no pior caso deve percorrer a lista inteira antes de remover um valor. A remoção total possui custo igual a 'n + (n-1) + (n-2) + ... + 1', logo tendo n adições e n remoções, terá-se uma complexidade no pior caso de '$n+n^2$', ou seja $O(n^2)$.

Já a segunda parte é a correspondente ao ControleCamadas, mais especificamente à construção de novas camadas. A construção de novas camadas também pode ser dividida em dois processos, o primeiro é quantas operações são necessárias para essa construção. Suponha que a camada anterior possua y valores, para cada valor desse será realizado n operações, ou seja essa parte terá uma complexidade de $O(n*y)$.

O outro processo é responsável pela quantidade de valores que haverá na próxima camada, correspondente ao valor y do processo anterior. Ao gerar novas camadas o algoritmo não armazena valores repitidos, portanto partindo da primeira camada que possui n valores, serão gerados n valores da nova camada usando o primeiro valor da camada anterior. Para o próximo valor da camada antiga será gerado n-1 valores para a nova camada e assim sucessivamente (lembrando que está sendo considerado o pior caso, onde operações que podem gerar números diferentes, gerarão números diferentes). Dessa forma a segunda camada terá no pior caso $[n*(n+1)]/2$ valores. Quando for construído a próxima camada ela terá $[n^2*(n+1)]/2$ valores e as próximas camadas seguirão o mesmo padrão.

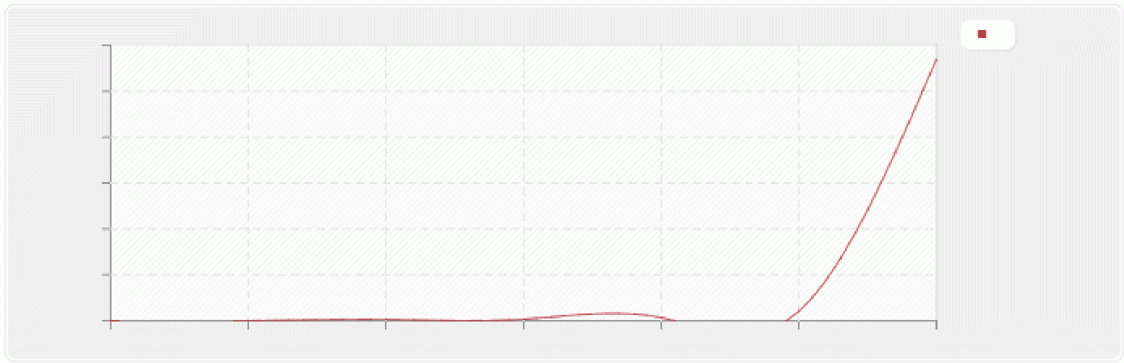
Portanto a complexidade correspondente ao cálculo de uma camada específica será:

$$\left[\left(n^{QC}-1\right)\cdot\frac{(n+1)}{2}\right]\cdot n$$

Onde 'QC' representa qual a camada específica, que representa também o número de movimentos já realizados pelo Rick e que pode ser representada como $O(n^{QC+1})$.

A questão necessária para concluir a complexidade da construção, é a quantidade de camadas que serão necessárias calcular durante a execução do problema. A principio essa questão depende da entrada específica e não de sua quantidade necessariamente, porém o cálculo dessa especifidade foge ao escopo desse projeto, portanto será utilizado como base a quantidade da entrada ‘n’. Logo a complexidade da construção como um todo pode ser descrita como $O(n^{n+1})$.

Logo a complexidade total do algoritmo dependerá da complexidade da construção, sendo portanto $O(n^{n+1})$. A figura abaixo demonstra a evolução do tempo de execução em milisegundos (eixo y) por quantidade de camadas construídas, e como pode ser visto a complexidade temporal é claramente exponencial.



Conclusão: A primeira leitura do problema gerou uma proposta inicial de solução muito mais complexa do que a implementada. Essa diminuição de complexidade é devida ao potencial existente na utilização de estruturas de dados como por exemplo uma Lista, a aplicada neste projeto, que apenas com a adição de mais uma camada para a gestão dessa estrutura foi possível alcançar uma complexidade muito menor do que a proposta inicialmente.

Esse trabalho apresentou também um desafio quanto a complexidade espacial e a necessidade de uma gestão de memória para solucioná-lo. Porém devido a limitação de tempo para o desenvolvimento desse projeto não foi possível alcançar o padrão esperado.

Bibliografia:

ZIVIANI, N. Projeto de Algoritmos com implementações em pascal e c. 3ª edição. Cengage Learning.

CORMEN, T. H. Et al. Algoritmos: Teoria e Prática. 3a edição. Elsevier, 2012

C++ reference. Disponível em: < https://en.cppreference.com/w/>. Acesso em: 03 out. 2019.