Universidade Federal de Minas Gerais

Instituto de Ciências Exatas

Departamento de Ciências da Computação

Documentação – TP Rick And Morty

**Nome:** BRUNO NICOLAU MACHADO

**Disciplina:** Estrutura de Dados

**Matrícula:** 2018014166

Belo Horizonte

<2019 / 2

**RESUMO.** Rick, um dos maiores cientistas da terra, possui um laboratório no qual ele faz seus experimentos. Em alguns experimentos Rick necessita de medidas de volume precisa, porém ele só conta com frascos de volume definido e não possui medidor variável de volume.

O Objetivo desse trabalho é ajudá-lo a otimizar suas medições de volume, o algoritmo deve retornar o menor número de operações possíveis(utilizando os recipientes de volume pré - definido para chegar a um valor de volume desejado). Afim de alcançar o objetivo do trabalho algumas estruturas de dados foram implementadas, sendo uma delas uma lista duplamente encadeada circular com sentinela e uma fila duplamente encadeada circular com sentinela.

A saída otimiza o trabalho de Rick e mostra a relação de duas listas interagindo em baixo nível, o problema poderia ser resolvido de maneira recursiva, mas por opção de maior clareza essa opção foi descartada e o uso de alguns loops foi implementado.

**1.Introdução**

Apesar o problema ser bastante relevante o verdadeiro objetivo desse trabalho é mostrar a implementação das estruturas de dados utilizadas e o relacionamento entre elas. Como implementas listas e filas? Como percorrer essas estruturas acessando cada dado sequencialmente?

Para isso a realidade de Rick deve ser reproduzida, o programa implementado deve conter as seguintes características:  ler a medida dos recipientes do laboratório (o número de recipientes máximo não é conhecido),ler a informação de que um recipiente foi quebrado, ler a medição requerida por Rick, Informar o número mínimo de operações necessárias para conseguir a medida exata desejada.

A seção 2 irá abordar os detalhes da implementação, a explicação detalhada de alguns tipos classes, structs, métodos e algumas análises de complexidade. Já a seção 3 irá descrever a máquina utilizada bem como instruções de compilação e execução. Já as duas últimas seções trataram de análise de instancias e resultados observados.

**2. Implementação**

Nesta seção algumas implementações serão comentadas evitando a redundância de comentários já escritos no código. Incialmente o programa faz várias leituras de entrada até que o usuário aperte Ctrl + D em um terminal Linux, dessa maneira os cálculos são feitos para todas as entradas antes mesmo de uma saída ser impressa, por isso o uso de uma estrutura de dados do tipo fila se fez necessário , pois ao final nescessita-se de uma impressão que saia na ordem dos comandos de entrada.

**2.1 Tipo Classe Fila**

**Structs No() -**  Antes da definição da classe o tipo No foi definido, contendo um valor do tipo inteiro e dois ponteiros, e dois construtores, um instanciando um tipo No com valores nulos e o outro um tipo No com ponteiros e valor definido. A ideia desse tipo No é servir como ponteiros de navegação para o tipo classe Fila que viria logo em seguida.

A classe possui os seguintes atributos privados : fim\_(tipo No), tamanho\_(tipo inteiro), o fim\_ funciona como um sentinela sendo um ponteiro que liga o fim da fila ao início não possuindo valor inteiro instanciado Por ser um fila o método de inserir insere o elemento no fim e realoca os ponteiros. Vale destacar que o destrutor desaloca elemento por elemento e depois apaga o sentinela ao invés de desalocar a lista inteira de uma só vez, isso ocorre pois é preciso imprimir o valor de cada elemento antes de desalocar a memória, e para evitar um loop que percorreria a fila e fosse imprimindo um elemento por e isso é feito no momento de deslocação. A complexidade da maioria dos métodos é O(1). Como não há muitas aplicações para essa estrutura não há motivo para ela não ser a mais simples possível.

**2.2 Tipo Classe ListaMedicoes**

Nessa classe a mesma lógica de marcação através de ponteiros de tipos No é utilizada, cronologicamente essa classe foi implementada primeiro e a lógica de tipo No foi aproveitada para no tipo Fila.

Os atributos privados são os mesmos empregados e descritos no tipo Fila, porém a variedade de métodos é bem maior nesse caso. Essa estrutura foi desenvolvida a fim de armazenar os recipientes disponíveis para Rick em seu laboratório, armazenar os subconjuntos de combinações de um mesmo número de operações e armazenar os volumes disponíveis.

A classe começar com dois construtores: um padrão e um que cria a cópia de uma outra lista, a ideia é utilizar essa cópia para instanciar listas com elementos repetidos e listas com os mesmos elementos, porém únicos. A função getSentinela() tem fundamental importância na implementação da lógica, pois ela permite que dentro do tipo Classe lista de operações um lista do tipo ListaMedicoes possa ser percorrida, proporcionando uma maior eficiência e clareza no que diz respeito à lógica . A função inserir, de complexidade O(i), insere o elemento de maneira ordenada na lista, tratando inicialmente os casos de lista vazia, maior, menor a fim de obter uma otimização mesmo que mínima da função, a ideia da ordenação surge para facilitar a organização e o acesso aos dados por parte do programa. As funções privadas, inserirInicio() e inserirFinal() são sub-métodos de inserir, pois só existem para complementar essa função.

A função retirar, retira um elemento existente na lista independente da posição, sua complexidade é o(size\_), além disso ele reorganiza os ponteiros a fim de manter a lista encadeada. Vale ressaltar que o destrutor trabalha em conjunto com dois submétodos removerPrimeiro() e limparLista(). Dessa forma é possível limpar a lista sem desalocar a memória do sentinela e isso é extremamente útil para a implementação do problema.

Por fim

**Tipo Classe ListaOperacoes**

Antes da declaração da classe em si há duas declarações de structs, um tipo No exatamente implementado como os dois últimos descritos, para percorrer as lista do tipo ListaMedicoes que são utilizadas dentro da estrutura da LsitaOperacoes, e outro structs do tipo apontador, que agora possui 2 valores o inteiros por posição, um desses valores corresponde ao número de operações necessárias, o outro um valor de volume possível. A ideia dessa estrutura foi armazenar todas as possibilidades de volumes que poderiam ser criados em x operações, para que o número mínimo fosse retornado a partir de uma pesquisa que revelasse se o valor pesquisado estava ou não armazenado na lista, caso eles estivesse armazenado o programa retornaria o atributo operações do tipo inteiro declarado no escopo do apontador.