O problema da medição de Rick Sanchez

Felipe Matheus Guimarães dos Santos[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

felipe.guimaraes@dcc.ufmg.br

1. Introdução

A situação problema do presente trabalho é a medição de substâncias de Rick Sanchez. Este precisa fazer medições com o menor número de operações - apenas soma e subtração - usando os frascos que seu neto Morty encontra pelo laboratório. Além dos frascos encontrados, Rick pode se atrapalhar e quebrar frascos.

Para resolver esse problema, serão usadas as estruturas de dados vistas em aula: fila e lista encadeada. A lista será usada para armazenar os frascos encontrados por Morty e remover os frascos quebrados por Rick. Por outro lado, a fila será utilizada para fazer a contagem de frascos para a medição, armazenando a quantidade de operações e a quantidade de ml.

2. Implementação

Para resolver o problema foram usadas duas estruturas de dados: fila, pois a inserção é feita no final e a remoção no início; e lista encadeada, pois precisa-se remover elementos em quaisquer posições.

O compilador utilizado foi o MinGW (versão 8.2.0), utilizando a versão 17 do c++: *g++ -std=c++17 [arquivos] -o [executável]*.

O programa principal roda em loop infinito, recebendo um inteiro e um caractere de entrada, separados por enter ou espaço. O único retorno do programa é o número mínimo de operações para fazer uma medição, o qual é impresso na saída padrão. O loop do programa principal é interrompido por EOF ou manualmente: inserindo um frasco de tamanho 0 (zero) e/ou a letra 'e', separados por espaço, ex: 0 e, 0 i, 12 e.

O nome da variável *int* foi trocado para '*TipoItem*' quando se tratava de atributos

da lista, apenas para o código ficar mais geral, porém, não há necessidade de tê-lo feito, pois os dados de entrada sempre serão inteiros.

2.1. Fila

A implementação da fila foi baseada nos slides disponibilizados pelo professor da disciplina Estrutura de Dados (EDs) Luiz Chaimowicz e modificados com a orientação a objetos aprendida na disciplina Programação e Desenvolvimento de Software 2 (PDS2) com o professor Flávio Figueiredo.

Foram usadas duas estruturas para a criação da fila: o struct *Medida*, o qual armazena um inteiro para a quantidade de medições e outro inteiro para a quantidade de mls; e o struct *Node_f*, o qual armazena um struct *Medida* como um atributo e um ponteiro *Node_f** para o próximo nó da lista. Os métodos do TAD Fila são:

- **Construtor**: A fila foi implementada com a célula cabeça, i.e., a primeira célula nunca será removida, tampouco terá valor armazenado. Além disso, também é armazenada a quantidade de números inseridos, para facilitar alguns métodos. O construtor da fila é igual ao da lista encadeada, mudando apenas o tipo de ponteiro – de *Node_t* para *Node_f*.
- **Destrutor**: Padrão.
- **Incluir elemento**: Os elementos na fila são sempre incluídos no fim da estrutura. A única diferença para o método homônimo da lista é o tipo de atributo.
- **Remover elemento**: A remoção de elementos da fila é diferente da lista, o primeiro elemento após a célula cabeça é removido por padrão.
- **Limpar**: Este método é necessário após realizar a medição, pois não se pode, em todos os casos, aproveitar o que foi feito antes. Logo, para evitar problemas, é necessário apagar toda a fila criada a cada medição.
- **Comparar**: Método criado para reduzir a quantidade de elementos na lista e, consequentemente, o número de operações repetidades. Está dentro do loop de medição da lista e recebe como parâmetro a soma - ou subtração - do primeiro elemento da fila com todos frascos, um de cada vez. Se esse valor já estiver presente na fila, retornará falso e não será armazenado novamente, evitando operações redundantes. Caso contrário, retorna 1 e o valor será armazenado na

fila para futuras operações.

- **Imprime**: Método criado para teste de inserção e remoção. Não necessário para o TP.

2.2. Lista Encadeada

A lista, assim como a fila, também foi baseada nos slides e modificada com orientação a objetos aprendida na displina PDS2.

A estrutura criada para a implementação da lista é um struct chamado *Node_t*, o qual armazena um valor inteiro (*TipoItem*) de ml e um ponteiro *Node_t** para o próximo nó da lista. Os métodos do TAD ListaSimples são:

- **Construtor**: A lista também foi implementada com a célula cabeça. Além disso, também é armazenada a quantidade de números inseridos, para facilitar alguns métodos.
- **Destrutor**: Padrão.
- **Incluir elemento**: A inserção de elementos na lista ocorre apenas no fim, pois a ordem dos elementos inseridos não é relevante para o problema.
- **Remover elemento**: A remoção de elementos é mais complexa que a inserção, pois, precisa-se saber a posição do elemento escolhido pelo usuário, o qual é passado por parâmetro para o método. Para tal, a lista é iterada usando dois ponteiros *Node_t **, um para a posição *anterior* e outro para a posição *seguinte*. O ponteiro '*seguinte*' compara o valor passado para o método com o atributo do struct. Se o valor não for igual, passa-se para o próximo nó da lista, indo até o fim da lista. Porém, se o valor passado para o método for igual ao atributo do struct, o ponteiro para o próximo nó de '*anterior*' aponta para o próximo de '*seguinte*' e deleta o ponteiro '*seguinte*' da lista.
- **Medição**: Para fazer a medição, primeiro compara-se o valor passado por parâmetro com a lista de frascos. Caso algum dos frascos for igual à medição, retorna o número de operações. Senão, os valores dos frascos são adicionados numa fila para que sejam feitas as operações. O número de operações começa com 1.

  O primeiro elemento da fila é escolhido para as operações e é instantaneamente

removido da fila. Esse valor é utilizado como índice e as operações são feitas dele para a lista de frascos. A cada operação - soma ou subtração - é somado um no atributo '*_item.qtde*', i.e., o número de operações aumenta. Se o valor for igual ao valor requisitado, retorna essa quantidade de operações. Senão, o valor do índice somado e subtraído dos frascos da lista, com suas respectivas quantidades de operações, é adicionado na fila para uma próxima iteração.

Esse loop é repetido até que se encontre o valor das operações com os frascos disponíveis.

Não foi criado um validador, pois todas as entradas de teste serão válidas.

- **Imprime**: Método criado apenas para testar a implementação da lista. Imprime todos os elementos da lista separados por um tab (\t) pulando de nó em nó até encontrar o fim da lista, ou seja, um ponteiro que aponta para nulo (*nullptr*). Não necessário para o TP.

2.3. Makefile

Para compilar e rodar o programa foi gerado um Makefile. Este arquivo facilita a compilação, pois elimina a necessidade de dar os comandos do g++ no terminal, apenas digitamos *make test* para rodar os testes ou *make run* para rodar o programa.

O Makefile já havia sido disponibilizado pelo monitor Matheus Diniz, portanto, apenas foram alterados os nomes dos arquivos *.cpp* e *.o*.

3. Instruções de compilação e execução

Para compilar e executar o programa usa-se o Makefile. Para tal, abra o Terminal e entre no diretório onde se encontra o programa, i.e., *.../felipe_guimaraes/src*. Dentro do diretório, dê o comando *make test* para rodar os testes ou make run para rodar o programa. Há um arquivo *README.txt* com as especificações.

4. Análise de complexidade

Os métodos do TAD **Fila** *incluir_elemento, remover_elemento* têm custo constante ($\theta(1)$). Os métodos *imprime* e *limpar* têm custo linear ($\theta(n)$). Já o método *comparar*, temos no melhor caso comportamento constante ($\Omega(1)$), quando o número passado como atributo é igual ao primeiro elemento da fila, e para o pior caso, quando

não está na fila, comportamento linear (O(n)). Todos os métodos da fila têm complexidade de espaço constante (θ(n)).

O método do TAD **Lista** *incluir_elemento* tem custo constante (θ(1)). Os métodos *imprime* e *limpar* têm custo linear (θ(n)).

O método *remover_elemento* tem custo linear no pior caso (O(n)) e custo constante no melhor caso (Ω(1)): o pior caso acontece quando o item a ser removido está no último nó da lista; e o melhor caso acontece quando o item a ser removido está na primeira posição.

E por fim, o método *medicao* é de ordem quadrática no pior caso, contando comparações como a operação relevante ($O(n^2)$, onde n é o número de frascos) e de ordem linear no melhor caso (Ω(n)): o melhor caso acontece quando o número medido é um frasco registrado; e o pior caso acontece quando o número de combinações de frascos é muito grande.

Dos métodos da lista, apenas o *medicao* não tem complexidade de espaço constante. A complexidade é exponencial ($O(2^n)$, considerando n o número de vezes que o loop é feito), pois, a cada iteração do loop, são adicionados dois novos valores na fila.

5. Conclusão

Portanto, para concluir o presente trabalho foram usadas duas estruturas de dados: lista encadeada e fila. A utilização de cada uma dessas EDs foi pensada de acordo com as especificações do professor e o auxílio do monitor Allison Svaigen.

A maior dificuldade do trabalho estava na iteração da fila para fazer as medições, porém, com as explicações alto-nível do monitor, ficou mais simples de ser implementada.

A lógica por trás da medição é testar todas as possíveis combinações de frascos: somar e subtrair os valores dos frascos até chegar na quantidade requisitada. Para isso, cada medição inconclusiva é adicionada na fila, com sua respectiva quantidade de operações, para ser reutilizada em outra operação posteriormente.

Logo, o trabalho cumpre o que era esperado: utilizar todo o conhecimento que foi acumulado desde o ingresso na Universidade e o conhecimento de EDs.

O código e a documentação deste trabalho estão disponíveis no GitHub: https://github.com/FelipeGuimaraes42/ListaFrascosRick

6. Bibliografia

FIGUEIREDO, Flavio. Lista Simplesmente Encadeada. Disponível em: https://github.com/flaviovdf/programacao-2/tree/master/exemplos/aula05-listas/listaponteiros. Acesso em 20 set. 2019.

TEODOROWITSCH, Roland. Adaptação para OpenOffice.org 1.1 feita em 29 mar. 2005. Disponível em: http://www.sbc.org.br/documentos-da-sbc/summary/169-templates-para-artigos-e-capitulos-de-livros/878-modelosparapublicaodeartigos. Acesso em 26 set. 2019.