TRABALHO PRÁTICO 1: O PROBLEMA DA MEDIÇÃO DE RICK SANCHEZ

Letícia da Silva Macedo Alves < leti.ufmg@gmail.com>

Matrícula: 2018054443

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

1. Introdução

O problema descrito consiste em desenvolver um programa que auxilie o cientista Rick em suas medições com reagentes na realização de experimentos. Recipientes disponíveis no laboratório são combinados para obter diferentes volumes de acordo com quantidades desejadas de reagentes. Além disso, a qualquer momento, podem haver quebra de recipientes (não estando mais disponíveis para uso) ou pode-se encontrar recipientes perdidos pelo laboratório (entrando para a lista de recipientes disponíveis). Para que determinada medição desejada seja feita, é preciso calcular o número mínimo de operações que devem ser realizadas a fim de obter tal medida.

Na realização deste projeto, a linguagem utilizada foi C++11 e foram implementadas duas classes: Celula e Lista (encadeada). A escolha da Lista Encadeada como estrutura de dados base deve-se ao fato de que não se sabe previamente o número máximo de elementos (células) que possuirá no decorrer da execução do programa. Além disso, foi considerada adequada, atendendo perfeitamente o que se esperava da solução do problema. Para a verificação de alocação e desalocação correta de memória, foi utilizado o Valgrind.

Uma célula pode representar um recipiente em si ou um "bloco" de operação realizada com os recipientes, informando o volume parcial obtido até um dado momento e o número de operações. Cada definição se aplica em um dos dois usos da classe Lista, o que será explicitado mais adiante.

Consideremos então dois casos de lista: lista com os recipientes disponíveis no laboratório e lista que guarda operações parciais realizadas para medição.

2. Implementação

Os formatos de entrada e saída usados foram os padrões do sistema (stdin e stdout).

2.1. A classe Celula

O uso da classe Celula é distinguível em dois casos, cada qual com suas especificidades.
A classe possui os atributos: *int volume*, *int numOperacoes* e um ponteiro *Celula *proximo*, que aponta para a próxima Celula da lista. O atributo *int volume* guarda a capacidade do recipiente.

No caso da lista contendo os recipientes disponíveis para uso, suas células utilizam apenas os atributos: *volume* e *proximo*, sendo *numOperacoes* inicializado sempre com 0 (valor que indica sua não utilidade neste caso).

Já para a lista que guarda o passo-a-passo das operações realizadas com os recipientes, as células utilizam de todos os três atributos. Assim, *int volume* guarda a medição obtida até o momento (através da combinação da capacidade dos recipientes) e *int numOperacoes* indica quantas operações foram necessárias para obter aquela dada combinação.

Uma observação é que também poderia ser usada uma herança para cobrir esses dois casos, porém foi considerado um esforço não necessário, uma vez que uma implementação de outra se diferiria em apenas um atributo int (*numOperacoes*). A maneira como foi implementado o problema não afeta sua solução, que segue pertinente e coerente e de codificação simplificada. Os requisitos da especificação foram atendidos.

2.1.1. Métodos da classe Celula

A classe possui três métodos, cada um deles é um tipo de construtor.

Celula(), sem parâmetros, é utilizado no momento de criar uma nova lista. Para a lista *recipientesDisponiveis*, será a cabeça da lista, sendo seus valores inicializados da seguinte forma:

```
this->volume = 0;
this->numOperacoes = 0;
this->proximo = nullptr;
```

Celula(int volume) é usado para adicionar um recipiente na lista *recipientesDisponiveis* com determinada capacidade (volume). Assim,

```
this->volume = volume;
this->numOperacoes = 0;
this->proximo = nullptr;
```

Já o construtor *Celula(int volume, int numOperacoes)* é responsável por adicionar um "bloco" (isto é, uma célula) que representa uma operação parcial na lista que calcula o mínimo de operações.

```
this->volume = volume;
this->numOperacoes = numOperacoes;
this->proximo = nullptr;
```

Em todos os casos de inicialização, *proximo* recebe *nullptr* porque a inserção dos recipientes é feita sempre ao final da lista. Logo, sendo tal elemento o último, seu *proximo* deve apontar para *nullptr*.

2.2. A classe Lista

Com base na natureza do problema, optou-se por utilizar a Estrutura de Dados do tipo lista encadeada. Foi implementada então uma classe Lista (que será composta por células). Tal classe possui apenas dois atributos: os ponteiros *primeiro* e *ultimo* para o tipo Celula. Esses apontam, respectivamente, para a primeira posição da lista e para a última.

O problema pôde ser resolvido utilizando duas declarações distintas de Lista, como citado anteriormente. Uma lista (que é declarada na *main*), contém os recipientes disponíveis no laboratório para as medições. A segunda lista usada é declarada sempre que a entrada for do tipo "p" (isto é, quando se deseja realizar um medição). Essa, por sua vez, é formada então pelo "passo a passo" das operações realizadas com os recipientes disponíveis até que se atinja a quantidade desejada em ml. Os dois "tipos" de lista possuem uma pequena diferença entre as células que as compõem, o que foi explicado anteriormente.

2.2.1. Métodos da classe Lista

A principal e mais relevante função de Lista a ser citada é a *int minimoOperacoes(int volumeDesejado)*, que é responsável por retornar o inteiro que é a quantidade mínima de operações necessárias a partir do caso base (0) para se obter a medida requerida. Ou seja, usa-se as medidas que necessitam de k operações para descobrir as que precisam de k+1. Esta função é atribuída à classe Lista para seu uso pela lista *recipientesDisponiveis*, uma vez que as operações são realizadas com base em seus próprios elementos. Além disso, isso simplifica seu uso e não é necessário deixá-la avulsa em outro arquivo (o que faria com que fosse necessário passar como parâmetro, além do volume desejado, também a lista de recipientes disponíveis).

Essa função é chamada sempre que se digita "p" como entrada (isto é, se deseja fazer uma medição).

O que essa função faz é alocar uma lista (declarada como *listaOperacoes*) que guarda o passo a passo das operações realizadas, retornando o número de operações que foram feitas (que é o mínimo) quando se atinge a medição desejada.

Primeiramente é verificado se a lista de *recipientesDisponiveis* está vazia e se o volume desejado é válido (imprimindo mensagem de erro e retornando -1 se estiver vazia ou se o volume não for válido, isto é, menor do que zero).

São declarados dois ponteiros para Celula, **recipienteAtual* e **operacaoAtual*, que iteram, respectivamente, sobre as células da lista de *recipientesDisponiveis* e *listaOperacoes* por meio um laço duplo de while. Ao final do laço mais interno, **recipienteAtual* retorna ao início da lista. Há também as variáveis *int volumeTempSoma* e *int volumeTempSubtr* guardam os volumes parciais obtidos ao se somar *recipienteAtual+operacaoAtual* ou subtrair *recipienteAtual-operacaoAtual*.

Cada volume parcial encontrado por *volumeTempSoma* e *volumeTempSubtr* são comparados com o *volumeDesejado*. Se alguma comparação resultar em True, *a listaOperacoes* é desalocada e o *numOperacoes* que é a solução do problema é retornado. Caso contrário, se *volumeTempSoma* e *volumeTempSubtr* forem válidos (isto é, maiores ou iguais a zero), são inseridos na lista de operações realizadas, juntamente com o parâmetro *numOperacoes*+1, já que está se contando uma nova operação.

Além da função *minimoOperacoes*, a classe possui outros métodos, citados a seguir.

Possui um construtor e um destrutor, que desaloca cada uma de suas células. E é relevante comentar que no construtor uma primeira célula é alocada. Esta célula atua como cabeça no caso da lista com recipientes (isto é, usada apenas para facilitar manipulações e operações para listas vazias). Já no caso da lista de cálculos de operações, tal célula é de fato a primeira da lista, não atuando como cabeça, mas sim como célula que representa o caso base de operação (isto é, volume 0 com 0 número de operações realizadas).

Há também uma função *bool estaVazia()*, que retorna True se lista está vazia, uma funçao que retira um elemento da lista e duas funções *insere* que diferem nos parâmetros passados.

A *void insereElemento (int volume)* é responsável por inserir um elemento em uma lista do tipo *recipientesDisponiveis* e *void insereElemento (int numOperacoes, int volume)* insere em listas do tipo *listaOperacoes*.

Na *Void retiraElemento (int volume)*, embora não fosse necessário se preocupar em tentar retirar algum elemento que não está na lista (já que nas especificações do problema diz que deve-se considerar

que para cada operação do tipo r houve, previamente, uma do tipo i), a função verifica se a lista está vazia (imprimindo uma mensagem de erro e retornando em caso de True). Verifica-se também se o valor procurado é o imediatamente depois da cabeça. (Já que, segundo o algoritmo utilizado, para se retirar um elemento, preciso saber quem é o imediantamente anterior, uma vez que seu *proximo* será o desejado. A princípio o único caso onde já se conhece o anterior é para o elemento logo após a cabeça).

Se não cair em nenhum desses dois casos, a função procura pelo valor até o final da lista. Caso o volume seja encontrado, o primeiro recipiente com tal medida será deletado da lista. Mas se não for encontrado nenhum recipiente com tal capacidade, o método imprime uma mensagem de erro e retorna. Enfim é verificado no final da função se o valor retirado ocupava a última posição. Se este for o caso, o ponteiro *ultimo* da lista aponta para o novo último elemento (que antes era o anterior ao último, que foi retirado).

2.3. A função main

Na *main*, uma lista *recipientesDisponiveis* é alocada. A leitura do teclado é feita em um while, e, para cada operação desejada, lê-se um inteiro e um caractere. Os caracteres são "p", "i" ou "r", cada qual representando uma instrução desejada distinta. Se for o caso i, um novo recipiente é adicionado à lista *recipientesDisponiveis* com o volume igual ao inteiro lido antes de i. Se for r, o recipiente (primeiro a ser encontrado) com o volume igual ao inteiro lido é retirado da lista. Já se o caso por p, será impresso o número inteiro que é o mínimo de operações para se obter o volume representado pelo inteiro, com base nos recipientes que a lista *recipientesDisponiveis* possui. Ao sair do while (por um comando Ctrl + D), a lista é desalocada.

3. Instruções de compilação e execução

A compilação é realizada utilizando o compilador GCC. Como o projeto faz uso de Makefile, presente na parta "*src*", o processo de compilação dos códigos-fonte é resumido a um único comando "*make*". Ou seja, uma possível sequência de comandos para compilação e execução do programa em sistema Unix seria:

"$ cd src"

"$ make"

"$ ./tp1"

Através do comando "Ctrl + D", correspondente ao EOF do sistema (final da entrada), o loop do while da *main* termina e a execução é encerrada.

4. Análise de Complexidade

A complexidade do algoritmo depende diretamente do tamanho da lista de recipientes disponíveis e do volume desejado para a medição.

Para a complexidade espacial do programa, sendo que a lista *recipientesDisponiveis* alocada na *main* tem tamanho n antes de uma instrução do tipo "p", pode-se considerar que, no pior caso, terão no máximo 2n elementos na lista de operações (alocada na função *minimoOperacoes*) para cada um desses n recipientes disponíveis. Assim, se o número de operações necessário na dada medição for i, no nível i da árvore, terão sido alocada de memória somatório de (2n)^j, para j de 0 até i -1. Assim, podemos dizer que o algoritmo é O((2n)^i).Já no melhor caso, quando o volume que se deseja obter é zero, o número de operações retornados é zero. Tal operação, sendo de simples retorno, é O(1).

Como as operações de combinação de resultados e as demais do programa que não envolvem diretamente as alocações e manipulações de lista podem ser ignoradas na análise de complexidade, a complexidade temporal também é ((2n)^i)no pior caso e O(1) no melhor.

5. Conclusão

A estrutura lista encadeada mostrou-se bastante viável no processo de inserção, remoção e busca de itens. Pode-se dizer, no entanto, que a resolução do algoritmo utiliza de "força bruta", testando todas as combinações de soma e subtração de medições parciais até que se atinja o volume desejado. Dessa forma, quanto maior o tamanho do problema (tamanho n da lista de recipientes disponíveis), mais computações deverão ser realizadas, aumentado consideravelmente a alocação de memória e o tempo de resposta, o que não é muito eficiente computacionalmente para n absurdamente grande (mas isso não é considerado para fins desse trabalho).

O projeto utilizou diversos conceitos diferentes como modularização, alocação dinâmica, uso de memória, complexidade de algoritmo e estruturas de dados. Puderam então ser praticados muitos dos conceitos e técnicas vistos durante o curso.

Bibliografia

ZIVIANI, Nívio. (1999) "Projeto de Algoritmos com implementações em Pascal e C", São Paulo, Pioneira.