Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Prof. Carlos H. C. Teixeira, Luiz Chaimowicz, Olga N. Goussevskaia
Monitores: Álisson R. Svaigen e Matheus A. Diniz

Trabalho Prático 1: O problema da medição de Rick Sanchez

Valor: 10 pontos
Data de entrega: 04 de outubro de 2019

Introdução

Rick Sanchez é o mais brilhante cientista que já caminhou pela humanidade, sendo responsável pelo início da exploração do multiverso por meio de seu aparato de portais interdimensionais. Cotidianamente, Rick realiza experimentos científicos, que necessitam de medições muito cautelosas de diferentes tipos de reagentes.

Rick utiliza recipientes próprios para fazer as medições. No entanto, as medições variam muito, de modo que Rick precise combinar diferentes recipientes pra obter a quantidade desejada. Um outro detalhe é que Rick é muito desastrado, e ele quebra, com certa frequência, seus recipientes. Além disso, seu neto Morty (que o ajuda nas medições) encontra recipientes que estavam perdidos pelo laboratório, que podem ser usados nas medições.

Como Rick é muito ocupado, ele pediu para você desenvolver um programa que, dados os recipientes a serem utilizados e uma determinada medição, calcule o número mínimo de operações necessárias para se obter a medida. Rick ainda deseja poder informar se um recipiente quebrou ou se um novo recipiente foi encontrado no laboratório.

Detalhes do problema

O objetivo deste trabalho é praticar os conceitos relacionados a estruturas de dados elementares. Você deverá desenvolver um programa que atenda os seguintes critérios:

- 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 por Rick;

Para ilustrar o problema, vamos considerar o seguinte caso: no laboratório de Rick, há apenas 3 recipientes: 100 ml; 300 ml; e 500 ml. Rick quer medir 400 ml. Há diferentes manipulações possíveis para atingir a quantidade, podemos citar:

1. Adicionar 500 ml e depois retirar 100 ml;
2. Adicionar 300 ml e depois adicionar mais 100 ml;
3. Adicionar 100 ml por quatro vezes consecutivas.

Embora as três manipulações apresentadas sejam válidas, a terceira não é uma solução para o problema, pois necessita de 4 operações, enquanto a primeira e segunda necessitam apenas de duas. Considere ainda que, após essa medição, Rick quebrou o recipiente de 300 ml. Caso ele quisesse medir novamente 400 ml, apenas a primeira operação apresentada seria uma solução válida.

Entrada e Saída

Os formatos desejados de entrada e saída são:

Entrada. Neste trabalho, a entrada será a padrão do sistema (`stdin`). Ela será composta por várias linhas, que são compostas por: um inteiro **Q** ($0 < \mathbf{Q} \leq 200{,}000$), indicando uma medição em ml; e um caractere **E** (após um espaço em branco), indicando o tipo de evento de entrada, cujo significado é descrito à seguir:

- **i**: indica a inclusão no seu programa de um recipiente de **Q** ml;
- **r**: indica que o recipiente de **Q** ml quebrou e deve ser removido do seu programa;
- **p**: indica que Rick deseja fazer uma medição de **Q** ml.

O final da entrada é indicado por `EOF` (`CTRL + D` no terminal). Lembre-se que Rick possui recipientes diferentes espalhados pela loja, e portanto, novos recipientes podem ser adicionados a qualquer momento, bem como podem quebrar a qualquer momento e serem removidos. Para essas linhas de entrada, nenhuma saída deverá ser produzida.

Saída. Neste trabalho, a saída será a padrão do sistema (`stdout`). Para as entradas relacionadas a uma medição desejada por Rick, você deve produzir uma saída que indica a quantidade mínima de operações necessárias. Para entradas relacionadas a inclusão ou remoção de recipientes, nenhuma saída deve ser produzida.

Informações adicionais. Para este problema, assuma que para toda entrada do tipo **r** já houve uma entrada do tipo **i**, não sendo necessário este tipo de verificação. Assuma também que para toda entrada do tipo **p** é possível encontrar um número mínimo de operações considerando os recipientes existentes, provenientes das entradas **i** e **r** anteriores.

Exemplos

Exemplo de Entrada	Exemplo de Saída
30 i 50 i 50 p 20 p 110 p	1 2 3

Exemplo de Entrada	Exemplo de Saída
50 i 10 i 20 p 50 r 100 p 25 i 12 i 74 p	2 10 4

Exemplos passo-a-passo

Para facilitar a resolução do problema, mostraremos, em alto nível, como o problema poderia ser resolvido com alguns exemplos. A Figura 1 mostra a notação que será utilizada no restante dessa seção. Cada bloco representa uma medição, que é descrita através de dois números. Na metade direita do bloco encontra-se a quantia medida, e na metade esquerda, a quantidade de operações/movimentos necessários para se obter esta quantia. Por exemplo, o bloco da Figura 1 indicaria que é possível realizar uma medição de 105ml com apenas 3 operações ou movimentos. A descrição textual desse bloco seria dada por [3|105].

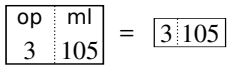

Figura 1: Notação utilizada nos exemplos.

A Figura 2 mostra um exemplo onde deseja-se medir 20ml e somente dois recipientes de 50 e 30ml estão disponíveis. As setas cheias indicam a utilização do recipiente de 50ml, e as setas tracejadas, a utilização do recipiente de 30ml. Como descrito anteriormente, é possível fazer duas operações diferentes com cada recipiente, adicionar a capacidade do recipiente, indicada pelo símbolo de adição na seta, ou removê-la, indicada pelo símbolo de subtração.

A resolução desse problema é feita de baixo para cima (bottom-up). Isto é, devemos resolver os problemas mais simples e fáceis primeiro. A partir da solução de um problema fácil, conseguiremos resolver problemas maiores e mais complexos. Neste trabalho, usaremos medidas que necessitam de k movimentos para descobrir as medidas que necessitam de $k+1$ movimentos.

A versão mais simples do nosso problema seria descobrir quantas operações são necessárias para medir 0ml, cuja solução é trivial, 0, como mostra a Fig 2a. A partir dessa solução com 0 movimentos, podemos descobrir quais são as soluções com 1 movimento. A Fig. 2b mostra que 4 medições poderiam ser feitas com um único movimento a partir da medição [0|0]. Para obter essas 4 medições, devemos simplesmente adicionar/remover o conteúdo de cada um dos recipientes disponíveis. Note que as medidas [1|−50] e [1|−30] representam volumes negativos, e portanto podem ser descartadas do nosso conjunto de soluções parciais. Por fim, a Fig. 2c constrói soluções de 2 movimentos a partir do sub-problema [1|50]. Uma das novas soluções encontradas é [2|20], que é a medida que nós procurávamos inicialmente. Portanto, podemos parar nossa busca aqui, e reportar que 2 operações são necessárias para obtermos 20ml.

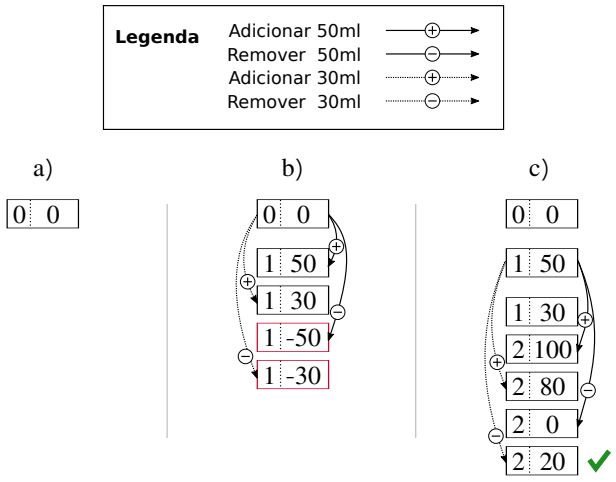

Figura 2: Passo a passo da construção da solução de 20ml, a partir de recipientes de 50 e 30 ml.

A Fig. 3 mostra um exemplo mais complexo. Nesse caso, temos recipientes que medem 35 e 17ml, e desejamos medir uma quantia de 53ml. Novamente, devemos começar do caso base, que mede 0ml com 0 operações, e adicionar/remover a quantia de cada recipiente. A Fig. 3c mostra os exemplos de dois movimentos construídos a partir da medida [1|35], enquanto a Fig. 3d mostra os que foram construídos a partir de [1|17]. Somente na quarta iteração, mostrada na Fig. 3e, conseguimos medir a quantia desejada de 53ml com 3 operações.

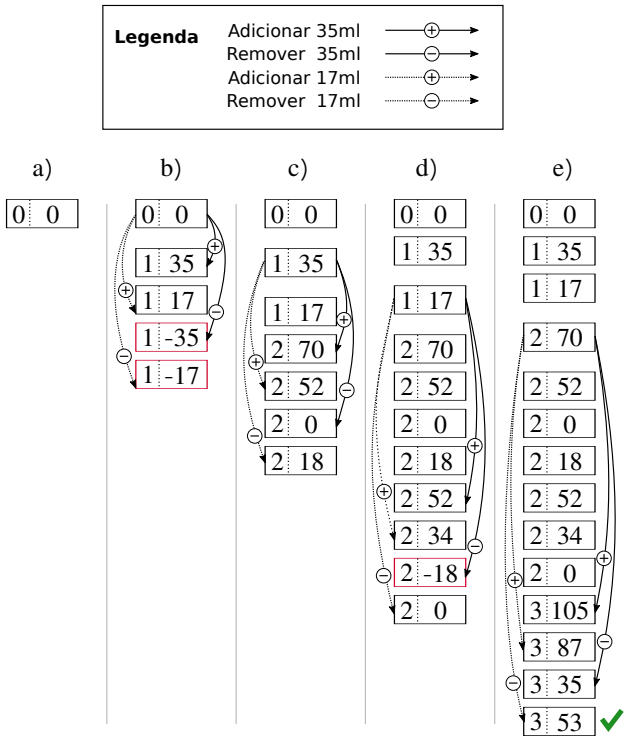

Figura 3: Passo a passo da construção da solução de 53ml, a partir de recipientes de 35 e 17 ml.

O funcionamento desse algoritmo está diretamente ligado com a ordem em que os problemas são avaliados. Note que antes de obter uma medida que utiliza k movimentos, **todas** as possíveis medidas com $k-1$ movimentos já foram avaliadas. Portanto, temos a garantia de que a primeira vez em que uma medida **Q** aparece é aquela com a menor quantidade de operações, visto que já vimos todas as medições possíveis entre 0 e $k-1$ movimentos, e **Q** não é uma delas. Se essa ordem não for cumprida, soluções não mínimas podem ser descobertas, comprometendo o funcionamento de seu algoritmo.

Entregáveis

Código-fonte. A implementação poderá ser feita utilizando as linguagens C ou C++. Não será permitido o uso da *Standard Library* do C++ ou de bibliotecas externas que implementem as estruturas de dados ou os algoritmos. **A implementação das estruturas e algoritmos utilizados neste trabalho deve ser sua**. Os códigos devem ser executáveis em um computador com *Linux*. Caso não possua um computador com *Linux*, teste seu trabalho em um dos computadores do laboratório de graduação do CRC[1]. A utilização de *Makefile*[2] é **obrigatória** para este trabalho.

Aplique boas práticas de programação e organize seu código-fonte em arquivos, classes e funções de acordo com o significado de cada parte. A separação de responsabilidades é um dos princípios da

[1] `https://crc.dcc.ufmg.br/infraestrutura/laboratorios/linux`
[2] `https://opensource.com/article/18/8/what-how-makefile`

engenharia de software: cada função deve realizar apenas uma tarefa e cada classe deve conter apenas métodos condizentes com sua semântica.

Documentação. A documentação de seu programa **deverá** estar em formato **PDF**, **seguir** o modelo de trabalhos acadêmicos da SBC (que pode ser encontrado online[3]) e ser **sucinta**. Deverá conter **todos** os seguintes tópicos:

- Cabeçalho. Título do trabalho, nome e número de matrícula do autor.
- Introdução. Apresentação do problema abordado e visão geral sobre o funcionamento do programa.
- Implementação. Descrição sobre a implementação do programa. Devem ser detalhados o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, compilador utilizado, bem como decisões tomadas relativas aos casos e detalhes que porventura estejam omissos no enunciado.
- Instruções de compilação e execução. Instruções de como compilar e executar o programa.
- Análise de complexidade. Estudo da complexidade de tempo e espaço do algoritmo de melhor e pior caso desenvolvido utilizando o formalismo da notação assintótica.
- Conclusão. Resumo do trabalho realizado, conclusões gerais sobre os resultados e eventuais dificuldades ou considerações sobre seu desenvolvimento.
- Bibliografia. Fontes consultadas para realização do trabalho.

O código-fonte e a documentação devem ser organizados como demonstrado pela árvore de diretórios na Figura 4. O diretório raiz deve ser nomeado de acordo seu **nome e último sobrenome**, separado por *underscore*, por exemplo, o trabalho de "Kristoff das Neves Björgman" seria entregue em um diretório chamado `kristoff_bjorgman`. Este diretório principal deverá conter um subdiretório chamado `src`, que por sua vez conterá os códigos (`.cpp, .c, .h, .hpp`) na estrutura de diretórios desejada. A documentação **em formato PDF** deverá ser incluída no diretório raiz do trabalho. Evite o uso de caracteres especiais, acentos e espaços na nomeação de arquivos e diretórios.

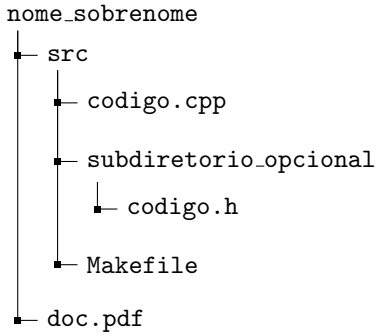

Figura 4: Estrutura de diretórios do entregável do TP1

O diretório deverá ser submetido em um único arquivo **'nome_sobrenome.zip'**, (onde nome e sobrenome seguem as mesmas diretrizes para o nome do diretório, explicado acima) através do Moodle da disciplina até as **23:59** do dia **04 de outubro de 2019**.

[3] `http://www.sbc.org.br/documentos-da-sbc/summary/169-templates-para-artigos-e-capitulos-de-livros/878-modelosparapublicaodeartigos`

Considerações Finais

Algumas considerações finais importantes:

- **Preste bastante atenção nos detalhes da especificação**. Cada detalhe ignorado acarretará em perda de pontos.
- O que será avaliado no trabalho:

  Boas práticas de programação: se o está código bem organizado e indentado, com comentários explicativos, possui variáveis com nomes intuitivos, modularizado, etc.

  Implementação correta dos algoritmos: se a árvore e o processo de decodificação foram implementados de forma correta e resolvem o problema aqui descrito.

  Conteúdo da documentação: se todo o conteúdo necessário está presente, reflete o que foi implementado e está escrito de forma coerente e coesa.
- Após submeter no Moodle seu arquivo **'.zip'**, faça o download dele e certifique-se que não está corrompido. Não será dada segunda chance de submissão para arquivos corrompidos.
- Em caso de dúvidas, **não hesite em perguntar** no Fórum de Discussão no Moodle ou procurar os monitores da disciplina – estamos aqui para ajudar!
- **PLÁGIO É CRIME**: caso utilize códigos disponíveis online ou em livros, **referencie** (inclua comentários no código fonte e descreva a situação na documentação). Trabalhos onde o plágio for identificado serão devidamente penalizados: o aluno terá seu trabalho anulado e as devidas providências administrativas serão tomadas. Discussões sobre o trabalho entre colegas são encorajadas, porém compartilhamento de código ou texto é plágio e as regras acima também se aplicam.
- Em caso de atraso na entrega, serão descontados $2^d - 1$ pontos, onde d é o número de dias (corridos) de atraso arredondado para cima.
- Comece o trabalho o mais cedo possível. Você nunca terá tanto tempo pra fazê-lo se começar agora!

Bom trabalho!

Apêndices

A Dicas para a documentação

O objetivo desta seção é apresentar algumas dicas para auxiliar na redação da documentação do trabalho prático.

1. **Sobre *Screenshots***: ao incluir *screenshots* (imagens da tela) em sua documentação, evite utilizar o fundo escuro. Muitas pessoas preferem imprimir documentos para lê-los e imagens com fundo preto dificultam a impressão e visualização. Recomenda-se o uso de fundo branco com caracteres pretos, para *screenshots*. Evite incluir trechos de código e/ou pseudocódigos utilizando *screenshots*. Leia abaixo algumas dicas de como incluir código e pseudocódigo em seu texto.

2. **Sobre códigos e pseudocódigos**: Ao incluir este tipo de texto em sua documentação procure usar a ferramenta adequada para que a formatação fique a melhor possível. Existem várias ferramentas para LaTeX, como o `minted`[4], o `lstlisting`[5], e o `algorithm2e`[6] (para pseudocódigos), e algumas para Google Docs (`Code Blocks`[7], Code Pretty[8]).

3. **Sobre URLs e referências**: evite utilizar URLs da internet como referências. Geralmente URLs são incluídas como notas de rodapé. Para isto basta utilizar o comando `\footnote{\url{}}` no LaTeX, ou ativar a opção nota de rodapé[9] no Google Docs/MS Word.

4. **Evite o Ctrl+C/Ctrl+V**: encoraja-se a modularização de código, porém a documentação é única e só serve para um trabalho prático. Reuso de documentação é auto-plágio[10]!

[4] https://www.overleaf.com/learn/latex/Code_Highlighting_with_minted
[5] https://www.overleaf.com/learn/latex/Code_listing
[6] https://en.wikibooks.org/wiki/LaTeX/Algorithms
[7] https://gsuite.google.com/marketplace/app/code_blocks/100740430168
[8] https://chrome.google.com/webstore/detail/code-pretty/igjbncgfgnfpbnifnnlcmjfbnidkndnh?hl=en
[9] https://support.office.com/en-ie/article/insert-footnotes-and-endnotes-61f3fb1a-4717-414c-9a8f-015a5f3ff4cb
[10] https://blog.scielo.org/blog/2013/11/11/etica-editorial-e-o-problema-do-autoplagio/#.XORgbdtKg5k