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

Professores: Luiz Chaimowicz e Raquel Prates
Monitores: Alysson R. da Silva e Tiago Alves
Estruturas de Dados 2021.1
Belo Horizonte, 17 de Agosto de 2021

Trabalho Prático III: Recebimento e Download das Consciências

---

Trabalho Individual. Valor: 10 pontos **Entrega: 08 de Setembro de 2021**

1 Introdução

No ano de 2076 as mega corporações dominaram o mundo, onde cada uma delas controla de forma vasta as tecnologias que regem o comportamento da sociedade. Em especial, a corporação Rellocator CO. utiliza uma tecnologia denominada Virtual Environment for Relocation Leisure and Autonomous Behavior (VERLAB). Essa tecnologia é responsável por permitir upload e download de consciências na mega-net (uma internet interplanetária).

Através dos serviços da Rellocator CO., uma pessoa pode fazer upload de sua mente para corpos sintéticos. No entanto, um dos grandes problemas eram os ataques hackers que furtavam as consciências durante o processo de upload. Dessa forma, a Rellocator CO. contratou uma equipe para desenvolver vários protótipos de um sistema de controle, detecção de anomalias, ordenação, e otimização dos mesmos para prevenir roubos. Devido aos esforços do time, a quantidade de roubos de consciências agora se encontra abaixo de 5%. Por esse motivo, agora a Rellocator CO. gostaria de contratar você novamente para otimizar o processo de Download em um corpo sintético destino.

Atualmente, o processo de download de consciências é muito desorganizado, onde cada fragmento/-dado (representado como número binário) pertencente a uma consciência é armazenado em uma grande lista de chegada nas estações de download. Uma vez dentro da lista de chegada, os dados de uma consciência são selecionados através de um processo de busca. Ao ser encontrados, os mesmos são fundidos, removidos da fila, e baixados pelo corpo destino por um processo de download. O maior problema desse processo é a falta de capacidade de processamento das estações de download para grandes volumes de consciências. Por esse motivo, a sua tarefa agora é criar uma mecanismo eficiente para armazenamento e busca de dados.

2 Especificações

Para abordar o problema enfrentado pela Rellocator CO., esse projeto foi dividido em duas etapas. A primeira etapa é responsável por substituir a estrutura de dados utilizada para o recebimento de dados binários das consciências por uma que propicie buscas mais eficientes (gaste menos tempo efetuando uma busca). Já a segunda etapa, é responsável por efetuar um processo de busca nessa estrutura, selecionar qual pessoa deve ser realocada, e gerar um índice a partir de seus dados para saber em qual corpo ela será realocada.

2.1 Etapa 1: Inserção de dados de consciências

Quando os dados binários de diversas consciências chegam na estação de download, os mesmos são organizados em uma fila onde esperam ser buscados e transferidos. Por consequência do processo de busca em uma fila ser ineficiente, a Rellocator CO. solicitou que você implemente uma árvore binária. Logo, você deve implementar uma árvore binária, onde cada nó armazena todos os dados pertencentes a uma mesma pessoa.

Para inserir os dados de uma pessoa na árvore, deve-se utilizar o próprio nome dela como chave. Uma vez que cada pessoa geralmente possui vários dados binários, devem ser utilizadas listas encadeadas para cada nó da árvore. Dessa forma, cada nó da árvore deve armazenar uma lista encadeada e cada nó dessa lista encadeada deve conter um número binário (fragmento de uma consciência).

2.2 Etapa 2: Busca e envio

A fase 2 resume-se em substituir o processo de busca antigo (que era realizado em uma lista) por um novo que é efetuado na árvore binária implementada. Esse processo visa buscar por uma pessoa na árvore, fundir todos os dados binários dentro da lista encadeada nesse nó em um único valor através da soma dos mesmo, e por fim remover esse nó da árvore (enviando ele ao corpo destino para download e reportando no terminal do programa).

Para efetuar uma busca na árvore, você deve receber o nome de uma pessoa e percorrer a árvore. Ao percorrer a árvore, caso o nome da pessoa seja encontrado em um nó, então deve-se recuperar a lista contendo os dados binários (i.e. fragmentos da consciência) daquela pessoa. Uma vez em posse desta lista, deve-se fundir todos os dados binários da mesma e gerar um índice. Para fundir os dados binários da lista de uma pessoa na árvore, deve-se:

1. Converter cada binário da lista de dados para seu respectivo inteiro.
2. Somar todos os inteiros gerados.
3. Remover nome e respectiva lista de dados da árvore de busca pois os mesmos já foram enviados.

3 Relatório de envios

Para facilitar sua vida, a Rellocator CO. disponibilizará para você um arquivo que representa um batch para construção da árvore de busca e para saber quais nomes devem ser buscados e enviados em um determinado momento.

O arquivo disponibilizado é dividido em duas partes de forma semelhante ao projeto. A primeira parte contém os nomes e dados de todas as pessoas para construção da árvore. Já a segunda parte, contém quais pessoas devem ser buscadas para transferência. Ambas as partes devem ser lidas em sequência para executar os processos de transferências. Durante a leitura, é esperado um relatório do estado do sistema, como especificado nas Seções 3.1 e 3.2.

3.1 Entrada

O executável do seu programa deve receber o arquivo de entrada como **UM** argumento de linha de comando e ler esse arquivo. A primeira linha do arquivo de entrada contém um número que identifica quantas inserções devem ser feitas para construção da árvore binária de busca. Cada linha subsequente, até que se alcance a quantidade especificada, contém os campos '`NOME DADOS`'. Após todas as linhas de construção da árvore binária, o arquivo contém N linhas de comandos. Um comando é composto pelo campo '`NOME`' que deve ser utilizado como chave para busca na árvore binária. A Entrada 1 ilustra um formato de entrada.

O exemplo ilustrado pela Entrada 1 contém 6 campos de construção para árvore. Além disso, após os campos para construção da árvore, a entrada também possui duas linhas de comandos referentes a '`RAQUEL`' e a '`PRATES`'.

```
6
CHAIMO 01011011
RAQUEL 10110101
CHAIMO 01111001
PRATES 01010001
CHAIMO 10101010
PRATES 11110000
RAQUEL
PRATES
```

Entrada 1: Exemplo de arquivo de entrada.

3.2 Saída

Após a leitura todos os conjuntos de '`NOME DADOS`', você deve imprimir no terminal todos os nomes inseridos na árvore utilizando percorrimento em ordem (in-order ou central). A impressão dos nomes da árvore construída deve ser feita em uma única linha, onde cada nome deve ser separado por espaço.

Em seguida, você deve processar o restante do arquivo de entrada. Para cada nome (um por linha de comando), você deve executar o processo de soma descrito na Seção 2.2. Ao finalizar a execução de uma linha, você deve imprimir no terminal o nome e a soma correspondente separados por um único espaço.

Quando o programa finalizar a execução de todas as linhas de comando, você deve imprimir no terminal a árvore remanescente utilizando percorrimento em ordem (in-order ou central) em uma única linha. Cada nome impresso da árvore remanescente deve ser separado por um único espaço. Um exemplo de saída válida para a entrada apresentada na Seção 3.1, é ilustrado pela Saída 1, onde, 181 e 321 são as respectivas somas geradas para a '`RAQUEL`' e a '`PRATES`'.

```
CHAIMO PRATES RAQUEL
RAQUEL 181
PRATES 321
CHAIMO
```

Saída 1: Exemplo de saída para a entrada apresentada na Seção 3.1.

Note que o valor 181 para '`RAQUEL`' é o valor inteiro do dado 10110101 e 321 é o número inteiro que representa a soma dos dados associados a '`PRATES`', ou seja, 01010001(= 81) e 11110000(= 240), que somados dão 321. Ao fim da saída, é impressa a árvore restante que, no caso, contém apenas o '`CHAIMO`' pois ele ainda não foi enviado e está em espera.

Para imprimir todas as informações solicitadas no terminal, você deve utilizar a saída padrão. Além disso, o seu programa não deve conter quebra de linha após a impressão da árvore remanescente.

Obs: No apêndice A, ao fim deste documento, é apresentado um exemplo de execução para a instância de entrada mostrada em Entrada 1.

4 Entregáveis

Você deve utilizar a linguagem C ou C++ para o desenvolvimento do seu sistema. O uso de estruturas pré-implementadas pelas bibliotecas-padrão da linguagem ou terceiros é **terminantemente vetado**. Caso seja necessário, use as estruturas que **você** implementou nos Trabalhos Práticos anteriores para criar **suas próprias implementações** para todas as classes, estruturas, e algoritmos.

Você **DEVE utilizar** a estrutura de projeto abaixo junto ao '`Makefile`' disponibilizado no *Moodle*:

```
- TP
    |- src
    |- bin
    |- obj
    |- include
    Makefile
```

A pasta '`TP`' é a raiz do projeto; a pasta '`bin`' deve estar vazia; '`src`' deve armazenar arquivos de código ('`*.c`', '`*.cpp`', ou '`*.cc`'); a pasta '`include`', os cabeçalhos (*headers*) do projeto, com extensão '`*.h`', por fim a pasta '`obj`' deve estar vazia. O **Makefile** disponibilizado no **Moodle** deve estar na **raiz do projeto**.

4.1 Documentação

A documentação do trabalho deve ser entregue em formato **pdf** e também **DEVE** seguir o modelo de relatório que será postado no Moodle. Além disso, a documentação deve conter **TODOS** os itens descritos a seguir **NA ORDEM** em que são descritos,

- Título, nome, e matrícula.
- **Introdução:** Contém a apresentação do contexto, problema, e qual solução será empregada.
- **Método:** Descrição da implementação que detalhe as estruturas, classes e métodos implementados.
- **Análise de Complexidade:** Contém a análise da complexidade de tempo e espaço dos procedimentos implementados, formalizada pela notação assintótica.
- **Conclusões:** A Conclusão deve conter uma frase inicial sobre o que foi feito no trabalho. Posteriormente deve-se sumarizar o que foi aprendido.
- **Bibliografia:** Contém fontes utilizadas para realização do trabalho. A citação deve estar em formato científico apropriado que deve ser escolhido por você.
- **Instruções para compilação e execução**: Esta seção deve ser colocada em um apêndice ao fim do documento e em uma página exclusiva (não divide página com outras seções).

4.2 Submissão

Todos os arquivos relacionados ao trabalho devem ser submetidos na atividade designada para tal no *Moodle*. A entrega deve ser feita **em um único arquivo** com extensão **.zip**, com nomenclatura '`nome_sobrenome_matricula.zip`', onde '`nome`', '`sobrenome`', e '`matricula`' devem ser substituídos por suas informações pessoais. O arquivo **.zip** deve conter a sua documentação no formato **.pdf** e a estrutura de projeto descrita na Seção 4.

5 Avaliação

O trabalho será avaliado em 10 pontos com a seguinte distribuição:

1. Corretude na execução dos casos de teste - **50% da nota total**
2. Apresentação da análise de complexidade das implementações - **25% da nota total**
3. Estrutura e conteúdo exigidos para a documentação - **10% da nota total**
4. Indentação do código fonte - **5% da nota total**
5. Comentários no código fonte em forma de documentação em cima da assinatura de **TODAS** as funções criadas - **5% da nota total**
6. Cumprimento total da especificação - **5% da nota total**

Se o programa submetido não compilar[1], seu trabalho não será avaliado e sua nota será 0. Trabalhos entregues com atrasos sofrerão penalização de $2^d - 1$ pontos, com $d =$ dias de atraso.

6 Considerações Finais

1. Leia **atentamente** o documento de especificação, pois o descumprimento de quaisquer requisitos obrigatórios aqui descritos causará penalizações na nota final.
2. Certifique-se de garantir que seu arquivo foi submetido corretamente no sistema.
3. **Plágio é CRIME**. Trabalho onde o plágio for identificado serão **automaticamente anulados** e as medidas administrativas cabíveis serão tomadas. Discussões a respeito do trabalho entre colegas são permitidas. É permitido consultar fontes externas, desde que exclusivamente para fins didáticos e devidamente registradas na sessão de bibliografia da documentação. **Cópia e compartilhamento de código não são permitidos**.

[1]Entende-se por compilar aquele programa que, independente de erros no Makefile ou relacionados a problemas na configuração do ambiente, funcione e atenda aos requisitos especificados neste documento em um ambiente Linux.

7 FaQ (Frequently asked Questions)

1. Posso utilizar o tipo String? **SIM.**
2. Posso utilizar o tipo String para simular minhas estruturas de dados? **NÃO.**
3. Posso utilizar alguma biblioteca para tratar exceções? **SIM.**
4. Posso utilizar alguma biblioteca para gerenciar memória? **SIM.**
5. As análises a apresentação dos resultados são importantes na documentação? **SIM.**
6. Os meus princípios de programação ligados a C++ e relacionados a engenharia de software serão avaliados? **NÃO.**
7. Com relação ao código fonte, será avaliado apenas o que foi explicitado na Seção 5, incluindo Indentação de código e comentário para documentação das funções? **SIM.**
8. Posso utilizar alguma estrutura de dados do tipo Queue, Stack, Vector, List, e **etc**..., do C++? **NÃO.**
9. Posso fazer o trabalho em dupla ou em grupo? **NÃO.**
10. Posso trocar informações com os colegas sobre a teoria? **SIM.**
11. Posso fazer o trabalho no Windows, Linux, ou MacOS? **SIM.**
12. Posso utilizar IDEs, Visual Studio, Code Blocks, Visual Code, Eclipse? **SIM.**

Appendices

A Exemplo de Execução para uma Instância de Entrada

Nesta seção é ilustrado um exemplo passo a passo da execução da especificação proposta para uma instância de entrada. O resultado de cada passo na árvore é ilustrado a **esquerda** e o estado atual do terminal a **direita**. Considere a seguinte entrada:

```
6
CHAIMO 01011011
RAQUEL 10110101
CHAIMO 01111001
PRATES 01010001
CHAIMO 10101010
PRATES 11110000
RAQUEL
PRATES
```

Obs: É importante notar que, ao representar os nós das árvores, a chave foi colocada na primeira linha em negrito e que as demais linhas representam os dados na fila, onde a fila cresce de cima para baixo.

A.1 Passo 1

O passo 1 serve para sinalizar que haverão 6 dados a serem inseridos na árvore. Isso é feito com o intuito de separar quais linhas dizem respeito à construção da árvore, sendo o restante à respeito da busca.

Linha lida do arquivo de entrada: '`6`'

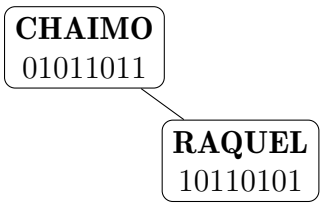
Nos 6 passos subsequentes cada comando é lido, um por vez, do arquivo de entrada e executado com o intuito de construir a árvore.

A.2 Passo 2

Linha lida do arquivo de entrada: '`CHAIMO 01011011`'

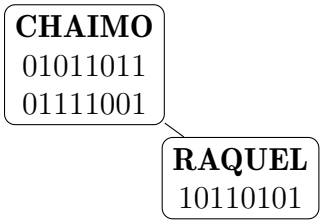
A.3 Passo 3

Linha lida do arquivo de entrada: ‘`RAQUEL 10110101`’



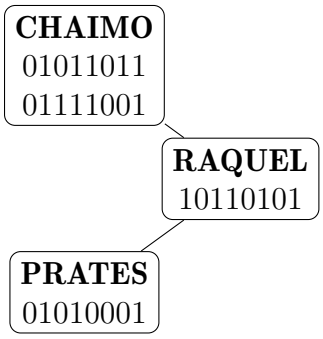
A.4 Passo 4

Linha lida do arquivo de entrada: ‘`CHAIMO 01111001`’



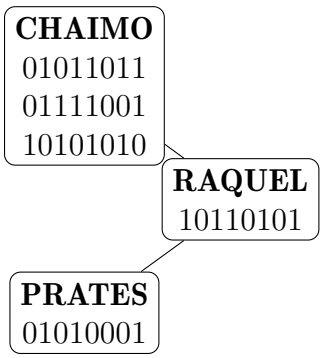
A.5 Passo 5

Linha lida do arquivo de entrada: ‘`PRATES 01010001`’



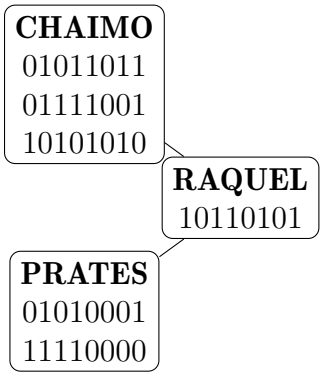
A.6 Passo 6

Linha lida do arquivo de entrada: ‘`CHAIMO 10101010`’



A.7 Passo 7

Linha lida do arquivo de entrada: ‘`PRATES 11110000`’

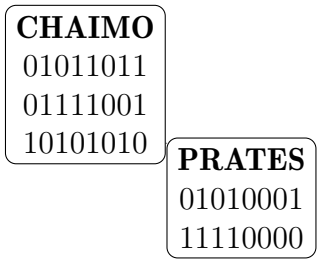



```
CHAIMO PRATES RAQUEL
```

Ao final da construção da árvore a mesma foi impressa. O restante das linhas até o final do arquivo dizem respeito ao envio das consciências. A busca e envio é feito um por vez de acordo com o especificado no tópico 2.2.

A.8 Passo 8

Linha lida do arquivo de entrada: ‘`RAQUEL`’




```
CHAIMO PRATES RAQUEL
RAQUEL 181
```

A.9 Passo 9

Linha lida do arquivo de entrada: 'PRATES'

CHAIMO
01011011
01111001
10101010

```
CHAIMO PRATES RAQUEL
RAQUEL 181
PRATES 321
CHAIMO
```

Note que, após ler e executar o último comando do arquivo de entrada o restante da árvore foi impressa, no caso apenas um nó e a execução do sistema deve ser finalizada.