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 2021.2
Belo Horizonte, 2 de Julho de 2021

Trabalho Prático I: Realocação Interplanetária 2076

Trabalho Individual. Valor: 10 pontos **Entrega: 25 de Julho 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 o VERLAB que age como uma espécie de ambiente virtual na mega-net. Dentro desse ambiente virtual, as diversas consciências podem interagir, se divertir através dos mais variados tipos de simulações, e mais importante, se realocar para outros corpos em diferentes planetas espalhados pelo universo e que foram colonizados pela raça humana.

A tecnologia de realocação de consciências foi o que permitiu que a humanidade se expandisse para outros planetas de forma eficiente, pois ela permite que cientistas e profissionais das mais diversas áreas possam contribuir com soluções para problemas espalhados pelo universo. Para realocar uma consciência a mesma precisa ser transferida através da mega-net para um corpo sintético, biológico ou robótico. O grande problema quando se trabalha com realocação de consciências é o fato de que hackers que agem de forma integrada com formas híbridas de Inteligência Artificial conseguem roubar as mesmas enquanto são transferidas.

Segundo dados liberados pela Rellocator CO., 27% das consciências que trafegam pela mega-net são roubadas e realocadas para corpos robóticos escavadores em minas de Composto Z espalhadas em diversos asteroides e planetas. Uma vez realocada para um robô escavador, a consciência é reprogramada para agir como um escravo e sua humanidade é completamente perdida. No ano de 2075, por exemplo, vários hackers atuaram durante a criação de robôs escravos em diversas expedições identificadas pelo codinome Extração-Z. Para reduzir o índice de roubo de consciências, a Rellocator CO. contratou você para implementar um sistema de segurança, detecção de anomalias, e implementação de planos de ação, durante um processo de transferência através da mega-net.

2 Especificações

Para implantar o seu sistema a Rellocator CO. lhe deu acesso a um sub-conjunto de servidores de transferência da mega-net que atuam utilizando buffers de dados de transferência. O seu sistema deve executar comandos (planos de ação de distribuição de dados), enviar dados para os buffers dos servidores de transferência, marcar anomalias nos dados enviados, detectar erros no sistema, e reiniciar toda a infra-estrutura caso seja necessário. Além disso, o sistema deve manter um histórico das consciências enviadas, esse histórico é único e global e pode ser utilizado para conferir as transferências realizadas.

Existem seis comandos que seu controlador pode executar, sendo eles o comando '`INFO`', '`WARN`', '`TRAN`', '`ERRO`', '`SEND`', '`FLUSH`'. Cada comando pode ter até 3 parâmetros no formato String, como descritos a seguir.

- '`INFO <S> <DADOS>`': Ao executar este comando, o conteúdo da variável '`DADOS`' deve ser inserido no buffer do servidor '`S`', em forma de uma mensagem (String), com política **First in, First out (FIFO)** para transferência na mega-net. Por exemplo, o comando '`INFO 5 "NIVIO 00010101"`' corresponde a enviar os dados '`NIVIO 00010101`' para o servidor 5.

- '`WARN <S> <I>`': Um comando '`WARN`' é executado pelo controlador central quando é detectada uma tentativa de hacking dos dados. Nesse caso, deve-se remover os dados da posição '`I`' do buffer do servidor '`S`' e inseri-los na primeira posição do buffer desse mesmo servidor para antecipar o seu envio. Por exemplo, o comando '`WARN 3 7`' corresponde a remover a mensagem que se encontra na posição 7 do buffer da máquina 3 e a inserir na primeira posição do mesmo.

- '`TRAN` $<S_1>$ $<S_2>$': Este comando tem como finalidade desabilitar um servidor 'S_1' quando são detectadas possíveis anomalias. Para que isso seja feito de forma a não perder o conteúdo do buffer do servidor 'S_1', deve-se transferir os seus dados para o buffer do servidor 'S_2'. A transferência deve ocorrer na ordem em que os dados estiverem dispostos em 'S_1' (Política de remoção FIFO). Já a inserção deve ser feita no fim do buffer de 'S_2', ou seja, os dados devem ser inseridos ao fim dos dados que já estão no buffer de 'S_2'. Por fim, deve-se deletar todo o conteúdo do buffer de 'S_1'. Por exemplo, o comando '`TRAN 2 13`' corresponde a transferir todo o conteúdo do buffer do servidor 2 para o buffer do servidor 13 e deletar todo o conteúdo do buffer do servidor 2.

- '`ERRO <S>`': A execução de um comando de ERRO identifica que algo estranho ocorreu na mega-net. Quando isso ocorre, deve-se imprimir no terminal a mensagem '`ERRO <S>`' e todo o conteúdo do buffer do servidor '`S`'. Posteriormente, deve-se deletar todo o conteúdo do buffer de '`S`'.

- '`SEND`': Quando o controlador central executa um comando '`SEND`', deve-se percorrer os buffers de todos os servidores enviando os dados da primeira posição de cada um, e removendo-os do buffer. Os dados são enviados para os corpos destino e também devem ser armazenados no histórico de consciências enviadas.

- '`FLUSH`': Ao executar um comando do tipo '`FLUSH`', deve-se imprimir no terminal o histórico de consciências na ordem que foram enviadas. Além disso, deve-se também imprimir no terminal, o conteúdo do buffer de **TODOS** os servidores (a impressão de um buffer individual deve ser feita utilizando política FIFO).

3 Entrada

O executável do seu programa deve receber um argumento de entrada via linha de comando que representa o nome do arquivo que contém os comandos a serem executados. No arquivo de entrada, a primeira linha contém a quantidade de servidores que estão disponíveis para você gerenciar. As demais linhas do arquivo contém os comandos descritos na Seção 2. As Strings '`DATA`' e '`MSG`' dos comandos da Seção 2 estarão contidas entre aspas duplas '`"`' como no exemplo a seguir.

```
20
INFO 0 "NIVIO 00110100"
INFO 0 "RAQUEL 00110101"
INFO 1 "CHAIMOWICZ 01001010"
WARN 0 1
TRAN 0 1
ERRO 1
```

Entrada 1: Exemplo de arquivo de entrada que contém a quantidade de máquinas disponíveis e todos os comandos a serem executados, onde os conteúdos das mensagens são armazenadas entre aspas duplas.

4 Saída

Utilize a **saída padrão** '`stdout`'/terminal para imprimir todas as informações. A saída deve conter a resposta dos comandos conforme especificado na Seção 2. As mensagens devem ser impressas sem aspas duplas '`"`'. A seguir é apresentado um exemplo de saída válida para entrada apresentada na Seção 3.

```
ERRO 1
CHAIMOWICZ 01001010
RAQUEL 00110101
NIVIO 00110100
```

Saída 1: Exemplo de saída para a entrada apresentada na Seção 3.

Obs: No apêndice A, ao fim deste documento, é apresentado um exemplo de execução para uma instância de entrada que contém todos os comandos apresentados.

5 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**. Utilize apenas as variáveis de tipos primitivos e derivados 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**.

5.1 Documentação

A documentação do trabalho deve ser entregue em formato **pdf** e também deve seguir o modelo de relatório que será postado no Moodle.

Ela deve conter **todos** os itens descritos abaixo.

- Título, nome, e matrícula.
- Introdução com apresentação do problema e visão geral sobre o programa.
- Instruções para compilação e execução.
- Descrição da implementação que detalhe as estruturas, classes e métodos. Comentários a respeito do eventual tratamento dado a casos e detalhes que não tenham sido previstos na especificação do trabalho.
- Estudo da complexidade de tempo e espaço dos procedimentos implementados, formalizado pela notação assintótica.

- Conclusões, observações e considerações finais.
- Bibliografia com as fontes consultadas para realização do trabalho.

5.2 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 5.

6 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, 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.

7 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**.

8 FaQ (Frequently asked Questions)

1. Posso utilizar o tipo String? **SIM.**
2. Posso utilizar alguma estrutura de dados do tipo Queue, Stack, Vector, List, e **etc**..., do C++? **NÃO.**
3. Posso fazer o trabalho em dupla ou em grupo? **NÃO.**
4. Posso trocar informações com os colegas sobre a teoria? **SIM.**
5. Posso fazer o trabalho no Windows, Linux, ou MacOS? **SIM.**
6. Posso utilizar IDEs, Visual Studio, Code Blocks, Visual Code, Eclipse? **SIM, desde que o trabalho seja entregue no formato especificado e que o mesmo funcione corretamente.**

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. A cada passo ilustrado é apresentado o conteúdo dos buffers a **esquerda** e também o estado atual do terminal a **direita**. Considere a seguinte entrada:

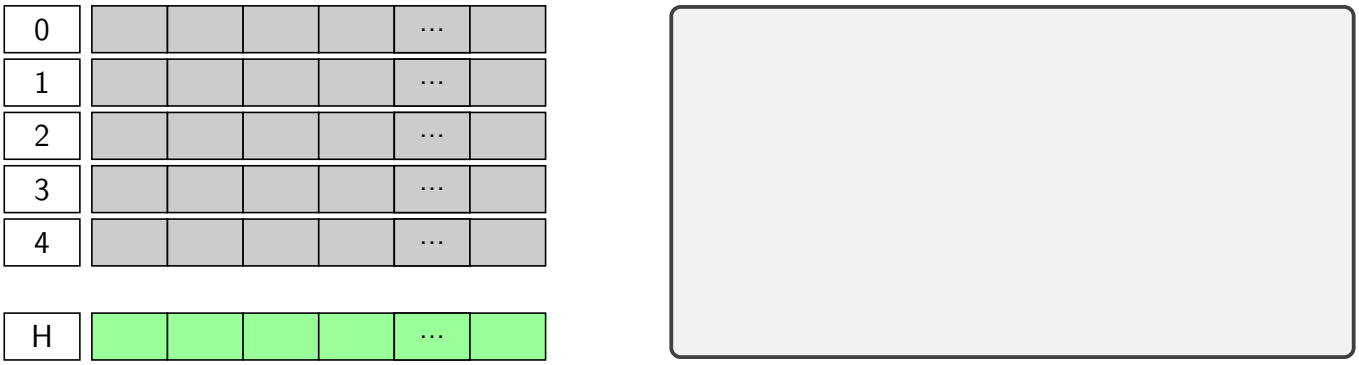
```
5
INFO 3 "CHAIMOWICZ 01011010"
INFO 1 "NIVIO 11010100"
INFO 3 "PRATES 00000010"
INFO 0 "RAQUEL 10000000"
INFO 2 "LUIZ 10000010"
INFO 2 "MARCO 11000110"
TRAN 3 2
INFO 4 "BRENO 11111100"
INFO 0 "FABIANO 10101110"
WARN 2 2
INFO 1 "GRERD 00110110"
SEND
INFO 1 "KERID 10111010"
ERRO 2
INFO 4 "EDSON 00110100"
FLUSH
```

Obs: É importante notar que ao representar os dados nos buffers dos servidores, apenas a primeira letra do nome está sendo utilizada para simplificar a visualização do exemplo. Porém, durante a execução, todo o conteúdo dos dados deve ser armazenado.

A.1 Passo 1

No passo 1 são criados 5 buffers, um para cada servidor especificado na primeira linha do arquivo de entrada e também o histórico. Os buffers criados são ilustrados a seguir, seguido pelo histórico (representado pela letra H e na cor verde).

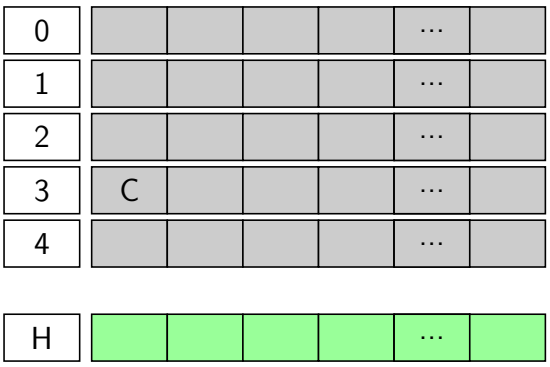
Linha lida do arquivo de entrada: '5'



Nos passos subsequentes cada comando é lido, um por vez, do arquivo de entrada e executado.

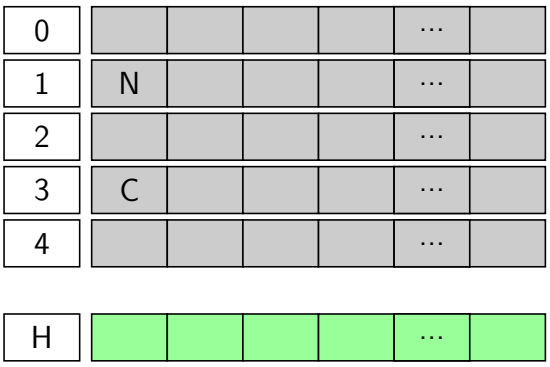
A.2 Passo 2

Linha lida do arquivo de entrada: `'INFO 3 "CHAIMOWICZ 01011010"'`



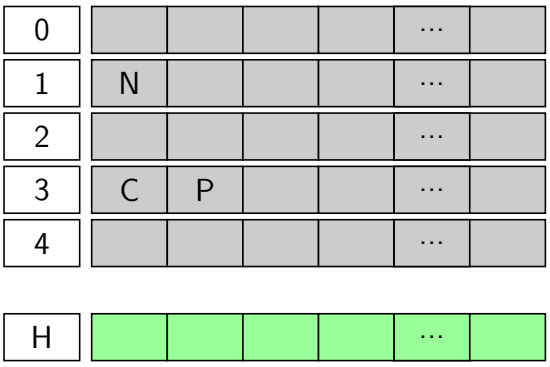
A.3 Passo 3

Linha lida do arquivo de entrada: `'INFO 1 "NIVIO 11010100"'`



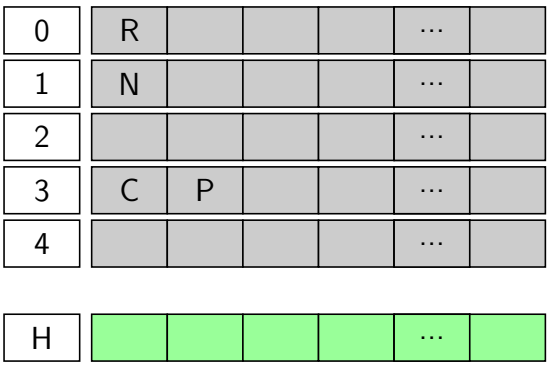
A.4 Passo 4

Linha lida do arquivo de entrada: `'INFO 3 "PRATES 00000010"'`



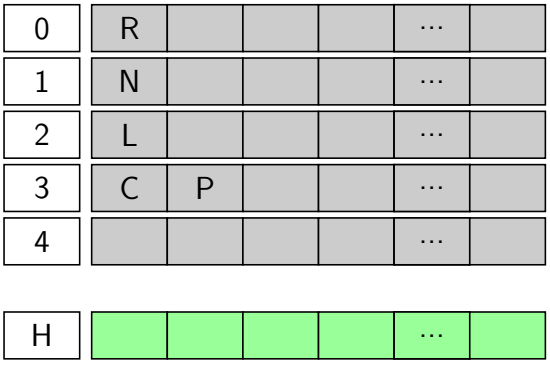
A.5 Passo 5

Linha lida do arquivo de entrada: `'INFO 0 "RAQUEL 10000000"'`



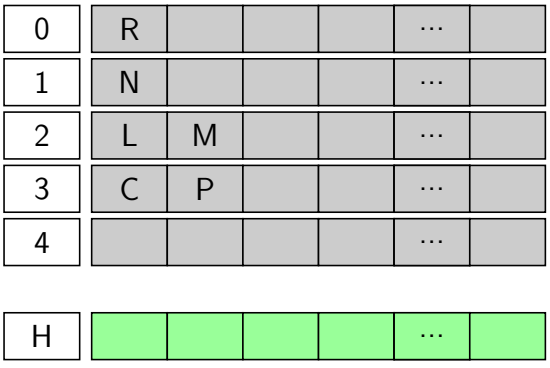
A.6 Passo 6

Linha lida do arquivo de entrada: `'INFO 2 "LUIZ 10000010"'`



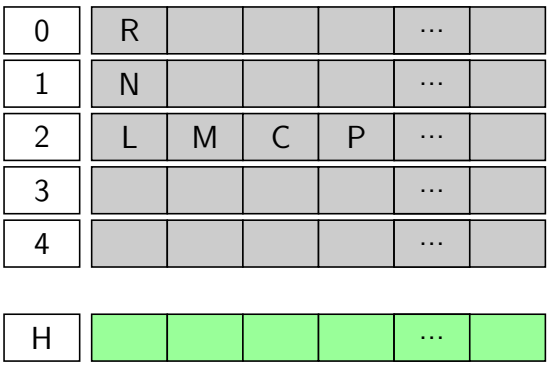
A.7 Passo 7

Linha lida do arquivo de entrada: `'INFO 2 "MARCO 11000110"'`



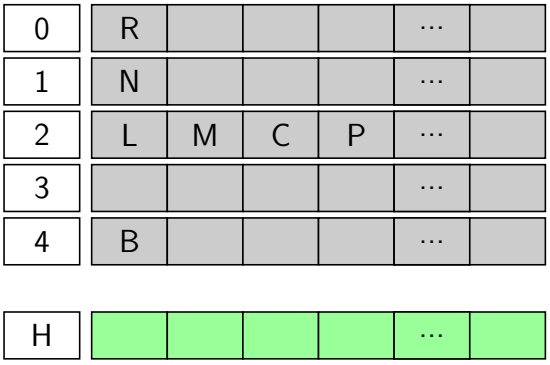
A.8 Passo 8

Linha lida do arquivo de entrada: ‘`TRAN 3 2`’



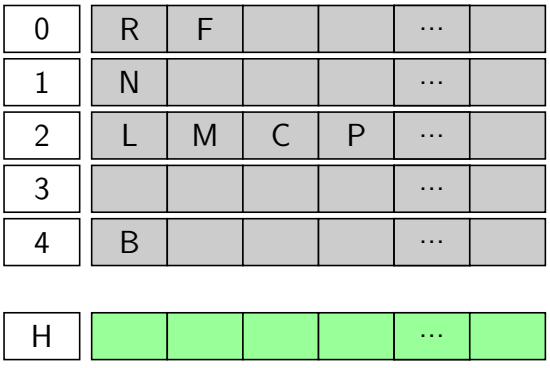
A.9 Passo 9

Linha lida do arquivo de entrada: ‘`INFO 4 "BRENO 11111100"`’



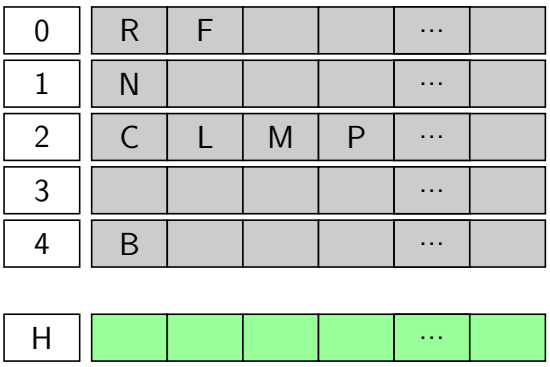
A.10 Passo 10

Linha lida do arquivo de entrada: ‘`INFO 0 "FABIANO 10101110"`’



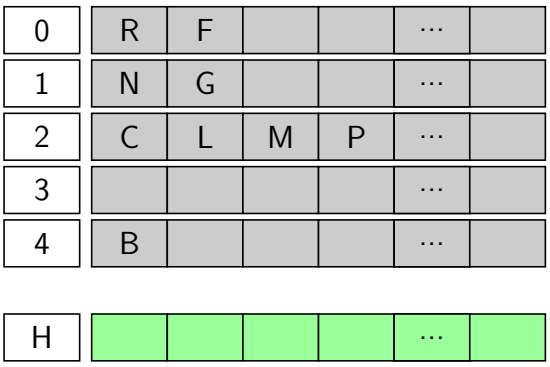
A.11 Passo 11

Linha lida do arquivo de entrada: ‘`WARN 2 2`’



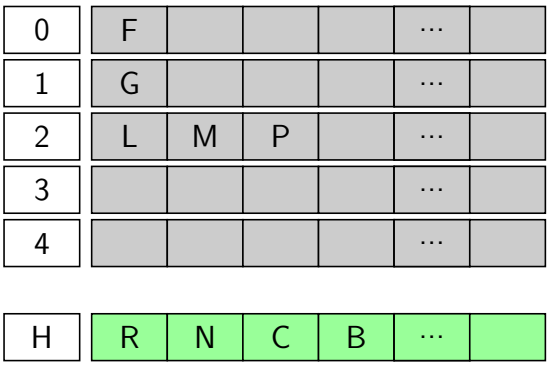
A.12 Passo 12

Linha lida do arquivo de entrada: ‘`INFO 1 "GRERD 00110110"`’



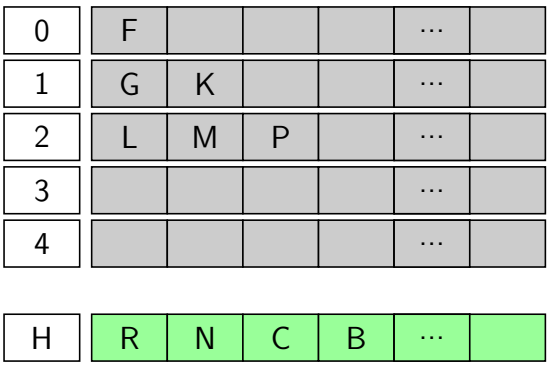
A.13 Passo 13

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



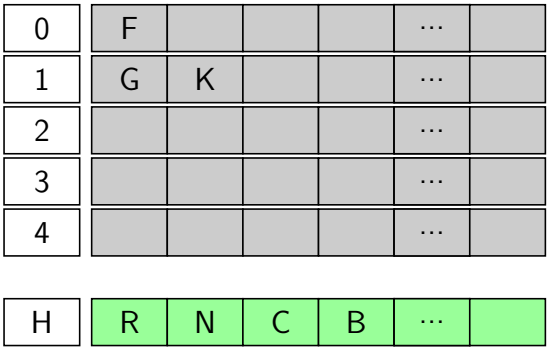
A.14 Passo 14

Linha lida do arquivo de entrada: ‘INFO 1 "KERID 10111010"’



A.15 Passo 15

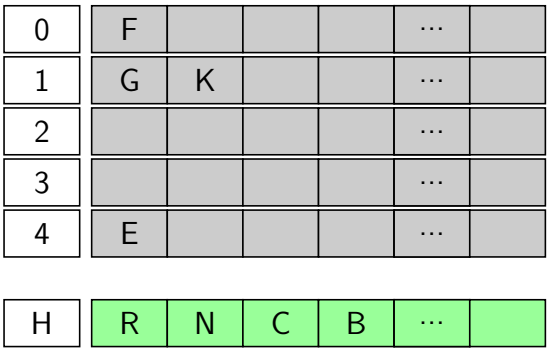
Linha lida do arquivo de entrada: ‘ERRO 2’




```
ERRO 2
LUIZ 10000010
MARCO 11000110
PRATES 00000010
```

A.16 Passo 16

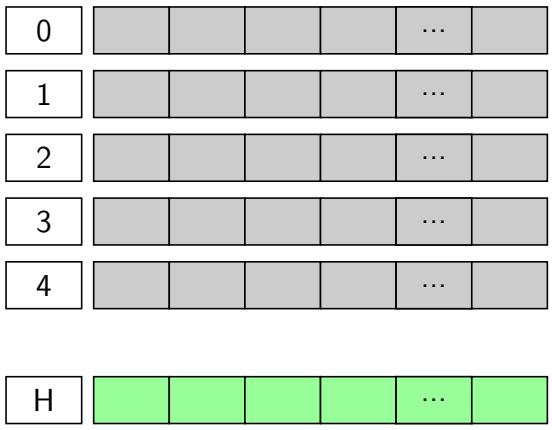
Linha lida do arquivo de entrada: ‘INFO 4 "EDSON 00110100"’




```
ERRO 2
LUIZ 10000010
MARCO 11000110
PRATES 00000010
```

A.17 Passo 17

Linha lida do arquivo de entrada: 'FLUSH'




```
ERRO 2
LUIZ 10000010
MARCO 11000110
PRATES 00000010
RAQUEL 10000000
NIVIO 11010100
CHAIMOWICZ 01011010
BRENO 11111100
FABIANO 10101110
GRERD 00110110
KERID 10111010
EDSON 00110100
```

Após ler e executar o último comando do arquivo de entrada a execução do sistema deve ser finalizada.