UFMG

Estruturas de Dados – Trabalho Prático 1

Realocação Interplanetária 2076

Nome: Arthur Pontes Nader

Matrícula: 2019022294

Universidade Federal de Minas Gerais

Belo Horizonte - MG – Brasil

arthurpn@ufmg.br

1. Introdução

Em um cenário futurístico, a raça humana colonizou diferentes planetas espalhados pelo universo. As consciências humanas podem ser realocadas entre diferentes corpos de cada um desse planetas pelo que é chamado de mega-net. Entretanto, o principal problema disso é que diversos hackers podem acabar roubando essas consciências que trafegam pelo universo e realocá-las para robores escavadores. Assim, deve-se implementar um sistema que seja capaz de fazer essas transferências de consciências de forma confiável e segura.

A manipulação de estruturas de dados é uma habilidade indispensável para cientistas da computação. Tem-se diversos tipos possíveis, como árvores, listas, filas, pilhas, etc. A principal estrutura de dados utilizada nesse trabalho foi a fila, que possui como principal característica a propriedade FIFO (“First in, first out”).

O principal objetivo desse trabalho foi a implementação dessa estrutura de dados em conjunto com classes que representassem as diversas entidades do programa, de tal forma que resolvesse o problema de Realocação Interplanetária apresentado. O programa foi desenvolvido na linguagem C++ e será testado passando como parâmetro um arquivo “.txt”.

A seção 2 explica como compilar e executar o programa. Já as seções 3 e 4 mostram a estruturação do código e a complexidade de cada função criada, respectivamente. A seção 5 apresenta uma breve conclusão e a seção 6 enumera as referências bibliográficas consultadas.

2. Compilação e execução

A compilação e a execução do programa foram realizadas em um computador com processador i3 e 4 GB de memória RAM, possuindo como sistema operacional o Linux Mint.

Primeiramente, adicione os arquivos ".txt" a serem lidos pelo programa na raiz do projeto. Para compilar o programa desenvolvido, acesse o diretório do programa e digite "make" no terminal de comando.

Já para executar, deve-se digitar "./bin/run.out", um espaço (" ") e digitar em seguida o nome de um arquivo ".txt" localizado no diretório. Esse arquivo deve conter os comandos válidos a serem realizados pelo programa.

Para finalizar o processo, digite "make clean" no terminal para limpar os arquivos gerados.

3. Descrição da implementação

Para implementação do programa, criou-se diversos tipos abstratos de dados que representassem cada uma das entidades que compõem as especificações do problema de realocação interplanetária a ser resolvido. A seguir, é apresentado as principais características de cada um desses TADs, uma breve descrição dos métodos implementados para cada TAD e algumas observações sobre a "main" e sobre o modo como o programa reage a entradas inválidas.

- **Classe Consciencia:**

  Essa classe é responsável por instanciar as consciências a serem realocadas pelo sistema. Cada uma dessas consciência terá uma string que armazenará o conteúdo (dados) da consciência em questão. Essa classe possui 3 métodos:

  Consciencia(): construtor padrão de objetos da classe.

  Consciencia(std::string dados): construtor que cria uma consciência que tem como conteúdo os dados passados como parâmetro.

  void exibirConsciencia(): exibe o conteúdo da consciência analisada.

- **Classe Capsula:**

  Para serem realocadas, as consciências precisam estar abrigadas em uma cápsula. Essas cápsulas são usadas para encadeamento de uma consciência em uma fila (Buffer) de realocação interplanetária. A cápsula,

portanto, além da consciência alojada, possui um ponteiro para a próxima cápsula da sequência. Os métodos definidos nessa classe são:

Capsula(): construtor padrão de objetos do tipo Capsula.

Capsula(std::string dados): com os dados passados como parâmetro, cria uma consciência diretamente na cápsula. Além disso também aponto o atributo “proxima” para um valor nulo.

void mostrarConscienciaEncapsulada(): mostra o conteúdo da consciência que está presente na cápsula.

std::string retornarDadosDaConsciencia(): retorna um tipo string que contém os dados da consciência presente na cápsula.

- **Classe Buffer:**

Essa classe é responsável por instanciar uma fila que conterá a ordem das consciências a serem realocadas. Ela é composta por um portal (parecido com uma célula cabeça) que aponta para a primeira consciência da fila. Diversos métodos foram implementados para realizar operações sobre um objeto de Buffer:

Buffer(): o construtor de Buffer cria o portal de realocação, define o tamanho do buffer como zero e aponta o atributo “ultima” para o portal, indicando que não há consciências a serem realocadas a princípio.

~Buffer(): o destrutor exclui todas as consciências do buffer e deleta o portal.

void inserirCapsula(std::string dados): cria um objeto da classe Consciencia diretamente em uma cápsula e a adiciona ao final da fila. O atributo “ultima” agora aponta para essa nova cápsula inserida.

void priorizarConsciencia(int localizacao): a consciência localizada na posição passada como parâmetro é movida para frente do portal, recebendo assim prioridade máxima de realocação.

void enviarTodasConsiencias(): envia todas as consciências pelo portal de realocação, imprimindo o conteúdo de cada uma delas e esvaziando o buffer.

std::string realocarConsciencia(): realiza a realocação da primeira consciência do buffer. Assim, a consciência é retirada da fila e em seguida é deletada.

int retornarTamanho(): retorna o tamanho atual do buffer

- **Classe ComandoCentral:**

O ComandoCentral é composto por um histórico de realocações e um conjunto de objetos Buffer. Esse conjunto será denominado servidores. Um método de um objeto de ComandoCentral será acionado sempre que um comando for lido do arquivo de entrada. Assim, o ComandoCentral realizará a operação equivalente sobre um dos servidores ou sobre todos eles, dependo do comando recebido. Os principais métodos definidos são:

ComandoCentral(int tamanho): o construtor da classe inicia um array de objetos do tipo Buffer com tamanho igual ao parâmetro passado.

void inserirEmServidor(int servidor, std::string dados): esse método seleciona um determinado servidor (que é do tipo Buffer) e chama o método inserirCapsula(std::string dados). Assim, a consciência é colocada no final desse servidor.

void anteciparConsciencia(int servidor, int posicao): seleciona o servidor do parâmetro passado e acionada o método de priorizar consciência para ele, passando como parâmetro a posição da consciência a ser mandada para frente do portal de realocação.

void desabilitarServidor(int servidorDesab, int servidorRecep): utiliza-se um laço “while” para que, enquanto o servidor a ser desabilitado não estiver vazio, chama-se o método realocarConsciencia() para ele. Assim, o conteúdo da consciência retornado por esse método é inserido ao final do servidor que irá receber a consciência. A identificação do servidor a ser desabilitado e do servidor receptor das consciências são passados como parâmetros para o método.

void tratarErro(int servidor): seleciona o servidor referente ao parâmetro passado e chama o método enviarTodasConsiencias() para ele.

void enviarPrimeiroDeCadaServidor(): em uma laço de repetição “for”, esse método envia a primeira consciência de cada servidor pelo portal de realocação. O conteúdo de cada consciência realocada é armazenado no histórico.

void imprimirHistoricoMaisTodosServidores(): imprime os dados salvos no histórico e, em seguida, utilizando um laço “for”, chama o método enviarTodasConsiencias() para cada um dos servidores criados.

- **Main:**

Na “main” ocorrerá a abertura do arquivo a ser lido. Primeiramente, lê-se o número de servidores e se cria um objeto de ComandoCentral chamado “sistema”, passando o número de servidores como parâmetro. A leitura dos comandos a serem transmitidos para o objeto “sistema” ocorrerá por meio de um “while” que lê linhas do arquivo até o seu fim. Essas linhas são formatadas para que as funções associadas ao objeto de ComandoCentral sejam acionadas com os parâmetros corretos. Em seguida, após realização de todas as operações, fecha-se o arquivo e o programa é encerrado.

- **Entradas imprevistas:**

Entradas com comandos não definidos ou com parâmetros que acessem posições inválidas do buffer ou dos servidores são simplesmente ignoradas pelo programa. Optou-se por isso a fim de preservar a integridade do output do programa, já que isso será avaliado pela correção automática.

4. Complexidade de espaço e de tempo

Para facilitar a análise assintótica, definimos duas variáveis m e n, sendo que m representará o tamanho de um determinado buffer e n será o número de servidores do sistema de Realocação Interplanetária.

- **Metodos da Classe Consciencia:**

Consciencia(std::string dados)

Complexidade de tempo: o construtor apenas atribui o parâmetro passado ao atributo “conteudo”, por isso esse método é O(1).

Complexidade de espaço: O(1), pois é criado apenas um objeto do tipo Consciencia.

Consciencia()

Complexidade de tempo: apenas cria um objeto do tipo Consciencia, portanto, O(1).

Complexidade de espaço: O(1), pelo mesmo motivo da complexidade de tempo.

void exibirConsciencia()

Complexidade de tempo: como se trata apenas da impressão de uma string, esse método é O(1).

Complexidade de espaço: nenhum espaço extra é utilizado, por isso esse procedimento é O(1).

- **Métodos da Classe Capsula:**

Capsula(std::string dados)

Complexidade de tempo: esse construtor apenas atribui o parâmetro “dados” ao conteúdo da consciência presente na cápsula, sendo assim, sua complexidade é O(1).

Complexidade de espaço: instanciar um objeto desse tipo tem custo de espaço constante, por isso o construtor é O(1).

Capsula()

Complexidade de tempo: construtor padrão apenas cria um objeto da classe, por esse motivo, é O(1).

Complexidade de espaço: o espaço utilizado na criação de um objeto é constante. Assim, o construtor é O(1).

void mostrarConscienciaEncapsulada()

Complexidade de tempo: apenas imprimi o conteúdo da consciência encapsulada, sendo assim, a complexidade de tempo é O(1).

Complexidade de espaço: nenhum espaço extra é alocado, portanto, O(1).

std::string retornarDadosDaConsciencia()

Complexidade de tempo: retornar os dados da consciência é feito em apenas um passo, o que indica que a complexidade associada é O(1).

Complexidade de espaço: O(1), pois nenhum espaço adicional é preciso para retornar os dados.

- **Métodos da Classe Buffer:**

Buffer()

Complexidade de tempo: apenas inicializa os atributos do objeto criado, portanto, o método é O(1).

Complexidade de espaço: o custo de espaço para instanciar um novo Buffer é constante, por isso, O(1).

~Buffer()

Complexidade de tempo: O(m), pois se deve iterar sobre cada consciência do buffer para desalocar a memória de cada uma delas.

Complexidade de espaço: como se trata de esvaziar um buffer, esse método na verdade libera espaço, sendo que por isso, ele é O(1).

void inserirCapsula(std::string dados)

Complexidade de tempo: como tem-se um ponteiro para a última cápsula do buffer, esse procedimento correspondente a manipular ponteiros, encadear a nova consciência em uma cápsula no final do buffer e atualizar o atributo “ultima”, portanto, é O(1).

Complexidade de espaço: a inserção de uma nova cápsula apenas instancia um novo objeto do tipo Consciencia, por isso, é O(1).

void priorizarConsciencia(int localizacao)

Complexidade de tempo: no pior caso, a consciência a ser priorizada é a última do buffer. Como deve-se iterar sobre todas as outras anteriores, essa função é O(m).

Complexidade de espaço: como só há manipulação de ponteiros, esse procedimento tem complexidade de espaço O(1).

void enviarTodasConsiencias()

Complexidade de tempo: nesse método, deve-se iterar sobre todas as consciências presentes no buffer, sendo sua complexidade, portanto, O(m).

Complexidade de espaço: esse método libera memórias alocadas anteriormente. Assim, sua complexidade é O(1).

std::string realocarConsciencia()

Complexidade de tempo: envolve apenas a manipulação de ponteiros e o retorno de uma string, por isso, tem complexidade de tempo O(1).

Complexidade de espaço: esse método libera espaço, tendo portanto, complexidade igual a O(1).

int retornarTamanho()

Complexidade de tempo: O(1), pois apenas retorna o tamanho atual do buffer.

Complexidade de espaço: O(1), já que nenhuma memória nova é necessária para realizar o procedimento.

- **Métodos da Classe ComandoCentral:**

ComandoCentral(int tamanho)

Complexidade de tempo: como são feitas somente atribuições aos atributos do novo objeto, o construtor possui complexidade O(1).

Complexidade de espaço: como deve-se criar um array de objetos Buffer a partir do tamanho passado como parâmetro, esse construtor possui complexidade de espaço O(n).

void inserirEmServidor(int servidor, std::string dados)

Complexidade de tempo: o acesso a um determinado servidor é feito diretamente pelo índice e o método chamado tem custo constante. Portanto, a complexidade de tempo é O(1).

Complexidade de espaço: apenas cria um objeto da classe Consciencia, o que possui custo constante. Assim, esse método possui complexidade de espaço O(1).

void anteciparConsciencia(int servidor, int posicao)

Complexidade de tempo: o método aciona um procedimento que é O(m) para um determinado servidor. Portanto, o método é O(m) para a complexidade de tempo.

Complexidade de espaço: como só há manipulação de ponteiros, o método é O(1).

void desabilitarServidor(int servidorDesab, int servidorRecep)

Complexidade de tempo: esse método chama funções de custo O(1) repetidas vezes até que o tamanho do buffer a ser desabilitado seja igual a zero. Portanto, a complexidade associada é O(m).

Complexidade de espaço: como esse método deleta e cria consciências na mesma proporção, sua complexidade de espaço é O(1).

void tratarErro(int servidor)

Complexidade de tempo: todas as consciências de um determinado servidor são enviadas, assim, esse método é O(m).

Complexidade de espaço: liberar espaço tem complexidade de espaço O(1).

void enviarPrimeiroDeCadaServidor()

Complexidade de tempo: esse método itera sobre cada servidor realizando operações de custo O(1). Assim, sua complexidade é O(n).

Complexidade de espaço: O(1), pois nenhuma memória extra precisa ser usada para realizar o procedimento.

void imprimirHistoricoMaisTodosServidores()

Complexidade de tempo: o procedimento itera sobre todas as consciências de cada servidor. Nesse caso, o método é O(mn).

Complexidade de espaço: esse método libera memória, portanto, tem custo constante. Sendo assim, o método é O(1).

- **Main:**

Complexidade de tempo: o programa principal pode chamar cada uma das funções criadas diversas vezes. Sua complexidade de tempo será limitada pelo método de maior custo computacional. Assim, o programa principal é O(mn).

Complexidade de espaço: uma instância de ComandoCentral é criada apenas uma vez, por isso, o programa principal será O(n) para a complexidade de espaço.

5. Conclusão

A realização do trabalho prático possibilitou uma boa aplicação dos conceitos ensinados na disciplina Estruturas de Dados. Saber como manipular uma fila utilizando ponteiros foi um fator decisivo para que as funcionalidades exigidas fossem devidamente implementadas.

A produção do programa ocorreu sem muitas dificuldades, sendo que todos os casos testes disponibilizados produziram as saídas esperadas quando executados pelo código implementado. O maior desafio encontrado foi na desabilitação de um servidor (comando “TRAN”), pois a manipulação de ponteiros nesse caso estava levando o programa a produzir resultados indesejados. Entretanto, isso foi resolvido rapidamente.

Enfim, a história desenvolvida (realocação interplanetária de consciências) fez com que a implementação ficasse mais interessante, pois se tinha um objetivo bem definido a ser cumprido, ao invés de simplesmente ficar manipulando ponteiros. Essa abstração computacional de um problema bem definido aumenta significativamente a motivação para encontrar uma solução que atenda a todas especificações exigidas. Além disso, a realização do trabalho possibilitou um acréscimo expressivo no conhecimento sobre estruturas de dados.

6. Bibliografia

- Ziviani, N**. Projeto de Algoritmos com Implementações em Pascal e C**. 3ª edição, Cengage Learning, 2011.
- Cormen, T., Leiserson, C., Rivest, R., Stein, C**. Introduction to Algorithms**. Third Edition, MIT Press, 2009.