Documentação do trabalho prático da disciplina de Estrutura de Dados

Caio César Silva
Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) Belo Horizonte – MG – Brazil
caiocs2019@ufmg.br

1. Introdução

Esta documentação se trata do trabalho prática de Estrutura de Dados, que visa a realização de um projeto que irá realizar a gestão de um sistema de robôs que tem como objetivo explorar um planeta outros planetas em busca do Composto Z, através da mineração e eliminação de formas de vida hostis encontradas no planeta em que se encontra a base. O nome da missão é Extração-Z.

A solução foi implementada utilizando estruturas de dados primitivas e uma abstração através do conceito de programação orientada a objetos. Nesta documentação irei fornecer detalhes sobre implementação, um tutorial de como compilar e executar o projeto, além de uma análise de complexidade de tempo e espaço das funções e procedimentos mais importantes do projeto.

Organizei a documentação da seguinte forma, na seção 2 consta os detalhes da implementação, na seção 3 as instruções de compilação e execução do projeto, na seção 4 é apresentado a análise de complexidade e na 5ª e última seção a conclusão obtida deste projeto.

2. Implementação

No projeto, foram criadas e utilizadas 7 classes, de forma que cada uma possua sua responsabilidade.

A primeira classe é a **Mapa,** que irá definir as propriedades e responsabilidades do mapa lido no arquivo de entrada “mapa.txt”. Ela possui alguns atributos como o número de linhas e colunas do mapa, o arquivo do mapa e a matriz que irá representar o mapa. Essa matriz é uma matriz de elementos do tipo char, que foi alocada dinamicamente de acordo com os valores lidos no arquivo de entrada. Todos os métodos do mapa são chamados no construtor da classe, pois não há necessidade de chamá-los separadamente. O primeiro método chamado é o

DefineTamanhoMapa que vai ler a primeira linha do arquivo mapa.txt usando a função **fscanf** e vai atribuir os números de linhas e colunas aos respectivos atributos da classe, que são inteiros. Após isso, chamamos o método **AlocaMapa,** que aloca dinamicamente a matriz de char usando através da função **malloc.** Após isso, chamamos a função **PreencheMapa,** que vai realizar uma iteração para cada elemento lido do arquivo do mapa e vai atribuir ele ao respectivo elemento da matriz alocada dinamicamente na função **AlocaMapa.** Também há uma função que permite classes externas a realizarem atribuição/recuperação de valores em um mapa de alguma instância dessa classe, que são, respectivamente a **SetDadoMapa** (recebe as posições e o valor a ser atribuido) e a **GetDadoMapa** (recebe as posições como parâmetro e retorna o valor da matriz desejado). Há também um método chamado **LimpaMapa,** que irá liberar o espaço de memória alocado pela matriz mapa através da função **free**.

Temos a classe **BaseComando,** que recebe o arquivo de comandos e o nome desse arquivo como parâmetro para seu construtor, onde é definido que o número de aliens e recursos coletados da base é 0, assim como atribuímos o nome do arquivo e o próprio arquivo aos respectivos atributos da classe. Os demais métodos e atributos serão introduzidos conforme a explicação. O primeiro método a ser chamado é o método **DelegaOrdens** (recebe um array de Robôs e o mapa como parâmetro)**,** que irá ler cada linha do arquivo de comandos e chamar a função **AnalisaComando** para cada uma. A função **AnalisaComando** (recebe a linha, array de Robôs e o mapa como parâmetro) irá criar uma instância da classe **Ordem**, para cada linha, chamando a função **AnalisaOrdem** (recebe a linha como parâmetro), após isso, irá verificar se a ordem é uma ordem direta ou ordem de comando. Caso seja uma ordem direta, ele irá mandar o robô a qual essa ordem é direcionada executar a ordem imediatamente, caso contrário, ele irá verificar se a ordem é prioritária ou não, e vai definir se vai inserir no início da fila de comandos ou na posição a qual ela pertenceria normalmente. A função **AnalisaOrdem** irá procurar por palavras chave na linha, que definem se a ação é ou não uma ordem direta, e se é ou não uma ordem prioritária. Através do uso de alguns métodos utilitários como **VerificaDigitoRobo**, **VerificaNumero**, **VerificaDigitoColuna** e **VerificaDigitoLinha**, irá definir para qual robô

a ordem é direcionada e para qual posição do mapa ela está se referindo, assim, o método cria a ordem e retorna ela. Esta classe possui alguns outros métodos, como o **ImprimeRelatorioFinal,** que irá imprimir a quantidade de aliens e recursos enviados pelos robôs ao todo, os métodos **GetAliensEliminados**, **GetRecursosColetados**, **AdicionaAliensEliminados** e **AdicionaRecursosColetados** servem apenas para adicionar ou recuperar os valores destes atributos **recursosColetados** e **aliensEliminados.**

A classe **Robo** é a classe que representa os robôs, que possui o método **ProcessaComando** (recebe a ordem, o mapa e a base como parâmetros), e irá analisar a ordem, de forma que se for uma ordem direta, ele irá tomar alguma ação, como executar ela, ou apenas imprimir algo. Temos também o método **ExecutaComando** (que recebe a ordem e o mapa com parâmetros) e vai identificar qual a tarefa desta ordem e vai executá-la, armazenando o relatório no atributo **histórico,** que é uma **FilaEncadeada.** Este método também altera os valores dos recursos e aliens coletados pelo robô e move ele também, pois ele possui os atributos **posicaoLinha**, **posicaoColuna**, **recursosColetados**, **aliensEliminados.** Esta classe também possui os métodos que possuem acesso aos métodos dos atributos que são filas encadeadas, como **InsereOrdemComPrioridade**, **InsereOrdemSemPrioridade**, **InsereHistorico**, **GetTamanhoFila**, **GetTamanhoHistorico**, **DesenfileiraExecutaItemFila**, **DesenfileiraHistorico**, **ImprimeHistorico**, **LimpaComandos** e **LimpaHistorico,** que somente chamam métodos referentes a filas encadeadas como o atributo **historico** e **filaComandos.** O robô também possui métodos que irão alterar ou recuperar outros valores de seus atributos como o **SetCodigo**, **SetAtivo**, **AdicionaAlienEliminado**, **AdicionaRecursosColetados**, **GetRecursosColetados** e **GetAliensEliminados**.
A classe **Ordem** possui alguns atributos como a **tarefa** (ação principal desta ordem)**, tipoOrdem** (tipo 1 é ordem direta, tipo 2 é ordem de comando), **posicaoLinha, posicaoColuna** (ambos descritos na ordem), **robo** (para o qual a ordem é direcionada), e **prioridade.** Todos os métodos dessa classe são métodos getters e setters, apenas recuperam ou definem os valores destes atributos.

A classe **Relatorio** representa cada item presente no histórico de comandos do robô através de seu atributo relato, ou seja, um conjunto de relatos forma o histórico do robô. Temos os métodos getters e setters para este atributo também.

A classe **FilaEncadeada** é apenas a implementação da estrutura de dados de uma fila encadeada por ponteiros. Ela pode possuir em suas células uma ordem ou um relato, possui métodos para Inserir ambos os tipos de item (**Enfileira**), no caso das ordens, também posso inseri-las no início da fila, caso ela tenha prioridade (**InsereInicio**). Podemos desenfileirar ambos os tipos de objeto (**Desenfileira** e **DesenfileiraHistorico**). Podemos recuperar o tamanho desta fila (**GetTamanho**) e limpá-la para liberar memória também (**Limpa**).

A classe **CelulaFila** representa cada célula da fila encadeada, ela pode receber uma ordem ou um relato, que são atributos dessa classe. Ela também possui um ponteiro que aponta para a próxima célula da fila. Ao inicializar a célula, todos estes atributos iniciam como **NULL**.

Explicando brevemente o funcionamento da função **main,** nós lemos os arquivos enviados como parâmetros, e inicializamos as classes **Mapa** e **BaseComando,** com base nesses arquivos. Logo após isso é feito a alocação de 50 objetos da classe **Robo,** e é realizado uma iteração de 0 a 50 passando a iteração atual como código para o robô e também o defino como inativo, para que as ordens enviadas os ativem individualmente. Logo após isso, eu passo este array de robôs e o mapa para o método **DelegaOrdens** do objeto instanciado da classe **BaseComando,** assim, é executada toda a lógica principal de gestão de comandos que chamam outros métodos internos (já explicados e citados acima) para delegar as ordens a cada robô. Fazendo com que eles executem esta ordem alterando o mapa de acordo com a ação executada e adicionando o número de aliens e recursos coletados desta instância da base. Logo após esse procedimento, é chamado o método que imprime o relatório final da base de comando (que imprime o número total de aliens e recursos), e então, nós liberamos a memória alocada para o histórico e a fila de comandos de cada robô, liberamos a memória alocada para o array de robôs, limpamos a memória alocada da matriz que foi alocada para receber o mapa e

deletamos as instâncias do mapa e da base. Dessa forma, concluímos a execução do nosso projeto.

Configurações do meu computador e do compilador:

- Sistema Operacional: Windows 10 Pro x64
- Linguagem de programação utilizada: C++
- Compilador utilizado: g++ através do terminal WSL com Ubuntu instalado, utilizando o Makefile fornecido pela equipe da disciplina de Estrutura de Dados.
- Processador/RAM: Intel Core i3-9100F 3.60GHz / 8GB RAM

3. Instruções de compilação e execução

- Acesse o diretório raíz do projeto, nele, haverão as pastas **bin**, **include**, **obj**, **src** e o arquivo **Makefile**.
- Utilizando um terminal Linux, certifique-se de que você tenha o make e o g++ instalado, caso não, rode os seguintes comandos:
    - sudo apt-get install build essentials
    - sudo apt-get install make
    - sudo apt-get install g++
- Com o make instalado, rode o seguinte comando:
    - make
- Dessa forma, os arquivos com formato ".o" serão gerados na pasta **obj** e o arquivo **run.out** será gerado na pasta **bin**.
- Agora, navegue a até a pasta **bin** usando o comando:
    - cd bin
- Coloque seus arquivos de entrada nesta pasta com os nomes de **mapa** e **comandos,** representando estas entradas, ambos arquivos devem estar no formato **".txt"**.
- Com os arquivos na pasta **bin** rode o comando:
    - ./run.out mapa.txt comandos.txt
- Com isso, os saída do projeto será impressa no terminal,
- Caso queira utilizar outros arquivos de entrada é só substituir pelos que estão na pasta bin e rodar o comando acima.
- Caso ocorra qualquer problema com sua execução, entre em contato comigo pelo e-mail: caiocs2019@ufmg.br

4. Análise de Complexidade

DefineTamanhoMapa - Complexidade de tempo: Como essa função não realiza, iterações, só 2 operações, a complexidade assintótica de tempo dela é de $\Theta(1)$.

DefineTamanhoMapa - Complexidade de espaço: Como essa função realiza somente duas atribuições, a complexidade assintótica de espaço dela é de $\Theta(1)$, pois é constante.

AlocaMapa - Complexidade de tempo: Essa função possui somente uma iteração, que é com base no número de linhas n, logo, a complexidade assintótica de tempo dela é de $\Theta(n)$.

AlocaMapa - Complexidade de espaço: Essa função vai realizar uma atribuição fora da iteração e n atribuições dentro da iteração, em que n é o número de linhas, logo, a complexidade assintótica de espaço dela é de $\Theta(n)$.

PreencheMapa - Complexidade de tempo: Essa função possui duas iterações aninhadas, com base no número de linhas e colunas do mapa, logo, a complexidade assintótica de tempo dela é $\Theta(n^2)$.

PreencheMapa - Complexidade de espaço: Essa função possui duas iterações aninhadas, com base no número de linhas e colunas do mapa, e a atribuição ocorre dentro do laço mais aninhado, logo, a complexidade assintótica de espaço dela é $\Theta(n^2)$.

LimpaMapa - Complexidade de tempo: Essa função possui somente uma iteração com base no número de linhas n do mapa, logo, sua complexidade assintótica de tempo é $\Theta(n)$.

LimpaMapa - Complexidade de espaço: Essa função possui somente uma iteração com base no número de linhas n do mapa em que é feito a liberação de memória para cada iteração, logo, sua complexidade assintótica de espaço é $\Theta(n)$.

DelegaOrdens - Complexidade de tempo: Essa função possui somente uma iteração com base no número de linhas n da entrada de comandos, e caso a linha lida contenha a ação EXECUTAR, ela irá realizar outra iteração com base no número n de comandos armazenados na fila do robô, além disso, se houver outra ação RELATORIO, ele também irá iterar pelos n comandos já realizados por este robô, e, se HOUVER também o comando RETORNAR, ele irá realizar a limpeza do histórico, que possui complexidade O(n), com isso, a complexidade assintótica de tempo da função seria de Θ(n^4).
DelegaOrdens - Complexidade de espaço: Seguindo a mesma lógica anterior, a complexidade assintótica de espaço da função seria de Θ(n^4), pois, dentro de cada laço, ocorrem atribuições e/ou limpeza de memória, o que implica na alteração da complexidade assintótica de espaço.

AnalisaComando - Complexidade de tempo: Essa função é a função mais próxima dentro das funções filhas da DelegaOrdens, ela dentro dela podem ocorrer 3 iterações, quando uma ação é executada, quando é solicitado a execução, relatório ou o retorno de um dos robôs, com isso, temos que a complexidade assintótica de tempo é de $\Theta(n^3)$.
AnalisaComando - Complexidade de espaço: Seguindo a mesma lógica anterior, a complexidade assintótica de espaço da função seria de $\Theta(n^3)$, pois, dentro de cada laço, ocorrem atribuições e/ou limpeza de memória, o que implica na alteração da complexidade assintótica de espaço.

AnalisaOrdem - Complexidade de tempo: Essa função possui complexidade assintótica de tempo de Θ(n), pois ela irá iterar por alguns dígitos da linha que estiver lendo, mas nenhuma dessas iterações são aninhadas.
AnalisaOrdem - Complexidade de espaço: Seguindo a mesma lógica anterior, a complexidade assintótica de espaço é de Θ(n), pois ocorrem atribuições nestas iterações, e nenhuma delas é aninhada.

ProcessaComando - Complexidade de tempo: Essa função possui complexidade assintótica de tempo de Θ(n), pois ela irá iterar pelo histórico do robô imprimindo e limpando ele, e por cada ordem armazenada na fila do robô, mas nenhuma iteração é aninhada.
ProcessaComando - Complexidade de espaço: Da mesma forma, a complexidade assintótica de espaço é de Θ(n), pois ocorrem atribuições e limpezas de memória nestas iterações, e nenhuma delas é aninhada em outra.

ExecutaComando - Complexidade de tempo: Essa função possui complexidade assintótica de tempo de Θ(1), pois não ocorre iteração nenhuma dentro de seu escopo e funções filhas, sendo constante.
ExecutaComando - Complexidade de espaço: Da mesma forma, a complexidade assintótica de espaço é de Θ(1), pois não há nenhuma iteração, ocorrem mais de uma atribuição, mas nenhuma delas aumenta a complexidade devido ao fato de não estarem aninhadas.

Enfileira - Complexidade de tempo: Essa função possui complexidade assintótica de tempo de Θ(1), pois não ocorre iteração nenhuma dentro de seu escopo e funções filhas, sendo constante.
Enfileira - Complexidade de espaço: Da mesma forma, a complexidade assintótica de espaço é de Θ(1), pois não há nenhuma iteração.

Desenfileira/DesenfileiraHistorico - Complexidade de tempo: Essa função possui complexidade assintótica de tempo de Θ(1), pois não ocorre iteração nenhuma dentro de seu escopo e funções filhas, sendo constante.
Desenfileira/DesenfileiraHistorico - Complexidade de espaço: Da mesma forma, a complexidade assintótica de espaço é de Θ(1), pois não há nenhuma iteração.

Limpa - Complexidade de tempo: Essa função possui complexidade assintótica de tempo de Θ(n), pois só ocorre uma iteração pelas células da fila encadeada.
Limpa - Complexidade de espaço: Da mesma forma, a complexidade assintótica de espaço é de Θ(n), pois a cada iteração há a liberação de um espaço de memória.

ImprimeHistorico - Complexidade de tempo: Essa função possui complexidade assintótica de tempo de Θ(n), pois só ocorre uma iteração pelas células da fila encadeada recuperando o relato da célula iterada.
ImprimeHistorico - Complexidade de espaço: Da mesma forma, a complexidade assintótica de espaço é de Θ(n), pois a cada iteração há a atribuição da célula atual recebendo o ponteiro para a próxima célula.

InsereInicio - Complexidade de tempo: Essa função possui complexidade assintótica de tempo de Θ(1), pois não ocorre nenhuma iteração.
InsereInicio - Complexidade de espaço: Da mesma forma, a complexidade assintótica de espaço é de Θ(1), pois a atribuição ocorre de forma constante.

main - Complexidade de tempo: Essa função possui complexidade assintótica de tempo de Θ(n^4), pois a função DelegaOrdens é executada dentro da main, mas não está aninhada dentro de nenhum outro laço, e a complexidade assintótica dela é a maior de todas, logo, isso faz com a main tenha a mesma complexidade.
main - Complexidade de espaço: Da mesma forma, a complexidade assintótica de espaço é de Θ(n^4), pois a complexidade vem da função DelegaOrdens.

5. Conclusão

Este trabalho buscou a realização de um projeto que irá realizar a gestão de um sistema de robôs que tem como objetivo explorar um planeta outros planetas em busca do Composto Z, foi utilizado estruturas de dados como Filas Encadeadas e uma arquitetura de orientação a objetos, de forma que cada entidade tivesse sua responsabilidade. Foi visto que o uso de POO é algo que ajuda bastante na implementação e estruturação da solução, por mais que eu não tenha um conhecimento muito aprofundado nessa questão, consegui perceber os benefícios que o seu uso tem em um projeto, da mesma forma que o uso específico de certas estruturas de dados, como filas encadeadas por ponteiros, se usadas corretamentes podem ser muito efetivas para alcançar os objetivos desejados. Eu tive bastante dificuldade com a passagem de referências e com a limpeza da memória, e sinto que poderia ter melhorado a arquitetura, mas me falta experiência e conhecimento para conseguir implementar algo mais consistente do que foi implementado.