Documentação do Trabalho Prático I da disciplina de Estruturas de Dados

Lucca Silva Medeiros

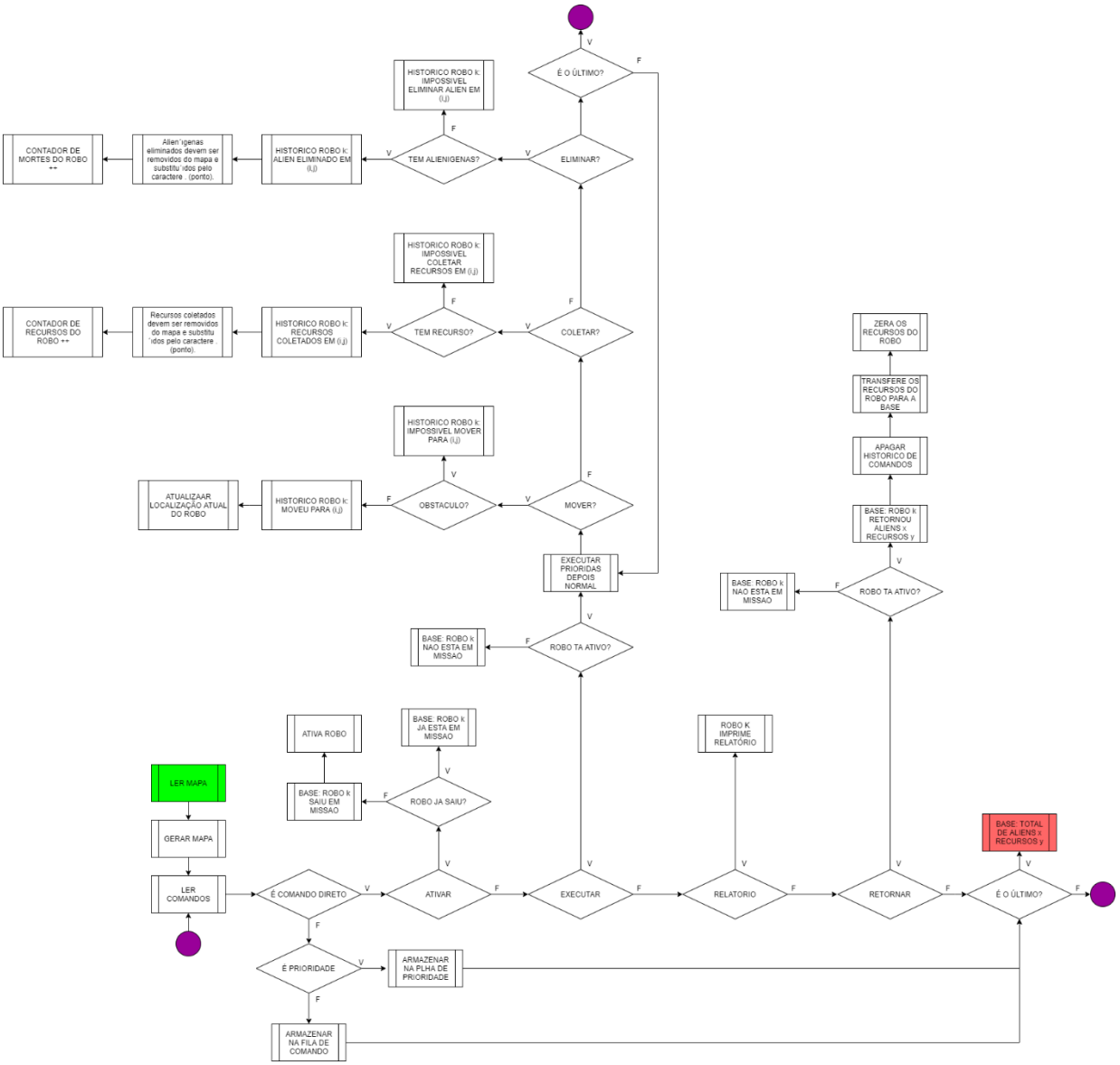
Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

luccasilva@dcc.ufmg.br

1. Introdução

Esta documentação lida com o problema da exploração do Composto Z, material essencial para desenvolver as tecnologias capazes de restaurar as condições de vida em uma Terra pós-apocalíptica. Haja vista essa situação, foi desenvolvido um software capaz de controlar 50 robôs que serão destinados à missão de exploração do planeta cujo Composto Z está presente. Esta documentação tem como objetivo detalhar a implementação do programa desenvolvido, apresentando a ideação da solução, a organização do código e explicando as principais funções (com suas respectivas análises de complexidade) e estruturas de dados que compõem o programa. Além disso, apresentar as instruções de compilação e execução do programa.

Para resolver o problema supracitado foi seguida uma abordagem guiada pelo seguinte Fluxograma:



Esta documentação está separada em seções. Na segunda é apresentado a implementação da solução do problema. Na terceira as instruções de compilação e execução do programa. Na quarta a Análise de complexidade de tempo e espaço de cada função e por último a conclusão do Trabalho.

2. Implementação

O código está organizado nos objetos listados a seguir:

- Base: responsável por armazenar a quantidade de Aliens e recursos que retornam da exploração de cada Robô. Além de ser a responsável por finalizar a exploração e emitir a mensagem final.

- Map: responsável por ler e armazenar o mapa da missão. Haja vista que o mapa deve ser constantemente atualizado e verificado (uma vez que Robôs não podem andar em cima de obstáculos ou que ao coletar um recurso o mesmo não poderá ficar disponível no mapa) esse objeto conta também com métodos de atualização e checagem do mapa original.

- Stack: estrutura de dado do tipo Pilha, responsável por armazenar as ordens de comando sinaladas com "*", uma vez que uma ordem desse tipo tem prioridade sobre as demais e necessariamente deverá ser a próxima a executar. Portanto a pilha é a estrutura de dado mais adequada para esse tipo de situação uma vez que o comando com prioridade mais recente será empilhado sobre os demais e ao executar será o primeiro a ser desempilhado.

- Queue: estrutura de dado do tipo Fila, responsável por armazenar as ordens de comando sem prioridade e o histórico de execução de cada Robô. A fila é a estrutura de dado mais adequada para esse tipo de situação uma vez que as ordens de comando e o histórico seguem uma política *first in first out (FIFO)* onde o próximo dado recebido sempre deve ser armazenado no final da mesma e a leitura é feita no início.

- Robot: principal objeto do programa, responsável por armazenar a pilha de ordens de comando com prioridade, a fila de ordens de comando, a fila do histórico de execução e os atributos básicos de cada Robô: (estado de ativação, quantidade de Aliens, recursos armazenados e posição atual no mapa). Esse objeto também possui todos os métodos de execução das ordens de comando (MOVER, COLETAR e ELIMINAR) e todos os métodos de execução das ordens diretas (ATIVAR, EXECUTAR, RELATORIO e RETORNAR).

Todos os objetos foram declarados separadamente nos respectivos arquivos.h presentes na pasta include. Os métodos de cada objeto estão implementados separadamente nos respectivos arquivos.cc presentes na pasta src. Ao compilar os arquivos.o de cada objeto estarão disponíveis na pasta obj.

A função main está no arquivo Main.cc, presente na pasta src. Ela é responsável por ler os argumentos passados pelo usuário (mapa e comandos) e realizar suas respectivas distribuições e separações ao longo do programa. O primeiro arquivo aberto é o mapa, onde a primeira linha do arquivo estipula a dimensão da matriz que o representa. Essas informações são coletadas e passadas para o construtor do objeto Map, que aloca dinamicamente a matriz e posteriormente a preenche com os *char* que simbolizam cada casa (método readMap).

Posteriormente a função readCommands lê todos os comandos do arquivo e, a cada leitura, chama a função identifyCommand que identifica qual comando é qual. Caso o comando atual for uma Ordem de Execução, a função imediatamente chama o método do robô identificado pelo número presente no comando, executando a ação necessária. Caso o comando atual for uma Ordem de Comando, a função identifica primeiramente se possui ou não a sinalização "*". Caso possua ela empilha o comando na pilha de comandos com prioridade do robô identificado pelo número presente na leitura. Caso contrário ela lista o comando na lista de comandos comuns do respectivo robô. Assim o programa executa até o último comando presente no arquivo.

Ao finalizar a leitura de todos os comandos é chamado o método de Base: endExploration que emite a mensagem final: BASE: TOTAL ALIENS X RECURSOS Y. Após esse passo os destrutores dos objetos são acionados, desalocando o mapa, os 50 robôs e a base, finalizando a execução do programa.

A configuração utilizada para testar o programa está especificada abaixo:

– **Sistema Operacional**: Windows 10 Home

– **Linguagem de programação implementada:** C++

– **Compilador utilizado:** mingw / G++

– **Processador**: Inter(R) Core(TM) i7-8750H CPU @2.20GHz 2.21GHz

– **Quantidade de memória RAM:** 16,0 GB

3. Instruções de compilação e execução

1- Baixe o arquivo.zip e extraia a pasta tp1;

2- A pasta deverá conter os seguintes arquivos;

/ bin – run.out, comandos.txt e mapa.txt

/ include – Base.h, Map.h, Queue.h, Robot.h e Stack.h

/ obj

/ src – Base.cc, Main.cc, Map.cc, Queue.cc, Robot.cc e Stack.cc

/ Makefile

3- Utilizando um terminal, abra o diretório da pasta tp1 e execute o arquivo [Makefile] utilizando o seguinte comando: < make >;

4- Com esse comando o Makefile irá compilar o programa e deverá criar com sucesso os objetos: ./obj/Main.o ./obj/Stack.o ./obj/Robot.o ./obj/Base.o ./obj/Map.o ./obj/Queue.o.

5- Utilizando um terminal, abra o diretório da pasta bin e execute o arquivo [run.out] utilizando o seguinte comando: < run.out argumento1.txt argumento2.txt >. O argumento 1 deverá ser o arquivo.txt referente ao **MAPA** e o argumento 2 deverá ser o arquivo.txt referente a **LISTA DE COMANDOS**.

Caso preferir poderá preencher os arquivos mapa.txt e comandos.txt com os valores desejados e executar o programa com o seguinte comando: < run.out mapa.txt comandos.txt >;

6- Com isso o programa irá executar e a saída esperada aparecerá no terminal.

4. Análise de Complexidade

Objeto Stack:

Função Node – complexidade de tempo – essa função realiza apenas operações constantes de atribuição em tempo O(1).

Função Node – complexidade de espaço – essa função realiza apenas a alocação em espaço O(1).

Função Stack – complexidade de tempo – essa função realiza apenas operações constantes de atribuição em tempo O(1).

Função Stack – complexidade de espaço – essa função realiza apenas a alocação em espaço O(1).

Função popStack – complexidade de tempo – essa função realiza apenas operações constantes em tempo O(1).

Função popStack – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função stackIsEmpty – complexidade de tempo – essa função realiza apenas operações em tempo O(1).

Função stackIsEmpty – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função cleanStack – complexidade de tempo – essa função realiza a chamada da função popStack para cada valor presente na pilha, logo a complexidade de tempo é O(n), sendo n o número de elementos da pilha.

Função cleanStack – complexidade de espaço – a complexidade de espaço é O(n), sendo n o número de elementos da pilha.

Função ~Stack – complexidade de tempo – essa função realiza apenas a chamada da função cleanStack e uma operação constante, portanto sua complexidade de tempo é a mesma de cleanStack: O(n).

Função ~Stack – complexidade de espaço – essa função realiza apenas a chamada da função cleanStack e uma operação constante, portanto sua complexidade de espaço é a mesma de cleanStack: O(n).

Função pushStack – complexidade de tempo – essa função possui complexidade de tempo O(1).

Função pushStack – complexidade de espaço – essa função possui complexidade de espaço O(1)

Função getSize – complexidade de tempo – essa função possui complexidade de tempo O(1).

Função getSize – complexidade de espaço – essa função possui complexidade de espaço O(1).

Objeto Queue:

Função Place – complexidade de tempo – essa função realiza apenas operações constantes de atribuição em tempo O(1).

Função Place – complexidade de espaço – essa função realiza apenas a alocação em espaço O(1).

Função Queue – complexidade de tempo – essa função realiza apenas operações constantes de atribuição em tempo O(1).

Função Queue – complexidade de espaço – essa função realiza apenas a alocação em espaço O(1).

Função toQueue – complexidade de tempo – essa função possui complexidade de tempo O(1).

Função toQueue – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função queueIsEmpty – complexidade de tempo – essa função realiza apenas operações em tempo O(1).

Função queueIsEmpty – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função cleanQueue – complexidade de tempo – essa função realiza o conjunto de operações para cada valor presente na fila, logo a complexidade de tempo é O(n), sendo n o número de elementos da fila.

Função cleanQueue – complexidade de espaço – a complexidade de espaço é O(n), sendo n o número de elementos da fila.

Função ~Queue – complexidade de tempo – essa função realiza apenas a chamada da função cleanQueue e uma operação constante, portanto sua complexidade de tempo é a mesma de cleanQueue: O(n).

Função ~Queue – complexidade de espaço – essa função realiza apenas a chamada da função cleanQueue e uma operação constante, portanto sua complexidade de espaço é a mesma de cleanQueue: O(n).

Função deQueue – complexidade de tempo – essa função possui complexidade de tempo O(1).

Função deQueue – complexidade de espaço – essa função possui complexidade de espaço O(1)

Função getSize – complexidade de tempo – essa função possui complexidade de tempo O(1).

Função getSize – complexidade de espaço – essa função possui complexidade de espaço O(1).

Objeto Base:

Funções Base, ~Base, setKillResources, getBaseKill, getBaseResources e endExploration – complexidade de tempo – possuem complexidade de tempo O(1).

Funções Base, ~Base, setKillResources, getBaseKill, getBaseResources e endExploration – complexidade de espaço – possuem complexidade de espaço O(1).

Objeto Map:

Função checkPlace – complexidade de tempo – essa função possui complexidade de tempo O(1).

Função checkPlace – complexidade de espaço – essa função possui complexidade de espaço O(1)

Função updatePlace – complexidade de tempo – essa função possui complexidade de tempo O(1).

Função updatePlace – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função readMap – complexidade de tempo – essa função possui complexidade de tempo O(i*j), sendo i o número de linhas da matriz e j o número de colunas.

Função readMap – complexidade de espaço – essa função possui complexidade de espaço O(1), um vez que só realiza atribuições em cada espaço da matriz.

Função deleteMap – complexidade de tempo – essa função possui complexidade de tempo O(i), sendo i o número de linhas da matriz.

Função deleteMap – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função Map – complexidade de tempo – essa função possui complexidade de tempo O(i*j), sendo i o número de linhas da matriz e j o número de colunas.

Função Map – complexidade de espaço – essa função possui complexidade de espaço O(i*j), sendo i o número de linhas da matriz e j o número de colunas.

Objeto Robot:

Função Robot – complexidade de tempo – essa função possui complexidade de tempo O(1).

Função Robot – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função Activate – complexidade de tempo – essa função possui complexidade de tempo O(1).

Função Activate – complexidade de espaço – essa função possui complexidade de espaço O(1).

Funções killNow, collectNow, moveNow – complexidade de tempo –complexidade de tempo O(1), uma vez que só possuem operações diretas e realizam chamadas para as funções checkPlace, setHistory e mapUpdate, todas com complexidade também O(1).

Funções killNow, collectNow, moveNow – complexidade de espaço – possuem complexidade de espaço O(1).

Função Execute – complexidade de tempo – essa função possui complexidade de tempo O(i+j), sendo i o número de comandos empilhados na pilha de prioridade e j sendo o número de comandos enfileirados na fila de comandos do robô.

Função Execute – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função Report – complexidade de tempo – essa função possui complexidade de tempo O(n), sendo n o número de comandos enfileirados no histórico do Robô.

Função Report – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função Return – complexidade de tempo – essa função possui complexidade de tempo O(1) no seu melhor caso, que é quando a fila de histórico do robô está vazia ou o robô não está ativo. E em seu pior caso O(n), onde n é o número de comandos enfileirados no histórico do Robô, uma vez que Return chama a função cleanQueue.

Função Return – complexidade de espaço – essa função possui complexidade de espaço O(1).

Funções setKill, getKill, setResources, isActive – complexidade de tempo – complexidade de tempo O(1), uma vez que só possuem operações diretas e/ou de atribuição simples;

Funções setKill, getKill, setResources, isActive – complexidade de espaço – possuem complexidade de espaço O(1).

Funções setCommonCommand, setHistory, setPriorityCommand, deQueueCommand, popPriorityCommand – complexidade de tempo – complexidade de tempo O(1), uma vez que só possuem chamadas das funções toQueue, pushStack, deQueue e popStack, todas com a mesma complexidade O(1).

Funções setCommonCommand, setHistory, setPriorityCommand, deQueueCommand, popPriorityCommand – complexidade de espaço – possuem complexidade de espaço O(1).

Main:

Funções getWidth, getHeight – complexidade de tempo – possuem complexidade de tempo O(1).

Funções getWidth, getHeight – complexidade de espaço – possuem complexidade de espaço O(1).

Função identifyCommand – complexidade de tempo – essa função possui no seu melhor caso complexidade de tempo O(1) se o comando for do tipo ATIVAR, MOVER, COLETAR ou ELIMINAR pois a função irá chamar apenas esses respectivos métodos os quais possuem também complexidade O(1). Mas no pior caso terá complexidade O(n), pois os comandos EXECUTAR, RELATORIO e RETORNAR irão chamar os respectivos métodos os quais possuem complexidade que dependem do tamanho da entrada.

Função identifyCommand – complexidade de espaço – essa função possui complexidade de espaço O(1).

Função readCommands – complexidade de tempo – essa função possui complexidade de tempo O(n), sendo n o número de comandos presente no arquivo de entrada.

Função readCommands – complexidade de espaço – essa função possui complexidade de espaço O(1).

No Programa como um todo:

Função main – complexidade de tempo – o programa terá uma complexidade de tempo geral O(n + i*j) sendo i o número de linhas da matriz do mapa, j sendo o número de colunas e n o número de comandos presentes no arquivo de entrada.

Função main – complexidade de espaço – o programa terá uma complexidade de espaço geral O(n + i*j) sendo i o número de linhas da matriz do mapa, j sendo o número de colunas e n o número de comandos presentes no arquivo de entrada.

5. Conclusão

Este trabalho lidou com o problema da extração do Composto Z, o qual a abordagem utilizada para resolução foi a implementação de um programa em C++ capaz de administrar a exploração de 50 robôs através de um mapa e uma lista de comandos executáveis. Por meio da resolução desse trabalho, foi possível praticar os conceitos relacionadas à disciplina Estrutura de Dados: Pilha e Fila, e ver na prática sua implementação e execução para a solução de um problema palpável. Além de relembrar conceitos relacionados à disciplina de Programação e Desenvolvimento de Software II como programação orientada à Objetos, boas práticas de programação e modularização de código.

6. Referências

Chaimowicz, L. and Prates, R. (2021). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.