Documentação Programa “Extração Z”

Henrique Alves Barbosa

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

1. Introdução
O Programa “Extração Z” é um software de interface por linha de comando que lida com o problema de realizar uma série de comandos decididos previamente em um arquivo, em um mapa, também presente em um arquivo, e exibir os acontecimentos advindos de tais comandos executados.

Foram implementadas no código em sua maioria classes que buscavam representar cada possível entidade existente no sistema em questão. Buscou-se manter em grande parte uma boa arquitetura, mantendo os princípios bases da POO sempre em primeiro lugar, mesmo que significasse criar interfaces de comunicação extras entre classes ou extrair métodos para manter uma maior legibilidade do código a medida do posível.

A resolução do problema baseou-se em separar as entidades do sistema e passar a cada uma delas as responsibilidades que deveriam possuir. O código principal cuida de abrir os arquivos passados pela linha de comando, e criar objetos no mais alto nível de abstração, que por sua conta criarão objetos em mais baixo nível de abstração, modularizando o sistema até o nível de execução das ordens do segundo arquivo passado como argumento no mapa do primeiro arquivo passado como argumento, buscando utilizar as estruturas de dados, sempre que possível, aprendidas até o momento na disciplina Estruturas de Dados ofertada pelo Departamento de Ciências da Computação da UFMG.

O resto da documentação presente segue a seguinte ordem:
- A seção 2 representa as decisões tomadas na hora da implementação do código e as responsabilidades tomadas por cada classe existente.
- A seção 3 apresenta instruções sobre a compilação e execução do código
- A seção 4 apresenta uma análise de complexidade de todos os métodos presentes na seção 2
- A seção 5 apresenta uma conclusão sobre o trabalho, incluindo os aprendizados advindos do mesmo e considerações finais de forma geral.

2. Implementação
O código referente ao programa em questão apresenta-se dividido em cinco pastas e um makefile.

O makefile apresenta as regras de compilação que o comando “make” e “make clean” devem seguir ao serem digitados na linha de comando. Mais explicações de compilação encontram-se na seção 4 da presente documentação.

A pasta “bin” apresenta um único arquivo binário resultado da compilação do código chamado “run.out”, sendo ele o executável do programa em questão.

A pasta “obj” guarda todos arquivos objetos advindos do processo da compilação e utilizados pelo compilador para a criação do executável.

A pasta “include” apresenta todos os headers das classes presentes no programa, assim como a própria implementação de classes Template, devido a forma como o compilador utilizado necessita que sejam implementadas tais classes junto a suas declarações.

A pasta “src” apresenta todas as outras implementações refeferentes ao código, incluindo o arquivo “main.cpp”, que é o arquivo base a ser compilado para a criação do programa.

Por último, uma pasta “files” está presente para que todos os arquivos utilizados pelo programa possam ser mais bem organizados, evitando a necessidade de navegar por diretórios ao passar argumentos ao programa pela linha de comando.

As configurações principais da máquina e softwares usados no desenvolvimento do programa são as seguintes:
OS: Linuz Ubuntu 20.04.1 LTS
Ram: 2 Slots de 16 GB
Processador: AMT ryzen 7 2700 8 núcleos
Linguagem: C++
Compilador: Compilador C++ GNU

O arquivo principal “main.cpp” é de relativa simplicidade, tomando apenas conta de criar dois objetos Input File Stream com os arquivos passados como parametrôs na execução que serão utilizados pela classe Base e Orders, estas que cuidam de toda a lógica de alto nível do programa. Por fim realiza um pequeno resumo de como o objeto da classe Base instanciado se apresenta ao fim da execução.

Uma vez que a abordagem do paradigma de orientação a objetos foi seguido como base de implementação no trabalho em questão, uma análise de cada classe presente, assim como suas variáveis, métodos e possíveis estruturas de dados que representem, apresenta-se vantajosa. Comecemos com as classe sem mais baixa hierarquia no código:

- Matrix: Essa classe têm como função representar um objeto do tipo matriz que possui elementos do tipo string como seus elementos. Assim como o objeto matemático que inspirou sua criação, ela possúi um número fixo de colunas e linhas representados por valores inteiros, além de um ponteiro duplo de strings.

Três constutores apresentam-se em sua implementação:
Matrix(): É o construtor default que, apesar de não ser utilizado no código, foi criado procurando seguir boas práticas da linguagem.
Matrix(std::ifstream &myFile): O construtor realmente utilizado, capaz de ler o arquivo que representa o mapa na formatação definida na explicação do trabalho e gravar os valores em suas respectivas posições
Matrix(const Matrix& that): Um construtor de cópia que, apesar de não ser utilizado, também foi criado proocurando-se manter boas práticas.

Um destrutor “~Matrix()” também está presente, sendo responsável por apagar os ponteiros criados na alocação dinâmica advinda de seu construtos.

Getters, setters, funções de impressão além do overload do operador[] também estão presentes, lembrando-se que um objeto matriz tem sua primeira linha como 1 e a primeira coluna como 1.

- QueueElement: Uma classe simples contendo apenas seu dado satélite e um ponteiro que aponta para outro QueueElement, no caso o próximo da file.

Possúi dois construtores, um contrutor default, que não executa nada, e um que recebe uma variável do tipo específicado na instanciação da classe Template, fazendo o ponteiro next apontar para NULL.
Possúi uma pequena função “print” para imprimir os dados satélites do elemento.
É friend class de Queue e PriorityQueue, classes explicadas a seguir.

- Queue: Uma classe template representando uma estrutura de dados fila, seguindo o padrão “FIFO”, possuindo um ponteiro para o nó inicial, um para o nó final e uma variável lenght representando o tamanho da lista

Possui um construtor que inicia os nós iniciais e finais como NULL, ou seja, não possúi um nó cabeça em sua implementação.

As funções de inserção e deleção realizam a alocação e desalocação de um nó com um atributo chave passado ao método, lembrando que mantém o padrão de “primeiro a entrar, último a sair”. Possui um getter para retornar o número de elementos, além de um método auxiliar de impressão e um de limpar a fila, ou seja, retirar todos nós presentes e atribuir ao seu atributo lenght o valor 0.

- PriorityQueue: Uma classe template que herda de Queue e representa uma estrutura de dados Fila de Prioridade com um único nível de prioridade existente.

Para atribuir a inserção por prioridade é utilizado seu método próprio “insert_with_priority”, que torna o novo nó criado como o último presente na fila, que pela propriedade da fila faz com que, ao retirar um elemento, o último elemento inserido com prioridade seja o primeiro a sair.

- Map: Uma classe que procura abstrair o conceito de mapa apresentado pelo arquivo.

Seu construtor recebe o arquivo do mapa e cria dinamicamente uma matriz com as dimensões correspondentes às passadas no arquivo, preenchendo os espaços com os valores também passados no arquivo

A classe possúi quatro variáveis estáticas que representam qual a string presente na matriz corresponde a um obstáculo, um recurso, um alien e à uma posição vazia.

Possui os seguintes métodos em sua implementação:
.move_to: recebe os parametros x e y correspondentes a posição a buscar e, caso essa posição não possua um obstáculo retorna true, caso contrário retorna false.
.destroy_alien: recebe os parametros x e y correspondentes a posição a buscar e, caso essa posição possua um alien substitui sua representação pela de um espaço vazio e retorna true, caso contrário retorna false.
.colect_resource: recebe os parametros x e y correspondentes a posição a buscar e, caso essa posição possua um recurso substitui sua representação pela de um espaço vazio e retorna true, caso contrário retorna false.
.print: chama o método print da sua variável matriz.

- Robot: Uma classe que apresenta uma entidade do tipo Robô, possuindo um id constante de identificação, sua posição no eixo x e y, número de recursos e aliens capturados, um atributo que revela se está ativo e duas listas de prioridades, uma para ordens de executar e outra de ordens já executadas.

O construtor de sua classe recebe seu id e posições no eixo x e y, inicializando as listas de prioridade dinamicamente e os outros atributos com valores default de 0 ou false, dependendo do seu tipo.

Possui getters padrões para todos seus atributos, excetuando-se as listas de prioridade.

Existem três métodos referentes à sua interação com o mapa. Todos eles recebem um mapa por referencia e uma posição com o eixo x e y. São elas:
.“move_to”, que analisa se a posição do mapa pode ser utilizada, caso seja altera os parâmetros do robô e retorna true, caso contrário apenas retorna false.
.”destroy_alien”, que analisa a posição passada e, caso possua um alien, aumenta o contador de aliens do robô, retira o alien da posição do mapa e coloca a representação de um espaço vazio nessa posição e retorna true, caso contrário apenas retorna false.
.”collect_resource”, que analisa a posição passada e, caso possua um recurso, aumenta o contador de recursos do robô, retira o recurso da posição do mapa e coloca a representação de um espaço vazio nessa posição e por fim retorna true, caso contrário apenas retorna false.

Existem também dois métodos referentes a retornar os recursos e aliens e instanciar seus valores como 0.

Outras funções importantes existentes são as seguintes:
print_relatory: Imprime as ordens presentes na fila executedOrders
execute_order: Executa as ordens armazenadas em toExecuteOrders e as adiciona em executedOrders
add_normal_order: Aciciona uma ordem sem prioridade em toExecuteOrders
add_order_with_priority: Adiciona uma ordem com prioridade em toExecuteOrders
move_order: executa uma ordem de movimentação do robô no mapa
collect_order: executa uma ordem de coletar um recurso
destroy_order: executa uma ordem de destruir um alien
process_order: Recebe uma string com o comando e descobre qual ordem deve ser adicionada à toExecuteOrders

- RobotArray: Uma classe que armazena todos os 50 robôs que podem ser utilizados durante o programa.

Seu construtor instancia um array de ponteiros, em que cada ponteiro aponta para uma instância de um robô criada dinamicamente.
Possui os seguintes métodos:
activate_robot: Chama a função activate do robô correspondente, retorna se ele já estava ativado ou não
is_active_robot: Retorna se o robô correspondente já estava ou não ativado.
return_to_base: chama o método Robot::return_to_base do robô correspondente
return_resources_to_base: Chama o método Robot::return_resources_to_base do robô correspondente
return_aliens_to_base: Chama o método Robot::return_aliens_to_base do robô correspondente
print_relatory: Chama o método Robot::print_relatory do robô correspondente
execute_order: Chama o método Robot::execute_order do robô correspondente
add_normal_order: Chama o método Robot::add_normal_order do robô correspondente
add_order_with_priority: Chama o método Robot::add_order_with_priority do robô correspondente

- Base: Essa classe corresponde à representação da base, possuindo variáveis de recursos e aliens, um Map correspondente e uma instância de RobotArray.

O construtor recebe o arquivo que representa o mapa e instância dinamicamente um mapa, além de instanciar um objeto RobotArray.

Possui duas funçôes de impressão, a “print_map” chama o método “print” do map e a “print_resources” imprime os recursos na forma definita pela orientação do trabalho.

Os outros métodos lidam com ordens referentes aos robôs, sendo eles:
.activate_robot, return_robot, print_relatory, execute_order: Recebem o id do robô e repassa a ordem para ser tratada pela instância de RobotArray.
.add_normal_order, add_order_with_priority: Recebem a string referente a ordem e o id do robô correspondente, repassando a ordem para ser tratada pela intância de RobotArray.

- Orders: Uma classe responsável por lidar com todas as ordens presentes no arquivo e chamar a base para repassar as ordens para cada robô.

Seu construtor recebe o arquivo que contém as ordens existentes e a instância do tipo Base instanciada no main, fazendo a leitura das ordens do arquivo linha a linha e chamando o método receive_order para cada uma delas.

Os métodos presentes na classe são:

.receive_order: Identifica se uma ordem é uma ordem direta e, caso não seja, se possui ou não prioridade e chama o seu próprio método correspondente que deve lidar com cada uma dessas ordens.
.add_order_with_priority e add_normal_order: Retiram o id do robô da ordem e chamam o método de mesmo nome da instância de Base passada com a ordem e o id do robô.
.execute_direct_order: Identifica o tipo de ordem direta que deve ser executada e chama seu respectivo método que deve ser utilizado.
.activation_order, relatory_order, return_order, execute_order: Extraem o id do robô da ordem e repassam a ordem para o método correspondente da instância de Base.
.is_direct_order: Identifica se uma ordem é ou não direta.

3. Instruções de compilação execução

Para o processo de compilação ser executado primeiramente navegue para a pasta projeto do projeto, caso haja alguma alteração nos headers da parte include, execute o comando “make clean” e execute o comando “clean”. Caso a alteração sejam nos arquivos da pasta “src” ou seja a primeira compilação na máquina, basta executar o comando “make”

Para o processo de execução, já estando na pasta projeto, primeiramente entenda qual a localização dos arquivos de ordem e mapa você possui. Uma boa dica é colocara seus arquivos na pasta “files” e executar o seguinte comando:
./bin/run.out files/<nome_do_arquivo_do_mapa> files/<nome_do_arquivo_de_ordens>

Caso decida não colocar seus arquivos nessa pasta, execute o comando com os caminhos relativos dos arquivos da seguinte forma:
./bin/run.out <caminho_relativo_do_arquivo_do_mapa> <caminho_relativo_do_arquivo_de_ordens>

Ao final desse comando você deve visualizar as impressões das ordens passadas pelo arquivo, da forma como especificado na definição do trabalho prático.

4. Análise de Complexidade

As complexidades de tempo e de espaço de todas as funções são analisadas a seguir uma a uma.

-Matrix:
.Construtor: Realiza operações de instanciação constantes no tempo O(1), além de possuir um laço while que executa n+1 vezes, com n sendo o número de linhas da matriz, além de nesse loop existirem dois laços for que, em conjunto executam um número m de vezes. Assim o tempo de execução é de O(nm). A complexidade de espaço é relativa à alocação das linhas e colunas da matriz, sendo ela O(nm)
.Destrutor: Realiza dois laços, um do tamanho das linhas O(n) e um dentro do primeiro por cada coluna O(m) sem alocar espaço extra em nenhuma delas. Logo a complexidade de tempo é de O(nm) e a de espaço é de O(1).
.getRows e getCols: Realiza operações constantes e não aloca nenhuma memória, tendo complexidade de memória e tempo de O(1)
.get: Possúi uma estrutura condicional que executa uma vez chamando getRows(complexidade O(1)) e getCols(complexidade O(1)), sem alocar nenhuma memória extra em sua execução. Dessa forma possui complexidade de tempo e espaço de O(1).
.set: Realiza dois acessos a um array(constantes) e altera seu valor sem alocar espaço extra. Possui assim complexidade de tempo e de espaço de O(1).
.operator[]: Retorna um ponteiro de strings sem alocar memória extra. Complexidade de tempo e de espaço de O(1)
.print: Realiza um único if O(1) e dois laços, um do tamanho das linhas O(n) e um dentro do primeiro por cada coluna O(m) sem alocar espaço extra. Logo a complexidade de tempo é de O(nm) e a de espaço é de O(1).

-PriorityQueue:
.Construtor: Executa operações de instanciação constantes sem alocar espaço. Complexidade de espaço e de tempo O(1)
.insert: Aloca o espaço para um novo elemento executando um if com operações constantes, possuindo complexidade de tempo constante O(1). Aloca o espaço para um novo nó, possuindo complexidade de espaço constante O(1).
.insert_with_priority: Aloca o espaço para um novo elemento executando um if com operações constantes, possuindo complexidade de tempo constante O(1). Aloca o espaço para um novo nó, possuindo complexidade de espaço constante O(1).
.delete: Desaloca o espaço para o ultimo elemento da fila utilizando o ponteiro para o último elemento, assim não precisa de percorrer toda a fila, tendo complexidade de tempo O(1). Como desaloca um elemento, possui complexidade de espaço O(1)
.get_element_number: retorna um valor sem executar mais nenhuma operação. Complexidade de tempo e espaço O(1).
.clear: Deleta todos elementos tendo de percorrer toda a fila em um laço. Complexidade de tempo O(n), complexidade de espaço O(1).
.print: Percorre toda a fila sem alocar memória extra. Complexidade de tempo O(n), de espaço O(1).

-Robot:
.Construtor: Realiza operações constantes, incluindo a alocação de dois PriorityQueue cujos construtores também tem complexidades de O(1). Assim a complexidade de tempo e de espaço é O(1).
.getters: Todos retornam um valor presente na instância sem realizar nenhuma outra operação. Complexidade de tempo e espaço O(1).
.activate: Executa apenas comandos contantes sem realizar nenhuma nova alocação. Complexidade de tempo e espaço O(1).
return_to_base: Possui um laço for que itera pela fila executedOrders de tamanho n, deletando cada um de seus elementos, assim possui complexidade de tempo O(n) e de espaço O(1).
.move_to, destroy_alien, collect_resource: Todos esses métodos possuem um teste condicional e não alocam memória extra, possuindo complexidade de tempo e de espaço O(1).
.return_resources_to_base, return_aliens_to_base: Esses dois métodos executam apenas operações constantes e instanciam apenas um inteiro, temdo complexidade de tempo e espaço O(1).
.print_relatory: Chama a função print de uma fila de prioridades, que possui complexidade de tempo O(n) e de espaço O(1). Logo, sua complexidade de tempo é O(n) e de espaço O(1).
.execute_order: Itera por toda a fila de prioridades chamando a cada iteração o método delete_element O(1) e o método process_order O(1). Sua complexidade de tempo é O(n) e de espaço O(1).
.process_order: Realiza um número fixo de operações O(1) que não alocam espaço. Complexidade de tempo e espaço O(1).
.add_normal_order, add_order_with_priority: Chamam um método para adicionar um novo nó na fila, que possui complexidade de tempo e espaço constantes. Logo possui complexidade de tempo e espaço O(1).
.move_order, collect_order, destroy_order: Possuem números fixos de operações com complexidades de tempo e espaço constantes. Possui complexidade de tempo e espaço O(1).

-RobotArray:
.Construtor: Executa a criação de um array de 50 posições e itera sobre ele criando um novo robô a cada chamada. Possui assim complexidade de tempo e espaço O(50)=O(1).
.activate_robot:Executa apenas comandos contantes sem realizar nenhuma nova alocação. Complexidade de tempo e espaço O(1).
.is_active_robot: Realiza apenas um teste de condição sem alocar memória extra. Complexidade O(1).
.return_to_base: realiza operações constantes e chama a função do Robot return_to_base com complexidade de tempo O(n) e espaço O(1). Possui assim complexidade de tempo O(n) e espaço O(1).

.return_resources_to_base, return_aliens_to_base: Acessam um elemento do array O(1) e chamam um método com ambas complexidades constantes. Complexidade de tempo e espaço O(1).
.print_relatory: Acessa um elemento do array e chama uma função de complexidade de tempo O(n) e de espaço O(1). Possui assim complexidade de tempo O(n) e de espaço O(1).
.execute_order: Acessa um elemento do array e chama uma função de complexidade de tempo e espaço O(1). Possui complexidade de tempo e espaço O(1).
.add_normal_order, add_order_with_priority: Complexidade de tempo e espaço O(1).

-Map:
.Construtor: Aloca uma matriz. Complexidade de tempo e espaço $O(n^2)$
.move_to, destroy_alien, colect_resource: Chamam um método que possui complexidades constantes. Complexidade de tempo O(1) e de espaço O(1).
.print: Chama um método que possui complexidade de tempo $O(n^2)$ e de espaço O(1). Possui assim complexidade de tempo $O(n^2)$ e de espaço O(1).

-Base:
.Construtor: Instancia um novo Map com construtor com ambas complexidades quadráticas e um novo RobotArray com construtor com ambas complexidades constantes. Possui complexidade de tempo e espaço $O(n^2)$.
.activate_robot: Possui uma condicional com teste de condição constante sem alocar memória extra. Complexidade de tempo e espaço O(1).
.return_robot: Possui um teste de condição e caso entre nela executa comandos constantes e um método de RobotArray return_to_base com complexidade de tempo linear e de espaço constante. Possui no melhor caso complexidade de tempo e espaço O(1) e no pior caso complexidade de tempo O(n) e de espaço O(1).
.print_relatory: Chama um método com complexidade de tempo linear e de espaço constante. Possui assim complexidade de tempo O(n) e de espaço O(1).
.execute_order: Chama um método com complexidade de tempo constante e de espaço constante. Possui assim complexidade de tempo O(1) e de espaço O(1).
.add_normal_order, add_order_with_priority: Complexidade de tempo e espaço O(1).

-Orders:
.is_direct_order: Realiza um teste condicional sem alocar nova memória. Complexidade de tempo e espaço O(1).
.add_normal_order, add_order_with_priority: Realizam testes condicionais e chama por fim uma função de complexidade de tempo e espaço O(1). Complexidade de tempo e espaço O(1).
.execute_order: Chama um método com ambas complexidades constantes. Complexidade de tempo e espaço O(1).
.return_order: Chama um método com complexidade de tempo linear e espaço constante. Complexidade de tempo O(n) e de espaço O(1).
relatory_order: Chama um método com complexidade de tempo linear e espaço constante. Complexidade de tempo O(n) e de espaço O(1).
.activation_order: Chama um método com ambas complexidades constantes. Complexidade de tempo e espaço O(1).
.execute_direct_order: Possui um teste de condição e, dependendo de qual caso cair, possui complexidade de tempo no pior caso de O(n) e de espaço O(1), no melhor caso possui complexidade de tempo e espaço O(1).
.Construtor: O construtor itera sobre o arquivo recebido como parâmetro e a cada iteração chama o método receive_order que possui no melhor caso ambas complexidades constantes e no pior caso ambas complexidades lineares. Assim possui no melhor caso complexidade de tempo e espaço O(n) e no pior caso complexidade de tempo e espaço $O(n^2)$.

5. Conclusão

O trabalho descrito se encarregou de tomar dois arquivos de texto e, por meio do uso correto de uma estrutura de dados matriz e outra estrutura de dados fila de prioridade, utilizar as ordens

passadas com as entidades robôs descritas no mapa criado, interagindo sempre com um objeto do tipo base para retornar as variáveis presentes no robô.

Utilizando duas filas de prioridades referentes as ordens passadas, uma representação do mapa em forma de matriz além de as funções e métodos relacionadas à strings presentes na biblioteca padrão, foi evidente a facilidade de abstração do problema ao possuir um bom entendimento de estruturas de dados e dos princípios gerais de orientação a objetos aplicados na linguagem C++.

O uso das soluções adotadas propiciou também uma possibilidade de modularização do código e testes em cima de cada objeto serem executados.

Durante a implementação do código, as maiores dificuldades apresentadas foram duas. A primeira foi a questão de alocação e desalocação de memória nos construtores e destrutores das classes, sendo muitas vezes necessário a revisão das classes para resolver problemas de alocação apontados pelo compilador. O segundo grande problema enfrentado foi a implementação da fila de prioridades por meio de ponteiros, sendo necessário um cuidado extra para conferir se as propriedades presentes na estrutura de dados estavam realmente sendo seguidas pelo código.

6. Referências

-Bruce Eckel. Thinking in C++
-Nivio Zaviano. Projeto de Algoritmos com Implementação em Pascal e C
-Thomas H. Cormen. Algoritmos
-Chaimowicz, L. and Prates, R. (2020). Slides virtuais da disciplina de estruturas de dados.