Trabalho Prático 1

O problema da frota intergaláctica do novo imperador

Fernanda Carolina da Silva Pereira

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

fernandacsp@ufmg.br

1. Introdução

Num universo cinematográfico paralelo de Star Wars, O Império derrotou a Aliança Rebelde. Neste contexto, o novo imperador segue insaciável por conquistar novos povos da galáxia, impondo o poder militar da sua frota intergaláctica de batalha. Dessa forma, o problema consiste em planejar e gerenciar os ataques das frotas de Darth Vader.

O problema consiste em três etapas, sendo elas preparação para a batalha, combate e nave avariada. Antes de um combate, as naves são organizadas de acordo com sua aptidão e serão posicionadas pelo imperador Vader da menos apta para a mais apta. Porém, para o combate as naves são acionadas da mais apta para a menos apta, de forma que ao comando para entrar em combate a nave mais apta deve ser a primeira a partir, e assim por diante.

Durante uma batalha uma nave pode ser afetada. Uma vez informado de que uma nave está avariada, o sistema deve imediatamente retirar essa nave de combate e colocá-la em fila para manutenção. E isso deve se repetir sempre que houver um caso de avaria. Uma vez interceptadas, as naves deverão ser consertadas para retornarem à batalha, de modo que a primeira nave interceptada é a primeira a ser consertada e a voltar para a estrutura de espera para acionamento de combate, tendo também preferência sobre as outras naves em espera, e assim sucessivamente.

2. Implementação

O sistema foi desenvolvido em C++ e compilado com o compilador GNU G++.

3. Estruturas de Dados

Em cada etapa foi utilizada uma estrutura de dados que melhor atendesse à demanda do imperador. Para a preparação para a batalha foi exigido que as naves fossem posicionadas da de menor aptidão para a de maior, mas que ao comando de combate fossem acionadas de maneira inversa, da mais apta para a menos apta. Nesse caso foi utilizada a estrutura de Pilha Encadeada, visto que esse tipo abstrato de dados tem a característica *LIFO – Last In First Out*, em que o último elemento a ser inserido é o primeiro a ser retirado, atendendo à demanda de Vader.

Já para a etapa de combate e naves avariadas, foi utilizada a estrutura Fila Encadeada, tendo como característica *FIFO – First In First Out*, atendendo mais uma vez às especificações.

Em ambas as estruturas foi utilizado o conceito de célula para melhor encapsulamento de manipulação dos dados.

3.1. Pilha

Como mencionado anteriormente, a estrutura pilha foi implementada utilizando lógica de apontadores. A implementação escolhida dispensa a alocação de estática de memória, permitindo que o espaço seja alocado à medida que naves são inseridas e removidas pelas opções de manipulação. Na estrutura foi criada a variável tamanho que armazena o tamanho da pilha e que é alterada em casos de inserção e remoção.

3.2. Fila

Em raciocínio semelhante ao da estrutura pilha, a implementação utilizada foi a de ponteiros, dispensando alocação estática de memória e permitindo a dinamicidade da alocação à medida que naves são inseridas e removidas. Há também uma variável tamanho armazenando o tamanho da fila, de modo a ser alterada em casos de inserção e remoção.

4. Classes

O sistema contém uma classe para manipulação da Fila Encadeada, uma para manipulação da Pilha Encadeada e uma para o Tipo Célula. Ambas as classes fila e pilha possuem seus itens encapsulados pelo tipo célula.

Aplicando-se às etapas, criamos a pilha chamada *waiting_for_combat*, que como o próprio nome diz manipula as naves aguardando por combate. Para a manipulação foram utilizadas

as funções *Empilha*, *Desempilha* e *Print*. Respectivamente, a primeira realiza empilhamento, incrementando o tamanho da pilha sempre que isso acontece; a segunda desempilha caso a pilha não esteja vazia, decrementando seu tamanho em caso de remoção; e finalmente, a terceira printa os itens da pilha.

Para a manipulação das naves em combate e naves avariadas criamos a fila *in_combat* e *broken*, respectivamente. Para sua manipulação foram utilizadas as funções *Enfileira, Desenfileira, EncontraPosição, Remove* e *Print*. Semelhante ao caso anterior, a primeira enfileira um item em caso de inserção; a segunda desenfilera em caso de remoção; a terceira encontra a posição de um item na fila; a quarta, utilizando a implementação da terceira remove o elemento de uma determinada posição; e a quinta printa os itens da fila.

5. Organização e Fluxo da Batalha

Primeiramente o imperador irá informar o tamanho de sua frota e após uma iteração executada sobre esse tamanho, o sistema recolhe os ids das naves.

Com essas informações podemos iniciar o fluxo de operações em que se desenvolve a batalha. A partir daí o imperador pode escolher que ação deseja executar sobre sua frota, sendo os comandos 0, -1, -2, -3 e qualquer valor que faça referência a um id de uma nave.

Caso a escolha seja 0, o sistema aciona a nave mais apta da pilha *waiting_for_combat* para entrar em combate. Caso a entrada seja um inteiro x em combate, o sistema remove a nave da fila *in_combat* e a transfere para a fila *broken*. Caso –1, a primeira nave da fila *broken* é consertada, e assim se houver próxima, ela assume a prioridade de conserto. Após consertada, uma nave retorna para a fila de aguardo para combate com prioridade sobre as outras que já estavam no aguardo. Caso –2 ou –3, os métodos print da pilha ou da fila são acionados para imprimir os valores de suas células.

Caso o imperador informe um comando EOF, o sistema irá parar sua execução.

6. Análise de Complexidade

No melhor caso o imperador não vai empilhar nenhuma nave, vai informar o comando EOF e o sistema não irá executar nenhuma opção, tomando complexidade constante.

As complexidades dos comandos 0 e –1 são também constantes, pois realizam apenas ações de inserção e remoção. Quanto aos comandos –2, -3 e entrada x (x sendo id de uma nave),

possuem complexidade linear, visto que realizam iteração para impressão dos dados, ou no caso de x, remover uma célula.

6.1. Complexidade de espaço

No melhor caso o imperador vai alocar as naves, mas não realizar ações sobre elas, e assim a complexidade seria constante. No pior caso, ele irá alocá-las e realizar ações livremente, portanto, a complexidade seria linear.

7. Conclusão

Após os estudos da matéria, foi possível identificar rapidamente as estruturas necessárias para a resolução e entendimento do problema. A contextualização também permitiu o afloramento da criatividade sobre a implementação.

Referências

- Ziviani, N. (2006). Projetos de Algoritmos com Implementações em Java e C++;
- Material disponibilizado pela disciplina de Estrutura de Dados.