Poker Face

Marcelo Andrade

github.com/marceloanje
linkedin.com/in/marcelo-andrade-10334b160
marcelo.anje@outlook.com

1. Introdução

O poker é um jogo de cartas amplamente praticado ao redor do globo, sendo considerado um dos jogos mais populares que existe. Dessa forma, foi proposto nesse trabalho o desenvolvimento de um sistema que simula as "engrenagens" do jogo, ou seja, o sistema irá tratar somente com a parte burocrática, sendo um decisor dos vencedores e seu montante de dinheiro virtual. Logo, os jogadores físicos podem manter toda dinâmica e emoção do jogo pessoalmente.

O programa recebe um arquivo texto com os dados de todas as rodadas e com base nesses dados ele realiza toda dinâmica para saber quem venceu cada rodada e quem acumulou mais dinheiro no final.

2. Método

2.1 Implementação

O programa foi desenvolvido na linguagem C++, compilada pelo WSL da Ubuntu e os testes foram realizados em um emulador de sistemas operacionais (VirtualBox) com Linux Ubuntu 20.04.4, sendo emulado em um computador com Windows 10, Ryzen 7 2.3 GHz e 8GB de memória RAM.

2.2 Estrutura de dados

A implementação do programa teve como base a estrutura de dados do tipo lista, que é a *Jogador _jogadores*, uma lista formada de jogadores da classe *Jogador*, essa lista foi alocada estaticamente devido ao número máximo de jogadores ser conhecido, (número máximo de jogadores é 10, devido ao uso de somente 1 baralho por jogo), e a maior facilidade na sua implementação.

Uma outra estrutura de dados usada foi uma lista da classe *Cartas*, a *Cartas mao[5]*, se caracteriza com um tamanho de exatamente 5 elementos para cada *Jogador*.

2.3 Formatos de dados

O sistema foi feito de tal forma, que durante sua execução é recebido um caminho o qual deve se encontrar um arquivo de entrada no formato *.txt*, o qual contém os dados de entradas que são tratados internamente, dessa forma a realizar todo mecanismo para determinar o vencedor de cada rodada e o dinheiro final de todos os participantes. O resultado final é expelido através de um arquivo no formato *.txt*, que pode ter seu caminho especificado através de argumentos fornecidos pelo usuário.

2.4 Classes e métodos

O programa foi dividido primordialmente em três classes diferentes, classe ***Carta***, classe ***Jogador*** e classe ***Partida***. Além dessas três classes, também há o arquivo ***main.cpp***.

A classe ***Carta*** possui dois construtores, um paramétrico e um default, também possui alguns métodos básicos, como *get* e *set* ,para manipulação dos seus atributos privados, que são *naipe* do tipo *Naipe* (uma constante que possui todos os naipes) e *valor* do tipo *int*.

A classe ***Jogador*** possui dois construtores, um paramétrico e um defaut, também possui métodos básicos, como *get* e *set* ,para manipulação dos seus atributos privados, que são *nome* do tipo *string*, *dinheiro* do tipo *int,* dentre outros. Na classe *Jogador*, também há o atributo privado *Carta mao[5]*, que se caracteriza como sendo uma lista do tipo *Carta*, com tamanho fixo de cinco, sendo assim, alocada estaticamente. Há também, alguns outros métodos relevantes, como o método *ordenaMao*, que é usado para ordenar por valor a mão de cada jogador, para dessa forma facilitar sua análise, o método *lavaMao*, que é usado para "zerar" a mão do jogador. Essa classe, possui outros métodos booleanos, que são usados para fazer a análise da mão de cada jogador, para dessa forma determinar qual o tipo de mão e sua respectiva força.

A classe ***Partida*** possui um construtor paramétrico e alguns métodos *get* e *set* para manipular os atributos privados, como o *tamanho*, que é definido quando a classe é inicializada, o *id*, que o índice do vencedor de cada rodada, o *montante*, que o montante que o vencedor de cada rodada recebe, dentre outros. Há também, um atributo do tipo lista, que é a *Jogador _jogadores*, que é uma lista do tipo *Jogador*, essa lista possui tamanho máximo igual a dez, pois cada rodada só pode possuir no máximo dez jogadores, tendo em vista que só é usado um baralho de cinquenta e duas cartas a cada rodada. Essa lista, é alocada estaticamente, devido ao tamanho máximo ser conhecido e não ser um valor alto, outro motivo é pela facilidade da implementação e o uso dos índices de cada item do tipo *Jogador*. É nessa classe que ocorre toda manipulação dos dados do sistema, nela há alguns métodos primordiais para o programa, como o método *addJogadorCarta*, que é responsável por criar um novo jogador, adicionar as cartas da sua mão e inserir na lista *Jogador _jogadores*, há também, o método *vencedorRodada*, que é responsável por manipular o dinheiro dos jogadores da rodada e comparar os jogadores para verificar quem é o vencedor da rodada.

O arquivo ***main.cpp*** é onde ocorre a execução do programa em si, essa parte recebe o arquivo de entrada em formato *.txt*, fornecido pelo usuário, lê todo esse arquivo, cria duas *Partida*, uma para ser usada a cada e outra global, na *Partida* global ocorre somente a

manipulação do dinheiro dos jogadores, e na segunda *Partida* ocorre todas as manipulações de cartas, dinheiro e etc. Nesse arquivo, também ocorre a impressão da saída do programa para um caminho determinado pelo usuário.

3. Análise de complexidade

A análise de complexidade do programa foi analisada os métodos relevantes de cada classe e o *main*. Dessa forma, sendo analisada no âmbito do tempo e espaço.

3.1 Complexidade temporal

Na classe ***Carta***, não há nenhum tipo de comparação, somente atribuições por meio dos métodos *set*,*get* e do construtor, podendo ser considerados como O(1).

Na classe ***Jogador***, temos os métodos *get*, *set* (exceto exceções que serão trabalhadas a seguir) e construtor que possuem apenas algumas comparações, logo, complexidade O(1).
lavaMao possui apenas alguns atribuições - O(1);
eFlush possui um "for" para realizar uma comparação - O(n);
ordenaMao possui dois "for" aninhados para realizar um ordenamento do tipo *bolha* - $O(n^2)$
eStraight possui um "for" para realizar uma comparação e aciona internamente o método *ordenaMao*, $O(n^2)$, sendo assim, suas ordens de complexidade são somadas - $O(n^2)$;
eRoyaStraightlFlush possui apenas algumas comparações que são O(1), mas aciona internamente os métodos *eFlush* e *ordenaMao*, que tem ordem de complexidade respectivamente, O(n) e $O(n^2)$, sendo assim, suas ordens de complexidade são somadas - $O(n^2)$;
eStraightFlush possui apenas algumas comparações que são O(1), mas aciona internamente os métodos *eFlush* e *eStraigh*, que tem ordem de complexidade respectivamente, O(n) e $O(n^2)$, sendo assim, suas ordens de complexidade são somadas - $O(n^2)$;
eFourOfKind possui dois "for" aninhados para realizar uma comparação - $O(n^2)$;
eFullHouse possui dois "for" aninhados para realizar uma comparação - $O(n^2)$;
eThreeKind possui dois "for" aninhados para realizar uma comparação - $O(n^2)$;
eTwoPairs possui dois "for" aninhados para realizar uma comparação - $O(n^2)$;
eOnePair possui dois "for" aninhados para realizar uma comparação - $O(n^2)$;
getMao aciona internamente todas os métodos booleanos da classe *Jogador* para verificar qual o tipo da mão, sendo assim, as ordens de complexidade de todas as funções são somadas
- $O(n^2)$;

Para saber a complexidade da classe, basta somar a complexidade de cada método, que resulta em $O(n^2)$.

Na classe ***Partida***, temos o método construtor que possui uma atribuição e pode ser considerado como O(1).
addJogadorCarta cria um novo jogador e atribui cartas a ele - O(1);
InsereInicio possui apenas algumas atribuições e uma comparação - O(1);

vencedorRodada possui dois “for” aninhados para realizar uma comparação, depois um “for” para realizar uma comparação, e possui alguns acionamentos internos de métodos da classe *Jogador*, sendo esses *get* e *set*, sendo assim, O(1), dessa forma, as ordens de complexidade de todas as funções são somadas - $O(n^2)$;
transNaipe possui apenas algumas atribuições e comparações - O(1); *transValor* possui apenas algumas atribuições e comparações - O(1); voltaNaipe possui apenas algumas comparações - O(1);
addJogadorDinheiro possui o acionamento interno de alguns métodos *set* da classe *Jogador* - O(1);
setnumJogadores possui apenas uma atribuição - O(1);
mudaDinheiro aciona internamente um método *set* da classe *Jogador* - O(1);
getDinheiroJogadores aciona internamente um método *get* da classe *Jogador* - O(1);
getNomeJogadores aciona internamente um método *get* da classe *Jogador* - O(1);
balancoJogador possui apenas algumas atribuições e comparações - O(1);
getnumJogadores possui apenas um retorno - O(1);
getMontante possui apenas um retorno - O(1);
getTipoMaoJogadores aciona internamente um método *get* da classe *Jogador* - O(1);
getID possui apenas um retorno - O(1);
ordenaJogadores possui dois “for” aninhados para realizar um ordenamento do tipo *bolha* - $O(n^2)$
getPingoRodada aciona internamente um método *get* da classe *Jogador* - O(1);

O ***main*** possui dois “for” aninhados para realizar os mecanismos do sistema, depois mais dois “for” aninhados para realizar uma outra parte e por fim mais um “for” para realizar a impressão, logo a soma dessas complexidades resulta em $O(n^2)$. Dentro do *main* são acionados alguns métodos das classes do sistema, mas como a maior ordem de complexidade das classes é $O(n^2)$, dessa forma podemos considerar que o *main* como um todo possui ordem de complexidade $O(n^2)$.

Com isso, tem se as ordens de complexidade de todos os métodos e partes do programa, como a ordem de complexidade total é a soma de todas ordens, pode-se concluir que a ordem de complexidade total do sistema é $O(n^2)$.

3.2 Complexidade espacial

A análise de espaço pode se avaliar o principal meio de armazenamento do sistema, que a lista de *_jogadores* do tipo *Jogador*, dessa forma, podemos considerar que o espaço ocupado é **O(n)**, pois a lista possui somente uma dimensão. Pode se analisar também que a complexidade não irá se alterar independente do tamanho da lista, visto que, $c*O(f(n)) = O(f(n))$.

4. Estratégias de robustez

Como se trata de manipulação de arquivos, uma estratégia de robustez utilizada foi verificar as consistências dos arquivos, como por exemplo, se o arquivo de entrada existe e se o arquivo de saída foi devidamente criado.

Outra estratégia usada foi na parte da estrutura de dados, como o sistema utiliza uma lista estática, deve se verificar se o tem armazenamento na lista ao adicionar um novo item.

Na parte da leitura e execução do programa, um estratégia utilizada para prevenção de erros, foi realizar a leitura de acordo com os dados de entrada, por exemplo, se inicialmente os dados dizem que são somente x rodadas, mesmo se o resto do arquivo conter algum erro e contiver mais rodadas, o programa não realiza a leitura.

5. Conclusões

Esse trabalho teve como objetivo a implantação de um sistema “poker face”, a partir disso foi desenvolvido um programa em C++ para atender a todas as especificações propostas.

Com esse desenvolvimento foi possível assimilar bem novos conceitos, como o de listas e sua manipulação.

Referências

POKER. [*S. l.*], 1 jun. 2022. Disponível em: https://en.wikipedia.org/wiki/Poker. Acesso em: 1 jun. 2022

Ziviani, N., **Projeto de Algoritmos com Implementações em Pascal e C**, 3ª Edição, Cengage Learning, 2011.

Instruções de compilação e execução

Para a execução do programa sigas os passos descritos abaixo, utilizando o Makefile para isso:

- Utilizando o terminal acesse o diretório;
- Execute o arquivo Makefile utilizando o seguinte comando: “make”;
- Com esse comando deve se gerar um arquivo intitulado como, “saida.txt”, que se encontra no diretório, e nele possui a saída do programa para uma entrada especificada por meio de um arquivo intitulado por padrão, como: “entrada.txt”;