Trabalho Prático 1 da disciplina de Estrutura de Dados

Alex Eduardo Alves dos Santos
Matrícula: 2021032145

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
alexeduardoaeas@ufmg.br

1. Introdução

O problema proposto foi criar um sistema onde qualquer um pode jogar Poker, mesmo aqueles que não sabem quase nada das regras do jogo. O sistema basicamente cuida da etapa de decisão, onde vê quem ganhou, e do montante de dinheiro virtual que cada jogador terá ao final das diversas rodadas. Dessa forma, os jogadores podem fisicamente fazer suas jogadas, blefarem, realizarem apostas e o sistema só precisa saber quais cartas e quanto foi a aposta total de cada participante no momento da decisão de cada rodada para determinar quem foi o vencedor.

2. Implementação

O programa foi desenvolvido na linguagem C++, compilada pelo compilador G++ da GNU Compiler Collection.

2.1. Estrutura de Dados

A implementação do programa teve como base a estrutura de dados de uma lista encadeada. A preferência desta ao invés da lista estática foi pela possível alocação dinâmica dos jogadores e todo o conteúdo que compõem esse objeto.

Essa estrutura de dados foi montada numa classe, como visto em aula, implementou se a lista com a chamada "célula-cabeça" antes do primeiro elemento, visando facilitar e, assim, reduzir o custo assintótico de algumas operações. Além disso, um dos membros da lista guarda a quantidade de elementos que ela possui, evitando um custo linear desnecessário (de iterar pela lista inteira contando o número de elementos) caso não houvesse esse atributo com essa informação.

As principais funções, como os construtores e os destrutores, a Vazia, bem como as operações de inserção e remoção das várias posições da lista são adaptações dos algoritmos vistos em sala.

2.2. Classes

Para modularizar a implementação, foram montadas 5 classes. A primeira delas é a Lista encadeada cujas células são structs com um ponteiro para a próxima célula e outra para o elemento(objeto) que está sendo guardado nessa célula e, consequentemente na lista.

A lista possui alguns métodos úteis como o GetItem para receber o jogador inserido na posição da lista e outros métodos comuns nesse tipo de estrutura de dados como os de inserção e remoção.

Uma variação da mesma lista também foi montada para guardar a classe Carta, e podendo assim criar a mão de cartas dos jogadores a cada rodada.

As outras classes são Jogador, Carta e Mão. Elas foram criadas para armazenar os dados das entradas e processá-los. A classe Jogador basicamente guarda as informações fornecidas a cada rodada, como o nome, o saldo, as maiores cartas, a classificação da sua jogada, além do valor da aposta e se ele iria participar da rodada.

A classe carta basicamente guarda a combinação que compõem uma carta sendo formada por um número e um naipe, podendo assim formar uma Lista de cartas, através da Lista Encadeada mencionada anteriormente, que é usada pela classe Mão onde é definido a classificação da jogada de cada jogador e suas respectivas maiores cartas.

2.3. Rodadas e Vencedores.

O problema foi abordado diferenciando a primeira rodada das rodadas seguintes, pois, é ali que as principais informações serão processadas, como a Lista de todos os jogadores que irão participar e os valores iniciais de dinheiro, além da quantidade de rodadas que irão ocorrer. Logo após popular a Lista de jogadores eles são ordenados em ordem alfabética através de chamada da função OrdenaAlfabetica, onde é usado o método de ordenação chamado de inserção, como visto em sala de aula.

Em seguida, o programa segue processando as informações das cartas dos jogadores, além das apostas e do Pingo, isso é feito a partir da classe mão onde é computada a Listas de cartas, inseridas em ordem crescente a partir do método InsereCard, e verificada as possíveis classificações das jogadas, a classificação e as maiores cartas são guardadas na classe jogador da Lista de jogadores.

Logo após, o programa define os vencedores da rodada e processa os valores perdidos e recebidos a cada rodada por cada jogador. De maneira mais detalhada, o programa verifica quais jogadores possuem a maior classificação da rodada atual, em seguida compara as maiores cartas desses jogadores para definir o(s) vencedor(es), ao término da primeira rodada o programa entra em um loop onde se repete até o fim da execução.

Por fim, neste Loop, o programa processa as informações de maneira similar a inicial, com mudanças sutis sobre a verificação dos jogadores que irão participar da rodada e os que não vão, e se está tudo certo com as jogadas quanto ao saldo dos jogadores. Ao final do programa a Lista de jogadores é ordenada em ordem crescente, a partir do saldo final dos jogadores, através da função Ordena onde é usado o método de ordenação Inserção, e retorna a Lista ao arquivo de saída na ordem inversa.

3. Análise de Complexidade

3.1 Tempo

Começaremos a análise a partir da primeira rodada onde a Lista jogadores é populada.

Nesse caso, a inserção é feita sempre no final da lista, o custo dessa operação é O(1) pois existe um parâmetro na lista que guarda a última célula e a nova sempre é inserida após a última. No entanto, essa operação é realizada N(quantidade de jogadores) vezes.

Dentro do mesmo loop é realizado outro onde é populada a Lista de Cartas desse mesmo jogador, essa operação se repete 5 vezes(quantidade de cartas na mão do jogador) e o método InsereCard é chamado todas as 5 vezes, o método possui complexidade O(n) pois insere Ordenado e caso a carta a ser inserida seja maior que as outras já inseridas o custo será o pior caso, que é linear. Além disso ao final do método InsereCard é chamado o método da Lista de Cartas InserePosição onde é chamado o Metodo Posiciona que possui complexidade O(n) pois percorre toda a lista ate encontrar a célula da posição onde sera inserida a nova célula. Portanto a complexidade é 5*(O(n)+O(n))= Max(O(n),O(n)), ou seja, o custo é linear, e nesse caso okay pois N é sempre menor que 5.

Ainda dentro do loop, após serem lidas as cartas dos jogadores é definida a classificação das suas respectivas jogadas, isso é feito através do método classificação da classe mão, ao chamar esse método ele realiza uma série de testes para definir a classificação da jogada, o custo dessa operação é na maior parte das vezes o de algumas comparações simples, e para evitar que a lista de cartas fosse iterada várias vezes foi escolhido popular essa lista de maneira ordenada, facilitando as comparações, além disso antes de realizar as comparações que definem as jogadas foram tratados alguns casos, como se a lista de cartas é uma sequência ou se são todas cartas de mesmo naipe, eliminando assim várias comparações desnecessárias de jogadas impossíveis dependendo da mão, portanto, o custo dessa operação está atrelado aos primeiro loops, sendo eles no máximo O(n^2) caso todas as cartas sejam ás, e os outros laços são O(n), logo a complexidade no pior caso é O(n^2) para n sempre menor que 6.

Após popular a Lista de jogadores ela é ordenada em ordem alfabética pelo Inserção, que possui um for e um while dentro desse for, portanto a ordem de complexidade é O(n^2), por mais que a complexidade seja quadrática a lista é o suficientemente pequena(n<=12) para o programa funcionar normalmente.

Em seguida, outro laço é inicializado, esse laço possui complexidade Linear e define a quantidade de jogadores vencedores na rodada, além de alterar o saldo dos jogadores.

Por fim é chegada a hora de definir o resultado da jogada, no melhor caso, onde somente um jogador possui a classificação da jogada vencedora essa operação é O(n) pois somente será necessário buscar qual jogador é esse. Se esse não é o caso então serão chamados dois fors para verificar se os jogadores possuem maiores cartas diferentes, se esse for o caso então a complexidade será O(n)+O(n)+O(n), ou seja, Linear.

Por outro lado se ainda assim os jogadores permanecerem empatados, então será repetida a operação anterior mas verificando a segunda maior carta dos

jogadores, e a complexidade no pior caso para definir o(s) vencedor(s) da rodada será 5*O(n).

Primeira rodada k*O(n^2) + ordenação em ordem alfabética O(n^2)= O(n^2).

Finalizando a primeira rodada, o programa entra em um loop para as rodadas restantes, e daí é feita primeiramente a limpeza dos lixos da rodada anterior (O(n)), e a busca dos jogadores que vão participar da rodada, essa operação custa O(n^3), caso todos jogadores participem da rodada, em seguida é verificado se a rodada é válida, para isso é necessário O(n) operações, logo é hora de definir os vencedores da rodada, essa operação é igual a da primeira rodada, logo 5*O(n) no pior caso.

Rodadas seguintes: O(n)+O(n^3)+O(n)+5*O(n) = O(n^3).

Finalmente é realizado a ordenação pelo valor do saldo final dos jogadores, essa operação como já discutido anteriormente custa O(n^2) por se tratar do método Inserção, por fim, é retornado o resultado final, sendo necessário O(n) para percorrer toda a lista.

Em resumo a complexidade para execução do programa é:

O(n)+O(n^2)+O(n^3)+O(n^2)+O(n) = O(n^3)

3.1 Espaço

Vamos considerar que cada jogador ocupa uma unidade de espaço, para nossa análise. Assim, o programa permanece com todos esses jogadores alocados na memória durante a maior parte da sua execução. Com isso, para uma entrada de n jogadores, o programa ocupa o espaço de n jogadores, ou seja, O(n), e a cada iteração de jogada também são alocadas as cartas dos jogadores, como esse valor não permanece na memória a cada jogada o custo é constante O(1), considerando a lista de cartas como uma unidade de espaço.

4. Estrategias de Robustez

Na existência de casos em que as entradas são inválidas foram tomadas medidas contra esse tipo de problema, uma delas se trata da quantidade de dinheiro que cada jogador irá apostar ao longo das rodadas, se o jogador não possuir dinheiro o suficiente para pagar a aposta ou até mesmo o pingo da rodada ela será invalidada e o programa continuará a partir da próxima rodada.

Ademais, caso o jogador faça uma aposta que não seja múltipla de 50 ela também será invalidada.

Outro problema que poderia acontecer é se a entrada não obedecer a definições especificadas, no caso do valor do pingo, se ele não for maior, e múltipla de 50 então a rodada também será invalidada.

Não foram implementadas estratégias de robustez para todas as possíveis entradas, como em casos de confusão no arquivo de entrada e ela estar completamente errada. Espera-se que esse tipo de problema não ocorra e que as entradas sejam minimamente correntes com o problema.

5. Conclusão

Após a implementação do programa, pode-se notar que havia duas partes distintas para o desenvolvimento do trabalho. A primeira era implementar a estrutura de dados escolhida de forma correta, coesa e funcional para poder dar suporte à parte seguinte. Essa segunda parte tinha como objetivo utilizar a estrutura de dados criada anteriormente para criar os algoritmos que efetivamente fariam a computação das rodadas da partida de Poker. Havia ainda uma pequena e mais simples implementação que compreendia a leitura da entrada e escrita da saída dos dados.

O grande esforço, sobretudo para o desenvolvimento da segunda parte, era de realizar a classificação das mãos que cada jogador possuía, além de definir quem seria o vencedor nos vários casos em que eles permaneceram empatados.

Além disso, tinha-se que estar atento a algumas de boas práticas de programação como modularização de trechos de códigos e encapsulamento, a fim de destinar a devida classe, no caso Lista Encadeada, Mão e Jogador, a referida responsabilidade. Assim, momentos de refatoração, sobretudo do trecho do código que realiza essas ações, foram bastante necessários para deixar o algoritmo mais legível, e com métodos menores.

Ao fim desse trabalho, ficou ainda mais evidente a importância dos testes de ˆ unidade e da refatoração. Ambas práticas possibilitaram um desenvolvimento mais produtivo e consciente de que, com a implementação de novas funcionalidades ou com a refatoração de trechos do código, as estruturas do programa continuavam corretas e funcionais. Além disso, pode-se protagonizar dois papéis como programador: aquele que cria a estrutura de dados e aquele que utiliza a estrutura de dados.

Referências

Ziviani, N. (2006). Projetos de Algoritmos com Implementações em Java e C + +: Capítulo 3: Estruturas de Dados Básicas . Editora Cengage

Chaimowicz, L. and Prates, R. (2020). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

https://m.cplusplus.com/doc/tutorial/files/

Instruções para compilação e execução

- Acesse o diretorio [TP/];
- Cole o arquivo [entrada.txt];
- Utilizando um terminal, execute o arquivo [MakeFile] utilizando o seguinte comando: <make>;
- Com esse comando, o programa irá compilar gerando os arquivos .o dentro do diretório [TP/obj/] e o [run.out] dentro do diretório [TP/bin/];
- Com o programa devidamente compilado, utilizando um terminal execute o arquivo [run.out] utilizando o seguinte comando: <make exec>;
- Após a execução desse comando será gerado o arquivo [saida.txt] no diretório raiz do programa.