Trabalho Prático 1

Poker Face

Leandro Freitas de Souza - 2021037902

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

leandrofreitas@dcc.ufmg.br

1. Introdução

O problema proposto foi implementar um jogo de poker simplificado. Para um jogo, deve-se computar suas rodadas, e, para cada rodada, computar os seus vencedores e contar o montante virtual de cada jogador em todas as rodadas. Ao final, deve-se imprimir a classificação final dos jogadores, de acordo com a quantidade final de dinheiro.

2. Método

O programa foi desenvolvido na linguagem C++, e compilado pelo compilador G++, da GNU Compiler Collection. Além disso, a máquina utilizada no desenvolvimento no programa possui 8GB de memória RAM, e um processador Intel Core i5 de 11ª Geração.

2.1. Estruturas de Dados

Para a construção do jogo, foram implementadas duas listas de arranjo, uma para enumerar os jogadores, e outra para enumerar as cartas de cada jogador. A escolha de listas de arranjo em detrimento à lista encadeada se deve ao tamanho limitado que elas podem atingir: um jogo pode ter no máximo 10 jogadores, e cada jogador possui exatamente 5 cartas. Além disso, o acesso a elementos se dá de forma mais fácil, a inserção só ocorre no final da lista (o que é uma operação O(1)), e a implementação de uma lista de arranjo é mais simples.

As duas estruturas de dados diferentes foram implementadas em duas classes: a classe ListaJogadores e a classe ListaCartas. A implementação das listas foi uma adaptação da implementação vista em sala de aula, em que, além de métodos padrões de uma lista, como os métodos “GetTam()” e “InsereFinal()”, foram

implementados outros métodos que ajudam no funcionamento do jogo de poker como um todo.

O funcionamento de uma lista de arranjo, como observado na figura 1, se dá pela alocação de uma *array* de tamanho MAXTAM, que corresponde ao maior tamanho que essa *array* pode atingir. No caso da lista de cartas, esse tamanho máximo seria 5, uma vez que uma mão possui 5 cartas. Já na lista de jogadores, o tamanho máximo é 10, dado que um baralho possui 52 cartas, e isso significa que mais que 10 pessoas não conseguem jogar poker com apenas um baralho. A alocação dessas listas é estática, ou seja, todas as posições são alocadas, independentemente de seu tamanho real.

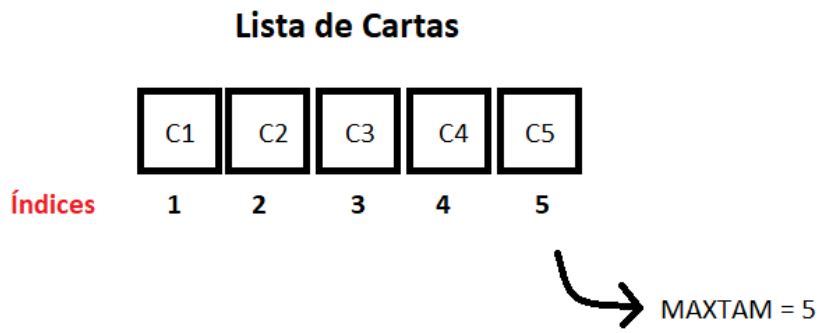

Figura 1: Ilustração da implementação estática da lista de cartas

Para a representação de uma carta, foi criada uma estrutura que armazena seu número e seu naipe, e para a representação de um jogador, foi criada outra estrutura que armazena seu nome, a quantidade de dinheiro que ele possui e uma Lista de Cartas, que representa as cartas que ele possui.

2.2. Classes

Para que a implementação do jogo fosse bem sucedida, foram implementadas duas classes e dois structs. As classes ListaJogadores e ListaCartas foram criadas para representar as estruturas de dados comentadas na sessão anterior.

As principais funções da classe ListaCartas envolvem uma função para ordenar as cartas baseado em seu número chamada *Ordena() e* uma função que classifica o tipo de mão que a lista de carta representa, retornando um inteiro de 1 a 10, relativos ao valor de importância de cada jogada, utilizada para determinar o jogador ganhador em uma determinada rodada.

A classe ListaJogadores, por sua vez, possui como uma das funções mais importantes a função Ganhadores, que retorna o ponteiro de uma lista de jogadores com os ganhadores de uma determinada rodada.

Os dois structs implementados no programa são utilizados para representar os atributos dos jogadores e das cartas.

2.3. Funções

- **ListaCartas – Ordena**

A função Ordena, pertencente à classe ListaCartas, ordena a lista de acordo com o número das cartas, utilizando o método bubblesort, visto em sala de aula. Após a chamada, a lista se encontra em ordem crescente em relação aos números das cartas.

- **ListaCartas – Classifica**

A função Classifica retorna um inteiro relativo ao valor do tipo de mão que a lista de cartas representa. Para definir o tipo de mão que um jogador possui, foram criadas 9 funções auxiliares a essa, uma para cada tipo de jogada diferente, que retornam se uma mão é ou não é desse tipo de jogada. Essas funções são nomeadas RSF, SF, FK, FH, F, S, TK, TP e OP. Assim, acionando essas funções em ordem de mais importante para menos importante, a função Classifica consegue determinar qual é o tipo de mão que uma lista representa, e, caso nenhuma dessas funções retorne *true*, então o jogador não possui nenhuma das funções, e a função retornará 1.

- **ListaCartas - Outras Funções**

Outras funções importantes que pertencem à classe ListaCartas são:

- AdicionaFinal: Adiciona uma carta no final da lista;
- GetCarta: Acessa a lista de cartas em uma posição determinada;
- ValQuadra: Retorna o valor da sua quadra, se a carta possuir uma;
- ValTrinca: Retorna o valor de sua trinca, se a carta possuir uma.

- **ListaJogadores – OrdenaPorCartas**

Essa função ordena a lista de jogadores tendo como base a classificação das jogadas dos jogadores da lista, em ordem decrescente. Para realizar a ordenação, foi utilizado o método bubblesort, invertido, para que o resultado seja uma ordem decrescente. Dessa forma, o jogador com a maior classificação ficará na primeira posição da lista, e o jogador com a menor classificação, em último.

- **ListaJogadores – Ganhadores**

A função Ganhadores é primordial para o funcionamento do programa, uma vez que ela é utilizada para determinar os ganhadores de uma jogada específica. Ela retorna uma lista de jogadores que contém apenas os ganhadores de uma determinada rodada. Para determinar os ganhadores, a função primeiramente ordena a lista de acordo com as classificações e separa os membros da lista que possuem a maior classificação.

Com os membros com a maior classificação separados, a função aplica os métodos de desempate para esses indivíduos. Esses métodos são aplicados caso a caso, um para cada classificação, e contam com uma *array* auxiliar que determina se um jogador deve ou não ser incluído nos ganhadores.

- **ListaJogadores – TesteSanidade e TesteSanidadeApostas**

Essas duas funções, pertencentes à lista de jogadores, realizam os testes de sanidade de uma rodada, um para o pingo dos jogadores, e outro para as suas apostas. Caso a rodada não passe nos testes de sanidade, a função retorna false.

- **ListaJogadores – BuscaAposta e BuscaRecebe**

As funções BuscaAposta e BuscaRecebe realizam uma busca em uma lista de jogadores por nome, e quando encontrado o nome em um dos jogadores, realizam uma aposta em determinado valor, ou adicionam determinado valor ao dinheiro do jogador.

- **ListaJogadores – OrdenaPorDinheiro**

Essa função é similiar à função OrdenaPorCartas, porém, ao invés de ordenar a lista em relação à classificação de suas cartas, ordena a lista em relação ao dinheiro que cada jogador possui. Caso a quantidade de dinheiro seja a mesma, o critério de desempate é a ordem alfabética dos nomes. Para a ordenação, foi também utilizado o método bubblesort.

- **ListaJogadores – Outras Funções**

Outras funções importantes na classe ListaJogadores são:

- Imprime: Imprime o nome de todos os jogadores da lista;

- ImprimeResultado: Imprime o nome e o dinheiro de cada jogador, em ordem decrescente;
- MaiorJogada: Retorna a maior jogada entre as cartas dos jogadores da lista;
- GetJogador: Acessa a lista de jogadores em uma posição determinada.

3. Análise de Complexidade

3.1. Tempo

O programa computa n jogadas, e para cada jogada, tendo j jogadores em cada jogada, realiza os seguinte processos:

1. Adiciona os jogadores das respectivas rodadas em uma lista (O(j) para todos os jogadores);
2. Adiciona 5 cartas ao final da lista de cartas para cada jogador ((O(5) para todas as cartas);
3. Realiza os testes de sanidade para todos os jogadores (O(j));
4. Realiza as apostas para todos os jogadores (O(j));
5. Encontra os jogadores ganhadores ($O(j^2)$ no pior caso);
6. Adiciona o dinheiro das apostas divididos entre os ganhadores (O(4) no pior caso).

Assim, temos que o programa realiza pelo menos $n(j + 5 + j + j + j + j^2 + 4)$ operações no pior caso. Como no pior caso, j será igual a 10, podemos considerar a parte que depende de j como uma constante.

Dessa maneira, a complexidade assintótica do programa é O(n).

3.2. Espaço

Como a alocação do programa se dá de maneira estática, o valor alocado é constante. Existe uma lista contendo todos os jogadores, uma lista para cada rodada específica e outra para os ganhadores, todas de tamanho 10. E para cada elemento dessas listas, existe uma lista de cartas de tamanho 5. Dessa forma, há 150 espaços alocados, e a complexidade de espaço do código é O(150).

4. Estratégias de Robustez

Para aumentar a robustez do programa e a sua tolerância a erros, o programa foi implementado de forma a considerar apenas os jogadores que apostaram na rodada inicial, nas rodadas seguintes. Assim, quaisquer jogadores que não jogaram na

primeira rodada, por mais que sejam declarados em alguma rodada seguinte, não farão parte do jogo. Além disso, foram implementados testes de sanidade, que invalidam rodadas em que algum jogador aposta mais dinheiro do que ele possui. Nesses casos, a rodada ainda ocorre, porém ninguém recebe ou paga nada.

Além disso, a implementação de listas e suas funções em TADs facilita a compreensão e a implementação do código.

5. Análise Experimental

A análise experimental do código foi feita a partir do programa *geracaocargapoquer*, feito pelo professor Wagner Meira Junior, que gera um arquivo com cargas aleatórias que serve como entrada para o código.

Nos casos de teste gerados pelo programa, não foram observadas anomalias, e o programa funcionou como esperado.

6. Conclusão

Por fim, foi possível implementar um jogo simplificado de poker, com apostas e diferentes tipos de mão.

A partir da realização desse trabalho, foi possível ampliar o meu conhecimento sobre a implementação de listas, e de suas propriedades. Implementar um jogo de poker, por mais que simplificado, se mostrou um desafio para mim, uma vez que eu tive que implementar as estruturas de dados. Além disso, outro ponto de dificuldade foi a criação de algoritmos para o desempate das cartas, que tinha que levar em consideração parâmetros diferentes, dependendo do tipo de jogos que havia em cada carta. Por fim, a superação dessas dificuldades possibilitou a fixação de diversos conhecimentos.

7. Referências

https://virtual.ufmg.br/20221/course/view.php?id=11607

Apêndice: Instruções para compilação e execução

Para compilar o código, primeiramente, deve-se acessar a pasta TP por meio do terminal, e, estando nessa pasta, digitar o comando *make all*. Assim, um arquvo chamado *tp1.exe* s*erá criado no diretório ./bin/*. Assim, basta executar o arquivo para que o código entre em funcionamento. Note que é necessário que exista um arquivo com nome e*ntrada.txt* no diretório principal para que o programa funcione de forma correta.

Uma vez executado, o programa gerará, também no diretório principal um arquivo de nome *saida.txt*, onde haverá as saídas do programa.