

Trabalho Prático 1

Poker Face

Ítalo Dell'Areti

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

italodellareti@ufmg.br

1. Introdução.

O problema proposto é a criação de um jogo de poker no qual não vamos utilizar dinheiro verdadeiro e todas as decisões serão gerenciadas pelo programa. Sendo assim, qualquer pessoa poderá jogar, mesmo aqueles que desconhecem as regras, pontuação das cartas, noção de rodadas e ganhadores, poderão participar do jogo. Com base nos tipos abstratos de dados, alocação dinâmica e processos de ordenação podemos implementar este problema de forma a respeitar todas as nuances de um jogo de poker. Tendo em vista isso, este documento possui como objetivo explicitar o funcionamento desta implementação.

2. Implementação

O programa foi desenvolvido utilizando a linguagem de programação C++, compilada pelo G++ da GNU Compiler Collection. Ademais, o programa foi produzido e testado em um ambiente Linux distribuição Ubuntu 20.04 LTS utilizando Visual Studio Code.

2.1 - Modularização e Organização

Temos como arquivos os headers `banco.h`, `carta.h`, `io.h`, `jogador.h`, `jogo.h`, `mao.h` e `sort.h`. A `sort.h` possui algoritmo de ordenação padrão com ordenação decrescente, sendo `arr` o vetor a ser ordenado e `n` o número de elementos no vetor.

A implementação do programa foi realizada utilizando as classes **banco**, **jogo**, **mao**, **jogador** e as estruturas **io** e **main**. A **main** é responsável pela chamada de todas as funções, suas operações de escrita e leitura dos arquivos texto onde são armazenados os resultados. Já o **io** é responsável pelo tratamento dos dados obtidos, tanto com leitura e as informações de cada round.

A classe **banco** possui a estrutura conta e é responsável por gerenciar a quantidade máxima de jogadores, o saldo e a modificação do saldo de cada jogador. Dessa forma, ela administra a parte financeira dos jogadores.

Na classe **jogo** é responsável pela criação de todos os requisitos para se criar o jogo, recebendo os arquivos de entrada e saída, lendo número de rounds, jogadores e o dinheiro inicial. Após isso, cria a partida, resolve os vencedores de cada rodada e distribui os potes.

Na Classe **mao** é onde se realiza as classificações das mãos dos jogadores. Além disso, é nesta parte que é resolvido as combinações especiais como Royal Flush, Full House e entre outros.

E por final a classe **jogador** possui como propósito receber o nome do jogador, mão do jogador e o valor da aposta.

2.2 - Estrutura de Dados

Para atender as demandas em relação às operações necessárias, vamos analisar as principais funções do programa.

Em banco temos as funções

banco : É o construtor da classe Banco, sendo n_contas o número de contas do banco, jogadores Vetor de jogadores e dinheiro_inicial o saldo inicial de cada conta.

modifica_saldo : Modifica o saldo de apenas uma conta, indexada pelo nome. Sendo nome o Nome da conta e quantia a Quantia a ser adicionada ou subtraída.

mostra_saldo : Imprime o saldo de todas as contas em saída.

modifica_todos_saldos : Modifica o saldo de todas as contas, sendo quantia a Quantia a ser adicionada ou subtraída.

Em jogo temos as funções

comeca : Primeira rodada do jogo. É especial pois nela descobriremos os jogadores e começaremos o banco.

primeira_rodada : Primeira rodada do jogo. É especial pois nela descobriremos os jogadores e começaremos o banco.

n_vencedores : Conta o número de jogadores que venceram a rodada.

nomes_jogadores : Escreve no arquivo de saída os n primeiros jogadores

distribui_pote : Distribui o pote entre os vencedores da rodada, sendo jogadores os jogadores da rodada, n número de jogadores que venceram e pote quantidade total do pote.

resultado : Imprime o resultado do jogo (nomes dos jogadores com o saldo, ordenados pelo saldo).

Em io temos as funções

inverte : Inverte a string (Para resolver o problema do nome composto)

parse_jogada : Lê uma jogada a partir de uma linha.

get_info_jogo : Lê as informações do jogo a partir de uma linha.

get_info_rodada : Lê as informações da rodada a partir de uma linha.

Em jogador temos as funções

get_mao : Getter da mão do jogador.

get_aposta : Getter da aposta do jogador.

get_nome : Getter do nome do jogador, retorna std::string com o nome do jogador.

Em mao temos as funções

eh_royal_flush : Função que verifica se a mão é um royal flush.

eh_straight_flush : Função que verifica se a mão é um straight flush.

eh_four_of_a_kind : Função que verifica se a mão é uma quadra.

eh_full_house : Função que verifica se a mão é um full house.

eh_flush : Função que verifica se a mão é um flush.

eh_straight : Função que verifica se a mão é um straight.

eh_three_of_a_kind : Função que verifica se a mão é uma trinca.

eh_two_pair : Função que verifica se a mão é dois pares.

eh_one_pair : Função que verifica se a mão é um par

high_card : Função que pega a carta alta de uma mão (desconsiderando o que já faz parte de uma combinação).

get_valor_par : Função que pega o valor do primeiro par que encontrar na mão (apenas um par mesmo, ignora trincas ou quadras).

par_menor : Função que pega o valor do menor par na mão (para as combinações de dois pares).

par_maior : Função que pega o valor do maior par na mão (para as combinações de dois pares).

get_tripla : Função que pega o valor da trinca na mão.

determina_forca : Função que determina a força da mão que o jogador possui. Com valores para o desempate de combinações "iguais". Também determina qual é a combinação da mão.

Em sort.h temos a função

insertion_sort : Insertion sort padrão com ordenação decrescente.

3. Análise Complexidade

3.1 - Tempo

Não foi especificado se o jogo seria de Texas Hold'em para que cada jogador recebesse apenas 2 cartas para formar sua mão junto às 5 comunitárias, então supõe-se que as cartas em jogo serão 5 cartas únicas para cada jogador, totalizando assim $4 \text{ naipes} \times 13 \text{ cartas} = 52 \text{ cartas}$, o que nos dá o limite máximo de 10 jogadores por partida.

Tendo em vista que o custo da maior parte dos métodos está limitado ao número de cartas na mão (5) e ao número máximo de jogadores (10), o custo assintótico de todas as funções, exceto uma, fica em $O(1)$. Então até mesmo a função mais custosa, sendo a de ordenação, com custo $O(n^2)$, ficando limitada à $n \leq 10$ pode ser considerada $O(1)$.

Sendo assim, o único método que vai variar na execução com o tamanho da entrada é o **Jogo::comeca()**, com custo $O(n)$, onde $n = n_{rodadas}$.

Após esta breve visão sobre o programa como um todo, agora vamos mostrar algumas ordens de complexidade das principais funções e classes do programa, tendo em vista que seus impactos influenciam no tempo de execução do programa. Dessa forma, assim, a complexidade das principais funções são :

banco : (classe) Custo: $O(n)$ pois n_contas é o número de contas do banco.

modifica_saldo : Custo: $O(n)$ por conta da busca sequencial pela conta a partir do nome.

mostra_saldo : Imprime o saldo de todas as contas de n jogadores, se for considerada como uma função genérica seu é Custo $O(n)$, mas também pode ser $O(1)$ pois temos no máximo 10 jogadores

modifica_todos_saldos : Modifica o saldo de todas n contas. Custo: $O(n)$.

comeca : Custo: $O(n)$.

primeira_rodada : n número de jogadores na rodada, Custo: $O(n)$.

n_vencedores : Custo: $O(n)$.

nomes_jogadores : Custo: $O(n)$.

distribui_pote : Distribui o poste para n jogadores, Custo: $O(n)$.

resultado : resultado de n jogadores. Custo: $O(n)$.

inverte : Inverte input de n jogadores. Custo: $O(n)$.

parse_jogada : Custo: $O(n)$. Onde n é o número de cartas na mão, mas as cartas que temos sempre 5. Então, neste caso é $O(1)$.

get_info_jogo : Custo $O(1)$.

get_info_rodada : Custo $O(1)$.

get_mao : Custo: $O(1)$.

get_aposta : Custo: $O(1)$.

get_nome : Custo: $O(1)$.

eh_royal_flush : Custo: $O(1)$.

eh_straight_flush : Custo: $O(1)$.

eh_four_of_a_kind : Custo: $O(1)$.

eh_full_house : Custo: $O(1)$.

eh_flush : Custo: $O(1)$.

eh_straight : Custo: $O(1)$.

eh_three_of_a_kind : Custo: $O(1)$.

eh_two_pair : Custo: $O(1)$.

eh_one_pair : Custo: $O(1)$.

high_card : Custo: $O(n)$. Onde n é o número de cartas na mão, que é sempre 5. Então, neste caso pode ser considerado $O(1)$.

get_valor_par : Custo: $O(n)$. Onde n é o número de cartas na mão, que é sempre 5. Então, neste caso pode ser considerado $O(1)$.

par_menor : Custo: $O(n)$. Onde n é o número de cartas na mão, que é sempre 5. Então, neste caso pode ser considerado $O(1)$.

par_maior : Custo: $O(n)$. Onde n é o número de cartas na mão, que é sempre 5. Então, neste caso pode ser considerado $O(1)$.

get_tripla : Custo: $O(n)$. Onde n é o número de cartas na mão, que é sempre 5. Então, neste caso pode ser considerado $O(1)$.

determina_forca : Custo: $O(1)$.

insertion_sort : Insertion sort padrão com ordenação decrescente de ordem $O(n^2)$, sendo o pior caso quando a entrada está ordenada em ordem decrescente com custo $O(n^2)$ e o melhor caso quando o array já está ordenado com custo $O(n)$.

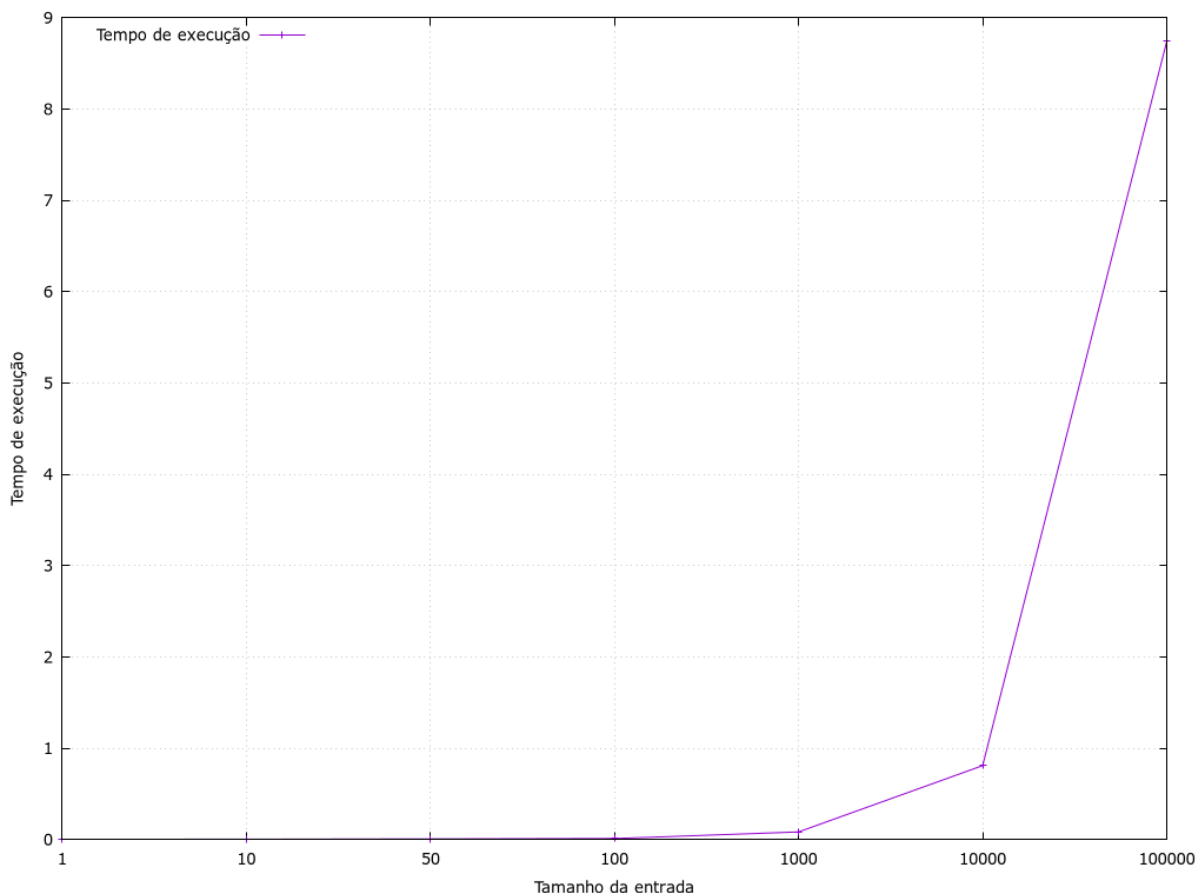
3.2 - Espaço

No pior dos casos, temos acessos à estruturas com $n_{\text{jogadores}} \times n_{\text{cartas}}$, sendo $n_{\text{jogadores}} \leq 10$ e $n_{\text{cartas}} = 5$, logo ocupando $O(1)$.

4. Análise Experimental

4.1 - Desempenho computacional

Utilizando os resultados de testes de tempo de todo o programa, incluindo as operações de leitura e escrita de arquivos e alocação de memória, com número de rodadas de 10 a 100 000, obtivemos o seguinte gráfico no gnuplot.



4.2 - Depuração de desempenho

Apenas com grandes quantidades de rodadas (acima de 10 mil) o **gprof** começa a acumular tempo na análise, com o método **Jogo::comeca()** tendo cerca de meio segundo de tempo de execução.

```
Flat profile:
Each sample counts as 0.01 seconds.
 %   cumulative   self           self      total
time  seconds    seconds   calls   s/call   s/call   name
41.38    0.48    0.48         1    0.48    1.15  Jogo::comeca()
 8.62    0.58    0.10    100000    0.00    0.00  Jogo::n_vencedores(Vetor<Jogador>, int)
 6.90    0.66    0.08    680935    0.00    0.00  void troca<Jogador>(Jogador&, Jogador&)
 6.03    0.73    0.07    600239    0.00    0.00  parse_jogada(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&)
 5.17    0.79    0.06    713162    0.00    0.00  Banco::modifica_saldo(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, int)
 4.31    0.84    0.05    6666649    0.00    0.00  Mao::Mao()
 3.45    0.88    0.04    100000    0.00    0.00  Jogo::nomes_jogadores(Vetor<Jogador>, int)
 2.59    0.91    0.03    600239    0.00    0.00  Mao::parse_cartas(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&*)
 2.59    0.94    0.03    600239    0.00    0.00  Mao::Mao(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&*)
 2.59    0.97    0.03    596877    0.00    0.00  Mao::high_card()
 1.72    0.99    0.02    3601434    0.00    0.00  get_token_inverso(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&)
 1.72    1.01    0.02    600239    0.00    0.00  Jogador::Jogador(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, Mao const&, int)
 1.72    1.03    0.02    600231    0.00    0.00  Mao::eh_four_of_a_kind()
 1.72    1.05    0.02    314184    0.00    0.00  Jogador::get_mao()
 1.72    1.07    0.02    200002    0.00    0.00  get_token(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&)
 0.86    1.08    0.01    4801912    0.00    0.00  invert(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&)
 0.86    1.09    0.01    1800717    0.00    0.00  Jogador::get_aposta()
 0.86    1.10    0.01    1799694    0.00    0.00  Mao::eh_flush()
 0.86    1.11    0.01    1200478    0.00    0.00  Mao::eh_straight_flush()
 0.86    1.12    0.01    600239    0.00    0.00  Mao::eh_royal_flush()
 0.86    1.13    0.01    600239    0.00    0.00  Mao::determina_forca()
 0.86    1.14    0.01    554506    0.00    0.00  Mao::eh_one_pair()
 0.86    1.15    0.01    100000    0.00    0.00  Jogo::distribui_pote(Vetor<Jogador>, int, int)
 0.86    1.16    0.01          _init
```

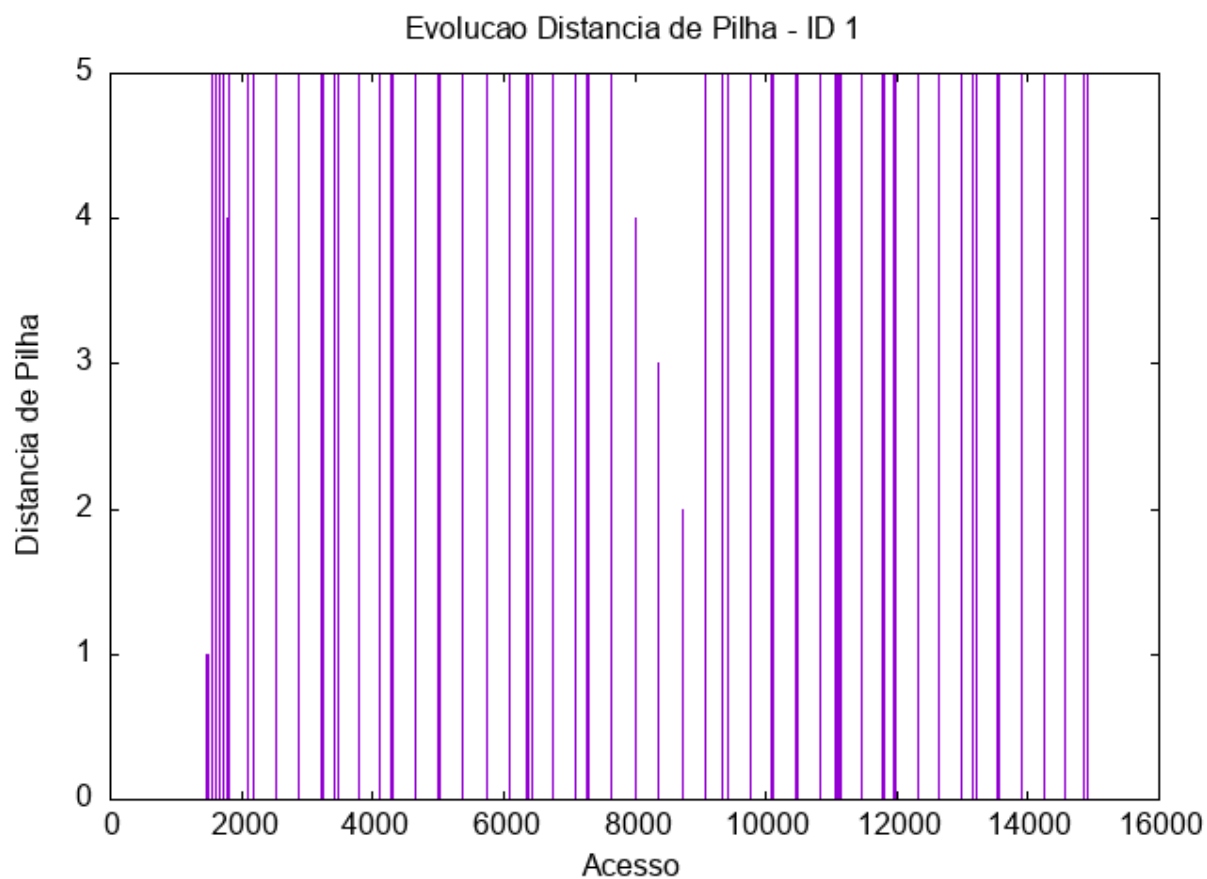
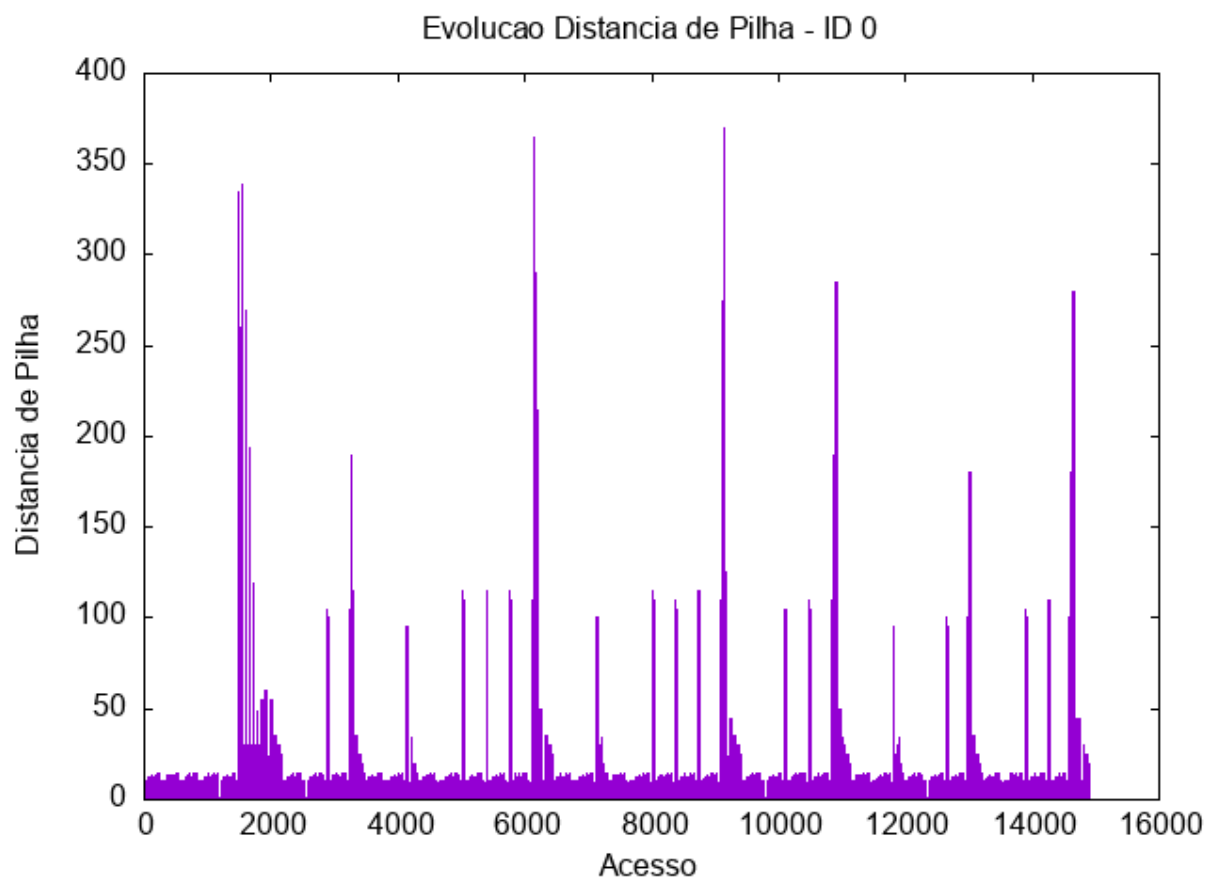
Saída da análise do gprof com 100 mil rodadas e 10 jogadores:

Os métodos com tempo igual à 0 foram cortados da imagem.

4.3 - Análise de Padrão de Acesso à Memória e Localidade de Referência

O teste foi realizado com 10 rodadas e 5 jogadores, para tal, a biblioteca **memlog** foi usada e os acessos de registro ao endereços foram salvos, utilizando o programa **analisamem**, foram plotadas gráficos e histogramas sobre o acessos e ocupação de espaço.

Para a análise dos gráficos de distância de pilha e acesso, considere o ID 0 corresponde à estrutura das cartas, o ID 1 à estrutura do banco e o ID 2 ao vetor com os jogadores.



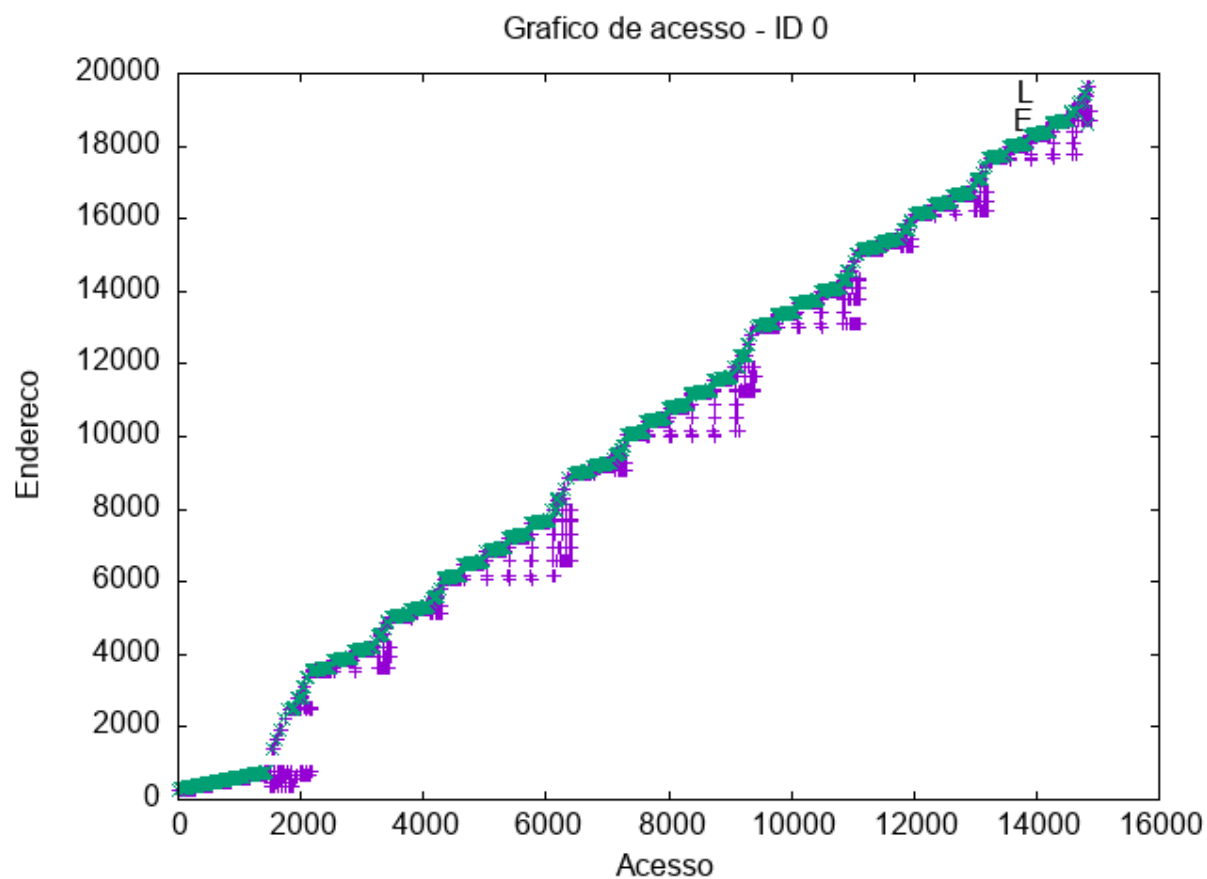
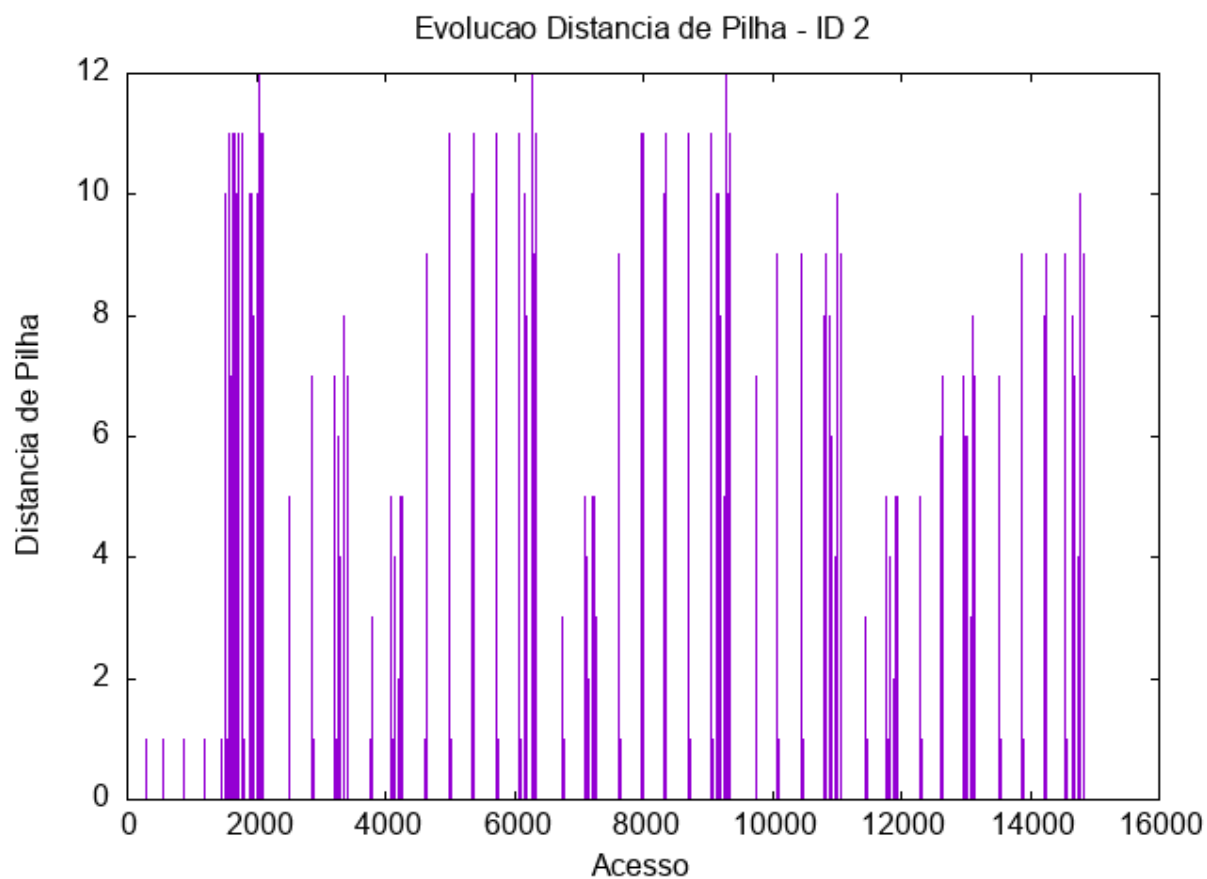


Grafico de acesso - ID 1

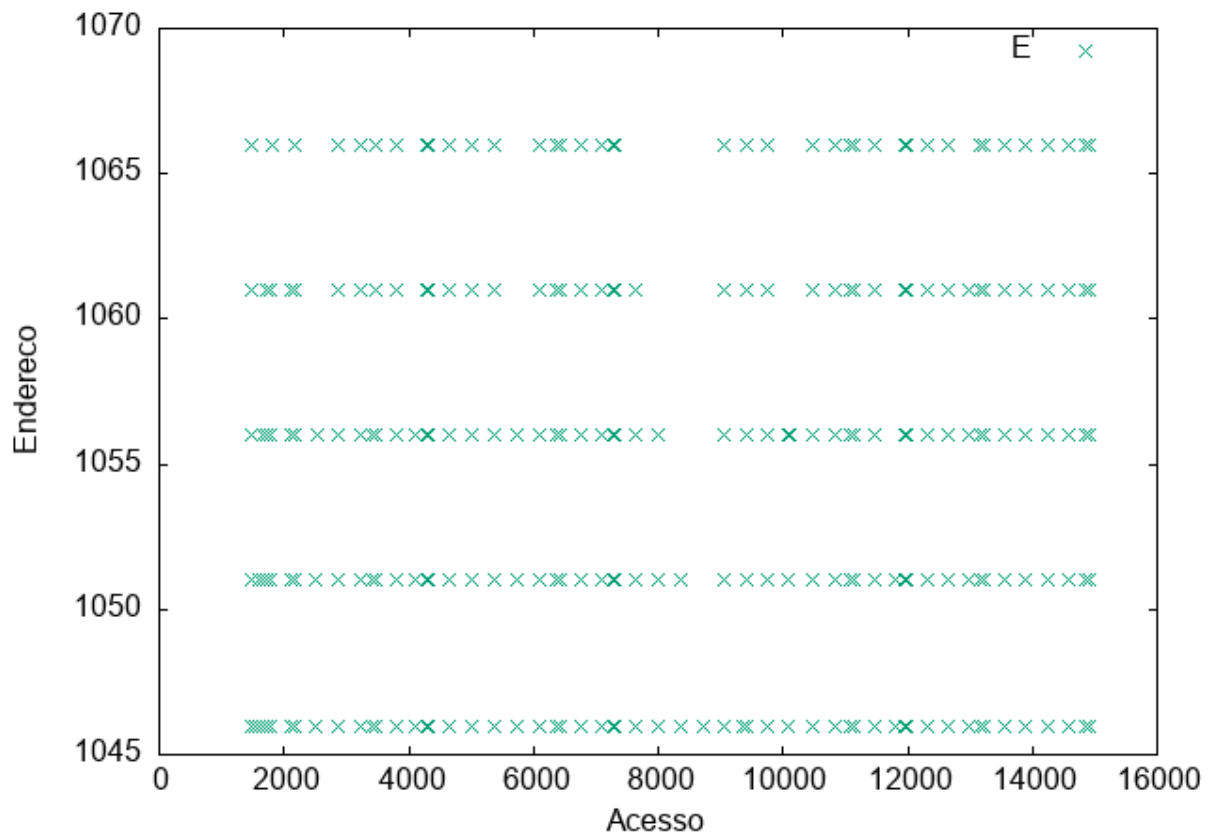
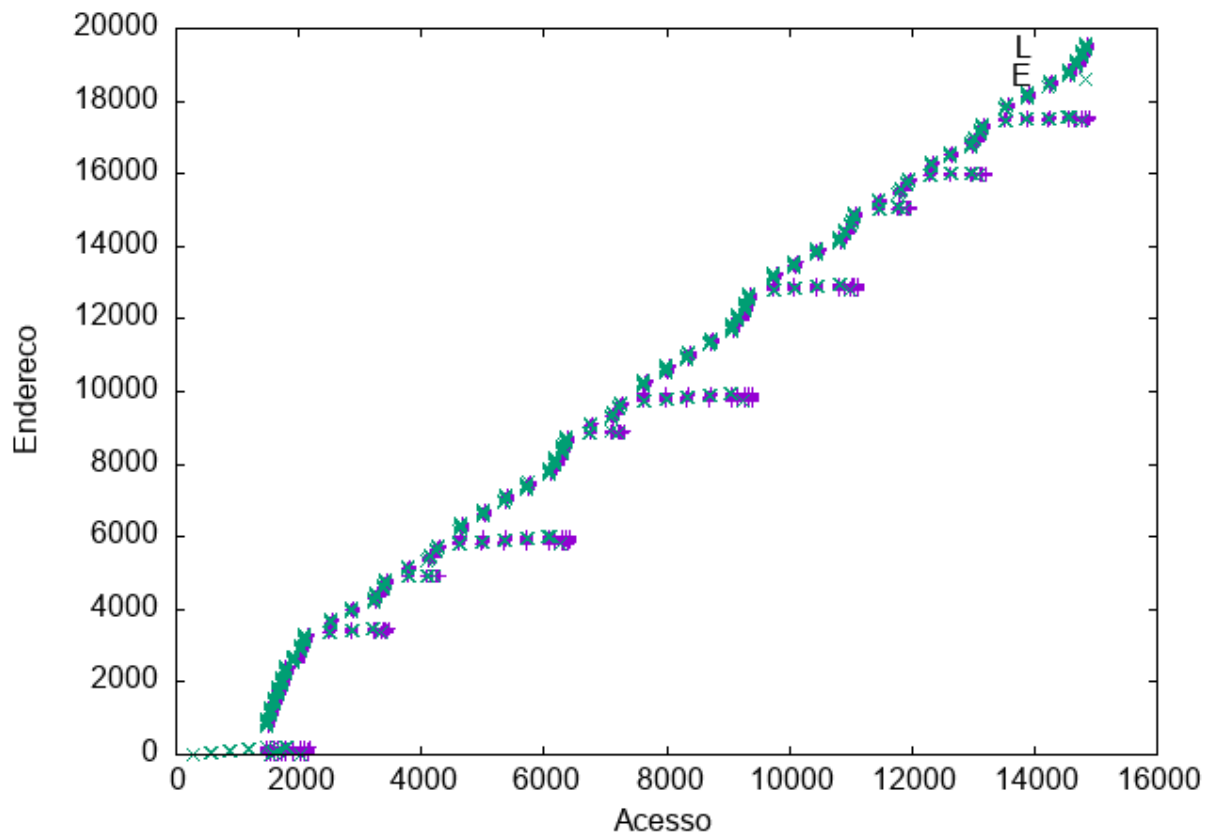


Grafico de acesso - ID 2



Através da interpretação destes gráficos podemos inferir que a distância de pilha não muda com o número de rodadas, ela só se altera com o número de jogadores. Também podemos notar que a distância se mantém praticamente toda no tamanho do objeto.

Ademais, o comportamento do gráfico de acesso possui as características esperadas de acordo com as estruturas de dados que são utilizados e são correspondentes às estrutura das cartas, estrutura do banco e o vetor com os jogadores.

5. Robustez

Para lidar com apostas inválidas, onde o valor não é múltiplo de 50 e quando o jogador não tem saldo para apostar, após a verificação, é devolvido o saldo descontado dos jogadores e a rodada é ignorada.

Para lidar com nomes compostos, a entrada da rodada é lida de trás para frente, dando *parse* nas 5 cartas primeiro, em seguida na aposta e o que resta na *string* é guardado como o nome.

6. Testes

6.1 Caso de teste de funcionamento

Para testarmos o funcionamento vamos utilizar o caso teste fornecido

```
3 1000

5 50

Giovanni 100 6O 3P 10E 11O 1O
John 200 3P 4E 3E 13C 13O
Thiago 100 12O 7P 12C 1O 13C
Gisele 300 12E 10C 11C 9C 13E
Wagner 50 5P 12P 5E 2E 1P

2 50

Wagner 200 2P 13E 9E 12C 2O
Gisele 350 11P 9P 2E 6E 4P

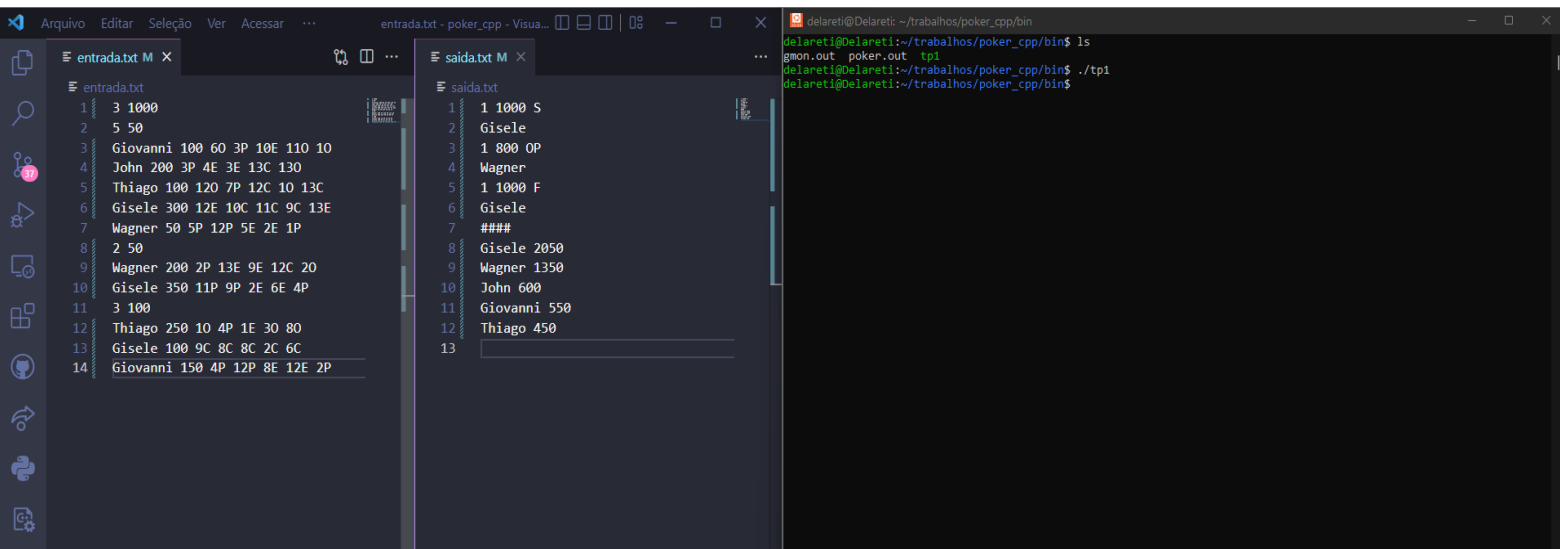
3 100

Thiago 250 1O 4P 1E 3O 8O
Gisele 100 9C 8C 8C 2C 6C
Giovanni 150 4P 12P 8E 12E 2P
```

Saída

```
1 1000 S
Gisele
1 800 OP
Wagner
1 1000 F
Gisele
####
Gisele 2050
Wagner 1350
John 600
Giovanni 550
Thiago 450
```

Resultado após a inserção e execução dos dados



The screenshot displays a development environment with three windows. The left window, titled 'entrada.txt M x', shows a list of 14 lines of input data. The middle window, titled 'saida.txt M x', shows the corresponding output data. The right window is a terminal showing the execution of the program, including directory listing and command execution.

```
Arquivo Editar Seleção Ver Acessar ... entrada.txt - poker_cpp - Visua... - - - x
```

```
entrada.txt M x
1 3 1000
2 5 50
3 Giovanni 100 60 3P 10E 110 10
4 John 200 3P 4E 3E 13C 130
5 Thiago 100 120 7P 12C 10 13C
6 Gisele 300 12E 10C 11C 9C 13E
7 Wagner 50 5P 12P 5E 2E 1P
8 2 50
9 Wagner 200 2P 13E 9E 12C 20
10 Gisele 350 11P 9P 2E 6E 4P
11 3 100
12 Thiago 250 10 4P 1E 30 80
13 Gisele 100 9C 8C 8C 2C 6C
14 Giovanni 150 4P 12P 8E 12E 2P
```

```
saida.txt M x
1 1 1000 S
2 Gisele
3 1 800 OP
4 Wagner
5 1 1000 F
6 Gisele
7 ####
8 Gisele 2050
9 Wagner 1350
10 John 600
11 Giovanni 550
12 Thiago 450
13
```

```
delareti@Delareti: ~/trabalhos/poker_cpp/bin
delareti@Delareti:~/trabalhos/poker_cpp/bin$ ls
gmon.out poker.out tp1
delareti@Delareti:~/trabalhos/poker_cpp/bin$ ./tp1
delareti@Delareti:~/trabalhos/poker_cpp/bin$
```

6.2 Caso de desempate (full house)

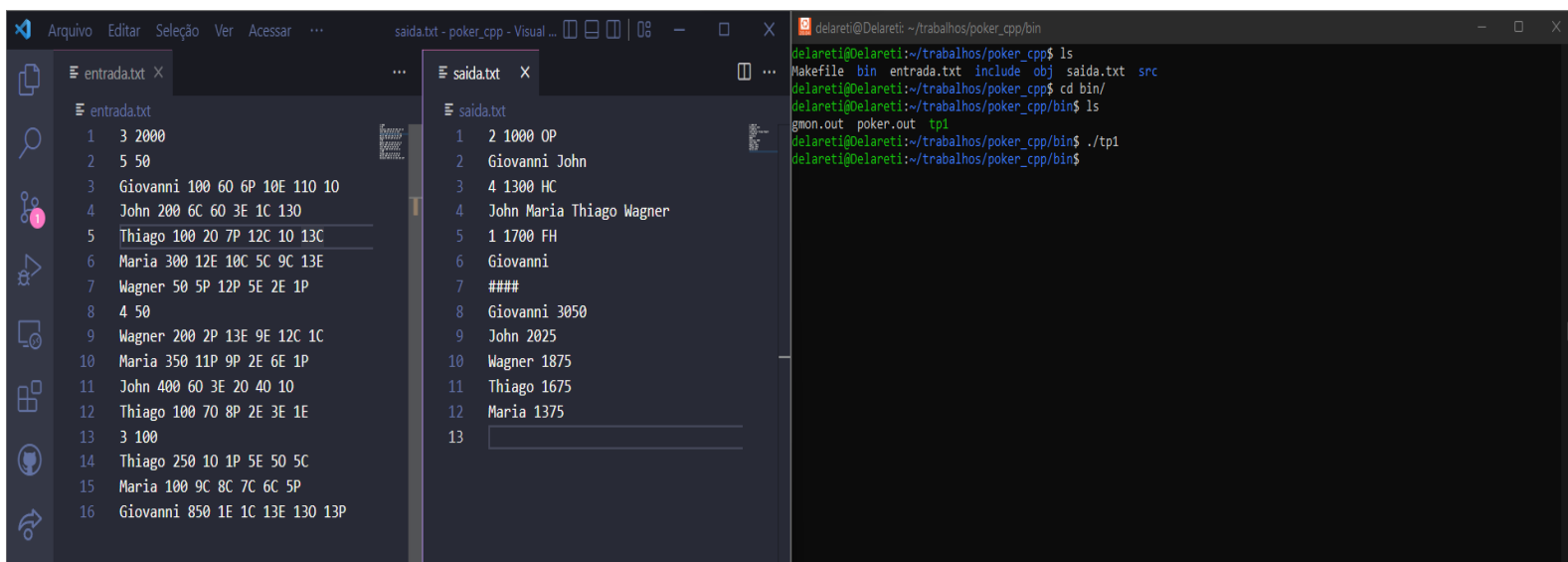
Para testarmos um caso de desempate de full house podemos considerar as seguintes entradas

```
3 2000
5 50
Giovanni 100 6O 6P 10E 11O 1O
John 200 6C 6O 3E 1C 13O
Thiago 100 2O 7P 12C 1O 13C
Maria 300 12E 10C 5C 9C 13E
Wagner 50 5P 12P 5E 2E 1P
4 50
Wagner 200 2P 13E 9E 12C 1C
Maria 350 11P 9P 2E 6E 1P
John 400 6O 3E 2O 4O 1O
Thiago 100 7O 8P 2E 3E 1E
3 100
Thiago 250 1O 1P 5E 5O 5C
Maria 100 9C 8C 7C 6C 5P
Giovanni 850 1E 1C 13E 13O 13P
```

Neste caso, as maiores são as de Thiago, com AA555 e Giovanni, com AAKKK. Dessa forma, a Giovana ganhou no desempate com AAKKK. Portanto, a saída esperada é :

```
2 1000 OP
Giovanni John
4 1300 HC
John Maria Thiago Wagner
1 1700 FH
Giovanni
####
Giovanni 3050
John 2025
Wagner 1875
Thiago 1675
Maria 1375
```

Resultado após a inserção e execução dos dados



```
Arquivo  Editar  Seleção  Ver  Acessar  ...  saida.txt - poker_cpp - Visual ...  delareti@Delareti: ~/trabalhos/poker_cpp/bin

entrada.txt  ...  saida.txt

1  3 2000
2  5 50
3  Giovanni 100 60 6P 10E 110 10
4  John 200 6C 60 3E 1C 130
5  Thiago 100 20 7P 12C 10 13C
6  Maria 300 12E 10C 5C 9C 13E
7  Wagner 50 5P 12P 5E 2E 1P
8  4 50
9  Wagner 200 2P 13E 9E 12C 1C
10 Maria 350 11P 9P 2E 6E 1P
11 John 400 60 3E 20 40 10
12 Thiago 100 70 8P 2E 3E 1E
13 3 100
14 Thiago 250 10 1P 5E 50 5C
15 Maria 100 9C 8C 7C 6C 5P
16 Giovanni 850 1E 1C 13E 130 13P

saida.txt

1  2 1000 OP
2  Giovanni John
3  4 1300 HC
4  John Maria Thiago Wagner
5  1 1700 FH
6  Giovanni
7  #####
8  Giovanni 3050
9  John 2025
10 Wagner 1875
11 Thiago 1675
12 Maria 1375
13

delareti@Delareti:~/trabalhos/poker_cpp/bin$ ls
delareti@Delareti:~/trabalhos/poker_cpp/bin$ makefile bin entrada.txt include obj saida.txt src
delareti@Delareti:~/trabalhos/poker_cpp/bin$ cd bin/
delareti@Delareti:~/trabalhos/poker_cpp/bin$ ls
gmon.out  poker.out  tp1
delareti@Delareti:~/trabalhos/poker_cpp/bin$ ./tp1
delareti@Delareti:~/trabalhos/poker_cpp/bin$
```

Vídeo dos testes funcionando

Google Drive:

<https://drive.google.com/file/d/1gAZbaz2pgYixUrsdRXu6c6JJdholtfqW/view?usp=sharing>

Dropbox: <https://www.dropbox.com/s/rcp92y2rkdhcmle/Teste%20casos.mp4?dl=0>

7. Conclusão

Ao final deste trabalho, podemos inferir que este possui o propósito de abordar e trabalhar e ampliar a visão sobre o funcionamento de ordenação de objetos e como elas se relacionam em uma aplicação real, sendo esta em um jogo no qual devemos realizar diversas comparações de calcular qual mão é maior que a outra. Sendo assim, por meio de poker a todas as nuances que envolvem o jogo, como distribuir cartas, verificar ganhador, distribuir o pote e entre outros, foi possível trabalhar com iterações e ordenações, podendo praticar as mais diversas operações.

Bibliografia

BACKES, A. R. (2018). *Linguagem C - Completa e Descomplicada*

Ziviani, N., *Projeto de Algoritmos com Implementações em Pascal e C*

[Geeks for geeks - Insertion Sort](#)

[GNU PLOT - A Brief Manual and Tutorial](#)

[GPROF Tutorial – How to use Linux GNU GCC Profiling Tool](#)

Instruções para compilação e execução

Para executar o programa basta dar um comando **make** para executar tudo.

make all - compila tudo

make distclean - remove objetos

make clean - remove objetos e executável

Além disso, o arquivo de texto entrada.txt **deve estar no diretório raiz da pasta do programa para o seu funcionamento.**

Os resultado do jogo vai aparecer na pasta raiz do projeto através do arquivo texto saida.txt

Sempre que mudar quaisquer valores de entrada, você deverá ir na pasta bin e executar novamente o comando ./tp1 para atualizar os valores da saída para os valores verdadeiros.