Trabalho Prático – AEDS II
Gustavo Cavalieri Fernandes – Universidade Federal de Minas Gerais
Documentação
Trabalho Prático 1 – 1º Semestre 2011

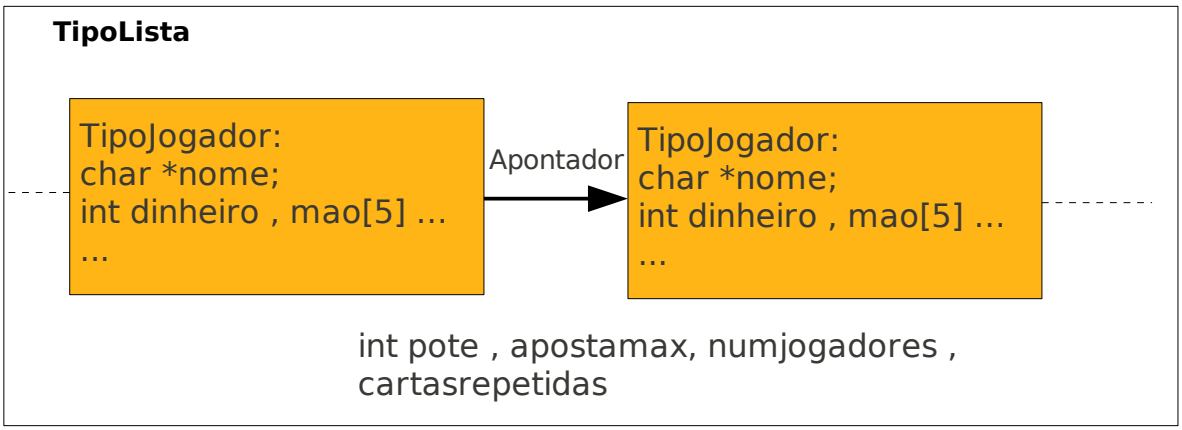
1. Introdução:

O objetivo desse trabalho foi implementar um programa para um jogo de poker tal que, dada uma entrada X, o programa produza uma saída Y com o nome dos vencedores de cada rodada, o montante ganho e ao final , o dinheiro final de cada jogador. Além disso deseja-se praticar os conceitos de TAD(Tipo Abstrato de Dados) e outros conceitos ,que são essenciais para o Curso de Ciência da Computação.

2. Implementação:

Estrutura de Dados:

O trabalho foi implementado através de uma Lista Encadeada de Jogadores lidos a partir de uma entrada. Cada Jogador(TipoJogador) foi inserido como uma célula da Lista(TipoLista). A figura abaixo ilustra bem a estrutura de dados utilizada:



Funções e Procedimentos:

O TAD da Lista encadeada possui as seguintes funções:

void FazListaVazia(TipoLista *l); Inicializa a estrutura da lista, inserindo o primeiro jogador como elemento Nulo e zerando variáveis da Mesa(Pote, apostamax..)

void insere(TipoJogador x , TipoLista *l); Insere um jogador no final da Lista

Apontador pesquisa(char *nome , TipoLista l); Pesquisa um determinado jogador na lista e retorna um apontador para o mesmo.

int lejogador(FILE *fp, int dinicial, TipoLista *l); Lê o arquivo e guarda os dados do jogador na lista. Nela é convertida a mão do jogador e são realizados os testes de sanidade. Retorna FALSO caso o teste de sanidade seja confirmado.

int convertemao(FILE *fp , TipoJogador *j , TipoLista *l); Lê e converte todas as cartas do arquivo o vetor de inteiros. Esse vetor é guardado num TipoJogador auxiliar para futura inserção.

void copiamao(TipoJogador j , Apontador aux); Copia a mão convertida da estrutura auxiliar para o Jogador já inserido na Lista(Apontador aux).

void pingo(TipoLista *l); Subtrai de todos os jogadores o valor do pingo.

int cartarepetida(int carta ,TipoLista *l); Recebe uma carta e verifica se ela é repetida em determinada rodada. Caso não seja, elá é guardada em um vetor.

void verificamao(TipoLista *l); Determina a jogada do jogador de acordo com suas cartas.

void verificasaida(TipoLista *l); Verifica se a aposta do jogador é igual a aposta máxima da mesa, caso negativo a variável **int saiu** do jogador se torna TRUE.

void ordenamao(int a[]); Ordena o vetor de inteiros que corresponde a mão do jogador.

void ordenadinheiro(TipoJogador *j); Ordena o dinheiro dos os jogadores da lista através de uma estrutura auxiliar para a impressão do resultado final.

void novarodada(TipoLista *l); Zera as variáveis da mesa(pote, apostamax) e as variáveis de cada jogador(saiu, aposta, jogada) , inicializando assim uma nova rodada.

int nomecomposto(char *nome); Verifica se a string recebida é um nome e retorna TRUE caso verdadeiro.

void determinaganhador(FILE *fw , TipoLista *l); Determina o ganhador da rodada de acordo com os jogadores que estão no jogo e suas jogadas.

int cartamaisalta(Apontador aux); Verifica se a jogada do jogador é desempatada pela carta mais alta da mão, caso os dois jogadores possuam essa jogada.

void dividepote(int ganhadores, Apontador *j , TipoLista *l); Divide o pote entre o número de jogadores.

void imprimerodada(FILE *fw , int ganhadores, Apontador *j , TipoLista *l); Imprime o resultado da rodada.

void imprimeresultado(FILE *fw, TipoLista *l); Imprime o resultado final

int RSF(Apontador aux); Verifica se o jogador possui RSF.

int SF(Apontador aux);Verifica se o jogador possui SF.

int FK(Apontador aux);Verifica se o jogador possui FK.

int FH(Apontador aux);Verifica se o jogador possui FH.

int Flush(Apontador aux);Verifica se o jogador possui Flush.

int Straight(Apontador aux);Verifica se o jogador possui Straight.

int TK(Apontador aux);Verifica se o jogador possui TK.

int TP(Apontador aux);Verifica se o jogador possui TP.

int OP(Apontador aux);Verifica se o jogador possui OP.

int VerificaAs(Apontador aux);Verifica se o jogador possui um Ás como carta mais alta.

void devolvedinheiro(TipoLista *l); Devolve o dinheiro para os jogadores que apostaram em uma rodada inválida.

void freeall(TipoLista *l); Desaloca a memória utilizada na estrutura

Programa Principal:

O programa principal abre os arquivos de leitura e escrita, lê o numero de rodadas e em seguida chama as funções dos TAD na seguinte ordem de procedimentos:

1.Inicializa a Lista → função FazListaVazia()
2.Inicializa uma nova rodada → função NovaRodada()
3.Recolhe o pingo de cada jogador → Função Pingo()
4.Verifica a aposta do jogador e a sua saída da rodada → Função VerificaSaida()
5.Verifica a jogada do jogador → Função VerificaMao()
6.Determina o ganhador → Função VerificaGanhador()

Organização do Código, Decisões de Implementação e Detalhes Técnicos:

- O código está dividido em 3 arquivos, sendo 2 sources(.c) e 1 headers(.h).O header lista.h guarda a estrutura e as funções da lista. Já os sources lista.c, main.c implementam as funções e procedimentos.
- Os valores definidos para as constantes MAXJ, MAO, JF, NAIPE, TRUE e FALSE foram : 100 , 5 , -1 , 13 , 1 , 0 respectivamente.
- As cartas foram implementadas como um vetor de inteiros. Cada carta pode variar de 0 a 51, sendo 0 o correspondente do Ás de Ouros, o 1 o correspondente do 2 de Ouros e assim por diante. Os naipes foram definidos da seguinte maneira:

Naipe	Vetor
Ouros	0 a 12
Espadas	13 a 25
Copas	26 a 38
Paus	39 a 51

- O compilador utilizado foi o GCC (GNU project C and C++ compiler) em um ambiente operacional Linux.

3.Análise de Complexidade

Seja **l** o número de jogadores da lista, **m** o número de jogadores em cada rodada e **c** o número de cartas presente em cada rodada. Genericamente representaremos a complexidade das funções por O(n) e a denotaremos por α.

Funções do TAD Lista:

void FazListaVazia(TipoLista *l); Realiza 4 atribuições. Considerando as atribuições a função é **O(4)≡O(1).**

void insere(TipoJogador x , TipoLista *l); Realiza alocações e atribuições, portanto assim como a função FazListaVazia a função é **O(1).**

Apontador pesquisa(char *nome , TipoLista l); Realiza uma pesquisa sequencial na lista encadeada. Para cada jogador pesquisado há uma comparação. Logo no melhor caso a função é O(1)(o jogador pesquisado é o primeiro) e no pior caso é **O(l)**(o jogador pesquisado pode não estar na lista ou ser o último. Logo a função é **O(l).**

int lejogador(FILE *fp, int dinicial, TipoLista *l); Inicialmente lê dados do arquivo(Número de Jogadores, pingo). Depois inicia uma interação de acordo com o número de jogadores da rodada, aí temos uma interação O(m). É então verificado se o nome do jogador é composto com a função nomecomposto(**O(1)**) e feita uma pesquisa (Função pesquisa(**O(l)**). Logo após são feitas atribuições e a mão do jogador é convertida(Função convertemao(**O(1)**). É ainda verificado se existem cartas repetidas na rodada(Função cartarepetida(**O(c)**). Portanto a função lejogador possui complexidade $O(1) \leq \alpha \leq O(l) \leq O(c)$. **Logo a função é genericamente O(n).**

int convertemao(FILE *fp , TipoJogador *j , TipoLista *l); Realiza 5 interações. Uma para cada carta a ser convertida a partir do arquivo de leitura. Para cada carta lida é são feitas no pior caso 4 comparações(correspondentes aos naipes) e no melhor caso 1 comparação, por fim

é verificada se a carta é repetida(Função cartarepetida(O(c)). Temos na função então $\alpha = 5(4 + O(c)) = 20 + O(c)$. **Genericamente a função é O(n).**

void copiamao(TipoJogador j , Apontador aux); Copia as 5 cartas do vetor auxiliar para o jogador inserido na lista. Faz 5 interações e para cada uma delas uma atribuição. Logo a complexidade da função é **O(5x1) ≡ O(5) ≡ O(1).**

void pingo(TipoLista *l); Para cada jogador presente na lista, subtrai do seu dinheiro o valor do pingo, soma o pingo ao pote a incrementa o apontador de pesquisa. Como temos **l** jogadores na lista , realizamos **3l** atribuições. **Logo a função é O(l).**

int cartarepetida(int carta ,TipoLista *l); Possui uma iteração que possui uma comparação. A iteração roda de acordo com o número de cartas inseridas no vetor de cartas repetidas. A função é chamada **c** vezes. Na sua última chamada o vetor de cartas repetidas possui (**c-1**) elementos. Logo iteração da função será O(c-1) de acordo com o número de comparações. **E a função será genericamente O(c).**

void verificamao(TipoLista *l); Para cada jogador da lista é feito no melhor caso uma comparação. No pior caso as cartas do jogador são ordenadas através da função Ordenamao(O(1)) e são feitas 9 comparações. A complexidade da função é no melhor caso **O(l)** e no pior caso $\mathbf{O(1) \leq \alpha \leq O(9l) \equiv O(l)}$ que é igual ao melhor caso.

void verificasaida(TipoLista *l); Para cada jogador da lista, verifica se a aposta do jogador é igual a maior aposta da mesa. Portanto para **l** jogadores temos **l** comparações. A função é então **O(l).**

void ordenamao(int a[]); Ordena a mão do jogador(vetor de inteiros) por inserção. No melhor caso a função é O(1) → vetor ordenado, e no pior caso $O(n^2)$. Mas o vetor possui 5 cartas então a função é
$\mathbf{O(5^2) \equiv O(25) \equiv O(1).}$

void ordenadinheiro(TipoJogador *j); Ordena por inserção o dinheiro dos jogadores. No melhor caso a função é O(1) → vetor ordenado, e no pior caso $\mathbf{O(l^2).}$

void novarodada(TipoLista *l); Para cada jogador da lista, realiza 4 atribuições. Fora da iteração realiza mais 3 atribuições. Portanto a função é **4.l +3 ≡ O(4l) ≡ O(l).**

int nomecomposto(char *nome); Faz 10 comparações para verificar se o primeiro caractere da string é um número. Portanto a função é **O(10) ≡ O(1).**

void determinaganhador(FILE *fw , TipoLista *l); Vai do segundo jogador até o ultimo procurando a melhor jogada. No melhor caso realiza **(l-1)** comparações → Apenas um jogador permaneceu no jogo. No pior caso, deve-se comparar os critérios de desempate para todos os jogadores. Nele, são feitas em média 7 comparações. Portanto no pior caso a função é 7(**l**-1) = O(7l) ≡ **O(l)**. No melhor caso a função é **O(l-1) ≡ O(l)**.

int cartamaisalta(Apontador aux); Faz 6 comparações , verificando se o jogador possui alguma jogada que é desempatada pela carta mais alta. A função é constantemente **O(6) ≡ O(1).**

void dividepote(int ganhadores, Apontador *j , TipoLista *l); No melhor caso há apenas um ganhador na rodada. Portanto a função realiza uma atribuição. No pior caso, há no máximo 4 ganhadores. Nele a função realiza 4 atribuições. Em ambos os casos a função é O(1) , pois **O(4) ≡ O(1).**

void imprimerodada(FILE *fw , int ganhadores, Apontador *j , TipoLista *l); No pior caso realiza 10 comparações. No melhor, realiza 1 comparação. Portanto a função é constantemente **O(1).**

void imprimeresultado(FILE *fw, TipoLista *l); Guarda cada jogador da lista em uma estrutura auxiliar para ordenação através da função OrdenaDinheiro($O(l^2)$ no pior caso). Depois

imprime o nome e o dinheiro de cada jogador da estrutura auxiliar(**l** iterações). Portanto complexidade da função é no melhor caso **O(l)** e no pior caso **O(l) ≤ α ≤ O(l²) ≡ O(l²).**

int RSF(Apontador aux); Faz 9 comparações no pior caso. No melhor faz 1 comparação. Portanto a função é **O(1) ≤ α ≤ O(9) ≡ O(1).**

int SF(Apontador aux); Faz 4 comparações no pior caso. No melhor faz 1 comparação. Portanto a função é **O(1) ≤ α ≤ O(4) ≡ O(1).**

int FK(Apontador aux); Possui 2 comandos “For” . O mais externo é nomeado pela variável **i** que vai de 0 a 4(máximo de cartas da mão)e possui uma comparação. Já o mais interno vai de i+1 até 4 e possui uma comparação. Portanto a função realiza 4 + 4 +3 + 2 + 1 comparações = **O(14) ≡ O(1).**

int FH(Apontador aux); Possui duas comparações envolvendo as funções: TP(O(1))e OP(O(1)). Portanto **O(1) ≤ α ≤ O(1) ≡ O(1).**

int Flush(Apontador aux); Realiza no melhor caso 2 comparações. Em seu pior caso realiza 5 comparações. Portanto temos: **O(2) ≤ α ≤ O(5) ≡ O(1).**

int Straight(Apontador aux); Realiza uma iteração para pegar apenas o naipe das 5 cartas presentes na mão do Jogador. Em seguida ordena o vetor através da função OrdenaMão(O(1)). Em seguida realiza no melhor caso uma comparação e no pior caso 4 comparações. Portanto a complexidade da função é: **O(1) ≤ α ≤ O(4) ≡ O(1).**

int TK(Apontador aux);Possui 2 comandos “For” . O mais externo é nomeado pela variável **i** que vai de 0 a 4(máximo de cartas da mão)e possui uma comparação. Já o mais interno vai de i+1 até 4 e possui uma comparação. Portanto a função possui a mesma complexidade da função **FK ≡ O(1).**

int TP(Apontador aux); Possui 2 comandos “For” . O mais externo é nomeado pela variável **i** que vai de 0 a 4(máximo de cartas da mão)e possui uma comparação. Já o mais interno vai de i+1 até 4 e possui duas comparações no melhor caso e três comparações no pior caso.

Portanto a função realiza 4 + 3(4 +3 + 2 + 1) comparações no pior caso = **O(44) ≡ O(1).**

int OP(Apontador aux);Possui 2 comandos “For” . O mais externo é nomeado pela variável **i** que vai de 0 a 4(máximo de cartas da mão)e possui duas comparações. Já o mais interno vai de i+1 até 4 e possui uma comparação. Portanto a função realiza 2x4 + 4 +3 + 2 + 1 comparações = **O(18) ≡ O(1).**

int VerificaAs(Apontador aux); Para cada carta da mão do jogador realiza uma comparação. Como a mão é limitada a 5 cartas a função é **O(5) ≡ O(1).**

void devolvedinheiro(TipoLista *l); Para **p** jogadores que apostaram na rodada , a função percorre os **l** jogadores da lista e faz a atribuição para os **p jogadores** o retorno do dinheiro. Logo a função é **O(l).**

void freeall(TipoLista *l); Percorre os **l** jogadores da lista e desaloca os seus respectivos ponteiros utilizados na estrutura TipoJogador. Logo a função faz **l** desalocações, sendo assim **O(l).**

4.Testes:

Testes foram realizados com o programa de forma a verificar o seu funcionamento. Os testes foram realizados em um AMD X2, com 4 Gb de memória. A figura abaixo mostra a saída escrita em um arquivo de uma execução para a seguinte entrada especificada:

Exemplo de entrada
3 1000
5 50
Giovanni 50 6O 3P 10E 11O 1C
John Holiver 200 3C 4E 3E 13P 13O
Thiago 100 12O 7P 12C 1O 13C
Gisele 300 12E 10C 11C 9C 13E
Clodoveu 50 5P 12P 5E 2E 1P
2 50
Clodoveu 250 2P 13E 9E 12C 2O
Gisele 250 11P 9P 2E 6E 4P
3 100
Thiago 250 1O 4P 1E 3O 8O
Gisele 250 9C 8C 3C 2C 6C
Giovanni 250 1P 1C 12E 12O 2P

```
1 1 950 S
2 Gisele
3 1 750 OP
4 Clodoveu
5 1 1250 F
6 Gisele
7 ########
8 Gisele 2200
9 Clodoveu 1250
10 John Holiver 600
11 Giovanni 500
12 Thiago 450
13
```

5.Conclusão:

A implementação do trabalho transcorreu sem maiores problemas e os resultados ficaram dentro do esperado. Entretanto, a maior dificuldade encontrada foi o número de comparações feitas para determinar uma certa jogada de um jogador.

Referências
[1] Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 2a Edição, Editora Thomson, 2004.
Curso de linguagem C – UFMG – disponível em http://mico.ead.cpdee.ufmg.br/cursos/C/index.html

Anexos:
• lista.h
• lista.c
• main.c