

Trabalho Prático 1

Jogo Tabuleiro

Aluno: Enzo Roiz

Professores: Gisele e Clodoveu.

Descrição do problema

O objetivo deste trabalho é simular um jogo de tabuleiro e a interação de um ou mais jogadores sobre ele. O tabuleiro é composto por casas, identificadas por códigos únicos, e cada casa pode ter uma regra de navegação específica. As regras de navegação indicam a ação que um jogador deve tomar ao cair numa determinada casa do tabuleiro, obedecendo as regras do jogo.

Implementação

Estrutura de dados:

Para a implementação do trabalho foi utilizada o tipo abstrato de dados “listas encadeadas” com a seguinte estrutura.

```
typedef struct Celula_str *Apontador;
typedef struct {
    char casa[TAM_BUFFER];
    char proximaCasa[TAM_BUFFER];
    char regraCasa[TAM_BUFFER];
    long long int energia;
    char jogador[TAM_BUFFER];
    long long int numDado;
    Apontador apontadorCasaAtual;
    Apontador enderecoCelula;
    Apontador apontadorRegraCasa;
    Apontador apontadorProximaCasa;
    int status;
    int ganhador;
} Tipoltem;
```

```
typedef struct Celula_str {
    Tipoltem Item;
    Apontador Prox;
} Celula;
```

```
typedef struct {
    Apontador Primeiro, Ultimo;
} TipoLista;
```

TipoLista: Contém dois apontadores, um para a célula cabeça e outro para o último item da lista.

TipoCelula: Contém um apontador para a célula seguinte ou para NULL caso seja a última célula e um Item do tipo Tipoltem.

Tipoltem: Onde são armazenadas as informações de cada célula da lista.

O **Tipoltem** implementado possui as seguintes informações:

casa: Campo texto de tamanho definido por TAM_BUFFER indicando o nome da casa no tabuleiro.

proximaCasa: Campo texto de tamanho definido por TAM_BUFFER indicando o nome da proximaCasa no tabuleiro.

regraCasa: Campo texto de tamanho definido por TAM_BUFFER indicando a regra de navegação da casa no tabuleiro.

energia: Armazena a energia do jogador e a energia proveniente da casa no tabuleiro.

jogador: Campo texto de tamanho definido por TAM_BUFFER indicando o nome do jogador.

numDado: Armazena o número do dado que o jogador tirou.

apontadorCasaAtual: Guarda o endereço da casa no tabuleiro onde o jogador está.

enderecoCelula: Guarda o apontador para a célula onde o item está.

apontadorRegraCasa: Guarda o apontador para a célula para onde o jogador será redirecionado, caso regraCasa seja zero, este apontador, aponta para a própria célula onde o item está.

apontadorProximaCasa: Guarda o apontador para a célula seguinte. Caso a célula seja a última, aponta para NULL.

status: Guarda o status do jogador, vivo ou morto.

ganhador: Se o jogador ganhar o jogo, esta variável é setada para 1.

Para a implementação foi utilizada três listas diferentes: jogadores contendo as informações de cada jogador, tabuleiro, jogadas e jogadores.

As variáveis utilizadas por cada lista foram:

Tabuleiro:

casa

proximaCasa

regraCasa

energia

enderecoCelula

apontadorProximaCasa

apontadorRegraCasa

Jogadas:

jogador
numDado

Jogadores:

energia
jogador
apontadorCasaAtual
status
ganhador

Funções

void PegaArgumentos(int , char **, char **, char **):

Recebe como parâmetros argc: número de argumentos com os quais a função main foi chamada na comando. **argv endereço da matriz de strings onde estão armazenados o nome do programa e os argumentos passados pela linha de comando. ** entrada: endereço da string com o nome do arquivo de entrada. **saída: endereço da string com o nome do arquivo de saída. Nesta função, entrada receberá o nome do arquivo de entrada passado pela linha de comando e saída receberá o nome do arquivo de saída passado pela linha de comando.

void FLVazia(TipoLista *);

Recebe como parâmetro um ponteiro para uma lista. Nesta função, é criada a lista. É feita uma célula cabeça para facilitar as operações na lista com os apontadores “Primeiro” e “Ultimo” apontando para ela própria.

int Vazia(TipoLista);

Recebe como parâmetro uma lista. Verifica se a lista está vazia e retorna non-zero caso esteja vazia e zero caso não esteja vazia.

void Insere(TipoItem, TipoLista *);

Recebe como parâmetros o item a ser inserido na lista e um ponteiro para uma lista. Cria uma célula nova na lista, inserindo nela o item passado por parâmetro e, nesse caso como é inserida na última posição da lista, o apontador “Ultimo” da lista, passa a apontar para a nova célula inserida.

void Imprime(char *, TipoLista);

Recebe como parâmetros um ponteiro para string contendo o nome do arquivo de saída onde serão impressos os resultados e a lista de jogadores. Esta função imprime os pontos vida de cada jogador, caso alguém ganhe imprime Ganhei (nome jogador) e caso alguém morra imprime Morri (nome jogador).

void PegaRegraCasa(TipoLista *);

Recebe como parâmetro um ponteiro para a lista tabuleiro. Nesta função, para cada célula da lista, a lista é percorrida. Caso a regra da casa seja igual à casa da célula apontada ao se percorrer a lista, a variável apontadorRegraCasa recebe o endereço da casa apontada ao se percorrer a lista. Caso a regra da casa seja zero, apontadorRegraCasa recebe o endereço da própria célula.

Apontador CriaJogador(TipoLista *, Apontador, Apontador);

Recebe como parâmetros um ponteiro para a lista jogadores, um apontador para a célula onde está armazenada a jogada que está sendo processada, e um apontador para a primeira casa do tabuleiro. A cada jogada processada, esta função é chamada, caso o jogador não exista, o jogador é criado. A função retorna um apontador para a célula do jogador que está realizando a jogada.

TipoLista ExecutaJogadas(TipoLista *, TipoLista *);

Recebe como parâmetros um ponteiro para a lista jogadas e um ponteiro para a lista tabuleiro. Esta função executa as jogadas passadas no arquivo de entrada. A cada jogada executada, ela chama a função CriaJogador(), criando um novo jogador caso ele não exista e aplicando as regras do jogo sobre as jogadas. A função retorna a lista de jogadores que participaram do jogo.

void PegaProximaCasa(TipoLista *);

Recebe como parâmetro um ponteiro para a lista tabuleiro. Nesta função, para cada célula da lista, a lista é percorrida. Caso a próxima casa seja igual à casa da célula apontada ao se percorrer a lista, a variável apontadorProximaCasa recebe o endereço da casa apontada ao se percorrer a lista.

void FazTabuleiro(char *, TipoLista *, TipoLista *);

Recebe como parâmetro a string contendo o nome do arquivo de entrada, um ponteiro para lista tabuleiro e um ponteiro para a lista jogadas. Nesta função é lido o arquivo de entrada passado por linha de comando e então são construídas as listas jogadas e tabuleiro.

Programa principal

Cria dois ponteiros para char, entrada e saída, ambos apontando para NULL. Declara as listas tabuleiro e jogadas e inicializa as listas por meio da função FLVazia(). Faz a chamada da função PegaArgumentos(), para receber os argumentos passados por linha de comando. Chama a função FazTabuleiro(), para ler o arquivo de entrada e criar as listas tabuleiros e jogadas. Chama as funções PegaRegraCasa() e PegaProximaCasa(), para pegar os apontadores para proxima casa e regra da casa de cada célula do tabuleiro. Chama a função Imprime() passando como um dos argumentos o retorno da função ExecutaJogadas(), a função Imprime() escreverá os resultados do jogo no arquivo de saída.

Análise de complexidade

A complexidade utilizará da notação $O(\text{Big-O})$ e será medida abordando a entrada de três diferentes formas.

Número de casas do tabuleiro: n .

Número de jogadas: m .

Número de jogadores: k .

void PegaArgumentos(int , char **, char **, char **);

A complexidade da função é $O(1)$ já que a função independe das entradas descritas.

void FLVazia(TipoLista *);

A complexidade da função é $O(1)$ já que a função independe das entradas descritas.

int Vazia(TipoLista);

A complexidade da função é $O(1)$ já que a função independe das entradas descritas.

void Insere(TipoItem, TipoLista *);

A complexidade da função é $O(1)$ já que a função independe das entradas descritas. Pois como estamos inserindo na última posição da lista não é necessário percorrê-la, apenas utilizar o apontador Lista->Ultimo para ter acesso à última posição.

void Imprime(char *, TipoLista);

Na função Imprime(), teremos três laços while independentes, ou seja, não estão um dentro do outro, que percorrem a lista de jogadores. A complexidade da função então será:

$$O(k)+O(k)+O(k) = 3O(k) = O(k).$$

void PegaRegraCasa(TipoLista *);

Na função PegaRegraCasa(), temos dois laços while aninhados. A função, é usada para andar cada célula, e a compará-la com todas as células da lista tabuleiro. No pior caso, a regra da casa de todas as células seria a última célula inserida na lista, assim, a lista seria percorrida todas as vezes em sua totalidade. A complexidade da função é portanto:

$$n(O(n)) = O(n^2)$$

Apontador CriaJogador(TipoLista *, Apontador, Apontador);

Na função CriaJogador() temos um laço while para verificar, na lista de jogadores, se o jogador que está executando a jogada existe. Para verificar que o jogador não existe a lista é percorrida em sua totalidade, ou seja sua complexidade será:

$$O(k)$$

TipoLista ExecutaJogadas(TipoLista *, TipoLista *);

Nesta função existe um laço while para que todas as jogadas da lista jogadas sejam executadas. Mas como a função chama a função CriaJogador() a cada execução de jogadas e a complexidade de CriaJogador() é $O(k)$, teremos no pior caso:

$$O(k)O(m) = O(k*m)$$

void PegaProximaCasa(TipoLista *);

Na função PegaProximaCasa(), temos dois laços while aninhados. A função, é usada para andar cada célula, e a compará-la com todas as células da lista tabuleiro. No pior caso, proxima casa de todas as celulas seria a última célula inserida na lista, assim, a lista seria percorrida todas as vezes em sua totalidade. A complexidade da função é portanto:

$$n(O(n)) = O(n^2)$$

void FazTabuleiro(char *, TipoLista *, TipoLista *);

Na função FazTabuleiro(), temos dois laços while independentes, ou seja, não estão um dentro do outro. Em um while lê-se todas as casas do tabuleiro, no outro lê-se todas as jogadas que serão realizadas pelo programa. Portanto a complexidade será:

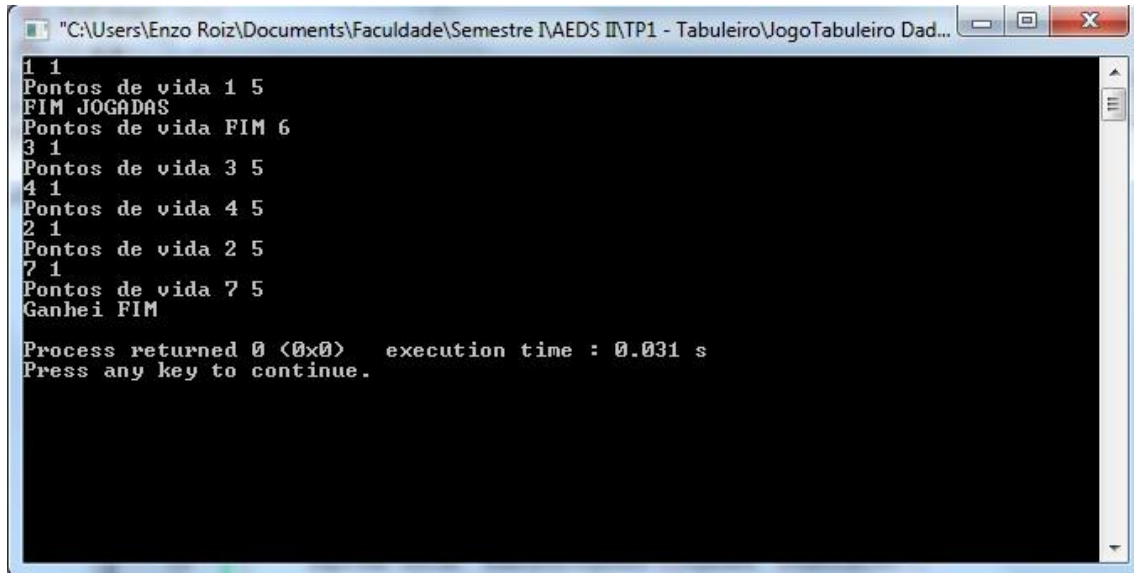
$$O(\max(n),(m))$$

Assim a complexidade do programa será representada pela função de maior ordem de complexidade, mas como temos três funções com ordem de complexidade semelhantes, PegaRegraCasa(), PegaProximaCasa() e ExecutaJogadas(), porém a complexidade de PegaRegraCasa() e PegaProximaCasa() é a mesma, n^2 , a complexidade do programa será dada então por:

$$O(\max(n^2),(k*m))$$

Testes

Para a validação da solução alguns testes foram feitos baseados em testes de outros colegas fornecidos no moodle.



```
"C:\Users\Enzo Roiz\Documents\Faculdade\Semestre I\AEDS II\TP1 - Tabuleiro\JogoTabuleiro Dad..."
1 1
Pontos de vida 1 5
FIM JOGADAS
Pontos de vida FIM 6
3 1
Pontos de vida 3 5
4 1
Pontos de vida 4 5
2 1
Pontos de vida 2 5
7 1
Pontos de vida 7 5
Ganhei FIM
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Saída para a entrada:

TABULEIRO

1 2 0 -3

3 4 0 0

2 3 4 -3

4 7 2 -2

333 0 0 0

7 333 0 5

JOGADAS

1 1

2 2

3 4

1 2

2 2

1 3

2 1

1 1

FIM

[illegible]

Conclusão

O trabalho foi interessante e aprimorou e fez que utilizasse na prática os meus conhecimentos sobre estruturas de dados, no caso as listas encadeadas. A implementação ocorreu em 2 etapas, onde na primeira foi implementada considerando o tabuleiro como sendo linear, ou seja, as casas só poderiam ser acessadas via “dado”. Porém este tipo de implementação não atenderia a casas que poderiam ser acessadas via “regra da casa”. Então a parte de execução de jogadas teve de ser refeita para atender tal padrão. A dificuldade maior encontrada foi encontrar erros e corrigir o TP considerado “pronto” para passar no maior número de testes. A saída foi quase como esperada, uma vez que falhei em dois dos 23 testes, 9 e 15 (contando de 0).

Referências:

- Slides passados em sala de aula.
- Curso de Linguagem C online: <http://www.mtm.ufsc.br/~azeredo/cursoC/>

Arquivos:

O TP é composto de três arquivos:

- main.c
- JogoTabuleiro.c
- JogoTabuleiro.h.