Universidade Presbiteriana Mackenzie

- Faculdade de Computação e Informática –

Ciência da Computação

Projeto e Análise de Algoritmo II

Participantes:

Nome	RA
Thomaz de Souza Scopel	10417183
Vinícius Sanches Cappatti	10418266

Listagem de código fonte:

- **Main.c**

```
/*Bibliotecas utilizadas*/
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <locale.h>

/*Arquivo com as funcoes implementadas para o jogo de resta um*/
#include "Funcoes.h"

int **tabuleiro;
int **tabuleiro2;
Movimento **jogadas;

int main(int argc, char **argv){
   setlocale(LC_ALL, "pt_BR.UTF-8");

   tabuleiro = inicializaTabuleiro(); // Alocacao da memoria dinamica do tabuleiro
   if(tabuleiro == NULL){
      return 1;
   }

   // Chama a função para ler o tabuleiro do arquivo e armazená-lo na matriz
   if (argc > 1) {
      lerTabuleiro(argv[1]);
   }
   else {
      lerTabuleiro("entrada.txt");
   }

   tabuleiro2 = copiaTabuleiro(tabuleiro, TAMANHO); // tabuleiro2 sera utilizado posteriormente para imprimir a saida

   jogadas = (Movimento**) malloc(NUM_INI_PECAS * sizeof(Movimento*)); // Lista contendo os movimentos feitos
para resolver o jogo

   /*Chamada do backtracking*/

   printf("*** JOGO DE RESTA UM ***\nA execução pode demorar um pouco.\n");

   jogaRestaUm(NUM_INI_PECAS);

   if(jogadas == NULL){
      printf("Erro na criação das jogadas.\n");
      return 1;
   }
```

```
    printf("Resultado encontrado!\nPassos para se resolver o resta um:\n");

    printJogadas();

    /*Imprimir as jogadas no arquivo de saida*/

    imprimeSaida("restaum.out");

    printf("Resultado salvo em restaum.out\n");

    // Libera a memória alocada dinamicamente antes de finalizar o programa
    for(int i = 0; i < cont; i++){ free(jogadas[i]); }
    free(jogadas);

    for (int i = 0; i < TAMANHO; i++){ free(tabuleiro[i]); }
    free(tabuleiro);

    for (int i = 0; i < TAMANHO; i++){ free(tabuleiro2[i]); }
    free(tabuleiro2);

    return 0;
}
```

- **Funcoes.h**

```
#include <stdbool.h>
#include <stdlib.h>

#ifndef FUNCOES_H
#define FUNCOES_H

/*Constantes do programa*/

#define TAMANHO 7
#define CENTRO 3
#define NUM_INI_PECAS 32
#define MAX_LINHA 100

/* ****************************************************************
* Struct Movimento representa um movimento com as caracteristicas:
* x0: linha de partida
* y0: coluna de partida
* xf: linha final
* yf: coluna final
**************************************************************** */
typedef struct movimento{
    int x0;
    int y0;
    int xf;
    int yf;
} Movimento;

/*Variaveis globais*/

/* ************************************************
* Matriz que contem um tabuleiro de Resta um
* onde:
* -1 representa uma posicao onde nao ha casa ('#')
```

```
* 0 representa uma casa vazia (' ')
* 1 representa uma casa ocupada ('o')
************************************************** */
extern int **tabuleiro;

//Copia da matriz tabuleiro que eh usada na saida do programa
extern int **tabuleiro2;

//Historico de jogadas feitas para se resolver o jogo de Resta Um
extern Movimento** jogadas;

//Contador utilizado no backtracking
static int cont = 0;

/*Funcoes do jogo Resta Um*/

int** inicializaTabuleiro();
void lerTabuleiro(char *nomeArquivo);
int** copiaTabuleiro();
void removeNovaLinha(char *linha);
bool movimentoValido(int x0, int y0, int xf, int yf);
Movimento movimenta(int **matriz, int x0, int y0, int xf, int yf);
void salvaMovimento(Movimento mov);
void desfazMovimento(int x0, int y0, int xf, int yf);
bool iteraBacktracking(int qtdPecas, int x0, int y0, char direcao);
int defineXf(int x, char direcao);
int defineYf(int y, char direcao);
bool jogaRestaUm(int qtdPecas);
void limpaMovimento(Movimento *mov);
void printMov(Movimento *mov);
void imprimeSaida(char *nomeArquivo);
void printJogadas();

#endif
```

- **Funcoes.c**

```
/*Funcao que le o arquivo de entrada e o converte em uma matriz*/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#include "Funcoes.h"

// A funcao remove linha evita o erro do fgets substituindo o \n por terminador nulo
void removeNovaLinha(char *linha) {
    char *pos;
    if ((pos = strchr(linha, '\n')) != NULL) {
        *pos = '\0';
    }
}

// A funcao inicializaTabuleiro aloca o espaco para ele e preenche com zeros
int** inicializaTabuleiro(){
    bool sucesso = true;

    int **tabuleiro = (int **) malloc (TAMANHO * sizeof(int *));
```

```
    if (tabuleiro == NULL){
        fprintf(stderr, "Erro ao alocar memória para o tabuleiro.\n");
        sucesso = false; // Ou outra forma de lidar com o erro
    }

    /*Alocacao da memoria para as linhas da matriz*/
    for (int i = 0; i < TAMANHO; i++){
        tabuleiro[i] = (int *)malloc(TAMANHO * sizeof(int));
        if (tabuleiro[i] == NULL){
            fprintf(stderr, "Erro ao alocar memória para a linha %d do tabuleiro.\n", i);
            // Libere a memória já alocada antes de retornar
            for (int k = 0; k < i; k++){
                free(tabuleiro[k]);
            }
            free(tabuleiro);
            sucesso = false;
        }

        // Inicializa a linha
        for (int j = 0; j < TAMANHO; j++){
            tabuleiro[i][j] = 0;
        }
    }

    return tabuleiro;
}

// A funcao copiaTabuleiro() passa os valores do tabuleiro1 para o tabuleiro2
int** copiaTabuleiro(){
    int** tabuleiro2 = (int**) malloc(TAMANHO * sizeof(int*));

    for(int x = 0; x < TAMANHO; x++){
        tabuleiro2[x] = (int*) malloc(TAMANHO * sizeof(int));
    }

    for(int i = 0; i < TAMANHO; i++){
        for(int j = 0; j < TAMANHO; j++){
            tabuleiro2[i][j] = tabuleiro[i][j];
        }
    }

    return tabuleiro2;
}

/* Função que le o arquivo de entrada e o converte em uma matriz de inteiros salva na variavel tabuleiro*/
void lerTabuleiro(char *nomeArquivo) {
    FILE *arquivo = fopen(nomeArquivo, "r");  // Abre o arquivo em modo de leitura
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo.\n");
        return;
    }

    char linha[MAX_LINHA];  // Ajusta para possível '\n' e '\0'

    // Percorre o arquivo, lendo linha por linha
    for (int i = 0; i < TAMANHO + 2; i++) {  // TAMANHO + 2 devido à margem (#)
        fgets(linha, sizeof(linha), arquivo);  // Lê uma linha do arquivo
        removeNovaLinha(linha);  // Remove a nova linha (\n) se estiver presente

        if (i == 0 || i == TAMANHO + 1) {
```

```
            // Ignora a primeira e a última linha (margens de #)
            continue;
        }

        // Processa apenas as linhas internas (sem a margem)
        for (int j = 1; j <= TAMANHO; j++) {  // Começa em 1 e vai até TAMANHO
            char caractere = linha[j];  // Pega o caractere correspondente

            // Converte os caracteres para os valores da matriz
            if (caractere == '#') {
                tabuleiro[i - 1][j - 1] = -1;  // Margem ou casa invalida
            } else if (caractere == 'o') {
                tabuleiro[i - 1][j - 1] = 1;   // Espaço ocupado
            } else if (caractere == ' ') {
                tabuleiro[i - 1][j - 1] = 0;   // Espaço vazio
            }
        }
    }

    fclose(arquivo);  // Fecha o arquivo
}

/*Metodo retorna se a peca da posicao tabuleiro[x0][y0] pode ser movida para tabuleiro[xf][yf]*/
bool movimentoValido(int x0, int y0, int xf, int yf){

            if(tabuleiro[x0][y0] != 1){ return false; }

            if(xf < 0 || xf >= TAMANHO || yf < 0 || yf >= TAMANHO){ return false; } // Fora dos limites da matriz

            if(tabuleiro[xf][yf] != 0){ return false; }

            int xmedia = (x0 + xf) / 2;
            int ymedia = (y0 + yf) / 2;

            if(tabuleiro[xmedia][ymedia] != 1){ return false; }

            return true;
}

/*Funcao que recebe uma coordenada inicial (x0, y0) e final (xf, yf) e altera os valores da matriz tabuleiro para
simular um movimento de resta um, de forma que a peca sai da posicao inicial para a final 'comendo' a peca
que estava na casa do meio e retorna qual o movimento resultante*/
Movimento movimenta(int **matriz, int x0, int y0, int xf, int yf) {
    matriz[x0][y0] = 0; /*Posicao inicial do movimento*/
    matriz[xf][yf] = 1; /*Posicao final do movimento*/

    int xmedia = (x0 + xf) / 2;
    int ymedia = (y0 + yf) / 2;

    /*Representa a casa onde esta a peca que sai do tabuleiro*/
    matriz[xmedia][ymedia] = 0;

    Movimento mov;

    mov.x0 = x0;
    mov.y0 = y0;
    mov.xf = xf;
    mov.yf = yf;

    return mov;
```

```
}

/*Funcao que armazena o movimento 'mov' na ultima posicao de 'jogadas'*/
void salvaMovimento(Movimento mov){
   jogadas[cont] = (Movimento*) malloc(sizeof(Movimento));
   *jogadas[cont] = mov;
   cont++;
}

/*Funcao que recebe uma coordenada inicial (x0, y0) e final (xf, yf) e altera os valores da matriz tabuleiro para
simular a exclusao de um movimento de resta um, de forma que a posicao inicial e do meio recebem uma peca cada
enquanto a final eh esvaziada*/
void desfazMovimento(int x0, int y0, int xf, int yf){
   tabuleiro[x0][y0] = 1; /*Retorna a peca para a posicao inicial*/
   tabuleiro[xf][yf] = 0; /*Esvazia a casa da posicao final*/

   int xmedia = (x0 + xf) / 2;
   int ymedia = (y0 + yf) / 2;

   /*Insere de volta no tabuleiro a peca que havia sido retirada*/
   tabuleiro[xmedia][ymedia] = 1;
}

/*Funcao que retorna qual a linha final de um movimento para determinada direcao*/
int defineXf(int x, char direcao){
   switch (direcao){
      case 'c': // Movimento para cima
         return x - 2;
      case 'b': // Movimento para baixo
         return x + 2;
      case 'd': // Movimento para a direita
         return x;
      case 'e': // Movimento para a esquerda
         return x;
      default:
         return -1;
   }
}

/*Funcao que retorna qual a coluna final de um movimento para determinada direcao*/
int defineYf(int y, char direcao){
   switch (direcao){
      case 'c': // Movimento para cima
         return y;
      case 'b': // Movimento para baixo
         return y;
      case 'd': // Movimento para a direita
         return y + 2;
      case 'e': // Movimento para a esquerda
         return y - 2;
      default:
         return -1;
   }
}

/*Metodo com backtracking do resta um*/
bool jogaRestaUm(int qtdPecas){

   /*Condicao de parada: quando se ha somente uma peca no tabuleiro e ela se localiza no centro dele*/
   if(qtdPecas == 1 && tabuleiro[CENTRO][CENTRO] == 1){
```

```
      return true;
   }

   for(int x = 0; x < TAMANHO; x++){
      for(int y = 0; y < TAMANHO; y++){
         if(tabuleiro[x][y] == 1){ // Somente itera backtracking se encontrar uma casa ocupada
            if(iteraBacktracking(qtdPecas, x, y, 'b')){ // Testa movimento para baixo
               return true;
            }
            if(iteraBacktracking(qtdPecas, x, y, 'c')){ // Testa movimento para cima
               return true;
            }
            if(iteraBacktracking(qtdPecas, x, y, 'd')){ // Testa movimento para a direita
               return true;
            }
            if(iteraBacktracking(qtdPecas, x, y, 'e')){ // Testa movimento para a esquerda
               return true;
            }
         }
      }
   }

   return false;
}

/*Funcao que itera o backtracking da funcao 'jogaRestaUm'.*/
bool iteraBacktracking(int qtdPecas, int x0, int y0, char direcao){

   int meio = TAMANHO / 2;

   int xf = defineXf(x0, direcao);
   int yf = defineYf(y0, direcao);

   /*Somente itera o backtracking se o movimento for valido*/
   if(movimentoValido(x0, y0, xf, yf)){

      Movimento mov = movimenta(tabuleiro, x0, y0, xf, yf); // Altera o tabuleiro e retorna o movimento feito

      // Salva o ultimo movimento em 'jogadas'
      salvaMovimento(mov);

      if(jogaRestaUm(qtdPecas - 1)){ // Chama novamente o jogaRestaUm com a quantidade de pecas atualizada
         return true; // Retorna true se a proxima iteracao chegar ao caso base
      }

      limpaMovimento(jogadas[cont - 1]); /*Retira o movimento de 'jogadas'*/
      cont--;
      desfazMovimento(x0, y0, xf, yf); /*Retorna o tabuleiro para a posicao anterior ao movimento*/
   }

   return false; // Se o movimento nao for valido ou se nao conseguir chegar a um caso base, retorna false
}

// Funcao que exclui uma instancia da struct Movimento da memoria
void limpaMovimento(Movimento *mov){
   mov->x0 = 0;
   mov->y0 = 0;
   mov->xf = 0;
   mov->yf = 0;
}
```

Universidade Presbiteriana Mackenzie

- Faculdade de Computação e Informática –

Ciência da Computação

Projeto e Análise de Algoritmo II

```
// Funcao que imprime as coordenadas iniciais e finais de um movimento
void printMov(Movimento *mov){
   printf("(%d, %d) -> (%d, %d)\n", mov->x0, mov->y0, mov->xf, mov->yf);
}

// Funcao que preenche o arquivo de saida com todos os estados do tabuleiro durante o decorrer do jogo
void imprimeSaida(char *nomeArquivo){
   FILE *saida;

   saida = fopen(nomeArquivo, "w");

   if(saida == NULL){
      printf("Erro ao abrir o arquivo de saída.");
      return;
   }

   for(int i = 0; i <= cont; i++){

      fprintf(saida, "#########\n");
      for(int x = 0; x < TAMANHO; x++){
         fprintf(saida, "#");
         for(int y = 0; y < TAMANHO; y++){
            if(tabuleiro2[x][y] == -1){
               fprintf(saida, "#");
            } else if(tabuleiro2[x][y] == 0){
               fprintf(saida, " ");
            } else if(tabuleiro2[x][y] == 1){
               fprintf(saida, "o");
            }
         }
         fprintf(saida, "#\n");
      }
      fprintf(saida, "#########\n\n");

      if (i == cont) break; // Evita o erro de out of bounds, ja que percorrendo 'jogadas' estamos a i+1 na frente da escrita
do arquivo

      // Agora com o tabuleiro reiniciado fazemos os movimentos nele e vamos escrevendo no arquivo de saida
      movimenta(tabuleiro2, jogadas[i]->x0, jogadas[i]->y0, jogadas[i]->xf, jogadas[i]->yf);
   }

   fclose(saida);
}

// Funcao que imprime todas as jogadas necessarias para se chegar na solucao do jogo
void printJogadas(){
   for(int i = 0; i < cont; i++){
      printMov(jogadas[i]);
   }
}
```