Autores: Lucas Garcia e Luis Augusto

Arquivo `.c` (implementação)

1. Função `abrirArquivo()`

- **Objetivo**: Abrir um arquivo para leitura ou escrita. Se o arquivo não puder ser aberto, o programa exibe uma mensagem de erro e é finalizado.

```c
FILE *abrirArquivo(char *nomeArq, char *modo) {
    FILE *arq = fopen(nomeArq, modo);
    if (arq == NULL) {
        printf("ERRO ao abrir o arquivo.\n");
        exit(-1);
    }
    printf("INFO: Arquivo Aberto! Bom uso.\n");
    return arq;
}
```

- **Explicação**: Esta função utiliza `fopen` para abrir um arquivo no modo especificado (leitura ou escrita). Caso o arquivo não seja encontrado, o programa imprime uma mensagem de erro e encerra.

2. Função `calcularTempo()`

- **Objetivo**: Calcular o tempo de execução de uma operação usando a função `clock_t` para medir o tempo.

```c
void calcularTempo(double ini, double fim) {
    double tempoDecorrido = (double)(fim - ini) / CLOCKS_PER_SEC;
    printf("Tempo de execucao: %f segundos\n", tempoDecorrido);
}
```

- **Explicação**: O tempo de execução de uma operação é calculado e impresso em segundos. É útil para medir o desempenho do programa.

3. Função `salvarDadosNoArquivo()`

- **Objetivo**: Salvar os dados da árvore binária em um arquivo de texto.

```c
void salvarDadosNoArquivo(ArvoreBinaria *arvore, FILE *arquivoLista) {
    for (int i = 0; i < arvore->capacidade; i++) {
        if (arvore->elementos[i].ocupado) {
            fprintf(arquivoLista, "%s\n%lld\n", arvore->elementos[i].nome, arvore->elementos[i].matricula);
        }
    }
    printf("INFO: Dados salvos com sucesso no arquivo.\n");
}
```

- **Explicação**: A função percorre todos os nós da árvore e, para os nós ocupados, grava o nome e a matrícula no arquivo.

4. Função `inicializarArvore()`

- **Objetivo**: Inicializar uma árvore binária com uma capacidade baseada no número de matrículas esperado, multiplicado por um fator de segurança.

```c
void inicializarArvore(ArvoreBinaria *arvore, int quantidadeMatriculas) {
    arvore->capacidade = (int)(quantidadeMatriculas * FATOR_SEGURANCA);
    arvore->tamanho = 0;
    arvore->elementos = (NoArvore *)malloc(arvore->capacidade * sizeof(NoArvore));

    if (arvore->elementos == NULL) {
        printf("Erro de alocação de memória.\n");
        exit(1);
    }

    for (int i = 0; i < arvore->capacidade; i++) {
        arvore->elementos[i].ocupado = 0; // Inicializa todos os nós como vazios
    }
}
```

- **Explicação**: A função aloca dinamicamente memória para a árvore, inicializando todos os nós como vazios. Usa um fator de segurança para alocar mais memória do que o necessário, prevenindo futuras alocações.

5. Função `redimensionarArvore()`

- **Objetivo**: Aumentar a capacidade da árvore binária quando necessário.

```c
void redimensionarArvore(ArvoreBinaria *arvore) {
    int novaCapacidade = arvore->capacidade * FATOR_SEGURANCA;
    NoArvore *novoArray = (NoArvore *)realloc(arvore->elementos, novaCapacidade * sizeof(NoArvore));

    if (novoArray == NULL) {
        printf("Erro ao redimensionar a árvore.\n");
        exit(1);
    }

    // Atualizar ponteiros e capacidade
    arvore->elementos = novoArray;
    arvore->capacidade = novaCapacidade;

    // Inicializa os novos espaços alocados como vazios
    for (int i = arvore->tamanho; i < arvore->capacidade; i++) {
        arvore->elementos[i].ocupado = 0;
    }

    printf("Arvore redimensionada para %d elementos.\n", arvore->capacidade);
}
```

- **Explicação**: Se a árvore atingir sua capacidade máxima, esta função aumenta sua capacidade em 50%. O conteúdo da árvore original é mantido, e novos nós são inicializados como vazios.

6. Função `imprimirEmOrdem()`

- **Objetivo**: Exibir a árvore binária em ordem (ordem crescente de matrículas).

```c
void imprimirEmOrdem(ArvoreBinaria *arvore, int indice) {
    if (indice >= arvore->capacidade || !arvore->elementos[indice].ocupado) {
        return;
    }

    // Percorrer o filho esquerdo
    imprimirEmOrdem(arvore, 2 * indice + 1);

    // Imprimir o nó atual
    printf("Matrícula: %lld, Nome: %s\n", arvore->elementos[indice].matricula, arvore->elementos[indice].nome);

    // Percorrer o filho direito
    imprimirEmOrdem(arvore, 2 * indice + 2);
}
```

- **Explicação**: A árvore é percorrida recursivamente em ordem, começando pelo filho esquerdo, depois o nó raiz, e por último o filho direito, imprimindo os nós ocupados.

Função `inserirAluno()`

- **Objetivo**: Inserir um aluno na árvore binária de acordo com a matrícula.

```c
void inserirAluno(ArvoreBinaria *arvore, long long int matricula, char *nome) {
    if (arvore->tamanho >= arvore->capacidade) {
        redimensionarArvore(arvore);
    }

    int i = 0;
    while (i < arvore->capacidade && arvore->elementos[i].ocupado) {
        if (matricula < arvore->elementos[i].matricula) {
            i = 2 * i + 1; // Vai para o filho esquerdo
        } else if (matricula > arvore->elementos[i].matricula) {
            i = 2 * i + 2; // Vai para o filho direito
        } else {
            printf("Matrícula já existente.\n");
            return;
        }
    }

    if (i < arvore->capacidade) {
        arvore->elementos[i].matricula = matricula;
        strcpy(arvore->elementos[i].nome, nome);
        arvore->elementos[i].ocupado = 1;
        arvore->tamanho++;
        printf("Aluno inserido com sucesso!\n");
    } else {
        printf("Erro ao inserir o aluno. Posição inválida.\n");
    }
}
```

- **Explicação**: A função insere um novo aluno na árvore. Se a árvore estiver cheia, é redimensionada. A inserção ocorre buscando a posição correta de acordo com a matrícula.

8. Função `buscarAluno()`

- **Objetivo**: Buscar um aluno na árvore usando a matrícula.

```c
void buscarAluno(ArvoreBinaria *arvore, long long int matricula) {
    int i = 0;
    while (i < arvore->capacidade && arvore->elementos[i].ocupado) {
        if (matricula < arvore->elementos[i].matricula) {
            i = 2 * i + 1; // Filho esquerdo
        } else if (matricula > arvore->elementos[i].matricula) {
            i = 2 * i + 2; // Filho direito
        } else {
            printf("Aluno encontrado: %s (Matrícula: %lld)\n", arvore->elementos[i].nome, arvore->elementos[i].matricula);
            return;
        }
    }
    printf("Aluno não encontrado.\n");
}
```

- **Explicação**: A função percorre a árvore de forma recursiva, buscando a matrícula. Se for encontrada, imprime as informações do aluno.

9. Função `removerAluno()`

- **Objetivo**: Remover um aluno da árvore binária com base na matrícula.

```c
void removerAluno(ArvoreBinaria *arvore, long long int matricula) {
    int i = 0;
    while (i < arvore->capacidade && arvore->elementos[i].ocupado) {
        if (matricula < arvore->elementos[i].matricula) {
            i = 2 * i + 1; // Filho esquerdo
        } else if (matricula > arvore->elementos[i].matricula) {
            i = 2 * i + 2; // Filho direito
        } else {
            // Encontrei o nó para remover
            printf("Removendo aluno: %s (Matrícula: %lld)\n", arvore->elementos[i].nome, arvore->elementos[i].matricula);
            arvore->elementos[i].ocupado = 0;
            arvore->tamanho--;
            return;
        }
    }
    printf("Matrícula não encontrada para remoção.\n");
}
```

- **Explicação**: A função busca o nó correspondente à matrícula e o marca como vazio, efetivamente removendo o aluno da árvore.

10. Função `pedirOpcao()`

- **Objetivo**: Exibir o menu principal e solicitar a opção escolhida pelo usuário.

```c
long long int pedirOpcao() {
    int op;
    printf("\n--- Menu Principal ---\n");
    do {
        printf("1 - Inserir na Arvore\n");
        printf("2 - Exibir a Arvore\n");
        printf("3 - Excluir da Arvore\n");
        printf("4 - Pesquisar na Arvore\n");
        printf("5 - Total de Matriculas\n");
        printf("6 - Percorrer toda a Arvore\n");
        printf("7 - Sair\n");
        printf("Digite a opção: ");
        scanf("%d", &op);
    } while ((op < 1) || (op > 7));
    return op;
}
```

- **Explicação**: Esta função exibe um menu de opções para o usuário e solicita uma escolha. O loop `do-while` garante que o programa continue solicitando uma opção válida até que o usuário insira um número entre 1 e 7.

11. Função `pedirNum()`

- **Objetivo**: Solicitar um número (matrícula) ao usuário, usado tanto para inserção quanto para exclusão.

```c
long long int pedirNum(int caminhoASerEscolhido) {
    long long int num;
    if (caminhoASerEscolhido == 0) {
        printf("Digite um numero para ser inserido: ");
        scanf("%lld", &num);
    } else {
        printf("Digite um numero para ser excluido: ");
        scanf("%lld", &num);
    }
    return num;
}
```

- **Explicação**: Dependendo do valor do parâmetro `caminhoASerEscolhido`, a função solicita um número ao usuário para inserção (caminho `0`) ou para exclusão (caminho `1`). O número é interpretado como a matrícula de um aluno.

12. Função `imprimirVetorCompleto()`

- **Objetivo**: Imprimir todo o conteúdo da árvore binária, incluindo os nós ocupados e vazios, em ordem de índices.

```c
void imprimirVetorCompleto(ArvoreBinaria *arvore) {
    printf("\n===| Impressão Completa do Vetor da Arvore |===\n\n");
    for (int i = 0; i < arvore->capacidade; i++) {
        if (arvore->elementos[i].ocupado) {
            if (i == 0) {
                printf("Índice %d: Matrícula: %lld, Nome: %s (Raiz)\n", i, arvore->elementos[i].matricula, arvore->elementos[i].nome);
            } else {
                int parentIndex = (i - 1) / 2;
                if (2 * parentIndex + 1 == i) {
                    printf("Índice %d: Matrícula: %lld, Nome: %s (Filho Esquerdo de %lld)\n", i, arvore->elementos[i].matricula, arvore->elementos[i].nome, arvore->elementos[parentIndex].matricula);
                } else {
                    printf("Índice %d: Matrícula: %lld, Nome: %s (Filho Direito de %lld)\n", i, arvore->elementos[i].matricula, arvore->elementos[i].nome, arvore->elementos[parentIndex].matricula);
                }
            }
        } else {
            printf("Índice %d: (vazio)\n", i);
        }
    }
    printf("\n=============================================\n");
}
```

- **Explicação**: A função imprime cada nó da árvore binária, informando a matrícula, o nome e se o nó é a raiz ou o filho esquerdo/direito de outro nó. Também imprime se o nó está vazio.

13. Função `menuPrincipal()`

- **Objetivo**: Gerenciar as operações de inserção, exclusão, pesquisa e exibição da árvore binária.

```c
void menuPrincipal(ArvoreBinaria *arvore) {
    long long int op;      You, há 2 semanas • ffddfd
    long long int numInseri;
    char nomeInseri[100];
    int repete = 0;
    do {
        op = pedirOpcao();
        switch (op) {
            case 1:
                // Inserir na Árvore Binária
                numInseri = pedirNum(0);
                printf("Digite o nome associado à matrícula: ");
                getchar();  // Para evitar problemas com a leitura do '\n' residual
                fgets(nomeInseri, sizeof(nomeInseri), stdin);
                nomeInseri[strcspn(nomeInseri, "\n")] = 0;  // Remover a quebra de linha
                inserirAluno(arvore, numInseri, nomeInseri);
                break;
            case 2:
                // Exibir a Árvore Binária em Ordem
                printf("\n\n===| Exibição da Árvore Binária (Em Ordem) |===\n\n");
                imprimirEmOrdem(arvore, 0);
                break;
            case 3:
                // Excluir da Árvore Binária
                numInseri = pedirNum(1);
                removerAluno(arvore, numInseri);
                break;
```

```c
            case 4:
                // Pesquisar uma matrícula na Árvore Binária
                numInseri = pedirNum(0);
                buscarAluno(arvore, numInseri);
                break;
            case 5:
                // Exibir total de matrículas na Árvore Binária
                printf("O total de matrículas na árvore é: %d\n", arvore->tamanho);
                break;
            case 6:
                // Imprimir Vetor completo
                imprimirVetorCompleto(arvore);
                break;
            case 7:
                // Sair
                repete = 1;
                break;
            default:
                printf("Opção inválida. Tente novamente.\n");
                break;
        }
    } while (repete == 0);
}
```

- **Explicação**: Esta função atua como o ponto central do programa, permitindo que o usuário escolha entre diversas operações na árvore binária, como inserção, exclusão, exibição e pesquisa.

14. Função `contarMatriculas()`

- **Objetivo**: Contar o número de matrículas presentes em um arquivo de texto.

```c
int contarMatriculas(FILE *arquivoLista) {
    char linha[100];
    int totalMatriculas = 0;
    while (fgets(linha, sizeof(linha), arquivoLista) != NULL) {
        totalMatriculas++;
    }
    return totalMatriculas / 2;  // Cada matrícula ocupa duas linhas (nome e matrícula)
}
```

- **Explicação**: A função conta o número de linhas em um arquivo e retorna o número de matrículas, assumindo que cada matrícula ocupa duas linhas (uma para o nome e outra para o número).

15. Função `lerEInserirMatriculas()`

- **Objetivo**: Ler as matrículas de um arquivo e inseri-las na árvore binária.

```c
void lerEInserirMatriculas(ArvoreBinaria *arvore, FILE *arquivoLista) {
    rewind(arquivoLista);  // Reposicionar para o início do arquivo
    long long int matricula;
    char nome[100];

    while (fgets(nome, sizeof(nome), arquivoLista) != NULL) {  // Ler o nome
        nome[strcspn(nome, "\n")] = 0;  // Remover o '\n' do nome
        if (fscanf(arquivoLista, "%lld\n", &matricula) != EOF) {  // Ler a matrícula
            inserirAluno(arvore, matricula, nome);  // Inserir na árvore binária
        }
    }
}
```

- **Explicação**: A função lê o nome e a matrícula do arquivo e insere esses dados na árvore binária. Ela percorre o arquivo até o fim, inserindo cada registro.

16. Função `iniciarCodigo()`

- **Objetivo**: Controlar o fluxo do programa, desde a inicialização até o encerramento.

```c
void iniciarCodigo(FILE *arquivoLista,ArvoreBinaria arvore){
    arquivoLista = abrirArquivo("nomes_matriculas.txt", "r");
    int totalMatriculas = contarMatriculas(arquivoLista);
    printf("Total de matrículas no arquivo: %d\n", totalMatriculas);
    rewind(arquivoLista); // Volta ao início do arquivo

    inicializarArvore(&arvore, totalMatriculas);
    double inicio = clock();    conversion from ‘clock_t’ {aka ‘long int’} to ‘doub
    // Ler as matrículas e inseri-las na árvore
    lerEInserirMatriculas(&arvore, arquivoLista);
    double fim = clock();    conversion from ‘clock_t’ {aka ‘long int’} to ‘double’
    fclose(arquivoLista);
    calcularTempo(inicio,fim);
    // Menu para interação
    menuPrincipal(&arvore);

    // Salvar os dados antes de encerrar
    arquivoLista = abrirArquivo("nomes_matriculas.txt", "w");
    salvarDadosNoArquivo(&arvore, arquivoLista);
    fclose(arquivoLista);

    // Liberar a memória
    liberarArvore(&arvore);

}
```

- **Explicação**: Esta função gerencia todo o processo de leitura dos dados, inicialização da árvore, inserção dos registros e o ciclo de interação com o usuário. Também cuida da gravação dos dados e da liberação da memória no final.

Arquivo `.h`

Este arquivo contém as definições de tipos, macros e as declarações das funções que serão implementadas no arquivo `.c`. Abaixo estão os detalhes de cada parte.

1. Bibliotecas Incluídas

- **Objetivo**: As bibliotecas incluídas fornecem as funcionalidades necessárias para a manipulação de strings, entrada e saída de dados, operações matemáticas, medição de tempo, entre outras.

```c
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
```

- **math.h**: Fornece funções matemáticas, como a função `pow` para cálculos de potência.
- **stdlib.h**: Inclui funções como `malloc` para alocação de memória dinâmica e `exit` para encerrar o programa.
- **stdio.h**: Inclui funções de entrada e saída, como `printf`, `scanf`, `fopen`, `fclose`, entre outras.
- **ctype.h**: Fornece funções para manipulação de caracteres, como `isdigit`.
- **string.h**: Inclui funções para manipulação de strings, como `strcpy` e `strcmp`.
- **time.h**: Fornece funções para manipulação de tempo, como `clock_t` e `clock` para medir o tempo de execução.

2. Definição de Tipos e Macros

a) Definição do tipo `string`

- **Objetivo**: Definir um tipo de dado `string` como um array de caracteres de tamanho 101

```
typedef char string[101];
```

Isso simplifica o uso de strings no programa, permitindo que seja referenciado como `string` ao invés de `char[101]`.

b) Definição do tipo `processTime`

- **Objetivo**: Definir um alias para o tipo `clock_t`, que é utilizado para medir o tempo de execução do programa.

```
typedef clock_t processTime;
```

c) Definição da macro `FATOR_SEGURANCA`

- **Objetivo**: Definir uma constante para o fator de segurança usado no cálculo da capacidade da árvore binária.

```
#define FATOR_SEGURANCA 100
```

Neste caso, o fator de segurança é 100 (ou 150%, de acordo com o comentário), para garantir que a capacidade da árvore seja dimensionada adequadamente.

3. Definição de Estruturas

a) Estrutura `NoArvore`

- **Objetivo**: Definir a estrutura de um nó na árvore binária. Cada nó armazena uma matrícula, o nome do aluno e se o nó está ocupado ou não.

```
typedef struct {
    long long int matricula; //Matrícula do aluno.
    string nome;             //Nome do aluno.
    int ocupado;             //Indica se o nó está ocupado (1) ou vazio (0).
} NoArvore;
```

- **`matricula`**: Um identificador único para o aluno.
- **`nome`**: Nome associado à matrícula do aluno.
- **`ocupado`**: Um valor `1` se o nó estiver ocupado ou `0` se estiver vazio.

b) Estrutura `ArvoreBinaria`

- **Objetivo**: Definir a estrutura da árvore binária, que é representada como um array de nós.

```
typedef struct {
    NoArvore *elementos; //Ponteiro para os nós da árvore.
    int tamanho;         //Número de elementos ocupados na árvore.
    int capacidade;      //Capacidade atual da árvore.
} ArvoreBinaria;
```

- **`elementos`**: Ponteiro para um array de nós (`NoArvore`), que representam os elementos da árvore.
- **`tamanho`**: Número de nós atualmente ocupados na árvore.
- **`capacidade`**: Capacidade total da árvore (quantidade máxima de nós que podem ser armazenados).

4. Declaração de Funções

As funções a seguir são declaradas para implementação no arquivo `.c`. Cada uma tem um papel específico no funcionamento da árvore binária.

```c
FILE *abrirArquivo(char *nomeArq, char *modo);
void calcularTempo(double ini, double fim);
void salvarDadosNoArquivo(ArvoreBinaria *arvore, FILE *arquivoLista);
void inicializarArvore(ArvoreBinaria *arvore, int quantidadeMatriculas);
void redimensionarArvore(ArvoreBinaria *arvore);
void imprimirEmOrdem(ArvoreBinaria *arvore, int indice);
void liberarArvore(ArvoreBinaria *arvore);
long long int pedirOpcao();
long long int pedirNum(int caminhoASerEscolhido);
void menuPrincipal(ArvoreBinaria *arvore);
int contarMatriculas(FILE *arquivoLista);
void inserirAluno(ArvoreBinaria *arvore, long long int matricula, char *nome);
void buscarAluno(ArvoreBinaria *arvore, long long int matricula);
void removerAluno(ArvoreBinaria *arvore, long long int matricula);
void lerEInserirMatriculas(ArvoreBinaria *arvore, FILE *arquivoLista);
void imprimirEmOrdem(ArvoreBinaria *arvore, int indice);
```

Tarefa 08 - Implementação do A*

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 NoCaminho Struct Reference

Representa um nó no caminho do algoritmo A*.

`#include <rascunho.h>`

Collaboration diagram for NoCaminho:

Public Attributes

- int x
- int y
- int custo_g
- int custo_h
- int custo_f
- struct NoCaminho * pai

3.1.1 Detailed Description

Representa um nó no caminho do algoritmo A*.

3.1.2 Member Data Documentation

3.1.2.1 custo_f

```
int NoCaminho::custo_f
```

Custo total (g + h).

3.1.2.2 custo_g

```
int NoCaminho::custo_g
```

Custo acumulado do início até este nó.

3.1.2.3 custo_h

```
int NoCaminho::custo_h
```

Heurística estimada até o objetivo.

3.1.2.4 pai

```
struct NoCaminho* NoCaminho::pai
```

Ponteiro para o nó pai (para reconstrução do caminho).

3.1.2.5 x

```
int NoCaminho::x
```

Coordenada x do nó.

3.1.2.6 y

```
int NoCaminho::y
```

Coordenada y do nó.

The documentation for this struct was generated from the following file:

- rascunho.h

Chapter 4

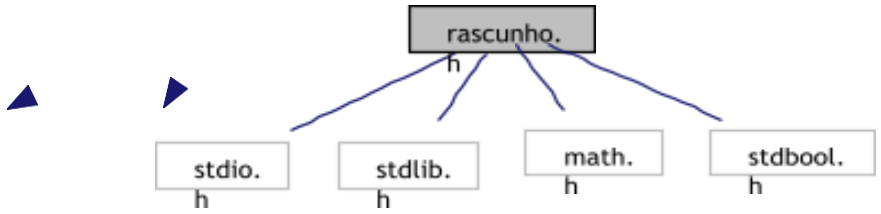
File Documentation

4.1 rascunho.h File Reference

Declarações de funções e estrutura para a implementação do algoritmo A*.

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

Include dependency graph for rascunho.h:



Classes

- struct NoCaminho

  Representa um nó no caminho do algoritmo A.*

Macros

- #define **RASCUNHO_H**
- #define LINHAS 10

Dimensões da matriz do mapa.

- #define COLUNAS 10

Typedefs

- typedef struct NoCaminho **TNoCaminho**

Functions

- void lerMapa (const char *arquivo, int mapa[LINHAS][COLUNAS])
  Lê um mapa de um arquivo e armazena em uma matriz.
- void imprimirMapa (int mapa[LINHAS][COLUNAS])
  Imprime o mapa no console.
- void buscaAestrela (int mapa[LINHAS][COLUNAS], int inicio_x, int inicio_y, int objetivo_x, int objetivo_y)
  Executa o algoritmo A para encontrar o caminho mais curto.*

4.1.1 Detailed Description

Declarações de funções e estrutura para a implementação do algoritmo A*.

4.1.2 Macro Definition Documentation

4.1.2.1 COLUNAS

```
#define COLUNAS 10
```

Número de colunas do mapa.

4.1.2.2 LINHAS

```
#define LINHAS 10
```

Dimensões da matriz do

mapa. Número de linhas do

mapa.

4.1.3 Function Documentation

4.1.3.1 buscaAestrela()

```
void buscaAestrela (
            int mapa[LINHAS][COLUNAS],
            int inicio_x,
            int inicio_y,
            int objetivo_x,
            int objetivo_y
            )
```

Executa o algoritmo A* para encontrar o caminho mais curto.

Parameters

mapa	Matriz representando o mapa.
inicio_x	Coordenada x do ponto inicial.
inicio_y	Coordenada y do ponto inicial.
objetivo↔_x	Coordenada x do ponto objetivo.
objetivo↔_y	Coordenada y do ponto objetivo.

Executa o algoritmo A* para encontrar o caminho mais curto.

Parameters

mapa	Matriz representando o mapa.
inicio_x	Coordenada x do ponto inicial.
inicio_y	Coordenada y do ponto inicial.
objetivo↔_x	Coordenada x do ponto objetivo.
objetivo↔_y	Coordenada y do ponto objetivo.

4.1.3.2 imprimirMapa()

```
void imprimirMapa (
            int mapa[LINHAS][COLUNAS] )
```

Imprime o mapa no console.

Parameters

ma↔pa	Matriz representando o mapa.

Exibe a matriz representando o mapa no console, incluindo as coordenadas iniciais e finais, além de marcar o caminho traçado.

- O valor `3` no mapa representa o ponto inicial (marcado como 'I').
- O valor `4` no mapa representa o ponto final (marcado como 'F').
- Outros valores são exibidos diretamente conforme a matriz.

Parameters

ma↔pa	Matriz representando o mapa.

4.1.3.3 lerMapa()

```
void lerMapa (
            const char *arquivo,
            int mapa[LINHAS][COLUNAS] )
```

Lê um mapa de um arquivo e armazena em uma matriz.

Parameters

arquivo	Caminho do arquivo do mapa.
mapa	Matriz onde o mapa será armazenado.

Lê um mapa de um arquivo e armazena em uma matriz.

Carrega os dados de um mapa, representados como uma matriz de inteiros, a partir de um arquivo de texto. Cada valor lido corresponde a uma célula do mapa.

- O valor `1` representa um obstáculo.
- O valor `0` representa uma célula transitável.

Parameters

arquivo	Caminho para o arquivo que contém o mapa.
mapa	Matriz onde o mapa será carregado.