



UNIVERSIDADE FEDERAL DE VIÇOSA - CAMPUS FLORESTAL

Trabalho Prático 1 de Projeto e Análise de Algoritmos

Nome: Aline Cristina Santos Silva
Gustavo Luca Ribeiro Da Silva
Luana Tavares Anselmo

Matrícula: 5791,5787,5364

Florestal - MG
2024

Sumário

1. Introdução	3
2. Compilação e Organização	4
3. Desenvolvimento	5
3.1. Estrutura Labirinto	
3.2. Função carregar_labirinto	
3.3. Função movimenta_estudante	
3.4. Função exibir_caminhos	
3.5. Função Extra: Geração de labirinto aleatório	
4. Resultados	8
5. Conclusão	11
6. Referências Bibliográficas	12

1. Introdução

O projeto desenvolvido é um programa em C que simula a navegação de um estudante em um labirinto. O labirinto é representado como uma matriz de células, onde o estudante parte de uma posição inicial e tenta alcançar a saída na linha superior. O projeto permite carregar um labirinto a partir de um arquivo, exibir o caminho percorrido e também gerar labirintos aleatórios(extra), facilitando a exploração de diferentes configurações e desafios para a simulação.

Este programa foi desenvolvido com o objetivo de aplicar e consolidar conceitos fundamentais de estruturas de dados e algoritmos, particularmente voltados para o uso de matrizes e a implementação de algoritmos de busca recursiva. Além disso, o código utiliza técnicas de memória dinâmica para gerenciar o espaço necessário para a matriz do labirinto e o caminho percorrido pelo estudante.

O programa conta com as seguintes funcionalidades:

1. **Carregar Labirinto de Arquivo:** Permite ao usuário carregar a configuração de um labirinto a partir de um arquivo especificado. O arquivo contém a estrutura da matriz e a posição inicial do estudante.
2. **Exibir Caminho Percorrido:** Uma vez que o estudante tenta resolver o labirinto, o programa armazena e exibe o caminho percorrido. Esta função permite que o usuário veja cada posição visitada na busca pela saída.
3. **Gerar Labirinto Aleatório:** Gera um labirinto com uma configuração aleatória de caminhos e barreiras, dando ao usuário a opção de definir o tamanho da matriz e o número de chaves ou pontos de interesse no labirinto.
4. **Buscar Saída:** A função principal que utiliza um algoritmo de busca recursiva para explorar o labirinto, registrando cada movimento e tentando alcançar a linha superior, considerada a saída do labirinto.

2. Compilação e Organização

O projeto está organizado em dois diretórios principais:

- **headers/**: contém todos os arquivos de cabeçalho (.h) utilizados no projeto. Esses arquivos definem as estruturas de dados e declarações de funções usadas em vários pontos do código. Exemplo: labirinto.h, gerador_labirinto.h, menu.h
- **sources/**: contém todos os arquivos de implementação (.c) do projeto, que possuem o código das funções definidas nos arquivos de cabeçalho. Exemplo: labirinto.c, gerador_labirinto.c, menu.c

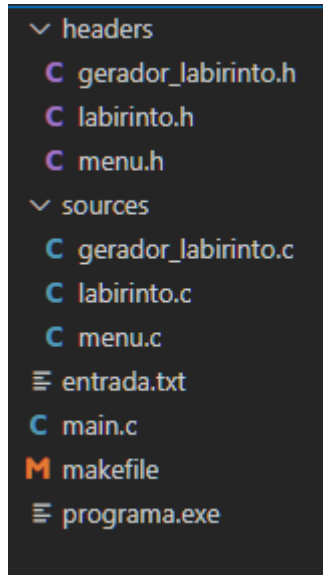


Figura 1: Organização das pastas

O Makefile é usado para automatizar o processo de compilação, simplificando a criação do executável. Ele define como os arquivos-fonte devem ser compilados e onde o executável final será salvo. Abaixo está um exemplo de Makefile para o projeto:

```
M makefile
1 all: main.c sources/gerador_labirinto.c sources/labirinto.c sources/menu.c
2   gcc -o programa main.c sources/gerador_labirinto.c sources/labirinto.c sources/menu.c -lm
3
```

Figura 2: MakeFile

Como Compilar o Projeto?

- Pré-requisitos: Certifique-se de que o gcc está instalado em seu sistema.
- Compilação: Para compilar o projeto, navegue até o diretório raiz do projeto e execute o comando: **make**
- Em seguida: **./programa**

3. Desenvolvimento

O projeto consiste em um programa que carrega um labirinto a partir de um arquivo de dados, movimenta o estudante dentro do labirinto em busca de uma saída, e exibe o caminho percorrido. A seguir, detalharemos as principais partes do código.

3.1 Estrutura Labirinto

A estrutura Labirinto é central para o programa, representando o estado do labirinto e armazenando informações importantes, como a matriz de células do labirinto, o número de chaves, posições iniciais e finais, e as posições visitadas pelo estudante.

3.2 Função carregar_labirinto

A função carregar_labirinto lê um arquivo de entrada com as dimensões e a disposição do labirinto, carregando esses dados para a estrutura Labirinto. Ela identifica a posição inicial do estudante (representada pelo valor 0 no arquivo) e armazena essa posição para iniciar a movimentação.

3.3 Função movimenta_estudante

A função movimenta_estudante aplica a técnica de backtracking para explorar todas as possíveis rotas no labirinto. Essa abordagem permite que o estudante retroceda quando encontra um caminho sem saída, explorando alternativas até encontrar uma solução ou concluir que não há caminho viável. O uso de backtracking aqui é ideal para resolver o problema, pois o labirinto pode conter muitos becos sem saída e vários caminhos possíveis. O funcionamento dela pode ser resumido da seguinte forma:

1. **Verificação de Limites:** Inicialmente, verifica se a posição atual está dentro dos limites do labirinto e se a célula é válida para movimentação (não visitada e não é uma parede).
2. **Condição de Saída:** Se o estudante atinge a linha superior do labirinto, a função registra a posição como a saída e retorna sucesso (1).
3. **Marca Visitada:** A posição atual é marcada como visitada para evitar retornos a ela durante a exploração.
4. **Movimento Recursivo:** A função tenta mover o estudante em quatro direções (cima, baixo, esquerda e direita), chamando-se recursivamente para cada direção. Se algum movimento encontrar a saída, a função retorna sucesso e atualiza o contador de movimentos.
5. **Backtracking:** Caso nenhuma direção leve à saída, a função "volta atrás", desfazendo a marcação na célula e permitindo que outras rotas sejam exploradas a partir dessa posição.

```
int movimenta_estudante(Labirinto *labirinto, int linha, int coluna, int *movimentos, int *max_recurso, int nivel_atual) {
    if (linha < 0 || coluna < 0 || linha >= labirinto->linhas || coluna >= labirinto->colunas || labirinto->matriz[linha][coluna] == 2 || labirinto->matriz[linha][coluna] == 1)
        return 0;

    if (linha == 0) {
        labirinto->pos_final_coluna = coluna;
        labirinto->posicoes_visitadas[labirinto->total_movimentos++] = (Posicao){linha, coluna}; // Registra posição final
        return 1;
    }

    labirinto->matriz[linha][coluna] = -1;
    labirinto->posicoes_visitadas[labirinto->total_movimentos++] = (Posicao){linha, coluna};

    if (ANALISE && nivel_atual > *max_recurso) *max_recurso = nivel_atual;

    int sucesso = movimenta_estudante(labirinto, linha - 1, coluna, movimentos, max_recurso, nivel_atual + 1) ||
        movimenta_estudante(labirinto, linha + 1, coluna, movimentos, max_recurso, nivel_atual + 1) ||
        movimenta_estudante(labirinto, linha, coluna - 1, movimentos, max_recurso, nivel_atual + 1) ||
        movimenta_estudante(labirinto, linha, coluna + 1, movimentos, max_recurso, nivel_atual + 1);

    if (sucesso) {
        (*movimentos)++;
        return 1;
    }

    labirinto->matriz[linha][coluna] = 1;
    labirinto->total_movimentos--;
    return 0; }

```

Figura 3: função movimenta_estudante

3.4 Função exibir_caminhos

A função exibir_caminho percorre o array de posições visitadas e exibe cada movimento do estudante no terminal, mostrando o caminho percorrido até encontrar a saída (ou até constatar que não há saída).

3.5 Função Extra: Geração de labirinto aleatório

Uma funcionalidade extra implementada no projeto é a geração de um labirinto aleatório, que permite ao usuário criar um labirinto de tamanho definido e com um número especificado de chaves. Esse labirinto é então salvo em um arquivo para uso futuro.

A função **gerar_labirinto_aleatorio** gera um labirinto de tamanho especificado com células aleatórias (caminhos, paredes e chaves). A posição inicial do estudante é definida aleatoriamente na última linha, enquanto a saída é considerada na primeira linha.

Após a geração, o labirinto pode ser salvo em um arquivo para reutilização, através da função **salvar_labirinto_em_arquivo**, que armazena as dimensões e a matriz do labirinto no formato adequado.

```

void gerar_labirinto_aleatorio(Labirinto *labirinto, int tamanho, int chaves) {
    labirinto->colunas = tamanho;
    labirinto->linhas = tamanho; // Número de linhas igual ao número de colunas
    labirinto->num_chaves = chaves;

    labirinto->matriz = malloc(labirinto->linhas * sizeof(int *));
    for (int i = 0; i < labirinto->linhas; i++) {
        labirinto->matriz[i] = malloc(labirinto->colunas * sizeof(int));
        for (int j = 0; j < labirinto->colunas; j++) {
            labirinto->matriz[i][j] = rand() % 2 + 1;
        }
    }

    labirinto->pos_inicial_linha = labirinto->linhas - 1;
    labirinto->pos_inicial_coluna = rand() % labirinto->colunas;
    labirinto->matriz[labirinto->pos_inicial_linha][labirinto->pos_inicial_coluna] = 0;

    for (int i = 0; i < (labirinto->linhas * labirinto->colunas) / 4; i++) {
        int linha = rand() % labirinto->linhas;
        int coluna = rand() % labirinto->colunas;
        labirinto->matriz[linha][coluna] = 2;
    }

    for (int i = 0; i < chaves; i++) {
        int linha = rand() % labirinto->linhas;
        int coluna = rand() % labirinto->colunas;
        while (labirinto->matriz[linha][coluna] != 1) {
            linha = rand() % labirinto->linhas;
            coluna = rand() % labirinto->colunas;
        }
        labirinto->matriz[linha][coluna] = 3;
    }
}

```

Figura 4: Função gerar_labirinto_aleatorio

```

void salvar_labirinto_em_arquivo(Labirinto *labirinto, const char *nome_arquivo) {
    FILE *arquivo = fopen(nome_arquivo, "w");
    if (!arquivo) {
        printf("Erro ao abrir o arquivo para gravação.\n");
        return;
    }

    // Escreve as dimensões do labirinto e o número de chaves no arquivo
    fprintf(arquivo, "%d %d %d\n", labirinto->linhas, labirinto->colunas, labirinto->num_chaves);

    // Escreve o labirinto no arquivo
    for (int i = 0; i < labirinto->linhas; i++) {
        for (int j = 0; j < labirinto->colunas; j++) {
            fprintf(arquivo, "%d", labirinto->matriz[i][j]);
        }
        fprintf(arquivo, "\n");
    }

    fclose(arquivo);
    printf("Labirinto salvo com sucesso no arquivo %s.\n", nome_arquivo);
}

```

Figura 5: Função salvar_labirinto_em_arquivo

4. Resultados

```
PROGRAMA LABIRINTO
=====
OPÇÕES DO PROGRAMA
=====
1) Carregar novo arquivo de dados
2) Exibir resultados
3) Caminho Percorrido
4) Gerar labirinto aleatório
5) Sair do programa
=====
Digite uma opção: 1
Por favor digite o nome do arquivo: entrada.txt
Labirinto carregado com sucesso.

Pressione Enter para voltar ao menu...

PROGRAMA LABIRINTO
=====
OPÇÕES DO PROGRAMA
=====
1) Carregar novo arquivo de dados
2) Exibir resultados
3) Caminho Percorrido
4) Gerar labirinto aleatório
5) Sair do programa
=====
Digite uma opção: 2
O estudante se movimentou 26 vezes e chegou na coluna 5 da primeira linha.
Nível máximo de recursão: 26

Pressione Enter para voltar ao menu...
```



```

1) Carregar novo arquivo de dados
2) Exibir resultados
3) Caminho Percorrido
4) Gerar labirinto aleatório
5) Sair do programa

Digite uma opção: 3
Caminho percorrido:
Linha: 9 Coluna: 4
Linha: 8 Coluna: 4
Linha: 8 Coluna: 3
Linha: 9 Coluna: 3
Linha: 9 Coluna: 2
Linha: 8 Coluna: 2
Linha: 8 Coluna: 1
Linha: 9 Coluna: 1
Linha: 9 Coluna: 0
Linha: 8 Coluna: 0
Linha: 7 Coluna: 0
Linha: 6 Coluna: 0
Linha: 5 Coluna: 0
Linha: 4 Coluna: 0
Linha: 3 Coluna: 0
Linha: 3 Coluna: 1
Linha: 4 Coluna: 1
Linha: 4 Coluna: 2
Linha: 3 Coluna: 2
Linha: 3 Coluna: 3
Linha: 4 Coluna: 3
Linha: 4 Coluna: 4
Linha: 3 Coluna: 4
Linha: 3 Coluna: 5
Linha: 2 Coluna: 5
Linha: 1 Coluna: 5
Linha: 0 Coluna: 5

Pressione Enter para voltar ao menu...

```

Figura 6: Saída das opções 1,2 e 3 em ordem

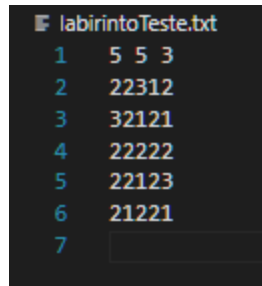
C labirinto.h
C menu.h
sources
C gerador_labirinto.c
C labirinto.c
C menu.c
entrada.txt
labirintoTeste.txt
main.c
makefile
programa.exe

PROGRAMA LABIRINTO
OPÇÕES DO PROGRAMA
1) Carregar novo arquivo de dados
2) Exibir resultados
3) Caminho Percorrido
4) Gerar labirinto aleatório
5) Sair do programa

Digite uma opção: 4
* Digite o tamanho do labirinto (linhas = colunas): 5
* Digite o número de chaves: 3
Labirinto gerado com sucesso.
* Digite o nome do arquivo para salvar o labirinto: labirintoTeste.txt
Labirinto salvo com sucesso no arquivo labirintoTeste.txt.

Pressione Enter para voltar ao menu...

Figura 7: Saída da opção 4



```
labirintoTeste.txt
1 5 5 3
2 22312
3 32121
4 22222
5 22123
6 21221
7
```

Figura 8: Arquivo gerado pela opção 4

5. Conclusão

Este projeto permitiu explorar o uso de algoritmos de busca e técnicas de backtracking aplicadas à resolução de problemas de navegação em labirintos.

Ao longo do desenvolvimento, enfrentamos desafios na implementação do backtracking, principalmente na gestão da recursão para garantir que o estudante percorresse todas as possibilidades até encontrar a saída ou concluir que o labirinto não possui solução.

Esta parte exigiu atenção especial para o controle de movimentos e armazenamento de posições visitadas, além de lidar com limitações de memória e eficiência para evitar loops desnecessários.

Entre as facilidades, destacamos a modularização do código, que simplificou o desenvolvimento e tornou a adição de novas funcionalidades mais organizada. A criação de uma função para gerar labirintos aleatórios foi um acréscimo interessante, proporcionando uma ferramenta para testar o algoritmo em diferentes cenários e garantindo que a solução fosse suficientemente flexível para adaptar-se a diferentes configurações de labirinto.

Esse trabalho ofereceu uma experiência prática em estrutura de dados, manipulação de memória em C e uso de algoritmos de busca e recursão, consolidando conhecimentos fundamentais para a resolução de problemas computacionais.

6. Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009).

Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language*. Prentice Hall

Weiss, M. A. (2013). *Data Structures and Algorithm Analysis in C*. Pearson.