

Universidade Federal de Viçosa - Campus Florestal



**Trabalho 01 da Disciplina de Projeto e Análise de Algoritmos**  
Backtracking

Isabella Menezes Ramos	3474
Larissa Isabelle Rodrigues e Silva	3871

**Professor:** Daniel Mendes Barbosa

Florestal - MG  
12 de Abril de 2021

Isabella Menezes Ramos 3474  
Larissa Isabelle Rodrigues e Silva 3871

## **Trabalho 01 da Disciplina de Projeto e Análise de Algoritmos**

Primeiro trabalho prático da disciplina Projeto e Análise de Algoritmos. O trabalho tem o intuito de tentar criar uma solução para a resolução de um labirinto.

Professor: Daniel Mendes Barbosa

Florestal - MG  
12 de Abril de 2021

## Sumário

<b>1 Introdução</b>	<b>4</b>
<b>2 Desenvolvimento</b>	<b>5</b>
<b>3 Resultados</b>	<b>10</b>
<b>4 Conclusão</b>	<b>15</b>
<b>5 Referências</b>	<b>16</b>

# 1 Introdução

O trabalho proposto tem o objetivo de fazer uma implementação de um programa, utilizando backtracking, que consiga resolver problemas de labirintos inseridos por meio de arquivos. Caso o labirinto não tenha solução, deve-se informar ao usuário. Além disso, deve-se informar quais células da matriz o estudante passou, a quantidade de movimentações que ele teve, e caso o labirinto tenha uma saída, fornecer ao usuário em qual coluna o estudante conseguiu chegar ao fim do labirinto.

Também foi implementado no trabalho um `#define` para que o usuário consiga executar o programa de duas formas: pelo modo normal ou pelo modo análise. Pelo modo normal, as informações descritas na tela deverão aparecer como dito anteriormente. Se estiver no modo análise, deverá fazer tudo, mas também contabilizar o número total de chamadas recursivas que foram feitas e o nível máximo de recursividade alcançado durante toda a solução.

Nos capítulos seguintes iremos detalhar sobre o desenvolvimento do trabalho. No capítulo 2, aprofundaremos sobre o desenvolvimento, explicando a maneira que o algoritmo foi implementado e as estruturas de dados que foram criadas. No capítulo 3, iremos mostrar como executar o trabalho e mostraremos os resultados obtidos, através de capturas de tela, e mostraremos os arquivos que foram testados no programa. No capítulo 4, fazemos a conclusão do trabalho.

## 2 Desenvolvimento

O código do trabalho está dividido da seguinte forma, o arquivo dados.c é composto pelas funções responsáveis por armazenar e manipular os dados solicitados ao fim da execução do labirinto. O arquivo labirinto.c possui as funções necessárias para a criação e manipulação do labirinto, além disso temos também as funções de backtracking do modo normal e do modo análise. O arquivo menu.c possui o menu visual do programa.

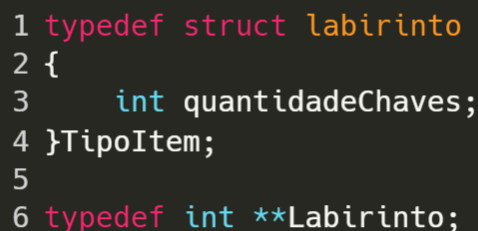
Para diferenciar o modo normal do modo análise, foi adicionado no menu principal um `#define` para ativar ou desativar o modo análise, caso o modo análise for igual a 1 ele está ativo, caso o modo análise for igual a 0, o modo normal está ativo.



```
1 #define MODOANALISE 1 //1 ativado, 0 desativado
```

**Imagem 01.** Define do modo análise presente no main.c

Para o labirinto, foi criado uma estrutura chamada de `TipoItem`, utilizada para guardar a quantidade de chaves que terá no bolso do estudante. Além disso, foi criado um `typedef` de inteiros com dois apontamentos para que o labirinto seja desse tipo.



```
1 typedef struct labirinto
2 {
3     int quantidadeChaves;
4 }TipoItem;
5
6 typedef int **Labirinto;
```

**Imagem 02.** Estruturas criadas no labirinto.h.

As funções criadas para o desenvolvimento do labirinto estão presentes na imagem 02. A função *InicializaLabirinto* é responsável por criar a matriz do labirinto e armazenar a quantidade de chaves que há no bolso do estudante. Como não

sabemos o tamanho exato da matriz, é utilizado a alocação dinâmica de memória para criar a matriz. Para o trabalho, foi utilizado a função `calloc` da biblioteca `stdlib.h`.

A função *PreencherLabirinto* é responsável por preencher cada célula da matriz criada na função anterior. Através da leitura de cada item do arquivo, ela preenche cada célula, sendo que os valores podem ser 0 caso for a posição inicial do aluno, 1 se for célula vazia, 2 se a célula for ocupada por uma parede e 3 caso for uma porta.

Para visualizar o labirinto, foi criada a função *ImprimirLabirinto*. Nela mostra os números ocupados por cada célula da matriz. Caso a célula for um caminho onde o estudante passou, ao invés de números, terá um asterisco na célula.

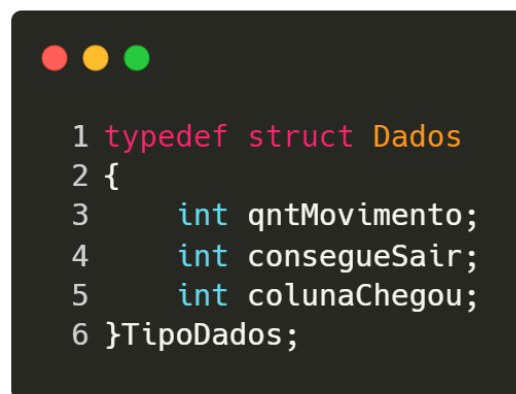
As funções seguintes, declaradas em `labirinto.h`, são funções auxiliares para a execução do algoritmo de backtracking. A função *EhParede* verifica se a célula tem uma parede. Já a função *EhPorta* verifica se a célula tem uma porta fechada. A função *ChegouNoFim* verifica se o aluno chegou na primeira linha da matriz, ou seja, no fim do labirinto. A função *MarcarCaminho* é responsável por marcar as células que o aluno passou. As funções *ColunaPosicaoAtual* e *LinhaPosicaoAtual* retornam a coluna e a linha que o aluno se encontra inicialmente, respectivamente. A função *PosicaoInicial* é responsável por verificar se determinada célula representa a posição inicial do aluno. A função *ForaDaMatriz* é responsável por verificar se a função de backtracking está verificando um espaço inválido. A função *CaminhoMarcado* é responsável por verificar se determinada célula é um local onde o estudante já passou. Já a função *AbrirPorta* é responsável por mudar o valor da célula de uma matriz para o valor correspondente a uma célula vazia.

```
1 int **InicializarLabirinto(int linha, int coluna, TipoItem *item, int chave);
2 void PreencherLabirinto(int **Labirinto, int linha, int coluna, int valor);
3 void ImprimirLabirinto(int **Labirinto, int linha, int coluna);
4 int EhParede(int **Labirinto, int linha, int coluna);
5 int EhPorta(int **Labirinto, int linha, int coluna);
6 int ChegouNoFim(int **Labirinto, int linha, int coluna);
7 void MarcarCaminho(int **Labirinto, int linha, int coluna);
8 int ColunaPosicaoAtual(int **Labirinto, int linha, int coluna);
9 int LinhaPosicaoAtual(int **Labirinto, int linha, int coluna);
10 int PosicaoInicial(int **Labirinto, int linha, int coluna);
11 int ForaDaMatriz(int i, int j, int linha, int coluna);
12 int CaminhoMarcado(int **Labirinto, int linha, int coluna);
13 int AbrirPorta(int **Labirinto, int linha, int coluna);
14 int MovimentaEstudante(int **Labirinto, TipoItem *item, int i, int j, int linha, int coluna,
    TipoDados *dados);
15 int MovimentaEstudanteAnalise(int **Labirinto, TipoItem *item, int i, int j, int linha, int
    coluna, TipoDados *dados, long long int *numRec);
```

**Imagem 03.** Funções declaradas no labirinto.h e criadas no labirinto.c.

As funções *MovimentaEstudante* e *MovimentaEstudanteAnalise* são as funções responsáveis por fazer o backtracking de fato, sendo que a função *MovimentaEstudante* é chamada durante a execução do modo normal do programa e a função *MovimentaEstudanteAnalise* é chamada quando o modo análise é ativado. A função de backtracking das duas funções é a mesma, é feito uma chamada recursiva para as células adjacentes, e é verificado se o estudante consegue acessar alguma das células adjacentes, caso ele não tenha ainda passado por elas, ou seja, se a célula é vazia ou se for uma porta, verificar se o estudante ainda tem chaves restantes para abri-la, lembrando que a cada movimentação feita pelo aluno, é contabilizado dentro da estrutura *TipoDados*. A diferença entre as duas é que na função *MovimentaEstudanteAnalise* é contado o número de análises recursivas, requerido no modo análise.

Para a contabilização dos dados requeridos na especificação do trabalho, foi criado uma estrutura chamada de *TipoDados*, onde nela é armazenada a quantidade de movimentos que o estudante fez, se ele conseguiu sair do labirinto e qual coluna da primeira posição ele conseguiu chegar, a estrutura está presente na imagem 04.

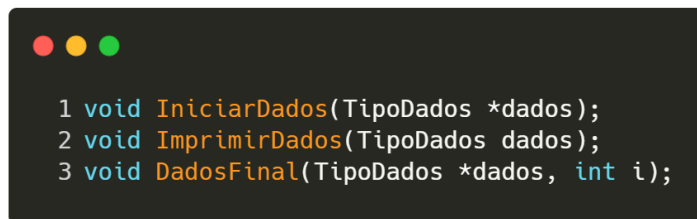


```
1 typedef struct Dados
2 {
3     int qntMovimento;
4     int consegueSair;
5     int colunaChegou;
6 }TipoDados;
```

**Imagem 04.** Estrutura TipoDados criada em dados.h.

As funções criadas para os dados solicitados estão presentes na imagem 05. A função *IniciarDados* é responsável por inicializar os dados da estrutura TipoDados. A função *ImprimirDados* é responsável por verificar se o estudante conseguiu sair do labirinto e informar ao usuário a quantidade de movimentações que o estudante fez. Já a função *DadosFinal* é responsável por incrementar pela

última vez a variável `qntMovimento` e atribuir os valores da coluna em que o aluno saiu do labirinto e da variável `consegueSair` para 1, caso o aluno consiga sair do labirinto.



```
1 void IniciarDados(TipoDados *dados);  
2 void ImprimirDados(TipoDados dados);  
3 void DadosFinal(TipoDados *dados, int i);
```

**Imagem 05.** Funções declaradas em `dados.h` e criadas em `dados.c`

Como tarefa extra foi implementado também um gerador de labirintos, com três funções que têm como objetivo a criação de arquivos em três níveis diferentes: níveis fácil, médio e difícil. Na criação das funções foram criadas variáveis recebendo valores aleatórios para definir a matriz. Para diferenciar os níveis, as condições de preenchimento do labirinto e quantidade de chaves iniciais foram mudadas de forma que ficasse balanceado com o nível. Ficou definido como matriz de nível fácil aquelas que possuem tamanho de 5x5 até 10x10; nível médio com tamanho de 10x10 até 25x25; e por fim, como nível difícil temos matrizes de tamanho 25x25 até 50x50. As funções declaradas estão presentes na imagem 6.



```
1 void modoFacil();  
2 void modoMedio();  
3 void modoDificil();
```

**Imagem 06.** Funções declaradas em `gerador.h` e criadas em `gerador.c`

Na imagem 7 podemos ver a implementação da função *modoMedio*.



```
1 if (arquivo == NULL){
2     printf("Erro ao abrir o arquivo!\n");
3 }
4 else{
5     fprintf(arquivo, "%d %d %d \n", linha, coluna, quantidadeChave); //primeira linha do arquivo
6     for (int i = 0; i < linha; i++){
7         for (int j = 0; j < coluna; j++){
8             if (i == alunoLinha && j == alunoColuna){
9                 celula = 0; //quando chegar na posição[i][j] correspondente à posição que o
aluno está, escreve 0 (aluno)
10             }
11             else{
12                 celula = 1 + rand() % 3; //soma 1 para não cair onde está o aluno
13             }
14             fprintf(arquivo, "%d", celula);
15         }
16         fputc('\n', arquivo); //adiciona ao final de cada linha uma quebra de linha
17     }
18     fclose(arquivo);
19 }
```

**Imagem 07.** Função *modoMedio()* criada em gerador.c.

### 3 Resultados

Nesta seção iremos mostrar algumas capturas de tela da execução do programa. Para executar, além de ter um compilador de c, deve-se digitar no terminal dentro da pasta do projeto:

```
make all
```

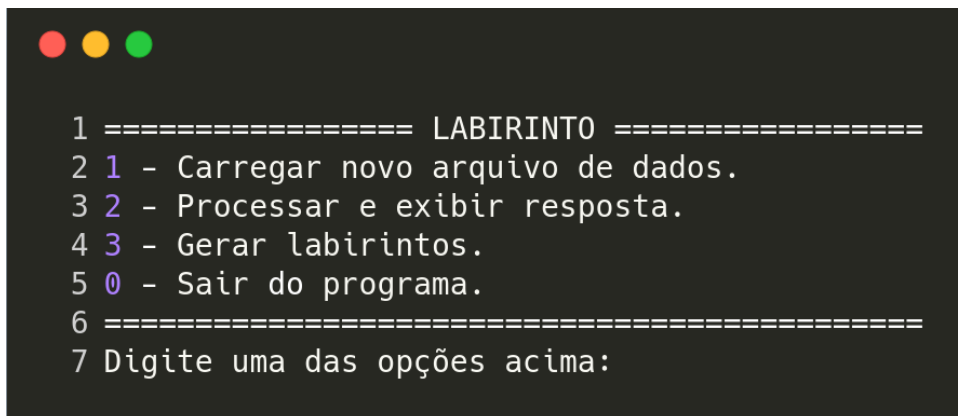
Ou pode digitar o comando:

```
gcc main.c -o EXEC sources/menu.c sources/dados.c  
sources/labirinto.c sources/gerador.c
```

E logo em seguida digitar o comando:

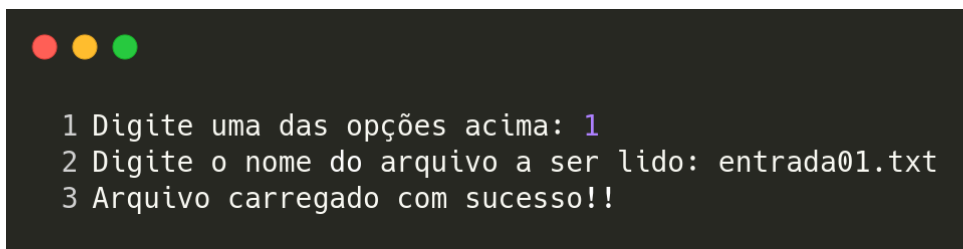
```
./EXEC
```

Segue imagens do programa em execução, com menus, resultados e alguns arquivos de teste:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The text displayed is a menu for a program called 'LABIRINTO'. It lists five options: 1 - Carregar novo arquivo de dados, 2 - Processar e exibir resposta, 3 - Gerar labirintos, and 0 - Sair do programa. The prompt asks the user to enter one of these options.

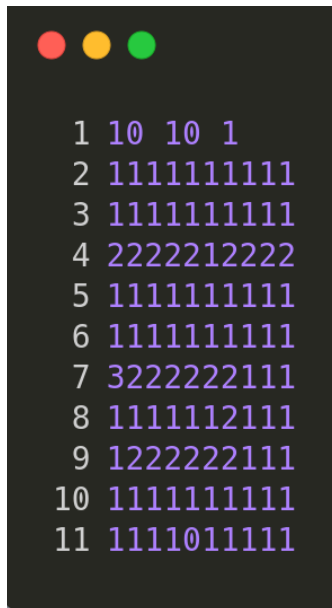
```
1 ===== LABIRINTO =====  
2 1 - Carregar novo arquivo de dados.  
3 2 - Processar e exibir resposta.  
4 3 - Gerar labirintos.  
5 0 - Sair do programa.  
6 =====  
7 Digite uma das opções acima:
```

**Imagem 08.** Menu principal, terminal.

A terminal window with a dark background and three colored window control buttons in the top-left corner. The text shows the user entering '1' at the prompt, followed by the program asking for the filename 'entrada01.txt' and then displaying a success message.

```
1 Digite uma das opções acima: 1  
2 Digite o nome do arquivo a ser lido: entrada01.txt  
3 Arquivo carregado com sucesso!!
```

**Imagem 09.** Ao entrar com o nome do arquivo desejado, é necessário escrevê-lo com sua extensão.



A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It displays 11 lines of text, each starting with a line number followed by a space and then a sequence of digits. The digits are colored purple in the original image. The lines are as follows:

Line	Content
1	10 10 1
2	1111111111
3	1111111111
4	2222212222
5	1111111111
6	1111111111
7	3222222111
8	111112111
9	1222222111
10	1111111111
11	1111011111

**Imagem 10.** Arquivo entrada01.txt utilizado como exemplo para os resultados.

```
1 Digite uma das opções acima: 2
2 Linha: 9 Coluna: 4
3 Linha: 8 Coluna: 4
4 Linha: 8 Coluna: 3
5 Linha: 8 Coluna: 2
6 Linha: 8 Coluna: 1
7 Linha: 8 Coluna: 0
8 Linha: 7 Coluna: 0
9 Linha: 6 Coluna: 0
10 Linha: 5 Coluna: 0
11 Linha: 4 Coluna: 0
12 Linha: 3 Coluna: 0
13 Linha: 3 Coluna: -1
14 Linha: 4 Coluna: 0
15 Linha: 3 Coluna: 1
16 Linha: 3 Coluna: 0
17 Linha: 4 Coluna: 1
18 Linha: 3 Coluna: 1
19 Linha: 4 Coluna: 0
20 Linha: 4 Coluna: 2
21 Linha: 3 Coluna: 2
22 Linha: 3 Coluna: 1
23 Linha: 4 Coluna: 2
24 Linha: 3 Coluna: 3
25 Linha: 3 Coluna: 2
26 Linha: 4 Coluna: 3
27 Linha: 3 Coluna: 3
28 Linha: 4 Coluna: 2
29 Linha: 4 Coluna: 4
30 Linha: 3 Coluna: 4
31 Linha: 3 Coluna: 3
32 Linha: 4 Coluna: 4
33 Linha: 3 Coluna: 5
34 Linha: 2 Coluna: 5
35 Linha: 1 Coluna: 5
36 Linha: 0 Coluna: 5
37
38 ===== MODO ANALISE =====
39 0 total de chamadas recursivas foi de: 45
40 =====
41
42 1 1 1 1 1 * 1 1 1 1
43 1 1 1 1 1 * 1 1 1 1
44 2 2 2 2 2 * 2 2 2 2
45 * * * * * 1 1 1 1
46 * * * * * 1 1 1 1
47 * 2 2 2 2 2 2 1 1 1
48 * 1 1 1 1 1 2 1 1 1
49 * 2 2 2 2 2 2 1 1 1
50 * * * * * 1 1 1 1
51 1 1 1 1 * 1 1 1 1
52
53 0 estudante se movimentou 34 vezes e chegou na coluna 5 da primeira linha.
```

**Imagem 11.** Resultado em modo análise do arquivo de teste *entrada01.txt*.

```
1 ===== GERADOR DE LABIRINTOS =====
2 1 - Modo facil.
3 2 - Modo medio.
4 3 - Modo dificil.
5 0 - Sair do programa.
6 =====
7 Digite uma das opções acima:
```

**Imagem 12.** Menu do programa gerador de labirintos.

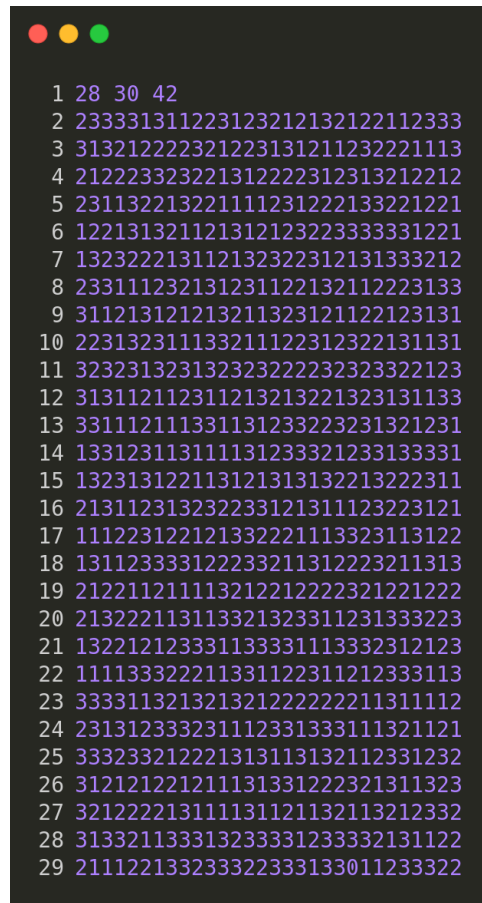
Abaixo, segue os arquivos de teste gerados nos três níveis.

```
1 7 8 1
2 11111111
3 11111111
4 11111111
5 11111211
6 11111111
7 11110311
8 11111111
```

**Imagem 13.** Arquivo teste gerado no modo fácil, nomeado *facil.txt*.

```
1 18 20 4
2 11221233122313133213
3 33231223132131331223
4 22121232132221312121
5 21331313313232112221
6 33323122212311122333
7 21323331111332133233
8 1112311113333222113
9 21231211313113333332
10 32112222111332113332
11 3213131212223333312
12 11321131332131231321
13 11112231122123121332
14 31232232212322121121
15 33311331331131332333
16 13112221321322233123
17 12313333223211111213
18 11323332232313223031
19 21221321232332223231
```

**Imagem 14.** Arquivo teste gerado no modo médio, nomeado *medio.txt*



```
1 28 30 42
2 233331311223123212132122112333
3 3132122232122313121123221113
4 2122332322131222312313212212
5 231132213221111231222133221221
6 12213132112131212322333331221
7 13232221311213232231213133212
8 233111232131231122132112223133
9 311213121213211323121122123131
10 223132311133211122312322131131
11 3232313231323232223232322123
12 313112112311213213221323131133
13 331112111331131233223231321231
14 133123113111131233321233133331
15 13231312211312131313221322311
16 213112313232233121311123223121
17 111223122121332221113323113122
18 131123333122233211312223211313
19 212211211113212212222321221222
20 213222113113321323311231333223
21 132212123331133331113332312123
22 111133322211331122311212333113
23 333311321321321222222211311112
24 231312333231112331333111321121
25 333233212221313113132112331232
26 312121221211131331222321311323
27 321222213111131121132113212332
28 313321133313233331233332131122
29 211122133233322333133011233322
```

**Imagem 15.** Arquivo teste gerado no modo difícil, nomeado *difícil.txt*.

## 4 Conclusão

O trabalho prático foi de extrema importância para nós, para que colocássemos em prática os conhecimentos sobre backtracking obtidos em sala, já que o trabalho é uma boa forma de exercitarmos o nosso conhecimento adquirido. Acreditamos que conseguimos cumprir todos os requisitos da especificação do trabalho prático e também de algumas tarefas adicionais.

No início houve algumas dificuldades sobre a lógica inicial para desenvolver o backtracking, mas após alguns rascunhos e testes acabou dando certo. A execução do trabalho foi de extrema importância para aprofundarmos o pensamento lógico.

## 5 Referências

CURSO de C: Alocação Dinâmica. Alocação Dinâmica. 1999. Disponível em: <https://www.pucsp.br/~so-comp/cursoc/aulas/ca60.html#cA62>. Acesso em: 04 abr. 2021.

DELGADO, Armando Luiz Nicolini. Alocação dinâmica de memória. 2013. Disponível em: [https://www.inf.ufpr.br/cursos/ci067/Docs/NotasAula/notas-32\\_Aloca\\_c\\_cao\\_dinamica\\_mem.html](https://www.inf.ufpr.br/cursos/ci067/Docs/NotasAula/notas-32_Aloca_c_cao_dinamica_mem.html). Acesso em: 09 abr. 2021.