

Universidade Federal de Viçosa - CAF  
Projeto e Análise de Algoritmos - Ciência da Computação

André Elias (3013) - Arthur Marciano (3019)

**Trabalho Prático 1: Estudante no labirinto**

Florestal

2019

# SUMÁRIO

1 INTRODUÇÃO	4
2 SOBRE O ALGORITMO	4
3 LIMITAÇÕES DO ALGORITMO	9
4 CONCLUSÃO	9
5 REFERÊNCIAS	10

## LISTA DE IMAGENS

Imagem 1 – Interface	4
Imagem 2 – Exemplo saída bem sucedida	5
Imagem 3 – Exemplo saída fracassada	5
Imagem 4 – Percurso básico	6
Imagem 5 – Percurso estudante inteligente	6
Imagem 6 – Parâmetros movimentoEstudante	6
Imagem 7 - Condições de parada movimentoEstudante	7
Imagem 8 – Condições de sucesso movimentoEstudante	7
Imagem 9 – Chamadas recursivas cima, esquerda, direita, baixo	8
Imagem 10 – Funções utilizadas	8
Imagem 11 – Matrizes utilizadas	9

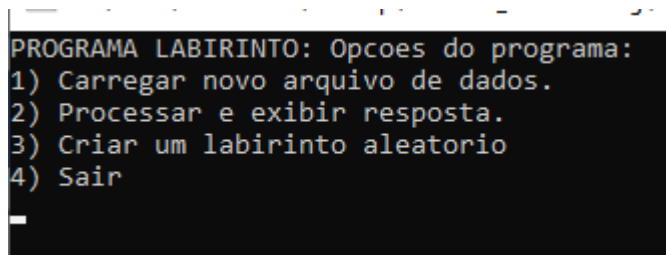
## 1. Introdução

Este trabalho consiste na implementação de um algoritmo utilizando backtracking para descobrir uma ou mais maneiras de um estudante sair de um labirinto qualquer, se possível. Além da possibilidade de se criar um novo labirinto com os parâmetros dados pelo usuário.

Foi utilizado dois tipos abstrato de dados, um “Menu.c” e “funcoes.c”, onde o Menu trata a interface do usuário com suas respectivas entradas, e a funcoes.c contém toda a lógica algorítmica da aplicação

## 2. Sobre o algoritmo

O algoritmo começa chamando o método MenuPrincipal que contém alguns switch’s para as opções escolhidas pelo usuário pela seguinte interface:



```
PROGRAMA LABIRINTO: Opcoes do programa:  
1) Carregar novo arquivo de dados.  
2) Processar e exibir resposta.  
3) Criar um labirinto aleatorio  
4) Sair  
_
```

Imagem 1 – Interface

Este algoritmo permite o carregamento de determinado arquivo parâmetro do labirinto com a quantidade de linhas, colunas e quantidade de chaves iniciais do estudante preso, como também processar e exibir resposta (abordado mais a frente), criar um labirinto de acordo com os parâmetros dado pelo usuário e sair.

Selecionando a opção de carregar o arquivo será pedido o caminho do arquivo com seu respectivo nome e extensão, para um arquivo localizado na pasta do programa só será necessário escrever o nome do arquivo com sua extensão (Ex: Entrada.txt).

Selecionando a opção de processar e exibir resposta (funcionará apenas se selecionado um arquivo antes) será exibido todo o caminho percorrido pelo estudante até encontrar a saída ou perceber que não é possível sair, como também o total de movimentos e a coluna que ele chegou. Eis um exemplo de um caso que o estudante conseguiu sair e outro que não:

```

Linha: 6 Coluna: 7
Linha: 5 Coluna: 7
Linha: 4 Coluna: 7
Linha: 3 Coluna: 7
Linha: 3 Coluna: 6
Linha: 3 Coluna: 5
Linha: 2 Coluna: 5
Linha: 1 Coluna: 5
Linha: 0 Coluna: 5
O estudante se movimentou 29 vezes e chegou na 6 coluna

Escolha o tipo de caminho a ser mostrado:
1)Mostrar todo o caminho do estudante
2)Mostrar caminho resumido

```

Imagem 2 – Exemplo saída bem sucedida

```

Linha: 8 Coluna: 6
Linha: 8 Coluna: 7
Linha: 8 Coluna: 8
Linha: 9 Coluna: 8
Linha: 9 Coluna: 7
Linha: 9 Coluna: 6
Linha: 9 Coluna: 5
O estudante se movimentou 16 vezes e percebeu que o labirinto nao tem saida

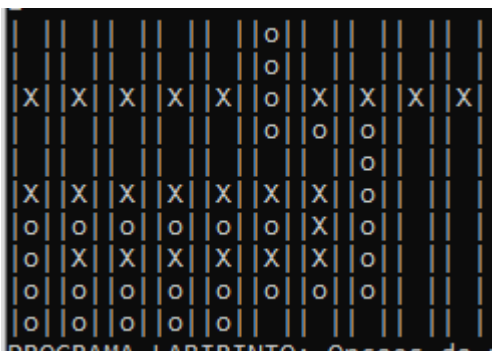
Escolha o tipo de caminho a ser mostrado:
1)Mostrar todo o caminho do estudante
2)Mostrar caminho resumido

```

Imagem 3 – Exemplo saída fracassada

Logo após isso é apresentado duas formas de visualização, a primeira mostra todo o caminho percorrido pelo estudante, ou seja, mostrando cada tentativa errada que ele cometeu (chegar em uma passagem sem saída) até a saída do labirinto. A segunda maneira é um estudante mais inteligente, que sabe que determinado caminho dará errado e o evita. Caso o labirinto não tenha saída o estudante, inteligente como é, saberá que qualquer esforço será desnecessário e ficará parado no lugar esperando o resgate. No caso de conter uma saída ele percorrerá um caminho “direto” a saída, importante ressaltar que não representa necessariamente o menor caminho possível.

Eis a diferença entre o estudante comum e o inteligente:



```
| | | | | | | | | | | | | | | | | | | | | |
```

X	X	X	X	X	^	X	X	X	X
					^	<	<		
X	X	X	X	X	X	X	^		
		X	X	X	X	X	^		
				>	>	>	^		
				^					

PROGRAMA LABIRINTO: Oquees do

O algoritmo de backtracking foi feito da seguinte maneira:

Imagem 6 –Parâmetros movimentaEstudante

A seguir as condições de parada do programa:

Imagem 7 – Condições de parada movimentada

Todos os “return false” representam tentativas frustradas do backtracking, acontece quando o estudante sai do escopo da matriz, quando chega em uma parede ou quando chega em uma porta sem chave.

As condições de qtdChaves são usadas para implementar a funcionalidade de pegar ou devolver chaves no chão, acrescentando ou diminuindo ao total de chaves do estudante.

Tentativas acertadas do backtracking acontece quando nenhuma condição de return false é ativada, o algoritmo sabe que chegou no fim quando chega na linha 0 do programa.

```
if (linhaAtual == 0) {  
    caminho[linhaAtual][colunaAtual] = 1;  
    (*pontColuna) = colunaAtual;  
    return true;  
}
```

Imagem 8 – Condição de sucesso movimentaEstudante

O algoritmo constantemente chama o método movimentaEstudante recursivamente, a diferença está se o movimento é pra cima, esquerda, direita ou pra baixo, nessa mesma ordem de prioridade.

```
// MOVIMENTA PRA CIMA.  
if (movimentaEstudante(tamLinhas, tamColunas, labirinto, visitado, caminho, linhaAtual-1, colunaAtual, movimentos, qtdChaves, pontColuna)) {  
    caminho[linhaAtual][colunaAtual] = 4;  
    return true;  
}  
  
// Movimenta pra esquerda.  
if (movimentaEstudante(tamLinhas, tamColunas, labirinto, visitado, caminho, linhaAtual, colunaAtual-1, movimentos, qtdChaves, pontColuna)) {  
    caminho[linhaAtual][colunaAtual] = 5;  
    return true;  
}  
  
// Movimento pra direita.  
if (movimentaEstudante(tamLinhas, tamColunas, labirinto, visitado, caminho, linhaAtual, colunaAtual+1, movimentos, qtdChaves, pontColuna)) {  
    caminho[linhaAtual][colunaAtual] = 6;  
    return true;  
}  
  
// Movimenta pra baixo.  
if (movimentaEstudante(tamLinhas, tamColunas, labirinto, visitado, caminho, linhaAtual+1, colunaAtual, movimentos, qtdChaves, pontColuna)) {  
    caminho[linhaAtual][colunaAtual] = 7;  
    return true;  
}  
  
// Retorna false se deu certo para todas as direções, mas não era a coordenada final.  
return false;
```

Imagem 9 – Chamadas recursivas cima, esquerda, direita e baixo

Caso o algoritmo chegue no último return false quer dizer que não encontrou saída para o labirinto.

Além do método movimentaEstudante, o algoritmo contém outros como gerarMatriz que é utilizada para a criação de uma matriz aleatória que posteriormente será salva em um arquivo, e o método imprimeMatriz que faz o que o nome sugere.

```

#endif // FUNCOES_H_INCLUDED

void imprimeMatriz(int tamLinhas, int tamColunas, int **matriz);

bool movimentaEstudante(int tamLinhas, int tamColunas, int **labirinto, int **visitado,
                        int **caminho, int linhaAtual, int colunaAtual, int *movimentos,
                        int qtdChaves, int *pontColuna);

void gerarMatriz (int linhas, int colunas, int portas, int chaves, int dificuldade, char nomeArquivo[20]);
//int labirinto(int l, int c, int mat[l][c], int deltaL[], int deltaC[], int Li, int Ci, int Lf, int Cf);
|

```

Imagem 10 – Funções utilizadas

Sobre o menu foi utilizado alguns outros métodos como o lerArquivo, alocarMatriz e liberarMatriz que será utilizado para armazenar os dados inserido pelo arquivo.

Importante ressaltar que para a resolução do problema foram utilizados 3 matrizes:

```

,
matrizCaminhos = alocarMatriz(valoresMatriz[0], valoresMatriz[1]);
visitado = alocarMatriz(valoresMatriz[0], valoresMatriz[1]);
caminho = alocarMatriz(valoresMatriz[0], valoresMatriz[1]);
free(pontArq, tamanhoString, SEFY SET \.

```

Imagem 11 – Matrizes utilizadas

A matrizCaminhos é simplesmente para armazenar os dados do arquivo e para servir de base de comparações, a visitado é para marcar locais da matriz que já foram visitados, útil para verificar se determinado caminho já foi verificado, evitando loops infinitos no programa. E por fim a caminho, que é usada para salvar o caminho utilizado pelo estudante inteligente, que evita caminhos sem saída.

### 3. Limitações do algoritmo

Não foram implementadas as seguintes funcionalidades opcionais: Modo análise, exibir todos os caminhos possíveis, caminho mais curto, menor número de chaves utilizadas.

### 4. Conclusão

Com isso terminamos o trabalho prático sobre backtracking, trabalho este que teve uma dificuldade considerável, e que nos fez aprender sobre o novo conceito na prática.

Apesar de faltar alguns recursos consideramos os resultados deste trabalho satisfatórios, pois aprendemos a aplicar muitas coisas novas além de aprender novos conceitos.



## 5. Referências

GeeksforGeeks. **Backtracking Algorithms**. Belo Horizonte, MG. Disponível em:  
<<https://www.geeksforgeeks.org/backtracking-algorithms/>>, Acesso em: 18 out. 2019.

UFSC. **Alocação Dinâmica de Vetores e Matriz**. Belo Horizonte, MG. Disponível em:  
<<http://mtm.ufsc.br/~azeredo/cursoC/aulas/ca70.html>>, Acesso em: 18 out. 2019.