UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS DE FLORESTAL
CIÊNCIA DA COMPUTAÇÃO

DANIEL FERNANDES PINHO (2634)
TAIANNE VALERIE A. MOTTA (2679)

TRABALHO PRÁTICO 1

PROJETO E ANÁLISE DE ALGORITMOS

FLORESTAL, MG
12 de abril de 2021

Sumário

1 Introdução 2
2 Desenvolvimento 3
2.1 Explicação do algoritmo projetado 3
2.1.1 Backtracking 3
2.1.2 Explicação 3
2.2 Implementação do algoritmo projetado 4
2.2.1 Menu principal 4
2.2.2 Estruturas de dados criadas 4
2.2.3 Função movimenta-estudante 6
2.2.4 Função Inicializações 9
2.2.5 Modo Análise 15
2.3 Resultados de execução 16
2.4 Arquivos de entrada usados nos testes 19
2.5 Como compilar 21
3 Conclusão 22

1 Introdução

Esse trabalho contém uma implementação em C, da solução de um labirinto. O objetivo é desenvolver nossos conhecimentos sobre Backtracking - matéria vista em sala de aula.

O problema resolvido, consistia em fazer com que estudantes escapem de um labirinto que contém paredes e portas, mas não se sabe se importa o número de tentativas erradas e nem o número de chaves usadas. Se a pessoa voltar atrás em uma porta que você usou uma chave, pode pegar de volta a chave, já que o caminho não levou à saída do labirinto, e então poderá usá-la em outra porta de outro caminho possível do labirinto.

Foi nos dado uma imagem de um labirinto inicial, feito por um dos estudantes do exemplo.

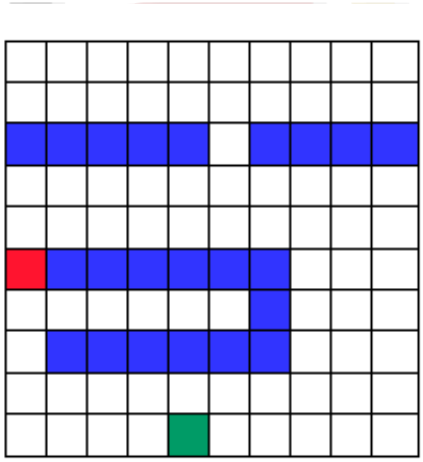


Figura 1 – Figura 1

Nosso algoritmo lê um arquivo com as informações do labirinto onde algum estudante está, bem como sua exata posição inicial. Encontramos, então, um caminho que leve o estudante de sua posição inicial até uma das células da primeira linha da tabela, mostrando na tela cada tentativa de movimento feita e qual o caminho encontrado. O arquivo tem um padrão. Na primeira linha dele é informado três números inteiros. O primeiro corresponde ao número de linhas do labirinto (em forma de matriz); o segundo corresponde ao número de colunas do labirinto (em forma de matriz), e a terceira linha é um inteiro representando o número de chaves que o aluno possui.

2 Desenvolvimento

2.1 Explicação do algoritmo projetado

2.1.1 Backtracking

Backtracking é um refinamento do algoritmo de busca por força bruta (ou enumeração exaustiva), no qual boa parte das soluções podem ser eliminadas sem serem explicitamente examinadas.

Se aplica em:

- Problemas cuja solução pode ser definida a partir de uma seqüência de decisões.
- Problemas que podem ser modelados por uma árvore que representa todas as possíveis seqüências de decisão.

2.1.2 Explicação

Como já foi dito antes, o problema apresentado na especificação do trabalho consiste de um estudante de ciência da computação preso em um labirinto, com o objetivo de encontrar uma saída (que pode ou não existir na primeira linha do labirinto).

O nosso algoritmo possui um arquivo ’main.c’ que é responsável por dar inicio ao programa e nele contém um menu; Um arquivo ’.c’ que contém a implementação das funções; E um arquivo ’.h’ que contém os cabeçalhos e as structs.

E nele temos dois tipos: o modo análise ou não. No não estiver no modo análise a execução será normal. Se estiver no modo análise, deverá fazer tudo, e também contabilizar o número total de chamadas recursivas que foram feitas e o nível máximo de recursividade alcançado durante toda a solução. Além de imprimir tudo na tela somente se o modo análise estiver ligado.

Dessa forma, lemos um arquivo (que contenha o número de linhas, colunas, chaves e portas) e criamos uma função para inicializar esses valores e transformá-los em uma uma matriz com os respectivos valores que tem no arquivo.

2.2 Implementação do algoritmo projetado

2.2.1 Menu principal

O menu foi implementado da forma simples. Não houve adição de extras.

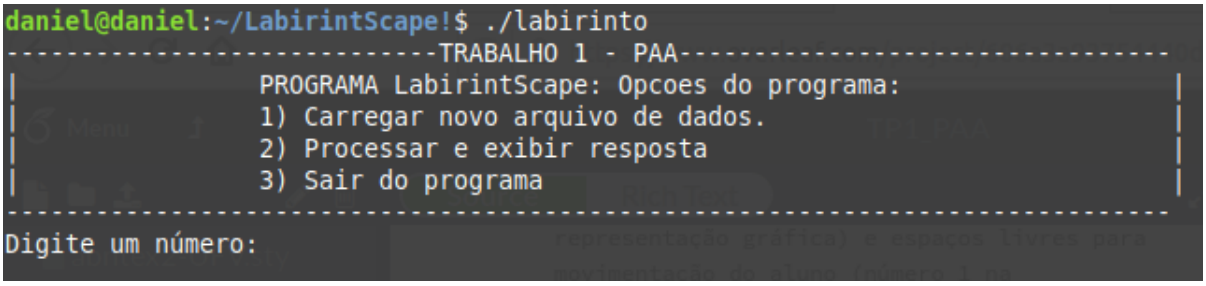


Figura 2 – Resultado da execução para o arquivo de texto 1.

2.2.2 Estruturas de dados criadas

As estruturas criadas foram implementadas no arquivo labirinto.h, como visto a seguir:

```
1     // TP01 PAA - 2020/2 - Daniel Fernandes Pinho (2634) e Taianne Mota
          (2679)
2
3  #ifndef LABIRINTO_H_INCLUDED
4  #define LABIRINTO_H_INCLUDED
5
6  // Coordenadas da posicao do estudante onde x é linha e y coluna
7  typedef struct Posicao{
8      int x;
9      int y;
10 }TipoPosicao;
11
12 //posicoes do estudante em alguns momentos
13 typedef struct Estudante{
14     TipoPosicao primeiraPosicao;
15     TipoPosicao posicaoAtual;
16     TipoPosicao ultimaPosicao;
17 }TipoEstudante;
18
19 //Dados do labrinto
20 typedef struct Labirinto{
21     char **posicionamento;
22     int qtdLinhas;
```

```
23      int qtdColunas;
24      int numeroChaves;
25  }TipoLabirinto;
26
27  // Dados das análise a serem feitas
28  typedef struct ModoAnalise{
29      int chamadasRecursivas;
30      int numeroMovimentos;
31      int qtdMaxChamadasRecursivas;
32      int maxAux;
33  }TipoModoAnalise;
34
35  //escopo das funcoes
36  void aloca_espaco_memoria(TipoLabirinto *Labirinto);
37  int lerArquivo(TipoLabirinto *labirinto, char *nomeArquivo);
38  int movimenta_estudante(TipoEstudante *estudante, TipoLabirinto
        *labirinto, TipoModoAnalise *analise, int
        caminho[labirinto->qtdLinhas][labirinto->qtdColunas], int x, int y,
        int chave[labirinto->qtdLinhas][labirinto->qtdColunas]);
39  int inicializacoes(TipoLabirinto *labirinto, TipoEstudante *estudante,
        TipoModoAnalise *analise, int opcao);
40
41  #endif // LABIRINTO_H_INCLUDED
```

Foram definidas *structs* para implementação das estruturas úteis ao programa. Foi criada a estrutura *ModoAnalise* que tem as variáveis que foram usadas para guardas as informações que são retornadas para o usuário quando o modo análise está ativo. A motivação desta estrutura é somente esta. Nela, a variáveis *chamadasRecursivas* armazena o número de chamadas recursivas, resultado da técnica de *backtracking*; a variável *número de movimentos* armazena o a quantidade de movimentos feito pelo aluno durante a execução; a variável *qtdMaxChamadasRecursivas* armazena o número max de chamadas recursivas e, por fim, a variável *maxAux* é uma variável auxiliar de uso na função *movimentaEstudante*. Na estrutura *Labirinto* é definidas as variáveis que armazenam informações sobre o labirinto. Foi acrescentada uma variável do tipo ponteiro para ponteiro char que evitaram alguns erros na execução do programa, mesmo embora, os dados lidos no arquivo são números do tipo inteiro. E, por fim, foi criada uma estrutura de nome *posicao* que é utilizada para definir as coordenadas das posições, variáveis as quais são utilizadas na estrutura Estudante. Esta última define variáveis relativas à posição inicial, posição final e posição atual do estudante, todas sendo manipuladas no grafo carregado dentro da memória em período de execução do programa. A motivação par a criação desta estruturas se deu na simplicidade de manipulação e na garantia de modularização do programa.

2.2.3 Função movimenta-estudante

O arquivo labirinto.c possui todas as funções implementadas necessárias para obter o resultado esperado. Nele encontra-se a função que movimenta o estudante. Nela foi passado parâmetros do tipo estrutura para estudante, labirinto e analise. X e Y que são as coordenadas do estudante, e a matriz de chaves. Ao se fazer as chamadas recursivas usando a técnica de backtracking, esses parâmetros são fundamentais para que o processo de recuperação da informação anterior seja feito.

```
1  int movimenta_estudante(TipoEstudante *estudante, TipoLabirinto
       *labirinto, TipoModoAnalise *analise, int
       caminho[labirinto->qtdLinhas][labirinto->qtdColunas], int x, int y,
       int chave[labirinto->qtdLinhas][labirinto->qtdColunas]){
2  int i, j, cont = 0, qtdChave;
3  estudante->posicaoAtual.x = x;
4  estudante->posicaoAtual.y = y;
5  analise->maxAux++;
6  analise->chamadasRecursivas++;
7
8  // conta quantas chaves tem na matriz chave
9  for (i = 0; i < labirinto->qtdLinhas; i++) {
10     for (j = 0; j < labirinto->qtdColunas; j++) {
11         if (chave[i][j] == 1) {
12             cont++;
13         }
14     }
15 }
16
17 qtdChave = labirinto->numeroChaves - cont;
18
19 //quando chegar na primeira linha da matriz (labirinto)
20 if (estudante->posicaoAtual.x == 0 && labirinto->posicionamento[x][y]
       != '2'){
21     if (labirinto->posicionamento[x][y] == '3'){ // caso seja uma porta
          verifica se há uma chave suficiente para abri-la
22         if (qtdChave > 0){
23             chave[x][y] = 1;
24             estudante->ultimaPosicao.x = x;
25             estudante->ultimaPosicao.y = y;
26             analise->numeroMovimentos++;
27             printf("Linha: %d Coluna: %d\n",
                   estudante->ultimaPosicao.x+1,
                   estudante->ultimaPosicao.y+1); //imprime a posicao final
                   do estudante
```

```
28              return 1;
29          }
30          return 0;
31      }
32
33      estudante->ultimaPosicao.x = x;
34      estudante->ultimaPosicao.y = y;
35      analise->numeroMovimentos++;
36      printf("Linha: %d Coluna: %d\n", estudante->ultimaPosicao.x+1,
           estudante->ultimaPosicao.y+1);
37      return 1;
38  }
39  // percorre o labirinto sem que repita posicoes ja verificadas
40  if ((x >= 0) && (x < labirinto->qtdLinhas) && (y >= 0) && (y <
       labirinto->qtdColunas) && (labirinto->posicionamento[x][y] != '2')
       && (caminho[x][y] == 0)){
41      caminho[x][y] = 1;
42      analise->numeroMovimentos++;
43      printf("Linha: %d Coluna: %d\n", x+1, y+1); // imprime os movimentos
44      if (labirinto->posicionamento[x][y] == '3'){
45          if (qtdChave > 0){
46              chave[x][y] = 1; // usou uma chave
47              // movimento para cima
48              if (movimenta_estudante(estudante, labirinto, analise,
                   caminho, x - 1, y, chave)){
49                  return 1;
50              }
51              //movimento para a direita
52              if (movimenta_estudante(estudante, labirinto, analise,
                   caminho, x, y + 1, chave)){
53                  return 1;
54              }
55              //movimento para a esquerda
56              if (movimenta_estudante(estudante, labirinto, analise,
                   caminho, x, y - 1, chave)){
57                  return 1;
58              }
59              //movimento para baixo
60              if (movimenta_estudante(estudante, labirinto, analise,
                   caminho, x + 1, y, chave)){
61                  return 1;
62              }
63              chave[x][y] = 0; // 0 para voltar pela porta
```

```
64          }
65          return 0;
66      }
67
68      if (movimenta_estudante(estudante, labirinto, analise, caminho, x -
            1, y, chave)){
69          return 1;
70      }
71
72      if (movimenta_estudante(estudante, labirinto, analise, caminho, x,
            y + 1, chave)){
73          return 1;
74      }
75
76      if (movimenta_estudante(estudante, labirinto, analise, caminho, x,
            y - 1, chave)){
77          return 1;
78      }
79
80      if (movimenta_estudante(estudante, labirinto, analise, caminho, x +
            1, y, chave)){
81          return 1;
82      }
83      return 0;
84  }
85  if (analise->qtdMaxChamadasRecursivas < analise->maxAux){
86      analise->qtdMaxChamadasRecursivas = analise->maxAux;
87  }
88  analise->maxAux = 0;
89  return 0;
90  }
```

Para mover o estudante a função gera tentativas.

Funcionamento das tentativas da função:

- Mover o estudante para cima (x,y+1);
- Mover o estudante para o lado direito (x+1, y);
- Mover o estudante para o lado esquerdo (x-1, y);
- Mover o estudante para baixo (x, y-1)

Para saber como o estudante poderá se movimentar, ou seja, se as tentativas acontecerão, foi utilizado os números de 1 a 3, oferecidos pelo professor:

1. - Vazio (o movimento acontece)
2. - Parede (o movimento não acontece)
3. - Porta (o movimento acontece se o estudante tiver a chave)

Caso o estudante não possa fazer seu movimento, acontecem as chamadas recursivas que vão desempilhando e fazendo o retorno para achar um novo caminho que ele consiga se movimentar. Caso o estudante se depare com uma porta, criamos uma matriz denominada *chave* que conta quantas chaves o estudante tem; E para saber se o estudante já passou por um caminho utilizamos a matriz chamada *caminho*.

2.2.4 Função Inicializações

É nessa função que o programa se inicializa e inicializa as variáveis necessárias para montar o labirinto. Nela é decidido qual função de movimentação será chamada.

```
1  // TP01 PAA - 2020/2 - Daniel Fernandes Pinho (2634) e Taianne Mota
       (2679)
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <time.h>
7  #include "labirinto.h"
8  #define modoAnalise 1
9
10 //funcao para alocar espacco na memoria de acordo com a informação do
       arquivo lido
11 void aloca_espaco_memoria(TipoLabirinto *labirinto){
12     int i;
13     labirinto->posicionamento = (char **) malloc(labirinto->qtdLinhas *
           sizeof(char *));
14     for (i = 0; i < labirinto->qtdLinhas; i++) {
15         labirinto->posicionamento[i] = (char *)
               malloc(labirinto->qtdColunas * sizeof(char));
16     }
17 }
18
19
20 int lerArquivo(TipoLabirinto *labirinto, char *nomeArquivo){
```

```
21      FILE *arquivo;
22      char caminhoArquivo[150], cor;
23      int i, j, linhas, colunas, chaves;
24      strcpy(caminhoArquivo, "/home/daniel/LabirintScape!/");
25      strcat(caminhoArquivo, nomeArquivo);
26      strcat(caminhoArquivo, ".txt");
27      arquivo = fopen(caminhoArquivo, "r");
28      if (arquivo == NULL) {
29          printf("\nErro de leitura do arquivo\n");
30          return 0; //retorna 0 caso nao seja possivel ler o arquivo
31      }else{
32          fscanf(arquivo, "%d %d %d\n", &linhas, &colunas, &chaves); //
               leitura linha a linha
33          // atribuição de valores na estrutura dinâmicamente alocada
34          labirinto->qtdLinhas = linhas;
35          labirinto->qtdColunas = colunas;
36          labirinto->numeroChaves = chaves;
37          aloca_espaco_memoria(labirinto);
38          // leitura do arquivo
39          while (!feof(arquivo)) {
40              for (i = 0; i < labirinto->qtdLinhas; i++) {
41                  for (j = 0; j < labirinto->qtdColunas; j++) {
42                      fscanf(arquivo, "%c ", &cor);
43                      labirinto->posicionamento[i][j] = cor; //preenche o
                           labirinto
44                  }
45              }
46          }
47      }
48      fclose(arquivo);
49      return 1;
50  }
51
52  //função para recursividade do backtracking.
53  int movimenta_estudante(TipoEstudante *estudante, TipoLabirinto
       *labirinto, TipoModoAnalise *analise, int
       caminho[labirinto->qtdLinhas][labirinto->qtdColunas], int x, int y,
       int chave[labirinto->qtdLinhas][labirinto->qtdColunas]){
54      int i, j, cont = 0, qtdChave;
55      estudante->posicaoAtual.x = x;
56      estudante->posicaoAtual.y = y;
57      analise->maxAux++;
58      analise->chamadasRecursivas++;
```

```
59
60      // conta quantas chaves tem na matriz chave
61      for (i = 0; i < labirinto->qtdLinhas; i++) {
62          for (j = 0; j < labirinto->qtdColunas; j++) {
63              if (chave[i][j] == 1) {
64                  cont++;
65              }
66          }
67      }
68
69      qtdChave = labirinto->numeroChaves - cont;
70
71      //quando chegar na primeira linha da matriz (labirinto)
72      if (estudante->posicaoAtual.x == 0 &&
           labirinto->posicionamento[x][y] != '2'){
73          if (labirinto->posicionamento[x][y] == '3'){ // caso seja uma
               porta verifica se há uma chave suficiente para abri-la
74              if (qtdChave > 0){
75                  chave[x][y] = 1;
76                  estudante->ultimaPosicao.x = x;
77                  estudante->ultimaPosicao.y = y;
78                  analise->numeroMovimentos++;
79                  printf("Linha: %d Coluna: %d\n",
                       estudante->ultimaPosicao.x+1,
                       estudante->ultimaPosicao.y+1); //imprime a posicao
                       final do estudante
80                  return 1;
81              }
82              return 0;
83          }
84
85          estudante->ultimaPosicao.x = x;
86          estudante->ultimaPosicao.y = y;
87          analise->numeroMovimentos++;
88          printf("Linha: %d Coluna: %d\n", estudante->ultimaPosicao.x+1,
               estudante->ultimaPosicao.y+1);
89          return 1;
90      }
91
92
93      // percorre o labirinto sem que repita posicoes ja verificadas
94      if ((x >= 0) && (x < labirinto->qtdLinhas) && (y >= 0) && (y <
           labirinto->qtdColunas) && (labirinto->posicionamento[x][y] !=
```

```
        '2') && (caminho[x][y] == 0)){
95          caminho[x][y] = 1;
96          analise->numeroMovimentos++;
97          printf("Linha: %d Coluna: %d\n", x+1, y+1); // imprime os
                movimentos
98
99          if (labirinto->posicionamento[x][y] == '3'){
100             if (qtdChave > 0){
101                 chave[x][y] = 1; // usou uma chave
102                 // movimento para cima
103                 if (movimenta_estudante(estudante, labirinto, analise,
                        caminho, x - 1, y, chave)){
104                     return 1;
105                 }
106                 //movimento para a direita
107                 if (movimenta_estudante(estudante, labirinto, analise,
                        caminho, x, y + 1, chave)){
108                     return 1;
109                 }
110                 //movimento para a esquerda
111                 if (movimenta_estudante(estudante, labirinto, analise,
                        caminho, x, y - 1, chave)){
112                     return 1;
113                 }
114                 //movimento para baixo
115                 if (movimenta_estudante(estudante, labirinto, analise,
                        caminho, x + 1, y, chave)){
116                     return 1;
117                 }
118                 chave[x][y] = 0; // 0 para voltar pela porta
119             }
120             return 0;
121         }
122
123         if (movimenta_estudante(estudante, labirinto, analise, caminho,
                x - 1, y, chave)){
124             return 1;
125         }
126
127         if (movimenta_estudante(estudante, labirinto, analise, caminho,
                x, y + 1, chave)){
128             return 1;
129         }
```

```
130
131         if (movimenta_estudante(estudante, labirinto, analise, caminho,
                x, y - 1, chave)){
132             return 1;
133         }
134
135         if (movimenta_estudante(estudante, labirinto, analise, caminho,
                x + 1, y, chave)){
136             return 1;
137         }
138         return 0;
139     }
140
141
142     if (analise->qtdMaxChamadasRecursivas < analise->maxAux){
143         analise->qtdMaxChamadasRecursivas = analise->maxAux;
144     }
145
146     analise->maxAux = 0;
147     return 0;
148 }
149
150 int inicializacoes(TipoLabirinto *labirinto, TipoEstudante *estudante,
       TipoModoAnalise *analise, int opcao){
151     int i, j, x , y ;
152     int caminho[labirinto->qtdLinhas][labirinto->qtdColunas];
153     int chave[labirinto->qtdLinhas][labirinto->qtdColunas];
154
155     //inicializa posicao
156     for (i = 0; i < labirinto->qtdLinhas; i++) {
157         for (j = 0; j < labirinto->qtdColunas; j++) {
158             if (labirinto->posicionamento[i][j] == '0') {//se a posicao
                    for a do estudante
159                 estudante->primeiraPosicao.x = i;
160                 estudante->primeiraPosicao.y = j;
161                 estudante->posicaoAtual.x = i;
162                 estudante->posicaoAtual.y = j;
163                 estudante->ultimaPosicao.x = 0;
164                 estudante->ultimaPosicao.y = 0;
165                 //x e y sao usados para enviar para as funcoes que
                        movimentam o estudante
166                 x = estudante->posicaoAtual.x;
167                 y = estudante->posicaoAtual.y;
```

```
168             }
169         }
170     }
171
172     //inicializa caminho
173     for (i = 0; i < labirinto->qtdLinhas; i++) {
174         for (j = 0; j < labirinto->qtdColunas; j++){
175             caminho[i][j] = 0;
176         }
177     }
178
179     //inicializa chaves
180     for (i = 0; i < labirinto->qtdLinhas; i++) {
181         for (j = 0; j < labirinto->qtdColunas; j++){
182             chave[i][j] = 0;
183         }
184     }
185
186     //inicializa analise
187     analise->chamadasRecursivas = -1; // -1 para chamada nao recursiva
188     analise->numeroMovimentos = -1; // -1 para dizer que nao ha
        movimento
189     analise->qtdMaxChamadasRecursivas = -1; // -1 para chamada nao
        recursiva pois é a primeira
190     analise->maxAux = 0;
191
192     return (movimenta_estudante(estudante, labirinto, analise, caminho,
        x, y, chave));
193
194 }
```

2.2.5 Modo Análise

Para fazermos o modo análise, definimos uma variável. Se o modo análise estiver ativo, a variável é definida como ’1’ e o programa irá retornar a quantidade de movimentos feitas e a posição final do usuário. Além disso, irá retornar todos os números de chamadas recursivas que foram realizadas e também qual foi o nível máximo de recursão atingido.

Se a variável análise é diferente de ’1’, o programa funcionará sem o relato da recursividade, mostrando apenas a quantidade de movimentos realizadas e a posição final do usuário.

Para salvarmos os dados do modo análise criamos uma struct ModoAnalise que contém uma variável para o número de chamadas recursivas, uma para o número de movimentos e uma para a quantidade de chamadas recursivas feitas.

```
1    typedef struct ModoAnalise{
2        int chamadasRecursivas;
3        int numeroMovimentos;
4        int qtdMaxChamadasRecursivas;
5        int maxAux;
6    }TipoModoAnalise;
```

2.3 Resultados de execução

O primeiro arquivo de teste foi executado e obteve-se 31 movimentos, com um total de 89 chamadas recursivas e número máximo de recursão igual a 7.

```
Digite um número:2
Linha: 10 Coluna: 5
Linha: 9 Coluna: 5
Linha: 9 Coluna: 6
Linha: 9 Coluna: 7
Linha: 9 Coluna: 8
Linha: 8 Coluna: 8
Linha: 7 Coluna: 8
Linha: 6 Coluna: 8
Linha: 5 Coluna: 8
Linha: 4 Coluna: 8
Linha: 4 Coluna: 9
Linha: 4 Coluna: 10
Linha: 5 Coluna: 10
Linha: 5 Coluna: 9
Linha: 6 Coluna: 9
Linha: 6 Coluna: 10
Linha: 7 Coluna: 10
Linha: 7 Coluna: 9
Linha: 8 Coluna: 9
Linha: 8 Coluna: 10
Linha: 9 Coluna: 10
Linha: 9 Coluna: 9
Linha: 10 Coluna: 9
Linha: 10 Coluna: 10
Linha: 10 Coluna: 8
Linha: 10 Coluna: 7
Linha: 10 Coluna: 6
Linha: 4 Coluna: 7
Linha: 4 Coluna: 6
Linha: 3 Coluna: 6
Linha: 2 Coluna: 6
Linha: 1 Coluna: 6

Modo analise ativo

O estudante se movimentou 31 vezes e chegou na coluna 5 da primeira linha
Numero total de chamadas recursivas: 89
Nivel maximo de recursao: 7
```

Figura 3 – Resultado da execução para o arquivo de texto 1.

O segundo arquivo de teste foi executado e obteve-se 19 movimentos, com um total de 43 chamadas recursivas e número máximo de recursão igual a 5.

```
Digite um número:2
Linha: 6 Coluna: 1
Linha: 5 Coluna: 1
Linha: 4 Coluna: 1
Linha: 3 Coluna: 1
Linha: 6 Coluna: 2
Linha: 6 Coluna: 3
Linha: 5 Coluna: 3
Linha: 5 Coluna: 4
Linha: 5 Coluna: 5
Linha: 6 Coluna: 5
Linha: 6 Coluna: 6
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: 3 Coluna: 4
Linha: 2 Coluna: 4
Linha: 1 Coluna: 4

Modo analise ativo

O estudante se movimentou 19 vezes e chegou na coluna 3 da primeira linha
Numero total de chamadas recursivas: 43
Nivel maximo de recursao: 5
```

Figura 4 – Resultado da execução para o arquivo de texto 2.

Já o terceiro arquivo de teste foi executado e obteve-se 53 movimentos, com um total de 216 chamadas recursivas e número máximo de recursão igual a 8, sem ser possível encontrar uma saída.

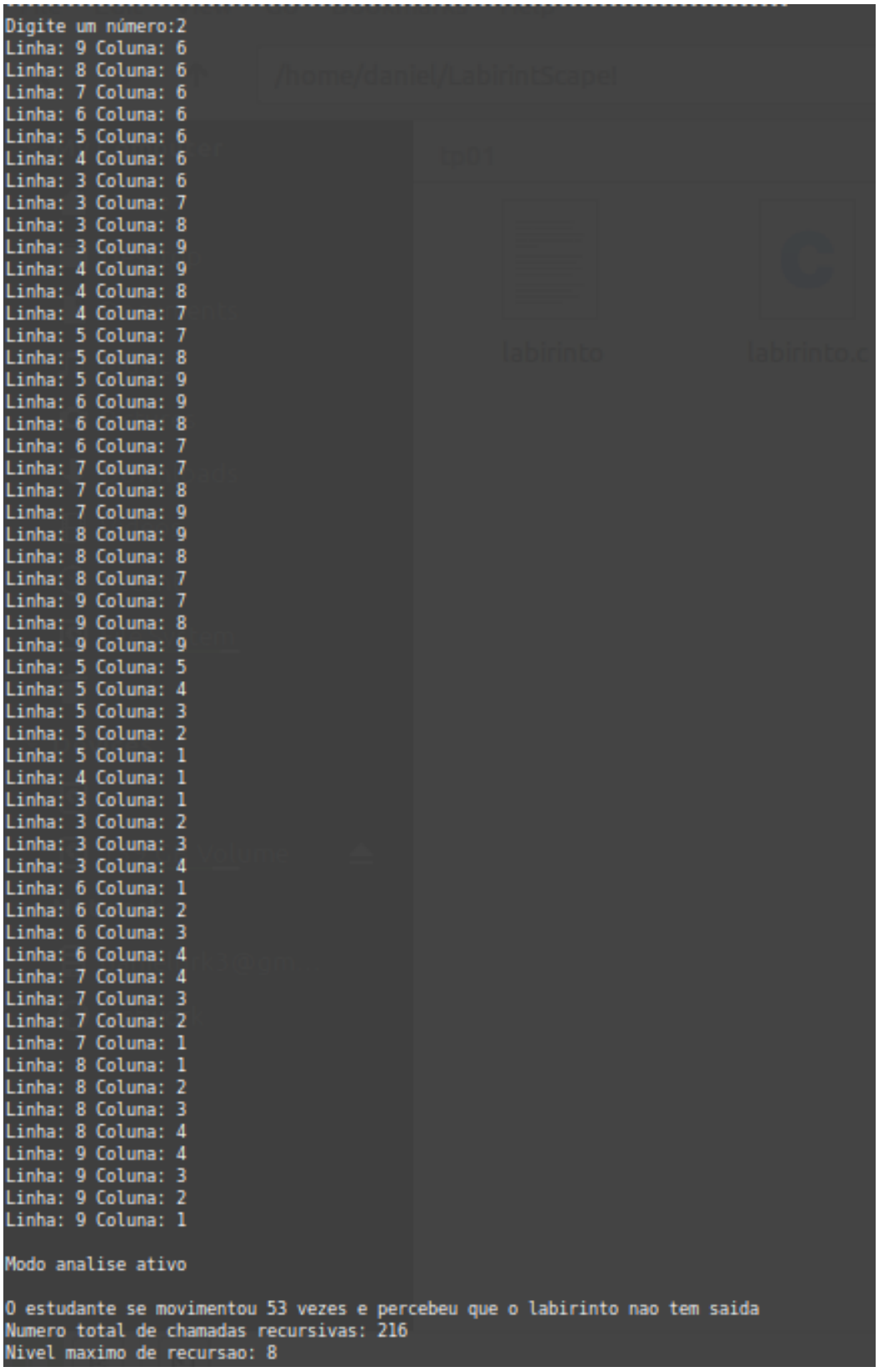


Figura 5 – Resultado da execução para o arquivo de texto 3.

2.4 Arquivos de entrada usados nos testes

Foram criados três arquivos (extensão .txt) de entrada para execução de testes do programa. O primeiro arquivo está descrito na documentação deste trabalho e chamamos de labirinto1. Consiste num exemplo simples, onde o labirinto há uma saída. Vale ressaltar que, o programa também cria labirintos matriciais não quadráticos. Isso se dá a partir das definições de linhas e colunas nos arquivos texto de entrada. A seguir, o arquivo criado e a representação gráfica do labirinto que ele forma (numa matriz, em tempo de execução do programa):

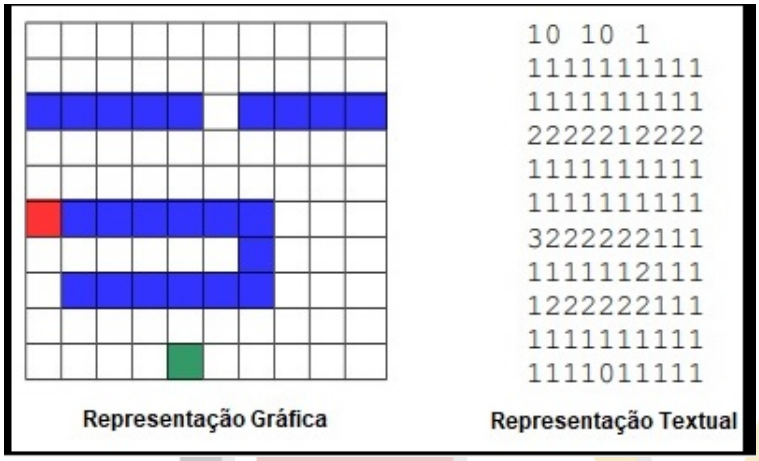
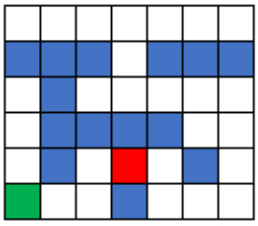


Figura 6 – Arquivo de teste 1, de nome labirinto1.txt e sua representação gráfica

No segundo arquivo, de nome labirinto2.txt, criamos um teste intermediário, onde formamos um labirinto não quadrático, com uma porta (número 3 na representação em forma de arquivo de texto, cor vermelho na representação gráfica), paredes (número 2 na representação em forma de arquivo texto, cor azul na representação gráfica) e espaços livres para movimentação do aluno (número 1 na representação em forma de arquivo texto, cor branca na representação gráfica). A seguir, um *screenshot* deste arquivo teste criado.



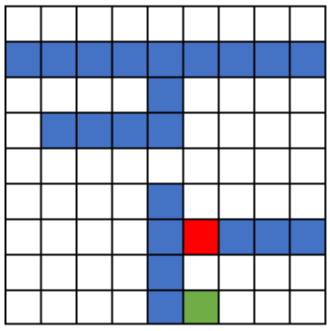
Representação gráfica

6 7 1
1111111
2221222
1211111
1222211
1213121
0112111

Representação textual

Figura 7 – Arquivo de teste 2, de nome labirinto1.txt e sua representação gráfica

No terceiro arquivo, de nome labirinto3.txt, criamos um teste mais avançado. O labirinto não há saída.



9 9 2
111111111
222222222
111121111
122221111
111111111
111121111
111123111
111121111
111120111

Representação textual

Figura 8 – Arquivo de teste 3, de nome labirinto1.txt e sua representação gráfica

2.5 Como compilar

O pré requisito para compilação e execução do programa é ter um compilador de linguagem C instalado na máquina. Para este trabalho foi utilizado o compilador GCC. Para compilar e executar o programa a partir deste compilador no ambiente Linux, basta executar os dois comandos a seguir no terminal:

```
1    gcc labirinto.c -o labirintScape // compilando
2    ./labirintScape                  // executando
```

Para execução no ambiente Windows, também se faz necessário a instalação de um compilador de linguagem C. Os seguintes comandos, feitos via terminal, compilam e executam o programa (para exemplificar será usado comandos compatíveis ao compilador MinGW [o mesmo que GCC]):

```
1    gcc labirinto.c -o labirintScape // compilando
2    labirintScape                    // executando
```

Em seguida, será gerado um arquivo compilado de nome *labirintScape*, e a interação gráfica será via terminal.

3 Conclusão

Com este trabalho conseguimos além reforçar os conceitos aprendidos em aula, colocamos na prática, o funcionamento do algoritmo backtracking e suas facilidades em resoluções de problemas. Dessa forma, conseguimos assimilar muito melhor os conceitos.

Portanto, concluímos que o trabalho foi de grande importância para incrementar em nosso conhecimento.