



UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS FLORESTAL
CURSO CIÊNCIA DA COMPUTAÇÃO

RICARDO SPÍNOLA 03471

LUCAS GABRIEL 03493

PROJETO E ANÁLISE DE ALGORITMOS
TRABALHO PRÁTICO - 1: LABIRINTO

Florestal, MG

2019

Introdução	3
Desenvolvimento	3
Funções implementadas	4
Função checar	4
Função movimentaEstudante	5
Função procuraSolucao	8
Função A_movimentaEstudante	9
Funções auxiliares	10
Funções de exibição da saída	10
Função Leitura de Arquivo - entradaArquivo	12
Como usar	12
Conclusão	16

1. Introdução

O trabalho proposto pelo professor Daniel Mendes, tem com objetivo implementar um programa que utiliza a metodologia de backtracking para tentar achar a saída de um labirinto.

O trabalho se mostra desafiador uma vez que envolve recursividade, logo para encontrar um erro na lógica deve-se por um esforço maior.

2. Desenvolvimento

Inicialmente foi criada uma função denominada *movimentaEstudante* com a finalidade de realizar cada movimento do estudante, porém o grupo percebeu que deveria haver uma checagem para ver de que tipo seria a próxima “casa” a ser percorrida pelo estudante, ou seja, se seria uma célula que contém uma chave, uma parede, uma porta, ou apenas um caminho livre.

2.1. Funções implementadas

Nesta parte daremos ênfase na explicação de algumas funções importantes em nosso código.

2.1.1. Função checar

```
bool checar(int N, int M, int x, int y, int lab[N][M], int *keys) {  
    if(x >= 0 && x < N && y >= 0 && y < M && lab[x][y] == 1){  
        return true;  
    }  
    else if(lab[x][y] == 0){  
        return true;  
    }  
    else if(x >= 0 && x < N && y >= 0 && y < M && lab[x][y] == 3){  
        if (*keys > 0){  
            *keys = *keys - 1;  
            return true;  
        }  
        else{  
            return false;  
        }  
    }  
    else if(x >= 0 && x < N && y >= 0 && y < M && lab[x][y] == 4){  
        *keys = *keys + 1;  
        return true;  
    }  
    else{  
        return false;  
    }  
}
```

Imagem 1: função checar

A função checar funciona da seguinte forma, inicialmente é passado como parâmetro as dimensões do labirinto e a posição atual do estudante, a qual queremos checar, juntamente com a matriz labirinto e o número de chaves que o estudante possui, então é checado se a célula a qual passamos é igual a 1, caso seja, a função *checar* retorna *true* pois se trata de um caminho livre, caso a posição seja igual a 0 a função novamente retorna *true* pois se trata da posição inicial do estudante, porém caso a posição seja igual a 3(uma porta) é feita uma checagem no número de chaves do estudante, caso o número de chaves do estudante seja

superior a zero então a função decrece o número total de chaves do estudante e retorna *true*, caso o estudante não possua chaves a função retorna *false*. Por último, caso a posição seja igual a 4(possui uma chave no chão) é acrescentado uma chave ao número total de chaves que o estudante possui.

2.1.2. Função *movimentaEstudante*

```
bool movimentaEstudante(int N, int M, int x, int y, int solucao[N][M], int lab[N][M], int *keys, int *movimentos, int *colunaFim){
    if(x == 0 && checar(N,M,x,y,lab,keys)){
        *colunaFim = y;
        solucao[x][y] = 1;
        return true;
    }
    if(checar(N,M,x,y,lab,keys)){
        solucao[x][y] = 1;
        *movimentos+=1;
        if(solucao[x-1][y] != 1) {
            if(movimentaEstudante(N,M,x-1,y,solucao,lab,keys,movimentos,colunaFim)){
                return true;
            }
        }
        if(solucao[x][y+1] != 1) {
            if(movimentaEstudante(N,M,x,y+1,solucao,lab,keys,movimentos,colunaFim)){
                return true;
            }
        }
        if(solucao[x][y-1] != 1) {
            if(movimentaEstudante(N,M,x,y-1,solucao,lab,keys,movimentos,colunaFim)){
                return true;
            }
        }
        if(solucao[x+1][y] != 1){
            if(movimentaEstudante(N,M,x+1,y,solucao,lab,keys,movimentos,colunaFim)){
                return true;
            }
        }
        if(lab[x][y]==3){ //ADICIONAR CHAVE AO USUARIO AO VOLTARMOS POR UMA PORTA QUE HAVIAMOS ABERTO
            *keys = *keys+1;
        }
        if(lab[x][y]==4){ //DEVOLVER CHAVE AO RETORNAR PELA CELULA QUE HAVIAMOS PEGADO A CHAVE
            *keys = *keys-1;
        }
        solucao[x][y] = 0;
        *movimentos+=1;
        return false;
    }
    return false;
}
```

Imagem 2: função *movimentaEstudante*

A função *movimentaEstudante* tem como parâmetros as dimensões do labirinto (N e M) a posição atual do estudante (x e y) uma matriz solução que irá conter a solução encontrada caso haja uma, a matriz labirinto, o número de chaves do estudante, o número de movimentos necessários até que fosse encontrado a saída e uma variável responsável por indicar em qual coluna a saída foi encontrada.

Inicialmente é checado se x (linha atual do estudante) é igual a zero (pois a linha zero indica a saída) e se a função checar com os parâmetros atuais retorna *true*, caso ambas as condições sejam cumpridas a variável *colunaFim* recebe y(coluna atual do estudante) e a matriz solução na posição atual recebe 1(indicar o caminho percorrido) então é retornado pela função *true* indicando que a saída foi encontrada.

Caso as condições do primeiro condicional não sejam cumpridas, então ainda estamos procurando pela saída, logo checamos se a posição atual do estudante é válida através da função *checar*, caso seja(ou seja caso a função checar retorne *true*)chamamos a função *movimentaEstudante* para todas as configurações, isto é, cima, direita, esquerda e baixo, logo após checarmos se a matriz solução naquela posição possui o valor igual a 1 como indicado em vermelho na *imagem 3*(pois isso indicaria que estaríamos retornando para uma célula já visitada então permaneceríamos em um loop avançando e recuando a célula).

```
if(solucao[x-1][y] != 1) {  
    if(movimentaEstudante(N,M,x-1,y,solucao,lab,keys,movimentos,colunaFim)) {  
        return true;  
    }  
}  
if(solucao[x][y+1] != 1) {  
    if (movimentaEstudante(N,M,x, y + 1, solucao, lab,keys,movimentos,colunaFim)) {  
        return true;  
    }  
}  
if(solucao[x][y-1] != 1) {  
    if (movimentaEstudante(N,M,x, y - 1, solucao, lab,keys,movimentos,colunaFim)) {  
        return true;  
    }  
}  
if(solucao[x+1][y] != 1){  
    if(movimentaEstudante(N,M,x+1, y, solucao, lab, keys,movimentos,colunaFim)){  
        return true;  
    }  
}
```

Imagem 3: condicional de repetição de caminho

Então as chamadas são empilhadas até que atinjam o nível mais interno, caso ao desempilhar ela entre em algum dos condicionais o retorno da função naquele nível será *true*, por outro lado caso ela não entre em nenhum dos condicionais checamos se aquela posição no labirinto se trata de uma porta ou uma célula com uma chave.

```

if(lab[x][y]==3){ //ADICIONAR CHAVE AO USUARIO AO VOLTARMOS POR UMA PORTA QUE HAVIAMOS ABERTO
    *keys = *keys+1;
}
if(lab[x][y]==4){ //DEVOLVER CHAVE AO RETORNAR PELA CELULA QUE HAVIAMOS PEGADO A CHAVE
    *keys = *keys-1;
}
solucao[x][y] = 0;
*movimentos+=1;
return false;

```

Imagem 4: condicional para checar se se trata de uma porta ou célula com chave

Tratar-se-á de uma porta, devolveremos a chave ao estudante($*keys = *keys + 1$), por outro lado, caso se trate de uma célula com uma chave colocaremos a chave novamente no chão($*keys = *keys - 1$) diminuindo a quantidade de chaves do estudante e a matriz solução naquela posição será preenchida com zero($solucao[x][y] = 0$) indicando que aquela casa não está contida no caminho solução e retornando *false*, uma vez que o último desempilhamento gere o retorno *false* para a função *movimentaEstudante* é concluído que o labirinto não possui solução.

2.1.3. Função procuraSolucao

A função *procuraSolucao* é a função responsável por realizar a chamada inicial da função *movimentaEstudante* e *A_movimentaEstudante* e retornar o resultado da função sendo *true* ou *false*, caso o resultado geral da chamada seja *true* a função *procuraSolucao* chamará uma função auxiliar denominada *imprimeSolucao* responsável por imprimir a matriz do caminho resultante que leva até a saída, caso o resultado geral seja *false* a função alertará que o labirinto em questão não possui saída.

```

bool procuraSolucao(int N, int M, int maze[N][M], int posInicial, int *keys, bool analise)
{
    int solucao[N][M];
    int movimentos = 0;
    int nivelRecursao = 0;
    int aux = 0;
    int colunaFim = 0;
    int tRecursivo = -1;
    for(int i = 0; i < N; i++){
        for(int j = 0; j < M; j++){
            solucao[i][j] = 0;
        }
    }
    if(analise){
        if (!A_movimentaEstudante(N,M,N-1,posInicial, solucao, maze, keys, &tRecursivo, &aux, &nivelRecursao,&movimentos,&colunaFim)) {
            printf(" Já percorri essa de uns %d metros, não consigo encontrar a saída !\n");
            printf(" Devia ter ficado em casa, jogando Megamania !\n");
            return false;
        }
    }
    else{
        if (!movimentaEstudante(N,M,N-1,posInicial, solucao, maze, keys,&movimentos,&colunaFim)) {
            printf("Solution doesn't exist");
            return false;
        }
    }
    printf("SOLUÇÃO DO LABIRINTO \n\n");
    imprimeSolucao(N,M,solucao);
    printf("\n O maior nível de recursão atingido foi de: %d\n\n", nivelRecursao);
    printf("\n Me movimenteí uns %d metros para encontrar essa saída.",movimentos);
    printf("\n Uhn, estranho ! Está escrito, coluna %d.\n",colunaFim);
    return true;
}

```

Imagem 5: função procuraSolucao

Como mostrado na *Imagem 5* ela recebe como parâmetros as dimensões do labirinto(N e M), a matriz labirinto(maze[N][M]), a posição inicial(posInicial) do estudante o número de chaves inicial(*keys) é uma variável booleana que definirá se o programa rodará no modo análise ou não (bool análise).

2.1.4. Função A_movimentaEstudante

De modo semelhante a execução da função movimentaEstudante descrita acima, esta adaptação irá acrescentar o modo análise.

Como parâmetros, irá ser acrescentado: número de recursos totais e nível máximo da recursividade(chamada mais profunda).

2.2. Funções auxiliares

Breve descrição de funções implementadas para facilitar o entendimento do código.

2.2.1. Funções de Exibição da Saída

A amostragem do labirinto fica responsável pelas funções: *imprimeSolucao*, *imprimeMat* que formatam a saída como mostra a imagem seguinte.

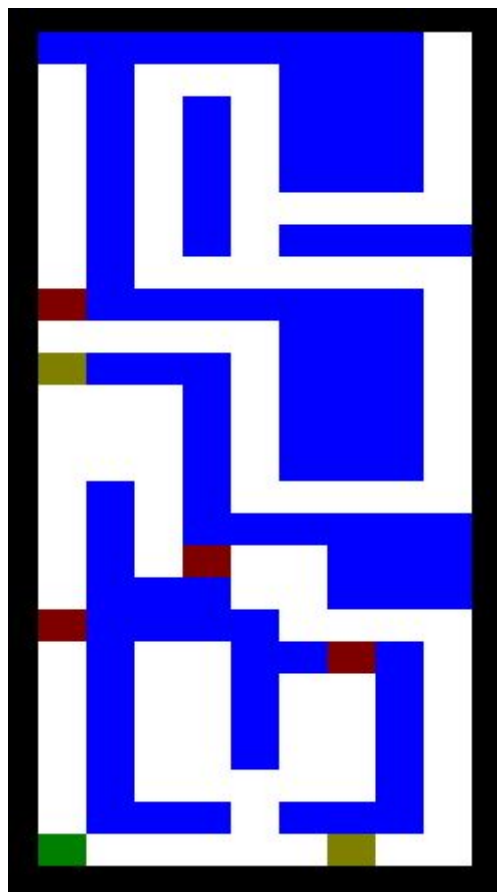


Imagem 6: Saída do labirinto no terminal

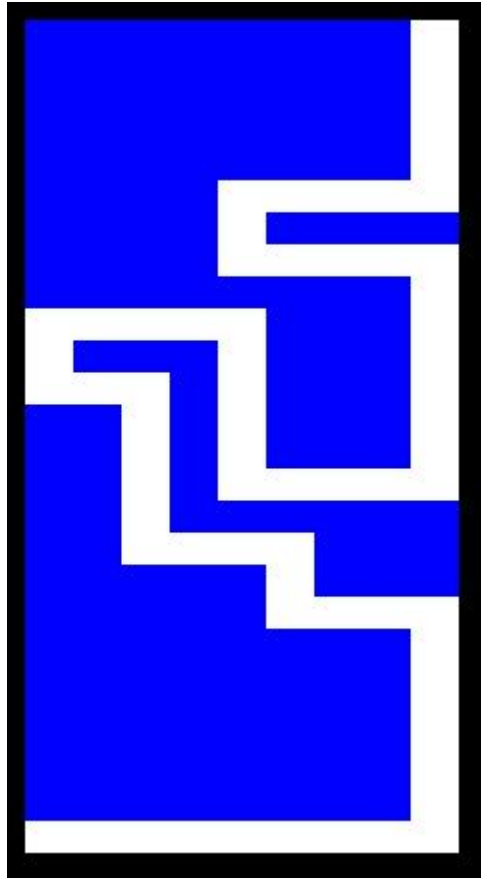


Imagem 7: Saída do labirinto solucionado no terminal

Para exibição do menu, criamos a função *menu*.

```
PROGRAMA Labirinto: Opções do programa:  
[1] Carregar novo arquivo no drone  
[2] Processar e exibir possível saída  
[3] Desligar drone  
[0] Limpar monitor
```

Imagem 8: Menu do programa

2.2.2 Função Leitura de Arquivo - *entradaArquivo*

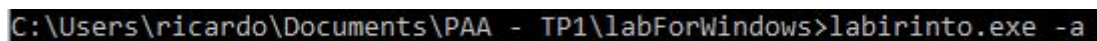
Função implementada para que, a partir de um arquivo de texto, pudesse obter dados do labirinto, como chaves, linhas e colunas.

Assim como especificado na documentação do trabalho proposto, fizemos nossa execução.

Funções auxiliares foram implementadas para facilitar a codificação do trabalho, como: *programa* -reuni todas informações e inicia o *procuraSolução*-, *convertMatriz* - converte uma matriz definida como ponteiros para matriz definida sem ponteiros-.

3. Como usar

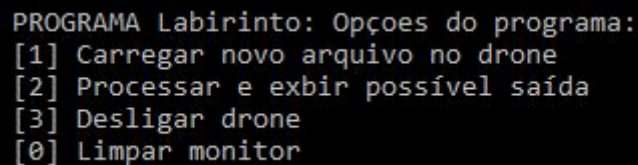
Primeiramente deve-se escolher o modo, para isso, basta executar o programa *labirinto.exe* com o parâmetro *-a*, caso queira em modo análise. Caso contrário, poderá executar sem nenhum parâmetro.



```
C:\Users\ricardo\Documents\PAA - TP1\labForWindows>labirinto.exe -a
```

Imagem 9: Modo de execução em análise

Seguido à isso, será apresentado o menu.



```
PROGRAMA Labirinto: Opções do programa:  
[1] Carregar novo arquivo no drone  
[2] Processar e exibir possível saída  
[3] Desligar drone  
[0] Limpar monitor
```

Imagem 10: Menu do programa

- 1 - Definir um arquivo para leitura (*arquivo.txt*)
- 2 - Iniciar a descoberta por um caminho.
- 3 - Fechar o programa
- 4 - Limpar saída e mostrar novamente o menu.

Depois de processado o caminho, mostrará a saída da seguinte forma:

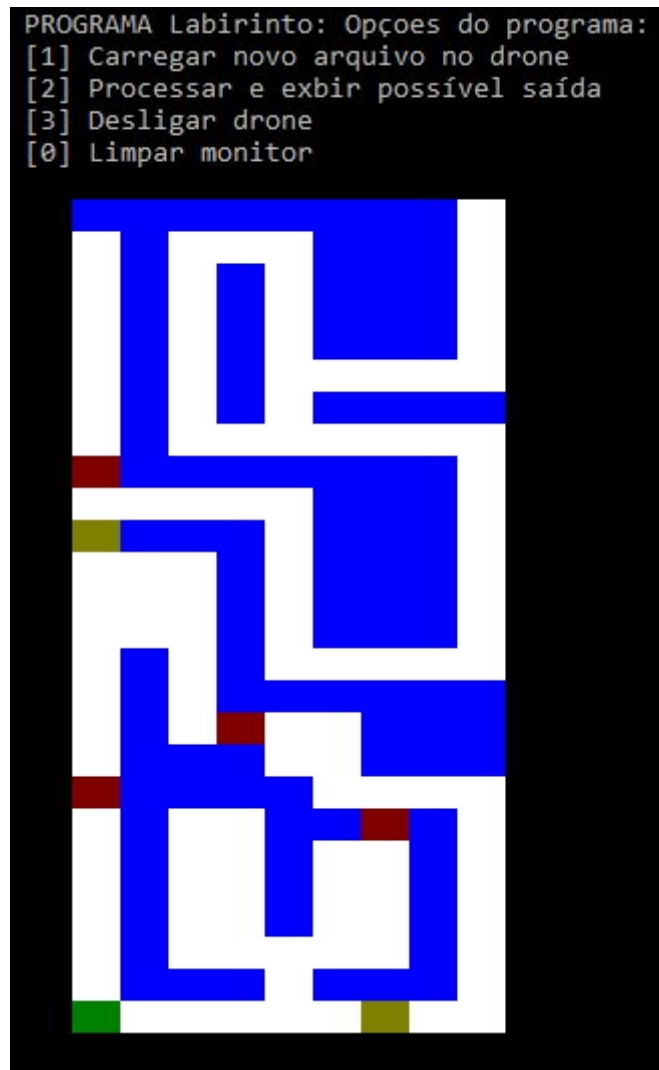


Imagem 11: Saída para o arquivo.txt

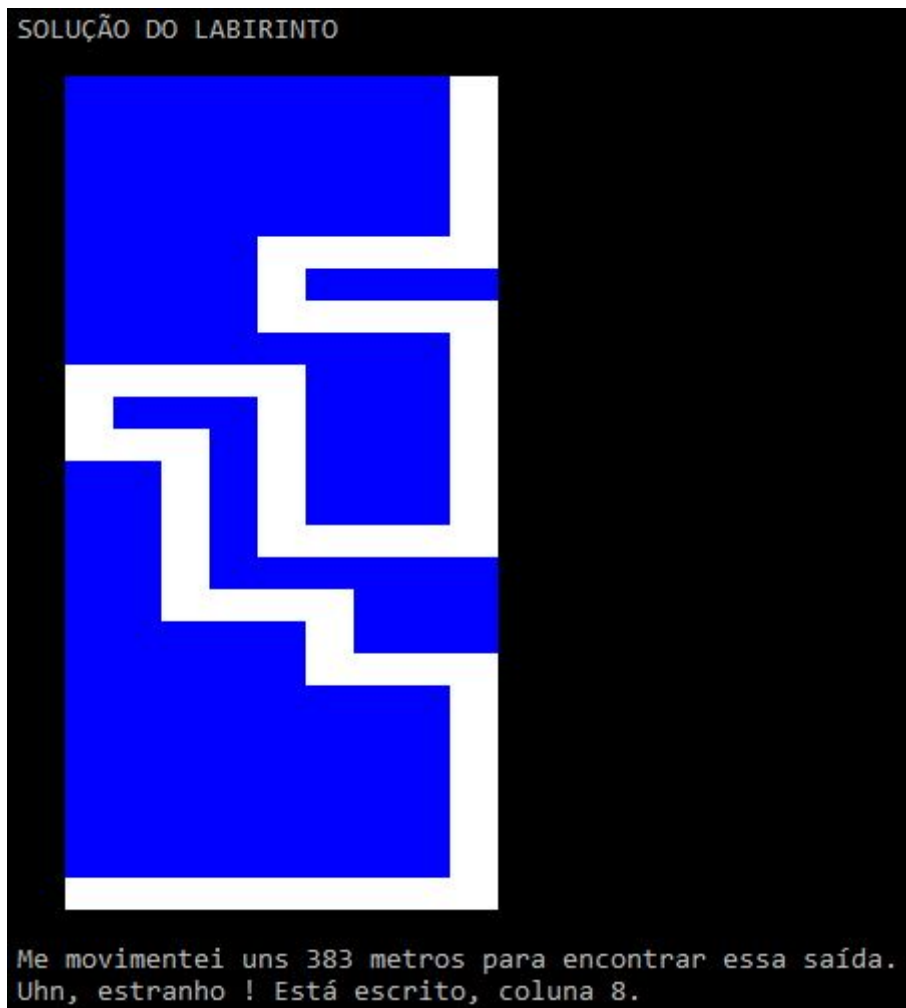


Imagem 12: Saída para o arquivo.txt - Part2

O arquivo de teste escolhido tem a seguinte configuração:

26 9 0

222222221

121112221

121212221

121212221

121212221

121211111

121212222

121111111

322222221

111112221

422212221

111212221

111212221

111212221

121211111

121222222

121311222

122211222

322221111

121122321

121121121

121121121

121121121

121111121

122212221

011111411

4. Conclusão

A Partir desse trabalho pudemos perceber as vantagens de se utilizar backtracking em relação a brute-force, uma vez que, pelo método de backtracking é possível realizar a diminuição de vezes que o programa necessita rodar.

Foi interessante utilizar o método que foi estudado em sala, de maneira teórica, em uma abordagem prática.