Algoritmo de Resolução de Labirintos

Luis Alexandre Ferreira Bueno e Vitor Bruno de Oliveira Barth

December 14, 2016

Abstract

Este trabalho tem por objetivo apresentar um algoritmo, que, por meio dos conceitos de Fila, encontre um caminho entre dois pontos de um labirinto.

1 Introdução

Nos foi apresentado o seguinte desafio: "Elaborar um algoritmo que encontre o caminho entre dois pontos de um labirinto, gerando imagens dele não resolvido e resolvido".

Tivemos de usar como base o conceito de Fila, ou seja, uma estrutura de armazenamento de dados do tipo FIFO (First In, First Out).

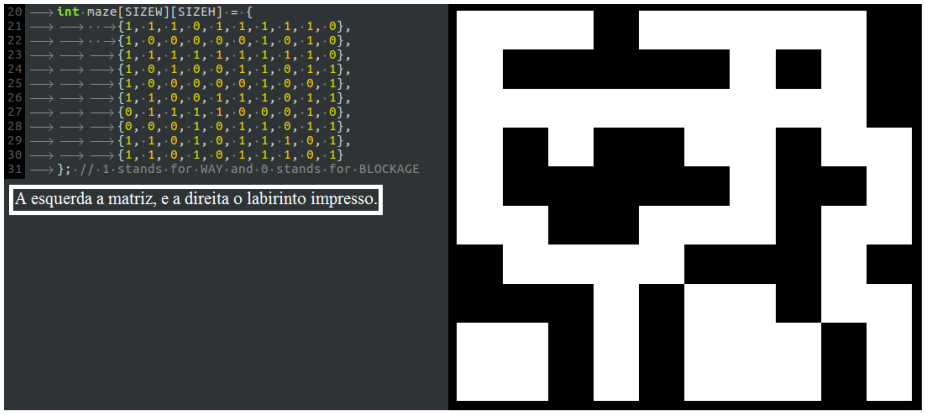
O labirinto foi inserido no programa na forma de uma matriz de tamanho variável, onde o valor (x, y) = 1 representa um caminho e (x, y) = 0 representa um obstáculo. Após ser resolvido, o caminho será marcado por (x, y) = -1.

Trabalhar com imagens pode ser relativamente difícil, e por isso escolhemos utilizar o formato Netpbm (extensões .pbm ou .ppm), qual usa o padrão ASCII para salvar as informações de cor.

```
P1
# This is an example bitmap of the letter "J"
6 10
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
1 0 0 0 1 0
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

Nosso programa é escrito na linguagem C, e possui duas funcões principais: a writePPM, que recebe uma matriz e gera uma imagem de resolução variável

no formato .ppm e a solveMaze, que recebe o labirinto, as posições inicial e final, e encontra um caminho que ligue estes dois pontos.



2 Geração de Imagem

Para gerar imagens em .ppm usamos a função writePPM, que possui quatro variáveis principais:

- resolutionWidth e resolutionHeight que armazenam, respectivamente, largura e altura (em pixels) da imagem a ser gerada.

- blockSizeWidth e blockSizeHeight que armazenam, respectivamente, a largura e altura (em pixels) de cada bloco do labirinto, sendo ele proporcional ao tamanho do labirinto e resolução da imagem a ser gerada.

A parte mais importante desta função é composta de quatro loops for encadeados. Os dois mais externos servem para iterar as linhas da imagem, e os dois mais internos servem iterar as colunas da imagem, e deste modo gerar cada bloco da linha.

Logo, se o valor do bloco a ser gerado é 0, este será pintado da cor preta, código RGB 0 0 0, e caso seja de valor 1, será pintado de branco, código RGB 255 255 255. E, para marcar o caminho, o bloco será parcialmente pintado de vermelho, RGB 255 0 0.

```
for (int i = 0; i < SIZEH; i++) // ITERATES MAZE LINES
  for (int j = 0; j < blockSizeHeight; j++) // ITERATES EACH LINE PIXELS HEIGHT
    for (int k = 0; k < SIZEW; k++) // ITERATES MAZE COLUMNS
          for (int l = 0; l < blockSizeWidth; l++) // ITERATES EACH COLUMN PIXEL WIDTH
                if (maze[i][k] == 0)
                  fprintf(image, "0 0 0  "); // PAINT WALL
                else if (maze[i][k] == -1 &&
                        (j > blockSizeHeight/3 && j < 2*blockSizeHeight/3
                        || l > blockSizeHeight/3 && l < 2*blockSizeHeight/3))
                  fprintf(image, "1 0 0  "); // PAINT CROSS IN WAY OUT
                else
                  fprintf(image, "1 1 1  "); // PAINT WAY
```

3 Resolução do Labirinto

Para achar os caminhos do ponto A ao ponto B préviamente estabelecidos, inicialmente enfileiramos o ponto inicial e utilizamos a seguinte ideia:

a) verifica-se as posições ao redor da posição atual

b) caso ela seja um caminho possivel, enfilera-se essa nova posição

c) define-se o valor da nova posição como posição atual + 1

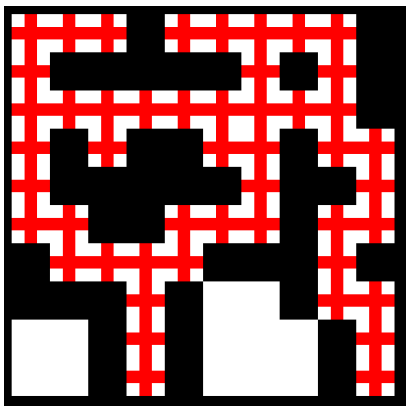
d) depois de verificadas todas as posições ao redor, desenfilera-se a posição atual

e) tratamos a primeira posição enfileirada, como posição atual

Esse código é repetido até que a posição atual seja o ponto onde desejamos chegar. Veja no pedaço do código abaixo:

```
while (!wayOut) {
        if (pos.x==finalPosition[0] && pos.y==finalPosition[1]) {
                wayOut = 1;
        }
        else {
                if (pos.x < SIZEW-1 && maze[pos.x+1][pos.y]==1) {
                        enqueue(&queue,(coordXY){pos.x+1,pos.y});
                        maze[pos.x+1][pos.y]=way+1;
                } // Usa-se essa mesma condição para os pontos x-1, y+1 e y-1
        }
```

Após encontrar o final, temos como resultado todos os caminhos possíveis, contudo nem todos levam ao ponto desejado.

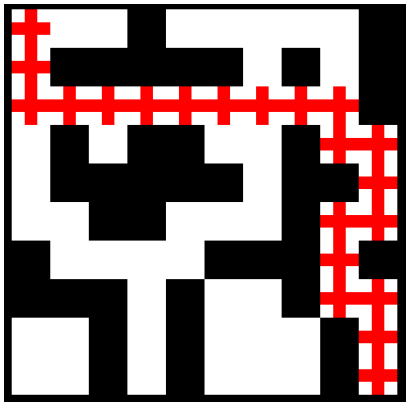


Para que seja mostrado apenas o(s) caminho(s) mais curto(s), tratamos o ponto final como o inicial, e percorremos enfileirando o caminho inverso, substituindo o valor da posição por -1, conforme mostra o código abaixo.

```
while (!wayOut) {
        coordXY pos = dequeue(&queue);
        if (pos.x < SIZEW-1 && maze[pos.x+1][pos.y]==maze[pos.x][pos.y]-1)
                enqueue(&queue,(coordXY){pos.x+1,pos.y});
```

```
        //  Usa-se essa mesma condição para os pontos x-1, y+1 e y-1

        maze[pos.x][pos.y]=-1;
}
```



E caso sejam percorridas todas as posições possíveis e não tenhamos encontrado um caminho de A para B, será mostrada uma mensagem de erro.

4 Conclusão

Para chegar ao resultado esperado tivemos algumas dificuldades. O esquema de linha/coluna em C é invertido, logo precisamos fazer algumas adaptações para que ele funcionasse no formato matriz[x][y], e não matriz[y][x].

Além disso, o reconhecimento de múltiplos caminhos é um tanto quanto confuso, afinal seguindo a lógica por nós escrita, caso dois caminhos possuam a mesma distância, ambos serão marcados.

Para que fosse otimizado o código, foi necessária uma pequena adaptação na biblioteca padrão de filas. Criamos a função empty(), que limpava a fila sem precisar que a reinicializássemos.

Chegar no código que cria imagens PPM também foi cansativo, pois para permitir o uso de labirintos de tamanho variável e imagens de resolução dinâmica tivemos de trabalhar com proporcionalidade entre blocos e pixels, afinal arquivos deste tipo e tamanho (uma imagem PPM em 720p possui cerca de 5MB em linguagem ASCII) não permitem multiplas edições por conta do tempo gasto de processamento.

Contudo alcançamos o resultado esperado, a não ser que seja inserido um labirinto demasiado complexo, ou resoluções fora do padrão.