Maze Solver

Júlio César Guimarães Costa – RA:2203049 – Organização de Computadores

Resumo – O relatório a seguir apresenta um trabalho desenvolvido pelo autor pertencente a disciplina de Organização de Computadores na UTFPR-PG que utiliza a linguagem de Assembly para a Resolução de Labirintos.

Palavras Chave – Organização de Computadores, Labirintos, Assembly MIPS, Vetores, PGM.

---

1 – Introdução

Neste projeto devemos primeiramente imaginar um mapa complexo com uma entrada e uma saída cheia de caminhos que nos levam a caminhos sem saída, espera-se então deste projeto que seja possível que um programa em Assembly encontre o menor caminho que possa unir a entrada e a saída, e tudo isso com o menor número de passos possíveis.

Para este feito, temos um arquivo de entrada e um arquivo de saída, ambos em formato PGM, um formato de imagem bem simples e vetorizado, contendo os valores da escala de cinza de cada pixel de uma matriz bidimensional definida.

2 – Desenvolvimento

Pode-se dividir o desenvolvimento do projeto em etapas: definição de referências globais; abertura e fechamento do arquivo de entrada; obtenção das linhas e colunas; procura do início do labirinto; marcação da entrada e da saída do labirinto; a resolução do labirinto através do método da eliminação dos caminhos falhos; escrita do arquivo PGM de saída.

2.1 – Definição de referências globais

Logo no início do programa são definidos o nome do arquivo de entrada e saída, algumas referências a tabela ASCII e o espaço que foi definido para o buffer que ira receber o arquivo de entrada.

2.2 – Abertura e fechamento do arquivo de entrada

Com as referências definidas foi feita a abertura e leitura do arquivo PGM de entrada através dos registradores de argumentos e do comando “system call” (syscall), o qual o endereçamento foi atribuído ao registrador $s0 que será utilizado no código.

2.3 – Obtenção das linhas e colunas

A obtenção das linhas e colunas é feita iterando-se no buffer do arquivo até a terceira linha a qual contem a quantidade de linhas e colunas, foi utilizado o comando mul, para aquisição das casas decimais.

2.4 – Procura do início do labirinto

Neste ponto, o programa itera no buffer do arquivo até encontrar um bit 0 (primeira célula preta do vetor do labirinto), que indica o início do labirinto, e o atribui a um registrador, que salva o endereço do início.

2.5 – Marcação da entrada e da saída do labirinto

A marcação dos pontos de entrada e saída foi feita através de uma alteração (somado em 1) no primeiro bit de ambos os pontos, para que dessa forma, o próximo passo do programa entenda que essas duas regiões são extremidades e não caminhos sem saída.

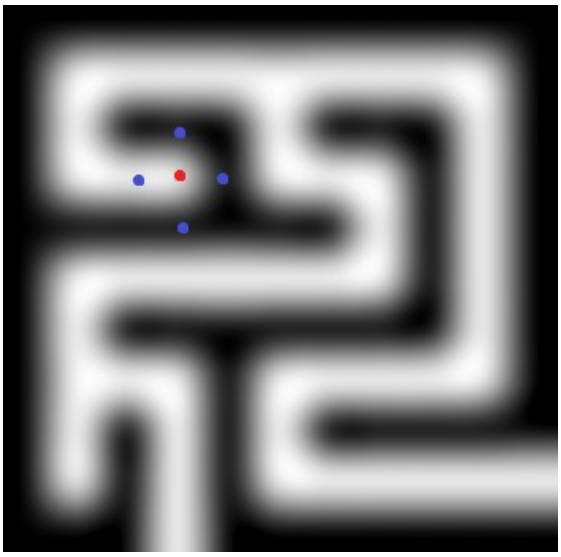
2.6 – A resolução do labirinto através do método da eliminação dos caminhos falhos

Primeiramente para contextualização do método precisamos entender que, um labirinto em formato PGM, nada mais é que um vetor que representa uma imagem com dimensões n x m, com escalas entre a cor preta e a cor branca.

Tendo em mente o contexto que foi apresentado, o método de solução do labirinto por exclusão de falhas age com um pensamento simples: “Excluir todas as falhas até que sobre apenas o caminho que ligue a entrada e a saída”, para facilitar o entendimento do método, apresentarei um exemplo:

Primeiramente devemos entender que a lógica segue um contexto de “looping” ou recursividade, seguindo uma sequência específica de tarefas:

- Primeiramente o algoritmo busca por uma célula branca no vetor da imagem.
- Logo após garantida a célula branca, o algoritmo conta o número de vizinhos nos 4 pontos cardiais que sejam da cor preta, ou seja “paredes do labirinto”.
- Com essa contagem em mãos o programa verifica se o número de vizinhos é igual a 3, indicando que foi encontrado uma posição sem saída do labirinto, como indicado na figura abaixo:



- Caso a condição anterior seja atendida o programa pinta aquela região de cinza, trocando o primeiro bit daquela posição para 0.

Por fim, o que temos ao excluir todos os caminhos que nos levam ao caminho sem saída, é somente o caminho correto, que une a entrada e a saída, desta forma:

2.7 – Escrita do arquivo PGM de saída

Por fim, é feita a escrita da imagem PGM de saída utilizando-se do comando 15 do system call (sys call), baseada no buffer que foi modificado, durante a resolução.

3 – Utilização

Para utilizarmos o programa, basta que obtenhamos uma IDE que rode o programa desenvolvido em assembly, que no caso do desenvolvimento inicial foi a MARS IDE, uma imagem de entrada no formato PGM contendo um labirinto com um pixel de largura entre as paredes e por fim a configuração do nome do arquivo de entrada que será executado em questão nesta parte a indicada a seguir:

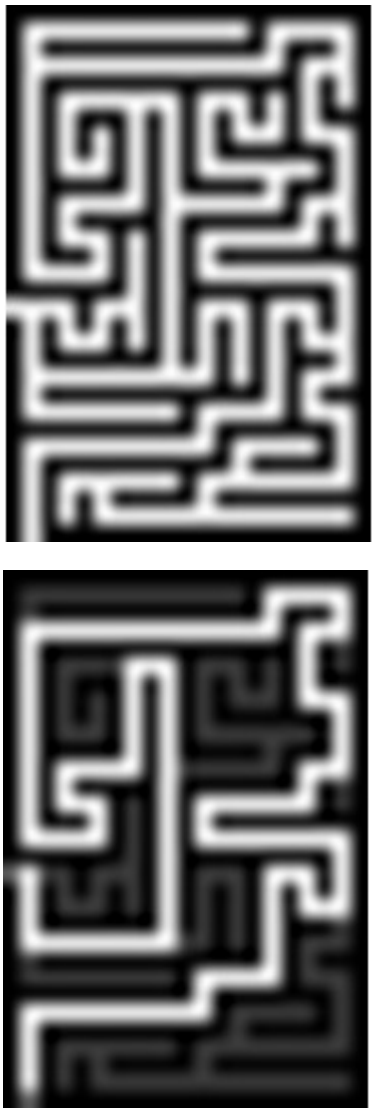
```
# 1.Definição de referências globais
.data
arquivo: .asciiz "maze-1-1.pgm"
nome_arquivo: .asciiz "Solucao.pgm"
buffer: .space 128000
```

4 – Resultados

Foram testados 4 labirintos, nos quais ficou notável a diferença de tempo de execução entre cada um deles, por conclusão foi percebido que devido à complexidade do algoritmo, e por se tratar de um algoritmo recursivo o tempo estará sempre em uma função exponencial em relação ao tamanho do arquivo de entrada.

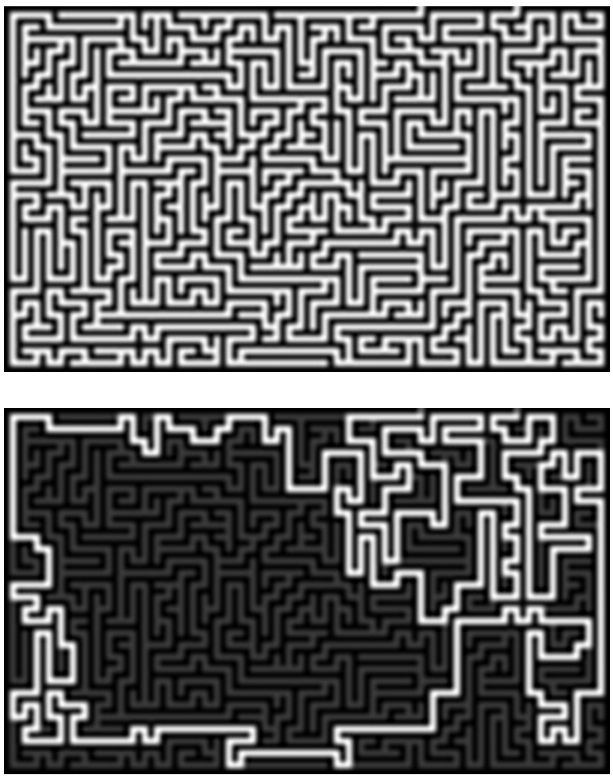
A seguir estão algumas imagens dos labirintos antes e depois de serem resolvidos e seus devidos tempos de execução:

Labirinto 1
Arquivo (maze-1-2)



Tamanho: 3 Kb
Tempo de execução: 30 segundos.

Labirinto 2
Arquivo (maze-1-3)



Tamanho: 24 Kb
Tempo de execução: 1 Hora e 30 minutos

Considerações Finais

Este trabalho foi desenvolvido em conjunto e colaboração com os alunos Francisco Correa Neto, e a aluna Mariana Regina Cabrinha de Lima, e o trabalho em equipe foi indispensável para o sucesso do mesmo.

Referências

[1] Assembly Commands
https://www.youtube.com/playlist?list=PL5b07qlmA3P6zUdDf-o97ddfpvPFuNa5A

[2] System Call Functions in MARS
http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html

[3] ASCII Table
https://www.rapidtables.com/code/text/ascii-table.html